

OPSCODE

1. Home 5
1.1 About 10
1.1.1 What is Chef? 12
1.1.2 Chef Core Principles 14
1.1.3 Why Chef? 15
1.1.4 The Different Flavors of Chef 17
1.1.5 FAQ 18
1.2 Chef Basics 21
1.2.1 Architecture Introduction 23
1.2.2 Core Components 26
1.2.3 Introduction to Cookbooks and More 30
1.2.4 Introduction to Search and Data Bags 35
1.3 Architecture 37
1.3.1 Authentication and Authorization 40
1.3.1.1 Hosted Chef Authorization 45
1.3.1.2 Making Authenticated API Requests 49
1.3.1.3 API Clients 55
1.3.2 Anatomy of a Chef Run 57
1.3.2.1 Evaluate and Run Resources at Compile Time 61
1.3.3 Chef Client 63
1.3.4 Chef Solo 65
1.3.5 Chef Server 70
1.3.5.1 Chef Indexer 73
1.3.5.2 CouchDB Administration for Chef Server 76
1.3.5.3 Backing Up Chef Server 78
1.3.6 Hosted Chef 80
1.3.6.1 Migrating to Hosted Chef from an Open Source Chef Server implementation 82
1.3.7 Server API 86
1.3.8 Cookbook Site API 112
1.3.9 Chef Concepts as UML 118
1.4 Chef Essentials 121
1.4.1 Attributes 124
1.4.1.1 Automatic Attributes 129
1.4.1.2 Deep Merge 132
1.4.2 Cookbooks 136
1.4.2.1 Cookbook Attributes 141
1.4.2.1.1 Setting Attributes (Examples) 144
1.4.2.2 Definitions 146
1.4.2.3 File Distribution 150

1.4.2.4 Libraries . . . 152
1.4.2.5 Metadata . . . 156
1.4.2.6 Recipes . . . 163
1.4.2.6.1 Creating a "First Run Only" Resource . . .
1.4.2.6.2 Unix Environment Variables and Chef Recipes . . . 174
1.4.2.7 Templates . . . 178
1.4.2.8 Version Constraints . . . 183
1.4.2.9 Cookbook Site Help . . . 187
1.4.2.10 Chef Repository . . . 192
1.4.2.11 Cookbook Workflows . . . 197
1.4.3 Data Bags . . . 198
1.4.3.1 Encrypted Data Bags . . . 205
1.4.4 Environments . . . 209
1.4.5 Exception and Report Handlers . . . 216
1.4.5.1 Distributing Chef Handlers . . . 221
1.4.6 Nodes . . . 223
1.4.6.1 Setting the run_list in JSON during run time . . . 226
1.4.7 Ohai . . . 227
1.4.7.1 Ohai Installation and Use . . . 228
1.4.7.1.1 Installing Ohai on Windows . . . 230
1.4.7.2 Writing Ohai Plugins . . . 231
1.4.7.3 Loading Custom Ohai Plugins . . . 234
1.4.7.4 Distributing Ohai Plugins . . . 235
1.4.7.5 Disabling Ohai Plugins . . . 237
1.4.8 Resources and Providers . . . 238
1.4.8.1 Resources . . . 240
1.4.8.1.1 Deploy Resource . . . 282
1.4.8.1.2 Yum Package Resource . . . 292
1.4.8.2 Providers . . . 296
1.4.8.3 Lightweight Resources and Providers (LWRP) . . . 298
1.4.8.4 Opscode LWRP Resources . . . 305
1.4.9 Roles . . . 332
1.4.10 Search . . . 338
1.5 Installation . . . 350
1.5.1 Fast Start Guide . . . 352
1.5.1.1 Setup Opscode User and Organization . . . 360
1.5.1.2 Client Bootstrap Fast Start Guide . . . 364
1.5.1.3 Fast Start Guide for Windows . . . 370
1.5.1.4 Fast Start Guide for CentOS . . . 380
1.5.1.5 EC2 Bootstrap Fast Start Guide . . . 388
1.5.1.6 Cookbook Fast Start Guide . . . 399
1.5.1.7 Getting Started with Shef . . . 405
1.5.2 Workstation Setup . . . 409
1.5.2.1 Workstation Setup for CentOS, Red Hat, Fedora . . . 411
1.5.2.2 Workstation Setup for Debian and Ubuntu . . . 414
1.5.2.3 Workstation Setup for Mac OS X . . . 417
1.5.2.4 Workstation Setup for Windows . . . 420
1.5.3 Installing Chef Server . . . 426
1.5.3.1 Installing Chef Server Manually . . . 429
1.5.3.2 Installing Chef Server on Debian or Ubuntu using Packages . . . 436
1.5.3.3 Installing Chef Server using Chef Solo . . . 441
1.5.4 Installing Chef Client and Chef Solo . . . 447
1.5.4.1 Installing Chef Client on CentOS . . . 449
1.5.4.2 Installing Chef Client on OS X . . . 452
1.5.4.3 Installing Chef Client on Other Operating Systems . . . 454
1.5.4.4 Installing Chef Client on Ubuntu or Debian . . . 459
1.5.4.5 Installing Chef Client on Windows . . . 463
1.5.5 Upgrade Instructions . . . 467
1.5.5.1 Upgrading Chef 0.8.x RubyGems . . . 468
1.5.5.2 Upgrading Chef 0.8.x to 0.9.x . . . 469
1.5.5.3 Prepare Debian and Ubuntu for 0.9 Upgrade . . . 471
1.5.5.4 Upgrading Chef 0.9.x to Chef 0.10.x . . . 473
1.5.5.5 Using Chef to Upgrade Chef . . . 478
1.5.6 Vagrant . . . 479
1.5.7 Installing Chef from HEAD . . . 485
1.5.8 Configure Management Workstation As A Chef-Client . . . 490
1.5.9 Quick Start . . . 493
1.5.10 Running Chef Client with Elevated Privileges . . . 494
1.6 Managing Chef . . . 496
1.6.1 Knife . . . 498
1.6.1.1 Knife Built In Subcommands . . . 504
1.6.1.2 Knife Bootstrap . . . 519
1.6.1.2.1 Custom Knife Bootstrap Script . . . 523
1.6.1.3 Knife Windows Bootstrap . . . 525
1.6.1.4 Knife Exec . . . 529

1.6.1.5 Launch Cloud Instances with Knife . . . 535
1.6.1.6 Managing API Clients With Knife . . . 542
1.6.1.7 Managing Cookbooks With Knife . . . 545
1.6.1.8 Managing Data Bags With Knife . . . 555
1.6.1.9 Managing Environments With Knife . . . 558
1.6.1.10 Managing Nodes With Knife . . . 561
1.6.1.11 Managing Roles With Knife . . . 566
1.6.2 Knife Plugins . . . 569
1.6.2.1 Community Plugins . . . 577
1.6.3 Management Console . . . 581
1.6.3.1 Hosted Chef Management Console . . . 582
1.6.3.1.1 Managing Clients with the Hosted Chef Management Console . . . 583
1.6.3.1.2 Managing Cookbooks with the Hosted Chef Management Console . . . 587
1.6.3.1.3 Managing Data Bags and Items with the Hosted Chef Management Console . . . 589
1.6.3.1.4 Managing Environments with the Hosted Chef Management Console . . . 594
1.6.3.1.5 Managing Groups with the Hosted Chef Management Console . . . 599
1.6.3.1.6 Managing Nodes with the Hosted Chef Management Console . . . 603
1.6.3.1.7 Managing Organizations with the Hosted Chef Management Console . . . 609
1.6.3.1.8 Managing Permissions with the Hosted Chef Management Console . . . 615
1.6.3.1.9 Managing Roles with the Hosted Chef Management Console . . . 619
1.6.3.1.10 Managing Users and your Private Key with the Hosted Chef Management Console . . . 624
1.6.3.1.11 Managing your Account and Billing Information . . . 628
1.6.3.1.12 Managing your Opscode User Account . . . 637
1.6.3.1.13 Setup an Opscode User and Organization . . . 642
1.6.3.1.14 Using Search in the Hosted Chef Management Console . . . 646
1.6.3.2 Open Source Chef Server Management Console . . . 648
1.6.3.2.1 Looking up node status through the Management Console . . . 649
1.6.3.2.2 Management Console Users . . . 650
1.6.3.2.3 Managing API Clients through the Management Console . . . 654
1.6.3.2.4 Managing Cookbooks through the Management Console . . . 659
1.6.3.2.5 Managing Data bags through the Management Console . . . 661
1.6.3.2.6 Managing Environments through the Management Console . . . 664
1.6.3.2.7 Managing Nodes through the Management Console . . . 665
1.6.3.2.8 Managing Roles through the Management Console . . . 671
1.6.3.2.9 Search on the Management Console . . . 676
1.6.4 Shef . . . 677
1.6.5 Spiceweasel . . . 685
1.7 Documentation . . . 692
1.7.1 A Can of Condensed Chef Documentation . . . 695
1.7.2 Glossary . . . 698
1.7.3 Chef Configuration Settings . . . 707
1.7.3.1 Advanced Configuration Settings with Ruby . . . 717
1.7.4 Just Enough Ruby for Chef . . . 718
1.7.5 Release Notes . . . 722
1.7.5.1 Breaking Changes from 0.7.x to 0.8.x . . . 748
1.7.5.2 Release Notes Older Versions . . . 750
1.7.6 Chef OmniGraffle & Visio Stencils . . . 772
1.8 Guides . . . 773
1.8.1 Build a Django Stack . . . 776
1.8.2 Build A Java Web Stack . . . 783
1.8.3 Build a LAMP Stack . . . 790
1.8.4 Build A Rails Stack . . . 797
1.8.5 Chef Gem Man Pages . . . 803
1.8.6 Deploying OpenStack with Chef . . . 804
1.8.7 Guide to Creating A Cookbook and Writing A Recipe . . . 810
1.8.8 Headless Branches for Cookbook Repositories . . . 813
1.8.9 How to Proxy Chef Server with Apache . . . 815
1.8.10 Nagios Quick Start . . . 820
1.8.11 Performance Tune Ohai for Large User Bases . . . 827
1.8.12 Scalability and High Availability . . . 829
1.8.13 Working with Git and Cookbooks . . . 830
1.9 Developers . . . 839
1.9.1 Get Involved . . . 840
1.9.2 How to Contribute . . . 841
1.9.2.1 Apache License . . . 845
1.9.2.2 Contributor License Agreement . . . 848
1.9.2.3 Corporate Contributor License Agreement . . . 849
1.9.2.4 Working with Git . . . 850
1.9.3 Approved Contributors . . . 853
1.9.4 Release Process . . . 874
1.9.5 Release MVPs . . . 876
1.9.6 Feature Proposals . . . 879
1.9.6.1 After hooks . . . 880
1.9.6.2 Chef 11 Release Checklist . . . 882
1.9.6.3 Chef Groups . . . 883

1.9.6.4 Configuration Based on Partial Convergence . . . 884
1.9.6.5 Cookbook Documentation Functional Specification . . . 887
1.9.6.6 Explicit and Opt-in Filespecificity . . . 888
1.9.6.7 git resource updates . . . 889
1.9.6.8 Improved Windows File Security . . . 892
1.9.6.9 Infrastructure Proposal . . . 895
1.9.6.10 Writing Chef Recipes in various languages . . . 898
1.9.7 Chef Users List . . . 899
1.9.8 Chef Presentations . . . 903
1.9.9 AMI List for Developers . . . 907
1.10 Community Events . . . 909
1.10.1 ChefConf . . . 911
1.10.2 Opscode Community Summit 1 . . . 912
1.10.2.1 Commemorative T-shirt Ideas . . . 914
1.10.2.2 Participants . . . 915
1.10.2.3 Sessions . . . 923
1.10.2.3.1 Closing Session - Decisions, Follow-Ups, and Next Steps . . . 924
1.10.2.3.2 Day 1 . . . 926
1.10.2.3.3 Day 2 . . . 945
1.10.2.4 Suggestions for Community Summit . . . 967
1.11 Support . . . 969
1.11.1 Common Errors . . . 972
1.11.2 Cookbook Support . . . 983
1.11.3 IRC . . . 986
1.11.4 Mailing Lists . . . 987
1.11.5 Troubleshooting and Technical FAQ . . . 988
1.11.5.1 User Environment PATH Sanity . . . 998

Home



Welcome to the Chef Wiki!

Chef is a systems integration framework, built to bring the benefits of configuration management to your entire infrastructure.

With Chef, you can:

- Manage your servers by writing code, not by running commands. (via Cookbooks)
- Integrate tightly with your applications, databases, LDAP directories, and more. (via Libraries)
- Easily configure applications that require knowledge about your entire infrastructure ("What systems are running my application?" "What is the current master database server?") (via Search)
- Create perfect clones of QA environments, pre-production environments, partner preview environments and more. (via Environments)

Once automated, you hold a blueprint for your infrastructure, enabling you to build, or rebuild, automatically in minutes or hours – not weeks or months. Better still, when you take those environments live and reality intrudes - which, trust us, it will - Chef gives you endless flexibility to adapt on the fly.

It works the way you want it to:

We provide the building blocks that help you to accomplish your goals. At the end of the day, it's about your infrastructure being easy to manage. Chef provides the horse-power to get you to where you need to be - it doesn't start out telling you where you should be going.

About Chef

The open-source systems integration framework built specifically for automating the cloud

- **Tell me why! Should I use Chef?**
- What is Chef anyway?
- What are Chef's core principles?
- How many types of Chef are there?
- How is it licensed and why?

Chef Basics

Some central concepts for obtaining configuration management benefits in your infrastructure

- **New to Chef? Start Here!**
- Includes an Architecture Introduction,
- an overview of the Chef concepts and terminology,
- an Introduction to cookbooks, recipes, resources, and roles, and
- an Introduction to Search and Data Bags

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Architecture

There are several clients for interacting with a Chef Server, Hosted or Private Chef, or Chef Solo

- **How does Chef run? Understand the architecture**
- Understand the authentication and authorization models applied,
- the anatomy of a chef-client run within the architecture,
- architectural components the Chef Client, Chef Server, and Chef Solo,
- interaction with the REST API and more.

Chef Essentials

Understand these essential components for utilizing the full capability and power of Chef

- **Ready for more in-depth information? Dive into Details**
- An in depth exploration of all the features and functions of Chef
- Explore Cookbooks more fully
- Reach into Attributes, Definitions, Environments, Exception and Report Handlers, Resources, Roles, Ohai, and more.

Installation

No matter the complexity of your business, Chef makes it easy to deploy servers and scale applications across infrastructure

- **Ready to start automating? Install Chef!**
- Set up your Chef Workstation,
- OS specific installation instructions for Installing Chef Client and Chef Solo, or Installing the Open Source Chef Server.
- Upgrade Instructions
- Use of Vagrant for building and distributing virtualized development environments.

Managing Chef

Manage your infrastructure via command line, a WebUI, an irb session, and through adding plugins for additional functionality

- **Manage your Infrastructure: With Chef**
- Use of Knife - the powerful Chef command line utility to bootstrap new nodes, launch cloud instances and more.
- The Management Console WebUI Interface,
- Use of Shef for running in an irb session,
- Community Plugins to extend Chef, Ohai and Knife, Spiceweasel for batch-loading, and more.

Documentation

Reference documentation and related information - Read up here!

- **Want to broaden your understanding? Read up!**
- A Can of Condensed Chef Documentation with a Glossary of

Guides & Tutorials

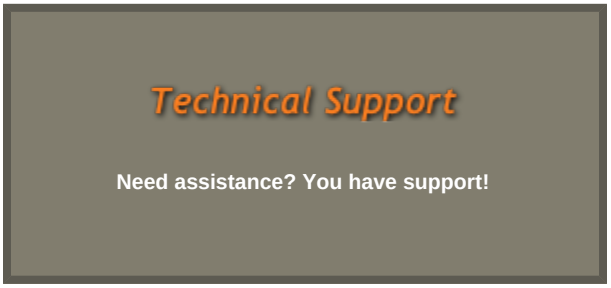
Walkthrough Guides, Webcasts, Tutorials, Training - Learn here!

- **Seeking Instruction? Learn Here!**
- Walkthrough guides and webcasts to go step by step through installing 6 common server stacks including a rails stack, a java web stack, and others.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

terms, technologies and concepts.
- Reference material on the Chef Configuration Settings and Just Enough Ruby for Chef, along with Chef OmniGraffle & Visio Stencils.
- Release Notes for each version
- All the Getting Started Docs, community member books and reference cards, and more.
- "How To Guides" including a Guide to Creating A Cookbook and Writing A Recipe, Deploying OpenStack with Chef and more.
- Multiple excellent tutorials put together by Chef community members, and
- Access to training courses and materials.





- **Want to Contribute? Join the Community!**
- When you use Chef, you join a vibrant community of professionals.
- Here you'll find How to Contribute and Approved Contributors.
- The Release MVPs from the Community,
- The Release Process for Opscode Open Source Projects.
- Feature Proposals that are larger than a more standard request, and more.



Official Community Events, including: **FoodFight** - a bi-weekly Chef Community Podcast, Opscode Events, MeetUps and Conferences we're attending, and the inaugural Opscode Community Summit 1 plus **#chefconf**: the Chef User Conference.

- **Seeking Help? You have Support!**
- Support resources for all Chef users - including:
  - Common Errors, and
  - a Troubleshooting and Technical FAQ.
- Real-time interactive community support from IRC, with IRC logs to review previous discussions.
- You can sign up for Chef Mailing Lists
- File an Open Source Ticket
- You can also follow us on Twitter through @opscode
- The Opscode Blog is updated regularly with company and product information, and has an RSS feed for your convenience.

Hosted and Private Chef users can also receive support by contacting support@opscode.com.

- Periodic maintenance is announced in advance via the Opscode Status site, as are any emergent issues with availability.
- Sign up for @opscode_status for timely notice of any information on Hosted Chef availability and performance.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Quick Links

- Installation
- Walkthrough Guides and Tutorials
- Chef Wiki As A PDF
- How to Contribute
- Source Code
- Open Source Ticket Tracking
- Just Enough Ruby for Chef
- Troubleshooting and Technical FAQ
- Resource Reference
- Support

Why Chef?



Watch a short video about Opscode Chef & Cloud Infrastructure Automation



Looking For Instant Gratification?

The **Fast Start Guide** will get you up and running quickly on Ubuntu or Mac OS X using Hosted Chef (as your chef server).

If you prefer to install your own server, use the Chef Server + Chef client directions at Installation.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Opscode.com Blog

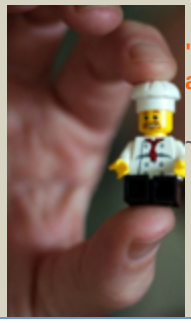
Automate All The Things!!! – Video of Jesse Robbins' Keynote @ Cloud Connect
Chef Support for Citrix CloudStack 3
Chef @Work: Telling Cool Customer Stories, Today and Everyday
Post-Hoc Index Design: From Regex to PEG
#ChefConf 2012 – Call for Proposals
Join Opscode at SCALE!

Opscode on



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

About



"Chef is like a little system admin robot ... you tell it how you want your system configured, and it will do all the dirty work."

~ Chef User

Chef is an open-source systems integration framework built specifically for automating the cloud.



No matter how complex the realities of your business, Chef makes it easy to deploy servers and scale applications throughout your entire infrastructure.

With Chef, you write abstract definitions as source code to describe how you want each part of your infrastructure to be built, and then apply those descriptions to individual servers.

***The result is a fully automated infrastructure*: when a new server comes on line, the only thing you have to do is tell Chef what role it should play in your architecture.**

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

What is Chef?

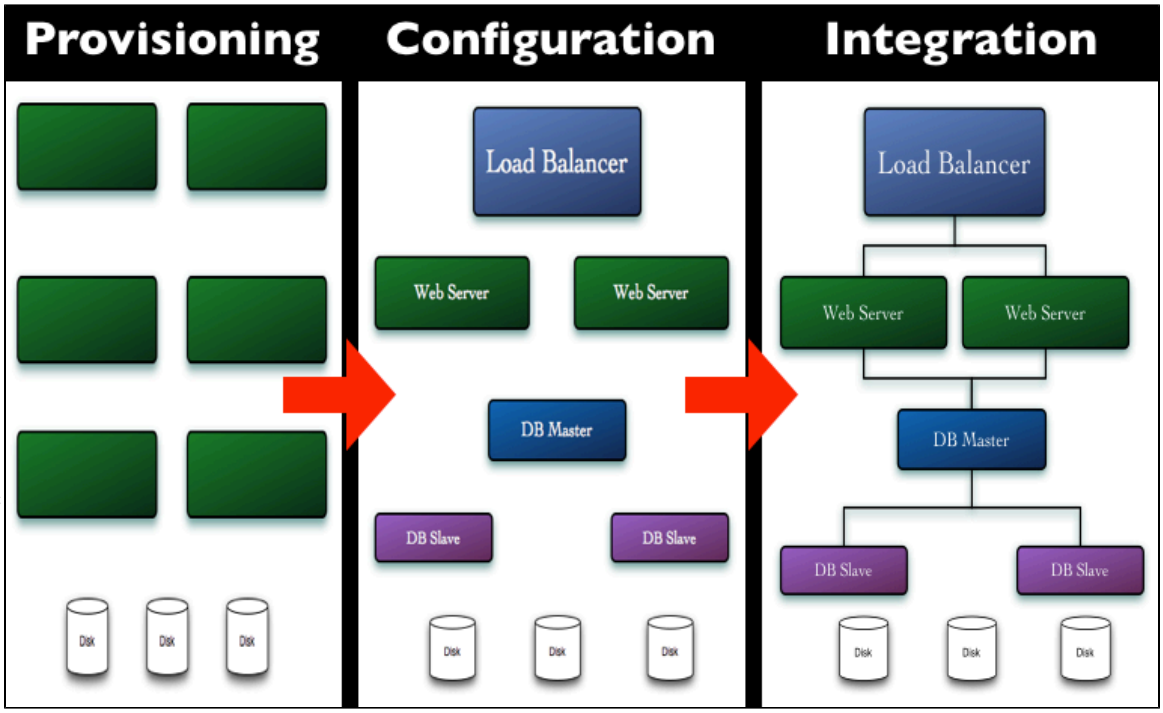
With Chef, you write abstract definitions as source code to describe how you want each part of your infrastructure to be built, and then apply those descriptions to individual servers.

The result is a fully automated infrastructure: **when a new server comes on line, the only thing you have to do is tell Chef what role it should play in your architecture.**

At a High Level, Chef Is...

1. A library for configuration management
2. A configuration management system
3. A systems integration platform
4. An API for your entire Infrastructure

Chef Provides



Provisioning

Chef invokes system and API calls to automate the provisioning of servers through RESTful API calls or through the command line interface knife. In the case of bare-metal, Chef integrates with things like Kickstart for Linux, Jumpstart for Solaris or NIM for AIX. For virtualized environments, Chef integrates with libvirt and hypervisor specific (XEN, KVM, VMware) API's. In Cloud scenarios Chef can invoke direct cloud API calls to provision servers for platforms such as AWS and vCloud. Chef is also integrated with many of the open source language specific cloud abstractions such as fog for Ruby and jclouds for Java.

Configuration Management

Chef is a fully functional configuration management tool. Roles and Recipes are used to describe how servers should be configured. Chef works by allowing you to write recipes that describe what roles a server (such as Apache, MySQL, or Hadoop) should be configured as. These recipes describe a series of resources that should be in a particular state - for example, packages that should be installed, services that should be running, or files that should be written. Chef then makes sure that each resource is properly configured, only taking corrective action when it's necessary. The result is a safe, flexible mechanism for making sure your servers are always running exactly how you want them to be.

Systems Integration

We like to call systems integration the last mile of fully automated infrastructure. One of the most powerful features of Chef is in its design of separating the configuration data from configuration logic. Data about your infrastructure is dynamically stored and indexed in an NoSQL database and a powerful search API is provided to query information about your infrastructure and applications. In other words Chef recipes can be data driven thereby providing dynamic system integration between servers. For example, when configuring a web server the search API can be called to discover the database and memcache servers and then automatically update the web server's configuration. Likewise a load balancer recipe can automatically add the web servers into its configuration.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



About

Chef Core Principles



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Core Principles



Chef is idempotent

You can run Chef recipes multiple times on the same system and the resulting system will always return to an identical state. In Chef, resources are defined in recipes and describe actions to be performed on the system. Chef ensures that actions are not performed if the resources have not changed. If you rerun a Chef recipe on a system and nothing has changed either on the system or in the recipe, Chef doesn't change anything.

Thick Clients, Thin Server

Chef does as much work as possible on the Chef Clients. The Chef Server is built to handle the easy distribution of data to the clients - the recipes to build, templates to render, files to transfer - along with storing the state of each Node. This orientation makes for a system that is easy to scale and extend - the work of deciding how to configure your infrastructure is distributed throughout your infrastructure, rather than centralized on set of configuration management servers.

Order Matters



When you are configuring systems, order matters. If you have not installed Apache, you can't start configuring it, and you certainly can't start the daemon. Configuration management tools have been struggling with this problem for years. Nodes in Chef apply lists of recipes, which in turn specify resources. Within a recipe, resources are applied in the order they appear. At any point in a recipe, you can include any other recipe - assuring that all of its resources are applied before continuing (Chef is smart enough never to apply the same recipe twice.) You specify dependencies only at the recipe level, not the resource level. This means that you only track dependencies between high level concepts - "I need Apache installed before I can start my Django Application". It also means that, given the same set of Cookbooks, Chef will always execute your resources in the same order.

Reasonability

Chef is designed to be easy to think about, easily changed and easy to extend. Chef assumes that you know best how your infrastructure is put together. Therefore, Chef makes as few decisions for you as possible, and when it does, it's easy to make it change its mind. When Chef does make decisions it sets reasonable defaults that can be easily changed. Moreover, Chef is pragmatic and opinion-less relying less on theory or dogma and more on deterministic results. This is why Chef uses Ruby as its reference language with extended DSL for specific resources. Chef provides a reasonable set of base primitives (i.e., resources) to automate an infrastructure; however, it also provides an easy way to modify and extend the base (via Ruby) – because ... TMTOWTDI.

 What is Chef?

Why Chef? 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Why Chef?

Gain Benefits In Three Areas

Economics

Chef saves you money by allowing your business to support larger, more complicated infrastructures with less manpower, less repetition, and fewer errors. Do more with less.

Efficiency

Automating your infrastructure with Chef means never having to go back in six months to repeat yourself. Plus, you won't have to do it all yourself. Leverage our vibrant community of experts who are sharing their recipes, collaborating on best practices, and helping each other to succeed. Get rid of all those little "5 minute" tasks, and get ready for the difference in your workday.

Scalability

The founders of Opscode have deep experience with what it takes to run infrastructure at scale. When we wrote Chef, we put that knowledge to work. Chef scales horizontally like a web application, so you'll be able to trust that Chef can keep up with even the fastest growing environment. Find out in the morning that you're being featured on CNN that afternoon? You're going to need some more horsepower! When your infrastructure is built with Chef, that's a snap, Just bring the new servers online, and let Chef take care of the rest.

It Works The Way You Want It To

Set It. Forget It.

Chef is infrastructure aware. When you add new servers, Chef automatically discovers thousands of data points about your system. When you need to know which web servers to load balance in production, you simply ask Chef – no need to change source code, or manually reconfigure based on static documentation. When you need to model new classes of data, from users to virtual machines, simply use the Chef API to store the information, and re-use the data instantly.

Your Problems. Your Solutions.

Other solutions impose their model on your process. But who knows your problems best, you or them? Chef allows you to automate your current processes - your model, your decisions.

Most infrastructure solutions force you to adapt to their "one-size-fits-all" approaches, even though every infrastructure challenge is different. That's crazy-talk. And it's precisely the reason why Chef is built directly on top of Ruby, a dynamic, open-source programming language. Integrated with well-known cloud platforms like Rackspace and EC2, Chef is built to seamlessly integrate into your existing environment. We designed Chef from the ground up to allow you to focus on solving your real business problems - because reality is chaotic enough. Dealing with it shouldn't be.

Modeling That's Truly Meaningful

Chef allows you to create perfect clones of your environments, from QA to pre-production to partner preview and more. Once automated, you hold a blueprint for your infrastructure, enabling you to build, or rebuild, automatically in minutes or hours – not weeks or months. Better still, when you take those environments live and reality intrudes - which, trust us, it will - Chef gives you endless flexibility to adapt on the fly. We all know business is "a series of managed disasters." Chef gives you the power to deal with them quickly, accurately and intelligently.

It is Truly a Community Effort

Sharing is the Norm

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



We released Chef on January 15th, 2009. As of Sept 17, 2011, more than 430 people and 90 companies have signed up to contribute to Chef. Companies like Dell, VMware and Engine Yard have integrated it into their business offerings directly. Many different cookbooks, automating everything from Apache web servers to Xen virtual machines have been published, and put to work in hugely different environments. When you use Chef, you join a community of professionals who have been there before, understand the problems we all face, and are working together to make things better for everyone. The results are far greater than any of us could have possibly achieved alone.

Chef is Apache Licensed

Opscode is an open source company, and we've written extensively about what that means to us . We released Chef under the Apache License to ensure that, no matter how you want to use Chef, you can. No matter how you contribute, the license ensures that we are all equals.

Chef Core Principles

The Different Flavors of Chef

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

The Different Flavors of Chef

One size doesn't fit all. There is an Opscode Chef product that can work *for you*.



Chef Solo

Chef Solo is an open source standalone version of Chef that runs locally on your node, detached from a Chef server. This means that all the information and Cook books required to configure the node have to be present on the local disk. They can be retrieved via a remote URL with a command-line or config file option.

Chef Client and Chef Server

The open source Chef Server is a client/server delivery of Chef functionality and power. Chef Client is a Chef agent that runs locally on the systems ("nodes") you are managing with Chef. Chef-client connects to a Chef Server to be told what to do on the node. This allows for more dynamic and flexible configuration management. For example, nodes can have roles applied to sets of them for consistency, and they can query information stored centrally on the server by other nodes, to dynamically configure themselves according to changing conditions elsewhere in your infrastructure.

Hosted Chef

Opscode's Hosted Chef combines the freedom and flexibility of Chef with the reliability and speed of Opscode. Hosted Chef is a highly available, scalable Chef Server in the cloud, plus additional features, such as fine-grained role-based access control. It is also the world's first configuration management platform delivered as a service (PaaS). This means that it is easier than ever to start using Chef to manage your infrastructure. As with Chef-Server, Chef-client connects to Hosted Chef to be told what to do on the local node: enabling you to automate your infrastructure, without having to setup and manage your own Chef-Server.

Private Chef

The power and speed of Opscode. The freedom and flexibility of Chef. Completely customized for your business. Private Chef is for Enterprises who want the power, flexibility, availability, and performance of Hosted Chef, but require that information never leave their private networks.

Why Chef?

FAQ

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

FAQ

This is an "About Chef" FAQ.

For common troubleshooting tips and a Technical FAQ, see Troubleshooting and Technical FAQ

Why did we create Chef?



Opscode began as a consulting company called HJK Solutions. We built fully automated infrastructure for startups - everything from OS installation to Application Deployment, fully integrated and ready to rock. (You can see a presentation on our approach on Slideshare)

Over the course of building a dozen or so different startups on the same infrastructure baseline over a year, we realized that we just were never going to get to a place where **everyone** could have a fully automated infrastructure, regardless of their Systems Administration expertise. A few things were in the way:

- A fully automated infrastructure is a fully integrated infrastructure - the different components need to be able to communicate with each other. (Your application informs your infrastructure, just like your infrastructure informs your application.)
- Configuration Management is the fundamental bedrock of infrastructure automation - for the infrastructure to be fully automated, you need to be able to expose Configuration Management as a Service.
- It was too difficult to share the code that builds the infrastructure - the tools at hand all required a level of specificity that made sharing difficult.

We solve these problems by:

- Building a systems integration framework on top of a configuration management system. That framework is powered by Ruby, but has a very simple DSL - making it very approachable for beginners.
- Making it possible to extend the capabilities of Chef easily, and by making Chef's Resources capable of taking action from ad-hoc sources. (This isn't released yet, but it will be soon - and trust us, it is awesome.)
- Every decision about Chef was made with an eye to keeping as much as possible inside of Cookbooks - sharable chunks of automation that are easily re-used and extended.

The goal is to remove ourselves from the process of building automated infrastructure entirely - Chef is the first part of a framework that allows us

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

to do that.

How can I help?

Join us on IRC, sign up for the Chef Mailing Lists, and read the instructions on how to contribute to an Opscode Open Source project.



Who is using Chef?

See a sample of the companies & organizations already using Chef.

Can I trust Chef?

Yes. Chef won't do anything to your system that isn't in a Recipe. Since Chef is an Open Source project, you have full access to the source code.

Do I need to know Ruby to use Chef?

It helps, but its not required. You can learn Just Enough Ruby for Chef.

Why did you create Chef rather than adapt an existing Open Source tool?

Chef was born out of our experience building fully automated production infrastructures with a variety of tools. That experience helped us define clear requirements for tool that went far beyond traditional configuration management. After surveying many different Open Source tools, we found that none met our needs.

Developers need a tool they can safely integrate into their code. Chef and Ohai are licensed under the Apache License Version 2.0 - a liberal, non-copyleft free software license. We maintain Contributor License Agreements, which allows anyone who uses Chef or Ohai to know they are free from any copyright or patent entanglements.

Why did you choose the Apache License?

We've written extensively about why we chose the Apache License on our blog.

How is Chef different than Puppet?

Puppet evolved from Cfengine and showed the potential of a new kind of configuration management.

The original design of Chef was strongly influenced by our own experiences working with and contributing to the Puppet project. However, Chef does not share any code from Puppet, and is not a "fork" of the Puppet project.



Chef is different from Puppet in a number of important ways:

- Chef uses Ruby as the configuration language, rather than a custom DSL.
- Chef is designed from the ground up to integrate with other tools, or to make that integration as simple as possible. Chef is not the canonical representation of your infrastructure. It is a service that exposes data about the state of your infrastructure.
- Chef applies resources in the order they are specified in your Recipes - there is no dependency management. This means

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

multiple Chef runs will always apply the Res
ources under management in the same order, every time.
- Chef Resources have Actions, which can be signaled.
- Resources can appear more than once in Chef, and they inherit the attributes of the earlier resource. (ie: you can tell Apache to start and stop in a recipe by specifying the resource twice, with the second one only changing the action attribute).

As Chef grows, the services we expose will likely be integrated with Puppet as well. There is more than one way to do it.

How is it different than Cfengine?

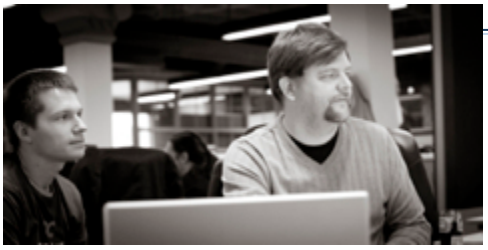
It bears very little in common with Cfengine, other than embracing Single Copy Nirvana.

 The Different Flavors of Chef

Chef Basics 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Basics



Chef is a systems integration framework, built to bring the benefits of configuration management to your entire infrastructure.

With Chef, you write abstract definitions as source code to describe how you want each part of your infrastructure to be built, and then apply those descriptions to individual servers.

Chef Basics



To use Chef, you first need to get acquainted with some central concepts. The following articles are basic introductions to core chef concepts.

- ***Architecture Introduction***

  An introduction to Nodes, Chef Workstation, and Chef Server. This article covers the role of each of these components and how they interact. If you are new to Chef, start here!

- ***Core Components***

  A single-page overview of the terminology and concepts within Chef.

- ***Introduction to Cookbooks and More***

  An introduction to cookbooks, recipes, resources, and roles. This article covers the core ideas of how one builds a node's configuration using the Chef DSL.

- ***Introduction to Search and Data Bags***

  An introduction to two of Chef's most powerful features: Search and Data bags. This article covers how node configuration can be generated dynamically using data about your infrastructure.

Guides and Tutorials

For those looking for hands-on tutorials, the **Guides** and **Fast Start Guide** section include a number of useful articles. Including:

- Guide to Creating A Cookbook and Writing A Recipe
- Deploying OpenStack with Chef
- Cookbook Fast Start Guide
- EC2 Bootstrap Fast Start Guide
- Client Bootstrap Fast Start Guide

Want to Get Started Quickly?

Head over to the Fast Start Guide to get up and running with Hosted Chef as quickly as possible, then head back here for a basic overview of Chef concepts.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

In addition to **Walkthrough Guides and Webcasts** take you step by step through the installation of 6 different common server stacks, *and more*.

From the Community

- **Introduction to Chef** - A nice and concise overview, from Warwick Poole
- **Chef in the Cloud and On the Ground** - Great 60 minute presentation demonstrating use of Chef, from Michael Nygard
- **Introduction to Chef** - slides from Opscode Team Member Sean O'Meara's presentation at LISA.

 FAQ

Architecture Introduction 

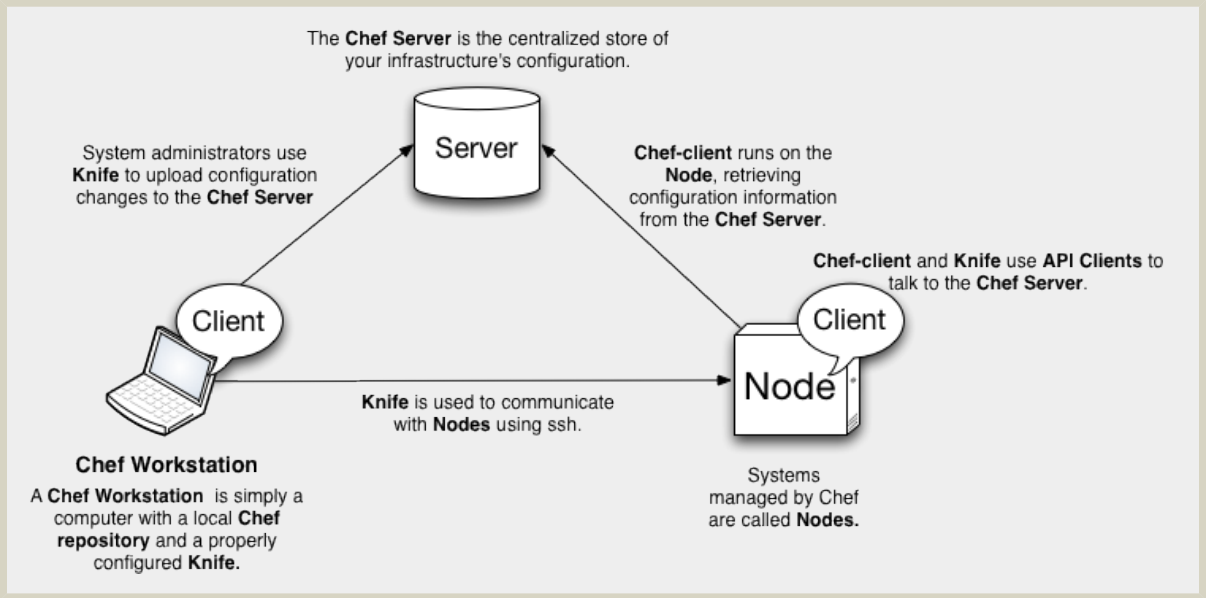
 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Architecture Introduction

This article is an introduction to Chef's architecture.
It covers the basic functions of the Chef Server, Nodes, and Chef Workstations when using Chef and how these components communicate.

For those using the Open Source Chef Server, you may also be interested in Chef Server, which goes into greater detail regarding the components of a Chef Server.

Chef Server, Nodes, and Workstations



When using Chef to manage your infrastructure, you will be dealing with three types of hosts: a Chef Server, Nodes, and Chef Workstations.

Chef Server

The Chef Server is the central store of your infrastructure's configuration data. The Chef Server stores the data necessary to configure your nodes and provides search, a powerful tool that allows you to dynamically drive node configuration based on data. A REST API makes this data accessible to nodes and chef workstations. Optionally, you can use the Chef Server's WebUI to manage your infrastructure via a web interface. Hosted Chef users have a hosted Chef Server, managed as a service by Opscode with a WebUI interface that is accessible at http://manage.opscode.com.

Nodes

A Node is any host that is configured using chef-client. Chef-client runs on your nodes, contacting the Chef Server for the information necessary to configure the node. From the point of view of the Chef Server, a node is little more than a **run list**, the list of recipes and roles that will be applied to the node and define its configuration, and **attributes**, a set of data about the node itself. Since a Node is a machine that runs the `chef -client` software, nodes are sometimes be referred to as "clients". (This is separate from API clients, that are authenticating against the Chef-Server API.)

When nodes are referenced as "clients", it implies the link between the chef-client executable, the API call made for authentication and authorization using the identity information in the API client object, to save the node object on the server.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Workstations

A Chef Workstation is the host you use to modify your cookbooks and other configuration data--typically the local workstation of a system administrator. Such a workstation has two key components:

1. the knife executable shipped with chef and
2. a repository containing your infrastructure's configuration documents.

As we will see, these configuration documents include cookbooks, data bags, roles, and more. Ideally, these configuration documents are managed by a version control system. Using `knife`, your workstation is used to upload configuration data to the Chef Server as well as communicate with individual nodes over SSH when necessary.

Clients, Authentication, and Authorization

Chef uses API Clients and a system of authentication and authorization to ensure that your configuration data is only read and updated by those authorized to do so. **Chef-client (running on nodes) and knife (running on workstations) communicate with the Chef Server using API clients.**

The Chef Server's role within your infrastructure is simple

It stores configuration information that is uploaded to it, and returns that information when asked by an authorized API client.

In other words, *Chef Client* is the 'chef-client' software executable itself. An *API Client* is the object that represents the cryptographic identity used for authentication and authorization in every API call.

To ensure that requests are coming from known API clients, Chef uses public key encryption, with Hosted Chef users having some additional fine-grained controls. API clients have associated private and public encryption keys. Every API request made by chef-executable is signed with a specific API client's private key. The Chef Server then verifies that the request is authentic with the client's public key.

Once the Chef-Server verifies that the request is authentic, it then ensures that the client making the request, has permission to take whatever action they are requesting (such as uploading a cookbook or retrieving a run_list). This process is known as **authorization**. Open source Chef Server and Hosted Chef use different authorization systems.

Open Source Chef

In Open Source Chef, certain API clients can be marked as "admin clients." Authorization for various actions is determined by whether the API clients is marked as an admin or not. Admin clients are used when making requests via knife and have all possible permissions. Non-admin clients have limited permissions and are used for nodes within your infrastructure.

Hosted Chef

Hosted Chef uses a more complex authorization system that provides fine-grained control on a client-by-client (or user-by-user) basis. In this system, API clients and users can have **CREATE**, **READ**, **UPDATE**, **DELETE**, or **GRANT** permission on the objects stored on Hosted Chef (node data, cookbook data, etc). Hosted Chef ensures that the requesting client has the appropriate permission for the requested action. In order to make managing permissions easy, Hosted Chef provides a number of pre-made groups that have permissions appropriate for different roles. Users and Clients that are members of the "admin" group have all permissions on all objects. Non-human API clients (such as those used by nodes) will automatically be placed in the client's group, which has permissions similar to non-admin clients in Open Source Chef, while human users can be placed in the "users" group, which has permissions similars to admin clients in Open Source Chef. For more details see Hosted Chef Authorization.

When chef-client runs on new nodes, Chef is designed to automatically create a new API client for the given node. Because of this, once your users or admin clients are set up properly to use knife and log into the WebUI or Management Console, there is often little need to think about the specifics of API clients, authentication, or authorization.

Hosted Chef

In addition to the more fine-grained authorization system, Hosted Chef has two concepts not used in Open Source Chef: Users and Organizations.

Users

Users are what you think they are: credentialed entities used by humans to connect to Hosted Chef. Like

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

clients, they have keys that they can use to connect to Hosted Chef programmatically, via api.opscode.com or knife. But they also have passwords that they can use to log in to the various web pages that Hosted Chef exposes.



Users can be associated with an organization, which gives them certain rights to interact with and manage that organization. However, unlike API clients, Users can be associated with more than one organization, for example a consultant could create a separate organization for each of his customers. When thinking about authentication and authorization issues, it is easiest to think about a user as just another API client.

Users have single accounts across all the Opscode web pages. The same username and password can be used to log in to Hosted Chef Management Console, the Opscode Cookbooks Site, or help.opscode.com.

Organizations

When you sign up for Hosted Chef, you create a user account and an organization. Your organization may represent your entire company, or the department you work in, or any other grouped set of server infrastructure you plan on managing. Organizations are the things that get billed for use of Hosted Chef. With the exception of users, nothing is shared between organizations. An API client in one organization cannot read node data from another organization. When using Hosted Chef, it is as if each organization has its own, private chef server, managed by Opscode.

Summary

- The Chef Server is the central store of configuration information.
- Nodes and Workstations communicate with the Chef Server using API Clients that make requests to a REST API.
- All API requests are checked for authenticity and authorization.

Chef Basics

Core Components

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Core Components



As you learn to use Chef, you'll need to understand what each of the different components are, and how they work together.

This page provides a brief overview of those components, and each has a more in-depth page.

Modeling Your Infrastructure

Nodes

A node is a host that runs the Chef client. The primary features of a node, from Chef's point of view, are its attributes and its run list. Nodes are the thing that Recipes and Roles are applied to. You want to work with Nodes any time you want to:

- Add a recipe to a system
- Update an attribute

Run List

In the simplest case, the run list is a list of the recipes that a node will run. Assuming the cookbook metadata is correct, you can put just the recipes you want to run in the run list, and dependent recipes will be run automatically as needed. Ordering is important: the order in which recipes are listed in the run list is exactly the order in which chef will run them.

In more advanced usage, the run list will include roles that a node has been assigned in addition to any recipes set on the node explicitly. In this case, when the Chef client runs, the run list is "expanded" into a list of recipes by replacing the role's entry in the run list with the list of recipes the role specifies in its run list.

Introduction to Chef

Opscode team member Sean O'Meara has a nice Introduction to Chef from a presentation at LISA in 2011.

Node Attributes

Nodes and roles have associated attributes, which are a structure of nested key–value pairs. Node and role attributes are commonly used as inputs for resource attributes. For example, your production boxes might be using one version of nginx, but you'd like to install a newer version on your staging servers for testing. By using a node or role attribute to specify the version, you can use the same recipe in both environments.

Chef allows attributes to be set in attribute files (among other myriad ways). Code in these files accesses the node chef is running on, and manages attributes on that node directly. In ruby terms, the value of `self` within attribute files is the node. Using attribute files, you can rely on a node having a sane value for an attribute when writing a recipe without having to worry that the node may not have defined that attribute.

Advanced Chef users often make heavy use of attributes defined in roles to manage attributes on many nodes at once.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Roles

A role provides a means of grouping similar features of similar nodes. At web scale, you almost never have just one of something, so you use roles to express the parts of the configuration that are shared by a group of nodes. Roles consist of the same parts as a node: attributes and a run list. When the Chef client runs, it merges its own attributes and run list with those of any roles it has been assigned.

Configuring Nodes

Cookbooks

A cookbook is a collection of recipe, resource definition, attribute, library, cookbook file and template files that chef uses to configure a system, plus metadata. Cookbooks are typically grouped around configuring a single package or service. The MySQL cookbook, for example, contains recipes for both client and server, plus an attributes file to set sane defaults for tunable values.

Cookbooks are the unit of distribution and sharing in Chef. Most of the time you are using Chef, you are writing cookbooks.

Recipes

Recipes are the files where you write your resources (described below). Recipes can also contain arbitrary ruby code, but you need to understand a little bit about how Chef runs to make productive use of this. Each Chef run is a two stage process: in the first step, called the compilation step, Chef evaluates the recipe files, building up a list of the resources. In the next step, Chef executes the desired action for each resource on the provider for that resource. Any arbitrary code in a recipe will be run during the compilation step, not the execution step. To defer execution until the execution phase of the Chef run, use the ruby_block resource. For a more detailed discussion of this process, refer to the anatomy of a chef run

Metadata

Cookbooks often rely on other cookbooks for pre-requisite functionality. In order for the server to know which cookbooks to ship to a client, a cookbook that depends on another one needs to express that dependency somewhere. That "somewhere" is in cookbook metadata. Dependency tracking is the most visible part of metadata, but metadata also can contain information about authorship, licensing, a description of the cookbook, what platforms the cookbook is expected to work on, and whether or not a cookbook plays nice with other cookbooks. At the moment, Chef supports many more fields for metadata than it actually uses, but maintaining accurate dependency information is absolutely essential, since nodes may not get all of the cookbooks they need if this information is incomplete.

Resources

A resource is *usually* a cross platform abstraction of the thing you're configuring on the host. For example, packages may be installed via apt, yum, or the BSD ports and packages systems, but the package resource abstracts these differences away so you can specify that a package should be installed in a cross-platform way. Chef's resources are mostly just containers for data, with some basic validation functionality.

Resources are declared in Recipes and Definitions. They are the basic unit of work in Chef.

Resource Attributes

Resource Attributes or Parameters

The Chef developers have discussed renaming Resource Attributes as "parameter" to avoid overloading the term "attribute," as that term is also used to describe data associated with nodes and roles. *The current documentation uses the term "attribute," so keep in mind that these are different from node attributes.*

As noted above, resources are mostly containers for data. Attributes are the pieces of data that resources contain. In the case of managing a package, this might be the name of the package you want to install, the version you want to install, or options to pass to the package manager.

Actions

The action is what you want Chef to do with the resource: should the package be installed? Upgraded to the latest version? Removed? Actions

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

are usually specific to the resource, but all resources support the *nothing* action, which does what its name suggests.

Providers

The provider is the platform-specific implementation of the thing a resource abstracts. On Red Hat or CentOS, a package resource will use the yum package provider to get the package installed, but on Debian and Ubuntu, the apt package provider will be used. Providers contain most of the intelligence: they're responsible for making Chef idempotent by checking if an action needs to be taken and issuing the commands to the system to take that action. In the case of package providers, they first check if the desired version of a package is installed and run the yum, apt-get, or or other package manager commands to install or upgrade as needed. When working with Chef, you normally don't need to worry about providers. For the occasions when you do, Chef provides "shortcut" resources which will always use the desired provider. The dpkg_package and rpm_package resources allow you to install packages directly from the filesystem, using providers specific to these package managers, for example.

Providers take action on Resources. A given Node decides what Provider should be used for a Resource by default, or a Resource can specify a provider explicitly.

Search

Search are built by the Chef Server, and allow you to query arbitrary data about your infrastructure. Most often, you utilize this service via search calls in a recipe.

Data Bags

Chef Version Requirement for Data Bags

As of version 0.10.4, Data Bags are available for all flavors of Chef. For Chef-Clients prior to version 0.10.4, you need to be using Hosted Chef, or an Open Source Chef Server at that version or later.

Data Bags store nested key–value data (like attributes) on the chef server. Data Bag data are searchable, and can also be loaded directly by name in a recipe. Data Bags are global for your chef-server installation–you can think of them as attributes for your whole infrastructure.

Environments

Environments provide a mechanism for managing different architectural segmented spaces such as production, staging, development, and testing, etc with one Chef setup (or one organization on Hosted Chef). With environments, you can specify per environment run lists in roles, per environment cookbook versions, and environment attributes: allowing you to set policies to dictate which versions of a given cookbook may be used in an infrastructure segment. Environments are not a multi-tenancy isolation of particular interactions on a single system, they act as cookbook version constraints upon a grouping of systems that span uses - for example: development, test, production - where there are application servers and database server, etc. Roles are groupings of systems that perform a like function and which can span environments - for example: application server, web server, database server. Thus you can have per-environment run lists within roles - for example: in your development environment you want your application server to connect to a different load balancer than you do in your test environment - which are included in the same recipe.

Managing Chef

Knife

Knife is the command line interface to the Chef server, though it also provides some features that are useful for chef-solo as well.

Management Console

The Management Console is a web UI interface to the chef-server API. Many components can be managed through the console including users, nodes, roles, cookbooks, data bags, and API clients. Search can also be done on the console. The Hosted Chef Management Console also allows customers to edit information in their user profile such as billing or account information, and to manage Hosted Chef Authorization.

Shef

Shef is the chef console—it allows you to write, run, and debug recipes interactively. As of Chef 0.9.8, it includes support for reading and editing data on a chef-server as well.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

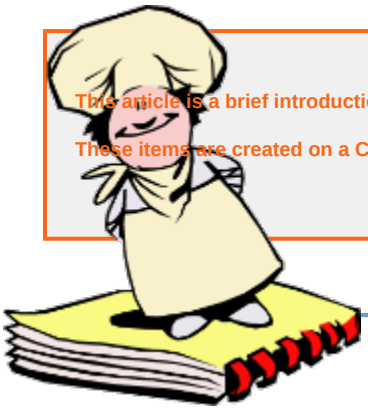
> ***NOTE:*** *Some of the information on this page is adapted from Chef Speak, with permission from author Opscode Team Member Dan DeLeo.*

 Architecture Introduction

Introduction to Cookbooks and More 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Introduction to Cookbooks and More



This article is a brief introduction to cookbooks, recipes, attributes, and roles.

These items are created on a Chef Workstation or via the WebUI, and are stored on the Chef Server.

Cookbooks

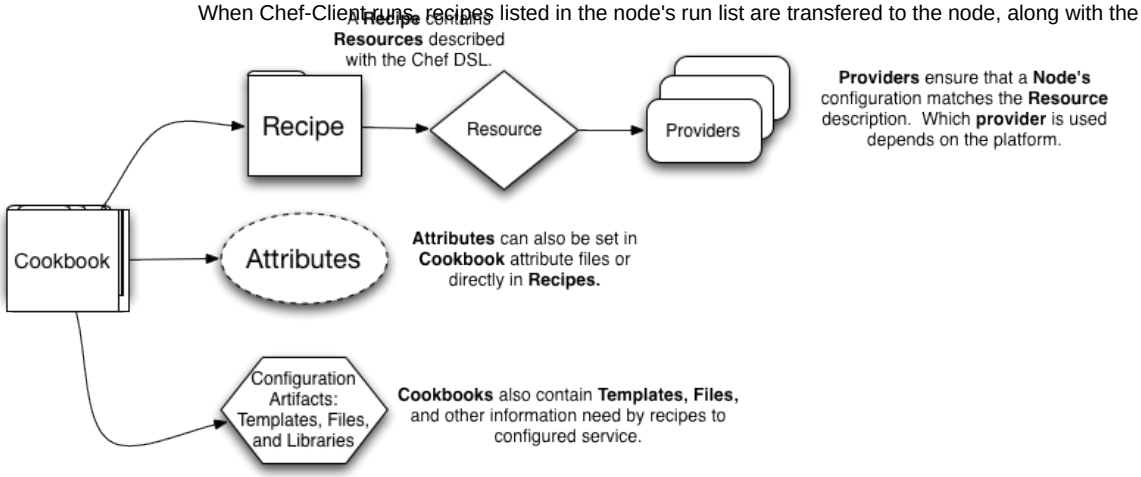
Cookbooks are Chef's fundamental units of distribution

Cookbooks are the way Chef and Hosted Chef users package up, distribute, and share configuration information. They encapsulate all the resources needed to automate your infrastructure, and are easily sharable with other Chef users as well.

They contain recipes, attribute files, templates, and other configuration artifacts.

When Chef-Client runs, recipes listed in the node's run list are transfered to the node, along with the

Typically, a single cookbook contains the information necessary to configure a single service or single part of the system. For instance, one may have a "users" cookbook for configuring the users who should have access to the system and an "apache" cookbook for configuring the apache web server.



Cookbooks can be authored by anyone with basic programming skills, and they can be written without storing any details about a particular deployed environment. This means they can be shared safely and reused across organization and company boundaries. Opscode encourages users to publish their cookbooks on the Opscode Community Site, where there are already over 300 cookbooks to choose from. As a result, you can install and configure many commonly used technologies without ever having to write a new cookbook.

If your Chef repository is using git (see Creating a Chef Repository for details), you can quickly obtain and use cookbooks written by the community using Chef's command line tool, Knife. For instance, to download the simple "getting-started" cookbook, run the following from a chef repository:

```
knife cookbook site install getting-started
```

To upload it to a Chef Server or Hosted Chef:

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife cookbook upload getting-started
```

And, to add the "default" recipe inside this cookbook to the run list of a node:

```
knife node run_list add my_node 'getting-started::default'
```

Chef also provides tools to make it easier to create new cookbooks. For instance, to create the basic structure of a new cookbook within your Chef Repository, you can use knife:

```
knife cookbook create new_cookbook_name
```

Recipes

Recipes are Chef's fundamental units of configuration

Recipes are Ruby files in which you use Chef's Domain Specific Language (DSL) to define how particular parts of a node should be configured. As will be seen in later sections, you can use data, combined with the ability to use ruby code within recipes to dynamically change a node's configuration.

Cookbooks can contain multiple recipes, with the recipes added to a run list by their full name using with the form `COOKBOOK_NAME::RECIPE_NAME`. Only roles and recipes are added to a run list.

Chef however provides a shortcut notation to name recipes and cookbooks in a run list command. (This is useful for the recipe that serves as a base configuration for a set of nodes.)

When you add `COOKBOOK_NAME` to a run list, Chef assumes that you want the recipe named "default" within the named cookbook. To see an example of this, we could modify the knife command used above to add a recipe to a node's run list. Using this shortcut, we can simplify the command given above:

```
knife node run_list add my_node getting-started
```

In this updated example, you've added the "default" recipe within the "getting-started" cookbook to the run_list of "my_node", just as before.

Resources and Providers

Resources are Chef's fundamental units of work

Resources are the building blocks you will use to to create a properly configured node. They are cross platform abstraction of that which you are configuring on the node - discrete chunks of a system's configuration that are placed in recipes and applied to your nodes.

The key feature of resources is that they allow you to focus on describing the node's configuration rather than the specifics of how a given task is accomplished.

For example, the following resource adds a user to a node:

```
user "sam" do
     home "/home/sam"
     shell "/bin/zsh"
     comment "Sam loves DevOps"
     action :create
end
```

This resource describes a user, "sam", that we would like to create on our node. The actual creation is performed by **Providers** which executes the necessary commands to create a new user.

An individual resource can have multiple providers, each of which knows how to perform the necessary task on a different platform. Chef-client will choose the best provider for the node's platform. Thus, this single resource description can be used to create the user "sam" on a Linux

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Providers are how Chef supports multiple platforms with a single Resource

system and a FreeBSD system, without any modification.

The user resource above is structured like all resource descriptions in the Chef Domain Specific Language (DSL):

```
resource_type "resource_name" do
  resource_attribute value
  ...
end
```

A **resource type** corresponds to the kind of resource that you need to configure. Many different resources are defined by the Chef DSL. A full list of these resources can be found on the Resources page. As you advance to more complex uses of Chef, you can even extend the Chef DSL with your own resources.

The **resource name** is a string that identifies this specific instance of a resource. For a given node's configuration you may need to define many user resources, each of which will have their own name and can be referred to by other resources. The resource name is also, by default, used as the value for one of the resource's "resource attributes". For example, in the user resource defined above, the name "sam" will be used as the username for the user chef-client will create.

Resource attributes and the associated **values** describe the desired state of the resource. Each resource has different attributes that are meaningful in the context of the given resource. The "action" attribute is an attribute used by all resources and defines what ought to be done with the resource. In our example above, we used the "create" action to indicate we want the user to be created. When sam leaves our organization, we may change this action to ":delete" to remove her user.

The available resource attributes and their default values are described in detail for each resource on the Resources page.

When chef-client loads the recipes in its run list,

1. It runs each recipe as a piece of ruby code.
2. Each time a resource is used, it is added to a collection of resources for that node.
3. The providers of these resources then decide whether or not action is needed to fulfill the resources' description.

In this way, a core principal of Chef is achieved: idempotency. **Idempotency** ensure that a resource can be applied to a machine multiple times and always have the same result: a properly configured machine.

Attributes and Templates

Attributes are data about a Node and able to be dynamically loaded

Attributes provide a nested key-value store of data about a node and its configuration. Some attributes are collected automatically at the start of each chef-client run and include information such as the nodes IP address, hostname, or loaded kernel modules. Other attributes are added from other sources such as cookbooks. (Roles and Environments can also set attributes, two features of Chef that are talked about in other sections.)

Attributes allow you to change configuration based on the characteristics of a node and set sane defaults for service configurations while also allowing them to be easily changed. One way to set attributes is from an attribute files within a cookbook.

For example the following sets some sane defaults for the location of a particular configuration file:

An attribute file

```
default["my_application"]["config_location"] = "/etc/myapp.conf"
```

This attribute could then be used directly in a resource controlling the configuration file:

```
template node["my_application"]["config_location"] do
         action :create
end
```

This defines a Templates, which is a feature of Chef that allows you to create general files whose contents can be generated dynamically. If this configuration file was referenced multiple

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

A template is a configuration file allowing dynamic generation of file contents based on variables or more complex logic

times in a recipe, changing its location is now a matter of changing a single attribute.

Node attributes also contain information about the node that can be used in recipes:

```
if node.attribute?("ec2")
   # Do EC2 specific configuration tasks.
end
```

This would allow a recipe to do some set of configuration tasks only if it was running on Amazon's EC2 cloud. For complete documentation on Attributes see Attributes.

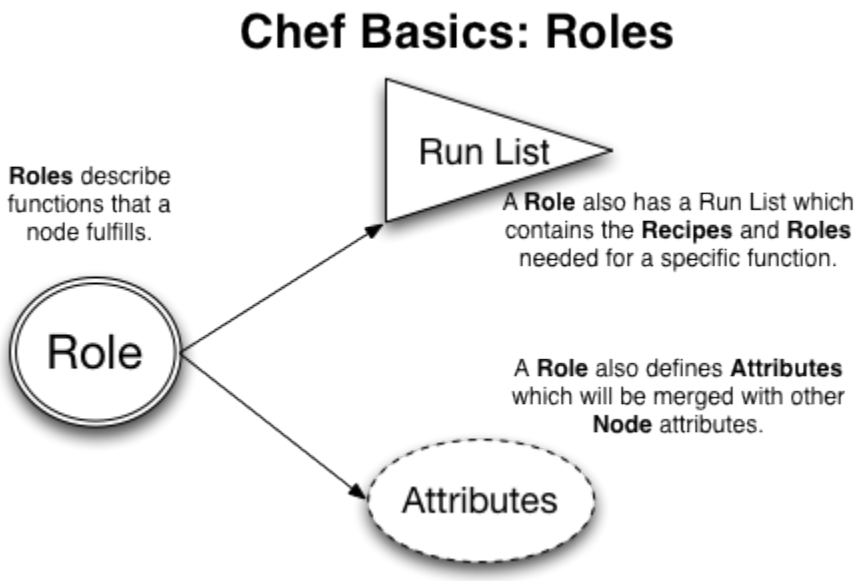
Roles

At web scale, you almost never have just one of something

Configuring a single node may involve many different cookbooks. In order to group similiar features of similar nodes, Chef provides Roles. Roles contain a run list (just like a node) and attributes pertaining to a specific function. For example, you may create a "webserver" role that would contain a run list with all of the services needed on a typical webserver within your infrastructure. A node's run list can contain these roles.

When a node's run list contains a role, the run list from the role itself is expanded and added to the node's run list, ensuring that all of the recipes for a given role are applied to the node.

Roles can also set attributes. Setting attributes in a roles allows you to override the default attribute of a general-purpose cookbook with values more appropriate for a node with a particular role.



Environments

Manage multiple spaces with one Chef instance

Environments in Chef provide a mechanism for managing different environments such as production, staging, development, and testing, etc within one Chef Chef Server or Hosted Chef organization. With environments, you can specify per environment run lists in roles, per environment cookbook version constraints, and environment attributes. Roles differ in that they are groupings of systems that perform a like function and which *can span environments* - for example: application server, web server, database server. Thus you can have per-environment run lists within roles - for example: in your development environment you want your application server to connect to a different load balancer than you do in your test environment, and this can be managed within a single recipe.

You can create and manage environments in multiple ways, thereby tailoring their use to your needs and infrastructure. See Environments for more details.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Summary

- Cookbooks contain recipes, attribute files, and other configuration information.
- Resources are the building blocks of recipes and describe a discrete piece of a Node's configuration.
- Resources are idempotent. Applying the same resource to a node twice should have the same result.
- Providers take the necessary actions required to ensure the Node's state matches the resource description.
- Attributes provide tunable parameters that can be used within a recipe as well as information about the node.
- Roles provide a way to describe a particular function or type of node. Roles have run_lists and attributes.
- Environments provide the means of managing different infrastructure spaces within one Chef instance.

Onward!

Now on to two of Chef's most Powerful Features:

- Next Steps: Introduction to Search and Data Bags

Advanced Reading

- Cookbooks
- Resources
- Providers
- Attributes
- Roles
- Templates
- Environments

 Core Components

Introduction to Search and Data Bags 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Introduction to Search and Data Bags

The article introduces the basics of search and data bags.

These are two of Chef's most powerful features and allow you to dynamically change the configuration of your infrastructure based on data.

Search



Search is a feature of the Chef Server that allows you to use a full-text search engine (based on Apache Solr) to query information about your infrastructure and applications. Information contained in Node attributes, data bags (see below), environments, roles, and API clients are all searchable.

Search can be used both within recipes and from the Chef Workstations using knife. For example, in a recipe, I might want a list of all my application servers so that I can add them to my load-balancers configuration. To do this, I might search for all of the nodes that are using the role "appserver":

```
app_servers = search(:node, 'role:appserver')
```

Note that the results of this search will likely not be static. Rather, each time chef-client runs, the results will reflect the appservers within your infrastructure at that time. *This gives you the power to scale infrastruture and ensure that new components will be properly integrated with existing components such as a load-balancer.*

Alternatively, you might want to quickly list the nodes serving as your application servers from your workstation:

```
knife search node 'role:appserver'
```

For more information on search see Search.

Data Bag



Data Bags store nested key–value data (like attributes) on the chef server. Data Bag data are searchable, and can also be loaded directly by name in a recipe. **Data Bags are global for your chef-server installation–you can think of them as attributes for your whole infrastructure.**

For example, a given node many need many user accounts. Rather than writing a new user resource description for each of those users, we could store information about each user in a data bag and simplify our recipe:

```
search(:users, '*:*') do |u|
  user u["username"] do
    home u["home"]
    shell u["shell"]
    comment u["comment"]
  end
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Now, whenever we want to add a user to our nodes, we can do so by editing the data bag rather than the recipe. *Not only does this simplify our recipes, but it ensures that important data about our infrastructure is now available over search.* Having this data be searchable means that we can quickly answer questions we might have about our infrastructure, using knife. For instance, say we wanted to know the default shell the user "hank" is using, we could do the following using knife:

```
knife search users 'username:hank' -a shell
```

For data bags containing sensitive data, Chef provides **Encrypted Data Bags** which allows you to securely store information on the Chef Server in an encrypted form. For more information about data bags and encrypted data bags see Data Bags and Encrypted Data Bags.

Summary

- Search allows you to query information about your infrastructure from within a recipe or via knife.
- Data bags allow you to store data in a searchable fashion. This data is then searchable and can be used within recipes.

Allons-y!

Now that you've read about the basic concepts of Chef, you may want to check out Guide to Creating A Cookbook and Writing A Recipe for a guided tour of writing your first cookbook or Fast Start Guide to quickly set up your Chef Workstation and a Node using Hosted Chef as your Chef Server.

Advanced Reading

- Search
- Data Bags
- Encrypted Data Bags

 Introduction to Cookbooks and More

Architecture 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Architecture

Clients

Chef includes several clients with the distribution:

- **Chef Solo:** is a client application that works entirely from on-disk data, and is a light-weight alternative to a full client-server configuration.
- **Chef Client:** is the client application that works with a Chef Server to persist data and download cookbooks. Chef Clients can also take advantage of Chef Server's search abilities to dynamically integrate your applications with the rest of your infrastructure.
- **Knife:** is the command line interface to Chef. Knife is primarily used to interact with the Chef Server API, and it can be used for local Chef Repository maintenance.
- **Shef:** is the interactive Chef shell. Shef allows you to write, run, and debug recipes interactively and also provides a programmatic interface for viewing and editing data on your Chef Server.

Server Architecture

The Chef Server that the Chef Client interacts with is either:

Hosted Chef

- Hosted Chef customers do not need to manage their Chef Server, as that is done for them by Opscode.
- If you are a Hosted Chef Customer and have a question, please contact support@opscode.com.

Private Chef

- Private Chef customers should refer to their separate Private Chef Administration Documention for the Private Chef Server.
- If you are a Private Chef Customer and have a question, please contact support@opscode.com.

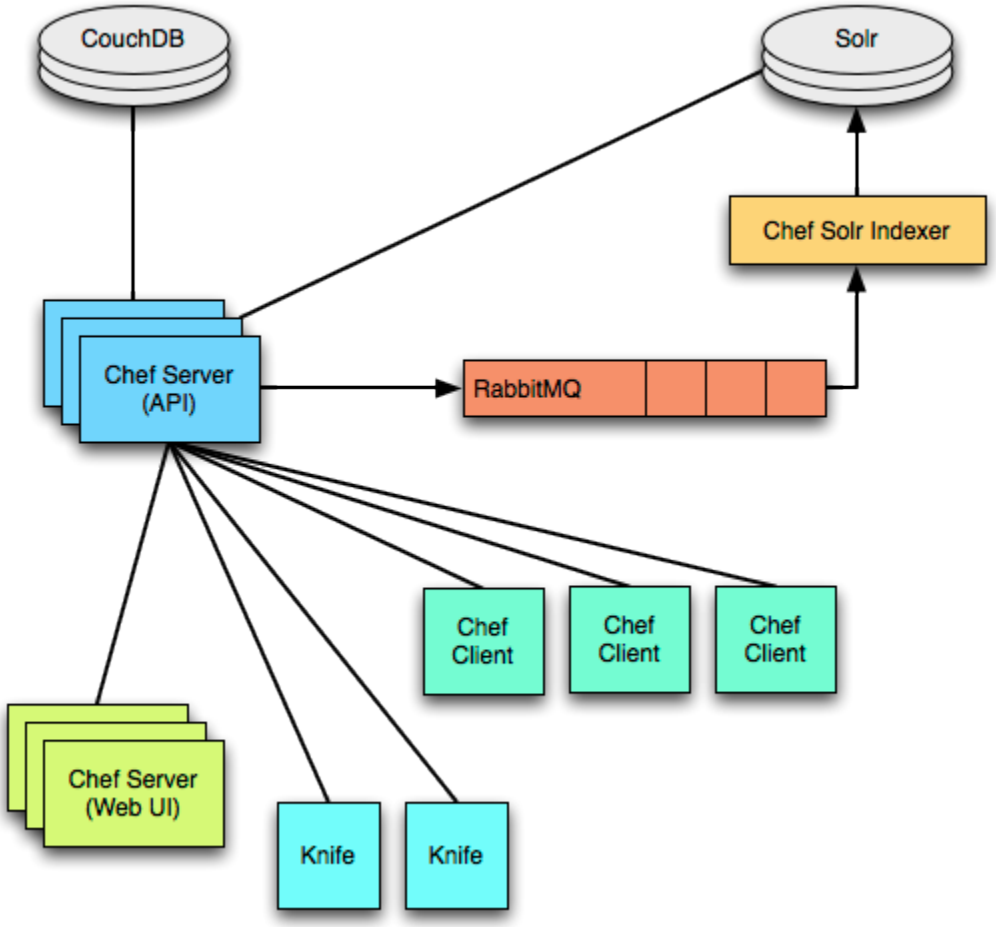
Open Source Chef Server

- An Open Source Chef Server is comprised of the following components:
  - **Chef Server:** services HTTP API requests from the Web UI, nodes, and other clients (see above).
  - **Chef Server Web UI:** the

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

web-based management console for the Chef Server. It manages your infrastructure by making API calls to Chef Server.

- **CouchDB:** the primary data store for a Chef server.
- **RabbitMQ:** stores and then forwards data from Chef Server to the Chef Solr Indexer. It acts as a buffer for cases when high write loads temporarily exceed the ability of the Chef Solr Indexer to update the search index.
- **Chef Solr Indexer:** flattens and expands data to enhance searchability, then writes the data to Chef Solr.
- **Chef Solr:** a thin wrapper around the Apache Solr search engine. Chef Solr allows you to find your way around your infrastructure by querying its metadata.



Further Detail

- Authentication and Authorization
- Anatomy of a Chef Run
- Chef Client
- Chef Solo
- Chef Server
- Hosted Chef
- Server API
- Cookbook Site API
- Chef Concepts as UML

Introduction to Search and Data Bags

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Authentication and Authorization



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Authentication and Authorization



Overview

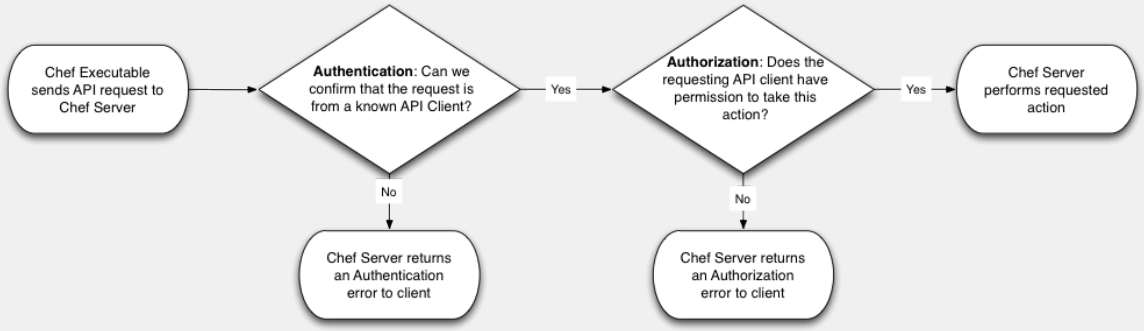
Chef-client and knife
both communicate with the Chef Server using a REST API.

Chef-clients make API requests to retrieve their run list, download their cookbooks, and save their state after a successful run.

When developing cookbooks and managing your infrastructure, you will regularly use Knife commands which make API request to the Chef Server.

Before being processed by the server, each request goes through two processes

Authentication: Verifies the identity of the API client communicating with the Chef Server.	**Authorization: Verifies that the identified client has the permission to perform the requested actions.**



After your initial configuration of nodes and your workstations, these processes are largely transparent. However, understanding both of these processes is often important when debugging potential problems.

Since **Chef-Solo** does not communicate with a Chef Server, authentication and authorization do not play a role in the normal use of chef-solo. Thi s article covers how a given API client's requests are authenticated and authorized. For information about specific errors see Common Errors.

Authentication

Authentication ensures that requests received by the Chef Server are from known API clients

The authentication process in Chef uses public key encryption, and is the same for users of the Open Source Chef Server as well as Hosted Chef and Private Chef.

When an API client is created, private and public encryption keys are created. The public key is stored on the Chef Server, while the private key is returned to the user for safe keeping. The private key is the ".pem" file that you put in your local `.chef` directory or in `/etc/chef`. Chef executables

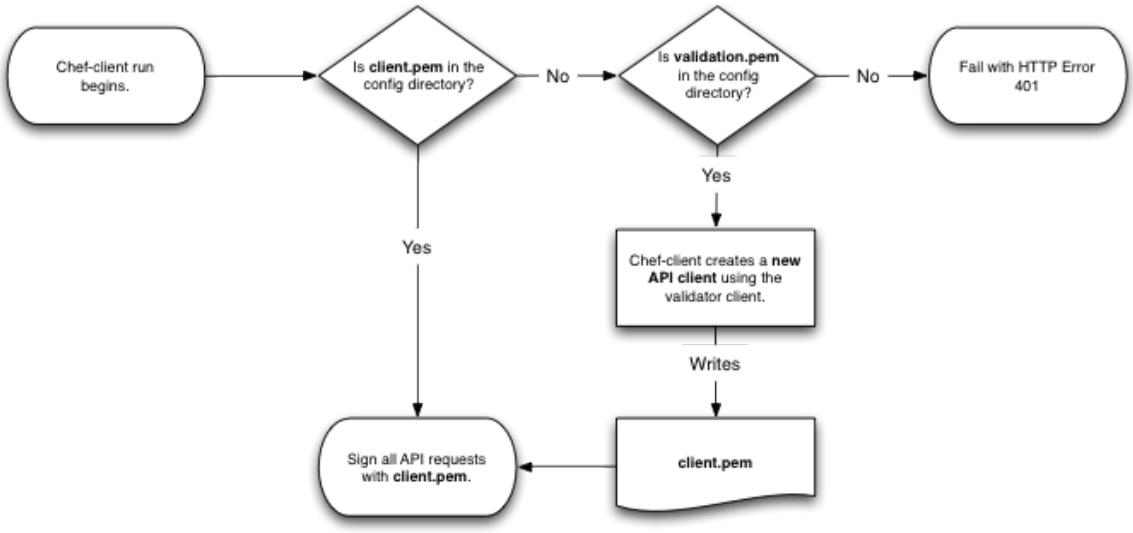
This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

such as `chef-client` and `knife` authenticate requests to the Chef Server by encrypting a specially crafted set of HTTP Headers with the private key. The Chef Server then uses the public key to de-crypt the headers and verifies their content.

This process helps ensure that requests are only accepted from known API clients.

For technical details on how API requests are authentication see Making Authenticated API Requests. To debug common problems that can result in Authorization errors, see Common Errors.

Initial Chef-client run



Since every request from chef-client to the Chef Server must be authenticated, every node must have an API client. To avoid requiring administrators to create new API clients for each node they would like to add to their infrastructure, Chef uses a special **validator client** on the first run of chef-client.

The validator client's key pair is created when you first install an Chef Server or create a Hosted Chef organization and is moved onto clients by the Knife Bootstrap process or manually during the chef-client install process. *This validator client creates a new API client that the node can then use for all future request.*

A check is done to see if a client key exists on the node (typically, `/etc/chef/client.pem`)

- If so, we use it to start signing requests.
- If not, we check if a validation key exists on the node (typically, `/etc/chef/validation.pem`).
  - If so, we request a new API Client Key, signing our request with the validation key.
  - If not, we fail.

If we were issued a new API Client Key, we start using it to sign subsequent requests.

Debugging Authentication Problems

Authentication problems are situations in which chef-client receives HTTP Status 401 in response to its request. If you are receiving HTTP Status 403, the problem is with **Authorization**.

An authentication error error may look like the following:

```
[Wed, 05 Oct 2011 15:43:34 -0700] INFO: HTTP Request Returned 401 Unauthorized: Failed to authenticate
as NODENAME. Ensure that your node_name and client key are correct.
```

To debug authentication problems, the first piece of information to determine is the API client that chef-client is attempting to authenticate as. You can usually see this in the log messages emitted by chef-client. For instance if you turn on debug logging (`chef-client -l debug`) you will see a line such as:

```
[Wed, 05 Oct 2011 22:05:35 +0000] DEBUG: Signing the request as SOMENODENAMEHERE
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

If the authentication is happening with your validator client, the problem is most likely with your validation key.

If the authentication is happening with the nodes API client, there are a number of common causes.

- Your client.pem file is incorrect.
  This can be fixed by deleting client.pem and re-running chef-client. When chef-client runs, it will register the API client and generate the correct key.
- You are trying to authenticate with a node_name that is different from the one you used on your first chef-client run.
  This can happen for a number of reasons. For example, if your client.rb file does not specify your node name and you have recently changed the systems hostname. You can fix this by explicitly setting the node name in the client.rb file or with chef-client's -N option to match the name originally used to register. Alternatively, you can re-register using the method described above.
- Your system clock has drifted from the actual time by more than 15 minutes. This can be fixed by syncing your clock with an NTP server.

WebUI

The WebUI also uses Chef Server's REST API to perform most operations. As such requests from the API are also authenticated. The WebUI authenticates using a keypair that is generated during the installation. Hosted Chef users need not worry about the management of the WebUIs authentication keys, as Opscode handles that for you. Private Chef users WebUI authentication keys are managed by the internal administrator(s) responsible for the Private Chef server.

Authorization

Authorization ensures the API client making a given request has the permission to perform the requested action

Open Source Chef has a single tenant access control model, which is very from the role-based access control (RBAC) model of Hosted Chef and Private Chef.
*See **Hosted Chef Authorization** for the Hosted Chef and Private Chef authorization model.*

Authorization in the Open Source Chef Server is governed by the designation of API clients as non-admin, admin, or validator clients.
Certain functions are available to all clients, others are available to either admin clients or the validator clients, and some are only available to admin clients. Further, API clients are permitted to perform certain actions on themselves or their related node but not on other clients.

Admin clients are created by setting the client's `admin` attribute to false.
This attribute can be set upon client create when using `knife client create` or by using `knife client edit`.

Validator clients are created during the Chef Server installation.

Admin Clients Only
The following API requests can only be made by admin clients.

Request Description	Knife Equivalent
Client Index	`knife client list`
Client Update	`knife client edit NAME`
Client Destroy	`knife client delete NAME`
Cookbook Update	`knife cookbook upload COOKBOOK`
Cookbook Destroy	`knife cookbook delete COOKBOOK`
Data Bag Create	`knife data bag create BAG_NAME`
Data Bag Destroy	`knife data bag delete BAG_NAME`
Data Bag Item Create	`knife data bag create BAG_NAME ITEM_NAME`
Data Bag Item Update	`knife data bag edit BAG_NAME ITEM_NAME`
Data Bag Item Destroy	`knife data bag delete BAG_NAME ITEM_NAME`
Environment Create	`knife environment create ENV_NAME`

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Environment Update	`knife environment edit ENV_NAME`
Environment Destroy	`knife environment delete ENV_NAME`
Role Create	`knife role create ROLE_NAME`
Role Update	`knife role edit ROLE_NAME`
Role Destroy	`knife role delete ROLE_NAME`
Search Reindex	**NA**
User Create	**NA**
User Update	**NA**
User Destroy	**NA**

Admin Clients or Validator Client
The following API requests can only be made by admin clients OR the validator client.

Request Description	Knife Equivalent
Client Create	`knife client create`

Note that only admin clients can create other admin clients.

Admin Clients or Clients Acting on Themselves
The following API requests can only me made by admin clients OR clients that are requesting that the action be taken on themselves or their related node.

Request Description	Knife Equivalent
Client Show	`knife client show NAME`
Node Update	`knife node edit NAME`
Node Destroy	`knife node delete NAME`
Node Cookbooks	**NA**

Any functions not listed in the above tables can be performed by any API client that has made a properly authenticated request.

Summary

Open Source Chef Server has a single tenant access control model.

1. All stored objects (roles, nodes, cookbooks, data bags, search indexes) are in the same virtual space and accessible to all API clients.
2. All API clients are authorized as either "admin" or "normal."
    - "Admin" API clients perform privileged tasks such as create/read/update/delete operations on all nodes.
    - "Normal" API clients act upon the node they create.

The role-based access control (RBAC) model of Hosted and Private Chef is detailed at Hosted Chef Authorization.

 Architecture

Hosted Chef Authorization

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Hosted Chef Authorization

Hosted Chef and Private Chef provide fine-grained control over the actions that Users and API Clients can perform

Unlike Open Source chef, permission to create, update, delete, or view objects (such as nodes) on the Chef Server can be controlled on an object-by-object and user-by-user basis.



Hosted and Private Chef provide the flexibility to control a user's access to Chef object in a manner that is commensurate with their role in your organization.

- The permissions available for each type of Chef object and what those permissions imply,
- The method used to determine whether a user is authorized to take a given action, and
- The permissions users will have on newly created objects within your organization.

For a description of how to manage a given user's or API client's permission see Managing Permissions with the Hosted Chef Management Console.

Definitions

This article uses the following definitions:

Every object on Hosted Chef is secured with fine-grained role-based access control (RBAC)

Each object has an access control list (ACL) composed of up to five access control entries (ACEs): create, read, update, delete, and grant.

- **Actor**: An *actor* is an API Client or Hosted Chef user. It is often easiest to think of Hosted Chef users simply as API clients.
- **Objects**: An *object* is an entity on which an *actor* can have permission to take action. For example, a node is an object.
- **Object Type:** Every object has a *type* (node, role, cookbook, etc).
- **Group**: A *group* is a collection of actors. Hosted Chef defines an **admins**, **users**, **clients**, and **billing-admins** group for every organization. Additional groups can be added as a means of grouping users based on their role within your organization. See Managing Users and your Private Key with the Hosted Chef Management Console for further details.

When an API request is made to Hosted Chef, a check occurs to ensure that the requesting **actor** has the necessary permission on either the **object** or **object type**.

Global Permissions

Actors and *groups* can have the following *Global Permissions* on a given *Object Type*:

- **LIST (READ)**: An actor with LIST permission on an object type can list all of the objects of the given type. This corresponds to the ability to run commands such as `knife node list`.
- **CREATE**: An actor with CREATE permission on an object type can create new objects of the given type. This corresponds to the ability to run commands such as `knife node create`.

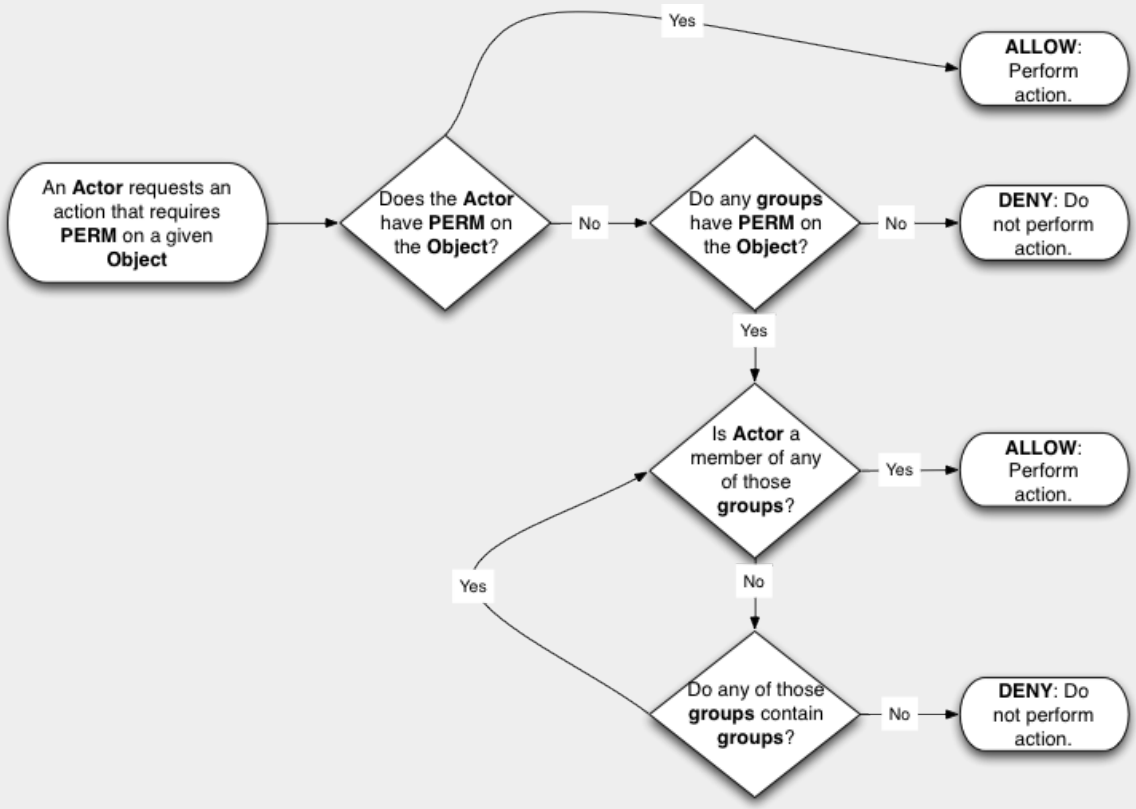
Object Specif

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

ic Permissions

Actors and *groups* can have the following permissions on a given *object*:

Note that this graph focuses on the case when permission is needed on an **Object**, but the same process applies to cases where permission is needed on an **Object Type**.



- **READ**: An actor with **READ** permission on an object can read the object's state. This corresponds to the ability to run commands such as `knife node show NODENAME`.
- **UPDATE**: An actor with **UPDATE** permission on an object can update the object's state. This corresponds to the ability to run commands such as `knife node edit NODENAME` and the ability for an API client to save node data at the end of a chef-client run.
- **DELETE**: An actor with **DELETE** permission on an object can delete the object. This corresponds to the ability to run commands such as `knife node delete NODENAME`.
- **GRANT**: An actor with **GRANT** permission on an object can grant or remove other actors' permissions on the object. This corresponds to the ability to edit permissions on the object's "Permissions" tab within the Management Console. See Managing Permissions with the Hosted Chef Management Console for more details.

Checking if an Actor is Authorized

When an actor makes an authenticated request via the REST API, Hosted Chef determines whether the actor has permission to perform the action using the following process:

- If the request requires **LIST** or **CREATE** permission, check if the user has **LIST** or **CREATE** permission for the object type.
  - If Yes, perform the action
  - If Not, recursively check whether the user is a member of a group which has **LIST** or **CREATE** permission for the object.
- If the request requires **READ**, **UPDATE**, **DELETE**, or **GRANT** permission, check if the user has the **READ**, **UPDATE**, **DELETE**, or **GRANT** permission for the object.
  - If Yes, perform the action
  - If Not, recursively check whether the user is a member of a group has the **READ**, **UPDATE**, **DELETE**, or **GRANT** permission for the object.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

For information on how to determine what permissions a given actor has on an object, changing an actors permissions, or on what permissions an actor will have by default, see Managing Permissions with the Hosted Chef Management Console

Default Permissions

In most cases, the required permission for a given option is fairly intuitive. Actions that retrieve an object's state requires READ, for instance. Deleting objects require DELETE. However, what permissions is a given actor likely to have?

Typically, actors will receive permission for a given object or object type via their group memberships. To facilitate easily managing permissions, all Hosted Chef organizations have four groups by default:

- **Users**: This group has the default permissions that make the most sense for non-admin Hosted Chef users. Hosted Chef users that you add to your organization are automatically added to this group.
- **Admins**: This group has the default permission for all-powerful admins.
- **Clients**: This group has the default permissions that make the most sense for non-human API clients. Newly created clients are put into this group automatically.
- **Billing-admins**: This group has permission to change billing information. See Managing your Account and Billing Information.

Placing users into the Users and/or Admins groups provides an easy way to grant permissions to a given user without much hassle. See Managing Users and your Private Key with the Hosted Chef Management Console to add users to groups.

Default Global Permissions

There are two ways an actor can receive LIST or CREATE permission on an object type:

- Manually granting the actor the global permission in the WebUI (see Managing Permissions with the Hosted Chef Management Console)
- Being a member of a group that has the given permission.

The default groups have the following global permissions by default:

- The **Admins** group has **LIST** and **CREATE** on every object type.
- The **Users** group has **LIST** on
    1. Clients
    2. Groups
    3. Nodes
    4. Roles
    5. Cookbook
    6. Databags
    7. Environments
- The **Users** group has **CREATE** on
    1. Nodes
    2. Roles
    3. Cookbook
    4. Databags
    5. Environments
- The **Clients** group has **LIST** on
    1. Nodes
    2. Roles
    3. Cookbook
    4. Databags
    5. Environments
- The **Clients** group has **CREATE** on
    1. Nodes
    2. Databags

Default Object-specific Permissions

There are three ways an actor can receive permissions on objects:

- If the actor created the object, she receive all

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

the permissions on the new object.

- If the actor is manually granted the actor permissions on the object in the WebUI (see Managing Permissions with the Hosted Chef Management Console).
- "Inheriting" the permissions when the object is created, based on a group or actor's Global permission on the Object Type.

To determine the permissions an actor will have on a new object, consider the following rules:

1. If the actor is a member of the **admins** group, she will receive all of the permissions on every new object.
2. If the actor is a member of the **users** group, she will receive the following permissions on new objects:

Object Type	READ	UPDATE	DELETE	GRANT
Environments	X	X	X	
Nodes	X	X	X	
Roles	X	X	X	
Cookbooks	X	X	X	
Data Bags	X	X	X	
Clients	X		X	

3. If an actor or a group has the global permission LIST on an Object Type, it will receive READ on any new objects of that type.

Think In Terms Of Group Membership

Technically, when you create a new object of a given type, the permissions list from the object type (also called the object container) is copied to the new object. Thus creating the "inheritance".

It is therefore possible for an object to have the CREATE, READ(LIST), UPDATE, DELETE, and GRANT permissions on the object type, just as with an object.

This is how the admin, users, and client groups receive permission on newly created objects.

Since only a subset of these permissions (LIST and CREATE) can be altered via the WebUI, it is typically easier for those managing a Hosted Chef organization to think about the permissions system in terms of group membership.

Summary

Every object on Hosted Chef is secured with fine-grained role-based access control (RBAC).

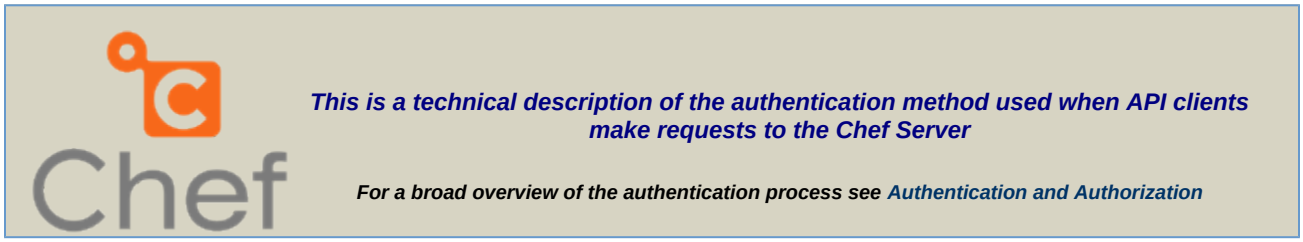
- Each object has an access control list (ACL) composed of up to five access control entries (ACEs): create, read, update, delete, and grant.
- Before allowing an authenticated user or client to perform an operation on an object, the appropriate ACE is consulted to determine whether the user is either explicitly in the ACE or in a group that is on the ACE (or in a group in a group on the ACE, etc.).

Authentication and Authorization

Making Authenticated API Requests

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Making Authenticated API Requests



Protocol Overview

Authentication occurs by constructing a specific set of HTTP headers, and then signing a subset of those headers and information about the request using the API client's private key. The server verifies the signature using the public key of the specified API client.

Required Authentication Headers

Every API request must contain the following headers:

- **X-Ops-Timestamp**: A timestamp in ISO-8601 format. The time must be in UTC, indicated by a trailing Z, and separated by the character "T". The following is an example of a time in this format:

```
2011-10-14T18:17:48Z
```

- **X-Ops-UserId**: The name of the API client whose private key will be used to create the authorization header.
- **X-Ops-Content-Hash**: The body of the request, hashed using SHA1 and encoded using Base64. See Hashing below.
- **X-Ops-Sign**: Set to "version=1.0"
- **X-Ops-Authorization-N**: One or more headers whose value are the signature of the "canonical headers", encoded in Base64. See **Canonical Headers** and **X-Ops-Authorization Headers** below.

API Calls against Hosted Chef

API calls against Hosted Chef must also include the header **X-Chef-Version** which should be set to the version of the Chef executable making the API call. Practically, this controls whether you will receive API responses in the form expected by 0.9.x or 0.10.x.

Note, that these are the headers required for authentication. Other headers such as "Accept" should also be included in your request.

Canonical Headers

The **Canonical Header** is a string in the following format:

```
"Method:HTTP_METHOD\nHashed
Path:HASHED_PATH\nX-Ops-Content-Hash:HASHED_BODY\nX-Ops-Timestamp:TIME\nX-Ops-UserId:USERID"
```

HTTP_METHOD is the method being used in the API request (e.g., GET or POST) and must be capitalized. HASHED_PATH is the path of the request (e.g., "/clients") hashed with SHA1 and encoded using Base64 (see **Hashing**). The path **must not** have repeated "/"s and **must not** end in a "/", unless the path is "/".

Hashing

All Hashing is done using SHA1 and encoded in Base64. The Base64 encoding should have line breaks every 60 characters.

X-Ops-Authorization Headers

To create the **Authorization Headers**, the canonical header is signed with the API client's private key and encoded using Base64. The private

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

key must be an RSA key in SSL's PEM format. This signature is then broken in into 60 character strings and placed in the headers **X-Ops-Authorization-1** through **X-Ops-Authorization-N** where N is the last header needed to send the entire string.

The Chef server decrypts the X-Ops-Authorization headers and ensures their content matches the content of the non-encrypted headers that were sent. Further, the timestamp of the message is checked to ensure that the request has been received within a reasonable amount of time.

Within Chef, this authorization method is implemented by mixlib-authentication.

Making API Requests in Ruby



From a Knife Plugin or Knife Exec script

When creating a Knife Plugin or Knife Exec script, the libraries required to make authenticated API requests have already been included. In the most cases, one will not need to make API calls directly when using knife plugins or knife exec scripts, as it is easier to interact with nodes, clients, and other objects via other means. See the **Knife Plugins** and **Knife Exec** page for details.

In a Knife Exec script (and Shef), the `api` object already includes all of the necessary configuration to make API calls using the following methods:

```
api.get
api.post
api.put
api.delete
```

API Request from a Knife Exec Script, Example 1

```
knife exec -E 'puts api.get("/nodes/nodename")'
```

API Request from a Knife Exec Script, Example 2

```
client_desc = {
  "name"  => "monkeypants",
  "admin" => false
}
new_client = api.post("/clients", client_desc)
puts new_client["private_key"]
```

Moreover, whenever possible, `api` will return an object of the relevant type, on which you can call methods.

API Request from a Knife Exec Script, Example 2

```
# We could also just call api.delete, but that wouldn't show
# that api returns a node object
silly_node = api.get("/nodes/foobar")
silly_node.destroy
```

In a Knife Plugin, the `rest` object is similar to the `api` client. The only difference is the function names:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
rest.get_rest
rest.put_rest
rest.post_rest
rest.delete_rest
```

API Request from a Knife Plugin

```
module MyCommands
  class MyNodeDelete < Chef::Knife
    #An implementation of knife node delete
    banner 'knife my node delete [NODE_NAME]'

    def run
      if name_args.length < 1
        show_usage
        ui.fatal("You must specify a node name.")
        exit 1
      end
      nodename = name_args[0]
      api_endpoint = "nodes/#{nodename}"
      # Again, we could just call rest.delete_rest
      nodey = rest.get_rest(api_endpoint)
      ui.confirm("Do you really want to delete #{nodey}")
      nodey.destroy
    end
  end
end
```

Note that knife will handle any HTTP exceptions that arise within your knife plugin or exec script.

Arbitrary Ruby Scripts

On a system with Chef installed, the easiest way to make authenticated API request is to use the chef libraries.

```
require 'rubygems'
require 'chef/config'
require 'chef/log'
require 'chef/rest'

chef_server_url="https://chefserver.com"
client_name = "cleintname"
signing_key_filename="/path/to/pem/for/clientname"

rest = Chef::REST.new(chef_server_url, client_name, signing_key_filename)
puts rest.get_rest("/clients")
```

Alternatively, you can use mixlib-authentication directly.

Making API Requests in Python

As with Knife Plugins and Knife Exec scripts, you will typically not want to make API calls directly, but use other facilities provided by the library.

For example, the following is the example provided in PyChef's README:

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Python API for interacting with Chef Server

Opscode team member Noah Kantrowitz has constructed **PyChef**: a Python library satisfying the Mixlib::Authentication pre-requisites, for interacting with the Chef Server.

```
from chef import autoconfigure, Node

api = autoconfigure()
n = Node('web1')
print n['fqdn']
n['myapp']['version'] = '1.0'
n.save()
```

However, it is also possible to make API calls directly:

```
from chef import autoconfigure

api = autoconfigure()
print api.api_request('GET', '/clients')
```

Note: Both of these examples assume that your current working directory is such that PyChef can find a valid chef configuration file in the same manner chef-client or knife would find its configuration.

Making API Requests with Curl

It is also possible to make authenticated API requests using `curl` or `wget` provided you have various common tools installed.

The following is an example of how one can create an authenticated request using curl. It is a Bash shell script which requires the following utilities: awk, openssl

```
_chef_dir () {
    # Helper function:
    # Recursive function that searches for chef configuration directory
    # It looks upward from the cwd until it hits /.  If no directory is found,
    # ~/.chef is chosen if it exists
    # You could simply hard-code the path below

    if [ "$PWD" = "/" ]; then
        if [ -d ".chef" ]; then
            echo "/.chef"
        elif [ -d "$HOME/.chef" ]; then
            echo "$HOME/.chef"
        fi
        return
    fi

    if [ -d '.chef' ];then
        echo "${PWD}/.chef"
    else
        (cd ..; _chef_dir)
    fi
}


_chomp () {
    # helper function to remove newlines
    awk '{printf "%s", $0}'
}
```


This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
chef_api_request() {
    # This is the meat-and-potatoes, or rice-and-vegetables, your preference really.

    local method path body timestamp chef_server_url client_name hashed_body hashed_path
    local canonical_request headers auth_headers

    chef_server_url="https://yourchefserver.com"
    client_name="client name"
    method=$1
    path=$2
    body=$3

    hashed_path=$(echo -n "$path" | openssl dgst -sha1 -binary | openssl enc -base64)
    hashed_body=$(echo -n "$body" | openssl dgst -sha1 -binary | openssl enc -base64)
    timestamp=$(date -u "+%Y-%m-%dT%H:%M:%SZ")

    canonical_request="Method:$method\nHashed
Path:$hashed_path\nX-Ops-Content-Hash:$hashed_body\nX-Ops-Timestamp:$timestamp\nX-Ops-UserId:$client_
name"
    headers="-H X-Ops-Timestamp:$timestamp -H X-Ops-Userid:$client_name -H X-Chef-Version:0.10.4 -H
Accept:application/json -H X-Ops-Content-Hash:$hashed_body -H X-Ops-Sign:version=1.0"

    auth_headers=$(printf "$canonical_request" | openssl rsautl -sign -inkey
"$(_chef_dir)/${client_name}.pem" | openssl enc -base64 | _chomp |  awk '{ll=int(length/60);i=0; while
(i<=ll) {printf " -H X-Ops-Authorization-%s:%s", i+1, substr($0,i*60+1,60);i=i+1}}')

    case $method in
        GET)
            curl_command="curl $headers $auth_headers ${chef_server_url}$path"
            $curl_command
            ;;
        *)
            echo "Unknown Method. I only know: GET" >&2
            return 1
            ;;
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
    esac
}
```

After sourcing this shell script into your local shell, you might use it as follows:

```
chef_api_request GET "/clients"
```

 Hosted Chef Authorization

API Clients 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

API Clients



Overview

This page provides an overview of the identity through which requests to the Chef server are authenticated through the Server API, how the clients become registered with a valid identity, and how the API clients can be managed. See Authentication and Authorization for details on the authentication and validation process itself.

Authentication Identity

In chef, node data is separate from the identity used to authenticate requests to the Chef server, which is managed by an API client. Each API client has a public/private key pair. The public half of the key is stored in the database on the server, and the private half is kept locally by the client.

- On a node running the Chef Client, the private key is generally stored in /etc/chef/client.pem
- For an administrator using knife, the private key is generally stored in ~/.chef/USERNAME.pem

Each request to the Chef server includes a request signature in the HTTP headers. The request signature is computed from the hash of the request content and encrypted with the client's private key to verify the identity of the user or machine making the request and prevent attempts to tamper with the content.

Client Registration

When a node runs chef-client for the first time, it generally does not yet have an API client identity, and so cannot make authenticated requests to the server. This is where the validation client—named "chef-validator" by default comes in.

1. When the chef-client runs, it checks if it has a client key.
2. If the client key does not exist, it then attempts to "borrow" the validation client's identity to register itself with the server.
3. In order to do that, the validation client's private key needs to be copied to the host and placed in /etc/chef/validation.pem.

Once the client machine has registered itself with the chef server, it no longer uses the validation client for anything. It is recommended that you delete the validation client's private key from the host after the host has registered.

API clients can be managed with Knife, the Server API or the Management Console.

Managing API Clients With Knife

Knife is Chef's powerful command line tool, and can be used to create, edit, view, list, and delete API clients. Details on this use are at Managing API Clients With Knife.

Managing API Clients through the Management Console

- Open Source Chef Server users, please refer to the Managing API Clients through the Management Console article for information.
- Hosted Chef customers please refer to Managing Clients with the Hosted Chef Management Console.
- Private Chef customers should refer to the separate Private Chef Administration Documentation.

Managing API Clients through the Server API

Please refer to the Server API article for information about managing API clients through the REST API.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Making Authenticated API Requests

Anatomy of a Chef Run



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Anatomy of a Chef Run

As you work with Chef, it will be important to understand the path we take during execution of the Chef Client.

Convergence

We call the process of running the Chef Client or Chef Solo and taking any needed actions on Resources "Convergence". In a nutshell, Convergence in systems automation speak means "bringing the system closer to correct with each action you take". Ideally, a single run of the Chef Client should always bring the system into the fully expected state - not just half-way. But if it doesn't work out, trying again should continue to bring the system closer to correct.

Build, Register, and Authenticate the Node

Build the Node

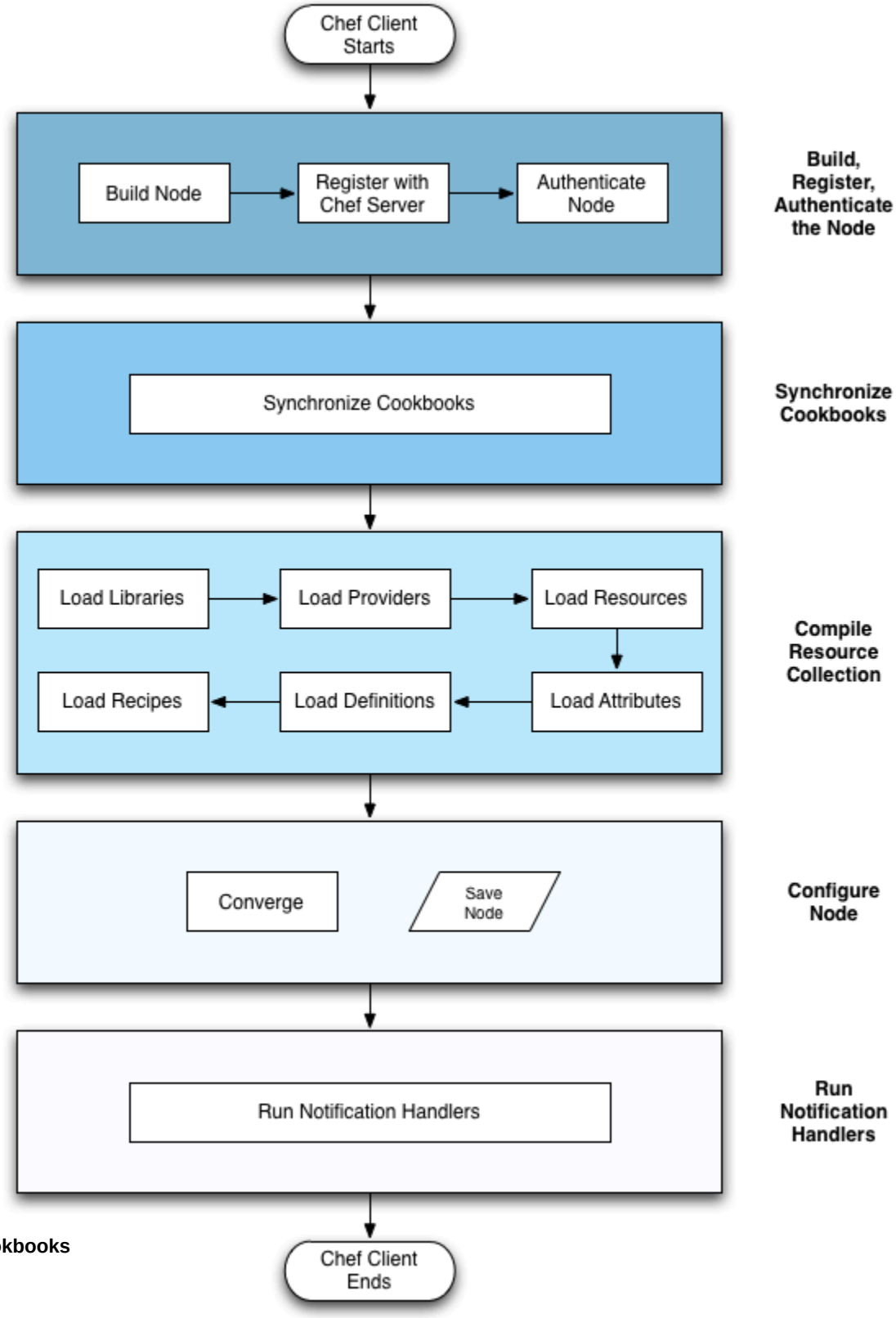
The first thing Chef Client or Chef Solo do is build a new Node. The node is constructed from:

1. a. Ohai - Ohai discovers the data about the Operating System first.
   b. The previous data for

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

the node is fetched from the Chef Server (unless we are running under Chef Solo).

c. Any JSON Attributes or Recipes are added.
d. All the Ohai attributes are added.



Registering With The Chef Server

Once the initial node is constructed, the Chef Client checks for the existence of its private key file, usually located in `/etc/chef/client.pem`. If the private key does not exist, the client will attempt to register itself. To register itself, the client temporarily assumes the identity of the *chef-validator* client. *chef-validator* is a special purpose client used exclusively for registering new clients. By default, the private key for *chef-validator* is stored in `/etc/chef/validation.pem`. Using the identity of the *chef-validator* chef-client (the application) will register a new client identity with the server and store its private key in `/etc/chef/client.pem`. From this point forward, chef-client will authenticate with the server using its own identity.

Synchronize Cookbooks

Synchronize Libraries, Attributes,

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Definitions and Recipes

The Chef Client then queries the Chef Server for a list of all the Libraries, Attributes, Definitions and Recipes in all Cookbooks, and transfers them to the local file cache.

Compile - Resource Collection

Now that we have all information from the Cookbooks, it's time to assemble the specific collection of resources needed to converge this node.

Load Libraries

We first load all the Libraries from every cookbook, making any language extensions or ruby classes available.

Load Attributes

Then we load all the Attribute files, which update the Node attributes and Recipes.

Load Definitions

Definitions must be loaded before Recipes, since they create new pseudo-Resources.

Load Recipes

At this point, the Recipes themselves are evaluated. We are not taking any action on the resources in the recipes at this stage - we are taking each evaluated resource and putting it in the **Resource Collection**.

- This is essentially an Array of each evaluated resource, along with some helpful functions.
- Plain Ruby code outside of resources is evaluated, however.
- If you would like Ruby code executed with other resources, use a Ruby Block Resource.
- **You can have Chef Evaluate and Run Resources at Compile Time, too.**

Execute - Configure Node

Now Chef is ready to configure the system.

Converge

This is the actual convergence step. Each Resource in the Resource Collection is mapped to a Provider, which then takes Action on it. The system is configured.

Save Node

After converging, Chef saves the state of the node to persist its node data and make it available for search.

Run Notifications

Finally, any notification handlers you've configured will be run.

If Something Went Wrong...

When chef-client fails, it finishes the run by executing any exception handlers you've configured.

API Clients

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License


Evaluate and Run Resources at Compile Time


 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Evaluate and Run Resources at Compile Time



Chef processes recipes in two phases, Compile and Execute.

1. During the compile phase, the recipes are evaluated as Ruby code and recognized resources are added to the resource collection. 2. During the execute phase, Chef takes the appropriate Provider action on each resource.

This is the normal way Chef runs, but sometimes you want to make sure that a resource is configured before anything else.

You do this by setting up a `Chef::Resource` object of the appropriate resource type, and then calling the method that runs the action.

Example One: Update Package Cache

It is important to make sure that your operating system's package cache is up to date before Chef tries to install packages, otherwise it might have stale references to versions that don't exist anymore. For example, on Debian and Ubuntu systems, the APT cache needs to be updated.

```
e = execute "apt-get update" do
  action :nothing
end

e.run_action(:run)
```

This creates e as a "`Chef::Resource::Execute`" Ruby object. The action is set to nothing because we're going to call the `run_action` meth od to tell Chef to run the specified command.

Opscode provides a cookbook for doing this with apt (Debian/Ubuntu) and pacman (ArchLinux). Put the recipe at the top of your node's run list to ensure it happens before Chef tries to install your packages.

Example Two: Installing a RubyGem to use later

One of the goals of Chef is that a single run should completely configure a system. One of the features of Chef is that it uses Ruby as the recipe language, which means you can do anything in Recipes that you can do in Ruby.

However, you might need to install a RubyGem first. For example, if you would like to interact with a MySQL database in a recipe directly with Ruby, you can do that by using the Compile phase resource creation.

```
g = gem_package "mysql" do
  action :nothing
end

g.run_action(:install)

Gem.clear_paths
require 'mysql'
```

Just like the earlier example, we're creating a new Ruby object. This time, "`Chef::Resource::Package`" using the Rubygems provider. Do note that in this case the `mysql` RubyGem compiles native extensions in C, so you'll need to have the appropriate packages for your OS installed.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

We introduce two new lines at the end here. The first, `Gem.clear_paths` ensures that Chef reloads the cache of available Rubygems. The second loads the mysql gem so it can be used to connect to a MySQL database.



Anatomy of a Chef Run

Chef Client



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Client



Clients do work for one or many Nodes

A Chef Client does work on behalf of a Node. A single Chef Client can run recipes for multiple Nodes.

Clients do all the work

Clients are where all the action happens - the Chef Server and Chef Indexer are largely services that exist only to provide the Client with information.

You can run the client periodically

If you want, you can run the Chef Client as a persistent daemon. To do this, make your startup script for the chef client execute something like:

Chef Client Interval and Splay

```
$ chef-client -i 3600 -s 600
```

The `-i` option provides an *Interval* - it's how often the Chef client will attempt to wake up and Converge this Node. The `-s` option is the *Splay* - a random piece of time added to the interval, which helps avoid the thundering herd problem.

Anatomy of a Chef Run

Anatomy of a Chef Run describes the process taken by the Chef Client to configure an individual Node in detail.

Chef-Client Cookbook

The Opscode chef-client cookbook is available for use on systems that should have a `chef-client` daemon running, such as when Knife bootstrap was used to install Chef on a new system.

It includes the interval set at 1800 seconds and the splay set at 20 seconds, as configurable defaults, and additional configurable detail information as detailed on this page.

The entire set of options available can be found with the incantation:

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Client Option List

```
$ chef-client --help
Usage: /usr/bin/chef-client (options)
    -S, --server CHEFSERVERURL       The chef server URL
    -k, --client_key KEY_FILE        Set the client key file location
    -c, --config CONFIG              The configuration file to use
    -d, --daemonize                  Daemonize the process
    -E, --environment ENVIRONMENT    Set the Chef Environment on the node
    -g, --group GROUP                Group to set privilege to
    -i, --interval SECONDS           Run chef-client periodically, in seconds
    -j JSON_ATTRIBS                  Load attributes from a JSON file or URL
        --json-attributes
    -l, --log_level LEVEL            Set the log level (debug, info, warn, error, fatal)
    -L, --logfile LOGLOCATION        Set the log file location, defaults to STDOUT - recommended for
daemonizing
    -N, --node-name NODE_NAME        The node name for this client
        --once                       Cancel any interval or splay options, run chef once and exit
    -P, --pid PIDFILE                Set the PID file location, defaults to /tmp/chef-client.pid
    -s, --splay SECONDS              The splay time for running at intervals, in seconds
    -u, --user USER                  User to set privilege to
    -K, --validation_key KEY_FILE    Set the validation key file location, used for registering new
clients
    -v, --version                    Show chef version
    -h, --help                       Show this message
```

The JSON_ATTRIBS can be a `run_list` and used to specify roles and/or recipes to add to the node.

Chef-Client Configuration Settings

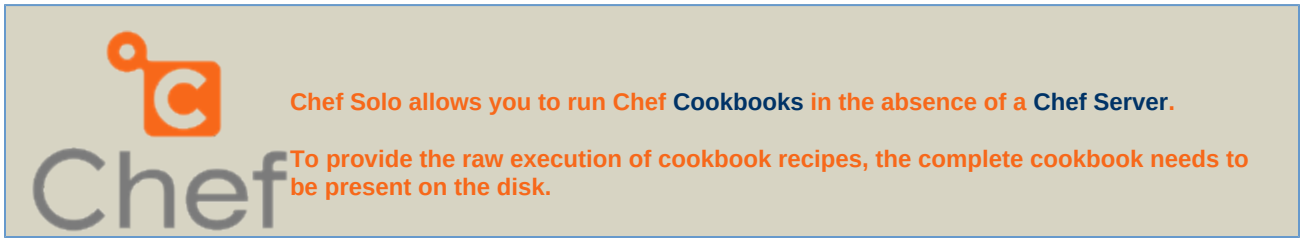
Chef Configuration Settings has detail on the configuration settings available for chef-client, and all the Chef executables.


Evaluate and Run Resources at Compile Time

Chef Solo


 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Solo



Chef Solo does not provide:

- Node data storage or search indexes.
- Centralized cookbook distribution.
- Environments, for setting policy of cookbook versions.
- A central API to interact with and use to integrate infrastructure components.

This document will show you how to set up cookbooks and Chef Solo.

We're going to look at three of the options for `chef-solo`

```
-c, --config CONFIG
-j, --json-attributes JSON_ATTRIBS
-r, --recipe-url RECIPE_URL
```

And then we'll look at some examples of actually running `chef-solo`.

Configure Chef Solo

The first option is `-c` or `--config`, which takes a filename as a configuration file. If this option is not specified, by default Chef Solo will look in `/etc/chef/solo.rb` for its configuration. This configuration file has two required variables: `file_cache_path` and `cookbook_path`. For example, in our instructions for Chef Configuration Settings, we use the following `~/solo.rb`:

~/solo.rb

```
file_cache_path "/var/chef-solo"
cookbook_path "/var/chef-solo/cookbooks"
```

For your own systems, you can change this to reflect any directory you like, but you'll need to specify absolute paths and the `cookbook_path` directory should be a subdirectory of the `file_cache_path`.

Please note: as mentioned below, the establishment of a Chef Repository will introduce considerations that you'll need to incorporate when determining your cookbooks directory structure. The result would determine use of the `cookbook_path` variable within `~/solo.rb`.

Multiple Cookbooks Directories

You can also specify `cookbook_path` as an array, passing multiple locations to search for cookbooks.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

~/solo.rb

```
file_cache_path "/var/chef-solo"
cookbook_path ["/var/chef-solo/cookbooks", "/var/chef-solo/site-cookbooks"]
```

Note that the order is significant, and changed in chef 0.8; earlier entries are now overridden by later ones. When you create the cookbooks tarball below, you'll need both directories to extract into the `file_cache_path` directory.

JSON, Attributes and Recipes

Since `chef-solo` doesn't have any interaction with a Chef Server, you'll need to specify node-specifc attributes in a JSON file. This can be located on the target system itself, or it can be stored on a remote server such as S3, or a web server on your network.

Within the JSON file, you'll also specify the recipes that Chef should run in the "run_list". An example JSON file, which sets a resolv.conf:

~/node.json

```
{
  "resolver": {
    "nameservers": [ "10.0.0.1" ],
    "search":"int.example.com"
  },
  "run_list": [ "recipe[resolver]" ]
}
```

(See the opscode/cookbook, resolver)

Preparing your Cookbooks

You can point `chef-solo` at a target directory on the local system, or at a URL containing a tarball archive.

Chef Repository Interaction

If you are establishing a Chef Repository to define a location where cookbooks and other Chef artifacts will be stored and version controlled, please note the the Chef Repository will have considerations that you'll need to incorporate when setting up your chef-solo managed server and establishing the target directory that `chef-solo` is directed towards.

Running from a Directory

Once that you have `chef-solo` configured for a directory, you can store the cookbooks there directly and run `chef-solo` against that directory if that is the approach that you are undertaking.

Running from a URL

Running `chef-solo` against a URL containing a tarball is probably more common than running against a directory, however. First create the cookbooks tarball.

Creating a Chef Solo Tarball

```
tar zcvf chef-solo.tar.gz ./cookbooks
```

If you're using multiple cookbook directories, note that chef-solo expects the tarball you give it to have a directory structure of the form:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
cookbooks/
       |---- cbname1/
              |--attributes/ ... etc
       ...
       |---- cbname2/
              |--attributes/
       ...
```

(See Rake Task "bundling cookbooks".)

Make sure you've modified the cookbook_path variable in solo.rb to include both paths as described above in Configure Chef Solo

Using Multiple Chef Cookbook Directories

```
tar zcvf chef-solo.tar.gz ./cookbooks ./site-cookbooks
```

Then, upload the resulting `chef-solo.tar.gz` to a web server that is accessible by the servers you wish to run `chef-solo` on. We do this for the bootstrap tarball that Configures Chef Server and Clients.

Examples of Running `chef-solo`

Now that you have configured `chef-solo`, prepared JSON data for your system(s), and prepared your Cookbooks, it's time to run it!

Running Solo with URL

```
chef-solo -c ~/solo.rb -j ~/node.json  -r http://www.example.com/chef-solo.tar.gz
```

Remember: Chef Solo is disconnected from a Chef Server

There is no authentication, no authorization, no search indexes, no persistent attributes. It's just the bare ability to execute Cookbooks.

This actually uses the Chef remote_file resource type to retrieve the specified `chef-solo.tar.gz` into the `file_cache_path`, and extracts it to `cookbooks_path`, then runs Chef with those cookbooks. So once this has occurred, you can modify the cookbooks on the local system and rerun Chef with the directory:

Running Solo with Directory

```
chef-solo -c ~/solo.rb -j ~/node.json
```

Simply remove the '`-r URL`', and the configuration file tells Solo where to look.

You can also specify the JSON data via URL.

Running Solo with URL for Cookbooks and JSON

```
chef-solo -c ~/solo.rb -j http://www.example.com/node.json  -r http://www.example.com/chef-solo.tar.gz
```

These two options, `-r` and `-j` correspond to the `recipe_url` and `json_attribs` config file options, respectively.

solo.rb

```
file_cache_path "/var/chef-solo"
cookbook_path "/var/chef-solo/cookbooks"
json_attribs "http://www.example.com/node.json"
recipe_url "http://www.example.com/chef-solo.tar.gz"
```

You can create the solo.rb in `/etc/chef/solo.rb` instead, as `chef-solo` will look there by default, thus making the '`-c ~/solo.rb`' optional.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Roles

Chef Solo can also use Roles. They can be defined in the JSON format or with the Ruby DSL, and you need to tell the solo config file where the roles live. Using the earlier example:

~/solo.rb

```
file_cache_path "/var/chef-solo"
cookbook_path "/var/chef-solo/cookbooks"
role_path "/var/chef-solo/roles"
```

The JSON file for the role would look something like:

/var/chef-solo/roles/test.json

```
{
  "name": "test",
  "default_attributes": { },
  "override_attributes": { },
  "json_class": "Chef::Role",
  "description": "This is just a test role, no big deal.",
  "chef_type": "role",
  "run_list": [ "recipe[test]" ]
}
```

The above file using the Ruby DSL would look like:

/var/chef-solo/roles/test.rb

```
name 'test'
description 'This is just a test role, no big deal.'
run_list 'recipe[test]'
```

Finally, the JSON data passed to chef-solo:

~/chef.json

```
{ "run_list": "role[test]" }
```

See how to set the run_list in JSON.

Environments

Currently chef-solo does not have support for environments.

 Chef Client

Chef Server 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Server



*The Chef Server provides a central point for the distribution of **Cookbooks**, management and authentication of **Nodes**, and the use of **Search**.*

*It provides two layers of functionality - a **REST API** and a human-readable Web Interface (**Open Source Chef Server Management Console**).*

Components

The Chef Server is a Merb web application with some additional components.

- API service.
- Management Console (optional).
- AMQP Server
- Search indexer and Search Engine.
- Data store with CouchDB.

API Service

Service Name	chef-server
Package/Gem	chef-server,chef-server-api
Default port	4000

The API service is what clients use to interact with the server to manage node configuration in Chef. By default, the service is started on port 4000 as a Merb application slice running with the thin server adapter.

The two methods of interaction with the API for humans are the command-line tool Knife and the Open Source Chef Server Management Console . The Chef Client library is used for interacting with the API for client nodes.

Bake in Scalability with Chef

Our friends at Cycle Computing use a CentOS 5.5 Chef Server to provide configuration management for hundreds of systems and automate the scalability of their infrastructure. Read their blog post to see how they do it.

Open Source Server Management Console

Service Name	chef-server-webui
Package/Gem	chef-server-webui

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Default port	4040

The Open Source Server Management Console is an optional component that provides a nice way for humans to work with the Chef Server. By default, the service is started on port 4040 also as a Merb application slice running with thin. The Management Console has Users that represent human logins and uses password-based authentication. Optionally, OpenIDs can be associated with Open Source Management Console users.

The Management Console itself is an API client, and the private key is located in `/etc/chef/webui.pem`, and the default name is `chef-webui`. Within the Management Console are users and the default Management Console user is `admin` with password `p@ssw0rd1`. When using the RubyGems bootstrap cookbook, the password is randomly generated by Chef, or can be specified by a JSON attribute.

Open Source Chef Server Management Console has details on its use, while Hosted Chef customers should review the Hosted Chef Management Console. Private Chef customers should turn to separately delivered Administration Documentation.

AMQP Server

Service Names	rabbitmq-server
Package	rabbitmq-server*
Default port	5672,4369,50229

*rabbitmq-server package name varies by distribution.

The Chef Server runs RabbitMQ as an AMQP server. Whenever data is stored in CouchDB that needs to be indexed by SOLR, the server sends a message and the data payload to the queue, and the indexer picks it up.

Search Indexes

Service Names	chef-solr-indexer	chef-solr
Package/Gem	chef-solr	chef-solr
Default port	-	8983

The search indexer, `chef-solr-indexer` listens to AMQP for messages about what to index, and then passes the results to `chef-solr`. Read more about the Chef Indexer or Search.

Data Store

Service Name	couchdb
Package	couchdb
Default port	5984

Chef Server utilizes CouchDB for storing JSON data about Nodes, Roles, and Data Bags. The server requires CouchDB 0.9.1 or above for API compatibility reasons. For more information about CouchDB, please see CouchDB's project page.



CouchDB Administration
CouchDB Administration for Chef Server

Information Stored

The Chef Server stores several bits of information about the configured objects in the infrastructure.

API Clients

Clients are entities that access the API. They are stored as JSON objects in CouchDB, are indexed and searchable. Each client has a public key

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

stored on the server, and a private key that should be copied to the client. Clients come in two flavors for accessing the API.

- Humans
- Non-humans

Humans interact with the API through Knife or the Management Console. In the Management Console, they have users, and the Management Console itself is a non-human API client whose name is chef-webui. Non-humans are Nodes running the `chef-client` command/daemon, or services that otherwise access the API programatically.

Cookbooks

The Chef Server distributes Cookbooks to Chef Clients - specifically, it distributes:

- Libraries
- Attributes
- Definitions
- Recipes

Directly to each node based on the dependencies specified in the cookbook Metadata.

The Server also serves up File Distribution and Templates on an as-needed basis.

Nodes

The Chef Server provides for the management and authentication of Nodes via pre-shared RSA keys. Node data is indexed and searchable. See the Nodes section for more details on how to manage Nodes. See Authentication and Authorization to learn how the model works.

Roles

The Chef Server stores JSON objects about Roles configured. Roles are indexed and searchable. Learn more about Roles and how to manage them on the Roles page.

Data Bags

The server can optionally store arbitrary JSON data known as data bags. Data bags are indexed and searchable. See the Data Bags page for more information on how to use this feature.

 Chef Solo

Chef Indexer 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Indexer



The Chef Indexer is comprised of a RabbitMQ message queue, chef-expander, which pulls messages from the queue and formats them, and chef-solr, a thin wrapper around Solr.

RabbitMQ

RabbitMQ provides a queuing service which stores requests for updates to the search index. This allows the API server to handle load spikes while maintaining an even load on Solr.

We recommend you read RabbitMQ's administration guide before deploying it into production.

Configuring Chef's RabbitMQ-Specific Settings

Setting	Default Value	Description
amqp_host	'0.0.0.0'	The IP address or resolvable hostname of the server running RabbitMQ
amqp_port	'5672'	The port on which RabbitMQ listens for requests
amqp_user	'chef'	The username to use when connecting to RabbitMQ
amqp_pass	'testing'	The password to use when connecting to RabbitMQ
amqp_vhost	'/chef'	The RabbitMQ vhost to use
amqp_consumer_id	nil	A tag to append to the queue name. The implications of this setting are discussed in the *redundancy and scaling* section below

More information:

- RabbitMQ User Management
- RabbitMQ Access Control and VHosts
- Getting status information about a running RabbitMQ **Note:** be sure to include `-p /chef` (or `-p /VHOST_NAME` if you are using a custom vhost) when running `rabbitmqctl` to query for information about the chef vhost.

Node in Knife, but not in Chef Searches?

This has happened when there has been an additional quotation mark, tripping up the indexer. (for example: 3.5" rather than 3.5)

There are two possible solutions: remove offending lines and then run `knife index rebuild`, or install fast_xs.

Configuring RabbitMQ for Chef

Chef requires you to configure a user and vhost in RabbitMQ. Normally, this configuration will be handled for you by either post-install scripts in Chef packages, or in the rabbitmq_chef recipe included in the bootstrap install. **You only need to run these commands if you are doing a fully manual installation or have destroyed your RabbitMQ configuration somehow.**

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

To configure RabbitMQ for Chef using the default vhost "/chef", user "chef", and password "testing". (You may need to do this as "rabbitmq" user or whatever user rabbitmq runs as):

```
rabbitmqctl add_vhost /chef
rabbitmqctl add_user chef testing
rabbitmqctl set_permissions -p /chef chef ".*" ".*" ".*"
```

To verify that your configuration is correct:

```
shell$ rabbitmqctl list_vhosts
Listing vhosts ...
/
/chef
...done.

shell$ rabbitmqctl list_users
Listing users ...
chef
guest
...done.

shell$ rabbitmqctl list_user_permissions chef
Listing permissions for user "chef" ...
/chef .* .* .*
...done.
```

RabbitMQ Persistence

As of Chef 0.10, you can configure Chef to use RabbitMQ persistence. This means that RabbitMQ will store queued messages on disk, allowing them to be recovered if RabbitMQ should fail (but not if the disk on the box running RabbitMQ fails). To enable persistence, make sure you are running RabbitMQ 2.2 or higher. Then set `persistent_queue` to true in your server.rb configuration file.

Chef Expander

New in Chef 0.10
Chef Expander is new in Chef 0.10 and replaces chef-solr-indexer

Chef Expander fetches messages from RabbitMQ, processes them into the correct format to be loaded into Solr and loads them into Solr.

Running Chef Expander

Chef Expander is designed for clustered operation, though small installations will only need one worker process. To run Chef Expander with one worker process, run `chef-expander -n 1`. You will then have a master and worker process, which looks like this in `ps`:

```
your-shell> ps aux|grep expander
you   52110   0.1  0.7  2515476  62748 s003  S+    3:49PM   0:00.80 chef-expander worker #1 (vnodes
0-1023)
you   52108   0.1  0.5  2492880  41696 s003  S+    3:49PM   0:00.91 ruby bin/chef-expander -n 1
```

Workers are single threaded and therefore cannot use more than 100% of a single CPU. If you find that your queues are getting backlogged (see Operation and Troubleshooting, below), increase the number of workers.

Chef Expander Operation and Troubleshooting

Chef Expander includes `chef-expanderctl`, a management program that allows you to get status information or change the logging verbosity

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

(without restarting). `chef-expanderctl` has the following commands:

- `chef-expanderctl help` prints usage.
- `chef-expanderctl queue-depth` Shows the total number of messages in the queues. See the design section for more explanation.
- `chef-expanderctl queue-status` Show the number of messages in each queue. This is mainly of use when debugging a Chef Expander cluster
- `chef-expanderctl log-level LEVEL` Sets the log level on a running Chef Expander or cluster.

If you suspect that a worker process is stuck, as long as you are using clustered operation, you can simply kill the worker process and it will be restarted by the master process.

Design

Chef Expander uses 1024 queues (called *vnodes* in some places) to allow you to scale the number of Chef Expander workers to meet the needs of your infrastructure. When objects are saved in the API server, they are added to queues based on their database IDs. These queues can be assigned to different Chef Expander workers to distribute the load of processing the index updates.

Redundancy and Scaling Options with Chef Indexer



0.9.x and Lower
The discussion below applies only to Chef 0.8.x and 0.9.x. In Chef 0.10 and higher, you can increase Chef Expander throughput by increasing the number of workers in the cluster

The point of setting the queue name is to switch between messages going from 1 producer to N consumers or messages going from 1 producer to (1 of N) consumers. When multiple consumers share the same queue name, you'll have the 1->(1 of N) behavior; with multiple consumers with distinct queue names, you'll get the 1-> N behavior.

The use cases these address in Chef are:

- With 1 to (1 of N) behavior, you can have multiple chef-solr-indexer processes reading data from the queue, munging the data and then posting the data to the same SOLR instance. You would want to do this if: you want redundancy in chef-solr-indexer and/or the data munging that chef-solr-indexer does becomes a bottleneck. To configure this behavior, configure each instance of chef-solr-indexer to use the same amqp_consumer_id setting.
- With 1 to N behavior, you have multiple chef-solr-indexer processes each passing the data to unique SOLR instances. You want to do this if: you want redundancy in chef-solr, and you prefer to achieve it this way instead of replicating at the SOLR level. To configure this behavior, configure each instance of chef-solr-indexer to use a different amqp_consumer_id setting, or set amqp_consumer_id to nil.

 Chef Server

CouchDB Administration for Chef Server 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

CouchDB Administration for Chef Server



Overview

This page gives some guidance on how to maintain and troubleshoot a CouchDB installation for the Chef Server.

It is not meant to be comprehensive, merely to cover the common topics that users will encounter when working with CouchDB and Chef.

For full details about CouchDB, please see its project page.

RESTful API

CouchDB has a RESTful API. This means you can interact with it using any tool that speaks HTTP. For example, curl.

```
% curl http://localhost:5984
{"couchdb":"Welcome","version":"0.10.0"}

% curl http://localhost:5984/chef
{"db_name":"chef","doc_count":36,"doc_del_count":177,"update_seq":66256,"purge_seq":0, /joined with
next line/
"compact_running":false,"disk_size":28434524,"instance_start_time":"1273693455462751","disk_format_ve
rsion":4}
```

Compaction

CouchDB stores a revision for every document when it is updated. When the Chef Client runs, it saves the node data to the server multiple times. Each time it saves, CouchDB creates a new revision. Over time with a number of clients, this can add up and cause the CouchDB database to become quite large.

CouchDB has a feature called Compaction that can be used to remove older revisions of documents. To compact a growing database, use curl to POST on the server:

```
% curl -H "Content-Type: application/json" -X POST http://localhost:5984/chef/_compact
{"ok":true}

% curl http://localhost:5984/chef
{"db_name":"chef","doc_count":36,"doc_del_count":177,"update_seq":66256,"purge_seq":0, /joined with
next line/
"compact_running":false,"disk_size":507996,"instance_start_time":"1273761005173758","disk_format_vers
ion":4}
```

Since this is a common thing to do on a Chef Server, the Opscode chef-server cookbook does this in its default recipe.

Use of the Opscode chef-server cookbook means that every time Chef runs on the Chef Server with the default Recipe for CouchDB compaction, it will compact the database. In addition to compacting the database, the default recipe also compacts the views.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Compact the Views
If you are managing the CouchDB compaction yourself (*rather than using the chef-server cookbook cookbook to compact the database and the views*), be sure to run compaction on the individual views as well as the database.

The Opscode chef-server cookbook contains the default recipe which compacts both the database and the views. You can review it as an example on one method for compacting the views. *Without compaction, the views may eventually consume all available disk space.*

Administrative Interface

CouchDB comes with a web interface for administration. It can be accessed via http://localhost:5984/_utils. Obviously since this is running on localhost by default, it won't be accessible from your management workstation, so create an SSH tunnel to the server to forward the port. If your Chef server is `chef.example.com`:

```
ssh -L 5984:localhost:5984 chef.example.com
```

Then point your local system's web browser to http://localhost:5984/_utils.

Client/Node ID from CouchDB

To get the ID of a client/node from CouchDB execute

```
curl -s http://localhost:5984/chef/_design/nodes/_view/all_id | grep NODE_OR_CLIENT_NAME
```

The returned data should look something like

```
{"id":"e497412a-f4be-4e73-affe-e5c0d557c94d","key":"lucid1test","value":"lucid1test"}
```

Chef Indexer

Backing Up Chef Server

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Backing Up Chef Server



There are three disaster recovery strategies for Chef

1. ***Manage Chef objects as code in source control***
2. ***Export objects from the Chef Server***
3. ***Backup and restore CouchDB (database)***

Which method you chose depends on your existing infrastructure. For instance, if your team is familiar with Chef and development practices then using the source control method is more flexible and less susceptible to data corruption. On the other hand, backing up the database and associated files on disk fits better into organizations with traditional disaster recovery systems with centralized backup.

Manage Chef objects in Source Control

Chef enables the reconstruction of your business from nothing but a source code repository, an application data backup, and bare metal machines. With Chef, you define in source code how each part of your infrastructure is built. Cookbooks, roles, data bags and environments should be created as JSON files in source control, and then uploaded to the Chef Server.

Disaster recovery then becomes:

1. Create new Chef Server using preferred method
2. Load objects from source control using knife

Using source control also means that previous versions are always available for restoration and provides a log for auditing changes.

Export objects from the Chef server

- Use this Knife Exec script to perform a backup of an existing Chef Server:

```
cd ~/path/to/chef/repository
curl -O https://raw.github.com/jtimberman/knife-scripts/master/chef_server_backup.rb
```

- Run the knife exec script

```
knife exec chef_server_backup.rb
```

This will export your data bags, roles, and nodes stored on the chef server as JSON files within the `.chef/chef_server_backup` directory. Simply copy that directory and all files within to your backup location, and you'll have all of the objects that you use within Chef available to you as JSON files. These files can be uploaded to a new Chef server using knife.

CouchDB Backup

As indicated in CouchDB Administration for Chef Server, the CouchDB database can become quite large over time, so you may want to use Compaction on the database prior to backup. Please review the Compaction section of CouchDB Administration for Chef Server for details if you desire to pursue this prior to backup, and ensure that you compact the views as well.

A minimal backup consists of configuration files and a dump of the CouchDB database. For all platforms, locate your database, configuration, and log files and perform a filesystem copy. Be careful to preserve file permissions, too. Archive these files to wherever you want-- ideally on a

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

different machine in a different physical location – with appropriate security limiting access.

For example, here are the directories to backup for a CouchDB install on Ubuntu:

- Configuration: /etc/couchdb/
- Database files: /var/lib/couchdb/
- Logs: /var/log/couchdb/

You'll want to backup both the .couch database file (which contains many objects, including the index to the cookbooks) and the directory in 'checksum_path' specified in your server.rb, which contains the file contents. One possible example:

```
tar -czf /tmp/chef_server_backup.`date +%F`.tar.gz /var/lib/chef /etc/chef /etc/couchdb
couchdb-dump http://127.0.0.1:5984/chef |gzip -9c > /tmp/chef.`date +%F`.couchdb.gz
```

The SOLR Search index the RabbitMQ queue Chef Indexer should be rebuilt upon restoration.

 CouchDB Administration for Chef Server

Hosted Chef 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Hosted Chef



Hosted Chef

(formerly The Opscode Platform)

Powered by Opscode's Chef systems integration software

Key Benefits:

Speed

Start immediately and accelerate deployment and configuration management

Scale

Deploy from tens to thousands of servers in public and/or private clouds

Agility

Roll out complex changes to your entire fleet of servers and infrastructure

Elegance

Solve complex problems with simple, dynamic code

Sign Up Now and Manage up to Five Nodes Free!

Sign up now to begin building automated cloud infrastructure with Hosted Chef, the world's first and only SaaS platform for automated infrastructure deployment and configuration management.

Get instant access to a highly available, dynamically scalable, fully managed and supported automation environment - powered by the experts at Opscode. Once you sign up and create an organization, you can choose from several different pricing plans.

The first five nodes are free!

Functionality

At its heart, Hosted Chef is a centrally managed data store into which Chef clients publish system information such as IP addresses, loaded kernel modules, OS versions and more. This data becomes useful across your entire infrastructure in several important ways:

*** Search-based Automation:**

All the data collected by Hosted Chef is indexed and searchable. You can then dynamically query this data from within in your recipes to configure services that require complex configuration. Examples might include: dynamically discovering a master mySQL server, configuring nagios, or finding all the memcached servers in a cluster.

*** Role-based Access Control:**

Layered on top of this data index is a fine-grained access control system that allows you to centrally manage the level of infrastructure access granted to employees, contractors, and other parties. Should auditors have read-only access to the entire

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

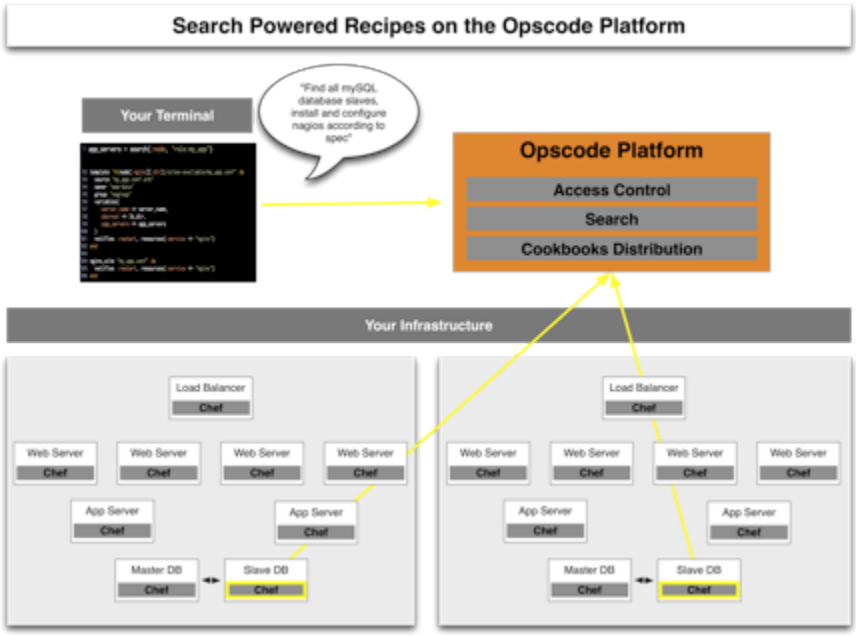
infrastructure? Should DBA contractors only have access to mySQL slaves?

*** Cookbooks Distribution:**

Hosted Chef allows you to centrally manage the cookbooks and roles that define your infrastructure. When you want to update every Apache web server in your development environment, just make one change and Hosted Chef will know where to go to work, whether in your own datacenter or in the cloud.

*** Portability:**

The data stored on Hosted Chef serves as a blueprint of your infrastructure, making it much easier to create multiple environments that are perfect clones of your production environment. Examples of this include QA environments, pre-production environments, and partner preview environments.



Built to Scale

Hosted Chef is built from the ground up by Opscode to be fully multi-tenant and horizontally scalable. In working with Opscode, you don't have to worry about the ability of your systems management tools to meet the growing demands of your infrastructure and of your business.

As configuration management becomes more core to your business, the cost of maintaining a highly available system only increases. *As you move from hundreds to thousands of systems under management, Hosted Chef will gracefully handle the load, with no concurrent increase in management overhead.*

When new versions of Chef and the Opscode API are released, Hosted Chef will immediately be available to you – data migrations and backwards compatibility are taken care of for you, virtually eliminating the work that may be required for major upgrades.

Sign Up Now and Manage up to Five Nodes Free!

Sign up now to begin building automated cloud infrastructure with Hosted Chef, the world's first and only SaaS platform for automated infrastructure deployment and configuration management. Once you sign up and create an organization, you can choose from several different pricing plans. ***The first five nodes are free!***

Backing Up Chef Server

Migrating to Hosted Chef from an Open Source Chef Server implementation

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Migrating to Hosted Chef from an Open Source Chef Server implementation

If you run your own Open Source Chef Server and want all the benefits of configuration management with none of the adminstration hassle, you can migrate to Hosted Chef without losing the work you've already done. Here's how...

Benefits



As a Hosted Chef Customer you...

- are free from managing your own Chef server(s) as your business scales. You get back to focusing on core business essentials.
- benefit immediately from new versions of Chef and the Opscode API when they are released. Data migrations and backwards compatibility are taken care of by Opscode, virtually eliminating the work that may be required for major upgrades.
- don't worry about scalability. As you move from tens to hundreds to thousands of systems under management, Hosted Chef will gracefully handle the load, with no concurrent increase in management overhead.
- are protected by an industry-leading SLA and 24x7x365 support options, giving you the confidence to run mission-critical systems.

Approach

Migrate by taking the following actions:

1. Prepare your backup.

- Download the following Knife Plugin (backup_export.rb):

```
cd ~/path/to/chef/repository
mkdir -p .chef/plugins/knife
curl https://raw.github.com/stevendanna/knife-hacks/master/plugins/backup_export.rb >
.chef/plugins/knife/backup_export.rb
```

- Run the knife plugin to backup your existing Chef Server:

```
knife backup export
```

This will export from the chef server all data bags, roles, environments, and nodes, storing them as JSON files within the `.chef/chef_server_backup` directory.

These JSON files will later be uploaded to Hosted Chef in step 5.

2. Create your accounts and organization on Hosted Chef.

See Setup Opscode User and Organization for detailed instructions on creating a new organization. Rather than creating a new, local chef repository, you can reconfigure your current repository to use Hosted Chef.

- Replace `.chef/knife.rb` with the version provided by the Management console.
- Move your Hosted Chef user's authentication key into the `.chef` directory, and

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- Move your Hosted Chef organization's validator key into the `.chef` directory.

3. Confirm that your new Opscode Hosted Chef organization has no nodes and just the validator client

```
knife node list
knife client list
```

The only client that should be listed is your organization's validator client. No nodes should be listed.

4. Register your existing servers with Hosted Chef

Use a custom Knife Bootstrap template to populate the nodes with the correct configuration file and validation certificate, delete the existing client certificate, and run chef-client to register the node as a new API client to the Opscode Hosted Chef Organization.

If your nodes are running a Microsoft Windows platform, you will need to use the Knife Windows Bootstrap and modify the template for the Windows platform. Be sure to remove the client.pem file, typically `c:\chef\client.pem`.

Note *This assumes, as above, that your Hosted Chef organization has no data for these nodes. The run list will be empty. We will populate that from the backed up node data in a later step.*

- Save the following script as `.chef/bootstraps/migration.erb` within your chef repository. **Note** *if you customized the client.rb, you will need to modify this script.*

```
bash -c '
# removed installation, presumably Chef is installed on existing nodes

rm /etc/chef/client.pem /etc/chef/validation.pem

(
cat <<'EOP'
<%= validation_key %>
EOP
) > /tmp/validation.pem
awk NF /tmp/validation.pem > /etc/chef/validation.pem
rm /tmp/validation.pem

(
cat <<'EOP'
<%= config_content %>
EOP
) > /etc/chef/client.rb

chef-client
'
```

- Run this bootstrap on each of your nodes. The exact command will depend on how your environment is configured. To run this on a node running Ubuntu you may execute the command:

```
knife bootstrap TARGET_IPADDR -d migration -x ubuntu --sudo
```

5. Upload data to your Hosted Chef organization

You can restore the node data using a Knife Plugin (backup_restore.rb):

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
cd ~/path/to/chef/repo
curl https://raw.github.com/stevendanna/knife-hacks/master/plugins/backup_restore.rb >
.chef/plugins/knife/backup_restore.rb
knife backup restore
```

Alternatively, can upload the data by hand:

- **node data**
  For each of the files found in the `nodes` directory of your chef backup, do the following:

  ```
  knife node from file /path/to/chef/backup/nodes/some_node.json
  ```

  Running this command will repopulate the newly created nodes with their previous run_list and attributes.

- **cookbooks**
  From a chef repository configured to use hosted chef and containing all of your cookbooks do:

  ```
  knife cookbook upload -a
  ```

- **roles**
  For each of the files found in the `roles` directory of your chef backup, do the following:

  ```
  knife role from file /path/to/chef/backup/roles/some_role.json
  ```

- **data bags**
  For each of the files found in the `data_bags` directory of your chef backup, do the following:

  ```
  knife data bag from file /path/to/chef/backup/data_bags/some_bag.json
  ```

6. Run chef-client on all of your nodes

Depending on your environment, this could be done via knife ssh:

```
knife ssh 'name:*' 'sudo chef-client'
```

Questions on the migration?

Open a Hosted Chef ticket with any questions or issues encountered, or send an email to support@opscode.com.

Need Specialized Assistance?

If you have a particularly complex environment, and would like to seek Professional Services assistance, please contact us with your needs and we'll go over the possibilities with you.



Hosted Chef

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Server API



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Server API

Overview

This page serves as an overview of the Chef Server REST API, and on managing API clients through this Server API.

API Clients can also be managed via Knife or the Open Source Chef Server Management Console, see API Clients for documentation on those approaches, and Authentication for the process of authentication and validation of the clients access to the server.

There are requirements in order to interact with the Chef Server REST API. In order to make any of these API calls, you will need to satisfy these pre-requisites:

- Have Chef 0.9.x or above
- Set your Accept header to "application/json"
- Set your X-Chef-Version header to the version of the API you are using
- Sign your request with Mixlib::Authentication

The easiest way to ensure a well formatted request is to use the Chef::REST library bundled with Chef. See the Authentication page for more information about Chef's authentication mechanism.

Clients

API Clients communicate with the Chef Server via REST, authenticates via Signed Header Authentication, and compiles and executes Cookbooks.

/clients

Manipulate or create API Clients.

HTTP Methods

GET

Returns a list of all the API clients. This includes the API Client registration for each Chef node, any other API users you may create, and the validation and webui clients.

Response

Response to a GET on /clients

```
{
  "chef-webui": "http://localhost:4000/clients/chef-webui",
  "chef-validator": "http://localhost:4000/clients/chef-validator",
  "adam": "http://localhost:4000/clients/adam"
}
```

- Returns "200 OK".

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Hosted and Private Chef
The Hosted and Private Chef APIs are compatible with the Chef Server API described in this article.

POST

Creates a new client.

The body for a POST to /clients

```
{
  "name": "monkeypants",
  "admin": false
}
```

The name field should be the name of the client you want created. The admin field controls whether or not this is an "admin" client (meaning it can manipulate most objects in the system.)

Response

The response to a POST to /clients

```
{
  "uri": "http://localhost:4000/clients/monkeypants",
  "private_key": "RSA PRIVATE KEY"
}
```

You will want to make sure you store the contents of the private_key attribute after creation. You can then use this client name and private key to access the REST API.

- Returns "201 Created".
- Returns "401 Unauthorized" if you are not a recognized user.
- Returns "403 Forbidden" if the user is not authorized to create a client.
- Returns "409 Conflict" if you try and create a client that already exists.

/clients/CLIENT_NAME

HTTP Methods

GET

Returns the client.

Response

The response to a GET of /clients/CLIENT_NAME

```
{
  "name": "monkeypants",
  "chef_type": "client",
  "json_class": "Chef::ApiClient",
  "public_key": "RSA PUBLIC KEY",
  "_rev": "1-68532bf2122a54464db6ad65a24e2225",
  "admin": true
}
```


This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- Returns "200 OK".
- Returns "401 Unauthorized" if you are not a recognized user.
- Returns "403 Forbidden" if the client is not allowed to view the client.
- Returns "404 Not Found" if the client does not exist.

PUT

Updates the client.

The request body should be:

The response to a PUT to /clients/CLIENT_NAME

```
{
  "name": "monkeypants",
  "private_key": true,
  "admin": false
}
```

If private_key is set to true, a new key-pair for this client will be generated, and the new private key will be returned in the response body.

If admin is set, this client will be promoted to an administrator.

Response

The response to a PUT to /clients/CLIENT_NAME

```
{
  "name": "monkeypants",
  "private_key": "RSA PRIVATE KEY",
  "admin": true
}
```

- Returns "200 OK".
- Returns "401 Unauthorized" if you are not a recognized user.
- Returns "403 Forbidden" if the user is not authorized to update the client.
- Returns "404 Not Found" if the client does not exist.

DELETE

Removes the client. Request has no body.

Response

- Returns "200 OK".
- Returns "401 Unauthorized" if you are not a recognized user.
- Returns "403 Forbidden" if the client is not authorized to delete the client.
- Returns "404 Not Found" if the client does not exist.

Cookbooks

In Chef, cookbooks have one or more versions, which are manifests containing the version's metadata and information about its component files.

return to top of page

In order to minimize storage and the amount of time required to iterate in the modify-upload-test cycle, cookbooks only require that files with checksums the system has not yet seen be uploaded. To accomplish this, a cookbook version's component files (each with its own particular

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

checksum) are uploaded using the sandbox API. These checksums are then used in the version's manifest in records that include the component file's description (name, specificity, etc.), as well as its checksum and a URL from which to retrieve the file's contents. This is why you will see only the files that you have updated get uploaded when doing a "knife cookbook upload COOKBOOK_NAME".

/cookbooks

HTTP Methods

GET

Chef 0.9.x:

Returns a list of all the cookbooks on this chef server. The request has no body, and returns a hash as follows:

Response

Response to GET /cookbooks

```
{
  "unicorn": "http://localhost:4000/cookbooks/unicorn",
  "apache2": "http://localhost:4000/cookbooks/apache2"
}
```

Chef 0.10.0 or above:

Returns a hash of all the cookbooks and their versions. Query parameter num_versions can be used to set the number of versions being returned. For example, GET to /cookbooks?num_versions=3 returns 3 latest versions, in descending order (high to low); /cookbooks?num_versions=all returns all available versions, in descending order (high to low). If num_versions is not specified, it defaults to 1. Note that 0 is a valid input that returns an empty array for the versions of each cookbooks.

The request has no body, and returns a hash as follows:

Response

Example response to GET /cookbooks?num_versions=all

```
{
  "apache2" => {
                "url" => "http://localhost:4000/cookbooks/apache2",
                "versions" => [
                   {"url" => "http://localhost:4000/cookbooks/apache2/5.1.0",
                    "version" => "5.1.0"},
                   {"url" => "http://localhost:4000/cookbooks/apache2/4.2.0",
                    "version" => "4.2.0"}
                 ]
  },
  "nginx" => {
                "url" => "http://localhost:4000/cookbooks/nginx",
                "versions" => [
                   {"url" => "http://localhost:4000/cookbooks/nginx/1.0.0",
                    "version" => "1.0.0"},
                   {"url" => "http://localhost:4000/cookbooks/nginx/0.3.0",
                    "version" => "0.3.0"}
                 ]
  }
}
```

- Returns "200 OK" normally.
- Returns "401 Unauthorized" if you are not a recognized client.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- Returns "403 Forbidden" if you do not have permission to see the cookbook list.

/cookbooks/COOKBOOK_NAME

HTTP Methods

GET

Chef 0.9.x

Returns a listing of the versions of the cookbook.

Response

Response to GET /cookbooks/unicorn

```
{
  "unicorn": [
    "0.1.2"
  ]
}
```

Chef 0.10.0 and above

Returns a hash in the same format as the **/cookbooks** API endpoint. The hash contains one key-value pair corresponding to the cookbook requested. The same query parameters as the **/cookbooks** endpoint apply. The **num_versions** query parameter defaults to "all".

Response to GET /cookbooks

```
{
  "apache2" => {
                "url" => "http://localhost:4000/cookbooks/apache2",
                "versions" => [
                   {"url" => "http://localhost:4000/cookbooks/apache2/5.1.0",
                    "version" => "5.1.0"},
                   {"url" => "http://localhost:4000/cookbooks/apache2/4.2.0",
                    "version" => "4.2.0"}
                 ]
               }
}
```

- Returns "200 OK" normally.
- Returns "401 Unauthorized" if you are not a recognized client.
- Returns "403 Forbidden" if you do not have permission to see the cookbook.

/cookbooks/COOKBOOK_NAME/COOKBOOK_VERSION

HTTP Methods

GET

Returns a description of the cookbook, including its metadata and links to all of its component files. COOKBOOK_VERSION can be "_latest" in order to float to head. It requires no request body.

Response

Response to GET /cookbooks/unicorn/0.1.0

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
{
    "definitions": [
      {
        "name": "unicorn_config.rb",
        "url": "https://s3.amazonaws.com/opscode-platform-production-data/organization-(...)",
        "checksum": "c92b659171552e896074caa58dada0c2",
        "path": "definitions/unicorn_config.rb",
        "specificity": "default"
      }
    ],
    "name": "unicorn-0.1.2",
    "attributes": [

    ],
    "files": [

    ],
    "json_class": "Chef::CookbookVersion",
    "providers": [

    ],
    "metadata": {
      "dependencies": {
        "ruby": [

        ],
        "rubygems": [

        ]
      },
      "name": "unicorn",
      "maintainer_email": "ops@opscode.com",
      "attributes": {
      },
      "license": "Apache 2.0",
      "suggestions": {
      },
      "platforms": {
      },
      "maintainer": "Opscode, Inc",
      "long_description": "= LICENSE AND AUTHOR:\n\nAuthor:: Adam Jacob
<adam@opscode.com>\n\nCopyright 2009-2010, Opscode, Inc.\n\nLicensed under the Apache License, Version
2.0 (...)",
      "recommendations": {
      },
      "version": "0.1.2",
      "conflicting": {
      },
      "recipes": {
        "unicorn": "Installs unicorn rubygem"
      },
      "groupings": {
      },
      "replacing": {
      },
      "description": "Installs/Configures unicorn",
      "providing": {
      }
    },
    "libraries": [
```


 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
    ],
    "templates": [
      {
        "name": "unicorn.rb.erb",
        "url": "https://s3.amazonaws.com/opscode-platform-production-data/organization-(...)",
        "checksum": "36a1cc1b225708db96d48026c3f624b2",
        "path": "templates/default/unicorn.rb.erb",
        "specificity": "default"
      }
    ],
    "resources": [

    ],
    "cookbook_name": "unicorn",
    "version": "0.1.2",
    "recipes": [
      {
        "name": "default.rb",
        "url": "https://s3.amazonaws.com/opscode-platform-production-data/organization-(...)",
        "checksum": "ba0dadcbca26710a521e0e3160cc5e20",
        "path": "recipes/default.rb",
        "specificity": "default"
      }
    ],
    "root_files": [
      {
        "name": "README.rdoc",
        "url": "https://s3.amazonaws.com/opscode-platform-production-data/organization-(...)",
        "checksum": "d18c630c8a68ffa4852d13214d0525a6",
        "path": "README.rdoc",
        "specificity": "default"
      },
      {
        "name": "metadata.rb",
        "url": "https://s3.amazonaws.com/opscode-platform-production-data/organization-(...)",
        "checksum": "967087a09f48f234028d3aa27a094882",
        "path": "metadata.rb",
        "specificity": "default"
      },
      {
        "name": "metadata.json",
        "url": "https://s3.amazonaws.com/opscode-platform-production-data/organization-(...)",
        "checksum": "45b27c78955f6a738d2d42d88056c57c",
        "path": "metadata.json",
        "specificity": "default"
      }
```


 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
    ],
    "chef_type": "cookbook_version"
}
```

- "200 OK".
- "401 Unauthorized".
- "403 Forbidden".
- "404 Not Found" if the cookbook or version does not exist.

PUT

Creates or updates the cookbook version. The payload is the cookbook version's manifest, which is identical to the response of a cookbook version GET except without the URL fields for each component. Note that all checksums must have been uploaded by the user at some point via the sandbox mechanism; however, once a file with a particular checksum has been uploaded by the user, redundant uploads are not necessary.

Response

- Returns "200 OK" normally.
- Returns "401 Unauthorized" if you are not a recognized client.
- Returns "403 Forbidden".

DELETE

Deletes a cookbook version.

Response

- Returns "200 OK" normally.
- Returns "401 Unauthorized" if you are not a recognized client.
- Returns "403 Forbidden".
- Returns "404 Not Found" if the cookbook or version does not exist.

Data Bags

Data Bags allow you to store arbitrary data about your infrastructure.

return to top of page

/data

Data Bags can be used for all sorts of things, for example, application configuration and user data, etc.

HTTP Methods

GET

Returns a list of all the data bags.

Response

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Response to GET /data

```
{
  "users": "http://localhost:4000/data/users",
  "applications": "http://localhost:4000/data/applications"
}
```

The key is the name of the data bag, and the value is the URI for that data bag.

- Returns "200 OK".

POST

Creates a new data bag.

Request body for a POST to /data

```
{
  "name": "users"
}
```

Response

Response to a POST on /data

```
{
  "uri" => "http://localhost:4000/data/users"
}
```

- Returns "201 Created".
- Returns "409 Conflict" if the bag already exists.
- Returns "401 Unauthorized" if the user is not authorized to create a new bag.

/data/DATA_BAG_NAME

HTTP Methods

GET

Returns a hash of all the entries in the given bag.

Response

Response to a GET on /data/DATA_BAG_NAME

```
{
  "adam": "http://localhost:4000/data/users/adam"
}
```

- Returns "200 OK".
- Returns "401 Unauthorized" if the user is not allowed to view the bag.
- Returns "404 Not Found" if the bag does not exist.

POST

Creates a new data bag item. The request must contain an "id" field:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Request body for a POST to /data/DATA_BAG_NAME

```
{
  "id": "adam",
  "real_name": "Adam Jacob"
}
```

Response

- Returns "200 OK".
- Returns "401 Unauthorized"
- Returns "404 Not Found"
- Returns "409 Conflict" if the user already exists.

/data/DATA_BAG_NAME/DATA_BAG_ITEM_ID

This endpoint allows you to show/manipulate data bag items within a bag. The data bag item itself can be any arbitrary JSON you construct.

HTTP Methods

GET

Returns the data bag item in the bag.

Response

Response to a GET on /data/DATA_BAG_NAME/DATA_BAG_ITEM_ID

```
{
  "real_name": "Adam Jacob",
  "id": "adam"
}
```

- Returns "200 OK"
- Returns "401 Unauthorized"
- Returns "404 Not Found"

PUT

Updates the data bag item in the bag. The request body **must** contain an "id" field, besides that, you can use any JSON you wish.

Request body for a PUT to /data/DATA_BAG_NAME/DATA_BAG_ITEM_ID

```
{
  "real_name": "Adam Brent Jacob",
  "id": "adam"
}
```

Response

The response body will be the current value of the item - which should be identical to your request.

- Returns "200 OK".
- Returns "401 Unauthorized" if the user is not authorized to update the entry.
- Returns "404 Not Found".

DELETE

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Removes a data bag item from the bag. Takes no request body, and returns the last state of the item in the response.

Response

Response to a DELETE on /data/DATA_BAG_NAME/DATA_BAG_ITEM_ID

```
{
  "id": "adam",
  "real_name": "Adam Brent Jacob"
}
```

- Returns "200 OK"
- Returns "401 Unauthorized"
- Returns "404 Not Found"

Nodes

Nodes are systems that run the chef client.

return to top of page

/nodes

HTTP Methods

GET

Returns a hash of uri's for the nodes.

Response Codes

Response to GET /nodes

```
{
  "latte": "http://localhost:4000/nodes/latte"
}
```

The key is the name of the node, and the value is the URI where it can be retrieved.

- Responds with a "200 OK".

POST

A POST request to /nodes will create a new node. A sample request body:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Request body for a POST to /nodes

```
{
  "name": "latte",
  "chef_type": "node",
  "json_class": "Chef::Node",
  "attributes": {
    "hardware_type": "laptop"
  },
  "overrides": {
  },
  "defaults": {
  },
  "run_list": [ "recipe[unicorn]" ]
}
```

The name should be the name of the node you want created. The attributes hash should contain any attributes you want set for this node - overrides and defaults correspond to override and default attributes. The run_list should be an array containing recipes and roles. Note that order matters in the run_list array.

Response

In response you should receive:

Response to a POST on /nodes

```
{ "uri": "http://localhost:4000/nodes/latte" }
```

- Responds with a "201 Created"
- Returns "401 Unauthorized"
- Responds with a "409 Conflict" if the node already exists.

/nodes/NODE_NAME

GET

A GET request to /nodes/NODE will return the node.

Response

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Response to a GET /nodes/NODE

```
{
  "overrides": {

  },
  "name": "latte",
  "chef_type": "node",
  "json_class": "Chef::Node",
  "attributes": {
    "hardware_type": "laptop"
  },
  "run_list": [
    "recipe[unicorn]"
  ],
  "defaults": {

  }
}
```

- Responds with a "200 OK"
- Responds with a "401 Unauthorized"
- Responds with a "404 Not Found"

PUT

A PUT request to /nodes/NODE will update the node. The request body should be the Node:

Request body for a PUT to /nodes/NODE

```
{
  "overrides": {

  },
  "name": "latte",
  "chef_type": "node",
  "json_class": "Chef::Node",
  "attributes": {
    "hardware_type": "laptop"
  },
  "run_list": [
    "recipe[apache2]"
  ],
  "defaults": {

  }
}
```

Response

The response will be the updated node.

- Responds with a "200 OK" if the node is updated.
- Responds with a "404 Not Found" if the node does not already exist.

DELETE

A DELETE request to /nodes/NODE will delete the node. It requires no request body, and returns the last known state of the node.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Request body for a DELETE to /nodes/NODE

```
{
  "overrides": {

  },
  "name": "latte",
  "chef_type": "node",
  "json_class": "Chef::Node",
  "attributes": {
    "hardware_type": "laptop"
  },
  "run_list": [
    "recipe[apache2]"
  ],
  "defaults": {

  }
}
```

Response Codes

- Responds with a "200 OK" if the node is deleted.
- Responds with a "404 Not Found" if the node does not exist.

/nodes/NODE_NAME/cookbooks

HTTP Methods

GET

Returns the complete list of cookbook attributes, definitions, libraries and recipes that are required for this node. The request body should be blank, and the response will look like:

Response

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Response to a /nodes/NODE/cookbooks request


```
{
  "unicorn": {
    "definitions": [
      {
        "name": "unicorn_config.rb",
        "uri": "http://localhost:4000/cookbooks/unicorn/definitions?id=unicorn_config.rb",
        "checksum": "72074229598611d7a6cdcc8176ffb76748adb4127be3a9651c89bb0afab6498f"
      }
    ],
    "files": [

    ],
    "providers": [

    ],
    "templates": [
      {
        "name": "unicorn.rb.erb",
        "uri": "http://localhost:4000/cookbooks/unicorn/templates?id=unicorn.rb.erb",
        "checksum": "c1df9c8ec2bda7b14ecadc1c5a56802442cd8d7503f2524d4fa3f73dba1a1250",
        "specificity": "default"
      }
    ],
    "libraries": [

    ],
    "resources": [

    ],
    "attributes": [

    ],
    "recipes": [
      {
        "name": "default.rb",
        "uri": "http://localhost:4000/cookbooks/unicorn/recipes?id=default.rb",
        "checksum": "87a3f720bf5bbd9227228ba22946436db6598a59f8aedff37515ebd4e1157644"
      }
    ]
  }
}
```


- Responds with a "200 OK"
- Responds with a "401 Unauthorized"
- Responds with a "404 Not Found"

Roles

A role provides a means of grouping similar features of similar nodes, providing a mechanism for easily composing sets of functionality.

return to top of page

/roles

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

HTTP Methods

GET

A GET call to /roles returns a data structure that contains links to each available role.

Response

Response to GET /roles

```
{
  "webserver": "http://localhost:4000/roles/webserver"
}
```

- Responds with a "200 OK".

POST

A POST call to /roles creates a new role. The request body should contain the JSON representation of the role.

Request body for a POST to /roles

```
{
  "name": "webserver",
  "chef_type": "role",
  "json_class": "Chef::Role",
  "default_attributes": {

  },
  "description": "A webserver",
  "run_list": [
    "recipe[unicorn]",
    "recipe[apache2]"
  ],
  "override_attributes": {

  }
}
```

Response

In response, you will receive:

Response to a POST on /roles

```
{ "uri": "http://localhost:4000/roles/webserver" }
```

- Responds with a "201 Created" if the Role is created.
- Responds with a "409 Conflict" if the Role already exists.

/roles/ROLE_NAME

HTTP Methods

GET

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

A GET to /roles/ROLE returns the JSON representation of the role.

Response

Response to GET /roles/ROLE

```
{
  "name": "webserver",
  "chef_type": "role",
  "json_class": "Chef::Role",
  "default_attributes": {

  },
  "description": "A webserver",
  "run_list": [
    "recipe[unicorn]",
    "recipe[apache2]"
  ],
  "override_attributes": {

  }
}
```

- Responds with a "200 OK" if the role exists.
- Responds with a "404 Not Found" if the role does not exist.

PUT

A PUT to /roles/ROLE updates the Role.

Request body for a PUT to /roles/ROLE

```
{
  "name": "webserver",
  "chef_type": "role",
  "json_class": "Chef::Role",
  "default_attributes": {

  },
  "description": "A webserver",
  "run_list": [
    "recipe[apache2]"
  ],
  "override_attributes": {

  }
}
```

Response

The response will be the updated role.

- Responds with a "200 OK" if the role is updated.
- Responds with a "404 Not Found" if the role does not exist.

DELETE

A DELETE to /roles/ROLE_NAME deletes the role. This request has no body.

Response

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

The response to a DELETE request is the role's state just before it was removed.

Response to DELETE /roles/ROLE_NAME

```
{
  "name": "webserver",
  "chef_type": "role",
  "json_class": "Chef::Role",
  "default_attributes": {

  },
  "description": "A webserver",
  "run_list": [
    "recipe[apache2]"
  ],
  "override_attributes": {

  }
}
```

- Responds with a "200 OK"
- Responds with a "404 Not Found" if the role does not exist.

Environments

Environments in Chef provide a mechanism for managing different environments such as production, staging, development, and testing, etc with one Chef setup (or one organization on Hosted Chef).

return to top of page

Chef 0.10 Feature
The Environments APIs are only available in Chef 0.10.0 or above.

/environments

HTTP Methods

GET

A GET call to /environments returns a data structure that contains links to each available environment.

Response

Response to GET /environments

```
{
  "webserver": "http://localhost:4000/environments/webserver"
}
```

- Responds with a "200 OK".

POST

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

A POST call to /environments creates a new environment. The request body should contain the JSON representation of the environment.

Request body for a POST to /environments

```
{
  "name": "dev",
  "attributes": {
  },
  "json_class": "Chef::Environment",
  "description": "",
  "cookbook_versions": {
  },
  "chef_type": "environment"
}
```

Response

In response, you will receive:

Response to a POST on /environments

```
{ "uri": "http://localhost:4000/environments/dev" }
```

- Responds with a "201 Created" if the Environment is created.
- Responds with a "409 Conflict" if the Environment already exists.

/environments/ENVIRONMENT_NAME

HTTP Methods

GET

A GET to /environments/ENVIRONMENT returns the JSON representation of the environment.

Response

Response to GET /environments/ENVIRONMENT

```
{
  "name": "dev",
  "attributes": {
  },
  "json_class": "Chef::Environment",
  "description": "",
  "cookbook_versions": {
  },
  "chef_type": "environment"
}
```

- Responds with a "200 OK" if the environment exists.
- Responds with a "404 Not Found" if the environment does not exist.

PUT

A PUT to /environments/ENVIRONMENT updates the Environment.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Request body for a PUT to /environments/ENVIRONMENT

```
{
  "name": "dev",
  "attributes": {
  },
  "json_class": "Chef::Environment",
  "description": "The Dev Environment",
  "cookbook_versions": {
  },
  "chef_type": "environment"
}
```

Response

The response will be the updated environment.

- Responds with a "200 OK" if the environment is updated.
- Responds with a "404 Not Found" if the environment does not exist.

DELETE

A DELETE to /environments/ENVIRONMENT_NAME deletes the environment. This request has no body.

Response

The response to a DELETE request is the environment's state just before it was removed.

Response to DELETE /environments/ENVIRONMENT_NAME

```
{
  "name": "dev",
  "attributes": {
  },
  "json_class": "Chef::Environment",
  "description": "",
  "cookbook_versions": {
  },
  "chef_type": "environment"
}
```

- Responds with a "200 OK"
- Responds with a "404 Not Found" if the environment does not exist.

/environments/:environment_id/cookbooks

HTTP Methods

GET

Returns a list of all the cookbooks and their versions that are available to the specified Environment. Query parameter num_versions can be used to set the number of versions being returned. For example, GET to /environment/production/cookbooks?num_versions=3 returns 3 latest versions, in descending order (high to low); /environments/production/cookbooks?num_versions=all returns all available versions in this environment, in descending order (high to low). If num_versions is not specified, it defaults to 1. Note that 0 is a valid input that returns an empty array for the versions of each cookbooks.

The request has no body, and returns a hash as follows:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Response

Response to GET /cookbooks

```
{
  "apache2" => {
                "url" => "http://localhost:4000/cookbooks/apache2",
                "versions" => [
                   {"url" => "http://localhost:4000/cookbooks/apache2/5.1.0",
                    "version" => "5.1.0"},
                   {"url" => "http://localhost:4000/cookbooks/apache2/4.2.0",
                    "version" => "4.2.0"}
                 ]
  },
  "nginx" => {
                "url" => "http://localhost:4000/cookbooks/nginx",
                "versions" => [
                   {"url" => "http://localhost:4000/cookbooks/nginx/1.0.0",
                    "version" => "1.0.0"},
                   {"url" => "http://localhost:4000/cookbooks/nginx/0.3.0",
                    "version" => "0.3.0"}
                 ]
  }
}
```

- Returns "200 OK" normally.
- Returns "401 Unauthorized" if you are not a recognized client.
- Returns "403 Forbidden" if you do not have permission to see the cookbook list.

/environments/:environment_id/cookbook_versions

HTTP Methods

POST

Returns a hash of the required versions of cookbooks and their dependencies for the given run_list array. Version constraints may be specified by using the '@' symbol after the cookbook name as a delimiter.

```
chef > rest = Chef::REST.new(Chef::Config[:chef_server_url])
chef > rest.post_rest("environments/_default/cookbook_versions", {:run_list => [ "zed@0.0.1", "bar",
"mysql", "gem", "nginx@0.99.2", "cron", "foo"] }).keys
 => ["mysql", "runit", "zed", "cron", "openssl", "gem", "nginx", "foo", "bar", "build-essential"]
```

/environments/:environment_id/cookbooks/COOKBOOK_NAME

HTTP Methods

GET

Returns a hash in the same format as the **/environments/:env_id/cookbooks** API endpoint. The hash contains one key-value pair corresponding to the cookbook requested. The versions displayed are filtered based on the environment specified. The same query parameters as the **/environments/:env_id/cookbooks** endpoint apply. The **num_versions** query parameter defaults to "all".

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Response to GET /cookbooks

```
{
  "apache2" => {
                 "url" => "http://localhost:4000/cookbooks/apache2",
                 "versions" => [
                    {"url" => "http://localhost:4000/cookbooks/apache2/5.1.0",
                     "version" => "5.1.0"},
                    {"url" => "http://localhost:4000/cookbooks/apache2/4.2.0",
                     "version" => "4.2.0"}
                  ]
                }
}
```

- Returns "200 OK" normally.
- Returns "401 Unauthorized" if you are not a recognized client.
- Returns "403 Forbidden" if you do not have permission to see the cookbook.

Search

Searches are built by the Chef Server, and allow you to query arbitrary data about your infrastructure.

return to top of page

/search

This endpoint allows you to search your entire organizations nodes, roles, and data bags.

HTTP Methods

GET

A GET call to /search returns a data structure that contains links to each available search index.

By default, the "role", "node" and "client" indexes will always be available.

Note that the search indexes may lag behind the most current data by at least 10 seconds at any given time. If you need to write data and immediately query it, you likely need to produce an artificial delay (or simply retry until the data is available.)

Response

Response to GET /search

```
{
  "node": "http://localhost:4000/search/node",
  "role": "http://localhost:4000/search/role",
  "client": "http://localhost:4000/search/client",
  "users": "http://localhost:4000/search/users"
}
```

- Responds with a "200 OK".

/search/INDEX

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

HTTP Methods

GET

A GET call to /search/INDEX without any request parameters will return all of the data within the index.

Response

Response to GET /search/node

```
{
  "total": 1,
  "start": 0,
  "rows": [
    {
      "overrides": {
        "hardware_type": "laptop"
      },
      "name": "latte",
      "chef_type": "node",
      "json_class": "Chef::Node",
      "attributes": {
        "hardware_type": "laptop"
      },
      "run_list": [
        "recipe[unicorn]"
      ],
      "defaults": {

      }
    }
  ]
}
```

The response contains the total number of rows that matched your request, the position this result set returns (useful for paging) and the rows themselves.

You can modify the search results with the following request parameters:

Param	Description
q	A valid search string.
sort	A sort string, such as 'name DESC'
rows	How many rows to return
start	The result number to start from

See the Search Query Syntax for help on composing search strings.

- Responds with "200 OK".
- Responds with "404 Not Found" if the index does not exist.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Sandboxes

Sandboxes are used to commit files so that they only have to be uploaded once, as opposed to each time a cookbook is uploaded, even if only one file has changed.

return to top of page

/sandboxes

Create or list sandboxes.

HTTP Methods

POST

Creates a new sandbox. It accepts a list of checksums as input and returns the URLs against which to PUT files that need to be uploaded.

Request

Request body for a POST to /sandboxes

```
{"checksums": {
  "385ea5490c86570c7de71070bce9384a":null,
  "f6f73175e979bd90af6184ec277f760c":null,
  "2e03dd7e5b2e6c8eab1cf41ac61396d5":null
  }
}
```

Response

Response to a POST to /sandboxes

```
{"uri":
  "https://api.opscode.com/organizations/testorg/sandboxes/eff7b6f8b3ef44c6867216662d5eeb5f",
 "checksums":
  {"385ea5490c86570c7de71070bce9384a":
    {"url":
      "https://s3.amazonaws.com/opscode-platform-production-data/organization-(...)",
     "needs_upload":true},
   "f6f73175e979bd90af6184ec277f760c"=>
    {"url":
      "https://s3.amazonaws.com/opscode-platform-production-data/organization-(...)",
     "needs_upload":true},
   "2e03dd7e5b2e6c8eab1cf41ac61396d5":
    {"url":
      "https://s3.amazonaws.com/opscode-platform-production-data/organization-(...)",
     "needs_upload":true}},
 "sandbox_id"=>"eff7b6f8b3ef44c6867216662d5eeb5f"}
```

- Returns "200 OK" with a hash mapping each checksum to a hash that contains a boolean "needs_upload" field and a URL if "needs_upload" is true.
- Returns "400 Bad Request" if the payload does not contain a well-formed "checksums" parameter that is a hash containing a key for each checksum.

/sandboxes/SANDBOX_ID

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

HTTP Methods

PUT

Commits the files the sandbox to their final location so that proceeding cookbook changes will not require re-uploading the same data.

Request

Request body for a PUT to /sandboxes/SANDBOX_ID

```
{"is_completed":true}
```

Response

- Returns "200 OK" if all checksums have been uploaded and the contents have the proper checksums.
- Returns "400 Bad Request" if the sandbox has already been committed or one or more of the checksums were not properly uploaded.
- Returns "404 Not Found" if the sandbox does not exist.

/sandboxes/SANDBOX_ID/CHECKSUM

HTTP Methods

PUT

Request

The contents of the file whose checksum is CHECKSUM.

Response

- Returns "200 OK".
- Returns "400" if the file's contents do not have the correct checksum.

Client Libraries

return to top of page

pychef

Opscode Team Member Noah Kantrowitz (coderanger) has written PyChef, a Python API allowing interaction with the Chef REST interface. While only a subset of Chef is currently wrapped as objects, any of the API calls below can be made via the `api_request` mechanism:

```
>>> import chef.api
>>> chef_api = chef.api.autoconfigure()
>>> chef_api.api_request('GET', '/data')
{u'sites': u'http://chef.example.com:4000/data/sites'}
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Migrating to Hosted Chef from an Open Source Chef Server implementation

Cookbook Site API



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook Site API

Opsocode maintains a public collection of cookbooks at the cookbooks.opscode.com. This collection is accessible via a REST API. This article documents the REST API methods accepted by the cookbooks site.

Introduction

A cookbook is the fundamental unit of distribution in Chef. They are how you encapsulate all the different resources you need to automate your infrastructure, and they are the component that is easily sharable between different Chef users. Opscode maintains a public collection of cookbooks at cookbooks.opscode.com. This site contains all the cookbooks authored by Opscode, as well as many cookbooks from other members of the community.

All the cookbooks on the site are accessible through a REST API rooted at cookbooks.opscode.com/api/v1/. The endpoints and their HTTP methods are described below.

In most cases, using this API directly is unnecessary. *Rather, you should use the `knife cookbook site` sub-commands or interact directly with the Cookbooks site.* See **Knife** and **Cookbook Site Help** for more details.

Work in Progress
This document is a work in progress and is subject to change.

API Endpoints

/cookbooks

HTTP METHODS

POST

Creates a new cookbook.

GET

Gets a listing of the available cookbooks. It is a good idea to use the optional parameters start and items to limit and paginate the results.

Parameters:

Parameter	Description
start	The offset into the cookbook list at which you would like the result set to start.
items	The number of items to return in the result set.

Example Request and Response

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Request

```
GET /cookbooks?start=20&items=5
```

Response

```
{
 "items":
  [
   {
    "cookbook_name": "apache",
    "cookbook_description": "installs apache.",
    "cookbook": "http://cookbooks.opscode.com/api/v1/cookbooks/apache",
    "cookbook_maintainer": "john"
   },
   {"cookbook_name": "fail2ban",
    "cookbook_description": "installs fail2ban.",
    "cookbook": "http://cookbooks.opscode.com/api/v1/cookbooks/fail2ban",
    "cookbook_maintainer": "jill"
   },
   {"cookbook_name": "mysql",
    "cookbook_description": null,
    "cookbook": "http://cookbooks.opscode.com/api/v1/cookbooks/mysql",
    "cookbook_maintainer": "barry"
   },
   {"cookbook_name": "capistrano",
    "cookbook_description": null,
    "cookbook": "http://cookbooks.opscode.com/api/v1/cookbooks/capistrano",
    "cookbook_maintainer": "pt"
   },
   {"cookbook_name": "ptapache",
    "cookbook_description": "an alternate apache recipe.",
    "cookbook": "http://cookbooks.opscode.com/api/v1/cookbooks/ptapache",
    "cookbook_maintainer": "pt"
   }
  ],
 "total": 5234,
 "start": 20
}
```

Each item in the array contains the name of the cookbook, the description, URI, and username of the maintainer of the cookbook. The total is the number of cookbooks on the site.

Response Codes:

- Returns "200 OK".

/cookbooks/{cookbook}

HTTP METHODS

DELETE

Deletes a cookbook. Not yet fully implemented.

GET

Gets a cookbook.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Parameters

None.

Example Request and Response

Request

```
GET /cookbooks/apache
```

Response

```
{
    "name": "apache",
    "category": "web servers",
    "updated_at": "2009-09-26T00:51:36Z",
    "maintainer": "jtimberman",
 "latest_version": "http://cookbooks.opscode.com/api/v1/cookbooks/apache/versions/2_0",
 "external_url": null,
 "versions":
  ["http://www.example.com/api/v1/cookbooks/apache/versions/1_0",
   "http://www.example.com/api/v1/cookbooks/apache/versions/2_0"],
 "description": "installs apache.",
 "average_rating": null,
 "created_at": "2009-09-26T00:51:36Z"
}
```

latest_version is the URI for the most recent version of the cookbook. external_url is an optional field that can be used by the cookbook maintainer to point to a related website. The versions array contains the URIs for all available versions of the cookbook.

Example Response for a Deprecated Cookbook

Request

```
GET /cookbooks/apache
```

Response

```
{
    "name": "apache",
 "category": "web servers",
 "updated_at": "2009-09-26T00:51:36Z",
 "maintainer": "jtimberman",
 "latest_version": "http://cookbooks.opscode.com/api/v1/cookbooks/apache/versions/2_0",
 "external_url": null,
 "versions":
  ["http://www.example.com/api/v1/cookbooks/apache/versions/1_0",
   "http://www.example.com/api/v1/cookbooks/apache/versions/2_0"],
 "description": "installs apache.",
 "replacement": "http://cookbooks.opscode.com/api/v1/cookbooks/ptapache",
 "deprecated": true,
 "average_rating": null,
 "created_at": "2009-09-26T00:51:36Z"
}
```

In this example, deprecated is set to true, indicating that the maintainer of the cookbook has deprecated it. In this case, another cookbook may be indicated in the replacement field. The replacement may also be "".

Response Codes:

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- Returns 200 when the cookbook exists.
- Returns 404 when the cookbook does not exist. In this case, the response body will be:

```
{
 "error_messages":
 ["Resource does not exist"],
 "error_code": "NOT_FOUND"
    }
```

/cookbooks/{cookbook}/versions/{version}

HTTP METHODS

GET

Gets a specific version of a cookbook.

Parameters

None.

Example Requests and Responses

Requests

```
GET /cookbooks/apache/versions/latest
    GET /cookbooks/apache/versions/2_0
    GET /cookbooks/apache/versions/0_1
    GET /cookbooks/apache/versions/2_0_1
    GET /cookbooks/apache/versions/4
```

The `version` is always in the form `INT[_INT[Staging:_INT]]` or "latest".

Response

```
{
 "license": "GPLv2",
 "updated_at": "2009-09-26T00:51:36Z",
 "tarball_file_size": null,
 "version": "2.0",
 "average_rating": null,
 "cookbook": "http://www.example.com/api/v1/cookbooks/ptapache",
 "created_at": "2009-09-26T00:51:36Z",
 "file": "/tarballs/original/missing.png"
    }
```

license is a string containing the name of the license under which the cookbook is distributed. It can contain anything the cookbook's maintainer chooses to put in it. file contains the link to the actual cookbook tarball.

Response Codes:

- Returns 200 when the cookbook exists.
- Returns 404 when the cookbook does not exist. In this case, the response body will be:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
{
 "error_messages":
 ["Resource does not exist"],
 "error_code": "NOT_FOUND"
    }
```

/search

HTTP METHODS

GET

Gets a listing of cookbooks whose names match a given string. It is a good idea to use the optional parameters start and items to limit and paginate the results.

Parameters

Parameter	Description
q	The search query.
start	The offset into the cookbook list at which you would like the result set to start.
items	The number of items to return in the result set.

Examples and Responses

Request

```
GET /search?q=apache GET /search?q=apache&start=20&items=5
```

Response

```
{
 "items":
  [{"cookbook_name": "apache",
    "cookbook_description": "installs a web server.",
    "cookbook": "http://cookbooks.opscode.com/api/v1/cookbooks/apache",
    "cookbook_maintainer": "jtimberman"},
   {"cookbook_name": "webserver",
    "cookbook_description": "installs apache.",
    "cookbook": "http://cookbooks.opscode.com/api/v1/cookbooks/webserver",
    "cookbook_maintainer": "raxmus"}],
 "total": 2,
 "start": 0
    }
```

Items that are in the result set contain the search query in the cookbook_name or cookbook_description. Each item in the array contains the name of the cookbook, the description, URI, and username of the maintainer of the cookbook. The total is the number of cookbooks on the site.

Response Codes:

- Returns "200 OK".

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Server API

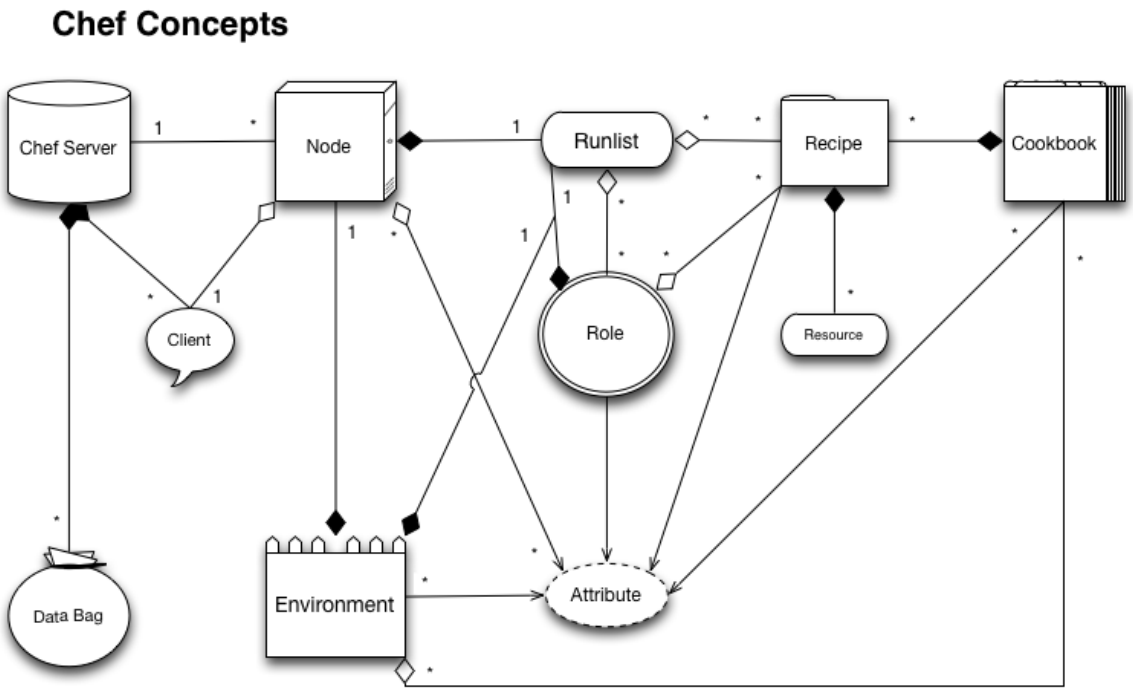
Chef Concepts as UML



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Concepts as UML

Despite rusty UML, we thought it would be worthwhile to share a diagram of relationships between the various Chef components. Feel free to update it or leave feedback if there appear to be incorrect connections. The original OmniGraffle file is attached.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Server

- There is 1 Server.

Client

- There may be multiple Clients.
- Each Client has 1 Server.
- A Client may or may not belong to a Node.

Environment

- There may be multiple Environments.
- Each Environment has 0 or more versions of a Cookbook.

Node

- There may be multiple Nodes.
- Each Node has 1 Server.
- Each Node has 1 Client.
- Each Node has 1 Run List.
- Each Node has 1 Environment (_default if none is specified)
- Each Node has 0 or more Attributes.

Run List

- The Run List may have 0 or more Roles.
- The Run List may have 0 or more Recipes.

Role

- There may be multiple Roles.
- Each Role has 1 Run List per Environment (with a default).

Recipe

- There may be multiple Recipes.
- Each Recipe has 1 Cookbook.
- Each Recipe has 0 or more Resources.

Resource

- There may be multiple Resources.

Cookbook

- There may be multiple Cookbooks.
- Each Cookbook has 0 or more Recipes.
- Each Cookbook has 1 or more versions.

Attribute

- There are precedence rules for setting Attributes within a Recipe, Cookbook, Role, Environment or directly on the Node.

Data Bag

- There may be multiple Data Bags.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License


Cookbook Site API

Chef Essentials



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Essentials



Understanding and use of these essential components will provide you the capability to take advantage of the power of Chef.



Attributes

Node data such as the IP address, hostnam… …nel mod… …rsion of programming languages available on the system and more.

Cookbooks

The fundamental units of distribution in Ch… …ce… …rces you need to au…nate your infrastructu… …d are easily sharable with other Chef users.

Definitions

Allow the creation of new Resources by stringing together existing resources.

Libraries

Allow inclusion of arbitrary Ruby code, either to extend Chef's language or to implement your own classes directly. They are the secret sauce that will allow you to plug in to your existing infrastructure and utilize it to inform how your systems are configured.

Metadata

A small amount of meta-data is required in cookbooks. This information is used to provide hints to the Chef Server as to what cookbooks should be deployed to a given node, and in the future it will be integral to an automated system for discovering and installing cookbooks.

Recipes

The fundamental configuration in Chef. Recipes encapsulate collections of Resources which are executed in the order defined to configure the Nodes.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Templates	Files written in a markup language that allows one to dynamically generate the file's final content based on variables or more complex logic. Templates are commonly used to manage configuration files with Chef.
Chef Repository	The place where cookbooks, roles, config files and other artifacts for managing systems with Chef will live.
Data Bags	Provide an arbitrary stores of globally available JSON data.
Environments	Provide a mechanism for managing different environments such as production, staging, development, and testing, etc with one Chef setup (or one organization on Hosted Chef).
Exception and Report Handlers	A feature of Chef that allow you to run code in response to a Chef run succeeding or failing.
Lightweight Resources and Providers (LWRP)	Resource and providers whose implementation is quick and easy, requiring less Ruby knowledge than their heavier counterparts.
Nodes	Hosts that runs the Chef client. The primary features of a node, from Chef's point of view, are its Attributes and its run list. Nodes are the thing that Recipes and Roles are applied to.
Providers	Take a resource, compare that resource to the current state of the part of the system it is managing, and then takes the Action specified in the resource. They are the way that Chef supports multiple platforms with a single Resource.
Resources	A cross platform abstraction of the thing you're configuring on the host. For example, packages may be installed via apt, yum, or the BSD ports and packages systems, but the package resource abstracts these differences away so you can specify that a package should be installed in a cross-platform way. Chef's resources are mostly just containers for data, with some basic validation functionality.
Roles	Provide a means of grouping similar features of similar nodes, providing a mechanism for easily composing sets of functionality. At web scale, you almost never have just one of something, so you use roles to express the parts of the configuration that are shared by a group of Nodes.
Search	A feature of the Chef Server that allows you to use a full-text search engine (based on Apache Solr) to query information about your infrastructure and applications. Searches are built by the Chef Server, and allow you to query arbitrary data about your infrastructure.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 Chef Concepts as UML

Attributes 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Attributes



Overview

Attributes are Node data such as the IP address, hostname, loaded kernel modules, version of programming languages available on the system and more. New attributes can be dynamically added to the node in a variety of ways.

During the Chef run, the Chef Client saves these node attributes on the Chef Server where they are indexed for Search. When the Chef Client runs again, it will retrieve the attributes that were saved previously and merge in attributes based on the priority rules described below.

Attribute Type and Precedence

There are three types of attributes in order of precedence, highest to lowest.

- override
- normal
- default

Write your cookbooks with default attributes, but override these with role-specific or node-specific values as necessary.



A fourth attribute type, `automatic`, contains Ohai data and has the highest precedence.

Chef doesn't provide any way to modify automatic attributes because any modifications you make will be overwritten with Ohai data again on the next run. Review **Automatic Attributes** for those reserved namespaces.

Attributes are a special key-value store called a **Mash** within the Ruby DSL context. A **Mash** is just a **Hash** where the key can be either a Ruby symbol (:key) or a string ("key"). The keys are also auto-vivified into accessor methods, which we'll talk about in a moment.

Attributes that Aren't
A number of pieces of data related to a node are methods rather than attributes. As such, they cannot be accessed using the syntax for attributes, but rather must be accessed as method. Important node methods include:

```
node.name
node.run_list
node.chef_environment
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Attribute Persistence

At the beginning of each `chef-client` run, `default`, `override`, and `automatic` attributes are completely reset.
They are then rebuilt using the currently applicable cookbooks, recipes, roles, environment and Ohai data.
Thus, default and override attributes set in cookbook attribute files, roles, recipes, or environments will disappear once the recipe, role, or environment is removed from the node and chef-client is run on the node.

0.10.x behavior
The behavior described in this section refers to the behavior of Chef 0.10+.

`Normal` attributes are not reset.
Rather, on each chef-client run, any new attributes passed to chef-client in a JSON file are merged with the existing `normal` attributes in the node's data(using Deep Merge).
This means that any `normal` attribute set in a recipe or cookbook attribute file will remain even after the cookbook or role has been removed from the node's run list.

Setting Attributes

Attributes may be set on the node from the following objects

- cookbooks
- environments *(Chef 0.10.0 or above only)*
- roles
- nodes

Precedence

The precedence of the attributes is as follows, from low to high:

- `default` attributes applied in an attributes file
- `default` attributes applied in an environment
- `default` attributes applied in a role
- `default` attributes applied on a node directly in a recipe
- `normal` or `set` attributes applied in an attributes file
- `normal` or `set` attributes applied on a node directly in a recipe
- `override` attributes applied in an attributes file
- `override` attributes applied in a role
- `override` attributes applied in an environment
- `override` attributes applied on a node directly in a recipe

`default` attributes applied in an attributes file have the lowest priority and `override` attributes applied on a node directly in a recipe have the highest priority.

Write your cookbooks with default attributes, but override these with role-specific or node-specific values as necessary.
See examples here...

Again, the exception to this are `automatic` attributes. They have the highest precedence and may not be modified, as they are regenerated by Ohai every time Chef runs.

Notation

Attributes are a special key-value store called a **Mash** within the Ruby DSL context. A **Mash** is just a **Hash** where the key can be either a Ruby symbol (:key) or a string ("key"). It's easier to stick with string notation if you are just starting out with Chef and/or Ruby. This allows you to just "quote it", rather than also having to learn about special cases, name space overlap (and method_missing), character constraints, and escaping work at the beginning.

That way, by the time you may want to use symbols stylistically or contextually, you can learn to wield them with an existing base of knowledge. Our examples and detail on attributes use a string notation for that reason. If you already know your way around symbols in ruby, you may of course use them.

Cookbook Attributes

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook attribute files are found in the `attributes` subdirectory of the cookbook. They are evaluated in the context of the Node object and Node methods are used to set attribute values.

e.g. from Opscode's Apache cookbook:

cookbooks/apache2/attributes/default.rb

The use of the node object is implicit here. The following is equivalent:

cookbooks/apache2/attributes/default.rb

Attributes can be set in a recipe as well, but `node` must be used.

Cookbook Attribute Methods

Use the following methods within a cookbook's attributes file or in a recipe. They correspond to the attribute type of the same name (set is an alias for normal).

- `override`
- `default`
- `normal` (or `set`)

Additionally, there are `_unless` methods available. See the end of this document for information on conditionally setting attributes

Another handy method available related to attributes is the `attribute?` method. This will check for the existence of an attribute, so you can do processing in an attributes file or recipe only if a specific attribute exists.

attribute?() in attributes file

attribute?() in recipe

In the recipe, we need to use the method on the node object. In the attributes file, the node object is implicit. In either, we can also look for a sub-key of an attribute by chaining the attribute as methods:

attribute?() in recipe

Cookbook Attribute File Ordering

When Chef loads cookbook attribute files, it does so in alphabetical order for all the cookbooks. If you need to ensure that one attribute file is loaded before another (for example, if your Rails cookbook requires that the Apache attributes are available first) you can use the `include_attribute` method, like so:

include_attribute

This loads `apache/attributes/default.rb` before continuing the processing of the current attribute file.

The syntax for this follows the same "double colon" pattern as `include_recipe`, so a statement like:

include_attribute

This loads the `attributes/tunables.rb` file in a rails cookbook.

Reloading Attribute Files From Recipes

Attributes sometimes depend on actions taken from within recipes, so it may be necessary to reload a given attribute from within a recipe. For example: if you have an attribute that reads firewall rules, and a recipe that installs a firewall package, the firewall attributes will not be set the first time you execute the cookbook. Since `include_attribute` is not available from inside recipes, you will need to manually reload your firewall::default attribute:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

reloading attributes from recipes

Attribute Accessor Methods

Attribute accessor methods are automatically created and the method invocation can be used interchangeably with the keys. The following is equivalent to the usage above:

cookbooks/apache2/attributes/default.rb

This is a matter of style, and may be seen when "retrieving" the value of an attribute.

Environment Attributes

Environments can set `default` and `override` attributes.

This is done with the `default_attributes` and `override_attributes` methods (respectively) in an Environment's Ruby DSL file, or the `default_attributes` and `override_attributes` hashes in the Environment's JSON data. It is common to assign attributes that pertain to a particular environment. For example, the external load balancer's public DNS may be different in "production" than in "staging".

Chef 0.10.0 or above
Environments attributes are only available in Chef 0.10.0 or above.

Role Attributes

Roles can only set `default` and `override` attributes, they cannot set `normal` attributes. This is done with the `default_attributes` and `override_attributes` methods in a Role's Ruby DSL file, or the `default_attributes` and `override_attributes` hashes in the Role's JSON data.

You can have deep merged Attributes
Deep merging allows you to layer role attributes.

It is common to assign attributes that pertain to a particular role. For example, a `php_apache2_server` role might use different tuning parameters for the same apache attributes than a `mod_perl_apache2_server`.

Node Attributes

Finally, the node object can be modified directly to set the attributes. Typically, this sets attributes at the `normal` priority level, and can be done by editing the node with knife or through the WebUI, or by passing JSON data to the node.

JSON Attributes

You can also specify node attributes with a JSON file. These are applied at the `normal` priority level.

```
chef-client --help
[...]
    -j JSON_ATTRIBS                  Load attributes from a JSON file or URL
        --json-attributes
```

For example, to set up some different ports for Apache to listen on:

JSON attribute example

```
{ "apache": { "listen_ports": ["81", "8080"] } }
```

Remember, that attributes passed via JSON file are merged with those stored on node and actually there is no way to override them in that way, however if there is a conflict, attributes from JSON file will win with those stored on node.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Automatic Attributes

This fourth attribute type cannot be modified, as any modifications you make will be overwritten with Ohai data again on the next run.

How to Use Attributes

Good Candidates for Attributes

- Cross-Platform abstractions for an application such as the path to a config file.
- Default values for "tunable" settings such as memory assigned to processes or number of workers to spawn.
- Anything else you may want to persist (in node data) between Chef runs.

Usage Best Practices

The general pattern for attributes precedence is that cookbooks and roles should be setting defaults.

- If you need to change the values for a specific node, use, the normal attributes on the node.
- Overrides are there so roles can force a certain value even if the node already has a value there.

There are certainly other ways to use it, but that is the pattern it was designed for.

Setting Attributes at the Same Precedence Level

A common use case is to set default attributes in a cookbook's attribute files, and also set the same default attributes, but with different values, using a role. In this scenario, the attributes set in the role will be deep merged on top of the attributes from the attributes file. *The attributes set by the role will win if there is a conflict.*

Setting a Value Only If the Attribute Has No Value

In attribute files, you can also set a value only if no value is currently set for that attribute using the `_unless` variants of the attribute priority methods: `default_unless`, `set_unless`, and `override_unless`. These can be handy in some use cases, but **be careful!**

When you use these methods, *the attributes applied to your nodes will become out-of-sync with the values in your cookbooks* whenever you update your cookbooks. This means that building a new node with an identical set of recipes and roles as an existing node could result in a different configuration--a problem that can be frustrating to debug. For this reason, you should avoid using the `_unless` methods whenever possible.

 Chef Essentials

Automatic Attributes 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Automatic Attributes



Review

***Attributes** are **Node** data such as the IP address, hostname, loaded kernel modules, version of programming languages available on the system and more.*

*During the **Chef run**, the **Chef Client** saves node attributes on the **Chef Server** where they are indexed for **Search**.*

Use of Automatic Attributes in Recipes

Node attributes can be set in recipes. This use of node attributes should do done when you want to calculate a derived value, or store some data on the node that should be persisted the next time Chef runs. For more information on the use within recipes, see **Node Attributes** within Recipes.

Automatic Attribute Precedence

Attributes are applied in precedence order - with the fourth attribute type, `automatic`, containing Ohai data and having the highest precedence.
As these `automatic` attributes will be re-written with each Ohai run - Chef doesn't provide any way to modify them.

The following attributes are those which are re-written with each Ohai run, so they should be recognized as being unmodifiable when considering the use of attributes.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
ohai$ grep -R "provides" -h lib/ohai/plugins|sed 's/^\s*//g'|sed "s/\\\"/\'/g"|sort|uniq|grep
^provides
provides 'block_device'
provides 'chef'
provides 'cloud'
provides 'command'
provides 'command/ps'
provides 'cpu'
provides 'dmi'
provides 'ec2'
provides 'etc', 'current_user'
provides 'eucalyptus'
provides 'filesystem'
provides 'fqdn', 'domain'
provides 'fqdn', 'hostname'
provides 'hostname', 'fqdn'
provides 'kernel'
provides 'kernel/os'
provides 'keys'
provides 'keys/ssh'
provides 'languages'
provides 'languages/c'
provides 'languages/erlang'
provides 'languages/groovy'
provides 'languages/java'
provides 'languages/lua'
provides 'languages/mono'
provides 'languages/perl'
provides 'languages/php'
provides 'languages/python'
provides 'languages/ruby'
provides 'lsb'
provides 'memory'
provides 'network'
provides 'network', 'counters/network'
provides 'network_ip_scope', 'privateaddress'
provides 'network/listeners'
provides 'ohai'
provides 'ohai_time'
provides 'os', 'os_version'
provides 'platform', 'platform_version'
provides 'platform', 'platform_version', 'platform_build'
provides 'rackspace'
provides 'system_profile'
provides 'uptime', 'idletime', 'uptime_seconds', 'idletime_seconds'
provides 'uptime', 'uptime_seconds'
provides 'virtualization'
```

You'll note that you can get this list directly yourself by running the command that produced it.

```
ohai$ grep -R "provides" -h lib/ohai/plugins|sed 's/^\s*//g'|sed "s/\\\"/\'/g"|sort|uniq|grep
^provides
```

Additional attributes and sub-attributes can be viewed on a particular node by running `ohai`. This returns data as JSON.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Attributes

Deep Merge



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Deep Merge



Overview

Chef does deep merging of mutilevel hashes and arrays for role and environment attributes. These can assist in code reuse by allowing you to layer your configuration among many roles, while still making use of default attributes set at the cookbook level.

See Attributes for more background information.

Examples

Simple merge

First our lowest level server role

roles/baseline.rb

```
name "baseline"
description "The most basic role for all configurations"
run_list "recipe[baseline]"

override_attributes(
  :apache => {
    :listen_ports => [ 80 ],
    :prefork => {
      :startservers => 20,
      :minspareservers => 20,
      :maxspareservers => 40
    }
  }
)
```

next our more customized web server role which uses the baseline role

roles/web.rb

```
name "web"
description "Web server config"
run_list "role[baseline]"

override_attributes(
  :apache => {
    :prefork => {
      :startservers => 30
    }
  }
)
```

then using a recipe like

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

cookbooks/baseline/recipes/default.rb

```
include_recipe "apache2"

Chef::Log.info(node['apache']['prefork'].to_hash)
```

and a run_list set to

run_list/web.json

```
{
 "run_list": [ "role[web]" ]
}
```

produces

```
[Tue, 16 Aug 2011 14:44:26 -0700] INFO: {"startservers"=>30, "minspareservers"=>20,
"maxspareservers"=>40, "serverlimit"=>400,
"maxclients"=>400, "maxrequestsperchild"=>10000}
```

which mixes the default attributes from cookbooks/apache2/attributes/default.rb, the overrides from baseline.rb and the final overrides from web.rb quite nicely, setting startservers to 30.

With this in mind the examples below will use short hand.

Substitution

Hash - Substitute an existing string

```
role 1 { :x => "1", :y => "2" }
+
role 2 { :y => "3" }
=
{ :x => "1", :y => "3" }
```

Hash - Substitute an existing boolean

```
role 1 { :x => true, :y => false }
+
role 2 { :y => true }
=
{ :x => true, :y => true }
```

Array - Substitute an existing string

```
role 1 [ "1", "2" ]
+
role 2 [ "!merge:2", "3" ]
=
[ "1", "3" ]
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Substitute an Array with a Hash

```
role 1 [ "1", "2", "3" ]
+
role 2 { :x => "1" , :y => "2" }
=
{ :x => "1", :y => "3" }
```

when items can't be merged the original data is overwritten

Addition

Hash - Add a string

```
role 1 { :x => "1", :y => "2" }
+
role 2 { :z => "3" }
=
{ :x => "1", :y => "2", :z => "3" }
```

Array - Add a string

```
role 1 [ "1", "2" ]
+
role 2 [ "3" ]
=
[ "1", "2", "3" ]
```

Hash - Multilevel Add

```
role 1 { :x => { :y => "2" } }
+
role 2 { :x => { :z => "3" } }
=
{ :x => { :y => "2", :z => "3" } }
```

Array - Multilevel Add

```
role 1 [ [ 1, 2 ] ]
+
role 2 [ [ 3 ] ]
=
[ [ 1, 2 ], [ 3 ] ]
```

Subtraction

Subtraction using !merge is only valid between roles. It will not work in environment or file attributes.



Subtraction using !merge is likely to be removed in the future, potentially in Chef 0.11. See the discussion here for more information.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Hash - Remove a string

```
A hash cannot be removed during a merge
```

Array - Remove a string

```
role 1 [ "1", "2", "3" ]
+
role 2 [ "!merge:3" ]
=
[ "1", "2" ]
```

Array - Remove all values

```
role 1 [ "1", "2", "3" ]
+
role 2 [ "!merge:" ]
=
[]
```

Array - Remove an integer

```
role 1 [ 4, 5, 6 ]
+
role 2 [ "!merge:", 5, 6 ]
=
[ 5, 6 ]
```

You can't use !merge:<value> with integers, to get rid of 4 you'll have to dump them all.

 Automatic Attributes

Cookbooks 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbooks



Cookbooks are the fundamental units of distribution in Chef.

They encapsulate all the resources you need to automate your infrastructure and are easily sharable with other Chef users.

What's in a Cookbook?

Cookbooks contain:

- **Attributes** that are values on Node to set default values used elsewhere in the cookbook.
- **Definitions** that allow you to create reusable collections of one or more Resources.
- **File Distribution** that are transferred to your Chef-administered machines via Cookbook File resource.
- **Libraries** that extend Chef or provide helpers with Ruby code.
- **Recipes** that specify Resources to manage, in the order they should be managed.
- **Lightweight Resources and Providers (LWRP)** that allow you to create your own custom resources and providers.
- **Templates** that are rendered on Chef-configured machines with your dynamically substituted values. Think config files on steroids, then read ERB templates.
- **Metadata** that tells Chef about your recipes, including dependencies, version constraints, supported platforms and more.

Each of these components is a directory or file. An example cookbook directory after running the `cookbook create` knife command:

Cookbook Contents

```
attributes/
definitions/
files/
libraries/
metadata.rb
providers/
README.rdoc
recipes/
resources/
templates/
```

You develop cookbooks on your local system in the Chef Repository, in the `cookbooks/` sub-directory.

How do I work with cookbooks?

We recommend you develop cookbooks in the **Chef Repository**. You can create your own cookbooks, or you can download other people's shared cookbooks.

Then, you can upload your cookbooks to your Chef Server or to the Opscode Community Cookbooks site through **Managing Cookbooks With Knife**.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

How do I create a new Cookbook?

You can create a new cookbook in your Chef Repository `cookbooks/` sub-directory with Knife.

```
% knife cookbook create MYCOOKBOOK
** Creating cookbook MYCOOKBOOK
** Creating README for cookbook: MYCOOKBOOK
** Creating metadata for cookbook: MYCOOKBOOK
```

Then start editing the various components, such as the default recipe, as needed. (See **Guide to Creating A Cookbook and Writing A Recipe** f or an additional examples.)



Updates on the Opscode Cookbooks Project

Check out the **Opscode Blog** for recent updates to Opscode provided cookbooks, and future plans.

Tutorials from the Community

Nagios and Chef at Fotopedia

Fotopedia uses Chef in the management of their infrastructure. Among many other things, Chef generates the Nagios configuration for all their services. Their Fotopedia Labs blog entry: **Nagios and Chef at Fotopedia** includes extensive detail on the use of recipes, roles, ruby blocks, helpers and other Chef components - all within their Nagios Cookbook.

Where can I find some Cookbooks?

Opscode maintains the **Chef Community Cookbook Site** as a location for finding and sharing Chef cookbooks. It includes dedicated listings for each cookbook, is easy to navigate and can be accessed via a RESTful API. Opscode publishes all our cookbooks there and encourages others to do the same. Cookbook Site Help includes access to help articles for the Chef Community Cookbook Site.

You can download cookbooks from the cookbooks site through the API using knife. This will retrieve a tar.gz of the cookbook that you can extract into the `cookbooks/` directory of the Chef Repository.

```
% knife cookbook site download COOKBOOK
```

If you are using Git, you can have cookbooks downloaded, then automatically extracted into `cookbooks/` and tracked with their own branch using:

```
% knife cookbook site install COOKBOOK
```

For more information about this command see Chef Repository#cookbooks.

Github Repositories

Opscode's cookbooks on the Chef Community site are considered "released", and the cookbooks repository on Github is considered

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

"development." *Opscode recommends that users download cookbooks from the Community site instead of using the Github repository, as changes may be incompatible with your version of Chef.*

Beyond the Opscode development Github repository, some members of the Chef Community have only shared their cookbooks on Github. Some repositories that are available:

Repository	Description	Maintainer
http://github.com/opscode/cookbooks	Recipes created by Opscode	Opscode
http://github.com/37signals/37s_cookbooks	37 Signals Repository	37 Signals
http://github.com/engineyard/ey-cloud-recipes	EY Cloud Recipes	Engine Yard

You can also discover other repositories on github with the network graph.

Cookbook Development Workflow

See Working with Git and Cookbooks for a development workflow for Chef cookbooks, using Git as the version control system.
If you are not using Git, you should be able to review these details and adapt your approach for the specifics of your version control system.

How do I upload cookbooks to the Chef Server?

Use knife cookbook upload sub-command. There is no difference in use for Open Source Chef Server, Hosted Chef or Private Chef.

To upload a single named COOKBOOK:

```
% knife cookbook upload COOKBOOK
```

To upload multiple cookbooks, as the command parses the name args as 1..n cookbook names:

```
% knife cookbook upload COOKBOOK COOKBOOK COOKBOOK ...
```

To upload all cookbooks:

```
% knife cookbook upload -a
```

Use of Knife to Manage Your Cookbooks



Knife is a powerful command line interface for Chef.

See **Managing Cookbooks With Knife** for additional detail to that presented here on using Knife to manage your cookbooks.

If you're using Chef Solo, you need to copy all the cookbooks necessary to the system that will be running Chef Solo. See the Chef Solo page for more information.

How do I customize existing cookbooks?

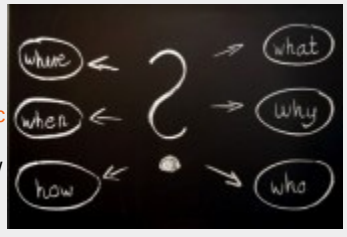
When you download cookbooks from the Community Site or Github repositories, you may wish to make changes. Using the "cookbook site install" knife command above makes this really easy.

Simply modify the cookbook as desired, and if you want to retrieve a new release of the cookbook, you can simply run the command again, and your changes will be preserved.

Need Help With the Community Site?

Cookbook Site Help is the place to go.

If you are using one of the Opscode Cookbooks, **Cookbook Support** provides detail on how



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

they are built and maintained -
and where to turn with any questions.

```
% knife cookbook site install COOKBOOK
% vi cookbooks/COOKBOOK/recipes/default.rb
% git add cookbooks/COOKBOOK/recipes/default.rb
% git commit -m 'made my own changes to COOKBOOK'
```

Then, as the maintainer of COOKBOOK releases a new version, you could merge the new version into your changes.

```
% knife cookbook site install COOKBOOK
```

If there are any conflicts, git will let you know, and you can resolve those and commit the changes.

Changing the location of your cookbooks

You can specify a cookbook path using the `-o` option after the cookbook subcommand. For example,

```
knife cookbook upload -o /path/to/cookbook mycookbookname
```

Alternatively, you can specify the location of your cookbooks by changing `cookbook_path` in your knife.rb.

Site Specific Cookbooks

You can also make your own site specific copies of cookbooks.

~/.chef/knife.rb or ~/chef-repo/.chef/knife.rb

```
cookbook_path      [ "chef-repo/cookbooks", "chef-repo/site-cookbooks" ]
```

Next, copy the entire contents of the cookbook, go forth and customize it, then upload the cookbook(s) to the server. When Chef runs, it will only use the cookbook from `site-cookbooks`, not the one in `cookbooks`.

For example, say you have:

```
chef-repo/cookbooks/djbdns
chef-repo/site-cookbooks/djbdns
```

When the cookbook is uploaded, Knife will use the cookbook in `chef-repo/site-cookbooks/djbdns`.

Customizing Templates and Files

If you would like to customize just the files or templates used by a cookbook, you can create just those as well, copying them over from the upstream version and making your local changes. For example, you're deploying OpenLDAP and want to customize the slapd.conf and add your own certificates.

Assuming you've followed along with the Chef Repository and have created the ldap certificate:

```
mkdir -p site-cookbooks/openldap/files/default/ssl
mkdir -p site-cookbooks/openldap/templates/default
cp certificates/ldap.example.com.pem site-cookbooks/openldap/files/default/ssl
cp cookbooks/openldap/templates/default/slapd.conf.erb
site-cookbooks/openldap/templates/default/slapd.conf.erb
vi site-cookbooks/openldap/templates/default/slapd.conf.erb
```

Make changes, update the repository and install the cookbooks, and when Chef runs, it will get the certificte and slapd.conf from the

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

site-cookbooks, but otherwise use the rest of the openldap cookbook.

Other Site-specific Cookbooks

You can also use `site-cookbooks` for setting up other site-specific cookbooks

chef-repo/site-cookbooks/web_server/recipes/default.rb

```
include_recipe "apache2"
include_recipe "php::php5"

apache_site "php_server.conf" do
  enable :true
end
```

Then add "web_server" to `recipes` for the node in the webui, and it will apply the configuration. Prior to Chef 0.7.0, you might have a cookbook that merely includes several other recipes/cookbooks. Now you'll use Roles to define that.

Cookbook Dependencies

The Chef Server tries to only distribute the cookbooks that are needed to configure each individual Node. In order to do that, we take the list of Roles and Recipes that are assigned directly to that system, expand the list of dependencies for them, and then ship that set to the Node.

If there is a dependency on a particular cookbook being in place in order to complete a configuration, edit the template metadata.rb file to specify that dependency through the 'depends' field. Metadata has details on this field, and the other fields that are available to you in the metadata.rb file.

Whenever you include a recipe in a cookbook via "include_recipe" you need to add the included cookbook to the depends list.

Use Case Study

In our environment we don't have only Windos / Ubuntu/Debian /RH servers. So my problem is that we want to create a maintenance role for updating these servers, but the server name doesn't describe the function or the OS running on it. Is there a way to create "server-groups" which include only windows servers or something like this because with these groups we can easily put a the correct "maintenance-role" (cookbooks for maintenance) to them if they are needed?

Probably the best and most direct way to go about this is to have a single "Maintenance Cookbook" as part of every system, which will selectively include recipes based on platform. It'd look something like this:

```
case node.platform
    when :debian
      include_recipe("maintenance::debian")
    when :ubuntu
      include_recipe("mainentance::ubuntu")
    else
      # ...
    end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook Attributes



Overview

***Attributes** are **Node** data such as the IP address, hostname, loaded kernel modules, version of programming languages available on the system and more. New attributes can be dynamically added to the node in a variety of ways.*

During the Chef run, the Chef Client saves these node attributes on the Chef Server where they are indexed for Search. When the Chef Client runs again, it will retrieve the attributes that were saved previously and merge in attributes based on the priority rules described below.

Setting Attributes

Attributes may be set on the node from the following objects

- cookbooks
- environments (Chef 0.10.0 or above only)
- roles
- nodes

Precedence

The precedence of the attributes is as follows, from low to high:

- `default` attributes applied in an attributes file
- `default` attributes applied in an environment
- `default` attributes applied in a role
- `default` attributes applied on a node directly in a recipe
- `normal` or `set` attributes applied in an attributes file
- `normal` or `set` attributes applied on a node directly in a recipe
- `override` attributes applied in an attributes file
- `override` attributes applied in a role
- `override` attributes applied in an environment
- `override` attributes applied on a node directly in a recipe

`default` attributes applied in an attributes file have the lowest priority and `override` attributes applied on a node directly in a recipe have the highest priority.

Write your cookbooks with default attributes, but override these with role-specific or node-specific values as necessary.
See examples here...

The exception to this are Automatic Attributes. *They have the highest precedence and may not be modified, as they are regenerated by Ohai ever y time Chef runs.*

Notation

Attributes are a special key-value store called a **Mash** within the Ruby DSL context. A **Mash** is just a **Hash** where the key can be either a Ruby symbol (:key) or a string ("key"). It's easier to stick with string notation if you are just starting out with Chef and/or Ruby. This allows you to just "quote it", rather than also having to learn about special cases, name space overlap (and method_missing), character constraints, and escaping work at the beginning.

That way, by the time you may want to use symbols stylistically or contextually, you can learn to wield them with an existing base of knowledge. Our examples and detail on attributes use a string notation for that reason. If you already know your way around symbols in ruby, you may of course use them.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook Attributes

Cookbook attribute files are found in the `attributes` subdirectory of the cookbook. They are evaluated in the context of the Node object and Node methods are used to set attribute values.

e.g. from Opscode's Apache cookbook:

cookbooks/apache2/attributes/default.rb

The use of the node object is implicit here. The following is equivalent:

cookbooks/apache2/attributes/default.rb

Attributes can be set in a recipe as well, but `node` must be used.

Cookbook Attribute Methods

Use the following methods within a cookbook's attributes file or in a recipe. They correspond to the attribute type of the same name (set is an alias for normal).

- `override`
- `default`
- `normal` (or `set`)

Additionally, there are `_unless` methods available. See the end of this document for information on conditionally setting attributes

Another handy method available related to attributes is the `attribute?` method. This will check for the existence of an attribute, so you can do processing in an attributes file or recipe only if a specific attribute exists.

attribute?() in attributes file

attribute?() in recipe

In the recipe, we need to use the method on the node object. In the attributes file, the node object is implicit. In either, we can also look for a sub-key of an attribute by chaining the attribute as methods:

attribute?() in recipe

Cookbook Attribute File Ordering

When Chef loads cookbook attribute files, it does so in alphabetical order for all the cookbooks. If you need to ensure that one attribute file is loaded before another (for example, if your Rails cookbook requires that the Apache attributes are available first) you can use the `include_attribute` method, like so:

include_attribute

This loads `apache/attributes/default.rb` before continuing the processing of the current attribute file.

The syntax for this follows the same "double colon" pattern as `include_recipe`, so a statement like:

include_attribute

This loads the `attributes/tunables.rb` file in a rails cookbook.

Reloading Attribute Files From Recipes

Attributes sometimes depend on actions taken from within recipes, so it may be necessary to reload a given attribute from within a recipe. For example: if you have an attribute that reads firewall rules, and a recipe that installs a firewall package, the firewall attributes will not be set the first

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

time you execute the cookbook. Since `include_attribute` is not available from inside recipes, you will need to manually reload your firewall::default attribute:

reloading attributes from recipes

Attribute Accessor Methods

Attribute accessor methods are automatically created and the method invocation can be used interchangeably with the keys. The following is equivalent to the usage above:

cookbooks/apache2/attributes/default.rb

This is a matter of style, and may be seen when "retrieving" the value of an attribute.

 Cookbooks

Setting Attributes (Examples) 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Setting Attributes (Examples)

Setting Attributes (Examples)

`default` attributes applied in an attributes file have the lowest priority and `override` attributes applied on a node directly in a recipe have the highest priority.

Again, the exception to this are Automatic Attributes, as these are regenerated by Ohai every time Chef runs.

The precedence of the attributes - from low to high

- `default` attributes applied in an attributes file
- `default` attributes applied in an environment
- `default` attributes applied in a role
- `default` attributes applied on a node directly in a recipe
- `normal` or `set` attributes applied in an attributes file
- `normal` or `set` attributes applied on a node directly in a recipe
- `override` attributes applied in an attributes file
- `override` attributes applied in a role
- `override` attributes applied in an environment
- `override` attributes applied on a node directly in a recipe



Default in ../cookbooks/../attributes/default.rb

Example Precedence #1

```
default["apache"]["dir"] = "/etc/apache2"
```

Default in ../environments/environment_name.rb

Example Precedence #2

```
default_attributes({ "apache" => {"dir" => "/etc/apache2"}})
```

Default in ../roles/rolename.rb

Example Precedence #3

```
default_attributes({ "apache" => {"dir" => "/etc/apache2"}})
```

Default in Node object in a recipe ../cookbooks/../recipe/default.rb

Example Precedence #4

```
node.default["apache"]["dir"] = "/etc/apache2"
```

Normal or Set in ../cookbooks/../attributes/default.rb

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Example Precedence #5

```
set["apache"]["dir"] = "/etc/apache2"

normal["apache"]["dir"] = "/etc/apache2"  #set is an alias of normal.
```

Setting a Normal/Set in a recipe ../cookbooks/../recipe/default.rb

Example Precedence #6

```
node.set["apache"]["dir"] = "/etc/apache2"
```

Override in ../cookbooks/../attributes/default.rb

Example Precedence #7

```
override["apache"]["dir"] = "/etc/apache2"
```

Override in .../roles/rolename.rb

Example Precedence #8

```
override_attributes({ "apache" => {"dir" => "/etc/apache2"}})
```

Override in ../environments/environment_name.rb

Example Precedence #9

```
override_attributes({ "apache" => {"dir" => "/etc/apache2"}})
```

Override in Node object (from recipe)

Example Precedence #10

```
node.override["apache"]["dir"] = "/etc/apache2"
```

Cookbook Attributes

Definitions

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Definitions

Definitions allow you to create new Resources by stringing together existing resources.



Definitions declarations must be placed in a definitions folder at the root of your cookbook. ***Do not declare them into a cookbook.***

Definitions aren't resources

They are replaced by the resources they contain. **You cannot notify a definition to take an action - you can only notify the resources it creates**. If you do need to send actions to the resource, you should look in to creating a new Provider.

When to Use Definitions

- You are repeating a pattern of resources
- You do not want to send actions directly to this resource - i.e., you never need to notify it
- You want to pass data from various recipes to one definition, to update your /etc/aliases, /etc/sudoers, or something similar, with entries from multiple recipes in a single chef run.

There are three components to a definition:
1. The resource name
2. The Prototype Parameters
3. The Params hash

You start with `define`, followed by the name of the new resource (`apache_site`). What follows is the Prototype Arguments for the definition - these are used to set default values for parameters within the definition.

Definition and Prototyping

```
define :apache_site, :action => :enable do
  ...
end
```

Those parameters are then accessed within your definition via the `params` hash. Since this is a hash, each parameter must have a value. If no default value is specified, use `nil`.

Hey, CGI programmers...
The params hash should look familiar to you - it's basically the exact same model we've been following ever since the CGI.pm days.

Example

This definition file creates a new resource `apache_site`:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

apache_site Definition

```
define :apache_site, :enable => true do
  include_recipe "apache2"

  if params[:enable]
    execute "a2ensite #{params[:name]}" do
      command "/usr/sbin/a2ensite #{params[:name]}"
      notifies :restart, resources(:service => "apache2")
      not_if do
        ::File.symlink?("#{node[:apache][:dir]}/sites-enabled/#{params[:name]}") or
          ::File.symlink?("#{node[:apache][:dir]}/sites-enabled/000-#{params[:name]}")
      end
      only_if do ::File.exists?("#{node[:apache][:dir]}/sites-available/#{params[:name]}") end
    end
  else
    execute "a2dissite #{params[:name]}" do
      command "/usr/sbin/a2dissite #{params[:name]}"
      notifies :restart, resources(:service => "apache2")
      only_if do ::File.symlink?("#{node[:apache][:dir]}/sites-enabled/#{params[:name]}") end
    end
  end
end
```

We utilize it by placing the definition in the recipe:

apache_site resource

```
# Enable my_site.conf
apache_site "my_site.conf" do
  enable true
end

# Disable my_site.conf
apache_site "my_site.conf" do
  enable false
end
```

We take the attributes of our new `apache_site` resource, and make them accessible via the `params` hash. This will be true of any attribute we set - you do not have to prototype all your parameters. Within the context of the Chef run, this definition will be replaced by all the resources you specify within the definition - in the "enabled" case, it is expanded to:

The resources created by this definition

```
execute "a2ensite my_site.conf" do
  command "/usr/sbin/a2ensite my_site.conf"
  notifies :restart, resources(:service => "apache2")
  not_if do
    ::File.symlink?("/etc/apache2/sites-enabled/my_site.conf") or
      ::File.symlink?("/etc/apache2/sites-enabled/000-my_site.conf")
  end
end
```

You can have as many resources in a definition as you want.

You can also pass data from various recipes to one definition.

Passing data from various recipes to one definition is possible. For example, you'd like to update your /etc/aliases, /etc/sudoers, or something similar, with entries from multiple recipes in a single chef run. Reopening resources is the way do this currently. Here's an example definition for

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

adding email aliases (via Mithrandir on #chef):

```
define :email_alias, :recipients => [] do
  execute "newaliases" do
    action :nothing
  end

  t = nil
  begin
    t = resources(:template => "/etc/aliases")
  rescue Chef::Exceptions::ResourceNotFound
    t = template "/etc/aliases" do
      source "aliases.erb"
      cookbook "aliases"
      variables({:aliases => {} })
      notifies :run, "execute[newaliases]"
    end
  end

  if not t.variables[:aliases].has_key?(params[:name])
    t.variables[:aliases][params[:name]] = []
  end
  t.variables[:aliases][params[:name]] << [ params[:recipients] ]
end
```

Use Case Scenario

I've two applications to deploy/run using chef on a single node under the same domain/subdomain. One is a Rails app and another is a Wordpress blog app. The Rails app will reside as the main app at say `example.com` and the wordpress at `blog.example.com`. And lets assume that I'll be using Apache2 or Nginx as the webserver to handle the VirtualHost. I can create 2 separate run_list adding different roles to it. Later, when I have to add another app on the same domain, e.g. `forum.example.com` on the same node, I can create a separate run_list for this. But how do I update/ammend the VirtualHost of the web-server since its only one per node?

Definitions are a feature of Chef cookbooks that allow one to create "macros" for resources so they can be reused. The `web_app` definition exists in our apache2 cookbook, and can be used like this:

```
web_app "blog_site" do
     server_name "blog"
     server_aliases [ "blog.#{node['domain']}", node['fqdn'] ]
     docroot "/srv/www/blog_site"
   end
```

When Chef processes the recipe containing this, it sees web_app and looks to see if it recognizes it as a "resource". It will find it as loaded from the Opscode apache2 cookbook, and then process the template according to how the definition was written.

It is important to note that definitions aren't actually resources, and to Chef they are replaced by the resources they contain. They cannot be signaled directly with a notification, though the contained resources can.

We also have a (relatively old) blog post about this specific definition.

 Setting Attributes (Examples)

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

File Distribution 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

File Distribution



Distribution

Often, you need to distribute files to your servers. The Files within a cookbook are where you can do just that.

They are reached via the cookbook_file resource.

A cookbook_file resource

```
cookbook_file "/usr/local/bin/apache2_module_conf_generate.pl" do
  source "apache2_module_conf_generate.pl"
  mode 0755
  owner "root"
  group "root"
end
```

The above resource would create `/usr/local/bin/apache2_module_conf_generate.pl` from the `apache2_module_conf_generate.pl` file contained in the cookbook that was most specific to the Node executing the recipe. By default, it looks for File Specificity from the remote_url setting in the Client configuration.

File Specificity

Cookbooks will work on multiple platforms frequently, or distribute files that are only subtly different depending on the platform. Chef Cookbooks allow you to only specify those files one time, but make sure that the right file ends up on each system. Within a Cookbook's `files` or `templates` directories, you might find a directory structure like this:

- files/ (or templates/)
  - host-foo.example.com
  - ubuntu-8.04
  - ubuntu-8
  - ubuntu
  - default

Given this resource:

A cookbook_file resource

```
cookbook_file "/usr/local/bin/apache2_module_conf_generate.pl" do
  source "apache2_module_conf_generate.pl"
  mode 0755
  owner "root"
  group "root"
end
```

We would match:

- host-foo.example.com/apache2_module_conf_generate.pl
- ubuntu-8.04/apache2_module_conf_generate.pl
- ubuntu-8/apache2_module_conf_generate.pl
- ubuntu/apache2_module_conf_generate.pl
- default/apache2_module_conf_generate.pl

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

In that order (given a Node named foo.example.com running Ubuntu 8.04). Then for example, if we put `apache_module_conf_generate.pl` under `files/host-foo.example.com/` directory it will be only copied to the machine with the domain name `foo.example.com`.

So, the rule distilled:

1. `host-node[:fqdn]`
2. `node[:platform]-node[:version]`
3. `node[:platform]-version_components`: The version string is split on decimals and searched from greatest specificity to least; for example, if the location from the last rule was centos-5.7.1, then centos-5.7 and centos-5 would also be searched.
4. `node[:platform]`
5. `default`



0.10.6
The searching of version components (e.g. searching centos-5 in addition to centos-5.6) was added in Chef 0.10.6.



Host Notation
The **host-** is literal. If your host was `foo.example.com`, then your folder needs to be named `host-foo.example.com`.



Hey, Cfengine people!
This is inspired by Single Copy Nirvana - if you think about it that way, this will click immediately.

When does a file get transferred?

When you run a cookbook_file resource, the client will calculate the checksum of local file. It compares this checksum against the checksum of the most specific available version of the file in the cookbook. If the checksums match, the file will not be transferred. Additionally, cookbook files are loaded lazily from the server--a cookbook may contain many cookbook files, but they are only transferred to the client as they are needed.


Definitions

Libraries


 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Libraries



Libraries allow you to include arbitrary Ruby code, either to extend Chef's language or to implement your own classes directly.

They are the secret sauce that will allow you to plug in to your existing infrastructure and utilize it to inform how your systems are configured.

Libraries are defined in your_cookbook/libraries/library_name.rb.

- Any libraries you include in a Cookbook will automatically be required, and available within all your Recipes, Attribute Files, Providers and Definitions.
- What this means is that the library files will automatically be read - however you may need to do something else to actually use them, depending on their contents.

Here's a simple example, showing how you'd use module defined in a Library in your recipe:

your_cookbook/libraries/your_example_library.rb

```
# define a module to mix into Chef::Recipe
module YourExampleLibrary
  def your_function()
    # ... do something useful
  end
end
```

your_cookbook/recipes/default.rb

```
# open the Chef::Recipe class and mix in the library module
class Chef::Recipe
  include YourExampleLibrary
end

your_function()
```

Basic Libraries

Lets say you have a customer record you are storing in an attribute file, that looks like this:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

A customer attribute file

```
mycompany_customers({
  :bob => {
    :homedir => "/home/bob",
    :webdir => "/home/bob/web"
  }
}
)
```

In a recipe, you could say:

Simple recipe

```
directory node[:mycompany_customers][:bob][:webdir] do
  owner "bob"
  group "bob"
  action :create
end
```

But that feels verbose, and you are going to do it all the time. You would rather say:

Advanced recipe

```
directory customer(:bob)[:webdir] do
  owner "bob"
  group "bob"
  action :create
end
```

You could achieve that with a simple library, like this:

Library to extend Chef::Recipe

```
class Chef
  class Recipe
    # A shortcut to a customer
    def customer(name)
      @node[:mycompany_customers][name]
    end
  end
end
```

Lets say you wanted some code to easily loop over every customer, like this:

Loop over customers

```
all_customers do |name, info|
  directory info[:webdir] do
    owner name
    group name
    action :create
  end
end
```

You could have a library like this:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Loop over customers

```
class Chef
  class Recipe
    def all_customers(&block)
      @node[:mycompany_customers].each do |name, info|
        block.call(name, info)
      end
    end
  end
end
```

Advanced Library Usage

You can also use Libraries to connect up to databases, talk to LDAP, or do anything else you can do with Ruby.. which is basically anything.
If you are going to go too crazy, consider putting your libraries in a new namespace, and calling them directly from your Chef Recipes - that way, there won't be too much pollution of the Chef::Recipe namespace.

Here is an example of connecting to a database and returning a list of customer Virtual Hosts.

Virtual Hosts from a Database

```
require 'sequel'

class Chef::Recipe::ISP
  # We can call this with ISP.vhosts
  def self.vhosts
    v = []
    @db = Sequel.mysql(
      'web',
      :user => 'example',
      :password => 'example_pw',
      :host => 'dbserver.example.com'
    )
    @db[
      "SELECT virtualhost.domainname,
              usertable.userid,
              usertable.uid,
              usertable.gid,
              usertable.homedir
         FROM usertable, virtualhost
        WHERE usertable.userid = virtualhost.user_name"
      ].all do |query|
      vhost_data = {
        :servername   => query[:domainname],
        :documentroot => query[:homedir],
        :uid          => query[:uid],
        :gid          => query[:gid],
      }
      v.push(vhost_data)
    end
    Chef::Log.debug("About to provision #{v.length} vhosts")
    v
  end
end
```



This rocking example..
Was provided by Arjuna (fujin). He's awesome.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

You could then use this in a recipe like so:

Using ISP.vhosts in a Recipe

```
ISP.vhosts.each do |vhost|
  directory vhost[:documentroot] do
    owner vhost[:uid]
    group vhost[:gid]
    mode 0755
    action :create
  end

  directory "#{vhost[:documentroot]}/#{vhost[:domainname]}" do
    owner vhost[:uid]
    group vhost[:gid]
    mode 0755
    action :create
  end
end
```

 File Distribution

Metadata 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Metadata

Overview



Chef Cookbooks require you to specify a small amount of meta-data. This information is used to provide hints to the Chef Server as to what cookbooks should be deployed to a given node, and in the future it will be integral to an automated system for discovering and installing cookbooks.

To get started, you need to create a `metadata.rb` file at the top of your cookbook. This file provides a simple ruby DSL for building the actual metadata file (which is stored as JSON). If you create a new cookbook by using `knife cookbook create` command, a template metadata.rb file will be created for you.

metadata.rb

We never interpret the metadata file directly, we only ever use the compiled JSON file! Currently, the metadata JSON is generated for you when you upload the cookbook or run the `knife cookbook metadata COOKBOOK` command. It's a good practice to always edit metadata.rb but not the JSON file, but if you need to edit metadata.json directly, remember that it will be overwritten by metadata.rb next time when you generate metadata or upload cookbook using `knife`, if metadata.rb exists.

At the moment, we utilize the following fields in the metadata:

- maintainer
- maintainer_email
- license
- description
- long_description
- depends
- version
- recipe

Meaningful Fields

The following fields are used by the Chef Server, WebUI, or Opscode Community Site.

name

The name of the cookbook. Normally this is inferred, but you can override it in the cookbook metadata.

name example
`name 'cats'`

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Did you get sent here by an error message?

If so, here is the deal. The Chef Server tries to only distribute the cookbooks that are needed to configure each individual Node. In order to do that, we take the list of Roles and Recipes that are assigned directly to that system, expand the list of dependencies for them, and then ship that set to the Node.

If you forget to specify a dependency in a cookbooks metadata, then the Chef Server will not realize that the cookbook is necessary to complete your configuration... resulting in the error message that brought you here. To fix it, simply add the right 'depends' entry to your metadata.rb file and try again.

version

The current version of this cookbook. Version numbers follow the simple three-number version scheme:

version example

```
version "1.9.0"
```

depends

Declares that this cookbook has a dependency on another cookbook, with an optional version constraint. This will require a cookbook with a matching name and version number to exist on the server, and will cause the server to include it in the set of cookbooks sent to the node when the chef-client runs.

Whenever you need to use a feature in another cookbook, it is a dependency and you need to add the included cookbook's feature to the depends in the current cookbook. For example, including recipes with `"include_recipe"`, `resources/providers` in LWRPs, `libraries`, or `definitions`.

Make sure this is accurate!
This is a critical piece of information - the Chef Server uses these dependency statements to ensure that the right cookbooks are distributed to the edge. Failure to properly fill out this field will result in Chef not completing, or partially completing, the configuration of your system!

The 'depends' are only used if you apply the recipe in a role, or directly to a node. If a cookbook doesn't depend on any others, that's okay.

Version Syntax

Version syntax and constraints are fully documented on their own page. In summary:

Operator	Description
<	Less than
<=	Less than or equal to
=	Equal to
>=	Greater than or equal to
~>	Approximately greater than
>	Greater than

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Examples

depends examples

```
# Depends on a cookbook named 'dogs', with any version
depends "dogs"
# Depends on a cookbook named 'dogs', with version checking
depends "dogs", "> 1.0"
```

In the actual recipe, "`depends dogs`" is required for either.

See Version Constraints for more information on the version syntax, and how the cookbook metadata can be used to limit the version of the cookbook that should be used.

description

A short description of this cookbooks function. Ideally a single line.

license example

```
description 'A fancy cookbook that manages a herd of cats!'
```

long_description

A longer description of this cookbook. Ideally contains full instructions on the proper use of this cookbook, descriptions of definitions it contains, libraries, etc. Often rendered from README.rdoc at the top of the cookbook.

long_description embedded example

```
long_description <<-EOH
= DESCRIPTION:

Complete Debian/Ubuntu style Apache2 configuration.

= REQUIREMENTS:

Debian or Ubuntu preferred.

Red Hat/CentOS and Fedora can be used but will be converted to a Debian/Ubuntu style Apache as it's
far easier to manage with chef.

= ATTRIBUTES:

The file attributes/apache.rb contains the following attribute types:

* platform specific locations and settings.
* general settings
* prefork attributes
* worker attributes

General settings and prefork/worker attributes are tunable.
EOH
```

long_description read in a file example

```
# Alternatively, put the above in README.rdoc and do this
long_description IO.read(File.join(File.dirname(__FILE__), 'README.rdoc'))
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

maintainer

This is the name of the current maintainer - it should be either an individual or an organization.

maintainer example

```
maintainer 'Adam Jacob'
```

maintainer_email

This is the email address of the current maintainer. You can only list one address here, but if you need this to forward to multiple people consider placing a mailing list here that is already setup to forward.

maintainer_email example

```
maintainer_email 'adam@example.com'
```

license

This is the license the cookbook is distributed under.

license example

```
license 'Apache v2.0'
license 'GPL v3'
license 'MIT'
license 'Proprietary - All Rights Reserved'
```

Optional Fields

The following fields are optional. The information in these fields is not used by the Chef server.

attribute

Adds an attribute that a user may need to configure for this cookbook. Takes a name (with the / notation for a nested attribute), followed by any of these options:

Option	Possible Values	Descriptions
display_name	String	What a UI should show for this attribute
description	String	A hint as to what this attr is for
choice	Array of strings	Array of choices to present to the user. If type is string, allow user one choice. If type is array, allow user many choices.
calculated	true or false	If true, the default value is calculated by the recipe and cannot be displayed. If value != nil, calculated default will be overwritten.
type	"string", "array"	The type of value passed to the recipe - default is "string".

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

required	"required", "recommended", "optional"	The level of user input this attribute requires - default is "optional" "required" : User must provide input. The "default" option must not be set. "recommended" : A default value is set, but the user probably should change it. (i.e. apache[:contact] = "ops@example.com") "optional": Default values are set and there is no need to change unless the user knows what they are doing. For backwards compatibility, "required: true" is interpreted as "required: required" and "required: false" is interpreted as "required: optional" **Note:** at present, the required attribute is advisory-only and is not enforced by Chef: if necessary, your recipe should check that the attribute is set and raise an exception if it is not.
recipes	Array	An array of recipes which need this attr set. If not set, the attribute is global to recipes in "this" cookbook
default	String , Array	The default value for this attribute. This value(s) must exist in any non-empty 'choice' array. Do not define if 'calculated' is true.

attribute description

```
attribute 'pets/cat/name',
:display_name => "Cat Name",
:description => "The name of your cat",
:choice => \[ 'kitty kitty', 'peanut', 'einstein', 'honey' \],
:type => "string",
:required => "recommended",
:recipes => \[ 'cats::eat' \],
:default => "kitty kitty"
```

grouping

Adds a displayable title and description to a group of attributes within a namespace. Takes a name (with the / notation for a nested grouping), followed by any of these options:

Option	Possible Values	Descriptions
title	String	Title for this group of attributes
description	String	A hint as to what these attributes are for

attribute grouping description

```
grouping 'pets/cat',
:title => "Cat Options",
:description => "Describe your cat using the options below"
```

conflicts

Warns that this cookbook conflicts with another cookbook.

conflicts examples

```
# Conflicts with a cookbook
conflicts "dogs"
# Conflicts with a cookbook named 'dogs', with mixed version checking
conflicts "dogs", "> 1.0"
```

provides

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Adds a recipe, definition, or resource provided by this cookbook. We auto-populate the list of provided recipes, so it would be rare to need to specify them directly.

Recipes are specified as expected:

provides a recipe

```
provides "cats::sleep"
provides "cats::eat"
```

Definitions are defined by using the name of the definition, followed by parenthesis, and a symbol-style list that matches the definitions parameters.

provides a definition

```
provides "here(:kitty, :time_to_eat)"
```

Individual resources are specified with the normal stringified resource syntax:

provides a resource

```
provides "service[snuggle]"
```

recipe

Adds a description for a recipe - used mainly for aesthetics in the UI.

recipe description

```
recipe "cats::sleep", "For something crazy, like 20 hours a day."
recipe "cats::eat", "When they are not sleeping."
```

recommends

Adds a dependency on another cookbook that is not required (ie: we'll work without it) but is a good idea.

supports examples

```
# Recommends a cookbook
recommends "dogs"
# Recommends on a cookbook named 'dogs', with mixed version checking
recommends "dogs", "> 1.0"
```

replaces

This cookbook replaces another - it can be used entirely in the place of it.

replaces examples

```
# Replaces with a cookbook
replaces "dogs"
# Replaces with a cookbook named 'dogs', with mixed version checking
replaces "dogs", ">> 1.0", "<< 4.0"
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

supports

Adds a supported platform, with version checking logic. To specify multiple platforms, simply provide this command multiple times, once for each platform.

If no version is provided, any version of the platform will match.

supports examples

```
# Supports every version of Ubuntu
supports 'ubuntu'
# Supports versions between 8.04 and 9.10 (8.04, 8.10, 9.04, 9.10)
supports 'ubuntu', ">= 8.04"
# Supports only 9.10
supports 'ubuntu', '= 9.10'
```

Currently, the platform matching and version checking logic is not implemented - Chef will happily run any cookbook on any platform, regardless of the data provided here.

suggests

Adds a suggestion for another cookbook. The difference between recommends and suggests is that, should a system be constructed for automatically resolving cookbook dependencies, recommended cookbooks will be included by default, and suggested ones will not.

supports examples

```
# Suggests a cookbook
suggests "dogs"
# Suggests on a cookbook named 'dogs', with mixed version checking
suggests "dogs", ">> 1.0", "<< 4.0"
```

 Libraries

Recipes 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Recipes



*Recipes are the fundamental configuration in Chef. Recipes encapsulate collections of **resources** which are executed in the order defined to configure the **system***

Recipes are an internal Ruby domain-specific language (DSL), but you do not need to have experience with Ruby to write recipes.

Most things in Chef recipes will be resources to configure. Some things in recipes will be Ruby syntax and helper code.

Basic Recipe Rules

When writing recipes, keep a few things in mind.

1. Recipes are stored in Cookbooks.
2. Resources are executed in the order they appear.
3. Recipes are evaluated as Ruby code on the node to find resources to execute. See Anatomy of a Chef Run for more information.
4. Recipes from other Cookbooks can be included with `include_recipe`.
5. All the attributes of the current Node are available via the `node` object.

Using Recipes

Recipes are used by assigning them to a node or role run list. They are assigned using the appropriate name. Recipes are stored in cookbook directories, and namespaced by the cookbook. The namespacing is used when assigning recipes to be run by a node. For example for a directory structure like:

```
cookbooks/
cookbooks/apache2/
cookbooks/apache2/recipes/
cookbooks/apache2/recipes/default.rb
cookbooks/apache2/recipes/mod_ssl.rb
```

There are two recipes: `apache2` and `apache2::mod_ssl`. On a node, these are assigned to the node's `run_list`. For example:

```
{
  "run_list": [
    "recipe[apache2]",
    "recipe[apache2::mod_ssl]"
  ]
}
```

(See Guide to Creating A Cookbook and Writing A Recipe for a simple example.)

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Just Enough Ruby for Chef

Ruby is a language designed to be easy to read and to behave exactly as you'd expect, so it shouldn't take you too long to get up to speed. See Just Enough Ruby for Chef for a starting point.



Applying Recipes to Nodes

If you are using Chef with client/server, use knife to add the recipe to the node's run list, then run Chef client on the node.

```
% knife node run list add NODENAME "recipe[apache2]"
```

If you are using Chef Solo, use a JSON file passed to `chef-solo -j`:

/etc/chef/dna.json

```
{
  "run_list": ["recipe[apache2]"]
}
```

Then run chef-solo:

```
% sudo chef-solo -j /etc/chef/dna.json
```

The cookbooks must be available in the `cookbook_path` on the system running `chef-solo`. See the Chef Solo page for more information.

Resources

Chef manages Resources on the node. Resources can be many things:

- packages
- services
- users
- files
- directories

And more. They are the bread and butter of recipes. Chef's Providers do the heavy lifting that configure resources, based on the platform of the node. Chef's providers take idempotent actions to configure the resource as it is declared in the recipe.

Composition

Resources have four components:

- type
- name
- parameter "attributes"
- action(s)

For example:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
package "tar" do
  version "1.16.1-1"
  action :install
end
```

This resource is a Ruby block. Its type is "package". Its name is "tar". It has a single parameter attribute "version". It defines a single action, ":install".

Parameter attributes are not the same as node Attributes. Context is important. These parameters specify various ways the resource should be configured. Each kind of resource in Chef has different parameters, and some have default values. If you want the default, you do not need to specify that parameter. Likewise, each type of resource has a default action. You do not need to specify the action if you want the default.

See the Resources page for documentation on each resource that comes with Chef and the default parameters and actions.

Meta Parameters

A variety of "meta" parameters are available to all resources. These are most commonly used to send notifications to other resources or set up conditional execution rules.

Meta parameters are documented in detail in the Meta section of the Resources page.

Custom Lightweight Resource Providers (LWRPs)

As discussed above, Resources represent a piece of system state and Providers are the underlying implementation which brings them into that state. For example, all database vendors support the abstract concept of database creation, but the underlying implementation is different for each. While typical Resources and Providers are implemented in Chef's core using Ruby classes, creating custom Lightweight Resources and Providers (LWRP) requires less Ruby knowledge than their heavier counterparts.

For the Light-weight Resources and Providers (LWRPs) that are used in Opscode's public open source cookbooks, see Opscode LWRP Resources.

Node Object

Recipes are executed in the context of the Chef node object, called `node`. A feature of Chef is using data about the Node, called Attributes in recipes to configure the system in particular ways based on that data.

Accessing Node Attributes

The node's attributes are like a Ruby hash. They are accessed with:

```
node['some_attribute']
```

This will use the value of 'some_attribute'. A number of attributes are detected automatically by ohai when the node runs Chef. Node attributes are a nested key/value store, so there can be sub-keys too:

```
node['some_attribute']['sub_attribute']
```

New custom attributes can be created in Cookbook Attributes and Roles. See below for creating new attributes in Recipes.

Common Automatic Attributes

Ohai detects attributes on the node based on its own plugin system. The most commonly accessed attributes are:

- `node['platform']` - The node's platform. This determines what Providers are used by Resources.
- `node['platform_version']` - The node's platform version. This may be used in determining Providers as well.
- `node['ipaddress']` - The node's ipaddress is IPV4 address of the interface that has the default route. If the node does not have a default route, this attribute will be nil. Using the IP of the interface with the default route is deemed the most sane default.
- `node['macaddress']` - The node's macaddress, from the interface detected for ipaddress above.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- `node['fqdn']` - The fully qualified domain name, for example from `hostname -f` on Unix/Linux systems. This is also used as the node name unless otherwise set.
- `node['hostname']` - The hostname is the first field from splitting the FQDN on "." (dot).
- `node['domain']` - The domain is the rest of the FQDN after splitting on "." (dot).
- `node['recipes']` - The node's Run List is expanded for roles and recipes, and recipes are stored in this attribute.
- `node['roles']` - The node's Run List is expanded for roles and recipes, and roles are stored in this attribute.
- `node['ohai_time']` - Not commonly used in recipes, but this value is the time epoch of the node when ohai was run, and gets saved to the Chef Server. It is also used in the Knife "status" sub command.

Automatic Attributes lists all the `automatic` node attributes detected by Ohai. Additional attributes and sub-attributes can be viewed on a particular node by running `ohai`. This returns data as JSON.

Setting Node Attributes

Node attributes can be set in recipes. This use of node attributes should do done when you want to calculate a derived value, or store some data on the node that should be persisted the next time Chef runs. Use the "set" method on the node.

```
node.set['some_attribute'] = "Some Value" + "Some other value"
node.set['some_attribute']['sub_attribute'] = "Sub attribute Value"
```

Attributes are applied in precedence order; node attributes are `automatic` and have the highest precedence. As these `automatic` attributes will be re-written with each Ohai run - Chef doesn't provide any way to modify them. See Setting Attributes for more detail on the attribute types and precedences.

Including Recipes

Recipes from other cookbooks can be included in a recipe with the `include_recipe` keyword. The included recipe's resources will be inserted in order, at the point where `include_recipe` was called. For example:

```
include_recipe "apache2::mod_ssl"
```

Will include the resources defined in recipe `apache2::mod_ssl`. Note, however, that subsequent calls to `include_recipe` for the same recipe will have no effect.

You can also pass data from various recipes to one definition. This would be useful if you'd like to update your /etc/aliases, /etc/sudoers, or something similar, with entries from multiple recipes in a single chef run.

Dependencies

When using Chef client/server, you must use cookbook Metadata to declare dependencies on cookbooks' recipes included with `include_recipe`. **This is specified with the "depends" keyword in the metadata.rb of the cookbook.**

For example, if the above inclusion of `apache2::mod_ssl` was in a recipe in the "my_app" cookbook then `cookbooks/my_app/metadata.rb` would have:

```
depends "apache2"
```

This is not required in Chef Solo because **all** the cookbooks the node will use must be available since Solo doesn't use a Chef Server to distribute cookbooks.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Command-line Cookbook Dependency Solving with Knife Exec

Need to replicate a node setup elsewhere, and aren't sure of all the cookbook dependencies? Or, just want to validate that you've uploaded all the cookbooks that are stated as being required?

Community member Stephen Nelson-Smith has a blog post on **Dependency Solving** using Knife Exec.

Other Data Sources

The Chef Server has core features that are useful in Recipes to build fully automated dynamic infrastructure.

* Search Indexes
* Data Bags
* Encrypted Data Bags

The Chef Recipe DSL has keywords for using each of these features.

Search

A basic search query in a recipe looks like this:

```
search(:node, "attribute:value")
```

Searches can be assigned to variables and then used elsewhere in a recipe. For example, we could search for all nodes with the role "webserver" assigned, then render a template with them.

```
webservers = search(:node, "role:webserver")

template "/tmp/list_of_webservers" do
  source "list_of_webservers.erb"
  variables(:webservers => webservers)
end
```

For more information on using search in recipes, and search queries in general, see the Search documentation. For more information on the Template resource, see its section on the Resources#Template page.

Data Bags

The Chef Server can have arbitrary stores of JSON data called Data Bags, which can also be Encrypted Data Bags. Each bag is a container of one or more items. These items can be loaded in recipes. Given the bag "apps", and the following item "my_app":

```
{
  "id": "my_app",
  "repository": "git://github.com/company/my_app.git"
}
```

We can access the item in a recipe:

```
my_bag = data_bag_item("apps", "my_app")
```

The item's keys and values can be accessed like a Ruby Hash.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
my_bag["repository"] #=> "git://github.com/company/my_app.git"
```

Other Recipe DSL Methods

There are other methods available in the Recipe DSL.

platform?

The `platform?` method will return true if one of the parameters matches the `node 'platform'`. This method takes a comma separated list of platforms.

```
if platform?("redhat", "centos", "fedora")
  # code for only redhat family systems.
end

if platform?("ubuntu")
  # code for only ubuntu systems
end
```

attribute?

The `node.attribute?` method will return true if the specified node attribute is defined.

```
if node.attribute?('ipaddress')
  # the node has an ipaddress
end
```

value_for_platform

The `value_for_platform` method uses a hash to select a particular value based on the `node 'platform'` and `node 'platform_version'`.

value_for_platform syntax

```
value_for_platform(platform => { platform_version => value })
```

For example, the following will set the `package_name` Ruby variable to "httpd" on Red Hat family distributions, or "apache2" on Debian family distributions.

value_for_platform example

```
package_name = value_for_platform(
  ["centos", "redhat", "suse", "fedora" ] => {
    "default" => "httpd"
  },
  ["ubuntu", "debian"] => {
    "default" => "apache2"
  }
)
```

Tags

To use tags in your recipe simply

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
tag('mytag')
```

If you want to test if a machine is tagged

```
tagged?('mytag')
```

will return true or false. tagged? Takes an array as an argument.

To remove a tag

```
untag('mytag')
```

So all in all

```
tag("machine")

if tagged?("machine")
Chef::Log.info("Hey I'm #{node[:tags]}")
end

untag("machine")

if not tagged?("machine")
Chef::Log.info("I has no tagz")
end
```

Will output

```
[Thu, 22 Jul 2010 18:01:45 +0000] INFO: Hey I'm machine
[Thu, 22 Jul 2010 18:01:45 +0000] INFO: I has no tagz
```

Ruby in Recipes

Regular Ruby code can also be used in Chef Recipes.

Variables

Assign values to variables in recipes using the assignment operator, "="

```
package_name = "apache2"
local_dir = node["apache"]["conf_dir"]
```

Conditionals

Common conditionals can be used to check for true/false. Note that in Ruby, only `nil` and `false` are false, **everything** else is true.

Case

Select a particular package name for a resource based on the platform. This is similar to the example above.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
package "apache2" do
  case node[:platform]
  when "centos","redhat","fedora","suse"
    package_name "httpd"
  when "debian","ubuntu"
    package_name "apache2"
  when "arch"
    package_name "apache"
  end
  action :install
end
```

If/Unless

If statements check for true or false values.

```
if platform?(“debian”, “ubuntu”)
  # do something if node[‘platform’] is debian or ubuntu
else
  # do other stuff
end
```

Unless is the opposite of if.

```
unless node[:platform_version] == "5.0"
  # do stuff on everything but 5.0
end
```

Loops

Use the .each method to an Enumerable object (typically an Array or Hash)

Array

Iterate over an array of package names and install them.

```
["apache2", "apache2-mpm"].each do |p|
  package p
end
```

Hash

Iterate over a hash of gem package names with specific versions.

```
{"fog" => "0.6.0", "highline" => "1.6.0"}.each do |g,v|
  gem_package g do
    version v
  end
end
```

Further Reading

- Just Enough Ruby for Chef

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- Ruby Programming Language



Metadata

Creating a "First Run Only" Resource



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Creating a "First Run Only" Resource



This page describes how to construct a resource that will perform an action on the first run of chef-client and do nothing on subsequent runs.

Use Sparingly
Most resources are idempotent by construction and do not require the methods discussed on this page. *Whenever possible, it is best to use an idempotent resource.*

One case where this would be desired, for example, is to ensure idempotency when a command within an execute resource would move the state of the node away from the desired state if run a second time.

The ideal way to achieve idempotence when using a non-idempotent resource is to inspect the state of the node and condition the resource on that state using the `not_if` or `only_if` conditional execution resource attributes. However, it may be difficult or impossible to correctly inspect the state of the system, or you may never want to run the command again, even if the system's state is incorrect.

In order to create a resource that runs only on the first run of chef-client:

1. set a node attribute after successfully completing the task
2. test for the presence of that attribute within the resource.

The following code will run the command "command to run once" on the first run of chef-client and will not run it upon subsequent runs:

```
execute "some_command" do
  command "command to run once"
  notifies :create, "ruby_block[some_command_run_flag]", :immediately
  not_if { node.attribute?("some_command_complete") }
end

ruby_block "some_command_run_flag" do
  block do
    node.set['some_command_complete'] = true
    node.save
  end
  action :nothing
end
```

This method has a couple of benefits over other possible approaches:

1. Like any other attribute, the flag will be searchable:

```
knife search node "some_command_complete:*"
```

2. All of the flags can be unset using knife:

```
knife exec -E 'nodes.all { |n| n.delete("some_command_complete"); n.save() }'
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



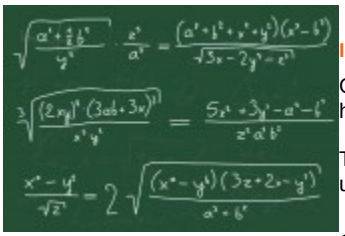
Recipes

Guide to Creating A Cookbook and Writing A Recipe



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Unix Environment Variables and Chef Recipes



In Unix, a process's environment is a set of key-value pairs made available to the process.

Often, programs expect their environment to contain information required for the program to run. The details of how these key-value pairs are accessed depends on the API of the language being used.

This article explains how environments of child processes interact with their parent process and how you can use Chef to ensure that services and applications are started with the proper environment.

Child Processes and Inheritance

Child processes inherit a copy of their parent's environment. In Bash (and other shells) the environment is accessible via shell variables. Shell variables can be added to the environment that is inherited by children processes using the `export` keyword.

Consider the following example:

```
% WOO="woo" #Set a shell variable
% echo $WOO
woo
% bash #Start a new process, a child of our first shell
% echo $WOO # Shell variable is empty because we didn't export it

% exit # Return to original shell
exit
% export WOO # Export variable
% bash
% echo $WOO #Shell variable now available in children.
woo
% exit
```

As mentioned, the child process gets a *copy* of its parent's environment. This means that any changes made to that environment do not affect the parent process. For example:

```
% WOO="woo" #Set a shell variable
% echo $WOO
woo
% bash #Start a new process, a child of our first shell
% export WOO="hello" # Change and export the shell variable
% exit # Return to original shell
exit
% echo $WOO #The parent's value remains unchanged.
woo
% exit
```

The principles mentioned above (a child process receives a copy of its parent's environment and cannot affect their parent's environment) apply in Ruby and Chef just as they do in Bash.

In Ruby (and thus Chef), the current environment can be altered via the `ENV` variable. Any changes made to the environment will also be available to child process started by Chef. For example, consider the following recipe:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
ENV['FOO'] = "bar"
bash "env_test0" do
    code<<-EOF
 echo $FOO
EOF
end
```

When run, the bash resource will correctly echo "bar" to its standard output.

However, just as in Bash, changes made in child processes have no affect on the parent, and thus no affect on subsequent child processes:

```
bash "env_test1" do
    code<<-EOF
    export BAZ="bar"
EOF
end

bash "env_test2" do
    code<<-EOF
 echo $BAZ
EOF
end
```

When run, the second bash resource will not cause anything to be echoed to standard out as BAZ is not part of its environment.

Managing Environments when Using Chef

Services and other processes often look to environment variables for important information needed at run time. There are a number of ways to ensure that processes have access to the environment variables they need to run properly.

Using the Service's Init Script

Ideally, a service's init script would contain everything needed to properly start that service, including the necessary environment. Ensuring that the init script itself contains the necessary environment changes ensures that the service will start properly whenever it is being started using its init script, whether that be from Chef's `service` resource or directly from the shell. In classic System V init scripts, the environment can be altered just as it can be altered in any other shell script, by using a shell variable marked with the `export` keyword:

```
export IMPORTANT_VAR="value"
```

Upstart Services

For services started using Upstart (the System V-compatible init system used by recent versions of Ubuntu and other distributions), their environment can be altered using `env`:

```
env IMPORTANT_VAR="value"
```

See init(5) (provided with the upstart package) for more information.

Systemd Services

For services started using systemd (the System V-compatible init system by the recent versions of Fedora and other distributions), their environment can be altered using the `Environment` or `EnvironmentFile` options:

```
Environment="IMPORTANT_VAR='value'"
```

See systemd.exec(5) for more details.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

If the init script provided by the package does not include the necessary environment variables, you can manage your altered init script using Chef's template resource.

Using ENV

Another method is to use Ruby's predefined ENV variable to set the environment variable. *This ensures that any child processes (including the service that a resource may be starting) have this value in their environment.* While not technically a Hash, ENV can be manipulated as if it were a Hash. For example:

```
ENV['IMPORTANT_VAR'] = "value"

# Some service that requires IMPORTANT VAR
service "example_service" do
  action :start
end
```

Note, however, that changes made to ENV only effect the environment of the chef-client process and its children processes. Altering the environment in this way will often ensure that Chef can start the service properly, but will not ensure that the service will start properly when started using other methods.

Using Resource Attributes

If you are starting a processes by using an `execute` or `script` resource, you can use the `environment` attribute to alter the environment that will be passed to the process.

```
bash "env_test" do
    code<<-EOF
 echo $FOO
EOF
    environment { 'FOO' => "bar" }
end
```

Note that the only environment being altered is the environment being passed to the child process being started by the `bash` resource. This will not affect the environment of chef-client or subsequently started child processes.

Other Issues

My init script works fine when I'm logged in but not over ssh or when launched from chef-client running as daemon!

Shells commonly alter their environment at startup by loading various initialization scripts. The files used for initialization vary based on whether the shell is started as an interactive or non-interactive shell and whether it is is started as a login or non-login shell. When you log in, you are likely starting an interactive, login shell. When you run a command via ssh, it is possible that you are starting a non-interactive shell. This can mean that the process in question is receiving different environments. Ensure that your service or process is being started in a way that ensures its environment has the necessary key-value pairs.

I want to change the environment for every process!

To change the environment for new processes, you will need to alter the initialization scripts for your system's shell. You can manage these scripts using a Chef template resource; however, there are a few caveats you should be aware of:

- The environments of existing processes will be unaffected.
- Shells look to different startup files when started with different options. See your shell's documentation for the definitive list of files that need to be altered and whether it is possible to alter the environment for every possible invocation of the shell.
- When you first change a shell's initialization file, it will have no affect on your current shell or process since its environment has already been initialized.
- From a shell, you can use the `source` command to reload the given initialization file; however, since child processes do not affect their parent's environment, using a `script` or `execute` resource to run `source` from inside a Chef recipe will have no effect on chef-client's environment.

Additional Reading

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

On most well configured unix-like systems, the following manual pages should be helpful:

- environ(7)
- env(1)
- The INVOCATION, COMMAND EXECUTION ENVIRONMENT, and ENVIRONMENT sections of bash(1) or the equivalent sections in the manual page for your preferred shell.

 Guide to Creating A Cookbook and Writing A Recipe

Templates 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Templates

Overview

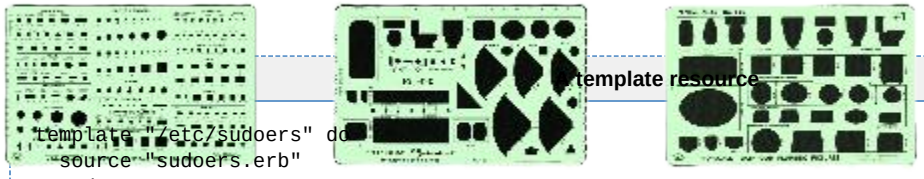
Templates get used all the time in Chef! We love templates.

A template is a file written in a markup language that allows one to dynamically generate the file's final content based on variables or more complex logic. Templates are commonly used to manage configuration files with Chef.

You utilize a template by adding a template resource to a recipe and creating a corresponding ERB template in the cookbook.

> **Chef Templates Just ERB (only faster!)**
>
> Chef utilizes Erubis for our templates - for those of you coming from the Rails, Merb, or Puppet communities, this means it's just ERB (only faster!).

For example, the following template and template resource may be used to manage the `/etc/sudoers` configuration file on your nodes. This example depends on node attributes that would have to be set elsewhere. Within a `recipes/default.rb` and save the template as `templates/default/sudoers.erb`:



A template resource

```
template "/etc/sudoers" do
  source "sudoers.erb"
  mode 0440
  owner "root"
  group "root"
  variables(
    :sudoers_groups => node[:authorization][:sudo][:groups],
    :sudoers_users => node[:authorization][:sudo][:users]
  )
end
```


This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

A template (sudoers.erb)

```
#
# /etc/sudoers
#
# Generated by Chef for <%= node[:fqdn] %>
#

Defaults        !lecture,tty_tickets,!fqdn

# User privilege specification
root          ALL=(ALL) ALL

<% @sudoers_users.each do |user| -%>
<%= user %>   ALL=(ALL) <%= "NOPASSWD:" if @passwordless %>ALL
<% end -%>

# Members of the sysadmin group may gain root privileges
%sysadmin     ALL=(ALL) <%= "NOPASSWD:" if @passwordless %>ALL

<% @sudoers_groups.each do |group| -%>
# Members of the group '<%= group %>' may gain root privileges
%<%= group %> ALL=(ALL) <%= "NOPASSWD:" if @passwordless %>ALL
<% end -%>
```

Variables and ERB

Chef templates are ERB templates. ERB templates allow you to embed Ruby code inside a text file within specially formated tags:

An ERB Template

```
Plain text content
<%= RUBY EXPRESSIONS GO HERE %>
<% RUBY STATEMENTS GO HERE -%>
More plain text content
```

When rendered, the ruby code delimitated by <%= and %> is evaluated and its output is placed in the final text file. You will mostly be using Ruby Expressions, as this is how you reference any variables you sent to the template. You can also use Ruby Statements though, such as each, if or end statements. These are surrounded by <% and -%> instead.

When a template is rendered, Chef passes the variables listed in the resource's `variables` attribute and the `node` object to the template. The variables will be accessible as **instance variables** within the template while the node object can be accessed just as it can be in a recipe, using the name `node`.

For example:

Simple template resource

```
node[:fqdn] = "latte"
template "/tmp/foo" do
  source "foo.erb"
  variables({
    :x_men => "are keen"
  })
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Simple ERB template

```
The node <%= node[:fqdn] %> thinks the x-men <%= @x_men %>
```

Would render:

Rendered template

```
The node latte thinks the x-men are keen
```

As can be seen in the example present in the Overview, more complex Ruby features can be used to dynamically generate configuration files using the data passed to the template resource. **For more information about how to use ERB, check out the Erubis Documentation.**

Template Location Specificity

Cookbooks are often designed to work on a variety of hosts and platforms. Templates often need to differ depending on the platform, host, or function of the node. When the differences are minor, they can be handled with a small amount of logic within the template itself. When templates differ dramatically, you can define multiple templates for the same file. Chef will decide which template to render based on the following rules.

Within a Cookbook's template directory, you might find a directory structure like this:

- templates
  - host-foo.example.com
  - ubuntu-8.04
  - ubuntu
  - default

For a node with FQDN of foo.example.com and the `sudoers.erb` resource above, we would match:

- host-foo.example.com/sudoers.erb
- ubuntu-8.04/sudoers.erb
- ubuntu/sudoers.erb
- default/sudoers.erb

In that order.

Then, for example: `sudoers.rb` placed under the `files/host-foo.example.com/` directory, means it will be only copied to the machine with the domain name `foo.example.com`. (Note the "host-" prefix to the directory name)

So, the rule distilled:

1. `host-node[:fqdn]`
2. `node[:platform]-node[:platform_version]`
3. `node[:platform]`
4. `default`

('default' here does not refer to the recipe in default.rb. Templates are not split up into different directories by recipe.)

Additionally, when using a cookbook downloaded from the community site, it is possible to override individual templates within that cookbook using Site Specific Cookbooks. Note, however, that this feature is being deprecated.



Host Notation
The **host-** is literal. If your host was `foo.example.com` than your folder needs to be named `host-foo.example.com`.

Hey, Cfengine people!
This is inspired by Single Copy Nirvana - if you think about it that way, this will click immediately.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

When does a template get transferred?

Chef caches a template when it is first requested. On all subsequent requests for that template, we compare that to the template on the server - if they are the same, we won't transfer the entire template again.

Use Case Scenario

We have an Apache cookbook writing out httpd.conf, then we have an application under site-cookbooks that needs its own very custom httpd.conf. How do we have the site cookbook override the httpd.conf template?

Review the `cookbook` attribute for the `template` resource. Use of this attribute allows you to specify an alternate cookbook in which to look for a template. You could therefore create application specific cookbooks that hold app-specific versions of the template in question. Usually the core `httpd.conf` file doesn't change between applications, but the vhost conf file may differ heavily.

For example, the `web_app` definition that is found within the Opscode apache2 cookbook is meant to simplify the process of creating an app specific vhost:

The web_app definition

```
define :web_app, :template => "web_app.conf.erb" do

  application_name = params[:name]

  include_recipe "apache2"
  include_recipe "apache2::mod_rewrite"
  include_recipe "apache2::mod_deflate"
  include_recipe "apache2::mod_headers"

  template "#{node[:apache][:dir]}/sites-available/#{application_name}.conf" do
    source params[:template]
    owner "root"
    group "root"
    mode 0644
    if params[:cookbook]
      cookbook params[:cookbook]
    end
    variables(
      :application_name => application_name,
      :params => params
    )
    if ::File.exists?("#{node[:apache][:dir]}/sites-enabled/#{application_name}.conf")
      notifies :reload, resources(:service => "apache2"), :delayed
    end
  end

  apache_site "#{params[:name]}.conf" do
    enable enable_setting
  end
end
```

The Opscode wordpress cookbook has a great example of using this definition:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Use of the web_app definition

```
(...)

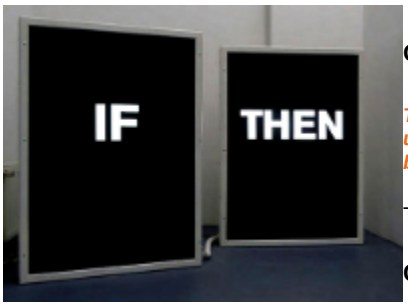
  web_app "wordpress" do
    template "wordpress.conf.erb"
    docroot "#{node['wordpress']['dir']}"
    server_name server_fqdn
    server_aliases node['fqdn']
  end
```

 Unix Environment Variables and Chef Recipes

Version Constraints 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Version Constraints



Overview

This page clarifies the syntax for cookbook versions and describes version constraints, which can be used in cookbook metadata and in environments feature to limit which versions of a cookbook should be used.

The departs from currently documented, but not used, version constraint features found here.

Cookbook Version Syntax

A cookbook version has the form x.y.z, where x, y and z are decimal numbers representing major, minor and patch versions, respectively. A two-part version (e.g. x.y) is also allowed.

Note that alpha-numeric version numbers, such as 3.4.rc4, and version numbers with more than three parts (e.g. 1.2.3.4) are not allowed.

Version Constraints

A version constraint is a string of the form "OP VERSION", where OP is one of the operators listed below and VERSION is a cookbook version as described above.

Operators

- = Equal to
- > Greater than
- < Less than
- >= Greater than or equal to
- <= Less than or equal to
- ~> Approximately greater than

Specifying `">= 2.6.5"` is an optimistic version constraint. All versions greater than the one specified, including major releases (e.g. 3.0.0) are allowed.

Conversely, specifying `"~> 2.6"` is pessimistic about future major revisions and `"~> 2.6.5"` is pessimistic about future minor revisions.

`"~> 2.6"` matches cookbooks `>= 2.6.0 AND < 3.0.0`
`"~> 2.6.5"` matches cookbooks `>= 2.6.5 AND < 2.7.0`



Requires Chef 0.10
The Version Constraints functionality is only available in Chef 0.10.x and after.

Cookbook Metadata

The following DSL functions can be used in a cookbook's metadata.rb file. Each function accepts a name and an optional version constraint. If the version constraint is not provided, the default constraint of `">= 0.0.0"` will be used.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
conflicts
depends
provides
recommends
replaces
suggests
supports
```

Example version constraints in a cookbook/metadata.rb file:

```
depends   "opscode-base"
depends   "opscode-github", "> 1.0.0"
depends   "runit", "~> 1.2.3"
conflicts "apache2", "< 3.0"
conflicts "daemon-tools"
```

Specifying a version constraint for these functions is **optional**. If no version constraint is provided, the default version constraint of ">= 0.0.0" will be used.

Environment Cookbook Versions

An environment can specify a list of allowed cookbook versions using version constraints. When defining a constraint, you must specify the cookbook name and the version constraint. Any cookbook not explicitly constrained in the environment file is assumed to have no version constrain and can be used at any version by every node in that environment.

```
cookbook "apache2", "~> 1.2.3"
cookbook "runit", "= 4.2.0"
```

Cookbook Versions in Run List Items

You can peg the cookbook version to use for recipe items in a node's run list. The syntax uses '@' to specify a version. Only the equality constraint is supported for specifying constraints in run lists.

Example run list JSON

```
{"run_list":["recipe[nginx@1.2.3]"]}
```

Freezing Cookbooks

You can also freeze your cookbooks, to prevent anyone from uploading the frozen cookbook at the same version number. Once a cookbook is frozen, you can only upload a cookbook with the same name and version by using the --force option:

```
% knife cookbook upload redis --freeze
Uploading redis…
upload completed

% knife cookbook show redis 0.1.6 | grep frozen
frozen?:        true

% knife cookbook upload redis
Uploading redis…
ERROR: Version 0.1.6 of cookbook redis is frozen. Use --force to override.
```

By combining frozen cookbook versions with environments you can make sure that production is safe from accidental updates when testing out

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

changes in your development infrastructure.

There are two main strategies for implementing version control:

- If you place supreme importance on always keeping every last bit of data in version control, you'll want to use the maximum version control strategy and drive your use of environments by editing files only.
- If you're already managing your cookbooks for separate environments using git branches, and you already have your versioning policy information stored in your cookbooks' metadata, you might opt for a lighter weight branch tracking strategy.

Below are examples of both of these strategies.

Branch Tracking Strategy

In the branch tracking strategy, you have a branch in your source control repo for each environment, and your cookbook versioning policy tracks whatever is in the tip of each branch. This strategy is simple and lightweight. For development environments that track the latest cookbooks, you just need to bump the version before you upload the cookbook for testing. For environments that need special protection, i.e., production, you always upload cookbooks using the -E ENVRIONMENT and --freeze flags.

In development:

1. Bump the version number as appropriate.
2. Make the changes needed to the cookbook.
3. Upload and test. Upload for dev is the same as always:

```
knife cookbook upload my-app
```

Repeat steps 2 and 3 as necessary.

When you're ready to move your changes into production:

1. Upload for prod with automatic version constraints:

```
knife cookbook upload  my-app -E production --freeze
```

Adding the -E ENVIRONMENT option will cause knife to automatically set the version constraint on that environment to match the version you're uploading.

Maximum Version Control Strategy

If you prefer to version control absolutely everything, then you need to use a file-editing based approach to environments. In your development environment, you work the same as the branch tracking strategy.

In development:

1. Bump the version number as appropriate.
2. Make the changes needed to the cookbook.
3. Upload and test. Upload for dev is the same as always:

```
knife cookbook upload my-app
```

Repeat 2 and 3 as necessary.

When you're ready to move your cookbooks into production, you'll do the following:

1. Upload and freeze your cookbooks:

```
knife cookbook upload my-app --freeze
```

2. Modify your environment to prefer the new version you uploaded:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
(vim|emacs|mate|ed) YOUR_REPO/environments/production.rb
```

3. Upload the updated environment:

```
knife environment from file production.rb
```

4. Deploy!

 Templates

Cookbook Site Help 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook Site Help



The Opscode Cookbook Site is a community site where you can share, contribute to, and/or download cookbooks through the web site or its REST API.

You must first have the capability to interact with the site, following which you'll be able to access, download, use and contribute to the community cookbooks.

Interaction with the Community Site

Opscode has Single Sign-On for the Community Site and for Hosted Chef - so if you have an account for one, then you have an account for both. You'll need to be logged in if you already have an account. If you don't have an account - sign up, it's free!

Resetting the Password to the Community Site

There are two ways to reset the password, depending on if you are able to log in with your current password or not.

Changing your password with your current password

1. On the homepage of the Opscode cookbooks site, click your username in the upper right corner next to *Logged in as*.
2. On the page showing your user details, click the change my password link, as shown below:



3. Enter your current password, new password, and confirm your new password, and then click the "Change Password" button.
   Note that password needs to be 6 to 50 characters long.

Changing your password without your current password

If you forget your current password, you can change your password without using the current password.

You will need to know your Opscode user name's email address so that you can get an email to reset your password.

1. On the homepage of the Opscode cookbooks site (http://cookbooks.opscode.com), click the *Recover Password* link in the upper right corner.
2. On the Recover Your Password page, enter the Email Address that you used to register with Opscode, and click the *Submit* button. You will receive an email with a link to reset your password.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

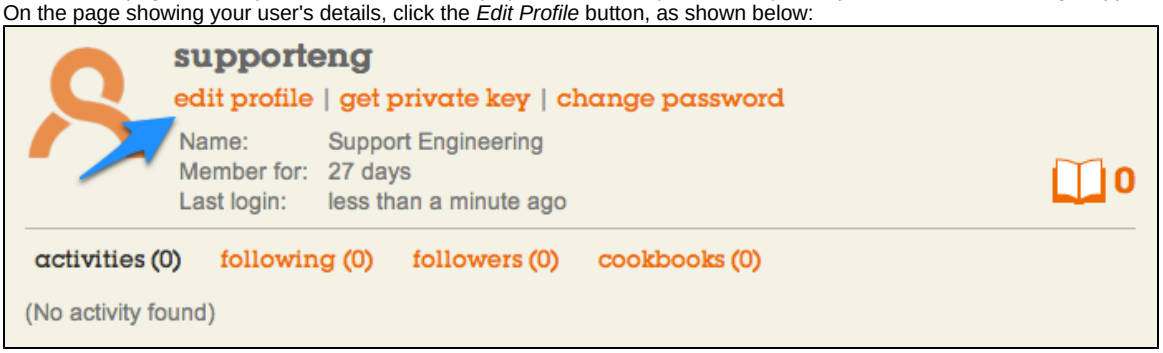
Recent Opscode Cookbook Updates

The PHP, Mysql and Database cookbooks are all now at release 1.0.0 and work across CentOS and RHEL.

Editing your Profile

Your Opscode user account is shared across the Opscode Platform, cookbooks.opscode.com, and help.opscode.com. You can update your name, twitter account information, geographic information, email address, profile image, and communication preferences.

1. On the homepage of the Opscode cookbooks site (http://cookbooks.opscode.com), click your username in the right upper corner.
2. On the page showing your user's details, click the *Edit Profile* button, as shown below:



3. On the Edit Your Profile page,
4. Update your information, and
5. Click the *Save Changes* button.
   First Name, Last Name, and Email Address are required fields.

Use of the Community Site

Once you have the capability to interact on the site, you can search for and review cookbooks of interest, access and download those that you may potentially use for managing your infrastructure, follow cookbooks to be notified of any changes, upload your own cookbooks to the site to share with the community.

Following Cookbooks on the Community Site

Users can follow cookbooks to be updated via email when the cookbooks are updated. In order to find the cookbook you wish to follow, you can either search or browse the categories. Both the search box and the categories are on the homepage of the Cookbook Site (http://cookbooks.opscode.com). The search box can also be found in the right upper corner of other pages.

When you are on a particular cookbook's page,

1. The option to follow the cookbook is available on the right side of the page.
2. Click the gray *Follow* button right under the *Download Now* button to follow the cookbook.
3. Once followed, clicking again allows you to unfollow it.

When you follow a cookbook, you will receive email notifications when new versions of the cookbook are uploaded, if you have chosen to receive email notifications from Opscode. The list of cookbooks you are following is found in your user profile: accessed by clicking your username in the right upper corner after logging in.

Access and Download Cookbooks from the Cookbook Site

You can download and search for Cookbooks from the web site without logging in, but all of the rest operations (sharing, contributing, or using the REST API) require you to log in with an Opscode user account, which can be registered for free on the Cookbook Site.

There are two ways to download and install a cookbook from the Cookbook Site:

Every Chef installation needs a Chef Repository

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- Download a cookbook from cookbooks.opscode.com, place it in your chef repository's cookbook directory and untar the file to use it. For example, if you downloaded the apache2 cookbook you could run the following to untar it:

  ```
  tar zxvf apache2.tar.gz
  ```

  The cookbook will uncompress and be ready to be uploaded to Hosted Chef or the Chef Server. You can delete the tar.gz file once the command completes.

- You can also use `knife` to install cookbooks. This command would download the apache2.tar.gz file to the cookbook directory, untar it, and remove the tar file afterwards for you:

  ```
  knife cookbook site install apache2
  ```

  Once this command completes the cookbook is ready to be uploaded to Hosted Chef or the Chef Server.

This is the place where cookbooks, roles, config files and other artifacts for managing systems with Chef will live.

Authors of the Cookbooks on the Cookbook Site

There are cookbooks provided and maintained by Opscode, while others are contributed by members of the Chef Community. If you have a cookbook downloaded and need to know who created it, you can find this at the bottom of the README.rdoc. This is located at `~/chef-repo/cookbooks/<cookbook name>/README.rdoc`.

This information is also available on the cookbook site, once you click on a cookbook the collaborators and maintainers are listed on the right-hand side of the page.

Opscode Supported Cookbooks
See **Cookbook Support** for the details of support provided for Opscode maintained cookbooks.

Creating and Installing Cookbooks

Each of these activities are sufficiently detailed to have the information on their own pages.

- Details on creating cookbooks can be found at **Guide to Creating A Cookbook and Writing A Recipe**.
- **Managing Cookbooks With Knife** has detail on use of the Knife command line interface to manage your cookbooks, including the installation

Adding a Cookbook to the Community Site

There are two ways to add a cookbook to the community site, you can either use knife or upload the tarball through the website.

Uploading a cookbook using Knife

share

Use of Knife's `share` subcommand uploads the specified cookbook using the given category to the Opscode cookbooks site. This requires a login user and certificate for the Opscode Community site.

By default, knife will use the username and API key you've configured in your configuration file; otherwise you must explicitly set these values on the command line or use an alternate configuration file. You need to be the owner or maintainer of the cookbook if it already exists on the web site.

Usage:

```
```

For example:

```
```

unshare

Use of the `unshare` subcommand stops sharing the given cookbook on the Opscode cookbooks site.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Usage:

Uploading a cookbook using the web site

An alternate to using Knife is to upload the cookbook directly through the site.

Prepare the Cookbook

First, you should ensure that your cookbook's metadata.rb file is correct. You must include the following metadata:

- Name
- Version
- License

We also strongly recommend that you fill out:

- Description
- Supported Platforms

Once you have done that, you need to bundle the cookbook into a tarball. New versions of the Chef Repository have a task called 'bundle_cookbook' which will do this for you. From the top of your Chef Repository:

The task will produce a tarball in the pkgs directory of your chef repository:

Add the cookbook to the Cookbook Site

To add a new cookbook to the Cookbook Site:

1. Visit http://cookbooks.opscode.com and log in.
2. Click on the 'Cookbooks' tab at the top.
3. Click the orange *Add a new cookbook* button.
4. On the "New cookbook" page, browse for your cookbook tarball, select a category, enter tags (separated by commas), and click *Create*.

Licensing Questions

Do I need to license my cookbooks with Apache 2.0, like Opscode uses for Chef?

A: No. You can use any license you wish for your cookbooks as long as you have the rights to share and publish them. You specify which license the cookbook is published under in the metadata of the cookbook itself.

Do I need to sign an Opscode Contributor License Agreement to post cookbooks?

A: No. The Opscode CLA is only required if you want to make contributions to Opscode's published cookbooks.

Maintaining the Cookbooks you have Added

If you uploaded a cookbook to the Opscode Cookbook Site, you are the maintainer of the cookbook. A maintainer can delete or deprecate the cookbook, edit the metadata of the cookbook, and add collaborators. In order to do any of these tasks, you need to log in to the Opscode Cookbook Site and navigate to the cookbook you uploaded. You will see a menu bar with the following options:

Deleting the cookbook

Click the *Delete cookbook* link and confirm the operation. Note: It's generally a better idea to deprecate a cookbook than to delete a cookbook, since other users may rely on that cookbook and will want to know what to use instead.

Deprecating the cookbook

Click the *Deprecate cookbook* link and enter the name of another cookbook that you would like to direct users to use instead. This cookbook can be maintained by another user entirely. However, it needs to have been uploaded before deprecating the old one.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Adding a collaborator to the cookbook

A collaborator can upload new versions of the cookbook. In order to add a collaborator to the cookbook, click the *Add a collaborator* link and enter the username you want to add.

Promoting collaborators to maintainers

A maintainer can promote collaborators to become maintainers by clicking the *promote* link under the collaborator. There can only be one maintainer on the cookbook. Think of maintainers as the admin for the cookbook; they can remove everyone else as collaborators, so be careful who you promote.

Uploading a new version of the cookbook

Click the *Upload a new version* link on the menu bar to upload a new version of the cookbook. Both maintainers and collaborators can do this.

Editing the metadata of a cookbook

1. Click *Edit Cookbook* button on the menu bar.
2. Enter new information (metadata) about the cookbook.

 Version Constraints

Chef Repository 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Repository



Overview

Every Chef installation needs a Chef Repository. This is the place where cookbooks, roles, config files and other artifacts for managing systems with Chef will live.

We strongly recommend storing this repository in a version control system and treating it like source code.

While we prefer Git, and make this repository available via GitHub, you are welcome to download a tar or zip archive and use your favorite version control system to manage the code.

Chef Terms used on this page

We use some terms regarding the Chef Repository that have further explanation on their own pages.

- Knife - Chef's command-line API tool.
- Cookbooks - Unit of code distribution in Chef.
- Roles - Describe what a node should be / do.
- Data Bags - Arbitrary stores of data about the environment / infrastructure.

Getting the chef-repo

To create your own repository, start by cloning Opscode's `chef-repo` from GitHub. *If you don't want to install Git, you can download a tarball.*

Cloning Opscode's Chef Repo from GitHub

Git Clone chef-repo

```
git clone git://github.com/opscode/chef-repo.git
```

If you want to wipe out the existing history and start fresh, feel free to remove the `.git` directory after you clone. Then you can initialize a new git repository, or create a subversion (or mercurial, or bazaar) repository.

If you don't want Git...

You can grab the tarball directly off GitHub:

Grab Tarball of Chef Repo

```
wget http://github.com/opscode/chef-repo/tarball/master
```

You'll need to extract the tarball, and move the directory it creates (by default, `username-project-commit/`) to `chef-repo`. The commit ID may be different than this.

Extract Tarball

```
tar -zxf master
mv opscode-chef-repo-a3bec38 chef-repo
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

We still strongly recommend you use some kind of version control tool to manage the source code in your chef-repo, so use your favorite tool to initialize the repository for tracking. If you do use Git, there is more info on Working with Git and Cookbooks.

Repository Directories

This repository contains several directories, and each directory contains a README file that describes what it is for in greater detail, and how to use it for managing your systems with Chef.

- `config/` - Contains the Rake configuration file, `rake.rb`.
- `cookbooks/` - Cookbooks you download or create.
- `data_bags/` - Store data bags and items in .json in the repository.
- `roles/` - Store roles in .rb or .json in the repository.
- `certificates/` - SSL certificates generated by `rake ssl_cert` live here.

config

Contains the Rake config file. See Configuration below.

cookbooks

This directory contains the cookbooks used to configure systems in your infrastructure with Chef. Configure knife to use your preferred copyright holder, email contact and license by adding the following lines to `~/chef-repo/.chef/knife.rb`:

```
cookbook_copyright "Example, Com."
    cookbook_email     "cookbooks@example.com"
    cookbook_license   "apachev2"
```

Supported values for `cookbook_license` are "apachev2" or "none". These settings are used to prefill comments in the default recipe, and the corresponding values in the metadata.rb. You are free to change these in those files.

Create new cookbooks in this directory with Knife.

```
knife cookbook create COOKBOOK
```

This will create all the cookbook directory components. You don't need to use them all, and can delete the ones you don't need. It also creates a README file, metadata.rb and default recipe.

You can also download cookbooks directly from the Opscode Community Cookbook Site. There are two subcommands to help with this depending on what your preference is. The first and recommended method is to use a vendor branch if you're using Git, this step is automatically handled with Knife.

```
knife cookbook site install COOKBOOK
```

This will:

- Download the cookbook tarball from cookbooks.opscode.com.
- Ensure its on the git master branch.
- Checks for an existing vendor branch, and creates if it doesn't.
- Checks out the vendor branch (chef-vendor-COOKBOOK).
- Removes the existing (old) version.
- Untars the cookbook tarball it downloaded in the first step.
- Adds the cookbook files to the git index and commits.
- Creates a tag for the version downloaded.
- Checks out the master branch again.
- Merges the cookbook into master.

The last step will ensure that any local changes or modifications you have made to the cookbook are preserved, so you can keep your changes through upstream updates.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

If you're not using Git, use the site download subcommand to download the tarball.

```
knife cookbook site download COOKBOOK
```

This creates the COOKBOOK.tar.gz from in the current directory (e.g., `~/chef-repo`). We recommend following a workflow similar to the above for your version control tool.

data_bags

This directory contains directories of the various data bags you create for your infrastructure. Each subdirectory corresponds to a data bag on the Chef Server, and contains JSON files of the items that go in the bag.

First, create a directory for the data bag.

```
mkdir data_bags/BAG
```

Then create the JSON files for items that will go into that bag.

```
$EDITOR data_bags/BAG/ITEM.json
```

The JSON for the ITEM must contain a key named "id" with a value equal to "ITEM". For example,

```
{
      "id": "foo"
    }
```

Next, create the data bag on the Chef Server.

```
knife data bag create BAG
```

Then upload the items in the data bag's directory to the Chef Server.

```
knife data bag from file BAG ITEM.json
```

roles

Create roles here, in either the Role Ruby DSL (.rb) or JSON (.json) files. To install roles on the server, use knife.

For example, create `roles/base_example.rb`:

```
name "base_example"
    description "Example base role applied to all nodes."
    # List of recipes and roles to apply. Requires Chef 0.8, earlier versions use 'recipes()'.
    #run_list()
    # Attributes applied if the node doesn't have it set already.
    #default_attributes()
    # Attributes applied no matter what the node has set already.
    #override_attributes()
```

Then upload it to the Chef Server:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife role from file roles/base_example.rb
```

certificates

Creating SSL certificates is a common task done in web application infrastructures, so a rake task is provided to generate certificates. These certificates are stored here by the ssl_cert task.

Configure the values used in the SSL certificate by modifying `config/rake.rb`. To generate a certificate set for a new monitoring server, for example:

```
rake ssl_cert FQDN=monitoring.example.com
```

Once the certificates are generated, copy them into the cookbook(s) where you want to use them.

```
cp certificates/monitoring.example.com.* cookbooks/COOKBOOK/files/default
```

In the recipe for that cookbook, create a cookbook_file resource to configure a resource that puts them in place on the destination server.

```
cookbook_file '/etc/apache2/ssl/monitoring.example.com.pem'
      owner 'root'
      group 'root'
      mode 0600
    end
```

Rake Tasks

The repository contains a `Rakefile` that includes tasks that are installed with the Chef libraries. To view the tasks available with in the repository with a brief description, run `rake -T`.

The default task (`default`) is run when executing `rake` with no arguments. It will call the task `test_cookbooks`.

The following tasks are not directly replaced by knife sub-commands:

- `bundle_cookbook` - Creates cookbook tarballs in the `pkgs/` dir.
- `install` - Calls `update`, `roles` and `upload_cookbooks` Rake tasks.
- `ssl_cert` - Create self-signed SSL certificates in `certificates/` dir.
- `update` - Update the repository from source control server, understands git and svn.

The following tasks duplicate functionality from knife and may be removed in a future version of Chef:

- `metadata` - replaced by `knife cookbook metadata -a`.
- `new_cookbook` - replaced by `knife cookbook create`.
- `role` - replaced by `knife role from file`.
- `roles` - iterates over the roles and uploads with `knife role from file`.
- `test_cookbooks` - replaced by `knife cookbook test -a`.
- `test_cookbook` - replaced by `knife cookbook test COOKBOOK`.
- `upload_cookbooks` - replaced by `knife cookbook upload -a`.
- `upload_cookbook` - replaced by `knife cookbook upload COOKBOOK`.

Configuration

The repository uses two configuration files.

- config/rake.rb
- .chef/knife.rb (optional)

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

The first, `config/rake.rb` configures the Rakefile in two sections.

- Constants used in the `ssl_cert` task for creating the certificates.
- Constants that set the directory locations used in various tasks.

If you use the `ssl_cert` task, change the values in the `config/rake.rb` file appropriately. These values were also used in the `new_cookbook` task, but that task is replaced by the `knife cookbook create` command which can be configured below.

The second config file, `.chef/knife.rb` is a repository specific configuration file for knife. If it exists, it will override your primary knife configuration file (in `~/.chef/knife.rb`). You do not need to have a `knife.rb` in your chef-repo if your primary knife configuration file is correct. If you're using the Opscode Platform, you can download a knife configuration for your organization from the management console. If you're using the Open Source Chef Server, you can generate a new one with `knife configure`. For more information about configuring Knife, see the Knife documentation.

Next Steps

Learn more about Chef Basics or get started on working with Cookbooks at the Cookbook Fast Start Guide. If you did proceed with Git, we have a guide on Working with Git and Cookbooks that you may want to review.

 Cookbook Site Help

Data Bags 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook Workflows

Workflows

Combined chef / cookbook repo

One git repository containing all cookbooks

One git repository per cookbook

Cookbooks as git sub-modules

Librarian

Use cases

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Data Bags



What are Data Bags?

Data bags provide an arbitrary stores of globally available JSON data.

Data Bags are not directly associated with Node or Role attributes. When using Data Bags with a chef server, they are stored on the server and indexed for searching. When using Data Bags with chef-solo, data bags are stored in a directory hierarchy on the machine running chef-solo.

Recipes can load data bags directly or search a data bag for specific values similar to attributes in node indexes.

Store Data Bags In Source Control

If you clone the Chef Repository from Opscode, there is a directory called `data_bags`. You can create a directory for each bag, and put the JSON files for each item in the bag's directory. This will provide for more easily managing data bags in source control systems and in understanding and use. Access to your data bag structure can then be done through the knife command to load from file.

Data Bag From File

A individual data bag can refer to either a sub-directory of the chef-repo/data_bags directory or a .json file in that directory or one of the sub-directories. From the earlier example of admins:

```
mkdir data_bags/admins
```

Either create the new data bag item as a JSON file in the admins directory, or if the item already exists on the Chef Server, write to a file.

```
vi data_bags/admins/charlie.json
```

Or:

```
knife data bag show admins charlie -Fj > data_bags/admins/charlie.json
```

Here is an example of what a complete data bag directory structure could look like:

```
data_bags
            |_admins
                        |_charlie.json
                        |_bob.json
                        |_tom.json
            |_db_users
                        |_charlie.json
                        |_bob.json
                        |_sarah.json
            |_db_config
                        |_small.json
                        |_medium.json
                        |_large.json
            |_standard_packages.json
            |_global_shell_settings.json
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

The directory structure is used by knife as a convenience so you don't have to type the full path when using the `data bag from file` sub-command.

The data bag itself provides a container of related items. For example, admins and db_users in your output would be separate bags. The "standard_packages" and "global_shell_settings" are things that might be under a "common" bag or whatever makes sense to you for grouping.

Each item in the bag gets its own JSON file. **The only structural requirement of items is that they have an id.**

```
{
     "id": "ITEM_NAME"
   }
```

Then when using the knife command to load the data bag item from the JSON file, you don't have to specify the path, just the name of the bag and the filename of the item. Knife's `data bag from file` automatically knows how to find the directory and the file under `data_bags`.

The command:

```
knife data bag from file BAG_NAME ITEM_NAME.json
```

Loads the file:

```
data_bags/BAG_NAME/ITEM_NAME.json
```

(Note: the bag "BAG_NAME" must exist already.)

If you are not in the root directory for chef, you can also specify the path with this command:

```
knife data bag from file BAG_NAME /path/to/file/ITEM_NAME.json
```



Deploying from your private repo using Data Bags?

Don't overlook the `deploy_key` option. It's there for you to use to add the deployment private key to the data bag.

```
{
     "id": "my_app",
     ... (truncated) ...
     "deploy_key": "SSH private key"
   }
```

The key would be the one from your private git repository, and needs to be a string with the newlines converted to "\n"'s



Command Output Change in 0.10

Prior to version 0.10, `knife data bag` output was a JSON representation of the relavent information. JSON output can still be received in 0.10 and above by passing the `-Fj` command line option to knife.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Data Bags With Knife

Data bags can be managed with the knife command line client. See **Managing Data Bags With Knife** to accomplish this.

Using Data Bags in Recipes

Data bags can be loaded by name using the DSL in a recipe, or can be accessed via the search indexes. When using a single data bag item, use the direct method whenever possible, as it avoids the search index entirely and so has lower overhead. On the other hand, if you plan to loop over all of the items in a data bag, using the search interface will cause Chef to bulk load the items, resulting in lower overhead.

Loading a Data Bag Directly

The recipe DSL provides access to data bags via the `data_bag(BAG)` method and access to data_bag_items via the `data_bag_item(BAG, ITEM)` method.

The `data_bag` method returns an array of the keys of the items that belong to the data bag. For the admins data bag shown in the above examples, we'd use the data_bag method like this:

```
data_bag("admins")
# => ["charlie"]
```

To get the contents of the 'charlie' data bag item, we use the data_bag_item method. Note that each time you use data_bag_item in a recipe, Chef makes an API call to the server to load the item.

```
data_bag_item('admins', 'charlie')
# => {"comment"=>"Crazy Charlie", "gid"=>1005, "id"=>"charlie", "uid"=>1005, "shell"=>"/bin/zsh"}
```

A More Complete Example

If we want to create a user on our systems for each admin in our admins data bag, we might do something like this:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Example Admins Recipe default.rb

```
# Load the keys of the items in the 'admins' data bag
admins = data_bag('admins')

admins.each do |login|
  # This causes a round-trip to the server for each admin in the data bag
  admin = data_bag_item('admins', login)
  home = "/home/#{login}"

  # for each admin in the data bag, make a user resource
  # to ensure they exist
  user(login) do
    uid       admin['uid']
    gid       admin['gid']
    shell     admin['shell']
    comment   admin['comment']

    home      home
    supports  :manage_home => true
  end

end

# Create an "admins" group on the system
# You might use this group in the /etc/sudoers file
# to provide sudo access to the admins
group "admins" do
  gid     999
  members admins
end
```

Loading Data Bags through the Search Interface

In some situations you may not know which data bag items you want information from in advance, or you may want to load all of the items in the data bag. In these cases you can use the search index to find the data you want. When searching for data bag items, you provide the name of the data bag to search and the query string as arguments to the `search(BAG, QUERY_STRING)` method. Using the admins data bag, you can search for items based on any value, for example:

find every admin in the admins data bag

```
search(:admins, "*:*")
```

search for an admin with the id "charlie"

```
search(:admins, "id:charlie")
```

search for admins with a gid of "ops"

```
search(:admins, "gid:ops")
```

search for admins with an id beginning with the letter 'c'

```
search(:admins, "id:c*")
```

Note that even though the search is returning the data bag items as Chef::DataBagItem objects, you can use these objects just like they were a

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Hash:

Use DataBagItem Objects Like Hashes

```
charlie = search(:admins, "id:charlie").first
# => variable 'charlie' is set to the charlie data bag item
charlie["gid"]
# => "ops"
charlie["shell"]
# => "/bin/zsh"
```

Here's the same "create a user for each admin" recipe as shown above, modified to load the admins using the search interface:

Example Admins Recipe default.rb

```
# An empty array to which we'll add the admins' logins as we go.
admins = []

# search for all items in the 'admins' data bag and loop over them
search(:admins, "*:*") do |admin|
  # Set `login` to the id of the data bag item
  login = admin["id"]

  # build up the list of the admins logins
  admins << login

  home = "/home/#{login}"

  # for each admin in the data bag, make a user resource
  # to ensure they exist
  user(login) do
    uid       admin['uid']
    gid       admin['gid']
    shell     admin['shell']
    comment   admin['comment']

    home      home
    supports  :manage_home => true
  end

end

# Create an "admins" group on the system
# You might use this group in the /etc/sudoers file
# to provide sudo access to the admins
group "admins" do
  gid     999
  members admins
end
```

Creating and Editing Data Bags within a Recipe

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Danger!
If two clients are simultaneously attempting to update a data bag, the data written last wins. This can lead to data loss if the user is not careful about multiple clients writing to data bags concurrently.

Altering data bags from the node when using the Open Source chef-server requires giving the node's API client admin privileges. In most cases, this is not advisable.

Please use this methods in this section with extreme care.

Data bags can be created within recipes using a variety of methods. The following examples describe how to create and edit data bag and data bag items within recipes by directly using the methods provided by the `Chef::DataBag` and `Chef::DataBagItem` objects.

Creating a Data Bag within a Recipe

```
users = Chef::DataBag.new
users.name("users")
users.save
```

Creating a Data Bag Item within a Recipe

```
sam = {
  "id" => "sam",
  "Full Name" => "Sammy",
  "shell" => "/bin/zsh"
}
databag_item = Chef::DataBagItem.new
databag_item.data_bag("users")
databag_item.raw_data = sam
databag_item.save
```

Editing a Data Bag Item within a Recipe

```
sam = data_bag_item("users", "sam")
sam["Full Name"] = "Samantha"
sam.save
```

Data Bags and Environments

Data Bags are global and can be accessed by nodes in any environment. For details on strategies that you can use to provide per-environment values in data bags, see Data Bags and Environments on the Environments page.

Using Data Bags with Chef Solo

Chef 0.10.4+ Only
Since 0.10.4 Data bags are available for use in **Chef Solo** as well as in the **Chef Server**.

Data bags can also be used by chef-solo. Chef solo loads data bags from a directory structure on the local file system. The location of the data bag directory is configurable with the `data_bag_path` configuration option in `solo.rb`. Each subdirectory within `data_bag_path` corresponds to a data bag. JSON files within each data bag directory corresponds to a data bag item. This is the same convention used by knife and described in the Data Bags from File section.

Since search is not available in recipes run with chef-solo, you must use the `data_bag()` and `data_bag_item()` functions to access data bags and items within a recipe.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Search using Chef Solo
A community member, Edelight, created a chef-solo search implementation for data bags. This functionality is provided via a cookbook that can be found on GitHub:

https://github.com/edelight/chef-solo-search



Cookbooks

Environments



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Encrypted Data Bags



Encrypted Data Bags

Data Bags are a great way to store and access data infrastructure wide, so long as it isn't associated directly with node or role attributes.

Encrypted Data Bags allow sensitive information, such as database passwords, to be stored in data bags in an encrypted form.

This protects the sensitive data from exposure if the Chef server is compromised and allows the data bag to be managed in a source control system without having plain-text passwords in revision history.

Only designated nodes will be able to decrypt values stored in an encrypted data bag because a shared secret is required for decryption.

Encrypted Data Bags provide the following features

1. Only the values of data bag items are encrypted, so that keys are still searchable.
2. The system uses shared key encryption; **users distribute the shared key to authorized nodes and users**.
3. Recipes can load encrypted data bags and access values like regular Data Bags. Decryption relies on the shared secret being present in a file on the node or accessible via a URI path.
4. Encrypted Data Bag support is integrated into knife. The `knife data bag *` commands make it possible to create, edit, and show encrypted data bags using the `--secret` and `--secret-file` options.

Chef 0.10 Overview

Check out the **Opscode Blog** for additional information on Encrypted Data Bags.

Encrypted data bags and knife

The encrypted data bag features of the `knife data bag *` subcommands are activated by specifying either a `--secret` or `--secret-file` option.

- `--secret SECRET` is used to specify the key used for encryption and decryption on the command line.
- `--secret-file FILE` is used to specify the path to a file that contains the encryption/decryption key.

Create a new encrypted data bag item

Verify that the data bag has been created and encrypted

Decrypt an encrypted data bag item

Other knife encrypted data bag commands

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

When given a `--secret` or `--secret-file` option,

- `knife data bag from file` will **encrypt** the data bag item contained in the specified JSON file;
- `knife data bag edit` will **decrypt** an existing data bag item, allow you to edit it, and then encrypt it before saving it to Chef

Accessing encrypted data bags from a recipe

In order for recipes running on a node to decrypt data bag items, the node must have access to the key used to encrypt the data bag. You can specify a secret directly by using `Chef::EncryptedDataBagItem.load` like this:

```
mysql_creds = Chef::EncryptedDataBagItem.load("passwords", "mysql", secret)
mysql_creds["pass"] # will be decrypted
```

You can store the encryption key in a file on the nodes that need it and configure Chef so that it knows where to look using `Chef::Config[:encrypted_data_bag_secret]` which defaults to `/etc/chef/encrypted_data_bag_secret`. In this case, you can omit the third argument to load and have Chef load the secret for you based on the `encrypted_data_bag_secret` configuration:

```
# look for secret in file pointed to by encrypted_data_bag_secret config.
# If not set explicitly use default of /etc/chef/encrypted_data_bag_secret
mysql_creds = Chef::EncryptedDataBagItem.load("passwords", "mysql")
mysql_creds["pass"] # will be decrypted
```

You can also store the encryption key in an alternate file on the nodes that need it and specify the path location to the file inside an attribute; however, EncryptedDataBagItem.load expects to see the actual secret as the third argument, rather than a path to the secret file. In this case, you can use EncryptedDataBagItem.load_secret to slurp the secret file contents and then pass them:

```
# inside your attribute file:
# default[:mysql][:secretpath] = "C:\\chef\\any_secret_filename"
#
# inside your recipe:
# look for secret in file pointed to by mysql attribute :secretpath
mysql_secret = Chef::EncryptedDataBagItem.load_secret("#{node[:mysql][:secretpath]}")
mysql_creds = Chef::EncryptedDataBagItem.load("passwords", "mysql", mysql_secret)
mysql_creds["pass"] # will be decrypted
```

Example

To demonstrate the use of encrypted data bags on a node, we'll start by copying the `secret_key` file created above to our example node using scp and moving it to `/etc/chef/encrypted_data_bag_secret`.

```
scp ./secret_key $MY_NODE_IP:~/
ssh $MY_NODE_IP
sudo mv ./secret_key /etc/chef/encrypted_data_bag_secret
```

As of 0.10.6, `knife bootstrap` supports the 'encrypted_data_bag_secret' setting in knife.rb. You will want to add this line to your knife.rb:

```
encrypted_data_bag_secret '/path/to/your/data_bag_key'
```

And change '/path/to/your/data_bag_key' to the location of where the data bag key is located. When you run knife bootstrap afterwards it automatically adds this line to the client.rb for the node you are bootstrapping and copies the key over.

Next, we'll create a recipe that will log the decrypted values for demonstration purposes (if these were real secrets, you would want to avoid logging them).

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife cookbook create edb_demo
```

Now edit `cookbooks/edb_demo/recipes/default.rb` so that it contains the following:

```
# cookbooks/edb_demo/recipes/default.rb
passwords = Chef::EncryptedDataBagItem.load("prod", "passwords")
mysql = passwords["mysql"]
Chef::Log.info("The mysql password is: '#{mysql}'")
```

Finally, upload the cookbook and run chef-client on the node. You should see something like this:

```
knife cookbook upload edb_demo
# output clipped
knife ssh name:i-8a436fe5 -a ec2.public_hostname &amp;#8216;sudo chef-client&amp;#8217;
INFO: *** Chef 0.10.0 ***
INFO: Run List is [recipe[edb_demo]]
INFO: Run List expands to [edb_demo]
INFO: Starting Chef Run for i-8a436fe5
INFO: Loading cookbooks [edb_demo]
INFO: The mysql password is: &amp;#8216;open-sesame-123&amp;#8242;
INFO: Chef Run complete in 3.122228 seconds
INFO: Running report handlers
INFO: Report handlers complete
```

As you can see, the recipe was able to decrypt the values in the encrypted data bag. It did so by using the shared secret located in the default location of `/etc/chef/encrypted_data_bag_secret`.



Encrypted Data Bag Implementation Example

Opscode team member Joshua Timberman put together a blog entry on setting up **Encrypted Data Bag for Postfix SASL Authentication** that should also serve as a nice example to those seeking use of encrypted data bags in their environment.

Implementation notes

- Only data bag item values are encrypted so that searches can be executed based on the keys.
- The values associated with the "id" key of a data bag item is **not encrypted** because it is needed for Chef to track the data bag item.
- Each data bag item value is encrypted by converting it to YAML, encrypting using AES 256 CBC via the openssl Ruby library, and then base 64 encoding the encrypted data.
- The EncryptedDataBagItem class is a read-only wrapper that decrypts data bag item values on the fly.
- The data bag secret can also reference any URI that Ruby's

  ```
  open-uri
  ```

  library supports.

Data Bags

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

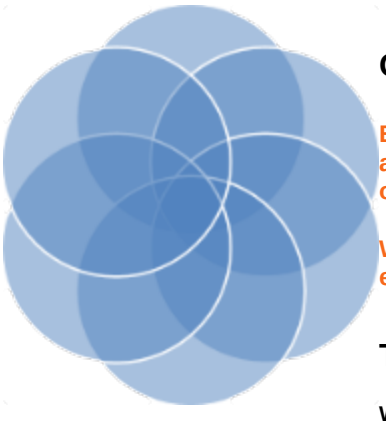

Environments



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Environments

Requires Chef 0.10
The Environments feature is only available in chef 0.10.x and higher.



Overview

Environments in Chef provide a mechanism for managing different environments such as production, staging, development, and testing, etc with one Chef setup (or one organization on Hosted Chef).

With environments, you can specify per environment run lists in roles, per environment cookbook versions, and environment attributes.

The _default Environment

When you first create an organization, only one environment will be setup and it will be named the "_default" environment. You can use this environment for viewing, but editing is restricted so you can't modify it. If you try to, you will get an error similar to this:

```
ERROR: Method Not Allowed
Response: Merb::ControllerExceptions::MethodNotAllowed
```

You'll want to create a new environment if you want to modify it, such as setting the attributes.

Creating Environments

You can create environments in Chef in 5 different ways:

1. through the creation of environment files in your chef repository (that utilize a ruby DSL, which gets compiled to JSON,)
2. the creation of the JSON files directly in your chef repository,
3. through the Open Source Chef Server Management Console,
4. with Knife, or
5. through the REST API.

We will cover each option here. When an environment exists, you can set a node to be in the environment using the node's chef_environment method. The remaining sections of this article will show the details of the work-flow.

Chef 0.10
Check out the Opscode Blog for an overview of the Environments feature within chef 0.10.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Choose Your Workflow

There are two workflows you can use to manage your environments: you can either write your environments as ruby or JSON files and push them to the server, or you can create environments with Knife or the Open Source Chef Server Management Console and edit them as you go along.

If you manage environments with files

- You can use the ruby DSL or JSON
- You can dynamically generate environment data using the ruby DSL
- You can keep your environments in your version control system and have a history of who changed what and when.
- You **should not** edit environments with Knife or the Open Source Chef Server Management Console, as your edits will be overwritten next time you upload from the file.

If you manage environments with Knife, the REST API and the Open Source Chef Server Management Console

- You can use the `-E ENVIRONMENT` switch to `knife cookbook upload` to automatically configure an environment for a given cookbook version.
- You can edit your environment with `knife` or the web interface.
- The canonical source of a environment's data will be the Chef Server, which makes it difficult to keep in version control
- You cannot use the ruby DSL.

***These workflows are mutually exclusive**: if you upload a environment to your Chef server from a file, then edit it with Knife or the Open Source Chef Server Management Console, and then edit the file and upload again, the changes you made with Knife or the Open Source Chef Server Management Console will be lost.*

The Ruby DSL

Environments created through this mechanism get compiled to JSON, and then are loaded in to the Chef Server. (We never execute ruby code directly on the chef server!) Each time you rake install your Chef Repository, we will re-compile the corresponding JSON and store it in the chef server.

You should create these files in the 'environments' subdirectory of your chef repository. If your repository doesn't have this directory, create it now. Each DSL file should have a .rb suffix.

A complete environment file looks like this:

The Dev Environment

```
name "dev"
description "The development environment"
cookbook_versions  "couchdb" => "11.0.0"
default_attributes "apache2" => { "listen_ports" => [ "80", "443" ] }
```

We'll go over each section below.

name

Each environment must have a unique name, which is made up of [A-Z][a-z][0-9] and [_-]. Spaces are not allowed.

name field

```
name 'dev'
```

description

A short description of this environment.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

description field

```
description 'The development environment'
```

cookbook_versions

By specifying cookbook_versions, when chef-client is running on a node within a certain environment, it will pick the specified versions of the cookbooks.

cookbook_versions

```
cookbook_versions({
  "couchdb"=>"= 11.0.0",
  "my_rails_app"=>"~> 1.2.0"
})
```

Please see Cookbook Version Constraints for more on the syntax for cookbook versions and version constraints, to limit which versions of a cookbook should be used.

cookbook

Specify the version constraints for only one cookbook.

cookbook

```
cookbook "couchdb", "= 11.0.0"
cookbook "my_rails_app", "~> 1.2.0"
cookbook "gems", "< 1.4.0"
```

attributes

Environments can set attributes at the `default` and `override` levels. See the attribute precedence list for a complete description of the precedence of attributes set in an environment. An environment attribute will be applied to all nodes within this environment, except in those places where it is overridden by another attribute with higher precedence.

attributes

```
default_attributes "apache2" => { "listen_ports" => [ "80", "443" ] }
```

In the above example, all nodes with this environment would have `node[:apache2][:listen_ports]` set to '80' and '443', unless otherwise specified by an attribute of higher precedence.

As JSON

The JSON format for environments maps directly to the Ruby DSL above. For the environment we describe in that section, the corresponding JSON is:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
{
  "name": "dev",
  "default_attributes": {
    "apache2": {
      "listen_ports": [
        "80",
        "443"
      ]
    }
  },
  "json_class": "Chef::Environment",
  "description": "",
  "cookbook_versions": {
    "couchdb": "11.0.0"
  },
  "chef_type": "environment"
}
```

The two additional fields are described below.

json_class

This should always be set to `Chef::Environment`. This is used internally by Chef to auto-inflate this type of object. It should be ignored if you are re-building objects outside of Ruby, and its value may change in the future.

chef_type

This should always be set to `environment`. This is the field you should rely on if you are building a system to consume Environments outside of Ruby.

Managing Environments With Knife

Knife is Chef's powerful command line interface tool. Environments can be managed through Knife. See **Managing Environments With Knife** for accomplishing this.

Managing Environments With the Management Console

You can create and manage environments through the Management Console. Hosted Chef customers should refer to **Managing Environments with the Hosted Chef Management Console**. Open Source Chef Server users should refer to **Managing Environments through the Management Console**.

The REST API

You can also create and manage environments directly in a running Chef Server via the REST API. Please refer to the Server API article for information.

Setting a Node's Environment

A node is considered by Chef associated with an environment by having the chef_environment "attribute" set

There are several ways to set this attribute:

- Edit the "chef_environment" attribute on the node directly using Knife or the Open Source Chef Server Management Console.
- Set "environment" config entry in Knife's config file knife.rb, and bootstrap the node by using "knife bootstrap" command, the instance bootstrapped will be in the specified environment.
- Set "environment" config entry in chef-client's config file (by default /etc/chef/client.rb), when chef-client runs, it will pick up the value and set the chef_environment attribute of the node based on the value.
- When creating/updating a node programmatically, suppose node is a Chef::Node object, calling node.chef_environment("dev") and

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

node.save will update the chef_environment attribute on the node to "dev".

Note: The chef_environment 'attribute' cannot be set with normal or override attributes (i.e. in a role) since it is actually a method. *It must be set explicitly via knife edit, knife exec, or one of the methods listed above.* It also cannot be accessed via the typical attribute syntax `node[:attribute_name]` but rather must be accessed by calling the chef_environment method on the node: `node.chef_environment`.

Moving nodes in bulk

You can move nodes from, for instance, "_default" to your "dev" environment on the command line with:

```
knife exec -E 'nodes.transform("chef_environment:_default") { |n| n.chef_environment("dev") }'
```

Knife Plugin to Set Environment

You can take advantage of another Chef 0.10.0 feature, custom Knife plugins, to facilitate setting the node environment. Here's a plugin to set the node environment.

Per Environment Run Lists in Roles

In addition to being able to pick specific cookbook versions in an environment, per environment run lists in a role allow you to specify a different run list for a role if the role is in the run list of the node in a specific environment. Before the existence of environments, each role has only one run list and it will be expanded and applied to the node when chef-client runs. With environments, in the role you can specify one run list for each environment, such as:

```
{
  "name": "webserver",
  "default_attributes": {
  },
  "json_class": "Chef::Role",
  "env_run_lists": {
    "production": [

    ],
    "preprod": [

    ],
    "test": [
      "role[base]",
      "recipe[apache]"
    ],
    "dev": [
      "role[base]",
      "recipe[apache]",
      "recipe[apache::copy_dev_configs]"
    ]
  },
  "run_list": [
    "role[base]",
    "recipe[apache]"
  ],
  "description": "The webserver role",
  "chef_type": "role",
  "override_attributes": {
  }
}
```

If you do not specify a per environment run list for a certain environment, such as "production" and "preprod" in the above example, chef will use the default run list (like before).

Roles has additional information on role based runlists, including an example of a per environment runlist in .rb format.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Searching within Environments

Within search, chef_environment is treated much like an attribute. Thus search results can be limited to a given environment by using a boolean operator and an extra search term.

To find all the servers running CentOS in the QA environment using search:

Search within a Chef environment.

```
knife search node "chef_environment:QA AND platform:centos"
```

In a recipe, a code block like this would be useful:

```
qa_nodes = search(:node,"chef_environment:QA")
qa_nodes.each do |qa_node|
    # Do useful specific to qa nodes only
end
```

Data Bags and Environments

Data bags are global to your organizations are available in any environment. If you would like to provide per-environment data within a data bag, you can use one of two possible strategies. The strategy that makes the most sense depends on your use case.

1. Within a databag item, use a top level key that corresponds to the environment. For instance, you might have a databag that looks like this:

   ```
   {
     "id": "some_data_bag_item",
     "production" : {
       # Hash with all your data here
     },
     "testing" : {
       # Hash with all your data here
     }
   }
   ```

   When using the data bag in a recipe, you would then always access the data bag items looking via code similar to:

   ```
   bag_item[node.chef_environment]["some_other_key"]
   ```

2. Within a data bag, use separate items for each environment. Depending on your data, all of your data may fit nicely in a single item. If this is the case, then creating different items for each environment is an easy way to provide per-environment values within a data bag.

For more information about using data bags, see Data Bags.

Environments

Exception and Report Handlers

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Exception and Report Handlers



Exception and Report handlers are a feature of Chef that allow you to run code in response to a Chef run succeeding or failing.

The most obvious use is to dispatch notifications if a Chef run fails, though it is also possible to gather rich data about your chef usage for trending and analysis.

Anatomy of a handler

A handler is a class inheriting from `Chef::Handler` and implementing `report()` function. Before `report()` is actually run, `run_status` is initialized by chef. `run_status` is an instance of `Chef::RunStatus`, a class that keeps track of the chef run status (gee, there's a surprise...). `run_status` provides several useful properties for reporting, have a look at the table:

Name	Description
success?/failed?	Whether the chef run was successful
backtrace	The backtrace from exception, if any
exception	The raw exception, if any
formatted_exception	The exception as a formatted string, e.g. "ExceptionClass: message"
node	The node for the client run
all_resources	The list of all resources in the current run context's `resource_collection`
updated_resources	The list of all resources in the current run context's `resource_collection` that are marked as updated
elapsed_time	The time elapsed between the start and finish of the chef run
start_time/end_time	The time the chef run started or ended
run_context	An instance `Chef::RunContext` that keep track of the context of the chef run. It provides access several useful properties: `cookbook_collection`, `resource_collection`, `definitions`

A handler may be assigned as a report or exception handler, there's no real difference in the code. Use `success?` or `failed?` if you need to know what situation caused it to run.

Tutorials from the Community

- ***A very simple report handler***

Opscode team member Joshua Timberman demonstrates a very simple report handler to make it clearer what resources were actually configured in the Chef run.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Writing a Handler

Let's start by writing a simple handler to send an email when a chef run fails. We'll keep things simple by using the pony library to send the email. For this example to work, you'll need to be able to send mail via `/usr/sbin/sendmail` or SMTP to localhost, but it should be pretty easy to extend the example if you need to.

```
require 'rubygems'
require 'pony'

module MyOrganization
  class EmailMe < Chef::Handler

    def initialize(from_address, to_address)
      @from_address = from_address
      @to_address   = to_address
    end

    def report
      # The Node is available as +node+
      subject = "Chef run failed on #{node.name}\n"
      # +run_status+ is a value object with all of the run status data
      message = "#{run_status.formatted_exception}\n"
      # Join the backtrace lines. Coerce to an array just in case.
      message << Array(backtrace).join("\n")

      Pony.mail(:to => @to_address, :from => @from_address, :subject => subject, :body => message)


    end
  end
end
```

Installing and Configuring a Handler

You can install and configure handlers either through use of the chef-handler LWRP, or manually.

In a recipe with the chef_handler LWRP

The `chef_handler` LWRP that ships as part of the chef_handler cookbook provides an easy to use idempotent resource that can be included in your recipe code.

Configuring the Handler with chef_handler LWRP

```
chef_handler "MyOrganization::EmailMe" do
  source "/var/chef/handlers/email_me"
  action :enable
end
```

To manually install and configure a handler, you'll need to use edit the Chef config file.

In the Chef config (client.rb) file

There is currently no default install location for handlers. The simplest way to distribute and install them is via rubygems, though any method of distributing the code (i.e., github, HTTP, etc.) will work. Once the handler is installed on the system, enable it in your client.rb file by requiring it. If you're using rubygems, you can require the handler by name:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Loading a Handler Installed via Rubygems

```
# /etc/chef/client.rb
require "rubygems"
require "my-sweet-handler"
```

If you installed the handler by some other method, you will probably need to require it using the full path to the file:

Loading a Handler Installed to an Arbitrary Location

```
# /etc/chef/client.rb
require "/var/chef/handlers/my-awesome-handler"
```

Once the handler is enabled, you may need to configure it. The way this is done will vary for each handler. If you have a simple handler that requires no additional configuration, you will most likely just need to create a new instance of the handler.

For example, if you have a handler `MyOrganization::EmailMe` in which you hardcoded all the values it needs to send email, you would simply create a new instance of it:

Create a New Instance of the MyOrganization::EmailMe Handler

```
email_handler = MyOrganization::EmailMe.new
```

Finally, configure the handler to notify you for either failed chef runs or successful ones.

Configuring Chef to Use Your Handler

```
report_handlers << email_handler     # these fire at the end of a successful run
exception_handlers << email_handler  # these fire at the end of a failed run
```

Putting it all together, your `client.rb` file would have a section like this:

Configuring the Handler in Your client.rb

```
# your handler is installed in /var/chef/handlers/email_me.rb
require "/var/chef/handlers/email_me"

# Assuming your handler is simple and doesn't need to be configured
email_handler = MyOrganization::EmailMe.new

# Configure Chef to use your handler
report_handlers << email_handler     # these fire at the end of a successful run
exception_handlers << email_handler  # these fire at the end of a failed run
```

Distributing Chef Handlers

See Distributing Chef Handlers for use of the Opscode provided cookbook to accomplish this task.

Existing Handlers

Chef ships with a handler that will serialize the run status data to a JSON file. To use it, put the following in /etc/chef/client.rb or /etc/chef/solo.rb.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

/etc/chef/client.rb to use Chef's JsonFile handler

```
require 'chef/handler/json_file'
report_handlers << Chef::Handler::JsonFile.new(:path => "/var/chef/reports")
exception_handlers << Chef::Handler::JsonFile.new(:path => "/var/chef/reports")
```

LWRP entry to use Chef's JsonFile handler

```
chef_handler "Chef::Handler::JsonFile" do
  source "chef/handler/json_file"
  arguments :path => '/var/chef/reports'
  action :enable
end
```

The data can be loaded and inspected via irb:

```
irb(main):001:0> require 'rubygems'
=> true
irb(main):002:0> require 'json'
=> true
irb(main):003:0> require 'chef'
=> true
irb(main):004:0> r = JSON.parse(IO.read("/var/chef/reports/chef-run-report-20110322060731.json"))
=> ... output truncated
irb(main):005:0> r.keys
=> ["end_time", "node", "updated_resources", "exception", "all_resources", "success", "elapsed_time",
"start_time",
 "backtrace"]
irb(main):006:0> r['elapsed_time']
=> 0.00246
```

Community Based Handlers

Exception and report handlers from the Chef Open Source Community.





- **Airbrake exceptions**
  Chef handler for sending exceptions to Airbrake (exceptions only)
- **Asynchronous Resources Handler**
  Chef report handler that pushes changes to a stomp queue, see blog post
- **Campfire handler**
  Chef handler for sending exceptions and reports to campfire
- **chef-handler-graphite**
  Chef report handler to push metrics to graphite. YEAH!
- **chef-irc-snitch**
  An exception handler for Opscode Chef runs (github gists & IRC)
- **Cloudkick handler**
  Chef handler for sending exceptions and reports to cloudkick
- **Datadog handler**

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef handler for getting Chef stats directly into the Datadog newsfeed each time the client runs, extended into a Rubygem from chef-data dog. See blog post

- **GELF/Graylog2 handler**
  Chef handler which can report run status, including any changes that were made, to a Graylog2 server.
- **Growl handler**
  Chef report handler to send Growl notification
- **HipChat room message**
  HipChat HTTP API Wrapper in Ruby with Capistrano hooks (exceptions only)
- **Mail report handler**
  Simple Chef report handler that sends Erubis template based reports via the Pony gem
- **nsca based chef report handler**
  Chef report/exception handler plugin that submits passive checks to nagios via nsca
- **Simple Updated Resources Handler**
  Chef report handler to display resources updated in the Chef Run, see blog post

There are Community Plugins for Chef, Ohai, and Knife as well.

 Environments

Distributing Chef Handlers 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Distributing Chef Handlers



Opscode provides a cookbook for distributing custom Chef Handlers.

Add cookbook to chef-repo

Download the cookbook to your Chef Repository.

```
% knife cookbook site install chef_handler
Installing chef_handler to /Users/schisamo/dev/chef_repositories/development/.chef/../cookbooks
Checking out the master branch.
Creating pristine copy branch chef-vendor-chef_handler
Downloading chef_handler from the cookbooks site at version 1.0.0 to
/Users/schisamo/dev/chef_repositories/development/.chef/../cookbooks/chef_handler.tar.gz
Cookbook saved:
/Users/schisamo/dev/chef_repositories/development/.chef/../cookbooks/chef_handler.tar.gz
[ ... SNIP ... ]
Cookbook chef_handler version 1.0.0 successfully installed
```

Write Chef Handler

See the Exception and Report Handlers page for instructions on how to write new Chef handlers. These files will go under `cookbooks/chef_handler/files/default/handlers`.

```
$EDITOR cookbooks/chef_handler/files/default/handlers/cloudkick_handler.rb
```

Change the Handler Path

If you wish to change the directory where the handlers are copied, edit the cookbook attributes file and change the path for the `node['chef_handler']['handler_path']` attribute. You can also set this value in a role.

```
default['chef_handler']['handler_path'] = "/var/chef/handlers"
```

Enable the Chef Handler with the `chef_handler` LWRP

After you distribute handler files down to the node, you have to enable it so Chef will know to fire it on the conclusion of Chef runs. This is very easy to do using the `chef_handler` LWRP. This LWRP requires, configures and enables handlers on the node for the current Chef run. It also has the ability to pass arguments to the handlers initializer, allowing initialization data to be pulled from a node's attribute data.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
chef_handler "CloudkickHandler" do
  source "#{node['chef_handler']['handler_path']}/cloudkick_handler.rb"
  arguments [node['cloudkick']['oauth_key'], node['cloudkick']['oauth_secret']]
  action :enable
end
```

It is best to declare `chef_handler` resources early on in the compile phase so they are available to fire for any exceptions during the Chef run.
If you have a base role you would want any recipes that register Chef handlers to come first in the run_list.

Upload to your Chef Server

The cookbook needs to be uploaded to the Chef Server to be used.

```
knife cookbook upload chef_handler
Uploading chef_handler             [1.0.0]
upload complete
```

Add the recipe to a run list

Add the `recipe[chef_handler]` to a node or role run list. Put it first to make sure your custom handlers are loaded and available to other recipes.
When Chef runs, during the compile phase the handlers will be copied into place and be available for enabling with the `chef_handler` LWRP.

Additional Information

See the README.md of the `chef_handler` cookbook for additional information.

 Exception and Report Handlers

Lightweight Resources and Providers (LWRP) 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Nodes



A node is a host that runs the Chef client. The primary features of a node, from Chef's point of view, are its Attributes and its run list. Nodes are the thing that Recipes and Roles are applied to.

In practice, there is usually a one-to-one mapping between a node and a physical device (a computer, a switch, a router, etc.), but in special cases a system may execute the recipes for multiple nodes.

Components of a Node

A node is made up of two prime components: a Run List and Attributes.
The Run List is an ordered list of Recipes or Roles to run. Recipes are the fundamental building block of Chef - they define the resources you want managed, in the order you want them managed. Roles are ordered lists of other Roles and Recipes, expanded at run time. Attributes are data about your node - things like the network interfaces, file systems, or how many clients your Apache server can accept.

Lifecycle of a Node

When you start Chef Client or Chef Solo, the first thing we do is create a Node object. We then:

1. load up Ohai, which detects information about the operating system ("Ohai speaks for the operating system"). This also gives us access to this hosts fully qualified domain name and hostname, which we use to
2. grab the last known state of the Node from the Chef Server (assuming we are using Chef Client). Once there,
3. we update all the Ohai attributes to their latest value, and add any extra attributes specified via JSON on the command line.
4. Finally, we will run all of the Attribute files in all of the cookbooks in your Chef repository.

At runtime, the Node is available within Recipes, Definitions, and Resources as they are being evaluated.

Authentication

Chef uses Signed Header Authentication for each request. Node data is separate from the identity used to authenticate requests to the Chef Server, which is managed by an API client. Each API client has a public/private key pair. The public half of the key is stored in the database on the server, and the private half is kept locally by the client. On a node running the Chef Client, the private key is generally stored in /etc/chef/client.pem. Each request to the Chef server includes a request signature in the HTTP headers. The request signature is computed from the hash of the request content and encrypted with the client's private key to verify the identity of the user or machine making the request and prevent attempts to tamper with the content.

When a node runs chef-client for the first time, it generally does not yet have an API client identity, and so cannot make authenticated requests to the server. This is where the validation client—named "chef-validator" by default comes in. When the chef-client runs, it checks if it has a client key. If the client key does not exist, it then attempts to "borrow" the validation client's identity to register itself with the server. In order to do that, the validation client's private key needs to be copied to the host and placed in /etc/chef/validation.pem. Once the client machine has registered itself with the chef server, it no longer uses the validation client for anything. It is recommended that you delete the validation client's private key from the host after the host has registered.

Authentication and Authorization provides additional detail on the Chef model for these actions.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Nodes

The recipes list and attributes of nodes may be managed through multiple means.

Managing Nodes via Knife

Knife is Chef's command line tool, and can be used to create, edit, view, list, tag and delete nodes. Refer to the Knife documentation for additional details on knife.

- **Managing Nodes With Knife** provides information on how this powerful command line tool can be used to manage the chef-client nodes in your infrastructure.

Managing Nodes via the Management Console

The Management Console is Chef's web interface. Nodes can be managed in the Management Console.

- **Managing Nodes through the Management Console** provides documentation on Open Source Chef users.
- Hosted and Private Chef customers should refer to **Managing Nodes with the Hosted Chef Management Console**.

Managing Nodes through JSON at the command line

Both Chef Client and Chef Solo allow you to write JSON to a file and read it in through the command line. The file should contain a hash, the elements of which will be added as attributes on your Node.

We have two special entries that can be used to set which cookbooks/recipes will be applied during run-time.

- `run_list` - specify roles and/or recipes to add to the node.
- `recipes` - add only recipes to the node. This should not be used, please always use run_list in this purpose.



For all you AWS users
This is an awesome way to use per-instance metadata. You can just stick some JSON in, and have it written to a file on disk, which you read with the Chef Client or Chef Solo to configure your nodes. Easy peasy!

Managing Nodes via a Cookbooks Attribute or Recipe file

You can also ensure that an attribute or is set via an Attribute or Recipe file.

Use Case Questions

How do I run more than one node on a system or change a node name?

You can use the `-N` option to `chef-client` or `chef-solo` to set the node name manually.

Can you name a node anything you choose?

The only restriction Chef places on node names is that they match this regular expression:

```
/^[\-[:alnum:]_:.]+$/
```

 Lightweight Resources and Providers (LWRP)

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Setting the run_list in JSON during run time 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Setting the run_list in JSON during run time

Work in progress on this page.
nuo@opscode.com

The JSON attributes file can be used to add specific Roles or Recipes to a node during chef-client/chef-solo run time.

For example, the JSON file should look like:

base.json

```
{ "run_list": [ "role[base]" ] }
```

ops_master.json

```
{ "run_list": [ "role[base]", "role[ops_master]" ] }
```

ops_monitor.json

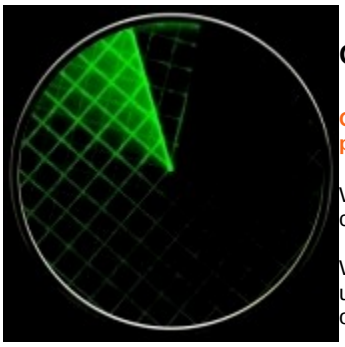
```
{ "run_list": [ "role[base]", "recipe[nagios::server]", "recipe[ganglia::server]" ] }
```

Then run chef-client/chef-solo with -j, for example:

```
chef-client -j base.json
```


This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Ohai



Ohai Overview

Ohai detects data about your operating system. It can be used standalone, but its primary purpose is to provide node data to Chef.

When invoked, it collects detailed, extensible information about the machine it's running on, including Chef configuration, hostname, FQDN, networking, memory, CPU, platform, and kernel data.

When used standalone, Ohai will print out a JSON data blob for all the known data about your system. When used with Chef, that JSON output is reported back via "automatic" node attributes to update the node object on the chef-server.

Ohai Use and Plugins

This section of the Wiki describes using Ohai and writing plugins for it.

- **Ohai Installation and Use**
- **Writing Ohai Plugins**
- **Loading Custom Ohai Plugins**
- **Distributing Ohai Plugins**
- **Disabling Ohai Plugins**
- **Ohai Release Notes**

Ohai Version Dependencies

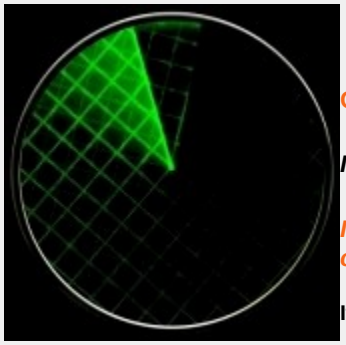
- Use of Ohai Version 0.6.x, requires the use of Chef version 0.9.x or greater.
- Chef version 0.10 requires Ohai Version 0.6.x or greater, and will not function with earlier versions.

Nodes

Ohai Installation and Use

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Ohai Installation and Use



Opscode distributes Ohai as a Gem, available from http://rubygems.org/

It is a requirement for Chef, so if you have Chef installed, you have Ohai.

If you would like to install Ohai separately, for testing or development, read on.

If you're running Windows, see Installing Ohai on Windows for more information.

Install Gem

Add Rubygems.org's gem server to your local sources.

Update Gem Sources

```
sudo gem source -a http://rubygems.org
```

Install the Gem.

Install Ohai

```
sudo gem install ohai
```

This will also install Ohai's dependency gems, json, extlib and systemu.

Ohai Version Dependencies

- Use of Ohai Version 0.6.x, requires the use of Chef version 0.9.x or greater.
- Chef version 0.10 requires Ohai Version 0.6.x or greater, and will not function with earlier versions.

Install from Git

We use GitHub for sharing our source code, so to install the source from Git:

clone git repository

```
git clone git://github.com/opscode/ohai.git
```

Install the gem with rake.

rake install

```
rake install
```

See the README.rdoc included in the repository (or on the GitHub page) for more about Ohai development. Also read about contributing to Opscode projects

Using Ohai

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- ***Node data from ohai is provided to Chef via an object named node.***

  When used with Chef, Ohai JSON output is reported back via "automatic" node attributes to update the node object on the chef-server.

- ***Ohai can also be run outside of Chef, so you can query your system for various data.***

  When run standalone, Ohai will print out a JSON data blob for all the known data about your system.

- ***Use as a Library***

  You can use Ohai as a library within a ruby program or script.

 Ohai

Writing Ohai Plugins 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Installing Ohai on Windows

You will need Ohai version 0.5.4 or better for functionality on Windows.

See Release MVPs for the latest Ohai version, linked off to release notes and announcement.

The following platforms are known to work and return gobs of data (mainly through WMI):

- Windows 7 Ultimate RC1
- Windows 7 Enterprise
- Windows XP Professional
- Windows Vista Ultimate
- Windows 2000 Server SP4
- Windows 2003 Server R2 Standard
- Windows 2008 Server R2 Standard

Ohai Version Dependencies

- Use of Ohai Version 0.6.x, requires the use of Chef version 0.9.x or greater.
- Chef version 0.10 requires Ohai Version 0.6.x or greater, and will not function with earlier versions.

Install Ruby via one-click installer:

- http://rubyforge.org/frs/download.php/69034/rubyinstaller-1.8.7-p249-rc2.exe

Install ohai and wmi bindings

```
C:\Ruby>gem install ohai ruby-wmi
```

Run Ohai:

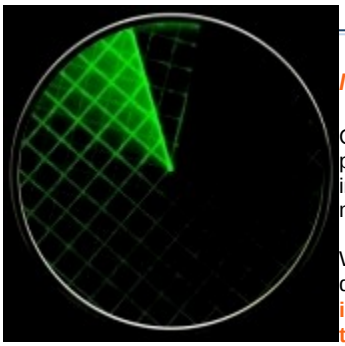
```
C:\Ruby>ohai
```

Example output:

- Windows 2003 Server R2

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Writing Ohai Plugins



Need to Gather Additional Information About Your Infrastructure?

Ohai detects data about your operating system, and can be used standalone. As it's primary purpose is to provide data to Chef, Ohai itself is included with Chef. When invoked, Ohai collects detailed, extensible information about the machine it's running on, including Chef configuration, hostname, FQDN, networking, memory, CPU, platform, and kernel data.

When used standalone, Ohai will print out a JSON data blob for all the known data about your system. That data is then available to you for whatever benefit you can derive through its use. **If you'd like additional information about your system infrastructure - you could write your own Custom Ohai Plugin to gather that information for you.**

Thanks to Community Member Christopher Sexton for this tutorial.

Create a simple plugin

Ohai plugins use a ruby DSL
The following is about as simple as it gets:

```
provides "orly"
orly "yeah, rly."
```

Loading Your Plugin

Create a "plugins" folder and put the above code in the plugins/orly.rb file.

Now to fire up irb (I am assuming you are in the directory that contains the "plugins" folder, if not adjust your path):

```
>> require 'ohai'
>> Ohai::Config[:plugin_path] << './plugins'
>> o = Ohai::System.new
>> o.all_plugins
>> o.orly #=> "yea, rly"
```

The entire script can be found in orly.rb.

If you run `o.orly` and get `nil` the chances are the plugin path is probably incorrect. I battled with this for hours banging my head against the wall. Turns out I just forgot the 's' on the end of './plugins'

Using a Mash

Most of the information we want to lookup would be nested in some way, and ohai tends to do this by storing the data in a Mash. This can be done by creating a new mash and setting the attribute to it.

In plugins/canhas.rb:

```
provides "canhas"
canhas Mash.new
canhas[:burger] = "want"
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Working with Different Platforms

One of the main reasons for using ohai is to gather information regardless of the operating system, luckily this is made easy by optionally loading Recipes based on the platform. With that platform specific calls abstracted away you can keep your code DRY.

The builtin plugins that come with ohai use the following trick to load platform specific code:

It works by creating a base cross-platform plugin that loads the platform specific plugin from a subdirectory. In plugins/lolcode.rb:

```
provides "languages/lolcode"
require_plugin "languages"
require_plugin "#{os}::lolcode"

languages[:lolcode] = Mash.new unless languages[:lolcode]
languages[:lolcode][:version] = "TEH VERSHIONS"
```

In plugins/darwin/lolcode.rb:

```
provides "languages/lolcode"
require_plugin "languages"
languages[:lolcode] = Mash.new unless languages[:lolcode]
languages[:lolcode][:platform] = "MACKERS"
```

Checkout ohai's os.rb for the list of platform names.

All of these examples can be found in the ohai-plugin-howto github repo, you should be able to clone that and run the ruby scripts in the repo's root directory. If you figure out any other handy tricks please fork the project and add them.

Extending an existing plugin

Ohai makes it very easy to extend a current plugin with new information. Simply require the plugin you want to extend and extend away. In this example we want to add LOLCODE to languages.

In plugins/lolcode.rb:

```
provides "languages/lolcode"
require_plugin "languages"
languages[:lolcode] = Mash.new
languages[:lolcode][:version] = "TEH VERSHIONS"
```

Chef Community Plugins

Review (or contribute to) a collection of **Community Plugins Found In The Wild**. You may find a plugin created by someone in the community that fills a need. And if you end up creating one that you believe may have appeal beyond your infrastructure, please contribute it to the community for others as well.

Loading and Distributing Custom Ohai Plugins

Loading Custom Ohai Plugins provides detail on how to load a custom plugin to chef-client. **Distributing Ohai Plugins** details how to distribute custom plugins to your Nodes.

Ohai Installation and Use

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

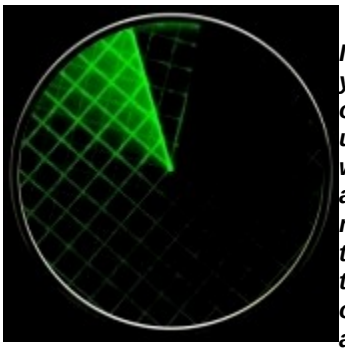


Loading Custom Ohai Plugins



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Loading Custom Ohai Plugins



If you want to add custom/unofficial Ohai plugins to a Chef client:

> **Handy Hint**
> As an example, you can find a custom plugin that extends virtualization attribute adding a guest_lists sub-attribute here:
>
> http://github.com/rubiojr/ohai-plugins/blob/master/virtualization_extensions.rb

1. Edit /etc/chef/client.rb
2. Add the following line to the end of the file: **You can choose any path you want.**

```
Ohai::Config[:plugin_path] << '/etc/ohai/plugins'
```

3. Add your custom plugins there (e.g. /etc/ohai/plugins)
4. Test your plugins with ohai

```
ohai -d /etc/ohai/plugins
```

5. Reload the client:

```
/etc/init.d/chef-client restart
```

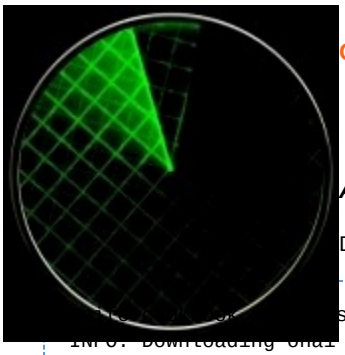
> Because of OHAI-126 you must NOT use a trailing slash with 'ohai -d /etc/ohai/plugins'

Writing Ohai Plugins

Distributing Ohai Plugins

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Distributing Ohai Plugins



Opscode provides a cookbook for distributing custom Ohai plugins.

Add cookbook to chef-repo

Download the cookbook to your Chef Repository.

```
stall ohai
INFO: Downloading ohai from the cookbooks site at version 0.9.0
INFO: Cookbook saved: /Users/jtimberman/chef-repo/cookbooks/ohai.tar.gz
INFO: Checking out the master branch.
INFO: Checking the status of the vendor branch.
INFO: Creating vendor branch.
INFO: Removing pre-existing version.
INFO: Uncompressing ohai version 0.9.0.
INFO: Adding changes.
INFO: Committing changes.
INFO: Creating tag chef-vendor-ohai-0.9.0.
INFO: Checking out the master branch.
INFO: Merging changes from ohai version 0.9.0.
[ ... SNIP ... ]
INFO: Cookbook ohai version 0.9.0 successfully vendored!
```

Write custom plugins

See the Writing Ohai Plugins page for instructions on how to write new Ohai plugins. These files will go under `cookbooks/ohai/files/default/plugins`. They should be Ruby files.

```
$EDITOR cookbooks/ohai/files/default/plugins/my_plugin.rb
```

Change the Plugin Path

If you wish to change the directory where the plugins are copied, edit the cookbook attributes file and change the path for the `node[:ohai][:plugin_path]` attribute.

```
default[:ohai][:plugin_path] = "/etc/chef/ohai_plugins"
```

Upload to your Chef Server

The cookbook needs to be uploaded to the Chef Server to be used.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife cookbook upload ohai
INFO: Saving ohai
INFO: Validating ruby files
INFO: Validating templates
INFO: Syntax OK
INFO: Generating Metadata
INFO: Uploading files
[ ... SNIP ... ]
```

Add the recipe to a run list

Add the `recipeOhai` to a node or role run list. Put it first to make sure your custom plugins are loaded and available to other recipes. When Chef runs, during the compile phase the plugins will be copied into place and then loaded and merged with the node. This does cause Ohai to be run twice which can add a few seconds to the total run time of Chef.

Additional Information

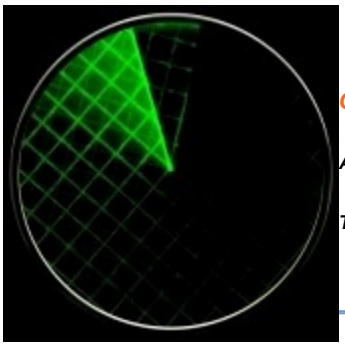
See the README.md of the Ohai cookbook for additional information.

 Loading Custom Ohai Plugins

Disabling Ohai Plugins 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Disabling Ohai Plugins



Often it's desirable not to load certain Ohai plugins.

An example of this is the passwd plugin, should your nodes be connected to a large LDAP directory.

To see long it takes to load various Ohai plugins, see https://gist.github.com/1323597

To do this, add the following line to `/etc/chef/client.rb`; this examples shows 4 plugins being disabled:

```
Ohai::Config[:disabled_plugins] = ["passwd", "rackspace", "dmi", "dmi_common"]
```

To see the list of available plugins run:

```
gem content ohai | grep lib/ohai/plugins
```

To test the effect of disabling plugins run the following command and look for the string "`Skipping disabled plugin XXXX`":

```
chef-client -l debug
```

No effect on the "ohai" executable

Please note that the `client.rb` configuration file is ignored by the `ohai` executable.

It is therefore not possible to disable plugins when running `ohai` manually.

Distributing Ohai Plugins

Providers

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Resources and Providers



In Chef, Resources represent a piece of system state and Providers are the underlying implementation which brings them into that state.

(All database vendors support the abstract concept of database creation, but the underlying implementation is different for each.)

More Information

Resources

Providers

Lightweight Resources and Providers (LWRP)

Opscode LWRP Resources

Resources are the basic units of work in Chef - a cross platform abstraction of a system's configuration that are declared in Recipes and Definitions, and applied to your nodes. For example, packages may be installed via apt, yum, or the BSD ports and packages systems, but the package resource abstracts these differences away so you can specify that a package should be installed in a cross-platform way. Chef's resources are mostly just containers for data, with some basic validation functionality.

Providers take a resource, compare that resource to the current state of the part of the system it is managing, and then takes the Action specified in the resource.



Providers are the way that Chef supports multiple platforms with a single Resource. Providers are applicable for an individual resource type. For instance, on Red Hat or CentOS, a package resource will use the yum package provider to get the package installed, but on Debian and Ubuntu, the apt package provider will be used. Providers contain most of the intelligence: they're responsible for making Chef idempotent by checking if an action needs to be taken and issuing the commands to the system to take that action.



While typical Resources and Providers are implemented in Chef's core using Ruby classes, implementing Lightweight Resources and Providers (LWRP) is quick and easy, requiring less Ruby knowledge than their heavier counterparts.

(LWRP's also become Ruby classes, but this is done for you, behind the scenes.) LWRPs are loaded from files in a Cookbook's "resources" and "providers" directories; through use of a DSL you can create custom LWRPs for your environment and infrastructure automation.

Opscode LWRP Resources are a set of LWRPs, provided by Opscode, that can be used in Opscode's public open source cookbooks.

These Opscode LWRP Resources are ready made for use within Recipes to increase automation of your environment.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



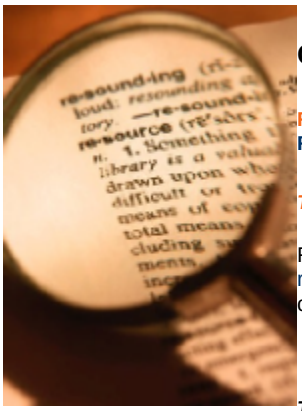
Ohai

Resources



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Resources



Overview

Resources are the basic units of work in Chef - discrete chunks of a system's configuration that are declared in Recipes and Definitions, and applied to your nodes.

They are usually a cross platform abstraction of the thing you're configuring on the host.

For example, packages may be installed via apt, yum, or the BSD ports and packages systems, but the package resource abstracts these differences away so you can specify that a package should be installed in a cross-platform way.

This page provides documentation of the Resources included in the Chef library.

Each description is broken in to several categories:

Category	Description
Actions	The list of actions for this resource
Attributes	The list of attributes for this resource
Providers	The list of providers for this resource, along with any shortcut resource name. (You can use the shortcut resource name instead of specifying the `provider` attribute - `apt_package` rather than `package` with `provider Chef::Provider::Package::Apt`.)
Examples	The first section, shows examples of using each action.

New custom resources can also be created using Chef's Lightweight Resources and Providers (LWRP) DSL. Opscode provides several cookbooks that define custom Lightweight Resources and Providers (LWRP). Opscode LWRP Resources details those customer LWRPs, grouped with the cookbook that provides them.

Meta

The attributes and actions in this section apply to all resources.

Common Actions

These actions are available in every resource.

Action	Description	Default
nothing	Do nothing - useful if you want to specify a resource, but only notify it of other actions.	Yes - in the absence of another default action, nothing is the default

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Examples

Action Nothing

```
service "memcached" do
  action :nothing
  supports :status => true, :start => true, :stop => true, :restart => true
end
```

Common Attributes

These attributes can be set on any resource.

Attribute	Description
ignore_failure	If `true`, we will continue running the recipe if this resource fails for any reason. (defaults to `false`)
provider	The class name of a provider to use for this resource.
retries	Number of times to catch exceptions and retry the resource (defaults to 0). Requires Chef >= 0.10.4.
retry_delay	Retry delay in seconds (defaults to 2). Requires Chef >= 0.10.4.
supports	A hash of options that hint providers as to the capabilities of this resource.

Examples:

ignore_failure

```
gem_package "syntax" do
  action :install
  ignore_failure true
end
```

provider

```
package "some_package" do
 provider Chef::Provider::Package::Rubygems
end
```

supports

```
service "apache" do
  supports :restart => true, :reload => true
  action :enable
end
```

Provider and Supports

```
service "some_service" do
  provider Chef::Provider::Service::Upstart
  supports :status => true, :restart => true, :reload => true
  action [ :enable, :start ]
end
```

The `supports` attribute is only utilized by a few providers in Chef. For example User and Service.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Conditional Execution

You can add additional guards around resources to prevent them from being run unless some other condition is met using `not_if` or `only_if`.
The most common use of these is to make the `execute` resource idempotent, but they can be used with any resource.

Conditional Execution Attributes

Attribute	Description
not_if	Do **not** apply *this* resource if the given string (shell command) or block (ruby code) results in return code 0 or is true.
only_if	Only apply *this* resource if the given string (shell command) or block (ruby code) results in return code 0 or is true.

`not_if` and `only_if` attributes can be passed as strings or blocks.

- Strings are executed as a shell command. The attribute is `true` if the command returns 0.
- Blocks are Ruby code. The attribute is `true` if the result is `true`.

The behavior of the resource depends on whether the attribute was true.

- `not_if {true}` - indicates the `action` should **never** be performed.
- `not_if {false}` - indicates the `action` should **always** be performed.
- `only_if {true}` - indicates the `action` should **always** be performed.
- `only_if {false}` - indicates the `action` should **never** be performed..

Optional arguments are available.

Argument	Description	Example
:user	Sets the user to run the command as	`not_if "grep adam /etc/passwd", :user => 'adam'`
:group	Sets the group to run the command as	`not_if "grep adam /etc/passwd", :group => 'adam'`
:environment	A Hash of environment variables to set	not_if "grep adam /etc/passwd", :environment => { 'HOME' => "/home/adam" }
:cwd	Sets the current working directory before running the command	`not_if "grep adam passwd", :cwd => '/etc'`
:timeout	Sets a timeout for the command	`not_if "sleep 10000", :timeout => 10`

Examples:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

not_if

```
# Any of these next three examples will create /tmp/somefile, but not if /etc/passwd exists.
# Execute a string
template "/tmp/somefile" do
  mode "0644"
  source "somefile.erb"
  not_if "test -f /etc/passwd"
end

# Execute a block
template "/tmp/somefile" do
  mode "0644"
  source "somefile.erb"
  not_if do
    File.exists?("/etc/passwd")
  end
end

# A block using curly braces
template "/tmp/somefile" do
  mode "0644"
  source "somefile.erb"
  not_if {File.exists?("/etc/passwd")}
end

# This next example will create /tmp/somefile but only if the node does not have the attribute
some_value
# A block using curly braces with node attributes
template "/tmp/somefile" do
  mode "0644"
  source "somefile.erb"
  not_if { node[:some_value] }
end
```

only_if

```
# Either of these next two examples will create /tmp/somefile only if /etc/passwd exists.
# Execute a string
template "/tmp/somefile" do
  mode "0644"
  source "somefile.erb"
  only_if "test -f /etc/passwd"
end

# Execute a block
template "/tmp/somefile" do
  mode "0644"
  source "somefile.erb"
  not_if do ! File.exists?("/etc/passwd") end
end

# This next example will create /tmp/somefile but only if the node has the attribute some_value
# A block using curly braces with node attributes
template "/tmp/somefile" do
  mode "0644"
  source "somefile.erb"
  only_if { node[:some_value] }
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Notifications

Attribute	Description
notifies	Notify another resource to take an action if this resource changes state for any reason.
subscribes	Take action on this resource if another resource changes state. Works similarly to **notifies**, but the direction of the relationship is reversed.

Syntax

New in 0.9.10
The syntax described here is new in 0.9.10 and supports notifying resources that are defined later in a recipe. See below for the older syntax (which is still valid in 0.9.10+).

Warning
Note that this syntax is not supported by `subscribes` as of 0.9.12. See Ticket CHEF-1994.

The basic form of a notification is:

```
resource "name" do
  notifies :action, "resource_type[resource_name]", :notification_timing
end
```

For example, if you need to restart the apache service when you modify a template that configures apache, you would write that like this:

```
template "/etc/www/configures-apache.conf" do
  notifies :restart, "service[apache]"
end

service "apache"
```

By default, notifications are `:delayed`, that is they are queued up as they are triggered, and then executed at the very end of the Chef run. If you want to run an action immediately, use `:immediately`:

```
template "/etc/nagios3/configures-nagios.conf" do
  # other parameters
  notifies :run, "execute[test-nagios-config]", :immediately
end

# this will fail and prevent us from restarting nagios if we broke the config
execute "test-nagios-config" do
  command "nagios3 --verify-config"
  action :nothing
end
```

You can also send notifications to multiple resources, just use multiple attributes. Multiple attributes will get sent to the notified resources in the order specified.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
template "/etc/netatalk/netatalk.conf" do
  notifies :restart, "service[afpd]", :immediately
  notifies :restart, "service[cnid]", :immediately
end

service "afpd"
service "cnid"
```

Examples

notifies

```
template "/tmp/somefile" do
  mode "0644"
  source "somefile.erb"
  notifies :reload, "service[apache]"
end

service "apache" do
  supports :restart => true, :reload => true
  action :enable
end
```

Syntax (0.9.8 and Previous)



Original Syntax
This is the original syntax. It works on all versions of Chef.



Warning
Note that when using the `resources` syntax, a resource must have already been defined before you can set a notification that points to it. This shortcoming is the reason the new syntax was introduced in Chef 0.9.10.

The basic form of a notification is like this:

```
resource "name" do
  notifies :action, resources(:resource_type => "resource_name"), :notification_timing
end
```

To restart apache when a template is updated, you would write something like this:

```
service "apache"

template "/etc/www/configures-apache.conf" do
  notifies :restart, resources(:service => "apache")
end
```

Valid syntax for the `resources()` list in `notifies` and `subscribes`:

- `resources(:resource => "name")`
- `resources("resource[name]")`

To notify / subscribe multiple resources with the same action:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- `resources(:resource => ["name1", "name2"])`
- `resources("resource[name1]", "resource[name2]")`
- `resources("resource[name1,name2]")`

Here, `resources` is a method in Chef. `resource` is a resource type as described on this page. `name/name1/name2` is the name of the resource. Examples below show both formats of specifying the arguments to the resources method, and either works.

Examples

notifies

```
service "apache" do
  supports :restart => true, :reload => true
  action :enable
end

template "/tmp/somefile" do
  mode "0644"
  source "somefile.erb"
  notifies :reload, resources(:service => "apache")
end
```

notifies multiple resources

```
template "/etc/chef/server.rb" do
  source "server.rb.erb"
  owner "root"
  group "root"
  mode "644"
  notifies :restart, resources( :service => [ "chef-solr", "chef-solr-indexer", "chef-server" ]),
:delayed
end
```

subscribes

```
template "/tmp/somefile" do
  mode "0644"
  source "somefile.erb"
end

service "apache" do
  supports :restart => true, :reload => true
  action :enable
  subscribes :reload, resources(:template => "/tmp/somefile"), :immediately
end
```

Relative Paths

These paths are available in every resource.

Path	Description
#{ENV['HOME']}	This will return the ~ path in Linux and Mac, or the `%HOMEPATH%` in Windows.

Examples

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Path Home

```
template "#{ENV['HOME']}/chef-getting-started.txt" do
  source "chef-getting-started.txt.erb"
  mode "0644"
end
```

Cookbook File

Cookbook File transfers files from the `files/` directory of a cookbook to a specified path on the host running chef-client or solo.

The file in the cookbook is selected according to file specificity, which allows you to use a different source file based on hostname, host platform (OS or distro, as appropriate), or specific platform version. Files under `COOKBOOK_NAME/files/default` will be used if a more specific version is not available.

return to top of page

Actions

Action	Description	Default
create	Create this file	Yes
create_if_missing	Create only if it doesn't exist yet	
delete	Delete this file	

Attributes

Attribute	Description	Name Attribute
backup	How many backups of this file to keep. Set to `false` if you want no backups.	5
group	The group owner of the file (string or id)	
mode	The octal mode of the file - e.g. `'0755'` default varies	
owner	The owner for the file	
path	Name attribute: The path to the file	name
source	The basename of the source file	The basename of the path (destination)
cookbook	The cookbook this file is stored in	The current cookbook

A note about mode: When specifying the mode, the value can be a quoted string, eg "644". For a numeric value (i.e., without quotes), it should be 5 digits, e.g., `00644` to ensure that Ruby can parse it correctly. For more detail, see Ticket CHEF-174.

Providers

Provider	Shortcut Resource	Default For Platforms
Chef::Provider::CookbookFile		All

Examples

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Transfer a file in this cookbook

```
cookbook_file "/tmp/testfile" do
  source "testfile" # this is the value that would be inferred from the path parameter
  mode "0644"
end
```

Cron

Manage a cron entry. Attributes for schedule will default to '*' if not provided so only specify the schedule attributes that are applicable to your cron entry.

Create will update an entry if it shares the same name. The delete action will delete an entry with a matching name. Both notes only apply on a per-user basis.

Requires that a package providing `crontab` program is installed (typically `cron`).

return to top of page

Actions

Action	Description	Default
create	Create the crontab entry	Yes
delete	Delete the crontab entry	

Attributes

Attribute	Description	Default Value
minute	The minute this entry should run (0 - 59)	*
hour	The hour this entry should run (0 - 23)	*
day	The day of month this entry should run (1 - 31)	*
month	The month this entry should run (1 - 12)	*
weekday	The weekday this entry should run (0 - 6) (Sunday=0)	*
command	The command to run	
user	The user to run command as. Note: If you **change** the crontab user then the original user will **still** have the same crontab running until you explicity delete that crontab	root
mailto	Set the MAILTO environment variable	
path	Set the PATH environment variable	
home	Set the HOME environment variable	
shell	Set the SHELL environment variable	

Providers

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Provider	Shortcut Resource	Default For Platforms
Chef::Provider::Cron		All

Examples

Run a program on the fifth hour of the day

```
cron "noop" do
  hour "5"
  minute "0"
  command "/bin/true"
end
```

Run an entry if a folder exists

```
cron "ganglia_tomcat_thread_max" do
  command "/usr/bin/gmetric -n 'tomcat threads max' -t uint32 -v `/usr/local/bin/tomcat-stat
--thread-max`"
  only_if do File.exist?("/home/jboss") end
end
```

Deploy

The `deploy` resource is quite detailed and has a separate document. Please see **Deploy Resource** for detail on its use.

return to top of page

Directory

Manage a directory.

return to top of page

Actions

Action	Description	Default
create	Create this directory	Yes
delete	Delete this directory	

Attributes

Attribute	Description	Default Value
group	The group owner of the directory (string or id)	
mode	The octal mode of the directory, e.g. `'0755'` default varies	

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

owner	The owner for the directory	
path	Name attribute: The path to the directory	name
recursive	When deleting the directory, delete it recursively. When creating the directory, create recursively (ie, mkdir -p). Note: owner/group/mode only applies to the leaf directory, regardless of the value of this attribute	false

Providers

Provider	Shortcut Resource	Default For Platforms
Chef::Provider::Directory		All

Examples

Create a directory

```
directory "/tmp/something" do
  owner "root"
  group "root"
  mode "0755"
  action :create
end
```

Create a group of directories

```
%w{dir1 dir2 dir3}.each do |dir|
   directory "/tmp/mydirs/#{dir}" do
      mode "0775"
      owner "root"
      group "root"
      action :create
      recursive true
   end
end
```

Remove a directory, and its contents

```
directory "/tmp/something" do
  recursive true
  action :delete
end
```

Env

Manages environment keys. Currently, this is **a windows-only resource**.

On Unix Systems, the best way to manipulate the environment is via Ruby's ENV variable; however, manipulating this variable will not have the same "permanent" effect as using the env resource on Windows.

return to top of page

Actions

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Action	Description	Default
create	Create new environment variable.	Yes
delete	Delete environment variable.	
modify	Modify an existing environment variable. This will append the given value to the existing value, delimited by **delim**.	

Attributes

Attribute	Description	Default Value
key_name	The name of the key to create, delete, or modify	`name`
value	The value to set **key_name** to.	
delim	The delimiter that separates multiple values for a single key.	

Providers

Provider	Shortcut Resource	Default For Platforms
Chef::Provider::Env::Windows		Windows

Examples

Set an environment variable

```
env "ComSpec" do
    value "C:\\Windows\\system32\\cmd.exe"
end
```

Erlang Call

Connects to a distributed erlang node and makes a call. See the documentation for erl_call in the OTP documentation for more information.

Like the Execute and Script resources, Erlang Call resources are not idempotent. Use the `not_if` and `only_if` meta parameters to guard the resource for idempotence.

Additionally, the erl_call command needs to be on the path for this resource to work properly.

return to top of page

Actions

Action	Description	Default
run	Run this erlang call	Yes
nothing	Do not run this command	

Attributes

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Attribute	Description	Default Value
code	The erlang code to execute on the distributed node	`q().`
cookie	The cookie of erlang node you connect to	
distributed	Starts a distributed erlang node if neccessary	false
name_type	Whether the node_name is an `sname` or `name`	`sname`
node_name	The node hostname to connect to	`chef@localhost`

Providers

Provider	Shortcut Resource	Default For Platforms
Chef::Provider::ErlCall		All

Examples

Run a command on an erlang node

```
erl_call "list names" do
  code "net_adm:names()."
  distributed true
  node_name "chef@latte"
end
```

Execute

Execute a command. Use Script resources to execute a script with a specific interpreter.

By their nature, Execute resources are not idempotent, as they are completely up to the user's imagination. Use the `not_if` or `only_if` meta parameters to guard the resource for idempotence.

return to top of page

Actions

Action	Description	Default
run	Run this command	Yes
nothing	Do not run this command	

Use `action :nothing` to set a command to only run if another resource notifies it.

Attributes

Attribute	Description	Default Value
command	Name attribute: The command to execute	name
creates	A file this command creates - if the file exists, the command will not be run.	nil
cwd	Current working directory to run the command from.	nil

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

environment	A hash of environment variables to set before running this command.	nil
group	A group name or group ID that we should change to before running this command.	nil
path	An array of paths to use when searching for the command. Note that these are **not** added to the command's environment $PATH.	nil, uses system path
returns	The return value of the command (may be an array of accepted values) - this resource raises an exception if the return value(s) do not match.	0
timeout	How many seconds to let the command run before timing it out.	3600
user	A user name or user ID that we should change to before running this command.	nil
umask	Umask for files created by the command	nil

Providers

Provider	Shortcut Resource	Default For Platforms
Chef::Provider::Execute		All

Examples

Execute a command, with environment variable

```
execute "slapadd" do
  command "slapadd < /tmp/something.ldif"
  creates "/var/lib/slapd/uid.bdb"
  action :run
  environment ({'HOME' => '/home/myhome'})
end
```

Execute a command only on notification

```
execute "slapadd" do
  command "slapadd < /tmp/something.ldif"
  creates "/var/lib/slapd/uid.bdb"
  action :nothing
end

template "/tmp/something.ldif" do
  source "something.ldif"
  notifies :run, resources(:execute => "slapadd")
end
```

File

Manage files and optionally set the content. See also CookbookFile, Template, and Remote File.

If you want to copy a file from a cookbook, you probably want to use CookbookFile instead. This resource is for working with files already on your node.

return to top of page

Actions

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Action	Description	Default
create	Create this file	Yes
create_if_missing	Create this file only if it does not exist. If it exists, do nothing.	
delete	Delete this file	
touch	Touch this file (update the mtime/atime)	

Attributes

Attribute	Description	Default
path	Name attribute: The path to the file	name
backup	How many backups of this file to keep. Set to `false` if you want no backups.	5
group	The group owner of the file (string or id)	
mode	The octal mode of the file - e.g. `'0755'` default varies	
owner	The owner for the file	
content	A string to write to the file. This will replace any previous content if set	nil (don't manage content)

A note about mode: When specifying the mode, the value can be a quoted string, eg "644". For a numeric value, it should be 5 digits, eg "00644" to ensure that Ruby can parse it correctly. For more detail, see Ticket CHEF-174.

Providers

Provider	Shortcut Resource	Default For Platforms
Chef::Provider::File		All

Examples

Create a file

```
file "/tmp/something" do
  owner "root"
  group "root"
  mode "0755"
  action :create
end
```

Remove a file

```
file "/tmp/something" do
  action :delete
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

File Modes

```
# These are both equivalent.
file "/tmp/something" do
  mode "644"
end

file "/tmp/something" do
  mode 00644
end
```

Git

Git is a provider for the SCM resource. This is closely linked with doing application code deployment, so it is documented along with the Deploy Resource.

return to top of page

Group

Manage a local group.

return to top of page

Actions

Action	Description	Default
create	Create the group.	Yes
remove	Remove the group.	
modify	Modify an existing group, raising an exception if the group does not exist.	
manage	Manage an existing group. This will take no action if the group does not exist.	

Attributes

Attribute	Description	Default Value
group_name	Name attribute: Name of the group.	name
gid	Group id.	nil
members	Include users in a group	nil
append	True=add these members to the group, false=these are the definitive group members	false

Providers

Provider	Shortcut Resource	Default for Platforms

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef::Provider::Group		
Chef::Provider::Group::Dscl		Mac OS X
Chef::Provider::Group::Gpasswd		All
Chef::Provider::Group::Pw		FreeBSD
Chef::Provider::Group::Usermod		Solaris
Chef::Provider::Group::Windows		Windows

Examples

Random user

```
group "admin" do
  gid 999
  members ['paco', 'vicente']
end
```

HTTP Request

Send an HTTP request with an arbitrary message. Handy for implementing your own callbacks!

return to top of page

Actions

Action	Description	Default
get	Send a GET request	Yes
put	Send a PUT request	
post	Send a POST request	
delete	Send a DELETE request	
head	Send a HEAD request	
options	Send an OPTIONS request	

Attributes

Attribute	Description	Default Value
url	The URL to send the request to	nil
message	Name attribute: The message to be sent to the URL (as the `message` parameter)	name
headers	Hash of custom headers	{}

Providers

Provider	Shortcut Resource	Default For Platforms

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef::Provider::HttpRequest		All

Examples

Send a GET request

```
# Sends "http://example.com/check_in?message=Send a GET request"
http_request "some message" do
  url "http://example.com/check_in"
end
```

Send a POST request, with JSON message body and basic authentication

```
http_request "posting data" do
  action :post
  url "http://example.com/check_in"
  message :some => "data"
  headers({"AUTHORIZATION" => "Basic #{Base64.encode64("username:password")}"})
end
```



JSON Body
The message is posted as application/data and not multipart/form-data or application/x-www-form-urlencoded

Ifconfig

You can manage interfaces with the `ifconfig` resource. This resource is still under development.

return to top of page

Actions

Action	Description	Default
add	Run `ifconfig` to configure the network interface and, on some platforms, write a configuration file for the network interface.	Yes
delete	Run `ifconfig` to bring the network interface down and, on some platforms delete the network interface's configuration file.	
enable	Run `ifconfig` to configure the network interface.	
disable	Run `ifconfig` to bring the network interface down.	

Attributes

Attribute	Description	Default Value
target	The address you would like to assign to the interface.	name
device	The network interface to be configured.	nil
hwaddr		nil

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

inet_addr		nil
bcast	Broadcast address. Note this is not set with ifconfig but is added to the interface's startup configuration file on select platforms.	nil
mask	The netmask.	nil
mtu	The maximum transmission unit(mtu) of the network interface.	nil
metric	The routing metric of the interface.	nil
onboot	Determines whether interface should be brought up on boot. Set to "yes" to bring interface up on boot.	nil
network	The network address.	nil
bootproto	Boot protocol, such as `"dhcp"`	nil
onparent	Determines whether the interface should be brought up when parent interface is brought up. Set to "yes" to bring up interface when parent interface brought up.	nil

Providers

Provider	Shortcut resource	Default For Platforms
Chef::Provider::Ifconfig		All

Examples

```
ifconfig "192.186.0.1" do
  device "eth0"
end
```

Notes

- Currently, the default provider only writes out a start-up configuration file for the interface on Red Hat-based platforms (it writes to `/etc/sysconfig/network-scripts/ifcfg-#{device_name}`).

Link

Create symbolic or hard links.

return to top of page

Actions

Action	Description	Default
create	Create a link	Yes
delete	Delete a link	

Attributes

Attribute	Description	Default Value
target_file	Name Attribute: The file name of the link	name

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

to	The real file you want to link to	
link_type	Either `:symbolic` or `:hard`.	`:symbolic`
owner	The owner of the symlink	
group	The group of the symlink	

Providers

Provider	Shortcut Resource	Default For Platforms
Chef::Provider::Link		All

Examples

Create a symbolic link

```
# ln -s /etc/passwd /tmp/passwd
link "/tmp/passwd" do
  to "/etc/passwd"
end
```

Create a hard link

```
# ln /etc/passwd /tmp/passwd
link "/tmp/passwd" do
  to "/etc/passwd"
  link_type :hard
end
```

Delete a link

```
link "/tmp/mylink" do
  action :delete
  only_if "test -L /tmp/mylink"
end
```

Log

Adds log entries from your recipes.

return to top of page

Actions

Action	Description	Default
write	Write to log	Yes

Attributes

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Attribute	Description	Default Value
level	:info, :warn, :debug, :error, or :fatal	:info

Providers

Provider	Shortcut Resource	Default For Platforms
Chef::Provider::Log::ChefLog	n/a	All

Examples

info log level

```
log "your string to log"
```

debug log level

```
log("a debug string") { level :debug }
```

Mdadm

Manage mdadm software RAID devices.

return to top of page

Actions

Action	Description	Default
create	Create a new array with per-device superblocks	Yes
assemble	Assemble a previously created array into an active array	
stop	Stop an active array	

Attributes

Attribute	Description	Default Value
raid_device	Name attribute:The RAID device name (/dev/md0)	name
devices	An array of devices to include in the RAID array	
level	The RAID level	1
chunk	The chunk size	16

Providers

Provider	Shortcut Resource	Default For Platforms
Chef::Provider::Mdadm		Linux

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Examples

Create and Assemble a RAID 1 Array from two disks with a 64k chunk size

```
mdadm "/dev/md0" do
  devices [ "/dev/sda", "/dev/sdb" ]
  level 1
  chunk 64
  action [ :create, :assemble ]
end
```

Mount

Manage a mounted filesystem.

return to top of page

Actions

Action	Description	Default
mount	Mount the device	Yes
umount	Unmount the device	
remount	Remount the device	
enable	Add entry to fstab	
disable	Remove entry from fstab	

When passing multiple actions order matters. For example, `mount` must happen before `enable`: `action [:mount, :enable]`

Attributes

Attribute	Description	Default Value
mount_point	Name attribute:The directory/path where the device should be mounted	name
device	The special block device or remote node, a label or an uuid to mount.	nil
device_type	The type of the device specified. Can be :device, :label or :uuid	:device
fstype	The filesystem type of the device.	nil
options	Array or string containing mount options.	defaults
dump	Dump frequency in days, for fstab entry.	0
pass	Pass number for fsck, for fstab entry.	2

For now, device is required for umount and remount in order to check the mount command output for presence. This requirement will be removed at a later date.

For now, fstype is required. This requirement will be removed at a later date.

If options is a string, it will be converted to an array.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Providers

Provider	Shortcut Resource	Default For Platforms
Chef::Provider::Mount		All, except Windows
Chef::Provider::Mount::Windows		Windows

Examples

Mount a local block device

```
mount "/mnt/local" do
  device "/dev/sdb1"
  fstype "ext3"
end
```

Mount a remote filesystem

```
mount "/export/www" do
  device "nas1prod:/export/web_sites"
  fstype "nfs"
  options "rw"
end
```

Mount a remote windows folder on local drive letter T:

```
mount "T:" do
  action :mount
  device "\\\\hostname.example.com\\folder"
end
```

un-mount remote windows D-drive attached as local drive letter T:

```
mount "T:" do
  action :umount
  device "\\\\hostname.example.com\\D$"
end
```

Mount a remote filesystem & add to fstab

```
mount "/export/www" do
  device "nas1prod:/export/web_sites"
  fstype "nfs"
  options "rw"
  action [:mount, :enable]
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Mount a labeled filesystem

```
mount "/mnt/volume1" do
  device "volume1"
  device_type :label
  fstype "xfs"
  options "rw"
end
```

Mount a non-block filesystem

```
mount "/mount/tmp" do
  pass     0
  fstype   "tmpfs"
  device   "/dev/null"
  options  "nr_inodes=999k,mode=755,size=500m"
  action   [:mount, :enable]
end
```

Ohai

Reload node's ohai configuration. This allows recipes that change system attributes (e.g. add a user) to refer to those attributes later in the recipe.

return to top of page

Actions

Action	Description	Default
reload	Reload the node's ohai configuration	Yes

Attributes

Attribute	Description	Default Value
plugin	Optionally restrict reload to a particular Ohai plugin. By default, all plugins are reloaded.	nil

Providers

Provider	Shortcut Resource	Default For Platforms	Options
Chef::Provider::Ohai		All	

Examples

Reload ohai

```
ohai "reload" do
  action :reload
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Reload ohai configuration after new user

```
ohai "reload_passwd" do
  action :nothing
  plugin "passwd"
end

user "daemonuser" do
  home "/dev/null"
  shell "/sbin/nologin"
  system true
  notifies :reload, resources(:ohai => "reload_passwd"), :immediately
end

ruby_block "just an example" do
  block do
    # These variables will now have the new values
    puts node[:etc][:passwd][:daemonuser][:uid]
    puts node[:etc][:passwd][:daemonuser][:gid]
  end
end
```

Package

Manage packages.

return to top of page

Actions

Action	Description	Default
install	Install a package - if version is provided, install that specific version	Yes
upgrade	Upgrade a package to the latest version	
remove	Remove a package	
purge	Purge a package (this usually entails removing configuration files as well as the package itself)	

Attributes

Attribute	Description	Default Value
package_name	Name attribute: The name of the package to install	name
version	The version of the package to install/upgrade	nil
response_file	An optional response file - used to pre-seed packages (note: the file is fetched by Cookbook File)	nil
source	Used to provide an optional package source for providers that use a local file (rubygems, dpkg, yum and rpm)	
options	Add additional options to the underlying package command	nil

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

gem_binary	A gem_package attribute to specify a gem binary. Useful for installing ruby 1.9 gems while running chef in ruby 1.8	nil (API)
arch	The architecture of the package to install/upgrade (yum_package only)	nil
flush_cache	Flush the internal YumCache before/after install/upgrade (yum_package only, chef >= 0.10.4)	{ :before => false, :after => false }
allow_downgrade	Allow yum to downgrade a package to satisfy a requested version (yum_package only, chef >= 0.10.4)	false

For providers that install from a local file (rubygems, dpkg, rpm), you must first grab the file via remote_file or cookbook_file.

Providers

Provider	Shortcut Resource	Default For Platforms	Options
Chef::Provider::Package::Apt	apt_package	Debian, Ubuntu	Yes
Chef::Provider::Package::Dpkg	dpkg_package		Yes
Chef::Provider::Package::EasyInstall	easy_install_package		
Chef::Provider::Package::Freebsd	freebsd_package	Freebsd	No
Chef::Provider::Package::Macports	macports_package	Mac OS X	No
Chef::Provider::Package::Portage	portage_package	Gentoo	Yes
Chef::Provider::Package::Rpm	rpm_package		Yes
Chef::Provider::Package::Rubygems	gem_package		Yes
Chef::Provider::Package::Yum	yum_package	Redhat, Centos	No
Chef::Provider::Package::Zypper	zypper_package	SuSE	No

The options attribute was added in 0.7.0.

Examples

Install Package

```
package "tar" do
  action :install
end
```

Install Specific Package Version

```
package "tar" do
  version "1.16.1-1"
  action :install
end
```

Install Package with options

```
package "debian-archive-keyring" do
  action :install
  options "--force-yes"
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Upgrade Package

```
package "tar" do
  action :upgrade
end
```

Remove Package

```
package "tar" do
  action :remove
end
```

Purge Package

```
package "tar" do
  action :purge
end
```

Install Package Using Specific Provider

```
package "tar" do
  action :install
  source "/tmp/tar-1.16.1-1.rpm"
  provider Chef::Provider::Package::Rpm
end
```

Install Yum package with specified architecture

```
yum_package "glibc-devel" do
  arch "i386"
end
```

Install a gem file from the local filesystem

```
gem_package "right_aws" do
  source "/tmp/right_aws-1.11.0.gem"
  action :install
end
```

Use of a response file is only supported on Debian and Ubuntu at this time. Providers need to be written to support use of a response_file. On Debian and Ubuntu, the file contains debconf answers to questions normally asked by the package manager on installation. Put the file in `files/default` of the cookbook where the package is specified and Chef will use the Cookbook File resource to retrieve it.

Install a package with a response_file

```
package "sun-java6-jdk" do
  response_file "java.seed"
end
```

Gem Package Options

As of Chef 0.9.0, the RubyGems package provider attempts to install gems using RubyGems' ruby api without spawning a new gem process, but falls back to spawning a gem command to install when necessary under the following conditions:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

1. If you specify a *gem_binary*, the provider will run that gem command to examine its environment settings, and again to install the gem.
2. If you specify install options as a String, the provider will spawn a gem command with those options when installing the gem.

Specifying Options With a Hash

If you're not using an explicit *gem_binary* parameter to the gem_package resource, it is preferable to provide install options as a Hash. This allows the provider to install the gem without needing to spawn an external gem process. The following options are available (options are passed to rubygems' DependencyInstaller):

Option	Description	Default Value
`:env_shebang`	whether or not the shebang line in executables uses `/usr/bin/env ruby`	false
`:force`		false
`:format_executable`	adds the ruby version to executables if your ruby interpreter has a trailing version number, i.e., `chef-client18`	false
`:ignore_dependencies`		false
`:prerelease`	installs prerelease gems	false
`:security_policy`	see http://docs.rubygems.org/read/chapter/21	nil
`:wrappers`	wraps executables so that you can specify which gem version to use with them	true

Installing a Gem with a Hash of Options

```
gem_package("bundler") do
  options(:prerelease => true, :format_executable => false)
end
```

Specifying Options With a String

When using an explicit *gem_binary*, you **must** pass in the options as a String. When not using an explicit *gem_binary*, you **may** still pass the options as a String, but this forces chef to spawn a `gem` process to install the gem, using more system resources. String options are passed verbatim to the gem command, so you specify them just as you would on the command line (i.e., `"--prerelease"` for prerelease gems).

Installing a Gem with an Options String

```
gem_package("nokogiri") do
  gem_binary("/opt/ree/bin/gem")
  options("--prerelease --no-format-executable")
end
```

PowerShell Script

Execute a script using the PowerShell interpreter.

Includes actions/attributes available to Execute resources: `creates`, `cwd`, `environment`, `group`, `path`, `timeout`, `user`. A temporary file is created and executed like other script resources, rather than run inline. This resource is used just like the regular Script resource (and providers for `bash`, `csh`, `perl`, `python` and `ruby`) with some small tweaks under the covers for the Windows platform and PowerShell interpreter.

By their nature, Script resources are not idempotent, as they are completely up to the user's imagination. Use the `not_if` or `only_if` meta parameters to guard the resource for idempotence.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

return to top of page

Actions

Action	Description	Default
run	Run the script	Yes

Attributes

Attribute	Description	Default Value
command	Name attribute: Name of the command to execute.	name
code	Quoted script of code to execute.	nil
interpreter	Script interpreter to use for `code` execution.	nil
flags	command line flags to pass to the `interpreter` when invoking	nil

Providers

Provider	Shortcut Resource	
Chef::Provider::PowershellScript		All

Examples

write out to an interpolated path

```
powershell "write-to-interpolated-path" do
  code <<-EOH
  $stream = [System.IO.StreamWriter] "#{Chef::Config[:file_cache_path]}/powershell-test.txt"
  $stream.WriteLine("In #{Chef::Config[:file_cache_path]}...word.")
  $stream.close()
  EOH
end
```

use the change working directory attribute

```
powershell "cwd-then-write" do
  cwd Chef::Config[:file_cache_path]
  code <<-EOH
  $stream = [System.IO.StreamWriter] "C:/powershell-test2.txt"
  $pwd = pwd
  $stream.WriteLine("This is the contents of: $pwd")
  $dirs = dir
  foreach ($dir in $dirs) {
    $stream.WriteLine($dir.fullname)
  }
  $stream.close()
  EOH
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

cwd to a winodws env variable

```
powershell "cwd-to-win-env-var" do
  cwd "%TEMP%"
  code <<-EOH
  $stream = [System.IO.StreamWriter] "./temp-write-from-chef.txt"
  $stream.WriteLine("chef on windows rox yo!")
  $stream.close()
  EOH
end
```

pass an env var to script

```
powershell "read-env-var" do
  cwd Chef::Config[:file_cache_path]
  environment ({'foo' => 'BAZ'})
  code <<-EOH
  $stream = [System.IO.StreamWriter] "./test-read-env-var.txt"
  $stream.WriteLine("FOO is $foo")
  $stream.close()
  EOH
end
```

Remote Directory

Recursively transfer a directory from the cookbook's files/default directory.

Remote directories obey file specificity so the directory you wish to copy should be located under `COOKBOOK_NAME/files/default/REMOTE_DIRECTORY` or a host- or distribution-specific path if desired.

Remote directory includes the attributes and actions of Directory. Name variable is still `path`.

return to top of page

Attributes

Attribute	Description	Default Value
cookbook	Specify the cookbook where the directory is located, default is current cookbook	nil
files_backup	Number of backup copies to keep for files in the directory	5
files_owner	The owner for files in the directory.	nil
files_group	The group for files in the directory.	nil
files_mode	The octal mode for files in the directory - `0755`.	0644
path	Name attribute: Path to the directory.	name
source	From the cookbook's `files/` directory.	
purge	Purge any extra files found in the target directory	false
overwrite	Overwrite files if different from the source	true

A note about mode: When specifying the mode, the value can be a quoted string, eg "644". For a numeric value, it should be 5 digits, eg "00644"

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

to ensure that Ruby can parse it correctly. For more detail, see Ticket CHEF-174.

Providers

Provider	Shortcut Resource	Default for Platforms
Chef::Provider::Directory::RemoteDirectory		All

Examples

Recursively transfer a directory from a remote location

```
# create up to 10 backups of the files, set the files owner different from the directory.
remote_directory "/tmp/remote_something" do
  source "something"
  files_backup 10
  files_owner "root"
  files_group "root"
  files_mode "0644"
  owner "nobody"
  group "nobody"
  mode "0755"
end
```

Remote File



Deprecated Behavior
In Chef 0.8.x and earlier, Remote File is also used to fetch files from the `files/` directory in a cookbook. This behavior is now provided by Cookbook File, and use of Remote File for this purpose is deprecated (though still valid) in Chef 0.9.0 and later.

Transfer a file from a remote `source`. Remote File includes the attributes and actions of File. Name variable is still `path`.

Read more about File Distribution to learn how Chef determines where to get the file

return to top of page

Actions

Action	Description	Default
create	Synchronize the file from the remote `source` to the filesystem	Yes
create_if_missing	Create the file locally by fetching from the remote source only if the file doesn't yet exist	

Attributes

Attribute	Description	Default Value
path	Name attribute: The path to the file	name
backup	How many backups of this file to keep. Set to `false` if you want no backups.	5
group	(optional) The group owner of the file (string or id)	

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

mode	(optional) The octal mode of the file - e.g. `'0755'` default varies	
owner	(optional) The owner for the file	
source	The source URL	the basename of the path attribute (see deprecated attributes)
checksum	(optional) the SHA-256 checksum of the file--if the local file matches the checksum, Chef will not download it	nil

Deprecated Attributes

The following attributes and attribute usage are relevant only to deprecated behavior that is now provided by cookbook file.

Attribute	Description	Default Value
cookbook	Specify the cookbook where the file is located, default is current cookbook	nil
source	The source file in `files/` directory	last path component of name

The source follows the same search path as Templates

Providers

Provider	Shortcut Resource	Default for Platforms
Chef::Provider::File::RemoteFile		All

Examples

Transfer a file from some URL

```
remote_file "/tmp/testfile" do
  source "http://www.example.com/tempfiles/testfile"
  mode "0644"
  checksum "08da002l" # A SHA256 (or portion thereof) of the file.
end
```

Transfer a file only if the remote source has changed (uses http_request resource)

```
remote_file "/tmp/couch.png" do
  source "http://couchdb.apache.org/img/sketch.png"
  action :nothing
end

http_request "HEAD #{http://couchdb.apache.org/img/sketch.png}" do
  message ""
  url http://couchdb.apache.org/img/sketch.png
  action :head
  if File.exists?("/tmp/couch.png")
    headers "If-Modified-Since" => File.mtime("/tmp/couch.png").httpdate
  end
  notifies :create, resources(:remote_file => "/tmp/couch.png"), :immediately
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Route

Manage the system routing table.

Currently Only works on Linux, and only outputs config files on redhat based distros

return to top of page

Actions

Action	Description	Default
add	Add a route	Yes
delete	Delete a route	

Attributes

Attribute	Description	Default Value
target	Name attribute: The address of the target route	name
netmask	The decimal representation of the network mask eg `255.255.255.0`	
gateway	The gateway for the route	
device	Network interface to apply this route to	

Providers

Provider	Shortcut Resource	Default For Platforms
Chef::Provider::Route		All

Examples

Add a host route

```
route "10.0.1.10/32" do
  gateway "10.0.0.20"
   device "eth1"
end
```

Delete a network route

```
route "10.1.1.0/24" do
  gateway "10.0.0.20"
   action :delete
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Ruby Block

The `ruby_block` resource can be used to execute a bit of Ruby code during a run. Ruby code in ruby_block resources is evaluated **with** other resources during convergence, whereas Ruby code outside ruby_block resources is evaluated **before** other resources, during recipe evaluation (compilation).

(See Evaluate and Run Resources at Compile Time for examples, and Anatomy of a Chef Run for more detail on code evaluation within a chef-client run.)

return to top of page

Actions

Action	Description	Default
create	Create the Ruby Block	Yes

Attributes

Attribute	Description	Default Value
block	A block of Ruby code	nil

Providers

Provider	Shortcut Resource	Default For Platforms
Chef::Provider::RubyBlock		All

Examples

Reread Chef client config in a run

```
ruby_block "reload_client_config" do
  block do
    Chef::Config.from_file("/etc/chef/client.rb")
  end
  action :create
end
```

SCM

Source code can be managed via the SCM resource. This is closely linked with doing application code deployment, so it is documented along with the Deploy Resource.

return to top of page

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Script

Execute a script using the specified interpreter. Includes all actions/attributes available to Execute resources. Predefined available script interpreters are `bash`, `csh`, `perl`, `python` and `ruby`.

The `ruby` script resource is different than the `ruby_block` resource described above, as it is created as a temporary file and executed like other script resources, rather than run inline.

By their nature, Script resources are not idempotent, as they are completely up to the user's imagination. Use the `not_if` or `only_if` meta parameters to guard the resource for idempotence.

return to top of page

Actions

Action	Description	Default
run	Run the script	Yes

Attributes

Attribute	Description	Default Value
command	Name attribute: Name of the command to execute.	name
code	Quoted script of code to execute.	nil
interpreter	Script interpreter to use for `code` execution.	nil
flags	command line flags to pass to the `interpreter` when invoking	nil

Providers

Providers for Chef::Resource::Script are aliases for different command interpreters.

Provider	Shortcut Resource
Chef::Resource::Script::Bash	bash
Chef::Resource::Script::Csh	csh
Chef::Resource::Script::Perl	perl
Chef::Resource::Script::Python	python
Chef::Resource::Script::Ruby	ruby

Examples

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Run a bash script

```
script "install_something" do
  interpreter "bash"
  user "root"
  cwd "/tmp"
  code <<-EOH
  wget http://www.example.com/tarball.tar.gz
  tar -zxf tarball.tar.gz
  cd tarball
  ./configure
  make
  make install
  EOH
end
```

Use bash provider/interpreter to run the same script

```
bash "install_something" do
  user "root"
  cwd "/tmp"
  code <<-EOH
  wget http://www.example.com/tarball.tar.gz
  tar -zxf tarball.tar.gz
  cd tarball
  ./configure
  make
  make install
  EOH
end
```

Service

Manage a service.

return to top of page

Actions

Action	Description	Default
enable	Enable this service	
disable	Disable this service	
nothing	Don't do anything with this service	Yes
start	Start this service	
stop	Stop this service	
restart	Restart this service	
reload	Reload the configuration for this service	

Use `action :start` to make sure the service is started and running. Use `action :enable` to enable it at boot. See examples for more.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Attributes

Attribute	Description	Default Value
service_name	Name attribute: Name of the service.	name
pattern	Pattern to look for in the process table.	service_name
start_command	Command used to start this service.	nil
stop_command	Command used to stop this service.	nil
status_command	Command used to check the service run status.	nil
restart_command	Command used to restart this service.	nil
reload_command	Command used to tell this service to reload its configuration.	nil
supports	Features this service supports, ie `:restart`, `:reload`, `:status`.	false

The supports parameter attribute controls how the Chef Provider will attempt to manage the service.

- :restart - the init script or other service provider can use a restart command. If this is not specified, Chef will attempt to stop then start the service.
- :reload - the init script or other service provider can use a reload command.
- :status - the init script or other service provider can use a status command to determine if the service is running. If this is not specified, Chef will attempt to match the service_name against the process table as a regular expression, unless `pattern` is specified as a parameter attribute.

Providers

By default, the init provider is used, which runs `/etc/init.d/service_name` with `_command`.

Provider	Shortcut Resource	Default for Platform
Chef::Provider::Service::Init		All
Chef::Provider::Service::Init::Debian		Debian, Ubuntu
Chef::Provider::Service::Upstart		
Chef::Provider::Service::Init::Freebsd		Freebsd
Chef::Provider::Service::Init::Gentoo		Gentoo
Chef::Provider::Service::Init::Redhat		Redhat, Centos
Chef::Provider::Service::Solaris		Solaris
Chef::Provider::Service::Windows		Windows

Examples

Starts the service if it is not running.

```
# typically this will run /etc/init.d/example_service start
service "example_service" do
  action :start
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Basic example, starts the service if it's not running and enables it to start at system boot time

```
# runs /etc/init.d/example_service (start|stop|restart), etc.
service "example_service" do
  supports :status => true, :restart => true, :reload => true
  action [ :enable, :start ]
end
```

Manage the ssh service, named depending on the node's platform

```
service "example_service" do
  case node[:platform]
  when "CentOS","RedHat","Fedora"
    service_name "redhat_name"
  else
    service_name "other_name"
  end
  supports :restart => true
  action [ :enable, :start ]
end
```

Change Provider for service depending on the node's platform

```
service "example_service" do
    case node[:platform]
    when "ubuntu"
      if node[:platform_version].to_f >= 9.10
        provider Chef::Provider::Service::Upstart
      end
    end
    action [:enable, :start]
  end
```

Process table has a different value than the name of the service script

```
service "samba" do
  pattern "smbd"
  action [:enable, :start]
end
```

Subversion

Subversion is a provider for the SCM resource. This is closely linked with doing application code deployment, so it is documented along with the Deploy Resource.

return to top of page

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Template

Manage File contents with an embedded ruby (erb) template. Includes actions and attributes from File. `path` is the Name attribute. Read more about templates.

Templates follow the same File Specificity rules as `remote_file` and `file` resources.

return to top of page

Actions

Action	Description	Default
create	Create the file	Yes
create_if_missing	Create only if it doesn't exist yet	

Attributes

Attribute	Description	Default Value
cookbook	Specify the cookbook where the template is located, default is current cookbook	nil
path	Name attribute: The path to the file	name
source	Template source file. Found in templates/default for the cookbook.	nil
group	The group owner of the file (string or id)	
mode	The octal mode of the file - e.g. `'0755'` default varies	
owner	The owner for the file	
variables	Variables to use in the template.	
local	Is the template already present on the node?	false

Source is found in the search path per the templates page

Providers

Provider	Shortcut Resource	Default for Platforms
Chef::Provider::File::Template		

Examples

Config file from a template

```
template "/tmp/config.conf" do
  source "config.conf.erb"
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Config file from a local template

```
template "/tmp/config.conf" do
  local true
  source "/tmp/config.conf.erb"
end
```

Config file from a template with variable map

```
template "/tmp/config.conf" do
  source "config.conf.erb"
  variables(
    :config_var => node[:configs][:config_var]
  )
end
```

User

Manage a local user.

return to top of page

Actions

Action	Description	Default
create	Create the user with given attributes. If the user exist, ensure that the resource is in the correct state. This implies **manage** .	Yes
remove	Remove the user.	
modify	Modify an existing user, raising an exception if the user does not exist.	
manage	Manage an existing user. This will take no action if the user does not exist.	
lock	Lock the user's password.	
unlock	Unlock the user's password.	

Attributes

Attribute	Description	Default Value
username	Name attribute: Name of the user.	name
comment	Gecos/Comment field.	nil
uid	Numeric user id.	nil
gid	Primary group id.	nil
home	Home directory location	nil
shell	Login shell.	nil
password	Shadow hash of password.	nil

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

system	Whether to create a system user.	nil
supports	User features supported, eg `:manage_home`.	{:manage_home => false, :non_unique => false }

- The **system** parameter can be used with useradd as the provider to create a "system" user which passes the `-r` flag to `useradd`.
- The **supports** attribute accepts a Mash where the keys represent features and the values are booleans indicating whether that particular feature is supported.
  Currently, there are two features of note:
  - `:manage_home`: Indicates whether the user's home directory will be created when the user is created.
    In the case of the Useradd provider, this indicates whether `-d` and `-m` will be passed to `useradd` (in the case of `:create`), or whether `-d` will be passed to `usermod` (in the case of `:manage` and `:modify`).
  - `:non_unique`: Indicates whether non-unique UIDs are allowed.
    Currently unused by the existing providers.
- The user resource does not include the -dm parameter on the usermod command when "supports :manage_home=>true". Chef-2706 is open to address this.

Looking to add your user's public SSH key to the authorized_keys file? **See the users cookbook**

Providers

Provider	Shortcut Resource	Default for Platforms
Chef::Provider::User		
Chef::Provider::User::Useradd		All
Chef::Provider::User::Pw		FreeBSD
Chef::Provider::User::Dscl		Mac OS X
Chef::Provider::User::Windows		Windows

Prerequisites

In order to use the "password" attribute in Chef 0.5.6, you must have "ruby-shadow" installed. You can get this by installing the debian package "libshadow-ruby1.8".

Password Shadow Hash

Different operating systems support different password encryption methods. MD5 is generally a supported fallback for the time being but newer methods such as SHA-512 should be used where available. Also, the tool you use to create the encrypted password may be different from one distribution to another.

To create a password shadow hash, used in the password attribute, run:

Create MD5 Password Shadow Hash

```
echo "theplaintextpassword" | makepasswd --clearfrom=- --crypt-md5 |awk '{ print $2 }'
```

Create MD5 Password Shadow Hash Alternate Method

```
openssl passwd -1 "theplaintextpassword"
```

Create SHA-512 Password Shadow Hash on Ubuntu 9.10+

```
sudo apt-get install mkpasswd
mkpasswd -m sha-512
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Examples

Random user

```
user "random" do
  comment "Random User"
  uid 1000
  gid "users"
  home "/home/random"
  shell "/bin/zsh"
  password "$1$JJsvHslV$szsCjVEroftprNn4JHtDi."
end
```

System User

```
user "systemguy" do
  comment "system guy"
  system true
  shell "/bin/false"
end
```

Opscode Cookbook LWRPs

Opscode provides several cookbooks that define custom Lightweight Resources and Providers (LWRP). They are documented on a separate page.

- Opscode LWRP Resources

return to top of page

Resources and Providers

Deploy Resource

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Deploy Resource



There are two resources for deployment, the Deploy Resource - with extensive attributes available to specify use, and the SCM Resources - which can be used within the Deploy Resource, or independently.

There are many approaches and strategies that can be undertaken to use these resources for managing and automating deployment within your infrastructure. **The discussion below goes over some of those approaches and provides some examples of use.**

SCM Resources

Source Control Management (SCM) Resources, which may be used within the Deploy resource, or independently.

Actions

"sync" is the default action.

Action	Description	Default
sync	Update the source to the specified revision, or get a new clone/checkout	
checkout	Clone or checkout the source. Does nothing if a checkout is available	
export	Export the source, excluding or removing any version control artifacts	
force_export	(Subversion only) Force an export of the source overwriting the existing copy if it exists, excluding or removing any version control artifacts	

Attributes

Attribute	Description	Default Value
destination	Name attribute Path to clone/checkout/export the source to.	
repository	URI of the repository	
revision	revision to checkout. can be symbolic, like "HEAD" or an SCM specific revision id	HEAD
reference	(Git only) alias for revision	
user	System user to own the checked out code	nil (user chef runs as)
group	System group to own the checked out code	nil(user chef runs as)
svn_username	(Subversion only) Username for Subversion operations	
svn_password	(Subversion only) Password for Subversion operations	
svn_arguments	(Subversion only) Extra arguments passed to the subversion command	
svn_info_args	(Subversion only) **svn_arguments** are not used when chef calls "svn info". If you need arguments to be passed to "svn info", put them in **svn_info_args**	
depth	(Git only) Number of past revisions to include in Git shallow clone	nil (full clone)
enable_submodules	(Git only) performs a *submodule init* and *submodule update*	false
remote	(Git only) remote repository to use for syncing an existing clone	origin

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

additional_remotes	(Git only) Array of additional remotes to add to the repositories configuration	
ssh_wrapper	(Git only) path to a wrapper script for running SSH with git. GIT_SSH environment variable is set to this.	



Doing Deployment from Your Git Repo?

Don't hit a dead end when you try to pull the code from your private git repo! If you do, *one of the following should help:*

1. Have a passphrase-less key.
You can strip your current key of the passphrase with:

```
ssh-keygen -p -P 'YOURPASSPHRASE' -N '' -f id_deploy
```

Run that from /srv/app_name, type your passphrase where 'YOURPASSPHRASE' appears, and the ssh-key will no longer require a passphrase.

or

2. Add the deployment private key to a Data Bag.

```
{
     "id": "my_app",
     ... truncated ...
     "deploy_key": "SSH private key for private git repository"
   }
```

The key needs to be a string with the newlines converted to "\n"'s

Providers

Provider	Shortcut Resource	Default For Platforms
Chef::Provider::Git	git	
Chef::Provider::Subversion	subversion	

SCM Examples

Did you need the absolute latest version of CouchDB?

grab_couchdb_from_svn

```
subversion "CouchDB Edge" do
  repository "http://svn.apache.org/repos/asf/couchdb/trunk"
  revision "HEAD"
  destination "/opt/mysources/couch"
  action :sync
end
```


This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Prefer the git mirror?

grab_couchdb_from_git_mirror

```
git "/opt/mysources/couch" do
  repository "git://git.apache.org/couchdb.git"
  reference "master"
  action :sync
end
```

Want to use different branches depending on the environment of the node?

using_different_branches_in_git

```
if node.chef_environment == "QA"
    branch_name = "staging"
else
    branch_name = "master"
end

git "/home/user/deployment" do
    repository "git@github.com:gitsite/deployment.git"
    revision branch_name
    action :sync
    user "user"
    group "test"
end
```

In the above example, the branch_name variable is being set to "staging" or "master" depending on the environment of the node. Once that is determined, it uses the branch_name variable to set the revision for the repository.

Keep in mind that if you use the command "git status" after running this recipe it will return the branch name as "deploy" regardless, as this is a default value. You should be able to see it checking out the correct branch when you run chef-client with debugging on:

```
sudo chef-client -l debug
```

Deploy Resource

A resource to help you manage and control your deployments. **Please note the number of attributes available for specifying behavior, and review the Discussion for strategies and examples of use.**

Actions

Action	Description	Default
deploy	Deploy the application	yes
force_deploy	For the revision deploy strategy, this removes any existing release of the same code version and re-deploys in its place	
rollback	Rollback the application to the previous release	

Attributes

Attribute	Description	Default Value
deploy_to	The "meta root" for your application.	

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

repository	URI of the repository	
repo	alias for *repository*	
revision	revision to checkout. can be symbolic, like "HEAD" or an SCM specific revision id	HEAD
branch	alias for *revision*	
user	System user to run the deploy as	nil (user chef runs as)
group	System group to run the deploy as	nil (group chef runs as)
svn_username	(Subversion only) Username for Subversion operations	
svn_password	(Subversion only) Password for Subversion operations	
svn_arguments	(Subversion only) Extra arguments passed to the subversion command	
shallow_clone	(Git only) boolean, true sets clone depth to 5	nil (full clone)
enable_submodules	(Git only) performs a *submodule init* and *submodule update*	false
remote	(Git only) remote repository to use for syncing an existing clone	origin
ssh_wrapper	(Git only) path to a wrapper script for running SSH with git. GIT_SSH environment variable is set to this.	
git_ssh_wrapper	alias for *ssh_wrapper*	
scm_provider	SCM Provider to use.	Chef::Provider::Git
repository_cache	Name of the subdirectory where the pristine copy of your app's source is kept	cached-copy
environment	A hash of the form {"ENV_VARIABLE"=>"VALUE"}	
purge_before_symlink	An array of paths, relative to app root, to be removed from a checkout before symlinking	%w{log tmp/pids public/system}
create_dirs_before_symlink	Directories to create before symlinking. Runs after purge_before_symlink	%w{tmp public config}
symlinks	A hash that maps files in the shared directory to their paths in the current release	{"system" => "public/system", "pids" => "tmp/pids", "log" => "log"}
symlink_before_migrate	A hash that maps files in the shared directory into the current release. Runs before migration	{"config/database.yml" => "config/database.yml"}
migrate	Should the migration command be executed? (*true* or *false*)	false
migration_command	A string containing a shell command to execute to run the migration	
restart_command	A code block to evaluate or a string containing a shell command	nil
rollback_on_error	Boolean value that controls whether to rollback on error. When true, the deploy resource will rollback to the previously deployed release if an error occurs when deploying the new release. (**Available in Chef 0.10.6+**)	false
before_migrate	A block or path to a file containing chef code to run before migrating	deploy/before_migrate.rb
before_symlink	A block or path to a file containing chef code to run before symlinking	deploy/before_symlink.rb
before_restart	A block or path to a file containing chef code to run before restarting	deploy/before_restart.rb
after_restart	A block or path to a file containing chef code to run after restarting	deploy/after_restart.rb

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Providers

Provider	Shortcut Resource	Default For Platforms
Chef::Provider::Deploy::TimstampedDeploy	timestamped_deploy	all
Chef::Provider::Deploy::Revision	deploy_revision, deploy_branch	

Discussion

The Deploy Strategies

The deploy resource is ported from Capistrano and works similarly.

In your deploy directory, you will need to create a subdirectory named *shared*, where your configuration and temporary files will be kept. For a rails application, you will typically have *config*, *log*, *pids*, and *system* directories within the shared directory to keep the files stored there independent of the code in your source repository.

In addition to these, the deploy process will create subdirectories named *releases* and *current* in your deploy directory. The *releases* directory holds up to the five most recently deployed versions of your application, and the *current* directory will be a link to the currently released version.

Deployment happens in four phases: checkout, migrate, symlink, and restart.

1. When the deploy is run, Chef uses the SCM resource to get the specified revision, placing the clone or checkout in a subdirectory of the deploy directory named *cached-copy*.
    - A copy of your application is then placed in a subdirectory under the releases directory.
2. Next, if a migration is to be run, Chef symlinks the database configuration file in to the checkout, by default *config/database.yml*, into the checkout and runs the migration command.
    - For a rails application, *migration_command* is usually set to *rake db:migrate*
3. After migration, the symlink proceeds by removing directories for shared and temporary files from the checkout, by default *log*, *tmp/pids*, and *public/system*.
    - After this step, any needed directories, by default *tmp*, *public*, and *config*, are created if they don't yet exist.
    - The symlink step completes by symlinking shared directories into the current release, *public/system*, *tmp/pids*, and *log* by default, and then symlinking the release directory to *current*.
4. Following the symlink step, the application is restarted according to the restart command set in the recipe.

Timestamped Deploy

The two available deploy providers mainly differ in how they name the directories that contain your releases. Although the difference might seem minor, it has a large effect on how the deployment behaves.

The "timestamped" deploy strategy is the default. It names release directories with a timestamp of the form "YYYYMMDDHHMMSS". For example, your release might be located at the path */my/deploy/dir/releases/20040815162342*.

The deploy provider determines whether or not to deploy your code based on the existence of the release dir it is attempting to deploy to.

- Because the timestamp will be different for every chef run, **the timestamped deploy provider is not idempotent**.
- If you choose to use this deployment strategy, *you will need to manually manage the action setting on the resource to prevent unintended continuous deployment.*

Deploy Revision

Although the "revision" deploy strategy is not the default for compatibility reasons, it is the recommended strategy.

With this strategy, the release subdirectory will be named after the revision id used by your SCM. For git users, this will be the familiar SHA checksum; For subversion users, it will be the integer revision number. *Note that even if you supply a branch name, tag, or other name instead of the revision id, Chef will lookup the revision id from your SCM and name this directory with the revision id.*

As noted above, the deploy providers determine whether or not to deploy based on whether or not the releases directory it is trying to deploy to exists. This means that deploy_revision will be completely idempotent if you specify an exact revision to deploy. You could also use this in more interesting ways, for example, if you have a "deploy" or "production" branch in git, you can set the provider to track this branch. This will cause your deployment to get updated the next time chef runs after pushing to that branch.

The deploy_revision resource results in deployed components under the destination location being owned by the user who runs the application. If this is an issue for your particular workflow, there are three approaches you can undertake:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- you'll need to incorporate changing permissions to their desired end state within your recipe, or
- add a before_restart block to fix up the permissions, or
- have an unprivileged user (e.g., "opscode") as the owner of the deploy directory, and another unprivileged user (e.g., "opscodeapp") run it. This is the approach we'd recommend.

The drawback to the revision deploy strategy is that if the deploy fails for any reason, and you want to re-deploy with the same code, you'll need to manually set the action to :force_deploy. Forcing a deploy will remove the old release directory and then proceed with the deployment as usual. *Be forewarned that this can cause downtime if you force a deployment over the current release.*

The deployed revisions are stored in (file_cache_path)/revision-deploys/(deploy_path)

Interacting with Your Infrastructure During Deployment

In between each step of the deployment process, you can run arbitrary ruby or chef code using callbacks. The available callbacks are:

- **before_migrate**
- **before_symlink**
- **before_restart**
- **after_restart**
- **restart_command**

All of the above callbacks support embedded recipes given in a block, but assume a shell command (instead of a deploy hook filename) when given a string.

Each of these callbacks can be used in one of three ways:

- You can pass a block:

callback_blocks

```
deploy_revision "/deploy/dir/" do
  # other attributes
  # ...

  before_migrate do
    # release_path is the path to the timestamp dir
    # for the current release
    current_release = release_path

    # Create a local variable for the node so we'll have access to
    # the attributes
    deploy_node = node

    # A local variable with the deploy resource.
    deploy_resource = new_resource

    python do
      cwd current_release
      user "myappuser"
      code<<-PYCODE
        # Woah, callbacks in python!
        # ...
        # current_release, deploy_node, and deploy_resource are all available
        # within the deploy hook now.
      PYCODE
    end
  end

end
```

- You can specify a file. Files are searched relative to the current release. The code in the file is evaluated exactly as if you had put that code in a block.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

callback_files

```
deploy "/deploy/dir/" do
  # ...

  before_migrate "callbacks/do_this_before_migrate.rb"
end
```

- If you don't provide a block or file, Chef will look for a file in the current checkout named *deploy/callback_name.rb*, such as *deploy/before_migrate.rb*. If it exists, it will be evaluated as above.

Within your callbacks, there are several ways to get access to information about the deployment.

- As shown in the example above, you can use *release_path* to get the path to the current release.
- You can access the deploy resource itself using the *new_resource* method, allowing you to access the environment variables you've set.
- If you've relied on access to an @configuration variable using the original chef-deploy gem, you can continue to use that as well, however, *new_resource* is the preferred way.

These are all available only at the "top level" within callbacks, so you need to assign any values you intend to use to local variables as shown in the callback_blocks example above.

Custom Application Layouts

By default, the deploy resource expects your application to be structured like a rails app, but you can customize the layout for any kind of application.

- **symlink_before_migrate**
  symlink_before_migrate is a hash mapping files in the shared directory to files in the current release. The primary use is to link required configuration files into place so they're available during the migration step. The default is
  `{"config/database.yml" => "config/database.yml"}`
  Note that if you try to do "symlink_before_migrate {}", the "{}" will be interpreted as a block rather than an empty hash. Instead, use "symlink_before_migrate({})" or "symlink_before_migrate nil".
- **purge_before_symlink**
  purge_before_symlink is an array of directories which will be removed before the symlink step. This removes directories before the production copy from the shared directory is symlinked in. Defaults to
  `["log", "tmp/pids", "public/system"]`
- **create_dirs_before_symlink**
  This list is used to create directories that need to exist in the release before the symlink step. To illustrate in the context of a rails app, you might use .gitignore to keep all tempfiles out of your repository and not have the tmp/ directory in your checked out code, but the symlink step will create a symlink from shared/pids to RELEASE/tmp/pids. Since this would fail if the RELEASE/tmp directory did not exist, the directory must be created for the deploy to proceed. Note that it is not necessary to create directories that will be symlinked in the symlink step, such as log/, as these don't rely on the existence of other directories in the release. Directories that already exist won't be overwritten. Defaults:
  `["tmp", "public", "config"]`
- **symlinks**
  symlinks is a hash mapping files and directories from their locations in the shared dir to locations in the release dir, used in the main symlinking step. Defaults to
  `{"system" => "public/system", "pids" => "tmp/pids", "log" => "log"}`

How to Nuke and Start Over

Chef keeps track of the order in which you've deployed each revision of your application in a cache file. *If you delete the deployed code from a box in order to force a redeployment, you need to delete this cache file as well.* Under the default configuration, this cache file is located in `/var/chef/cache/revision-deploys/APPNAME/`

Deploy Examples

Basic attribute usage

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

basic_usage

```
deploy "/my/deploy/dir" do
  repo "git@github.com/whoami/project"
  revision "abc123" # or "HEAD" or "TAG_for_1.0" or (subversion) "1234"
  user "deploy_ninja"
  enable_submodules true
  migrate true
  migration_command "rake db:migrate"
  environment "RAILS_ENV" => "production", "OTHER_ENV" => "foo"
  shallow_clone true
  action :deploy # or :rollback
  restart_command "touch tmp/restart.txt"
  git_ssh_wrapper "wrap-ssh4git.sh"
  scm_provider Chef::Provider::Git # is the default, for svn: Chef::Provider::Subversion
end
```

Any recipes using the git-deploy gem can continue using the same API

backwards_compatible_API

```
deploy "/srv/#{appname}" do
  repo "git://github.com/radiant/radiant.git"
  revision "HEAD"
  user "railsdev"
  enable_submodules false
  migrate true
  migration_command "rake db:migrate"
  # Giving a string for environment sets RAILS_ENV, MERB_ENV, RACK_ENV
  environment "production"
  shallow_clone true
  action :deploy
  restart_command "touch tmp/restart.txt"
end
```

The layout of the application matches a Rails app by default, but you can customize it to fit your needs

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

customizing_the_application_layout

```
deploy "/my/apps/dir/deploy" do
  # Use a local repo if you prefer
  repo "/path/to/gitrepo/typo/"
  environment "RAILS_ENV" => "production"
  revision "HEAD"
  action :deploy
  migration_command "rake db:migrate --trace"
  migrate true
  restart_command "touch tmp/restart.txt"
  create_dirs_before_symlink  %w{tmp public config deploy}

  # You can use this to customize if your app has extra configuration files
  # such as amqp.yml or app_config.yml
  symlink_before_migrate  "config/database.yml" => "config/database.yml"

  # If your app has extra files in the shared folder, specify them here
  symlinks  "system" => "public/system", "pids" => "tmp/pids", "log" => "log",
            "deploy/before_migrate.rb" => "deploy/before_migrate.rb",
            "deploy/before_symlink.rb" => "deploy/before_symlink.rb",
            "deploy/before_restart.rb" => "deploy/before_restart.rb",
            "deploy/after_restart.rb" => "deploy/after_restart.rb"
end
```

When deploying non-rails code and you don't want any symlinks to the shared folder, use parentheses or Hash.new to avoid an ambiguity in Ruby

How to Deploy Without Symlinking to Shared

```
deploy "/my/apps/dir/deploy" do
  # OTHER PARAMETERS
  # ...

  # THIS DOESN'T WORK, the {} gets parsed as a code block:
  symlinks {}
  # Instead, use parentheses:
  symlinks({})
  # Or create your hash by calling new instead of using a literal:
  symlinks Hash.new
end
```

Using resources from within your callbacks as blocks or within callback files distributed with your application's source code

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

using_embedded_recipes_for_callbacks

```
deploy "#{node[:tmpdir]}/deploy" do
  repo "#{node[:tmpdir]}/gitrepo/typo/"
  environment "RAILS_ENV" => "production"
  revision "HEAD"
  action :deploy
  migration_command "rake db:migrate --trace"
  migrate true

  # Callback awesomeness:
  before_migrate do
    current_release = release_path

    directory "#{current_release}/deploy" do
      mode "0755"
    end

    # creates a callback for before_symlink
    template "#{current_release}/deploy/before_symlink_callback.rb" do
      source "embedded_recipe_before_symlink.rb.erb"
      mode "0644"
    end

  end

  # This file can contain Chef recipe code, plain ruby also works
  before_symlink "deploy/before_symlink_callback.rb"

  restart do
    current_release = release_path
    file "#{release_path}/tmp/restart.txt" do
      mode "0644"
    end
  end

end
```

Open Issues

- Only One Level of Submodules
  If you have submodules that themselves depend on submodules, the second level of submodules won't be initialized/updated. See the comments below.
- Directories are Assumed to Exist (CHEF-630)
  You'll need to have most of the directories needed by deploy already created before the deploy runs. *This issue is addressed with version 0.10.6+.*

Resources

Yum Package Resource

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Yum Package Resource



Yum Package Resource

Manage packages.

In addition to supporting the typical package resource options for installation, upgrade and removal the Yum Package Resource has several additional attributes and hidden options that Enterprise Linux users may find useful.

This document is only valid for Chef >= 0.10.4

The most notable difference with the yum package provider is being able to resolve Provides data for packages much like yum on the command line. This enables a variety of interesting options for installing packages by minimum versions, virtual provides and library names. The only portion the provider doesn't support at the moment is installing via file names, eg: yum_package "/bin/sh", like yum can. The volume of data to parse for this is excessive.

Actions

Action	Description	Default
install	Install a package - if version is provided, install that specific version (see allow_downgrade)	Yes
upgrade	Upgrade a package to the latest available version	
remove	Remove a package - if version is provided, remove that specific version	

Attributes

Attribute	Description	Default Value	yum_package only
package_name	Name attribute: Either package name, package name + architecture or dependency name (Provides) to install	name	No
version	The version of the package to install/upgrade/remove. Must be a complete version-release	nil	No
source	Can be used to yum localinstall a file, yum will use repositories to satisfy remote dependencies		No
options	Add additional options to yum (eg: enable/disable repos, disable gpg checks)	nil	No
arch	The architecture of the package to install/upgrade (can also be passed in package name)	nil	Yes
flush_cache	Flush the internal YumCache before/after install/upgrade/remove	false	Yes
allow_downgrade	Allow yum to downgrade a package to satisfy a requested version	false	Yes

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Some attributes are unique to the Yum provider and require the use of the 'yum_package' resource instead of the 'package' resource.

Examples

These are in addition to the standard Resources - Package examples.

Install package

```
yum_package "netpbm" do
  action :install
end

# or with the default action
yum_package "netpbm"

# or via Provides data (at least 2x slower for intial install/upgrade to gather
# remote Provides data)
yum_package "libnetpbm.so.10"
```

Install a specific version

```
yum_package "netpbm" do
  # must be a complete version-release
  version "10.35.58-8.el5"
end

# or via Provides data with an exact version
yum_package "netpbm = 10.35.58-8.el5"

# or via Provides data with a minimum version
yum_package "netpbm >= 10.35.58-8.el5"

# or via Provides data with a partial minimum version
yum_package "netpbm >= 10"
```

Install a specific version, even if older than the installed one

```
# To see what packages are available for this, add showdupesfromrepos=1 to /etc/yum.conf
yum_package "tzdata" do
  version "2011b-1.el5"
  allow_downgrade true
end
```

Install a specific arch

```
yum_package "netpbm" do
  arch "i386"
end

# or via package name
yum_package "netpbm.x86_64"
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Install new Yum repositories from package

```
# This package contains /etc/yum.repos.d/ repository configs. Some of these repos have
# newer packages and possibly a higher priority than what's in the cache, so we dump it.
yum_package "foo-release" do
  action :install
  flush_cache [ :after ]
end

# This package will now be picked up immediately instead of first looking at Provides
# data for a match which is slower. Also you are sure to get the correct version based
# on the repository configs.
yum_package "only-in-new-repo" do
  action :install
end
```

Install new Yum repositories from file

```
cookbook_file "/etc/yum.repos.d/custom.repo" do
  source "custom"
  mode "0644"
end

# Ensure the cache is dumped and the new repository is used immediately to ensure we
# find the correct package to install.
yum_package "only-in-custom-repo" do
  action :install
  flush_cache [ :before ]
end

# Unfortunately this is only useful if cookbook_file and yum_package are called within
# the same recipe with the correct order - a pretty narrow use case.
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Install new Yum repositories from file - improved

```
# Thanks to gaffneyc @ https://gist.github.com/918711 this is much nicer general
# solution that doesn't rely on the provider at all: we have the install of the
# repository trigger a creation of the yum cache and force the internal Chef cache
# to reload as well.
execute "create-yum-cache" do
  command "yum -q makecache"
  action :nothing
end

ruby_block "reload-internal-yum-cache" do
  block do
    Chef::Provider::Package::Yum::YumCache.instance.reload
  end
  action :nothing
end

cookbook_file "/etc/yum.repos.d/custom.repo" do
  source "custom"
  mode "0644"
  notifies :run, resources(:execute => "create-yum-cache"), :immediately
  notifies :create, resources(:ruby_block => "reload-internal-yum-cache"), :immediately
end

# Deleting a repo is similar but we have yum scrub it's cache to avoid any issues
execute "clean-yum-cache" do
  command "yum clean all"
  action :nothing
end

file "/etc/yum.repos.d/bad.repo" do
  action :delete
  notifies :run, resources(:execute => "clean-yum-cache"), :immediately
  notifies :create, resources(:ruby_block => "reload-internal-yum-cache"), :immediately
end
```

 Deploy Resource

Providers 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Providers

Providers take a resource, compare that resource to the current state of the part of the system it is managing, and then takes the Action specified in the resource.

Providers allow Chef to support multiple platforms with a single Resource.



What providers are available?

The real question you should be asking is *"What providers are available for a given Resource?"* You can find that answer by looking at the Resources page, in the Providers section for any individual resource type.

How does Chef know which Provider to use?

Chef maps Providers to Platforms (and Platform Versions) via `Chef::Platform`. In this class you will find a class instance variable `@platforms`, which consists of a Hash of various Platforms and Versions. We look up the resource type in this hash according to your Nodes platform and platform_version Attributes (which are provided by Ohai).

How do Providers... Provide?

We did our best to keep adding new Providers to Chef as simple as possible.

For each resource:

1. We look up the proper provider
2. We build a new instance of that provider
3. We call `load_current_resource`, which uses the `@new_resource` as a "guide" to look up the current state of the resource under management.
4. We then call the `action_` method for the action provided.
5. Ensure we mark the resource as `updated` if any changes happened.

For example, given a resource such as:

"Sample Resource"

```
directory "/tmp/monkey" do
  owner "root"
  group "root"
  mode 0755
  action :create
end
```

We would:

1. Look up the provider for a directory resource, which happens to be Chef::Provider::Directory
2. Call `load_current_resource`, which creates a new `directory["/tmp/monkey"]` resource, based on the directories current state (which goes in `@current_resource`).
3. Call `action_create`, which is responsible making sure the directory exists, and that it's attributes are correct.
4. If we changed the directory in any way, we mark the resource as updated.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

That's it!

How do I write new Providers?

If you want to write your own providers for resources that don't exist in Chef, you can use Lightweight Resources and Providers (LWRP) in your cookbook(s).

If you want to write a new provider for Chef itself and contribute to the project, that's awesome! We will help you out as much as we can. Go read the instructions for how to contribute to an Opscode project, hop on the Chef IRC channel, and get started.

 Resources

Lightweight Resources and Providers (LWRP) 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Lightweight Resources and Providers (LWRP)

Overview



*In Chef, **Resources** represent a piece of system state and **Providers** are the underlying implementation which brings them into that state.*

For example, all database vendors support the abstract concept of database creation, but the underlying implementation is different for each.

While typical Resources and Providers are implemented in Chef's core using Ruby classes, implementing Lightweight Resources and Providers (LWRP) is quick and easy, requiring less Ruby knowledge than their heavier counterparts. (LWRP's also become Ruby classes, but this is done for you, behind the scenes).

This document covers the DSL for creating Resources and Providers. It is not meant to be an in-depth description of the implementation, but details are sprinkled throughout for the curious reader.

For the Light-weight Resources and Providers (LWRPs) in Opscode's public open source cookbooks, see Opscode LWRP Resources.

File Locations

Lightweight Resources and Providers are loaded from files in a Cookbook's "resources" and "providers" directories. Resource and Provider names combine the cookbook name and the file name with an underscore. The only exception to this convention are files named 'default.rb'. In this case, the Resource or Provider is named according to the cookbook name only.

Examples

Note: there is no default LWRP for the AWS cookbook; they are referenced only for illustrating how names correspond.

Filename	Resource or Provider Name, as used in the DSL	Generated Class
/cookbooks/aws/resources/default.rb	aws	Chef::Resource::Aws
/cookbooks/aws/resources/elastic_ip.rb	aws_elastic_ip	Chef::Resource::AwsElasticIp
/cookbooks/aws/providers/default.rb	aws	Chef::Provider::Aws
/cookbooks/aws/providers/elastic_ip.rb	aws_elastic_ip	Chef::Provider::AwsElasticIp

Understanding LWRP Development

Want a different way of looking at it? View presentation slides from a talk on Understanding LWRP Development.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Custom LWRPs From the Community

Chef-Dominodes - A Chef resource for mutual exclusion of blocks of recipe code. Useful for cross-cluster rolling restarts.

iis7 - A Resource/Provider to create and configure websites in IIS 7 / 7.5.

minitest - Opscode Chef cookbook with lightweight resources and providers to help you functionally test your services during convergence

Resources

A Resource can be thought of as an abstract interface. Each is defined by its attributes and their validation rules, as well as the names of the actions it can take.

Keyword: actions

Actions are specified using the "actions" keyword followed by a comma-separated list of names. For example, the line

```
actions :foo, :bar
```

specifies that the list of allowed actions for this resource should include foo and bar and ultimately corresponds to the implementing Provider's "action_foo" and "action_bar" methods.

Note: several "actions" declarations will append to, not overwrite, the list of allowed actions.

Keyword: attribute

Attributes are specified using the "attribute" keyword followed by the attribute's name and an optional set of validation rules. For example, the line

```
attribute :foo
```

creates an attribute named foo, accessible to the implementing Provider via the resource's "foo" method, with no validation; whereas, the line

```
attribute :bar, :kind_of => String
```

creates an attribute named bar (accessible to the implementing Provider via the resource's "bar" method) that enforces that values specified in recipes must be of type String.

Validation parameters

The full set of options that can be passed to the attribute keyword in order to validate a parameter set on a Resource in a Recipe is:

:default	Sets the default value for this parameter.
:kind_of	Ensure that the value is a kind_of?(Whatever). If passed an array, it will ensure that the value is one of those types.
:required	Raise an exception if this parameter is missing. Valid values are true or false, by default, options are not required.
:regex	Match the value of the parameter against a regular expression.
:equal_to	Match the value of the parameter with ==. An array means it can be equal to any of the values.
:name_attribute	Specifies that this is set to the name of the resource when used. Valid value is true or false.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

:callbacks	Takes a hash of Procs, which should return true if the argument is valid. The key will be inserted into the error message if the Proc does not return true: "Option #{key}'s value #{value} #{message}!"
:respond_to	Ensure that the value has a given method. Takes one method name or an array of method names.

Default Provider when invoking an LWR in a Recipe

If you omit the provider attribute when using an LWR in a recipe, Chef will look for an LWP of the same name in the same cookbook by default.

So, you can write:

"Example LWR usage without an explicit Provider"

```
aws_elastic_ip :my_ip do
  ip 1.2.3.4
end
```

instead of:

"Example LWR usage with an explicit Provider"

```
aws_elastic_ip :my_ip do
  ip 1.2.3.4
  provider :aws_elastic_ip # unnecessary, as this is the default
end
```

Example, creating a Lightweight Resource

In order to demonstrate, let's contrast the implementation of the existing (heavyweight) File Resource with a Lightweight Resource of the same functionality:

"Chef::Resource::File circa Chef 0.7.8"

```
require 'chef/resource'

class Chef
  class Resource
    class File < Chef::Resource

      def initialize(name, collection=nil, node=nil)
        super(name, collection, node)
        @resource_name = :file
        @path = name
        @backup = 5
        @action = "create"
        @allowed_actions.push(:create, :delete, :touch, :create_if_missing)
      end

      def backup(arg=nil)
        set_or_return(
          :backup,
          arg,
          :kind_of => [ Integer, FalseClass ]
        )
      end

      def checksum(arg=nil)
        set_or_return(
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
      :checksum,
      arg,
      :regex => /^[a-zA-Z0-9]{64}$/
    )
  end

  def group(arg=nil)
    set_or_return(
      :group,
      arg,
      :regex => [ /^([a-z]|[A-Z]|[0-9]|_|-)+$/, /^\d+$/ ]
    )
  end

  def mode(arg=nil)
    set_or_return(
      :mode,
      arg,
      :regex => /^0?\d{3,4}$/
    )
  end

  def owner(arg=nil)
    set_or_return(
      :owner,
      arg,
      :regex => [ /^([a-z]|[A-Z]|[0-9]|_|-)+$/, /^\d+$/ ]
    )
  end

  def path(arg=nil)
    set_or_return(
      :path,
      arg,
      :kind_of => String
    )
  end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
    end
  end
end
```

The above code is simple, traditional ruby--no magic at all. We are creating a number of getter/setter methods and validating that the inputs match some criteria (a regex, a string, true/false, etc.). The lightweight version looks like:

Lightweight File Resource

```
actions :create, :delete, :touch, :create_if_missing

attribute :backup,   :kind_of => [ Integer, FalseClass ]
attribute :group,    :regex => /^([a-z]|[A-Z]|[0-9]|_|-)+$/, /^\d+$/
attribute :mode,     :regex => /^0?\d{3,4}$/
attribute :owner,    :regex => [ /^([a-z]|[A-Z]|[0-9]|_|-)+$/, /^\d+$/ ]
attribute :path,     :kind_of => String
attribute :checksum, :regex => /^[a-zA-Z0-9]{64}$/
```

Hopefully this is simpler to write and understand.

Providers

A provider is the underlying implementation which brings the resource into the desired state.

Background

The chef-client runs in two stages

1. **The compilation phase**
   The client examines each Recipe in order and adds its Resources to the ResourceCollection
2. **The execution phase**
   The client iterates over the ResourceCollection, and performs the following:
   a. Based on the Resource's "provider" attribute, a new instance of the specified Provider is created (if the attribute is not set, one is selected based on the local platform).
      The originating Resource is stored in the new Provider as the `@new_resource` instance variable and is accessible when writing LWPs as `new_resource.name`, for example.
   b. The `load_current_resource` method is then called on the provider instance, giving it an opportunity to populate `@current_resource` with a new resource that represents the current state of the relevant part of the system.
   c. For each action specified in `@new_resource.actions`, the `action_` method that corresponds to each action is called (e.g. `action :create` will invoke the `action_create` method of the Provider instance.)

Keyword: action

Actions are defined using the "action" keyword. Attributes from the originating Resource are accessible through the `@new_resource` instance variable, or, more idiomatically, simply as `new_resource` (see example below). When updating the resource, actions should call `@new_resource.updated_by_last_action(true)` to ensure that notifications are delivered.

Default Action

The DSL doesn't provide a semantic for specifying a default action. If you would like to specify a particular action to be the default for the resource, create an initialize method in the resource's .rb file. For example, to have the `:create` action from the file resource above be the default (as it is in the Chef library itself):

```
def initialize(*args)
  super
  @action = :create
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

In-line Resources in Provider Actions

The Recipe DSL has been extended to Providers, meaning Resources can be constructed and executed in-line in the bodies of Provider actions (see example using the "execute" Resource below.)

Implementation

For the curious, when a Provider references new Resources in-line, they are inserted into the ResourceCollection in order of appearance after the currently-executing Resource. For example, if after phase 1, the ResourceCollection contains the Resources [A,B] and during phase 2, the action run on A's Provider references Resources C and D in-line, the ResourceCollection (and execution order) will end up as [A,C,D,B].

Example, creating a Lightweight Provider

Taking the database example, our Resource might be defined by:

/cookbooks/opscode/resources/database.rb

```
actions :create, :delete

attribute :name, :kind_of => String, :name_attribute => true
attribute :type, :kind_of => String
```

A mysql Provider might look like:

/cookbooks/opscode/providers/mysql.rb

```
action :create do
  execute "create database" do
    not_if "mysql -e 'show databases;' | grep #{new_resource.name}"
    command "mysqladmin create #{new_resource.name}"
  end
end

action :delete do
  execute "delete database" do
    only_if "mysql -e 'show databases;' | grep #{new_resource.name}"
    command "mysqladmin drop #{new_resource.name}"
  end
end
```

This would create a new Provider (Chef::Provider::OpscodeMysql) with a load_current_resource that does nothing, along with two methods, action_create and action_delete. When either of these methods is invoked, the corresponding block is executed, including properly resolving `@new_resource`.

Using our resource in a recipe:

using the database resource

```
opscode_database "monkeynews" do
  type "innodb"
  action :create
  provider "opscode_mysql"
end
```

Would create a database called monkeynews. It would also allow you to trivially switch out the database back-end.

Extending An Existing Provider

If you'd like to write a LWP that extends another provider class, you can accomplish that as a mixin, which you would then place in a library under

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

the library directory of the cookbook using that extended class.

Your LWRP would then be written to include that library in the provider implementation to get access to the extended core resource. For an example, see the Transmission Cookbook, which includes a `transmission_torrent_file` LWRP that allows you to download a file via the BitTorrent protocol. This `transmission_torrent_file` LWRP are an extension of the existing file and remote_file resources.

Further Reading

View slides from a talk on Understanding LWRP Development

 Providers

Opscode LWRP Resources 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Opscode LWRP Resources



*This page documents the Light-weight Resources and Providers (LWRPs) in Opscode's public open source **cookbooks**. These can be used within **Recipes** to increase automation of your environment. Each document section links off to the Opscode public open source cookbook discussed.*

For additional information on LWRPs, including the DSL for creating Resources and Providers, see Lightweight Resources and Providers (LWRP).

apt

apt_repository

Manage apt repositories by dropping off a "sources.list" in `/etc/apt/sources.list.d`. Only relevant on distributions that have APT installed, e.g. Debian and Ubuntu. The sources.list filename will be named after the name attribute of the resource (`repo_name`) - see the examples.

Actions

Action	Description
add	Add a new repository
remove	Delete a repository

Attributes

Attribute	Description	Default Value
repo_name	Name Attribute: Name of the repository	name
uri	URI for the repository	nil
distribution	Distribution shortname (e.g. "lucid")	
components	Array of components (e.g. ["main"])	[]
deb_src	Whether to add the deb-src source repository	false
keyserver	Keyserver to download the PGP key from	nil
key	if a key_server is provided, this is assumed to be the fingerprint, otherwise it is the URI to the PGP key for the repo	nil

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Understanding LWRP Development

Opscode team member Joshua Timberman has a slideshare presentation on Understanding LWRP Development, available for your review.

Examples

/etc/apt/sources.list.d/zenoss.list

```
apt_repository "zenoss" do
  uri "http://dev.zenoss.org/deb"
  components ["main","stable"]
  action :add
end
```

/etc/apt/sources.list.d/opscode.list (signed repository)

```
apt_repository "opscode" do
  uri "http://apt.opscode.com"
  distribution node['lsb']['codename'] # or "lucid" if lsb isn't installed :)
  components ["main"]
  key "2940ABA983EF826A"
  keyserver "pgpkeys.mit.edu"
  action :add
end
```

/etc/apt/sources.list.d/opscode.list (signed repository) v2

```
apt_repository "opscode" do
  uri "http://apt.opscode.com"
  distribution node['lsb']['codename'] # or "lucid" if lsb isn't installed :)
  components ["main"]
  key "http://apt.opscode.com/packages@opscode.com.gpg.key"
  action :add
end
```

/etc/apt/sources.list.d/hardy-rsyslog-ppa.list (signed repository)

```
apt_repository "hardy-rsyslog-ppa" do
  uri "http://ppa.launchpad.net/a.bono/rsyslog/ubuntu"
  distribution "hardy"
  components ["main"]
  keyserver "keyserver.ubuntu.com"
  key "C0061A4A"
  action :add
  notifies :run, "execute[apt-get update]", :immediately
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

/etc/apt/sources.list.d/cloudkick.list (signed repository)

```
apt_repository "cloudkick" do
  uri "http://packages.cloudkick.com/ubuntu"
  distribution node['lsb']['codename']
  components ["main"]
  key "http://packages.cloudkick.com/cloudkick.packages.key"
  action :add
end
```

Remove /etc/apt/sources.list.d/zenoss.list

```
apt_repository "zenoss" do
  action :remove
end
```

aws

The AWS cookbook by Opscode provides two LWRPs.

- aws_ebs_volume - manages EBS volumes
- aws_elastic_ip - manages elastic IPs

These LWRPs use the right_aws RubyGem to access the AWS EC2 API.

Both of these LWRPs require passing in the AWS authentication credentials, aws_access_key and aws_secret_access_key. Generally we recommend storing these in a data bag item.

knife data bag show aws main

```
{
  "id": "main",
  "aws_access_key_id": "YOUR AWS ACCESS KEY",
  "aws_secret_access_key": "YOUR AWS SECRET ACCESS KEY"
}
```

This can be loaded in a recipe by using:

Recipe to load a data bag item

```
aws = data_bag_item("aws", "main")
```

Then the elements of the data bag item can be accessed as the hash keys in the `aws` hash:

```
aws['aws_access_key_id']
aws['aws_secret_access_key']
```

The examples below will use these values.

aws_ebs_volume

This LWRP only handles manipulation of the EBS volumes, it does not create filesystems, mount points or other storage management tasks. You'll need to write recipes to do that.

Actions

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

The create action will only make the API call to create a new volume, which will get a random volume ID generated by EC2. It can be attached in the same resource by specifying the actions as an array. See the examples.

Action	Description
create	Creates a new EBS volume
attach	Attaches an existing volume to the node
detach	Detaches the specified volume
snapshot	Creates a snapshot of the volume
prune	Prunes older snapshots

Attributes

Attribute	Description	Default Value
aws_access_key	Amazon AWS Access Key (username)	
aws_secret_access_key	Amazon AWS Secret Access Key (password)	
size	Size of volume to create in gigabytes	
snapshot_id	Snapshot to build a new EBS volume	
availability_zone	EC2 availability zone	
device	Local block device to attach the volume	
volume_id	Use to specify a volume to attach	
timeout	Connection timeout to EC2 API	3 minutes
snapshots_to_keep	Used with action **:prune** to specify how many snapshots to maintain	2

Examples

Create a new volume and attach it to the node

```
aws_ebs_volume "database_volume" do
  aws_access_key aws['aws_access_key_id']
  aws_secret_access_key aws['aws_secret_access_key']
  size 50
  device "/dev/sdm"
  action [:create, :attach]
end
```

Create a new volume out of an existing volume's snapshot

```
aws_ebs_volume "db_ebs_volume_from_snapshot" do
  aws_access_key aws['aws_access_key_id']
  aws_secret_access_key aws['aws_secret_access_key']
  size 50
  device "/dev/sdi"
  snapshot_id "snap-ABCDEFGH" # creates a new volume based on this snapshot
  action [ :create, :attach ]
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Prune all but 1 snapshot for the specified volume

```
aws_ebs_volume "prune_snapshots" do
  aws_access_key aws['aws_access_key_id']
  aws_secret_access_key aws['aws_secret_access_key']
  volume_id "vol-ABCDEF12" # must specify the volume ID to prune
  snapshots_to_keep 1
  action :prune
end
```

aws_elastic_ip

This LWRP can only associate or disassociate existing elastic IPs. You'll need to allocate new IPs in your AWS account manually.

Actions

Note that when an elastic IP address is associated with the node, network connectivity will be lost.

Action	Description
associate	Associate the specified IP with the node
disassociate	Disassociate the specified IP from the node

Attributes

Attribute	Description	Default Value
aws_access_key	Amazon AWS Access Key (username)	
aws_secret_access_key	Amazon AWS Secret Access Key (password)	
ip	The allocated IP address to use	
timeout	Connection timeout to EC2 API	3 minutes

Examples

The IP below is not valid in Amazon EC2, and is used just as an example.

Assocating a new IP address

```
aws_elastic_ip "eip_load_balancer_production" do
  aws_access_key aws['aws_access_key_id']
  aws_secret_access_key aws['aws_secret_access_key']
  ip "192.168.1.42"
  action :associate
end
```

Disassocating a new IP address

```
aws_elastic_ip "eip_load_balancer_production" do
  aws_access_key aws['aws_access_key_id']
  aws_secret_access_key aws['aws_secret_access_key']
  ip "192.168.1.42"
  action :disassociate
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

bluepill

Bluepill is a process management tool written in Ruby. More information about bluepill can be found on its project page.

bluepill_service

The bluepill_service LWRP can be used with the normal Chef service resource by using the `provider` parameter attribute.

Actions

Action	Description
start	Starts the service
stop	Stops the service
enable	Enable the service by linking the pill
disable	Disable the service by removing the pill link
load	Loads the service's pill into bluepill
restart	Restarts the service

Attributes

Attribute	Description	Default Value
service_name	Name Attribute: Name of the service	
variables	A hash of variables to pass into the pill template	
supports	Meta-parameter supports can be used like with the service resource	{ :restart => true, :status => true }

Examples

The recipe using the bluepill_service LWRP must contain a template resource for the pill and it must be named `NAME.pill.erb` where `NAME` is the service_name or name attribute, above.

template resource in the example recipe

```
template "/etc/bluepill/my_app" do
  source "my_app.pill.erb"
end
```

my_app.pill.erb

```
Bluepill.application("my_app") do |app|
  app.process("my_app") do |process|
    process.pid_file = "/var/run/my_app.pid"
    process.start_command = "/usr/bin/my_app"
  end
end
```

Actual bluepill_service resources:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Using the service resource

```
service "my_app" do
  provider bluepill_service
  action [:enable, :load, :start]
end
```

Using the bluepill_service resource directly

```
bluepill_service "my_app" do
  action [:enable, :load, :start]
end
```

chef_handler

The chef_handler LWRP allows enabling of Chef report and exception handlers from within recipe code (as opposed to hard coding in the client.rb file). This is useful for cookbook authors who may want to ship a product specific handler.

chef_handler

Requires, configures and enables handlers on the node for the current Chef run. Also has the ability to pass arguments to the handlers initializer. This allows initialization data to be pulled from a node's attribute data.

It is best to declare `chef_handler` resources early on in the compile phase so they are available to fire for any exceptions during the Chef run. If you have a base role you would want any recipes that register Chef handlers to come first in the run_list.

Actions

Action	Description
enable	Enables the Chef handler for the current Chef run on the current node
disable	Disables the Chef handler for the current Chef run on the current node

Attributes

Attribute	Description	Default Value
class_name	Name Attribute: The name of the handler class (can be module name-spaced).	
source	full path to the handler file. can also be a gem path if the handler ships as part of a Ruby gem.	
arguments	an array of arguments to pass the handler's class initializer	
supports	type of Chef Handler to register as, ie :report, :exception or both	{ :report => true, :exception => true }

Examples

register the Chef::Handler::JsonFile handler that ships as part of the Chef gem

```
chef_handler "Chef::Handler::JsonFile" do
  source "chef/handler/json_file"
  arguments :path => '/var/chef/reports'
  action :enable
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

enable handler during the compile phase

```
chef_handler "Chef::Handler::JsonFile" do
  source "chef/handler/json_file"
  arguments :path => '/var/chef/reports'
  action :nothing
end.run_action(:enable)
```

handle exceptions only

```
chef_handler "Chef::Handler::JsonFile" do
  source "chef/handler/json_file"
  arguments :path => '/var/chef/reports'
  supports :exception => true
  action :enable
end
```

enable the CloudkickHandler which was dropped off in the default handler path. Also passes the oauth key/secret to the handler's initializer

```
chef_handler "CloudkickHandler" do
  source "#{node['chef_handler']['handler_path']}/cloudkick_handler.rb"
  arguments [node['cloudkick']['oauth_key'], node['cloudkick']['oauth_secret']]
  action :enable
end
```

daemontools

daemontools is a set of utilities for managing Unix services written by Dan J Bernstein. (More information)

daemontools_service

This LWRP focuses on managing services using the svc command.

Actions

Almost the actions available correspond to command-line parameters to the svc command, which is itself idempotent.

Action	Description
start	Starts the service with `svc -u`. If the service stops, restart it
stop	Sends the STOP signal to stop the service with `svc -p`
restart	Restarts the service via TERM signal with `svc -t`
up	Starts the service with `svc -u`. If the service stops, restart it
once	Starts the service with `svc -o`. Does not restart if it stops
pause	Sends the STOP signal to pause the service with `svc -p`
cont	Sends the CONT signal with `svc -c`
hup	Sends the HUP signal with `svc -h`
alrm	Sends the ALRM signal with `svc -a`

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

int	Sends the INT signal with `svc -i`
term	Sends the TERM signal with `svc -t`
kill	Sends the KILL signal with `svc -k`
enable	Enables the service by setting up the service directory and link
disable	Disables the service by removing the link to the service directory

Attributes

Attribute	Description	Default Value
service_name	Name Attribute: Name of the service	
directory	Directory where the service should be set up (required)	
template	Name given to the template files	name
cookbook	Cookbook where the templates are located if not the current cookbook	
variables	Hash of variables to pass to the templates	{}
owner	Owner of the service directory and scripts	
group	Group of the service directory and scripts	
finish	Whether the service has a finish script	nil
log	Whether the service has a custom logging script	nil
env	If the service requires a special environment set up	{}

The run script is set up by a template named `sv-NAME-run.erb`, where NAME is the service name. Similarly, the log/run is `sv-NAME-log-run.erb` and the finish script is `sv-NAME-finish.erb`. These templates should be placed in the templates directory of the cookbook where this LWRP is used.

The env attribute above is used to create an env directory for the service containing files named for each environment variable required, with contents of its value in the hash. For more information on how daemontools uses env with the envdir program:

- http://cr.yp.to/daemontools/envdir.html

Examples

The following example will set up /etc/sv/chef-client directory, /etc/sv/chef-client/run script from the specified template, and the /etc/sv/chef-client/log/run script.

chef-client service

```
daemontools_service "chef-client" do
  directory "/etc/sv/chef-client"
  template "chef-client"
  action [:enable,:start]
  log true
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

sv-chef-client-run.erb

```
#!/bin/sh
PATH=/usr/local/bin:/usr/local/sbin:/bin:/sbin:/usr/bin:/usr/sbin
exec 2>&1
exec /usr/bin/env chef-client -i <%= node["chef_client"]["interval"] %> -s <%=
node["chef_client"]["splay"] %>
```

sv-chef-client-log-run.erb

```
#!/bin/sh
exec svlogd -tt ./main
```

djbdns

djbdns is a set of DNS daemons and tools written by Dan J Bernstein. (More information)

djbdns_rr

The djbdns_rr LWRP creates new DNS resource records for tinydns servers (usually internal) using the tinydns-edit command.

Actions

Action	Description
add	Adds a new resource record to tinydns

Attributes

Attribute	Description	Default Value
fqdn	Name Attribute: Fully qualified domain name	
ip	IP Address (required)	
type	Type of record, available: ns, childns, host, alias, mx	host
cwd	Current working directory where the tinydns data file lives	

Examples

In the example, the `node:djbdns:tinydns_internal_dir` is the djbdns cookbook attribute for the default location where the internal tinydns service is configured.

```
djbdns_rr "www.example.com" do
  cwd "#{node[:djbdns][:tinydns_internal_dir]}/root"
  ip "192.168.1.10"
  type "host"
  action :add
end
```

dmg

dmg_package

Package provider for installing applications from Mac OS X DMG disk images.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

This resource will install a DMG "Package". It will retrieve the DMG from a remote URL, mount it using OS X's `hdid`, copy the application (.app directory) to the specified destination (/Applications), and detach the image using `hdiutil`. The dmg file will be stored in the `Chef::Config[:file_cache_path]`. If you want to install an application that has already been downloaded (not using the `source` parameter), copy it to the appropriate location. You can find out what directory this is with the following command on the node to run chef:

```
knife exec -E 'p Chef::Config[:file_cache_path]' -c /etc/chef/client.rb
```

Actions

Action	Description
install	Installs the application (default)

The install action is the only one that makes sense, as uninstalling is as easy as dragging the application to the trash can. It is beyond the scope of this LWRP to do full package management for Mac OS X applications, as they have different installed artifacts.

Attributes

Attribute	Description	Default Value
app	Name Attribute: Name of the application used for the /Volumes directory and .app directory copied to /Applications.	name
source	Remote URL for the DMG to download, if specified.	nil
destination	Directory to copy the .app into.	/Applications
checksum	SHA256 checksum of the DMG to download.	nil
volumes_dir	Directory under /Volumes where the dmg is mounted.	app
dmg_name	Name of he DMG if it is not the same as `app` or if the name has spaces.	nil

Examples

The following examples all use the SHA256 checksum that was current when the application was originally installed, and may not be current for the latest releases.

Install `/Applications/Tunnelblick.app` from the primary download site.

```
dmg_package "Tunnelblick" do
      source "http://tunnelblick.googlecode.com/files/Tunnelblick_3.1.2.dmg"
      checksum "a3fae60b6833175f32df20c90cd3a3603a"
      action :install
    end
```

Install Google Chrome. Uses the `dmg_name` because the application name has spaces. Installs in `/Applications/Google Chrome.app`.

```
dmg_package "Google Chrome" do
      dmg_name "googlechrome"
      source "https://dl-ssl.google.com/chrome/mac/stable/GGRM/googlechrome.dmg"
      checksum "7daa2dc5c46d9bfb14f1d7ff4b33884325e5e63e694810adc58f14795165c91a"
      action :install
    end
```

Install Dropbox. Uses `volumes_dir` because the mounted directory is different than the name of the application directory. Installs in `/Applications/Dropbox.app`.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
dmg_package "Dropbox" do
      volumes_dir "Dropbox Installer"
      source "http://www.dropbox.com/download?plat=mac"
      checksum "b4ea620ca22b0517b75753283ceb82326aca8bc3c86212fbf725de6446a96a13"
      action :install
    end
```

Install MacIrssi to `~/Applications` from the local file downloaded to the cache path into an Applications directory in the current user's home directory.

Chef should run as a non-root user for this.

```
directory "#{ENV['HOME']}/Applications"

    dmg_package "MacIrssi" do
      destination "#{ENV['HOME']}/Applications"
      action :install
    end
```

dynect

dynect_rr

Dyn, Inc's dynect service has a REST API, and this LWRP can be used to create resource records through the API using Adam Jacob's dynect_rest rubygem.

Actions

Action	Description
delete	Delete the record
create	Create a new record
update	Update an existing record
replace	Replace an existing record

Attributes

Attribute	Description	Default Value
record_type	Type of record (A, CNAME, etc)	
rdata	Record data, see the Dyn API documentation	
ttl	Record time to live	
fqdn	Fully qualified domain name	
username	Dynect username	
customer	Dynect customer ID	
password	Dynect password	
zone	DNS zone	

Examples

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

A record, with authentication credentials as node attributes

```
dynect_rr "webprod" do
  record_type "A"
  rdata({"address" => "10.1.1.10"})
  fqdn "webprod.example.com"
  customer node[:dynect][:customer]
  username node[:dynect][:username]
  password node[:dynect][:password]
  zone     node[:dynect][:zone]
end
```

firewall

The firewall cookbook currently only has a 'ufw' provider, but additional implementations are under development.

firewall

The firewall resource handles the state of the firewall itself (enabled, disabled, reset).

Actions

Action	Description
enable	enable the firewall. this will make any rules that have been defined 'active'.
disable	disable the firewall. drop any rules and put the node in an unprotected state.
reset	reset the firewall. drop any rules and puts the node in the default state. Does not enable or disable the firewall.

Attributes

Attribute	Description	Default Value
name	arbitrary name to uniquely identify this resource	

Examples

enable platform default firewall

```
firewall "ufw" do
  action :enable
end
```

firewall_rule

Add new rules to the firewall.

Actions

Action	Description
allow	the rule should allow incoming traffic.
deny	the rule should deny incoming traffic.
reject	the rule should reject incoming traffic.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Attributes

Attribute	Description	Default Value
name	name attribute. arbitrary name to uniquely identify this firewall rule.	
protocol	valid values are: :udp, :tcp. defaults to all protocols.	
port	port number.	
source	ip address or subnet incoming traffic originates from. default is `0.0.0.0/0` (ie Anywhere).	
destination	ip address or subnet traffic routing to.	
position	position to insert rule at. if not provided rule is inserted at the end of the rule list.	

Examples

open standard ssh port, enable firewall

```
firewall_rule "ssh" do
  port 22
  action :allow
  notifies :enable, "firewall[ufw]"
end
```

open standard http port to tcp traffic only; insert as first rule; enable firewall

```
firewall_rule "http" do
  port 80
  protocol :tcp
  position 1
  action :allow
  notifies :enable, "firewall[ufw]"
end
```

mysql

mysql_database

The mysql cookbook includes a LWRP for interacting with a MySQL database via the Ruby library provided by the mysql gem. The various actions available correspond to queries sent to the database.

Actions

The flush_tables_with_read_lock and unflush_tables actions are used when performing replication. The create_db action is used to create databases. The query action is used to query databases.

Action	Description
flush_tables_with_read_lock	Flush tables with read lock
unflush_tables	Unflush tables
create_db	Create the named database
query	Query the database

Attributes

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Attribute	Description	Default Value
host	Host of the database server	
username	Username to connect as	
password	Password of the specified user	
database	Name of the database to create (only with create_db action)	

Examples

flush tables with read lock

```
mysql_database "locking tables for ebs snapshot" do
  action :flush_tables_with_read_lock
  host "localhost"
  username "root"
  password "SuperSecret"
end
```

unlock tables

```
mysql_database "unlocking tables after ebs snapshot" do
  action :unflush_tables
  host "localhost"
  username "root"
  password "SuperSecret"
end
```

create an application database

```
mysql_database "create my_app database" do
  host "localhost"
  username "root"
  password "SuperSecret"
  database "my_app_production"
  action [:create_db]
end
```

query a database

```
mysql_database "flush the privileges" do
  action :query
  host "localhost"
  username "root"
  password "SuperSecret"
  sql "flush privileges"
end
```

pacman

The pacman cookbook provides LWRPs for working with the pacman packaging system on ArchLinux.

pacman_aur

Builds and installs packages from ArchLinux User Repository.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Through a custom pkgbuild source file, a package can be built without hitting AUR at all.

Actions

Action	Description
build	Builds the package with the PKGBUILD data
install	Installs the package built with the build action

Attributes

Attribute	Description	Default Value
package_name	Name Attribute: Name of the package	
version	Version of the package	
builddir	Directory to download and build the package in	`Chef::Config:file_cache_path/builds`
options	Options appended to configure command	
pkgbuild_src	Whether to use a custom PKGBUILD file	false
patches	Array of files to use as patches to the package	[]

If pkgbuild_src is true, then the pacman_aur provider will look for a cookbook_file named `PKGBUILD` to use as package instructions.

Examples

Simple AUR package

```
pacman_aur "djbdns" do
  action [:build, :install]
end
```

Custom package with PKGBUILD

```
pacman_aur "netatalk" do
  pkgbuild_src "PKGBUILD"
  action [:build,:install]
end
```

pacman_group

Manages a pacman package group.

Actions

Action	Description
install	Installs the group
remove	Removes the group

Attributes

Attribute	Description	Default Value

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

package_name	Name Attribute: Name of the package group	
options	Additional options passed to the pacman command	

Examples

```
pacman_group "base-devel"
```

php

php_pear

PEAR is a framework and distribution system for reusable PHP components. PECL is a repository for PHP Extensions. PECL contains C extensions for compiling into PHP. As C programs, PECL extensions run more efficiently than PEAR packages. PEARs and PECLs use the same packaging and distribution system. As such this LWRP is clever enough to abstract away the small differences and can be used for managing either. This LWRP also creates the proper module .ini file for each PECL extension at the correct location for each supported platform.

Actions

Action	Description
install	Install a pear package - if version is provided, install that specific version
upgrade	Upgrade a pear package - if version is provided, upgrade to that specific version
remove	Remove a pear package
purge	Purge a pear package (this usually entails removing configuration files as well as the package itself). With pear packages this behaves the same as remove

Attributes

Attribute	Description	Default Value
package_name	Name Attribute: The name of the pear package to install	
version	the version of the package to install/upgrade. If no version is given latest is assumed.	
preferred_state	PEAR by default installs stable packages only, this allows you to install pear packages in a devel, alpha or beta state	stable
directives	extra extension directives (settings) for a pecl. on most platforms these usually get rendered into the extension's .ini file	
options	Add additional options to the underlying pear package command	

Examples

upgrade a pear

```
php_pear "XML_RPC" do
  action :upgrade
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

install a specific version

```
php_pear "XML_RPC" do
  version "1.5.4"
  action :install
end
```

install the mongodb pecl

```
php_pear "mongo" do
  action :install
end
```

install apc pecl with directives

```
php_pear "apc" do
  action :install
  directives(:shm_size => 128, :enable_cli => 1)
end
```

install the YAML pear from the symfony project

```
sc = php_pear_channel "pear.symfony-project.com" do
  action :discover
end
php_pear "YAML" do
  channel sc.channel_name
  action :install
end
```

install the beta version of Horde_Url from the Horde channel

```
hc = php_pear_channel "pear.horde.org" do
  action :discover
end
php_pear "Horde_Url" do
  preferred_state "beta"
  channel hc.channel_name
  action :install
end
```

php_pear_channel

PEAR Channels are alternative sources for PEAR packages. This LWRP provides and easy way to manage these channels.

Actions

Action	Description
discover	Initialize a channel from its server
add	Add a channel to the channel list, usually only used to add private channels. Public channels are usually added using the `:discover` action
update	Update an existing channel

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

remove	Remove a channel from the list

Attributes

Attribute	Description	Default Value
channel_name	Name Attribute: The name of the channel to discover	
channel_xml	file of the channel you are adding	

Examples

discover the horde channel

```
php_pear_channel "pear.horde.org" do
  action :discover
end
```

download xml then add the symfony channel

```
remote_file "#{Chef::Config[:file_cache_path]}/symfony-channel.xml" do
  source "http://pear.symfony-project.com/channel.xml"
  mode 0644
end
php_pear_channel "symfony" do
  channel_xml "#{Chef::Config[:file_cache_path]}/symfony-channel.xml"
  action :add
end
```

update the main channels

```
php_pear_channel 'pear.php.net' do
  action :update
end
php_pear_channel 'pecl.php.net' do
  action :update
end
```

python

python_pip

Install packages using the new hotness in Python package management...pip. Yo dawg...easy_install is so 2009, you better ask your local Pythonista if you don't know! The usage semantics are like that of any normal package provider.

Actions

Action	Description
install	Install a pip package - if version is provided, install that specific version
upgrade	Upgrade a pip package - if version is provided, upgrade to that specific version
remove	Remove a pip package
purge	Purge a pip package (this usually entails removing configuration files as well as the package itself). With pip packages this behaves the same as remove

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Attributes

Attribute	Description	Default Value
package_name	Name Attribute: The name of the pip package to install	
version	the version of the package to install/upgrade. If no version is given latest is assumed.	
virtualenv	virtualenv environment to install pip package into	
options	Add additional options to the underlying pip package command	

Examples

Install latest gunicorn into system path

```
python_pip "gunicorn" do
  action :install
end
```

Target a virtualenv

```
python_pip "gunicorn" do
  virtualenv "/home/ubunut/virtualenvs/myapp"
  action :install
end
```

Install a specific pip version

```
python_pip "django" do
  version "1.1.4"
  action :install
end
```

Use this provider with the core package resource

```
package "django" do
  provider Chef::Provider::PythonPip
  action :install
end
```

python_virtualenv

virtualenv is a great tool that creates isolated python environments. Think of it as RVM without all those hipsters and tight jeans.

Actions

Action	Description
create	Creates a new virtualenv
delete	Deletes an existing virtualenv

Attributes

Attribute	Description	Default Value

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

path	Name Attribute: The path where the virtualenv will be created	\
interpreter	The Python interpreter to use.	python2.6
owner	The owner for the virtualenv	
group	The group owner of the file (string or id)	

Examples

create a 2.6 virtualenv owned by ubuntu user

```
python_virtualenv "/home/ubunut/virtualenvs/myapp" do
  owner "ubuntu"
  group "ubuntu"
  action :create
end
```

create a Python 2.4 virtualenv

```
python_virtualenv "/home/ubuntu/virtualenvs/myapp" do
  interpreter "python2.4"
  owner "ubuntu"
  group "ubuntu"
  action :create
end
```

riak

riak_cluster

Joins a Riak node into a cluster. If you're using chef-client, the other cluster members will be automatically discovered by searching on their `riak.core.cluster_name` attribute. This resource is used in the `riak::autoconf` recipe.

Actions

Action	Description
join	Joins the local node to a cluster
wait_for_ringready	Waits until cluster membership converges

Attributes

Attribute	Description	Default Value
cluster_name	Name Attribute: the name of the cluster	
cluster_members	Other node names that should be part of the cluster	(discovered by Search)
node_name	The name of the local node (`riak.erlang.node_name` attribute)	
timeout	How long, in seconds, to wait for ring convergence	30
riak_admin_path	Where the `riak` and `riak-admin` scripts are located	`"/usr/sbin"`

Examples

From the `riak::autoconf` recipe:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
riak_cluster node[:riak][:core][:cluster_name] do
  node_name node[:riak][:erlang][:node_name]
  action :join
  riak_admin_path bin_path
end
```

samba

samba_user

Manages the smbpasswd entries for samba users.

Actions

Action	Description
create	Creates the user
enable	Enables the user
delete	Deletes the user

Attributes

Attribute	Description	Default Value
password	Password for the user passed to smbpasswd	

Examples

```
samba_user "jtimberman" do
  password "SuperSecret"
  action :create
end
```

transmission

transmission_torrent_file

Download a file via the BitTorrent protocol. The usage semantics are like that of the existing file and remote_file resources. This allows very fast downloads for files that are part of large BitTorrent swarms. The Ubuntu 10.04 LTS ISO (around 700MB) downloads in about 50 seconds.

Actions

Action	Description
create	Download a file via the BitTorrent protocol

Attributes

Attribute	Description	Default Value
path	Name Attribute: the path to the file	

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

torrent	torrent file of the swarm to join. can either be a url or local file path	
blocking	should the file be downloaded in a blocking way? If `true` Chef will download the file in a single Chef run, if `false` Chef will check for a completed download during each Chef run until the download is complete	true
continue_seeding	should the file continue to be seeded to the swarm after download?	false
owner	The owner for the file	
group	The group owner of the file (string or id)	
rpc_host	the address of the Transmission RPC host to connect to	localhost
rpc_port	the port of the Transmission RPC host to connect to	9091
rpc_username	the username of the Transmission RPC account.	transmission
rpc_password	the password of the Transmission RPC account . should probably be `node'transmission''rpc_password'` which by default is a secure password generated for this node.	

Examples

Download the Lucid iso

```
transmission_torrent_file "/home/ubuntu/ubuntu.iso"  do
  torrent "http://releases.ubuntu.com/lucid/ubuntu-10.04.1-server-i386.iso.torrent"
  owner 'ubuntu'
  group 'ubuntu'
  rpc_username node['transmission']['rpc_username']
  rpc_password node['transmission']['rpc_password']
  action :create
end
```

do the same thing but continue seeding after download

```
transmission_torrent_file "/home/ubuntu/ubuntu.iso"  do
  torrent "http://releases.ubuntu.com/lucid/ubuntu-10.04.1-server-i386.iso.torrent"
  owner 'ubuntu'
  group 'ubuntu'
  continue_seeding true
  rpc_username node['transmission']['rpc_username']
  rpc_password node['transmission']['rpc_password']
  action :create
end
```

windows

windows_package

Manage Windows application packages in an unattended, idempotent way.

The following application installers are currently supported:

- MSI packages
- InstallShield
- Wise InstallMaster
- Inno Setup
- Nullsoft Scriptable Install System

If the proper installer type is not passed into the resource's installer_type attribute, the provider will do it's best to identify the type by introspecting the installation package. If the installation type cannot be properly identified the `:custom` value can be passed into the installer_type attribute

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

along with the proper flags for silent/quiet installation (using the `options` attribute..see example below).

PLEASE NOTE - the resource's `package_name` should be the same as the 'DisplayName' registry value in the uninstallation data that is created during package installation. The easiest way to definitively find the proper 'DisplayName' value is to install the package on a machine and search for the uninstall information under the following registry keys:

- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall`
- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Uninstall`
- `HKEY_LOCAL_MACHINE\Software\Wow6464Node\Microsoft\Windows\CurrentVersion\Uninstall`



You may also be able to see this value from the "Name" column in the Windows control panel, programs and features, uninstall dialog. (Start->"Uninstall"->"Uninstall a program").

For maximum flexibility the `source` attribute supports both remote and local installation packages.

Actions

Action	Description
install	install a package
remove	remove a package

Attributes

package_name	**Name Attribute: The 'DisplayName' of the application installation package.**
source	The source of the windows installer. This can either be a URI or a local path.
installer_type	They type of windows installation package. valid values are: :msi, :inno, :nsis, :wise, :installshield, :custom. If this value is not provided, the provider will do it's best to identify the installer type through introspection of the file.
options	Additional options to pass the underlying installation command

Examples

install PuTTY (InnoSetup installer)

```
windows_package "PuTTY version 0.60" do
  source "http://the.earth.li/~sgtatham/putty/latest/x86/putty-0.60-installer.exe"
  installer_type :inno
  action :install
end
```

install 7-Zip (MSI installer)

```
windows_package "7-Zip 9.20 (x64 edition)" do
  source "http://downloads.sourceforge.net/sevenzip/7z920-x64.msi"
  action :install
end
```

install Notepad++ (Y U No Emacs?) using a local installer

```
windows_package "Notepad++" do
  source "c:/installation_files/npp.5.9.2.Installer.exe"
  action :install
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

install VLC for that Xvid (NSIS installer)

```
windows_package "VLC media player 1.1.10" do
  source "http://superb-sea2.dl.sourceforge.net/project/vlc/1.1.10/win32/vlc-1.1.10-win32.exe"
  action :install
end
```

install Firefox as custom installer and manually set the silent install flags

```
windows_package "Mozilla Firefox 5.0 (x86 en-US)" do
  source
"http://3347-mozilla.voxcdn.com/pub/mozilla.org/firefox/releases/5.0/win32/en-US/Firefox%20Setup%205.
0.exe"
  options "-ms"
  installer_type :custom
  action :install
end
```

Google Chrome FTW (MSI installer)

```
windows_package "Google Chrome" do
  source
"https://dl-ssl.google.com/tag/s/appguid%3D%7B8A69D345-D564-463C-AFF1-A69D9E530F96%7D%26iid%3D%7B806F
36C0-
                CB54-4A84-A3F30CF8A86575E0%7D%26lang%3Den%26browser%3D3%26usagestats%3D0%26appname%3D

Google%2520Chrome%26needsadmin%3Dfalse/edgedl/chrome/install/GoogleChromeStandaloneEnterprise.msi"
  action :install
end
```

remove Google Chrome (but why??)

```
windows_package "Google Chrome" do
  action :remove
end
```

remove 7-Zip

```
windows_package "7-Zip 9.20 (x64 edition)" do
  action :remove
end
```

windows_registry

Create and modify Windows registry keys.

Actions

Action	Description
create	create a new registry key with the provided values.
modify	modify an existing registry key with the provided values.

Attributes

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

key_name	**Name Attribute: The registry key to create/modify.**
values	hash of the values to set under the registry key. The individual hash items will become respective 'Value name' => 'Value data' items in the registry key.

Examples

make the local windows proxy match the one set for Chef

```
proxy = URI.parse(Chef::Config[:http_proxy])
windows_registry 'HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings' do
  values 'ProxyEnable' => 1, 'ProxyServer' => "#{proxy.host}:#{proxy.port}", 'ProxyOverride' =>
'<local>'
end
```

enable Remote Desktop and poke the firewall hole

```
windows_registry 'HKLM\SYSTEM\CurrentControlSet\Control\Terminal Server' do
  values 'FdenyTSConnections' => 0
end
```

windows_zipfile

Most version of Windows do not ship with native cli utility for managing compressed files. This resource provides a pure-ruby implementation for managing zip files. *Be sure to use the `not_if` or `only_if` meta parameters to guard the resource for idempotence, or action will be taken on the zip file every Chef run.*

Actions

Action	Description
unzip	unzip a compressed file

Attributes

path	**Name Attribute: The path where files will be unzipped to.**	**"."**
source	The source of the zip file. This can either be a URI or a local path.	
overwrite	force an overwrite of the files if the already exists.	false

Examples

unzip a remote zip file locally

```
windows_zipfile "c:/bin" do
  source "http://download.sysinternals.com/Files/SysinternalsSuite.zip"
  action :unzip
  not_if {::File.exists?("c:/bin/PsExec.exe")}
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

unzip a local zipfile

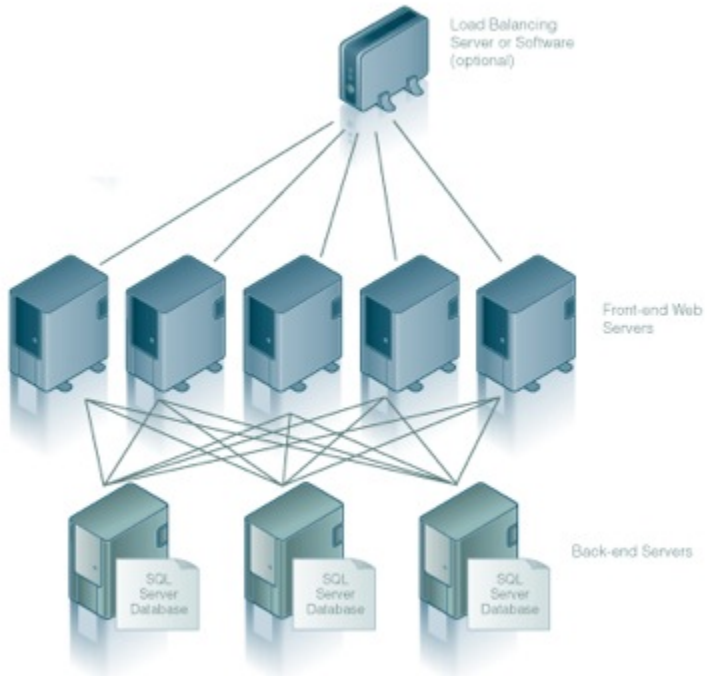
```
windows_zipfile "c:/the_codez" do
  source "c:/foo/baz/the_codez.zip"
  action :unzip
end
```

 Lightweight Resources and Providers (LWRP)

Roles 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Roles



Overview

A role provides a means of grouping similar features of similar nodes, providing a mechanism for easily composing sets of functionality.

At web scale, you almost never have just one of something, so you use roles to express the parts of the configuration that are shared by a group of Nodes.

- **Roles consist of the same parts as a node: Attributes and a Run List.**
- **Nodes can have multiple roles applied, and they will be expanded in place, providing for a complete recipe list for that node.**
- **When the Chef client runs, it merges its own attributes and run list with those of any roles it has been assigned.**

For example, let's say I have a list of Recipes that should be applied on all my Ubuntu servers, and a list that is specific to machines that are Web Servers. This would map cleanly to two roles: 'ubuntu' and 'web_server'.

Each role would specify the list of recipes it requires, in the order they should be applied. Where there is overlap (i.e., a recipe appears twice) it will be run when the first role is expanded (not run twice.)

You can create roles in Chef in 4 different ways:

1. through the creation of role files in your chef repository (that utilize a ruby DSL, which gets compiled to JSON,)
2. the creation of the JSON files directly in your chef repository,
3. through Knife or the Open Source Chef Server Management Console,
4. or through the REST API.

Once you have created a role, you can add it to a node through the Management Console by dragging it from the Roles list to the client's "run list", you can use the Chef command line tool Knife, or you can add it with a JSON file.

Roles work with chef-solo too!
You can use Roles with Chef Solo as well as the Client/Server. See the Chef Solo documentation.

Choose Your Workflow

There are two workflows you can use to manage your roles: you can either write your roles as ruby or JSON files and push them to the server, **OR** you can create roles with Knife or the Open Source Chef Server Management Console and edit them as you go along.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

If you create and edit roles by editing files:

- You can use the ruby DSL or JSON
- Your roles will be useable with chef-solo
- You can dynamically generate role data using the ruby DSL
- You can keep your roles in your version control system and have a history of who changed what and when.
- You **should not** edit roles with the Open Source Chef Server Management Console, as your edits will be overwritten next time you upload from the file.

If you create and edit roles with Knife and the Open Source Chef Server Management Console:

- You can edit a role's run list visually with the Open Source Chef Server Management Console
- You can make changes to a role "on the fly"
- The canonical source of a role's data will be the Chef Server, which makes it difficult to keep in version control
- You cannot use the ruby DSL.
- You cannot use your roles with chef-solo

These workflows are mutually exclusive:

- *If you upload a role to your Chef server from a file, then edit it with the Open Source Chef Server Management Console, and then edit the file and upload again, the changes you made with the Open Source Chef Server Management Console will be lost.*
- The same is true with Knife: when managing roles as files, you will generally upload your changes to the chef server with knife; however,
- *if you use `knife role edit` to modify a role, your changes will be overwritten when you next upload the role from the file.*

The Ruby DSL

Roles created through this mechanism get compiled to JSON, and then are loaded in to the Chef Server. (We never execute ruby code directly on the chef server!) Each time you rake install your Chef Repository, we will re-compile the corresponding JSON and store it in the chef server.

You should create these files in the 'roles' subdirectory of your chef repository - if you are using the opscode canonical Chef Repository as a baseline, the latest version includes this directory and rake tasks to manipulate roles as documented on the repository page. If your repository doesn't have this directory, create it now. Each DSL file should have a .rb suffix.

A complete role file looks like this:

A Very Simple Web Server Role

```
name "webserver"
description "Simple Web App"
run_list(
  "recipe[apache2]"
)
```



Chef 0.10.x feature
The env_run_lists feature is only available in Chef 0.10.0 or above.

The Web Server Role

```
name "webserver"
description "The base role for systems that serve HTTP traffic"
run_list "recipe[apache2]", "recipe[apache2::mod_ssl]", "role[monitor]"
env_run_lists "prod" => ["recipe[apache2]"], "staging" => ["recipe[apache2::staging]"]
default_attributes "apache2" => { "listen_ports" => [ "80", "443" ] }
override_attributes "apache2" => { "max_children" => "50" }
```

We'll go over each section below.

name

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Each role must have a unique name, which is made up of [A-Z][a-z][0-9] and [_-]. Spaces are not allowed.

name field

```
name 'webserver'
```

description

A short description of what functionality is covered by this role.

description field

```
description 'The base role for systems that serve HTTP traffic'
```

run_list

The recipes attribute is replaced with a run_list, which is identical to the one you specify for Node. This is the list of recipes or roles to apply for this role, in the order they should be applied.

recipes

```
run_list "recipe[apache2]", "recipe[apache2::mod_ssl]", "role[monitor]"
```

Would apply the "apache2" recipe, the "apache2::mod_ssl" recipe, and then anything required by the role "monitor".

Note: You may also see `recipes` used in place of `run_list`. *These are equivalent in function, though `recipes` is deprecated*, so you should use `run_list` instead.

env_run_lists

Chef 0.10.x Feature
The env_run_lists feature is only available in Chef 0.10.0 or above.

In Chef 0.10.x, per environment run lists in a role allow you to specify a different run list for a role if the role is in the run list of the node in a specific environment. Before the existence of environments, each role has only one run list and it will be expanded and applied to the node when chef-client runs. With environments, in the role you can specify one run list for each environment, such as:

per environment run lists

```
env_run_lists "prod" => ["recipe[apache2]"], "staging" => ["recipe[apache2::staging]"]
```

If you do not specify a per environment run list for a certain environment, such as "production" and "preprod" in the above example, chef will use the default run list (like before). If it is the default run list that you desire, there is no need to specify it.

Environments has an additional example, based in the JSON format that would be used for interacting directly with the REST API.

default_attributes

An optional set of attributes that should be applied to all nodes with this role, assuming the node does not already have a value for that attribute. Use this to set site-wide defaults that can be overridden on a node-specific basis. **The merge is 'deep', meaning that we will preserve nested attributes properly.**

default_attributes

```
default_attributes "apache2" => { "listen_ports" => [ "80", "443" ] }
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

In the above example, all nodes with this role would have `node[:apache2][:listen_ports]` set to '80' and '443', assuming they do not already have a value. *If more than one role attempts to set a default value for the same attribute, the last role applied will win.*

override_attributes

An optional set of attributes that should be applied to all nodes with this role, regardless of whether a node already has a value for that attribute. U seful for setting site-wide values that will always be set. **The merge is 'deep', meaning that we will preserve nested attributes properly.**

override_attributes

```
override_attributes "apache2" => { "max_children" => "50" }
```

In the example above, `node[:apache2][:max_children]` will **always** be set to '50'. *If more than one role attempts to set an override value for the same attribute, the last role applied will win.*

The parameters in role `.rb` files are actually Ruby method calls, so you can add parentheses and provide the attribute hashes on separate lines for clarity if specifying numerous or deeply-nested attributes:

Overriding Nested Attributes

```
override_attributes(
  :apache2 => {
    :prefork => { :min_spareservers => "5" }
  }
)
```

Multiple attributes can be overridden like this:

Overriding Multiple Attributes

```
override_attributes(
  :apache2 => {
    :prefork => { :min_spareservers => "5" }
  },
  :tomcat => {
    :worker_threads => "100"
  }
)
```

As JSON

The JSON format for roles maps directly to the Ruby DSL above. For the role we describe in that section, the corresponding JSON is:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
{
  "name": "webserver",
  "chef_type": "role",
  "json_class": "Chef::Role",
  "default_attributes": {
    "apache2": {
      "listen_ports": [
        "80",
        "443"
      ]
    }
  },
  "description": "The base role for systems that serve HTTP traffic",
  "run_list": [
    "recipe[apache2]",
    "recipe[apache2::mod_ssl]",
    "role[montior]"
  ],
  "env_run_lists" : {
    "production" : [],
    "preprod" : [],
    "dev": [
      "role[base]",
      "recipe[apache]",
      "recipe[apache::copy_dev_configs]",
    ],
    "test": [
      "role[base]",
      "recipe[apache]"
    ]
   },
  "override_attributes": {
    "apache2": {
      "max_children": "50"
    }
  }
}
```

The two additional fields are described below.

json_class

This should always be set to `Chef::Role`.
This is used internally by Chef to auto-inflate this type of object. It should be ignored if you are re-building objects outside of Ruby, and its value may change in the future.

chef_type

This should always be set to `role`.
This is the field you should rely on if you are building a system to consume Roles outside of Ruby.

Managing Roles through Knife

Knife is Chef's command line tool. Roles can be managed through Knife. Please refer to Managing Roles With Knife for how this is done.

Managing Roles through the Open Source Chef Server Management Console

You can create and manage roles through the Open Source Chef Server Management Console. Please refer to the Managing Roles through the

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Management Console articles for information about this topic.

The REST API

You can also create and manage roles directly in a running Chef Server via the REST API. Please refer to the Server API article for information.

 Resources and Providers

Search 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Search



Overview

*Search is a feature of the **Chef Server** that allows you to use a full-text search engine (based on **Apache Solr**) to query information about your infrastructure and applications.*

*Searches are built by the Chef Server, and allow you to query arbitrary data about your infrastructure. You can utilize this service via **search calls in a recipe** or the `knife search` command.*

Most data that Chef stores is automatically indexed in Solr including Data Bags, API Clients, Nodes, and Roles.

Search Index Names

Chef's built-in types are indexed in the following search indexes (don't worry about the search syntax for now):

Data Type	Index Name	Example Knife Search
Roles	role	`knife search role "name:production*"`
Nodes	node	`knife search node "name:app*"`
API Clients	client	`knife search client "name:c*"`
Environments	environment	`knife search environment "*:*"`

Data bags are indexed by the data bag's name. For example, to search for items in a data bag named 'bag_o_data' in knife, use a query like:

```
knife search bag_o_data "arbitrary_key:some_value"
```

The 'client' index is currently affected by a bug and will return incorrect results.

Query Syntax

Queries have the form "field:search_pattern" where "field" is a key in the JSON description of the objects being searched. The search pattern supports exact, wildcard, range, or fuzzy matching. The field name has limited support for wildcard matching.

Note: Both the field and search pattern is case sensitive.

Field Name Syntax

Field names are keys in the JSON description of the object being searched.

To search in a nested key, for example `dmi / system / product_name`, insert an underscore "_" between key names:

```
knife search node "dmi_system_product_name:ProLiant*"
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Note that wildcard use with nested field has changed between version 0.9 and 0.10. See Nested Fields below for details.

Discovering Key Names

The definitive list of search keys are the keys in the JSON description of the object being searched. To see the keys available when searching nodes:

```
knife node show staging -Fj | less
```

This command will open a full JSON description of the node "staging" in the pager `less`. Similarly `knife data bag show`, `knife client show`, and `knife role show` can be used to find keys available for their respective objects. The json keys are the definitive search keys to be used in any search context (knife, recipe, Hosted Chef, etc).

Wildcard Matching for Field Names

The field name also has limited support for wildcard matching. Both the "*" and "?" wildcards (see below) can be used within a field name; however, they cannot be the first character of the field name. For example, the following are valid queries:

```
knife search node 'platfo*:ubuntu'
knife search node 'platfor?:ubuntu'
```

Tutorials from the Community

- ***Chef Search As A Management and Automation Tool***

Community member Christian Paredes put together a blog post on **using Search in Recipes** to gather information on the database nodes and then to take action based upon that defined criteria.

Exact Matches

To search for an exact match in a field, use a query of the form "field:TEXT_TO_MATCH". Be sure to quote any search patterns with spaces using double quotes. You'll still need to quote the entire query to prevent your shell or ruby from trying to interpret it. The best way to do this is to quote your query with single quotes and the search pattern with double quotes, as shown in the second example below.

Examples:
Search for a data bag item in the 'admin' data bag with the exact value of 'charlie' for the key 'id':

```
knife search admins 'id:charlie'
```

EXAMPLE RESULT:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
1 items found
_rev:       1-39ff4099f2510f477b4c26bef81f75b9
chef_type:  data_bag_item
comment:    Charlie the Unicorn
data_bag:   admins
gid:        ops
id:         charlie
shell:      /bin/zsh
uid:        1005
```

Search for a data bag item with the exact text 'Charlie the Unicorn' for the key 'comment':

```
$knife search admins 'comment:"Charlie the Unicorn"'
```

EXAMPLE RESULT:

```
1 items found

_rev:       1-39ff4099f2510f477b4c26bef81f75b9
chef_type:  data_bag_item
comment:    Charlie the Unicorn
data_bag:   admins
gid:        ops
id:         charlie
shell:      /bin/zsh
uid:        1005
```

Wildcard Matches



Change to Query Syntax in Chef 0.10
Prior to Chef 0.10, search queries could not include the wildcards '*' or "?" as the first character of the search pattern.

In version Chef 0.10 and beyond, the "*" wildcard can now be used as the first character of a search pattern (but not a field/key name). "?" still cannot be used as the first character of the search pattern.

In Chef 0.10+, the following queries will return any node in which the key `foo` exists:

```
knife search node 'foo:*'
```

This same behavior can be achieved in 0.9 with the following query:

```
knife search node 'foo:[* TO *]'
```

Chef searches can use the familiar * (asterisk) and ? wildcard symbols to query for substring matches. These work much as they do in the shell: the * symbol matches zero or more characters, while the ? matches exactly one character. If we have a node named 'app1.example.com', the following queries will find it (and possibly other nodes as well):

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife search node 'name:app*'
knife search node 'name:app1*.example.com'
knife search node 'name:app?.example.com'
knife search node 'name:app1.example.???'
```

Range Search

Chef also supports range searches. To see how these work, suppose we have a data bag named 'sample' with items 'abc', 'bar', 'baz', and 'qux'. We can search for all of the items between 'bar' and 'foo', inclusive, using the search pattern `[bar TO foo]` ("TO" **must** be capitalized):

```
knife search sample "id:[bar TO foo]"
```

To search an *exclusive* range, use curly braces, like this:

```
knife search sample "id:{bar TO foo}"
```

Fuzzy Search



This search feature is currently affected by a bug and does not work.

Solr supports fuzzy matching of terms using an edit distance measure. To specify a fuzzy match, add a ~ (tilde) symbol to the end of your query term. For example, if you have an API client named 'FOO', you can find it by searching for 'BOO~', like this:

```
knife search client "name:BOO~"
```

```
{
  "total": 1,
  "start": 0,
  "rows": [
    {
      "public_key": "too long didn't read",
      "name": "FOO",
      "_rev": "1-f11a58043906e33d39a686e9b58cd92f",
      "json_class": "Chef::ApiClient",
      "admin": false,
      "chef_type": "client"
    }
  ]
}
```

Joining Multiple Query Criteria with Boolean Operators

For more exacting searches you can join multiple criteria with 'AND' or 'OR' or you can negate a query with 'NOT' (all three keywords **must** be capitalized).

The following examples assume you have a data bag named 'sample' with items 'abc', 'foo', 'bar', 'baz', and 'qux'.

Negating with NOT
`knife search sample "(NOT id:foo)"`

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Result:

```
{
  "total": 4,
  "start": 0,
  "rows": [
    {
      "comment": "an item named bar",
      "id": "bar",
      "animal": "cat"
    },
    {
      "comment": "an item named baz",
      "id": "baz"
      "animal": "dog"
    },
    {
      "comment": "an item named abc",
      "id": "abc",
      "animal": "unicorn"
    },
    {
      "comment": "an item named qux",
      "id": "qux",
      "animal", "penguin"
    }
  ]
}
```

Joining Queries with OR

```
knife search sample "id:foo OR id:abc"
```

Results:

```
{
  "total": 2,
  "start": 0,
  "rows": [
    {
      "comment": "an item named foo",
      "id": "foo",
      "animal": "pony"
    },
    {
      "comment": "an item named abc",
      "id": "abc",
      "animal": "unicorn"
    }
  ]
}
```

Joining Queries with AND

```
knife search sample "id:b* AND animal:dog"
```

Results:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
{
  "total": 1,
  "start": 0,
  "rows": [
    {
      "comment": "an item named baz",
      "id": "baz",
      "animal": "dog"
    }
  ]
}
```

Finding All

To find all of the items in an index, use an asterisk for both the field (key) name and search pattern, for example:

```
knife search node "*:*"
```

Will find all nodes.

Special Characters:

The following characters have special meaning to the query parser. If you need to use them in a search pattern, escape them with a \ (backslash):

```
+ - && || ! ( ) { } [ ] ^ " ~ * ? : \
```

As previously noted, Chef's search feature uses Apache SOLR, which is built on Apache Lucuene. Definitive documentation for the query syntax can be found in the projects' documentation sites:

- http://wiki.apache.org/solr/SolrQuerySyntax
- http://lucene.apache.org/java/2_3_2/queryparsersyntax.html

Nested Fields

Nested Field Syntax Changes in 0.10.0
The syntax for searching within nested fields will change in Chef 0.10.0. Instead of using 'X' as the wildcard character, you must now use '*'. In addition, you will now have full wildcard expansion on the search keys. A pre-0.10.0 query written as **'network_interfaces_X_addresses:192.168'** will now be written as **'network_interfaces_*_addresses:192.168'** or any other wildcard expansion of the key, such as **'network*addresses:192.168'**.

It is quite common in Chef for data you want to be a few levels deep in a data structure. For example, information about a network interface might be in a node's attributes at `node[:network][:interfaces][:eth0]`. Here's how Chef's search feature handles this. Consider this snippet of ohai data from my laptop:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
{"network":
  [
//snipped...
    "interfaces",
      {"en1": {
        "number": "1",
        "flags": [
          "UP",
          "BROADCAST",
          "SMART",
          "RUNNING",
          "SIMPLEX",
          "MULTICAST"
        ],
        "addresses": {
          "fe80::fa1e:dfff:fed8:63a2": {
            "scope": "Link",
            "prefixlen": "64",
            "family": "inet6"
          },
          "f8:1e:df:d8:63:a2": {
            "family": "lladdr"
          },
          "192.168.0.195": {
            "netmask": "255.255.255.0",
            "broadcast": "192.168.0.255",
            "family": "inet"
          }
        },
        "mtu": "1500",
        "media": {
          "supported": {
            "autoselect": {
              "options": [

              ]
            }
          },
          "selected": {
            "autoselect": {
              "options": [

              ]
            }
          }
        },
        "type": "en",
        "status": "active",
        "encapsulation": "Ethernet"
      },
// snipped
```

Before adding this node data to the indexer, Chef extracts nested fields into the top level:

```
"broadcast" => "192.168.0.255",
"flags"     => ["UP", "BROADCAST", "SMART", "RUNNING", "SIMPLEX", "MULTICAST"]
"mtu"       => "1500"
```

So the following searches will find this node:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife search node "broadcast:192.168.0.*"
knife search node "mtu:1500"
knife search node "flags:UP"
```

Chef will also flatten nested items into compound keys, like this:

```
# ...snip...
"network_interfaces_en1_addresses_192.168.0.195_broadcast" => "192.168.0.255",
"network_interfaces_en1_addresses_fe80::fa1e:tldr_family"  => "inet6",
"network_interfaces_en1_addresses"                         =>
["fe80::fa1e:tldr","f8:1e:df:tldr","192.168.0.195"]
# ...snip...
```

So you can also find the node with the following search:

```
knife search node "network_interfaces_en1_addresses:192.168.0.195"
```

Chef also creates "wildcard" compound keys, where 'X' is the wildcard character:

```
"network_interfaces_X_flags"     => ["UP", "BROADCAST", "SMART", "RUNNING", "SIMPLEX", "MULTICAST"]
"network_interfaces_X_addresses" => ["fe80::fa1e:dfff:fed8:63a2", "192.168.0.195",
"f8:1e:df:d8:63:a2"]
"network_interfaces_en0_media_X" => ["autoselect", "none", "1000baseT", "10baseT/UTP", "100baseTX"]
"network_interfaces_en1_X"       => ["1", "UP", "BROADCAST", "SMART", "RUNNING", "SIMPLEX",
"MULTICAST",
                                     "fe80::fa1e:dfff:fed8:63a2", "f8:1e:df:d8:63:a2",
"192.168.0.195",
                                     "1500", "supported", "selected", "en", "active", "Ethernet"]
```

These compound keys can be extremely helpful when a key you don't care about is between two keys that you do care about. For example to find every node with an IP address starting with 192.168, regardless of the interface name, you can search for:

```
knife search node 'network_interfaces_X_addresses:192.168*'
```

You could even use a range search to find every node with an address in a given subnet:

```
knife search node 'network_interfaces_X_addresses:[192.168.0.* TO 192.168.127.*]'
```

You can see that by combining these wild card fields with range and wildcard queries, it is possible to perform very powerful searches such as using the vendor part of the MAC address to find every node that has a network card made by a given vendor (Broadcom, Intel, etc.).

Using Search in Recipes

The Chef DSL provides the `search(INDEX, QUERY)` method for accessing search indexes in recipes. Basic use of search looks like this:

```
# Returns an array of all nodes
search(:node, "*:*")
```

If you just want to use each result of the search and don't care about the aggregate result you can provide a code block to the search method. Each result will be passed to the block in turn:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
# Print every node matching the search pattern
search(:node, "*:*") do |matching_node|
  puts matching_node.to_s
end
```

Search for Nodes Based on Run List Entries

Find Nodes With a Given Recipe in the Run List

To find nodes with a specified recipe in the run list, just search within the `run_list` field. Be careful, Lucene treats : and [as special characters, hence escaping is needed (also note the single quoting):



Use the `recipe` (singular!) keyword for searching in the top-level run list.

```
search(:node, 'run_list:recipe\[foo\:\:bar\]')
```

If you also want to interpolate variables into the search string using ruby's alternate quoting syntax makes this easier:

```
search(:node, %Q{run_list:"recipe[#{the_recipe}]"} )
```

Find Nodes with a Recipe in the Expanded Run List



0.9.8 and up
This is a new feature as of Chef 0.9.8. The change is made on the client, so you'll only be able to find nodes running chef-client 0.9.8 and greater when using this feature.

Chef saves expanded list of roles and recipes to the `roles` and `recipes` attributes on the node. This makes it possible to find nodes that run a given recipe even if it's included by a role (or a role-within-a-role). For example, consider a "tomcat_server" role that includes `apache2::mod_proxy_ajp` in its run list. A node that is in this role would be found by the following search:



Use the `recipes` (plural! Note the extra "s"!) keyword for searching in the expanded run list. This field is updated on each run of chef-client, thus changes to the run list will not affect "recipes" until chef-client has been run on the node.

```
search(:node, 'recipes:apache2\:\:mod_proxy_ajp')
```

Find Nodes with a Role in the Run List

To find a node that includes a role in the top level of its run list, use a search of the form `role:ROLE_NAME`.



Use the `role` (singular!) keyword for searching in the top-level run list.

```
search(:node, 'role:load_balancer')
```

```
knife search node role:load_balancer
```


This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Find Nodes with a Role in the Expanded Run List

0.9.8 and up
This is a new feature as of Chef 0.9.8. The change is made on the client, so you'll only be able to find nodes running chef-client 0.9.8 and greater when using this feature.

Since Chef 0.9.8, chef-client automatically saves the expanded list of roles (i.e., all roles applied to a node, including nested roles) to the node's automatic attributes. You can find every node that has a given role by a search of the form `roles:ROLE_NAME`.

Use the `roles` (plural! Note the extra "s"!) keyword for searching in the expanded run list. This field is updated on each run of chef-client, thus changes to a node's run list will not affect "roles" until chef-client has been run on the node.

```
search(:node, 'roles:load_balancer')
```

```
knife search node roles:load_balancer
```

Client/Server Settings for a Database

If you don't have roles fully defined and implemented, you may find it necessary on a webserver to place the hostname, ip or private ip of another machine in a settings file so that it can connect to a database, solr or other persistence server. It is assumed here that if a webserver is named mysqlchef that its database server is mysqlchefutil.

Consider this simplified settings file for the webserver mysqlchef1:

```
username: "mysql"
password: "MoveAlong"
host:     "10.40.64.202"  #Private IP of mysqlchefutil
port:     "3306"
```

You can see all the information about the node with the following command:

```
knife search node "name:mysqlchefutil" --long
```

To access it as part of a recipe that is to run on mysqlchef (the webserver):

```
db_server = search(:node,"name:mysqlchefutil")
private_ip = "#{db_server[0][:rackspace][:private_ip]}"
puts private_ip
```

Note the 0 (zero) index for the db_server identifier is used because you will get a single document returned from solr because the node is being searched on its unique name. The identifier "private_ip" now has the value "10.40.64.202" and can be used in templates as a variable, etc.

Searching for nodes having a given recipe applied (pre 0.9.8)

If you want to handle role inclusion gracefully, you can patch the Chef Language (see this gist) to allow that kind of search:

```
# returns all node having this role applied, even if the role is included in a role)
 nodes = esearch(:node, 'role\[admin\]')
```


This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

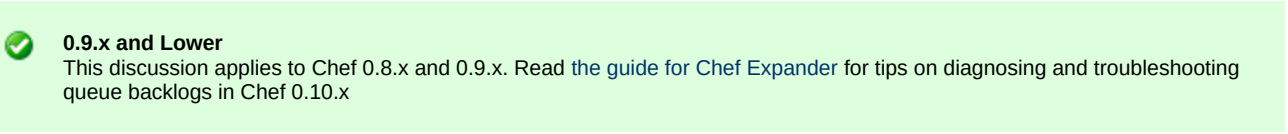
Searching within Environments

To search within a specific environment, see Searching within Environments.

Problems and Solutions

The following are solutions to problems/questions that arise in use of the Search feature and in maintaining the Indexes.

Updates not propagating to search results



0.9.x and Lower
This discussion applies to Chef 0.8.x and 0.9.x. Read the guide for Chef Expander for tips on diagnosing and troubleshooting queue backlogs in Chef 0.10.x

Sometimes you will notice that your search results get stale. For instance, you may have a node visible from "knife node show", but "knife search node" does not show the node. The first thing to check is the size of your message queue:

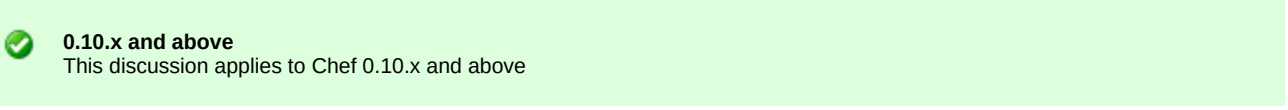
```
$ sudo /usr/sbin/rabbitmqctl list_queues -p /chef
Listing queues ...
chef-index-consumer-default     4
...done.
```

A low number is expected here. *If it gets past 15 or 20, and especially if it is constantly increasing, you need to add more chef-solr-indexer instances.*

Adding indexers using Runit

Instructions on adding services is here: http://smarden.org/runit/faq.html. You can reuse the scripts from /etc/sv/chef-solr-indexer.

Fields missing from certain nodes



0.10.x and above
This discussion applies to Chef 0.10.x and above

If you are getting strange results from your search queries, like certain nodes missing from searches for attributes you know exist (such as those confirmed to exist via the webui), you may have run into solr's maxFieldLength issue described in CHEF-2346. So far this has been noted on linux servers with complex configurations and Windows machines - anywhere with a lot of attributes and Ohai data. This is currently unresolved as of 0.10.8.

Search only returns IP Address of the Node, not of a specific interface

The issue is complicated, as the JSON data for interfaces is structured as hierarchical attributes of the parent. (Where the interface has an attribute, which is an address, which has an attribute which is a family) However, the way our minds tend to work is that the interface has an address of a particularly family, usually ipv4 (inet). There is a tendency to want something like: `node[:ipaddress_eth1] or node[:ipaddress][:eth1]` but these diverge from the data hierarchy. OHAI-88 is based in this issue.

Pending the determination on how to best address this circumstance, the network_addr.rb Ohai plugin, from Opscode Team Member Joshua Timberman, extends the Ohai network attribute with additional `ipaddrtype_iface` attributes to make semantically easier to retrieve the addresses.

Additionally, the following code should get the ip address of a specific interface, as it is constructed consistently with how the JSON data for interfaces is structured.

```
ipaddress_eth1 = host["network"]["interfaces"]["eth1"]["addresses"].select { |address, data|
data["family"] == "inet" }[0][0]
```


This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Roles

Installation



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Installation



The Flavors of Chef

No matter how complex the realities of your business, Chef makes it easy to deploy servers and scale applications throughout your entire infrastructure. Opscode provides different flavors of Chef, so there is a means for you to manage your infrastructure.

To install Chef, follow the directions for the Chef "flavor" that applies to you.

Chef Server + Chef Clients

To use your own Chef Server to manage your nodes, you will need to

1. Install a Chef Server
2. Configure your local workstation
3. Install chef-client on your nodes

Hosted Chef + Chef Clients

Those using Hosted Chef do not need to install a Chef Server. Rather, to begin using Chef to manage your infrastructure, you need to

1. Create a Hosted Chef Account,
2. Configure your local workstation, and
3. Install chef-client on your nodes

The Fast Start Guide provides a tutorial-style step-by-step introduction to using Chef for those using Hosted Chef.

Chef Solo

Those using Chef-Solo to configure nodes within their infrastructure need to

1. Install chef-solo on your nodes, and
2. Configure chef-solo for use

Note that chef-solo is shipped in the same package as chef-client.

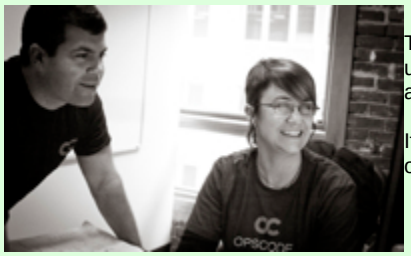
Current Release
These directions cover the installation of the current release. The current release of Chef is 0.10.8.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Additional Fast Start or Installation Documentation

Cookbook Fast Start Guide Getting Started with Shef Installing Chef from HEAD
Vagrant Management Workstation as Chef Client

Looking for instant gratification?



The **Fast Start Guide** will get you up and running quickly on Ubuntu or Mac OS X using Hosted Chef (as your chef server). There is also a Fast Start Guide for CentOS as well as a Fast Start Guide for Windows.

If you prefer to install your own server, use the Chef Server + Chef client directions on this page.

Search

Fast Start Guide

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Fast Start Guide



This page is an install guide that describes how to get up and running with Hosted Chef as quickly as possible and ends with a fast demonstration on how to work with cookbooks.

Assumptions

In order to make this as quick as possible, we make some assumptions. *If your system does not meet these assumptions, you will need to use the Installation instructions that apply for the "flavor" of Chef you are installing.*

Operating System

Chef runs on many popular Unix and Linux platforms as well as Max OSX and Windows. *We will describe how to set up using **Ubuntu 10.04** as a workstation and a client, but these are general directions that apply to most installations.*

There is a Fast Start Guide for CentOS as well as a Fast Start Guide for Windows which goes over these same directions on CentOS 6.0 and Windows 2008 R2. For other OS specific instructions, go to the section of Installation that applies to the "flavor" of Chef you are installing, and proceed from there.

Management Workstation as a Client

In this guide we will be setting up a management workstation as a client. This means that the one node we setup here can be used to both manage your cookbooks and configuration in Hosted Chef, as well as use chef-client to run recipes on it. *(If you already have a workstation configured and just want to add a client node, see the Client Bootstrap Fast Start Guide or the EC2 Bootstrap Fast Start Guide.)*

The Difference Between a Management Workstation and a Client

- *Management Workstations* have git installed so you can download and upload cookbooks with knife, and
- *Clients* have chef-client configured so Chef can be run on them.

If you'd like more information on these terms, the wiki page on Architecture Introduction has a very detailed explanation.

We will be setting up the client node as both in this guide.

- Once you add more nodes you will not need to set them all up as workstations.
- You do however need at least one management workstation to manage your client nodes.

You don't have to set up your management workstation to be a chef-client. If you prefer not to:

- Follow this guide through Step 4, and then
- use the Client Bootstrap Fast Start Guide or the EC2 Bootstrap Fast Start Guide to configure a client node separately.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

System Requirements

Chef-client is supported on the following platforms

- Ubuntu (10.04, 10.10, 11.04, 11.10)
- Debian (5.0, 6.0)
- RHEL & CentOS (5.x, 6.x)
- Fedora 10+
- Mac OS X (10.4, 10.5, 10.6, 10.7)
- Windows 7
- Windows Server 2003 R2, 2008 R2

Additionally, chef-client is known to run on the following platforms

- Ubuntu (6.06, 8.04-9.10)
- Gentoo (11.1,11.2)
- FreeBSD (7.1)
- OpenBSD (4.4)
- OpenSolaris (2008.11)
- Solaris 5.10 (u6)
- SuSE (11.x)
- Windows XP, Vista

We will be installing (or updating) the following in this Installation Guide.

Ruby
Ruby 1.8.5, 1.8.6, 1.8.7, 1.9.1 or 1.9.2 with SSL bindings is required.

RubyGems
Version 1.3.7 or greater. On Ubuntu and Debian Rubygems should be installed from source.

Chef Client Version
Hosted Chef is compatible with chef-client v0.9.0 and greater. Older clients need to be upgraded before connecting.

Steps

This guide is going to direct you through the following steps:

1. Create a Hosted Chef account
2. Install and Update dependencies - ruby, ruby gems, ruby-dev and git-core
3. Install Chef and create directories needed
4. Connect to Hosted Chef
5. Configure the workstation as a client
6. Download a Cookbook from the Community Site
7. Upload the cookbook to Hosted Chef
8. Add the Cookbook to the run_list
9. Run chef-client with the new Cookbook

At the end of this guide you will have a workstation setup with chef-client installed, a recipe downloaded and added to the node's run_list. These steps end with the recipe being run on the node by Chef.

Step 1: Create a Hosted Chef account

Working with a Chef is primarily done from a local workstation. Changes are made locally and uploaded to the Chef Server. Since we're using Hosted Chef as the Chef Server, you'll need to sign up for an account if you haven't already. (Don't worry! We won't charge you to use Hosted

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef for this guide - you can setup up to 5 nodes for free.)

> **Would you like to have screenshots walk you through the actions in this step instead?** If so, head over to *Setup Opscode User and Organization* which will guide you through creating a new organization, and then return here to the next step once complete.
>
> If you have already created your Organization, there are directions on how to recreate the organization key and download the knife config on the *Managing Organizations with the Hosted Chef Management Console* page and recreating existing user keys is explained on the *Managing Users with the Hosted Chef Management Console* page.

Sign up for a Hosted Chef account. You will get an email validation shortly after you sign up. After you complete the sign-up, create an organization in the Console page, and then download the following files:

- Your Organization validation key. This is used to automatically register new Chef Clients (like servers you manage).
- The Knife configuration file.
- Your User key. This is used to authenticate your user with Hosted Chef.

The private keys are not stored by Hosted Chef. Download them to your local system, we'll use them in Step #5.

Step 2: Install and Update Dependencies

We're going to use Ruby 1.8.7, the default available on our chosen operating systems. We also need to install the ruby-dev package, as well as git and a few other dependencies. We will install RubyGems from source.

Git isn't needed to setup a node as a client, but it will be needed to setup a node as a workstation and follow the steps on this page. *Workstations* are able to download recipes from the Community Site, edit them, and them upload them to Hosted Chef. Once they are uploaded to the server any node that has been setup as a client or workstation will be able to use them. *Clients* do not need git installed but are unable to directly edit the recipes or upload changes to Hosted Chef without it. You will want at least one workstation setup to manage your organization. These directions walk you through setting up a workstation, for specific instructions on setting up a client instead, see the Installation page.

Below is an example of how you would install these in Ubuntu. For OS specific instructions, see the Installation page.

Install Ruby and Development Tools

```
sudo apt-get update
sudo apt-get install ruby ruby-dev libopenssl-ruby rdoc ri irb build-essential wget ssl-cert git-core
```

> **Chef 0.10 Note**
> There are open issues with Chef 0.10.x and Ruby 1.8.6. If you are using Chef 0.10.x, you should get 1.8.7 with SSL bindings. If you'd like to use 1.9.1 or 1.9.2, you'll want to follow the Installing Chef Client on Other Operating Systems directions to install manually with gems.

Install RubyGems 1.3.7+

First, check to see if your operating system already has RubyGems installed and is at least version 1.3.7 by running the following:

```
% gem -v
1.3.7
```

If you're using Ubuntu, we recommend installing RubyGems from source as the default RubyGems package disables `update --system`. If you have 1.3.7 installed and are not on Ubuntu, you have completed installing the dependencies needed for chef-client and you can move on to Step 3: Install Chef.

If you have an older version

You will need to update rubygems if you have a version older than 1.3.7, this can be done with this command:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Update Rubygems

```
sudo gem update --system
```

Installing RubyGems from Source

If you do not have RubyGems installed at all, or you are using Ubuntu, install it from source. *(why?)*

Install RubyGems

```
cd /tmp
wget http://production.cf.rubygems.org/rubygems/rubygems-1.8.10.tgz
tar zxf rubygems-1.8.10.tgz
cd rubygems-1.8.10
sudo ruby setup.rb --no-format-executable
```

Step 3: Install Chef and create the directories that are needed

Now that all the dependancies are installed, you can install the chef-client with this command:

```
sudo gem install chef
```

NOTE: as the message says, this could take a while. Be patient.

You can verify the installation was successful with:

```
% chef-client -v
```

You should see "Chef: 0.10.8" returned as the version number.

Building infrastructure with Chef is like managing source code, so one of the first things we need to do is build the repository you'll use. You will need git installed to follow this step.

```
cd ~
git clone https://github.com/opscode/chef-repo.git
```

Knife reads configuration from the .chef directory, we'll need to create that one as well:

```
mkdir -p ~/chef-repo/.chef
```

Copy the keys and knife configuration you downloaded earlier into this directory:

```
cp USERNAME.pem ~/chef-repo/.chef
cp ORGANIZATION-validator.pem ~/chef-repo/.chef
cp knife.rb ~/chef-repo/.chef
```

Replace USERNAME above with your Hosted Chef Username, and ORGANIZATION with the short-name you provided when you signed up.

Step 4: Connect to Hosted Chef

Run the following command to confirm knife is working with the Hosted Chef API.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
% cd ~/chef-repo
% knife client list
    ORGANIZATION-validator
```

The Workstation part of the install is now complete

You should now have a fully functional Management Workstation from which to interact with Chef and automate your infrastructure. The workstation is now able to download and upload recipes, and manage nodes with knife.

At this point you have two options.

1. You can either continue this guide and configure this workstation as a client so it can run recipes as well, or
2. You can setup chef-client on a separate node and use the workstation you just created to manage it remotely.

If you'd like to setup a separate node as a client so you can manage it remotely, please follow the Client Bootstrap Fast Start Guide or the EC2 Bootstrap Fast Start Guide. The rest of this guide assumes that you want to configure this workstation as a client.

Step 5: Configure the workstation as a client

You can configure chef-client with these commands:

```
% cd ~/chef-repo
% knife configure client ./client-config
Creating client configuration
Writing client.rb
Writing validation.pem
```

You'll now have a client-config directory in your local repository:

```
% ls -l client-config
-rw-r--r--   1 adam  staff   155 Jun 18 16:03 client.rb
-rw-r--r--@  1 adam  staff  1679 Jun 18 16:03 validation.pem
```

To configure your workstation as a chef client, you just need to copy this directory to /etc/chef:

```
sudo mkdir /etc/chef
sudo cp -r ~/chef-repo/client-config/* /etc/chef
```

You will now need to run chef-client for the first time, so your node is added to the client list:

```
% sudo chef-client
[Wed, 14 Sep 2011 19:24:59 -0700] INFO: *** Chef 0.10.x ***
[Wed, 14 Sep 2011 19:25:00 -0700] INFO: Client key /etc/chef/client.pem is not present - registering
[Wed, 14 Sep 2011 19:25:04 -0700] INFO: HTTP Request Returned 404 Not Found load node NODENAME
[Wed, 14 Sep 2011 19:25:07 -0700] INFO: Run List is []
... output truncated ...
[Wed, 14 Sep 2011 19:25:11 -0700] INFO: Report handlers complete
```

Once this is done, you can confirm this node was added with knife:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
% cd ~/chef-repo
% knife client list
    ORGANIZATION-validator
    NODENAME
```

Instead of NODENAME you should see the name of your node here.

The Client install is now complete

At this point, your node is now configured as a management workstation and a client. The next steps explain how to add a recipe and run it on this node. If you are already familiar with Chef, you can skip the rest of the guide and it will have no impact on your install.

Step 6: Download a Cookbook from the Community Site

Now that Chef is installed, we can add a recipe to the run list and run it on this node. We're going to use the `getting-started` cookbook as an example.

First, download the `getting-started` cookbook from the Community Site, and add it to the local repository in the cookbooks directory. The `knife cookbook site install` sub-command does this for you.

```
% cd ~/chef-repo
% knife cookbook site install getting-started
Installing getting-started to /home/username/chef-repo/.chef/../cookbooks
... output truncated ...
Cookbook getting-started version 0.3.0 successfully installed
```

Step 7: Upload a Cookbook to Hosted Chef

Next, upload the cookbook to Hosted Chef so it is available for systems that run the chef-client.

```
% knife cookbook upload getting-started
Uploading getting-started          [0.3.0]
upload complete
```

Step 8: Add a Cookbook to the run_list

Now you will want to add this cookbook to the run_list for this node. The run_list is a list of the recipes and roles that chef-client will run on the node.

You can add it with this command, substitute the name of your node for NODENAME:

```
% knife node run_list add NODENAME "recipe[getting-started]"
run_list: recipe[getting-started]
```

Step 9: Run chef-client

You can now run the chef-client on this node and it will run the recipe you've added:

```
% sudo chef-client
[Wed, 14 Sep 2011 19:31:48 -0700] INFO: *** Chef 0.10.x ***
[Wed, 14 Sep 2011 19:31:48 -0700] INFO: Run List is [recipe[getting-started]]
... output truncated ...
[Wed, 14 Sep 2011 19:31:52 -0700] INFO: Report handlers complete
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

The first thing Chef did was download the getting-started cookbook. Then it ran the default recipe, which then wrote out our template. Lets look at what it wrote:

```
% cat ~/chef-getting-started.txt
Welcome to Chef!

This is Chef version 0.10.x.
Running on ubuntu.
Version 10.04.
```

Your output will of course be dependent on the data we have stored about your node. You can also find this file it created by going to your home directory and opening chef-getting-started.txt.

You can also see what code the recipe contained by opening it with a text editor:

```
cd ~/chef-repo
vim cookbooks/getting-started/recipes/default.rb
```

When you are ready to start configuring this node, you can remove this test recipe from the run_list with this command:

```
% knife node run_list remove NODENAME "recipe[getting-started]"
run_list:
```

(For more information on editing and creating recipes, see the Cookbook Fast Start Guide.)

If something goes wrong

When you run chef-client, do you receive the error below?

You can resolve this by running the command as root. See **Running Chef Client with Elevated Privileges** for how to accomplish this.

Aside from that, it only takes a small change to a dependency or environment to cause these instructions to not be exact. If something goes wrong, first check the **Installation** page and refer to documents specific to your OS. If this doesn't help, head over to the **Support** page for information on where and how to find help from Opscode and the rest of the Chef community.

What should I look at next?

Another fast start...

Cookbook Fast Start Guide

Cookbooks are how things are distributed and shared in Chef. Continue on the fast start path and visit this page for a quick guide to writing and using Chef Cookbooks.

Perhaps detail on Chef and its use...

Chef Basics

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Learn some of the central concepts of configuration management benefits for your infrastructure.

1. We begin with an Architecture Introduction, covering the basic functions of the Chef Server, Nodes, and Chef Workstations and how these components communicate.
2. Then an overview of the Core Components, which introduces all of the aspects and components of modeling your infrastructure, Configuring Nodes and Managing Chef.
3. Onward to Cooking School and begin an Introduction to Cookbooks and More. Cookbooks are Chef's fundamental units of distribution, the way Chef users package up, distribute and share configuration information. Recipes, Resources, Attributes, Roles and more are also introduced.
4. The final basic section is an Introduction to Search and Data Bags, two of Chef's most powerful features allowing you to dynamically change the configuration of your infrastructure based on data.

Chef Architecture

With all that under your belt, it's time to tackle the dirty secrets of what's happening behind the scenes with Chef Architecture. We'll give you the scoop on Chef's Authentication and Authorization system and go over the Anatomy of a Chef Run, where we go in-depth with the process by which your systems get configured. From there we'll review all the executable parts of Chef - Chef Client, Chef Solo, Chef Server, Chef Indexer, and Server API and Cookbook Site API interaction.

Or view video tutorials and training...

Guides

We have a number of Walkthrough Guides available on the building of common stacks - including: Rails, Java, LAMP, and more. There are also How To Guides such as Deploying OpenStack with Chef, Guide to Creating A Cookbook and Writing A Recipe, and more - as well as a full 2.25 hour Webcast of a Chef Tutorial Session. These should be good resources for you as you move forward in the automation of your infrastructure.

 Installation

Cookbook Fast Start Guide 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Setup Opscode User and Organization



This page is to provide addition details on using Hosted Chef to setup a user and organization

You should already have Ruby installed on your system.
If you don't, return to the Fast Start Guide and complete that preceding step prior to returning here.

Signup and Account Validation

Head over to the Hosted Chef Sign Up page to create your account. You will get an email validation shortly after you sign up.
After you validate your email address with Hosted Chef, you need to do the following:

- Download your User Key, which you will use to interact with Hosted Chef directly.
- Create an organization on Hosted Chef
- Download your Organization Key, which will be used to associate new nodes with your Organization. (Also sometimes called a validation key).
- Download a Knife configuration file, which will configure our command line tools for you.

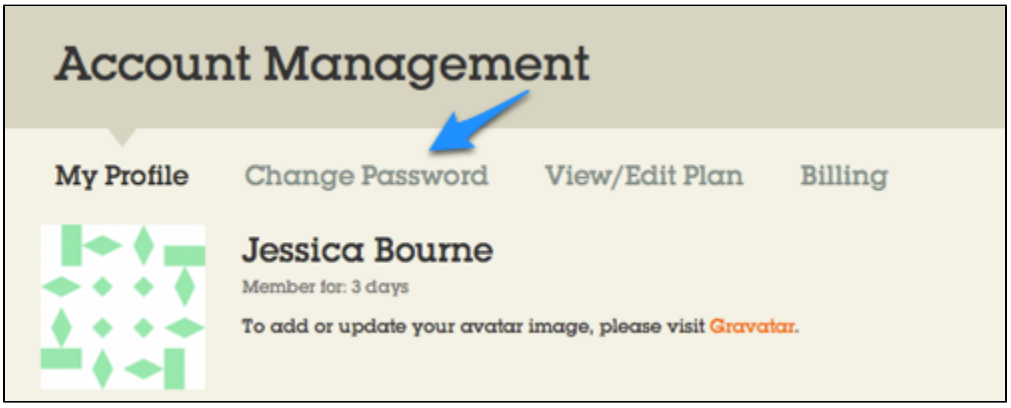
Each of these steps are detailed below. Place all of the files you download in this section in the same directory.

Download Your User Key

Login to the Management Console and navigate to your user profile page:

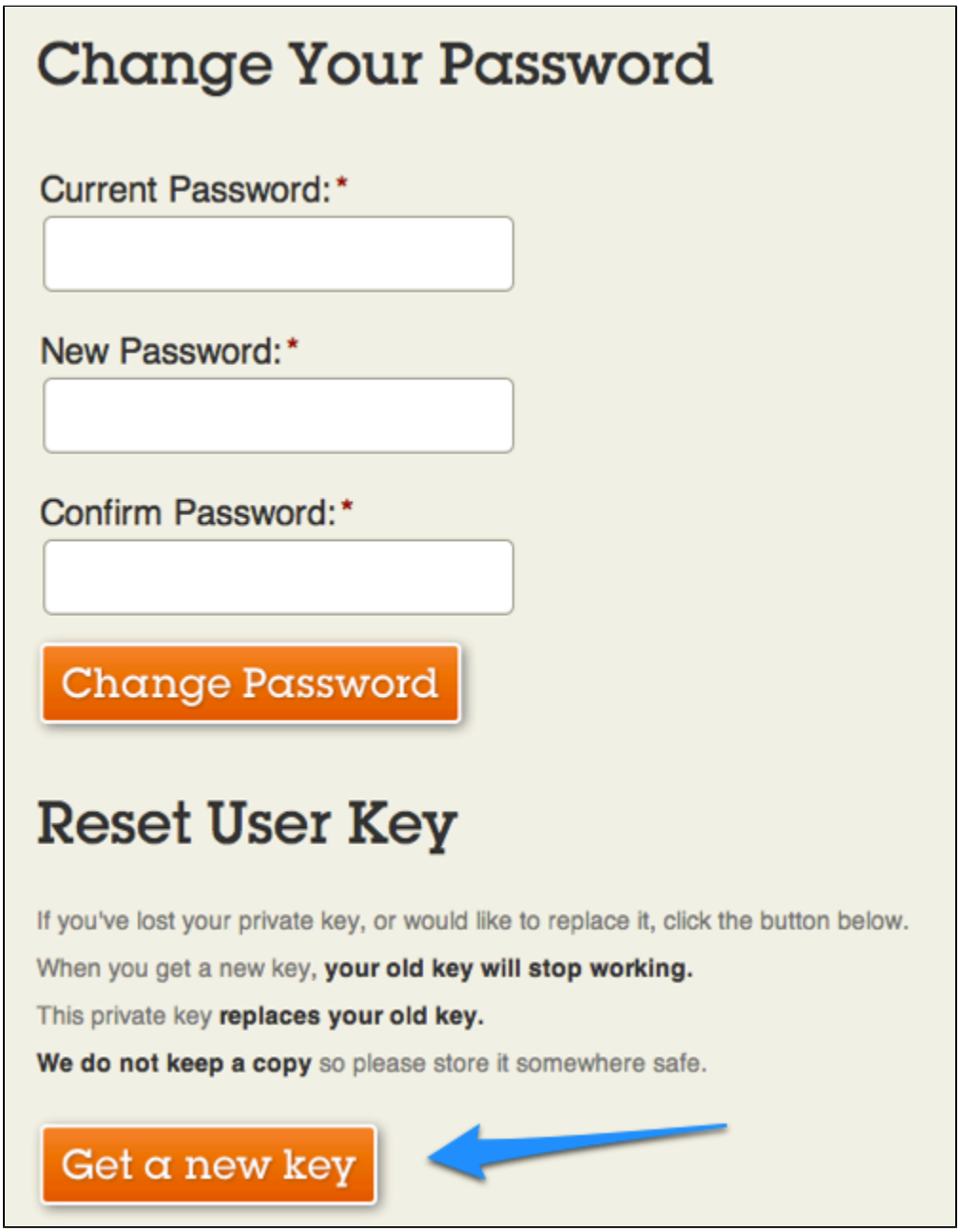


Follow the link to get a new User Key:



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

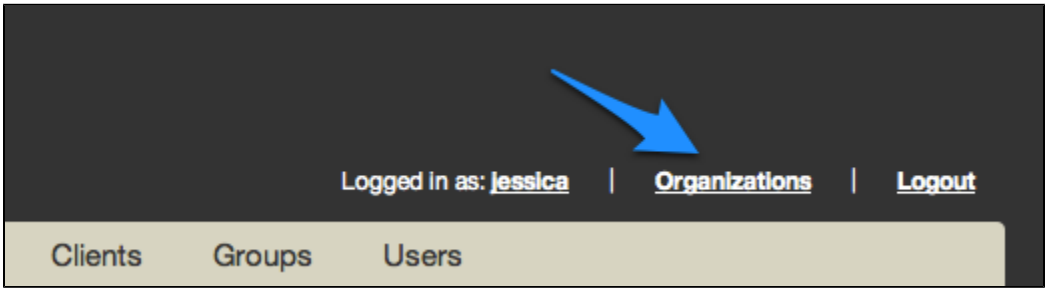
Download your new User Key:



Create an Organization

Login to the Management Console and navigate to the organizations page:

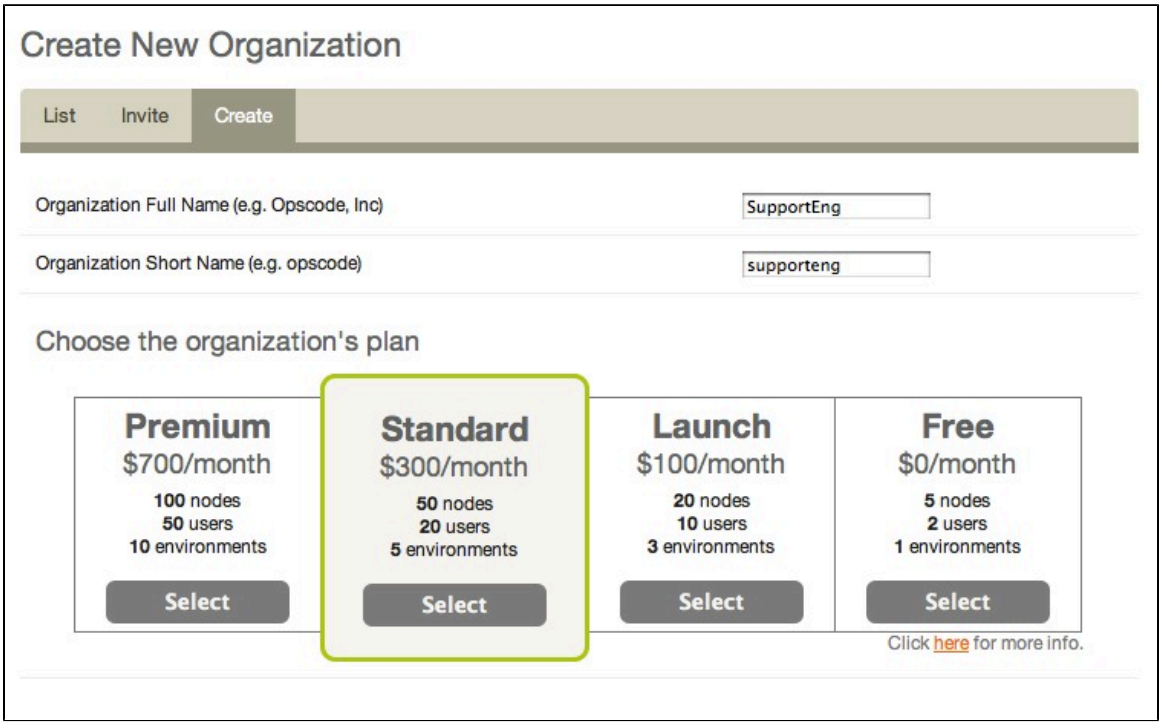
 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Click on the 'create' sub-tab:



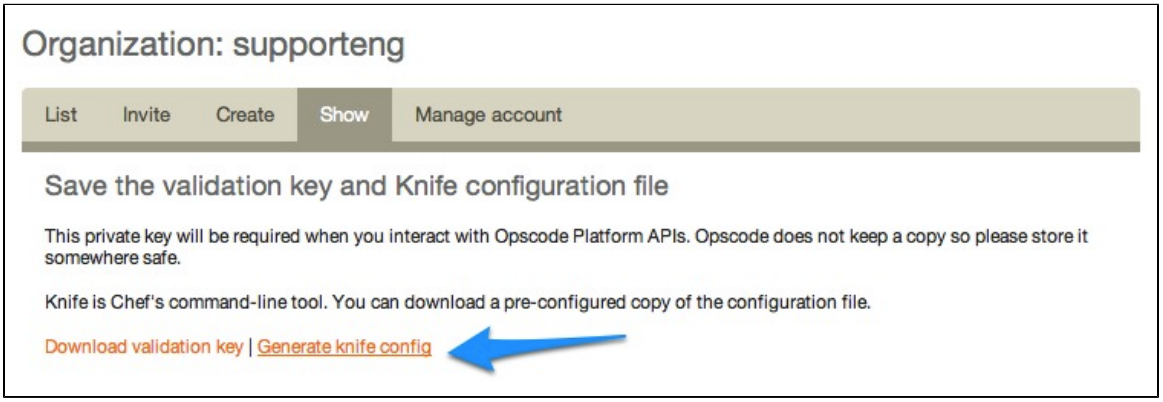
Enter a name for the organization and select a service plan:



Download Your Organization Key and Knife Configuration File

For the organization you would like to manage, download your Organization Key and your Knife Configuration File:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Opscode does not store copies of these keys! Keep them safe!



You now have a Hosted Chef user account and organization! Yea you!

return to the fast start guide

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Client Bootstrap Fast Start Guide



This installation guide describes how to get up and running with Chef as quickly as possible, and will walk you through adding a client node to your organization in Hosted Chef.

You can run your own Chef Server, but for this guide we will use Hosted Chef as the Chef Server, so you don't have to set one up.

- *Clients* are able to run chef-client, but will be unable to download, edit, and upload recipes with knife.
- *Management Workstations* are able to download, edit and upload recipes, and can be setup to work as clients as well so they can run chef-client.

You can find directions for setting up workstations in the Fast Start Guides:

- Fast Start Guide for Ubuntu
- Fast Start Guide for CentOS
- Fast Start Guide for Windows



Assumptions

In order to make this as quick as possible, we make some assumptions. *If your system does not meet these assumptions, you will need to use the Installation instructions that apply for the "flavor" of Chef you are installing.*

Hosted Chef

We're using Hosted Chef so we can get started right away without setting up a Chef Server. If you want to set up your own server instead, head over to the Installation page.

Operating System

Chef runs on many popular Unix and Linux platforms as well as Max OSX and Windows. *We will describe how to set up using **Ubuntu 10.04** as a client, but these are general directions that apply to most installations.* For OS specific instructions, go to the Installation page.

You already have a Management Workstation setup

In this guide we will be setting up a node as a client. We assume you already have another node setup as your Management Workstation.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

If you do not already have a workstation setup, you can find directions in the Fast Start Guides:

- Fast Start Guide for Ubuntu
- Fast Start Guide for CentOS
- Fast Start Guide for Windows

You'll want to follow one of these guides through at least step 4 and confirmed the knife commands worked properly. Once this is completed, you can return to this guide to setup another node as a client.

All of the commands in this guide will be ran from the Management Workstation by using knife commands to configure the client node remotely.

Steps

We will complete the following steps from the Management Workstation:

1. Bootstrap the client node
2. Verify the install completed
3. Manage recipes on the new client node
4. Run chef-client on the new client node remotely
5. Running chef-client as a daemon

When you are done with this guide you will have a node added as a client to the Hosted Chef organization, and the getting-started recipe will of been ran on it. You will also have a basic understanding of how you can manage this new client node remotely from your workstation.



Using Amazon's EC2 for your clients?
The EC2 Bootstrap Fast Start Guide goes over these same steps, and includes some additional steps that are specific to bootstrapping EC2 instances.

Step 1: Bootstrap the Ubuntu system

Use Knife Bootstrap to set up the target Ubuntu 10.04 client node with Ruby, Chef, and run Chef automatically. All of the steps in this guide are done from the management workstation you've already setup, the node must be accessible from the workstation via SSH. The base OS installation assumes that a user named ubuntu is created and has permission to run sudo. (See the Knife Bootstrap page for more information on how the sub-command works and other options available.)

Substitute IP_ADDRESS for the IP you want knife to SSH to, and PASSWORD for the sudo password for the ubuntu user:

```
% knife bootstrap IP_ADDRESS -x ubuntu -P PASSWORD --sudo
Bootstrapping Chef on IP_ADDRESS
... output truncated ...
192.168.1.27 Successfully installed chef-0.10.x
192.168.1.27 15 gems installed
192.168.1.27 [Fri, 16 Sep 2011 16:41:16 -0700] INFO: *** Chef 0.10.x ***
192.168.1.27 [Fri, 16 Sep 2011 16:41:16 -0700] INFO: Client key /etc/chef/client.pem is not present -
registering
192.168.1.27 [Fri, 16 Sep 2011 16:41:20 -0700] INFO: HTTP Request Returned 404 Not Found: Cannot load
node NODENAME
192.168.1.27 [Fri, 16 Sep 2011 16:41:22 -0700] INFO: Setting the run_list to [] from JSON
192.168.1.27 [Fri, 16 Sep 2011 16:41:22 -0700] INFO: Run List is []
192.168.1.27 [Fri, 16 Sep 2011 16:41:22 -0700] INFO: Run List expands to []
192.168.1.27 [Fri, 16 Sep 2011 16:41:22 -0700] INFO: Starting Chef Run for ubuntu-laptop
192.168.1.27 [Fri, 16 Sep 2011 16:41:22 -0700] INFO: Loading cookbooks []
192.168.1.27 [Fri, 16 Sep 2011 16:41:22 -0700] WARN: Node ubuntu-laptop has an empty run list.
192.168.1.27 [Fri, 16 Sep 2011 16:41:23 -0700] INFO: Chef Run complete in 1.488516 seconds
192.168.1.27 [Fri, 16 Sep 2011 16:41:23 -0700] INFO: Running report handlers
192.168.1.27 [Fri, 16 Sep 2011 16:41:23 -0700] INFO: Report handlers complete
```

You should see the current version of chef in the output of this command, Chef 0.10.8. Now chef-client has been installed on the node and it is

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

ready for recipes to be added to it. The rest of this guide will explain how to add and run a recipe remotely.



Using a private key for SSH authentication?
You can specify that you want to use a private key with the `-i` switch instead of sending it `-P`. For example you could use a command like this:

```
knife bootstrap IP_ADDRESS -x ubuntu -i ~/.ssh/id_rsa --sudo
```

With Chef 0.10.6+, you can also allow `knife bootstrap` to prompt you for a password. More information on the other options available with the `knife bootstrap` command can be found on the Knife Bootstrap wiki page.

Step 2: Verify the install completed

Now that the client has registered with Hosted Chef, it should show up on the node list. You can use knife to display a list of nodes:

```
% knife client list
    ORGANIZATION-validator
    NODENAME
```

Instead of NODENAME you should see the name of your node here.

The Client install is now complete

At this point, your node is now configured as a client. The next steps explain how to add a recipe and run it on this node remotely from the workstation, as well as run the client as a daemon. If you are already familiar with Chef, you can skip the rest of the guide and it will have no impact on your install.

Step 3: Manage recipes on the new client node

Now that your client is setup, you'll need to know how to add a recipe to it so you can start configuring it. This step will walk you through adding the getting-started recipe to it as an example. These steps are also ran from the management workstation.

We will need to download the getting-started cookbook from the Community Site:

```
% cd ~/chef-repo
% knife cookbook site install getting-started
Installing getting-started to /home/username/chef-repo/.chef/../cookbooks
... output truncated ...
Cookbook getting-started version 0.3.0 successfully installed
```

And then upload the recipe to Hosted Chef so it is available for our nodes:

```
% knife cookbook upload getting-started
Uploading getting-started          [0.3.0]
upload complete
```

Once this is completed, you can add this new recipe to the new nodes run list:

```
% knife node run_list add NODENAME 'recipe[getting-started]'
run_list:  recipe[getting-started]
```

Step 4: Run chef-client on the new client node

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

You can just SSH into the node at this point and run `sudo chef-client`, but it may be easier to run this command remotely from the workstation with knife. This way, once you have multiple nodes added you can run `chef-client` on all of them at once if needed by using a wildcard (*) as the NODENAME.

You can run chef-client remotely from the management workstation with knife with the SSH subcommand:

```
knife ssh name:NODENAME -x ubuntu -P PASSWORD "sudo chef-client"
```

If the node doesn't have a fully qualified domain name (FQDN) and is only accessible through a private IP address, you may need to specify that it uses the IP to SSH instead of the FQDN:

```
knife ssh name:NODENAME -a ipaddress -x ubuntu -P PASSWORD "sudo chef-client"
```

For more information on the knife SSH subcommand, review the Knife Built In Subcommands.

Running `chef-client` will run the getting-started recipe you've added to the client node's run_list. The getting-started recipe added a template to the client node in ~/chef-getting-started.txt. Once Chef has completed on the client, you can log into the client and look at the template:

```
% cat ~/chef-getting-started.txt
Welcome to Chef!

This is Chef version 0.10.x.
Running on ubuntu.
Version 10.04.
```

Your output will of course be dependent on the data we have stored about your node.

You can also see what code the recipe contained by opening it with a text editor from the Management Workstation:

```
cd ~/chef-repo
vim cookbooks/getting-started/recipes/default.rb
```

For more information on editing and creating recipes, see the Cookbook Fast Start Guide.

When you are ready to start configuring this node, you can remove this test recipe from the run_list with this command:

```
% knife node run_list remove NODENAME 'recipe[getting-started]'
run_list:
```

Step 5: Running chef-client as a daemon

You can setup chef-client to run as a daemon, so it will periodically run chef-client and get any updates you have submitted without needing to SSH to it every time. The easiest way to do this is with the chef-client cookbook. This will setup the chef-client as a daemon and add a time splay, you can add it the same way you added the getting-started recipe:

```
knife cookbook site install chef-client
knife cookbook upload chef-client
knife node run_list add NODENAME 'recipe[chef-client]'
```

Once it is added you'll need to run chef-client on the node once more for the recipe to configure the server:

```
knife ssh name:NODENAME -x ubuntu -P PASSWORD "sudo chef-client"
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Next time you bootstrap a new client node, you can include the `chef-client` cookbook in the Knife Bootstrap command from Step 1 if desired. This will add it to the run_list for the node and run it when installing Chef. You can do this with the `-r` switch:

```
knife bootstrap IP_ADDRESS -r 'recipe[chef-client]' -x ubuntu -P PASSWORD --sudo
```

You can read the cookbook's README to get more information on altering the default attributes.

If something goes wrong

It only takes a small change to a dependency or environment to cause these instructions to not be exact. If something goes wrong, first check the Installation page and refer to documents specific to your OS. If this doesn't help, head over to the **Support** page for information on where and how to find help from Opscode and the rest of the Chef community.

What should I look at next?

Another fast start...

Cookbook Fast Start Guide

Cookbooks are how things are distributed and shared in Chef. Continue on the fast start path and visit this page for a quick guide to writing and using Chef Cookbooks.

Perhaps detail on Chef and its use...

Chef Basics

Learn some of the central concepts of configuration management benefits for your infrastructure.

1. We begin with an Architecture Introduction, covering the basic functions of the Chef Server, Nodes, and Chef Workstations and how these components communicate.
2. Then an overview of the Core Components, which introduces all of the aspects and components of modeling your infrastructure, Configuring Nodes and Managing Chef.
3. Onward to Cooking School and begin an Introduction to Cookbooks and More. Cookbooks are Chef's fundamental units of distribution, the way Chef users package up, distribute and share configuration information. Recipes, Resources, Attributes, Roles and more are also introduced.
4. The final basic section is an Introduction to Search and Data Bags, two of Chef's most powerful features allowing you to dynamically change the configuration of your infrastructure based on data.

Chef Architecture

With all that under your belt, it's time to tackle the dirty secrets of what's happening behind the scenes with Chef Architecture. We'll give you the scoop on Chef's Authentication and Authorization system and go over the Anatomy of a Chef Run, where we go in-depth with the process by which your systems get configured. From there we'll review all the executable parts of Chef - Chef Client, Chef Solo, Chef Server, Chef Indexer, and Server API and Cookbook Site API interaction.

Or view video tutorials and training...

Guides

We have a number of Walkthrough Guides available on the building of common stacks - including: Rails, Java, LAMP, and more. There are also How To Guides such as Deploying OpenStack with Chef, Guide to Creating A Cookbook and Writing A Recipe, and more - as well as a full 2.25 hour Webcast of a Chef Tutorial Session. These should be good resources for you as you move forward in the automation of your infrastructure.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



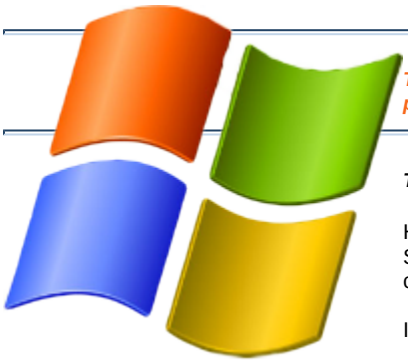
Installation

Cookbook Fast Start Guide



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Fast Start Guide for Windows



This page is an install guide that describes how to get up and running with Hosted Chef as quickly as possible and ends with a fast demonstration on how to work with cookbooks.

This Installation Guide uses the new Windows Full Installer.

Hosted and Private Chef customers should contact support@opscode.com with any issues. Open Source users may interact in open source support channels, open a Bug Report with project "CHEF", component "Packages", label "omnibus", if necessary.

If you want to install Chef manually instead, see the Installation page.

Assumptions

In order to make this as quick as possible, we make a few assumptions. *If your system does not meet these assumptions, you will need to use the Installation instructions that apply for the "flavor" of Chef you are installing.*

Hosted Chef

We're using Hosted Chef so we can get started right away without setting up a Chef Server. *If you want to set up your own server instead, head over to the Installation page.*

Operating System

Chef runs on many popular Unix and Linux platforms as well as Mac OSX and Windows. *We will describe how to set up using **Windows 2008 R2** as a workstation and a client, but these are general directions that will also apply to Windows 7.* These directions are known to not work for Windows XP and Server 2003 or on systems that have spaces in the %HOMEPATH%. For instructions on these, see the workstation installation page or the client installation page for Windows.

There is a Fast Start Guide for Ubuntu and a Fast Start Guide for CentOS which goes over these same instructions for Ubuntu 10.04 and CentOS 6.0. For other OS-specific instructions, go to the section of Installation that applies to the "flavor" of Chef you are installing, and proceed from there. (*See system requirements for details on what platforms Chef is known to run upon.*)

Management Workstation as a Client

In this guide we will be setting up a management workstation as a client. This means that the one node we set up here can be used to both manage your cookbooks and configuration in Hosted Chef, as well as use chef-client to run recipes on it. *(If you already have a workstation configured and just want to add a Linux or Mac client node, see the Client Bootstrap Fast Start Guide. Installation for Windows clients is described in the Installing Chef Client on Windows guide)*

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

The Difference Between a Management Workstation and a Client

- *Management Workstations* have git (or other version control system) installed so you can download and upload cookbooks with knife, and
- *Clients* have chef-client configured so Chef may be run on them.

If you'd like more information on these terms, the wiki page on Architecture Introduction has a very detailed explanation.

We will be setting up the client node as both in this guide.

- Once you add more nodes you will not need to set them all up as workstations.
- You do however, need at least one management workstation to manage your client nodes.

You do not have to set up your management workstation to be a chef-client. If you prefer not to:

- Follow this guide through Step 4, and then
- use the Client Bootstrap Fast Start Guide to configure a Linux or Mac client node separately, or use the Installing Chef Client on Windows guide to configure a Windows node separately.

Steps

This guide is going to direct you through the following steps:

1. Create a Hosted Chef account
2. Install Chef and Dependancies
3. Configure Chef and create directories needed
4. Connect to Hosted Chef
5. Configure the workstation as a client
6. Download a Cookbook from the Community Site
7. Upload the cookbook to Hosted Chef
8. Add the Cookbook to the run_list
9. Run chef-client with the new Cookbook

At the end of this guide you will have a workstation setup with chef-client installed, a recipe downloaded and added to the node's run_list. These steps end with the recipe being run on the node by Chef.

Step 1: Create a Hosted Chef account

Working with Chef is primarily done from a local workstation. Changes are made locally and uploaded to the Chef Server. Since we're using Hosted Chef as the Chef Server, you'll need to sign up for an account if you haven't already. (Don't worry! We won't charge you to use Hosted Chef for this guide - you can set up up to 5 nodes for free.)

Would you like to have screenshots walk you through the actions in this step instead? If so, head over to *Setup Opscode User and Organization* which will guide you through creating a new organization, and then return here to the next step once complete.

If you have already created your Organization, there are directions on how to recreate the organization key and download the knife config on the *Managing Organizations with the Hosted Chef Management Console* page and recreating existing user keys is explained on the *Managing Users with the Hosted Chef Management Console* page.

Head over to the Hosted Chef Sign Up page to create your account. You will get an email validation shortly after you sign up. After you complete the sign-up, create an organization in the Console page, and then download the following files:

- Your Organization validation key. This is used to automatically register new Chef Clients (like servers you manage).
- The Knife configuration file.
- Your User key. This is used to authenticate your user with Hosted Chef.

The private keys are not stored by Hosted Chef. Download them to your local system, we'll use them in Step #5.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Step 2: Install Chef and Dependencies

This Installation Guide uses the *new* Windows Full Installer.

It has been tested, but as it is newly released, Hosted and Private Chef customers should contact support@opscode.com with any issues. Open Source users may interact in open source support channels, opening a Bug Report with project "CHEF", component "Packages", label "omnibus", if necessary.

Install Chef

Download the Chef Full Installer for Windows and open it. Choose defaults for any options.

Once it is installed there will be no icon for it. If needed, you can confirm it was installed correctly with these commands in a new command prompt window:

```
C:\> tar --version
bsdtar 2.8.3 - libarchive 2.8.3

C:\> chef-client --version
```

You should see "Chef: 0.10.8" returned as the version number.

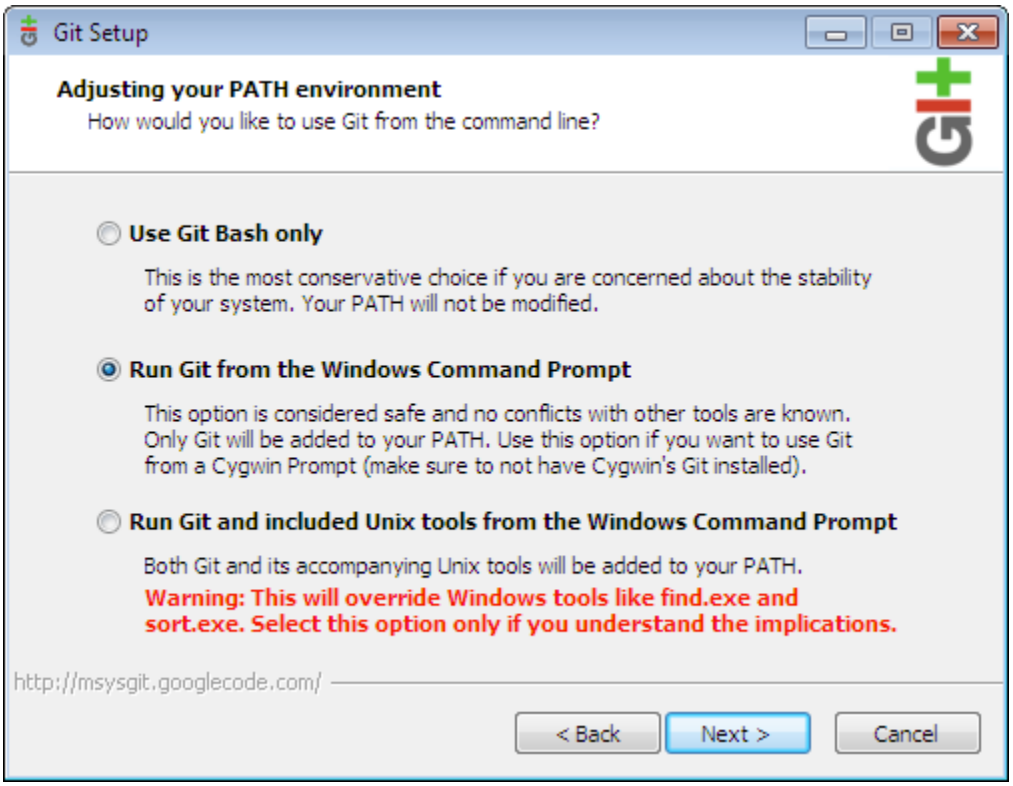
Install Git

Git isn't needed to set up a node as a client, but it will be needed to set up a node as a workstation and follow the steps on this page. *Workstation s* are able to download recipes from the Community Site, edit them, and them upload them to Hosted Chef. Once they are uploaded to the server any node that has been set up as a client or workstation will be able to use them. *Clients* do not need git installed but are unable to directly edit the recipes or upload changes to Hosted Chef without it. You will want at least one workstation set up to manage your organization. These directions walk you through setting up a workstation, for specific instructions on setting up a client instead, or using a workstation without git and tar on Windows, see the Installation page.

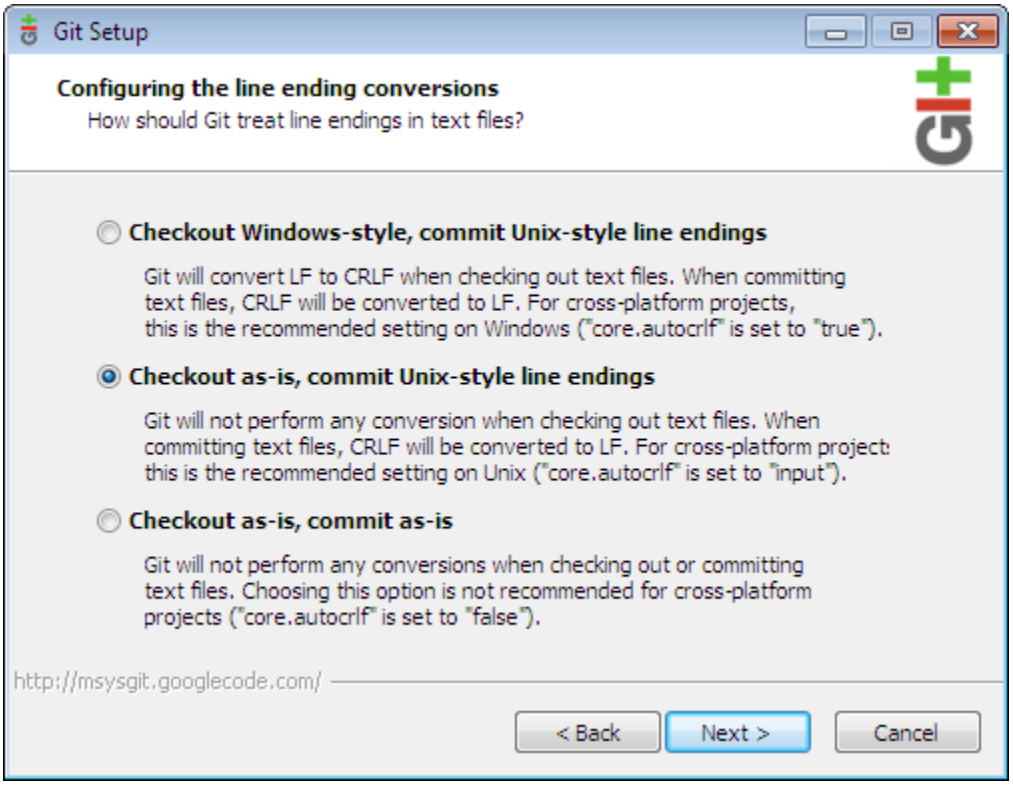
Download the Git Full installer for Windows and install it.

To make Git easily accessible from normal Windows Command Prompts, be sure to choose the 'Run Git from the Windows Command Prompt' option.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



For the line ending conversions option be sure to choose: 'Checkout as-is, commit Unix-style line endings'. Your fellow developers on *nix platforms will thank you!



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

If you'd like, you can confirm Git is now installed properly with this command in a new command prompt window:

```
C:\> git --version
git version 1.7.6.msysgit.0
```

Make sure that you close any command prompt windows after installing Git and Chef and re-open them, or some commands will not be available.

Step 3: Configure Chef and create the directories that are needed

Now that Chef and all the dependancies are installed, we will need to get the chef repository and set up the authentication files. You will see the variable %HOMEPATH% mentioned in these directions. You do not need to change this variable to an path and you can keep this as %HOMEPATH% in commands, Windows will know this is an alias for the path to your home directory.

Building infrastructure with Chef is like managing source code, so one of the first things we need to do is build the repository you'll use. You will need git installed to follow this step.

```
C:\> cd %HOMEPATH%
C:\Users\username> git clone git://github.com/opscode/chef-repo.git
```

Knife reads configuration from the .chef directory, so we'll need to make that directory:

```
C:\> mkdir %HOMEPATH%\chef-repo\.chef
```

Copy the keys and knife configuration you downloaded earlier into this directory. For example if they are in the 'C:\Users\<username>\Downloads' directory you could use these commands:

```
C:\> move %HOMEPATH%\Downloads\knife.rb %HOMEPATH%\chef-repo\.chef
            1 file(s) moved.
C:\> move %HOMEPATH%\Downloads\USERNAME.pem %HOMEPATH%\chef-repo\.chef
            1 file(s) moved.
C:\> move %HOMEPATH%\Downloads\ORGANIZATION-validator.pem %HOMEPATH%\chef-repo\.chef
            1 file(s) moved.
```

Replace USERNAME above with your Hosted Chef Username, and ORGANIZATION with the short-name you provided when you signed up.

Editing the knife.rb with the cookbook path

You will also need to edit the knife.rb to provide the full path to the cookbooks. You can do this by going to your home directory, entering the chef-repo directory, and then the .chef directory, and editing the knife.rb file. You will want to open it in Wordpad, and look for this line:

```
cookbook_path    ["#{current_dir}/../cookbooks"]
```

If you see this, you'll want to change it to the directory below or some knife commands will fail:

```
cookbook_path    ["#{ENV['HOME']}/chef-repo/cookbooks"]
```

Step 4: Connect to Hosted Chef

Run the following command to confirm knife is working with the Hosted Chef API.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
C:\> cd %HOMEPATH%\chef-repo
C:\Users\username\chef-repo> knife client list
    ORGANIZATION-validator
```

You should start out already in the home directory when you first launch command prompt, so normally you can just enter `cd chef-repo` to get to the correct directory. The `cd` command above will work no matter which directory you are in.

NOTE: knife commands will only work in the chef-repo directory unless you add `-c %HOMEPATH%\chef-repo\.chef\knife.rb` to the end of the command to specify where the knife config is located.

This command would work from any directory:

```
C:\> knife client list -c %HOMEPATH%\chef-repo\.chef\knife.rb
    ORGANIZATION-validator
```

The Workstation part of the install is now complete

You should now have a fully functional Management Workstation from which to interact with Chef and automate your infrastructure. The workstation is now able to download and upload recipes, and manage nodes with knife.

At this point you have two options. You can either continue this guide and configure this workstation as a client so it can run recipes as well, or you can set up chef-client on a separate node and use the workstation you just created to manage it remotely.

If you'd like to set up a separate Ubuntu client so you can manage it remotely, please follow the Client Bootstrap Fast Start Guide. If you'd like to set up a separate Windows node as a client, you could do this with the Knife Windows Bootstrap plugin or with the Installation guide for Windows clients. The rest of this guide assumes that you want to configure this workstation as a client.

Step 5: Configure the workstation as a client

You can configure chef-client with these commands:

```
C:\> cd %HOMEPATH%\chef-repo
C:\Users\username\chef-repo> knife configure client C:\chef
Creating client configuration
Writing client.rb
Writing validation.pem
```

You will need to edit the client.rb that's in the C:\chef directory to add a few lines to it, when you are done it should look like this. You will also want to replace ORGANIZATION here will your organization's name.

```
log_level        :info
log_location     STDOUT
chef_server_url  'https://api.opscode.com/organizations/ORGANIZATION'
validation_client_name 'ORGANIZATION-validator'
validation_key  "C:/chef/validation.pem"
client_key  "C:/chef/client.pem"
```

- Note that this file uses forward slashes [/] and not backslashes [\ ].

You will now need to run chef-client for the first time, so your node is added to the client list:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
C:/> chef-client -c C:\chef\client.rb
[Wed, 14 Sep 2011 19:24:59 -0700] INFO: *** Chef 0.10.x ***
[Wed, 14 Sep 2011 19:25:00 -0700] INFO: Client key C:/chef/client.pem is not present - registering
[Wed, 14 Sep 2011 19:25:04 -0700] INFO: HTTP Request Returned 404 Not Found load node NODENAME
[Wed, 14 Sep 2011 19:25:07 -0700] INFO: Run List is []
... output truncated ...
[Wed, 14 Sep 2011 19:25:11 -0700] INFO: Report handlers complete
```

Currently you need to specify the path to the client.rb with `-c` due to bug CHEF-2317. If you are already in the C:\chef directory, you can specify the location with `-c client.rb` instead of needing to provide the full path.

Once this is done, you can confirm this node was added with knife:

```
C:\> cd %HOMEPATH%\chef-repo
C:\Users\username\chef-repo> knife client list
    ORGANIZATION-validator
    NODENAME
```

Instead of NODENAME you should see the name of your node here.

The Client install is now complete

At this point, your node is now configured as a management workstation and a client. The next steps explain how to add a recipe and run it on this node. If you are already familiar with Chef, you can skip the rest of the guide and it will have no impact on your install.

Step 6: Download a Cookbook from the Community Site

Now that Chef is installed, we can add a recipe to the run list and run it on this node. We're going to use the `getting-started` cookbook as an example.

First, download the `getting-started` cookbook from the Community Site, and add it to the local repository in the cookbooks directory. Due to bug CHEF-2250 we need to use the `knife cookbook site download` command to do this instead of `knife cookbook site install`.

```
C:\> cd %HOMEPATH%\chef-repo\cookbooks
C:\Users\username\chef-repo\cookbooks> knife cookbook site download getting-started
C:\Users\username\chef-repo\cookbooks> tar zxvf getting-started-0.3.0-tar.gz
C:\Users\username\chef-repo\cookbooks> del getting-started-0.3.0-tar.gz
```

These commands download the cookbook tar file to the current directory, and untar the archive (similar to unziping for .zip files) which creates a getting-started directory. Then we can delete the tar file with `del` as it is no longer needed afterwards.

Step 7: Upload a Cookbook to Hosted Chef

Next, upload the cookbook to Hosted Chef so it is available for systems that run the chef-client. First, we will need to move back to the chef-repo as knife commands will not work without `-c` in other directories.

```
C:\> cd %HOMEPATH%\chef-repo
C:\Users\username\chef-repo> knife cookbook upload getting-started
Uploading getting-started          [0.3.0]
upload complete
```

Step 8: Add a Cookbook to the run_list

Now you will want to add this cookbook to the run_list for this node. The run_list is a list of the recipes and roles that chef-client will run on the node.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

You can add it with this command, substitute the name of your node for NODENAME:

```
C:\Users\username\chef-repo> knife node run_list add NODENAME 'recipe[getting-started]'
run_list: recipe[getting-started]
```

Step 9: Run chef-client

You can now run the chef-client on this node and it will run the recipe you've added:

```
C:/> chef-client -c C:\chef\client.rb
[Wed, 14 Sep 2011 19:31:48 -0700] INFO: *** Chef 0.10.x ***
[Wed, 14 Sep 2011 19:31:48 -0700] INFO: Run List is [recipe[getting-started]]
... output truncated ...
[Wed, 14 Sep 2011 19:31:52 -0700] INFO: Report handlers complete
```

The first thing Chef did was download the getting-started cookbook. Then it ran the default recipe, which then wrote out our template which is located in the home directory as chef-getting-started.txt. Lets look at what it wrote:

```
Welcome to Chef!

This is Chef version 0.10.x.
Running on windows.
Version 6.1.7600.
```

Your output will of course be dependent on the data we have stored about your node.

You can also see what code the recipe contained by opening it with a text editor. It will be located in your home directory, in chef-repo\cookbooks\getting-started\recipes\default.rb. You can open it in Wordpad if you do not have an editor installed.

For more information on editing and creating recipes, see the Cookbook Fast Start Guide.

When you are ready to start configuring this node, you can remove this test recipe from the run_list with this command:

```
C:\Users\username\chef-repo> knife node run_list remove NODENAME 'recipe[getting-started]'
run_list:
```

If something goes wrong

When you run chef-client, does it appear to run successfully, but when you go to review the changes - none have been made?

One possible reason is that the chef-client had insufficient privileges. See **Running Chef Client with Elevated Privileges** for how to address this.

Aside from that, it only takes a small change to a dependency or environment to cause these instructions to not be exact. If something goes wrong, first check the Installation page and refer to documents specific to your OS. If this doesn't help, head over to the **Support** page for information on where and how to find help from Opscode and the rest of the Chef community.

Some Resources/Providers Are Known To Not Work

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Not all Resources work on Windows yet. The following Resources do not work on Windows yet:

- Cron
- Deploy
- Erlang call
- Http request
- Ifconfig
- Link
- Mdadm
- Package (but gem_package - for ruby gems, does work)
- Route
- Script
- Subversion

Also, using 'user' or 'group' attributes within a resource, specifically the template or directory resource is known to cause issues. Related bug: CHEF-2633

What should I look at next?

Another fast start...

Cookbook Fast Start Guide

Cookbooks are how things are distributed and shared in Chef. Continue on the fast start path and visit this page for a quick guide to writing and using Chef Cookbooks.

Perhaps detail on Chef and its use...

Chef Basics

Learn some of the central concepts of configuration management benefits for your infrastructure.

1. We begin with an Architecture Introduction, covering the basic functions of the Chef Server, Nodes, and Chef Workstations and how these components communicate.
2. Then an overview of the Core Components, which introduces all of the aspects and components of modeling your infrastructure, Configuring Nodes and Managing Chef.
3. Onward to Cooking School and begin an Introduction to Cookbooks and More. Cookbooks are Chef's fundamental units of distribution, the way Chef users package up, distribute and share configuration information. Recipes, Resources, Attributes, Roles and more are also introduced.
4. The final basic section is an Introduction to Search and Data Bags, two of Chef's most powerful features allowing you to dynamically change the configuration of your infrastructure based on data.

Chef Architecture

With all that under your belt, it's time to tackle the dirty secrets of what's happening behind the scenes with Chef Architecture. We'll give you the scoop on Chef's Authentication and Authorization system and go over the Anatomy of a Chef Run, where we go in-depth with the process by which your systems get configured. From there we'll review all the executable parts of Chef - Chef Client, Chef Solo, Chef Server, Chef Indexer, and Server API and Cookbook Site API interaction.

Or view video tutorials and training...

Guides

We have a number of Walkthrough Guides available on the building of common stacks - including: Rails, Java, LAMP, and more. There are also How To Guides such as Deploying OpenStack with Chef, Guide to Creating A Cookbook and Writing A Recipe, and more - as well as a full 2.25 hour Webcast of a Chef Tutorial Session. These should be good resources for you as you move forward in the automation of your infrastructure.

Knife Windows Bootstrap

The knife windows plugin may be used to bootstrap new Windows nodes as clients. The plugin may also be used from Mac or Linux workstations.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

You can install it from Rubygems by using this command:

```
gem install knife-windows
```

There are also plugins for managing EC2 or Rackspace nodes. For a more complete list see Community Plugins.

 Installation

Cookbook Fast Start Guide 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Fast Start Guide for CentOS



This page is an install guide that describes how to get up and running with Hosted Chef as quickly as possible and ends with a fast demonstration on how to work with cookbooks.

Assumptions

In order to make this as quick as possible, we make some assumptions. *If your system does not meet these assumptions, you will need to use the Installation instructions that apply for the "flavor" of Chef you are installing.*

Hosted Chef

We're using Hosted Chef so we can get started right away without setting up a Chef Server. If you want to set up your own server instead, head over to the Installation page.

Operating System



Chef runs on many popular Unix and Linux platforms as well as Max OSX and Windows. *We will describe how to set up using **CentOS 6.0** as a workstation and a client, but these are general directions that apply to most installations.*

There is a Fast Start Guide for Ubuntu and a Fast Start Guide for Windows which goes over these same directions for Ubuntu 10.04 and Windows 2008 R2. For other OS specific instructions, go to the section of Installation that applies to the "flavor" of Chef you are installing, and proceed from there.

Management Workstation as a Client

In this guide we will be setting up a management workstation as a client. This means that the one node we setup here can be used to both manage your cookbooks and configuration in Hosted Chef, as well as use chef-client to run recipes on it. *(If you already have a workstation configured and just want to add a client node, see the Client Bootstrap Fast Start Guide .)*

The Difference Between a Management Workstation and a Client

- *Management Workstations* have git installed so you can download and upload cookbooks with knife, and
- *Clients* have chef-client configured so Chef can be run on them.

If you'd like more information on these terms, the wiki page on Architecture Introduction has a very detailed explanation.

We will be setting up the client node as both in this guide.

- Once you add more nodes you will not need to set them all up as workstations.
- You do however need at least one management workstation to manage your client nodes.

You don't however have to set up your management workstation to be a chef-client. If you prefer not to:

- Follow this guide through Step 4, and then
- use the Client Bootstrap Fast Start Guide to configure a client node separately.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

System Requirements

Chef-client is supported on the following platforms

- Ubuntu (10.04, 10.10, 11.04)
- Debian (5.0, 6.0)
- RHEL & CentOS (5.x, 6.x)
- Fedora 10+
- Mac OS X (10.4, 10.5, 10.6)
- Windows 7
- Windows Server 2003 R2, 2008 R2

Additionally, chef-client is known to run on the following platforms

- Ubuntu (6.06, 8.04-9.10)
- Gentoo (11.1,11.2)
- FreeBSD (7.1)
- OpenBSD (4.4)
- OpenSolaris (2008.11)
- Solaris 5.10 (u6)
- Windows XP, Vista

We will be installing (or updating) the following in this Installation Guide.

Ruby
Ruby 1.8.5, 1.8.6, 1.8.7, 1.9.1 or 1.9.2 with SSL bindings is required.

RubyGems
Version 1.3.7 or greater. On CentOS Rubygems should be installed from source.

Chef Client Version
Hosted Chef is compatible with chef-client v0.9.0 and greater. Older clients need to be upgraded before connecting.

Steps

This guide is going to direct you through the following steps:

1. Create a Hosted Chef account
2. Install and Update dependencies - ruby, ruby gems, ruby-dev and git-core
3. Install Chef and create directories needed
4. Connect to Hosted Chef
5. Configure the workstation as a client
6. Download a Cookbook from the Community Site
7. Upload the cookbook to Hosted Chef
8. Add the Cookbook to the run_list
9. Run chef-client with the new Cookbook

At the end of this guide you will have a workstation setup with chef-client installed, a recipe downloaded and added to the node's run_list. These steps end with the recipe being run on the node by Chef.

Step 1: Create a Hosted Chef account

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Working with a Chef is primarily done from a local workstation. Changes are made locally and uploaded to the Chef Server. Since we're using Hosted Chef as the Chef Server, you'll need to sign up for an account if you haven't already. (Don't worry! We won't charge you to use Hosted Chef for this guide - you can setup up to 5 nodes for free.)

> **Would you like to have screenshots walk you through the actions in this step instead?** If so, head over to *Setup Opscode User and Organization* which will guide you through creating a new organization, and then return here to the next step once complete.
>
> If you have already created your Organization, there are directions on how to recreate the organization key and download the knife config on the *Managing Organizations with the Hosted Chef Management Console* page and recreating existing user keys is explained on the *Managing Users with the Hosted Chef Management Console* page.

Sign up for a Hosted Chef account. You will get an email validation shortly after you sign up. After you complete the sign-up, create an organization in the Console page, and then download the following files:

- Your Organization validation key. This is used to automatically register new Chef Clients (like servers you manage).
- The Knife configuration file.
- Your User key. This is used to authenticate your user with Hosted Chef.

The private keys are not stored by Hosted Chef. Download them to your local system, we'll use them in Step #5.

Step 2: Install and Update Dependencies

We're going to use Ruby 1.8.7, the default available on our chosen operating systems. We also need to install the ruby-devel package, as well as git and a few other dependencies. We will install RubyGems from source.

Below is an example of how you would install these in CentOS 6.0. For OS specific instructions, see the Installation page.

Install the RBEL repo

```
sudo rpm -Uvh http://rbel.frameos.org/rbel6
```

Install Ruby and Development Tools

```
sudo yum install ruby ruby-devel ruby-ri ruby-rdoc ruby-shadow gcc gcc-c++ automake autoconf make curl
dmidecode
```

> **Chef 0.10 Note**
> There are open issues with Chef 0.10.x and Ruby 1.8.6. If you are using Chef 0.10.x, you should get 1.8.7, 1.9.1, or 1.9.2 with SSL bindings.

Install Git

Git isn't needed to setup a node as a client, but it will be needed to setup a node as a workstation and follow the steps on this page. *Workstations* are able to download recipes from the Community Site, edit them, and them upload them to Hosted Chef. Once they are uploaded to the server any node that has been setup as a client or workstation will be able to use them. *Clients* do not need git installed but are unable to directly edit the recipes or upload changes to Hosted Chef without it. You will want at least one workstation setup to manage your organization. These directions walk you through setting up a workstation, for specific instructions on setting up a client instead, see the Installation page.

First you will want to install the rpm from the Extra Packages for Enterprise Linux (EPEL) yum repository:

```
sudo rpm -Uvh http://download.fedora.redhat.com/pub/epel/6/i386/epel-release-6-5.noarch.rpm
```

Then you can use yum to install git:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
sudo yum install -y git
```

Install RubyGems 1.3.7+

First, check to see if your operating system already has RubyGems installed and is at least version 1.3.7 by running the following:

```
% gem -v
1.3.7
```

If you're using CentOS, we recommend installing RubyGems from source rather than use the OS-provided version (if any), as it is cross platform, so we know what to expect. If you have 1.3.7 installed already, you have completed installing the dependencies needed for chef-client and you can move on to Step 3: Install Chef.

If you have an older version

You will need to update rubygems if you have a version older than 1.3.7, this can be done with this command:

Update Rubygems

```
sudo gem update --system
```

Installing RubyGems from Source

If you do not have RubyGems installed at all, install it from source. *(why?)*

Install RubyGems

```
cd /tmp
curl -O http://production.cf.rubygems.org/rubygems/rubygems-1.8.10.tgz
tar zxf rubygems-1.8.10.tgz
cd rubygems-1.8.10
sudo ruby setup.rb --no-format-executable
```

Step 3: Install Chef and create the directories that are needed

Now that all the dependancies are installed, you can install the chef-client with this command:

```
sudo gem install chef --no-ri --no-rdoc
```

NOTE: as the message says, this could take a while. Be patient.

You can verify the installation was successful with:

```
% chef-client -v
Chef: 0.10.8
```

Building infrastructure with Chef is like managing source code, so one of the first things we need to do is build the repository you'll use. You will need git installed to follow this step.

```
cd ~
git clone https://github.com/opscode/chef-repo.git
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Knife reads configuration from the .chef directory, we'll need to create that one as well:

```
mkdir -p ~/chef-repo/.chef
```

Copy the keys and knife configuration you downloaded earlier into this directory:

```
cp USERNAME.pem ~/chef-repo/.chef
cp ORGANIZATION-validator.pem ~/chef-repo/.chef
cp knife.rb ~/chef-repo/.chef
```

Replace USERNAME above with your Hosted Chef Username, and ORGANIZATION with the short-name you provided when you signed up.

Step 4: Connect to Hosted Chef

Run the following command to confirm knife is working with the Hosted Chef API.

```
% cd ~/chef-repo
% knife client list
    ORGANIZATION-validator
```

The Workstation part of the install is now complete

You should now have a fully functional Management Workstation from which to interact with Chef and automate your infrastructure. The workstation is now able to download and upload recipes, and manage nodes with knife.

At this point you have two options. You can either continue this guide and configure this workstation as a client so it can run recipes as well, or you can setup chef-client on a separate node and use the workstation you just created to manage it remotely.

If you'd like to setup a separate node as a client so you can manage it remotely, please follow the Client Bootstrap Fast Start Guide. The rest of this guide assumes that you want to configure this workstation as a client.

Step 5: Configure the workstation as a client

You can configure chef-client with these commands:

```
% cd ~/chef-repo
% knife configure client ./client-config
Creating client configuration
Writing client.rb
Writing validation.pem
```

You'll now have a client-config directory in your local repository:

```
% ls -l client-config
-rw-r--r--   1 adam  staff   155 Jun 18 16:03 client.rb
-rw-r--r--@  1 adam  staff  1679 Jun 18 16:03 validation.pem
```

To configure your workstation as a chef client, you just need to copy this directory to /etc/chef. First, make the directory:

```
sudo mkdir /etc/chef
```

Then copy the files from client-config into this new directory:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
sudo cp -r ~/chef-repo/client-config/* /etc/chef
```

You will now need to run chef-client for the first time, so your node is added to the client list:

```
% sudo chef-client
[Wed, 14 Sep 2011 19:24:59 -0700] INFO: *** Chef 0.10.8 ***
[Wed, 14 Sep 2011 19:25:00 -0700] INFO: Client key /etc/chef/client.pem is not present - registering
[Wed, 14 Sep 2011 19:25:04 -0700] INFO: HTTP Request Returned 404 Not Found load node NODENAME
[Wed, 14 Sep 2011 19:25:07 -0700] INFO: Run List is []
... output truncated ...
[Wed, 14 Sep 2011 19:25:11 -0700] INFO: Report handlers complete
```

Once this is done, you can confirm this node was added with knife:

```
% cd ~/chef-repo
% knife client list
    ORGANIZATION-validator
    NODENAME
```

Instead of NODENAME you should see the name of your node here.

The Client install is now complete

At this point, your node is now configured as a management workstation and a client. The next steps explain how to add a recipe and run it on this node. If you are already familiar with Chef, you can skip the rest of the guide and it will have no impact on your install.

Step 6: Download a Cookbook from the Community Site

Now that Chef is installed, we can add a recipe to the run list and run it on this node. We're going to use the `getting-started` cookbook as an example.

First, download the `getting-started` cookbook from the Community Site, and add it to the local repository in the cookbooks directory. The `knife cookbook site install` sub-command does this for you.

```
% cd ~/chef-repo
% knife cookbook site install getting-started
Installing getting-started to /home/username/chef-repo/.chef/../cookbooks
... output truncated ...
Cookbook getting-started version 0.3.0 successfully installed
```

Step 7: Upload a Cookbook to Hosted Chef

Next, upload the cookbook to Hosted Chef so it is available for systems that run the chef-client.

```
% knife cookbook upload getting-started
Uploading getting-started          [0.3.0]
upload complete
```

Step 8: Add a Cookbook to the run_list

Now you will want to add this cookbook to the run_list for this node. The run_list is a list of the recipes and roles that chef-client will run on the node.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

You can add it with this command, substitute the name of your node for NODENAME:

```
% knife node run_list add NODENAME "recipe[getting-started]"
run_list: recipe[getting-started]
```

Step 9: Run chef-client

You can now run the chef-client on this node and it will run the recipe you've added:

```
% sudo chef-client
[Wed, 14 Sep 2011 19:31:48 -0700] INFO: *** Chef 0.10.8 ***
[Wed, 14 Sep 2011 19:31:48 -0700] INFO: Run List is [recipe[getting-started]]
... output truncated ...
[Wed, 14 Sep 2011 19:31:52 -0700] INFO: Report handlers complete
```

The first thing Chef did was download the getting-started cookbook. Then it ran the default recipe, which then wrote out our template. Lets look at what it wrote:

```
% sudo cat /root/chef-getting-started.txt
Welcome to Chef!

This is Chef version 0.10.8.
Running on centos.
Version 6.0.
```

Your output will of course be dependent on the data we have stored about your node. You can also find this file it created by going to your home directory and opening chef-getting-started.txt.

You can also see what code the recipe contained by opening it with a text editor:

```
cd ~/chef-repo
vim cookbooks/getting-started/recipes/default.rb
```

When you are ready to start configuring this node, you can remove this test recipe from the run_list with this command:

```
% knife node run_list remove NODENAME "recipe[getting-started]"
run_list:
```

(For more information on editing and creating recipes, see the Cookbook Fast Start Guide.)

If something goes wrong

When you run chef-client, do you receive the error below?

```
```

You can resolve this by running the command as root. See **Running Chef Client with Elevated Privileges** for how to accomplish this.

Aside from that only takes a small change to a dependency or environment to cause these instructions to not be exact. If something goes wrong, first check the Installation page and refer to documents specific to your OS. If this doesn't help, head over to the **Support** page for information on where and how to find help from Opscode and the rest of the Chef community.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

What should I look at next?

Another fast start...

Cookbook Fast Start Guide

Cookbooks are how things are distributed and shared in Chef. Continue on the fast start path and visit this page for a quick guide to writing and using Chef Cookbooks.

Perhaps detail on Chef and its use...

Chef Basics

Learn some of the central concepts of configuration management benefits for your infrastructure.

1. We begin with an Architecture Introduction, covering the basic functions of the Chef Server, Nodes, and Chef Workstations and how these components communicate.
2. Then an overview of the Core Components, which introduces all of the aspects and components of modeling your infrastructure, Configuring Nodes and Managing Chef.
3. Onward to Cooking School and begin an Introduction to Cookbooks and More. Cookbooks are Chef's fundamental units of distribution, the way Chef users package up, distribute and share configuration information. Recipes, Resources, Attributes, Roles and more are also introduced.
4. The final basic section is an Introduction to Search and Data Bags, two of Chef's most powerful features allowing you to dynamically change the configuration of your infrastructure based on data.

Chef Architecture

With all that under your belt, it's time to tackle the dirty secrets of what's happening behind the scenes with Chef Architecture. We'll give you the scoop on Chef's Authentication and Authorization system and go over the Anatomy of a Chef Run, where we go in-depth with the process by which your systems get configured. From there we'll review all the executable parts of Chef - Chef Client, Chef Solo, Chef Server, Chef Indexer, and Server API and Cookbook Site API interaction.

Or view video tutorials and training...

Guides

We have a number of Walkthrough Guides available on the building of common stacks - including: Rails, Java, LAMP, and more. There are also How To Guides such as Deploying OpenStack with Chef, Guide to Creating A Cookbook and Writing A Recipe, and more - as well as a full 2.25 hour Webcast of a Chef Tutorial Session. These should be good resources for you as you move forward in the automation of your infrastructure.

 Installation

Cookbook Fast Start Guide 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

EC2 Bootstrap Fast Start Guide



This installation guide describes how to get up and running with Chef as quickly as possible, and will walk you through adding a client node on Amazon's EC2 to your organization in Hosted Chef.

You can run your own Chef Server, but for this guide we will use Hosted Chef as the Chef Server, so you don't have to set one up.

- *Clients* are able to run chef-client, but will be unable to download, edit, and upload recipes with knife.
- *Management Workstations* are able to download, edit and upload recipes, and can be setup to work as clients as well so they can run chef-client.

You can find directions for setting up workstations in the Fast Start Guides:

- Fast Start Guide for Ubuntu
- Fast Start Guide for CentOS
- Fast Start Guide for Windows



Assumptions

In order to make this as quick as possible, we make some assumptions. *If your system does not meet these assumptions, you will need to use the Installation instructions that apply for the "flavor" of Chef you are installing.*

You already have a Management Workstation setup

In this guide we will be creating an EC2 Instance and setting it up as a client. We assume you already have another node setup as your Management Workstation.

If you do not already have a workstation setup, you can find directions in the Fast Start Guides:

- Fast Start Guide for Ubuntu
- Fast Start Guide for CentOS
- Fast Start Guide for Windows

You'll want to follow one of these guides through at least step 4 and confirmed the knife commands worked properly. Once this is completed, you

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

can return to this guide to setup another node as a client.

All of the commands in this guide will be ran from the Management Workstation by using knife commands to configure the client node remotely.

Amazon's EC2

This guide assumes you are using AWS EC2 to setup instances. You should already have an account created with AWS, and this guide assumes that you do not have the EC2 instance setup yet. If you've already created the EC2 instance, you can either delete it or use the regular Client Bootstrap Fast Start Guide to configure it.

Operating System

Chef runs on many popular Unix and Linux platforms as well as Max OSX and Windows.

- **Management Workstation OS:** We will describe how to set up using **Ubuntu 10.04** as the workstation by default, but these are general directions that will work on most workstations.
- **Client EC2 Instance OS:** There are directions included on setting up the EC2 client as Ubuntu, Debian, RHEL, or CentOS but as long as you have an AMI and a custom bootstrap file that works you can use any OS as the EC2 client.

Steps

We will complete the following steps from the Management Workstation:

1. Install the Knife-EC2 Plugin
2. Configure AWS settings and the knife.rb file
3. Bootstrap the EC2 Instance
4. Verify the install completed
5. Manage recipes on the new client node
6. Run chef-client on the new client node remotely
7. Running chef-client as a daemon

When you are done with this guide you will have a workstation setup to work with EC2. You will also have a node on EC2 added as a client to the Hosted Chef organization and AWS, and the getting-started recipe will of been ran on it. It will also explain the basics of how you can manage this new client node remotely from your workstation.

For more in-depth information, please review the Launch Cloud Instances with Knife page.

Step 1: Install the Knife-EC2 Plugin

If you have already installed and configured the knife-ec2 plugin then you can skip to Step 3. All of the steps in this guide are done from the management workstation you've already setup.

Install dependancies

First you will need to install the dependancies needed for the knife-ec2 plugin on your Ubuntu management workstation where you use knife:

```
sudo apt-get update
sudo apt-get install ruby1.8-dev ruby1.8 ri1.8 rdoc1.8 irb1.8
sudo apt-get install libreadline-ruby1.8 libruby1.8 libopenssl-ruby
sudo apt-get install libxslt-dev libxml2-dev
```

More information on why these are needed can be found on the Launch Cloud Instances with Knife page.

Install the plugin

Once the dependancies are installed you can install the knife-ec2 plugin with this command:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
sudo gem install knife-ec2 --no-rdoc --no-ri
```

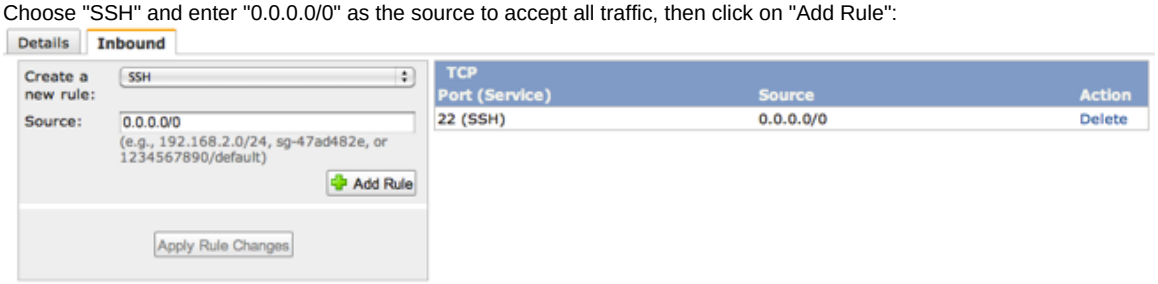
Step 2: Configure AWS settings and the knife.rb file

To authenticate with AWS, you'll need to setup a few things within your AWS account and on your local workstation.

Configure your Amazon security group

You will need to setup SSH access to your EC2 instances from external IPs, so knife will be able to reach them. You can do this by following these steps:

1. Log into the AWS Management Console and select the EC2 tab
2. Click on "Security Groups" from the menu on the left side of the screen
3. Select the default group. (You can select or create a new group in this step, but you will need to remember to add it to your `knife ec2 server create` command later with `-G` as it will use 'default' otherwise.)
4. Click on the "Inbound" tab
5. Choose "SSH" and enter "0.0.0.0/0" as the source to accept all traffic, then click on "Add Rule":



Create your SSH key pair for EC2 and save your identity file on the local workstation

1. Log into the AWS Management Console and select the EC2 tab
2. Click on "Key Pairs" from the menu on the left side of the screen
3. Select the "Create New Pair" button
4. Enter a name for your key pair and create it, this will download a .pem file to your local machine
5. Save the key pair to `~/.ssh/` and remember the name of it
6. make sure ssh-agent is running. To start it:

```
exec ssh-agent /usr/bin/bash
```

   If you do not use exec, you will get the following error message :
   Could not open a connection to your authentication agent

7. Now you can add this key pair to ssh with this command, substituting KEY_PAIR_NAME for the name of your key pair:

```
ssh-add ~/.ssh/KEY_PAIR_NAME.pem
```

   Note: This step is optional, if you skip it you'll need to remember to add the path to the identity file to your `knife ec2 server create` command later with `-i`.

8. You will also want to add this line to the bottom of your knife.rb, substituting KEY_PAIR_NAME for the name of your key pair:

```
knife[:aws_ssh_key_id] = "KEY_PAIR_NAME"
```

   Note: This step is optional, if you skip it you'll need to remember to add the name of your identity file to your `knife ec2 server create` command later with `-S`.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Configure knife.rb with your AWS Cloud Credentials

Add your AWS Cloud Credentials to the bottom of your knife.rb:

```
knife[:aws_access_key_id]     = "Your AWS Access Key"
knife[:aws_secret_access_key] = "Your AWS Secret Access Key"
```

Step 3: Bootstrap the EC2 Instance

Before bootstrapping the instance you'll first need to choose an AMI to use. If you want a small instance, you will need to choose a 32 bit AMI as you need a large or greater for 64 bit. You can use an AMI you've created, an AMI found online, and *we also have an unsupported AMI List for Developers that is useful for testing*.

You will need to know the login name for the AMI, as well as the bootstrap file you want to use.



Where are the bootstrap files located?
The knife bootstrap sub-command looks in three locations for the template to use:

- `bootstrap` directory in the installed Chef Knife library (Location varies depending how you installed Chef).
- `bootstrap` directory in the `$PWD/.chef`, e.g. in `~/chef-repo/.chef`.
- `bootstrap` directory in the users `$HOME/.chef`.

You can then launch the instance with a command like this, substituting in the AMI ID, the LOGIN_NAME and the BOOTSTRAP_FILE as needed:

```
knife ec2 server create -I ami-XXXXXXXX -x LOGIN_NAME -d BOOTSTRAP_FILE
```

Knife will automatically generate a node name based on the instance id. If you'd like to specify a name you can do this with the `-N` flag.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Further examples using the knife-ec2 plugin
If you were creating an **Ubuntu** instance your command may look like this:

```
knife ec2 server create -I ami-6936fb00 -x ubuntu
```

Note that you do not need to use `-d` with ubuntu as this will be used as the default.

If you were creating a **CentOS** instance your command may look like this instead:

```
knife ec2 server create -I ami-43c2032a -d centos5-gems
```

Note that you do not need to add `-x root` to this command as this will be used as the default.

If you wanted to launch a 64-bit AMI you'll need to use a large instance or greater. You can specify this with the `-f` switch. Here's an example bootstrapping a **64-bit RHEL** instance:

```
knife ec2 server create -I ami-31d41658 -d fedora13-gems -f m1.large
```

Here's an example where it's bootstrapping an instance named "server01" with the base and webserver roles. In this example it is also being added to the "www" security group:

```
knife ec2 server create -r "role[base],role[webserver]" -I ami-6936fb00 -G www,default
-x ubuntu -N server01
```

Note that you do not need to use `-d` with ubuntu as this will be used as the default.

Note: The AMIs used above are just an example. You should use the most current AMIs for your OS. For instance: the Ubuntu AMIs are found at their site, in the version subdirectories under "releases".

Configuration options

You can see the full list of options that you can add to the knife.rb config file by reading the readme for the knife-ec2 plugin. You can see the full list of switches available at the command line by using this command:

```
knife ec2 server create --help
```

Here is a partial list, of the commands you are most likely to need:

`-I` - The AMI for the server
`-x` - The ssh username, by default it will use "root"
`-d` - Bootstrap a distro using a template, by default it will use ubuntu
`-N` - The Chef node name for your new node, if this is not specified it will use the instance name. For example "i-91f66cf4".
`-r` - Comma separated list of roles/recipes to apply. For example: `knife ec2 server create -r 'role[webserver],recipe[users]'`
...
`-G` - Comma separated list of security groups for this server, by default it will just use "default"
`-f` - The flavor of server (m1.small, m1.medium, etc), this will default to m1.small if not specified
`--region` - Your AWS region, by default it will use us-east-1
`-Z` - The Availability Zone

Expected Output

This is the output you can expect after entering this command. First you will see it create the instance:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
Instance ID: i-XXXXXXXX
Flavor: m1.small
Image: ami-XXXXXXXX
Region: us-east-1
Availability Zone: us-east-1b
Security Groups: default
SSH Key: KEY_PAIR_NAME

Waiting for server.......................
Public DNS Name: ec2-XXX-XXX-XXX-XXX.compute-1.amazonaws.com
Public IP Address: XXX.XXX.XXX.XXX
Private DNS Name: ip-XXX-XXX-XXX-XXX.ec2.internal
Private IP Address: XXX.XXX.XXX.XXX

Waiting for sshd
```

It may pause at this step for a few minutes while SSHing into the instance. If it gets stuck at this step, make sure that you've properly setup the security group and if you are not using the "default" group you'll want to specify it with the `-G` switch. You'll also want to confirm you've followed all of the steps to create and save your SSH Key.

Once it has logged in it'll start running the bootstrap script, installing Ruby and Chef-client:

```
Waiting for sshd..done
Bootstrapping Chef on ec2-50-17-140-80.compute-1.amazonaws.com
ec2-50-17-140-80.compute-1.amazonaws.com --2012-02-08 23:06:13--
http://download.fedora.redhat.com/pub/epel/5/i386/epel-release-5-4.noarch.rpm
ec2-50-17-140-80.compute-1.amazonaws.com Resolving download.fedora.redhat.com...
ec2-50-17-140-80.compute-1.amazonaws.com 209.132.181.26, 209.132.181.27
```

After a few minutes you should see it report that the Chef install was successful, and it'll run chef-client for the first time:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
ec2-107-22-155-130.compute-1.amazonaws.com Successfully installed chef-0.10.X
ec2-107-22-155-130.compute-1.amazonaws.com 15 gems installed
ec2-107-22-155-130.compute-1.amazonaws.com [Thu, 09 Feb 2012 04:02:28 +0000] INFO: *** Chef 0.10.X ***
ec2-107-22-155-130.compute-1.amazonaws.com [Thu, 09 Feb 2012 04:02:29 +0000] INFO: Client key
/etc/chef/client.pem is not present - registering
ec2-107-22-155-130.compute-1.amazonaws.com [Thu, 09 Feb 2012 04:02:31 +0000] INFO: Setting the
run_list to [] from JSON
ec2-107-22-155-130.compute-1.amazonaws.com [Thu, 09 Feb 2012 04:02:31 +0000] INFO: Run List is []
ec2-107-22-155-130.compute-1.amazonaws.com [Thu, 09 Feb 2012 04:02:31 +0000] INFO: Run List expands to
[]
ec2-107-22-155-130.compute-1.amazonaws.com [Thu, 09 Feb 2012 04:02:31 +0000] INFO: Starting Chef Run
for NODENAME
ec2-107-22-155-130.compute-1.amazonaws.com [Thu, 09 Feb 2012 04:02:31 +0000] INFO: Running start
handlers
ec2-107-22-155-130.compute-1.amazonaws.com [Thu, 09 Feb 2012 04:02:31 +0000] INFO: Start handlers
complete.
ec2-107-22-155-130.compute-1.amazonaws.com [Thu, 09 Feb 2012 04:02:32 +0000] INFO: Loading cookbooks
[]
ec2-107-22-155-130.compute-1.amazonaws.com [Thu, 09 Feb 2012 04:02:33 +0000] WARN: Node NODENAME has
an empty run list.
ec2-107-22-155-130.compute-1.amazonaws.com [Thu, 09 Feb 2012 04:02:34 +0000] INFO: Chef Run complete
in 2.029583 seconds
ec2-107-22-155-130.compute-1.amazonaws.com [Thu, 09 Feb 2012 04:02:34 +0000] INFO: Running report
handlers
ec2-107-22-155-130.compute-1.amazonaws.com [Thu, 09 Feb 2012 04:02:34 +0000] INFO: Report handlers
complete

Instance ID: i-XXXXXXXX
Flavor: m1.small
Image: ami-XXXXXXXX
Region: us-east-1
Availability Zone: us-east-1b
Security Groups: default
SSH Key: KEY_PAIR_NAME
Root Device Type: instance-store
Public DNS Name: ec2-XXX-XXX-XXX-XXX.compute-1.amazonaws.com
Public IP Address: XXX.XXX.XXX.XXX
Private DNS Name: ip-XXX-XXX-XXX-XXX.ec2.internal
Private IP Address: XXX.XXX.XXX.XXX
Environment: _default
```

You should see the current version of chef in the output of this command, Chef 0.10.8. Now chef-client has been installed on the EC2 Instance and it is ready for recipes to be added to it. You can also add to the run_list when creating the instance by using the `-r` switch.

This process does not set up the Chef Client service as a daemon, you'll need to add a recipe in the run list that does this if needed. Step 7 also explains this further.

Step 4: Verify the install completed

Now that the client has registered with Hosted Chef, it should show up on the ec2 server list as well as the node list. You can use knife to display a list of ec2 servers:

```
% knife ec2 server list
Instance ID      Public IP        Private IP       Flavor           Image            SSH Key
Security Groups  State
i-XXXXXXXX       XXX.XXX.XXX.XXX    XXX.XXX.XXX.XXX   m1.small         ami-XXXXXXXX     KEY_PAIR_NAME
default          running
```

Instead of i-XXXXXXXX you should see the instance id of your new node here. If it does not show up on this list, that means the EC2 instance

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

was not created.

You can also check that the client shows up on the node list with Hosted Chef:

```
% knife node list
    NODENAME
```

Instead of NODENAME you should see the name of your new node here. If it does not show up on this list, that means it was unable to finish convergence and save the node at the end of the chef-client run. You can review your logs, or SSH into the node and try running `sudo chef-client -l debug` to get more information from the debug logs on why this failed.

The Client install is now complete

At this point, your instance has been created and is now configured as a client. The next steps explain how to add a recipe and run it on this node remotely from the workstation, as well as run the client as a daemon. If you are already familiar with Chef, you can skip the rest of the guide and it will have no impact on your install.

If you need to setup another EC2 instance, you can just repeat Step 3 as many times as needed. If you need to delete the instance you've created, there are instructions on how to do this on the Launch Cloud Instances with Knife page.

Step 5: Manage recipes on the new client node

Now that your client is setup, you'll need to know how to add a recipe to it so you can start configuring it. This step will walk you through adding the getting-started recipe to it as an example. These steps are also ran from the management workstation.

We will need to download the getting-started cookbook from the Community Site:

```
% cd ~/chef-repo
% knife cookbook site install getting-started
Installing getting-started to /home/username/chef-repo/.chef/../cookbooks
... output truncated ...
Cookbook getting-started version 0.3.0 successfully installed
```

And then upload the recipe to Hosted Chef so it is available for our nodes:

```
% knife cookbook upload getting-started
Uploading getting-started          [0.3.0]
upload complete
```

Once this is completed, you can add this new recipe to the new nodes run list:

```
% knife node run_list add NODENAME 'recipe[getting-started]'
run_list:  recipe[getting-started]
```

Step 6: Run chef-client on the new client node

You can just SSH into the node at this point and run `sudo chef-client`, but it may be easier to run this command remotely from the workstation with knife. This way, once you have multiple nodes added you can run `chef-client` on all of them at once if needed by using a wildcard (*) as the NODENAME.

You can run chef-client remotely from the management workstation with knife with the SSH subcommand:

```
knife ssh name:NODENAME -x ubuntu "sudo chef-client"
```

By default, knife will use the internal FQDN to connect to nodes. When connecting from outside of your cloud you may need to force this to use the external FQDN. You will want to avoid doing this unless necessary, as this will route all traffic out through the NAT infrastructure and could

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

cause performance hits or additional usage charges.

If you're sure you want to do this, you can do it with the -a option, such as in this example:

```
knife ssh name:NODENAME -x ubuntu "sudo chef-client" -a ec2.public_hostname
```

You do not want to edit the `-a ec2.public_hostname` portion when using this, as this is telling it to connect using the public_hostname attribute instead of the specific FQDN to connect to. For more information on the knife SSH subcommand, review the Knife Built In Subcommands .

Running `chef-client` will run the getting-started recipe you've added to the client node's run_list. The getting-started recipe added a template to the client node in ~/chef-getting-started.txt. Once Chef has completed on the client, you can log into the client and look at the template:

```
% cat ~/chef-getting-started.txt
Welcome to Chef!

This is Chef version 0.10.x.
Running on ubuntu.
Version 10.04.
```

Your output will of course be dependent on the data we have stored about your node.

You can also see what code the recipe contained by opening it with a text editor from the Management Workstation:

```
cd ~/chef-repo
vim cookbooks/getting-started/recipes/default.rb
```

For more information on editing and creating recipes, see the Cookbook Fast Start Guide.

When you are ready to start configuring this node, you can remove this test recipe from the run_list with this command:

```
% knife node run_list remove NODENAME 'recipe[getting-started]'
run_list:
```

Step 7: Running chef-client as a daemon

You can setup chef-client to run as a daemon, so it will periodically run chef-client and get any updates you have submitted without needing to SSH to it every time. The easiest way to do this is with the chef-client cookbook. This will setup the chef-client as a daemon and add a time splay, you can add it the same way you added the getting-started recipe:

```
knife cookbook site install chef-client
knife cookbook upload chef-client
knife node run_list add NODENAME 'recipe[chef-client]'
```

Once it is added you'll need to run chef-client on the node once more for the recipe to configure the server:

```
knife ssh name:NODENAME -x ubuntu "sudo chef-client"
```

Next time you bootstrap a new client node, you can include the `chef-client` cookbook in the Knife Bootstrap command from Step 3 if desired. This will add it to the run_list for the node and run it when installing Chef. You can do this with the `-r` switch:

```
knife ec2 server create -I ami-6936fb00 -x ubuntu -r 'recipe[chef-client]'
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

You can read the cookbook's README to get more information on altering the default attributes.

If something goes wrong

It only takes a small change to a dependency or environment to cause these instructions to not be exact. If you need to delete the instance you've created, there are instructions on how to do this on the Launch Cloud Instances with Knife page.

If you want to troubleshoot the issue, first check the Installation page and refer to documents specific to your OS. You may also want to try SSHing into the new instance and seeing if `sudo chef-client -l debug` gives you any additional debug information. If this doesn't help, head over to the **Support** page for information on where and how to find help from Opscode and the rest of the Chef community.

What should I look at next?

More information on Knife Cloud Plugins...

Launch Cloud Instances with Knife

The Knife Cloud Plugins create an instance and then bootstrap it with chef-client. More information on the plugins available can be found on the Launch Cloud Instances with Knife page.

Another fast start...

Cookbook Fast Start Guide

Cookbooks are how things are distributed and shared in Chef. Continue on the fast start path and visit this page for a quick guide to writing and using Chef Cookbooks.

Perhaps detail on Chef and its use...

Chef Basics

Learn some of the central concepts of configuration management benefits for your infrastructure.

1. We begin with an Architecture Introduction, covering the basic functions of the Chef Server, Nodes, and Chef Workstations and how these components communicate.
2. Then an overview of the Core Components, which introduces all of the aspects and components of modeling your infrastructure, Configuring Nodes and Managing Chef.
3. Onward to Cooking School and begin an Introduction to Cookbooks and More. Cookbooks are Chef's fundamental units of distribution, the way Chef users package up, distribute and share configuration information. Recipes, Resources, Attributes, Roles and more are also introduced.
4. The final basic section is an Introduction to Search and Data Bags, two of Chef's most powerful features allowing you to dynamically change the configuration of your infrastructure based on data.

Chef Architecture

With all that under your belt, it's time to tackle the dirty secrets of what's happening behind the scenes with Chef Architecture. We'll give you the scoop on Chef's Authentication and Authorization system and go over the Anatomy of a Chef Run, where we go in-depth with the process by which your systems get configured. From there we'll review all the executable parts of Chef - Chef Client, Chef Solo, Chef Server, Chef Indexer, and Server API and Cookbook Site API interaction.

Or view video tutorials and training...

Guides

We have a number of Walkthrough Guides available on the building of common stacks - including: Rails, Java, LAMP, and more. There are also

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

How To Guides such as Deploying OpenStack with Chef, Guide to Creating A Cookbook and Writing A Recipe, and more - as well as a full 2.25 hour Webcast of a Chef Tutorial Session. These should be good resources for you as you move forward in the automation of your infrastructure.



Installation

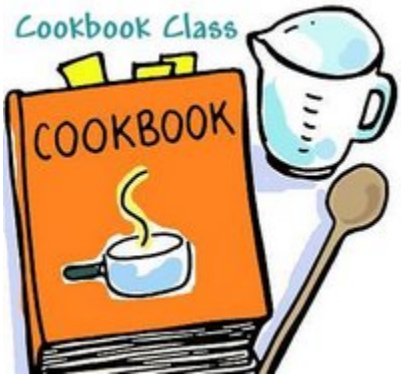
Cookbook Fast Start Guide



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook Fast Start Guide

Overview



Cookbooks are how things are distributed and shared in Chef. Most of your time building infrastructure in Chef will be spent writing Cookbooks; this page is a quick guide to writing and using Chef Cookbooks. It isn't comprehensive, but it will show how some basic components work and we'll explain some terms along the way. To learn more about Chef's terminology, check out the Chef Basics section. If you want more details about how Chef is structured architecturally, check out the Chef Architecture section.

At this point, you should have completed the installation process and have a working Chef server.

If you are using Hosted Chef, you can run these commands from the management workstation you've setup.

Get a Chef Repository

A Chef repository is just a directory structure where you keep your cookbooks and other files. If you haven't done so already, clone Opscode's Chef repository from github:

Clone the Chef Repository from Opscode

```
shell > cd /tmp/sandbox
shell > git clone git://github.com/opscode/chef-repo.git
```

Using Git?
If so, you may want to review our guide on **Working with Git and Cookbooks** for the development workflow with git as your version control system.

Using Cookbooks from the Community Site

When you find a cookbook at the Community Site that you would like to use, you can use these steps to add it to the Chef Server/Hosted chef.

Downloading the Cookbook, Uploading it to the Chef Server/Hosted Chef

You can use this command to download a new cookbook, substitute COOKBOOK for the name of the cookbook you want to download:

```
knife cookbook site install COOKBOOK
```

Once this command completes, the cookbook will have been downloaded to the chef-repo directory, in the cookbooks folder. You can feel free to edit it if needed, then you can upload it to the Chef Server/Hosted Chef with this command:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife cookbook upload COOKBOOK
```

Now this cookbook will be available to add to a node's run_list. If you want to make any changes to it, just edit it from you workstation and upload it again to Chef Server/Hosted Chef. If you'd like some quick information on how to add this to a node's run_list you can find this in Step 3 of the Client Bootstrap Fast Start Guide guide.

For more information on using cookbooks that are on the Community Site see the Cookbooks page.

Creating your own Cookbooks

You can also create your own cookbooks, and upload these into the Chef Server/Hosted chef. The steps below go over creating a new cookbook named `fast_start` and running it on a node.



What is a Recipe? (or a Resource, for that matter!)
Recipes allow you to specify Resources to manage, in the order they should be managed. In this case, you are creating a recipe called "fast_start", which has a single template resource called `template["/tmp/deep_thought.txt"]`. All of this will be part of the cookbook, although a cookbook can contain multiple related recipes if needed.

Create a new Cookbook called "fast_start"

Change your working directory to the `cookbooks/` directory of your Chef repository. If you cloned the repository into `/tmp/sandbox/` as shown above:

```
shell > cd /tmp/sandbox/chef-repo/cookbooks
```

Now create the cookbook with `knife cookbook create`:

Create the fast_start cookbook

```
shell > knife cookbook create fast_start -o ./
** Creating cookbook fast_start
** Creating README for cookbook: fast_start
** Creating metadata for cookbook: fast_start
```

This will create the cookbook's directory structure:

```
shell > ls -1p fast_start
README.rdoc
attributes/
definitions/
files/
libraries/
metadata.rb
providers/
recipes/
resources/
templates/
```

This cookbook is going to have a simple recipe that renders a simple text file using some Attributes we will define.

Add an attribute file called "fast_start.rb"

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

With your favorite text editor, create a file in `cookbooks/fast_start/attributes` called `fast_start.rb`, with the following contents.

cookbooks/fast_start/attributes/fast_start.rb

```
default['deep_thought'] = "If a tree falls in the forest..."
```



What is an Attribute?
Attributes in a Cookbook allow you create settings on a node, which you can then access from within a recipe. Attributes are persistent between Chef runs, and edit-able through the Chef Server web interface!

Add a template resource to the default recipe

Next, edit `cookbooks/fast_start/recipes/default.rb` and add the following:

cookbooks/fast_start/recipes/default.rb

```
template "/tmp/deep_thought.txt" do
  source "deep_thought.txt.erb"
  variables :deep_thought => node['deep_thought']
  action :create
end
```

Add the template file itself

In the template resource we defined above, we referenced a `source` parameter. This is the actual template file, and it is placed in `cookbooks/fast_start/templates/default/deep_thought.txt.erb`. Lets create it now:

cookbooks/fast_start/templates/default/deep_thought.txt.erb

```
Today's deep thought: <%= @deep_thought %>
```



What kind of Template is that?
Chef uses Erubis, a fast version of ERB, to render templates. Read more about it in the Templates section.

Upload the Cookbook to the Server

You should be keeping your cookbooks in version control, and you should start getting in the habit now, so commit your work using your favorite version control system.

Now that your work is in version control, upload the cookbook to the server.

Update the Cookbooks on the Chef Server

```
shell > cd cookbooks
shell > knife cookbook upload -a -o ./
```

The output will look like the following, though your file checksums will be different:

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
INFO: ** fast_start **
INFO: Saving fast_start
INFO: Validating ruby files
INFO: Validating templates
INFO: Syntax OK
INFO: Generating Metadata
INFO: Uploading files
INFO: Uploading /tmp/sandbox/chef-repo/cookbooks/fast_start/recipes/default.rb
(checksum hex = d69c4ce851301dbf1b6d2e8c52ef71db) to
http://localhost:4000/sandboxes/d1ea800f40914d47b179ffd2b8371925/d69c4ce851301dbf1b6d2e8c52ef71db
INFO: Uploading /tmp/sandbox/chef-repo/cookbooks/fast_start/metadata.rb
(checksum hex = 3ba2dc81f4ff8b3b67c611980184c50a) to
http://localhost:4000/sandboxes/d1ea800f40914d47b179ffd2b8371925/3ba2dc81f4ff8b3b67c611980184c50a
INFO: Upload complete!
```

Confirm that the server has the cookbook you just uploaded:

```
shell> knife cookbook list
[
  "fast_start"
]
```

Register a node with chef-client

For this step, you'll need a new host to run chef-client. *We recommend you use a virtual machine with snapshots for the maximum effectiveness when developing cookbooks.* There is also a Client Bootstrap Fast Start Guide guide if you would prefer to set this up on a separate node.

If you'd like to set up a VM, simply create it with the base OS of your choice and install chef client on it. *We also recommend you take a snapshot of the VM in this pristine state—if trouble strikes, you'll be able to roll back and try again.*

Once you have chef-client installed on the host, you'll need to copy the validation key, `/etc/chef/validation.pem`, from your Chef server box to the same location on the chef-client box. The validation key is used by the chef-client as a temporary identity so it can register itself; after it registers it will have its own key in `/etc/chef/client.pem` (read more about Chef's authentication system). (If you don't have an `/etc/chef/validation.pem` or `/etc/chef/client.rb` generate them with `knife configure client...`.)

With the validation key in place, run chef client:

Run chef-client

```
shell > sudo chef-client
[Mon, 16 Aug 2010 15:30:47 -0700] INFO: Client key /tmp/client.pem is not present - registering
[Mon, 16 Aug 2010 15:30:48 -0700] INFO: Starting Chef Run (Version 0.9.9)
[Mon, 16 Aug 2010 15:30:48 -0700] WARN: Node fermi.local has an empty run list.
[Mon, 16 Aug 2010 15:30:48 -0700] INFO: Chef Run complete in 0.468781 seconds
[Mon, 16 Aug 2010 15:30:48 -0700] INFO: Running report handlers
[Mon, 16 Aug 2010 15:30:48 -0700] INFO: Report handlers complete
```

Add the fast_start recipe to your new Node

From your management workstation (where you run knife from) run the `knife node run_list add` command. Use the name of the client machine for `HOSTNAME` (If you're not sure what to put here, check the output of `knife node list`).

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
shell > knife node run_list add HOSTNAME 'recipe[fast_start]'
{
  "run_list": [
    "recipe[fast_start]"
  ]
}
```

You can see that your change was made on the server using `knife node show`:

```
shell > knife node show HOSTNAME -r
{
  "run_list": [
    "recipe[fast_start]"
  ]
}
```

Run chef-client again on your new Node.

This time, executing `chef-client` will cause `/tmp/deep_thought.txt` to appear!

Run chef-client

```
sudo chef-client
```

If you run the command again, you'll notice that it does nothing at all, because the `deep_thought.txt` file has not changed, either on disk or on the Chef Server.

Congratulations!

You have just written your first Chef Cookbook! You'll be managing your entire infrastructure in no time!

Next Steps

Chef Basics

Learn some of the central concepts of configuration management benefits for your infrastructure.

1. We begin with an Architecture Introduction, covering the basic functions of the Chef Server, Nodes, and Chef Workstations and how these components communicate.
2. Then an overview of the Core Components, which introduces all of the aspects and components of modeling your infrastructure, Configuring Nodes and Managing Chef.
3. Onward to Cooking School and begin an Introduction to Cookbooks and More. Cookbooks are Chef's fundamental units of distribution, the way Chef users package up, distribute and share configuration information. Recipes, Resources, Attributes, Roles and more are also introduced.
4. The final basic section is an Introduction to Search and Data Bags, two of Chef's most powerful features allowing you to dynamically change the configuration of your infrastructure based on data.

Chef Architecture

With all that under your belt, it's time to tackle the dirty secrets of what's happening behind the scenes with Chef Architecture. We'll give you the scoop on Chef's Authentication and Authorization system and go over the Anatomy of a Chef Run, where we go in-depth with the process by which your systems get configured. From there we'll review all the executable parts of Chef - Chef Client, Chef Solo, Chef Server, Chef Indexer, and Server API and Cookbook Site API interaction.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Installation

Managing Chef



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Getting Started with Shef

Shef: The Chef REPL



Shef is a way to run chef in an Interactive Ruby (IRb) session. Shef currently supports recipe and attribute file syntax, as well as interactive debugging features.

Run Modes

Shef has three run modes:

- **Standalone**
  No cookbooks are loaded, and the run list is empty. This mode is the default.
- **Solo**
  Shef acts as a chef-solo client. It attempts to load chef-solo's configuration file and JSON attributes. If the JSON attributes set a run list, it will be honored. Cookbooks will be loaded in the same way that chef-solo loads them. Solo mode is activated with the `-s` or `--solo` com mand line option, and JSON attributes are specified in the same way as for chef-solo, with `-j /path/to/chef-solo.json`.
- **Client**
  Shef acts as a chef-client. During startup, it reads the chef client configuration file and contacts the server to get attributes and cookbooks. The run list will be set in the same way as normal chef client runs. Client mode is activated with the `-z` or `--client` options. You can also specify the configuration file with `-c CONFIG` and the server URL with `-S SERVER_URL`.

For more detail on the Chef REPL, see **Shef**.

Getting Started

First, let's start Shef in "standalone" mode, so it doesn't inherit any other configuration. We run this with sudo so we can install a simple package in this example.

```
% sudo shef -a
loading configuration: none (standalone shef session)
Loading...done.

This is shef, the Chef shell.
 Chef Version: 0.8.16
 http://www.opscode.com/chef
 http://wiki.opscode.com/display/chef/Home

run `help' for help, `exit' or ^D to quit.

Ohai2u jtimberman@localhost!
chef >
```

The next step is to enter recipe mode, so we can create actual resources that Chef will manage. We also turn off echoing return values to keep the terminal output clean.

```
chef > recipe
chef:recipe > echo off
```

Now we can start writing Resources directly into the console, and then run Chef to have them created on the system.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
chef:recipe > file "/tmp/shef_example" do
chef:recipe >   action :create
chef:recipe ?>  mode "0600"
chef:recipe ?> end
chef:recipe > run_chef
[Wed, 02 Jun 2010 09:07:24 -0600] DEBUG: Processing file[/tmp/shef_example]
[Wed, 02 Jun 2010 09:07:24 -0600] DEBUG: file[/tmp/shef_example] using Chef::Provider::File
[Wed, 02 Jun 2010 09:07:24 -0600] INFO: Creating file[/tmp/shef_example] at /tmp/shef_example
[Wed, 02 Jun 2010 09:07:24 -0600] INFO: Setting mode to 600 for file[/tmp/shef_example]
```

If we look at the local filesystem, we'll see this file has been created:

```
% ls -l /tmp/shef_example
-rw------- 1 jtimberman jtimberman 0 2010-06-02 09:07 /tmp/shef_example
```

Neat!

Now let's take a look at how we can use a node attribute. Here, we'll set a node attribute, and then simply display it to the output log.

```
chef:recipe > quit
chef > attributes
chef:attributes > set[:shef_example] = "Hello world!"
 => "Hello world!"
chef:attributes > quit
 => :attributes
chef > recipe
chef:recipe > log node.shef_example
chef:recipe > run_chef
[Wed, 02 Jun 2010 09:24:08 -0600] DEBUG: Processing log[Hello world!]
[Wed, 02 Jun 2010 09:24:08 -0600] DEBUG: log[Hello world!] using Chef::Provider::Log::ChefLog
[Wed, 02 Jun 2010 09:24:08 -0600] INFO: Hello world!
```

We first enter attributes editing mode, putting shef into the context of a node's attribute space. We set the attribute value, then we enter recipe mode to create a log resource.

Now let's put the content of this attribute into a file.

```
chef:recipe > file "/tmp/hello_world" do
chef:recipe >   content node.shef_example
chef:recipe ?>  action :create
chef:recipe ?> end
chef:recipe > run_chef
[Wed, 02 Jun 2010 09:29:36 -0600] DEBUG: Processing file[/tmp/shef_example]
[Wed, 02 Jun 2010 09:29:36 -0600] DEBUG: file[/tmp/shef_example] using Chef::Provider::File
[Wed, 02 Jun 2010 09:29:36 -0600] DEBUG: Processing log[Hello world!]
[Wed, 02 Jun 2010 09:29:36 -0600] DEBUG: log[Hello world!] using Chef::Provider::Log::ChefLog
[Wed, 02 Jun 2010 09:29:36 -0600] INFO: Hello world!
[Wed, 02 Jun 2010 09:29:36 -0600] DEBUG: Processing file[/tmp/hello_world]
[Wed, 02 Jun 2010 09:29:36 -0600] DEBUG: file[/tmp/hello_world] using Chef::Provider::File
[Wed, 02 Jun 2010 09:29:36 -0600] INFO: Creating file[/tmp/hello_world] at /tmp/hello_world
[Wed, 02 Jun 2010 09:29:36 -0600] INFO: Setting content for file[/tmp/hello_world]
```

If we look at this file, we see it contains the string from the attribute we set earlier.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
% cat /tmp/hello_world
Hello world!
```

The next example will install a package on the local system.

```
chef:recipe > package "netcat" do
chef:recipe >   action :install
chef:recipe ?> end
chef:recipe > run_chef
[Wed, 02 Jun 2010 09:32:40 -0600] DEBUG: Processing file[/tmp/shef_example]
[Wed, 02 Jun 2010 09:32:40 -0600] DEBUG: file[/tmp/shef_example] using Chef::Provider::File
[Wed, 02 Jun 2010 09:32:40 -0600] DEBUG: Processing log[Hello world!]
[Wed, 02 Jun 2010 09:32:40 -0600] DEBUG: log[Hello world!] using Chef::Provider::Log::ChefLog
[Wed, 02 Jun 2010 09:32:40 -0600] INFO: Hello world!
[Wed, 02 Jun 2010 09:32:40 -0600] DEBUG: Processing file[/tmp/hello_world]
[Wed, 02 Jun 2010 09:32:40 -0600] DEBUG: file[/tmp/hello_world] using Chef::Provider::File
[Wed, 02 Jun 2010 09:32:40 -0600] DEBUG: Processing package[netcat]
[Wed, 02 Jun 2010 09:32:40 -0600] DEBUG: package[netcat] using Chef::Provider::Package::Apt
[Wed, 02 Jun 2010 09:32:40 -0600] DEBUG: Checking apt-cache policy for netcat
[Wed, 02 Jun 2010 09:32:40 -0600] DEBUG: Current version is nil
[Wed, 02 Jun 2010 09:32:40 -0600] DEBUG: Current version is 1.10-38
[Wed, 02 Jun 2010 09:32:40 -0600] INFO: Installing package[netcat] version 1.10-38
[Wed, 02 Jun 2010 09:32:40 -0600] DEBUG: Executing apt-get -q -y install netcat=1.10-38
[Wed, 02 Jun 2010 09:32:43 -0600] DEBUG: ---- Begin output of apt-get -q -y install netcat=1.10-38
----
[Wed, 02 Jun 2010 09:32:43 -0600] DEBUG: STDOUT: Reading package lists...
Building dependency tree...
Reading state information...
The following packages were automatically installed and are no longer required:
  liblua5.1-0
Use 'apt-get autoremove' to remove them.
The following extra packages will be installed:
  netcat netcat-traditional
The following NEW packages will be installed:
  netcat netcat-traditional
0 upgraded, 2 newly installed, 0 to remove and 5 not upgraded.
Need to get 75.9kB of archives.
After this operation, 291kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com karmic/main netcat-traditional 1.10-38 [69.8kB]
Get:2 http://us.archive.ubuntu.com karmic/main netcat 1.10-38 [6,178B]
Fetched 75.9kB in 1s (73.5kB/s)
Selecting previously deselected package netcat-traditional.
(Reading database ... 89514 files and directories currently installed.)
Unpacking netcat-traditional (from .../netcat-traditional_1.10-38_amd64.deb) ...
Selecting previously deselected package netcat.
Unpacking netcat (from .../netcat_1.10-38_all.deb) ...
Processing triggers for man-db ...
Setting up netcat-traditional (1.10-38) ...

Setting up netcat (1.10-38) ...
[Wed, 02 Jun 2010 09:32:43 -0600] DEBUG: STDERR:
[Wed, 02 Jun 2010 09:32:43 -0600] DEBUG: ---- End output of apt-get -q -y install netcat=1.10-38 ----
[Wed, 02 Jun 2010 09:32:43 -0600] DEBUG: Ran apt-get -q -y install netcat=1.10-38 returned 0
```

The next example will install three gems that are optional for use with Knife sub-commands (fog, net-ssh-multi and highline)

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
chef > echo off
chef > recipe
chef:recipe > %w{ fog net-ssh-multi highline }.each do |g|
chef:recipe > gem_package g
chef:recipe ?> end
 => ["fog", "net-ssh-multi", "highline"]
chef:recipe > run_chef
[Sat, 10 Jul 2010 20:16:43 -0600] DEBUG: Processing gem_package[fog] on cider.local
[Sat, 10 Jul 2010 20:16:43 -0600] DEBUG: gem_package[fog] using Chef::Provider::Package::Rubygems
[Sat, 10 Jul 2010 20:16:43 -0600] DEBUG: Found installed gem fog version 0.1.8 matching fog (>= 0,
runtime)
[Sat, 10 Jul 2010 20:16:43 -0600] DEBUG: Processing gem_package[net-ssh-multi] on cider.local
[Sat, 10 Jul 2010 20:16:43 -0600] DEBUG: gem_package[net-ssh-multi] using
Chef::Provider::Package::Rubygems
[Sat, 10 Jul 2010 20:16:43 -0600] DEBUG: Found installed gem net-ssh-multi version 1.0.1 matching
net-ssh-multi (>= 0, runtime)
[Sat, 10 Jul 2010 20:16:43 -0600] DEBUG: Processing gem_package[highline] on cider.local
[Sat, 10 Jul 2010 20:16:43 -0600] DEBUG: gem_package[highline] using Chef::Provider::Package::Rubygems
[Sat, 10 Jul 2010 20:16:43 -0600] DEBUG: Found installed gem highline version 1.5.2 matching highline
(>= 0, runtime)
 => true
```

Uses of Shef

Onward to **Shef** for more details, including its use for managing the Chef Server, debugging Recipes, and more.

 Fast Start Guide

Managing Chef 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Workstation Setup

This guide will explain how to setup your workstation for Cookbook development and Chef infrastructure management with Knife.



So what's the difference between a workstation and a node?

A workstation manages Chef
A Node is managed by Chef

Your workstation *can be* set up as a client of the chef-server just like managed nodes, but you will mainly be invoking commands and uploading Roles, Cookbooks and other Chef artifacts to the server.

Chef itself is written in Ruby, as are Cookbooks, Knife plugins, and Chef Exception and Report Handlers. You don't have to be a Ruby expert to be an effective Chef Cook, but as you dive into Chef you will find yourself embracing Chef's Ruby roots.

The operating system specific guides found below explain how to:

1. Install Ruby and Chef on your workstation,
2. Install git on your workstation, and
3. Create and configure a chef repository on your workstation

System Requirements

Chef-client is supported on the following platforms

- Ubuntu (10.04, 10.10, 11.04, 11.10)
- Debian (5.0, 6.0)
- RHEL & CentOS (5.x, 6.x)
- Fedora 10+
- Mac OS X (10.4, 10.5, 10.6, 10.7)
- Windows 7
- Windows Server 2003 R2, 2008 R2

Additionally, chef-client is known to run on the following platforms

- Ubuntu (6.06, 8.04-9.10)
- Gentoo (11.1,11.2)
- FreeBSD (7.1)
- OpenBSD (4.4)
- OpenSolaris (2008.11)
- Solaris 5.10 (u6)
- SuSE (11.x)
- Windows XP, Vista

The Components of a Chef Workstation

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Ruby and Chef

As mentioned above, Chef is written in Ruby. A Chef Workstation must have ruby installed. On Unix based platforms (Mac OS X, Ubuntu, CentOS, Debian etc.) we will be using packages provided either by the distribution's repository or from well-known 3rd party repositories. On the Windows platform we will install Ruby 1.9 with the RubyInstaller provided by the Ruby community. We will also install the Ruby Development Kit t o allow the native extensions of certain gems to compile correctly.

Rubygems will be installed from source on Unix-like platforms and via the RubyInstallerChef on windows. Chef will be installed using Rubygems.

Git ***(optional)***

Git is a distributed version control system. The Git project describes itself best:

> *Git is a free & open source, distributed version control system designed to handle everything from small to very large projects with speed and efficiency. Every Git clone is a full-fledged repository with complete history and full revision tracking capabilities, not dependent on network access or a central server. Branching and merging are fast and easy to do.*

You don't have to use Git to effectively use Chef, but Git is used heavily by the Chef community as a whole and the larger Ruby development community.

A Chef Repository

A Chef Repository is a folder that holds your configuration files, cookbooks, and the other files that drive your infrastructure's configuration. Most of the work you do will be from within a chef repository. We recommend using the Git version control system to effectively manage your chef repository.

Select a Guide

And with that we are ready to go. *Please select the appropriate guide for your platform and happy cooking!*

- *Workstation Setup for Mac OS X*
- *Workstation Setup for CentOS, Red Hat, Fedora*
- *Workstation Setup for Debian and Ubuntu*
- *Workstation Setup for Windows*

 Installation

Installing Chef Server 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Workstation Setup for CentOS, Red Hat, Fedora



This article explains how to setup a Chef Workstation on CentOS, RHEL, and related Operating Systems.

A Chef workstation is where you develop cookbooks, interact with your chef-server, and interact with nodes.

Such a workstation typically includes:

- A local repository of your cookbooks and possibly other data.
- A well-configured knife client

To create such a workstation, this article explains how to:

1. Install Ruby and other Chef dependencies
2. Install Chef
3. Install Git
4. Create a bare-bones Chef repository
5. Configure Knife



This workstation setup has been tested and verified on the following versions of CentOS:

- CentOS 5.6
- CentOS 6.0



Commands Run as Root
Commands that require root privileges are run with `sudo` in these directions. If your system is not configured to use `sudo`, run these commands as root using a method appropriate for your configuration.

Install Ruby

Chef is written in Ruby and thus requires Ruby as well as other system dependencies to run. Ruby 1.8.7+ is required to run Chef. Run the following command to install ruby and other required dependencies:

On Red Hat and CentOS (Version 5)

Enable AegisCo repository to get Ruby 1.8.7.

If you will be installing a Chef Server on this host, you will also need to enable the RBEL repo:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Install Ruby and other development tools:

On Red Hat and CentOS (Version 6)

Install the RBEL repo

Install Ruby and other development tools:

Install RubyGems from Source

We prefer to install RubyGems from source rather than use the OS-provided version (if any), as it is cross platform, so we know what to expect.

Install RubyGems

Install Chef Gem

To install Chef and its dependencies, run the following code:

Install Chef

Git (optional)

Git is easily installed from the Extra Packages for Enterprise Linux (EPEL) yum repository.

```
sudo rpm -Uvh http://download.fedora.redhat.com/pub/epel/5/i386/epel-release-5-4.noarch.rpm
sudo yum install -y git
```

```
sudo rpm -Uvh http://download.fedora.redhat.com/pub/epel/6/i386/epel-release-6-5.noarch.rpm
sudo yum install -y git
```

For more in depth Git information please refer to the help guide on github.com.

Create a Chef Repository on your Workstation

The following directions will create a Chef repository--a folder on your local workstation from which you can manage your infrastructure. (Advanced users may find is productive to deviate from the basic setup outlined in these directions.)

Copy the skeleton Chef Repository available on Github

Using Git

Using git, we can create a chef repository by cloning a skeleton repository provided by Opscode:

This command will create a folder called `chef-repo` in your home directory. You can clone the repository to a different directory using a command such as:

Without Git

If you do not have git, you can copy the skeleton repository using standard unix tools:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Create a Configuration Folder

Now, we need to copy the necessary keys and knife configuration into this directory.

For Hosted and Private Chef customers, USERNAME.pem can be generated from your community.opscode.com profile; ORGANIZATION-validator.pem and knife.rb can be generated from the organizations picker in Management Console:

where ORGANIZATION should be replaced with your organizations short name and USERNAME should be replaced by your Hosted Chef username. Note, that the paths to `USERNAME.pem`, `ORGANIZATION-validator.pem`, and `knife.rb` should be change to the path to which they were downloaded.

For those using the Open Source Chef Server, you first need to create an admin client for use with knife.
Directions for creating an admin client can be found in the post-install section of the server configuration. Once you have an admin client, copy the user's private key and the validator's private key into your chef repository:

Now, you can then create a knife configuration file using the following command:

`knife configure` will prompt you for information regarding your admin client as well as your chef server.

Confirm your configuration is working

To confirm that your configuration is working, attempt to run a simple knife command

Install other knife plugins (optional)

Additional Knife commands are shipped as additional gems, and would be installed separately if desired. These include Opscode Knife Plugins to Launch Cloud Instances with Knife on popular cloud services such as ec2 and rackspace. There are also Community Plugins for knife, chef and ohai - developed and made available by the open source community to perform a number of different functions.

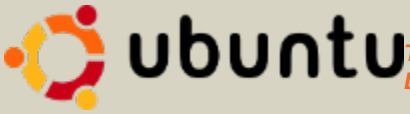
These plugins can be install using Rubygems:

Installation

Installing Chef Server

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Workstation Setup for Debian and Ubuntu



This article explains how to setup a Chef Workstation on Ubuntu or Debian.

A Chef workstation is where you develop cookbooks, and interact with your chef-server and nodes.

Such a workstation typically includes:

- A local repository of your cookbooks and possibly other data.
- A well-configured knife client

To create such a workstation, this article explains how to

- Install Ruby and other Chef dependencies
- Install Chef
- Install Git
- Create a bare-bones chef repository
- Configure Knife

This workstation setup has been tested and verified on the following versions of Debian:

- Debian 5.0 (Lenny)
- Debian 6.0
- Ubuntu 10.04
- Ubuntu 10.10
- Ubuntu 11.04



Install Ruby and other Dependencies

Chef is written in Ruby and thus requires Ruby as well as other system dependencies to run. Run the following command to install ruby and other required dependencies:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef 0.10 Note
There are open issues with Chef 0.10.x and Ruby 1.8.6. If you are using Chef 0.10.x, you should get 1.8.7 with SSL bindings. If you'd like to use 1.9.1 or 1.9.2, you'll want to follow the Installing Chef Client on Other Operating Systems directions to install manually with gems.

Install RubyGems from Source

We prefer to install RubyGems from source rather than use the OS-provided version (if any), as it is cross platform, so we know what to expect.

Install RubyGems

Install Chef Gem

To install Chef and its dependencies, run the following code:

Install Chef

Git (optional)

Git is easily installed from Debians's main apt repository

```
$ sudo apt-get -y install git-core
$ git --version
git version 1.5.6.5
```

For more in depth Git information please refer to the help guide on github.com.

Create a Chef Repository on your Workstation

The following directions will create a Chef repository--a folder on your local workstation from which you can manage your infrastructure. (Advanced users may find is productive to deviate from the basic setup outlined in these directions.)

Copy the skeleton Chef Repository available on Github

Using Git

Using git, we can create a chef repository by cloning a skeleton repository provided by Opscode:

This command will create a folder called `chef-repo` in your home directory. You can clone the repository to a different directory using a command such as:

Without Git

If you do not have git, you can copy the skeleton repository using standard unix tools:

Create a Configuration Folder

Now, we need to copy the necessary keys and knife configuration into this directory.

For Hosted and Private Chef customers, USERNAME.pem can be generated from your community.opscode.com profile;

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

ORGANIZATION-validator.pem and knife.rb can be generated from the organizations picker in Management Console:

where ORGANIZATION should be replaced with your organizations short name and USERNAME should be replaced by your Hosted Chef username. Note, that the paths to `USERNAME.pem`, `ORGANIZATION-validator.pem`, and `knife.rb` should be change to the path to which they were downloaded.

For those using the Open Source Chef Server, you first need to create an admin client for use with knife.
Directions for creating an admin client can be found in the post-install section of the server configuration. Once you have an admin client, copy the user's private key and the validator's private key into your chef repository:

Now, you can then create a knife configuration file using the following command:

`knife configure` will prompt you for information regarding your admin client as well as your chef server.

Confirm your configuration is working

To confirm that your configuration is working, attempt to run a simple knife command

Install other knife plugins (optional)

Additional Knife commands are shipped as additional gems, and would be installed separately if desired. These include Opscode Knife Plugins to Launch Cloud Instances with Knife on popular cloud services such as ec2 and rackspace. There are also Community Plugins for knife, chef and ohai - developed and made available by the open source community to perform a number of different functions.

These plugins can be install using Rubygems:

Installation

Installing Chef Server

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Workstation Setup for Mac OS X



This article explains how to setup a Chef Workstation on OS X.

A Chef workstation is where you develop cookbooks, interact with your chef-server, and interact with nodes.

Such a workstation typically includes:

- A local repository of your cookbooks and possibly other data.
- A well-configured knife client

To create such a workstation, this article explains how to:

1. Install Ruby and other Chef dependencies
2. Install Chef
3. Install Git
4. Create a bare-bones chef repository
5. Configure Knife



This has been tested and verified on the following versions of Mac OS X:

- Mac OS X 10.6.7 (Snow Leopard)

Developer Tool Chain

Download and install Xcode 3.x (ie the **free** version). You can also purchase Xcode 4.0 from the Mac app store but this version is not required. A copy of Xcode should also be present on the DVDs that shipped with your Mac.

Xcode provides the required development tools and headers (GCC and friends) that are used when compiling Rubies in RVM and native extensions in Ruby gems.

Install RubyGems from Source

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

We prefer to install RubyGems from source rather than use the OS-provided version (if any), as it is cross platform, so we know what to expect.

Install RubyGems

Install Chef Gem

To install Chef and its dependencies, run the following code:

Install Chef

Git (optional)

Download and install the latest version of the Git for OS X installer.

Note Don't worry that you don't see an icon when it's done. It's not that kind of application.

For more in depth Git information please refer to the help guide on github.com.

Create a Chef Repository on your Workstation

The following directions will create a Chef repository--a folder on your local workstation from which you can manage your infrastructure. (Advanced users may find is productive to deviate from the basic setup outlined in these directions.)

Copy the skeleton Chef Repository available on Github

Using Git

Using git, we can create a chef repository by cloning a skeleton repository provided by Opscode:

This command will create a folder called `chef-repo` in your home directory. You can clone the repository to a different directory using a command such as:

Without Git

If you do not have git, you can copy the skeleton repository using standard unix tools:

Create a Configuration Folder

Now, we need to copy the necessary keys and knife configuration into this directory.

For Hosted and Private Chef customers, USERNAME.pem can be generated from your community.opscode.com profile; ORGANIZATION-validator.pem and knife.rb can be generated from the organizations picker in Management Console:

where ORGANIZATION should be replaced with your organizations short name and USERNAME should be replaced by your Hosted Chef username. Note, that the paths to `USERNAME.pem`, `ORGANIZATION-validator.pem`, and `knife.rb` should be change to the path to which they were downloaded.

For those using the Open Source Chef Server, you first need to create an admin client for use with knife.
Directions for creating an admin client can be found in the post-install section of the server configuration. Once you have an admin client, copy the user's private key and the validator's private key into your chef repository:

Now, you can then create a knife configuration file using the following command:

`knife configure` will prompt you for information regarding your admin client as well as your chef server.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Confirm your configuration is working

To confirm that your configuration is working, attempt to run a simple knife command

Install other knife plugins (optional)

Additional Knife commands are shipped as additional gems, and would be installed separately if desired. These include Opscode Knife Plugins to Launch Cloud Instances with Knife on popular cloud services such as ec2 and rackspace. There are also Community Plugins for knife, chef and ohai - developed and made available by the open source community to perform a number of different functions.

These plugins can be install using Rubygems:

 Installation

Installing Chef Server 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Workstation Setup for Windows

This article explains how to setup a Chef Workstation on Windows.

A Chef workstation is where you develop cookbooks, and interact with your chef-server and nodes.

Such a workstation typically includes:

- A local repository of your cookbooks and possibly other data.
- A well-configured knife client

To create such a workstation, this article explains how to:

1. Install Chef and Git
2. Create a bare-bones chef repository
3. Configure Knife

Note: If you are using Hosted Chef with Windows 2008 R2 you can use the Fast Start Guide for Windows for directions on setting up your node as a workstation and a client. This guide also contains a quick walkthrough on using recipes on the client.

Due to bug CHEF-2250, `knife cookbook site install` does not currently work on Windows. A workaround to this bug is explained below.



This workstation setup has been tested and verified on the following versions of Windows:

- Windows 7
- Windows 2003
- Windows 2008 R2

Install Chef

Download the Chef Full Installer for Windows and open it. Choose defaults for any options.

Once it is installed there will be no icon for it. If needed, you can confirm it was installed correctly with these commands in a new command prompt window:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
C:\> tar --version
bsdtar 2.8.3 - libarchive 2.8.3

C:\> chef-client --version
```

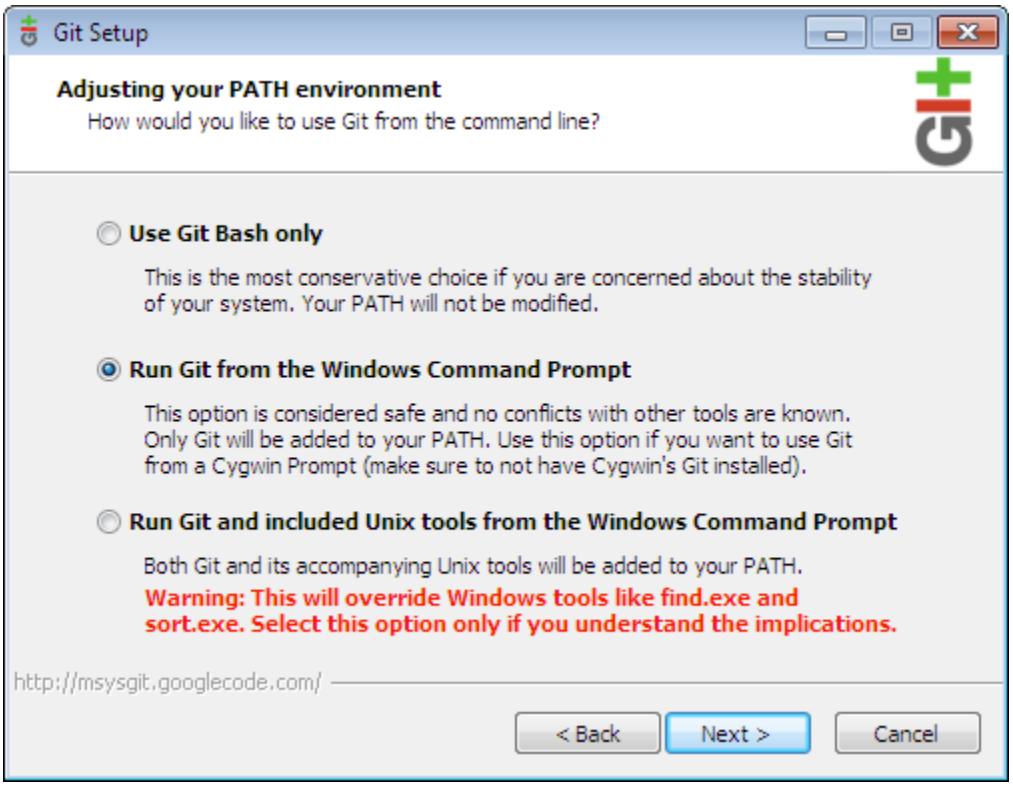
You should see "Chef: 0.10.8" returned as the version number.

Install Git (optional)

Git is used to download cookbooks, and the Chef Repository easily. You can skip this step and you will still be able to bootstrap nodes, and add recipes and roles to run_lists. If you skip this step and want to work on your cookbooks, you will need to manually download the cookbook you need from the Community site instead of using knife.

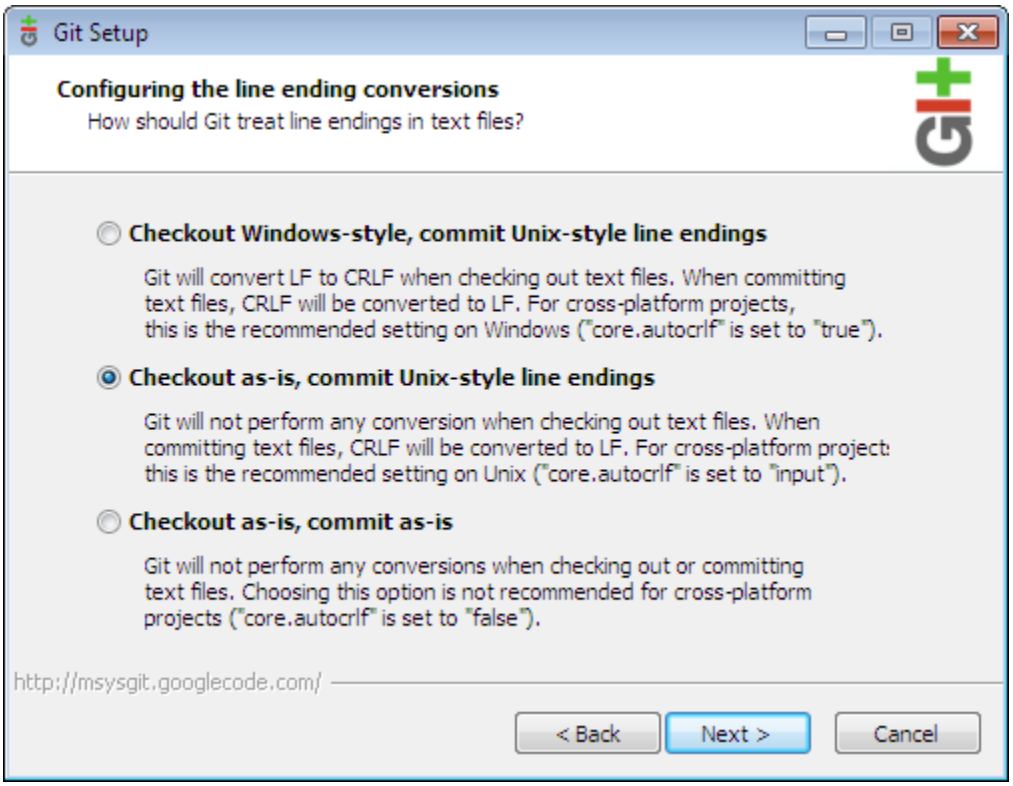
Download the Git Full installer for Windows and install it.

To make Git easily accessible from normal Windows Command Prompts, be sure to choose the 'Run Git from the Windows Command Prompt' option.



For the line ending conversions option be sure to choose: 'Checkout as-is, commit Unix-style line endings'. Your fellow developers on *nix platforms will thank you!

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Once Git has installed, close any command line windows and re-open them so the git command can be used.

Note Don't worry if you don't see an icon when it's done. It's not that kind of application.

For more in depth Git information please refer to the help guide on github.com.

Create a Chef Repository on your workstation

The following directions will create a Chef repository--a folder on your local workstation from which you can manage your infrastructure. Advanced users may find is productive to deviate from the basic setup outlined in these directions.

Copy the skeleton Chef Repository available on Github

Using Git

Using git, we can create a chef repository by cloning a skeleton repository provided by Opscode:

```
C:\Users> cd "%HOMEPATH%"
C:\Users\USERNAME> git clone git://github.com/opscode/chef-repo.git
```

This command will create a folder named chef-repo in your home directory. You can clone the repository to a different directory using a command such as:

```
C:\> git clone git://github.com/opscode/chef-repo.git C:\path\to\alternate\directory
```

If you choose to use an alternate directory, you'll need to change this in the commands below.

Without Git

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

If you skipped installing Git, you can copy the skeleton repository by going onto github.com.

1. Browse to https://github.com/opscode/chef-repo
2. Click on 'Downloads' in the upper right corner

- If you **do not want to use tar**, you can select the .zip version and then extract the contents to the %HOMEPATH%/chef-repo directory.
- If you **want to use tar** through the command line to extract the file, you can select the tar.gz version, download it to your user's directory, and then extract it with these commands:

```
C:\> cd "%HOMEPATH%"
C:\Users\USERNAME> tar zxvf chef-repo.tar.gz
```

After this has been completed, you should be able to browse to the %HOMEPATH%\chef-repo\cookbooks directory.

Create a Configuration Folder

First we will need to create the .chef file where the pem keys and knife.rb are stored:

```
C:\> mkdir "%HOMEPATH%\chef-repo\.chef"
```

Now, we need to copy the necessary keys and knife configuration into this directory.

The rest of this step will be a bit different depending on the platform you are using:

- **Hosted Chef**
  For Hosted Chef users, these files can be downloaded from the Management Console, they need to be moved to the new %HOMEPATH%\chef-repo\.chef directory. If you downloaded them to "C:\Users\USERNAME\Downloads" you could use these commands to move them for example:

  ```
  C:\> cd "%HOMEPATH%\Downloads"
  C:\Users\USERNAME\Downloads> copy USERNAME.pem "%HOMEPATH%\chef-repo\.chef"
  C:\Users\USERNAME\Downloads> copy ORGANIZATION-validator.pem "%HOMEPATH%\chef-repo\.chef"
  C:\Users\USERNAME\Downloads> copy knife.rb "%HOMEPATH%\chef-repo\.chef"
  ```

  Where ORGANIZATION should be replaced with your organizations short name and USERNAME should be replaced by your Hosted Chef username. Note, that the paths to USERNAME.pem, ORGANIZATION-validator.pem, and knife.rb should be change to the path to which they were downloaded.

  You will also need to edit the knife.rb to provide the path to the cookbooks. You can do this by opening %HOMEPATH%\chef-repo\.chef\knife.rb in Wordpad, and looking for this line:

  ```
  cookbook_path    ["#{current_dir}/../cookbooks"]
  ```

  If you see this, you'll want to change it to the directory below or some knife commands may fail:

  ```
  cookbook_path    ["#{ENV['HOME']}/chef-repo/cookbooks"]
  ```

  If your %HOMEPATH% has any spaces in it, which will be the case if you are using **Windows 2003 or XP** you'll want to use the full path to cookbooks instead:

  ```
  cookbook_path    ['C:\Documents and Settings\USERNAME\chef-repo\cookbooks']
  ```

- **Open Source Chef Server**

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

For users hosting their own chef-server, you first need to create an admin client for use with knife. Directions for creating an admin client can be found in the post-install section of the server configuration). Once you have an admin client, copy the user's private key and the validation key into your chef repository:

```
C:\> cd "%HOMEPATH%\Downloads"
C:\Users\USERNAME\Downloads> copy USERNAME.pem "%HOMEPATH%\chef-repo\.chef"
C:\Users\USERNAME\Downloads> copy validation.pem "%HOMEPATH%\chef-repo\.chef"
```

Now, you can create a knife configuration file using the `knife configure` command. Here is an example of what you could enter for each question, if you used the directions on this page:

```
C:\> cd "%HOMEPATH%\chef-repo"
C:\Users\USERNAME\chef-repo> knife configure
WARNING: No knife configuration file found
Where should I put the config file? [~/.chef/knife.rb] .chef\knife.rb
Please enter the chef server URL: [http://localhost:4000] http://chef-server.example.com:4000
Please enter an existing username or clientname for the API: [user] USERNAME
Please enter the validation clientname: [chef-validator]
Please enter the location of the validation key: [/etc/chef/validation.pem]
C:\Users\USERNAME\chef-repo\.chef\validation.pem
Please enter the path to a chef repository (or leave blank): C:\Users\USERNAME\chef-repo
```

Once this is completed, it will create the `knife.rb` file in `%HOMEPATH%\chef-repo\.chef`.

Confirm your configuration is working

To confirm that your configuration is working, attempt to run a simple knife command

```
C:\> cd "%HOMEPATH%\chef-repo"
C:\Users\USERNAME\chef-repo> knife client list
    ORGANIZATION-validator
```

NOTE: knife commands will only work in the `%HOMEPATH%\chef-repo` directory unless you add `-c "%HOMEPATH%\chef-repo\.chef\knife.rb"` to the end of the command to specify where the knife config is located.

This command would work from any directory:

```
C:\> knife client list -c "%HOMEPATH%\chef-repo\.chef\knife.rb"
```

Install a Cookbook

We recommend that you install git per the instructions above, and then use these commands to install a cookbook:

```
C:\> cd "%HOMEPATH%\chef-repo\cookbooks"
C:\Users\USERNAME\chef-repo\cookbooks> knife cookbook site download getting-started
C:\Users\USERNAME\chef-repo\cookbooks> tar zxvf getting-started-0.2.0-tar.gz
C:\Users\USERNAME\chef-repo\cookbooks> del getting-started-0.2.0-tar.gz
```

This will create the file chef-repo\cookbooks\getting-started in your user's home directory, you can see the default recipe contained in this cookbook at chef-repo\cookbooks\getting-started\recipes\default.rb.

Upload a Cookbook

You can then upload the cookbook to the Chef Server/Hosted Chef with this command. This will make the recipe available to be added to a run_list for a node:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
C:\> cd "%HOMEPATH%\chef-repo"
C:\Users\USERNAME\chef-repo> knife cookbook upload getting-started
```

If you get an error where it cannot find the cookbook in your path, you can specify where they are located with the `-o` switch:

```
C:\> knife cookbook upload getting-started -o "%HOMEPATH%\chef-repo\cookbooks"
```

If you edit the knife.rb with the cookbook_path instructions above and relaunch a cmd window you should no longer need to use the `-o` switch.

Configure this workstation as a client

If you decide that you would also like to run chef-client on this node, you can follow the directions on Step 5 of the Fast Start Guide for Windows t o configure chef-client.

Knife plugins

The knife windows plugin can be used to bootstrap new Windows nodes as clients. The plugin can also be used from Mac or Linux workstations. You can install it with Rubygems by using this command:

```
C:\> gem install knife-windows
```

There are also plugins for managing EC2 or Rackspace nodes. For a more complete list see Community Plugins.

Some Resources/Providers Are Known To Not Work

Not all Resources work on windows yet. The Following Resources do not work in windows:

- Cron
- Deploy
- Erlang call
- Git
- Http request
- Ifconfig
- Link
- Mdadm
- Package (but gem_package - for ruby gems, does work)
- Route
- SCM
- Script
- Subversion

Also, using 'user' or 'group' attributes within a resource, specifically the template or directory resource is known to cause issues. Related bug: CH EF-2633

 Installation

Installing Chef Server 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Installing Chef Server



This guide explains how to install Chef Server on your own hardware.

The Chef Server provides a central point for the distribution of Cookbooks, management and authentication of Nodes, and the use of Search.

Installation Methods

Choose one of the following methods to install chef-server based on the operating system of the host

Installation via Ubuntu or Debian Package

On Debian or Ubuntu, install Chef from Opscode's apt repository with apt-get. This method works with the following operating systems:

- Ubuntu 11.04
- Ubuntu 10.10
- Ubuntu 10.04
- Debian (unstable)
- Debian (testing)
- Debian 6 (stable)

Installation via Bootstrap

Using the bootstrap method, you install ruby with your system's package manager, install Chef with rubygems, and then install chef-server using a chef-solo cookbook. This method works on the following operating system and possibly others:

- Ubuntu 10.04-11.04
- Debian 6.0 (stable, testing, unstable)
- CentOS 5.x and 6.x (should work on RHEL, too)

Manual Installation

If packages are not available for your operating system, or you simply enjoy staying close to the metal, you can install the Chef server manually.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

System Requirements

Chef-client is supported on the following platforms

- Ubuntu (10.04, 10.10, 11.04, 11.10)
- Debian (5.0, 6.0)
- RHEL & CentOS (5.x, 6.x)
- Fedora 10+
- Mac OS X (10.4, 10.5, 10.6, 10.7)
- Windows 7
- Windows Server 2003 R2, 2008 R2

Additionally, chef-client is known to run on the following platforms

- Ubuntu (6.06, 8.04-9.10)
- Gentoo (11.1,11.2)
- FreeBSD (7.1)
- OpenBSD (4.4)
- OpenSolaris (2008.11)
- Solaris 5.10 (u6)
- SuSE (11.x)
- Windows XP, Vista



Hosted Chef is a a highly available, scalable Chef Server in the cloud offered by Opscode. **To use Hosted Chef as your Chef Server, follow the instructions for "Hosted Chef + Chef Clients".**

Dependencies

Chef Solr, which manages denormalizing and indexing data, and is usually run on the same host as Chef Server in smaller installations, requires the following additional gems:

- libxml-ruby

In addition to the client dependencies above, Chef Server requires:

- merb-assets
- merb-core
- merb-helpers
- merb-param-protection
- merb-slices
- thin

Additionally for the webui:

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- merb-haml
- haml
- coderay

Non-Ruby dependencies:

- CouchDB (0.9.1-0.10+)
- Java
- RabbitMQ

For ease of installation, we recommend using an OS platform for the Chef Server that has these available as native packages.

 Installation

Installing Chef Client and Chef Solo 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Installing Chef Server Manually



This page describes how to manually set up a Chef Server. This is the only option if your platform does not have native Chef packages and your platform is not supported by the RubyGem installation bootstrap.

These procedures will assume installation of Chef is done with RubyGems, and commands are executed as root through sudo. If your system does not have sudo installed, we recommend it. Otherwise, login as root and run the command w/o sudo.

Install Ruby and Chef

Chef-server is written in Ruby. Per Operating System directions for installing Ruby and Chef can be found at Installing Chef Client and Chef Solo.

Install CouchDB

Chef requires CouchDB version 0.9.1 or higher. If CouchDB 0.9.1+ is not available as a package on your platform, you'll need to install it from source. You may also need to install Erlang from source. *See the CouchDB installation instructions for more information.*

Once CouchDB is installed, you'll need to start it. If you installed it from a package for your platform, this might be done automatically. If you installed from source, check the source tarball for init scripts or instructions for your platform. You shouldn't need any other special configuration—Chef Server will create its database when it starts for the first time.

Install RabbitMQ

The Chef Indexer sends messages across an AMQP queue to get them indexed for search. We use RabbitMQ for this. You'll need to download and install RabbitMQ as appropriate for your platform, see their site.

Once RabbitMQ is installed, make sure the service is running.

Configure RabbitMQ

Once RabbitMQ is installed, you'll need to set up the queue. The following commands assume v1.6+ syntax.

```
sudo rabbitmqctl add_vhost /chef
sudo rabbitmqctl add_user chef testing
sudo rabbitmqctl set_permissions -p /chef chef ".*" ".*" ".*"
```


Commands Run as Root
Commands that require root privileges are run with `sudo` in these directions. If your system is not configured to use `sudo`, run these commands as root using a method appropriate for your configuration.

Install Java

Chef 0.8.0+ uses SOLR for the search engine. You need to install Java for your platform. This is packaged for most common Linux distributions but may not be in the default repositories. For example, enable on Debian or Ubuntu:

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- Debian: non-free
- Ubuntu: multiverse

This varies by platform.



OpenJDK May Work too!
Some Linux distributions include OpenJDK, which may work for SOLR as well, but isn't as widely tested.

Install zlib and libxml

The SOLR search engine used in Chef 0.8.0+ also requires zlib and libxml. Make sure to install the development headers for these libraries on your system.

Install gecode

Install Using Deb Package

On Ubuntu lucid, maverick, or Debian lenny release, you can install gecode deb package from the Opscode APT repository.

Add the Opscode APT Repository

Create `/etc/apt/sources.list.d/opscode.list`.

/etc/apt/sources.list.d/opscode.list

- For Chef 0.9.x, replace codename with the supported distribution codename, such as "lucid".
- For Chef 0.10.x, replace codename with the codename, suffixed with "-0.10", for example, "lucid-0.10".

If you would like to be able to download source packages, add an additional identical line, but change `deb` to `deb-src` line.

You can copy and paste these examples to create the necessary sources.list entry:

Ubuntu for Chef 0.9.x

Ubuntu for Chef 0.10.x

Debian users will likely need to run 'apt-get install sudo wget lsb-release' as root before pasting the examples.

Add the GPG Key and Update Index

Before you install the packages, make sure you add the Opscode GPG key to apt.

Issues downloading from gnupg.net?
If you get an error when trying to download the key that states that the "keyserver timed out", try downloading the key directly from our apt repository instead:

Afterwards you'll also want to run this command again:

Now, we update apt-get with the newly added Opscode repository:

Installing the opscode-keyring package will keep the key up-to-date:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Upgrade Existing Packages

To ensure you are using the latest versions of libraries that chef depends on, you may wish to update your existing packages:

Install deb package

```
sudo apt-get install libgecode-dev
```

Build and Install from Source

```
cd /tmp
curl -C - -O  http://www.gecode.org/download/gecode-3.5.0.tar.gz
tar zxvf gecode-3.5.0.tar.gz
cd gecode-3.5.0 && ./configure
make && make install
```

Install Chef Server

Install the `chef-server` gems.

```
sudo gem install chef-server chef-server-api chef-server chef-solr
```

Optionally install the WebUI.

```
sudo gem install chef-server-webui
```

Configure Chef Server

The server configuration file is `/etc/chef/server.rb`, and is what the chef-indexer will use by default as well. Here's a sample to get you started.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

/etc/chef/server.rb

```
log_level          :info
log_location       STDOUT
ssl_verify_mode    :verify_none
chef_server_url    "http://chef.example.com:4000"

signing_ca_path    "/var/chef/ca"
couchdb_database   'chef'

cookbook_path      [ "/var/chef/cookbooks", "/var/chef/site-cookbooks" ]

file_cache_path    "/var/chef/cache"
node_path          "/var/chef/nodes"
openid_store_path  "/var/chef/openid/store"
openid_cstore_path "/var/chef/openid/cstore"
search_index_path  "/var/chef/search_index"
role_path          "/var/chef/roles"

validation_client_name "chef-validator"
validation_key         "/etc/chef/validation.pem"
client_key             "/etc/chef/client.pem"
web_ui_client_name     "chef-webui"
web_ui_key             "/etc/chef/webui.pem"

web_ui_admin_user_name "admin"
web_ui_admin_default_password "somerandompasswordhere"

supportdir = "/srv/chef/support"
solr_jetty_path File.join(supportdir, "solr", "jetty")
solr_data_path  File.join(supportdir, "solr", "data")
solr_home_path  File.join(supportdir, "solr", "home")
solr_heap_size  "256M"

umask 0022

Mixlib::Log::Formatter.show_time = false
```

Replace chef.example.com with the proper FQDN for the server. Feel free to change the paths to fit your environment or your platform's preferences (for example, FHS on Linux).

The web_ui_admin_user_name and web_ui_admin_default_password settings are optional, the default admin user is 'admin', and the webui will instruct you how to change the password when you log in.

Start Chef Indexer

*Again, **we recommend runit for this**, but if runit is not available for your platform or you're using init scripts or a different init scheme, you'll need to set it up to start at boot time.* To get up and running quickly for testing, you can simply run the indexer program.

This runs as root to write to the file location specified by `search_index_path`.

```
sudo chef-solr-indexer
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Using chef 0.10.0 or greater?
Since Chef 0.10.0, indexer has been replaced with chef-expander (http://www.opscode.com/blog/2011/05/02/chef-0-10-0-released/).

```
sudo chef-expander -n1
```

Start Chef SOLR Server

Next we need to start the SOLR search engine. You'll want to configure this as a service that starts at boot time for your system (runit, init script, etc).

```
sudo chef-solr
```

Start Chef Server

To get running immediately, run the `chef-server` command. See above spiel for system startup methods.

The chef-server API and webui are separate, and the webui is optional. Set these up as system services like the others. By default these use the 'thin' web server adapter for merb.

```
sudo chef-server -N -e production
sudo chef-server-webui -p 4040 -e production
```

Verify That All Components are Running

Now that you have Chef Server installed, you should have the following processes running.

Name	Listen Port	Example Program Name in ps (Erlang programs truncated)
Chef Server	4000	`merb : chef-server (api) : worker (port 4000)`
Chef Server WebUI	4040	`merb : chef-server-webui : worker (port 4040)`
CouchDB	5984	`beam.smp -Bd -K true – -root /usr/local/lib/erlang -progname erl – -noshell -noinput -couch_ini /usr/local/etc/couchdb/default.ini /usr/local/etc/couchdb/local.ini -s couch`
RabbitMQ	5672	{{beam.smp -W w -K true -A30 – -root /usr/local/lib/erlang -progname erl – -noshell -noinput -s rabbit -sname rabbit -rabbit tcp_listeners [{"0.0.0.0", 5672}]}}
Chef Solr	8983	`/usr/bin/java -Xmx250M -Xms250M -Dsolr.data.dir=/opscode/chef/features/data/solr/data -Dsolr.solr.home=/opscode/chef/features/data/solr/home -jar /opscode/chef/features/data/solr/jetty/start.jar`
Chef Expander	*none*	`ruby ./chef-solr/bin/chef-expander -c /etc/chef/solr.rb -l debug`

Configure the Command Line Client

Once you've verified that all of Chef's components are working, it's time to configure the knife command line tool. On your Chef Server, run `knife configure -i` to interactively configure your knife client and create an admin account on the server. You can accept the default responses shown in brackets by pressing `<enter>`.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

First, create the ~/.chef directory and copy the required certificates created by the server.



Certificates Read Only
Presumably you're running this as a normal non-privileged user. When the `chef-server-api` starts, it creates the validation and webui certificates as read/write only by the user that starts the process (`chef`).

Next run the `knife configure` command, and pass the `-i` flag so the initial client that will be used to authenticate with the API.

API Client creation must use an existing admin client's credentials to create the new account. On a brand new server, the `chef-webui` is created by `chef-server-api` startup. Use the default for the clientname, and specify the webui.pem copied above.

Finally, specify a location for the Chef Repository. This is used to configure knife to point to the directory where Cookbooks will be stored.

Verify Your Knife Client Configuration

You can now run some basic knife commands to verify that you can communicate with the server.

Create a Knife Client for Your Laptop/Desktop

Create Your Client Account

When working with chef, you will spend a lot of time editing recipes and other files, and you'll find it much more convenient to edit them on your laptop/desktop (your management workstation), where you have your editor configured just to your liking. **To facilitate this mode of working, we recommend you create a knife client to use knife on your development machine.**

Make sure you've configured knife on your chef server as described above before proceeding with this step.

This command creates the client and writes its private key to `/tmp/my-username.pem`. To verify the operation, use the `knife client show` command:

Copy Your Key and Configure Knife

Now you need to copy the key you just created to your development machine using scp (or some other file copy mechanism).

In order to use knife on your laptop, you'll need to install the chef-client. If your laptop's OS comes with rubygems (Mac OS X, for example) you can run `sudo gem install chef`; otherwise, check out the client installation instructions for your OS. Once you have chef installed, you can use knife's interactive configuration on your laptop (note we're **not** using the `-i` option here):

Knife looks for its configuration in `HOME/.chef/knife.rb by default`:

Now, enter your client name, exactly as you did when running `knife client create` above:

For these next settings, you can accept the defaults for now and update them later by editing your knife.rb file. The validation client name and key are used with knife's cloud computing commands:

We'll also leave the path to the chef repository blank for now. After you've created a chef repository, you'll want to configure it by editing knife.rb

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Verify Your Configuration

You can run some `list` and `show` commands to verify everything is working:

You're now ready to use Knife Bootstrap to automatically set up systems to become new Chef Clients to the server.

Next?

You'll probably want to set up a Chef Repository for your configuration, or learn How to Proxy Chef Server with Apache, and then go to Cooking School.

 Installation

Installing Chef Client and Chef Solo 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Installing Chef Server on Debian or Ubuntu using Packages



Installing using native Ubuntu or Debian ensures that Chef is installed in the same way as other software on your system.

These instructions use the Opscode APT repository. Opscode maintains the packages in this APT repo, as well as the packages that get included in Debian and Ubuntu's repositories.

The following distributions are supported within the Opscode APT repository:

- Ubuntu 11.10 oneiric
- Ubuntu 11.04 natty
- Ubuntu 10.10 maverick
- Ubuntu 10.04 lucid
- Debian (unstable) sid
- Debian (testing) wheezy
- Debian 6 (stable) squeeze

Install Processes

Installing chef-server using the Opscode provided packages for Debian and Ubuntu requires that you:

1. Add the Opscode Apt Repository to your apt sources,
2. Install the chef-server package and its dependencies,
3. Verify the installation succeeded, and
4. Complete the Post-Install configuration

The following sections take you through each of these steps in detail.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Add the Opscode APT Repository

Create `/etc/apt/sources.list.d/opscode.list`.

/etc/apt/sources.list.d/opscode.list

- For Chef 0.9.x, replace codename with the supported distribution codename, such as "lucid".
- For Chef 0.10.x, replace codename with the codename, suffixed with "-0.10", for example, "lucid-0.10".

If you would like to be able to download source packages, add an additional identical line, but change `deb` to `deb-src` line.

You can copy and paste these examples to create the necessary sources.list entry:

Ubuntu for Chef 0.9.x

Ubuntu for Chef 0.10.x

Debian users will likely need to run 'apt-get install sudo wget lsb-release' as root before pasting the examples.

Add the GPG Key and Update Index

Before you install the packages, make sure you add the Opscode GPG key to apt.

Issues downloading from gnupg.net?
If you get an error when trying to download the key that states that the "keyserver timed out", try downloading the key directly from our apt repository instead:

Afterwards you'll also want to run this command again:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Now, we update apt-get with the newly added Opscode repository:

Installing the opscode-keyring package will keep the key up-to-date:

Upgrade Existing Packages

To ensure you are using the latest versions of libraries that chef depends on, you may wish to update your existing packages:

Install the chef-server package

```
sudo apt-get install chef chef-server
```

You will be prompted by debconf for the Chef Server URL. Put in a value like "http://chef.example.com:4000". You will also be prompted for the rabbitmq consumer password and the webui admin password.

This will:

- Install all the dependencies for Chef Server, including Merb, CouchDB, RabbitMQ, etc.
- Starts CouchDB (via the couchdb package).
- Starts RabbitMQ (via the rabbitmq-server package).
- Start chef-server-api via /etc/init.d/chef-server, running a merb worker on port 4000
- Start chef-server-webui via /etc/init.d/chef-server-webui, running a merb worker on port 4040
- Start chef-solr-indexer via /etc/init.d/chef-solr-indexer, connecting to the rabbitmq-server
- Start chef-solr via /etc/init.d/chef-solr, using the distro package for solr-jetty
- Start chef-client via /etc/init.d/chef-client
- Add configuration files in /etc/chef for the client, server, solr/solr-indexer and solo
- Create all the correct directory paths per the configuration files

If you do not want the webui installed, install the server-api and expander packages separately:

```
sudo apt-get install chef chef-server-api chef-expander
```

This does the same as above without the chef-server-webui service.

Verify That All Components are Running

Now that you have Chef Server installed, you should have the following processes running.

Name	Listen Port	Example Program Name in ps (Erlang programs truncated)
Chef Server	4000	`merb : chef-server (api) : worker (port 4000)`
Chef Server WebUI	4040	`merb : chef-server-webui : worker (port 4040)`
CouchDB	5984	`beam.smp -Bd -K true – -root /usr/local/lib/erlang -progname erl – -noshell -noinput -couch_ini /usr/local/etc/couchdb/default.ini /usr/local/etc/couchdb/local.ini -s couch`
RabbitMQ	5672	{{beam.smp -W w -K true -A30 – -root /usr/local/lib/erlang -progname erl – -noshell -noinput -s rabbit -sname rabbit -rabbit tcp_listeners [{"0.0.0.0", 5672}]}}
Chef Solr	8983	`/usr/bin/java -Xmx250M -Xms250M -Dsolr.data.dir=/opscode/chef/features/data/solr/data -Dsolr.solr.home=/opscode/chef/features/data/solr/home -jar /opscode/chef/features/data/solr/jetty/start.jar`

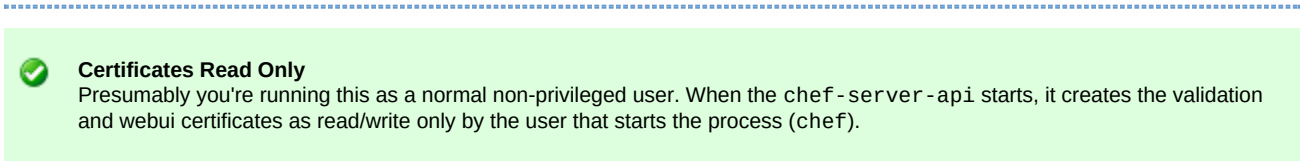
 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Expander	*none*	`ruby ./chef-solr/bin/chef-expander -c /etc/chef/solr.rb -l debug`

Configure the Command Line Client

Once you've verified that all of Chef's components are working, it's time to configure the knife command line tool. On your Chef Server, run `knife configure -i` to interactively configure your knife client and create an admin account on the server. You can accept the default responses shown in brackets by pressing `<enter>`.

First, create the ~/.chef directory and copy the required certificates created by the server.



Certificates Read Only
Presumably you're running this as a normal non-privileged user. When the `chef-server-api` starts, it creates the validation and webui certificates as read/write only by the user that starts the process (`chef`).

Next run the `knife configure` command, and pass the `-i` flag so the initial client that will be used to authenticate with the API.

API Client creation must use an existing admin client's credentials to create the new account. On a brand new server, the `chef-webui` is created by `chef-server-api` startup. Use the default for the clientname, and specify the webui.pem copied above.

Finally, specify a location for the Chef Repository. This is used to configure knife to point to the directory where Cookbooks will be stored.

Verify Your Knife Client Configuration

You can now run some basic knife commands to verify that you can communicate with the server.

Create a Knife Client for Your Laptop/Desktop

Create Your Client Account

When working with chef, you will spend a lot of time editing recipes and other files, and you'll find it much more convenient to edit them on your laptop/desktop (your management workstation), where you have your editor configured just to your liking. **To facilitate this mode of working, we recommend you create a knife client to use knife on your development machine.**

Make sure you've configured knife on your chef server as described above before proceeding with this step.

This command creates the client and writes its private key to `/tmp/my-username.pem`. To verify the operation, use the `knife client show` command:

Copy Your Key and Configure Knife

Now you need to copy the key you just created to your development machine using scp (or some other file copy mechanism).

In order to use knife on your laptop, you'll need to install the chef-client. If your laptop's OS comes with rubygems (Mac OS X, for example) you can run `sudo gem install chef`; otherwise, check out the client installation instructions for your OS. Once you have chef installed, you can use knife's interactive configuration on your laptop (note we're **not** using the `-i` option here):

Knife looks for its configuration in `HOME/.chef/knife.rb by default`:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Now, enter your client name, exactly as you did when running `knife client create` above:

For these next settings, you can accept the defaults for now and update them later by editing your knife.rb file. The validation client name and key are used with knife's cloud computing commands:

We'll also leave the path to the chef repository blank for now. After you've created a chef repository, you'll want to configure it by editing knife.rb

Verify Your Configuration

You can run some `list` and `show` commands to verify everything is working:

You're now ready to use Knife Bootstrap to automatically set up systems to become new Chef Clients to the server.

 Installation

Installing Chef Client and Chef Solo 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Installing Chef Server using Chef Solo



"Configuring my chef server with chef - it doesn't get any more meta than this" - bbrowning

Bootstrap in the context of this document refers to the initial configuration of a Chef Server for your infrastructure.

The bootstrap install works using `chef-solo` to run chef recipes that install a full Chef Server.

The bootstrap install does this because RubyGems does not have a way to handle several things package management systems do easily:

- Create configuration files.
- Create system-level init scripts.
- Start new services from init scripts.
- Install non-Ruby programs/libraries.

This bootstrap is handled by a special recipe in the Opscode Chef Cookbook, `chef-server::rubygems-install`. This page will describe the basics of how to use this recipe. The documentation for the cookbook, including the details of the settings available to configure these recipes, is in the README for the server and README for the client side.

Install Process

Installing chef-server using Chef Solo requires that you

1. Install Chef Solo,
2. Create the necessary Chef Solo configuration files,
3. Run chef-solo using the chef-server cookbook,
4. Verify the installation succeeded, and
5. Complete post-installation configuration task.

The following sections explain each part of this process in detail.



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Commands Run as Root
Commands that require root privileges are run with `sudo` in these directions. If your system is not configured to use `sudo`, run these commands as root using a method appropriate for your configuration.

Install Chef-Solo

Chef-solo is packaged together with Chef-client. To install chef-solo on the host that will become your chef server, follow the directions to install chef-client:

- Installing Chef Client and Chef Solo

Configure Chef-solo

You need to give chef-solo a minimal configuration before it can run the bootstrap recipes. You need to configure two parts: chef-solo's own configuration file (`/etc/chef/solo.rb` by default) tells chef-solo where to store its files, and a JSON attributes file gives chef-solo the values it uses to configure your Chef Server.

Chef Solo Configuration File: solo.rb

We're going to use Chef Solo to run the bootstrap recipes, so it needs to be configured to point the right locations.

First, create a chef directory in `/etc`

```
sudo mkdir /etc/chef/
```

Now create a chef-solo configuration file. Save it as `/etc/chef/solo.rb`.

Edit /etc/chef/solo.rb

```
file_cache_path "/tmp/chef-solo"
cookbook_path "/tmp/chef-solo/cookbooks"
```

Chef Solo Attributes Configuration: chef.json

Chef can use JSON data passed to Solo or the Client to specify a certain list of recipes to run and specific Attributes to configure on the system. We're going to configure the node with a JSON file particular to whether it will be a Chef Server or a Chef Client. Create the file `~/chef.json` wit h the applicable contents below.

Chef Server with no webui:

Server Attributes (API Only)

```
{
  "chef_server": {
    "server_url": "http://localhost:4000"
  },
  "run_list": [ "recipe[chef-server::rubygems-install]" ]
}
```

Chef Server with the webui installed and enabled:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Server Attributes (API and WebUI)

```
{
  "chef_server": {
    "server_url": "http://localhost:4000",
    "webui_enabled": true
  },
  "run_list": [ "recipe[chef-server::rubygems-install]" ]
}
```

Set The Init Style For Your System

The bootstrap recipes currently default to using runit as the init system. We're big fans of runit, but you might prefer a different init system.



The default Init Style for RHEL and RHEL-like Distros is "init"
RHEL, CentOS, and related distros don't have a runit package. The default is "init" init style for these systems.

If you need to specify a different init system, add the `init_style` attribute to your chef.json:

Server Attributes for Using Your System's Default Init System

```
{
  "chef_server": {
    "server_url": "http://localhost:4000",
    "init_style": "init"
  },
  "run_list": [ "recipe[chef-server::rubygems-install]" ]
}
```

There are a lot of other attributes available to configure for the Chef Server and Chef Clients. See the Chef Cookbook README for complete details.

Bootstrap Chef Server

This procedure will set up the `chef-server-api` to run on port 4000 using the `thin` adapter, and optionally the `chef-server-webui` on port 4040. SSL encryption can be provided by proxying with Apache.

The webui is now separate from the API, so it is now optional. Setting the attribute `webui_enabled` to true will install and configure it.

Run `chef-solo`

Chef Solo Installs and Configures Chef Server

```
sudo chef-solo -c /etc/chef/solo.rb -j ~/chef.json -r
http://s3.amazonaws.com/chef-solo/bootstrap-latest.tar.gz
```



CentOS/RHEL Issues
There are a number of issues setting up Chef Server on CentOS/RHEL, please see the CentOS/RHEL Installation Notes

This command will bring up the Chef Server by doing the following:

- Bootstrap the system as a Client (see section below for what this entails).
- Install RabbitMQ if possible (see below).
- Install CouchDB if possible (see below).
- Install development libraries zlib and xml, for `chef-solr`.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- Install the `chef-server`, `chef-server-api`, `chef-solr` gems.
- Optionally (if webui_enabled) install `chef-server-webui` gem.
- Create the server configuration file, `/etc/chef/server.rb`.
- Create some directories the server needs.
- If `init_style` is "runit", set up `chef-expander`, `chef-solr`, `chef-server` (API) as runit services. If webui_enabled, it `chef-server-webui` will be added as a runit service as well. If "init", copy the init scripts for these services from the installed Chef gem for the current platform (Debian and Red Hat families supported). If "bsd", display a hint about startup commands. Otherwise, display a message about manual setup.

Verify That All Components are Running

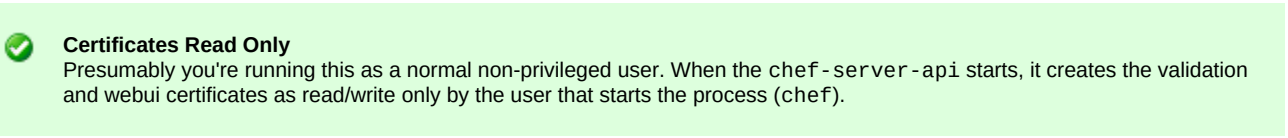
Now that you have Chef Server installed, you should have the following processes running.

Name	Listen Port	Example Program Name in ps (Erlang programs truncated)
Chef Server	4000	`merb : chef-server (api) : worker (port 4000)`
Chef Server WebUI	4040	`merb : chef-server-webui : worker (port 4040)`
CouchDB	5984	`beam.smp -Bd -K true – -root /usr/local/lib/erlang -progname erl – -noshell -noinput -couch_ini /usr/local/etc/couchdb/default.ini /usr/local/etc/couchdb/local.ini -s couch`
RabbitMQ	5672	{{beam.smp -W w -K true -A30 – -root /usr/local/lib/erlang -progname erl – -noshell -noinput -s rabbit -sname rabbit -rabbit tcp_listeners [{"0.0.0.0", 5672}]}}
Chef Solr	8983	`/usr/bin/java -Xmx250M -Xms250M -Dsolr.data.dir=/opscode/chef/features/data/solr/data -Dsolr.solr.home=/opscode/chef/features/data/solr/home -jar /opscode/chef/features/data/solr/jetty/start.jar`
Chef Expander	*none*	`ruby ./chef-solr/bin/chef-expander -c /etc/chef/solr.rb -l debug`

Configure the Command Line Client

Once you've verified that all of Chef's components are working, it's time to configure the knife command line tool. On your Chef Server, run `knife configure -i` to interactively configure your knife client and create an admin account on the server. You can accept the default responses shown in brackets by pressing `<enter>`.

First, create the ~/.chef directory and copy the required certificates created by the server.



Certificates Read Only
Presumably you're running this as a normal non-privileged user. When the `chef-server-api` starts, it creates the validation and webui certificates as read/write only by the user that starts the process (`chef`).

Next run the `knife configure` command, and pass the `-i` flag so the initial client that will be used to authenticate with the API.

API Client creation must use an existing admin client's credentials to create the new account. On a brand new server, the `chef-webui` is created by `chef-server-api` startup. Use the default for the clientname, and specify the webui.pem copied above.

Finally, specify a location for the Chef Repository. This is used to configure knife to point to the directory where Cookbooks will be stored.

Verify Your Knife Client Configuration

You can now run some basic knife commands to verify that you can communicate with the server.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Create a Knife Client for Your Laptop/Desktop

Create Your Client Account

When working with chef, you will spend a lot of time editing recipes and other files, and you'll find it much more convenient to edit them on your laptop/desktop (your management workstation), where you have your editor configured just to your liking. **To facilitate this mode of working, we recommend you create a knife client to use knife on your development machine.**

Make sure you've configured knife on your chef server as described above before proceeding with this step.

This command creates the client and writes its private key to `/tmp/my-username.pem`. To verify the operation, use the `knife client show` command:

Copy Your Key and Configure Knife

Now you need to copy the key you just created to your development machine using scp (or some other file copy mechanism).

In order to use knife on your laptop, you'll need to install the chef-client. If your laptop's OS comes with rubygems (Mac OS X, for example) you can run `sudo gem install chef`; otherwise, check out the client installation instructions for your OS. Once you have chef installed, you can use knife's interactive configuration on your laptop (note we're **not** using the `-i` option here):

Knife looks for its configuration in `HOME/.chef/knife.rb by default`:

Now, enter your client name, exactly as you did when running `knife client create` above:

For these next settings, you can accept the defaults for now and update them later by editing your knife.rb file. The validation client name and key are used with knife's cloud computing commands:

We'll also leave the path to the chef repository blank for now. After you've created a chef repository, you'll want to configure it by editing knife.rb

Verify Your Configuration

You can run some `list` and `show` commands to verify everything is working:

You're now ready to use Knife Bootstrap to automatically set up systems to become new Chef Clients to the server.

CentOS/RHEL Installation Notes

- You must have the RBEL or other similar 3rd party repository installed to run the bootstrap cookbook.
- CentOS 5.x users will need to have version 4.2 or better of gcc in order for this to function. You may install gcc44 and g++44 via yum to fulfill this requirement. If you do so, make sure you export
  CXX=`which g++44` and CC=`which gcc44` when running the bootstrap.
- If you run into the following error when running the bootstrap:

```
FATAL: ArgumentError: directory[/var/lib/couchdb] (couchdb::default line 30) had an error: can't
find group for couchdb
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Attempt to run the bootstrap again.

- If you find that chef-server did not start after running the bootstrap, attempt to run the following commands:

```
echo "/usr/local/lib" >> /etc/ld.so.conf
ldconfig
```

Then attempt to restart the chef services. The easiest way to restart all the services at once is often to just rerun the bootstrap.

- The chef-server::apache-proxy recipe will fail on CentOS/RHEL 6 because apache and chef-server::apache-proxy cookbooks included in http://s3.amazonaws.com/chef-solo/bootstrap-latest.tar.gz are out-of-date. Try using the following tarball instead. See COOK-973 for details.

Chef Solo Installs and Configures Chef Server

```
sudo chef-solo -c /etc/chef/solo.rb -j ~/chef.json -r
https://s3.amazonaws.com/chef-solo/bootstrap-latest.tar.gz
```

 Installation

Installing Chef Client and Chef Solo 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Installing Chef Client and Chef Solo

These directions explain installing chef-client or chef-solo on hosts that you would like to manage using Chef.



Chef Client is a Chef agent that runs on your nodes.
Chef-client connects to a Chef Server to be told what to do on the node.

Chef Solo is a standalone version of Chef that runs locally on your node, detached from a Chef server.
This means that all the information and Cookbooks required to configure the node have to be present on the local disk.

Both are shipped together and can be installed using the Operating System-specific directions listed below.

Per Operating System Directions

The following are Operating System-specifc guides to installing Chef Client (or chef-solo) on a host. If your operating system is not listed, use the Other Operating Systems guide, which provides general instructions for install Chef using Rubygems as well as specific notes for various operating systems.

- Ubuntu or Debian
- CentOS, Scientific Linux, Fedora, or RHEL
- OS X
- Windows Server 2008
- Other Operating Systems

Dependencies

The following are the packages and gems required to install and run chef-client. The directions above handle the installation of all require dependencies. They are listed here for those wishing to install Chef via alternate means.

- Ruby 1.8.7+ (with SSL bindings and development tools)
- Rubygems 1.7.2+
- ri
- rdoc
- gcc
- gcc-c++
- automake
- autoconf
- make

Gems written and provided by Opscode:

- mixlib-authentication
- mixlib-cli
- mixlib-config
- mixlib-log
- ohai

External gems:

- bunny
- erubis
- extlib
- highline

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- json
- moneta
- rest-client
- uuidtools

System Requirements

Chef-client is supported on the following platforms

- Ubuntu (10.04, 10.10, 11.04, 11.10)
- Debian (5.0, 6.0)
- RHEL & CentOS (5.x, 6.x)
- Fedora 10+
- Mac OS X (10.4, 10.5, 10.6, 10.7)
- Windows 7
- Windows Server 2003 R2, 2008 R2

Additionally, chef-client is known to run on the following platforms

- Ubuntu (6.06, 8.04-9.10)
- Gentoo (11.1,11.2)
- FreeBSD (7.1)
- OpenBSD (4.4)
- OpenSolaris (2008.11)
- Solaris 5.10 (u6)
- SuSE (11.x)
- Windows XP, Vista

We will be installing (or updating) the following in this Installation Guide.

Ruby
Ruby 1.8.5, 1.8.6, 1.8.7, 1.9.1 or 1.9.2 with SSL bindings is required.

RubyGems
Version 1.3.7 or greater. On Ubuntu and Debian Rubygems should be installed from source.

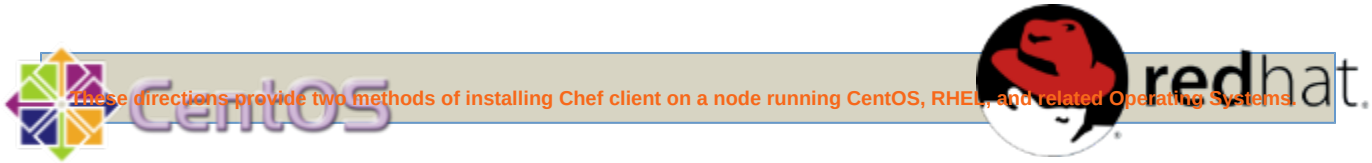
Chef Client Version
Hosted Chef is compatible with chef-client v0.9.0 and greater. Older clients need to be upgraded before connecting.

Installing Chef Server

Upgrade Instructions

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Installing Chef Client on CentOS



Nodes are hosts whose configuration that you want to manage using Chef.

In general, installing chef-client on a node includes:

- Installing Ruby and Other Dependencies
- Installing Chef
- Configuring Chef

Typically, chef-client is installed on CentOS systems using one of two possible methods:

- Using Rubygems, or
- Using a knife bootstrap script

You only need to use one of these three methods on any given node. Choose the method that makes the most sense for your environment.

Commands Run as Root
Commands that require root privileges are run with `sudo` in these directions. If your system is not configured to use `sudo`, run these commands as root using a method appropriate for your configuration.

BUG!
If you receive the following error during the installation process it is most likely the result of CHEF-2388, which is being actively investigated.

```
/usr/lib/ruby/1.8/stringio.so: /usr/lib/ruby/1.8/stringio.so: wrong ELF class:
ELFCLASS32 - /usr/lib/ruby/1.8/stringio.so (LoadError)
```

Install Ruby

Chef is written in Ruby and thus requires Ruby as well as other system dependencies to run. Ruby 1.8.7+ is required to run Chef. Run the following command to install ruby and other required dependencies:

On Red Hat and CentOS (Version 5)

Enable AegisCo repository to get Ruby 1.8.7.

If you will be installing a Chef Server on this host, you will also need to enable the RBEL repo:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Install Ruby and other development tools:

On Red Hat and CentOS (Version 6)

Install the RBEL repo

Install Ruby and other development tools:

Install RubyGems from Source

We prefer to install RubyGems from source rather than use the OS-provided version (if any), as it is cross platform, so we know what to expect.

Install RubyGems

Install Chef Gem

To install Chef and its dependencies, run the following code:

Install Chef

If you are installing Chef to use chef-client, proceed below. If you are installing Chef-solo as a means of installing Chef Server, return to Installing Chef Server using Chef Solo. If you wish to use Chef-solo as your primary Chef client, see Chef Solo.

Configure chef-client

The initial configuration of a chef-client requires a `client.rb` configuration file and a `validation.pem` file to be placed in `/etc/chef`. At a minimum `client.rb` must contain the configuration settings necessary for chef-client to communicate with chef-server. `validation.pem` is the permission certificate for your validator client. This API client is used on the first-run of chef-client and is used to create a new API client for the node. To create these we do the following:

1. Create `/etc/chef`:

2. Create `client.rb` and `validation.pem`:

   If you have a properly configured workstation, you should be able to run the following command from your workstation's chef repository

   This will place `client.rb` and `validation.pem` in your current working directory on your chef workstation. You can then use `scp` or another method to move both files into `/etc/chef` on the node you are configuring.

   If you do not have a configured workstation, you should create a minimal `/etc/chef/client.rb`. If you are using your own chef-server `client.rb` should look as follows (substituting the correct address of your chef server):

   client.rb

   If you are using Hosted Chef, `client.rb` should look as follows (with ORGNAME replaced with your Organization Short Name}}:

   client.rb for Hosted Chef

   Your `validation.pem` can be downloaded from your chef-server if you are running your own chef-server. On your Chef Server,

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

`validation.pem` should be in `/etc/chef/`. Users of Hosted Chef can find their validator client's key from the Hosted Chef Management Console. Note that if you are using Hosted Chef, your key is named `ORGNAME-validator.pem`.

Chef client has a number of configuration settings that do not appear above. See Chef Configuration Settings for further details.

Bootstrap

It is often easiest to install Chef on a node using knife-bootstrap. Bootstrapping runs a script on the target system which install Ruby. This method requires the node to be accessible over SSH. In most cases, bootstrapping a new node will be as simple as running a single command. For instance, to bootstrap a CentOS 5.6 workstation you might run the following command:

```
knife bootstrap IP_ADDR -d centos5-gems
```

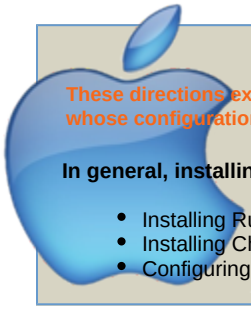
See Knife Bootstrap and Launch Cloud Instances with Knife for complete documentation. The bootstrap scripts distributed with knife for CentOS use an install method similar to the Rubygems installation methods above.

 Installing Chef Client and Chef Solo

Upgrade Instructions 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Installing Chef Client on OS X



These directions explain how to install Chef on a node running OS X via Rubygems. Nodes are hosts whose configuration you want to manage using Chef.

In general, installing chef-client on a node includes:

- Installing Ruby and Other Dependencies
- Installing Chef
- Configuring Chef



Install XCode

Install XCode from your OS X installation media or by downloading and installing it from http://developer.apple.com/xcode/

Install Ruby

Ruby 1.8.7 or better is required to run Chef. Ruby is already installed in recent versions of OS X. If you do not have Ruby installed or if your version of Ruby is too old, you can install Ruby using the information provided at http://www.ruby-lang.org/en/downloads/

Install RubyGems from Source

We prefer to install RubyGems from source rather than use the OS-provided version (if any), as it is cross platform, so we know what to expect.

Install RubyGems

Install Chef Gem

To install Chef and its dependencies, run the following code:

Install Chef

- If you are installing Chef to use chef-client, proceed below.
- If you are installing Chef-solo as a means of installing Chef Server, return to Installing Chef Server using Chef Solo.
- If you wish to use Chef-solo as your primary Chef client, see Chef Solo.

Configure chef-client

The initial configuration of a chef-client requires a `client.rb` configuration file and a `validation.pem` file to be placed in `/etc/chef`. At a minimum `client.rb` must contain the configuration settings necessary for chef-client to communicate with chef-server. `validation.pem` is the permission certificate for your validator client. This API client is used on the first-run of chef-client and is used to create a new API client for the node. To create these we do the following:

1. Create `/etc/chef`:

2. Create `client.rb` and `validation.pem`:

   If you have a properly configured workstation, you should be able to run the following command from your workstation's chef repository

   This will place `client.rb` and `validation.pem` in your current working directory on your chef workstation. You can then use `scp` or another method to move both files into `/etc/chef` on the node you are configuring.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

If you do not have a configured workstation, you should create a minimal `/etc/chef/client.rb`. If you are using your own chef-server `client.rb` should look as follows (substituting the correct address of your chef server):

client.rb

If you are using Hosted Chef, `client.rb` should look as follows (with ORGNAME replaced with your Organization Short Name}}:

client.rb for Hosted Chef

Your `validation.pem` can be downloaded from your chef-server if you are running your own chef-server. On your Chef Server, `validation.pem` should be in `/etc/chef/`. Users of Hosted Chef can find their validator client's key from the Hosted Chef Management Console. Note that if you are using Hosted Chef, your key is named `ORGNAME-validator.pem`.

Chef client has a number of configuration settings that do not appear above. See Chef Configuration Settings for further details.

 Installing Chef Client and Chef Solo

Upgrade Instructions 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Installing Chef Client on Other Operating Systems

Chef can be installed on a variety of operating systems via Rubygems.



In general, you will need to

1. Install Ruby 1.8.7+ and other dependencies
2. Install Rubygems
3. Install Chef

We recommend installing Ruby and dependencies other than Rubygems using a method common for your operating system.

Ubuntu or Debian Example

Below we give an example that works on Ubuntu or Debian. **Alter this command to the equivalent command on your system. Check the Installation Notes section for tips relevant to your platform.**

Install Ruby and other Dependencies

Chef is written in Ruby and thus requires Ruby as well as other system dependencies to run. Run the following command to install ruby and other required dependencies:



Chef 0.10 Note
There are open issues with Chef 0.10.x and Ruby 1.8.6. If you are using Chef 0.10.x, you should get 1.8.7 with SSL bindings. If you'd like to use 1.9.1 or 1.9.2, you'll want to follow the Installing Chef Client on Other Operating Systems directions to install manually with gems.

Install RubyGems from Source

We prefer to install RubyGems from source rather than use the OS-provided version (if any), as it is cross platform, so we know what to expect.

Install RubyGems

Install Chef Gem

To install Chef and its dependencies, run the following code:

Install Chef

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License







We have OS-specific instructions for the following operating systems

- Ubuntu or Debian
- CentOS, Scientific Linux, Fedora, or RHEL
- OS X
- Windows Server 2008

The bottom of this page also has tips for installing on various operating systems.

If you are installing Chef to use chef-client, proceed below. If you are installing Chef-solo as a means of installing Chef Server, return to Installing Chef Server using Chef Solo. If you wish to use Chef-solo as your primary Chef client, see Chef Solo.

Configure chef-client

The initial configuration of a chef-client requires a `client.rb` configuration file and a `validation.pem` file to be placed in `/etc/chef`. At a minimum `client.rb` must contain the configuration settings necessary for chef-client to communicate with chef-server. `validation.pem` is the permission certificate for your validator client. This API client is used on the first-run of chef-client and is used to create a new API client for the node. To create these we do the following:

1. Create `/etc/chef`:

2. Create `client.rb` and `validation.pem`:

   If you have a properly configured workstation, you should be able to run the following command from your workstation's chef repository

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

This will place `client.rb` and `validation.pem` in your current working directory on your chef workstation. You can then use `scp` or another method to move both files into `/etc/chef` on the node you are configuring.

If you do not have a configured workstation, you should create a minimal `/etc/chef/client.rb`. If you are using your own chef-server `client.rb` should look as follows (substituting the correct address of your chef server):

client.rb

If you are using Hosted Chef, `client.rb` should look as follows (with ORGNAME replaced with your Organization Short Name}}:

client.rb for Hosted Chef

Your `validation.pem` can be downloaded from your chef-server if you are running your own chef-server. On your Chef Server, `validation.pem` should be in `/etc/chef/`. Users of Hosted Chef can find their validator client's key from the Hosted Chef Management Console. Note that if you are using Hosted Chef, your key is named `ORGNAME-validator.pem`.

Chef client has a number of configuration settings that do not appear above. See Chef Configuration Settings for further details.

Bootstrap

Once you know the install procedure for your platform you can create a Custom Knife Bootstrap Script template, allowing you to automate future installations of Chef using Knife.

Installation Notes for Various Operating Systems

FreeBSD 7.1



Pre-Installation Planning

You'll need Rubygems >= 1.3.0, which isn't in the version of ports that ships with 7.1-RELEASE, so upgrade ports first.

Upgrading Ports

```
portsnap fetch
portsnap extract
```

Install Ruby and Rubygems

Make sure you have Ruby and Rubygems installed. On FreeBSD with ports installed:

Installing Ruby and Rubygems

```
cd /usr/ports/devel/ruby-gems
make install
```

OpenBSD

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Tested Versions

These instructions have been verified to work with the following versions of OpenBSD, though they may work with older versions.

OpenBSD 5.0
OpenBSD 4.9

Assumptions

This assumes that you have done a base install and have installed the compXY.tgz (i.e. comp50.tgz for OpenBSD 5.0) compiler file set during the installation. If you haven't, please see Adding a file set after install in the OpenBSD FAQ.

Pre-installation

You will want to add the following to the *~/.profile* of the root user or a user capable of running *pkg_add(1)* as root:

Prepare PKG_PATH

```
PKG_PATH=ftp://ftp3.usa.openbsd.org/pub/OpenBSD/`uname -r`/packages/`machine -a`/
export PKG_PATH
```

You may wish to change the location of the **PKG_PATH** to a mirror closer to your location. Once set, either source the new *.profile* or logout and login again.

Install Prerequisites

The following will install ruby, rubygems, and ruby-iconv.

Installing Ruby, Rubygems and other prerequisites

```
sudo pkg_add ruby-gems ruby-iconv
```

You will want to make the installed binaries the default for your installation:

Post-install Administrivia

```
sudo ln -sf /usr/local/bin/ruby18 /usr/local/bin/ruby
sudo ln -sf /usr/local/bin/erb18 /usr/local/bin/erb
sudo ln -sf /usr/local/bin/irb18 /usr/local/bin/irb
sudo ln -sf /usr/local/bin/rdoc18 /usr/local/bin/rdoc
sudo ln -sf /usr/local/bin/ri18 /usr/local/bin/ri
sudo ln -sf /usr/local/bin/testrb18 /usr/local/bin/testrb
sudo ln -sf /usr/local/bin/gem18 /usr/local/bin/gem
```

Gentoo

Installing Chef using Ebuilds

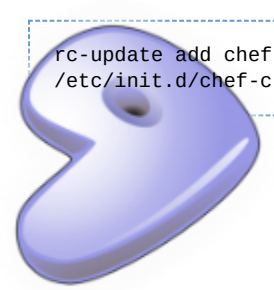
You will need to set some package.keywords, then simply emerge the Chef client.

```
vim /etc/portage/package.keywords
emerge chef
```

You may start using `chef-solo` or edit `/etc/chef/client.rb` (at least edit `chef_server_url`) and start the client with it's init script.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
rc-update add chef-client default
/etc/init.d/chef-client start
```



OpenSolaris





Installing Ruby

From Download Ruby:

- Ruby 1.8.7 are available for Solaris 8 through Solaris 10 on Sunfreeware and Ruby 1.8.7 is available at Blastwave. Using RVM can get you the latest version of Ruby 1.9.2.
- To install Ruby on OpenIndiana, please use the Image Packaging System, or IPS client. This will install the latest Ruby binaries and Rubygems directly from the OpenSolaris network repository for Ruby 1.9. It's easy:

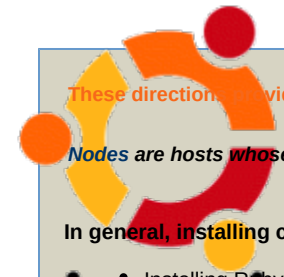
```
% pkg install runtime/ruby-18
```

 Installing Chef Client and Chef Solo

Upgrade Instructions 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Installing Chef Client on Ubuntu or Debian

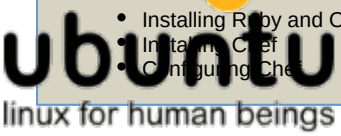


These directions provide two methods of installing Chef client on a node running Ubuntu or Debian.

Nodes are hosts whose configuration that you want to manage using Chef.

In general, installing chef-client on a node includes:

- Installing Ruby and Other Dependencies
- Installing Chef
- Configuring Chef







Typically, chef-client is installed on Ubuntu and Debian systems using one of three possible methods:

1. Using the Opscode package repository,
2. Using Rubygems, or
3. Using a `knife bootstrap` script

You only need to use one of these three methods on any given node. Choose the method that makes the most sense for your environment.



Commands Run as Root
Commands that require root privileges are run with `sudo` in these directions. If your system is not configured to use `sudo`, run these commands as root using a method appropriate for your configuration.

Package Installation

Installing using native Ubuntu or Debian ensures that Chef is installed in the same way as other software on your system. These instructions use the Opscode APT repository. Opscode maintains the packages in this APT repo, as well as the packages that get included in Debian and Ubuntu's repositories. The following distributions are supported within the Opscode APT repository.

- Ubuntu 11.10 oneirc
- Ubuntu 11.04 natty
- Ubuntu 10.10 maverick
- Ubuntu 10.04 lucid
- Ubuntu 9.10 karmic
- Debian (unstable) sid
- Debian (testing) wheezy
- Debian 6 (stable) squeeze
- Debian 5 (oldstable) lenny

Add the Opscode APT Repository

Create `/etc/apt/sources.list.d/opscode.list`.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

/etc/apt/sources.list.d/opscode.list

- For Chef 0.9.x, replace codename with the supported distribution codename, such as "lucid".
- For Chef 0.10.x, replace codename with the codename, suffixed with "-0.10", for example, "lucid-0.10".

If you would like to be able to download source packages, add an additional identical line, but change `deb` to `deb-src` line.

You can copy and paste these examples to create the necessary sources.list entry:

Ubuntu for Chef 0.9.x

Ubuntu for Chef 0.10.x

Debian users will likely need to run 'apt-get install sudo wget lsb-release' as root before pasting the examples.

Add the GPG Key and Update Index

Before you install the packages, make sure you add the Opscode GPG key to apt.

Issues downloading from gnupg.net?
If you get an error when trying to download the key that states that the "keyserver timed out", try downloading the key directly from our apt repository instead:

Afterwards you'll also want to run this command again:

Now, we update apt-get with the newly added Opscode repository:

Installing the opscode-keyring package will keep the key up-to-date:

Upgrade Existing Packages

To ensure you are using the latest versions of libraries that chef depends on, you may wish to update your existing packages:

Install Chef using apt-get

WIth the Opscode repository installed, you can now install chef using apt-get:

```
sudo apt-get install chef
```

You will be prompted for the Chef Server URL.

- If you are using Hosted Chef, enter your organization's Hosted Chef URL, e.g., "https://api.opscode.com/organizations/ORGNAME".
- If you installed a local Chef Server, enter your Chef Server's FQDN, e.g., "http://chef.example.com:4000".
- If you are using chef-solo, enter "none".

Upon installation, this will:

- Install all the dependencies for Chef as a client.
- Start chef-client via /etc/init.d/chef-client.
- Configure /etc/chef/client.rb

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

After installation, you should move your validation client's private key onto the node and copy it to `/etc/chef/validation.pem`.

Hosted Chef Users should add the following line to the end of `/etc/chef/client.rb`, replacing ORGNAME with your organization name:

```
validation_client_name 'ORGNAME-validator'
```

If you are planning to run chef-solo or use just knife, you can stop and disable the chef-client service. An example config file for chef-solo is created in /usr/share/chef/solo.rb.

Install Supporting Packages

The following packages are not required to run the chef-client or chef-solo and are optional.

- libopenssl-ruby - if the chef-server has SSL (like the Opscode Platform), this is required.
- ruby-dev and build-essential - required to build gems that have native extensions from source with gem_package

Scripting the apt-get installation

If you plan on using this from a bash script, you can preseed the values so no popup occurs.

```
echo "chef chef/chef_server_url string https://api.opscode.com/organizations/ORGNAME" | sudo
debconf-set-selections && sudo apt-get install chef -y
```

Rubygem

Install Ruby and other Dependencies

Chef is written in Ruby and thus requires Ruby as well as other system dependencies to run. Run the following command to install ruby and other required dependencies:

Chef 0.10 Note
There are open issues with Chef 0.10.x and Ruby 1.8.6. If you are using Chef 0.10.x, you should get 1.8.7 with SSL bindings. If you'd like to use 1.9.1 or 1.9.2, you'll want to follow the Installing Chef Client on Other Operating Systems directions to install manually with gems.

Install RubyGems from Source

We prefer to install RubyGems from source rather than use the OS-provided version (if any), as it is cross platform, so we know what to expect.

Install RubyGems

Install Chef Gem

To install Chef and its dependencies, run the following code:

Install Chef

If you are installing Chef to use chef-client, proceed below. If you are installing Chef-solo as a means of installing Chef Server, return to Installing Chef Server using Chef Solo. If you wish to use Chef-solo as your primary Chef client, see Chef Solo.

Configure chef-client

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

The initial configuration of a chef-client requires a `client.rb` configuration file and a `validation.pem` file to be placed in `/etc/chef`. At a minimum `client.rb` must contain the configuration settings necessary for chef-client to communicate with chef-server. `validation.pem` is the permission certificate for your validator client. This API client is used on the first-run of chef-client and is used to create a new API client for the node. To create these we do the following:

1. Create `/etc/chef`:

2. Create `client.rb` and `validation.pem`:

   If you have a properly configured workstation, you should be able to run the following command from your workstation's chef repository

   This will place `client.rb` and `validation.pem` in your current working directory on your chef workstation. You can then use `scp` or another method to move both files into `/etc/chef` on the node you are configuring.

   If you do not have a configured workstation, you should create a minimal `/etc/chef/client.rb`. If you are using your own chef-server `client.rb` should look as follows (substituting the correct address of your chef server):

   client.rb

   If you are using Hosted Chef, `client.rb` should look as follows (with ORGNAME replaced with your Organization Short Name}}:

   client.rb for Hosted Chef

   Your `validation.pem` can be downloaded from your chef-server if you are running your own chef-server. On your Chef Server, `validation.pem` should be in `/etc/chef/`. Users of Hosted Chef can find their validator client's key from the Hosted Chef Management Console. Note that if you are using Hosted Chef, your key is named `ORGNAME-validator.pem`.

Chef client has a number of configuration settings that do not appear above. See Chef Configuration Settings for further details.

Bootstrap

It is often easiest to install Chef on a node using knife-bootstrap. Bootstrapping runs a script on the target system which install Ruby, install chef-client, configure chef-client, and start a chef-client run. This method requires the node to be accessible over SSH. In most cases, bootstrapping a new node will be as simple as running a single command. For instance, to bootstrap an Ubuntu workstation using the APT-based installation method you might run the following command:

```
knife bootstrap IP_ADDR -d ubuntu10.04-apt --sudo
```

See Knife Bootstrap and Launch Cloud Instances with Knife for complete documentation. The bootstrap scripts distributed with knife for Ubuntu use an install method similar to the installation methods above.

Installing Chef Client and Chef Solo

Upgrade Instructions

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Installing Chef Client on Windows



The following instructions walk you through the steps involved to prepare a Windows node for management with Chef and the `chef-client`.

If you would like a more automated approach to bootstrapping your Windows nodes, please look Windows-specific knife bootstrap subcommands that ship as part of the Knife Windows Bootstrap plugin.

Use of these knife bootstrap subcommands is the preferred way to prepare a node for management by Chef.

If you are using Windows 2008 R2 with Hosted Chef and would like to set your node up as a workstation and a client, you can use the Fast Start Guide for Windows. This guide also contains a quick walkthrough on using recipes on the client.

Install Chef

Download the Chef Full Installer for Windows and open it. Choose defaults for any options.

Once it is installed there will be no icon for it. If needed, you can confirm it was installed correctly with these commands in a new command prompt window:

```
C:\> tar --version
bsdtar 2.8.3 - libarchive 2.8.3

C:\> chef-client --version
```

You should see "Chef: 0.10.8" returned as the version number.

Configure Chef

Now we will need to copy over the validation key and create the client.rb so chef-client will be able to authenticate with your organization or Chef Server.

Copy over the validation key

First you will want to create a file in C:\ named "chef" so we can put configuration information into it. You can move this file to another location, but you would need to change the paths to it in your client.rb.

You'll then want to copy the validation key onto this node:

- **Hosted Chef**
  Copy your ORGANIZATION-validator.pem key to `C:\chef\ORGANIZATION-validator.pem`.
- **Open Source Chef Server**
  Copy your validation.pem key to `C:\chef\validation.pem`.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Opscode Windows Cookbooks

Providing the setup, automation and maintenance of Windows-based servers and applications

- **Powershell**
- **IIS**
- **SQL-Server**
- **7-zip**
- **Windows Primitives**
- **Web Platform Installer (WebPI)**
- **Windows Installer XML (WiX)**

Custom LWRP for IIS 7/7.5 Configuration

iis7 - is a community developed custom Lightweight Resource and Provider, to create and configure websites in IIS 7 / 7.5.

Create client.rb

With wordpad or your favorite editor, create a file in C:\chef named client.rb.

What you place in it's contents is dependent on the type of Chef install you have:

- **Hosted Chef**
  Change ORGANIZATION to the name of your organization when creating this file. Put the following in it's contents:

```
log_level :info
log_location STDOUT
chef_server_url "https://api.opscode.com/organizations/ORGANIZATION"
validation_client_name "ORGANIZATION-validator"
validation_key  "c:/chef/ORGANIZATION-validator.pem"
client_key "c:/chef/client.pem"
file_cache_path   "c:/chef/cache"
file_backup_path  "c:/chef/backup"
cache_options     ({:path => "c:/chef/cache/checksums", :skip_expires => true})
```

  Note that this file uses forward slashes [/] and not backslashes [\ ].

- **Open Source Chef Server**
  Change the chef_server_url when creating this file to point to your chef server and include the port if you are not using port 80. Put the following in it's contents:

```
log_level :info
log_location STDOUT
chef_server_url 'http://CHEF-SERVER-URL:4000'
validation_client_name "chef-validator"
validation_key  "c:/chef/validation.pem"
client_key "c:/chef/client.pem"
file_cache_path   "c:/chef/cache"
file_backup_path  "c:/chef/backup"
cache_options     ({:path => "c:/chef/cache/checksums", :skip_expires => true})
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Note that this file uses forward slashes [/] and not backslashes [\ ].

Run the chef-client

Right now you have to run `chef-client` with a `-c` option and the full path to the client.rb file:

```
C:\> chef-client -c c:\chef\client.rb
```

This requirement will go away once Chef adjusts it's default paths based on the platform; follow CHEF-2317 for updates.

If this command completes without errors, your install is now completed.

Known issues - addressed with version 0.10
Chef versions earlier than 0.10 will encounter these issues:

- `knife cookbook upload` is known to not work.
  There is no Windows work-around to this issue on being unable to upload cookbooks from a Windows client, other than the above stated approach of running Chef in a VM on their Windows system or manually moving them to the appropriate directory.
- If you're using Ruby 1.9.2, you no longer need the win32-open3 gem, but things will be broken until you are using chef v 0.10.

Ohai

If you are using Ohai 0.6.x or higher, the following issue has been addressed. If you're running Ohai 0.5.6 on Windows 7 or Server 2008, add the following to the top of your client.rb :

```
require 'win32ole'
WIN32OLE.codepage = WIN32OLE::CP_UTF8
```

If you find that the start time on your Windows installation seems excessively slow, you may choose to disable a number of the Ohai plugins that aren't required for Chef to function on Windows. To do this, you would add the following to your client.rb file:

```
Ohai::Config[:disabled_plugins] = [
"c",
"groovy",
"lua",
"java",
"mono",
"passwd",
"perl",
]
```

Some Resources/Providers Are Known To Not Work

Not all Resources work on windows yet. The Following Resources do not work in windows:

- Cron
- Deploy
- Erlang call
- Git
- Http request
- Ifconfig
- Link
- Mdadm
- Package (but gem_package - for ruby gems, does work)

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- Route
- SCM
- Script
- Subversion

Also, using 'user' or 'group' attributes within a resource, specifically the template or directory resource is known to cause issues. Related bug: CHEF-2633

Bootstrapping Windows Nodes

The knife-windows Knife plugin has some additional Windows-specific bootstrap subcommands. Although these bootstrap subcommands target Windows nodes they are written in pure Ruby and thus can be run from any platform Chef supports. So yes you can use your MacBook Pro or Linux box to bootstrap your Windows nodes!

Please see the **knife-windows README** for in depth usage instructions.

 Installing Chef Client and Chef Solo

Upgrade Instructions 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Upgrade Instructions

This page collects the various guides on how to upgrade Chef from one version to another.



- *Upgrading Chef 0.8.x RubyGems*
- *Upgrading Chef 0.8.x to 0.9.x*
- *Prepare Debian and Ubuntu for 0.9 Upgrade*
- *Upgrading Chef 0.9.x to Chef 0.10.x*
- *Using Chef to Upgrade Chef*

Installation

Vagrant

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Upgrading Chef 0.8.x RubyGems

Upgrading Chef RubyGems in the version 0.8.x series is generally easy.

Upgrading to 0.9.x from 0.8.x
Upgrading to 0.9.x from 0.8.x requires you to upgrade your clients and server in lock-step. Please see Upgrading Chef 0.8.x to 0.9.x

Client

```
sudo /etc/init.d/chef-client stop
sudo gem uninstall -aIx chef # if mixlib-authentication needs update remove that too.
sudo gem cleanup
sudo gem install chef
sudo /etc/init.d/chef-client start
```

Server

```
sudo /etc/init.d/chef-client stop
for x in solr-indexer solr server server-webui
do
  sudo /etc/init.d/chef-$x stop
done
ps awux | egrep "chef|merb|solr" # verify the services are no longer running
sudo gem install chef chef-server chef-server-api chef-server-webui chef-solr
sudo gem cleanup
for x in solr-indexer solr server server-webui
do
  sudo /etc/init.d/chef-$x start
done
```

Upgrading to 0.8.10
When upgrading from earlier versions in the 0.8 series to 0.8.10, restart rabbitmq so the consumer ID gets recreated correctly.

```
sudo /etc/init.d/rabbitmq-server restart
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Upgrading Chef 0.8.x to 0.9.x

This guide is for the installation methods supported by Opscode (RubyGems and Debian packages).

Though it is a smoother transition than from 0.7 to 0.8, upgrading from 0.8 to 0.9 must be done lock-step. That is, 0.8 client and server are incompatible with 0.9 client and server.

New configuration values available:

- `sandbox_path` - `/var/chef/sandboxes`
- `checksum_path` - `/var/chef/checksums`

These are for the new cookbook upload process to the Chef Server. You will need to re-upload your cookbooks to the Chef server using Knife.

With that in mind, the basic upgrade process for chef-server is described below. As always, remember to make backups and try the upgrade in a test environment first.

Stop Server Services

The deb packaging will stop the services for you. If you're using the RubyGems installation, you'll need to stop the services the server runs manually.

```
for svc in server server-webui solr-indexer solr
do
  sudo /etc/init.d/chef-${svc} stop
done

# client only:
sudo /etc/init.d/chef-client stop
```

Make sure the 'merb' processes for the chef-server and chef-server-webui are no longer running. They will look like this when running (port 4000 is the chef-server, port 4040 is the chef-server-webui):

```
root      1035  0.1  0.0 112216  4276 ?        Sl   Jul07   5:44 merb : merb : master
root      1784  6.5  0.6 146760 51984 ?        S    Jul07 290:40 merb : worker (port 4040)
root      7547  0.1  0.4 110788 34104 ?        Sl   Jul09   3:00 merb : merb : master

root      7554  5.8  0.7 149912 57820 ?        S    Jul09 125:16 merb : worker (port 4000)
```

Upgrade Chef Server

For RubyGems installation:

```
# server:
sudo gem install chef chef-server
```

Note that the 'chef-server' gem depends on chef-solr, chef-server-api and chef-server-webui. If you do not want the webui:

```
# server only w/ API:
sudo gem install chef chef-server-api chef-solr
```

For a Debian/Ubuntu installation:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
sudo apt-get update
sudo apt-get install chef chef-server
```

Modify /etc/chef/server.rb

If you wish to use non-default values for sandbox_path and checksum_path, edit the server configuration file (server.rb). If you are using the Chef cookbook to manage this, update the template.

The Debian packaging already handles this for you.

Restart the Services

This is handled by the deb packaging for you.

```
for svc in server server-webui solr-indexer solr
do
  sudo /etc/init.d/chef-${svc} start
done

# client only:
sudo /etc/init.d/chef-client start
```

Re-upload all cookbooks.

You need to upload all the cookbooks. If your chef-repo is on the same system as the chef-server, then simply run:

```
knife cookbook upload -a
```

From the directory where your chef-repo is. If your chef-repo is on a different system such as your local workstation, then upgrade the Chef gem and run the knife command above.

Upgrade Clients

Upgrading the clients can now be done. Simply install the new gem and restart the chef-client service.

```
sudo gem install chef
sudo /etc/init.d/chef-client restart
```

Debian packaging; the service will be stopped and started for you.

```
sudo apt-get update
sudo apt-install chef
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Prepare Debian and Ubuntu for 0.9 Upgrade

This document describes the steps required to prepare for updating from 0.7.x Debian/Ubuntu packages to 0.9.



Coming Soon
The apt.opscode.com repository will be updated soon with the new packages!

First, be aware of Breaking Changes from 0.7.x to 0.8.x; For more information about Chef version 0.9, see the 0.9.0 release post on the Opscode Blog.

Pin Package Versions

First, pin the package versions of the installed Chef packages so you don't accidentally get upgraded. On systems where Chef 0.7.x is installed, add the following to `/etc/apt/preferences`.

/etc/apt/preferences

```
Package: *chef*
Pin: version 0.7*
Pin-Priority: 1001
```

Or use dpkg to hold packages.

```
for x in chef chef-indexer chef-server chef-server-slice libchef-ruby libchef-ruby1.8
do
  echo "$x hold" | sudo dpkg --set-selections
done
```

Important Changes

- chef-server-slices is deprecated, replaced by chef-server-api and chef-server-webui.
- chef-indexer is deprecated, replaced by chef-solr.
- The server API, webui and SOLR daemons run as `chef` user/group.
- init scripts are configured with `/etc/default/chef*` like many other Debian packages.
- chef-server-webui is optional.
- The webui daemon process runs on port 4040.
- The webui no longer uses openid. You will be prompted for a password for the `admin` user during chef-server-webui installation.
- Registrations are now called clients.
- Clients are validated with a certificate, `/etc/chef/validation.pem`. This is created when chef-server starts. Clients are no longer validated through the webui. See Authentication.

Upgrading

When ready to upgrade the server, remove the pin entries from `/etc/apt/preferences` and install the new version.

```
sudo apt-get install chef chef-server chef-server-webui
```



RubyGems
You should have RubyGems installed either from source or from package, or merb will fail to launch the chef-server and chef-server-webui daemons. Add it to the packages list here if it is not currently installed, or install from source.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

If you run into issues, see the troubleshooting section below.

After the server is up and running again, configure Knife.

```
% sudo knife configure -i
Your chef server URL? [http://localhost:4000]
Select a user name for your new client: [jtimberman]
Your existing admin client user name? [chef-webui]
The location of your existing admin key? [/etc/chef/webui.pem]
Your validation client user name? [chef-validator]
The location of your validation key? [/etc/chef/validation.pem]
Path to a chef repository (or leave blank)? /home/jtimberman/chef-repo
WARN: Creating initial API user...
INFO: Created (or updated) client[jtimberman]
WARN: Configuration file written to /home/jtimberman/.chef/knife.rb
```

Then in your Chef Repository, run rake install.

```
% cd ~/chef-repo
% rake install
```

Your client systems will need to have the validation.pem copied to their `/etc/chef/validation.pem`, and chef-client run, or you will need to create clients using "`knife client create`".

Upgrade Troubleshooting

Debian packages preserve existing configuration files. When configuring the chef-server package, `amqp_pass` may not be set, preventing the chef-server from starting. The installation prompts for this setting, and it gets written to `/etc/chef/solr.rb` but not an existing `/etc/chef/server.rb`. Add the value to /etc/chef/server.rb and restart the server.

```
% sudo grep amqp_pass /etc/chef/server.rb
amqp_pass "my_awesome_password"
% sudo /etc/init.d/chef-server restart
% sudo /etc/init.d/chef-server-webui restart
```

If the webui is installed, restart it, because when the API failed to start, the webui doesn't get initialized and create the admin user with the password specified.

Because the daemon processes now run as the unprivileged `chef` user, some paths may be unwritable.

```
% sudo chown chef:chef -R /var/chef /srv/chef /var/*/chef
```

When opening the webui (now on port 4040), the server may toss a "Tampered with Cookie" error page. Clear out the cookie `_chef_server_session_id` and reload the page.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Upgrading Chef 0.9.x to Chef 0.10.x

Before Upgrading

Before upgrading, make sure you have backups that you can recover from. When you're ready to proceed, stop the chef-server, chef-server-webui, chef-solr, and chef-solr-indexer on the server, and chef-client on the clients.

```
for svc in server server-webui solr solr-indexer
do
  sudo /etc/init.d/chef-${svc} stop
done

# client only:
sudo /etc/init.d/chef-client stop
```

Make sure the 'merb' processes for the chef-server and chef-server-webui are no longer running. They will look like this when running (port 4000 is the chef-server, port 4040 is the chef-server-webui):

```
root      1035  0.1  0.0 112216  4276 ?        Sl   Jul07   5:44 merb : merb : master
root      1784  6.5  0.6 146760 51984 ?        S    Jul07 290:40 merb : worker (port 4040)
root      7547  0.1  0.4 110788 34104 ?        Sl   Jul09   3:00 merb : merb : master
root      7554  5.8  0.7 149912 57820 ?        S    Jul09 125:16 merb : worker (port 4000)
```

Install gecode on the Servers

Gecode is a pre-requisite of dep_selector gem, which Chef Server depends on. It's only needed on the server side.

Install Using Deb Package

On Ubuntu lucid, maverick, or Debian lenny release, you can install gecode deb package from the Opscode APT repository.

Add the Opscode APT Repository

Create `/etc/apt/sources.list.d/opscode.list`.

/etc/apt/sources.list.d/opscode.list

- For Chef 0.9.x, replace codename with the supported distribution codename, such as "lucid".
- For Chef 0.10.x, replace codename with the codename, suffixed with "-0.10", for example, "lucid-0.10".

If you would like to be able to download source packages, add an additional identical line, but change `deb` to `deb-src` line.

You can copy and paste these examples to create the necessary sources.list entry:

Ubuntu for Chef 0.9.x

Ubuntu for Chef 0.10.x

Debian users will likely need to run 'apt-get install sudo wget lsb-release' as root before pasting the examples.

Add the GPG Key and Update Index

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Before you install the packages, make sure you add the Opscode GPG key to apt.

Issues downloading from gnupg.net?
If you get an error when trying to download the key that states that the "keyserver timed out", try downloading the key directly from our apt repository instead:

Afterwards you'll also want to run this command again:

Now, we update apt-get with the newly added Opscode repository:

Installing the opscode-keyring package will keep the key up-to-date:

Upgrade Existing Packages

To ensure you are using the latest versions of libraries that chef depends on, you may wish to update your existing packages:

Install deb package

```
sudo apt-get install libgecode-dev
```

Build and Install from Source

```
cd /tmp
curl -C - -O  http://www.gecode.org/download/gecode-3.5.0.tar.gz
tar zxvf gecode-3.5.0.tar.gz
cd gecode-3.5.0 && ./configure
make && make install
```

Upgrade Chef Server

For RubyGems installation:

```
# server:
sudo gem install chef chef-server
```

For a Debian/Ubuntu installation:

```
sudo apt-get update
sudo apt-get install chef chef-server
```

Upgrade Chef Solr and Install Chef Expander

Chef 0.10 includes an upgraded version of Solr and uses a different schema with Solr, so you will need to upgrade Solr and then rebuild the index. Before doing anything, make sure to back up your Solr Data directory. By default the data directory is located at `/var/chef/solr/data`, though it may be located elsewhere if you've installed from packages or customized the installation. You can backup your Solr data by stopping Solr and making a copy of the data directory and its contents.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Upgrade Solr

For a Debian/Ubuntu installation:

Skip chef-solr-installer on Ubuntu/Debian
Skip this section on upgrading Solr if you're upgrading using debs on Ubuntu/Debian. The deb packages automatically install this for you. If you get complaints about "start.config" not being found when launching chef-solr after the upgrade, try running:

```
rm /etc/chef/solr.rb
apt-get install -f chef-solr
```

You will need to rebuild your indexes after the upgrade. To do so, run:

```
knife index rebuild
```

Once your data is backed up, run `chef-solr-installer` to upgrade your Chef Solr installation. Answer "yes" when prompted for confirmation. The process should look like this:

```
yourshell> chef-solr-installer
Configuration setting solr_heap_size is unknown and will be ignored

Chef Solr is already installed in /var/chef/solr
Do you want to overwrite the current install? All existing Solr data will be lost. [y/n] y
Removing the existing Chef Solr installation
  rm -rf /var/chef/solr
  rm -rf /var/chef/solr-jetty
  rm -rf /var/chef/solr/data
Creating Solr Home Directory
  mkdir -p /var/chef/solr
  entering /var/chef/solr
  tar zxvf /Users/ddeleo/opscode/chef/chef-solr/solr/solr-home.tar.gz
Creating Solr Data Directory
  mkdir -p /var/chef/solr/data
Unpacking Solr Jetty
  mkdir -p /var/chef/solr-jetty
  entering /var/chef/solr-jetty
  tar zxvf /Users/ddeleo/opscode/chef/chef-solr/solr/solr-jetty.tar.gz

Successfully installed Chef Solr.
You can restore your search index using `knife index rebuild`
```

Replace Chef Solr Indexer with Chef Expander

In Chef 0.10+, chef-solr-indexer is replaced by Chef Expander. Complete documentation for Chef Expander is provided on the Chef Indexer architecture page.. To start Chef Expander using your existing solr.rb configuration:

```
shell> chef-expander -n 1
```

If you wish to use a configuration file other than `/etc/chef/solr.rb`, pass the config file with `-c`:

```
shell> chef-expander -n 1 -c /etc/custom-location/config-file.rb
```


This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Make Chef Expander a runit Service

If you manage the Chef-Server with chef-solo, you can use the update your cookbook that manages the server to add Chef Expander as a runit service.

For example:

In the recipe you use through chef-solo to manage your server

```
runit_service "chef-expander"
```

Add templates sv-chef-expander-run.erb

```
#!/bin/sh
PATH=/usr/local/bin:/usr/local/sbin:/bin:/sbin:/usr/bin:/usr/sbin<% if
@node[:languages][:ruby][:gems_dir] %>:<%= @node[:languages][:ruby][:gems_dir] %>/bin<% end -%>
exec 2>&1
exec /usr/bin/env chef-expander -n 1
```

Add template sv-chef-expander-log-run.erb

```
#!/bin/sh
exec svlogd -tt ./main
```

Make sure you remove chef-server-indexer from your recipe before running chef-solo.

Remove Chef-Solr-Indexer Service

Delete chef-solr-indexer in `/etc/sv`, `/etc/service/` and `/etc/init.d`.

Start the Services

This is handled by the deb packaging for you.

```
for svc in server-webui solr expander server
do
  sudo /etc/init.d/chef-${svc} start
done
```

Upgrade Clients

Upgrading the clients can now be done. Simply install the new gem or package and restart the chef-client service.

```
sudo gem install chef
sudo /etc/init.d/chef-client restart
```

Debian packaging; the service will be stopped and started for you.

```
sudo apt-get update
sudo apt-install chef
```

Rebuild Your Index

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Once Solr has been upgraded and you have chef-expander and all other chef-server services running, you can rebuild your index by running `knife index rebuild` from whatever machine you normally run knife from.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Using Chef to Upgrade Chef

Overview

We do not recommend the approach of having Chef upgrade itself. It is tricky from a process management perspective during the upgrade process, and can also result in you being surprised by something in the new release. We recommend following the specific guide for the upgrade being performed. That said, if there is some reason that you'd like to proceed with this approach within your environment, the following information may be helpful.

Upgrading via apt

Doing an upgrade through package distribution, as done on the individual version update guides, causes apt to restart chef-client. So in order to update chef via apt with chef, you'll need to fork off a new process that can survive the chef process being killed. For example:

```
fork do
    Process.setsid
    # code to upgrade the package
  end

  Chef::Log.info("Exiting for upgrade...")
  exit! 0
```

Clearly, not as nice as using the package resource. Alternatively, you could create a custom resource by using Chef's Lightweight Resources and Providers (LWRP) DSL.

Install via apt with Knife Use to Upgrade

As mentioned, it's tricky to have chef upgrade itself. For most software that gets significant updates, chef included, we don't recommend having the system update automatically.

If doing so, when using apt packages, we would use action `:install`, not `:upgrade`. This greatly reduces the potential of being surprised by anything in a new release. You can use knife to upgrade individual environments when you're ready to test and deploy. A few knife examples for consideration:

1. all systems

   ```
   knife ssh 'name:[* TO *]' 'sudo apt-get update ; sudo apt-get install chef'
   ```

2. if you've set up an attribute for each environment

   ```
   knife ssh 'environment:preprod' 'sudo apt-get update ; sudo apt-get install chef'
   ```

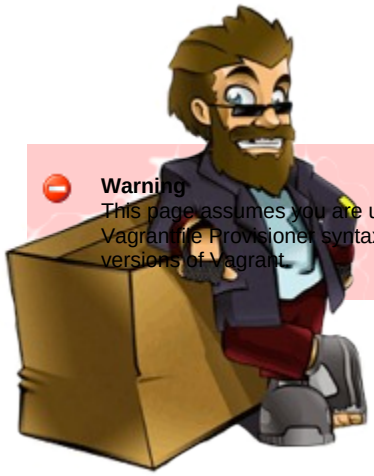
3. Chef 0.10 Environments

   ```
   knife ssh -E preprod 'name:[* TO *]' 'sudo apt-get update ; sudo apt-get install chef'
   ```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Vagrant

Vagrant is a tool for building and distributing virtualized development environments.



Vagrant can be a great way to leverage production Chef cookbooks in development and test/qa. It can also be a great way to develop and test new cookbooks on a local workstation.

Warning
This page assumes you are using Vagrant 0.7+ and VirtualBox 4+. Please be sure to update both before proceeding. The Vagrantfile Provisioner syntax has undergone massive changes so be sure to not use the following Vagrantfile with older versions of Vagrant.

Using Vagrant with Hosted Chef

(formerly the Opscode Platform)

The following Vagrantfile can be used to connect a Vagrant instance to Hosted Chef:

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
# Before running vagrant, export the shell variable for the organization on Hosted Chef and
# make sure the validator certificate is in ~/.chef.
#
# export ORGNAME=your_platform_organization
#
# Also be sure to export the shell variable for the vagrant box (linux flavor) you will be using
#
# export VAGRANT_BOX=ubuntu10.04-gems
#
# You can optionally export a shell variable for your Chef server username if it is different
# from your OS user export OPSCODE_USER=bofh

user = ENV['OPSCODE_USER'] || ENV['USER']
base_box = ENV['VAGRANT_BOX'] || 'lucid32'

Vagrant::Config.run do |config|
  config.vm.box = base_box

  config.vm.provision :chef_client do |chef|

    # Set up some organization specific values based on environment variable above.
    chef.chef_server_url = "https://api.opscode.com/organizations/#{ENV['ORGNAME']}"
    chef.validation_key_path = "#{ENV['HOME']}/.chef/#{ENV['ORGNAME']}-validator.pem"
    chef.validation_client_name = "#{ENV['ORGNAME']}-validator"

    # Change the node/client name for the Chef Server
    chef.node_name = "#{user}-vagrant"

    # Put the client.rb in /etc/chef so chef-client can be run w/o specifying
    chef.provisioning_path = "/etc/chef"

    # logging
    chef.log_level = :info

    # adjust the run list to suit your testing needs
    chef.run_list = [
      "recipe[some_cookbook::some_recipe]"
      "role[some_role]"
    ]
  end
end
```

This Vagrantfile requires the export of the following environment variables before using it:

- **ORGNAME** = your Hosted Chef organization name.
- **VAGRANT_BOX** = the vagrant box you will be using. More information about Vagrant Boxes can be found in the Vagrant documentation.



Setting up development environments with Vagrant and Hosted Chef ~ **from Cloudspace**

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

More Information

- Our friends at Automation Inc. put together a blog post on **Three easy steps to test/install a Chef Server 0.10**
- Our friends at Carbon Five put together a blog post on **provisioning VMs with Vagrant and Chef Solo**.
- Community member Fadhli Rahim **tutorial on use of Vagrant, Veewee, Chef Solo and knife-solo** for creating servers to deploy code that simulate servers in a production environment.
- **Getting Started With Vagrant**
  Vagrant uses Oracle's VirtualBox to create its virtual machines and then uses Chef to provision them.

Creating New Vagrant Boxes with Veewee

Opscode Vagrant Boxes
Opscode is no longer maintaining Vagrant boxes as we do not widely use Vagrant internally. You can use the following instructions to build your own Vagrant boxes with the latest version of Chef installed using Veewee. It only takes about 30 minutes.

Veewee by Patrick Debois is a great tool for automating Vagrant base box creation. Follow the instructions below to create base boxes that function similarly to the Knife Bootstrap templates that bear the same name.

Use `vagrant basebox templates` to generate a list of available base box templates for Veewee. We're going to use Ubuntu 10.04 server 64bit.

```
% vagrant basebox templates
...
vagrant basebox define '<boxname>' 'ubuntu-10.04.2-server-amd64'
...
```

Define a basebox that will be built similar to `ubuntu10.04-gems` bootstrap template.

```
vagrant basebox define ubuntu10.04-gems ubuntu-10.04.2-server-amd64
```

We're going to leave the generated definition.rb and preseed.cfg alone and only customize the postinstall.sh. By default, this installs Ruby Enterprise Edition from source, but we're going to use the Ruby provided by Ubuntu instead.

postinstall.sh

```
# postinstall.sh created from Mitchell's official lucid32/64 baseboxes

# Apt-install various things necessary for Ruby, guest additions,
# etc., and remove optional things to trim down the machine.
apt-get -y update
apt-get -y upgrade
apt-get -y install linux-headers-$(uname -r) build-essential
apt-get -y install zlib1g-dev libssl-dev libreadline5-dev

# Setup sudo to allow no-password sudo for "admin"
cp /etc/sudoers /etc/sudoers.orig
sed -i -e '/Defaults\s\+env_reset/a Defaults\texempt_group=admin' /etc/sudoers
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
sed -i -e 's/%admin ALL=(ALL) ALL/%admin ALL=NOPASSWD:ALL/g' /etc/sudoers

# Install NFS client
apt-get -y install nfs-common

apt-get install -y ruby ruby1.8-dev build-essential wget libruby-extras libruby1.8-extras
apt-get clean

cd /tmp
wget http://production.cf.rubygems.org/rubygems/rubygems-1.7.2.tgz
tar zxf rubygems-1.7.2.tgz
cd rubygems-1.7.2
ruby setup.rb --no-format-executable
cd ..
rm -rf rubygems-1.7.2

gem install chef --no-rdoc --no-ri --verbose

# Installing vagrant keys
mkdir /home/vagrant/.ssh
chmod 700 /home/vagrant/.ssh
cd /home/vagrant/.ssh
wget --no-check-certificate 'http://github.com/mitchellh/vagrant/raw/master/keys/vagrant.pub' -O
authorized_keys
chmod 600 /home/vagrant/.ssh/authorized_keys
chown -R vagrant /home/vagrant/.ssh

# Installing the virtualbox guest additions
VBOX_VERSION=$(cat /home/vagrant/.vbox_version)
cd /tmp
wget http://download.virtualbox.org/virtualbox/$VBOX_VERSION/VBoxGuestAdditions_$VBOX_VERSION.iso
mount -o loop VBoxGuestAdditions_$VBOX_VERSION.iso /mnt
sh /mnt/VBoxLinuxAdditions.run
umount /mnt

rm VBoxGuestAdditions_$VBOX_VERSION.iso

# Remove items used for building, since they aren't needed anymore
apt-get -y remove linux-headers-$(uname -r) build-essential
apt-get -y autoremove

# Zero out the free space to save space in the final image:
dd if=/dev/zero of=/EMPTY bs=1M
rm -f /EMPTY

# Removing leftover leases and persistent rules
echo "cleaning up dhcp leases"
rm /var/lib/dhcp3/*

# Make sure Udev doesn't block our network
# http://6.ptmc.org/?p=164
echo "cleaning up udev rules"
rm /etc/udev/rules.d/70-persistent-net.rules
mkdir /etc/udev/rules.d/70-persistent-net.rules
rm -rf /dev/.udev/
rm /lib/udev/rules.d/75-persistent-net-generator.rules
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
echo "Adding a 2 sec delay to the interface up, to make the dhclient happy"
echo "pre-up sleep 2" >> /etc/network/interfaces
exit
```

Build the base box. It will prompt you to download the ISO for the installation. If you already have an ISO for the target, put it in `iso/` in the current directory.

During this process, VirtualBox will open a console session. Veewee will interactively enter all the configuration for you, so best to leave it alone and let it work.

```
vagrant basebox build 'ubuntu10.04-gems'
```

Next, export the VM to a .box file.

```
vagrant basebox export 'ubuntu10.04-gems'
```

Import the box into Vagrant.

```
vagrant box add 'ubuntu10.04-gems' 'ubuntu10.04-gems.box'
```

Now you're ready to use it in Vagrant!

```
vagrant init 'ubuntu10.04-gems'
```

Modify the Vagrantfile as desired, then launch the box with vagrant.

```
vagrant up
vagrant ssh
```



Ubuntu 10.10 NFS Bug
If you run into errors using Vagrant mount the /etc/chef folder, and if you have NFS installed on your Vagrant base box, you'll need to do one of two things to overcome a known ubuntu/nfs bug:

1. remove it and vagrant will mount another less performant way
2. Tell Vagrant not to use nfs mounting, e.g.

```
csc.vm.share_folder("chef-server-etc", "/etc/chef",
"#{@cs_kitchen}/mnt/etc", :nfs => false)
```



Installation

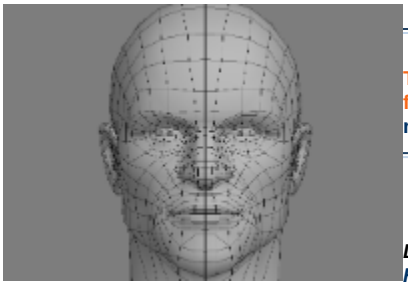
Managing Chef



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Installing Chef from HEAD



This document is focused on users who wish to get the latest development version of Chef installed for testing and development, or for the latest and greatest features that haven't been officially released yet.

Developers who wish to contribute to the project are encouraged to read the README.rdoc in the Git Hub repository.

Each released version of Chef is tagged in the Git Repository. You can work with a previous version's code by checking out the tag.

Version Numbering Style
We follow Linux kernel-style versioning. If the patch-level version is odd, it's a 'dev' release, as installing from HEAD is certainly 'dev'.

Requirements

First, you'll want to designate a system for development and deployment. This is the place where you'll clone the repository and create the gems. It can be your local workstation, or it can be the Chef Server itself.

- Git - the source is on GitHub, so you'll need to have Git installed for your platform.
- RubyGems - we have Rake tasks for generating and installing the gems, and these instructions should work anywhere with RubyGems. We recommend installing RubyGems from source.
- Rake - we like Rake, and use it liberally, install the `rake` gem.
- RSpec - we do specs in rspec, install the `rspec` gem v 1.3.1. v2.0.0 does not work with our current testsuite.
- Cucumber - we do feature tests with cucumber, install the `cucumber` gem.
- Gemcutter and Jeweler - we use jeweler for gemspec generation and may want to push to gemcutter, install these gems.

When installing the gems that are built from these instructions and using the Installing Chef Server using Chef Solo on Chef Server or Chef Client systems, other dependencies are handled as well such as CouchDB, RabbitMQ and Java.

You can also install the these for a Chef Server manually.

Install RubyGem Development Requirements

```
gem install rspec rake cucumber jeweler gemcutter
```

Some other gems are required to build the gems, but are automatically installed when we use `rake install` below. If you want to have them available first:

Install Additional Gem Building Requirements

```
gem install merb-core merb-slices merb-assets merb-helpers merb-haml moneta bunny uuidtools
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Bring behaviour-driven development to infrastructure as code

Community member Stephen Nelson-Smith has written **Test-Driven Infrastructure with Chef**.

He demonstrates a radical approach to developing web infrastructure that combines the powerful Chef configuration management framework with Cucumber, the leading Behavior-driven development (BDD) tool.

Learn how developing code test-first allows you to make significant changes without the fear of unexpected side effects.



Ubuntu Git Package
Ubuntu's git package to actually get usable git binaries installed is 'git-core', not 'git'.

Don't need RDoc and Ri?
By default, Gems install with the RDoc and Ri documentation available. You can disable this for gems you install by adding "`gem: --no-rdoc --no-ri`" to ~/.gemrc. This will save time on installation, too.

Git Repositories

Clone the `ohai` and `chef` repositories fresh from github. If you want the latest and greatest from Chef, you probably want the same from Ohai. You'll also need the Opscode mixlibs which are used in Chef and Ohai.

Clone Git Repositories

```
cd ~
git clone git://github.com/opscode/ohai.git
git clone git://github.com/opscode/chef.git
git clone git://github.com/opscode/mixlib-authentication.git
git clone git://github.com/opscode/mixlib-config.git
git clone git://github.com/opscode/mixlib-log.git
git clone git://github.com/opscode/mixlib-cli.git
```

Installing the Gems

We're going to install the gems from the repositories checked out above. Each of these has gem dependencies that build on each other. This is the order which you should install the gems.

In each rake install, the respective gem was generated. We recommend setting up an internal RubyGems server in your environment to host the development gems, so you don't have to copy them manually to client systems for testing. Opscode provides a gems::server recipe that will help you set up a RubyGems server.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Install mixlibs

Ohai and Chef use some "mixin" libraries that Opscode wrote. We need to install these first or the Ohai and Chef gems will fail to build.

Install mixlib components

```
#!/bin/sh
for mixlib in config cli log authentication
do
  pushd mixlib-$mixlib
  rake install
  popd
done
```

Install Ohai

Ohai needs to be installed as well so the Chef gems can be built.

Install Ohai

```
cd ~/ohai
rake install
```

Install Client

For a chef-client, just install the `chef` gem, it provides the binaries and code needed to run Chef.

Install Chef

```
cd ~/chef/chef
rake install
```

Install gecode

On Ubuntu lucid, maverick, or Debian lenny release, you can install gecode deb package from the Opscode APT repository.

Add the Opscode APT Repository

Create `/etc/apt/sources.list.d/opscode.list`.

/etc/apt/sources.list.d/opscode.list

- For Chef 0.9.x, replace codename with the supported distribution codename, such as "lucid".
- For Chef 0.10.x, replace codename with the codename, suffixed with "-0.10", for example, "lucid-0.10".

If you would like to be able to download source packages, add an additional identical line, but change `deb` to `deb-src` line.

You can copy and paste these examples to create the necessary sources.list entry:

Ubuntu for Chef 0.9.x

Ubuntu for Chef 0.10.x

Debian users will likely need to run 'apt-get install sudo wget lsb-release' as root before pasting the examples.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Add the GPG Key and Update Index

Before you install the packages, make sure you add the Opscode GPG key to apt.

Issues downloading from gnupg.net?
If you get an error when trying to download the key that states that the "keyserver timed out", try downloading the key directly from our apt repository instead:

Afterwards you'll also want to run this command again:

Now, we update apt-get with the newly added Opscode repository:

Installing the opscode-keyring package will keep the key up-to-date:

Upgrade Existing Packages

To ensure you are using the latest versions of libraries that chef depends on, you may wish to update your existing packages:

Install deb package

```
sudo apt-get install gecode
```

Build and Install from Source

```
cd /tmp
curl -C - -O  http://www.gecode.org/download/gecode-3.5.0.tar.gz
tar zxvf gecode-3.5.0.tar.gz
cd gecode-3.5.0 && ./configure
make && make install
```

Install Server

If you are not using a RubyGems server, you'll want to install the Chef Server gems.

Gemcutter jeweler are also prerequisites to do a rake install for the server. So before you do the rake install for the server:

Install Gemcutter Jeweler

```
sudo gem install gemcutter jeweler
```

Then you can install the server:

Install Chef Server

```
cd ~/chef
rake install
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Configure Chef

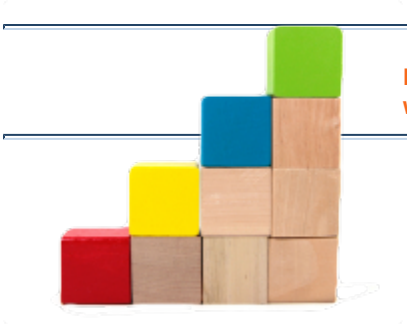
The Installing Chef Server Manually procedures can be used to get the server configured. If you're planning to run systems on the development version, we strongly recommend a RubyGems server with Chef and all the dependencies available to ease this process. Or, if you're using Amazon EC2, build an AMI with the gems installed.

 Installation

Managing Chef 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Configure Management Workstation As A Chef-Client



IF you want to configure your Management Workstation as a Chef-Client, this page will explain how to do so.

Node vs. Client vs. User vs. Workstation

First, to help avoid any confusion, lets go over terminology. The system where you are doing development and maintaining a cookbook repository we can call a workstation or your management workstation. It is where you run knife commands. A server you want to manage with chef we call a node, and the only command you will run on it should be "sudo chef-client".

A client allows you to access the server API. In general, clients map 1:1 to the list of nodes (every node is a client, and every client represents 1 node.) However, it is possible for a single client to manage multiple nodes - this is most often done for programmatic access to the nodes via the API. A user is an individual: credentialed entities used by humans to connect to the Hosted Platform to manage the organization.

The user is the credentialed entity that manages the organization:

- using their management workstation
- connecting through the Web UI (Management Console),
- using the knife command line tool

The organization is comprised of nodes

- systems that are configured and managed by chef

Which run the chef-client software

- which has a certificate to authorize its interaction with the server/platform to download cookbooks it needs to run and compile the configuration of the node.

Must I Have My Workstation Be A Chef-Client?

NO You are *not* required to configure your workstation as a chef-client for interacting with Chef to automate your infrastructure.

Managing your workstation via Chef is a choice, not a requirement. It would be driven by a desire to expand configuration management to the desktop, not by a need for Chef functionality.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Terminology Documentation References

A Can of Condensed Chef Documentation
Includes links to pages with more detail on each topic (if desired), and a call-out link to a **glossary**

Chef Basics
Includes a call-out link to a nice **Chef Speak** overview blog post by team member Dan DeLeo.

Having Chef Manage the Configuration of your Management Workstation

To have the Chef Server (or Hosted Chef, as the case may be) manage the configuration of your workstation, you need to configure the client. These directions assume you have already installed knife by following the Installation Guides.

```
$ cd ~/chef-repo
$ knife configure client ./client-config
INFO: Creating client configuration
INFO: Writing client.rb
INFO: Writing validation.pem
```

You'll now have a client-config directory in your local repository:

```
$ ls -l client-config
-rw-r--r--   1 adam  staff   155 Jun 18 16:03 client.rb
-rw-r--r--@  1 adam  staff  1679 Jun 18 16:03 validation.pem
```

To configure your workstation as a chef client, you just need to copy this directory to /etc/chef:

```
$ sudo mkdir /etc/chef
$ sudo cp -r ~/chef-repo/client-config/* /etc/chef
```



You have now configured your management workstation as a chef client, and can manage its configuration through Chef!
Yea you!

Installation

Managing Chef

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Quick Start



The former Quick Start is now the Fast Start Guide

Please go to there to get started, quickly.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Running Chef Client with Elevated Privileges



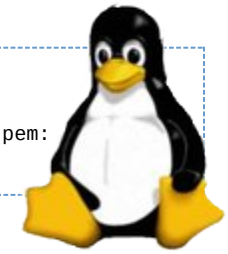
Chef-client needs to run with elevated privileges to get most recipes to converge correctly.

On Unix and Unix-like operating systems this is done by running the command as root.

On Windows this is done by running command prompt as administrator.

Linux

On Linux, you may see this error if you do not have the correct permissions to run chef-client:



```
$ chef-client
[Tue, 29 Nov 2011 19:46:17 -0800] INFO: *** Chef 0.10.x ***
[Tue, 29 Nov 2011 19:46:18 -0800] WARN: Failed to read the private key /etc/chef/client.pem:
#<Errno::EACCES: Permission denied - /etc/chef/client.pem>
```

You can resolve this by running the command as root. There are a few ways you can accomplish this:

- Logging in as root and running chef-client
- Using su to become the root user and running chef-client:

```
su
chef-client
```

- Using the `sudo` utility

```
sudo chef-client
```

You could also resolve this by giving your user access to read /etc/chef and the files accessed by chef-client, but *this is not advised as most recipes will need super user privileges to complete correctly.*

Windows



In Windows, this issue normally fails silently. **It appears that chef-client converged correctly, but when you go to check the changes they have not been made.**

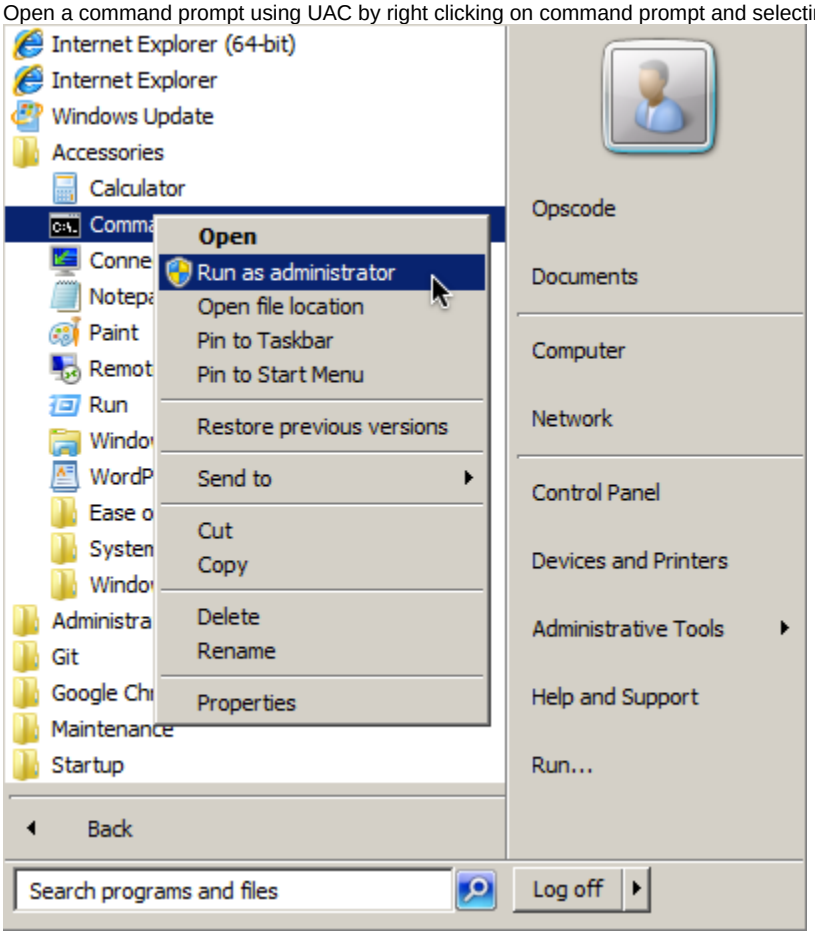
If you see this occur, follow one of these options to run chef-client as administrator and see if this resolves the issue.

- Login into the Administrator account. **NOTE:** *This is different than an account in the Administrator's group.*
- Run the chef-client process from the Administrator account while being logged into another account (you will be prompted for your Administrator password). You can do this by entering this command into a command prompt window:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
runas /user:Administrator "cmd /C chef-client -c c:/chef/client.rb"
```

- Open a command prompt using UAC by right clicking on command prompt and selecting 'run as administrator':



Once the command prompt is open, you can run chef-client. Keep in mind that you will need to provide the full path to the client.rb as the current working directory will now be the Administrator's home directory. You can do this with a command such as `chef-client -c C:\chef\client.rb`.

Installation

Managing Chef

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Chef



Knife

Knife is a command-line tool that comes with Chef. It is used by administrators to interact with the Chef Server API, the local Chef repository and more.

Community Plugins

Plugins created by community members that you may find useful in the management of your infrastructure with Chef.

Management Console

The Management Console is Chef Server's web interface. Nodes, roles, cookbooks, data bags, and API clients can be managed through the Management Console. Search can also be done on the console.

Shef

Shef is a way to run chef in an irb session. Shef currently supports recipe and attribute file syntax, as well as interactive debugging features.

Spiceweasel

Spiceweasel is a command-line tool that uses JSON or YAML to work with knife for batch loading Chef infrastructure.

Installation

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Knife



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Knife



What is Knife?

Knife is a powerful command-line interface (CLI) that comes with Chef.

It is used by administrators to interact with the Chef Server API and the local Chef repository. It provides the capability to manipulate nodes, cookbooks, roles, databags, environments, etc., and can also be used to provision cloud resources and to bootstrap systems.

Overview

As the administrators command line tool for interacting for the chef server, Knife is run from the management workstation, acting as the intersection between the Chef Server and your infrastructure.

Knife commands all have the following form

```
knife sub-command [ARGUMENTS] (options)
```

Knife interacts with the Chef Server through the same REST API that the Chef Client software uses, authenticating as an API client. Hosted Chef and Private Chef then have the additional role based authentication controls (RBAC) that authorize changes based upon users and organizations. See Authentication and Authorization for more information on this model.

Knife comes with a series of built in subcommands, which provide the functionality and capability for specific actions upon cookbooks, nodes, roles, etc. within your infrastructure. Knife also has the capability to be a container for Knife Plugins - which extend its functionality beyond the built in commands to features including launching cloud instances and bootstrapping systems.

There is a built in contextual help system for obtaining more information on the commands and subcommands, options for their use, or reviewing manpage documentation. When you go through Workstation Setup, one of the actions includes establishing and configuring knife for use. Subsequent modifications can be made through editing the knife.rb configuration file.

Knife Built In Subcommands

The following subcommands are built into Knife, and available for managing their respective components from the Knife command line interface.
See **Knife Built In Subcommands** for details on use of each.

return to top of page

Subcommands built into Knife:

- Bootstrap
- Client
- Cloud Plugins
- Configure

Knife sub-commands are structured to take the form
`NOUN verb NOUN` (options)

Because the Chef Server API is RESTful, the `verb` of a knife command is generally a CRUD operation:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- Cookbook
- Cookbook Site
- Data Bag
- Environment
- Exec
- Node
- Recipe
- Role
- Search
- SSH
- Status
- Tag
- Bash command completion

- **create** (create)
- **list** and show (read)
- **edit** (update)
- **delete** (destroy)

Contextual Help

Knife has a smart contextual help system built right in.

return to top of page

If you invoke knife with the `--help` option at the top level you will see options common to all subcommands.

```
% knife --help
Usage: knife sub-command (options)
    -s, --server-url URL             Chef Server URL
    -k, --key KEY                    API Client Key
        --color                      Use colored output
    -c, --config CONFIG              The configuration file to use
        --defaults                   Accept default values for all questions
    -e, --editor EDITOR              Set the editor to use for interactive commands
    -E, --environment ENVIRONMENT    Set the Chef environment
    -F, --format FORMAT              Which format to use for output
        --no-color                   Don't use colors in the output
    -n, --no-editor                  Do not open EDITOR, just accept the data as is
    -u, --user USER                  API Client Username
        --print-after                Show the data after a destructive operation
    -V, --verbose                    More verbose output. Use twice for max verbosity
    -v, --version                    Show chef version
    -y, --yes                        Say yes to all prompts for confirmation
    -h, --help                       Show this message

Available subcommands: (for details, knife SUB-COMMAND --help)

...
```

If you invoke knife with the `--help` option at a specific subcommand level, knife presents the options specific to that subcommand.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
% knife ssh --help
knife ssh QUERY COMMAND (options)
    -a, --attribute ATTR             The attribute to use for opening the connection - default is fqdn
    -s, --server-url URL             Chef Server URL
    -k, --key KEY                    API Client Key
        --color                      Use colored output
    -C, --concurrency NUM            The number of concurrent connections
    -c, --config CONFIG              The configuration file to use
        --defaults                   Accept default values for all questions
    -e, --editor EDITOR              Set the editor to use for interactive commands
    -E, --environment ENVIRONMENT    Set the Chef environment
    -F, --format FORMAT              Which format to use for output
    -i IDENTITY_FILE,                The SSH identity file used for authentication
        --identity-file
    -m, --manual-list                QUERY is a space separated list of servers
        --no-color                   Don't use colors in the output
    -n, --no-editor                  Do not open EDITOR, just accept the data as is
        --no-host-key-verify         Disable host key verification
    -u, --user USER                  API Client Username
        --print-after                Show the data after a destructive operation
    -P, --ssh-password PASSWORD      The ssh password
    -p, --ssh-port PORT              The ssh port
    -x, --ssh-user USERNAME          The ssh username
    -V, --verbose                    More verbose output. Use twice for max verbosity
    -v, --version                    Show chef version
    -y, --yes                        Say yes to all prompts for confirmation
    -h, --help                       Show this message
```

Further, you can directly access manpages with content similar to this page using the `help` sub-command. Use `knife help list`.

```
> knife help list
Available help topics are:
  bootstrap
  client
  configure
  cookbook
  cookbook-site
  data-bag
  environment
  exec
  index
  knife
  node
  role
  search
  shef
  ssh
  status
  tag
```

You can then read individual pages by using a command in the form of:

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife help topic
```

Knife has a Man Page
The man page for knife is the authoritative reference for knife. It is also shipped with the chef gem, rpm and deb. It can be easily accessed using knife's built-in help system:

```
knife help knife
```

Configuring Your System For Knife

Knife configuration

The knife configuration file is a Ruby DSL to set configuration parameters for Knife's common options.

return to top of page

Knife's configuration file is named `knife.rb`. *Unless directly passed the configuration file location via the `-c` command line option, Knife searches for this configuration by searching for `.chef/knife.rb`, starting at the current working directory and moving upward.* If no configuration is found `~/.chef/knife.rb` is used if it exists.

The following parameters can be set in `knife.rb`:

- `node_name`:
  User or client identity (i.e., name) to use for authenticating requests to the Chef Server.
- `client_key`:
  Private key file to authenticate to the Chef server. Corresponds to the -k or --key option.
- `chef_server_url`:
  URL of the Chef server. Corresponds to the -s or --server-url option. This is requested from the user when running this sub-command.
- `cache_type`:
  The type of cache to use. Default is BasicFile. This can be any type of Cache that moneta supports: BasicFile, Berkeley, Couch, DataMapper, File, LMC, Memcache, Memory, MongoDB, Redis, Rufus, S3, SDBM, Tyrant, Xattr, YAML.
- `cache_option`:
  Specifies various options to use for caching. These options are dependent on the cache_type.
- `validation_client_name`:
  Specifies the name of the client used to validate new clients.
- `validation_key`:
  Specifies the private key file to use when bootstrapping new hosts. See knife-client(1) for more information about the validation client.
- `cookbook_copyright`, `cookbook_email`, `cookbook_license`:
  Used by knife cookbook create sub-command to specify the copyright holder, maintainer email and license (respectively) for new cookbooks. The copyright holder is listed as the maintainer in the cookbook's metadata and as the Copyright in the comments of the default recipe. The maintainer email is used in the cookbook metadata. The license determines what preamble to put in the comment of the default recipe, and is listed as the license in the cookbook metadata. Currently supported licenses are "apachev2" and "none". Any other values will result in an empty license in the metadata (needs to be filled in by the author), and no comment preamble in the default recipe.

Knife Plugins

As Chef will load commands from a set of specific locations, you can create plugins for reuse across projects in your home directory, share plugins with your team by including them in your cookbook repo, and share plugins with the whole Chef community by distributing them as Ruby gems.

return to top of page

With Chef 0.10, knife will load commands from the following locations:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- The core set of knife commands shipped with Chef
- Commands in your home chef directory: ~/.chef/plugins/knife/
- Commands in a .chef/plugins/knife/ directory in your cookbook repo
- Commands located in a chef/knife/ directory in a Ruby Gem you have installed.

See **Knife Plugins** for details on creating and installing plugins, and **Community Plugins** for a listing of plugins that have been made available by Chef community members.

EDITOR Environment Variable

Many knife commands use the EDITOR environment variable when create or editing objects that will be saved on the Chef Server.

return to top of page

You can set this variable when issuing the command.

```
> EDITOR=nano knife node edit NODENAME
```

Alternatively you can set the variable for the rest of your shell's session.

```
> export EDITOR=nano
> knife node edit NODENAME
```

See your shell's documentation for information on how to ensure that EDITOR is always set when you start a new shell.

Command and Output Options

Common Options

The following options can be passed to any subcommand. For sub-command specific options, see the section of Knife Built In Subcommands for the relevant subcommand.

- `-s`, `--server-url URL`:
  Chef Server URL
- `-k`, `--key KEY`:
  API Client Key
- `--color`:
  Use colored output
- `-c`, `--config CONFIG`:
  The configuration file to use
- `--defaults`:
  Accept default values for all questions
- `-e`, `--editor EDITOR`:
  Set the editor to use for interactive commands
- `-E`, `--environment ENVIRONMENT`:
  Set the Chef environment
- `-F`, `--format FORMAT`:
  Which format to use for output. See the Output Formats section below.
- `--no-color`:
  Don't use colors in the output
- `-n`, `--no-editor`:
  Do not open EDITOR, just accept the data as is
- `-u`, `--user USER`:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

API Client Username
- `--print-after`:
  Show the data after a destructive operation
- `-V`, `--verbose`:
  More verbose output. Use twice for max verbosity
- `-v`, `--version`:
  Show chef version
- `-y`, `--yes`:
  Say yes to all prompts for confirmation
- `-h`, `--help`:
  Show this message

Output Formats

The amount of content displayed and the output format can be modified by the `--format` option. If no alternate format is selected, the default is summary.



Knife Output Change

The default knife output changed with version 0.10. Previous version of knife output JSON while 0.10 formats its output for human consumption.

Knife will still output JSON if passed the option ***`-Fj`*** *at the command line.*

Valid formats are:

- `summary`:
  displays the node in a custom, summarized format (default)
- `text`:
  displays the node data in its entirety using the colorized tree display
- `json`:
  displays the node in JSON format
- `yaml`:
  displays the node in YAML format
- `pp`:
  displays the node using Ruby's pretty printer.

For brevity, only the first character of the format is required, for example, -Fj will produce JSON format output.


Managing Chef

Knife Built In Subcommands


 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Knife Built In Subcommands



Subcommands for managing components from the Knife command line interface

This page contains documentation for each of the built-in knife sub-commands as well as popular plugins.

Many sub-commands are documented on their own pages, though all are listed here for completeness.

Common Options provides detail on the options can be passed to any subcommand, while sub-command specific options are reviewed in the relevant section of this page. Output Formats information on the amount of content displayed and modification options for the output.

Knife commands all have the same form

```
knife sub-command [ARGUMENTS] (options)
```

For all commands and subcommands, there is Contextual Help for the options available, including direct access to manpages.

Knife sub-commands are structured such that commands have the form: `NOUN verb NOUN` (options).

Because the Chef Server API is RESTful, the `verb` of a knife command is generally a CRUD operation:

- **create** (create)
- **list** and show (read)
- **edit** (update)
- **delete** (destroy)

Built In Subcommands

Bootstrap

The **Knife Bootstrap** page explains the bootstrap subcommand. For Chef versions 0.10 and greater, the **Knife Windows Bootstrap** page describes the knife-windows plugin, which allows one to bootstrap Windows nodes.

return to top of page

Client

The **Managing API Clients With Knife** page explains the client subcommand used to manage API clients.

return to top of page

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cloud Plugins

Using the cloud provider subcommands is explained on the **Launch Cloud Instances with Knife** page.

return to top of page

Configure

The configure sub-commands provides an easy way to create `knife.rb` and `client.rb` files that can then be distributed to workstations and nodes.

return to top of page

There are two configure commands:

configure

Create a configuration file for knife. This will prompt for values to enter into the file. Default values are listed in square brackets if no other entry is typed.

Usage:

```
knife configure (options)
```

Additional Options:

- `-i`, `--initial`:
  Create an initial API Client
- `-r`, `--repository REPO`:
  The path to your chef-repo

Use `-i` if you would like to create an admin client (**Hosted Chef** users should note that creating a client as an admin does not change its permissions.)

configure client

Read the `knife.rb` config file and generate a config file suitable for use in `/etc/chef/client.rb` and copy the validation certificate into the specified directory.

Usage:

```
knife configure client DIRECTORY
```

Cookbook

Managing Cookbooks With Knife provides detail on the use of this command to interact with cookbooks on your Chef Server.

return to top of page

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook Site

Managing Cookbooks With Knife provides detail on the use of this command to interact with cookbooks on the Opscode Community Site.

return to top of page

Data Bag

Data bags are stores of arbitrary JSON data that and indexed and available via search. **Managing Data Bags With Knife** provides detail on the use of this command for managing data bags and data bag items.

return to top of page

Environment

The environment sub-command allows you to manage environments. See **Managing Environments With Knife** for use of the environments subcommand.

return to top of page

Exec

`knife exec` allows you to run Ruby scripts in a context similar to that of Shef. See **Knife Exec** for documentation on this command.

return to top of page

Node

See **Managing Nodes With Knife** on use of the node command to modify the Nodes supported by the Chef Server.

return to top of page

Recipe

Recipes on the Chef Server can be listed using the recipe list sub-command.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

return to top of page

Specify **PATTERN** as a regular expression to limit the results.

```
% knife recipe list 'couchdb::*'
  couchdb::main_monitors
  couchdb::master
  couchdb::default
  couchdb::org_cleanu
```

Role

The role sub-commands allow you to manage roles saved on the Chef server and are structured similarly to the other knife sub-commands. See **Managing Roles With Knife** for details on its use.

return to top of page

Search

Search is one of the most useful features of Chef Server.

return to top of page

Search query syntax is covered in greater depth in **Search**. The following is the basic structure of the `knife search` command as well as some useful examples.

Usage:

```
knife search INDEX QUERY (options)
```

Additional Options:

- `-a`, `--attribute ATTR`:
  Show only one attribute
- `-i`, `--id-only`:
  Show only the ID of matching objects
- `-q`, `--query QUERY`:
  The search query; useful to protect queries starting with -
- `-R`, `--rows INT`:
  The number of rows to return
- `-r`, `--run-list`:
  Show only the run list
- `-o`, `--sort SORT`:
  The order to sort the results in
- `-b`, `--start ROW`:
  The row to start returning results at
- `-m`, `--medium`:
  Display medium sized output when searching nodes using the default summary format
- `-l`, `--long`:
  Display long output when searching nodes using the default summary format

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Examples:

```
Search for the ids of all nodes running in EC2:
{code:bash}
> knife search node 'ec2:*' -i
4 items found

ip-0A7CA19F.ec2.internal

ip-0A58CF8E.ec2.internal

ip-0A58E134.ec2.internal

ip-0A7CFFD5.ec2.internal
```

Search for the EC2 instance type (flavor) of all nodes running in EC2

```
> knife search node 'ec2:*' -a ec2.instance_type
4 items found

ec2.instance_type:  m1.large
id:                 ip-0A7CA19F.ec2.internal

ec2.instance_type:  m1.large
id:                 ip-0A58CF8E.ec2.internal

ec2.instance_type:  m1.large
id:                 ip-0A58E134.ec2.internal

ec2.instance_type:  m1.large
id:                 ip-0A7CFFD5.ec2.internal
```

Search for all nodes running Ubuntu:

```
> knife search node 'platform:ubuntu'
```

Search for all nodes running CentOS in the production environment.

```
> knife search node 'chef_environment:production AND platform:centos'
```

SSH

The SSH subcommand allows you to invoke commands in parallel on a subset of the nodes in your infrastructure.

return to top of page

Usage

```
knife ssh QUERY COMMAND (options)
```

The QUERY part of the command uses the same syntax as the search subcommand. This will return a list of servers that it will ssh into and run the COMMAND you've listed.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Additional Options:

- `-a`, `--attribute ATTR`:
  The attribute to use for opening the connection - default is fqdn
- `-C`, `--concurrency NUM`:
  The number of concurrent connections
- `-m`, `--manual-list`:
  QUERY is a space separated list of servers
- `-P`, `--ssh-password PASSWORD`:
  The ssh password
- `-x`, `--ssh-user USERNAME`:
  The ssh username
- `-i`, `--identity-file IDENTITY_FILE`:
  The SSH identity file used for authentication
- `-p`, `--ssh-port PORT`:
  The ssh port
- `--no-host-key-verify`:
  Disable host key verification

Examples:

Here's a simple example of this command:

```
knife ssh "role:webserver" "sudo chef-client"
```

This would ssh into all nodes that have the role "webserver" and run the command "sudo chef-client".

You could also use this to upgrade all of your nodes:

```
knife ssh name:* "sudo aptitude upgrade -y"
```

The SSH Subcommand could also be used to find the uptime of all your web servers:

```
% knife ssh "role:web" "uptime" -x ubuntu -a ec2.public_hostname
ec2-174-129-127-206.compute-1.amazonaws.com  13:50:47 up 1 day, 23:26,  1 user,  load average: 0.25,
0.18, 0.11
ec2-67-202-63-102.compute-1.amazonaws.com    13:50:47 up 1 day, 23:33,  1 user,  load average: 0.12,
0.13, 0.10
ec2-184-73-9-250.compute-1.amazonaws.com     13:50:48 up 16:45,  1 user,  load average: 0.30, 0.22,
0.13
ec2-75-101-240-230.compute-1.amazonaws.com   13:50:48 up 1 day, 22:59,  1 user,  load average: 0.24,
0.17, 0.11
ec2-184-73-60-141.compute-1.amazonaws.com    13:50:48 up 1 day, 23:30,  1 user,  load average: 0.32,
0.17, 0.15
```

Force a Chef run on each node returned by the search query:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
% knife ssh "role:web" "sudo chef-client" -x ubuntu -a ec2.public_hostname
ec2-67-202-63-102.compute-1.amazonaws.com   [Fri, 22 Oct 2010 14:18:37 +0000] INFO: Starting Chef Run
(Version 0.9.10)
ec2-174-129-127-206.compute-1.amazonaws.com [Fri, 22 Oct 2010 14:18:37 +0000] INFO: Starting Chef Run
(Version 0.9.10)
ec2-184-73-9-250.compute-1.amazonaws.com    [Fri, 22 Oct 2010 14:18:38 +0000] INFO: Starting Chef Run
(Version 0.9.10)
ec2-75-101-240-230.compute-1.amazonaws.com  [Fri, 22 Oct 2010 14:18:38 +0000] INFO: Starting Chef Run
(Version 0.9.10)
ec2-184-73-60-141.compute-1.amazonaws.com   [Fri, 22 Oct 2010 14:18:38 +0000] INFO: Starting Chef Run
(Version 0.9.10)
ec2-174-129-127-206.compute-1.amazonaws.com [Fri, 22 Oct 2010 14:18:39 +0000] INFO: Chef Run complete
in 1.419243 seconds
ec2-174-129-127-206.compute-1.amazonaws.com [Fri, 22 Oct 2010 14:18:39 +0000] INFO: cleaning the
checksum cache
ec2-174-129-127-206.compute-1.amazonaws.com [Fri, 22 Oct 2010 14:18:39 +0000] INFO: Running report
handlers
ec2-174-129-127-206.compute-1.amazonaws.com [Fri, 22 Oct 2010 14:18:39 +0000] INFO: Report handlers
complete
ec2-67-202-63-102.compute-1.amazonaws.com   [Fri, 22 Oct 2010 14:18:39 +0000] INFO: Chef Run complete
in 1.578265 seconds
ec2-67-202-63-102.compute-1.amazonaws.com   [Fri, 22 Oct 2010 14:18:39 +0000] INFO: cleaning the
checksum cache
ec2-67-202-63-102.compute-1.amazonaws.com   [Fri, 22 Oct 2010 14:18:39 +0000] INFO: Running report
handlers
ec2-67-202-63-102.compute-1.amazonaws.com   [Fri, 22 Oct 2010 14:18:39 +0000] INFO: Report handlers
complete
ec2-184-73-9-250.compute-1.amazonaws.com    [Fri, 22 Oct 2010 14:18:40 +0000] INFO: Chef Run complete
in 1.638884 seconds
ec2-184-73-9-250.compute-1.amazonaws.com    [Fri, 22 Oct 2010 14:18:40 +0000] INFO: cleaning the
checksum cache
ec2-184-73-9-250.compute-1.amazonaws.com    [Fri, 22 Oct 2010 14:18:40 +0000] INFO: Running report
handlers
ec2-184-73-9-250.compute-1.amazonaws.com    [Fri, 22 Oct 2010 14:18:40 +0000] INFO: Report handlers
complete
ec2-75-101-240-230.compute-1.amazonaws.com  [Fri, 22 Oct 2010 14:18:40 +0000] INFO: Chef Run complete
in 1.540257 seconds
ec2-75-101-240-230.compute-1.amazonaws.com  [Fri, 22 Oct 2010 14:18:40 +0000] INFO: cleaning the
checksum cache
ec2-75-101-240-230.compute-1.amazonaws.com  [Fri, 22 Oct 2010 14:18:40 +0000] INFO: Running report
handlers
ec2-75-101-240-230.compute-1.amazonaws.com  [Fri, 22 Oct 2010 14:18:40 +0000] INFO: Report handlers
complete
ec2-184-73-60-141.compute-1.amazonaws.com   [Fri, 22 Oct 2010 14:18:40 +0000] INFO: Chef Run complete
in 1.502489 seconds
ec2-184-73-60-141.compute-1.amazonaws.com   [Fri, 22 Oct 2010 14:18:40 +0000] INFO: cleaning the
checksum cache
ec2-184-73-60-141.compute-1.amazonaws.com   [Fri, 22 Oct 2010 14:18:40 +0000] INFO: Running report
handlers
ec2-184-73-60-141.compute-1.amazonaws.com   [Fri, 22 Oct 2010 14:18:40 +0000] INFO: Report handlers
complete
```

Note the `-x` option tells knife to invoke ssh as the "ubuntu" user. The `-a` option tells knife to use the "ec2.public_hostname" attribute for opening the connection - default is fqdn.

Interpolation Rules for the SSH Subcommand

Because knife runs at a command prompt, it is important to be clear about interpolation rules for the SSH subcommand.

For example: assume that the nodes in a db role have a schema that is the same as the node, and that said schema contains a table named "dollars".

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

The command below will be interpolated in the context of the local machine. That is to say the `$(hostname)` will be the name of the machine from which knife was invoked. This is to be expected because the local shell sees double quotes and makes the substitution **before** the command is sent to the node(s). A mysql error from each node should be expected in this case.

```
% knife ssh "role:db" "mysql $(hostname) -u<username> -p<secret> -e 'select count(*) from dollars'"
```

Now consider the next command. It will be interpolated in the context of the node. That is to say the `$(hostname)` will be that of the node on which the command is run for each node in the db role. This is to be expected because the local shell sees single quotes and takes the string as a literal. The command is then interpolated by the each node's shell. A count of the number of rows in the dollars table from each node should be expected in this case.

```
% knife ssh "role:db" 'mysql $(hostname) -u<username> -p<secret> -e "select count(*) from dollars"'
```

The lesson here is to keep the interpolation rules of your local shell in mind when passing a command to your nodes via knife ssh.

Node Environment Customization Considerations

The interpolation rules for the SSH command can cause the behavior of knife ssh to appear erratic when custom configurations are used on the node. To understand how this can happen, consider the following scenario:

A user named "fred" places a .my.cnf file in his home directory on all of his nodes. Below is a template for this.

```
[client]
user        = fred
password    = myex-wifesmaidenname
database    = #{node[:hostname]}
```

The .my.cnf file is read when a mysql client is invoked and substitutes the username, password and database behind the scenes so they do not need to be typed.

If, on his local machine, Fred invokes knife like this:

```
% knife ssh "role:db" "mysql ${hostname} -e 'select count(*) from dollars'"
```

The result in this case will still be a count of rows in the dollars table for each node. In other words, it **appears** that interpolation takes place in the context of the node rather than locally.

However, what actually happened is the command `mysql -e 'select count(*) from dollars'` was sent to the node and the mysql client read the coincidentally correct database name from the .my.cnf file. Note that there are curly braces around the hostname, not parenthesis, so `${hostname}` will evaluate to an empty string. It becomes quickly apparent that setting a different database name in the .my.cnf table from that of the node in this case could lead to even greater confusion when mysql returns saying the table does not exist in that schema. Or, if another user issues the same knife command and has no .my.cnf set, they will get an authentication error from mysql.

The lesson here is to be aware of any personal and global customizations because they can cause apparently erratic behavior that is, in fact, exactly correct.

Ensuring Process Runs Survive SSH Session

When a node script is run to start a background process, the script can complete and exit correctly, however the service that was to be run in the background had not completed prior to script closure. This appears most often with custom start-up scripts in `/etc/init.d`. These apparent inconsistencies are a product of how ssh handles terminal connections remotely, not issues with the knife-ssh subcommand.

Consider a custom server process called spaminatord, that connects to various university websites and gathers student information. If the start up script contains the below line to start spaminatord, it is at risk for exhibiting this behavior.

```
su -l spaminator -c "spaminatord --host=$PRIVATE_IP >> /var/log/my-corp/spaminatord.start.log &"
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

To insure the process runs beyond the life of the ssh session, the following need to be added to the command starting the process.

```
su -l spaminator -c "nohup spaminatord --host=$PRIVATE_IP >> /var/log/my-corp/spaminatord.start.log
2>> spaminatord.start.log < /dev/null &"
```

It is important to note that `nohup` and the redirection rules are all necessary to insure a clean start.

Alternate SSH Modes

The SSH subcommand supports a few alternate modes.

Interactive Mode

start an interactive ssh session. Commands will run agains all nodes (or a subset) returned by search query.

```
% knife ssh "role:web" interactive -x ubuntu -a ec2.public_hostname
Connected to ec2-184-73-9-250.compute-1.amazonaws.com, ec2-75-101-240-230.compute-1.amazonaws.com,
ec2-174-129-127-206.compute-1.amazonaws.com, ec2-67-202-63-102.compute-1.amazonaws.com and
ec2-184-73-60-141.compute-1.amazonaws.com

To run a command on a list of servers, do:
  on SERVER1 SERVER2 SERVER3; COMMAND
  Example: on latte foamy; echo foobar

To exit interactive mode, use 'quit!'

knife-ssh> uptime
ec2-174-129-127-206.compute-1.amazonaws.com  13:59:31 up 1 day, 23:35,  1 user,  load average: 0.06,
0.11, 0.09
ec2-67-202-63-102.compute-1.amazonaws.com    13:59:31 up 1 day, 23:42,  1 user,  load average: 0.08,
0.11, 0.09
ec2-184-73-9-250.compute-1.amazonaws.com     13:59:31 up 16:53,  1 user,  load average: 0.08, 0.13,
0.10
ec2-75-101-240-230.compute-1.amazonaws.com   13:59:31 up 1 day, 23:08,  1 user,  load average: 0.11,
0.15, 0.10
ec2-184-73-60-141.compute-1.amazonaws.com    13:59:31 up 1 day, 23:39,  1 user,  load average: 0.17,
0.16, 0.14
knife-ssh> sudo service apache2 status
ec2-75-101-240-230.compute-1.amazonaws.com  Apache is running (pid 10551).
ec2-184-73-60-141.compute-1.amazonaws.com   Apache is running (pid 10330).
ec2-184-73-9-250.compute-1.amazonaws.com    Apache is running (pid 6013).
ec2-174-129-127-206.compute-1.amazonaws.com Apache is running (pid 10453).
ec2-67-202-63-102.compute-1.amazonaws.com   Apache is running (pid 29696).
```

terminal multiplexer support - launch a tmux (or gnu screen) session with a window for each node returned by the search query:

tmux



Chef 10.6 and tmux
Typically, sessions are given names that match the search term used. Chef 0.10.6+ replaces any occurrences of ':' with '=' when generating the session name, since ':' is an illegal character for tmux session names.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
% knife ssh "role:web" tmux -x ubuntu -a ec2.public_hostname

Warning: Permanently added 'ec2-184-73-9-250.compute-1.amazonaws.com,184.73.9.25
0' (RSA) to the list of known hosts.
Linux domU-12-31-38-00-39-D1 2.6.32-308-ec2 #16-Ubuntu SMP Thu Sep 16 14:28:38 U
TC 2010 i686 GNU/Linux
Ubuntu 10.04.1 LTS

Welcome to Ubuntu!
 * Documentation:  https://help.ubuntu.com/

  System information as of Fri Oct 22 14:02:21 UTC 2010

  System load:  0.08              Processes:           59
  Usage of /:   7.2% of 9.84GB   Users logged in:     0
  Memory usage: 10%               IP address for eth0: 10.252.62.31
  Swap usage:   0%

  Graph this data and manage this system at https://landscape.canonical.com/
----------------------------------------------------------------------
At the moment, only the core of the system is installed. To tune the
system to your needs, you can choose to install one or more
predefined collections of software by running the following
command:

   sudo tasksel --section server
----------------------------------------------------------------------

27 packages can be updated.
12 updates are security updates.

A newer build of the Ubuntu lucid server image is available.
It is named 'release' and has build serial '20101020'.
Last login: Fri Oct 22 14:01:23 2010 from c-76-20-240-131.hsd1.ga.comcast.net
ubuntu@domU-12-31-38-00-39-D1:~$


[knife] 0:zsh- 1:ec2-184-73-9-250.compute-1.amazonaws.com* 2:ec2-75-101-240-230.compute-1.a>
"helsinki.chisamore.com" 10:02 22-Oct-10
```

GNU Screen

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
Warning: Permanently added 'ec2-184-73-60-141.compute-1.amazonaws.com,184.73.60.141' (RSA) to the list
of known hosts.
Linux domU-12-31-38-00-7C-B7 2.6.32-308-ec2 #16-Ubuntu SMP Thu Sep 16 14:28:38 UTC 2010 i686 GNU/Linux
Ubuntu 10.04.1 LTS

Welcome to Ubuntu!
 * Documentation:  https://help.ubuntu.com/

  System information as of Fri Oct 22 14:12:08 UTC 2010

  System load:  0.15              Processes:           61
  Usage of /:   7.2% of 9.84GB   Users logged in:     1
  Memory usage: 10%               IP address for eth0: 10.252.131.69
  Swap usage:   0%

  Graph this data and manage this system at https://landscape.canonical.com/
----------------------------------------------------------------------
At the moment, only the core of the system is installed. To tune the
system to your needs, you can choose to install one or more
predefined collections of software by running the following
command:

   sudo tasksel --section server
----------------------------------------------------------------------

27 packages can be updated.
12 updates are security updates.

Last login: Fri Oct 22 14:06:50 2010 from c-76-20-240-131.hsd1.ga.comcast.net
ubuntu@domU-12-31-38-00-7C-B7:~$



-129-127-206.  3- ec2-67-202-63-102.co  4* ec2-184-73-60-141.compute-1.amazonaws
knife ssh role:web
```

csshX



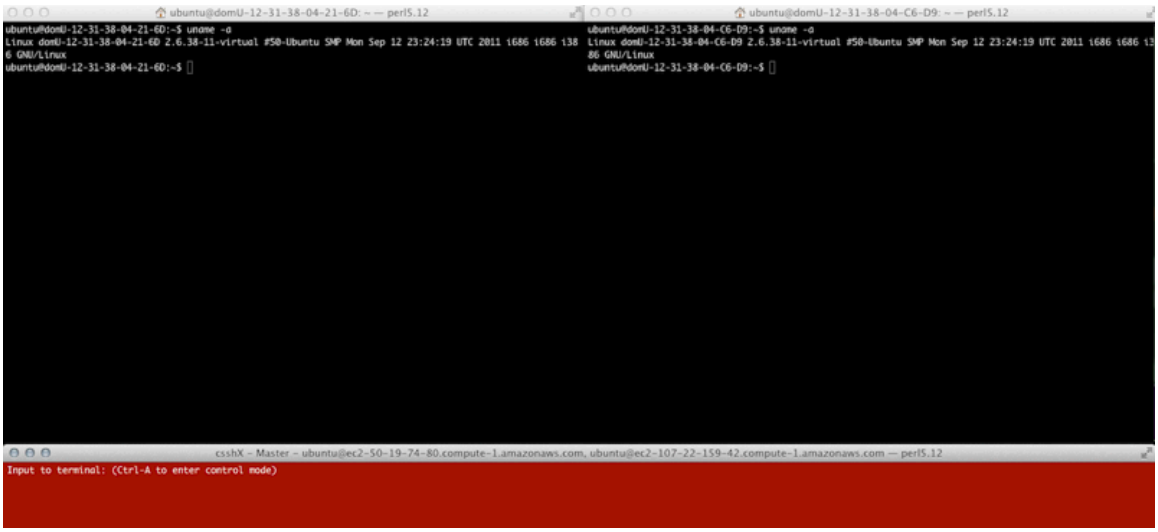
key-based authentication
Using csshX requires that you can authenticate to your nodes using key-based authentication and that your key has been added to your ssh-agent.

You can launch concurrent ssh sessions using csshX:

```
knife ssh 'ec2:*' -a ec2.public_hostname -x ubuntu csshx
```

This will open a new terminal window for each machine as well as a master terminal which can be used to send commands to all of the nodes.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Mac Terminal

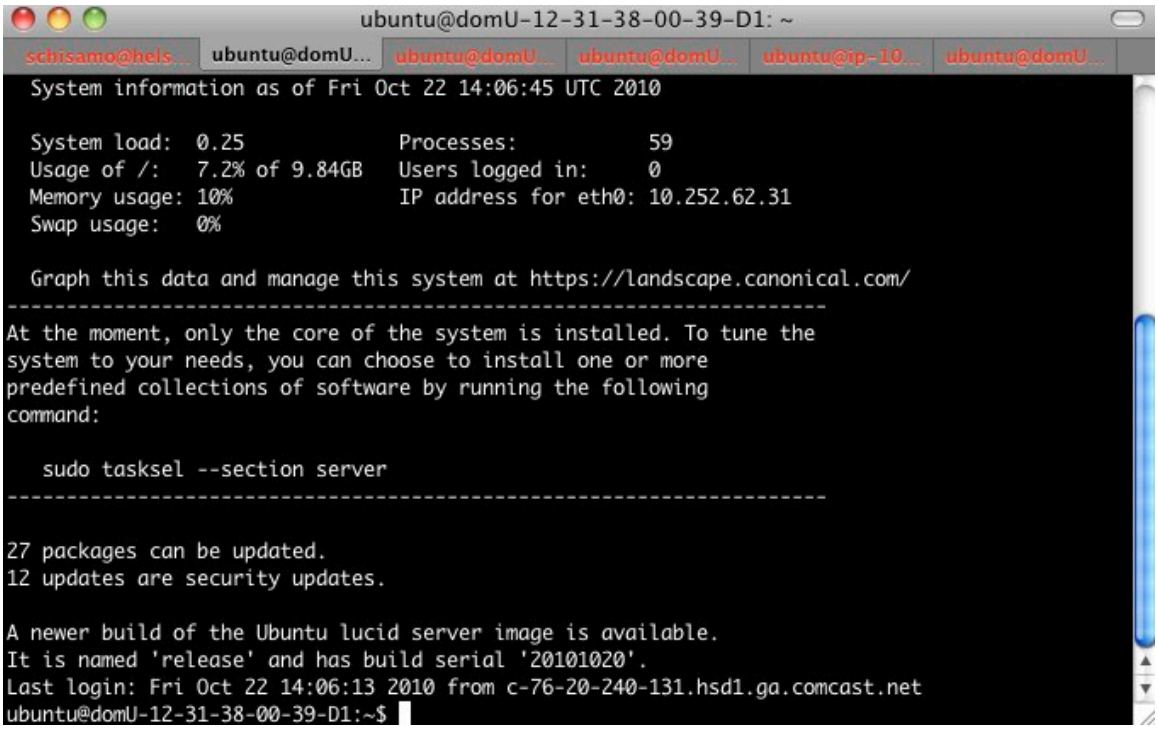
launch a new terminal window with a separate tab (and ssh session) for each node returned by the search query.

This sub-command requires extra gems!
The ssh macterm sub-command interact with mac terminal via applescript. You will need to install the rb-appscript gem on your local management workstation where you use knife.

```
sudo gem install rb-appscript
```

```
%knife ssh "role:web" macterm -x ubuntu -a ec2.public_hostname
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Status

The status}] sub-command searches the Chef Server for all nodes and displays information about the last time the node checked into the server and executed a {{node.save.

return to top of page

Usage:

```
knife status QUERY(options)
```

Specifying a QUERY allows you to restrict the results to a subset of the nodes.

Additional Options:

- `-r`, `--run-list RUN_LIST`:
  Show the run list

Examples:

Show the status of all nodes in your organization

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
% knife status
20 hours ago, dev-vm.chisamore.com, ubuntu 10.04, dev-vm.chisamore.com, 10.66.44.126
3 hours ago, i-225f954f, ubuntu 10.04, ec2-67-202-63-102.compute-1.amazonaws.com, 67.202.63.102
3 hours ago, i-a45298c9, ubuntu 10.04, ec2-174-129-127-206.compute-1.amazonaws.com, 174.129.127.206
3 hours ago, i-5272a43f, ubuntu 10.04, ec2-184-73-9-250.compute-1.amazonaws.com, 184.73.9.250
3 hours ago, i-226ca64f, ubuntu 10.04, ec2-75-101-240-230.compute-1.amazonaws.com, 75.101.240.230
3 hours ago, i-f65c969b, ubuntu 10.04, ec2-184-73-60-141.compute-1.amazonaws.com, 184.73.60.141
```

Include the nodes' run list in the status

```
% knife status --run-list
20 hours ago, dev-vm.chisamore.com, ubuntu 10.04, dev-vm.chisamore.com, 10.66.44.126, role[lb].
3 hours ago, i-225f954f, ubuntu 10.04, ec2-67-202-63-102.compute-1.amazonaws.com, 67.202.63.102,
role[web].
3 hours ago, i-a45298c9, ubuntu 10.04, ec2-174-129-127-206.compute-1.amazonaws.com, 174.129.127.206,
role[web].
3 hours ago, i-5272a43f, ubuntu 10.04, ec2-184-73-9-250.compute-1.amazonaws.com, 184.73.9.250,
role[web].
3 hours ago, i-226ca64f, ubuntu 10.04, ec2-75-101-240-230.compute-1.amazonaws.com, 75.101.240.230,
role[web].
3 hours ago, i-f65c969b, ubuntu 10.04, ec2-184-73-60-141.compute-1.amazonaws.com, 184.73.60.141,
role[web].
```

show the status of a subset of nodes returned by a search query

```
% knife status "role:web" --run-list
3 hours ago, i-225f954f, ubuntu 10.04, ec2-67-202-63-102.compute-1.amazonaws.com, 67.202.63.102,
role[web].
3 hours ago, i-a45298c9, ubuntu 10.04, ec2-174-129-127-206.compute-1.amazonaws.com, 174.129.127.206,
role[web].
3 hours ago, i-5272a43f, ubuntu 10.04, ec2-184-73-9-250.compute-1.amazonaws.com, 184.73.9.250,
role[web].
3 hours ago, i-226ca64f, ubuntu 10.04, ec2-75-101-240-230.compute-1.amazonaws.com, 75.101.240.230,
role[web].
3 hours ago, i-f65c969b, ubuntu 10.04, ec2-184-73-60-141.compute-1.amazonaws.com, 184.73.60.141,
role[web].
```

Tag

Apply tags to nodes on a Chef Server.

return to top of page

create

Adds one or more tags to node.

```
knife tag create NODE TAGS...
```

delete

Removes one or more tags from node.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife tag delete NODE TAGS...
```

list

Lists the tags applied to node.

```
knife tag list NODES...
```

See also Managing Nodes with Knife

Extras

Bash command completion

Additional convenience may be provided using bash command completion.

return to top of page

Download the script and either source it in `.bashrc` or drop it in `/etc/bash_completion.d/`.

The completion employs optional caching if the directory `$CHEF_HOME/.completion_cache` exists and is writable (`CHEF_HOME` is `~/.chef` by default). To clear the cache (e.g. because new roles or nodes are available) erase the files in the cache directory, they will be regenerated on the next completion run.

Note for OSX users: The script requires GNU sed to work, install `gsed` with macports or `gnu-sed` with Homebrew. You will also need the `bash-completion` script library if you do not already have it installed – a package exists named as such in macports and Homebrew (e.g. `brew install bash-completion`). Use the `etc/bash_completion.d` prefix appropriate for your environment.

 Knife

Knife Bootstrap 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Knife Bootstrap



The Bootstrap subcommand for Knife performs a Chef Bootstrap on the target node.

Knife commands all have the same form

```
knife sub-command [ARGUMENTS] (options)
```

The goal of the bootstrap is to get Chef installed on the target system so it can run Chef Client with a Chef Server (or Hosted Chef).

The main assumption is a baseline OS installation exists. This sub-command is used internally by some cloud computing server create commands and the others will be migrated in a future version of Chef. It is primarily intended for Chef Client systems that talk to a Chef server.

As of Chef 0.9.8, the bootstrap sub-command supports supplying a template to perform the bootstrap steps. If the distro is not specified (via `-d` or `--distro` option), an Ubuntu 10.04 with RubyGems is assumed. The `DISTRO` value corresponds to the base filename of the template, in other words `DISTRO.erb`. A template file can be specified with the `--template-file` option in which case the `DISTRO` is not used. The sub-command looks in the following locations for the template to use:

- `bootstrap` directory in the installed Chef Knife library (Location varies depending how you installed Chef).
- `bootstrap` directory in the `$PWD/.chef`, e.g. in `~/chef-repo/.chef`.
- `bootstrap` directory in the users `$HOME/.chef`.

Man Page in Chef 0.10+

Additional knife-bootstrap detail is available on your local system as a man page. Type `knife help bootstrap` to view it.

Additional Options

- `-i`, `--identity-file IDENTITY_FILE`:
  The SSH identity file used for authentication
- `-N`, `--node-name NAME`:
  The Chef node name for your new node
- `-P`, `--ssh-password PASSWORD`:
  The ssh password
- `-x`, `--ssh-user USERNAME`:
  The ssh username
- `-p`, `--ssh-port PORT`:
  The ssh port
- `--bootstrap-version VERSION`:
  The version of Chef to install
- `--bootstrap-proxy PROXY_URL`:
  The proxy server for the node being bootstrapped
- `--prerelease`:
  Install pre-release Chef gems
- `-r`, `--run-list RUN_LIST`:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Comma separated list of roles/recipes to apply
- `--template-file TEMPLATE`:
  Full path to location of template to use
- `--sudo`:
  Execute the bootstrap via sudo
- `-d`, `--distro DISTRO`:
  Bootstrap a distro using a template
- `--no-host-key-verify`:
  Disable host key verification

Supported Distros

The default bootstrap templates are scripts that get copied to the target node (FQDN). As of Chef 0.9.8, the following distros are supported:

- centos5-gems
- fedora13-gems
- ubuntu10.04-gems
- ubuntu10.04-apt

Authentication

`Knife bootstrap` communicates with the target node over SSH. We recommend using key-based authentication. You can pass the path to the private key used to authenticate to the node using the `-i` command line option or use a program such as `ssh-agent` to manage your keys automatically. `knife bootstrap` also supports password-based authentication. You can use the `-P` option to pass the password directly on the command line or, with Chef 0.10.6+, allow `knife bootstrap` to prompt you for a password.

Examples

Here are two example bootstrap commands, using different types of authentication. In this example we pass it the SSH password as part of the command:

```
knife bootstrap IP_ADDRESS -x ubuntu -P PASSWORD --sudo
```

And in this example, we use a private key file:

```
knife bootstrap IP_ADDRESS -x ubuntu -i ~/.ssh/id_rsa --sudo
```

Afterwards you'll be prompted for your sudo password on the node you are SSHing into. You will want to change IP_ADDRESS to the ip or hostname you are trying to bootstrap, and this assumes you want to login with the username 'ubuntu'.

Known Authentication Bug

*If you have not customized your ssh configuration, you will most likely **not** be affected by this bug.*

Knife uses the **net-ssh** gem to create ssh connections to your nodes. Currently, net-ssh processes your ssh configuration files (e.g. ~/.ssh/config) differently than your system's ssh client. One difference is that if you explicitly set various authentication methods to "yes", then all authentications *not* explicitly set to "yes" will be excluded. For example, if you set the following in your configuration file:

```
PasswordAuthentication yes
```

Other methods of authentication, such as key based authentication, will not be attempted. You can fix this either by removing the explicit authentication directive in your configuration file, or by ensuring that **all** methods you want to use are set to "yes", even if "yes" is typically their default value on your system. To explicitly allow key-based authentication, use the following setting

```
PubkeyAuthentication yes
```

For more information about this bug see CHEF-2783.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Creating Custom Bootstrap Templates

Custom bootstrap templates can be created and stored in `.chef/bootstrap/DISTRO.erb`, replacing `DISTRO` with the value passed with the `-d` or `--distro` option. **Custom Knife Bootstrap Script** has a couple of examples on the creation of a custom bootstrap template..

Installs RubyGems and Chef as a gem

The gems installations will use RubyGems 1.3.6 and Chef installed as a gem. A future release will install RubyGems 1.3.7, though you can customize this yourself by copying the template to one of the locations specified above. The apt installation will use the Opscode APT repository.

Writes Config Files

In addition to handling the software installation, these bootstrap templates do the following:

- Write the validation.pem per the local knife configuration.
- Write a default config file for Chef (`/etc/chef/client.rb`) using values from the `knife.rb`.
- Create a JSON attributes file containing the specified run list and run Chef.

In the case of the RubyGems, the `client.rb` will be written from scratch with a minimal set of values, and you may want to customize it. In the case of APT Package installation, `client.rb` will have the `validation_client_name` appended if it is not set to `chef-validator` (default config value), and the `node_name` will be added if `chef_node_name` option is specified.

Using encrypted data bags

As of 0.10.6, `knife bootstrap` supports the 'encrypted_data_bag_secret' setting in knife.rb. You will want to add this line to your knife.rb:

```
encrypted_data_bag_secret '/path/to/your/data_bag_key'
```

And change '/path/to/your/data_bag_key' to the location of where the data bag key is located. When you run knife bootstrap afterwards it automatically adds this line to the client.rb for the node you are bootstrapping and copies the key over.

Completed State

When this is complete, the bootstrapped node will have:

- Latest Chef version installed from RubyGems or APT Packages from Opscode. This may be a later version than the local system.
- Be validated with the configured Chef Server.
- Have run Chef with its default run list if one is specified.

If The Run Does Not Complete

The run list for the first run of a newly provisioned node is passed in by knife as a file with the special command line:

```
chef-client -j /etc/chef/first-boot.json
```

At the end of the first run, the node object is saved with this list, shows up on the console and knife commands, and is used on subsequent chef-client runs. If the run does not complete, (the instance is created but the installs are not completed), then the node with the run list applied to it is never saved.

Run the above command if this occurs. The normal state is for the first chef-client run to complete - an incomplete run is the sort of thing that will happen while debugging changes and new recipes. A typical cause is an unmet dependency in cookbooks. Review the metadata.rb files of the cookbooks in your base role, or any other roles being established in the initial run for any dependencies.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Knife Built In Subcommands



Custom Knife Bootstrap Script



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Custom Knife Bootstrap Script



The Knife Bootstrap scripts included with Chef by default are meant to be generalized, and implement instructions found for installing Chef clients.

This means that they may not be perfect for your specific needs in their default state.

If that is the case, you can modify them, or create your own custom scripts. The following are a couple of examples of creating custom scripts or bootstrap templates.

Modifying the default script for ubuntu 10.04.

Copy the bootstrap template from the default location. For example I installed Chef as a RubyGem, so I can get the full path of the file out of the gem contents:

```
% gem contents chef | grep ubuntu10.04-gems
/Users/jtimberman/.rvm/gems/ruby-1.9.2-p180/gems/chef-0.10.2/lib/chef/knife/bootstrap/ubuntu10.04-gem
s.erb
```

Copy the template to the Chef Repository in the `.chef/bootstrap` directory.

```
% cp
/Users/jtimberman/.rvm/gems/ruby-1.9.2-p180/gems/chef-0.10.2/lib/chef/knife/bootstrap/ubuntu10.04-gem
s.erb ~/chef-repo/.chef/bootstrap/ubuntu10.04-gems-mine.erb
```

Modify the template with your favorite editor, then use it with the `-d` or `--distro` option in the Knife Bootstrap or cloud computing plugin commands.

```
% knife bootstrap 192.168.1.100 -r 'role[webserver]' -d ubuntu10.04-gems-mine
```

Alternatively, an example bootstrap template can be found in the chef source repository. Copy the template to `~/.chef-repo/.chef/bootstrap/ubuntu10.04-apt.erb` and modify to suit your environment.

Create a bootstrap for Debian systems using APT packages.

Example client config (`/etc/chef/client.rb`) with Hosted Chef as the Chef Server:

```
log_level        :info
log_location     STDOUT
chef_server_url  'https://api.opscode.com/organizations/ORGNAME'
validation_client_name 'ORGNAME-validator'
```

As it says on Knife Bootstrap, The knife bootstrap sub-command looks in three locations for the template to use:

- `bootstrap` directory in the installed Chef Knife library (Location varies depending how you installed Chef).
- `bootstrap` directory in the `$PWD/.chef`, e.g. in `~/chef-repo/.chef`.
- `bootstrap` directory in the users `$HOME/.chef`.

So, using the second location from the above, we'd first create `.chef/bootstrap/` in your Chef Repository directory, then create the ERB template file.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
mkdir ~/.chef/bootstrap
vi ~/.chef/bootstrap/debian5.0-apt.erb
```

Edit the template to run the commands, set up the validation certificate and the client configuration file, and finally to run chef-client on completion with the given run list (-r option).

The bootstrap template can be called with:

```
knife bootstrap mynode.example.com -r 'role[webserver]','role[production]' --distro debian5.0-apt
```

 Knife Bootstrap

Knife Windows Bootstrap 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Knife Windows Bootstrap



New in 10.0
This feature is new for Chef 0.10.

Description

This Knife Plugin adds additional functionality to the Chef Knife CLI tool for configuring/interacting with nodes running Microsoft Windows.

The subcommands should function on any system running Ruby 1.9.1+ but nodes being configured via these subcommands require Windows Remote Management (WinRM) 1.0+. WinRM allows you to call native objects in Windows, which includes, but is not limited to:

- running batch scripts,
- powershell scripts and
- fetching WMI variables.

For more information on WinRM, please visit Microsoft's WinRM site.

You will want to familiarize yourself with (certain key aspects) of WinRM because you will be writing scripts/running commands with this tool to get you from (specific point A) to (specific point B). WinRM is built into Windows 7 and Windows Server 2008+. It can also be easily installed in older version of Windows, including:

- Windows XP
- Windows Server 2003
- Windows Vista

More information can be found on Microsoft Support article 968930.

Opscode Windows Cookbooks

Providing the setup, automation and maintenance of Windows-based servers and applications

- **Powershell**
- **IIS**
- **SQL-Server**
- **7-zip**
- **Windows Primitives**
- **Web Platform Installer (WebPI)**
- **Windows Installer XML (WiX)**

Requirements/Version

- Chef is supported on Windows Server 2008 R2, 2003 R2 and Windows 7. Additionally, it is known to run on Windows Vista and XP.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- Knife Plugins require Chef 0.10 or greater.
- knife-windows requires Ruby 1.9.1 or greater.
- The following modifications should be made to hosts that you would like to communicate with using WinRM:

  The following should be run within cmd.exe, not Powershell, in order to avoid issues with escaping the command line.

  - A server running WinRM must be configured properly to allow outside connections and the entire network path from the knife workstation to the server. The easiest way to accomplish this is to use WinRM's quick configuration option:

    ```
    C:\Users\Administrator> winrm quickconfig -q
    ```

  - The Chef and Ohai gem installations (that occur during bootstrap) take more memory than the default 150MB WinRM allocates per shell. Bump it up to 300MB with the following setting:

    ```
    C:\Users\Administrator> winrm set winrm/config/winrs @{MaxMemoryPerShellMB="300"}
    ```

  - Bootstrap commands can take longer than the WinRM default 60 seconds to complete, bump to 30 minutes:

    ```
    C:\Users\Administrator> winrm set winrm/config @{MaxTimeoutms="1800000"}
    ```

    WinRM supports both the HTTP and HTTPS transports and the following authentication schemes: Kerberos, Digest, Certificate and Basic. The details of these authentication transports are outside of the scope of this README but details can be found on the WinRM configuration guide. Currently, this plugin support Kerberos and Basic authentication schemes.//

  - For development and testing purposes, unencrypted traffic with Basic authentication can make things easier:

    ```
    C:\Users\Administrator> winrm set winrm/config/service @{AllowUnencrypted="true"}
    C:\Users\Administrator> winrm set winrm/config/service/auth @{Basic="true"}
    ```

WinRM installation must be configured correctly
Before any WinRM related knife subcommands will function correctly a node's WinRM installation must be configured correctly. The above settings should be added to your base server image (AMI) or passed in using some sort of user-data mechanism provided by your cloud provider.

Installation

Knife Plugins ship as Ruby gems so knife-windows is easily installable.

```
gem install knife-windows
```

Since knife-windows is written in pure Ruby (just like Chef) you should be able to install it on any platform that Chef supports. This means you can manage your Windows nodes from the comfort of bash on your favorite Linux distro or better yet zsh on OS X if you'd like!

Subcommands

This plugin provides the following Knife subcommands. **Specific command options can be found by invoking the subcommand with a —help flag.**

knife winrm

The winrm subcommand allows you to invoke commands in parallel on a subset of the nodes in your infrastructure. The winrm subcommand uses

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

the same syntax as the search subcommand; you could could find the uptime of all your web servers using the command:

```
% knife winrm "role:web" "net stats srv" -m -x Administrator -P 'super_secret_password'
```

You could also invoke everyone's favorite Windows command ipconfig against a single server (vs a search query in the prior example) using the following command:

```
knife winrm 'ec2-50-xx-xx-124.compute-1.amazonaws.com' 'ipconfig' -m -x Administrator -P
'secret_password'
```

Or force a chef run:

```
% knife winrm 'ec2-50-xx-xx-124.compute-1.amazonaws.com' 'chef-client -c c:/chef/client.rb' -m -x
Administrator -P 'super_secret_password'
    ec2-50-xx-xx-124.compute-1.amazonaws.com [Fri, 04 Mar 2011 22:00:49 +0000] INFO: Starting Chef Run
(Version 0.9.12)
    ec2-50-xx-xx-124.compute-1.amazonaws.com [Fri, 04 Mar 2011 22:00:50 +0000] WARN: Node ip-0A502FFB
has an empty run list.
    ec2-50-xx-xx-124.compute-1.amazonaws.com [Fri, 04 Mar 2011 22:00:53 +0000] INFO: Chef Run complete
in 4.383966 seconds
    ec2-50-xx-xx-124.compute-1.amazonaws.com [Fri, 04 Mar 2011 22:00:53 +0000] INFO: cleaning the
checksum cache
    ec2-50-xx-xx-124.compute-1.amazonaws.com [Fri, 04 Mar 2011 22:00:53 +0000] INFO: Running report
handlers
    ec2-50-xx-xx-124.compute-1.amazonaws.com [Fri, 04 Mar 2011 22:00:53 +0000] INFO: Report handlers
complete
```

This subcommand operates in a very similar manner as knife ssh leveraging the WinRM protocol for communication. It also include's knife ssh's "interactive session mode"

knife bootstrap windows winrm

Performs a Chef Bootstrap (via the WinRM protocol) on the target node. The goal of the bootstrap is to get Chef installed on the target system so it can run Chef Client with a Chef Server. The main assumption is a baseline OS installation exists. It is primarily intended for Chef Client systems that talk to a Chef server.

In the future this subcommand will be used internally by some cloud computing server create commands like the current **knife bootstrap** subcommand is. This subcommand operates in a very similar manner as knife bootstrap leveraging the WinRM protocol for communication. An initial run_list for the node can also be passed to the subcommand. Example usage:

```
knife bootstrap windows winrm ec2-50-xx-xx-124.compute-1.amazonaws.com -r
'role[webserver]','role[production]' -x Administrator -P 'super_secret_password'
```

knife bootstrap windows ssh

Performs a Chef Bootstrap (via the SSH protocol) on the target node. The goal of the bootstrap is to get Chef installed on the target system so it can run Chef Client with a Chef Server. The main assumption is a baseline OS installation exists. It is primarily intended for Chef Client systems that talk to a Chef server.

This subcommand assumes the SSH session will use Windows native cmd.exe command shell vs a bash shell through an emulated cygwin layer. Most popular Windows based SSHd daemons like freeSSHd and WinSSHD behave this way.

An initial run_list for the node can also be passed to the subcommand. Example usage:

```
knife bootstrap windows ssh ec2-50-xx-xx-124.compute-1.amazonaws.com -r
'role[webserver],role[production]' -x Administrator -i ~/.ssh/id_rsa
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Bootstrap Templates

This gem provides the following bootstrap templates:

windows-chef-client-msi

This bootstrap template does the following:

1. Installs the latest version of Chef (and all dependencies) using the `chef-client` msi.
2. Writes the validation.pem per the local knife configuration.
3. Writes a default config file for Chef (C:\chef\client.rb) using values from the knife.rb.
4. Creates a JSON attributes file containing the specified run list and run Chef.

This is the default bootstrap template used by both the windows bootstrap subcommands.

windows-shell

This bootstrap template does the following:

1. Installs Ruby 1.8.7 via the Ruby Installer for Windows which also includes the latest version of RubyGems
2. Installs the RubyInstaller Development Kit (DevKit). The RubyInstaller Development Kit (DevKit) is a MSYS/MinGW based toolkit than enables you to build many of the native C/C++ extensions available for Ruby.
3. Installs required Windows-related gems from RubyGems.org
4. Installs latest Chef version from RubyGems.org including Ohai and Chef.
5. Writes the validation.pem per the local knife configuration.
6. Writes a default config file for Chef (C:\chef\client.rb) using values from the knife.rb.
7. Creates a JSON attributes file containing the specified run list and run Chef.

This should be considered a legacy bootstrap template and will most likely be removed in a future version.

 Custom Knife Bootstrap Script

Knife Exec 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Knife Exec



Knife sub-command usage.

Overview

The `knife exec` subcommand allows you to write Ruby scripts that are executed in the context of a fully configured Chef API Client using the knife configuration file. Knife exec scripts also share the same DSL as Shef's main mode - any commands available in the top-level of shef `help` output are available.

The purpose of Knife Exec is slightly different than Knife Plugins. *This sub-command is often used for scripts that should access the Chef Server API for one-off type operations and don't warrant full blown*

Note the Context

As you are writing Ruby scripts that are executed in the context of a fully configured Chef API Client using the knife configuration file, the following are some details of note:

- The script lives on your management workstation, not the nodes. (i.e. on the system from which you are invoking knife, not on the systems that you are managing with chef)
  - The context of shell commands issued will therefore be your management workstation.
  - Something like `%x[ls -lash /opt/only-on-a-node]` would give you the directory listing for your /opt/only-on-a-node directory, or the error "no such file exists" if the directory does not exist locally.
- While the Shef's DSL is available, the chef-client DSL is not
  - Unless your management workstation is also a chef-client being managed by the chef-server.
  - Without the chef-client DSL, you cannot use a bash block to run bash commands.

Invocation

Invoke knife exec in one of three ways to write scripts.

```
knife exec /path/to/script_file
```

```
knife exec -E 'RUBY CODE'
```

```
knife exec
RUBY CODE
^D
```

Examples

Some one-line examples

List the search indexes available.

```
knife exec -E 'puts api.get("search").keys'
```

Show all the nodes and their free memory available.

```
knife exec -E 'nodes.all {|n| puts "#{n.name} has #{n.memory.total} free memory"}'
```


This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Tutorials from the Community

- **Knife Exec to Search for Systems Needing Updates**

Opscode team member Bryan McLellan wrote a blog post on **Knife Reporting: apt + updates**, using a Knife exec script to search for and describe systems needing updates.

This blog is helpful for knowing how to issue shell commands in the context of your managed nodes.

- **Knife Exec + Fog to Reconcile EC2 Instances**

Community member James Sulinski has a most excellent blog post on using Knife Exec and Fog and Highline gems to **reconcile the EC2 and Chef API's** and output it in a way which provides a beneficial overview. (Code - knife.reconcile)

- **Command-line Cookbook Dependency Solving with Knife Exec**

Need to replicate a node setup elsewhere, and aren't sure of all the cookbook dependencies? Or, just want to validate that you've uploaded all the cookbooks that are stated as being required?

Community member Stephen Nelson-Smith has a blog post on **Dependency Solving** using Knife Exec.

More detailed examples

A slightly different take on the knife status command.

scripts/status.rb

```
printf "%-5s %-12s %-8s %s\n", "Check In", "Name", "Ruby", "Recipes"
nodes.all do |n|
  checkin = Time.at(n['ohai_time']).strftime("%F %R")
  rubyver = n['languages']['ruby']['version']
  recipes = n.run_list.expand.recipes.join(", ")
  printf "%-20s %-12s %-8s %s\n", checkin, n.name, rubyver, recipes
end
```

```
knife exec scripts/status.rb
```

Post some status information to a google spreadsheet.

Requires that you install the google-spreadsheet-ruby gem and configure GOOGLE_EMAIL and GOOGLE_PASSWORD as environment variables with the appropriate values in your shell.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

scripts/google_status.rb

```
require 'google_spreadsheet'

session = GoogleSpreadsheet.login ENV["GOOGLE_EMAIL"], ENV["GOOGLE_PASSWORD"]
sheet = session.create_spreadsheet "Instances #{Time.now.strftime "%Y%m%d"}"

ws = sheet.worksheets.first

ws[1,1] = "Check In"
ws[1,2] = "Node Name"
ws[1,3] = "Ruby Version"
ws[1,4] = "Recipes"

i = 2
nodes.all do |n|
  ws[i,1] = Time.at(n['ohai_time']).strftime("%F %R")
  ws[i,2] = n.name
  ws[i,3] = n['languages']['ruby']['version']
  ws[i,4] = n.run_list.expand.recipes.join(", ")
  i = i + 1
end

ws.save
```

```
knife exec scripts/google_status.rb
```

(Thanks, Darrin Eden, for the Google Spreadsheet code!)

Query a node for multiple attributes.

This is a ruby script executed in the context of a fully configured Chef client that can talk to the API.

scripts/search_attributes.rb

```
% cat scripts/search_attributes.rb
query = ARGV[2]
attributes = ARGV[3].split(",")
puts "Your query: #{query}"
puts "Your attributes: #{attributes.join(" ")}"
results = {}
search(:node, query) do |n|
  results[n.name] = {}
  attributes.each {|a| results[n.name][a] = n[a]}
end

puts results
exit 0
```

Example usage and output:

```
% knife exec scripts/search_attributes.rb "hostname:test_system" ipaddress,fqdn
Your query: hostname:test_system
Your attributes: ipaddress fqdn
{"test_system.example.com"=>{"ipaddress"=>"10.1.1.200", "fqdn"=>"test_system.example.com"}}
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Using Chef Knife to Delete Node Attributes

There are a number of items that must be determined initially, and apply against the script settings

1. Determine the type of attribute that needs to be set
2. Determine the name of the attribute
3. Determine the set of nodes that it needs to be set on
    - organization?
    - environment?
    - role?

Note: Because of the way node attributes work, you need to use override_attrs.delete instead of override.delete

```
$ knife exec -E 'nodes.all { |n| puts n[:thing] }'
override-thing
nil
nil
nil
$ knife exec -E 'nodes.all { |n| n.override.key?("thing") ? ( n.override_attrs.delete("thing") ;
n.save ) : false  } '
$ knife exec -E 'nodes.all { |n| puts n[:thing] }'
normal-thing
nil
nil
nil
```

Example of changing a 'normal' attribute using set for all nodes with a certain role via knife exec:

```
$ knife node show blackie -a important_setting
important_setting:  orange
$ knife exec -E "nodes.find(:role => 'fixme') {|n| n.set['important_setting'] = 'purple' ; n.save }"
$ knife node show blackie -a important_setting
important_setting:  purple
```

Passing Arguments to Knife Scripts

Command-line arguments can be passed into a Knife script and read in using Ruby's global ARGV array. To ensure command line arguments are not later treated as additional files to be executed by knife, end your script with `exit 0` (see CHEF-1973).

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

scripts/stale_nodes.knife

```
# returns a list of nodes that have
# checked-in within the last MINUTES

abort("usage: knife exec stale_nodes.knife MINUTES") unless ARGV[2]
mintues_ago = ARGV[2].to_i
stale_nodes = search(:node, '*:*').select{|n| n.attribute?('ohai_time') && (n.ohai_time < (Time.now -
mintues_ago*60).to_f)}

# sort the list by checkin time
stale_nodes.sort!{|r1,r2| r1.ohai_time <=> r2.ohai_time}

printf "%-30s %-40s %-30s\n", "Last Checkin", "Host", "Run List"
stale_nodes.each do |n|
  printf "%-30s %-40s %-30s\n", Time.at(n.ohai_time), n.name, n.run_list.sort{|r1,r2| r1.name <=>
r2.name}
end

# Prevents knife from trying to execute any command line arguments as addtional script files, see
CHEF-1973
exit 0
```

```
knife exec scripts/stale_nodes.knife 60
```

Use Case Scenario

Is there a reason why deleting a cloud instance using knife does not delete the instance in the management console? We spin instances up and down constantly, and while we do not have more than 5 instances at any given time there are now 40 instances in the dashboard. Managing them manually using a dashboard seems counterintuitive.

The reason the node is left behind is because there are cases in which one would want to preserve the node data beyond the life of the specific cloud instance. There is an active discussion about best practices, and a number of people have requested the behavior you desire (instance and node deleted together) as well. While best practices are in discussion and an automated solution that addresses both needs is considered, in the current scenario, there are various approaches that can be undertaken. Some use a small shell function that runs `knife ec2 server delete`, `knife node delete`, and `knife client delete` in sequence. You could also construct a small `knife exec` script that does all three operations.

Here is an example:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
require 'fog'
def remove(name)

   ## Delete the Server
   delete_ec2 = Chef::Knife::Ec2ServerDelete.new()
   delete_ec2.name_args = [nodes.show(name)['ec2']['instance_id']]
   delete_ec2.run

   ## Delete the Node
   delete_node = Chef::Knife::NodeDelete.new()
   delete_node.name_args = [name]
   delete_node.run

   ## Delete the client
   delete_client = Chef::Knife::ClientDelete.new
   delete_client.name_args = [name]
   delete_client.run

end

remove(ARGV[2])
exit 0
```

This `knife-exec` script could be used by saving it as server-delete.rb and running it as follows:

```
knife exec server-delete.rb node_name
```

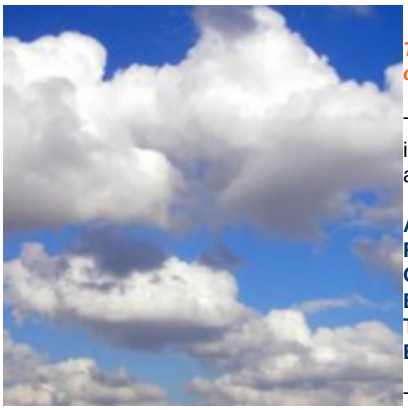
 Knife Windows Bootstrap

Launch Cloud Instances with Knife 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Launch Cloud Instances with Knife

Overview



This page describes how to launch API-driven "Cloud" servers with Knife and automatically run Chef on them.

This page covers the general use of the available cloud plugins. Reference documentation for the individual plugins is maintained in the README of the plugin's code repository, linked below where available. **Currently supported services (version 0.10.0 with plugins):**

Amazon EC2
Rackspace Cloud
Openstack
Bluebox
Terremark vCloud
Eucalyptus

These are supported via sub-commands in knife that use the Ruby library fog. During the "bootstrap" process that runs Chef, knife will SSH to the systems, and uses the net-ssh-multi library. Finally, knife uses colorized output via highline.

Install Dependancies

You will need to install the fog, net-ssh-multi and highline gems / libraries on your local management workstation where you use knife.

```
sudo gem install net-ssh net-ssh-multi fog highline --no-rdoc --no-ri --verbose
```

On Ubuntu you will also need to install some additional apt packages in order to compile the nokogiri gem (a fog dependency):

```
# ruby developer packages
sudo apt-get install ruby1.8-dev ruby1.8 ri1.8 rdoc1.8 irb1.8
sudo apt-get install libreadline-ruby1.8 libruby1.8 libopenssl-ruby

# nokogiri requirements
sudo apt-get install libxslt-dev libxml2-dev
```

Once complete, you'd move on to continuing to setup knife with your "cloud" credentials.

Cloud Credentials

Add the Cloud Credentials to your knife.rb. They are specifically named for each of the service providers Knife supports as keys under the `knife` configuration option.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
# EC2:
knife[:aws_access_key_id]     = "Your AWS Access Key"
knife[:aws_secret_access_key] = "Your AWS Secret Access Key"

# Rackspace:
knife[:rackspace_api_key]      = "Your Rackspace API Key"
knife[:rackspace_api_username] = "Your Rackspace API username"

# Slicehost
knife[:slicehost_password] = "Your Slicehost API password"

# Terremark
knife[:terremark_password] = "Your Terremark Password"
knife[:terremark_username] = "Your Terremark Username"
knife[:terremark_service]  = "Your Terremark Service name"

# Eucalyptus
knife[:euca_access_key_id]     = "Your Eucalyptus Access Key"
knife[:euca_secret_access_key] = "Your Eucalyptus Secret Access Key"
knife[:euca_api_endpoint]      = "http://ecc.eucalyptus.com:8773/services/Eucalyptus"
```

Add the appropriate lines for your preferred Cloud Computing service to your `knife.rb`.



Desire more EC2 specific information?

In addition to the use of the knife-ec2 detailed here - there is also an **EC2 Bootstrap Fast Start Guide** aimed at getting you up and running on EC2 as quick as possible, and an **AMI List for Developers** which can be useful for cookbook testing.

Tutorials from the Community

- Be Agile with EC2 and Knife

Our friends at Agile Web Operations provided this **Blog Post on Amazon EC2 Instances Using Knife**. Check it out!

- Chef and AWS Auto-Scaling

Community member Edward Sargisson has a nice blog post up on **Keeping an Amazon Elastic Compute Cloud (EC2) Instance Up with Chef and Auto Scaling.**

- Chef & Eucalyptus

Opscode joins Brady Murray from Eucalyptus Systems to conduct a Webinar on how to **Build Automated Private Cloud Infrastructure With Chef & Eucalyptus**. Check it out!

- Chef Managed EC2 Fedora Instances

Community member Steven Craig has a most excellent blog post providing an all-in-one to get you from zero to N-1 fully chef-managed **Fedora Amazon EC2 instances backed with custom EBS root devices in less than 60 minutes!**. Worthwhile!

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef 0.10: Command examples on this page

The command examples on this page now use Chef 0.10's knife plugin syntax. See the **Opscode Blog Post** for an overview.

Cloud Sub-commands



Cloud sub-commands for knife are available as Knife Plugins. The following plugins are provided by Opscode as RubyGems. Install the desired plugin to make the sub-commands described below available.

- knife-ec2
- knife-rackspace
- knife-openstack
- knife-bluebox
- knife-terremark
- knife-eucalyptus

Installing the knife-ec2 plugin

```
% sudo gem install knife-ec2 --no-rdoc --no-ri --verbose
```

The sub-commands for cloud management in knife follow the pattern:

```
knife SERVICE server ACTION
# server creation:
knife SERVICE server create -r [RUN LIST...] (options)
```

Where service is ec2, `rackspace`, `openstack`, `bluebox`, `openstack`, or `terremark`, euca each of the services has a `server create` sub-command that uses their API to launch a new instance of the specified flavor or image with a run list.

Enclose the run list items in quotes, as the brackets for `role[thing]` or `recipe[thing]` may be interpreted by your login shell. Multiple items can be specified as a comma separated list.

Creating the server with the knife plugin

Once the plugin is installed, you can create the server using the `knife SERVICE server create` command:

Launch a new EC2 instance with the webserver role

```
knife ec2 server create -r "role[webserver]" -I ami-2d4aa444 --flavor m1.small -G www,default -x
ubuntu -N server01
# multiple items in the run list:
knife ec2 server create -r "role[base],role[webserver]" -I ami-2d4aa444 -G www,default -x ubuntu
--node-name server01
```

EC2 Note: The AMI used above is just an example. You should use the most current AMI's for your OS. For instance: the Ubuntu AMIs are found at their site, in the version subdirectories under "releases".

To launch a new Rackspace instance with the webserver role

```
knife rackspace server create -r 'role[webserver]' --server-name server01 --node-name server01 --image
49 --flavor 2
knife rackspace server create -r 'role[base],role[webserver]' --server-name server01 -N server01
--image 49 --flavor 2
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Node name
If no node name is specified with the -N or --node-name flags, knife will automatically generate a node name based on information supplied by the cloud provider, such as a numeric server id. These machine generated names are generally less descriptive than is usually desirable.

Syntax for Run List in Chef 0.10
In the knife plugins used for these cloud computing commands with Chef 0.10, *the run list must be specified with the -r option*. Earlier versions of Chef did not use -r.

See the Knife documentation for more information on the syntax for the subcommands and further options.

Instance Bootstrap

The server creation sub-command `bootstrap` the instance. When the instance is launched, knife determines whether it is available, and once it is, ssh's to the instance using the specified username (or your local user if none specified) and does the following:

1. Installs Ruby and packages to support installing RubyGems with native extensions.
2. Installs RubyGems from source.
3. Installs Chef from RubyGems.
4. Creates the /etc/chef directory.
5. Writes your validation certificate to /etc/chef/validation.pem.
6. Writes an /etc/chef/client.rb config file.
7. Writes a JSON file, /etc/chef/first-boot.json with the roles and recipes specified as the run list.
8. Executes chef-client with the first-boot.json, connecting to the server in the client.rb.

Once the Chef run is complete, the new instance will be registered with the Chef Server and be saved as a node object. This process does not set up the Chef Client service, you'll need to add a recipe in the run list that does this.

You can see the config file that will be copied by creating one with:

```
knife configure client client-config
```

The client.rb and validation.pem will be copied to the specified directory, in this case `client-config`.

If The Run Does Not Complete

The run list for the first run of a newly provisioned node is passed in by knife as a file with the special command line:

```
chef-client -j /etc/chef/first-boot.json
```

At the end of the first run, the node object is saved with this list, shows up on the console and knife commands, and is used on subsequent chef-client runs. If the run does not complete, (the instance is created but the installs are not completed), then the node with the run list applied to it is never saved.

Run the above command if this occurs. The normal state is for the first chef-client run to complete - an incomplete run is the sort of thing that will happen while debugging changes and new recipes. A typical cause is an unmet dependency in cookbooks. Review the metadata.rb files of the cookbooks in your base role, or any other roles being established in the initial run for any dependencies.

Bootstrap Subcommand

Currently, the bootstrap subcommand is used on EC2 and Rackspace instances. This isn't cloud-provider specific, and can be used on any kind of system, like a bare metal server or a VMware machine.

The bootstrap sub-command is not used by the `terremark` or `openstack` or `euca` server creation yet:

- CHEF-1731 is an open ticket for Terremark to have this functionality
- Use the Deploying OpenStack with Chef Guide for Openstack

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

SSH Configuration

Some providers will require you to authenticate with them through an SSH Key. There are two parts you will need to setup, after creating the key with the hosting provider.

Identity File

The bootstrap process uses SSH, so you'll need to configure your system to SSH in a particular way. The easiest way to make sure that SSH uses your Cloud service's SSH key is to add it to your SSH key agent. For example, if you have saved the private key for EC2 to `~/.ssh/ec2_ssh_key`:

```
$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-RHHBa22270/agent.22270; export SSH_AUTH_SOCK;
SSH_AGENT_PID=22271; export SSH_AGENT_PID;
echo Agent pid 22271;

$ ssh-add ~/.ssh/ec2_ssh_key
```

On Mac OS X Snow Leopard, ssh-agent is already running by default when you use Terminal.

You also need to specify the current user to SSH as:

- On Canonical's Ubuntu AMIs, this is `ubuntu`.
- If you're using an image that uses a different user, specify it with `-x` in the server create command.

Once this is added, you will not need to supply an identity_file option when using the knife command. You will still need to provide the AWS SSH Key ID.

SSH Key ID

You can supply the key id with `-S` or by adding this line to your knife.rb file.

For EC2:

```
knife[:aws_ssh_key_id] = "KEY_PAIR_NAME"
```

For Openstack:

```
knife[:openstack_ssh_key_id] = "KEY_PAIR_NAME"
```

For Eucalyptus:

```
knife[:euca_ssh_key_id] = "KEY_PAIR_NAME"
```

The ID is the name of the key file from when you created it. For instance if your key was named KeyId.pem when you downloaded it, the KEY_PAIR_NAME will be "KeyId".

Configuration Options

You can find the full list of configuration options for each plugin by reading the readme included:

Amazon EC2
Rackspace Cloud
Openstack

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Bluebox
Terremark vCloud
Eucalyptus

Most of them support setting the flavor, image, distro, template_file through the knife.rb and some also support setting SSH options or the region.

For a list of all of the switches available with the plugin, you can use the command `knife SERVICE server create --help`.

Connecting to your cloud from its external FQDN

By default, knife will use the internal FQDN to connect to nodes. When connecting from outside of your cloud you may need to force this to use the external FQDN. You will want to avoid doing this unless necessary, as this will route all traffic out through the NAT infrustructure and could cause performance hits or additional usage charges.

If you're sure you want to do this, you can do it with the -a option, such as in this example:

```
knife ssh "role:webserver" "sudo chef-client" -a ec2.public_hostname
```

Deleting a server

To delete a server once you are done using it, you will want to use this command:

```
knife <plugin name> server delete <instance id>
```

The instance id can be located with the command '`knife <plugin name> server list`' or '`knife status`'. For example, you could delete a server using these commands if you were using the rackspace plugin:

```
$ knife rackspace server list
   Instance ID    Public IP      Private IP     Flavor         Image          Name           State
   12345678       1.1.1.1   192.168.1.1  3              49             slice12345678  active

$ knife rackspace server delete 12345678
   Instance ID: 12345678
   Host ID: testexample
   Name: slice12345678
   Flavor: 1GB server
   Image: Ubuntu 10.04 LTS (lucid)
   Public DNS Name: 1-1-1-1.static.cloud-ips.com
   Public IP Address: 1.1.1.1
   Private IP Address: 192.168.1.1

   Do you really want to delete this server? (Y/N) Y
   WARNING: Deleted server 12345678 named slice12345678
```

Make sure that you also clean up the associated 'node' and 'client' objects from hosted chef:

```
knife node delete 12345678
knife client delete 12345678
```

Knife Exec

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Managing API Clients With Knife



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing API Clients With Knife



Overview

Knife is a command-line tool that comes with Chef.

*It is used by administrators to interact with the Chef **Server API**, the local Chef repository and and can be used to create, edit, view, list, and delete API clients.*

Knife's `client` sub-command provides the ability to manage API clients.

For more information about Knife, refer to the **Knife** documentation.
See **API Clients** for more information on API Clients.

Knife commands all have the same form

```
knife sub-command [ARGUMENTS]
(options)
```

The following client arguments are available:

bulk delete

Delete clients where the client name matches the regular expression regex on the Chef Server. The regular expression should be given as a quoted string, and not surrounded by forward slashes.
Usage:

```
knife client bulk delete REGEX (options)
```

create

Create a new client. This generates an RSA keypair. The private key will be displayed on **STDOUT** or written to the named file. The public half will be stored on the Server. For **chef-client** systems, the private key should be copied to the system as `/etc/chef/client.pem`. If you do not copy the key and try running **chef-client** for the first time without it, you will get a 403 error because you won't have sufficient permissions to re-create the key.

Admin clients should be created for users that will use **knife** to access the API as an administrator. The private key will generally be copied to `~/.chef/client_name.pem` and referenced in the `knife.rb` configuration file. **Hosted Chef** users should note that marking clients as "Admin clients" will have no effect when using Hosted Chef.

Usage:

```
knife client create CLIENT_NAME (options)
```

Additional Options:

- `-a`, `--admin`:
  Create the client as an admin
- `-f`, `--file FILE`:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Write the key to a file

delete

Deletes a registered client.

Usage:

```
knife client delete CLIENT_NAME
```

edit

Edit a registered client.

Usage:

```
knife client edit CLIENT_NAME (options)
```

The clients record will be open with EDITOR and can be edited by the user.

list

List all registered clients

Usage:

```
knife client list (options)
```

Additional Options:

- `-w`, `--with-uri`:
  Show corresponding URIs

Example Output:

```
> knife client list
  exampleorg-validator
  i-12345678
  rs-123456
```

reregister

Regenerate the RSA keypair for a client. The public half will be stored on the server and the private key displayed on **STDOUT** or written to the named file. **This operation will invalidate the previous keypair used by the client, preventing it from authenticating with the Chef Server.** Use care when reregistering the validator client.

Usage:

```
knife client reregister CLIENT_NAME (options)
```

Additional Options:

- `-f`, `--file FILE`:
  Write the key to a file

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

show

Show a client. Output format is determined by the --format option.

Usage:

```
knife client show CLIENT_NAME (options)
```

Additional Options:

- `-a`, `--attribute ATTR`:
  Show only one attribute

Example Output

```
> knife client show testclient
admin:       false
chef_type:   client
json_class:  Chef::ApiClient
name:        testclient
public_key:
```

 Launch Cloud Instances with Knife

Managing Cookbooks With Knife 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Cookbooks With Knife



Overview

Knife is a command-line tool that comes with Chef.

*It is used by administrators to interact with the Chef **Server API**, the local Chef repository and and can be used to interact with cookbooks on the Chef Server and on the Cookbook Site.*

Knife's cookbook and cookbook `site` sub-commands provide this capability.

- For more information about Knife, refer to the **Knife** documentation.
- See **Cookbooks** for more information on Cookbooks.
- See **Cookbook Site Help** for assistance in interaction with the Opscode Community Cookbook Site.
- See **Cookbook Support** for support of cookbooks.

Knife commands all have the same form

```
knife sub-command [ARGUMENTS] (options)
```

Cookbook

The cookbook sub-commands are for interacting with cookbook on the Chef server. For commands related to interacting with cookbooks on the community site, see the documentation for **Cookbook Site**.

The following cookbook arguments are available:

bulk delete

Delete cookbooks on the chef server based on a regular expression. The regular expression (regex) should be in quotes, not in //'s.

Usage:

```
knife cookbook bulk delete REGEX (options)
```

Additional Options:

- `-p`, `--purge`:
  Purge files from backing store. This will disable any cookbook that contains any of the same files as the cookbook being purged.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

create

This is a helper command that creates a new cookbook directory in the `cookbook_path` configuration option.

Usage:

```
knife cookbook create COOKBOOK_NAME (options)
```

Additional Options

- `-o`, `--cookbook-path path`:
  the directory where the cookbook will be created
- `-r`, `--readme-format format`:
  format of the readme file md, mkd, txt, rdoc
- `-c`, `--copyright copyright`:
  name of copyright holder
- `-i`, `--license license`:
  license for cookbook, apachev2 or none
- `-e`, `--email email`:
  email address of cookbook maintainer

Details
The following directories and files are created for the named cookbook.

- cookbook/attributes
- cookbook/definitions
- cookbook/files/default
- cookbook/libraries
- cookbook/metadata.rb
- cookbook/providers
- cookbook/readme.md
- cookbook/recipes/default.rb
- cookbook/resources
- cookbook/templates/default

Supported readme formats are 'md' (default), 'mkd', 'txt', 'rdoc'. the readme file will be written with the specified extension and a set of helpful starting headers.

Specifying `-c` or `--copyright` with the name of the copyright holder as your name or your company/organization name in a quoted string will place the appropriate copyright notice in the pre-created files. If this value is not specified an all-caps string `your_company_name` is used, which can be easily changed with find/replace.

Specifying `-i` or `--license` with the license that the cookbook is distributed under for sharing with other people or posting to the opscode cookbooks site will place the appropriate license notice in the pre-created files. Be aware of the licenses of files you put inside the cookbook and follow any restrictions they describe.

The cookbook copyright, license, email and readme_format settings can be filled in the `knife.rb`, for example with default values:

```
cookbook_copyright "your_company_name"
    cookbook_license "none"
    cookbook_email "your_email"
    readme_format "md"
```

delete

Deletes the specified **version** of the named **cookbook**. If no version is specified, and only one version exists on the server, that version will be deleted. If multiple versions are available on the server, you will be prompted for a version to delete.

Usage:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife cookbook delete cookbook [VERSION] (options)
```

Additional Options:

- `-a`, `--all`:
  delete all versions
- `-p`, `--purge`:
  purge files from backing store. this will disable any cookbook that contains any of the same files as the cookbook being purged.

download

Downloads a cookbook from the chef server. If no version is specified and only one version exists on the server, that version will be downloaded. If no version is specified and multiple versions are available on the server, you will be prompted for a version to download.

Usage:

```
knife cookbook download cookbook [version]__ _(options)_
```

Additional Options:

- `-d`, `--dir download_directory`:
  the directory to download the cookbook into
- `-f`, `--force`:
  overwrite an existing directory with the download
- `-n`, `--latest`:
  download the latest version of the cookbook

list

Lists the cookbooks available on the Chef server.

Usage:

```
knife cookbook list (options)
```

Additional Options:

- `-a`, `--all`:
  show all versions of a cookbook instead of just the most recent
- `-w`, `--with-uri`:
  show corresponding uris

metadata from file

Loads the cookbook metadata from a specified file.

Usage:

```
knife cookbook metadata from file FILE (options)
```

show

Shows information about a particular cookbook, particular part of a cookbook, or a particular file in a cookbook.

Usage

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife cookbook show COOKBOOK [VERSION] [PART] [FILENAME] (options)
```

Additional Options

- `-f`, `--fqdn fqdn`:
  the fqdn of the host to see the file for
- `-p`, `--platform platform`:
  the platform to see the file for
- `-v`, `--platform-version version`:
  the platform version to see the file for
- `-w`, `--with-uri`:
  Show corresponding URIs

The following parts can be viewed using `knife cookbook show COOKBOOK VERSION PART`

- attributes
- definitions
- files
- libraries
- providers
- recipes
- resources
- templates

Example Output

Calling `show` against a cookbook without other arguments returns a list of the available versions:

```
> knife cookbook show getting-started
getting-started   0.3.0  0.2.0
```

Giving both the cookbook name and version will list a large amount of data about the cookbook:

```
> knife cookbook show getting-started 0.3.0
attributes:
    checksum:     fa0fc4abf3f6787aeb5c3c5c35de667c
    name:         default.rb
    path:         attributes/default.rb
    specificity:  default
    url:          https://somelongurlhere.com
chef_type:      cookbook_version
cookbook_name:  getting-started
definitions:    []
files:          []
frozen?:        false
json_class:     Chef::CookbookVersion
libraries:      []
```

Further specifying the part will show information regarding that part of the cookbook:

```
knife cookbook show getting-started 0.3.0 templates
  checksum:     a29d6f254577b830091f140c3a78b1fe
  name:         chef-getting-started.txt.erb
  path:         templates/default/chef-getting-started.txt.erb
  specificity:  default
  url:          https://someurlhere.com
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

test

Test the specified cookbooks for syntax errors. This uses the built-in ruby syntax checking option for files in the cookbook ending in `.rb`, and the erb syntax check for files ending in `.erb` (templates).

Usage:

```
knife cookbook test [COOKBOOKS...] (options)
```

Additional Options:

- `-a`, `--all`:
  test all cookbooks, rather than just a single cookbook
- `-o`, `--cookbook-path PATH:PATH`:
  a colon-separated path to look for cookbooks in

upload

Uploads one or more cookbooks from your local cookbook repository to the Chef Server. Only files that don't yet exist on the server will be uploaded.

Usage:

```
knife cookbook upload [COOKBOOKS...] (options)
```

Additional Options:

- `-a`, `--all`:
  upload all cookbooks, rather than just a single cookbook
- `-o`, `--cookbook-path path:path`:
  a colon-separated path to look for cookbooks in
- `-d`, `--upload-dependencies`:
  Uploads additional cookbooks that this cookbook lists in as dependencies in its metadata.
- `-E`, `--environment ENVIRONMENT`:
  An **ENVIRONMENT** to apply the uploaded cookbooks to. Specifying this option will cause knife to edit the **ENVIRONMENT** to place a strict version constraint on the cookbook version(s) uploaded.
- `--freeze`:
  Sets the frozen flag on the uploaded cookbook(s) Any future attempt to modify the cookbook without changing the version number will return an error unless `--force` is specified.
- `--force`:
  Overrides the frozen flag on a cookbook, allowing you to overwrite a cookbook version that has previously been uploaded with the `--freeze` option.

chefignore, with version 0.10
With chef version 0.10, you can use a single `chefignore` file placed in the root of your cookbook repository (your cookbook path, usually "cookbooks/" under your chef repository), to determine which files to ignore when uploading cookbooks. See the Opscode Blog Post on this enhanced functionality for details.

Cookbook Site

The *cookbook site* sub-commands are meant for interacting with the community cookbook site: community.opscode.com.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

return to top of page

For commands that simply read from the cookbok site (such as download, search, install, and list) you do not need an account on community.opscode.com. For commands that write to the site you need an account on the community site.

download

Downloads a specific cookbook from the community site, optionally specifying a certain version. The cookbook will be downloaded as a `tar.gz` a rchive and placed in your current working directory.

Usage:

```
knife cookbook site download COOKBOOK_NAME [VERSION] (options)
```

Additional Options:

- `-f`, `--file FILE`:
  The filename to write to
- `--force`:
  Force download deprecated cookbook

Example:

```
> knife cookbook site download getting-started
Downloading getting-started from the cookbooks site at version 0.3.0 to
/Users/sdanna/opscodesupport/getting-started-0.3.0.tar.gz
Cookbook saved: /Users/sdanna/opscodesupport/getting-started-0.3.0.tar.gz
```

install

Installs a cookbook from the community site to your local repository using the git version control system.

Usage:

```
cookbook site install COOKBOOK [VERSION] (options)
```

Additional Options:

- `-D`, {{--no-dependencies }}:
  Do not install dependencies automatically
- `-o`, `--cookbook-path PATH`:
  Install cookbooks to PATH
- `-B`, `--branch BRANCH`:
  Default branch to work with [defaults to master]

Details:

`knife cookbook site install` uses the git version control system in conjunction with the cookbook site to install community contributed cookbooks to your local cookbook
repository. Running `knife cookbook site install` does the following:

1. A new "pristine copy" branch is created in git for tracking the upstream;
2. All existing cookbooks are removed from the branch;
3. The cookbook is downloaded from the cookbook site in tarball form;
4. The downloaded cookbook is untarred, and its contents commited via git;
5. The pristine copy branch is merged into the master branch.

By installing cookbook with this process, you can locally modify the upstream cookbook in your master branch and let git maintain your changes as a separate patch. When an updated upstream version becomes available, you will be able to merge the upstream changes while maintaining

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

your local modifications.

Example:
The following example is how one would install the **getting-started** cookbook from the community site.

```
> knife cookbook site install getting-started
Installing getting-started to /Users/sdanna/opscodesupport/.chef/../cookbooks
Checking out the master branch.
Creating pristine copy branch chef-vendor-getting-started
Downloading getting-started from the cookbooks site at version 0.3.0 to
/Users/sdanna/opscodesupport/.chef/../cookbooks/getting-started.tar.gz
Cookbook saved: /Users/sdanna/opscodesupport/.chef/../cookbooks/getting-started.tar.gz
Removing pre-existing version.
Uncompressing getting-started version /Users/sdanna/opscodesupport/.chef/../cookbooks.
removing downloaded tarball
1 files updated, committing changes
Creating tag cookbook-site-imported-getting-started-0.3.0
Checking out the master branch.
Updating 4d44b5b..b4c32f2
Fast-forward
 cookbooks/getting-started/README.rdoc              |    4 +++
 cookbooks/getting-started/attributes/default.rb    |    1 +
 cookbooks/getting-started/metadata.json            |   29 ++++++++++++++++++++
 cookbooks/getting-started/metadata.rb              |    6 ++++
 cookbooks/getting-started/recipes/default.rb       |   23 +++++++++++++++
 .../templates/default/chef-getting-started.txt.erb |    5 +++
 6 files changed, 68 insertions(+), 0 deletions(-)
 create mode 100644 cookbooks/getting-started/README.rdoc
 create mode 100644 cookbooks/getting-started/attributes/default.rb
 create mode 100644 cookbooks/getting-started/metadata.json
 create mode 100644 cookbooks/getting-started/metadata.rb
 create mode 100644 cookbooks/getting-started/recipes/default.rb
 create mode 100644 cookbooks/getting-started/templates/default/chef-getting-started.txt.erb
Cookbook getting-started version 0.3.0 successfully installed
```

list

Show all cookbooks currently on the community site.

Usage:

```
knife cookbook site list (options)
```

Additional Options:

- `-w`, `--with-uri`:
  Show corresponding URIs

Example:

Output from the following example has been truncated:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
> knife cookbook site list
1password             homesick              rabbitmq
7-zip                 hostname              rabbitmq-management
AmazonEC2Tag          hosts                 rabbitmq_chef
R                     hosts-awareness       rackspaceknife
accounts              htop                  radiant
ack-grep              hudson                rails
activemq              icinga                rails_enterprise
ad                    id3lib                redis-package
ad-likewise           iftop                 redis2
ant                   iis                   redmine
[...snip..]
```

Search

Search for a given cookbook on the community cookbook site.

Usage:

```
knife cookbook site search QUERY (options)
```

Example:

```
> knife cookbook site search apache*
apache2:
  cookbook:              http://cookbooks.opscode.com/api/v1/cookbooks/apache2
  cookbook_description:  Installs and configures all aspects of apache2 using Debian style symlinks
with helper definitions
  cookbook_maintainer:   opscode
  cookbook_name:         apache2
instiki:
  cookbook:              http://cookbooks.opscode.com/api/v1/cookbooks/instiki
  cookbook_description:  Installs instiki, a Ruby on Rails wiki server under passenger+Apache2.
  cookbook_maintainer:   jtimberman
  cookbook_name:         instiki
kickstart:
  cookbook:              http://cookbooks.opscode.com/api/v1/cookbooks/kickstart
  cookbook_description:  Creates apache2 vhost and serves a kickstart file.
  cookbook_maintainer:   opscode
  cookbook_name:         kickstart
[...snip...]
```

share

See section below on Adding a Cookbook to the Community Site.

show

Show information about the given cookbook cookbook from the community site.

Usage:

```
knife cookbook site show COOKBOOK_NAME
```

Example:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
% knife cookbook site show haproxy
average_rating:
category:        Networking
created_at:      2009-10-25T23:51:07Z
description:     Installs and configures haproxy
external_url:
latest_version:  http://cookbooks.opscode.com/api/v1/cookbooks/haproxy/versions/1_0_3
maintainer:      opscode
name:            haproxy
updated_at:      2011-06-30T21:53:25Z
versions:
    http://cookbooks.opscode.com/api/v1/cookbooks/haproxy/versions/1_0_3
    http://cookbooks.opscode.com/api/v1/cookbooks/haproxy/versions/1_0_2
    http://cookbooks.opscode.com/api/v1/cookbooks/haproxy/versions/1_0_1
    http://cookbooks.opscode.com/api/v1/cookbooks/haproxy/versions/1_0_0
    http://cookbooks.opscode.com/api/v1/cookbooks/haproxy/versions/0_8_1
    http://cookbooks.opscode.com/api/v1/cookbooks/haproxy/versions/0_8_0
    http://cookbooks.opscode.com/api/v1/cookbooks/haproxy/versions/0_7_0
```

unshare

See section below on Adding a Cookbook to the Community Site.

Adding a Cookbook to the Community Site

There are two ways to add a cookbook to the community site, you can either use knife or upload the tarball through the website.

Uploading a cookbook using Knife

share

Use of Knife's `share` subcommand uploads the specified cookbook using the given category to the Opscode cookbooks site. This requires a login user and certificate for the Opscode Community site.

By default, knife will use the username and API key you've configured in your configuration file; otherwise you must explicitly set these values on the command line or use an alternate configuration file. You need to be the owner or maintainer of the cookbook if it already exists on the web site.

Usage:

For example:

unshare

Use of the `unshare` subcommand stops sharing the given cookbook on the Opscode cookbooks site.

Usage:

Uploading a cookbook using the web site

An alternate to using Knife is to upload the cookbook directly through the site.

Prepare the Cookbook

First, you should ensure that your cookbook's metadata.rb file is correct. You must include the following metadata:

- Name

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- Version
- License

We also strongly recommend that you fill out:

- Description
- Supported Platforms

Once you have done that, you need to bundle the cookbook into a tarball. New versions of the Chef Repository have a task called 'bundle_cookbook' which will do this for you. From the top of your Chef Repository:

The task will produce a tarball in the pkgs directory of your chef repository:

Add the cookbook to the Cookbook Site

To add a new cookbook to the Cookbook Site:

1. Visit http://cookbooks.opscode.com and log in.
2. Click on the 'Cookbooks' tab at the top.
3. Click the orange *Add a new cookbook* button.
4. On the "New cookbook" page, browse for your cookbook tarball, select a category, enter tags (separated by commas), and click *Create*.

Licensing Questions

Do I need to license my cookbooks with Apache 2.0, like Opscode uses for Chef?

A: No. You can use any license you wish for your cookbooks as long as you have the rights to share and publish them. You specify which license the cookbook is published under in the metadata of the cookbook itself.

Do I need to sign an Opscode Contributor License Agreement to post cookbooks?

A: No. The Opscode CLA is only required if you want to make contributions to Opscode's published cookbooks.

 Managing API Clients With Knife

Managing Environments With Knife 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Data Bags With Knife



Overview



Knife is a command-line tool that comes with Chef.

It is used by administrators to interact with the Chef Server API, the local Chef repository and and can be used to manage the arbitrary stores of globally available JSON data known as Data Bags.

Knife's `data bag` sub-command provides this capability.

- For more information about Knife, refer to the **Knife** documentation.
- See **Data Bags** for more information on Data Bags, and **Encrypted Data Bags** if the data stored in the data bags is sensitive and shouldn't be stored in plain text or viewable when checked into version control systems.

Managing Data Bags with Knife

Data bags can be managed with the knife command line client. Make sure you have correctly configured knife before trying the examples here.

Creating Data Bags and Items

To create a data bag with knife, use

```
knife data bag create BAG
```

For example, to create a data bag to store data about admins, run this command:

```
$ knife data bag create admins
Created data_bag[admins]
```

Now that the data bag is created, you can put data items in it with `knife data bag create BAG[ITEM]`. This command will create a skeleton of the JSON for the data bag and drop you into your `$EDITOR` to edit it.

To add an administrator to our admins data bag, we'd do the following:

```
$ knife data bag create admins charlie
```

Now, in your editor, you'll see this:

```
{
 "id": "charlie"
}
```

To follow along with the example, edit the data bag item as follows:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
{
 "id": "charlie",
 "uid": 1005,
 "gid":"ops",
 "shell":"/bin/zsh",
 "comment":"Crazy Charlie"
}
```

When you exit your editor, knife will save this data to the server.

Showing Data Bags and Items

To list all of the data bags on the server, use `knife data bag list`. If you created the admins data bag above, running the command will produce the following result:

```
$ knife data bag list
  admins
```

To view the contents of a data bag, use `knife data bag show BAG`, for example:

```
$ knife data bag show admins
  charlie
```

To view the contents of a data bag item, use `knife data bag show BAG ITEM`:

```
$ knife data bag show admins charlie
comment:  Crazy Charlie
gid:      ops
id:       charlie
shell:    /bin/zsh
uid:      1005
```

Create or Update Data Bag Items from JSON

To create or update a data bag item using a JSON file, use

```
knife data bag from file BAG_NAME /path/to/file
```

If file is a relative or absolute path to the file, that file will be used. Otherwise, the file parameter is treated as the base name of a data bag file in a Chef repository, and knife will search for the file in ./data_bags/bag_name/file. For example knife data bag from file users dan.json would attempt to load the file ./data_bags/users/dan.json. The JSON file in question should be a Hash that contains at least an "id" key. The "id" key will be the item name.

```
{
   "id": "item_name"
}
```

Other keys can be added to this file just as they can when using `knife data bag BAG_NAME ITEM_NAME create`.

Editing Data Bag Items

Data bags are edited via knife with `knife data bag edit BAG ITEM`. This works similarly to creating data bag items, loading your `$EDITOR` with the data from the server, and saving it back to the server when you exit the editor.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Deleting Data Bags and Items

To delete a data bag with knife, use `knife data bag delete BAG`. Individual data bag items are deleted with `knife data bag delete BAG ITEM`.

Deleting a Data Bag Item

```
$ knife data bag delete admins charlie
Do you really want to delete charlie? (Y/N) Y
Deleted data_bag_item[charlie]
```

Deleting a Data Bag

```
$ knife data bag delete admins
Do you really want to delete admins? (Y/N) Y
Deleted data_bag[admins]
```



Encrypted data bags and knife

The encrypted data bag features of the `knife data bag *` subcommands are activated by specifying either a `--secret` or `--secret-file` option.

- `--secret SECRET` is used to specify the key used for encryption and decryption on the command line.
- `--secret-file FILE` is used to specify the path to a file that contains the encryption/decryption key.

Create a new encrypted data bag item

Verify that the data bag has been created and encrypted

Decrypt an encrypted data bag item

Other knife encrypted data bag commands

When given a `--secret` or `--secret-file` option,

- `knife data bag from file` will **encrypt** the data bag item contained in the specified JSON file;
- `knife data bag edit` will **decrypt** an existing data bag item, allow you to edit it, and then encrypt it before saving it to Chef

Managing Cookbooks With Knife

Managing Environments With Knife

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Environments With Knife



Overview

Knife is a command-line tool that comes with Chef.

It is used by administrators to interact with the Chef Server API, the local Chef repository and and can be used to manage environments within one Chef setup (or one organization on Hosted Chef).

Knife's `environment` sub-command provides this capability.

- For more information about Knife, refer to the **Knife** documentation.
- See **Environments** for more information on Environments.

Knife commands all have the same form

```
knife sub-command [ARGUMENTS] (options)
```

create

Creates a new environment object on the Chef Server.

Usage:

```
knife environment create ENVIRONMENT (options)
```

Knife will open $EDITOR and allow you to edit the ENVIRONMENT description. Once you are done editing, knife will save the environment to the Chef Server.

Additional Options:

- `-d`, `--description DESCRIPTION`:
  The value of the description field.

delete

Deletes an environment on the Chef Server.

Usage:

```
knife environment delete ENVIRONMENT (options)
```

edit

Edit an existing environment using $EDITOR.

Usage:

```
knife environment edit ENVIRONMENT (options)
```

from file

Create or update an environment from a JSON or Ruby DSL description of the environment. See Environment for more details.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Usage:

```
knife environment from file FILE (options)
```

list

List all the of the environments saved on the Chef Server.

Usage:

```
knife environment list (options)
```

Additional Options:

- `-w`, `--with-uri`:
  Show the resource URI for each environment

Example:

```
% knife environment list
_default
dev
```

To list nodes within an environment, use search. Within the context of search, the chef_environment of a node is treated like an attribute. The following will result in a list of nodes within a the `dev` environment:

```
knife search node "chef_environment:dev"
```

see Searching within Environments below for an additional example.

show

Show the information for a given environment.

Usage:

```
> knife environment show ENVIRONMENT_NAME (options)
```

Examples:

```
% knife environment show dev
chef_type:            environment
cookbook_versions:
default_attributes:
description:
json_class:           Chef::Environment
name:                 dev
override_attributes:

\\
\\
\\
\\
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Managing Data Bags With Knife

Managing Nodes With Knife



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Nodes With Knife



Overview

Knife is a command-line tool that comes with Chef.

It is used by administrators to interact with the Chef Server API, the local Chef repository and can be used to create, edit, view, list, tag and delete nodes.

Knife's node sub-command provides the ability to manage nodes.

For more information about Knife, refer to the **Knife** documentation.
See **Nodes** for more information on nodes.

Knife commands all have the same form

```
knife sub-command [ARGUMENTS] (options)
```

Creating a node with Knife

Create a new node.

Note: **A node is created the first time you run chef-client, or when you use Knife Bootstrap commands.**
Use of this knife subcommand to create nodes isn't required if you have taken either of those steps.

```
knife node create NODE
```

Unless the --no-editor option is given, an empty node object will be created and displayed in your text editor. If the editor exits with a successful exit status, the node data will be posted to the Chef Server to create the node.

Through an EDITOR

1. set EDITOR environment variable, for example:

   ```
   export EDITOR=vi
   ```

2. create a new node, for example:

   ```
   knife node create foobar
   ```

3. enter the content of the node in JSON, for example:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
{
    "normal": {
    },
    "name": "foobar",
    "override": {
    },
    "default": {
    },
    "json_class": "Chef::Node",
    "automatic": {
    },
    "run_list": [
       "recipe[zsh]",
       "role[webserver]"
    ],
    "chef_type": "node"
}
```

4. Save it.
   If there isn't an existing node, you will see a warning message about that, which is normal. For example:

```
WARN: HTTP Request Returned 404 Not Found: Cannot load node foobar
Created (or updated) node[foobar]
```

Create a node from a JSON file

```
knife node from file FILE
```

Editing a node with Knife

Edit an existing node.

```
knife node edit NAME (options)
```

The JSON description of the node will be opened with EDITOR and saved back to the Chef Server once editing is complete. See Nodes for more information on the node description.

Through an EDITOR

1. set EDITOR environment variable, for example:

```
export EDITOR=vi
```

2. edit a node, for example:

```
knife node edit foobar
```

3. Update the JSON in the editor.
4. Save it.

Add to a node's run list.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife node run_list add NODE RUNLIST_ITEM (options)
```

This will add RUNLIST_ITEM to the run list of NODE.

Additional Options:

- `-a`, `--after ITEM`:
  Place the ENTRY in the run list after ITEM

Details:
Run list items may be either roles or recipes.

Recipe addition

When adding a recipe to a run list, there are several valid formats:

- Fully Qualified Format:
  `'recipe[COOKBOOK::RECIPE_NAME]'`, for example, `'recipe[chef::client]'`
- Cookbook Recipe Format:
  For brevity, the recipe part of the fully qualified format may be omitted, and recipes specified as `'COOKBOOK::RECIPE_NAME'`, e.g., `'chef-client::service`
- Default Recipe Format:
  When adding the default recipe of a cookbook to a run list, the recipe name may be omitted as well, e.g., `chef-client::default` may be written as just `chef-client`

Roles addition

Roles can be added to a run list using the following format:

```
'role[ROLENAME]'
```

Note that you need to put the role in quotes when using the knife node run_list add subcommand.

```
knife node run_list add node 'role[ROLE NAME]'
```

Example:

```
> knife node run_list add serverA getting-started
run_list:
    recipe[getting-started]
```

Remove an item from a node's run list.

```
knife node run_list remove NODE_NAME RUN_LIST_ITEM (options)
```

Viewing or Showing a node with Knife

Displays information about a node.

```
knife node show NODE_NAME (options)
```

Knife 0.10+ and above do not display all node attributes by default. The full list of attributes is very large and are thus suppressed from the default output. *You can see all of the node data by choosing JSON formatted output via the `-Fj` option.*

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Additional Options:

- `-a`, `--attribute ATTR`:
  Show only one attribute
- `-r`, `--run-list`:
  Show only the run list
- `-F`, `--format FORMAT`:
  Display the node in a different format.
- `-m`, `--medium`:
  Display more, but not all, of the node's data when using the default *summary* format

Examples

```
> knife node show brand-new-node
Node Name:   brand-new-node
Environment: _default
FQDN:
IP:
Run List:
Roles:
Recipes:
Platform:
```

```
% knife node show i-12345678 -a fqdn
fqdn: ip-10-251-75-20.ec2.internal
```

Note that when displaying a single attribute, the value shown is determined via attribute precedence. (See Attributes for more information.)

Examples:
Viewing all data in a node:

```
knife node show foobar
```

Viewing the run list of a node

```
knife node show foobar -r
```

Listing the nodes

List all nodes.

```
knife node list
```

Additional Options:

- `-w`, `--with-uri`:
  Show corresponding URIs

Examples:

```
% knife node list
  i-12345678
  rs-123456
```

Deleting a node with Knife

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Removing a node from being managed via the Chef Server.

bulk delete

Deletes nodes for which the name matches the regular expression on the Chef Server. The regular expression should be given in quotes, and should not be surrounded with forward slashes.

```
knife node bulk delete REGEX(options)
```

delete an individual node

Deletes a node from the Chef Server.

```
knife node delete NODE_NAME (options)
```

Additional Options:

- `-a`, `--all`:
  Display all node data in the editor. By default, default, override, and automatic attributes are not shown.

Note: Deleting the node does not delete any corresponding API client for the node.

Using TAGS

You can also "tag" nodes and then manage them with knife through those tags.

Creating a new node with a TAG

```
knife tag create NODE TAG ...
```

List a node's TAGs

```
knife tag list NODE
```

Deleting Node based on TAGs

```
knife tag delete NODE TAG ...
```

 Managing Environments With Knife

Managing Roles With Knife 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Roles With Knife



Overview

Knife is a command-line tool that comes with Chef.

It is used by administrators to interact with the Chef Server API, the local Chef repository and can be used to create, edit, view, list, and delete roles on nodes.

Knife commands all have the same form

```
knife
sub-command
[ARGUMENTS]
(options)
```

Knife's `role` subcommand provides this capability.

For more information about Knife, refer to the **Knife** documentation, and **Managing Nodes With Knife** for detail on the use of Knife to manage **Nodes**.
See **Roles** for more information on roles.

Adding a role to a node with Knife

Note that you need to put the role in quotes when using the knife node run_list add subcommand.

```
knife node run_list add node "role[ROLE NAME]"
```

Through an EDITOR

1. set EDITOR environment variable, for example:

```
export EDITOR=vi
```

2. create a new role, for example:

```
knife role create foobar
```

3. enter the data of the role in JSON, for example:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
{
    "name": "foobar",
    "default_attributes": {
    },
    "json_class": "Chef::Role",
    "run_list": [

    ],
    "description": "",
    "chef_type": "role",
    "override_attributes": {
    }
}
```

4. Save it. If there isn't an existing role with the same name, you will see a warning message about that, which is normal. For example:

```
WARN: HTTP Request Returned 404 Not Found: Cannot load role foobar
Created (or updated) role[foobar]
```

Editing a role with Knife

1. set EDITOR environment variable, for example:

```
export EDITOR=vi
```

2. edit a role, for example:

```
knife role edit foobar
```

3. Update the JSON in the editor.
4. Save it.

Viewing a role with Knife

```
knife role show foobar
```

Listing roles with Knife

```
knife role list
```

Deleting a role with Knife

```
knife role delete foobar
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 Managing Nodes With Knife

Knife Plugins 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Knife Plugins

Knife Plugins

Opscode Knife Plugins

See **Launch Cloud Instances with Knife** for detail on how to launch API-driven "Cloud" servers with Opscode Knife Plugins and automatically run Chef on the launched servers.



Installing Plugins

With Chef 0.10, knife will load commands from the following locations:

- The core set of knife commands shipped with Chef
- Commands in your home chef directory: ~/.chef/plugins/knife/
- Commands in a .chef/plugins/knife/ directory in your cookbook repo
- Commands located in a chef/knife/ directory in a Ruby Gem you have installed.

This allows you to conveniently keep a set of knife plugins that you reuse across projects in your home directory, share plugins with your team by including them in your cookbook repo, and share plugins with the whole Chef community by distributing them as Ruby gems.

To install a knife plugin from a file, such as a gist on github or email attachment, simply copy the file to your *~/.chef/plugins/knife/* directory. If you share a chef repo with your team and would like to share the plugin, copy the file to CHEF_REPO/.chef/plugins/knife/.

You can name the file anything you'd like as long as it's in this directory and has a `.rb` extension. You can also have multiple classes in one plugin file.

To use a plugin distributed as a rubygem, just install the gem.

Creating Plugins

Knife commands are Ruby classes that inherit from the Chef::Knife class. Knife will load your command regardless of how it is namespaced, so it is recommended that you put your commands in your own namespace. Knife commands are run by calling the `#run` method on an instance of your command class. For example, here is a "hello, world" knife command:

Chef Community Plugins

Review (or contribute to) a collection of **Community Plugins Found In The Wild**.

Knife, Chef, and Ohai plugins are available.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Notifying and Logging Resources with Knife Plugins

Nuclear Rooster did a nice blog post on using **Chef's flexible reporting handlers and knife plugins** for logging node updates, for easy debugging.

```
# You can namespace however you like, but it's a good idea to use your own
    # instead of the Chef::Knife namespace.
    module MyKnifePlugins
      # Make sure you subclass from Chef::Knife
      class HelloWorld < Chef::Knife

        # This method will be executed when you run this knife command.
        def run
          puts "Hello, World!"
        end

      end
    end
```

You should now be able to run this command using `knife hello world`. If you wanted the command to be `knife helloworld` instead, you would want to use `class Helloworld` with just the first letter capitalized.

You can also "override" current knife subcommands by using the class as the name of the subcommand, for example you could override the current functions of `knife cookbook upload` by using `class CookbookUpload < Chef::Knife`.

Command Line Options And Banners

If you run `knife --help` after creating the hello world plugin, you'll see this usage message:

```
** HELLO COMMANDS **
    Usage: /Users/ddeleo/opscode/chef/chef/bin/knife (options)
```

This isn't very helpful. Let's add a usage banner to match the other knife commands. You can add a banner by calling the `banner` method in the class body of your knife command class, like this:

```
class HelloWorld < Chef::Knife

      banner "knife hello world"

      def run
        puts "Hello, World!"
      end

    end
```

Now the help for this command will look like this:

```
** HELLO COMMANDS **
    knife hello world
```

Command line options are added to your command using the Mixlib::CLI library. Let's add a flag to give our plugin more emotion. As with the banner, we define new command line options by calling the `option` method inside the class body, like this:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
option :omg,
      :short => '-O',
      :long => '--omg',
      :boolean => true,
      :description => "I'm so excited!"
```

If we look at the help for our command, by running `knife hello world --help`, we can see that this option was added to the list:

```
knife hello world
        -s, --server-url URL             Chef Server URL
        ...snipped...
        -O, --omg                        I'm so excited!
        ...snipped...
        -h, --help                       Show this message
```

When knife runs our command, it will parse the options from the command line and make the settings available to us as a Hash that we can access using the `config` method. We can update the run method of our class to change its behavior based on the config flag like this:

```
# Update the +run+ method to use our config setting
    def run
      if config[:omg]
        # Oh yeah, we are pumped.
        puts "OMG HELLO WORLD!!!1!!11"
      else
        # meh
        puts "Hello, World!"
      end
    end
```

You can now run `knife hello world --omg` and you should see 'OMG HELLO WORLD!!!1!!11' in the output.

Command Line Arguments

Knife commands also take command line arguments that are not specified via an option flag (for example, `knife node show NODE`). You don't need to add any code to your knife command class to configure it to accept command line arguments; arguments are automatically made available via the `name_args` method. To illustrate, lets make our hello world command say hello to anyone:

```
banner "knife hello world WHO"

    def run
      # knife does not automatically check the number of arguments given:
      unless name_args.size == 1
        puts "You need to say hello to someone!"
        show_usage
        exit 1
      end

      # Access arguments via +name_args+
      who = name_args.first

      if config[:omg]
        puts "OMG HELLO #{who.upcase}!!!1!!11"
      else
        puts "Hello, #{who.capitalize}!"
      end
    end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Now that our command requires additional user input, we need to check the number of arguments and fail if the input doesn't make sense. Here we used the `show_usage` method to display the correct usage before exiting. Notice also that we updated the banner with the new usage for the command. With this update to our command, we have the following behaviors:

```
shell > knife hello world
    You need to say hello to someone!
    USAGE: knife hello world WHO
      # options snipped...

    shell > knife hello world chefs
    Hello, Chefs!

    shell > knife hello world chefs --omg
    OMG HELLO CHEFS!!!1!!11
```

User Interaction

The `ui` object provides methods for user interaction that ensure a consistent user experience across knife plugins. Use these methods in favor of manually handling user interaction whenever possible. The following table summarizes the most useful methods:

Method	Description
`ui.msg(message)`	Presents a message to the user
`ui.warn(message)`	Presents a warning to the user.
`ui.error(message)`	Presents an error to the user.
`ui.fatal(message)`	Presents a fatal error to the user.
`ui.output(data)`	Present a data structure to the user. This function will respect the output the user requested with the `-F` command-line option. The generic presenter used by default is suitable for most data that will be passed to it in a knife plugin; however, a custom presenter can be provided using `ui.use_presenter`
`ui.ask_question(question, opts={})`	Asks the user the question contained in the string `question`. If `:default => default_value` is passed as the second argument, `default_value` will be used if the user does not provide an answer. This respects Knife's `--default` command line option.
`ui.confirm(question)`	Ask a Y/N question. Immediately exit with status code 3 if user responds with N.
`ui.highline`	Provides direct access to the HighLine object used by many `ui` methods. See http://highline.rubyforge.org/doc/ for more information.

Using these classes we can improve the error messaging in our example:

```
unless name_args.size == 1
    ui.fatal "You need to say hello to someone!"
    show_usage
    exit 1
end
```

This ensures that our fatal error is shown in the same way as all other fatal errors in knife.

Handling Exceptions

Exceptions can be handled within your Knife plugin just as they can be handled in any Ruby program. However, in many cases the exception handling that Knife provides will be enough and ensure that your plugin's exception handling is consistent with the rest of Knife.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Handling Dependencies

You can declare all of your dependencies at the top of the file, but this will make knife load slower for all commands. It's recommended that you use knife's lazy loader within the plugin for any dependencies, so they are only used when you use the plugin command.

You can do this by adding this command below the `class` command for each plugin:

```
class HelloWorld < Chef::Knife
  deps do
    require 'chef/search/query'
  end

  [...SNIP...]
end
```

This would use the chef/server/query library, only whenever the `knife hello world` command is used.

Interacting With the Chef API using shef's DSL

Our example knife command is friendly, but it isn't very useful. In order to write useful knife commands, you will most likely want to interact with the Chef Server API in some way. There are two ways to do this. One way is to extend your knife command with shef's DSL for the API. This is an attractive way to go if you are looking to extend any existing `knife exec` scripts you already have. To do this, just require `chef/shef/ext` at the top of your file and then activate the Shef DSL by calling `Shef::Extensions.extend_context_object(self)` inside your `run` method. For example, we could write a knife command to search nodes like this:

```
module MyKnifePlugins
      class NodeSearch < Chef::Knife

        banner "node search QUERY"

        deps do
          require 'chef/shef/ext'
        end

        def run
          unless name_args.size == 1
            ui.fatal "You need to supply a query"
            show_usage
            exit 1
          end

          query = name_args.first

          Shef::Extensions.extend_context_object(self)
          ui.output(ui.format_for_display(nodes.find(query)))
        end

      end
    end
```

You can also interact with the Chef API by using the model classes directly. Consult the chef source code and existing knife commands for examples.

Using Search

You can also use search within a plugin, to return information about your infrastructure and run knife commands. To do this you will need to add these lines to the top, so Ruby knows it'll need additional libraries for search:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
require 'chef/search/query'
```

Then you will want to create a Chef::Search::Query object and assign it to a variable:

```
query_nodes = Chef::Search::Query.new
```

Afterwards you can use the search method on this object to search, in this example it is searching for nodes with the webserver role and then prints the name of each of the nodes it finds:

```
query = "role:webserver"

query_nodes.search('node', query) do |node_item|
  puts "Node Name: #{node_item.name}"
end
```

You can also then use this to edit nodes, in this example it searches for any nodes with the webserver role and changes the runlist to a role named apache2:

```
query = "role:webserver"

query_nodes.search('node', query) do |node_item|
  ui.msg "Changing the run_list to role[apache2] for #{node_item.name}"
  node_item.run_list("role[apache2]")
  node_item.save
  ui.msg "New run_list: #{node_item.run_list}"
end
```

This will overwrite any existing data on the attribute you edit. It's also possible to specify multiple items to add to the run_list:

```
node_item.run_list("role[apache2]", "recipe[mysql]")
```

You can also use the arguments you've sent with the plugin command to search. For instance in this example, if you used the command "knife envchange "web*" it would search for any nodes in roles beginning with "web" and change their environment to "web":

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
module MyKnifePlugins

  class Envchange < Chef::Knife
    banner "knife envchange ROLE"

    deps do
      require 'chef/search/query'
    end

    def run
      if name_args.size == 1
        role = name_args.first
      else
        ui.fatal "Please provide a role name to search for"
        exit 1
      end

      query = "role:#{role}"
      query_nodes = Chef::Search::Query.new

      query_nodes.search('node', query) do |node_item|
        ui.msg "Moving #{node_item.name} to the web environment"
        node_item.chef_environment("web")
        node_item.save
      end

    end
  end
```

Using Plugins within Other Plugins

You can access the functionality of other knife plugins from within your own plugin. For instance, suppose that you write a plugin that needs to run a custom bootstrap command on a node. Rather than executing a knife command using the ` (backtick) operator, you can directly run the plugin from inside your own plugin. To do this, you need to do three things:

1. `require` the necessary files to use the plugin's classes. The necessary files will change for each plugin. However, in most cases the library you want to include is in the form of `chef/knife/name_of_command`.

   ```
   require 'chef/knife/bootstrap'
   ```

2. Create a new object of the Class of the plugin you want to use.

   ```
   bootstrap = Chef::Knife::Bootstrap.new
   ```

3. Pass any arguments or options to the object. You do this by altering the object's config and name_arg variables.

   ```
   # Set some options. Currently, you will typically need
   # too look up the option name in the source file of the plugin
   bootstrap.config[:ssh_user] = "myuser"
   bootstrap.config[:distro] = "ubuntu10.04-gems"
   bootstrap.config[:use_sudo] = true
   # Set the arguments
   bootstrap.name_args = "some_host_name"
   ```

4. Call the object's run method.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
bootstrap.run
```

Existing Plugins

See **Community Plugins** for the Opscode Knife Plugins, in addition to plugins that have been developed and shared by the Chef Community. See **Launch Cloud Instances with Knife** for information on use of the Opscode Knife Plugins for launching cloud servers, including configuration and sub-command options.

Knife and Chef plugins can be distributed as gems. So an alternate way to find them is to just do a quick gem search.

```
gem search -r knife-
gem search -r chef-
```

Managing Roles With Knife

Community Plugins

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Community Plugins



Knife, Chef and Ohai Plugins Found in the Wild

Knife and Chef plugins can be distributed as gems. Those that are can be found with a gem search.

Found One, Write One, Share One

You can help the Chef community by expanding this page, and adding useful plugins that you created or "found in the wild".

Chef

- **chef-deploy**
  A gem that provides resources and providers for deploying ruby web apps from chef recipes
- **chef-gelf**
  Provides a Chef handler which can report run status, including any changes that were made, to a Graylog2 server.
- **chef-handler-twitter**
  A Chef handler that Tweets
- **chef-hatch-repo**
  Contains a Knife plugin and a Vagrant provisioner to launch a self-managed Chef server in VM or EC2
- **chef-irc-snitch**
  An exception handler for Opscode Chef runs (github gists & IRC)
- **chef-jenkins**
  Use Jenkins to drive continuous deployment and synchronization of your Chef Environments from a git repository
- **chef-rundeck**
  Provides a resource endpoint for RunDeck from a Chef Server
- **chef-solo-search**
  A cookbook library that adds data bag search powers to Chef Solo. (Requires Chef 10.04 or later)
- **chef-trac-hacks**
  This plugin is meant to fill the coordination gap between AWS and Chef
- **chef-vim**
  Vim plugin to make cookbook navigation quick and easy
- **chef-vpc-toolkit**
  The Chef VPC Toolkit is a set of Rake tasks that provide a framework to help automate the creation and configuration of “identical” VPC server groups in the cloud.
- **cluster_chef**
  Orchestrate complete clusters of machines. Powerful tools to let you develop clusters in VMs and deploy directly to the cloud

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- **cucumber-chef**
  Framework for behaviour-drive infrastructure development. Uses knife.rb for credentials with separate namespace in knife.rb
- **jclouds-chef**
  java and clojure components to use the Opscode chef rest api
- **metachef**
  Orchestrate distributed services: discover your redis server by saying `discover(:redis, :server)`, not hardcoding in values; discover all the ports to monitor in your organization by saying `components_with(:port)`, not living in danger.

Ohai

- **dell.rb**
  adds some useful Dell server info to ohai, like service tag, express service code, storage info, RAC info, etc. To use this plugin the server needs to have installed OMSA and SMBIOS applications.
- **ladvd.rb**
  adds ladvd (LLDP daemon for Unix) info, if it exists, to ohai
- **lxc_virtualization.rb**
  extends virtualization attributes to give host and guest information about linux lxc-containers.
- **network_addr.rb**
  extends network attribute with additional ipaddrtype_iface attributes to make semantically easier to retrieve the addresses. e.g., `node['network']['ipaddress_eth0']` vs `node['network']['interfaces']['eth0']['addresses'].guesswhichisfirst`.
- **network_ports.rb**
  extends the network attribute by adding which tcp / udp ports are bound on which interfaces
- **parse_host_plugin.rb**
  an oHai plugin that parses a host name, sets 3 top level attributes and 5 nested attributes
- **r.rb**
  collects some basic information about R
- **sysctl.rb**
  adds systems sysctl info to ohai
- **vserver.rb**
  extends virtualization attributes to give Linux VServer host and guest information including cgroup usage information.
- **xenserver.rb**
  extends virtualization attributes to load up xenserver host and guest info

Automate Ohai Plugin Use

The **Opscode Ohai Cookbook** creates a configured plugin path for distributing custom Ohai plugins, and reloads them via Ohai within the context of a Chef Client run during the compile phase.

Community Exception and Report Handlers

Chef includes Exception and Report Handlers that allow you to run code in response to a Chef run succeeding or failing. You can also write your own.

See **Community Based Handlers** for some written and shared by the chef open source community.

Knife

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



- **knife-audit**
  Shows you how many (and which) of your nodes have each of your cookbooks in their runlists
- **knife-batch**
  A wonderful little plugin for executing commands a la knife ssh, but doing it in groups of n with a sleep between execution iterations.
- **knife-brightbox**
  This plugin gives knife the ability to create, bootstrap, and manage instances in Brightbox Cloud .
- **knife-bulk-change-environment**
  A knife plugin to bulk move nodes from one environment to another
- **knife-canon**
  Compare command output across hosts to what you want it to be
- **knife-cloudstack** from *Clogeny*
  Provides the ability to create, bootstrap, and manage instances in Cloudstack Compute clouds.
- **knife-cloudstack** from *CloudStack* and *Edmunds*
  Gives Knife the ability to create, bootstrap and manage CloudStack instances, available as a gem.
- **knife-crawl**
  Displays the roles that are included recursively within a role and optionally all the roles that include it
- **knife-ec2-amis-ubuntu**
  RubyGem that provides an interface to retrieve a list of Canonical's released Ubuntu AMIs, and includes a knife plugin.
- **knife-env-diff**
  Diff the cookbook versions of two or more environments.
- **knife-esx**
  VMware ESX(i) 4/5 support for knife
- **knife-file**
  Utilities for manipulating files in a Chef code repository. (Right now, supports decrypting/encrypting data bag JSON files.)
- **knife-flip**
  An improved version of knife-set-environment with added functionality and failsafes
- **knife-github-cookbooks**
  Like `knife cookbook site install` *git vendor branch technique* but instead of only being able to access cookbooks in *community.opscode.com* you can create vendor branches automatically from any github cookbook.
- **knife-kvm**
  Linux KVM support for knife
- **knife-ohno**
  Show nodes in your environment that haven't checked into the platform for N hours
- **knife-OneHai**
  Get the last seen time of a single node using knife
- **knife-playground**
  Misc tools for Knife
- **knife-plugins**
  A set of plugins to make managing data bags in versions easier
- **knife-pocket**
  Save off a knife search to use later, either in knife ssh or for other commands you might need it for
- **knife-preflight**
  To let you easily check which nodes and roles use the cookbook you're about to change
- **knife_role_copy**
  Get data from one role, setup a new role with that data and a new name. This plugin will actively refuse to create a role with a name that already exists.
- **knife-rvc**
  Integrates a subset of Chef's knife functionality with RVC (Ruby vSphere Console).
- **knife-set-environment**
  A knife plugin to set node environment
- **knife-solo**
  To help with bootstrapping and running chef-solo. Supports search and data bags.
- **knife-spork**
  Simplifies the environments workflow to make it easier for teams to work on the same cookbooks and environments simultaneously
- **knife-ssh cheto**
  Some extra features to be used with ssh, for example, use ssh

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- **knife-gandi**
  This plugin gives knife the ability to create, bootstrap, and manage servers on the Gandi Hosting platform.
- **knife-gather**
  Collate multiline output from parallel knife ssh into one section per host

with proxy, knife ask you for your password, a new option that launches Cluster SSH.

- **knife-voxel**
  To provision Voxel.net cloud instances (physical as well as virtual) and then bootstrap them as chef clients.
- **knife-vsphere**
  VMware vSphere support for knife

Opscode Knife Plugins

- **knife-bluebox**
  This plugin gives knife the ability to create, bootstrap, and manage BlueBox servers.
- **knife-ec2**
  This plugin gives knife the ability to create, bootstrap, and manage instances in Amazon EC2.
- **knife-eucalyptus**
  This plugin gives knife the ability to create, bootstrap, and manage instances in Eucalyptus clouds.
- **knife-openstack**
  This plugin gives knife the ability to create, bootstrap, and manage instances in OpenStack Compute clouds.
- **knife-rackspace**
  This plugin gives knife the ability to create, bootstrap, and manage instances in the Rackspace Cloud.
- **knife-terremark**
  This plugin gives knife the ability to create, bootstrap, and manage Terremark servers.
- **knife-windows**
  Plugin that adds functionality to Chef's Knife CLI for configuring/interacting with nodes running Microsoft Windows

See Knife Plugins for installation details on the Opscode Knife Plugins, including information on interacting with the Chef API and creating your own plugin. See Launch Cloud Instances with Knife for information on their use for launching cloud servers, including configuration and sub-command options.

Knife Plugins

Management Console

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Management Console



The Management Console is a web interface for the Chef Server.

Many components can be managed through the console including users, nodes, roles, cookbooks, data bags, and API clients. Search can also be done on the console.

Most of these operations can also be completed by command line with Knife as well.

The Management Console will look and operate differently based upon the flavor of Chef you are using.

Hosted Chef

If you are a Hosted Chef customer, you can reach the management console at **http://manage.opscode.com**. The guide on the **Hosted Chef Management Console** explains this webui in more detail, including how to edit information in your user profile such as billing or account information.

Private Chef

If you are a Private Chef customer, please refer to the separately provided administration documentation.

Open Source Chef Server

If you are using your own Chef Server, you'll need to install and setup the server webui to use it. The guide on the **Open Source Chef Server Management Console** explains this further.

Community Plugins

Hosted Chef Management Console

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Hosted Chef Management Console



These guides provide examples and guidance on using the Host Chef Management Console.

You'll need to have a Hosted Chef Account, and have verified your email address in order to use the Hosted Chef Management Console.

Managing your Account, Users, and Permissions

- Setup an Opscode User and Organization
- Managing your Account and Billing Information
- Managing Users and your Private Key with the Hosted Chef Management Console
- Managing Groups with the Hosted Chef Management Console
- Managing Permissions with the Hosted Chef Management Console

Managing your Chef data

Most of these changes can also be made by command line with Knife if preferred.

- Using Search in the Hosted Chef Management Console
- Managing Organizations with the Hosted Chef Management Console
- Managing Clients with the Hosted Chef Management Console
- Managing Nodes with the Hosted Chef Management Console
- Managing Cookbooks with the Hosted Chef Management Console
- Managing Data Bags and Items with the Hosted Chef Management Console
- Managing Environments with the Hosted Chef Management Console
- Managing Roles with the Hosted Chef Management Console

 Management Console

Managing Clients with the Hosted Chef Management Console 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Clients with the Hosted Chef Management Console

The Management Console can be used to manage your clients

- Listing Clients
- Deleting Clients
- Creating a New Client
- Regenerating the Private Key for the Client API

Clients

On Hosted Chef, "API Clients" are entities that are similar to Users, except that clients are always scoped to your organization and only have access to the Hosted Chef API. API Clients are used by chef-client to authenticate when connecting to Hosted Chef. Typically, there is one client for each node under management.

- **To follow these directions, first log into the Hosted Chef Management Console with your existing username.** If you have not signed up for an account yet, follow the directions on the Set up Opscode User and Organization page.
- **If you're interested in editing account, node or other data in the Management Console instead, return to the Hosted Chef Management Console, and make the appropriate selection.**

When you first sign up, a "validator" client is created for your organization named `ORGANIZATION-validator.pem`. This "validator" client can be used to create new clients for your organization via the Hosted Chef API. *If the "validator" client is deleted for any reason, you will need to recreate it to authenticate new nodes.* Be careful not to let this happen, if it does, following directions on the Common Errors page will to recreate it.

Anyone in possession of a client's private key can do anything on your Hosted Chef account that the client is authorized to do, so be sure to protect you clients' private keys. These keys are typically created the first time a given chef-client connects to Hosted Chef, and stored locally on the node.

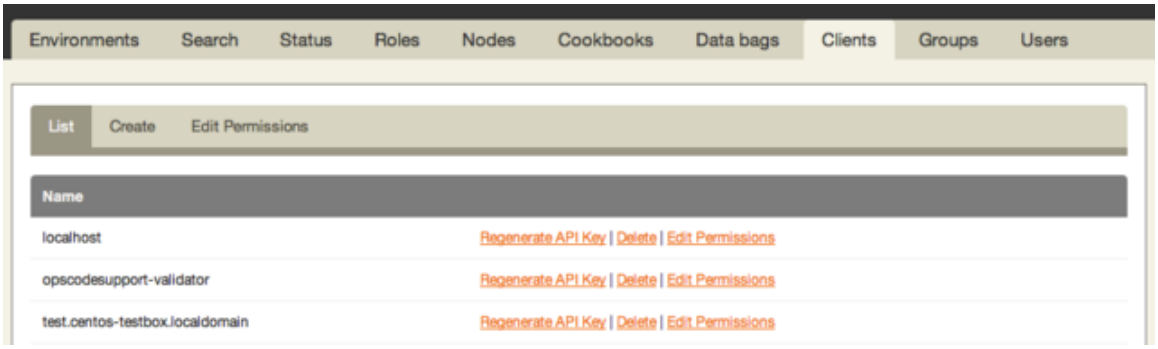
You can also manage clients with the command line tool Knife.

Listing Clients

In order to see a list of the clients in the organization:

1. Log in to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Select the *Clients* tab and you will see a list similar to this:



Note: You need list permissions on the global clients level to list the clients.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Deleting Clients

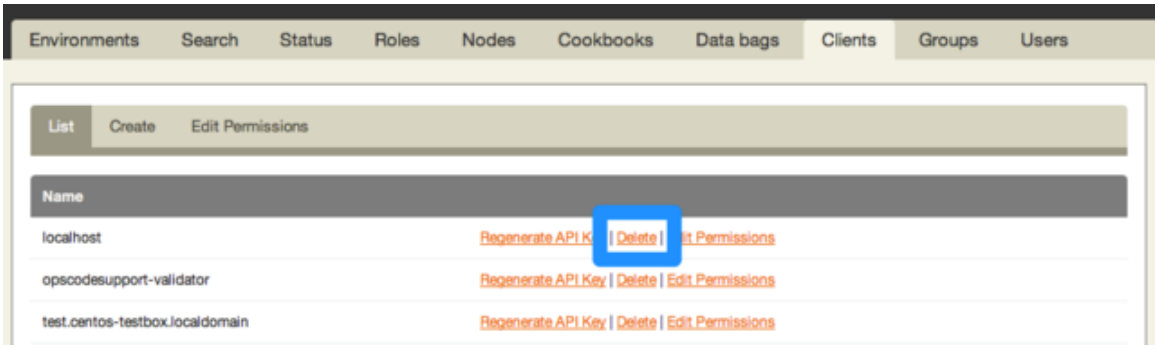
In order to delete a client:

1. Log in to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Select the *Clients* tab

3. Click on the *delete* link next to the client you want to delete.

Do not delete the validator client, which will be named ORGANIZATION-validator or you will be unable to authenticate new nodes until you recreate the validator key.



This will prevent that client from authenticating to Hosted Chef.

Note: You need delete permissions on the client to delete it.

Creating a New Client

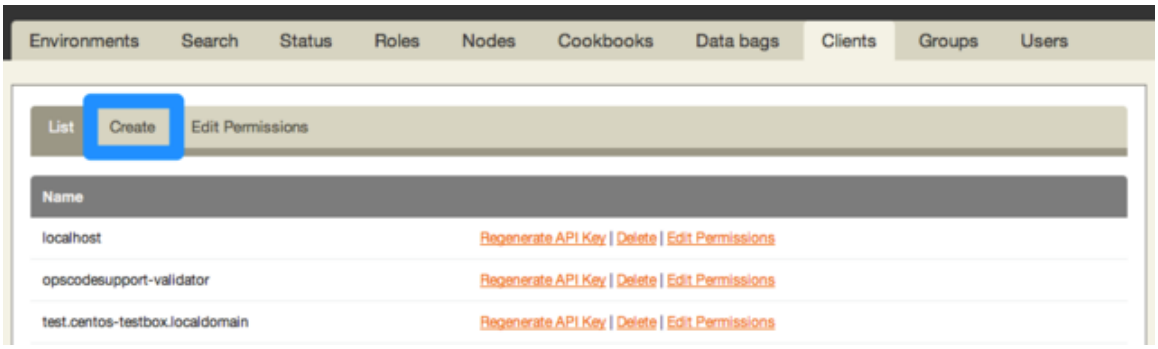
New clients are typically created by chef-client when it first connects to Hosted Chef, so there is no need in that case to manually create a new client.

If you need to manually create one, you can follow these steps:

1. Log in to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Select the *Clients* tab

3. Click on the *Create* sub-menu



4. Enter a name for the client and press *Create Client*. A link to your client's private key will be shown. Opscode does not store your private keys, so be sure to download and save the private key. You can generate a new private key if you lose it, however.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Note: You need create permissions on the global clients level to create a client.

Regenerating a private key for an Opscode API Client

Regenerating a client's private key invalidates the client's old private key and replaces it with a new one. Regenerating the key will be useful if you have lost the client's private key, if the client's private key has been exposed to someone you don't want to have access to your Hosted Chef account, or if you wish to regularly rotate keys for security purposes.

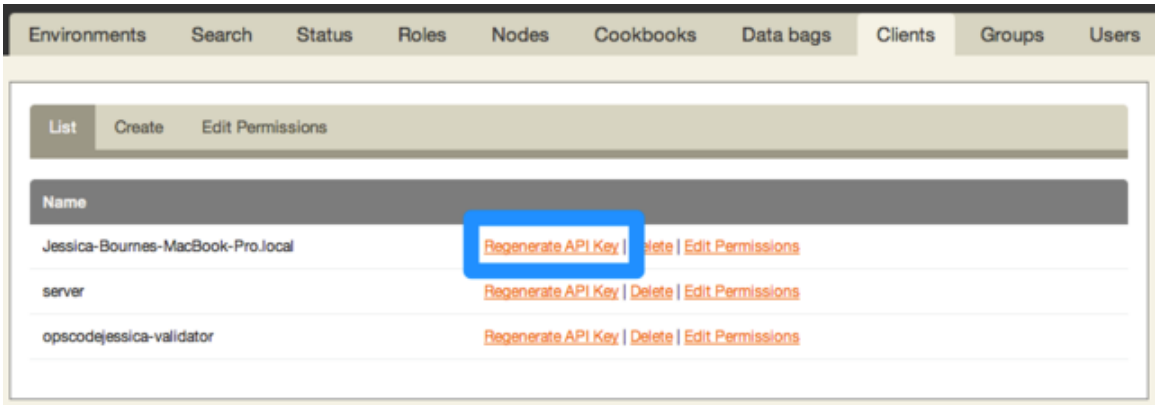
Regenerating the Key via the Hosted Chef Management Console

In order to regenerate a client's private key:

1. Log in to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Select the *Clients* tab

3. Click on the *regenerate private key* link next to the client whose key you want to regenerate:



Note: You need update permissions on the client to recreate this key.

Your new private key will start downloading. Save this key! There is no way for Opscode to recover the key if you lose it, though you can generate a new key again.

Regenerating the Key with Knife

To regenerate a client's key with knife, use `knife client reregister CLIENT`. This is explained further in the API Clients article.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License


Hosted Chef Management Console

Managing Cookbooks with the Hosted Chef Management Console


 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Cookbooks with the Hosted Chef Management Console

Using the Management Console to Manage your Cookbooks provides for:

- Listing Cookbooks
- Viewing Cookbook Contents

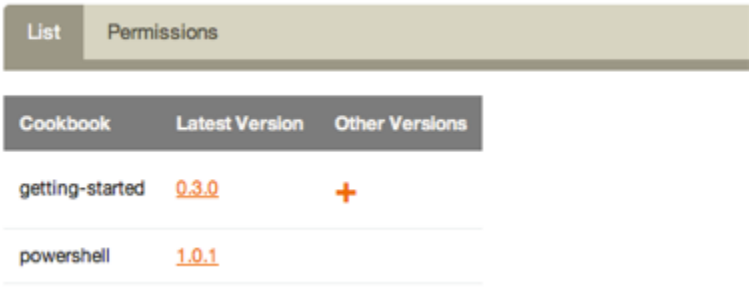
Listing Cookbooks in the Organization

- **To follow these directions, first log into the Hosted Chef Management Console with your existing username.** If you have not signed up for an account yet, follow the directions on the Set up Opscode User and Organization page.
- **If you're interested in editing account, nodes or other data in the Management Console instead, return to Hosted Chef Management Console, and make the appropriate selection.**

In order to list cookbooks in the organization:

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click *Cookbooks* on the main navigation menu. This will display the list of cookbooks that have been uploaded to the Hosted Chef Server:



Note: You need list permissions on the global cookbooks level in order to see the list of cookbooks.

Viewing the Contents of a Specific Cookbook

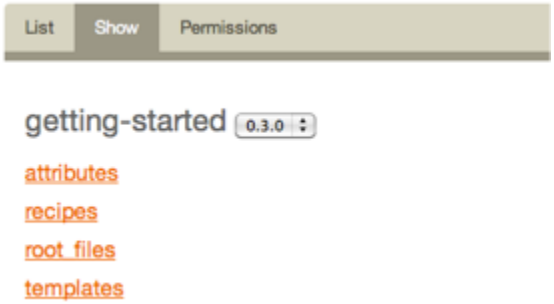
To view the contents of a specific cookbook:

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click *Cookbooks* on the main navigation menu.

3. Click the version of the cookbook you want to view. This may take longer for very large cookbooks.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



4. Click on *Library Files*, *Attribute Files*, *Definition Files*, *Recipe Files*, or *Template Files* to view the contents of the cookbook. If you do not see one of these links, the cookbook does not have files of that type.

Note: You need read permissions on the specific cookbook in order to view the details of the cookbook.

Managing Clients with the Hosted Chef Management Console

Managing Data Bags and Items with the Hosted Chef Management Console

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Data Bags and Items with the Hosted Chef Management Console

The Hosted Chef Management Console can be used to manage your data bags:

- Listing Data Bags
- Create a Data Bag
- Viewing or Editing a Data Bag
- Deleting a Data Bag

And manage data bag items:

- Creating Data Bag Items
- Editing Data Bag Items
- Viewing contents of Data Bag Items
- Deleting Data Bag Items

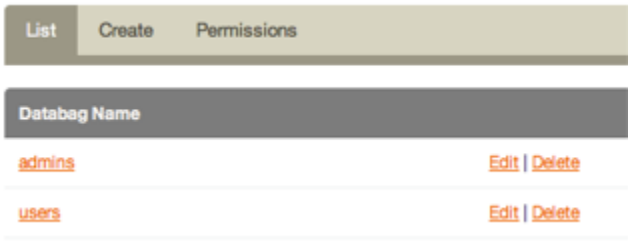
Listing Data Bags in the Organization

- **To follow these directions, first log into the Hosted Chef Management Console with your existing username.** If you have not signed up for an account yet, follow the directions on the Set up Opscode User and Organization page.
- **If you're interested in editing account, nodes or other data in the Management Console instead, return to Hosted Chef Management Console, and make the appropriate selection.**

To list data bags in your organization:

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click *Data bags* on the main navigation menu.



Note: You need the list permission on the global data bag level to see the list of data bags.

Creating a Data Bag

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click *Databags* on the main navigation menu.

3. Click *Create* on the sub-navigation bar. Enter a name for the data bag and click on the *Create data bag* button.

Note: You need the create permission on the global data bag level to create new data bags.

Viewing or Editing a Specific Data Bag in the Organization

To view or edit a specific data bag:

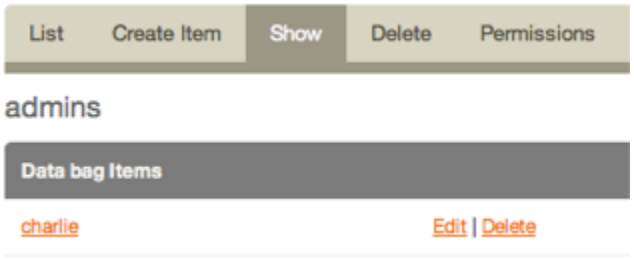
1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

2. Click *Databags* on the main navigation menu.

3. Click on the specific data bag you want to view or edit.

4. On the following page, you can create, edit, and delete items in the data bag. For details, refer to the sections below.



Deleting a Data Bag

To delete a data bag:

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click *Databags* on the main navigation menu.

3. Click the *Delete* link next to the name of the data bag you want to delete, and click *OK* on the confirmation message box. Or, click the link of the data bag to view its details, and click *Delete* on the sub navigation menu.

Note: You need delete permissions on a data bag to delete it. If you don't have this permission, you will not see a *Delete* link or tab.

Creating Data Bag Items

To create a data bag item in a specific data bag:

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click *Databags* on the main navigation menu.

3. Click on the specific data bag you want to view or edit.

4. Click *Create Item* on the sub navigation bar. You will see a page with a JSON editor.

5. Click the *Source* tab in the JSON editor to edit the data bag item directly, or use the tree view on the left and the *Editor* tab to add and edit keys and values. Further directions on these two methods are below. You should not delete the "id" key as this contains the name of the data bag item.

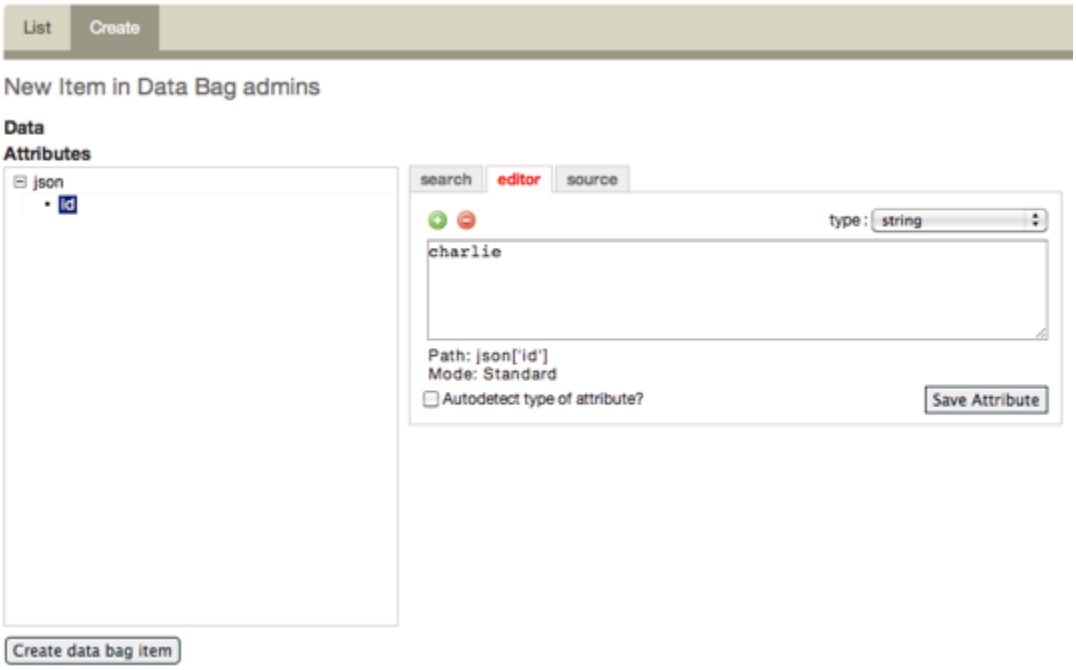
Note: You need create permissions on the parent data bag in order to create a new item within it.

Using the *Editor* tab

1. Expand the Data Attributes tree on the left and click "id".

2. Enter a value for the name of the data bag item, and then click *Save Attribute*. This will be the name of the data bag item when you are completed.

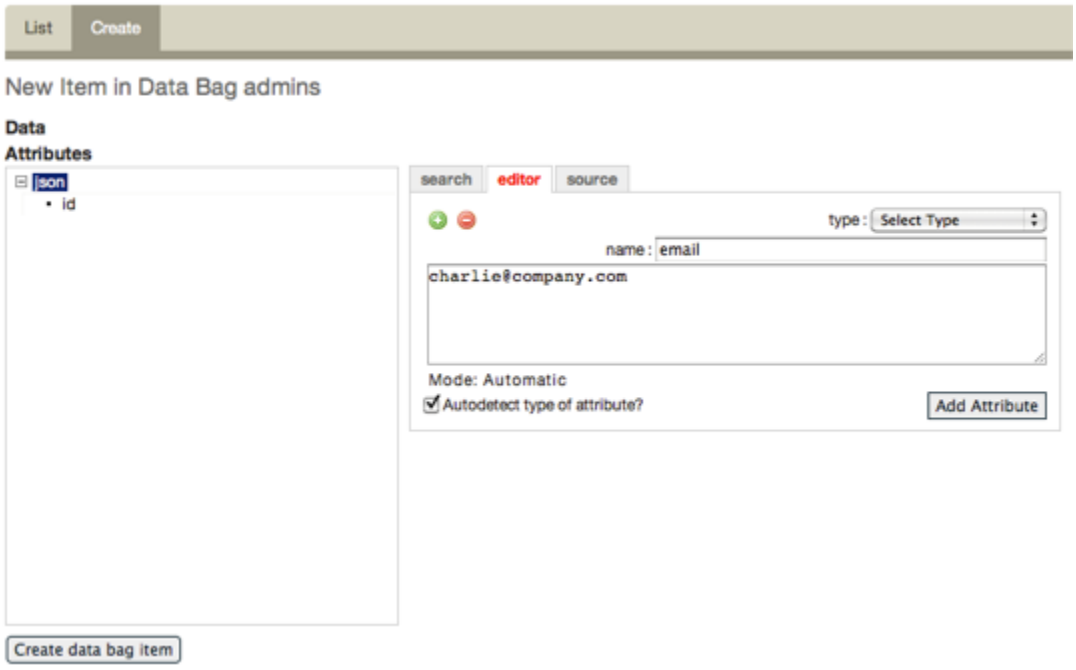
This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



3. Now you can add contents to the data bag. Click on "json" from the Data Attributes tree on the left, and click the *Editor* tab on the right.

4. Click the green add button under the *Editor* tab:  .

5. Enter a name as the key and a body as the value and click *Add Attribute*. The following figures show the Data Attribute tree and the JSON editor.



6. Add more contents to the data bag by following steps 3 through 5 as many times as needed. Click *Create Databag Item* to save the data bag item.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

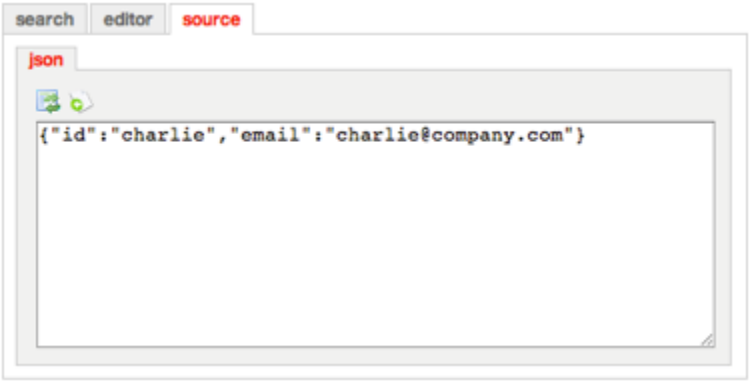
Using the *Source* tab

1. Enter the JSON body of your data bag item in the editor, and click the *Load JSON from source* button: 

Here's an example with just the name set:



You can also add contents to the data bag item by editing the json with further key/value pairs. Here's an example:



2. Click *Create Databag Item* button to save the data bag item when you are done.

Editing Data Bag Items

To edit a data bag item in a specific data bag:

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click *Databags* on the main navigation menu.

3. Click the *Edit* link next to the name of the data bag item you want to edit.

4. Click *Edit Databag Item* on the sub navigation bar. You will see a page with a JSON editor.

5. Click the *Source* tab in the JSON editor to edit the data bag item directly, or use the tree view on the left and the *Editor* tab to add and edit keys and values. You should not delete the "id" key as this contains the name of the data bag item. More information on using the *Editor* and *Source* ta bs can be found in the sections above.

Note: You need update permissions on the parent data bag in order to edit a data bag item within it.

Viewing contents of Data Bag Items

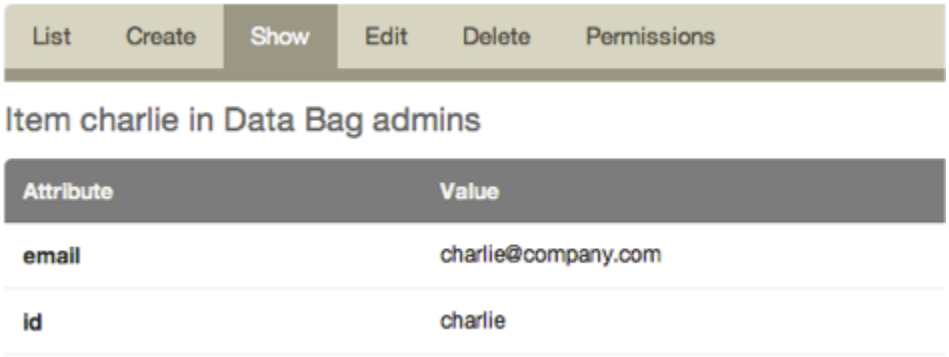
1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

2. Click *Databags* on the main navigation menu.

3. Click on the specific data bag that contains the data bag item you want to view.

4. Click on the specific data bag item you want to view. This will show a page similar to this:



Deleting Data Bag Items

To delete a data bag item in a specific data bag:

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click *Databags* on the main navigation menu.

2. Click the *Delete* link next to the name of the data bag item you want to delete, and click *OK* on the confirmation message box. Or, click the link of the data bag item to view its details, and click *Delete* on the sub navigation menu.

Note: If you do not see a *Delete* link or tab, you do not have delete permissions on the parent data bag.

Managing Cookbooks with the Hosted Chef Management Console

Managing Environments with the Hosted Chef Management Console

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Environments with the Hosted Chef Management Console

The Management Console can be used to manage environments.

- Listing Environments
- Viewing Environments
- Creating Environments
- Editing Environments
- Modifying Environment Attributes
- Adding a Node to an Environment
- Deleting Environments

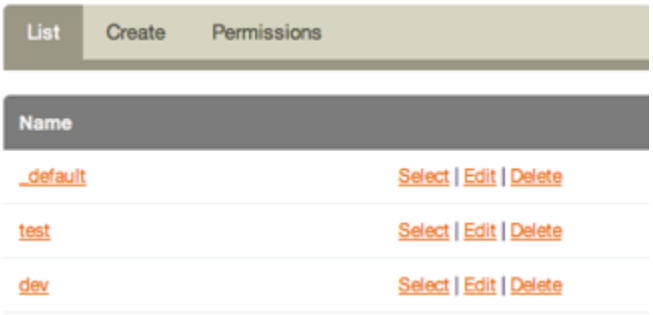
Listing Environments in the Organization

To list environments in your organization:

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

- **To follow these directions, first log into the Hosted Chef Management Console with your existing username.** If you have not signed up for an account yet, follow the directions on the Set up Opscode User and Organization page.
- **If you're interested in editing account, nodes or other data in the Management Console instead, return to Hosted Chef Management Console, and make the appropriate selection.**

2. Click *Environments* on the main navigation menu.



Note: You need list permissions on the global environments level to see the list of environments.

Viewing an Environment

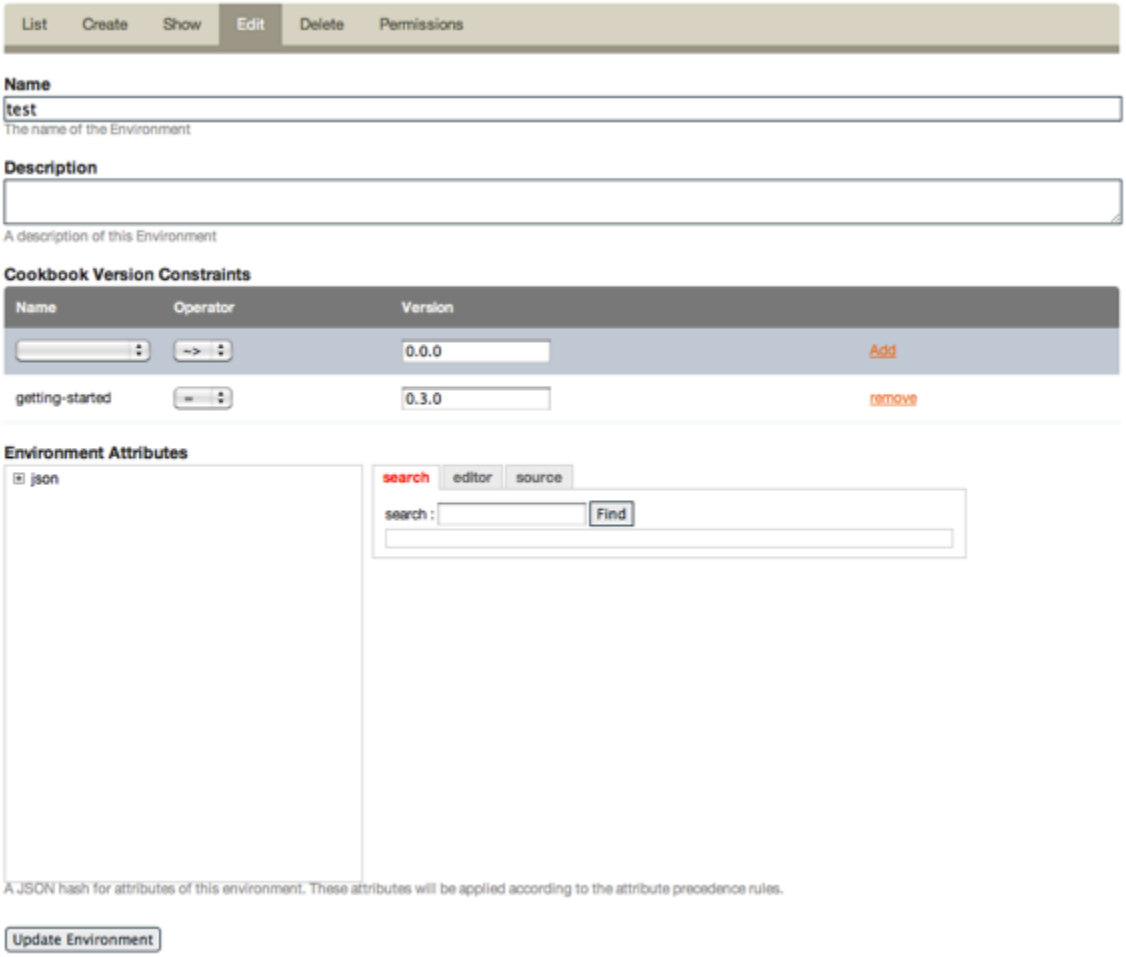
In order to view a specific environment:

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

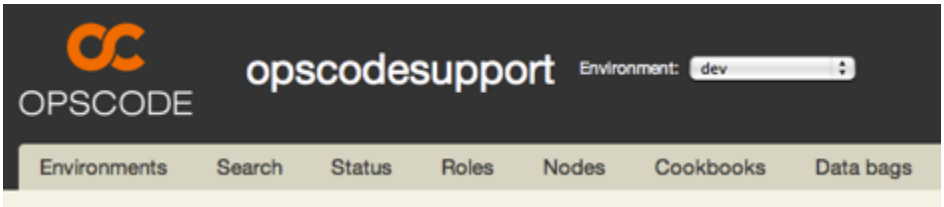
2. Click *Environments* on the main navigation menu.

3. Click on the name of the specific environment you want to view on the environments list page. This will show you data such as the Cookbook Version Constraints and Attributes for the Environment:

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



In order to see other data in the environment, such as the nodes that are members of it, you can select the environment from the drop down box at the top of the page and then go to the relevant tab in the Management Console:



For instance, to view nodes in the environment you could click on the *Node* tab after selecting environment from the drop down. To view data for all environments select *None* from the drop down. This can also get set by clicking on the *Select* link next to an environment on the *Environments* tab.

Note: You need read permissions on the environment to view it.

Creating an Environment

The environment creation page allows you to create a new environment and set its cookbook version constraints and attributes.

In order to create an environment:

1. Log on to the Hosted Chef Management Console and navigate to the *Environments* tab.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

2. Click *Create* on the sub navigation bar.

3. Give the environment a name and an optional description

4. If necessary, add any cookbook version constraints by selecting the cookbook name, operator, and entering the version number in the format of x.y or x.y.z, and then click *add*.

5. If necessary, add environment attributes in the JSON attribute editor. More information on using the JSON editor can be found in the next section.

6. Click *Create Environment*.

Note: You need create permissions on the global environments level to create an environment.

Editing an Environment

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click *Environments* on the main navigation menu.

3. Click on the *edit* link next to the name of the environment you want to edit.

4. On this page you can add any cookbook version constraints by selecting the cookbook name, operator, and entering the version number in the format of x.y or x.y.z, and then click *add*. You can also edit the Attributes, which is described further below.

5. Click the *Create Environment* or *Update Environment* button when you are done to save the environment.

Note: You need update permissions on the environment to edit it.

Modifying Environment Default and Override Attributes

You can edit the override Attributes of an environment, to override the values of an included recipe or role, and you can edit the default attributes of an environment to specify attributes that do not exist in included recipes and roles. You can use the *Editor* or *Source* tabs on this page to edit the attributes. With the *Editor* tab you can edit the attributes by selecting the location on the tree and adding the details. You can use the *Source* t ab to paste in JSON directly.

Note: You need update permissions on the environment to edit the attributes.

Using the Editor tab

1. Click on the plus sign next to "json" and select "default" or "override" from the tree on the left, depending on the type of attribute you want to create. If you're unsure, you should select "default" unless you are trying to override a default attribute that is already set in the node or in a recipe.

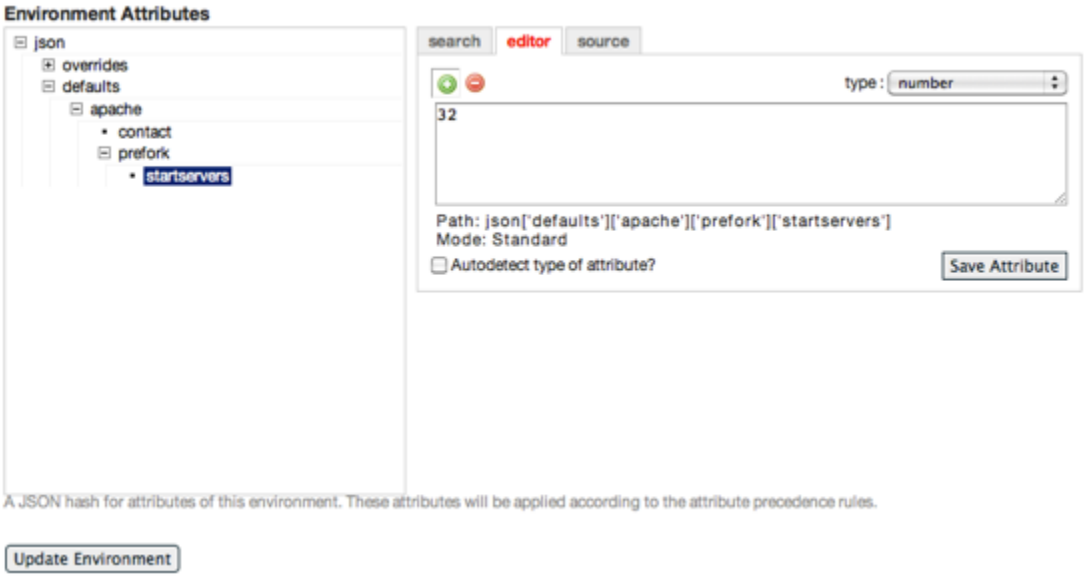
2. Click on the *Editor* tab on the right.

3. Push the green add button under the *Editor* tab:  .

4. Select a type of attribute from the drop down box. If you are unsure, you most likely want "string" if the value is text, or "number" if the value is a number.

5. Enter a name as the key and a body as the value and click *Add Attribute*. The following figure shows the Default and Override Attributes tree and the JSON editor.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



6. Click the *Create Environment* or *Update Environment* button to save the environment.

Using the Source tab

1. Enter the JSON body in the editor, and click the *Load JSON from source* button: 

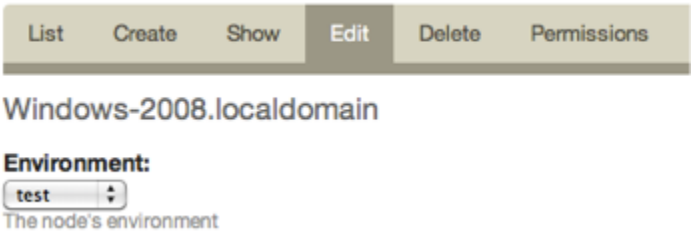


2. Click the *Create Environment* or *Update Environment* button to save the environment.

Moving a Node to an Environment

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.
2. Click *Nodes* on the main navigation menu.
3. Click on the *edit* link next to the node you want to move to the environment.
4. Select the environment from the drop down box under *Environment*.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



5. Click on the *Save Node* button.

Note: You need update permissions on the node to edit it.

Deleting an Environment

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click *Environments* on the main navigation menu.

3. Click on the *delete* link next to the environment you want to delete.

Note: You need delete permissions on the environment to delete it.

Managing Data Bags and Items with the Hosted Chef Management Console

Managing Groups with the Hosted Chef Management Console

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Groups with the Hosted Chef Management Console

The Management Console can be used to Manage Groups

- Listing Groups
- Viewing a Group
- Editing a Group
- Adding or Removing a User from a Group
- Deleting a Group

Groups

- **To follow these directions, first log into the Hosted Chef Management Console with your existing username.** If you have not signed up for an account yet, follow the directions on the Set up Opscode User and Organization page.
- **If you're interested in editing account, nodes or other data in the Management Console instead, return to Hosted Chef Management Console, and make the appropriate selection.**

Users, clients, and groups can be organized into groups for easier management of permissions. When a Hosted Chef organization is created, there are three default groups within the org: admins, users, and clients. By default, all associated users get added to the users group, all existing and new clients get added to the clients group; and the user who created the organization account belongs to the admins group.

Users with appropriate permissions can create new groups, edit existing groups, view current groups, and change the permissions bestowed by membership in the groups.

Listing Groups in Your Organization

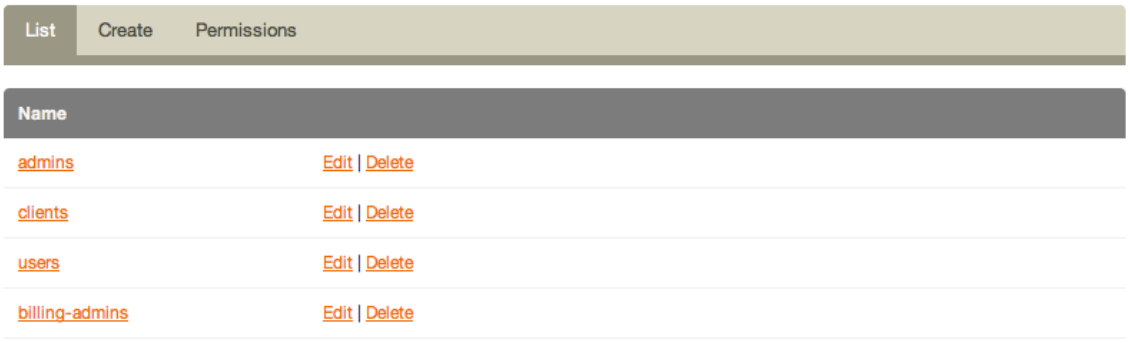
Note: In order to list groups in your organization, you need the list permission on the global level. Refer to the Managing Permissions article for more information about how to retrieve and update permissions.

In order to list groups in your organization:

1. Log in to the Hosted Chef Management Consoleand select the appropriate organization if the user is associated to more than one.

2. Click *Groups* on the main navigation menu.

This will direct you to this page where it lists out the groups in your organization:



Viewing a Specific Group

Note: In order to view the details of a specific group, you need read permissions on that group. Refer to the Managing Permissions article for

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

more information about how to retrieve and update permissions.

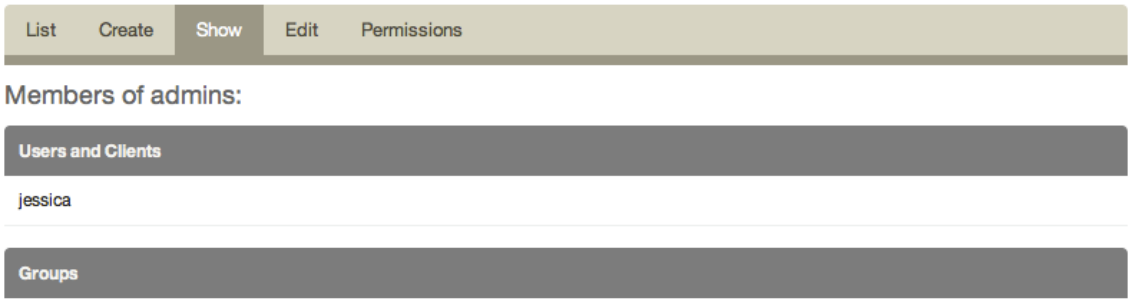
In order to view a specific group in your organization:

1. Log in to the Hosted Chef Management Console and select the appropriate organization if the user is associated to more than one.

2. Click *Groups* on the main navigation menu.

3. Click the name of the specific group you want to view.

This will display the list of users and clients within the group, as well as the groups it is a part of:



Editing a Specific Group

Note: In order to edit a specific group, you need update permissions on that group. Refer to the Managing Permissions article for more information about how to retrieve and update permissions.

In order to edit a specific group in your organization:

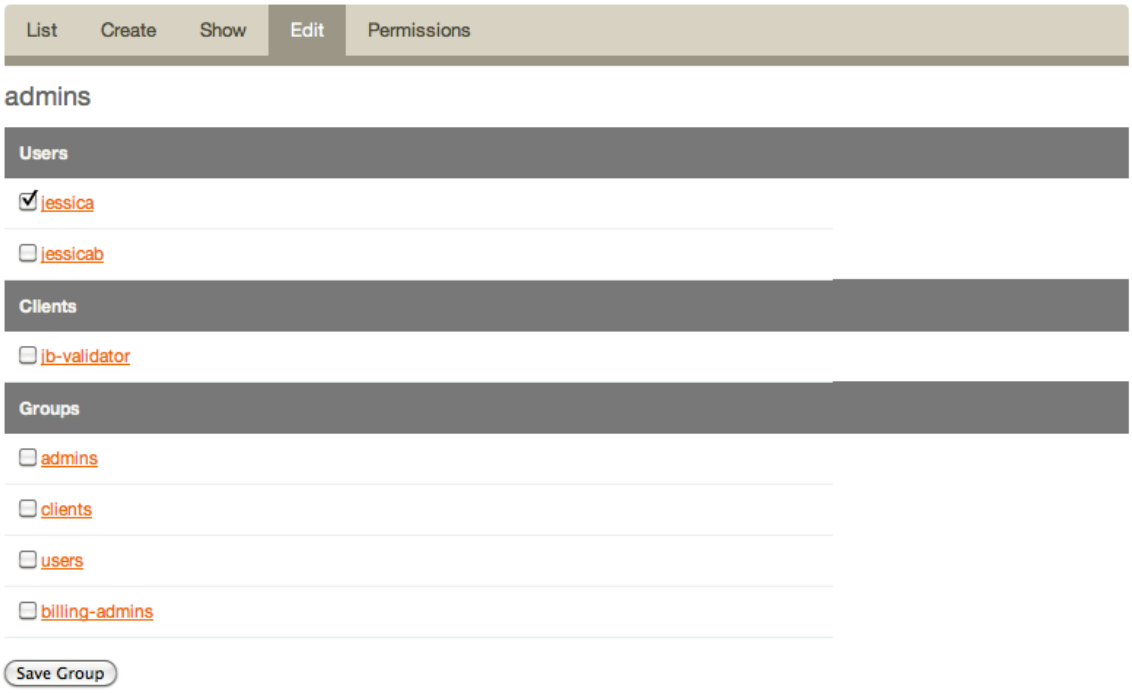
1. Log in to the Hosted Chef Management Console and select the appropriate organization if the user is associated to more than one.

2. Click *Groups* on the main navigation menu.

3. Click the *Edit* link next to the group you want to edit. Or, click the name of the specific group you want to edit, and in the group details page, click the *Edit* tab.

This will direct you to a page where you can select which users, clients, and groups to include in this group:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Note: If you don't see an *Edit* link or tab, you don't have update permissions on that group.

4. Select the users and groups you want, and click *Save Group*

Adding or Removing a User from a Group

In order to add or remove a user from a group:

1. Log in to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Add a user to the organization by following the Managing Organizations page.

3. Click *Groups* on the main navigation menu.

4. Click on *Edit* next to the group you want to add the user to.

5. Click on the checkbox next to the user's name, and push the *Save Group* button. You can remove a user from the group by unselecting the checkbox and clicking on the *Save Group* button.

Deleting a Specific Group

In order to delete a specific group in your organization:

1. Log in to the Hosted Chef Management Console and select the appropriate organization if the user is associated to more than one.

2. Click *Groups* on the main navigation menu.

3. Click the *Delete* link next to the group you want to delete. Or, click the link of the specific group you want to delete, and in the group details page, click the *Delete* tab.

4. Click *OK* on the confirmation message box.

Note: If you don't see a *Delete* link or tab, you don't have delete permissions on that group. You cannot delete admins, users, and clients groups.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 Managing Environments with the Hosted Chef Management Console

Managing Nodes with the Hosted Chef Management Console 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Nodes with the Hosted Chef Management Console

Manage your nodes with the Management Console

- Listing Nodes
- Viewing a Node
- Creating a node
- Editing a node
- Moving a Node to an Environment
- Modifying Node Attributes
- Deleting a Node
- Checking Node Status

You can also manage nodes with the command line tool Knife.

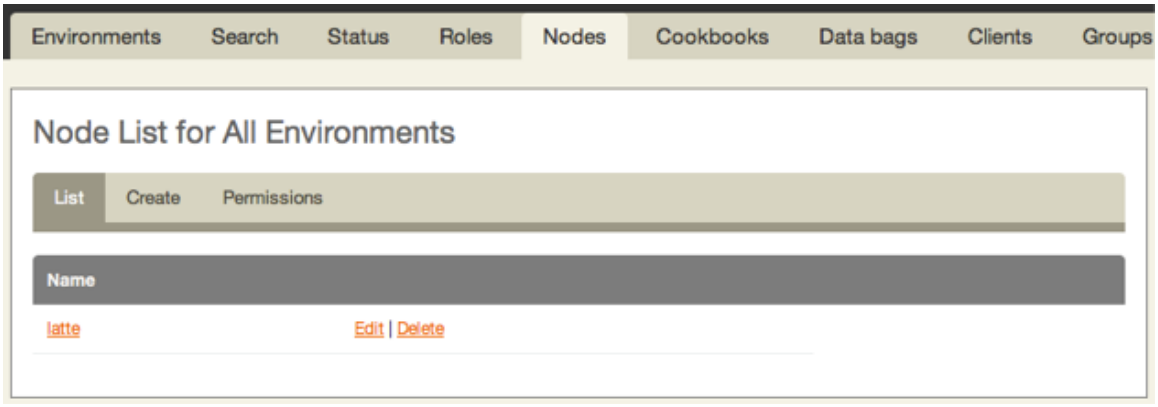
- **To follow these directions, first log into the Hosted Chef Management Console with your existing username.** If you have not signed up for an account yet, follow the directions on the Set up Opscode User and Organization page.
- **If you're interested in editing account, nodes or other data in the Management Console instead, return to Hosted Chef Management Console, and make the appropriate selection.**

Listing Nodes in the Organization

In order to list nodes in your organization:

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click *Nodes* on the main navigation menu.



Note: If a specific environment is selected on the top of the page, then only the nodes in that environment will be shown. You also need list permissions on the global nodes level to list the nodes.

Viewing a Node

Viewing a node shows the recipes and roles in the node's run list, as well as the node's attributes.

In order to view a specific node:

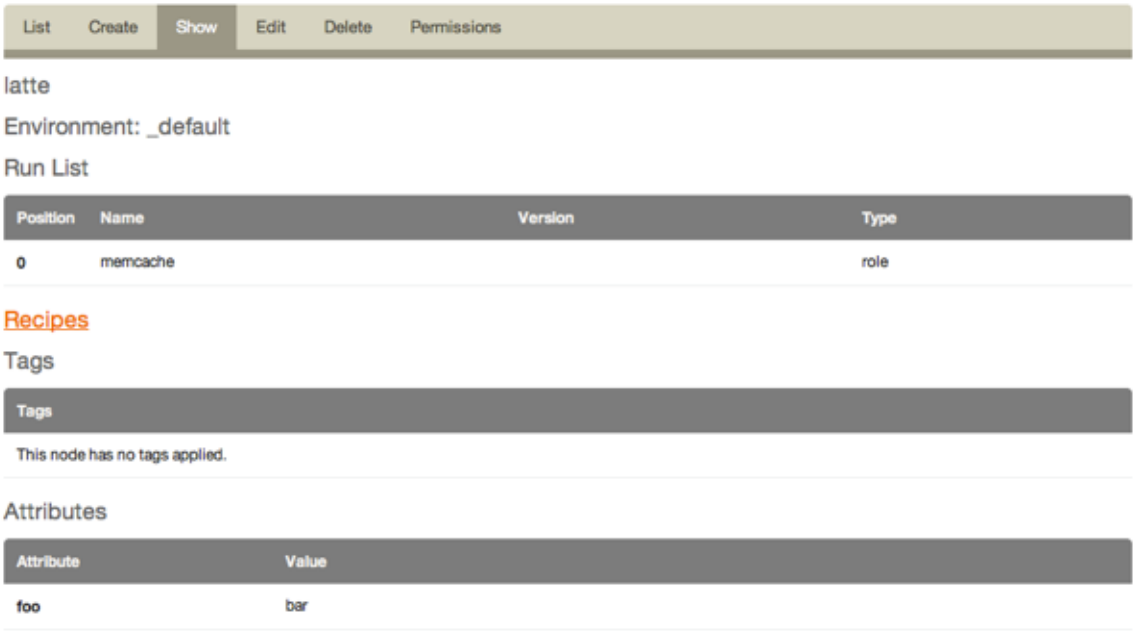
1. Login to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click on the *Nodes* tab.

3. Click on the name of the node you want to view.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

On this page you can see the roles and recipes in the node's run list, as well as an expandable tree view of the JSON attributes on the node.



Note: You need read permissions on the node to view it.

Creating a Node

The node create page lists all available recipes and roles in the organization, and allows you to include these in the run list of the node you are creating. There is also a JSON editor for editing, adding and removing the node's attributes.

A node is also created the first time you run chef-client, or when you use Knife Bootstrap commands.

In order to create a node through the Management Console:

1. Login to the Hosted Chef Management Console and select an organization.

2. Click on the *Nodes* tab.

3. Click *Create* on the sub navigation bar.



4. Give the node a name and select an Environment for the node.

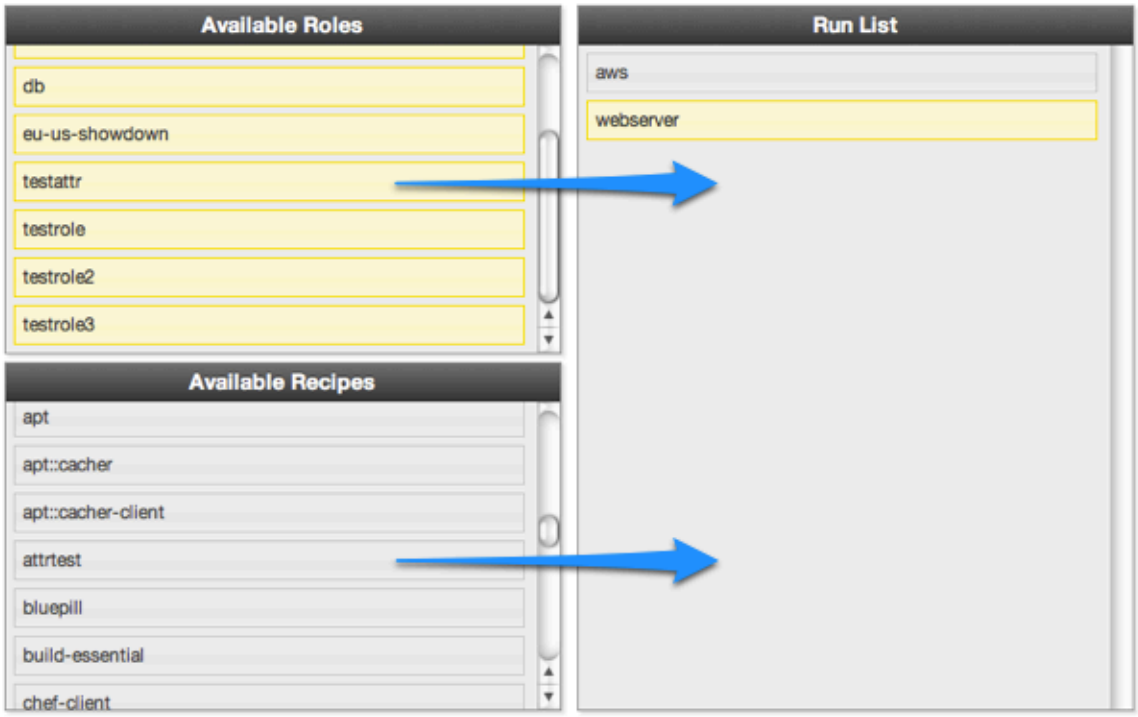
5. Edit the node, further information is in the next section. Click *Create Node* to save the new node when you are done.

Note: You need create permissions on the global nodes level to create a node.

Editing a Node

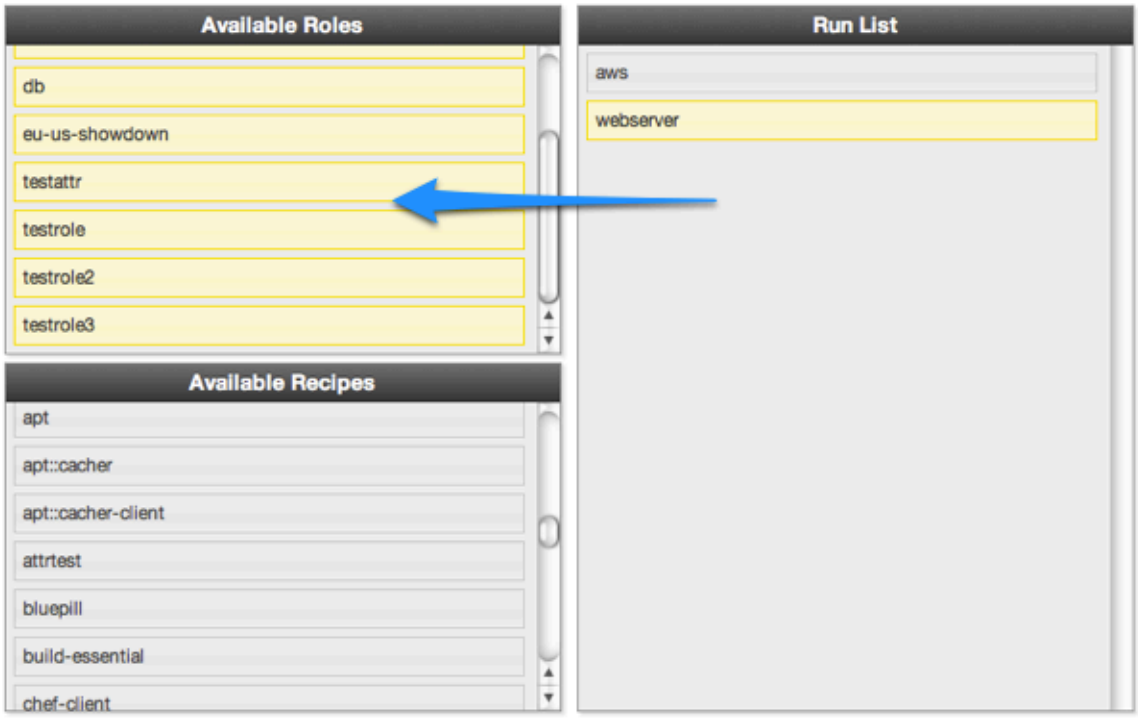
 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

1. Login to the Hosted Chef Management Console and select an organization.
2. Select the *Nodes* tab.
3. Click on the *Edit* link next to the node you want to edit.
4. To **add** a role or recipe to the node being edited, drag it from *Available Roles* or *Available Recipes* to *Run List*:



5. To **remove** a role or recipe from the node being edited, drag it from the *Run List* back to an *Available* section:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



6. Click *Save Node* to save.

Note: You need update permissions on the node to edit it.

Moving a Node to an Environment

You can move the node to an environment with the drop down box on the top of this page. More information on this can be found on the Managing Environments page.

Modifying Node Attributes

To modify the Attributes of a node, such that they override the values of the included recipe or role, you can use the *Editor* or *Source* tabs on this page. With the *Editor* tab you can edit the attributes by selecting the location on the tree and adding the details. You can use the *Source* tab to paste in JSON directly.

Note: You need update permissions on the node to edit the attributes.

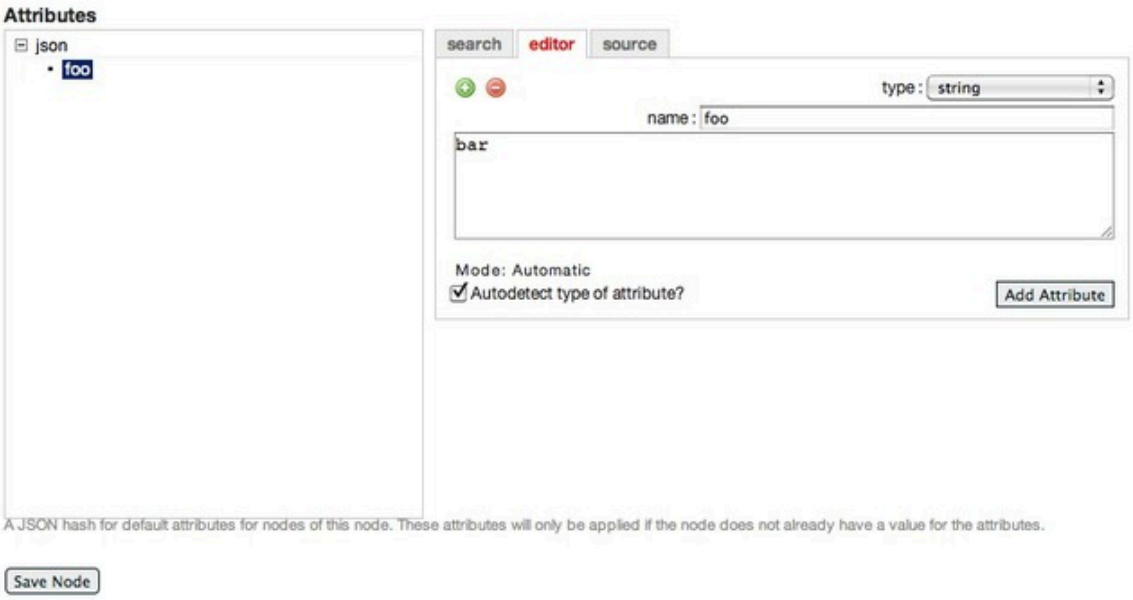
Using the Editor tab

1. Select "json" from the tree on the left and click on the *Editor* tab on the right.

2. Push the green add button under the *Editor* tab: .

3. Enter a name as the key and a body as the value and click *Add Attribute*. The following figure shows the Attributes tree and the JSON editor.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



4. Click the *Create Node* or *Save Node* button to save the node.

Using the Source tab

1. Enter the JSON body in the editor, and click the *Load JSON from source* button: 



2. Click the *Create Node* or *Save Node* button to save the node.

Deleting a Node

In order to delete a node:

1. Login to the Hosted Chef Management Console and select an organization.

2. Click on the *Nodes* tab.

2. Click the *Delete* link next to the name of the node you want to delete, and click *OK* on the confirmation message box. Or, click the link of the

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

node to view its details, and click *Delete* on the sub navigation menu.

Note: You need delete permissions on the node to delete it. If you don't have this permission, you may not see a *Delete* link or tab.

Checking Node Status

Node status such as platform, FQDN, IP Address, up time, last check-in time, and run list can be checked on the Management Console.

In order to check node status:

1. Log in to the Hosted Chef Management Console.

2. Click *Status* tab.

 Managing Groups with the Hosted Chef Management Console

Managing Organizations with the Hosted Chef Management Console 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Organizations with the Hosted Chef Management Console

Manage your organizations with the Management Console

- Selecting the Organization
- Creating an Organization
- Adding or Removing a User from an Organization
- Regenerating the Private Key and the Knife Config

Selecting an Organization to Use

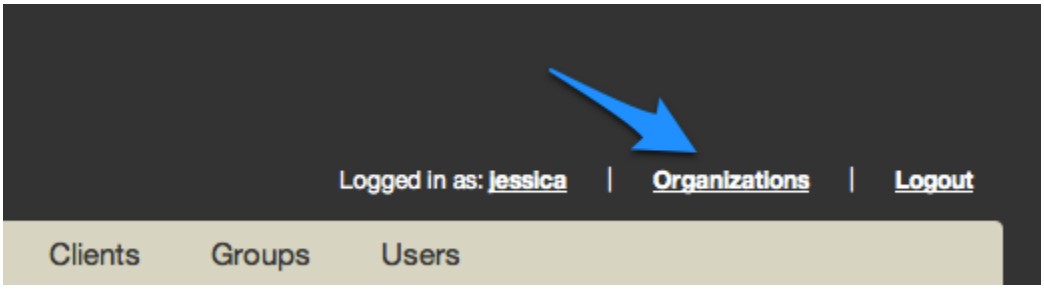
- **To follow these directions, first log into the Hosted Chef Management Console with your existing username.** If you have not signed up for an account yet, follow the directions on the Set up Opscode User and Organization page.
- **If you're interested in editing account, nodes or other data in the Management Console instead, return to Hosted Chef Management Console, and make the appropriate selection.**

A user is allowed to be associated with multiple organizations. When there are multiple orgs, you can select which organization to use from the list of associated organizations in the Hosted Chef Management Console. Subsequent actions will be scoped to that organization.

To see which organization is currently selected, or to view the list of associated organizations:

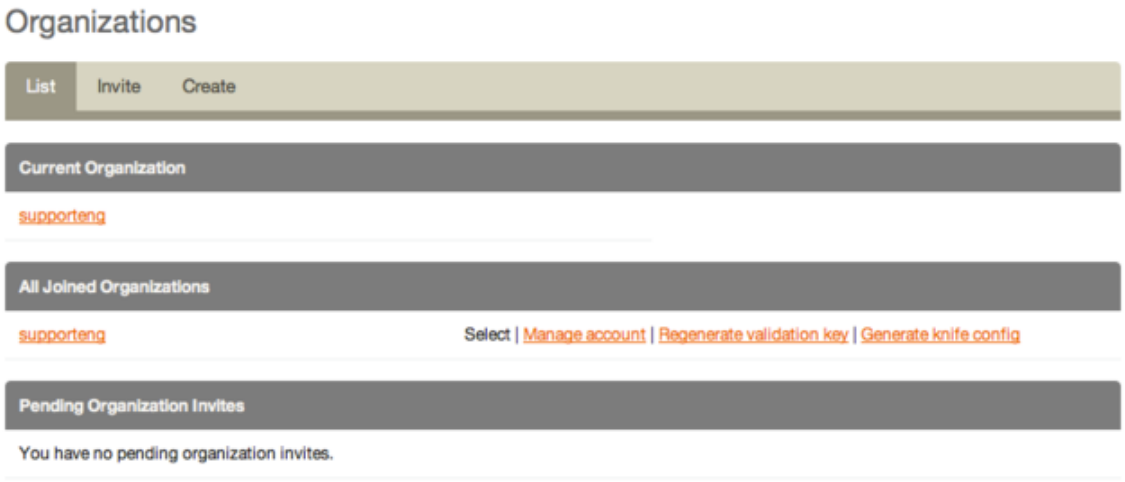
1. Log into the Hosted Chef Management Console

2. Click *organizations* in the upper right of the screen, next to your username.



3. On the Organizations page, click the *Select* link next to the organization you want to select to use. If there are no associated organizations listed, please make sure your user account is associated with an organization or contact Opscode Support.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

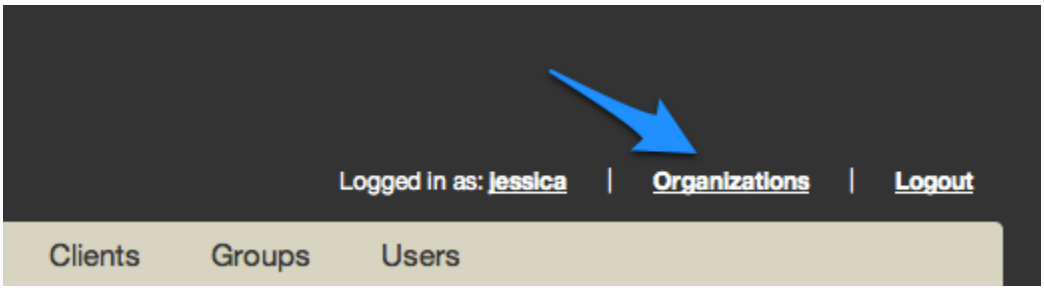


Creating an Organization

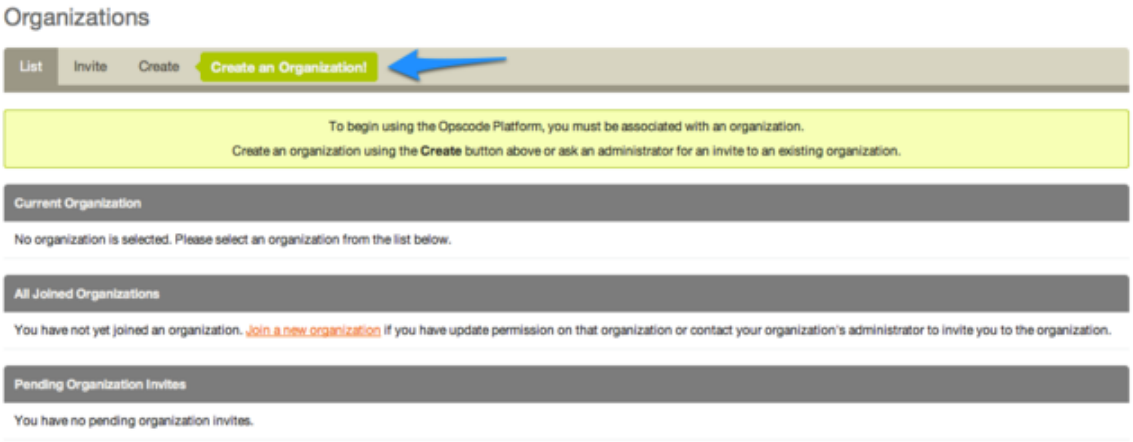
In order to create a new organization:

1. Log in to the Hosted Chef Management Console using your existing username.

2. Go to the *Organizations* page. If you do not have an organization associated with your user, you are on that page automatically. Otherwise, click the *Organizations* link on the upper right corner.



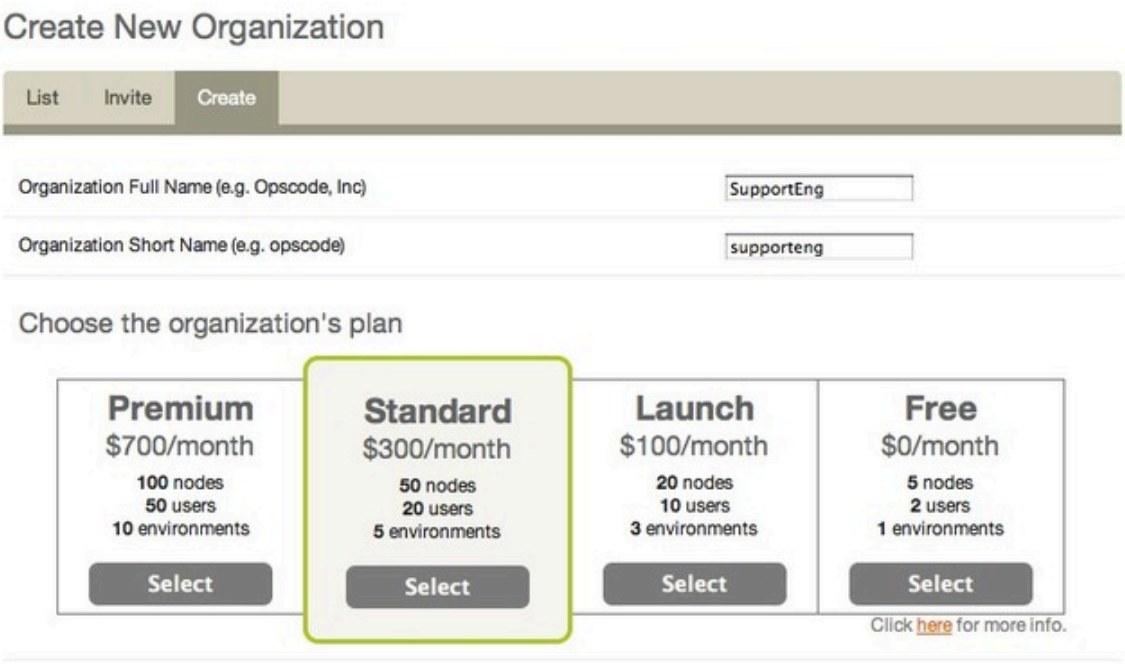
3. On the *Organizations* page, click *Create*.



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

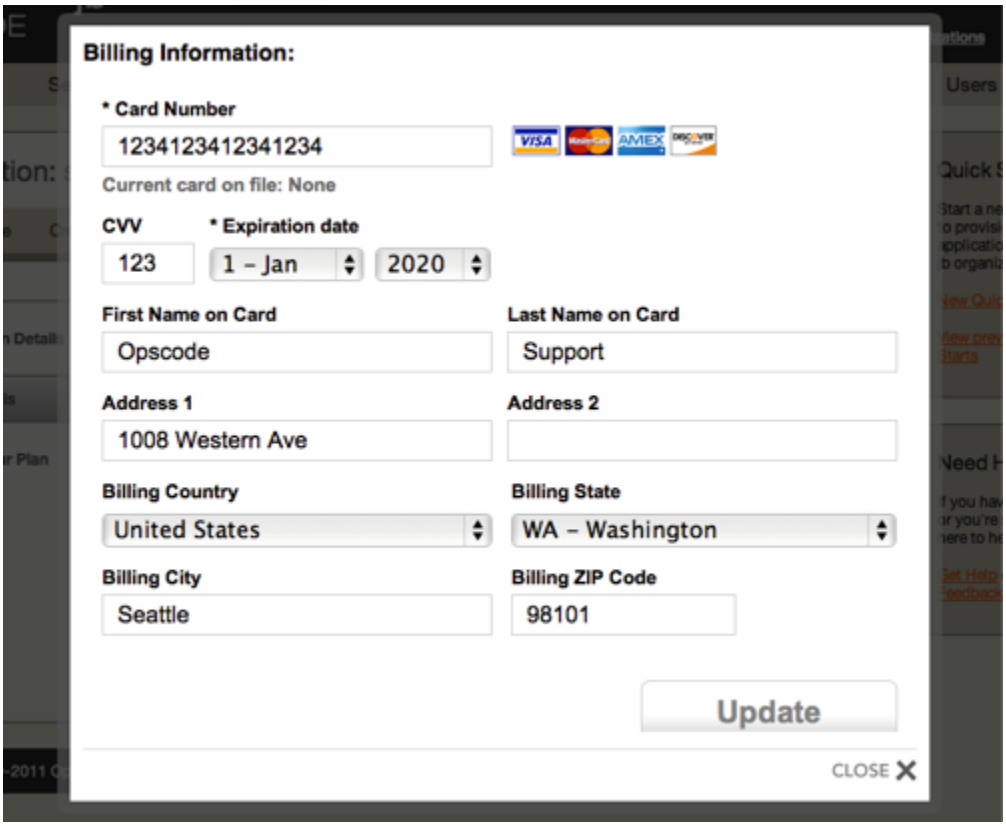
4. Enter a full name and a short name. The short name is like an account name for your new organization, only lower case a-z, -, and _ are supported. You cannot use spaces in the short name.

5. Select your desired billing plan to create your organization.

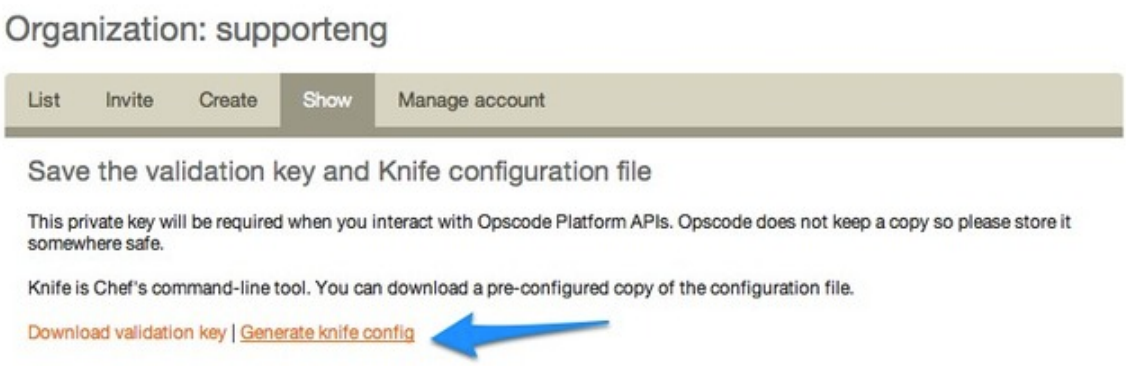


6. If you selected any of the non-free billing plans in the previous step, you will be prompted to enter your billing information at this time. Please do so before you proceed.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



7. Download your validation key and a sample Knife configuration file. You need to use the validation key to register new clients using this organization. The Knife configuration file allows you to start using Chef's command-line tool, Knife, immediately.



If you will be using Knife, be sure to download the User's Private Key as well.

Adding or Removing a User from an Org

If you want other users to be able to access the organization with Knife or the REST API, you will need to add them to the organization with the co rrect permissions so they are authorized to make changes to it.

Associate a user to the organization

In order to associate a user with the current organization:

1. Ask them to sign up for a Hosted Chef account and give you their username.

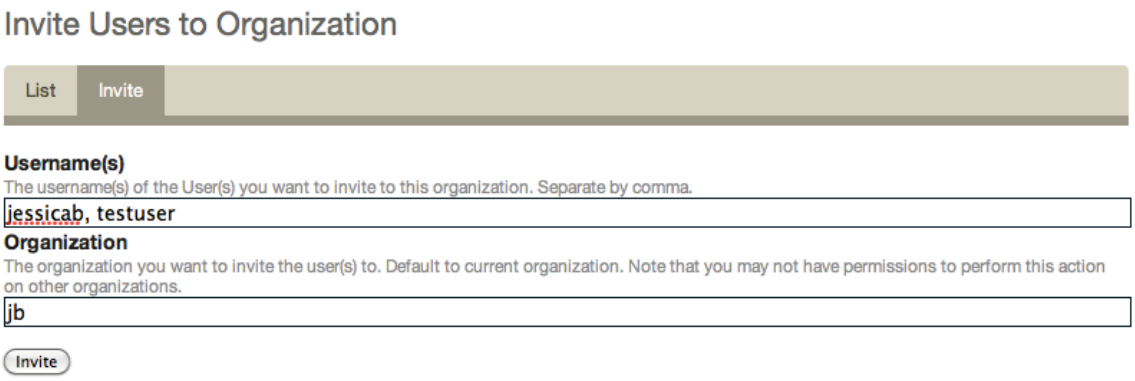
 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

2. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

3. Click the *Users* tab in the main navigation menu.

4. Click *Invite* in the sub-menu

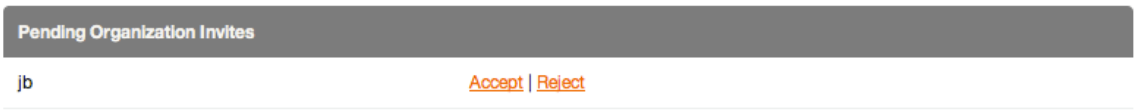
5. Enter the users you want to associate to the organization shown, separated by comma if there is more than one. And then click on the *Invite* bu tton:



Afterwards you should see this user appear in the Pending User Invites section of the *Users* tab.

6. Ask the user to log into their account to accept the invite.

7. The user will need to click on *organizations* in the upper right hand corner. Then they can click on the *Accept* link next to the invite at the bottom of the page:



Note: You need update permissions on the organization to be able to associate a user with the organization. Users in the admins group have update permissions by default.

Remove a user from the organization

In order to dissociate a user from the current organization:

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click the *Users* tab in the main navigation menu.

3. Click *Dissociate* next to the username. Click *OK* on the warning message box.

Note: You need update permissions on the organization to be able to dissociate a user from the organization. Users in the admins group have update permissions by default.

Regenerating the Private Key and the Knife Config

The private key for the organization is also referred to as the validation key and is used to authenticate with both Knife and chef-client. Regenerating the key will be useful if you have lost the organization's private key, if the private key has been exposed to someone you don't want to have access to your Hosted Chef account, or if you wish to regularly rotate keys for security purposes.

You can also download a default Knife config on the same page.

In order to download the private key and the knife.rb file:

1. Log into the Hosted Chef Management Console

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

2. Click *organizations* in the upper right of the screen, next to your username.

3. On the Organizations page, click the *Regenerate validation key* and the *Generate knife config* links to regenerate the private key and download a Knife config file. If there are no associated organizations listed, please make sure your user account is associated with an organization or contact Opscode Support.

If you will be using Knife, be sure to download the User's Private Key as well.

 Managing Nodes with the Hosted Chef Management Console

Managing Permissions with the Hosted Chef Management Console 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Permissions with the Hosted Chef Management Console

This page explains how to manage your permissions:

- Viewing and Editing Permissions
- Updating Billing Permissions

Viewing and Editing Permissions

- **To follow these directions, first log into the Hosted Chef Management Console with your existing username.** If you have not signed up for an account yet, follow the directions on the Set up Opscode User and Organization page.
- **If you're interested in editing account, nodes or other data in the Management Console instead, return to Hosted Chef Management Console, and make the appropriate selection.**

The Hosted Chef Management Console allows users with grant permissions to view and modify permissions on Roles, Nodes, Cookbooks, Databags, Clients, and Groups. For each type of object listed, there are two levels of permission settings: the global level and object level.

Permissions on the global level of an object type determines whether you are able to create new objects of this type and whether you can view the list of the objects of this type (for example, creating new groups and listing groups). Permissions on the object level determines whether you are able to do anything with that specific object (for example, showing, editing, or deleting the "foobar" group).

For more information on how permissions work, and default permissions see the Hosted Chef Authorization page.

Viewing and editing permissions on global object levels

Note: Roles are used here as an example, but it works the same way for Nodes, Cookbooks, Databags, Clients, and Groups. The permissions on Databag Items are inherited from their parent Databags.

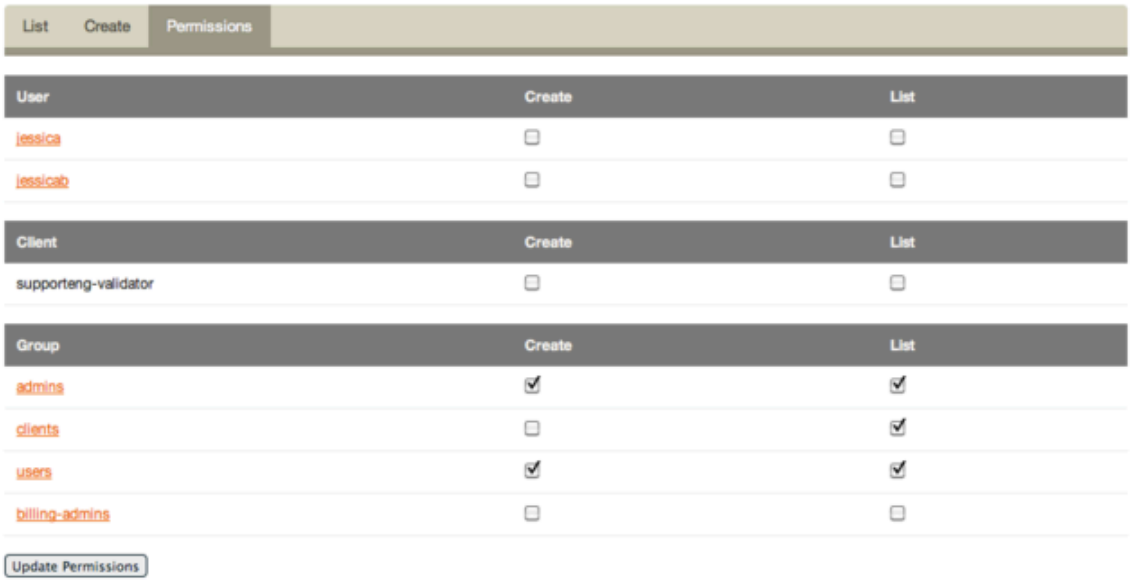
In order to view and update the global level permissions on an object type (Roles in this case):

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click the *Roles* tab on the main navigation menu.

3. Click the *Permissions* tab on the sub-navigation menu, select or deselect the permission checkboxes as appropriate, and click *Update Permissions*.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Viewing and updating permissions on object level

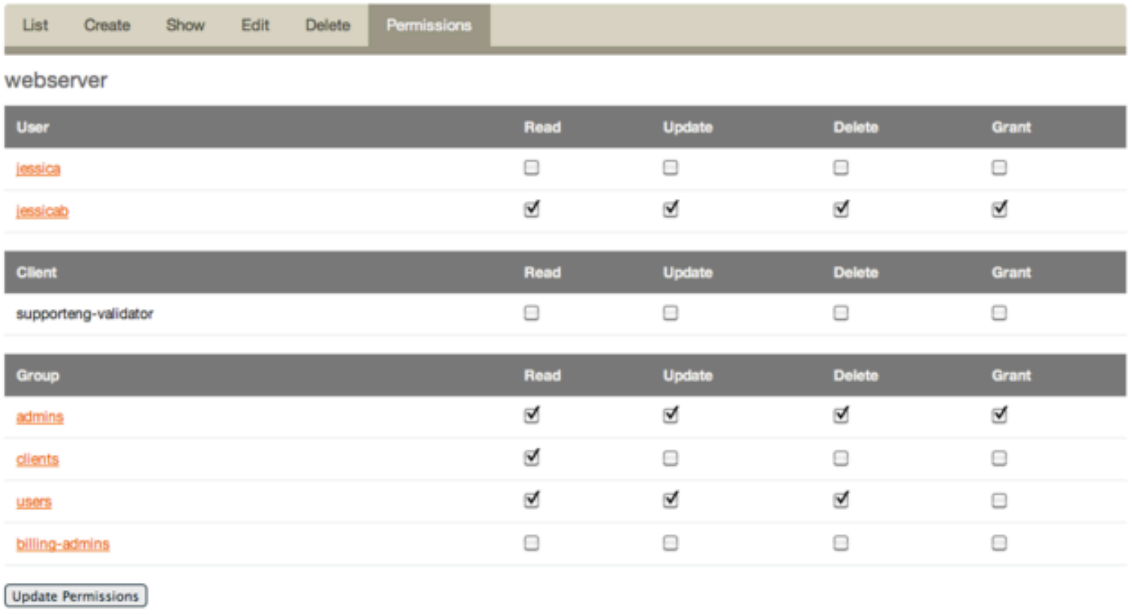
In order to view and update the object-level permissions on a specific object (a role, in this example):

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click the *Roles* tab on the main navigation menu.

3. Click the role to view the details of the role.

4. Click the *Permissions* tab on the sub-navigation menu, select or deselect the permission checkboxes as appropriate, and click *Update Permissions*.

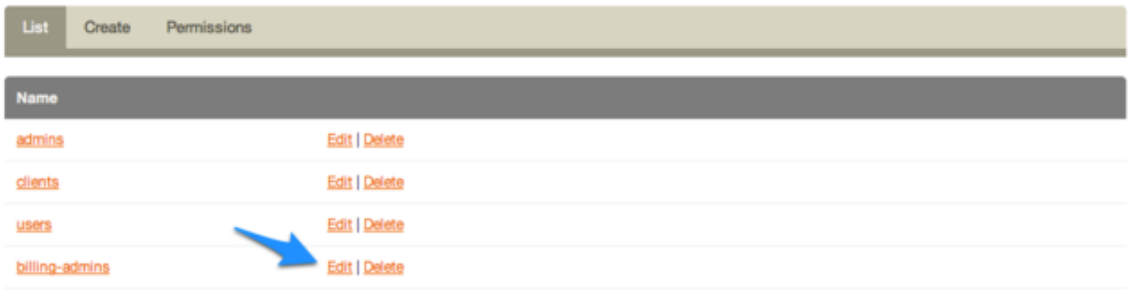


 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

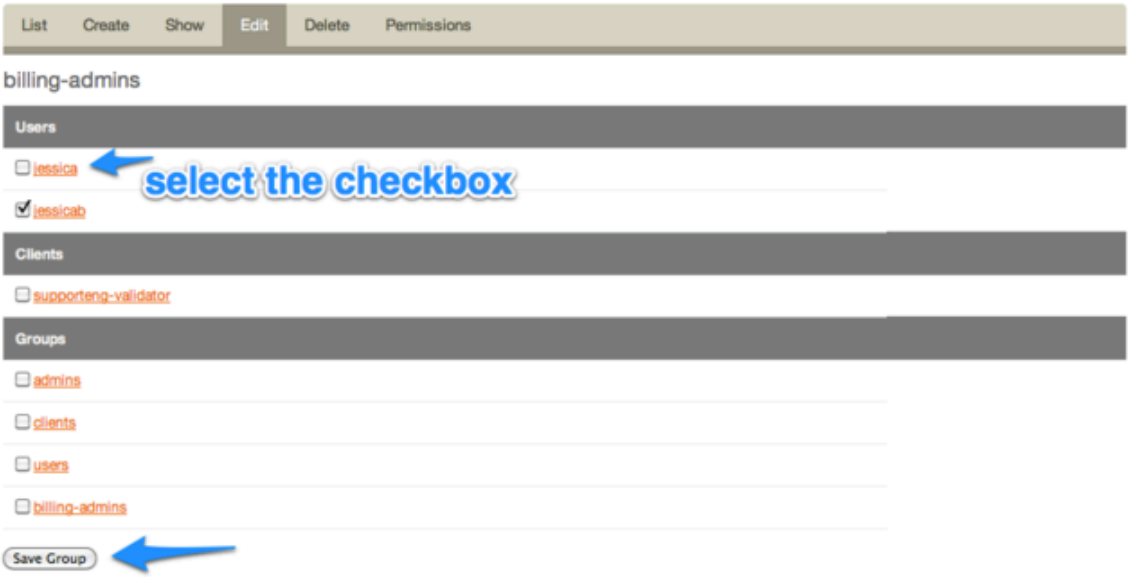
Updating permissions for the billing admin

You can also change who is the billing admin, the users that are able to see and edit the billing information on the account:

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.
2. If the user is not a member of the organization yet, add them to it by following the directions on the Managing Organizations page.
3. Click on the *Groups* tab.
4. Click on the *edit* link next to the *billing-admins* group:



5. Check the checkbox for any users you want to be billing admins, and then click on the *Save Group* button:



Managing Organizations with the Hosted Chef Management Console

Managing Roles with the Hosted Chef Management Console

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Roles with the Hosted Chef Management Console

This page explains how to manage your roles:

- Listing Roles
- Viewing a Role
- Creating a Role
- Editing a Role
- Modifying Role Attributes
- Editing Roles on a Node
- Deleting a Role

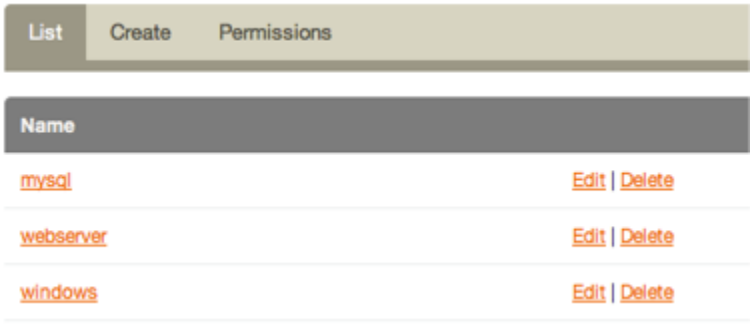
Listing Roles in the Organization

- **To follow these directions, first log into the Hosted Chef Management Console with your existing username.** If you have not signed up for an account yet, follow the directions on the Set up Opscode User and Organization page.
- **If you're interested in editing account, nodes or other data in the Management Console instead, return to Hosted Chef Management Console, and make the appropriate selection.**

To list roles in your organization:

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click *Roles* on the main navigation menu.



Note: You need list permissions on the global roles level to see the list of roles.

Viewing a Role

Viewing a role shows the recipes and roles included in the role, as well as a tree view of the attributes associated with the role.

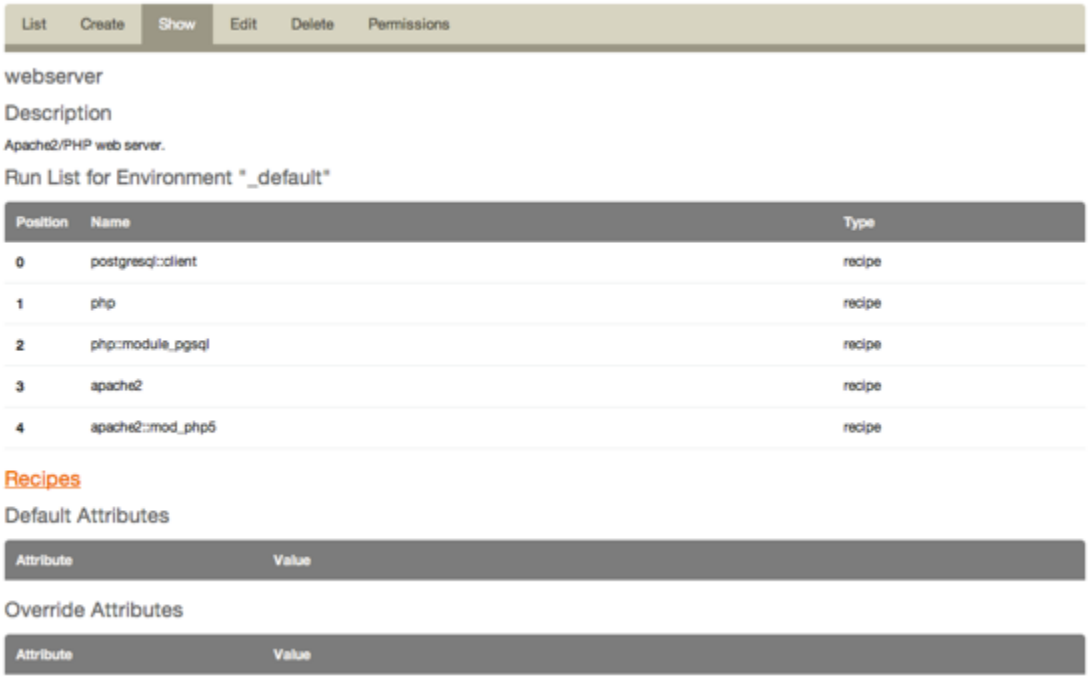
In order to view a specific role:

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click *Roles* on the main navigation menu.

3. Click on the name of the specific role you want to view. This will bring you to a page where you can see the run_list and attributes for the role:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Note: You need read permissions on the role to view it.

Creating a Role

The role create page allows you to create a role. You can add recipes and roles to the run list. There is also a JSON editor for editing, adding and removing JSON attributes associated with the role.

In order to create a role:

1. Login to the Hosted Chef Management Console and navigate to the *Roles* tab.

2. Make sure that the correct environment is selected at the top of the screen, select *None* if you want this role to work the same on all environments. Click *Create* on the sub navigation bar.

3. Give the role a name and an optional description.

4. Continue onto editing the role, by adding to it's run_list and editing the attributes. This is described further in the next section.

5. When you have finished editing the role, click *Create Role* to save the new role.

If you selected a custom environment and need the run list setup differently with another environment, you can now change the environment and edit the run list. If an environment is selected from the drop down at the top of the page, you will only see the run_list and attributes specific to this environment. You can see the default run list and attributes by selecting *None* or *_default* from the drop down.

Note: You need create permissions on the global roles level to create new roles.

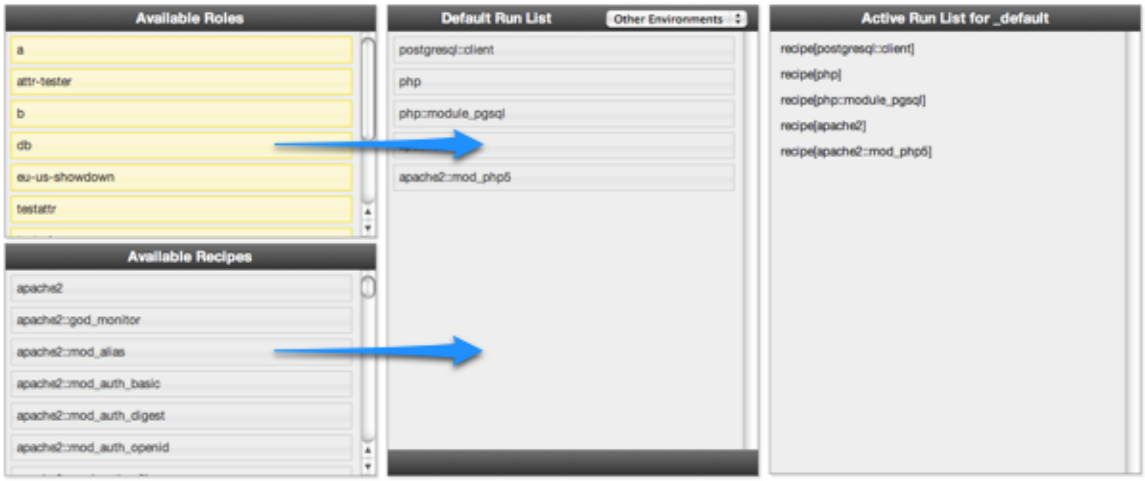
Editing a Role

1. Login to the Hosted Chef Management Console and navigate to the *Roles* tab.

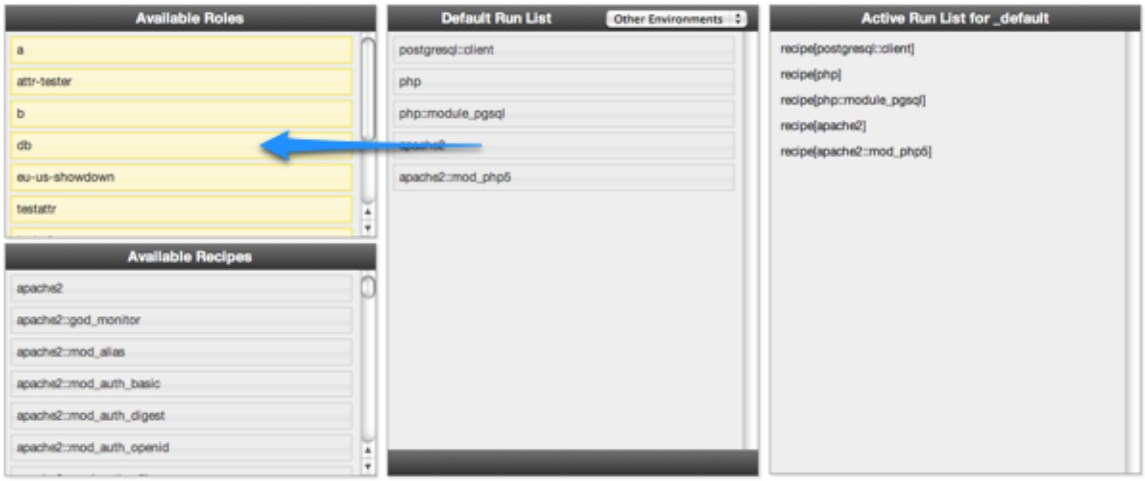
2. Make sure that the correct environment is selected at the top of the screen, select *None* if you want this role to work the same on all environments. Click on *edit* next to the role you want to edit.

3. To **add** a role or recipe to the role being edited, drag it from *Available Roles* or *Available Recipes* to *Run List*:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



4. To **remove** a role or recipe from the role being edited, drag it from the *Run List* back to an *Available* section:



The *Active Run List* panel on the right shows the current active run list for the environment selected with the drop down box at the top of the page. If the *None* or *_default* environments are selected it shows the default run list. If you've selected an environment that does not have a specific run list yet it will also show the default run list. If you edit the run list while an environment is selected at the top of the page, your changes will be saved as a run list specific for nodes in that environment.

5. Click *Save Role* to save.

Note: You need update permissions on the role to edit it.

Modifying Role Default and Override Attributes

You can edit the override Attributes of a role, to override the values of an included recipe or role, and you can edit the default attributes of a role to specify attributes that do not exist in included recipes and roles. You can use the *Editor* or *Source* tabs on this page to edit the attributes. With the *Editor* tab you can edit the attributes by selecting the location on the tree and adding the details. You can use the *Source* tab to paste in JSON directly.

Note: You need update permissions on the role to edit the attributes.

Using the Editor tab

1. Click on the plus sign next to "json" and select "default" or "override" from the tree on the left, depending on the type of attribute you want to create. If you're unsure, you should select "default" unless you are trying to override a default attribute that is already set in the node or in a recipe.

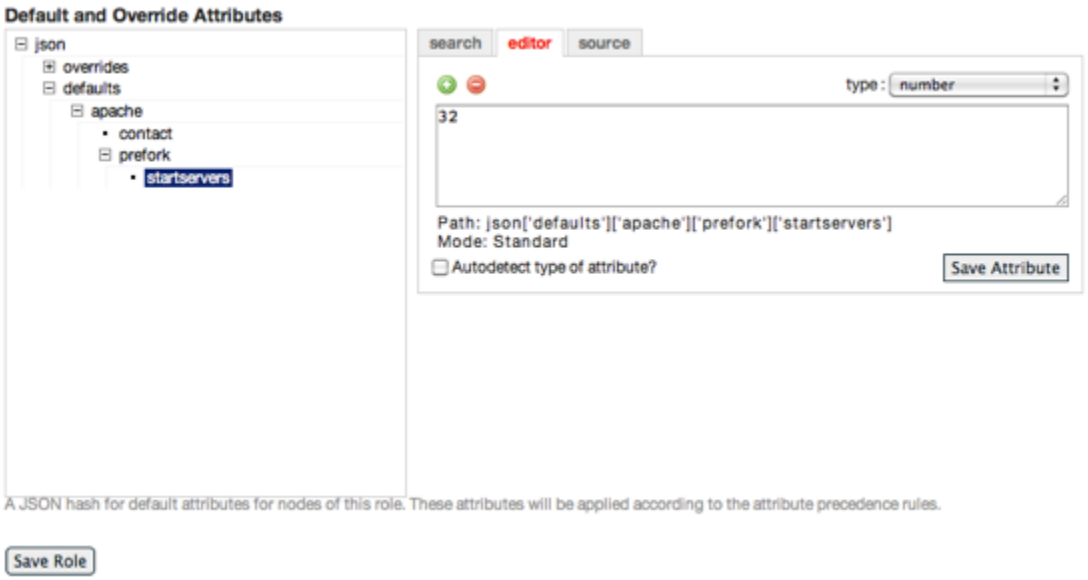
 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

2. Click on the *Editor* tab on the right.

3. Push the green add button under the *Editor* tab:  .

4. Select a type of attribute from the drop down box. If you are unsure, you most likely want "string" if the value is text, or "number" if the value is a number.

5. Enter a name as the key and a body as the value and click *Add Attribute*. The following figure shows the Default and Override Attributes tree and the JSON editor.



6. Click the *Create Role* or *Save Role* button to save the role.

Using the Source tab

1. Enter the JSON body in the editor, and click the *Load JSON from source* button: 



2. Click the *Create Role* or *Save Role* button to save the role.

Editing the Roles on a Node's Run List

To edit the roles that are in a node's run list, follow the Managing Nodes page. You can also use Knife to edit the run list for the node.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Deleting a Role

In order to delete a role:

1. Login to the Hosted Chef Management Console and navigate to the *Roles* tab.

2. Click the *Delete* link next to the name of the role you want to delete, and click *OK* on the confirmation message box. Or, click the role to view its details, and click *Delete* on the sub navigation menu.

Note: You need delete permissions on the role to delete it. If you don't have delete permissions, you may not see a *Delete* link or tab.

 Managing Permissions with the Hosted Chef Management Console

Managing Users and your Private Key with the Hosted Chef Management Console 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Users and your Private Key with the Hosted Chef Management Console

This page explains how to update your user information:

- Editing your User Profile and Changing your User Password
- Downloading and Using your Private Key

And manage other users in your organization:

- Listing Users
- Showing User Details
- Adding or Removing a User from an Organization
- Adding or Removing a User from a Group

Editing your User Profile and Changing your User Password

- **To follow these directions, first log into the Hosted Chef Management Console with your existing username.** If you have not signed up for an account yet, follow the directions on the Set up Opscode User and Organization page.
- **If you're interested in editing account, nodes or other data in the Management Console instead, return to Hosted Chef Management Console, and make the appropriate selection.**

Your Opscode user account is shared across the Opscode Platform, the Hosted Chef Management Console, the Community Site, and http://help.opscode.com. You can update your name, twitter account information, geographic information, email address, profile image, and communication preferences. Information on changing this data or your password can be found on the Managing your Opscode User Account page.

Downloading and Using your Private Key

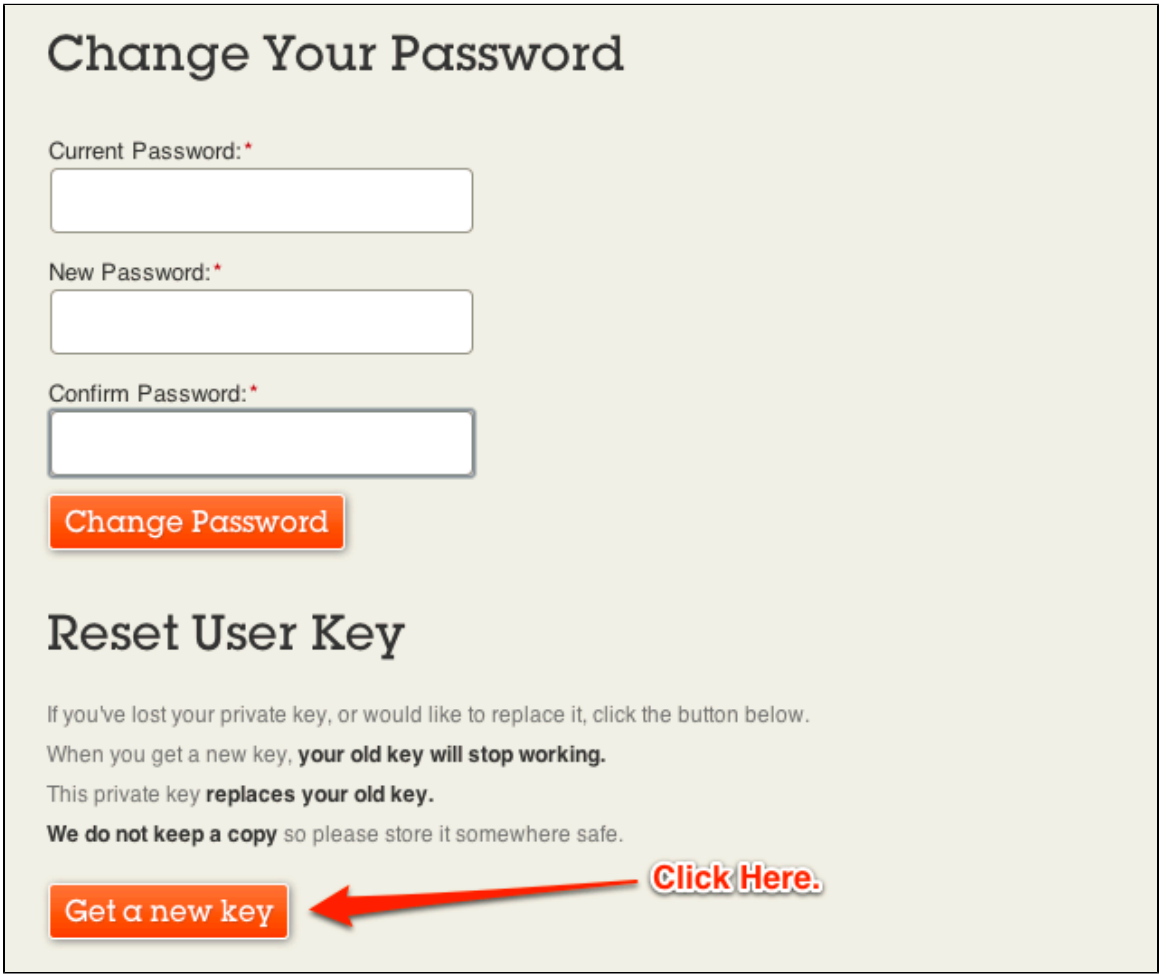
This section explains how to download and use your private key, which is used to authenticate with Knife or directly via the REST APIs. For security purposes, Opscode does not keep a copy of this private key. We keep only the corresponding public key to verify your requests. Your private key is like a password or bank PIN number, so keep it safe.

You can also find information on downloading the organization key and knife config file on the Setup Opscode User and Organization page.

Downloading your User's Private Key

You will need a new key if you have lost your old key or simply want to rotate your key for security. On the password page of the Opscode Account Management Tool, click "Get a new key" as shown in the figure below.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Your browser will prompt you to save the new key file. Save it in the same location as your old one, typically ~/.chef/. You should check that knife is configured to look for your user key in the correct location by inspecting knife.rb.

Using your User's Private Key

The file you downloaded is a private key for the user account you created. It is used for both authentication, "are you a valid user?", and authorization, "are you allowed to perform this operation on this object?". More information on this can be found on the Authentication and Authorization page.

If you open the file with a text editor (Notepad.exe on Windows, TextEdit on the Mac, gEdit / Vi / Emacs on Linux), it should look like the following, but the actual value will differ:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEArp5wl2DsS3WtUa9T3fAfREEbvm9FkhCcYAW/6vHearRYH9oi
YJLYEPuD6ccjxotti69VQ4nkGc/55fc6os8N2qkslGjRwSkk6JUt0wxTiOyEvcsC
LNap4osjkEDErBWGPc34PEEk77BmFUxxuNNqr/M8bheE4qJrFaBXnz+RJ3OPcTuv
lwrIu8f32+Sl8Ex9Aaw2ESFuKSgVOACc6BVi07rQR9CIWlFM0dgM8pFTzM96Gqdp
Xza7QQKBgCHREdDQ3LsPgSF+mAskjUcrihL1ycBQoaB9DoGLqOc8XrIJvdDn+IY+
4wD+07Oo1WBmcu1pka/eUSIfiEEq+wEBFFjNcQIDAQABAoIBADpHdQqA9bxlpRf7
LUGIb9cG5+ySFKUgWCCQKrKOUQ5J7JOM6SZUPDrGxwsSRLVyOXiCZzGoZvHagOwo
rHBELLqK9e7+YOtYARKDH40tsv+i+NajV7GBX6fVimQNydYpROmBwdTSZTJ+LEP1
hIsMPLJGuyd/QAGn9Zd9MWFcBOUmVg2aLAQ/smmKTpqu9PkNBzyiEw5pdSuxMFEN
H8/k5XtV5UaqJrUGDEqMphUFXYsACSVeJZ3Vd/B/XgwnI2y3ZqrreS9Fm1KTRTIt
vZ5jqpbsxf18IQ4Nx/3KvgXf+9glpj7CNIayCQDuG4GfjB+ATWDBRX3e+YeEnehm
1KA3kbBZURRlnBxEwIym83fyXzpMDUvlnPzc5CVisIkl60MZDEmGCrpTVXcPn2k1
mKuO/YWhg5mINRK3W7IouPIROXklZGQfR1/WmwV9N3q44x5wMnldKWeO8S7I/oky
c/zY03z3HBaYEHG7QeqMy9F4MbS8sa+GzRLZuY+Opn863+B5O9pcYp0CgYEAxgsn
C1snRWrugqXyNE3LnEM3oE57+NVPNGn/viu+dcVMcbRGkFgv6/siMJ/K+FSOxKEu
LxdeIssm+73wzEpsVJIUg4yhSgpj6VWKIMrodMiK6xLnfGNjrLpEoeCyARc/dvQd
OZsA0FJVKXyeuSPWEbioctTV1FHzNgNRZ1/Dw+UCgYBAZiSLVuMnqMH9Dee3Brbr
Y8J+X/ECgYEA4bhhSIdb50y4o2r+j7y4X+M/uNTRBnRYQ3pJ/+bTxS8OPJ0hbJ8J
KejGrdg93eumEOcd33Zu+7bSpPB/fpFAemE6Zary/pdACdLNSagF9qt+MvyrDVM+
ayRnRsgCRMZ9XUP5iaFWGQKBgQC0fVWIM/sGeP4Gc9bZdE0sjNYVc3Hoxf75deFu
WqfmIzuhSRSFZegHuu30Bi5vmoCiEXC6Sd0pivIBE8SWJ6YAuaSvLtBZpIiPq0SG
0Pl99Ycl1oTCCJK+yMPkBDQ2akmcz74S0gFb2q1isgC2GsLuQ6zBetemOUm+4CdA
tJrck5bF3y1115hrusd5nzKipPwSjNZKYsqovXQevOjjQU8q24vZBTPQXiNZ/cqR
T7hl6b5fMp/c3d27aCg+9fBnjP9uGGtIftKigOZDpd5hvl72SsRxrHSm8ZFumWY3
TT6BQEheN21G6rXaYDVQ1WvQBpnMasWPBaHzLfbs4yS5dBVYYlS8
-----END RSA PRIVATE KEY-----
```

Any tool which sends a request to Hosted Chef on behalf of your user account uses this key to sign the request. This includes but is not limited to the command line tool 'knife', and the 'chef-client' which runs on machines under configuration management.

Make sure you can find this file later. You'll need its path when configuring other tools. Here's an example of the configuration file for the 'knife' command line tool. Notice your user's private key file path is referred to here as 'client_key':

```
log_level        :info
log_location     STDOUT
node_name        'your-user'
client_key       '/Users/you/.chef/your-user.pem'
chef_server_url  'http://127.0.0.1/organizations/your-org/'
cache_type       'BasicFile'
cache_options( :path => '/Users/you/.chef/checksums' )
```

More Information on Private Keys

For more information on public-key cryptography and on Opscode's API request signing refer to the wiki page on Authentication or http://en.wikipedia.org/wiki/Public-key_cryptography

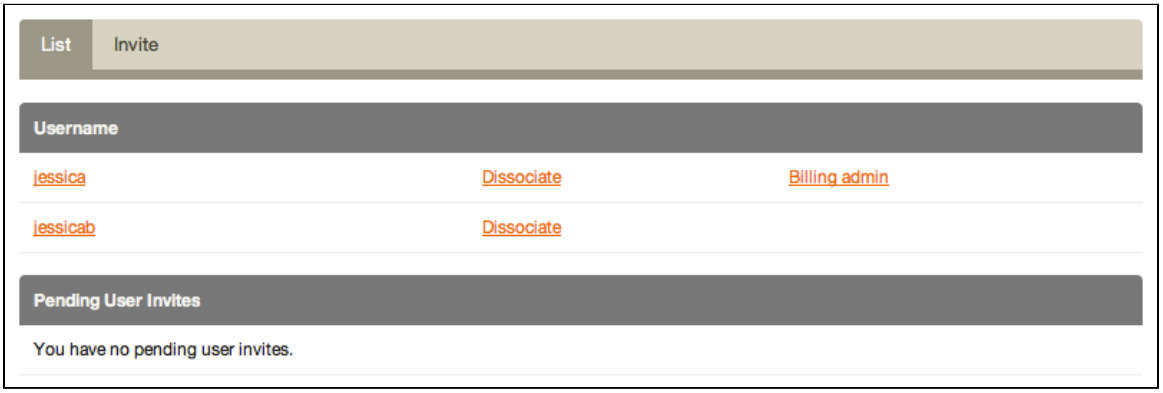
Listing users

In order to view the list of users in the organization:

1. Log in to the Hosted Chef Management Console and select an organization to work with if you are associated with multiple organizations.

2. Click the *Users* tab on the main navigation menu.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Note: Listing users requires the read permission on users. Anyone associated with the organization who has the grant permission on users can grant you such permission, specifically, users in the admins group by default have the grant permission.

Showing User Details

Users in the admins group or who have read permission on a particular user can view the details of the particular user, such as name, e-mail, and public key.

In order to view the details of a particular user:

1. Log in to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click the *Users* tab on the main navigation menu.

3. Click on the user you'd like to view.

Note: You need read permissions on the particular user in order to view the user's details. Anyone with grant permissions on the user can grant you read permissions.

Adding or Removing a User from an Organization

Information on adding users to your org can be found on the Managing Organizations page.

Adding or Removing a User from a Group

Information on adding users to your groups can be found on the Managing Groups page.

Managing Roles with the Hosted Chef Management Console

Managing your Account and Billing Information

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing your Account and Billing Information

You are able to update account information for your organization from the Hosted Chef Management Console

- View and Edit Billing Details
- Change Billing Plan
- Cancel a Hosted Chef Subscription

Selecting the Organization to Edit

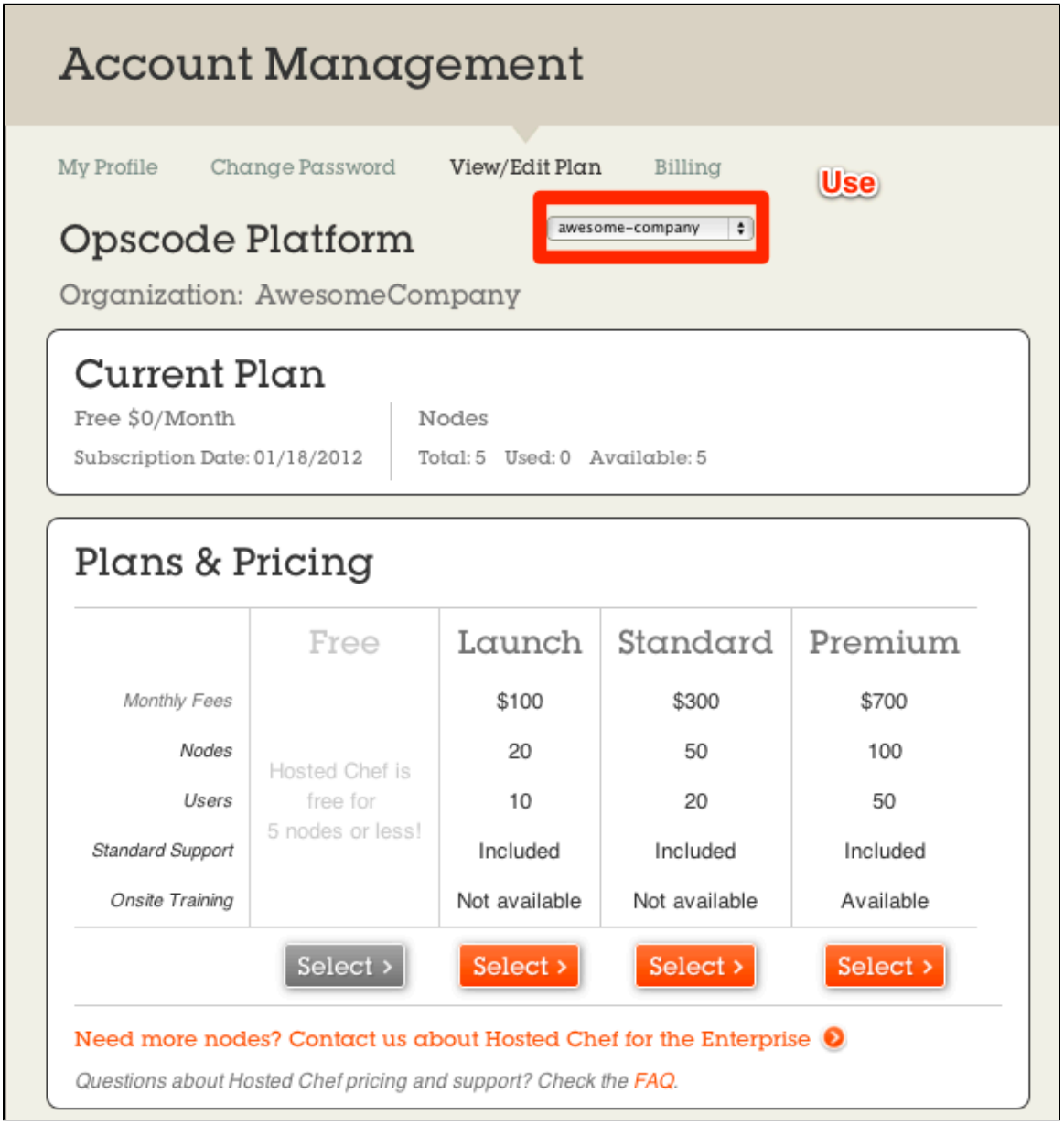
From the *Manage Account* page of the Hosted Chef Management Console you can update the details about your organization. The only information you can change about your organization is the *Organization Full Name* field . This field represents the formal way in which somebody would address your organization. To change your Organization Full name,

This article assumes you have already setup an opscode account. If you haven't yet, please follow the Setup Opscode User and Organization guide.

If you're interested in editing user, node or other data in the Management Console instead, return to Hosted Chef Management Console, and make the appropriate selection.

1. Log into http://manage.opscode.com,
2. Click on "Organizations" in the upper right-hand corner,
3. Click on "Manage Account" for the relevant organization,
4. The Hosted Chef Management console will redirect you to the Account Management Tool.

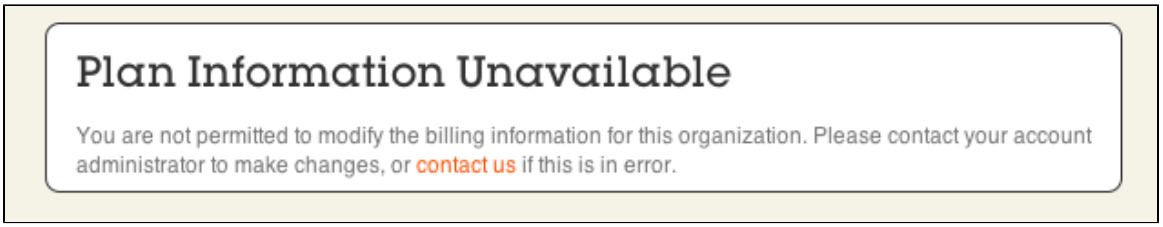
This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



From this tool, you can also switch between organization:

1. Click on either "View/Edit Plan" or "Billing Details" depending on the task (see below).
2. Organizations that you are a member of will appear in a drop down near the top of the page.

You will need to be a member of the billing-admins group to view or modify the billing or plan information. If you are not a billing-admin you will see this error instead:



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

To become a billing-admin, a member of the billing-admins group must add your user to the billing-admins group.

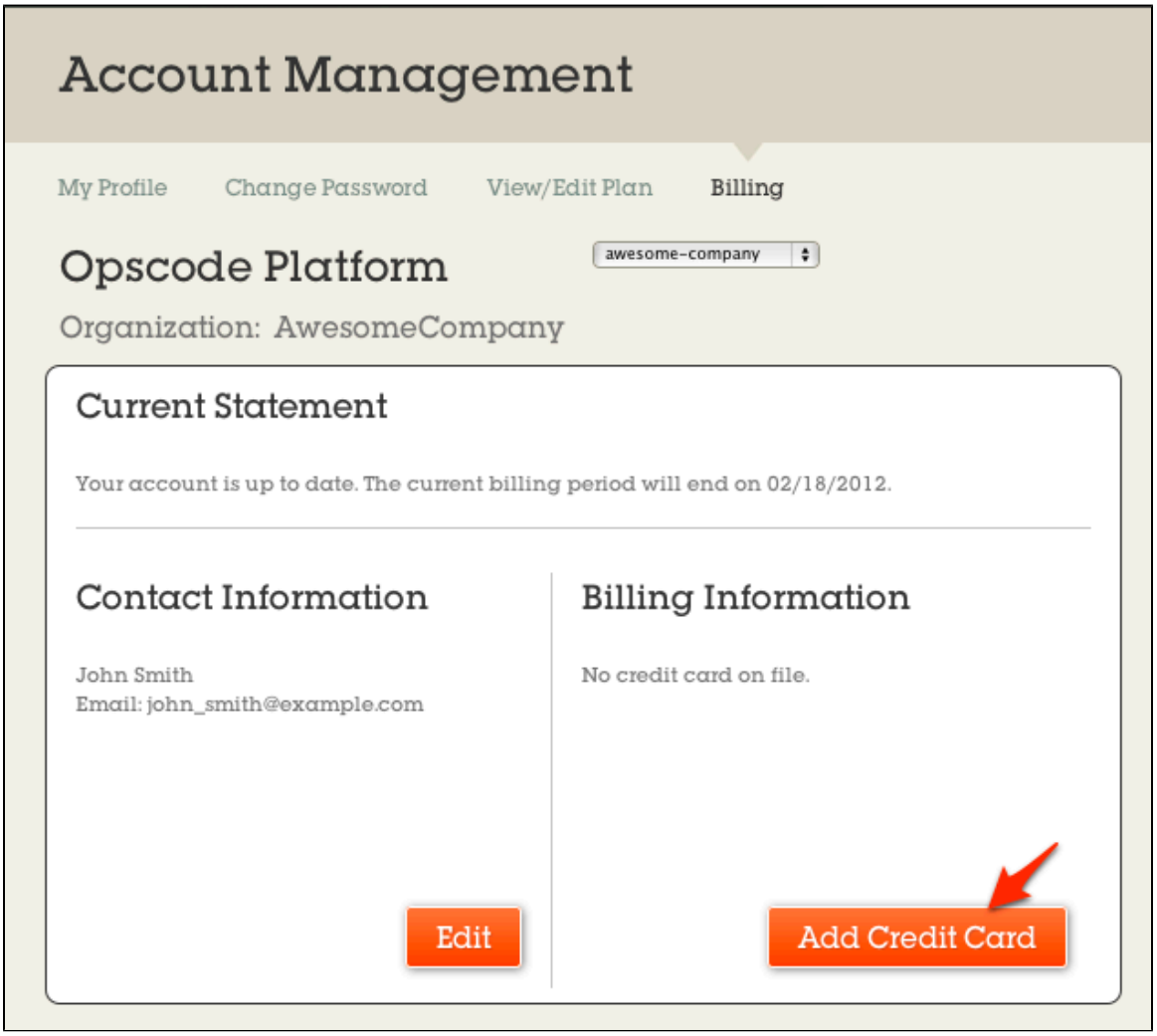
Billing Details

From the *Billing* page of the Account Opscode Management tool you can edit the billing details for your organization. The *Billing Details* section contains the following information:

- Credit Card Information
- Billing Contact Information
- Billing History

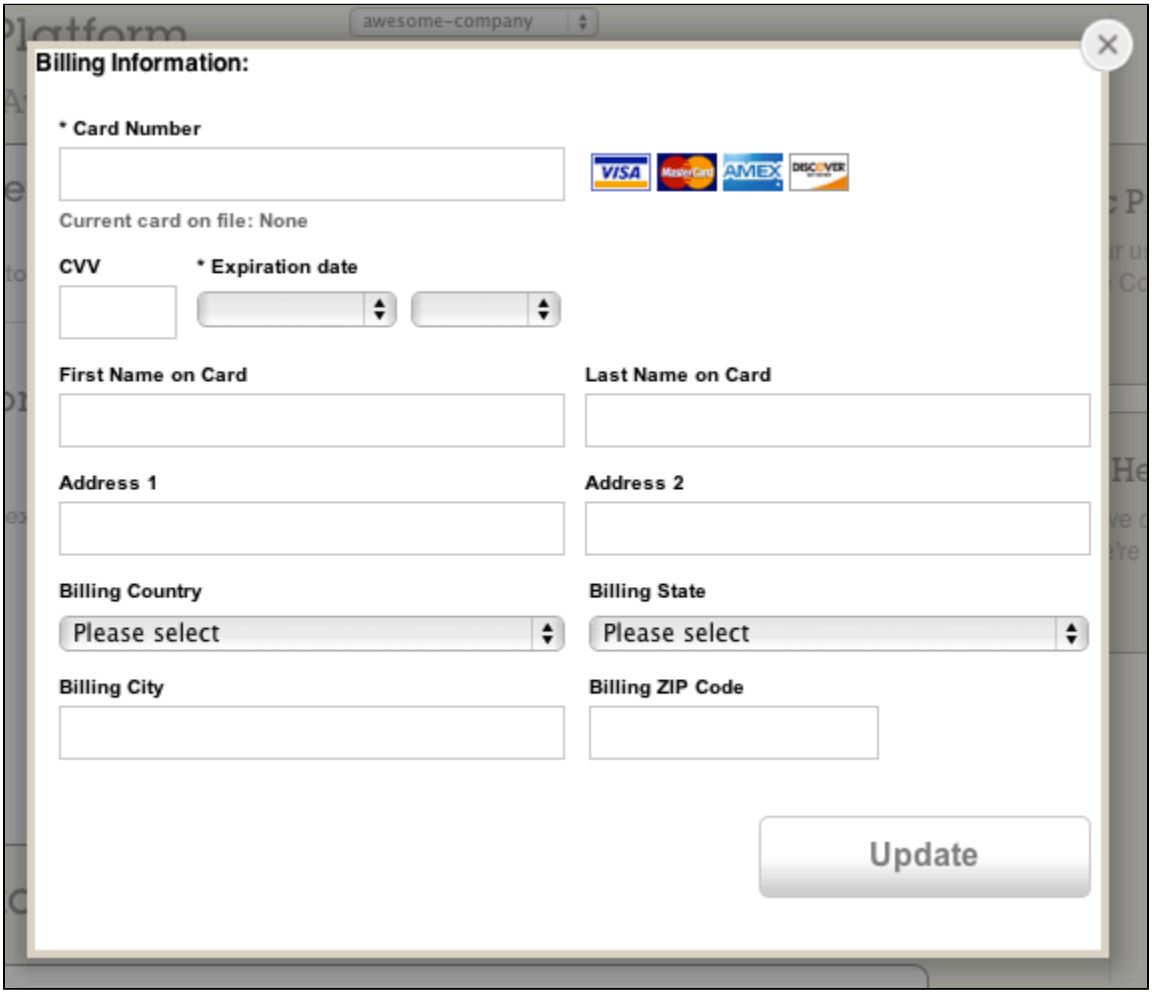
Credit Card Information

The *Billing Information* section of the *Billing Page* page shows you the information about the credit card you have on file. If no credit card is on file, you can use the "Add Credit Card" button to add a credit card.



Enter your billing information in the form that appears and click the update button.

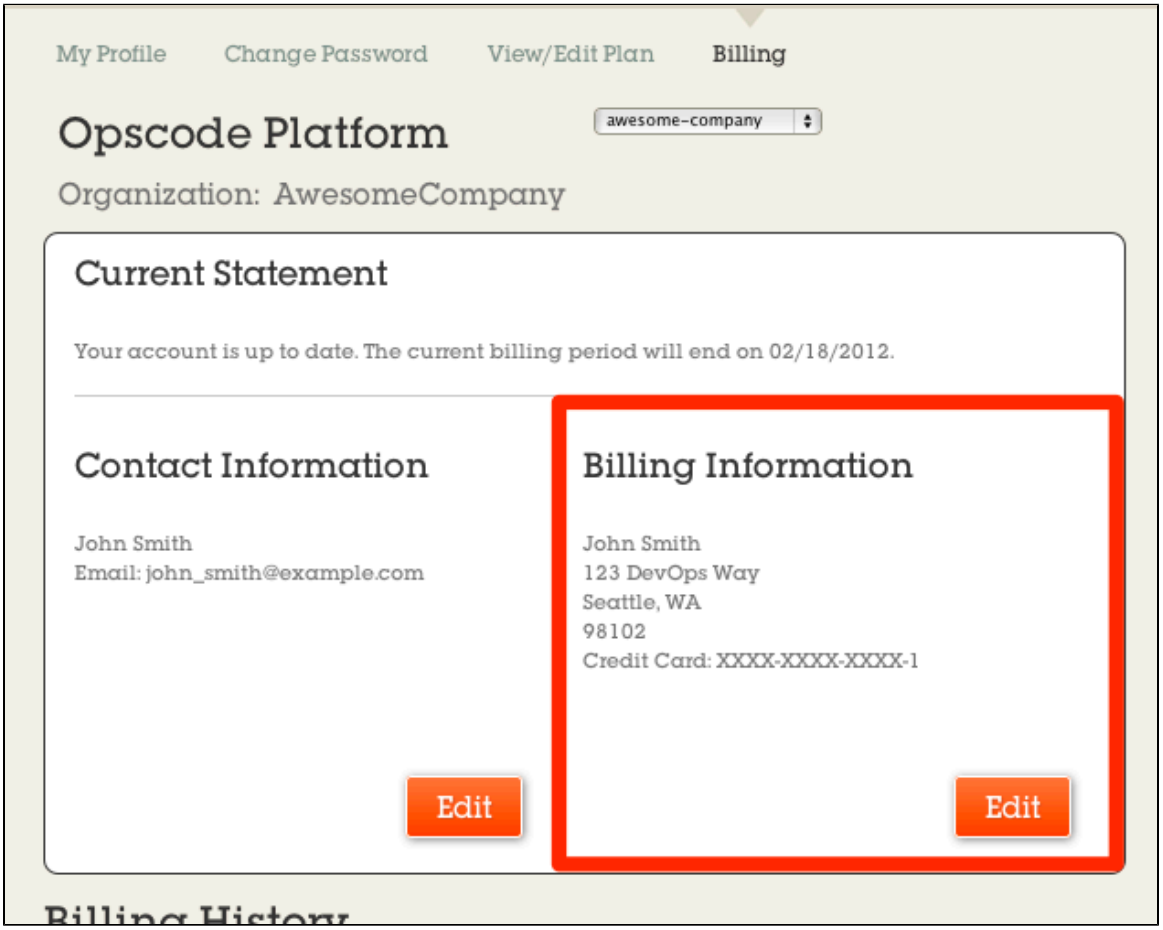
This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Showing Credit Card Info

If you have previously entered your billing information, a summary of that information will appear:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

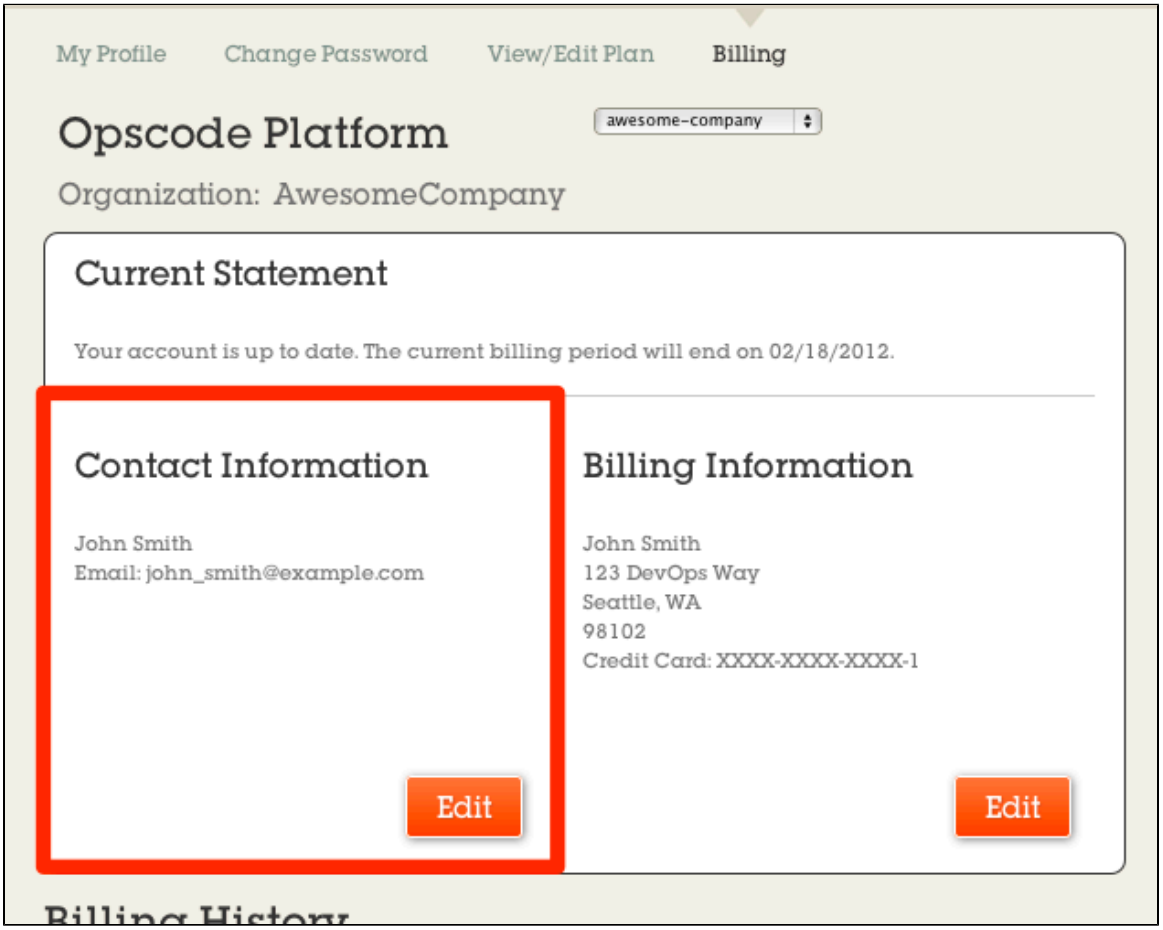


You can also update your billing information by clicking the *Edit* button.

Billing Contact Information

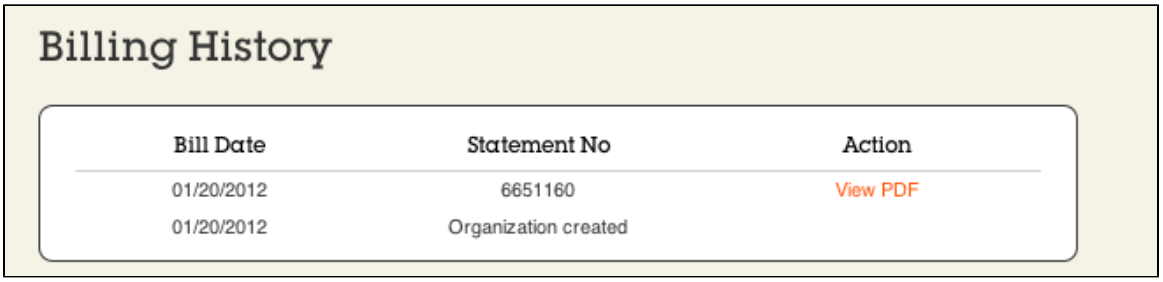
The *Billing Contact Information* is the information that Opscode will use to contact your organization if any problems arise during the billing process. The current billing contact information will appear in the left pane of the Billing information page. You can update this information by clicking the *Edit* button.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Billing History

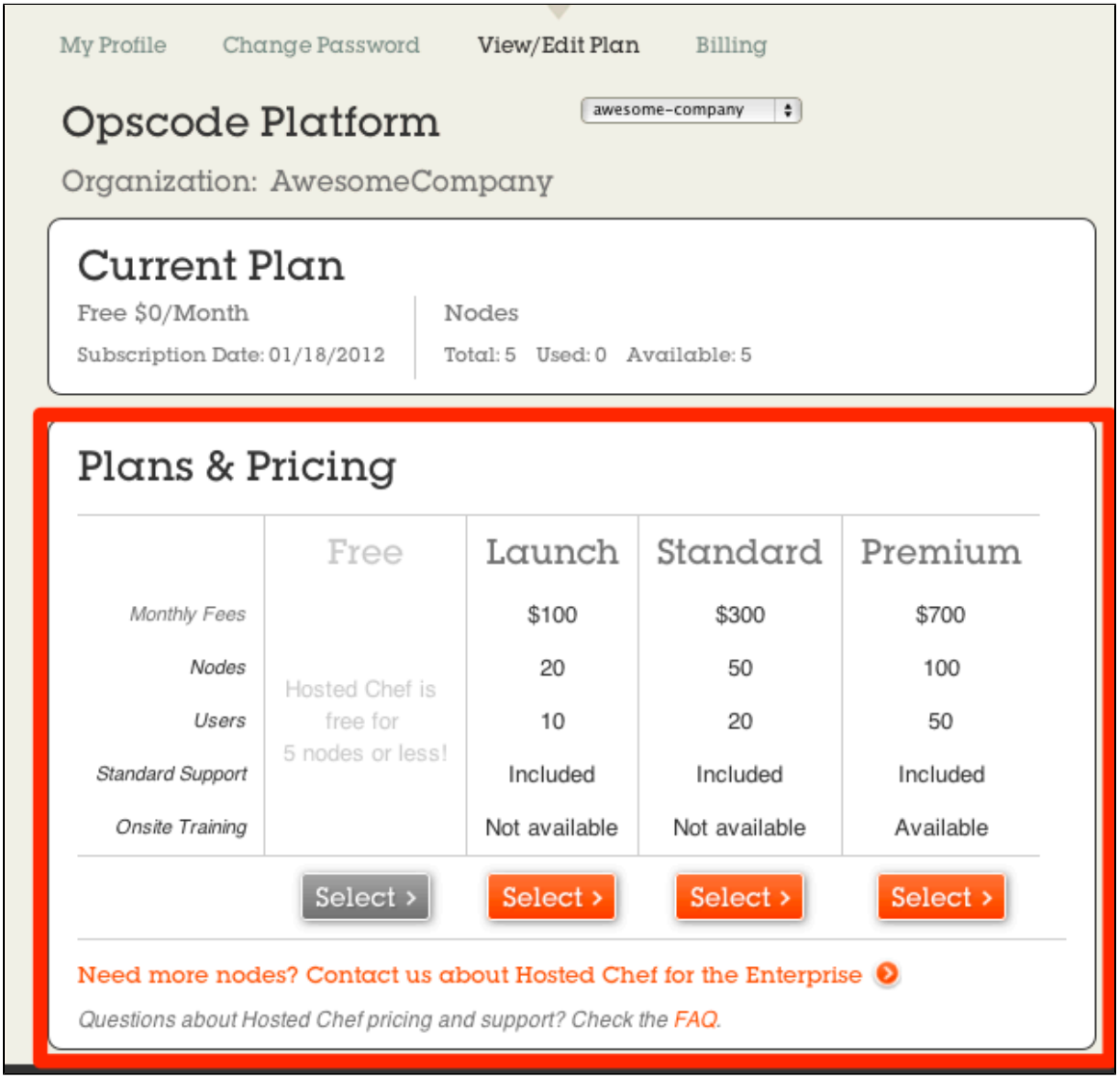
The *Billing History* section of the *Billing Page* allows you to view PDFs of previous statements. You can retrieve the statement for a given period by clicking the "View PDF" button.



Billing Plan

From the *View/Edit Plan* page of the Account Management tool, you can change your organization's subscription to Hosted Chef. Selecting your desired plan will update your subscription and you will be charged according to the billing information currently on file.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



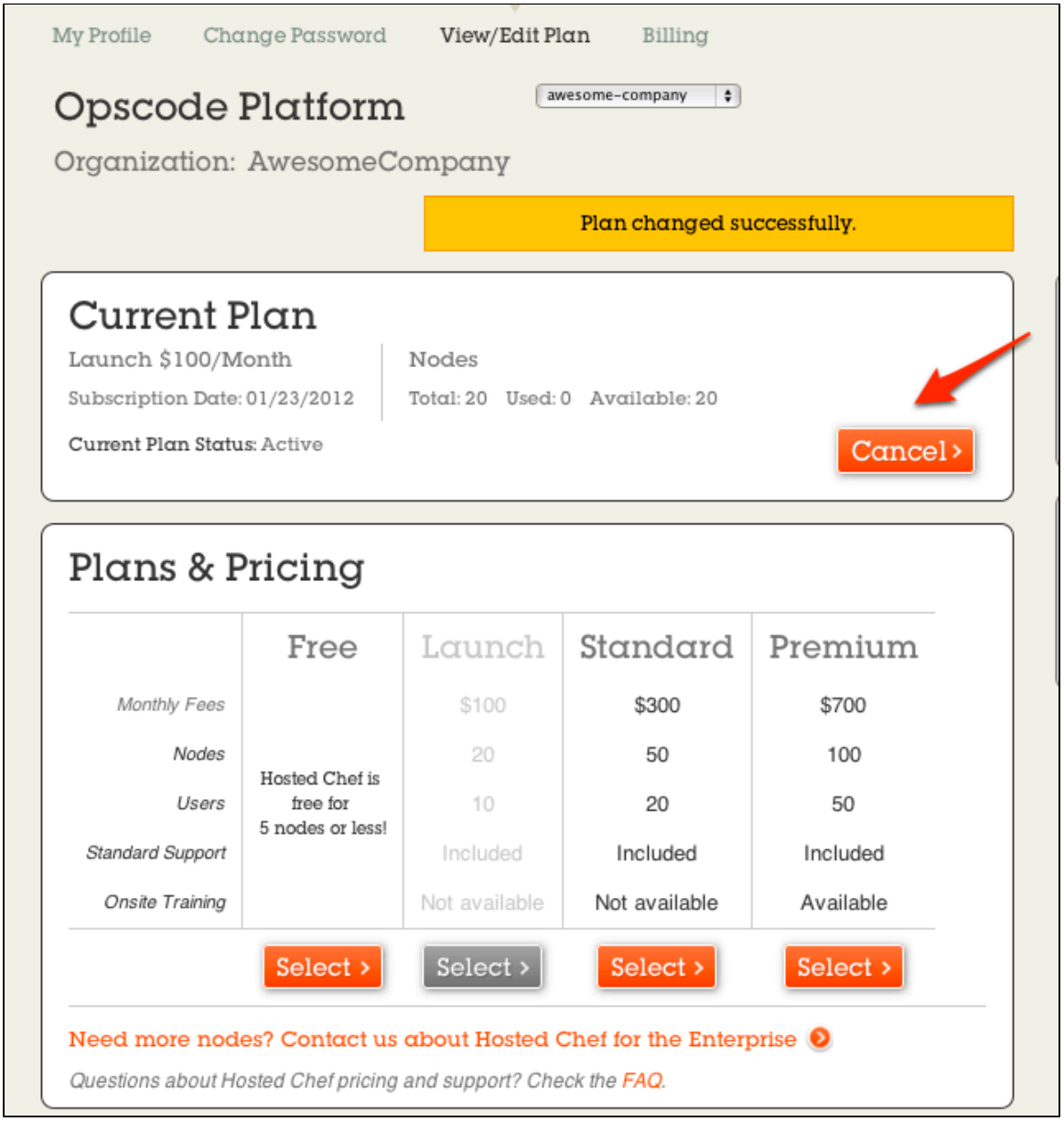
Canceling your Subscription

Be Careful: Canceling your subscription to Hosted Chef will cut off your access to the Hosted Chef API and your data.

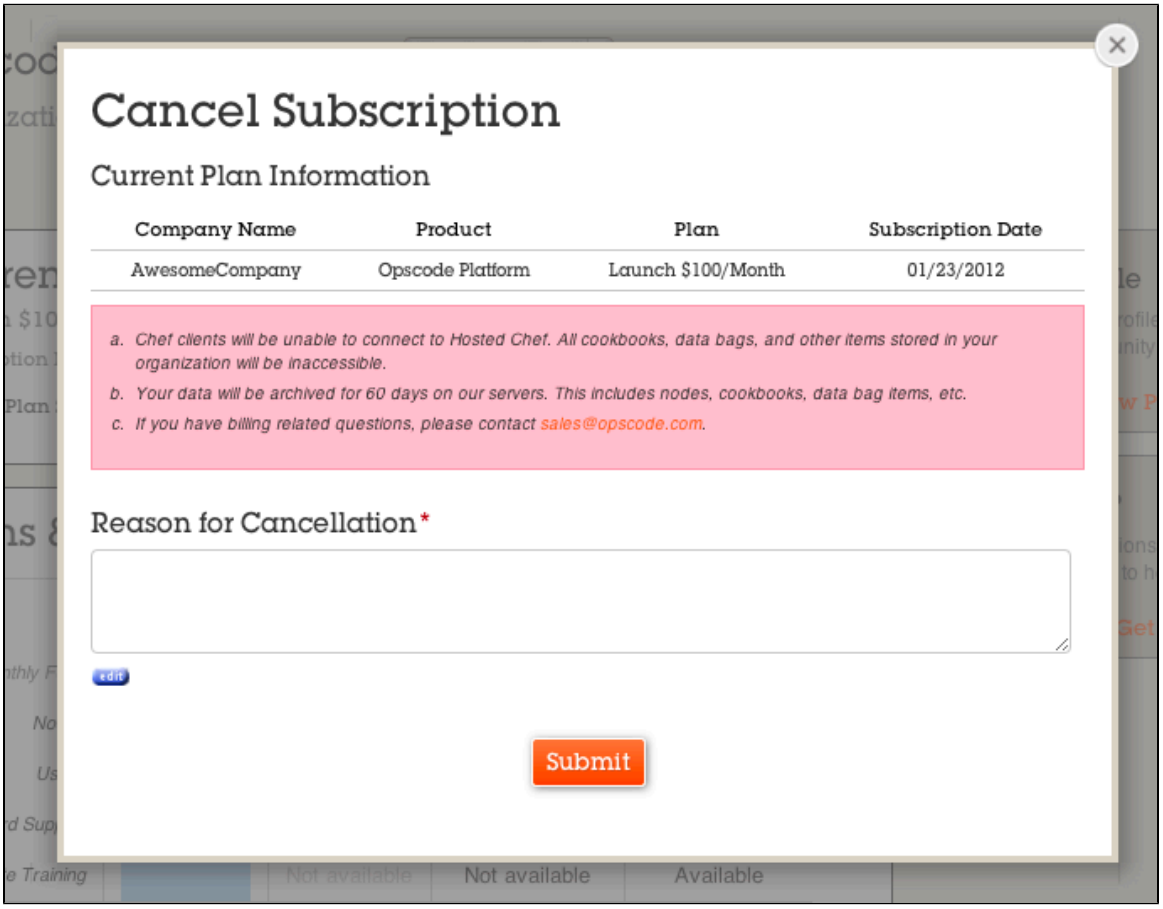
From the *View/Edit Plan* page of the Account Management tool, you can cancel your organization's subscription to Hosted Chef.

1. Select the "Cancel"" button,
2. Carefully read the presented warning,
3. Enter a reason for cancellation, and
4. Press "Submit"

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Managing Users and your Private Key with the Hosted Chef Management Console

Setup an Opscode User and Organization

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing your Opscode User Account

You are able to update your Opscode user account information from the Account Management Tool.

- Edit User Details
- Changing Your Password
- Recovering a Lost Password
- Retrieve Private Key

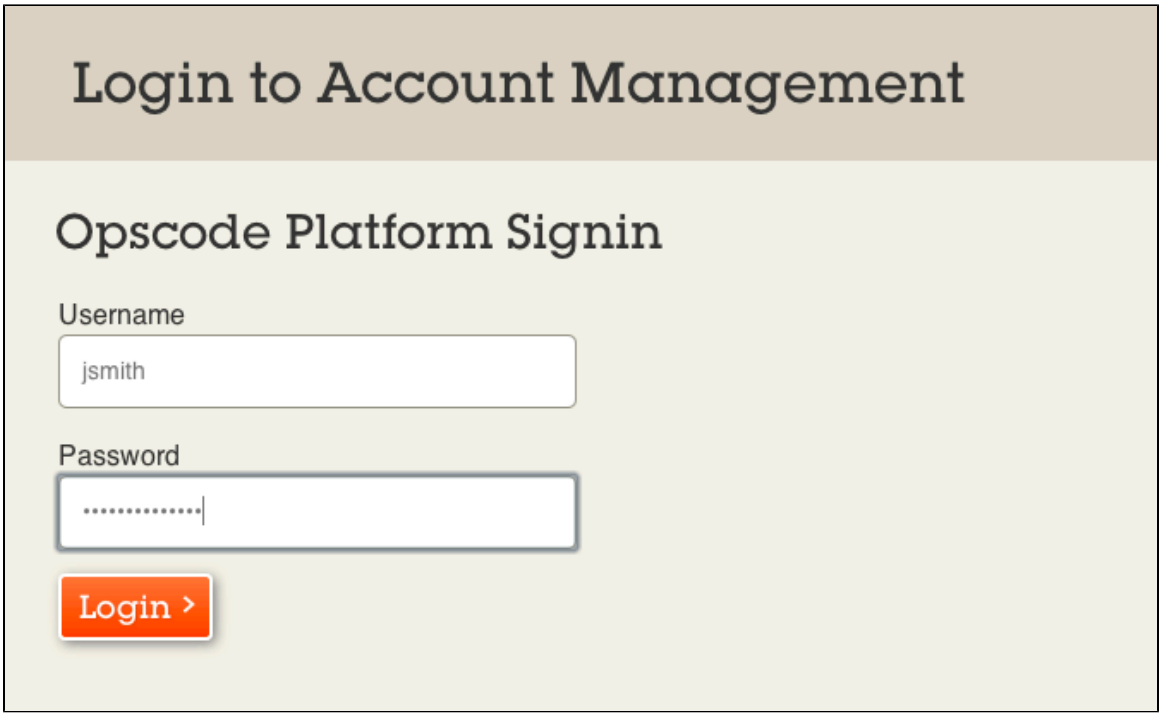
Log into the Account Management Tool

In order to update your account information you'll need to login to the account management tool

1. Go to the Account Management site (https://www.opscode.com/account) and login using your existing username.

This article assumes you have already setup an Opscode user account. If you haven't yet, please follow the Setup Opscode User and Organization guide.

If you're interested in editing user, node or other data in the Management Console instead, return to Hosted Chef Management Console, and make the appropriate selection.



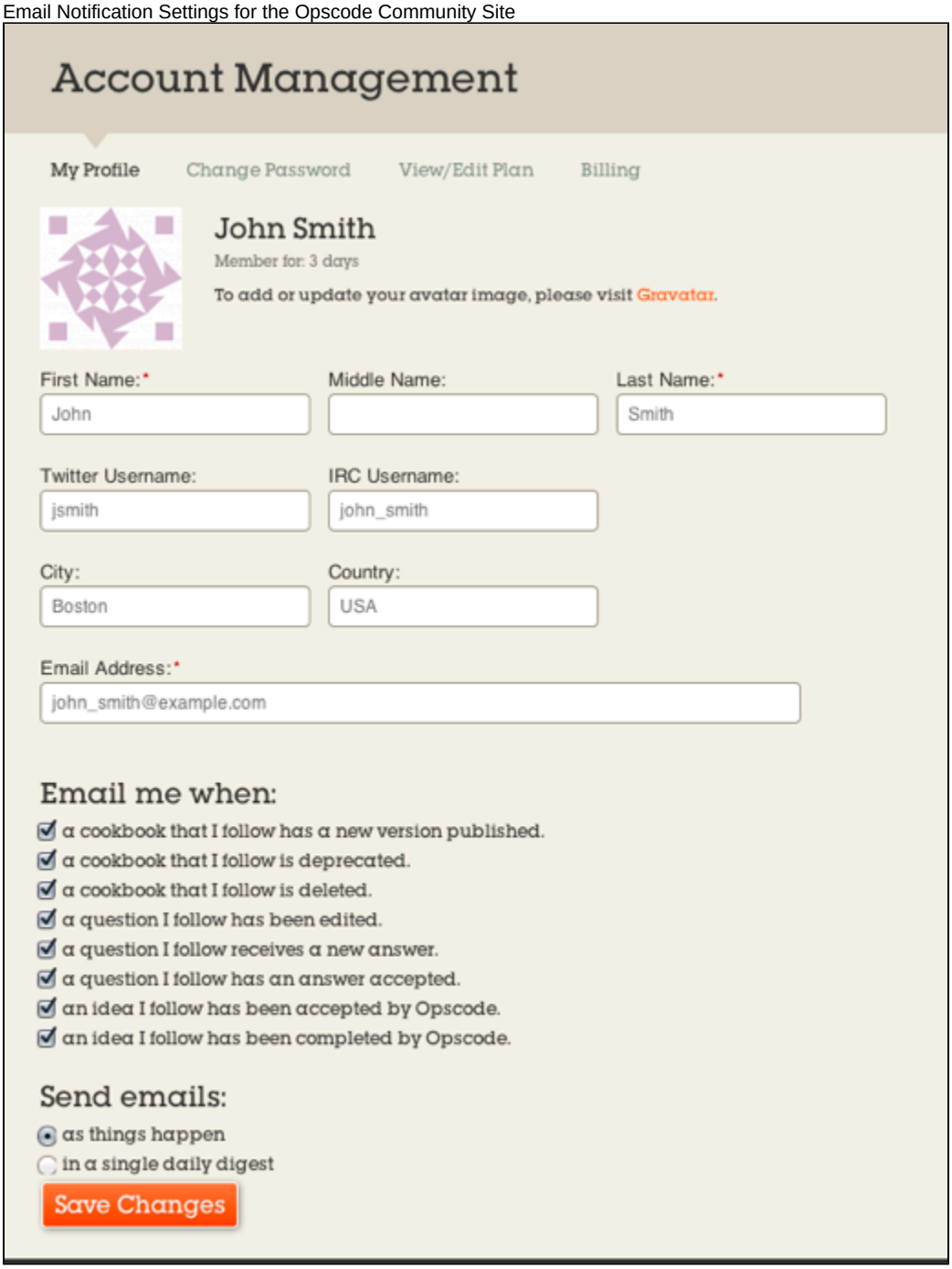
User Details

From the *My Profile* page of the account management tool, you can update the details about your Opscode User account.

1. Go to the Account Management site (https://www.opscode.com/account) and login using your existing username.
2. Go to the *My Profile* page by clicking on *My Profile* in the navigation bar.
3. On the *My Profile* page, you can edit any of the following:
    - First, Middle, and Last name
    - Twitter and IRC Usernames
    - Email Address

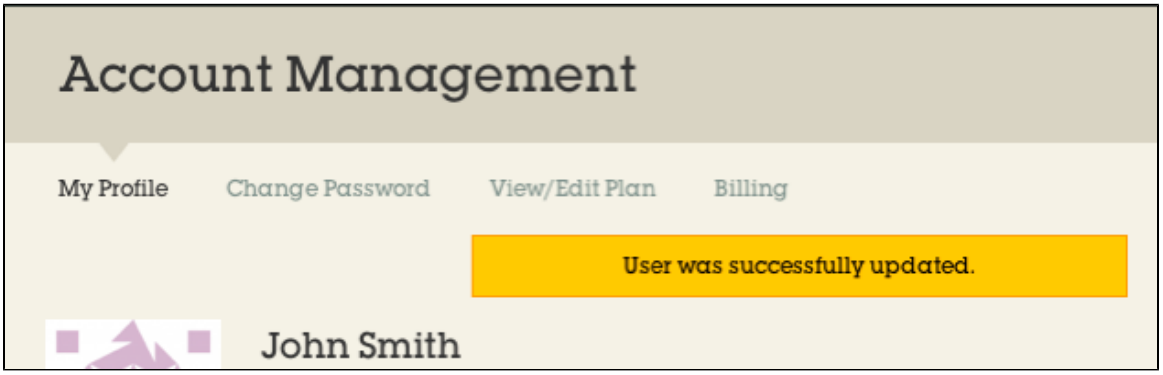
 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- Email Notification Settings for the Opscode Community Site



4. Make the desired changes and press "Save"
5. If your save was successful, you should see the following information:

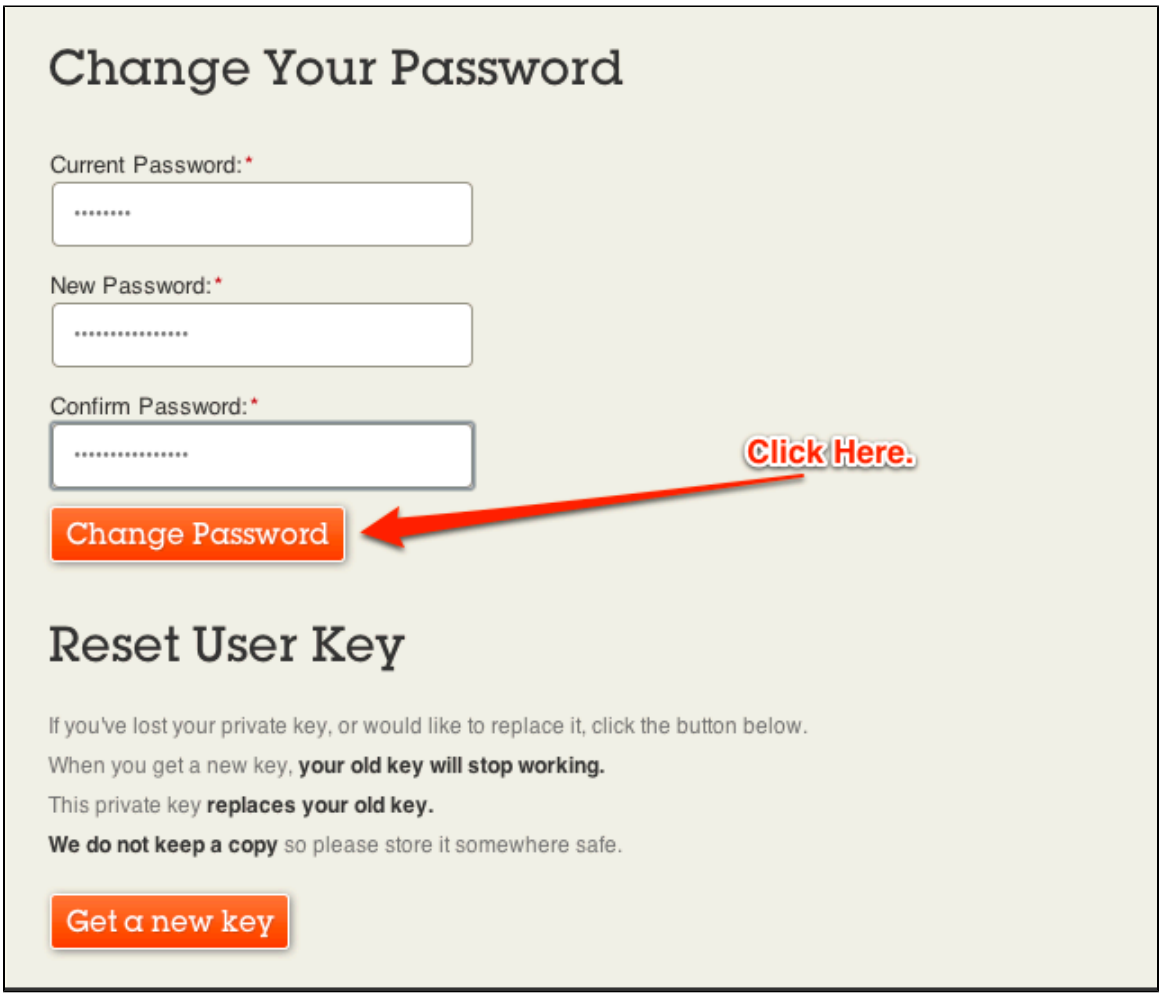
 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Changing your Password

From the *Change Password* page of the Account Management tool, you can reset your user's current password.

1. Go to the Account Management site (https://www.opscode.com/account) and login using your existing username.
2. Go to the *Change Password* page by clicking on *Change Password* in the navigation bar.
3. Enter your old password and your new password.
4. Click "Change Password"



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Recovering a Lost Password

If you lose your password, you can recover it using the email address currently associated with your account.

1. Go to the Account Management site (https://www.opscode.com/account).
2. Click "Recover Password" on the right hand side of the screen.
3. You will be redirected to a password reset page. Enter your email address and press submit.
4. Follow the directions that are sent in the email you will recieve.

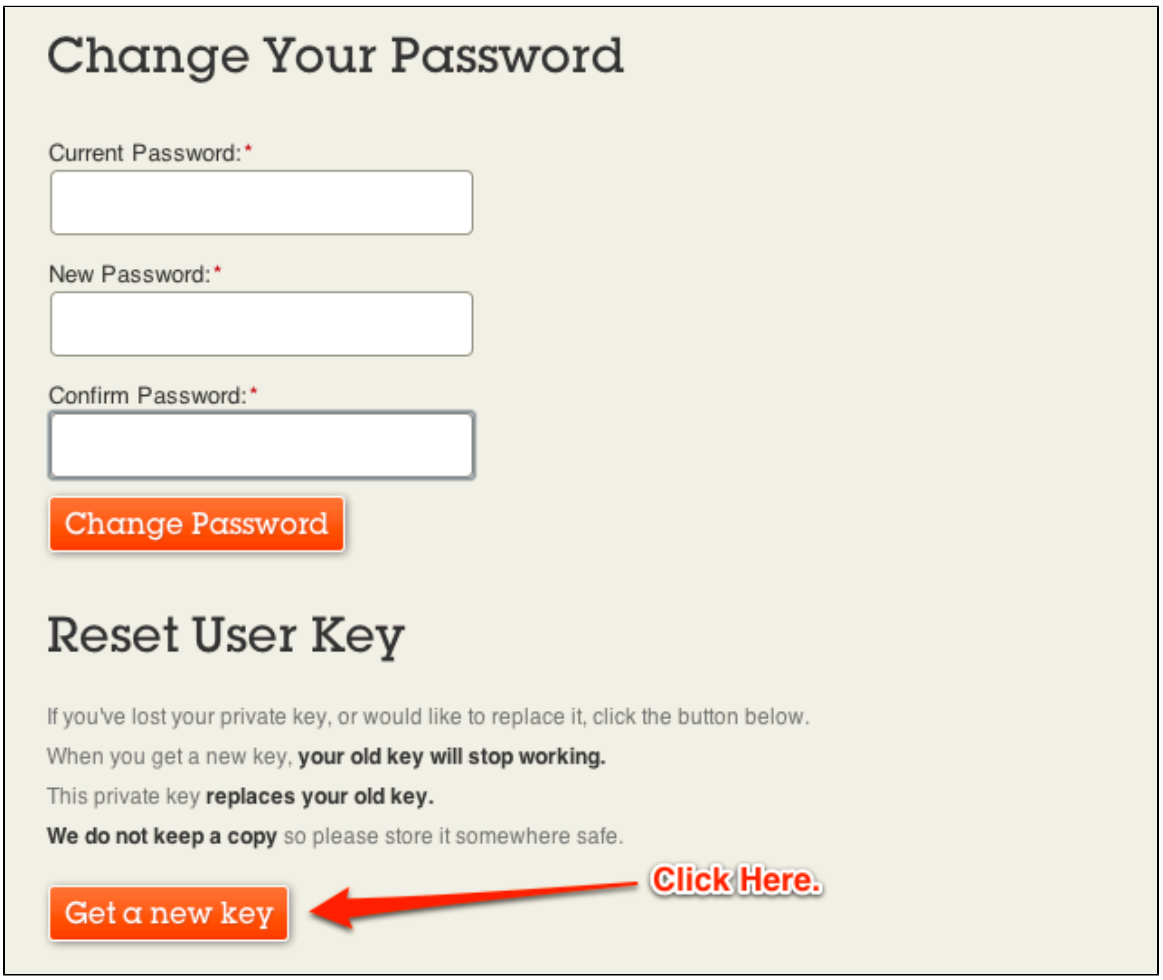


Regenerate Private Key

The *Change Password* section of the Account management tool, also allows you to regenerate your private key. This private key is what you use to make authenticated API requests when working with Staging:Knife.

1. Go to the Account Management site (https://www.opscode.com/account) and login using your existing username.
2. Go to the *Change Password* page by clicking on *Change Password* in the navigation bar.
3. Click "Get a new key"
4. Your new key will now download. Please save this in a safe location.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Managing your Account and Billing Information

Setup an Opscode User and Organization

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Setup an Opscode User and Organization



This page is to provide addition details on using Hosted Chef to setup a user and organization

You should already have Ruby installed on your system.
If you don't, return to the Fast Start Guide and complete that preceding step prior to returning here.

Signup and Account Validation

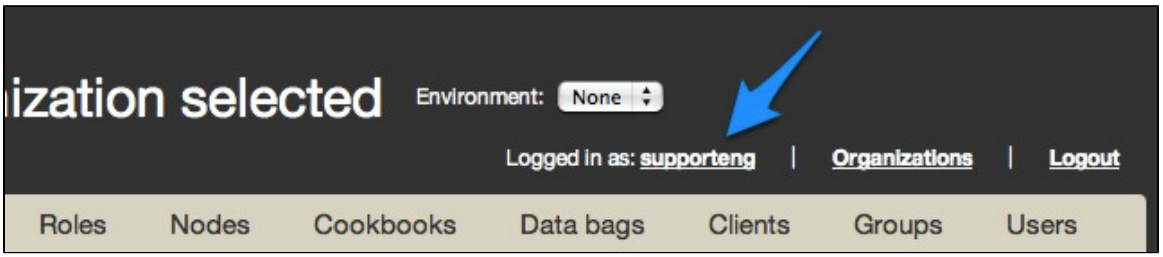
Head over to the Hosted Chef Sign Up page to create your account. You will get an email validation shortly after you sign up.
After you validate your email address with Hosted Chef, you need to do the following:

- Download your User Key, which you will use to interact with Hosted Chef directly.
- Create an organization on Hosted Chef
- Download your Organization Key, which will be used to associate new nodes with your Organization. (Also sometimes called a validation key).
- Download a Knife configuration file, which will configure our command line tools for you.

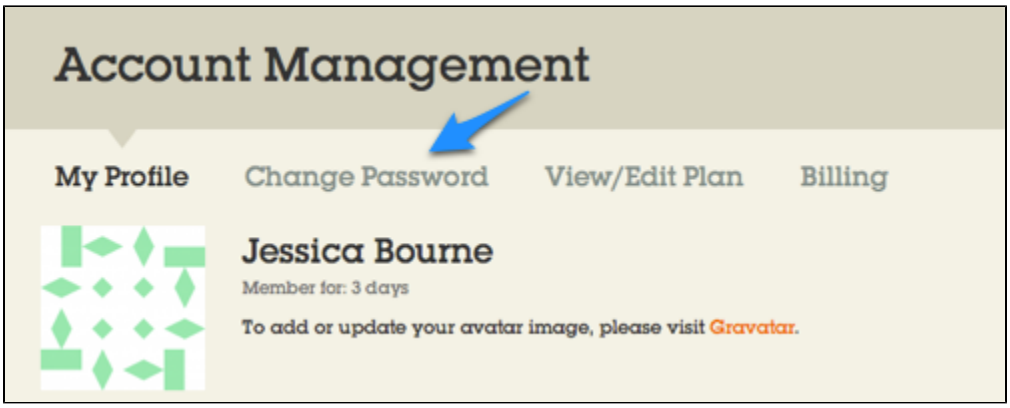
Each of these steps are detailed below. Place all of the files you download in this section in the same directory.

Download Your User Key

Login to the Management Console and navigate to your user profile page:

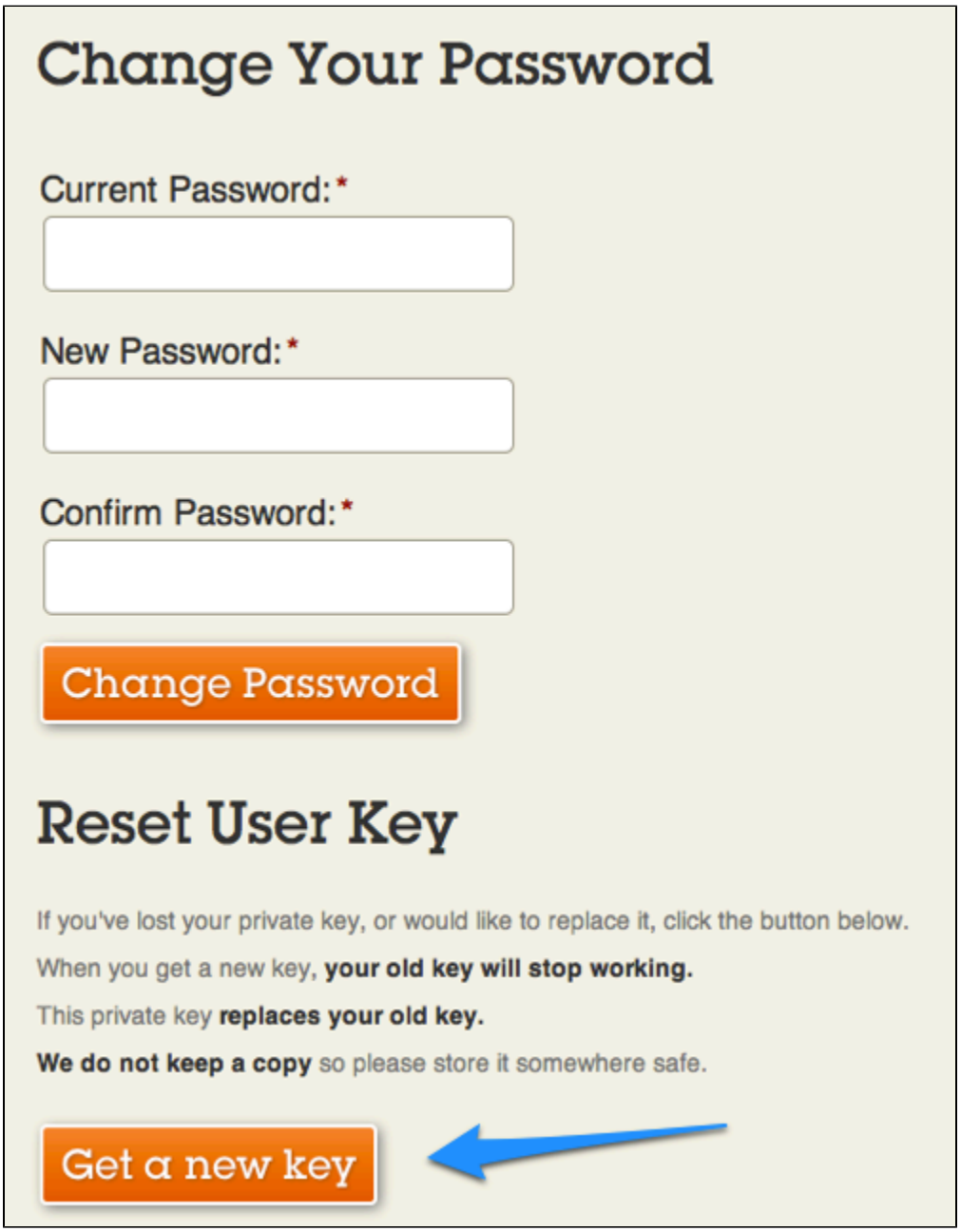


Follow the link to get a new User Key:



Download your new User Key:

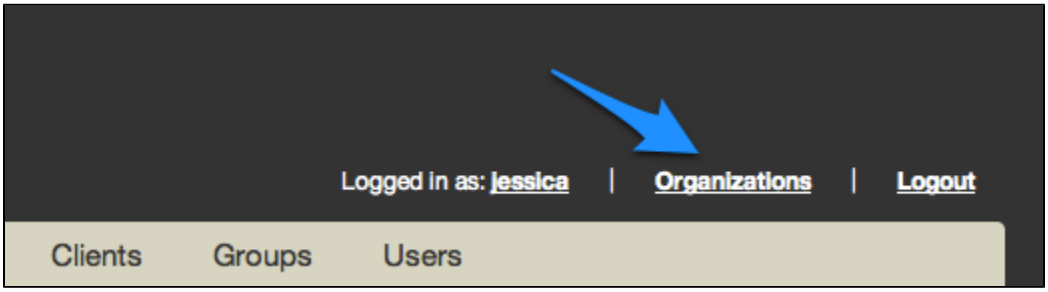
This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Create an Organization

Login to the Management Console and navigate to the organizations page:

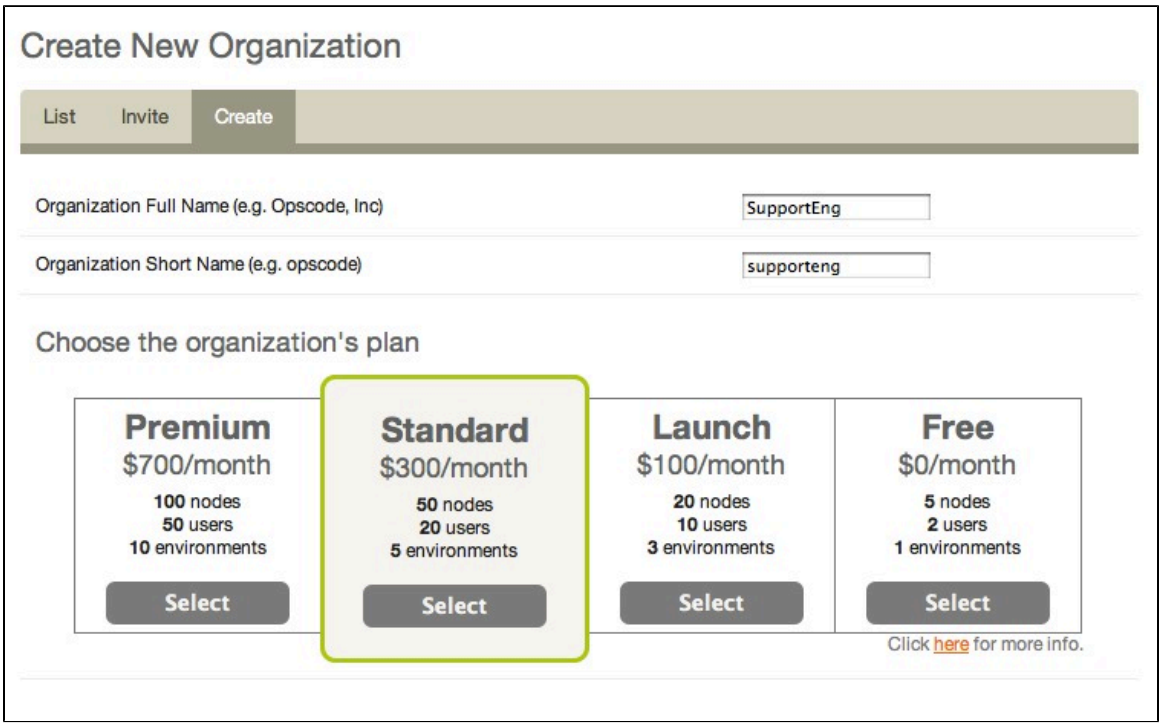
 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Click on the 'create' sub-tab:



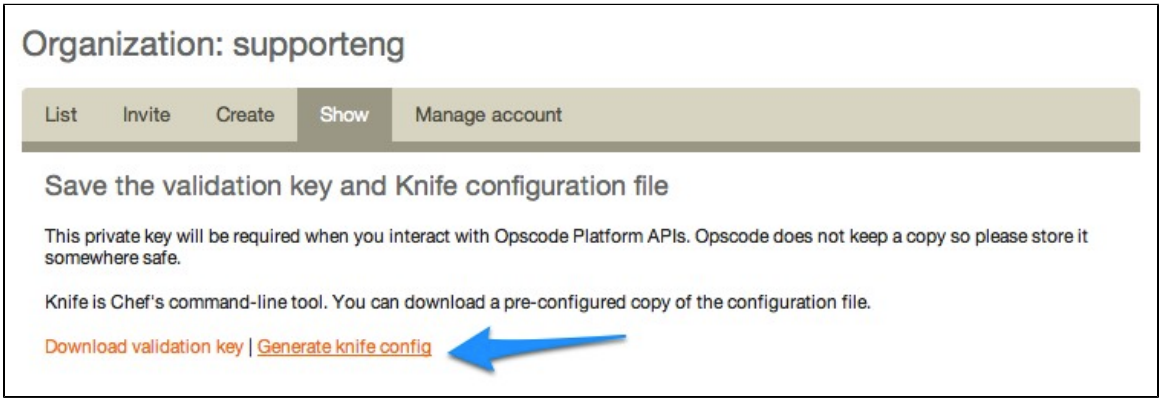
Enter a name for the organization and select a service plan:



Download Your Organization Key and Knife Configuration File

For the organization you would like to manage, download your Organization Key and your Knife Configuration File:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Opscode does not store copies of these keys! Keep them safe!

Managing your Account and Billing Information

Using Search in the Hosted Chef Management Console

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Using Search in the Hosted Chef Management Console

This page explains how to use search:

- Listing Search Indexes
- Searching the Indexes

Listing Search Indexes

More information on searching can be found on the Search page. You can also use the command line tool Knife to search.

- **To follow these directions, first log into the Hosted Chef Management Console with your existing username.** If you have not signed up for an account yet, follow the directions on the Set up Opscode User and Organization page.
- **If you're interested in editing account, nodes or other data in the Management Console instead, return to Hosted Chef Management Console, and make the appropriate selection.**

In order to see the list of Search Indexes:

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click on the *Search* tab on the main navigation menu.

This page will list all of the Indexes you can search within such as Environments, Clients, Roles, Nodes, or Data Bags.

Searching the Indexes

You can search through the Search Indexes including Environments, Clients, Roles, Nodes, or Data Bags by using the "attribute:value" syntax on the search page in the Management Console. More information about this syntax can be found in the Search article.

For example, if you want to search a client whose name is "foobar":

1. Log on to the Hosted Chef Management Console and select an organization to use if you are associated with multiple organizations.

2. Click on the *Search* tab on the main navigation menu.

3. In the text box under the "Client" search index, enter "clientname:foobar" and click on the *Search client* button.



The results will look like this if there is a client named "foobar":



If you leave the search box empty, it returns all the objects of that type. Searching for Environments, Data Bags, Nodes, and Roles work the same way.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Setup an Opscode User and Organization

Shef



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Open Source Chef Server Management Console



The Management Console is Chef Server's web interface.

Nodes, roles, cookbooks, data bags, and API clients can be managed through the Management Console. Search can also be done on the console.

Setting up the Management Console

In order to set up and start the management console, use the chef::bootstrap_server recipe with `webui_enabled`. The management console will be running on port 4040.

Using the Management Console

In order to start using the Management Console, you need to first create a user or change the default password on the "admin" user.

The following articles shows how to use the Management Console in detail:

- Managing Management Console Users
- Managing Nodes through the Management Console
- Managing Roles through the Management Console
- Managing Cookbooks through the Management Console
- Managing Data bags through the Management Console
- Managing API Clients through the Management Console
- Search on the Management Console
- Looking up node status through the Management Console

 Management Console

Looking up node status through the Management Console 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

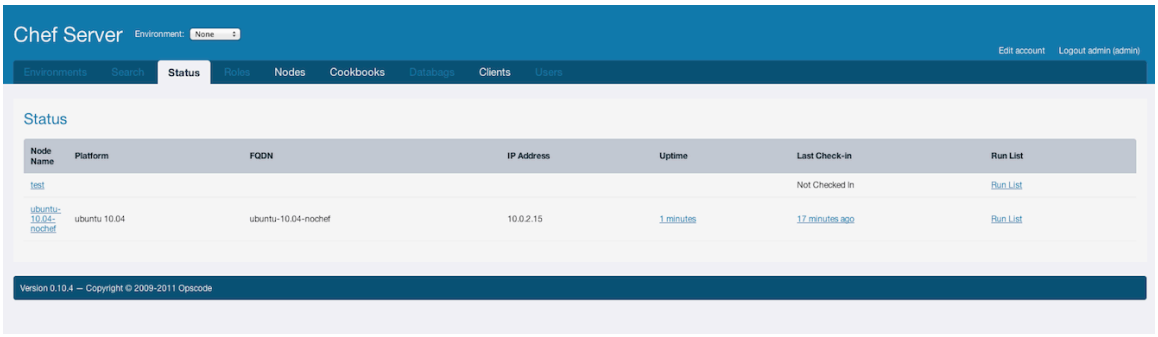
Looking up node status through the Management Console



Node status such as platform, FQDN, IP Address, up time, last check-in time, and run list can be checked on the Management Console.

In order to check node status:

1. Log in to the Management Console.
2. Click `Status` tab.



Open Source Chef Server Management Console

Management Console Users

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Management Console Users

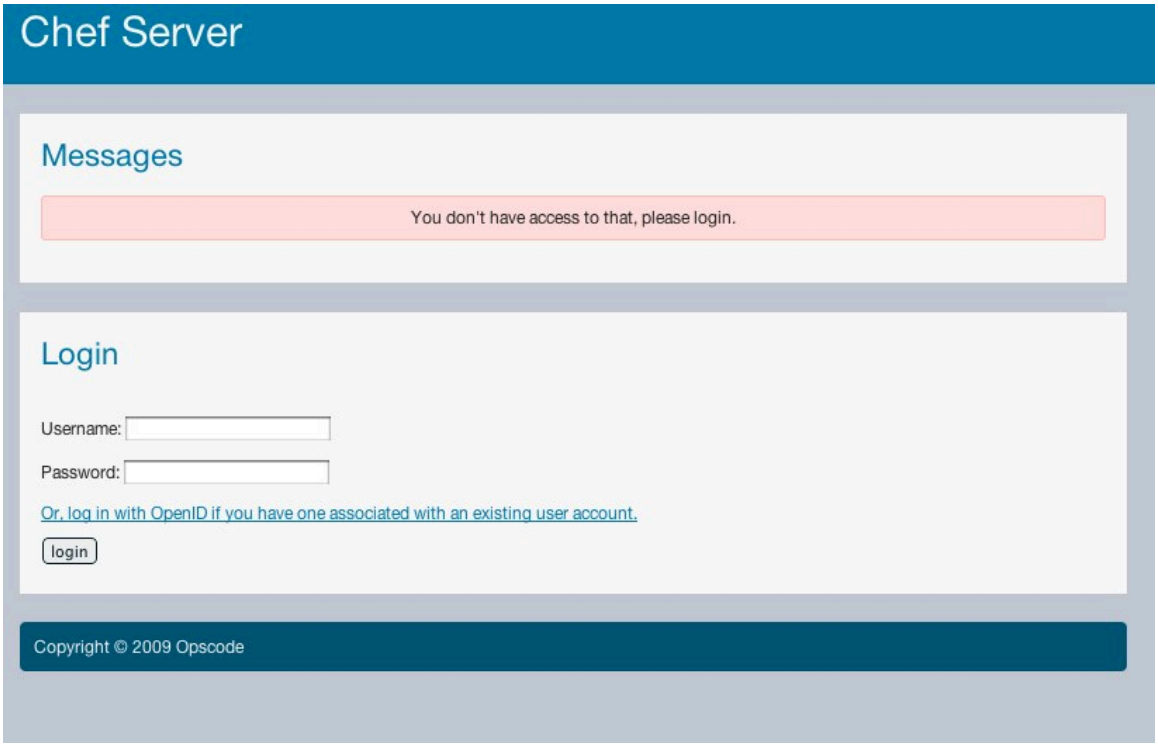


In Chef 0.8.2 and above, Management Console users are maintained separately from API Client and Node credentials .

You can associate an OpenID to your user and use that to authenticate, or authenticate using your user credentials in Chef's CouchDB database.

Logging in For the First Time

The Management Console runs on port 4040 by default. Navigate your browser to http://chef.example.com:4040/ (replacing chef.example.com with the actual hostname of your Chef server, of course). You'll be redirected to the login screen.





Change the Default Password

Default passwords are no better than not having a password. They're evil. They must die. Please change the default password and never use it again.

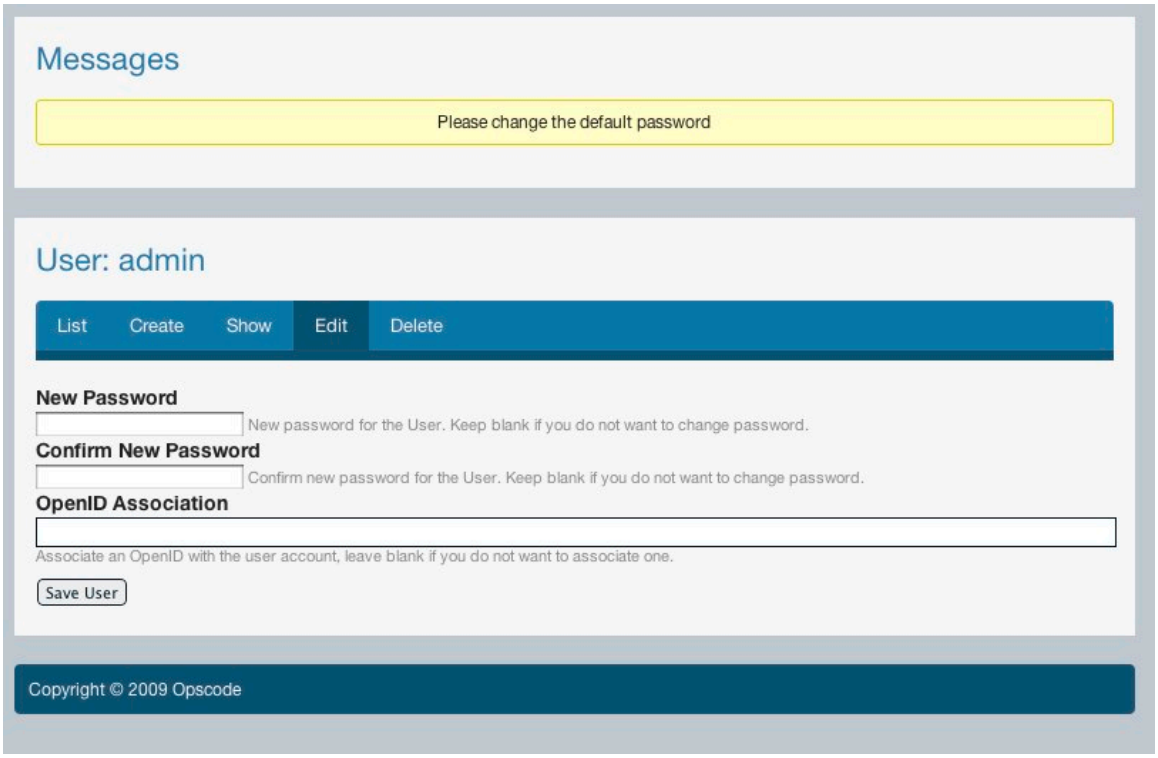
When logging in for the first time, use the default credentials

Username:	admin
Password:	p@ssw0rd1

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

When you log in as the default admin user, you're immediately redirected to the edit page for that user. After changing the default password, you can add add new users or use any other features of the Management Console.

Using the Bootstrap Recipe
When using the bootstrap::server recipe, this is actually a randomly generated password, and set in `/etc/chef/server.rb` through the template, with the setting `web_ui_admin_default_password`. This can be set to a different value by setting `webui_admin_password` in the chef.json.



Managing Users with the Web Interface

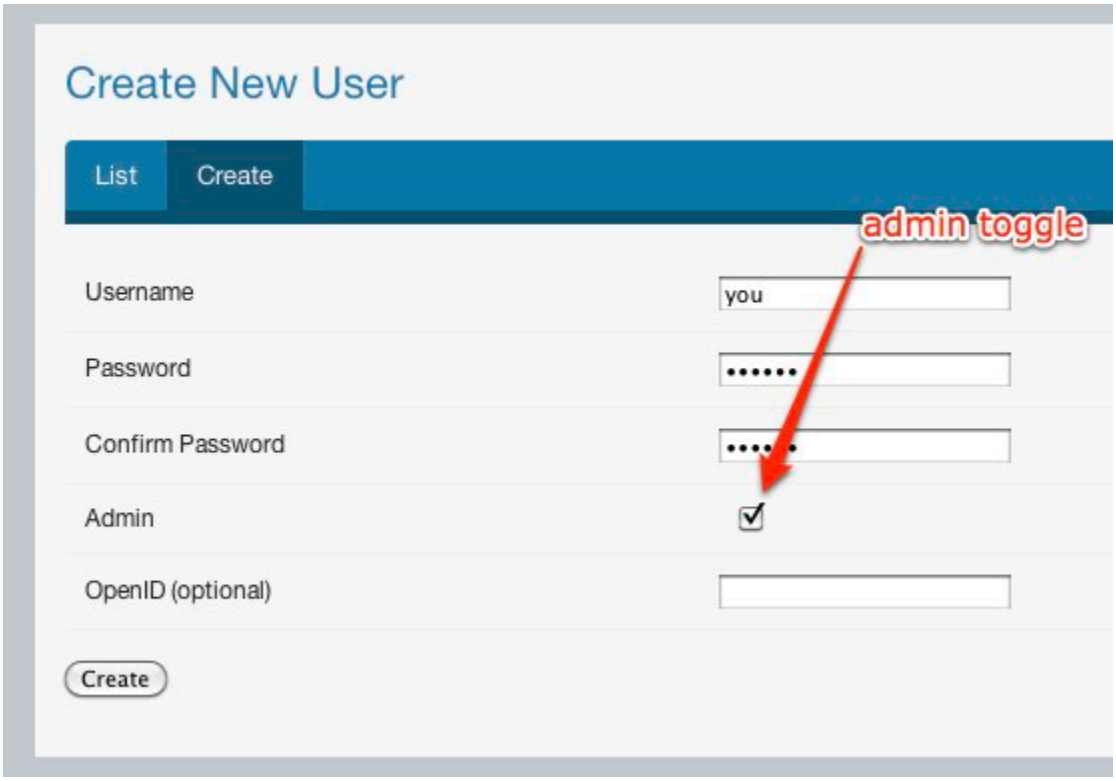
Users are managed from the users tab:



Creating New Users

To create a new user, click "create" on the Users page, then fill in the user's name and password. Tick the box if the new user will be an administrator, then hit the "create" button.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Deleting Users

To delete a user, click the link for "delete" next to the user's name in the list. Chef will ask if you're sure about deleting the user. Click "ok" to confirm.



Editing Users

To edit a user, click the edit link next to the user's name in the list. You'll see the same edit form you saw when you first logged in as the "admin" user.



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 Looking up node status through the Management Console

Managing API Clients through the Management Console 

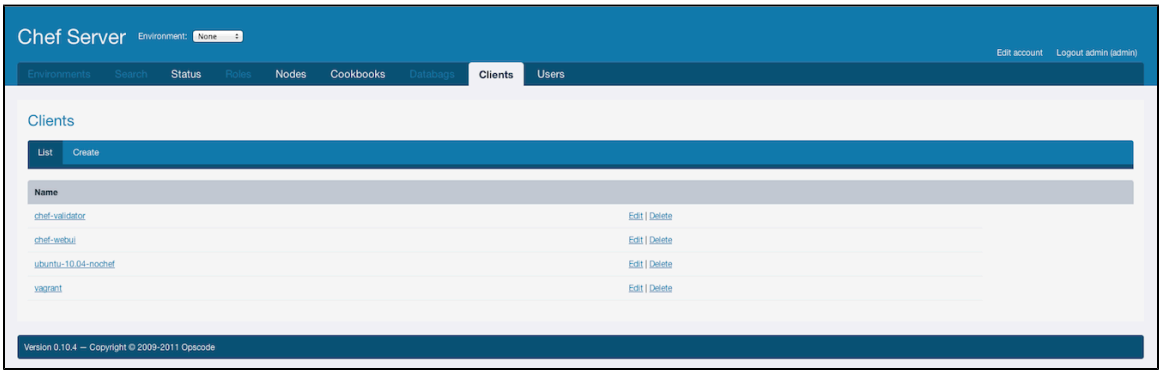
 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing API Clients through the Management Console

The Management Console allows users to create, read, edit, and delete API clients.

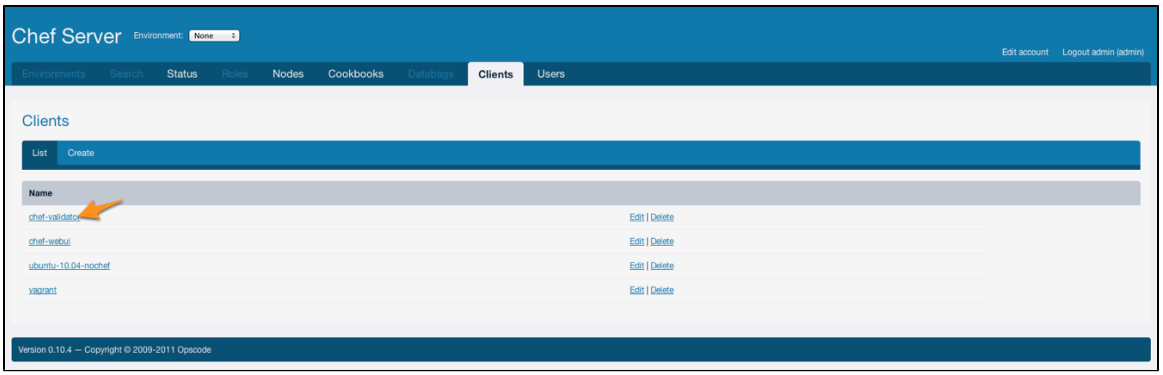
Listing Clients

1. Log on to the Management Console.
2. Click `Clients`.

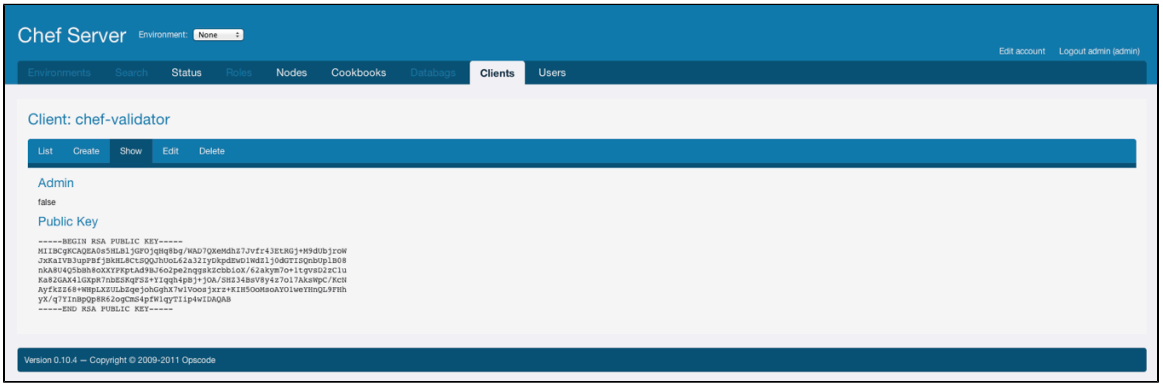


Viewing a Specific Client

1. Log on to the Management Console and click the `Clients` tab.
2. Click on the link of the name of the client you want to view.

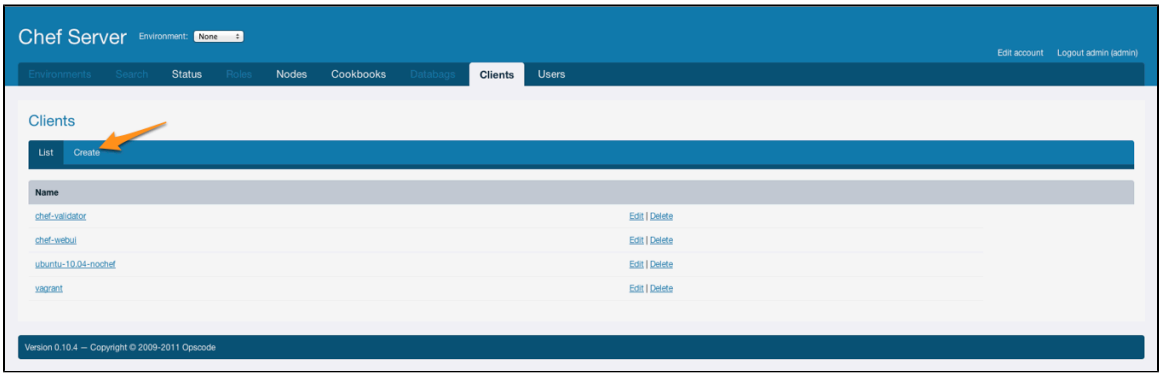


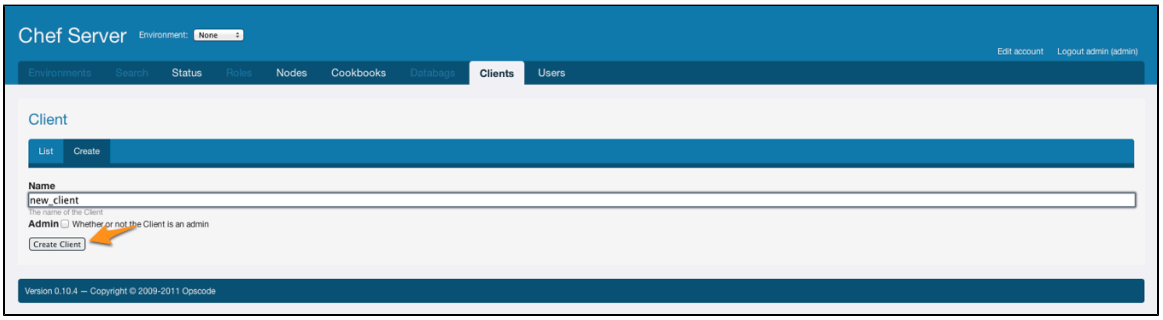
 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



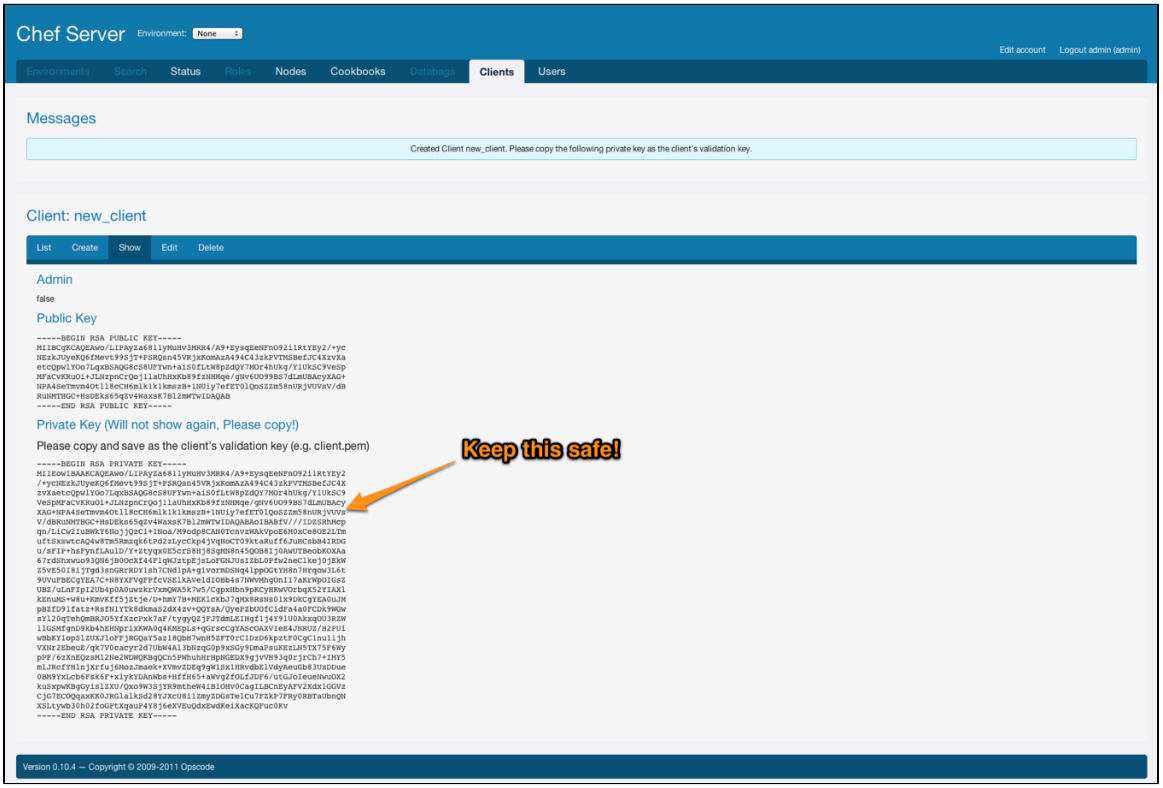
Creating a Client

1. Log on to the Management Console and click the `Clients` tab.
2. Click `Create` in the sub-navigation menu.
3. Enter a name, and select whether or not you want the new client to be an admin.
4. Click `Create Client`
5. Save the private key somewhere safe.



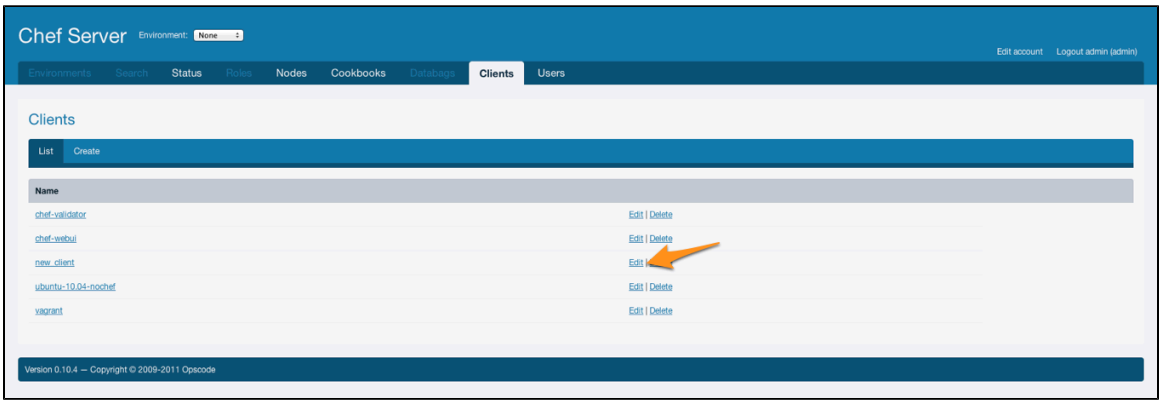


 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

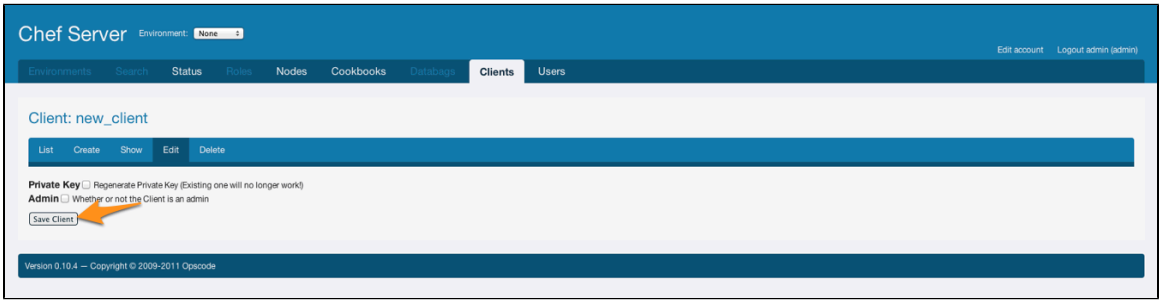


Editing a Client

1. Log on to the Management Console and click the `Clients` tab.
2. Next to the name of the client you want to edit, click the `Edit` link.
3. You are allowed to regenerate a private key or change its admin status. Click `Save Client` to save your change.
4. If you chose to regenerate the private key, copy and save it somewhere safe.

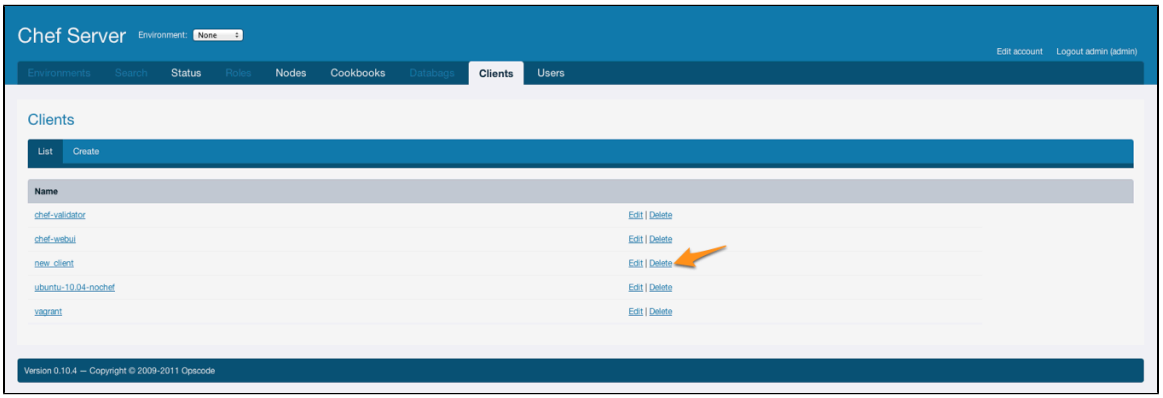


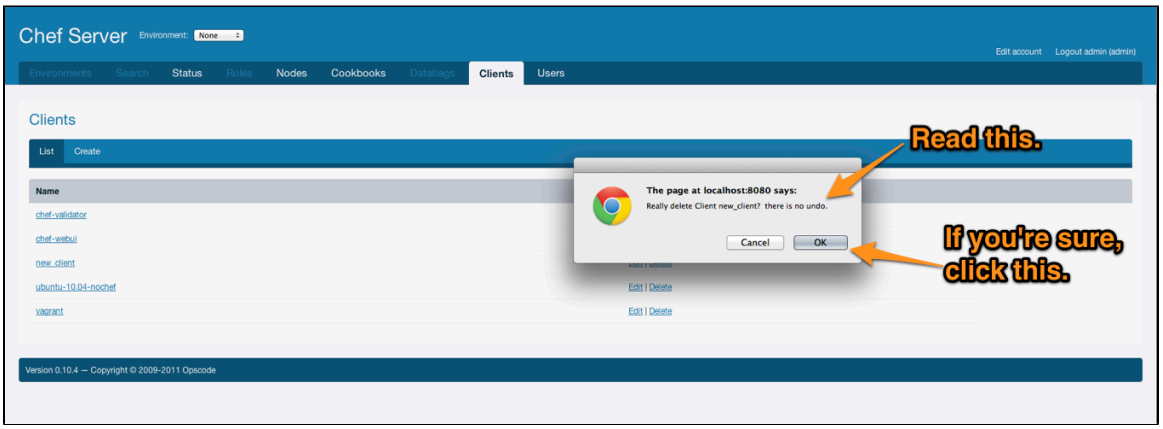
 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Deleting a Client

1. Log on to the Management Console and click the `Clients` tab.
2. Next to the name of the client you want to delete, click the `Delete` link. Or, click on the name of the client and click `Delete` on the sub-navigation menu. Click `OK` on the confirmation message box.





Management Console Users

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Cookbooks through the Management Console



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Cookbooks through the Management Console

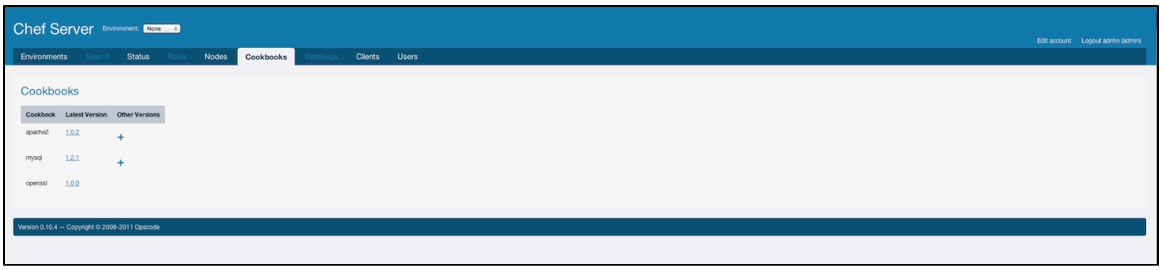
The Management Console allows users to list and view the content of cookbooks.

Listing Cookbooks

In order to list cookbooks:

1. Log on to the Management Console.

2. Click `Cookbooks` on the main navigation menu.



Viewing the Contents of a Specific Cookbook

To view the contents of a specific cookbook:

1. Log on to the Management Console.

2. Click `Cookbooks` on the main navigation menu.

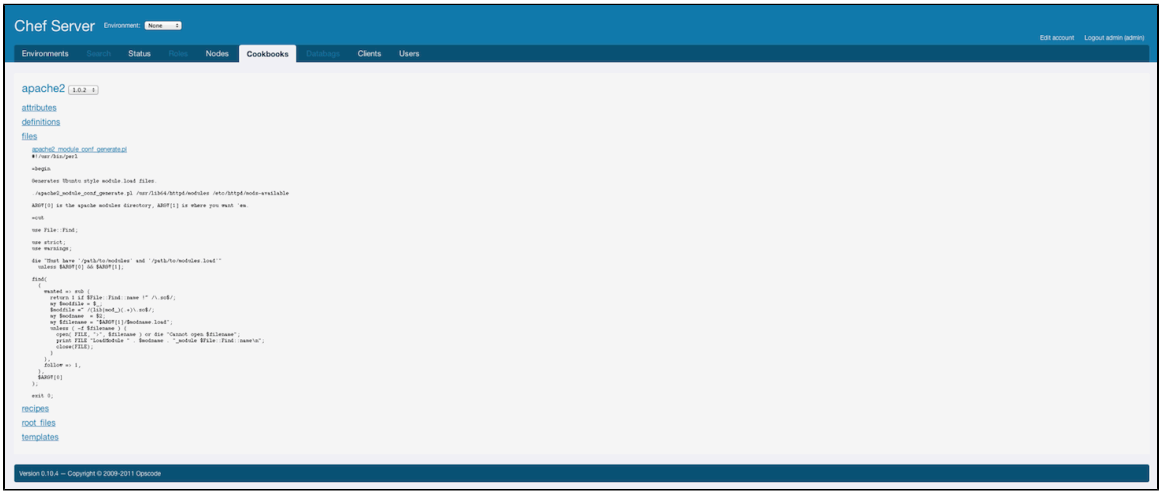
3. Click the link of the cookbook you want to view.

Note: This may take longer for very large cookbooks.

Click on Library Files, Attribute Files, Definition Files, Recipe Files, or Template Files to view the contents of the cookbook. If you do not see one of these links, the cookbook does not have files of that type.



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Managing API Clients through the Management Console

Managing Data bags through the Management Console

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Data bags through the Management Console

The Management Console allows users to create, read, update, and delete data bags and data bag items.

Listing Data Bags

To list data bags:

1. Log on to the Management Console.

2. Click `Databags` on the main navigation menu.

Viewing or Editing a Specific Data Bag

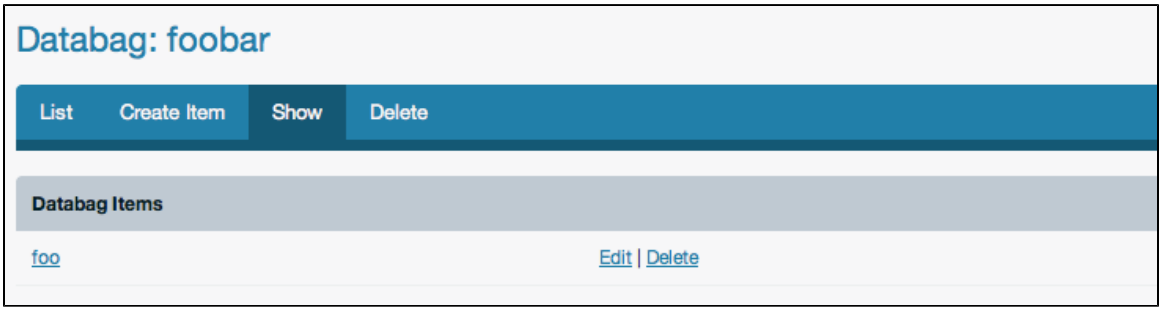
To view or edit a specific data bag:

1. Log on to the Management Console.

2. Click `Databags` on the main navigation menu.

3. Click on the specific data bag you want to view or edit.

4. On the following page, you can create, edit, and delete items in the data bag. For details, refer to the sections below.



Deleting a Data Bag

To delete a data bag:

1. Log on to the Management Console.

2. Click `Databags` on the main navigation menu.

3. Click the `Delete` link next to the name of the data bag you want to delete, and click `OK` on the confirmation message box. Or, click the link of the data bag to view its details, and click `Delete` on the sub navigation menu.

Creating Data Bag Items

To create a data bag item in a specific data bag:

1. Repeat the 4 steps in the "Viewing or Editing a Specific Data Bag" section.

2. Click `Create Item` on the sub navigation bar. You will see a page with a JSON editor.

3. Click the Source tab in the JSON editor to edit the data bag item directly, or use the tree view on the left and the Editor tab to add and edit keys and values. You should not delete the "id" key.

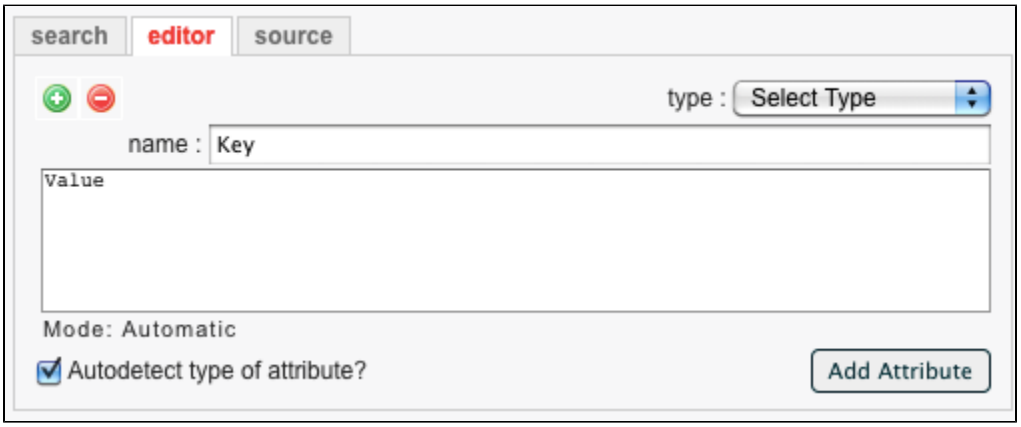
 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Using the Editor tab

1. Expand the Data Attributes tree on the left, click "id", and enter an value for the name of the data bag item, and then click "Save Attribute".

2. Click "json" from the Data Attributes tree on the left, and click the Editor tab on the right, and then click the green add button under the Editor tab, enter a name as the key and a body as the value and click "Add Attribute". The following figures show the Data Attribute tree and the JSON editor.





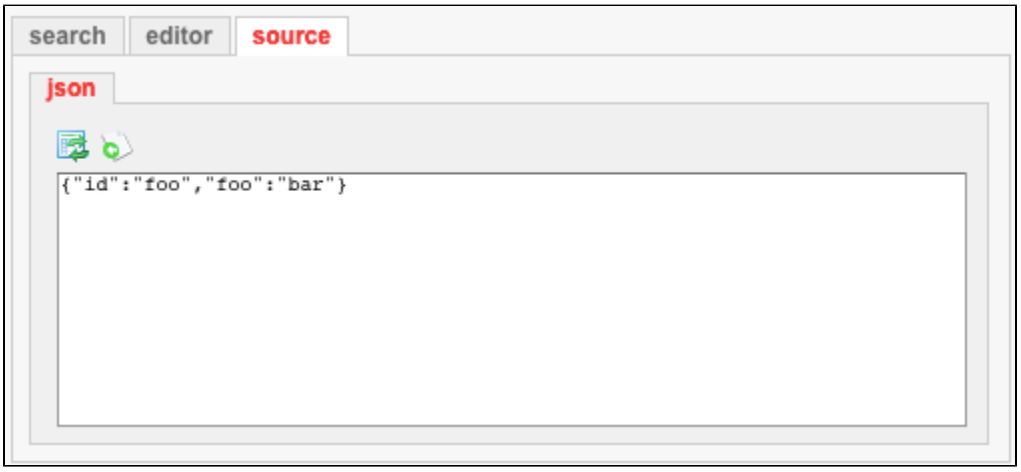
3. Click "Create Databag Item" to save the data bag item.

Using the Source tab

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

1. Enter the JSON body of your data bag item in the editor, and click the "Load JSON from source" button .

2. Click Create Databag Item button to save the data bag item. Here's an example.



Editing Data Bag Items

To edit a data bag item in a specific data bag:

1. Repeat the 4 steps in the "Viewing or Editing a Specific Data Bag" section.

2. Click the `Edit` link next to the name of the data bag item you want to edit.

3. Follow step 3 in the "Creating Data Bag Items" section, except that in the last step click `Edit Databag Item` button instead of `Create Databag Item` button.

Deleting Data Bag Items

To delete a data bag item in a specific data bag:

1. Repeat the 4 steps in the "Viewing or Editing a Specific Data Bag" section.

2. Click the `Delete` link next to the name of the data bag item you want to delete, and click `OK` on the confirmation message box. Or, click the link of the data bag item to view its details, and click `Delete` on the sub navigation menu.

Managing Cookbooks through the Management Console

Managing Environments through the Management Console

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Environments through the Management Console

Chef 0.10.0+ only feature
Environments feature is only available in (and after) Chef 0.10.0.

The Management Console allows users to create, read, update, delete environments, and to select a specific environment to view nodes and cookbooks in that environment.

Selecting an environment

On the header of the Management Console, there is an Environment drop down box where you can select either "none" or a specific environment.

If "none" is selected, the Management Console shows everything when you view nodes and cookbooks; if a specific environment is selected, the Management Console only shows the available objects in that environment for nodes and cookbooks. For cookbooks, if you do not define any restriction in the environment, the Management Console shows all cookbooks.

When viewing a specific role, the per environment run list for the selected environment shows up; if "none" or "_default" is selected in the Environment drop down box, or there is no environment specific run list for the selected environment, the role page shows the "default" run list.

Objects other than nodes, cookbooks, and roles are not affected by the environment selection.

Listing Environments

The List page under Environments tab shows all existing Environments. Clicking the link of the Environment name allows you to view the Environment. You can also select the Environment by clicking "Select" next to the name; doing so is equivalent to selecting an environment from the drop down box on the top of the page.

Viewing an Environment

Click the name of the Environment on the List page under Environments allows you to view the Environment. You are able to see the description, cookbook version constraints, and attributes defined in the Environment you are viewing.

Creating an Environment

By clicking the Create tab under Environments, you are able to create a new Environment. In the Create form you can specify a name (required), description, cookbook version constraints, and attributes.

Deleting an Environment

An Environment can be deleted by clicking the "delete" link on the List page next to the name of the Environment; or by clicking the "Delete" tab on the Show page when you view a specific Environment.

 Managing Data bags through the Management Console

Managing Nodes through the Management Console 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Nodes through the Management Console

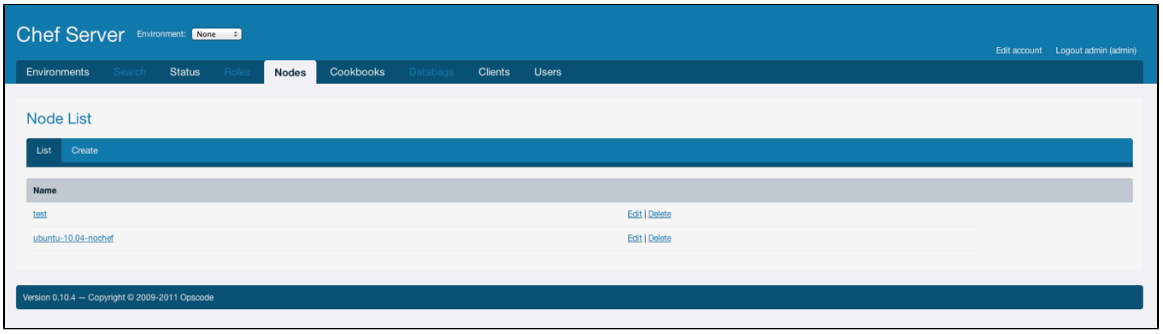
The Management Console allows users to create, read, update, and delete nodes.

Listing Nodes

In order to list nodes:

1. Log on to the Management Console.

2. Click Nodes on the main navigation menu.



Viewing a Node

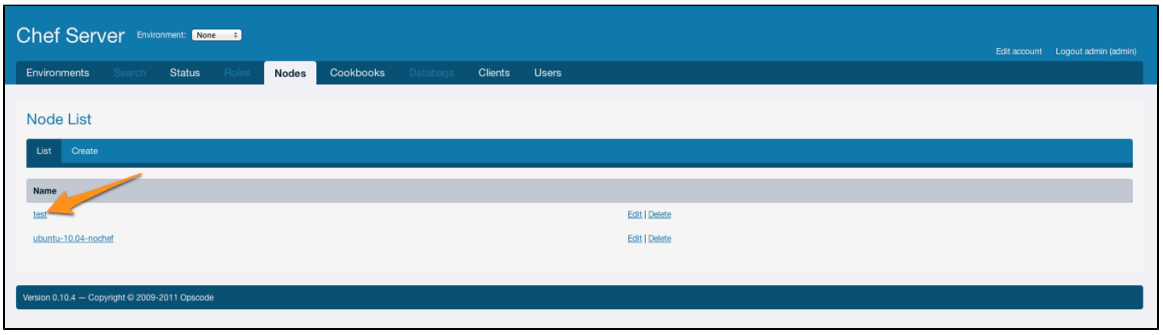
Viewing a node shows the recipes and roles in the node's run list, as well as the node's attributes.

In order to view a specific node:

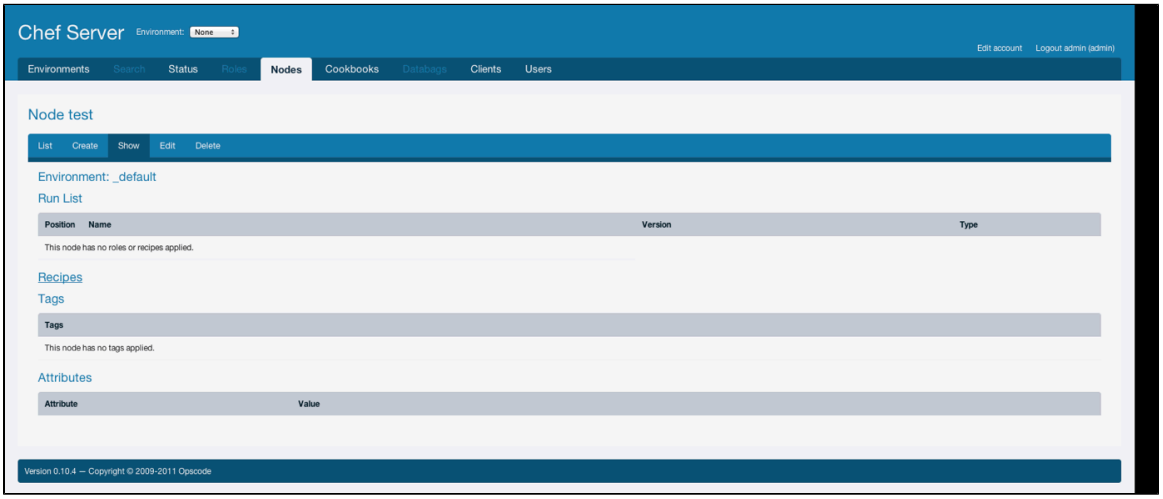
1. Login to the Management Console and navigate to the Nodes tab.

2. Click on the node you want to view.

On this page you can see the roles and recipes in the node's run list, as well as an expandable tree view of the JSON attributes on the node.



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Creating a Node

The node creation page lists all available recipes and roles, and allows you to include these in the run list of the node you are creating. There is also a JSON editor for editing, adding and removing the node's attributes.

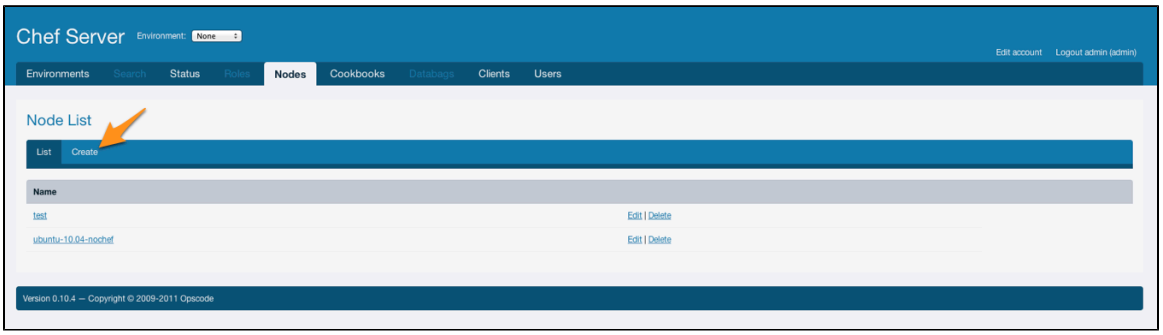
In order to create a node:

1. Login to the Management Console and navigate to the Nodes tab.

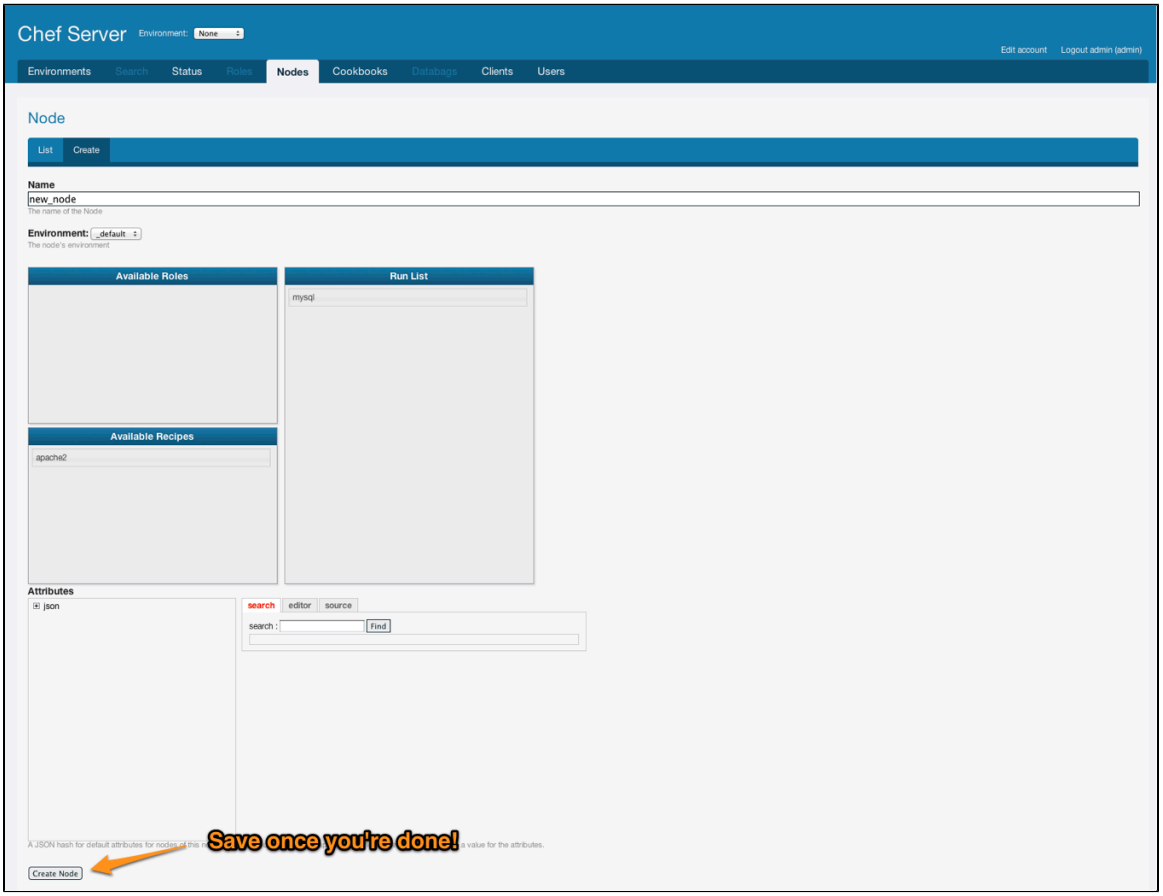
2. Click Create on the sub navigation bar.

3. Give the node a name.

4. Edit the node, as described in the next section. Continue with your normal edit steps and click "Create Node" to save the new node.

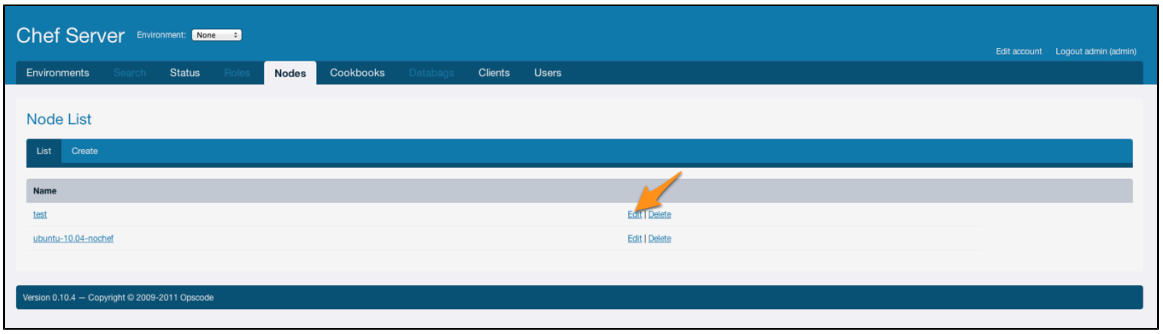


 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

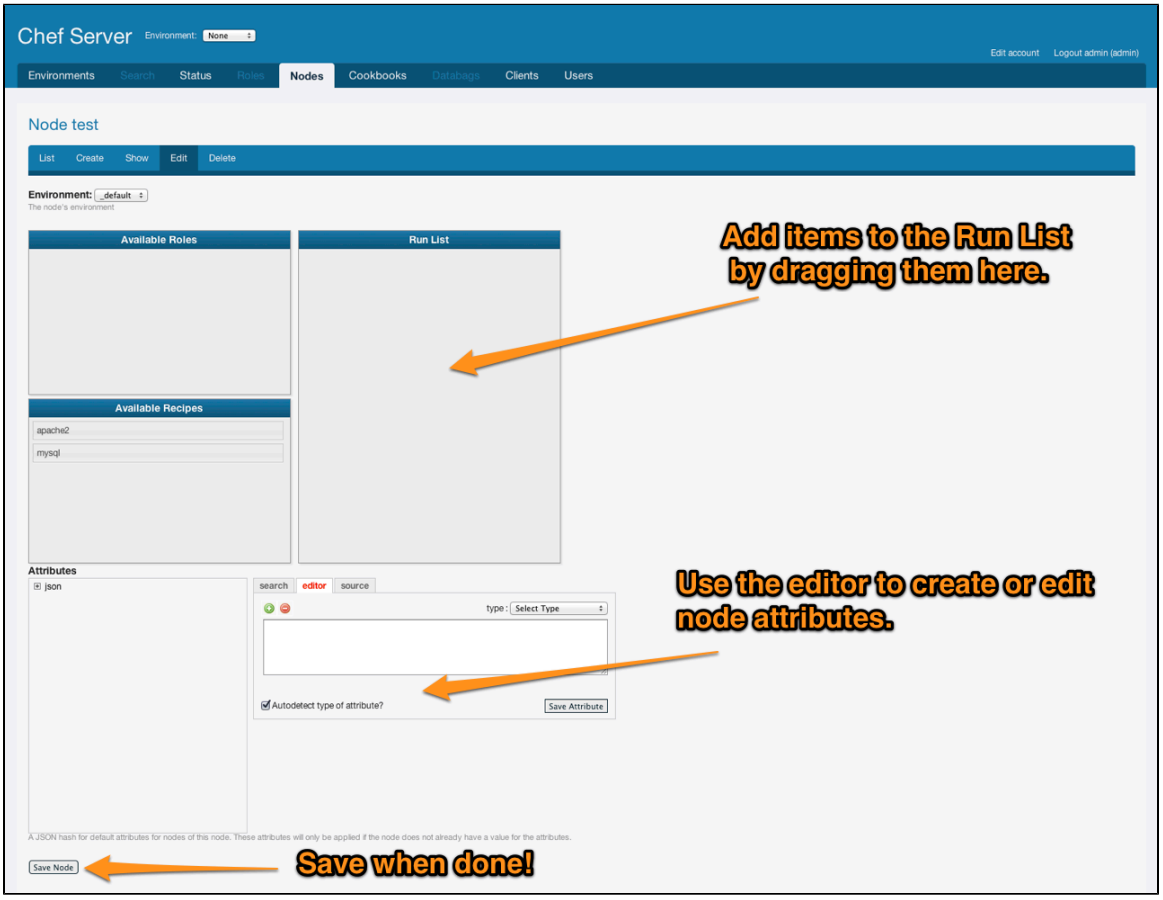


Editing a Node

NOTE: In any of the following actions where a drag is involved, you must drag the item to the lighter grey area of the target list.



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Modifying the Node's Run List

1. To include a role in the run list of the node being edited, drag it from "Available Roles" to "Run List".

2. To include a recipe in the run list of node being edited, drag it from "Available Recipes" to "Run List".

3. To remove an included role or recipe from the node being edited, drag it from the "Run List" back to the respective "Available" section.

4. Click "Save Node" to save.

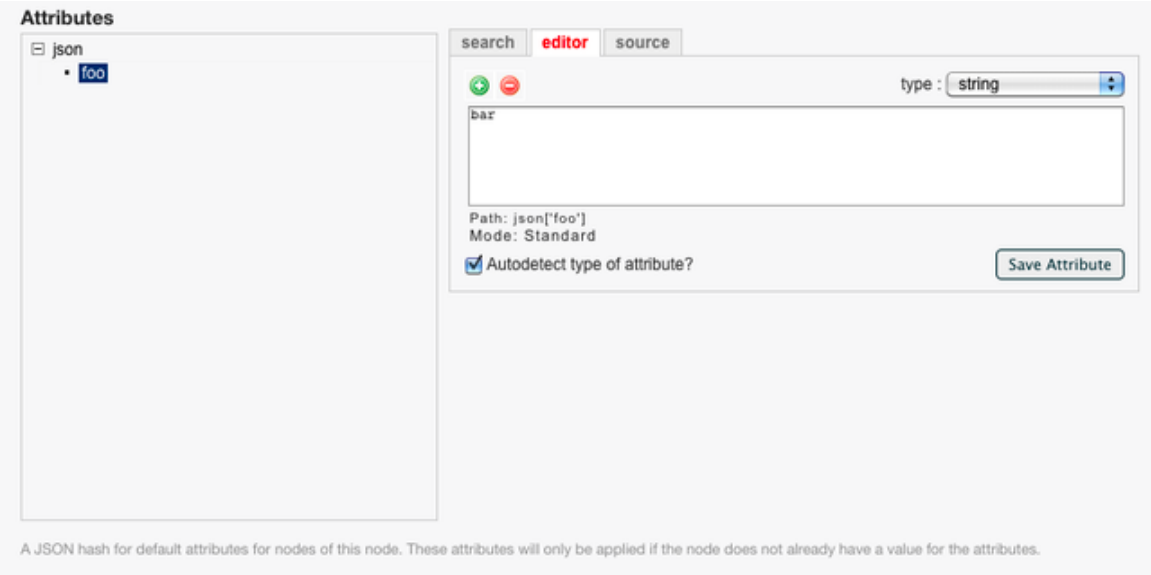
Modifying Node Attributes

To modify the attributes of a node, such that they override the values of the included recipe or role, click "json" from the "Default and Override Attributes" tree on the bottom left.

Using the Editor tab

Click the Editor tab on the right, click the green add button under the Editor tab, enter a name as the key and a body as the value and click "Add Attribute". The following figure shows the Default and Override Attributes tree and the JSON editor.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Using the Source tab

Enter the JSON body in the editor, and click the "Load JSON from source" button.

Click the Create Node or Save Node button to save the node. The following figure shows an example.



Deleting a Node

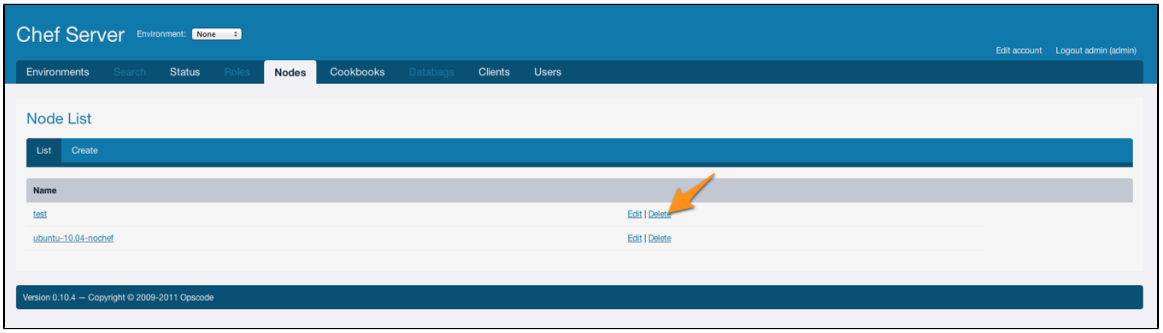
In order to delete a node:

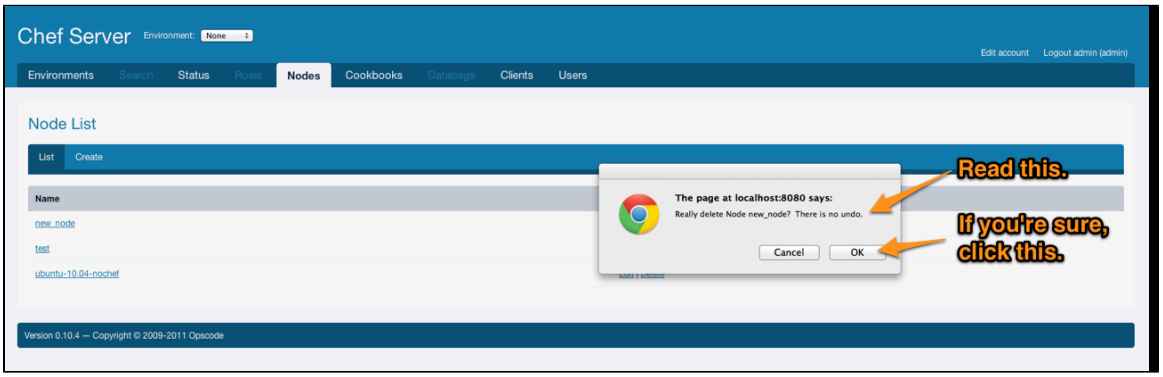
1. Login to the Management Console and navigate to the Nodes tab.

2. Click the Delete link next to the name of the node you want to delete, and click OK on the confirmation message box. Or, click the link of the node to view its details, and click Delete on the sub navigation menu.

Note: You need delete permissions on the node to it. If you don't have permission, you may not see a Delete link or tab.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License





Managing Environments through the Management Console

Managing Roles through the Management Console

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managing Roles through the Management Console

The Management Console allows users to create, read, update, and delete roles.

Listing Roles

To list roles:

1. Log on to the Management Console.

2. Click `Roles` on the main navigation menu.

Viewing a Role

Viewing a role shows the recipes and roles included in the role, as well as a tree view of the attributes associated with the role.

In order to view a specific role, on the roles page, click the link of the specific role you want to view.

Creating a Role

The role creation page allows you to add recipes and roles to the role you are editing. There is also a JSON editor for editing, adding and removing JSON attributes associated with the role.

In order to create a role:

1. Login to the Management Console and navigate to the `Roles` tab.

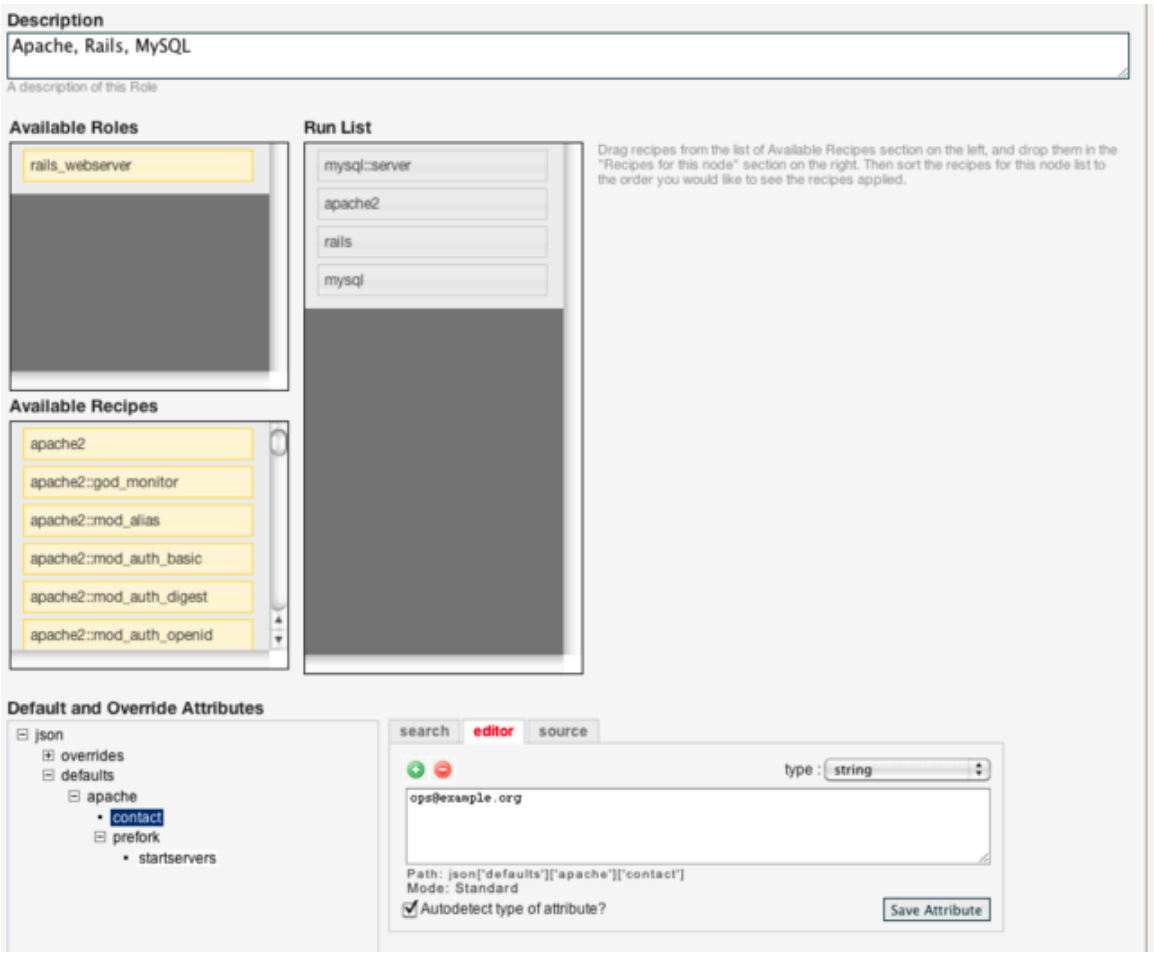
2. Click `Create` on the sub navigation bar.

3. Give the role a name and an optional description.

4. Continue onto editing the role, as described in the next section. Continue with your normal edit steps and click "Create Role" to save the new role.

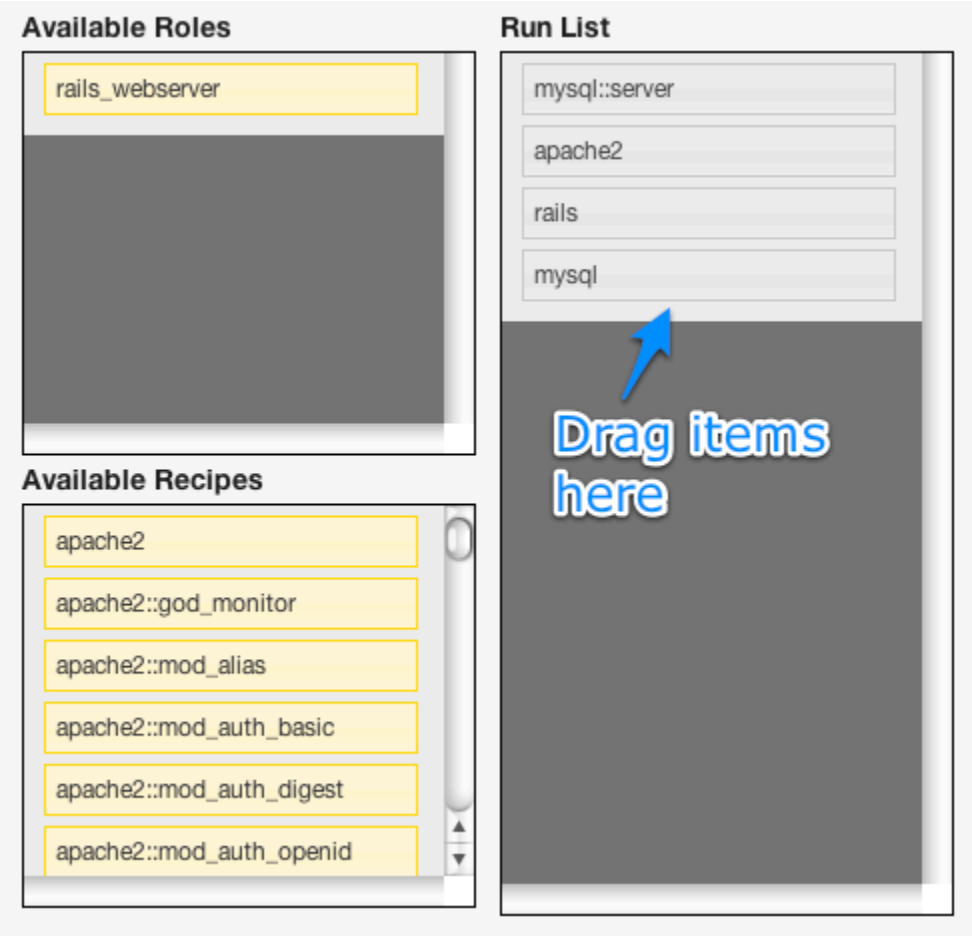
Editing a Role

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



NOTE: In any of the following actions where a drag is involved, you must drag the item to the lighter grey area of the target list.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



1. To add a role to the role being edited, drag it from "Available Roles" to "Run List".

2. To add a recipe to the role being edited, drag it from "Available Recipes" to "Run List".

3. To remove a role or recipe from the role being edited, drag it from the "Run List" back to the respective "Available" section.

4. Click `Save Role` to save.

Modifying Role Default and Override Attributes

You can edit the override attributes of a role, to override the values of an included recipe or role, and you can edit the default attributes of a role to specify attributes that do not exist in included recipes and roles. To do this, click "json" from the "Default and Override Attributes" tree on the bottom left. Then, click the Editor tab on the right, click the green add button under the Editor tab, enter a name as the key and a body as the value and click "Add Attribute". The following figures show the Default and Override Attributes tree and the JSON editor.

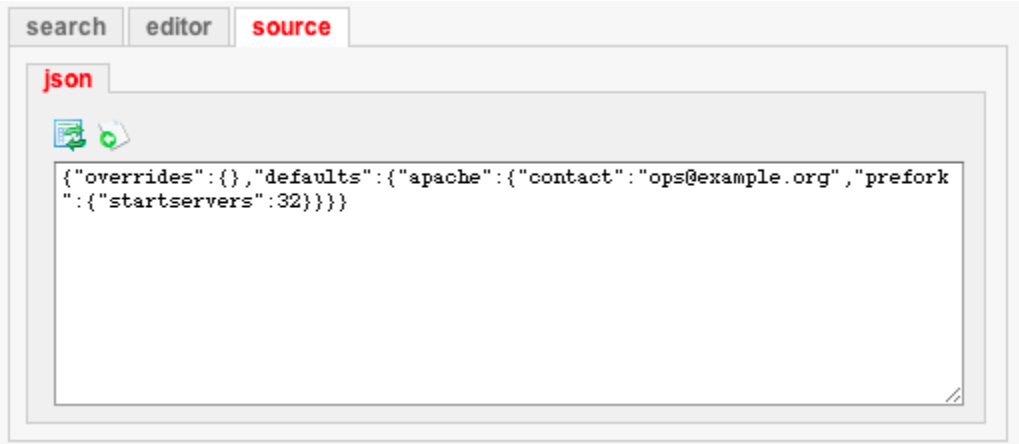
 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Using the Source tab

1. Enter the JSON body of your role in the editor, and click the "Load JSON from source" button.

2. Click the Create Role or Save Role button to save the role. The following figure shows an example.



Deleting a Role

In order to delete a role:

1. Login to the Management Console and navigate to the Roles tab.

2. Click the `Delete` link next to the name of the role you want to delete, and click `OK` on the confirmation message box. Or, click the role to view its details, and click `Delete` on the sub navigation menu.

Managing Nodes through the Management Console

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Search on the Management Console 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Search on the Management Console

You can view search indexes and search for nodes, API clients, roles, and data bags on the Management Console.

Viewing Available Search Indexes

In order to view available search indexes:

Log on to the Management Console and click the `Search` tab.

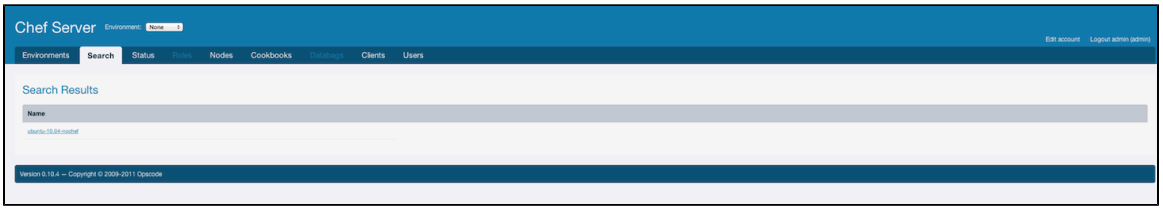
The search indexes should be listed. Nodes, Roles, and Clients are always available. Data bags are indexed by the data bag's name.

Searching

The query syntax described in the Search page also applies to the Management Console. The very basic syntax is "attribute:value". If you do not enter anything in any search box, it searches for everything ("*:*").

For more information, please read the Query Syntax section in the Search page.





Managing Roles through the Management Console

Shef

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Shef

Shef: The Chef REPL



Shef is a way to run chef in an Interactive Ruby (IRb) session. Shef currently supports recipe and attribute file syntax, as well as interactive debugging features.

Run Modes

Shef has three run modes:

- **Standalone**
  No cookbooks are loaded, and the run list is empty. This mode is the default.
- **Solo**
  Shef acts as a chef-solo client. It attempts to load chef-solo's configuration file and JSON attributes. If the JSON attributes set a run list, it will be honored. Cookbooks will be loaded in the same way that chef-solo loads them. Solo mode is activated with the `-s` or `--solo` com mand line option, and JSON attributes are specified in the same way as for chef-solo, with `-j /path/to/chef-solo.json`.
- **Client**
  Shef acts as a chef-client. During startup, it reads the chef client configuration file and contacts the server to get attributes and cookbooks. The run list will be set in the same way as normal chef client runs. Client mode is activated with the `-z` or `--client` options. You can also specify the configuration file with `-c CONFIG` and the server URL with `-S SERVER_URL`.

"Hello World" in Shef: A Simple Example

For this example, we'll use the standalone mode. (For solo or client modes, you would need to run shef using the `-s` or `-z` command line options as specified above, and take into consideration the configuration settings detailed below.)

If chef is installed from gems or your platform's package manager, shef should be in your path already. (If you are running chef from a git clone, shef is in `chef/bin/shef`.) To start just run `shef` with no options. You'll see the loading message, then the banner, and then the shef prompt:

```
$ bin/shef
./bin/../lib/chef.rb:30: warning: already initialized constant VERSION
loading configuration: none (standalone shef session)
Loading.......done.

This is shef, the Chef shell.
 Chef Version: 0.10.4
 http://www.opscode.com/chef
 http://wiki.opscode.com/display/chef/Home

run `help' for help, `exit' or ^D to quit.

Ohai2u danielsdeleo@eigenstate.local!
chef >
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Tutorials from the Community

- **Shef Tips and Tricks**

Opscode team member Steven Danna has put together a blog post with a number of **tricks and tools for productively debugging problems using Shef**.

Debug Chef-client runs by stepping through a node's run list, breaking before or after specific resources in order to inspect the state of the system.

- **Chef Development With Shef**

How to use Shef, the interactive chef console, for **iterative cookbook development** for those times when you just want to experiment without uploading anything to the server. This is a useful workflow oriented blog post by Community Member Jon Cowie .

At any point, you may use the `help` command to print a list of supported commands.

Use the `recipe` command to switch to recipe context:

```
chef > recipe
chef:recipe >
```

Now your typing will be evaluated the same context as recipes. We can create a file resource:

```
chef:recipe > file "/tmp/ohai2u_shef"
 => #<Chef::Resource::File:0x1b691ac @enclosing_provider=nil, @resource_name=:file, @before=nil,
@supports={}, @backup=5, @allowed_actions=[:nothing, :create, :delete, :touch, :create_if_missing],
@only_if=nil, @noop=nil, @collection=#<Chef::ResourceCollection:0x1b9926c @insert_after_idx=nil,
@resources_by_name={"file[/tmp/ohai2u_shef]"=>0}, @resources=[#<Chef::Resource::File:0x1b691ac ...>]>,

@updated=false, @provider=nil, @node=<Chef::Node:0xdeeaae @name="eigenstate.local">, @recipe_name=nil,
@not_if=nil, @name="/tmp/ohai2u_shef", @action="create", @path="/tmp/ohai2u_shef",
 @source_line="/Users/danielsdeleo/ruby/chef/chef/(irb#1) line 1", @params={}, @actions={},
@cookbook_name=nil, @ignore_failure=false>
```

At this point, shef has created the resource and put it in the run list, but not yet created the file. To initiate the chef run, use the `run_chef` command:

```
chef:recipe > run_chef
[Fri, 15 Jan 2010 10:42:47 -0800] DEBUG: Processing file[/tmp/ohai2u_shef]
[Fri, 15 Jan 2010 10:42:47 -0800] DEBUG: file[/tmp/ohai2u_shef] using Chef::Provider::File
[Fri, 15 Jan 2010 10:42:47 -0800] INFO: Creating file[/tmp/ohai2u_shef] at /tmp/ohai2u_shef
 => true
```

Shef also lets you switch to the same context as attribute files. You can set an attribute with the familiar syntax:

```
chef:recipe > attributes
chef:attributes > set[:hello] = "ohai2u-again"
 => "ohai2u-again"
chef:attributes >
```

Now you can switch back to recipe context and use the attribute you set:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
chef:attributes > recipe
 => :attributes
chef:recipe > file "/tmp/#{node.hello}"
```

Now, run chef again:

```
chef:recipe > run_chef
[Fri, 15 Jan 2010 10:53:22 -0800] DEBUG: Processing file[/tmp/ohai2u_shef]
[Fri, 15 Jan 2010 10:53:22 -0800] DEBUG: file[/tmp/ohai2u_shef] using Chef::Provider::File
[Fri, 15 Jan 2010 10:53:22 -0800] DEBUG: Processing file[/tmp/ohai2u-again]
[Fri, 15 Jan 2010 10:53:22 -0800] DEBUG: file[/tmp/ohai2u-again] using Chef::Provider::File
[Fri, 15 Jan 2010 10:53:22 -0800] INFO: Creating file[/tmp/ohai2u-again] at /tmp/ohai2u-again
 => true
chef:recipe >
```

Note that the first resource we created, (`file[/tmp/ohai2u_shef]`) is still in the run list, and gets executed again. Of course, since that file already exists, chef doesn't attempt to create it.

Finally, we can see that the files were created using shef's ls method:

```
chef:recipe > ls("/tmp").grep(/ohai/)
 => ["ohai2u-again", "ohai2u_shef"]
```



Shef Tutorial
You may want to review **Getting Started with Shef** for some additional high level examples on its use.

Client and Solo Modes

Shef provides client and solo modes which allow it to emulate running chef-client or chef-solo with additional debugging features. When running in client mode, shef will parse any provided JSON attributes file, then connect to the chef server and synchronize cookbooks. The node will have the same attributes and run list as it would during a chef client run. Solo mode is similar, except that attributes are loaded from a JSON file instead of a Chef server. Solo mode is activated with the `-s` or `--solo` command line option. Client mode is activated with the `-z` or `--client` options. Shef will load based upon its configuration settings.

Configuring Shef

Shef's configuration file support is new in Chef 0.9.8. (With older versions of chef, use the `-c` option to specify a configuration file.) Shef chooses the configuration file to load according to the following logic:

1. If a config file is specified with the `-c` option, it is used.
2. When shef is started with a named configuration as an argument, shef will search for a `shef.rb` file in that directory under `~/.chef`. For example, if shef is started as `shef production`, chef will load a configuration file from `~/.chef/production/shef.rb`
3. If no named configuration is given, shef attempts to load shef.rb from your .chef directory, i.e., `~/.chef/shef.rb`
4. If no shef.rb is found, shef will try to load `/etc/chef/client.rb` when started in client mode (i.e., `-z`)
5. If no shef.rb is found, shef will try to load `/etc/chef/solo.rb` when started in solo mode.

The shef configuration file

In your shef.rb you can configure shef just as you would configure the chef-client. To configure shef to properly authenticate with your Chef server, you might copy the `node_name`, `client_key`, and `chef_server_url` settings from your knife configuration:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Sample shef.rb

```
node_name                'your-knife-clientname'
client_key               File.expand_path('~/.chef/my-client.pem')
chef_server_url          'https://api.opscode.com/organizations/myorg'
```

You can also disable ohai plugins to speed up shef's boot process or include any arbitrary ruby code in your shef configuration file.

Running Shef as a Client Using Your Knife Credentials

By default, shef loads in standalone mode that doesn't connect to a server. On your laptop/desktop, you might want to run shef in client mode in order to test a feature that's only available when using Chef in the client-server configuration (i.e., Search or Data Bags).

If you've already configured Knife on your box, you can borrow knife's credentials to connect to the server. In order for this to work, you need to create a node with the same name that your knife client uses to authenticate. This is the `node_name` configuration parameter in your knife.rb file (typically located at `~/.chef/knife.rb` – you can find this with `grep node_name ~/.chef/knife.rb`. Then, run `knife node create $YOUR_KNIFE_NODE_NAME` to create a node with this name (the node attributes aren't important unless you need them for some other pupose).

Once you've done this, you can run shef in client mode, where it will connect to the server via the config value with 'chef_server_url'. By passing it your knife.rb, it will connect to the server via the config value.

```
shef -z -c ~/.chef/knife.rb
```

No Longer Needed for Chef REST api interaction
As of chef 0.9.8, you no longer need to take this approach if you just want to be able to access the Chef REST api from within shef. Just configure your shef.rb similar to the example in the previous section.

Managing Chef Server with Shef

When properly configured to access your Chef Server, shef can list, show, search for, and edit cookbooks, clients, nodes, roles, and databags.

These commands all have a syntax of the form *items.command* so to list nodes, you use `nodes.list`:

```
chef (preprod) > nodes.list
 => [node[i-f09a939b], node[i-049a936f], node[i-eaaaa581], node[i-9154b1fb], node[i-6a213101],
node[i-c2687aa9], node[i-7abeaa11], node[i-4eb8ac25], node[i-9a2030f1],
node[i-a06875cb], node[i-145f457f], node[i-e032398b], node[i-dc8c98b7], node[i-6afdf401],
node[i-f49b119c], node[i-5abfab31], node[i-78b8ac13], node[i-d99678b3],
node[i-02322269], node[i-feb4a695], node[i-9e2232f5], node[i-6e213105], node[i-cdde3ba7],
node[i-e8bfb083], node[i-743c2c1f], node[i-2eaca345], node[i-aa7f74c1],
node[i-72fdf419], node[i-140e1e7f], node[i-f9d43193], node[i-bd2dc8d7], node[i-8e7f70e5],
node[i-78f2e213], node[i-962232fd], node[i-4c322227], node[i-922232f9],
node[i-c02728ab], node[i-f06c7b9b]]
```

The *list* command can take a code block, which is applied (but not saved) to each object returned from the server. You can list all of your nodes and their run lists:

```
chef (preprod) > nodes.list {|n| puts "#{n.name}: #{n.run_list}" }
i-f09a939b: role[lb], role[preprod], recipe[aws]
i-049a936f: role[lb], role[preprod], recipe[aws]
i-9154b1fb: recipe[erlang], role[base], role[couchdb], role[preprod],
i-6a213101: role[chef], role[preprod]
# more...
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

To find a specific node, use `nodes.show`:

```
chef (preprod) > load_balancer = nodes.show('i-f09a939b')
 => node[i-f09a939b]
chef (preprod) > load_balancer.ec2.public_hostname
 => "ec2-111-22-333-44.compute-1.amazonaws.com"
```

You can also use search from within shef. To search for nodes, use `nodes.find`:

```
chef (preprod) > pp nodes.find(:ec2_public_hostname => 'ec2*')
```

Again, you can give a code block to "format" the results:

```
chef (preprod) > pp nodes.find(:ec2_public_hostname => 'ec2*') {|n| n.ec2.ami_id } and nil
["ami-f8927a91",
 "ami-f8927a91",
 "ami-a89870c1",
 "ami-a89870c1",
 "ami-a89870c1",
 "ami-a89870c1",
 "ami-a89870c1"
# and more...
chef (preprod) > amis = nodes.find(:ec2_public_hostname => 'ec2*') {|n| n.ec2.ami_id }
chef (preprod) > puts amis.uniq.sort
ami-4b4ba522
ami-a89870c1
ami-eef61587
ami-f8927a91
```

Debugging Recipes with Shef

Shef gives you the ability to manipulate your current position in the run list during a chef run. To use this feature, you'll need at least one breakpoint in one recipe in the client's run list.

The Breakpoint Resource

Breakpoints are implemented as a chef resource. The breakpoint resource has no custom attributes. Its default action is *break*. When the *break* a ction is called on the breakpoint provider, the provider tests if it is running under shef, or a normal chef-client (or solo) run. If it's running under chef-client or chef-solo, no action is taken; if it's running under shef, the chef run will be paused.

Stepping Through the Run List

To explore how breakpointing and manually controlling chef execution works, create a simple recipe in shef:

```
chef > recipe
chef:recipe > echo off
chef:recipe > file "/tmp/before-breakpoint"
chef:recipe > breakpoint "foo"
chef:recipe > file "/tmp/after-breakpoint"
```

Now, run chef:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
chef:recipe > run_chef
[Fri, 15 Jan 2010 14:17:49 -0800] DEBUG: Processing file[/tmp/before-breakpoint]
[Fri, 15 Jan 2010 14:17:49 -0800] DEBUG: file[/tmp/before-breakpoint] using Chef::Provider::File
[Fri, 15 Jan 2010 14:17:49 -0800] INFO: Creating file[/tmp/before-breakpoint] at
/tmp/before-breakpoint
[Fri, 15 Jan 2010 14:17:49 -0800] DEBUG: Processing
[./bin/../lib/chef/mixin/recipe_definition_dsl_core.rb:56:in `new']
[Fri, 15 Jan 2010 14:17:49 -0800] DEBUG: [./bin/../lib/chef/mixin/recipe_definition_dsl_core.rb:56:in
`new'] using Chef::Provider::Breakpoint
```

We can see that chef ran the first resource before the breakpoint, `file[/tmp/before-breakpoint]`, but it stopped after executing the breakpoint. We also see that chef attempted to name the breakpoint resource after its position in the source file, but we confused it by entering the resource interactively.

From here, we can tell shef to finish the chef run using `chef_run.resume`

```
chef:recipe > chef_run.resume
[Fri, 15 Jan 2010 14:27:08 -0800] INFO: Creating file[/tmp/after-breakpoint] at /tmp/after-breakpoint
```

If we list our `/tmp` directory, we see that both files were created:

```
chef:recipe > puts ls("/tmp").grep(/break/)
after-breakpoint
before-breakpoint
```

We can also rewind the chef run and step through it:

```
chef:recipe > Chef::Log.level = :debug # debug logging won't turn on automatically in this case
 => :debug
chef:recipe > chef_run.rewind
 => 0
chef:recipe > chef_run.step
[Fri, 15 Jan 2010 14:40:52 -0800] DEBUG: Processing file[/tmp/before-breakpoint]
[Fri, 15 Jan 2010 14:40:52 -0800] DEBUG: file[/tmp/before-breakpoint] using Chef::Provider::File
 => 1
chef:recipe > chef_run.step
[Fri, 15 Jan 2010 14:40:54 -0800] DEBUG: Processing
[./bin/../lib/chef/mixin/recipe_definition_dsl_core.rb:56:in `new']
[Fri, 15 Jan 2010 14:40:54 -0800] DEBUG: [./bin/../lib/chef/mixin/recipe_definition_dsl_core.rb:56:in
`new'] using Chef::Provider::Breakpoint
 => 2
chef:recipe > chef_run.step
[Fri, 15 Jan 2010 14:40:56 -0800] DEBUG: Processing file[/tmp/after-breakpoint]
[Fri, 15 Jan 2010 14:40:56 -0800] DEBUG: file[/tmp/after-breakpoint] using Chef::Provider::File
 => 3
```

From the output, we see that we rewound the run list, but when we executed the resources again, they repeated their checks for the existence of the files and found that they now exist, so chef skipped creating them. If we delete the files:

```
chef:recipe > ls("/tmp").grep(/breakpoint/).each {|f| rm "/tmp/#{f}" }
 => ["after-breakpoint", "before-breakpoint"]
```

Then we can rewind and resume the chef run and get the expected results:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
chef:recipe > chef_run.rewind
chef:recipe > chef_run.resume
[Fri, 15 Jan 2010 14:48:56 -0800] DEBUG: Processing file[/tmp/before-breakpoint]
[Fri, 15 Jan 2010 14:48:56 -0800] DEBUG: file[/tmp/before-breakpoint] using Chef::Provider::File
[Fri, 15 Jan 2010 14:48:56 -0800] INFO: Creating file[/tmp/before-breakpoint] at
/tmp/before-breakpoint
[Fri, 15 Jan 2010 14:48:56 -0800] DEBUG: Processing
[./bin/../lib/chef/mixin/recipe_definition_dsl_core.rb:56:in `new']
[Fri, 15 Jan 2010 14:48:56 -0800] DEBUG: [./bin/../lib/chef/mixin/recipe_definition_dsl_core.rb:56:in
`new'] using Chef::Provider::Breakpoint
chef:recipe > chef_run.resume
[Fri, 15 Jan 2010 14:49:20 -0800] DEBUG: Processing file[/tmp/after-breakpoint]
[Fri, 15 Jan 2010 14:49:20 -0800] DEBUG: file[/tmp/after-breakpoint] using Chef::Provider::File
[Fri, 15 Jan 2010 14:49:20 -0800] INFO: Creating file[/tmp/after-breakpoint] at /tmp/after-breakpoint
```

Turn Debugging up to 11

In shef, it's possible get get extremely verbose debugging using irb's tracing feature. Shef provides a shortcut to turn tracing on and off, using `tracing on` and `tracing off`.

For example, here's the output from turning tracing on and off:

```
chef > tracing on
/Users/danielsdeleo/.rvm/ree-1.8.7-2009.10/lib/ruby/1.8/tracer.rb:150: warning: tried to create Proc
object without a block
/Users/danielsdeleo/.rvm/ree-1.8.7-2009.10/lib/ruby/1.8/tracer.rb:146: warning: tried to create Proc
object without a block
tracing is on
 => nil
chef > tracing off
#0:(irb):2:Object:-: tracing off
#0:./bin/../lib/chef/shef/ext.rb:78:Shef::Extensions::Object:>:       def off
#0:./bin/../lib/chef/shef/ext.rb:79:Shef::Extensions::Object:-:         :off
#0:./bin/../lib/chef/shef/ext.rb:79:Shef::Extensions::Object:<:         :off
#0:./bin/../lib/chef/shef/ext.rb:259:Object:>:   def tracing(on_or_off)
#0:./bin/../lib/chef/shef/ext.rb:260:Object:-:     conf.use_tracer = on_or_off.on_off_to_bool
#0:./bin/../lib/chef/shef/ext.rb:135:Shef::Extensions::Symbol:>:       def on_off_to_bool
#0:./bin/../lib/chef/shef/ext.rb:136:Shef::Extensions::Symbol:-:         self.to_s.on_off_to_bool
#0:./bin/../lib/chef/shef/ext.rb:136:Symbol:>:         self.to_s.on_off_to_bool
#0:./bin/../lib/chef/shef/ext.rb:136:Symbol:<:         self.to_s.on_off_to_bool
#0:./bin/../lib/chef/shef/ext.rb:122:Shef::Extensions::String:>:       def on_off_to_bool
#0:./bin/../lib/chef/shef/ext.rb:123:Shef::Extensions::String:-:         case self
#0:./bin/../lib/chef/shef/ext.rb:124:Shef::Extensions::String:-:         when "on"
#0:./bin/../lib/chef/shef/ext.rb:124:Kernel:>:         when "on"
#0:./bin/../lib/chef/shef/ext.rb:124:String:>:         when "on"
#0:./bin/../lib/chef/shef/ext.rb:124:String:<:         when "on"
#0:./bin/../lib/chef/shef/ext.rb:124:Kernel:<:         when "on"
#0:./bin/../lib/chef/shef/ext.rb:126:Shef::Extensions::String:-:         when "off"
#0:./bin/../lib/chef/shef/ext.rb:126:Kernel:>:         when "off"
#0:./bin/../lib/chef/shef/ext.rb:126:String:>:         when "off"
#0:./bin/../lib/chef/shef/ext.rb:126:String:<:         when "off"
#0:./bin/../lib/chef/shef/ext.rb:126:Kernel:<:         when "off"
#0:./bin/../lib/chef/shef/ext.rb:127:Shef::Extensions::String:-:           false
#0:./bin/../lib/chef/shef/ext.rb:127:Shef::Extensions::String:<:           false
#0:./bin/../lib/chef/shef/ext.rb:136:Shef::Extensions::Symbol:<:         self.to_s.on_off_to_bool
tracing is off
 => nil
chef >
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Shef Help

```
================================================================================
| Command                  | Description
================================================================================
| shef_help                | prints this help message
| version                  | prints information about chef
| recipe                   | switch to recipe mode
| attributes               | switch to attributes mode
| node                     | returns the current node (i.e., this host)
| ohai                     | pretty print the node's attributes
| run_chef                 | run chef using the current recipe
| chef_run                 | returns an object to control a paused chef run
| chef_run.skip_forward    | move forward in the run list
| chef_run.resume          | resume the chef run
| chef_run.step            | run only the next resource
| chef_run.skip_back       | move back in the run list
| reset                    | resets the current recipe
| echo                     | turns printout of return values on or off
| echo?                    | says if echo is on or off
| tracing                  | turns on or off tracing of execution. *verbose*
| tracing?                 | says if tracing is on or off
| ls                       | simple ls style command
================================================================================
```

Use Case Scenario

To get a list of nodes with recipe postfix I'm using search(:node,"recipe:postfix"), but in this case I actually only want nodes with the postfix sub-recipe "delivery" (postfix::delivery). How do I express that correctly in the search syntax? (just putting in "postfix::delivery" returns an error)

One approach would be to use shef:

```
search(:node, 'recipes:postfix\:\:delivery')
```

Single vs. double quotes is significant here since you want to include a backslash in the string instead of having ruby interpret it as an escape.

 Management Console

Spiceweasel 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Spiceweasel



Spiceweasel is a command-line tool for batch loading Chef infrastructure.

It provides a simple syntax in either JSON or YAML for describing and deploying infrastructure in order with the Chef command-line tool Knife.

Please note, Spiceweasel is not an "official Chef tool", but it is developed by Opscode team member Matt Ray.

Requirements

- Spiceweasel currently depends on knife to run commands for it. Infrastructure files must either end in .json or .yml to be processed.
- It is written with Chef 0.9.12 and the 0.10.x series and supports cookbooks, environments, roles, data bags and nodes.
- Also, as Spiceweasel is not an "official Chef tool", *it is not supported for Hosted Chef customers*.

Installation

The source code is available on github, but Spiceweasel is distributed and installable as a gem.

```
gem install spiceweasel
```

Syntax

The syntax for the spiceweasel file may be either JSON or YAML format of Chef primitives describing what is to be instantiated. Below are 2 examples describing the same infrastructure.

OpenStack Swift, Chef and SpiceWeasel

Our friends at Voxel put together two great blog posts on the use of OpenStack, Chef, and Spiceweasel to automate the deployment, configuration and management of their infrastructure.

Read **Part One**, and **Part Two**, and love the weasel.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Spiceweasel 1.0

For more information, check out the **blog post detailing Spiceweasel 1.0** put together by Opscode Team Member Matt Ray.

YAML

Given the `example.yml`:

```
cookbooks:
- apache2:
- apt:
- 1.2.0
- mysql:

environments:
- development:
- qa:
- production:

roles:
- base:
- iisserver:
- monitoring:
- webserver:

data bags:
- users:
- alice
- bob
- chuck
- data:
- *
- passwords:
- secret secret_key
- mysql
- rabbitmq

nodes:
- serverA:
- role[base]
- -i ~/.ssh/mray.pem -x user --sudo -d ubuntu10.04-gems
- serverB serverC:
- role[base]
- -i ~/.ssh/mray.pem -x user --sudo -d ubuntu10.04-gems -E production
- ec2 4:
- role[webserver] recipe[mysql::client]
- -S mray -i ~/.ssh/mray.pem -x ubuntu -G default -I ami-7000f019 -f m1.small
- rackspace 3:
- recipe[mysql],role[monitoring]
- --image 49 --flavor 2
- windows_winrm winboxA:
- role[base],role[iisserver]
- -x Administrator -P 'super_secret_password'
- windows_ssh winboxB winboxC:
- role[base],role[iisserver]
- -x Administrator -P 'super_secret_password'
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Spiceweasel generates the following knife commands:

```
knife cookbook upload apache2
knife cookbook upload apt
knife cookbook upload mysql
knife environment from file development.rb
knife environment from file qa.rb
knife environment from file production.rb
knife role from file base.rb
knife role from file iisserver.rb
knife role from file monitoring.rb
knife role from file webserver.rb
knife data bag create users
knife data bag from file users alice.json
knife data bag from file users bob.json
knife data bag from file users chuck.json
knife data bag create data
knife data bag create passwords
knife data bag from file passwords mysql.json --secret-file secret_key
knife data bag from file passwords rabbitmq.json --secret-file secret_key
knife bootstrap serverA -i ~/.ssh/mray.pem -x user --sudo -d ubuntu10.04-gems -r 'role[base]'
knife bootstrap serverB -i ~/.ssh/mray.pem -x user --sudo -d ubuntu10.04-gems -E production -r
'role[base]'
knife bootstrap serverC -i ~/.ssh/mray.pem -x user --sudo -d ubuntu10.04-gems -E production -r
'role[base]'
knife ec2 server create -S mray -i ~/.ssh/mray.pem -x ubuntu -G default -I ami-7000f019 -f m1.small -r
'role[webserver],recipe[mysql::client]'
knife ec2 server create -S mray -i ~/.ssh/mray.pem -x ubuntu -G default -I ami-7000f019 -f m1.small -r
'role[webserver],recipe[mysql::client]'
knife ec2 server create -S mray -i ~/.ssh/mray.pem -x ubuntu -G default -I ami-7000f019 -f m1.small -r
'role[webserver],recipe[mysql::client]'
knife ec2 server create -S mray -i ~/.ssh/mray.pem -x ubuntu -G default -I ami-7000f019 -f m1.small -r
'role[webserver],recipe[mysql::client]'
knife rackspace server create --image 49 --flavor 2 -r 'recipe[mysql],role[monitoring]'
knife rackspace server create --image 49 --flavor 2 -r 'recipe[mysql],role[monitoring]'
knife rackspace server create --image 49 --flavor 2 -r 'recipe[mysql],role[monitoring]'
knife bootstrap windows winrm winboxA -x Administrator -P 'super_secret_password' -r
'role[base],role[iisserver]'
knife bootstrap windows ssh winboxB -x Administrator -P 'super_secret_password' -r
'role[base],role[iisserver]'
knife bootstrap windows ssh winboxC -x Administrator -P 'super_secret_password' -r
'role[base],role[iisserver]'
```

Cookbooks

The `cookbooks` section of the manifest currently supports `knife cookbook upload FOO` where `FOO` is the name of the cookbook in the `cookbooks` directory. *The default behavior is to download the cookbook as a tarball, untar it and remove the tarball.* The `--siteinstall` option will allow for use of `knife cookbook site install` with the cookbook and the creation of a vendor branch if git is the underlying version control. If a version is passed, it is validated against the existing cookbook `metadata.rb` and it must match the `metadata.rb` string exactly. You may pass any additional arguments if necessary.

Assuming the apt cookbook is not present, the YAML snippet

```
cookbooks:
- apache2:
- apt:
  - 1.1.0
- mysql:
```

produces the knife commands

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife cookbook upload apache2
knife cookbook site download apt 1.1.0 --file cookbooks/apt.tgz
tar -C cookbooks/ -xf cookbooks/apt.tgz
knife cookbook upload apt
knife cookbook upload mysql
```

Environments

The `environments` section of the manifest currently supports `knife environment from file FOO` where `FOO` is the name of the environment file ending in `.rb` or `.json` in the `environments` directory. Validation is done to ensure the filename matches the environment and that any cookbooks referenced are listed in the manifest.

The YAML snippet

```
environments:
- development:
- qa:
- production:
```

produces the knife commands

```
knife environment from file development.rb
knife environment from file qa.rb
knife environment from file production.rb
```

Roles

The `roles` section of the manifest currently supports `knife role from file FOO` where `FOO` is the name of the role file ending in `.rb` or `.json` in the `roles` directory. Validation is done to ensure the filename matches the role name and that any cookbooks or roles referenced are listed in the manifest.

The YAML snippet

```
roles:
- base:
- monitoring:
- webserver:
```

produces the knife commands

```
knife role from file base.rb
knife role from file monitoring.rb
knife role from file webserver.rb
```

Data Bags

The `data bags` section of the manifest currently creates the data bags listed with `knife data bag create FOO` where `FOO` is the name of the data bag. Individual items may be added to the data bag as part of a JSON or YAML sequence, the assumption is made that they `.json` files and in the proper `data_bags/FOO` directory. You may also pass a wildcard as an entry to load all matching data bags (ie. `*`). Encrypted Data Bags are supported by listing `secret filename` as the first item (where `filename` is the secret key to be used). Validation is done to ensure the JSON is properly formatted, the id matches and any secret keys are in the correct locations.

Assuming the presence of `dataA.json` and `dataB.json` in the `data_bags/data` directory, the YAML snippet

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
data bags:
- users:
  - alice
  - bob
  - chuck
- data:
  - *
- passwords:
  - secret secret_key
  - mysql
  - rabbitmq
```

produces the knife commands

```
knife data bag create users
knife data bag from file users data_bags/users/alice.json
knife data bag from file users data_bags/users/bob.json
knife data bag from file users data_bags/users/chuck.json
knife data bag create data
knife data bag from file users data_bags/data/dataA.json
knife data bag from file users data_bags/data/dataB.json
knife data bag create passwords
knife data bag from file passwords data_bags/passwords/mysql.json --secret-file secret_key
knife data bag from file passwords data_bags/passwords/rabbitmq.json --secret-file secret_key
```

Nodes

The `nodes` section of the manifest bootstraps a node for each entry where the entry is a hostname or provider and count. A shortcut syntax for bulk-creating nodes with various providers where the line starts with the provider and ends with the number of nodes to be provisioned. Windows nodes need to specify either `windows_winrm` or `windows_ssh` depending on the protocol used, followed by the name of the node(s). *Each node requires 2 items after it in a sequence.* You may also use the `--parallel` flag from the command line, allowing provider commands to run simultaneously for faster deployment.

The first item after the node is the run_list and the second are the CLI options used. The run_list may be space or comma-delimited. Validation is performed on the run_list components to ensure that only cookbooks and roles listed in the manifest are used. Validation on the options ensures that any Environments referenced are also listed. You may specify multiple nodes to have the same configuration by listing them separated by a space.

The YAML snippet

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
nodes:
- serverA:
  - role[base]
  - -i ~/.ssh/mray.pem -x user --sudo -d ubuntu10.04-gems
- serverB serverC:
  - role[base]
  - -i ~/.ssh/mray.pem -x user --sudo -d ubuntu10.04-gems -E production
- ec2 3:
  - role[webserver] recipe[mysql::client]
  - -S mray -i ~/.ssh/mray.pem -x ubuntu -G default -I ami-7000f019 -f m1.small
- rackspace 3:
  - recipe[mysql],role[monitoring]
  - --image 49 --flavor 2
- windows_winrm winboxA:
  - role[base],role[iisserver]
  - -x Administrator -P 'super_secret_password'
- windows_ssh winboxB winboxC:
  - role[base],role[iisserver]
  - -x Administrator -P 'super_secret_password'
```

produces the knife commands

```
knife bootstrap serverA -i ~/.ssh/mray.pem -x user --sudo -d ubuntu10.04-gems -r 'role[base]'
knife bootstrap serverB -i ~/.ssh/mray.pem -x user --sudo -d ubuntu10.04-gems -E production -r
'role[base]'
knife bootstrap serverC -i ~/.ssh/mray.pem -x user --sudo -d ubuntu10.04-gems -E production -r
'role[base]'
knife ec2 server create -S mray -i ~/.ssh/mray.pem -x ubuntu -G default -I ami-7000f019 -f m1.small -r
'role[webserver],recipe[mysql::client]'
knife ec2 server create -S mray -i ~/.ssh/mray.pem -x ubuntu -G default -I ami-7000f019 -f m1.small -r
'role[webserver],recipe[mysql::client]'
knife ec2 server create -S mray -i ~/.ssh/mray.pem -x ubuntu -G default -I ami-7000f019 -f m1.small -r
'role[webserver],recipe[mysql::client]'
knife ec2 server create -S mray -i ~/.ssh/mray.pem -x ubuntu -G default -I ami-7000f019 -f m1.small -r
'role[webserver],recipe[mysql::client]'
knife rackspace server create --image 49 --flavor 2 -r 'recipe[mysql],role[monitoring]'
knife rackspace server create --image 49 --flavor 2 -r 'recipe[mysql],role[monitoring]'
knife rackspace server create --image 49 --flavor 2 -r 'recipe[mysql],role[monitoring]'
knife bootstrap windows winrm winboxA -x Administrator -P 'super_secret_password' -r
'role[base],role[iisserver]'
knife bootstrap windows ssh winboxB -x Administrator -P 'super_secret_password' -r
'role[base],role[iisserver]'
knife bootstrap windows ssh winboxC -x Administrator -P 'super_secret_password' -r
'role[base],role[iisserver]'
```

Usage

To run a spiceweasel file, run the following from you Chef repository directory:

```
spiceweasel path/to/infrastructure.json
```

or

```
spiceweasel path/to/infrastructure.yml
```

This will generate the knife commands to build the described infrastructure. *Infrastructure files must end in either `.json` or `.yml`.*

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

`-c --knifeconfig`
Specify a `knife.rb` configuration file to use with the knife commands.

`--debug`
This provides verbose debugging messages.

`-d --delete`
The delete command will generate the knife commands to delete the infrastructure described in the file. This includes each cookbook, role, data bag, environment and node listed. Node deletion will specify individual nodes, attempt to pass the list of nodes to the cloud provider for deletion, and finish with `knife node bulk delete`. If you are mixing individual nodes with cloud provider nodes it is possible that nodes may be missed from cloud provider deletion and you should double-check (ie. `knife ec2 server list`).

`--dryrun`
This is the default action, printing the knife commands to be run without executing them.

`-h --help`
Print the currently-supported usage options for spiceweasel.

`--novalidation`
Disable validation ensuring existence of the cookbooks, environments, roles, data bags and nodes in infrastructure file.

`--parallel`
Use the GNU `parallel` command to execute `knife VENDOR server create` commands that may be run simultaneously.

`-r --rebuild`
The rebuild command will generate the knife commands to delete and recreate the infrastructure described in the file. This includes each cookbook, role, data bag, environment and node listed.

`--siteinstall`
Use the 'install' command with 'knife cookbook site' instead of the default 'download'.

`-v --version`
Print the version of spiceweasel currently installed.

 Shef

Documentation 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Documentation

Useful Documentation



- ### Chef Wiki as a PDF

  The entire wiki, as a single PDF file - *As of 1/1/2012* - (21.1Mb)

- ### A Can of Condensed Chef Documentation

  A brief explanation of each chef component, linking off to additional detail.

- ### Glossary

  A glossary of terms, technologies and concepts used by Chef and in working with Chef.

- ### Chef Configuration Settings

  Each Chef executable is configured through the Chef::Config object. In this page, settings are first listed in the Ruby DSL context, followed by a description of the setting and applicable executable context.

- ### Just Enough Ruby for Chef

  Ruby is a language designed to be easy to read and to behave exactly as you'd expect, so it shouldn't take you too long to get up to speed.

- ### Release Notes

  The release notes for the current version of chef, with historical release note for previous versions.

- ### Chef OmniGraffle & Visio Stencils

  A set of stencils for use with OmniGraffle for creating diagrams.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Test-Driven Infrastructure with Chef

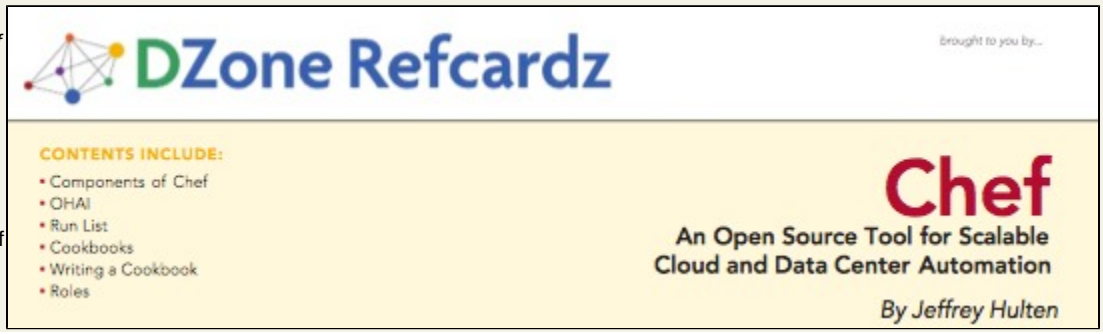
Community member Stephen Nelson-Smith has written **Test-Driven Infrastructure with Chef**.

He demonstrates a radical approach to developing web infrastructure that combines the powerful Chef configuration management framework with Cucumber, the leading Behavior-driven development (BDD) tool. Learn how developing code test-first allows you to make significant changes without the fear of unexpected side effects.

DZone Refcardz

Community member and IRC participant Jeffrey Hulten put together this handy **DZone Refcardz for Chef**.

Download this *free* "cheat sheet" from DZone, and use it and Chef to easily create a fully automated infrastructure of any size.



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Lesson Plan



You want to learn more about Chef? We want to teach you about Chef! *How convenient!* Lets start at the top, shall we?

Fast Start Guide

Head over to the Fast Start Guide, which will walk you through the following:

- Install Chef on your local workstation
- Set up a Chef Repository for storing your cookbooks and other "infrastructure as code."
- Download a cookbook for managing a new Chef Node.

It takes a few assumptions to get you up and running as quickly as possible.

Chef Basics

Learn some of the central concepts of configuration management benefits for your infrastructure.

1. We begin with an Architecture Introduction, covering the basic functions of the Chef Server, Nodes, and Chef Workstations and how these components communicate.
2. Then an overview of the Core Components, which introduces all of the aspects and components of modeling your infrastructure, Configuring Nodes and Managing Chef.
3. Onward to Cooking School and begin an Introduction to Cookbooks and More. Cookbooks are Chef's fundamental units of distribution, the way Chef users package up, distribute and share configuration information. Recipes, Resources, Attributes, Roles and more are also introduced.
4. The final basic section is an Introduction to Search and Data Bags, two of Chef's most powerful features allowing you to dynamically change the configuration of your infrastructure based on data.

Chef Architecture

With all that under your belt, it's time to tackle the dirty secrets of what's happening behind the scenes with Chef Architecture. We'll give you the scoop on Chef's Authentication and Authorization system and go over the Anatomy of a Chef Run, where we go in-depth with the process by which your systems get configured. From there we'll review all the executable parts of Chef - Chef Client, Chef Solo, Chef Server, Chef Indexer, and Server API and Cookbook Site API interaction.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

A Can of Condensed Chef Documentation



Chef's documentation is vast and broken up into many pages on the Opscode wiki

The goal here is to index this information and give a brief explanation of each topic

Architecture

http://wiki.opscode.com/display/chef/Architecture

Chef is a configuration management platform in the same class of tools as Cfengine, Bcfg2, and Puppet. The idea is to define policy at a centralized, version controlled place, and then have the machines under management pull down their policy and converge onto that state at regular intervals. This gives you a single point of administration allowing for easier change management and disaster recovery. Combined with a compute resource provisioning layer (such as knife's ability to manipulate EC2 or Rackspace servers), entire complex infrastructures can pop into existence in minutes.

Chef Server

http://wiki.opscode.com/display/chef/Chef+Server

Chef server has various components under the hood. Assorted information (cookbooks, databags, client certificates, and node objects), are stored in CouchDB as JSON blobs. CouchDB is indexed by chef-solr-indexer. RabbitMQ sits between the data store and A RESTful API service that exposes all this to the network as chef-server. If you don't want to run chef-server yourself, Opscode will do it for you for with their Hosted Chef service for a pittance. The management console is really handy during development since it provides a nice way to examine JSON data. However, it should be noted that real chefs value knife techniques.

Glossary
See the Glossary for a list of the terms, technologies and concepts used by Chef and in working with Chef.

Clients

http://wiki.opscode.com/display/chef/API+Clients

In Chef, the term "client" refers to an SSL certificate for an API user of chef-server. This is often a point of confusion, and should not be confused with chef-client. Most of the time, one machine has one client certificate, which corresponds to one node object.

Nodes

http://wiki.opscode.com/display/chef/Nodes

Nodes are JSON representations of machines under Chef management. They live in chef-server. They contain two important things: The node's run list, and a collection of attributes. The run list is a collection of recipes names that will be ran on the machine when chef-client is invoked. Attributes are various facts about the node, which can be manipulated either by hand, or from recipes.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Attributes

http://wiki.opscode.com/display/chef/Attributes

Attributes are arbitrary values set in a node object. Ohai provides a lot of informational attributes about the node, and arbitrary attributes can be set by the line cooks. They can be set from recipes or roles, and have a precedence system that allow you to override them. Examples of arbitrary attributes are listen ports for network services, or the names of a package on a particular linux distribution (httpd vs apache2).

Ohai

http://wiki.opscode.com/display/chef/Ohai

Ohai is the Chef equivilent of Cfengine's cf-know and Puppet's facter. When invoked, it collects a bunch of information about the machine its running on, including Chef related stuff, hostname, FQDN, networking, memory, cpu, platform, and kernel data. This information is then output as JSON and used to update the node object on chef-server. It is possible to write custom Ohai plugins, if your're interested in something not dug up by default.

Chef Client

http://wiki.opscode.com/display/chef/Chef+Client

Managed nodes run an agent called chef-client at regular intervals. This agent can be ran as a daemon or invoked from cron. The agent pulls down policy from chef-server and converges the system to the described state. This lets you introduce changes to machines in your infrastructure by manipulating data in chef-server. The pull (vs push) technique ensures machines that are down for maintenance end up the proper state when turned back on.

Resources

http://wiki.opscode.com/display/chef/Resources

Resources are the basic configuration items that are manipulated by Chef recipes. Resources make up the Chef DSL by providing a declarative interface to objects on the machine. Examples of core resources include files, directories, users and groups, links, packages, and services.

Recipes

http://wiki.opscode.com/display/chef/Recipes

Recipes contain the actual code that gets ran on machines by chef-client. Recipes can be made up entirely of declarative resources statements, but rarely are. The real power of Chef stems from a recipes's ability to search chef-server for information. Recipes can say "give me a list of all the hostnames of my web servers", and then generate the configuration file for your load balancer. Another recipe might say "give me a list of all my database servers", so it can configure Nagios to monitor them.

Cookbooks

http://wiki.opscode.com/display/chef/Cookbooks

Cookbooks allow you to logically group recipes. Cookbooks come with all the stuff the recipes need to make themselves work, such as files, templates, and custom resources (LWRPs).

Roles

http://wiki.opscode.com/display/chef/Roles

Roles allow you to assemble trees of recipe names, which are expanded into run lists. Roles can contain other roles, which serve as vertices, and recipe names, which are the leaves. The tree is walked depth first, which makes ordering intuitive when assembling run lists. It is possible to apply many of these trees to a single node, but you don't have to. Roles can also contain lists of attributes to apply to nodes, potentially changing recipe behavior.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Databags

http://wiki.opscode.com/display/chef/Data+Bags

Databags are arbitrary JSON structures that can be searched for by Chef recipes. They typically contain things like database passwords and other information that needs to be shared between resources on nodes. You can think of them as read only global variables that live on chef-server. They also have a great name that can be used to make various jokes in Campfire.

Knife

http://wiki.opscode.com/display/chef/Knife

knife is the CLI interface to the chef-server API. It can manipulate databags, node objects, cookbooks, etc. It can also be used to provision cloud resources and bootstrap systems.

Environments

http://wiki.opscode.com/display/chef/Environments

Environments in Chef provide a mechanism for managing different environments such as production, staging, development, and testing, etc with one Chef setup (or one organization on Hosted Chef). With environments, you can specify per environment run lists in roles, per environment cookbook versions, and environment attributes: allowing you to set policies to dictate which versions of a given cookbook may be used in an infrastructure environments.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Glossary



This page is a glossary of terms, technologies and concepts used by Chef and in working with Chef.

Each section describes one term and may link to other terms in this page. The "See Also" points to another page on the wiki, or to an external site.

API

Application Programming Interface. An interface implemented by software that enables it to interact with other software.

See Also:

- API on Wikipedia

Action

Providers in Chef take idempotent actions to configure resources

Attribute

Attributes are data about Nodes.

See Also:

- Attributes

Authentication

Chef clients authenticate to a Chef server using pre-shared RSA keys and signed HTTP headers.

See Also:

- Authentication

Auto-vivify

Internally in the Chef library, attributes are automatically created as methods.

See Also:

- Attributes

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Bootstrap

In the context of Chef, bootstrap means to get Chef installed and ready to run on the target system.

See Also:

- How to bootstrap new systems with Knife or How to bootstrap a Chef Client/Server

Client

The chef client communicates with a Chef server to download the cookbooks it needs to compile and run its configuration.

See Also:

- Chef Client.

Cloud

Cloud computing has been defined ad nauseum across the internet. Chef works well in several cloud computing environments such as Amazon EC2, Rackspace Cloud and Terremark vCloud.

See Also:

- Launch Cloud Instances with Knife
- Cloud computing on Wikipedia

Configuration Management

Setting up all the various components and services on a server so it can fulfill a role is configuration management.

See Also:

- Configuration Management on wikipedia

Convergence

The process by which systems are brought in line with the overall configuration management policy.

Cookbook

Chef cookbooks are packages for code used to configure some aspect of a system.

See Also:

- Cookbooks

CouchDB

A document-oriented database that provides a REST JSON API, used by the Chef Server as a back-end datastore for Node, Role, Client, and Data Bag data.

See Also:

- Apache CouchDB Project page

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

DSL

Programming or specification language dedicated to a specific problem domain. Chef makes use of meta-programming features in Ruby to create a simple DSL for writing Recipe, Role, and Metadata files.

Data Bag

Arbitrary store of JSON data that is indexed for Search.

See Also:

- Data Bags

Definition

Allow creation of new Resource macros that string together other resources.

See Also:

- Definitions

Environments

Provide a mechanism for managing different segmented spaces such as production, staging, development, and testing, etc with one Chef setup (or one organization on Hosted Chef): allowing you to set policies to dictate which versions of a given cookbook may be used within an Infrastructure segment.

See Also:

- Environments

File

Chef can manage static File and Directory assets as resources in recipes.

See Also:

- File Distribution
- cookbook_file Resource
- remote_file Resource
- remote_directory Resource

File Specificity

File specificity is a special order which Chef looks for host-, platform-version- or platform-specific files to use when downloading/configuring File and Template resources.

See Also:

- File Distribution#File Specificity

Git

Git is a distributed version control system.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

See Also:

- Git Home Page
- Pro Git Book
- GitHub's Git reference

Group

Groups are Hosted Chef-specific authorization objects that group together users in an organization. They are not used in the Open Source Chef Server.

See Also:

- Managing Groups with the Hosted Chef Management Console

Idempotent

A mathematical term that means multiple applications of the same action should not change the result.

Index

Most data (all but Cookbooks) stored on the Chef Server are indexed for Search.

See Also:

- Search

Infrastructure

Applications run on Infrastructure. Infrastructure in this context isn't physical or virtualized things like servers or networking. Rather, infrastructure is the application itself, plus all the underlying software prerequisites, server settings, tweaks, and configuration files need for it to function properly. Infrastructure typically spans nodes, and often networks.

See Also:

- Infrastructures.org

JSON

JavaScript Object Notation is a lightweight data format that is easy to read and write. All the APIs used in Chef are driven by JSON data.

See Also:

- JSON home page

Knife

A command-line tool used to work with a Chef Server and local Chef Repository.

See Also:

- Knife Documentation

Library

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

In a Chef cookbook, a Library is arbitrary Ruby code that can be used to extend Chef's language, or implement new features.

See Also:

- Libraries

Merb

Merb is a lightweight MVC framework used by the Chef Server to provide the API that clients communicate with.

See Also:

- Merb home page

Metadata

Chef cookbooks use metadata to provide hints to the Chef Server about what cookbooks should be deployed to a node, and can be used in user interfaces built on top of Chef.

See Also:

- Metadata

Node

A node is a system that is configured by Chef.

See Also:

- Nodes

Ohai

Ohai is a tool and library used by Chef to automatically detect information about the node it is running on.

See Also:

- Ohai

Operating System

An operating system manages hardware and provides services for running application software.

Hosted Chef

The Hosted Chef is a scalable, multi-tenant, secure, online service, supporting all the functionality of Chef and offered commercially by Opscode.

See Also:

- Opscode Home Page

Organization

In Hosted Chef, an organization represents a company, department or other grouped set of servers Infrastructure managed by Chef.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

See Also:

- Managing Organizations with the Hosted Chef Management Console

Platform

Chef detects the Operating System it is running on through Ohai, and uses that platform primarily to determine what Provider to use for particular resources.

Provider

A provider is an abstraction on top of system commands or API calls that is used to configure a Resource. Providers are often Operating System specific, such as the provider that installs packages for Debian (APT), Red Hat (Yum) or ArchLinux (Pacman).

See Also:

- Providers
- Lightweight Resources and Providers (LWRP)

Provision

The act of installing an operating system on bare metal, virtual machine or cloud computing instances is provisioning. Before Chef can configure a nd integrate systems, the system must first be provisioned, and then bootstrapped. This process varies by Operating System platform.

Queue

The Chef SOLR Search Index uses a queue for incoming data that needs to be indexed for search on the Chef Server.

See Also:

- Search

Recipe

A recipe is a Ruby DSL configuration file that you write to encapsulate resources that should be configured by Chef.

See Also:

- Recipes

Repository

A Chef Repository is a directory where you store all the various code used to configure your infrastructure with Chef.

See Also:

- Chef Repository

Resource

A resource is an abstraction that represents a particular thing that needs to be configured, such as a package or a service.

See Also:

- Resources

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- Recipes

REST

"REpresentational State Transfer (REST) is a style of software architecture for distributed hypermedia systems," commonly associated with HTTP. APIs conforming to REST constraints are "RESTful". Chef uses a RESTful API.

See Also:

- REST on Wikipedia

Role

A role describes a set of functionality for nodes through recipes and attributes.

See Also:

- Roles

Ruby

Ruby is an object-oriented programming language. Chef is written in Ruby and uses a number of Ruby DSLs for writing Recipe, Role, Metadata a s code.

See Also:

- Ruby Programming Language

Run List

A run list is an array of recipes and roles that should be applied, in order, on the Node, or in another Role.

See Also:

- Nodes
- Roles

SOLR

SOLR is a full text search engine platform written in Java.

See Also:

- Apache SOLR home page

Search

Data stored by the Chef Server is indexed for Search and can be queried with SOLR's Lucene search syntax.

See Also:

- Search

Server

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Shef

The Chef read-eval-print loop (REPL), Shef, is a way to run Chef in an IRB session. IRB is an interactive Ruby console.

See Also:

- Shef
- REPL on Wikipedia

Solo

Chef Solo is a standalone, non-client/server way to execute Chef recipes on nodes.

See Also:

- Chef Solo

Subversion

Subversion is a version control system.

See Also:

- Subversion home page

System Integration

The act of making disparate systems in an Infrastructure work together to provide application services and business value is system integration. It is where all the systems that have been configured are brought together to do their job.

Tags

Tags are an array Attribute of nodes.

See Also:

- Tags

Template

Chef uses ERB templates to create dynamically generated configuration files. The template files themselves are stored in cookbooks and generated using the Erubis library as it is faster than the default implementation of ERB in the Ruby standard library.

See Also:

- Templates
- Erubis home page
- ERB in the Ruby standard library

User

In the Open Source Chef Server, users log into the Webui Management Console.

In the Hosted Chef, users are credentialed entities used by humans to connect to Hosted Chef to manage the Organization.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

See Also:

- Management Console Users for Open Source Server
- Managing Users and your Private Key with the Hosted Chef Management Console

Webui

Hosted Chef and the Open Source Chef Server has a web user interface Management Console that can be used to view and modify various parts of the environment

See Also:

- Management Console

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Configuration Settings



This document describes all the configuration settings available for the Chef executables.

Chef Executables

Each Chef executable is configured through the Chef::Config object. When the executables are run, they load the default config file. Some settings are applicable only when the executable is run as a daemon, so whether daemonizing is possible is noted here.

Executable	Config File	Daemon?
chef-solo	`/etc/chef/solo.rb`	No
chef-client	`/etc/chef/client.rb`	Yes
chef-server	`/etc/chef/server.rb`	Yes
chef-server-webui	`/etc/chef/server.rb`	Yes
chef-solr	`/etc/chef/solr.rb`	Yes
chef-solr-indexer	`/etc/chef/solr.rb`	Yes
chef-solr-rebuild	`/etc/chef/solr.rb`	No
knife	`~/.chef/knife.rb`	No

If the file is not found, default values are used from Chef::Config.

In this page, settings are first listed in the Ruby DSL context, followed by a description of the setting and applicable executable context. Similar settings, such as those used only in the webui, are grouped together.



Ruby DSL
These files are written using a Ruby DSL that specifies the setting and its value. Some values will be specified as strings, others as Integers, Floats or Arrays.

Configuration values

amqp settings

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
amqp_host '0.0.0.0'
amqp_port '5672'
amqp_user 'chef'
amqp_pass 'testing'
amqp_vhost '/chef'
amqp_consumer_id "default"
```

These are for the AMQP host and vhost connection. For the consumer_id, `chef-indexer` is prepended automatically.

Context: chef-server, chef-solr, chef-solr-indexer

cache settings

```
cache_type "BasicFile"
cache_options({ :path => "/var/chef/cache/checksums", :skip_expires => true })
```

Checksums of files used in remote_file and template Resources are stored in a cache using Moneta, which can be configured using these options.

Context: chef-solo, chef-client

certificate settings

```
client_key "/etc/chef/client.pem"
validation_key "/etc/chef/validation.pem"
validation_client_name "chef-validator"
```

Authenticate the client using the file named in client_key. Use the valdiation_key to automatically register a new client with the server. The validation_key is signed using the validation_client_name for authentication. The validation_client_name specified for the server must match the same name when with the client configuration.

Context: chef-client, chef-server

chef_server_url

```
chef_server_url "http://localhost:4000"
```

Specifies the Chef Server to connect to. Can be any valid HTTP URL, including HTTPS. Learn more about the Chef Server.

When using the Opscode Platform, this will be `https://api.opscode.com/organizations/ORGNAME`, where ORGNAME is your organization short name that you created when signing up for the service.

Context: chef-client, chef-server, chef-server-webui, Knife

client_url

```
client_url "http://localhost:4042"
```

Specify the URL clients should use to connect to for registering with the server.

Context: chef-client

client_registration_retries

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
client_registration_retries 5
```

The number of times the client should attempt to register with the server.

Context: chef-client

cookbook_path

```
cookbook_path [ "/var/chef/cookbooks", "/var/chef/site-cookbooks" ]
```

A string or array of filesystem locations to search for cookbooks. Processed in the order specified so that the first is considered "upstream" and the last is considered overriding local modifications.

Context:

* chef-solo - Loads cookbooks from the specified directory.
* chef-client - Set to the 'cookbooks' subdirectory under `Chef::Config:file_cache_path` at runtime.
* chef-server - Used for storing cookbook tarball artifacts.
* knife - used for uploading, see Knife's documentation.

cookbook_tarball_path

```
cookbook_tarball_path "/var/chef/cookbook-tarballs"
```

Location where the Chef Server stores cookbook tarballs for distribution to clients.

Context: chef-server

couchdb_*

```
couchdb_database "chef"
couchdb_url "http://localhost:5984"
couchdb_version nil
```

Configure CouchDB settings for the Chef Server. These specify the database and URL to use, as well as the version. Chef attempts to determine the CouchDB version at runtime, and this setting is used to determine what API capability can be used.

Context: chef-server

data_bag_path

```
data_bag_path "/var/chef/databags"
```

Location where chef-solo can load data bag json files from.

Context: chef-solo

environment

```
environment "production"
```

Context:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

* chef-client - with -E or in the client.rb config file sets the environment for the node running chef.
* knife - with -E or in knife.rb sets the environment for relevant knife commands such as knife bootstrap, cloud creation, and more. See Knife's documentation.

file_cache_path

```
file_cache_path "/var/chef/cache"
```

Where to locally store cache files like cookbooks and other transient data.

Context:

* chef-solo - When downloading cookbooks from a remote URL, store them in this location.
* chef-client - Where synchronized cookbooks are stored.

This value can also be used in Recipes to download files with the remote_file resource.

file_backup_path

```
file_backup_path "/var/chef/backup"
```

Configurable location to store backups of files instead of the directory of the target file, for template, remote_file and file Resources.

Context: chef-solo, chef-client

user and group

```
user nil
group nil
```

System user and group to own the process. Applicable when starting any of the executables as a daemon.

Context: chef-client, chef-server, chef-server-webui, chef-solr, chef-solr-indexer

http_proxy*, https_proxy, no_proxy

```
http_proxy nil
https_proxy nil
http_proxy_user nil
http_proxy_pass nil
no_proxy nil
```

Specify the proxy server for HTTP/HTTPS with the `http_proxy` and `https_proxy` settings respectively. If the proxy server requires a username and password, use `http_proxy_user` and `http_proxy_pass`. For URLs that don't need a proxy, specify a comma separated list of URLs to exclude from proxying with `no_proxy`.

For example:

```
http_proxy "http://proxy.vmware.com:3128"
https_proxy "http://proxy.vmware.com:3128"
http_proxy_user "my_username"
http_proxy_pass "Awe_some_Pass_Word!"
no_proxy "*.vmware.com,10.*"
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Context: chef-client

http_retry_*

```
http_retry_count 5
http_retry_delay 5
```

Number of times to retry HTTP connections, and the delay between each retry, respectively.

Context: chef-client

interval and splay

```
interval nil
splay nil
```

Run the client on the specified interval, in seconds. The default value is 1800, configured in the chef-client application run time, rather than in the `Chef::Config`. The splay value is a random number of seconds to add to the interval to reduce potential server load of many clients running at the same time.

Context: chef-client

json_attribs

```
json_attribs nil
```

Specify JSON attributes, including a default run_list, for the node. Overrides whatever other attributes are set from other places like Cookbooks a nd Roles.

Context: chef-solo, chef-client

log output

```
log_level :info
log_location STDOUT
verbose_logging nil
```

Settings for log output. When Chef runs, it will display messages of the log_level or higher, to the specified log_location. If the log_location is set to another location than STDOUT, verbose_logging will tell Chef to also log to STDOUT, otherwise there would be no output other than to the file.

Chef uses Opscode mixlib-log library to handle logging. See Advanced Configuration Settings with Ruby for information on how to log Chef executable output to Unix syslog.

Valid log_levels are, in order of priority:

- :debug
- :info
- :warn
- :error
- :fatal

In Chef 0.10.6+, setting `verbose_logging` to `false` will suppress notifications about individual resources being processed. These notifications are output at the `:info` log level. Setting `verbose_logging` to `false` can be useful for running chef-client as a daemon. Below is an example of the difference in output when verbose_logging is set to `false`:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

verbose_logging set to true or nil

```
[Mon, 21 Nov 2011 09:36:50 -0800] INFO: *** Chef 0.10.6.rc.1 ***
[Mon, 21 Nov 2011 09:36:52 -0800] INFO: Setting the run_list to ["recipe[test-verbose-logging]"] from
JSON
[Mon, 21 Nov 2011 09:36:52 -0800] INFO: Run List is [recipe[test-verbose-logging]]
[Mon, 21 Nov 2011 09:36:52 -0800] INFO: Run List expands to [test-verbose-logging]
[Mon, 21 Nov 2011 09:36:52 -0800] INFO: Starting Chef Run for some_node
[Mon, 21 Nov 2011 09:36:52 -0800] INFO: Running start handlers
[Mon, 21 Nov 2011 09:36:52 -0800] INFO: Start handlers complete.
[Mon, 21 Nov 2011 09:36:53 -0800] INFO: Loading cookbooks [test-verbose-logging]
[Mon, 21 Nov 2011 09:36:53 -0800] INFO: Processing file[/tmp/test1] action create
(test-verbose-logging::default line 20)
[Mon, 21 Nov 2011 09:36:53 -0800] INFO: Processing file[/tmp/test2] action create
(test-verbose-logging::default line 21)
[Mon, 21 Nov 2011 09:36:53 -0800] INFO: Processing file[/tmp/test3] action create
(test-verbose-logging::default line 22)
[Mon, 21 Nov 2011 09:36:53 -0800] INFO: Processing file[/tmp/test4] action create
(test-verbose-logging::default line 23)
[Mon, 21 Nov 2011 09:36:54 -0800] INFO: Chef Run complete in 1.802127 seconds
[Mon, 21 Nov 2011 09:36:54 -0800] INFO: Running report handlers
[Mon, 21 Nov 2011 09:36:54 -0800] INFO: Report handlers complete
```

verbose_logging set to false

```
[Mon, 21 Nov 2011 09:37:38 -0800] INFO: *** Chef 0.10.6.rc.1 ***
[Mon, 21 Nov 2011 09:37:39 -0800] INFO: Setting the run_list to ["recipe[test-verbose-logging]"] from
JSON
[Mon, 21 Nov 2011 09:37:39 -0800] INFO: Run List is [recipe[test-verbose-logging]]
[Mon, 21 Nov 2011 09:37:39 -0800] INFO: Run List expands to [test-verbose-logging]
[Mon, 21 Nov 2011 09:37:39 -0800] INFO: Starting Chef Run for some_node
[Mon, 21 Nov 2011 09:37:39 -0800] INFO: Running start handlers
[Mon, 21 Nov 2011 09:37:39 -0800] INFO: Start handlers complete.
[Mon, 21 Nov 2011 09:37:40 -0800] INFO: Loading cookbooks [test-verbose-logging]
[Mon, 21 Nov 2011 09:37:41 -0800] INFO: Chef Run complete in 1.565369 seconds
[Mon, 21 Nov 2011 09:37:41 -0800] INFO: Running report handlers
[Mon, 21 Nov 2011 09:37:41 -0800] INFO: Report handlers complete
```

Context: All executables

node_name

```
node_name nil
```

Sets the name of this node, which determines what configuration to apply. Learn more about Nodes. Also sets the client_name, which is used as the name to use when authenticating with a Chef Server. Learn more about Authentication. The default value is set automatically to the fully qualified domain name (fqdn) as detected by Ohai.

Context: chef-solo, chef-client

node_path

```
node_path "/var/chef/node"
```

Location to look for node-specific recipes.

Context: chef-solo, chef-client

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

pid_file

```
pid_file nil
```

When started as a daemon, the executable will write the PID to the specified file. Otherwise, it will use `/tmp/executable.pid`.

Context: chef-client, chef-server, chef-solr, chef-solr-indexer

recipe_url

```
recipe_url nil
```

URL location of remote cookbook tarball to download with Chef Solo.

Context: chef-solo

rest_timeout

```
rest_timeout 300
```

Timeout for HTTP REST requests

Context: All executables.

role_path

Where Chef Solo should look for role files.

```
role_path "/var/chef/roles"
```

Context: chef-solo.

solo

```
solo  false
```

Whether Chef is being run in "solo" mode or not. Determines if it should attempt to communicate with a Chef Server. It gets set automatically when running chef-solo and if running Shef in solo mode.

Context: chef-solo, Shef.

solr settings

```
solr_url "http://localhost:8983"
```

URL of the Solr search engine server.

Context: chef-server, chef-solr, chef-solr-indexer

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
solr_jetty_path "/var/chef/solr-jetty"
solr_data_path "/var/chef/solr/data"
solr_home_path "/var/chef/solr"
solr_heap_size "256M"
solr_java_opts nil
```

Settings that control the Solr jetty environment. Points to the jetty environment, the Solr indexes, Solr's home directory, the amount of memory for the JVM running Solr and additional options to pass to Solr's Java, respectively.

Context: chef-solr, chef-solr-indexer

ssl settings

```
ssl_client_cert ""
ssl_client_key ""
ssl_ca_path nil
ssl_ca_file nil
```

Settings used to create certificate files used for client/server Authentication. These are OpenSSL X509 certificate, key and the CA.

These values are generated automatically and used internally. Most users won't need to modify these settings.

Context: chef-client, chef-server, chef-server-webui

```
ssl_verify_mode :verify_none
```

Controls HTTP verify mode, can be none or peer, for HTTPS requests.

Opscode Hosted Chef customers can use `:verify_peer` as the `ssl_verify_mode`. Depending on how OpenSSL was installed, the CA path may need to be specified. For example, on an Ubuntu system:

```
ssl_verify_mode :verify_peer
ssl_ca_path     '/etc/ssl/certs'
```

Context: All executables.

umask

```
umask 0022
```

Default file creation umask.

Context: All executables.

webui options

```
web_ui_client_name "chef-webui"
web_ui_key "/etc/chef/webui.pem"
web_ui_admin_user_name  "admin"
web_ui_admin_default_password "p@ssw0rd1"
```

The Chef Server WebUI is a client to the Chef Server API. It must have a corresponding client name and a client key, specified here. The WebUI default login user for humans is specified with the default password here. The password should be changed immediately when logging in.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

With Knife, the chef-webui client and certificate are used to create the initial API client with Knife's `-i` option.

Context: chef-server-webui, Knife

```
openid_cstore_couchdb false
openid_cstore_path "/var/chef/openid/cstore"
authorized_openid_identifiers nil
authorized_openid_providers nil
```

OpenID is now only used in the Chef Server WebUI. It is optional. WebUI login users can have an OpenID associated. The OpenIDs are not by default stored in CouchDB, instead in the specified location on the filesystem. The difference between identifiers and providers is:

- identifiers are the actual OpenID like username.myopenid.com or http://username.myopenid.com.
- providers are the OpenID provider like myopenid.com.

Context: chef-server-webui

signing_ca settings

```
signing_ca_cert "/var/chef/ca/cert.pem"
signing_ca_key "/var/chef/ca/key.pem"
signing_ca_user nil
signing_ca_group nil
signing_ca_country "US"
signing_ca_state "Washington"
signing_ca_location "Seattle"
signing_ca_org "Chef User"
signing_ca_domain "opensource.opscode.com"
signing_ca_email "opensource-cert@opscode.com"
```

Chef automatically generates the CA certificate and key for the Authentication keys it uses. These settings are passed to OpenSSL.

Context: chef-server

Unused Settings

The following unused settings never had their potential fulfilled and may be used in the future.

```
executable_path ENV['PATH'] ? ENV['PATH'].split(File::PATH_SEPARATOR) : []
run_command_stderr_timeout 120
run_command_stdout_timeout 120
```

Deprecated Settings

The following settings are deprecated and no longer used in Chef. They are kept here for historical purposes.

```
search_url "http://localhost:4000"
template_url "http://localhost:4000"
role_url "http://localhost:4000"
registration_url "http://localhost:4000"
remotefile_url "http://localhost:4000"
```

Specifies alternate URLs for various services provided by the Chef Server. "Deprecated" in favor of `chef_server_url`. These URLs can still be used to split out Chef Server API functionality, but that is an undocumented capability of the Chef Server.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
delay 0
```

Use `http_retry_delay` instead.

```
validation_token nil
```

Originally used to automatically validate new clients. Deprecated for `validation_key` and `validation_client_name`.

```
openid_store_path "/var/chef/openid/db"
openid_store_couchdb false
openid_providers nil
openid_url "http://localhost:4001"
```

These settings are deprecated in Chef 0.8+ due to the OpenID authentication scheme for nodes changing to the new pre-shared key based Authentication. OpenID is still used as an **optional** authentication method for users to the WebUI.

```
queue_host "localhost"
queue_password ""
queue_port 61613
queue_retry_count 5
queue_retry_delay 5
queue_user ""
```

These settings were used for configuring the STOMP queue, which has been deprecated for AMQP (RabbitMQ).

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Advanced Configuration Settings with Ruby

This document is a stub

This document is a stub, and requires more information and updating. See the discussion in the Chef Mailing List that led to its creation in the current state.
As approaches/solutions are developed, they should be added to this page.

Mixlib::Log provides a mixin for enabling a class based logger object, a-la Merb, Chef, and Nanite. To use it:

```
require 'mixlib/log'

class Log
  extend Mixlib::Log
end
```

You can then do:

```
Log.debug("foo")
Log.info("bar")
Log.warn("baz")
Log.error("baz")
Log.fatal("wewt")
```

By default, `Mixlib::Logger` logs to STDOUT. To alter this, you should call `Log.init`, passing any arguments to the standard Ruby Logger.

For example:

```
Log.init("/tmp/logfile")
# log to /tmp/logfile

Log.init("/tmp/logfile", 7)
# log to /tmp/logfile, rotate every day Enjoy!
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Just Enough Ruby for Chef

If you're new to Ruby, don't worry too much.

It's a language designed to be easy to read and to behave exactly as you'd expect, so it shouldn't take you too long to get up to speed.

First, it's useful to know how to check the syntax of a ruby file (like a cookbook):

```
$ ruby -c my_cookbook_file.rb
Syntax OK
```

If you're using the Chef Repository, you can just run 'rake test', which will remember which files have been tested based on mtime:

```
$ rake test
```

Now for a few basics:

Ruby Quick Reference for Clever Sysadmins

```
# anything after # is a comment.

# assigning a local variable:
x = 1

# some basic arithmetic:
1 + 2           # => 3
2 * 7           # => 14
5 / 2           # => 2   (because both arguments are whole numbers)
5 / 2.0         # => 2.5 (because one of the numbers had a decimal place)
1 + (2 * 3)     # => 7   (you can use parens to group expressions)

# strings
'single quoted' # => "single quoted"
"double quoted" # => "double quoted"
'It\'s alive'   # => "It's alive!"
"1 + 2 = 5"     # => "1 + 2 = 5" (numbers surrounded by quotes may exhibit string-like behavior)

# a string with embedded ruby
x = "Bob"
"Hi, #{x}"      # => "Hi, Bob"
'Hello, #{x}'   # => "Hello, \#{x}" Notice that single quotes don't work with #{}

# some basic truths
true            # => true
false           # => false
nil             # => nil
1 == 1          # => true ( == tests for equality )
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
1 == true       # => false ( == tests for equality )

# ! means not
!true           # => false
!false          # => true
!nil            # => true
1 != 2          # => true (1 is not equal to 2)
1 != 1          # => false (1 is not not equal to itself)

# !! (not not) converts something to either true or false
!!true          # => true
!!false         # => false
!!nil           # => false (when pressed, nil is false)
!!0             # => true (zero is NOT false).

# arrays are lists
x = ["a", "b", "c"] # => ["a", "b", "c"]
x[0]                # => "a" (zero is the first index)
x.first             # => "a" (see?)
x.last              # => "c"
x + ["d"]           # => ["a", "b", "c", "d"]
x                   # => ["a", "b", "c"] ( x is unchanged)
x = x + ["d"]       # => ["a", "b", "c", "d"]
x                   # => ["a", "b", "c", "d"]

# a hash is a list with keys and values
# - but no set order (!)
h = {
  "first_name" => "Bob",
  "last_name"  => "Jones"
}
# => { "first_name => "Bob", "last_name" => "Jones" }
h.keys              # => ["first_name", "last_name"]
h["first_name"]     # => "Bob"
h["last_name"]      # => "Jones"
h["age"] = 23
h.keys              # => ["first_name", "age", "last_name"]
h.values            # => ["Jones", "Bob", 23]

# perl-style regular expressions
"I believe"  =~ /I/                     # => 0 (matches at the first character)
"I believe"  =~ /lie/                   # => 4 (matches at the 5th character)
"I am human" =~ /bacon/                 # => nil (no match - bacon comes from pigs)
"I am human" !~ /bacon/                 # => true (correct, no bacon here)
/give me a ([0-9]+)/ =~ "give me a 7"     # => 0 (matched)

# you can do things conditionally

# with an if statement
if false
  # this won't happen
elsif nil
  # this won't either
else
  # code here will run though
end

# or a case statement
x = "dog"
case x
when "fish"
  # this won't happen
when "dog", "cat", "monkey"
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
  # this will run
else
  # the else is an optional catch-all
end

# def defines a method (functions, if you like)

def do_something_useless( first_argument, second_argument)
  puts "You gave me #{first_argument} and #{second_argument}"
end

do_something_useless( "apple", "banana")
# => "You gave me apple and banana"
do_something_useless 1, 2
# => "You gave me 1 and 2"
# see how the parens are optional if there's no confusion about what to do

# call a method on something with .method_name()
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
x = "My String"
x.split(" ")            # => ["My", "String"]
x.split(" ").join(", ") # => "My, String"
```

More Information

Ruby In Twenty Minutes



A small Ruby tutorial that should take no more than 20 minutes to complete.

The Power of Chef and Ruby

Opcode team member Bryan McLellan has a great personal blog post on **The Power of Chef and Ruby**

More in depth reading:
http://mislav.uniqpath.com/poignant-guide/
http://www.rubycentral.com/book/
http://www.ruby-doc.org/stdlib/

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Release Notes

Current Release Notes
This page contains the release notes for the current version series (e.g. 0.10.0+) of Chef, and earlier releases.

Release Notes - Chef - Version 0.10.8

Release announcement

Version 0.10.8 (1 issues)		
Type	**Key**	**Summary**
	CHEF-2819	Chef::ShellOut::Windows is exhibiting multiple issues in 0.10.6

Release Notes - Chef - Version 0.10.6

Release announcement

Version 0.10.6 (79 issues)		
Type	**Key**	**Summary**
	CHEF-2787	knife help search examples reference 'nodes' when it should be 'node'
	CHEF-2721	not_if and only_if work incorrectly for commands on windows
	CHEF-2712	rake install is broken in master due typo in Rakefile
	CHEF-2699	Don't depend on the value of an masgn
	CHEF-2684	Windows Ruby 1.9: uninitialized constant Chef::Mixin::Command::Windows::Open4
	CHEF-2676	Sandboxes#update fails if the sandbox is already complete
	CHEF-2675	Typos in Sandboxes#update
	CHEF-2674	Failing specs in chef-server-api
	CHEF-2672	Bad conditional in yum_spec.rb
	CHEF-2661	Chef::Cookbook::SyntaxCheck shouldn't shell_out to run erubis via sh
	CHEF-2660	chef-expander: undefined method `stop' for Chef::Expander::VNodeSupervisor:Class
	CHEF-2657	Failing specs in chef-expander
	CHEF-2655	Refactor Checksum to support more storage strategies
	CHEF-2652	solr-expander hardcodes solr_url
	CHEF-2651	solr_url ignores the path
	CHEF-2649	validation client should not be able to create admin clients
	CHEF-2647	improve file specificity with component versions
	CHEF-2637	yum provider - packages installed but non zero exit status

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-2634	FreeBSD packages: fail to build when using ports - incorrect use of port Makefile and working dir?
	CHEF-2626	Platform/distribution identification for Solaris derivitives
	CHEF-2607	Missing spec for knife configure client
	CHEF-2595	Can't use the --description option when creating a role
	CHEF-2588	webui status page doesn't respect environment selection
	CHEF-2580	unable to delete via the webui
	CHEF-2579	~ in cookbook_path fails to expand on cookbook create
	CHEF-2573	knife ssh interactive mode via tmux fails on OS X when search queries contain a colon
	CHEF-2571	mixlib-cli does not support negation of boolean parameters
	CHEF-2570	new :reconfig action for the package resource
	CHEF-2569	Cannot bootstrap CentOS instances because of Aegisco RPMs fixed to specifc version in bootstrap
	CHEF-2563	Missing specs for some of the knife cookbook commands
	CHEF-2559	--with-uri option broken for multiple commands
	CHEF-2553	chef-solr configs are packed within tar.gz
	CHEF-2549	Add start handlers to a chef-client run.
	CHEF-2548	Options list in knife manpages are out of date
	CHEF-2547	yum provider - arch in package name triggers provides search
	CHEF-2542	Missing specs for some of the knife client commands
	CHEF-2541	windows service provider reports success on a failed restart
	CHEF-2534	Chef debian package should depend on ucf
	CHEF-2532	Clean up spec output
	CHEF-2531	chef-expander, chef-expander-vnode argument parsing
	CHEF-2530	chef-solr-rebuild is broken
	CHEF-2519	remote_directory create always logging
	CHEF-2516	Shef::SoloSession#rebuild_context raises NameError for undefined local variable or method run_status in solo mode
	CHEF-2509	"knife configure -i" incorrectly adds the deprecated cookbook shadowing configuration
	CHEF-2492	Ifconfig provider doesn't allow modeling of onparent
	CHEF-2491	init scripts should implement reload
	CHEF-2462	chef-expander in 0.10.2 requires git
	CHEF-2439	"owner changed" info notice on cookbook_files is always repeated
	CHEF-2437	clear attribute list after each Knife::NodeShow.run
	CHEF-2434	bootstrap templates should transfer/rendering of encrypted_data_bag_secret in client.rb
	CHEF-2431	Present node JSON data in a similar manner to editing a node
	CHEF-2378	Chef expander's "rake install" fails with undefined method `version'

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- CHEF-2377 Broken client scenario "Synchronize dependent cookbooks"
- CHEF-2376 Broken client scenario "Remove cached file checksums that are no longer needed"
- CHEF-2363 knife node edit: "EOFError: end of file reached" on save
- CHEF-2358 shell_out should support passing environment variables on windows platform
- CHEF-2357 shell_out should support cwd on windows platform
- CHEF-2344 "knife configure -i" overwrites client key with 'nil' when run for an existing client name
- CHEF-2342 knife bootstrap script for centos should use rhel.rpm from aegisco to install yum repository
- CHEF-2339 Node search results disappear just after node performs save
- CHEF-2260 knife cookbook create should use markdown as default README file format.
- CHEF-2207 Add gemfile and development dependencies to chef client
- CHEF-2193 knife bootstrap - search for bootstrap templates in external knife plugins
- CHEF-2179 new log output for chef runs is a bit too verbose
- CHEF-2107 windows service provider works incorrectly on windows versions other than english
- CHEF-2061 knife bootstrap not finding add-on bootstrap dir
- CHEF-1959 file provider ignores create_if_missing
- CHEF-1958 Misleading error message when files entry isn't under "default"
- CHEF-1901 Rakefile for mixlib-cli should require yaml
- CHEF-1898 Knife error messages get sent to stdout
- CHEF-1830 knife bootstrap no longer prompts for password
- CHEF-1697 Chef service resource confused by services names with spaces under Windows
- CHEF-1677 Support csshX in knife ssh
- CHEF-1656 The knife man page has no description of the -F/--format options
- CHEF-1506 group resource should support 'system' similar to user
- CHEF-909 Rollback on deploy errors
- CHEF-630 Deploy should create the directories it needs if they don't exist
- CHEF-597 Debian service provider should use invoke-rc.d <script> rather than /etc/init.d/<script>
- CHEF-294 response_file handles grabbing a file via remote_file, but doesn't allow for dynamic preseeding through a template

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Release Notes - Chef - Version 0.10.4

Release announcement

Version 0.10.4 (78 issues)		
Type	**Key**	**Summary**
	CHEF-2325	Knife cookbook upload needs to verify that dependencies of a cookbook are/will be available when uploading
	CHEF-2493	New additional_remotes feature in git resource breaks deploy resource
	CHEF-1439	apt package provider should be able to detect and install 'virtual' packages
	CHEF-2518	NoMethodError running chef-solo
	CHEF-1245	Yum package provider fails poorly when it encounters unexpected output from yum-dump.py
	CHEF-2396	NoMethodError accessing non-existent members of Encrypted Data Bag Items
	CHEF-2105	"knife cookbook site vendor" in a git branch besides master it changes back to "master" and overwrites the cookbook
	CHEF-1054	Chef SOLR gem has an empty README.rdoc
	CHEF-1907	OSX Macports package provider does not respect options
	CHEF-1790	Modify package/yum.rb upgrade_package to support "downgrade" in certain situations
	CHEF-1982	mount provider fails to recognize existing /etc/fstab entries
	CHEF-2399	default group provider does not work on suse Linux
	CHEF-2301	general yum provider improvements
	CHEF-2235	chef-client failed when trying to install freebsd package from ports
	CHEF-1287	speed up yum provider
	CHEF-1576	yum gets confused with available packge in different arch
	CHEF-2416	Chef web ui is unable to edit data bag items
	CHEF-2316	knife help can't find its man pages outside of a gem file heiarchy
	CHEF-1819	Chef isn't recognizing yum .x86_64 syntax

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

CHEF-1424	Upstart provider fails restart action if service is not started yet
CHEF-2309	knife bootstrap throws optparse error for -E when bootstrapping less than 0.10.0
CHEF-2449	yum provider - slow parsing of yum-dump output
CHEF-2444	yum provider - yum-dump.py broken with older versions of yum
CHEF-1968	windows caching configuration problem
CHEF-1956	We should upgrade jQuery to 1.5.2 and remove old jQuery and jquery.livequery plugin
CHEF-2041	Allow file paths in package names to populate source attribute
CHEF-2222	mount resource issue with its "options" attribute
CHEF-2314	"knife cookbook site install" fails with cookbook name partially matching another cookbook name
CHEF-2456	knife cookbook site install -D description is incorrect and opposite behavior of -d from the old cookbook site vendor
CHEF-2354	Notifies :restart always triggers in remote_directory resource
CHEF-2293	bump rubygems version in bootstrap templates
CHEF-1413	FreeBSD "Unexpected form for PKGNAME variable in ..."
CHEF-1565	mount resource does not respect action :nothing
CHEF-311	yum provider - better cache refresh
CHEF-2300	net-ssh-multi dependancy it too restrictive
CHEF-2386	Make knife bootstrap (and ssh) use ui.msg for output
CHEF-39	Resources need a retry parameter which allows the action to repeat a set number of times until success is achieved.
CHEF-1946	Lightweight resources should have access to run_context and node during class definition, for use in DSL (e.g. attribute default values)
CHEF-2375	knife bootstrap needs a bootstrap-proxy option
CHEF-2311	Knife default display does not correctly display cloud providers public IP
CHEF-1891	When using rake new_cookbook, deprecation message should show correct syntax of creating cookbook via knife tool
CHEF-2096	Yum provider should support installing packages via provides when the name is not an exact match
CHEF-2350	rubygems spec relies on rspec version which changed its format in rspec2
CHEF-2379	Allow recipe to specify additional remotes to git resources
CHEF-2272	yum-dump.py - commas are valid in package names
CHEF-1063	yum query/yum-dump.py fails when a locally-compiled Python is in $PATH
CHEF-2085	YumCache needs unit tests
CHEF-2298	Add support for remote encryption key support to knife
CHEF-1848	http_request provider does not propagate request headers when following redirects
CHEF-2127	chef-client daemon does not pick up changes from chef server
CHEF-2471	knife exit specs fail on Ubuntu
CHEF-1838	Remove dead code from Chef::Runner
CHEF-1887	Mount provider should not check existence of Fuse devices

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- CHEF-1929 'knife ssh' should read config file for attribute, ssh-user, identity-file
- CHEF-1947 Alphabetical sort of keys in JSON editor
- CHEF-2312 webui doesn't show fully expanded run list as expected
- CHEF-2313 cookbook_uploader exception when exception is raised in uploader_function_for method
- CHEF-2307 knife or the chef-server ui to warn of cookbook download failure rather than silently fall back to previous version
- CHEF-2234 dpkg package provider ignores ~ in versions
- CHEF-519 value_for_platform should accept an array for version
- CHEF-1588 knife cookbook upload should have a -d option to upload the specified cookbooks dependencies
- CHEF-2129 Old zypper Versions will crash because they don'T know the commandline arguments
- CHEF-2394 Add data bag support for chef-solo
- CHEF-2378 Chef expander's "rake install" fails with undefined method `version'
- CHEF-2398 easy_install package provider doesn't use options attribute
- CHEF-2387 systemd service provider
- CHEF-2280 general yum-dump.py improvements
- CHEF-2271 knife client bulk delete not consistent with node delete
- CHEF-2330 yum provider - allow flushing of cache
- CHEF-2373 GET /environments/:environment/recipes fails if there is a cookbook with no available versions
- CHEF-2295 README.rdoc in Chef source is outdated
- CHEF-2283 yum/rpm providers - rpm version handling
- CHEF-2432 yum-dump - provides broken on centos 4
- CHEF-2429 yum provider - remote provides loaded for :remove and :purge
- CHEF-2428 yum provider raises an exception when asked to remove an uninstalled package
- CHEF-2310 knife ssh documentation missing -i option
- CHEF-2474 When loading a role from a json file, a role object is not returned
- CHEF-2367 support multiple lines in DAEMONS list in rc.conf on Arch linux

Release Notes - Chef - Version 0.10.2

Release announcement

Version 0.10.2 (1 issues)		
Type	**Key**	**Summary**
	CHEF-2436	Permissions not checked when maniupulating cookbookds

Release Notes - Chef - Version 0.10.0

Release announcement

Version 0.10.0 (225 issues)		

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Type	Key	Summary
	CHEF-1802	chef server broken with rack-1.2.1
	CHEF-1808	RabbitMQ connection retry broken in multi-queue indexing
	CHEF-1836	knife cookbook purge fails if checksum documents are missing from couchdb
	CHEF-1845	Git Provider fails when the directory you want to clone the repo to already exists
	CHEF-1861	chef-client hanged on windows after running around 100 times
	CHEF-1879	Resource#inspect uses all available memory
	CHEF-1905	Numeric#fdiv is not available on ruby 1.8.6
	CHEF-2044	Chef 0.10 server components should be easily installable via chef cookbook and chef-solo
	CHEF-2084	Roles create/edit form broken in the WebUI
	CHEF-2125	Shef is broken
	CHEF-2132	Cookbook Uploads must check users' dependency syntax and help them update to the newer syntax
	CHEF-2142	knife cookbook * -- uninitialized constant Chef::REST
	CHEF-2147	knife uninitialized constant Chef::DataBagItem
	CHEF-2155	execute resource, creates attribute -- undefined method
	CHEF-2153	knife bootstrap -- uninitialized constant
	CHEF-2154	knife data bag from file -- uninitialized constant
	CHEF-2152	knife cookbook site vendor -- uninitialized constant
	CHEF-2162	knife rackspace server create -- uninitialized constant
	CHEF-2177	knife cookbook site install -- undefined local variable or method `cookbook_path'
	CHEF-2188	Knife cookbook upload fails when an environment is not specified
	CHEF-2190	Chef::Environment::Mash nameerror on knife environment create production
	CHEF-2195	knife bootstrap scripts should install a specific version of chef
	CHEF-2198	Cookbook uploads broken
	CHEF-2211	Chef 0.10 and Ruby 1.8.6 issues
	CHEF-2206	knife recipe list is broken
	CHEF-2217	Knife should turn off colors on windows
	CHEF-2219	core bootstrap templates need to all support environments
	CHEF-2229	chef-solr-installer fails to chown
	CHEF-2224	Chef no longer saves the node before executing the resource collection
	CHEF-2239	knife ssh role:blah tmux fails with uninitialized constant Chef::Mixin::Command (NameError)
	CHEF-2258	Chef::Handler::ErrorReport causes a deprecation volcano to erupt
	CHEF-2265	knife bootstrap fails with uninitialized constant Chef::Mixin::Command (NameError)
	CHEF-2277	Object#tap monkey patch not loaded

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- CHEF-2289 EncryptedDataBagItem causes name error (missing require) when called from recipe code
- CHEF-2236 run_context not propagated to GemPackage during deploy provider run
- CHEF-1623 knife bootstrap mentions [RUN LIST] in the banner, but its an option
- CHEF-1844 A node's attributes says it is a kind_of? Hash, but does not support all Hash methods
- CHEF-1852 Better error message when moving a temporary file to a directory that doesn't exist
- CHEF-2130 Chef should enforce a sane path setting
- CHEF-2159 Chef::REST should set a unique user agent for chef-client, knife, and webui
- CHEF-2175 uninitialized constant Chef::Knife::Core::NodeFormattingOptions
- CHEF-2200 CookbookVersionSelector loads all data for all cookbook versions in the system
- CHEF-2210 Cookbook uploads should not rely on generating metadata.json
- CHEF-2227 chef-solr-indexer files in distro should be updated for chef-expander
- CHEF-2228 chef-expander daemon names are confusing
- CHEF-2290 knife data bag * commands have non-standard underscore in argument names
- CHEF-2092 Deploy resource with SVN failing in Chef 0.9.14
- CHEF-1469 Refactor UI internals for knife
- CHEF-1154 marsal data too short errors from moneta
- CHEF-1248 Plugin System for Knife
- CHEF-1276 Apt package provider doesn't distinguish between installed and available versions in the debug messages
- CHEF-1290 Cookbook loading must support versioned dependencies
- CHEF-1311 Metadata generation should be deterministic
- CHEF-1407 changes in update-rc.d breaks Chef::Provider::Service::Debian
- CHEF-1405 extlib gem dependancy provides bugs in rails application
- CHEF-1423 Support versions of recipes on the run list
- CHEF-1446 Improve error message for knife cookbook site vendor when working dir is not a git clone of chef-repo
- CHEF-1508 nested roles and override_attributes
- CHEF-1526 knife cookbook site vendor gives no indication of unmet dependencies
- CHEF-1562 Knife should not issue warnings when successfully updating a role
- CHEF-1583 Chef Server API root page throw errors
- CHEF-1584 redhat init.d chef-client script does not write pidfile
- CHEF-1587 node.recipe? was removed, no easy way to check for included recipes
- CHEF-1590 Chef isn't compatible with Amazon's Virtual Private Cloud
- CHEF-1600 rubygems provider should not force you to use rubygems.org as a gem source
- CHEF-1620 fog is not listed as an dependency but is used by knife/tests - similar to net-ssh-multi
- CHEF-1628 Inconsistent Documentation

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- CHEF-1633 does not work on centos with python 2.6
- CHEF-1634 easy_install provider can't install some packages
- CHEF-1682 Knife fails with NameError (Fcntl::F_SETFD) when uploading a cookbook on Win32 build of Ruby 1.8.7
- CHEF-1703 Xenserver is not listed in platform
- CHEF-1704 knife uses deprecated fog class Fog::AWS::EC2
- CHEF-1708 Upgrade Solr to 1.4.1
- CHEF-1717 knife cookbook show fails to display file / part contents
- CHEF-1715 Add support for Blue Box Group Blocks
- CHEF-1757 create knife subcommands for OpenStack
- CHEF-1781 IOError - closed stream in erl_call provider
- CHEF-1777 chef-client logging to stdout doesn't sync buffer under runit
- CHEF-1794 Cannot disable services on debian lenny
- CHEF-1818 Unify cookbook version syntax and version contraints
- CHEF-1834 debian service provider cannot enable a disabled service
- CHEF-1833 Not ignoring cookbook version constraints from dependency metadata when expanding cookbook dependencies
- CHEF-1835 run_interval feature test is unreliable as it depends on ps|grep for determining the chef-client child process
- CHEF-1832 knife status should handle a null ohai_time instead of failing noisily
- CHEF-1858 Modify load path and use a standard require instead of path-based require in chef.gemspec
- CHEF-1868 Centos bootstrap template references non-existant URL for EPEL repo
- CHEF-1878 WebUIUser design documents are not created when running feature tests, making WebUI unusable
- CHEF-1925 Data Bag Item inspect is not verbose enough
- CHEF-1924 Possible Regression -- Cookbook Not Found errors not suggesting metadata as a possible cause
- CHEF-1926 Git Provider has a confusing error message when accessing a non-existant repo over http(s)
- CHEF-1931 inconsistency specifying run list for knife ec2 server create and bootstrap
- CHEF-1930 Subversion resource fails if the destination directory isn't already a working copy
- CHEF-1937 Teach knife ssh tmux not to craete empty windows and to keep separate invocations in separate tmux sessions
- CHEF-1932 Knife SSH macterm error prone
- CHEF-1938 chef/distro boot scripts for archlinux
- CHEF-1943 'knife bootstrap' broken on chef-0.9.12
- CHEF-1945 Chef should support simple encrypted data bags
- CHEF-1955 Shef does not correctly configure Chef::Cookbook::FileVendor causing template and cookbook file resources to fail.
- CHEF-1988 DataBagItem.load returns a Hash and not a DataBagItem
- CHEF-1993 Version (only latest) shouldn't be shown on the Cookbooks list page
- CHEF-1999 Make Environments CRUD on the WebUI pretty and shippable

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

CHEF-2002	subversion provider checkout action not idempotent
CHEF-2022	Remove Cookbooks and Nodes tabs under Environment in the WebUI
CHEF-2023	Knife cookbook list should respect to the environment selected and show only the ones available in the selected environment.
CHEF-2032	Create a separate installer for chef-solr, and make the chef-solr executable responsible only for running solr.
CHEF-2028	/cookbooks and /environment/:env_id/cookbooks should return data in consistent format and the returned data structure should be extensible
CHEF-2039	Empty array should not be shown as a cookbook name in the cookbook version constraints picker on the environment create/edit form
CHEF-2043	chef-solr-installer doesn't respect paths in /etc/chef/solr.rb
CHEF-2033	verbose logger
CHEF-2049	Add multiple cookbook version and environment filtering to cookbook list/show in webui
CHEF-2067	knife rackspace_server_* need to support additional config option (for UK-based accounts)
CHEF-2076	RFE: option to prevent upload of a cookbook with the same version
CHEF-2080	Shell_out should not set @cwd to Dir.tmpdir by default
CHEF-2081	chef-0.9.14.rc.1 yum provider fails to correctly select packages when arch is provided
CHEF-2078	0.9.14.rc.1 - shef -z does not load run_list, contrary to behavior documented on wiki
CHEF-2090	DSL methods need to check arguments. In particular, Language#data_bag_item
CHEF-2083	0.9.14.rc.1 - unexpected nil referring to prior revision in git provider called from deploy provider
CHEF-2103	Search result is incorrect with syntax (NOT...) AND ... and (NOT...) ...
CHEF-2102	Need correct URI encoding and decoding of the query string for search
CHEF-2126	Don't show attributes the user shouldn't edit in knife node edit
CHEF-2137	extract class for knife subcommand loading
CHEF-2150	Knife should not use log levels to control output verbosity
CHEF-2164	Implement retries for 503
CHEF-2165	Knife should hide stack traces when not running with max verbose
CHEF-2173	Knife's default data output should be less noisy and verbose
CHEF-2170	remove retries for 403
CHEF-2174	knife cookbook site share -- undefined method 'validate_cookbook'
CHEF-2171	colorize message tags in knife, create option for turning colors off, make interrupts quieter
CHEF-2180	knife ssh - NameError: uninitialized constant Chef::Knife::Ssh::Readline
CHEF-2183	Environments need default and override attributes
CHEF-2185	enforce_path_sanity tries to modify frozen string (ENV["PATH"]) with "<<"
CHEF-2181	knife bootstrap - not showing colorized output
CHEF-2223	knife "Cannot find sub command for:" error should be more friendly
CHEF-2230	Environment search is broken in the WebUI.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- CHEF-2244 data doesn't make it to solr when re-indexing data with chef-expander
- CHEF-2243 knife index rebuild doesn't reindex environments
- CHEF-2281 Pass in inflated object instead of raw data from client / webui when saving a data bag item
- CHEF-2294 centos5 bootstrap template should still install EPEL
- CHEF-1187 Knife cookbook upload will break in windows, because the cookbook uploader assumes '/' as the file separator
- CHEF-1996 Create and edit node supports environment selection
- CHEF-1916 chef_environment attribute cannot be displayed with knife node show
- CHEF-1473 Add ability for SIGUSR1 to wakeup the chef daemon and start a chef run
- CHEF-1635 Debian/Ubuntu Chef package doesn't have a versioned dependency on rubygems
- CHEF-1074 /etc/init.d/chef-solr on centos doesn't set pidfile correctly
- CHEF-1521 enhance knife cloud functionality
- CHEF-2124 Knife should not load plugins from old gems
- CHEF-1853 Debian init scripts for chef-client don't pass on $PIDFILE option
- CHEF-1254 Implement cookbook version constraints in metadata dependencies per the wiki
- CHEF-1129 "knife cookbook upload" also bundles temporary files, could be avoided by a configurable parameter?
- CHEF-1292 JSON nesting error is not indicated in user interfaces
- CHEF-1319 knife node tag (add|remove)
- CHEF-1445 knife rackspace server creation should use bootstrap subcommand
- CHEF-1540 New rake tasks databag:upload[databag], databag:upload_all, databag:create[databag], databag:create_item[databag,item]
- CHEF-1558 knife rackspace command should be able to take and list flavors and distros instead of relying on numeric IDs
- CHEF-1612 "knife cookbook upload" uploads .svn/* files
- CHEF-1615 Chef/Windows/Ruby 1.9.2 incompatibility
- CHEF-1651 archlinux bootstrap template needs base-devel
- CHEF-1660 Create init scripts for archlinux
- CHEF-1679 Service provider on Windows does not acknowledge failure
- CHEF-1690 Mac OS X Server platform provider support
- CHEF-1765 make knife ec2 support :availability_zone in config
- CHEF-1767 AIX group provider
- CHEF-1763 make knife ec2 support :region in config
- CHEF-1803 file mode regex is overly restrictive
- CHEF-1807 run_interval functional test sometimes waits indefinitely for the child chef-client to exit
- CHEF-1821 Unable to knife search for negative queries
- CHEF-1840 Unclear message generated by chef-client when purging unused files
- CHEF-1851 Features tests will sometimes fail at beginning due to 412 from creating chef_integration; retrying solves it.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

CHEF-1850	In features couchdb replicate, change debug level of message about retrying of couchdb DB create
CHEF-1866	@api_search features tests don't work
CHEF-1919	knife bootstrap throws uninitialized constant Net::SSH (NameError)
CHEF-1954	Shef in client mode reports that it's in solo mode
CHEF-1990	Ability to set initial EBS device size and boot from it
CHEF-2003	chef-client should have an -E / --environment option that sets Chef::Config[:environment]
CHEF-2011	Relative path to client_key not handled well
CHEF-2013	Ability to specify the branch to merge back to when performing a knife cookbook vendor
CHEF-2042	chef-solr-indexer is now chef-expander
CHEF-2057	Rename Chef::JSON to avoid conflict with ::JSON class
CHEF-2062	yum provider claims to install nonexistent versions if specified
CHEF-2073	knife cookbook create should use FileUtils.mkdir_p instead of shell_out to run mkdir -p
CHEF-2135	cookbook_version.metadata and cookbook_version.manifest["metadata"] can be out of sync due to support for deprecated metadata syntax
CHEF-2172	move non-subcommand classes from chef/knife/FOO into chef/knife/core/FOO
CHEF-2187	friendly knife node show run list output isn't entirely friendly
CHEF-2191	knife-openstack uses fog's deprecated #ip_address
CHEF-2209	No need to dump the list of all dbs every time we create a design document
CHEF-2214	data_bag_item and data_bag DSL methods should continue to support the data bag name to be passed in using format :foo
CHEF-2225	Cookbook deprecation feature and integration with knife
CHEF-2233	Run list cannot be correctly specified with Windows client running knife ec2 server create
CHEF-1714	knife text format should dump lists in plain text
CHEF-1862	knife status doesn't show the IP on windows 6.0.6002
CHEF-2408	knife documentation is out of date for logging
CHEF-1935	When chef-client is running via runit w/ log_location STDOUT, it buffers log output and doesn't write to the log file
CHEF-442	Reclarify existing log messages levels, give more information in INFO mode.
CHEF-606	Runner#run_action duplicates code from Resource#run_action
CHEF-1691	Javascript alert message when deleting a client is wrong
CHEF-1718	create artifacts for testing apt repository in feature tests
CHEF-1769	improve chef-solr startup detection in debian init script
CHEF-1800	Color escape sequences do not work on Windows
CHEF-1864	Feature tests for apt package provider failing
CHEF-1904	EPEL URL in centos knife configuration now 404
CHEF-1964	archlinux-gems needs to preserve chef environment

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-2000	default group provider does not work on HP-UX
	CHEF-2026	specify other kinds of licenses with knife cookbook crate
	CHEF-2231	WebUI: Available recipes list in Node/Role creation/edit form should sort alphabetically
	CHEF-1727	knife using deprecated fog syntax as of fog 0.3.7
	CHEF-2091	spaces between items in the comma-separated run_list fails during knife bootstrap
	CHEF-2111	Replace per-cookbook ignores with per-repo ignores
	CHEF-2145	Back compat for cookbooks api is broken
	CHEF-2140	Chef server should rescue search query parse errors and return 400 instead of 500
	CHEF-2143	Search query transformation not handling wildcard character '?' properly
	CHEF-2146	"knife cookbook bulk delete .*" fails with 0.10.0 beta 2
	CHEF-2176	knife environment list `<class:Environment>': uninitialized constant Chef::Mixin::FromFile (NameError)
	CHEF-2178	When Chef 0.9.x is installed, attempting to load plugins that were formerly in core causes a load error
	CHEF-2184	Scrub log output at INFO level
	CHEF-2189	Set arbitrary HTTP headers for Chef::Rest via config
	CHEF-2194	knife-openstack doesn't deal with nil items in the image list
	CHEF-2203	"knife role bulk delete .*" fails with 0.10.0 beta 8
	CHEF-2201	Make man pages accessible via knife
	CHEF-2213	features/support/env.rb requires spec/expectations which changed in rspec
	CHEF-2208	inconsistent environment output with search, node list in 0.10.0.beta.8
	CHEF-2254	"Exception: NameError: uninitialized constant Chef::EncryptedDataBagItem" when creating an encrypted data bag item
	CHEF-2262	Solr Indexer: dies randomly every so often and doesn't properly restart on its own
	CHEF-2269	knife cookbook list --show-all fails with 0.10.0 rc1
	CHEF-2204	"knife cookbook list" with an empty list throws an error with Chef 0.10 beta 8
	CHEF-2430	knife ssh gets stuck on sockets in CLOSE_WAIT

Release Notes - Chef - Version 0.9.18

Release announcement

Version 0.9.18 (5 issues)		
Type	**Key**	**Summary**
	CHEF-2436	Permissions not checked when maniupulating cookbookds
	CHEF-2274	Shef does not seem to include the chef libraries
	CHEF-2234	dpkg package provider ignores ~ in versions
	CHEF-2129	Old zypper Versions will crash because they don'T know the commandline arguments
	CHEF-2367	support multiple lines in DAEMONS list in rc.conf on Arch linux

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Release Notes - Chef - Version 0.9.16

Release announcement

Version 0.9.16 (14 issues)		
Type	**Key**	**Summary**
	CHEF-2195	knife bootstrap scripts should install a specific version of chef
	CHEF-2159	Chef::REST should set a unique user agent for chef-client, knife, and webui
	CHEF-1154	marsal data too short errors from moneta
	CHEF-1584	redhat init.d chef-client script does not write pidfile
	CHEF-1757	create knife subcommands for OpenStack
	CHEF-1938	chef/distro boot scripts for archlinux
	CHEF-2067	knife rackspace_server_* need to support additional config option (for UK-based accounts)
	CHEF-2090	DSL methods need to check arguments. In particular, Language#data_bag_item
	CHEF-2100	Input to Solr not correctly escaped
	CHEF-1768	add ssh port CLI parameter to knife commands
	CHEF-1558	knife rackspace command should be able to take and list flavors and distros instead of relying on numeric IDs
	CHEF-1612	"knife cookbook upload" uploads .svn/* files
	CHEF-2189	Set arbitrary HTTP headers for Chef::Rest via config
	CHEF-2212	centos5-gems bootstrap template rubygems version issues

Release Notes - Chef - Version 0.9.14

Release announcement

Version 0.9.14 (78 issues)		
Type	**Key**	**Summary**
	CHEF-1802	chef server broken with rack-1.2.1
	CHEF-1836	knife cookbook purge fails if checksum documents are missing from couchdb
	CHEF-1845	Git Provider fails when the directory you want to clone the repo to already exists
	CHEF-1861	chef-client hanged on windows after running around 100 times
	CHEF-1905	Numeric#fdiv is not available on ruby 1.8.6
	CHEF-1332	shef fails to find cookbooks
	CHEF-1844	A node's attributes says it is a kind_of? Hash, but does not support all Hash methods
	CHEF-2069	chef-solr: Depends: solr-jetty (>= 1.4.0) which is a virtual package on debian squeeze
	CHEF-1248	Plugin System for Knife
	CHEF-1276	Apt package provider doesn't distinguish between installed and available versions in the debug messages
	CHEF-1407	changes in update-rc.d breaks Chef::Provider::Service::Debian
	CHEF-1562	Knife should not issue warnings when successfully updating a role

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-1587	node.recipe? was removed, no easy way to check for included recipes
	CHEF-1633	does not work on centos with python 2.6
	CHEF-1634	easy_install provider can't install some packages
	CHEF-1703	Xenserver is not listed in platform
	CHEF-1704	knife uses deprecated fog class Fog::AWS::EC2
	CHEF-1717	knife cookbook show fails to display file / part contents
	CHEF-1715	Add support for Blue Box Group Blocks
	CHEF-1781	IOError - closed stream in erl_call provider
	CHEF-1777	chef-client logging to stdout doesn't sync buffer under runit
	CHEF-1794	Cannot disable services on debian lenny
	CHEF-1801	Add rake tasks to generate the GEMNAME.gemspec files, and check them in to git
	CHEF-1834	debian service provider cannot enable a disabled service
	CHEF-1835	run_interval feature test is unreliable as it depends on ps|grep for determining the chef-client child process
	CHEF-1832	knife status should handle a null ohai_time instead of failing noisily
	CHEF-1858	Modify load path and use a standard require instead of path-based require in chef.gemspec
	CHEF-1868	Centos bootstrap template references non-existant URL for EPEL repo
	CHEF-1878	WebUIUser design documents are not created when running feature tests, making WebUI unusable
	CHEF-1892	issues with easy_install package installation status and version
	CHEF-1925	Data Bag Item inspect is not verbose enough
	CHEF-1926	Git Provider has a confusing error message when accessing a non-existant repo over http(s)
	CHEF-1930	Subversion resource fails if the destination directory isn't already a working copy
	CHEF-1937	Teach knife ssh tmux not to craete empty windows and to keep separate invocations in separate tmux sessions
	CHEF-1932	Knife SSH macterm error prone
	CHEF-1955	Shef does not correctly configure Chef::Cookbook::FileVendor causing template and cookbook file resources to fail.
	CHEF-1988	DataBagItem.load returns a Hash and not a DataBagItem
	CHEF-2002	subversion provider checkout action not idempotent
	CHEF-2040	erl_call doesnt cause exception when code fails
	CHEF-2046	files are constatly downloaded from server (windows)
	CHEF-2066	In some conditions, the retry logic in rest.register method causes unbalance of client keypair
	CHEF-2080	Shell_out should not set @cwd to Dir.tmpdir by default
	CHEF-2081	chef-0.9.14.rc.1 yum provider fails to correctly select packages when arch is provided
	CHEF-2078	0.9.14.rc.1 - shef -z does not load run_list, contrary to behavior documented on wiki
	CHEF-2083	0.9.14.rc.1 - unexpected nil referring to prior revision in git provider called from deploy provider
	CHEF-2098	debian json package far too old

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- CHEF-2104 Git provider error message is very confusing when the revision specified does not exist
- CHEF-1756 bootstrap scripts should set a better $PATH
- CHEF-1853 Debian init scripts for chef-client don't pass on $PIDFILE option
- CHEF-1890 include support for SELINUX filesystem labels
- CHEF-1292 JSON nesting error is not indicated in user interfaces
- CHEF-1316 rabbit-mqserver is not started on install, breaking dependency-based install
- CHEF-1445 knife rackspace server creation should use bootstrap subcommand
- CHEF-1575 Portage provider should search /var/db/pkg/*/* if a category is not specified
- CHEF-1651 archlinux bootstrap template needs base-devel
- CHEF-1660 Create init scripts for archlinux
- CHEF-1679 Service provider on Windows does not acknowledge failure
- CHEF-1758 improve arch bootstrap script
- CHEF-1765 make knife ec2 support :availability_zone in config
- CHEF-1763 make knife ec2 support :region in config
- CHEF-1803 file mode regex is overly restrictive
- CHEF-1807 run_interval functional test sometimes waits indefinitely for the child chef-client to exit
- CHEF-1851 Features tests will sometimes fail at beginning due to 412 from creating chef_integration; retrying solves it.
- CHEF-1850 In features couchdb replicate, change debug level of message about retrying of couchdb DB create
- CHEF-1866 @api_search features tests don't work
- CHEF-1919 knife bootstrap throws uninitialized constant Net::SSH (NameError)
- CHEF-1954 Shef in client mode reports that it's in solo mode
- CHEF-2013 Ability to specify the branch to merge back to when performing a knife cookbook vendor
- CHEF-2057 Rename Chef::JSON to avoid conflict with ::JSON class
- CHEF-2062 yum provider claims to install nonexistent versions if specified
- CHEF-1691 Javascript alert message when deleting a client is wrong
- CHEF-1718 create artifacts for testing apt repository in feature tests
- CHEF-1769 improve chef-solr startup detection in debian init script
- CHEF-1863 apt package provider debug output about current vs candidate is confusing
- CHEF-1904 EPEL URL in centos knife configuration now 404
- CHEF-2026 specify other kinds of licenses with knife cookbook crate
- CHEF-1727 knife using deprecated fog syntax as of fog 0.3.7
- CHEF-2110 Debian package of chef 0.9.14 fails to install on debian lenny

Release Notes - Chef - Version 0.9.12

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Release announcement

Version 0.9.12 (11 issues)		
Type	**Key**	**Summary**
	CHEF-1480	User resource appends '-r' option (system user) even when running 'usermod' in 'manage_user', which doesn't support such an option.
	CHEF-1759	Sometimes knife ssh fails because of fqdn attribute is missing from the node object returned form search
	CHEF-1792	Post-run cookbook cleanup is getting triggered on chef-solo, deleting files from cookbooks
	CHEF-1484	New gem provider is incompatible with rubygems 1.2.0
	CHEF-1706	Accessing the node before Shef has finished loading causes shef to reload infinitely
	CHEF-1785	shef -z
	CHEF-1179	Ruby 1.9 Support on Chef Server, WebUI, Solr and Solr Indexer
	CHEF-1444	knife ec2 server create should use the public fqdn for ssh
	CHEF-876	require 'json' interferes with require 'generator' (which is required for Chef::Mixin::Retry)
	CHEF-1110	Enable if_exists option for File (remote_file, template) resources.
	CHEF-1629	update CONTRIBUTING doc to new wiki location

Release Notes - Chef - Version 0.9.10

Release announcement

Version 0.9.10 (87 issues)		
Type	**Key**	**Summary**
	CHEF-1286	JSON Attribs and precedence
	CHEF-1322	Need to rescue timeouts in chef-solr-indexer
	CHEF-1452	Executes with action :nothing still trigger immediate notifications
	CHEF-1505	Chef Solr also needs to hide the net/http bug (undefined method `closed?' for nil:NilClass)
	CHEF-1532	knife ec2 server create broken on ruby 1.9
	CHEF-1536	knife cookbook metadata assumes cookbook_path is an array though it's allowed to be a string
	CHEF-1567	Update mixlib config dependency to the latest
	CHEF-1607	Platform cookbook version detection appears broken
	CHEF-1671	Bug in the current Apt Package provider - not handling dependencies right
	CHEF-1741	Nodes saved without :roles attribute during chef-client run
	CHEF-1743	notifies seem to trigger on cookbook_file even when not run
	CHEF-1747	Cookbook version 11.0.0 and such cause troubles
	CHEF-1746	Fix for CHEF-1344 makes chef run incorrectly under runit and upstart
	CHEF-1751	Knife rackspace server create and bootstrap don't correctly set the ssh password on the underlying knife ssh command
	CHEF-1753	using dot-style in an attributes file results in a NoMethodError
		"Knife (node|role|client|data bag) edit" doesn't think an object is changed after returning from editor, so doesn't actually

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

CHEF-1745	save it
CHEF-1397	chef-client does not clean up cached template checksums
CHEF-1364	Cannot purge the checksum documents from couchdb
CHEF-1152	Scientific Linux missing from chef/lib/platform.rb
CHEF-767	Git resource setting updated too often, causing notifications to trigger inappropriately
CHEF-871	Chef::Node::Attribute can't handle false/nil
CHEF-1422	rpm_package returns NoMethodError
CHEF-1571	Uploading new version of cookbook fails on first attempt, succeeds on second attempt
CHEF-1581	Chef Solr Indexer is slow
CHEF-899	Service can't depend on config file that restarts service
CHEF-1302	using content parameter with file results in empty file getting created, then clobbered
CHEF-1333	Route resource puts in "via" when gateway is not specified
CHEF-1442	Rescue 'no acceptor' from thin and re-raise with a clearer error message
CHEF-1460	knife --version prints a FATAL message for no reason
CHEF-1488	Remote Directory resource should support not overwriting existing files
CHEF-1485	knife ec2 server create hangs on bootstrap
CHEF-1507	New resource and provider: ohai (there should be a way to reload ohai data in a recipe)
CHEF-1520	Remove the requirement to click the 'login' button all the time on the Login screen by enabling 'default' behaviour on Enter keypress.
CHEF-1518	Action:export does not work for subversion
CHEF-1519	only_if and not_if aren't checked when resource is run by a notifies or subscribes
CHEF-1531	Knife ec2 server create should give the option to specify node name
CHEF-1534	cookbook_version.rb defines several methods multiple times
CHEF-1533	knife exec - run scripts or snippets of code with chef configured
CHEF-1537	Cannot Determine if root user is locked on CentOS
CHEF-1544	yum provider improperly tries to update lesser versions of packages
CHEF-1542	route provider is improperly using @collection
CHEF-1556	Cannot see Files content of cookbooks in webui
CHEF-1560	'knife cookbook upload' fails to upload freshly generated metadata.json
CHEF-1564	knife bootstrap should always try to install chef
CHEF-1570	knife "cookbook site share" throws NoMethodError
CHEF-1577	action :head support for http_request provider
CHEF-1601	Chef solo should not reset json attributes for each run
CHEF-1599	RSA key format check is too strict
CHEF-1613	default group provider does not work on Solaris

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- CHEF-1610 Refactor Chef::Node::Attribute#method_missing
- CHEF-1616 Chef Solr Indexer should not inflate objects it receives from the queue
- CHEF-1632 Windows mount provider does not assign local drive letter for network shares
- CHEF-1639 Chef can't load under Rubinius because of a missing io/wait
- CHEF-1646 Spurious 403s on first chef-client run coming from S3 cookbook download
- CHEF-1654 knife ec2 server create fails in bootstrap.rb line 118. undefined method 'first'
- CHEF-1702 Ability to pass flags when spawning process from the abstract script provider
- CHEF-1728 Chef does not apply any command line configuration options if it has no config file
- CHEF-1969 remote_file combined with the debug parameter shows every % of file downloaded
- CHEF-1417 use the node name vs fqdn in "knife status"
- CHEF-1569 New knife "windows bootstrap" subcommand
- CHEF-1167 Request to create functionality to perform bulk operation on a list of nodes
- CHEF-1344 chef-client run with a tty should not use interval and splay in client.rb
- CHEF-1538 knife ssh doesn't support the -P or --ssh-pasword options correctly
- CHEF-1539 Minor tweak for rest-client upgrade
- CHEF-1548 stack level too deep error appears resolved in json upstream at v1.4.4+
- CHEF-1578 provider/link spec test failure
- CHEF-1580 user::dscl provider tests fail with Errno::ENOENT: No such file or directory - dscl . -list /Users uid
- CHEF-1582 knife configure client should use IO objects instead of system cp
- CHEF-1606 knife cookbook site vendor <cookbook> fails without cookbook path specified
- CHEF-1614 Use DIETIME/STARTTIME in debian init scripts to give processes time to restart
- CHEF-1617 log provider does not set new_resource.updated flag
- CHEF-1619 bootstrap template for archlinux (w/ gems)
- CHEF-1664 Roles are not properly highligted in the node edit interface.
- CHEF-1667 support amazon linux ami (platform.rb) requires OHAI-216
- CHEF-1673 Chef::REST Spec failure, added newline for certificate
- CHEF-1709 Modify cucumber tests to use manual CouchDB replication instead of Couch's _replicate URL
- CHEF-1733 knife status should take an optional query to limit results
- CHEF-1739 Fix manual couchdb replication and organize it so it can be re-used for other projects
- CHEF-1798 Resource parser doesn't evaluate definitions
- CHEF-1722 Incorrect Regex in Cucumber test for checksum cache cleanup
- CHEF-462 http_request should allow for setting HTTP Header
- CHEF-1529 file_spec expects Chef::Config.file_backup_path to be writeable
- CHEF-1543 Solaris returns "Not owner" instead of "Operation not permitted" in daemon_spec.rb

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-1579	Chef::REST.stream_to_tempfile should only log progress when verbose logging is configured
	CHEF-1574	rubygems bootstrap script templates should use rubygems 1.3.7
	CHEF-1605	spec/unit/provider/service/redhat_spec.rb broken on systems without /sbin/service - false positives when running the test suite
	CHEF-1675	Knife status improvements (ip, fqdn, run list)

Release Notes - Chef - Version 0.9.8

Release announcement

Version 0.9.8 (70 issues)		
Type	**Key**	**Summary**
	CHEF-1216	add 10.04 to ubuntu versions to check in upstart provider
	CHEF-1337	Chef is too 1337 4 u
	CHEF-1416	Knife cookbook delete is broken for all cases except when a specific version is specified
	CHEF-1441	Typo in amqp client fix
	CHEF-1448	specs broken by recent commits
	CHEF-1474	Extra node saves in Chef::Client can leave node data in a broken state
	CHEF-1486	bundle the rackup files with the gem for unicorn users
	CHEF-1496	Bump required ohai version to 0.5.6
	CHEF-1498	knife cookbook create should use CLI opts instead of ARGs
	CHEF-1499	update knife markdown and man page for 0.9.8 changes
	CHEF-1500	knife bootstrap's sudo option conflicts with general chef-server-url option (-s)
	CHEF-1503	knife ec2 server create needs to pass bootstrap options for template use
	CHEF-1513	Deprecation notice for remote_file -> cookbook_file should print the line where the resource was declared
	CHEF-1501	bootstrap template for ubuntu 10.04 using apt.opscode.com packages
	CHEF-1514	Value for platform fails when the value to be returned is an array
	CHEF-1185	Mixlib authentication should be have a less cryptic failure message
	CHEF-1369	save the expanded run list into a searchable property on nodes
	CHEF-1415	link resource is broken when using shef in client mode
	CHEF-1451	Negative UID heuristic fails for UIDs between 2**30 and 2**31
	CHEF-1453	Knife status is broken on Ruby 1.9
	CHEF-1458	Trash and rebuild the Chef::Client for every run
	CHEF-1470	Refactor Knife's subcommand loader
	CHEF-1479	Chef::Provider::Group overwrites the new resource's gid with the existing gid
	CHEF-1487	Configure the name setting in merb so chef-server and chef-server-webui show up in the process listing
	CHEF-1497	Remove as much monkey patching as possible from shef

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-1145	filenames with brackets, filenames regular expression escape
	CHEF-187	Mount Resource should handle LABEL and UUID
	CHEF-366	Ability to specify service priority on action enable
	CHEF-587	no package provider for Solaris
	CHEF-1100	Yum provider doesn't want to install packages with specified architecture
	CHEF-1111	possible unclosed handle in easy_install package provider
	CHEF-1115	chef-solo incorrectly requires /etc/chef/client.pem
	CHEF-1176	Need a way to support http_proxy shell setting for remote_file and other resources
	CHEF-1215	knife ssh subcommand often does nothing after receiving search query results
	CHEF-1223	execute provider does not work on Windows platforms
	CHEF-1312	webui is missing a lot of admin checks
	CHEF-1329	Readable inspect for Chef::Node::Attribute
	CHEF-1343	Add an option to 'knife configure -i' commands to accept the defaults.
	CHEF-1345	Refactor help output for knife
	CHEF-1351	remove dependency on the merb-slices gem
	CHEF-1359	chef-server -v does not report version
	CHEF-1391	support region option in ec2 commands
	CHEF-1393	Extend knife bootstrap to support templates
	CHEF-1412	missing uuidtools dep?
	CHEF-1438	value_for_platform fails on 1.9.2
	CHEF-1450	Shef can't resume chef run due to change on Session
	CHEF-1447	The script provider does not account for possible spaces in the interpreter or script path on Windows
	CHEF-1449	Moving cookbook creation task to Knife, and deprecating the rake task.
	CHEF-1456	dead code in Resource#to_text
	CHEF-1461	Use Knife to share/unshare cookbook to/from the Opscode Cookbooks Site
	CHEF-1462	Factor out cookbook uploader from Knife::CookbookUpload
	CHEF-1481	Add support for Chef::Mixin::Command.popen4 on Windows platforms
	CHEF-1504	chef source should have a NEWS file
	CHEF-1502	Shef can masquerade as other nodes
	CHEF-1512	DataBagItem#to_s uses obsolete @name variable so it always prints 'data_bag_item[]'
	CHEF-1511	give shef a pretty interface to the api
	CHEF-1509	file provider backup fails on Windows platforms
	CHEF-1517	knife node run_list add broken on 0.9.8 beta
	CHEF-1525	open handle fix in CHEF-1111 causes easy_install candidate version to always be 256

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-1524	improve easy_install previously installed package detection
	CHEF-1557	update centos/redhat packages to 0.9.8
	CHEF-184	chef-client debug output doesn't display the version
	CHEF-301	Directories with a dot (.ssh for example) aren't copied with the remote_directory resource
	CHEF-886	knife subcommands should take --help argument and show sub-command specific help
	CHEF-1082	Configuration settings for stomp server are obsolete and should be removed
	CHEF-1386	Knife bootstrap. Should support specifying a ssh private key file
	CHEF-1387	JSON attribs aren't loaded correctly on first run of chef-client
	CHEF-1455	Resource#to_text triggers a deprecation warning
	CHEF-1516	Unable to delete cookbooks on server when there are multiple versions
	CHEF-1430	Ruby_block resources should be able to send notifications to other resources

Release Notes - Chef - Version 0.9.6

Release announcement

Version 0.9.6 (7 issues)		
Type	**Key**	**Summary**
	CHEF-1404	Shef no longer gets chef/recipe pulled in with its other requires, causing it to crash on start
	CHEF-1410	chef-solo does not always set Chef::Config[:solo] to true, Chef::Application::Solo should require 'chef'
	CHEF-1260	file resource checksums everything
	CHEF-1412	missing uuidtools dep?
	CHEF-1411	knife cookbook upload should be more resilient to 400's from the sandboxes controlled caused by S3 index inconsistency
	CHEF-1495	update centos/redhat packages to 0.9.6
	CHEF-1432	metadata.rb does not accept symbol as an attribute type

Release Notes - Chef - Version 0.9.2 and 0.9.4

A backwards compatibility issue was discovered in 0.9.2, and 0.9.4 was released immediately.

Release announcement

Version 0.9.4 (2 issues)		
Type	**Key**	**Summary**
	CHEF-1400	Need to support deprecated usage of the @node instance variable within resources
	CHEF-1432	metadata.rb does not accept symbol as an attribute type

Version 0.9.2 (14 issues)		
Type	**Key**	**Summary**
	CHEF-1337	Chef is too 1337 4 u
	CHEF-1365	Expectation of return code 0 is incorrect for chkconfig

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-1390	Upstart Script for WebUI runs the wrong command
	CHEF-1396	Template provider does not set access controls correctly when creating/updating a file
	CHEF-1394	Notification Handlers are broken
	CHEF-1363	ChefServerApi::SandboxFile assumes input is a StringIO
	CHEF-1380	update chef-server-webui's gemspec to require merb 1.1.x
	CHEF-1384	Template resource looks for the raw template file in the wrong part of the cache
	CHEF-1366	Add slicehost support to knife
	CHEF-1373	TypeError: true can't be coerced into Fixnum when you upload cookbooks
	CHEF-1378	Add Upstart Scripts
	CHEF-1388	Move monkey patches to individual files so they can be required individually
	CHEF-1432	metadata.rb does not accept symbol as an attribute type
	CHEF-1381	Error in reporting error in knife

Release Notes - Chef - Version 0.9.0

Release announcement

Version 0.9.0 (89 issues)		
Type	**Key**	**Summary**
	CHEF-1072	chef-server-webui incompatible with merb 1.1.0
	CHEF-1168	RubyGems 1.3.7 will introduce an issue where Chef's gem_package won't be able to install arch-specific packages
	CHEF-1210	updated man pages, init scripts and supporting distro specifics in source to make packaging consistent
	CHEF-1232	knife --help doesn't display full help installed as debian package or w/ debian's rubygems
	CHEF-1236	DeepMerge fails to merge production data
	CHEF-1246	Override attributes from roles get written to the node
	CHEF-1253	Solr configuration uses dynamicFields Declaration, creates havoc during indexation
	CHEF-1269	The webui needs to be updated for cookbook and run list changes
	CHEF-1273	Chef::Mixin::Language uses defunct @node should use node (method)
	CHEF-1275	knife doesn't work without highline, and highline is not in the gemspec
	CHEF-1277	Create a chef-server meta-gem
	CHEF-1280	Cookbook upload dies on some files
	CHEF-1291	Deploy resource seems to re-apply the whole recipe stack, not just the portion we specify
	CHEF-1288	@node can no longer be accessed directly in recipes, so we need to provide a check for this on cookbook upload
	CHEF-1294	Cookbook uploads that are only metadta changes fail
	CHEF-1296	Regression: We download all cookbook files, even those we may not need.
	CHEF-1295	cookbook file for preseed files in package resource needs to have its run context set
	CHEF-1301	gem_package prints gem installation messages on stdout with log

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

CHEF-1304	Setting attributes fails when a hash of another precedence has an intermediate value
CHEF-1307	chef-client does not error when non-existing role encountered in run_list
CHEF-1331	Update knife's manpage source
CHEF-1336	Chef::REST should paper over a bug in net/http
CHEF-1337	Chef is too 1337 4 u
CHEF-1341	rewrite the handler API
CHEF-1342	Updates to mixlib-config interact poorly with method stubs, resulting in spec failures
CHEF-1346	Add elapsed time to the list of methods that Chef::Handler delegates to @run_status
CHEF-1305	ShellOut segfaults older ruby patchlevels
CHEF-1104	intermittent closed stream error on packages and templates
CHEF-1289	API does not check for admin rights for user management
CHEF-1293	Knife cookbook delete should default to latest if a version is not specified
CHEF-1308	File Cache purging may incorrectly purge or not purge cookbook files (nee remote files) and templates
CHEF-1335	Mixlib Config is defining methods on a metaclass of a metaclass, causing config_attr_writer to fail
CHEF-1325	knife cookbook download should not fail when a version is not specified
CHEF-1361	knife search -a option does not handle nil attributes gracefully
CHEF-1358	The source line in resource objects is always recipe_definition_dsl_core.rb
CHEF-585	no service provider for Solaris
CHEF-682	Add an exception notification hook
CHEF-702	chef server should 404 when recipes can't be found
CHEF-838	attributes set with the "default" keyword should not be persisted to the node, and should have lower precedence
CHEF-979	Include status module for knife
CHEF-1041	Cookbook (up)loader should be version aware
CHEF-1161	undefined method `cookbook_loader=' for nil:NilClass
CHEF-1189	Authenticated subversion checkouts fail due to prompt
CHEF-1211	Versions of packages should be specified in at most one place per sub-project
CHEF-1219	Data bag item should throw Chef::Exceptions::ValidationFailed when validation failed
CHEF-1221	Exception if package needs updating (zypper)
CHEF-1220	knife ssh tmux
CHEF-1231	Implicit "::default" for include_attribute broken
CHEF-1228	env provider
CHEF-1230	Knife : -f from --format overwrites -f from other options
CHEF-1234	remote_file does not work with binary files on Windows platforms
CHEF-1243	Remote file should be deprecated for fetching cookbook files--this should be a cookbook file resource/provider

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

CHEF-1258	Knife ec2 support
CHEF-1263	The docs/ directory in chef should be removed
CHEF-1259	Knife should support per-directory configuration files
CHEF-1264	Chef::Solr::Query initialization should take couchdb object instead of just the database
CHEF-1267	Add mount provider for Windows platforms
CHEF-1272	Merge the completed work on chef 1269 into master so we can release alpha 4
CHEF-1270	chef-solr-indexer dies converting an argument error to a string (possibly ruby1.9 related)
CHEF-1271	The client needs to trust the server's provided manifest when fetching cookbooks
CHEF-1262	Add user and group providers for Windows platforms
CHEF-1268	Cookbooks should sync on a file-by-file basis, support versioning, and be updated atomically
CHEF-1281	Knife cannot upload a cookbook without metadata
CHEF-1300	chef-server Rakefile should have install / uninstall tasks
CHEF-1299	knife configure should ask different question for client name based on whether -i is specified or not
CHEF-1306	knife cookbook site vendor fails to extract due to incorrect cwd
CHEF-1314	Non existent roles are silently skipped in run_list expansion when running chef-client
CHEF-1326	500 error when attempting to show a cookbook with a bad/non-existent version
CHEF-1323	when chef tries to match a process against the ps output, it should print the regex with #inspect and not #to_s
CHEF-1324	Bring back the syntax check cache
CHEF-1347	knife recipe list shows '.rb' at the end of the recipe names
CHEF-1348	undefined local variable or method full_recipe_list in views/nodes/show.html.haml
CHEF-1353	Cookbook uploading fails
CHEF-45	Some attributes (e.g. Ohai's) are effectively immutable and should be read-only, lockable or namespaced.
CHEF-289	Initial JSON endpoint in REST API
CHEF-349	Refactor rubygems provider to use Gem classes, internal rubygems API instead of calling out to the CLI
CHEF-771	node :edit page slow to load in webui
CHEF-914	knife data bag subcommands should include data bag from file
CHEF-1026	Updated cookbooks stored even though cookbooks haven't been updated
CHEF-1085	cookbook loader fails somewhat silently when given an invalid cookbook
CHEF-1096	backups of config files gets included in various programs
CHEF-1130	sync distro work from rpm packages
CHEF-1141	Nodes webui screen needs to be sorted by name.
CHEF-1282	Specs throw warning "parenthesize argument(s) for future version"
CHEF-1320	Teach knife ssh screen to respect user's .screenrc
CHEF-1318	/files lists nonexistent files when source is a substring of another source directory

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- CHEF-1309 rubygems providers tests don't run on older versions of rubygems
- CHEF-1432 metadata.rb does not accept symbol as an attribute type
- CHEF-1381 Error in reporting error in knife

Release Notes - Chef versions prior to 0.9.0

Release Notes Older Versions

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Breaking Changes from 0.7.x to 0.8.x

This page documents the breaking changes from Chef version 0.7.x and earlier to Chef version 0.8.x



Without a doubt...
The biggest breaking change is that the 0.8 series is not backwards compatible between client and server to **any** previous Chef version. Make sure you follow the instructions for Upgrading Chef 0.7.x to 0.8.x, and remember that the Server will need to be upgraded before the clients.

Authentication

We have dramatically altered the Authentication model, and now rely on digitally signing each request. What you need to know is this: when you upgrade your Chef Server, it will create a "Validation Key" called `validation.pem`. This key should be put in `/etc/chef/validation.pem` o n all your Chef Clients, and will be used to allow them to generate their individual client keys. Once a client has registered, the validation key should be removed from it. Josh Timberman has a recipe that can do this automatically for you.

REST API

The Server API has been made more consistent, and now exists for every operation the Chef Server can perform. It is also dramatically different in many places from the previous API - so if you have tools that hit the Chef Server REST API, make sure you update them.

cookbook_path now has a more consistent order

The configuration option 'cookbook_path' has been reversed, so that it goes in order from least-to-most specific. In Chef 0.7, if you had:

Chef 0.7 cookbook_path configuration setting

```
cookbook_path [ "/var/chef/site-cookbooks", "/var/chef/cookbooks" ]
```

You should now have:

Chef 0.8 cookbook_path configuration setting

```
cookbook_path [ "/var/chef/cookbooks", "/var/chef/site-cookbooks" ]
```

Search

The syntax for Searching inside of Chef Recipes has changed. Previously, it was possible to return only certain attributes from a search:

Chef 0.7 Search

```
search(:node, "ipaddress:192*", [ :hostname, :fqdn ]) do |item|
  ...
end
```

In this example, the result was always a list of items that matched, and the item itself was simply a hash with the data requested.

With the API updates in Chef 0.8, we now return full objects from search queries - this makes utilizing them much more consistent (from a user point of view) but means that it's no longer reasonable to return sub-sets of the data. In Chef 0.8:

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef 0.8 Search

```
search(:node, "ipaddress:192*") do |node|
  node[:hostname]
end
```

Will work. Notice that we're now getting the full node object itself. 

In addition, the creation of custom search indexes is replaced by Data Bags. If you were using the previous custom search index functionality, move to Data Bags.

See the full Search documentation for more about the search feature of Chef.

Run Lists, not Recipe Lists

In Chef 0.7 we introduced the concept of a Run List, which allows you to mix Recipes and Roles. Due to the awesome work of Thom May, Chef 0.8 allows Roles to contain other Roles. This means that where in Chef 0.7 you would have written:

Chef 0.7 Recipe List

```
node.recipes [ "apache2", "unicorn" ]
```

You should now be writing:

Chef 0.8 Run List

```
node.run_list [ "recipe[apache2]", "role[webserver]" }
```

Your previous code will still work, but we will log a warning every time we encounter it, and expect it to be deprecated in the next major Chef release.

CouchDB 0.8 unsupported

CouchDB 0.8 is no longer supported by the Chef Server. Opscode provides a recipe for installing CouchDB 0.9.1+ from source, and this is handled on supported server platforms in the bootstrap. When upgrading CouchDB, you need to dump the database from the old version and load it from the dump file.

- CouchDB Breaking Changes
- Third party dump/restore tool for CouchDB databases.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Release Notes Older Versions

Older Release Notes
This page contains release notes for older releases of Chef. For the latest version series (e.g., 0.9.0+), see Release Notes

Release Notes - Chef - Version 0.8.16

Release announcement

Version 0.8.16 (9 issues)		
Type	**Key**	**Summary**
	CHEF-1196	Commit bc411a1ded418a385af23bbec7d1cc6b013cb08b breaks template owner setting
	CHEF-1207	Leftover debug output in chef-solo
	CHEF-1337	Chef is too 1337 4 u
	CHEF-1204	Mixlib CLI update breaks shef
	CHEF-1208	CHEF-1207 breaks chef-client
	CHEF-878	Put guards in require()s in the Rakefile so gems can be built w/o dev dependencies installed
	CHEF-1194	WebUI - show of cookbook fails
	CHEF-1297	knife ssh commands missing net-ssh package installation
	CHEF-1432	metadata.rb does not accept symbol as an attribute type

Release Notes - Chef - Version 0.8.12 and 0.8.14

We encountered a hiccup in the publishing process for 0.8.12, and moved right along to 0.8.14.

Release announcement

Version 0.8.14 (8 issues)		
Type	**Key**	**Summary**
	CHEF-1184	JSON gem version 1.4.3 does not work with ohai or Chef::Node, a dependency on 1.4.2 or less needs to be added to the gem spec
	CHEF-1337	Chef is too 1337 4 u
	CHEF-1188	Chef::Client squashes errors in its save_node method, destroying valuable debugging info
	CHEF-966	useradd provider should support creation of system users
	CHEF-1021	Pass environment, group and cwd to run on deploy
	CHEF-1191	hotfix for all applications always cd-ing to root leaves a failing test
	CHEF-600	Refactoring RubyGems provider to use output_of_command for better diagnostics
	CHEF-1432	metadata.rb does not accept symbol as an attribute type

Version 0.8.12 (51 issues)

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Type	Key	Summary
	CHEF-674	uuidtools gem needs to be packaged for debian
	CHEF-751	mixlib-authentication needs version tags
	CHEF-1108	attribute note for streaming_cookbook_uploader.rb from author
	CHEF-1106	remove suse copyrighted init script
	CHEF-1173	clients can turn themselves into admins
	CHEF-1172	file_spec.rb test is time zone dependent
	CHEF-1181	knife configure needs to be updated to reflect that the validator is no longer an administrator
	CHEF-1337	Chef is too 1337 4 u
	CHEF-489	Ruby 1.9 Support for Chef
	CHEF-785	svn_arguments not used when svn info is called (deploy resource)
	CHEF-1050	FreeBSD: chef commands freeze during ohai calls
	CHEF-1053	Chef source code repository need a CONTRIBUTING file
	CHEF-1091	chef_repo rake task fails to build metadata
	CHEF-424	Chef's Tempfile leaves lot of chef-rest files in /tmp
	CHEF-557	Remove rubygems from any libraries
	CHEF-641	Improvements to RPM packages
	CHEF-675	Allow for rendering file contents with a 'content' variable
	CHEF-845	Wrong number of arguments in preseed_package
	CHEF-1008	validation key should not be an admin key
	CHEF-1025	mdadm provider is broken
	CHEF-1060	Deep merge json attributes passed on the command line
	CHEF-1087	chef-solr truncating logs, not logging startup
	CHEF-1092	chef problems with dropping privileges
	CHEF-1099	chef-solr problems with dropping privileges
	CHEF-1105	bring back the ruby/template test cache
	CHEF-1107	Revision Provider for Deployment Resource cannot recover if cache file storing deployed revisions is lost
	CHEF-1121	Group/user creation / null group append fails on OSX
	CHEF-1126	Deleting a role is extremely slow when using couchdb 10
	CHEF-1155	Allow 'execute' resource to accept multiple return values without error.
	CHEF-1157	Make Chef-Client run on Windows with cookbooks using file, directory, remote file, remote directory, and template providers
	CHEF-1160	Fresh chef install on Ruby 1.8.7 can't run rake test
	CHEF-1166	There is no need to save the node after syncing cookbooks

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-1150	"knife configure -i" uses hardcoded validation key path
	CHEF-1164	Service Provider on Windows
	CHEF-1151	Chef Solr should not run ohai in its startup sequence
	CHEF-1170	File lookup problems cause specs to fail on ruby 1.9.2 when run with rake
	CHEF-1174	Knife output format: Allow more simple output for single attribute
	CHEF-1178	rake role[role_name] does not work at all
	CHEF-632	remote_file resource should display an INFO message at start of a download
	CHEF-669	optional rubygems?
	CHEF-724	Chef should complain when it can't find any cookbooks
	CHEF-721	support for purging managed directories
	CHEF-1030	Properly escape existing cron-job names
	CHEF-1032	rake test no longer performs syntax check on ruby and erb files
	CHEF-1432	metadata.rb does not accept symbol as an attribute type
	CHEF-510	be able to use debian/ubuntu libjs-jquery packaged library for chef-server
	CHEF-791	Help text for -T refers to openid
	CHEF-804	make knife 'configure' give an example of the server url requested
	CHEF-1001	gem provider specify's the source of 'gems.rubyforge.org' if the source is not specified, which is a deprecated mirror...
	CHEF-1109	Add descriptive error message identifying the file when an invalid metadata.json is loaded from a cookbook.
	CHEF-1120	Regex validation in the params validation mixin should print regex.inspect instead of regex.to_s in the error message

Release Notes - Chef - Version 0.8.10

Release announcement

Version 0.8.10 (24 issues)		
Type	**Key**	**Summary**
	CHEF-468	chef-server version needs to not rely on walking gem objectspace
	CHEF-554	chef-server/indexer should run as 'chef' user in packaging
	CHEF-833	recursive copy of cached-copy in deploy resource fails on symlinked file
	CHEF-946	if rabbitmq is down, the generated web ui and validation keys are lost
	CHEF-1039	chef-solr doesn't actually log to specified log file.
	CHEF-1044	Nested arrays of hashes break node indexing
	CHEF-1064	Installing gems with an implicit --remote flag on gem fails because of merb 1.1.x
	CHEF-1337	Chef is too 1337 4 u
	CHEF-252	If a template's source doesn't end in .erb, but exists, we still get a 500 error.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-653	Cron resource fails when the program takes a numerical argument.
	CHEF-735	Web Slice needs a override config file.
	CHEF-828	Mixlib CLI - Preserve ARGV after @opt_parser.parse!
	CHEF-1048	Creating an Invalid Role in WebUI causes 500
	CHEF-1057	Support templates that are already on the file system
	CHEF-1069	Chef::Node does not properly validate the name parameter
	CHEF-1081	Chef::Config should specify an amqp_consumer_id by default
	CHEF-1077	Running the features should not require sudo (mac-dev-start)
	CHEF-1083	chef-server-webui doesn't correctly rescue load error when it tries to load itself from source checkout
	CHEF-1076	Role error messages are non-helpful to track down which role caused the problem (when loading roles from filesystem)
	CHEF-425	portage provider in Gentoo reinstalls the packages over and over again; does not matter if :install or :upgrade is used.
	CHEF-527	debian/ubuntu packages should use /etc/default/chef-{client,server} for settings to the init script
	CHEF-782	debian packaging should include -e production
	CHEF-1432	metadata.rb does not accept symbol as an attribute type
	CHEF-510	be able to use debian/ubuntu libjs-jquery packaged library for chef-server

Release Notes - Chef - Version 0.8.8

Release announcement

Version 0.8.8 (17 issues)		
Type	**Key**	**Summary**
	CHEF-457	When using merb installed from debian packages, chef-server tries to load merb-slices gem.
	CHEF-823	release and package mixlib-log 1.1.0
	CHEF-965	chef-server logs excessively regardless of the log_level setting
	CHEF-1014	Remote directory does not work with chef-solo.
	CHEF-1009	"WebUIUser" fails to index, snake-casing name to invalid value.
	CHEF-1337	Chef is too 1337 4 u
	CHEF-961	Bad use of sudo in chef-server rake tasks
	CHEF-1015	Permission denied to /var/chef/ca/key.pem when running chef-server under non-root user
	CHEF-1018	chef-solr should report version
	CHEF-1022	chef-* commands should support -P or --pidfile that sets Chef::Config[:pid_file]
	CHEF-1036	chef .debs should conflict with outdated packages
	CHEF-780	Invalid file parameters of a cookbook upload request cause an authentication failure
	CHEF-959	Add a command-line switch for the pid_file option
	CHEF-1020	Features tests should not start a merb inside the cucumber process

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-1033	knife ssh should be able to use screen for real interactivity
	CHEF-1432	metadata.rb does not accept symbol as an attribute type
	CHEF-649	cucumber.yml tags in --tags option must always start with @

Release Notes - Chef - Version 0.8.6

Release announcement

Version 0.8.6 (15 issues)		
Type	**Key**	**Summary**
	CHEF-457	When using merb installed from debian packages, chef-server tries to load merb-slices gem.
	CHEF-1000	Runlist expand does not pass couchdb value on
	CHEF-1006	fedora missing from chef/lib/platform.rb
	CHEF-1337	Chef is too 1337 4 u
	CHEF-926	cleanup! of old releases sometimes deletes current release
	CHEF-1010	Yum provider ignores specified version in some circumstances
	CHEF-707	Change "gid" to "group" for the User Resource
	CHEF-964	Adding a bad role name to run list breaks everything for that node.
	CHEF-968	Chef rest should be more flexible with user keys and headers
	CHEF-1005	knife cookbook upload should support a list of multiple cookbooks
	CHEF-714	yum with a absolute path should try localinstall and honor option settings
	CHEF-992	A node created with a role that did not exist can not be edited or deleted
	CHEF-994	chef no longer requires ruby-hmac
	CHEF-1432	metadata.rb does not accept symbol as an attribute type
	CHEF-1012	knife help text needs a scrub for accuracy and consistent formatting

Release Notes - Chef - Version 0.8.4

Release announcement

Version 0.8.4 (4 issues)		
Type	**Key**	**Summary**
	CHEF-996	CA key not properly protected
	CHEF-1003	Cookbook Uploading does not contain an X-Chef-Version
	CHEF-1337	Chef is too 1337 4 u
	CHEF-1432	metadata.rb does not accept symbol as an attribute type

Release Notes - Chef - Version 0.8.2

Release announcement

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Version 0.8.2 (208 issues)		
Type	**Key**	**Summary**
	CHEF-219	Attribute files need include_attribute
	CHEF-328	CRUD Cookbooks via API
	CHEF-491	Should suport SSL verification
	CHEF-505	Nested Roles
	CHEF-530	chef-client daemon dies with segfault
	CHEF-540	Allow association of user accounts with OpenIDs
	CHEF-542	Cookbook Uploading
	CHEF-539	User Accounts
	CHEF-555	Default adapter for chef-server merb should be thin instead of mongrel
	CHEF-573	Add a verbose setting to allow logging to a TTY
	CHEF-598	Upstart service provider
	CHEF-607	cookbook loader doesn't get attributes in correct order
	CHEF-623	Nodes are able to be created without names
	CHEF-639	git resource fails on subsequent checkouts of the same repostiry
	CHEF-642	Services will always issue a WARN when status is not present
	CHEF-661	mixlib-authentication not available on apt.opscode.com
	CHEF-666	Chef should have separate Rabbitmq users for "nanite" and "mapper" roles
	CHEF-670	knife should be smarter about the command line arguments
	CHEF-680	When rake upload_cookbooks fails, it leaves behind a tempdir that causes the next run to fail
	CHEF-696	Cache file checksums
	CHEF-698	Validate JSON in the Web UI
	CHEF-703	chef search: make the default always be to iterate over the results
	CHEF-705	chef-repo with rake install should update roles via the api
	CHEF-708	add data bags support to Chef DSL
	CHEF-710	knife fails silently if you lack the EDITOR env var
	CHEF-704	Ruby block device does not have a default action
	CHEF-728	Data bags could be Mash, not Hash
	CHEF-742	Cron provider hangs installing new crontab
	CHEF-740	debian package for nanite
	CHEF-741	Web UI allows creation of a user with no name
	CHEF-744	rake upload_cookbook and upload_cookbooks tasks should complain when run from the wrong location.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-737	Starting chef-server with '-c2' fails to create a couch database
	CHEF-749	cache is not maintained with run_list; so we load items from the cache that are no longer set to run in the run_list
	CHEF-747	webui needs a binary so you can run the slice outside of git
	CHEF-748	chef-server-webui - config.ru and bin script
	CHEF-760	Use AMQP drivers directly for indexing queues
	CHEF-764	webui crashes when attributes are Fixnum
	CHEF-758	Webui prints generated keys with spaces and not newlines
	CHEF-759	Package bunny for debian + ubuntu
	CHEF-762	chef-solr gem builds from chef root but doesn't include the lib dir
	CHEF-765	when creating a user in the web-ui, it causes an indexer failure to occur in solr and thus users never get indexed.
	CHEF-777	hostnames with an _ in it, create 2 node entry's and then fail during a chef-client run
	CHEF-787	Make bunny DFSG-free
	CHEF-799	remote_directory does not work
	CHEF-806	knife refactored should show all the sub commands with --help
	CHEF-809	man page for knife
	CHEF-811	knife subcommand options are invalid, not processed
	CHEF-812	file backup permissions less secure than file
	CHEF-808	Rakefile for mixlib-authentication uses obsolete cucumber task syntax
	CHEF-824	The WebUI is not detecting thin
	CHEF-832	chef-solr does not install any binaries what so ever.
	CHEF-855	ruby-openid required by chef-server-webui, and missing in rakefile
	CHEF-858	Provider detection broken if Resource type and Provider name matches
	CHEF-860	chef-web-ui status button shows 500 error
	CHEF-856	knife configure should allow you to create a new api user
	CHEF-864	Unable to reindex chef via knife or chef-solr-rebuild
	CHEF-879	knife data bag show groups fails ... with something like undefined method `keys' for ["http://localhost:4000/data/groups/sysadmin"]:Array
	CHEF-898	chef-solr-indexer needs to depend on uuidtools
	CHEF-900	security vulnerability in 0.8 webui
	CHEF-902	mixlib-authentication fails to generate SHA1
	CHEF-912	deep_merge should be a Chef::Mixin
	CHEF-916	on chef-server startup, webui_user.rb:203:in `create_design_document': uninitialized constant Chef::Couchdb (NameError)
	CHEF-918	chef-solo fails to download remote recipes because application/solo.rb no longer uses open-uri

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-921	Remove deprecated gem dependencies
	CHEF-935	chef-server-webui data bag item editing is broken
	CHEF-951	update redhat distro files for 0.8
	CHEF-974	undefined chef_server_rest
	CHEF-975	client.pem is mode 644 by default
	CHEF-989	Search for client in the webui returns 404 when trying to show the client in the search result
	CHEF-1337	Chef is too 1337 4 u
	CHEF-919	chef-server-webui requires merb-param-protection but rakefile doesn't list it.
	CHEF-537	Authenticate Signed API Requests
	CHEF-601	Extend metadata spec based on wiki comments
	CHEF-681	500 error when trying to retrieve a file w/o a default dir
	CHEF-688	Deploy revision strategy does not fetch tags
	CHEF-695	No backups kept if backup is set to 1
	CHEF-756	Specifying the run list for instance data with knife should ignore commas
	CHEF-774	Chef-server does not log anything to server.log
	CHEF-822	Web UI differentiates between "user not found" and "wrong password"
	CHEF-120	OpenID auth bits should default to HTTPS
	CHEF-136	bad permissions on search_indexes files hangs chef-server
	CHEF-209	make use of Joshua Sierles's thorfile for recipe management
	CHEF-291	No high-level cookbook endpoint in REST API
	CHEF-358	Refactor the REST API
	CHEF-417	Add rake task to update roles in running server.
	CHEF-438	Chef::Provider::Execute doesn't honor user attribute for not_if/only_if
	CHEF-440	Running chef-client with a JSON file should override the node's run_list on server
	CHEF-447	Zypper provider - suse support
	CHEF-454	Centos4 yum provider failure
	CHEF-470	role attribute deep merge only goes one level deep
	CHEF-474	Switch chef-server syntax highlighting to coderay
	CHEF-536	Add server side key-pair creation
	CHEF-538	Allow for the creation of API Clients
	CHEF-543	Web UI should call the API exclusively
	CHEF-551	erl_call provider
	CHEF-576	provider for python easy_install

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-584	launching chef-client init script hangs chef-solo
	CHEF-629	Users *must* be able to set the action on the SCM resource used by deploy
	CHEF-635	Portage incorrectly detects currently installed packages
	CHEF-644	Shebang lines should respect user's ruby
	CHEF-643	Fix CHEF-570, as it doesn't catch package -revisions
	CHEF-650	Freebsd and Yum package shortcut resources don't exist
	CHEF-645	cd to /tmp breaks merb bootloader in features
	CHEF-659	UI expands some escaped characters from JSON, then fails to encode them again
	CHEF-668	cron resource seems to always run for non-numeric time entries
	CHEF-677	Webui users object (for login)
	CHEF-687	shef: irb/REPL mode for chef
	CHEF-684	Should be possible for roles to be created without anything in the run_list
	CHEF-693	Nodes should auto-expand
	CHEF-697	Cookbook metadata does not convert booleans to "required" or "optional" on ruby 1.8.7 as shown by spec failures
	CHEF-694	role override and default attributes don't get indexed
	CHEF-699	solo and client modes in shef
	CHEF-706	mount provider fails to mount samba/cifs devices (Device does not exist)
	CHEF-725	'rake gem' fails calling 'rake package' on chef-solr
	CHEF-753	chef server slice should finish activating
	CHEF-755	"knife create_client" could accept :admin option
	CHEF-766	Chef should print the error message generated by the Chef Server when an HTTP Exception occurs
	CHEF-776	users and groups for mac os x
	CHEF-775	if a client has a hostname with an _ such as hasoffers_3.adappsolutions.com it gets truncated improperly and mangled into hasoffers.3.adappsolutions.com and then returns a 403 during chef-client run
	CHEF-800	The response from /search contains nil in some situations while the Chef::Search::Query library tries to process every result even it's nil
	CHEF-801	Deprecated dependencies in chef-server
	CHEF-805	Deploy resource's scm_provider should accept a short string/symbol as name instead of a fully qualified class name
	CHEF-807	SVN provider uses undefined local variables to create an error message, obscuring the true cause of the error
	CHEF-813	Spec failure: 'Chef::Application::Knife run should exit 2 if run without a sub command' actually returns 1
	CHEF-814	Spec failure: 'Chef::Application::Knife run should exit 2 if run without a sub command' actually returns 1
	CHEF-818	rake install: no longer require sudo
	CHEF-826	rest fails with an obscure error if node_name is not determinable
	CHEF-841	knife keeps trying to upload a cookbook despite HTTP 401

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-837	Delayed actions excute in unpredicatable order
	CHEF-854	shef executable needs to be added to the gemspec
	CHEF-859	route provider will incorrectly configures centos/rhel networking
	CHEF-865	Implement retry logic when making server requests
	CHEF-866	@node and node should be available in attributes
	CHEF-870	does not close stderr when daemonising
	CHEF-873	Template provider doesn't respect the template resource's cookbook option
	CHEF-877	fix :default and :required in lwrp
	CHEF-882	slowdown due to debugging in ruby_block provider
	CHEF-880	chef-server-api should require admin privileges to update data bag items
	CHEF-897	package provider does not tell you about which package is the cause of problems
	CHEF-904	"Could not create work tree dir" on chef deploy
	CHEF-906	chef-server-webui refers to JSONeditor which doesn't exist
	CHEF-908	Indexer fails on node properties that have invalid XML character sequences
	CHEF-922	knife ssh should accept -a ec2.public_hostname
	CHEF-931	deep_merge mixin spits out a warning
	CHEF-932	git provider fetch strategy will not get all updates
	CHEF-927	Override and Default attributes get clobbered at recipe load time
	CHEF-936	chef no longer requires deep_merge gem
	CHEF-937	chef-server-api and Chef::Solr::Query bans queries for api_users
	CHEF-943	add section for knife ssh to man page
	CHEF-948	pacman provider / resource - archlinux support
	CHEF-944	chef-solr needs LICENSE information
	CHEF-949	service daemons - archlinux support
	CHEF-957	arch as platform - archlinux support
	CHEF-956	Embed the Chef::VERSION as X-Chef-Version in HTTP requests
	CHEF-969	Api client should expect 409 not 403 in save when one with same name already exists
	CHEF-970	Need both class and instance methods for chef_server_rest
	CHEF-971	Changes to webui error handling
	CHEF-977	`knife configure` should prompt for validation_client name and validation_key
	CHEF-978	Knife should not show HTTP Request Returned 404 Not Found: Cannot load node foo as WARN when the node gets created successfully
	CHEF-390	Software raid provider

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-406	typos in specs, fix resulting failures
	CHEF-444	document using the chef-server API from a script using 'knife' as an example
	CHEF-617	Install to chef repository on a remote machine
	CHEF-616	rake install in chef-repo breaks if there is no git origin
	CHEF-626	Template Context should have a #node method so users don't have to remember to use @node
	CHEF-634	UI fails silently when unable to save node
	CHEF-633	HTTP Request uses Chef::REST incorrectly
	CHEF-638	Deploy with a revision should make that revision current if it already exists
	CHEF-646	Enable Chef::Provider::Cron under FreeBSD
	CHEF-647	Deploy resource should always run symlinks before migrate.
	CHEF-654	chef-client -j against a self signed cert fails
	CHEF-656	0.8 Integration tests should setup Vhosts and users for nanite
	CHEF-657	Deploy with revision strategey did not clean up cache on rollback
	CHEF-664	libxml required but not a gem dependency
	CHEF-667	Specs run really slooooow because of ohai
	CHEF-673	uuidtools gem is required with a silent rescue and not dependend on by chef server
	CHEF-671	SCM providers should be able to specify the group
	CHEF-676	redhat init script update
	CHEF-713	missing alias for rpm_package (and probably also freebsd_package, yum_package)
	CHEF-709	Support for backup up files in another directory than the original file
	CHEF-722	Link provider doesn't understand paths requiring expansion
	CHEF-723	creating ssl-certs with fqdn "*.example.com" should name w/ wildcard instead of *
	CHEF-732	Remove references to chef-indexer
	CHEF-734	LWRP resources should look for provider named same by default
	CHEF-736	chef-solr gem rake install task does not use sudo
	CHEF-750	Refactor the caching code so logic for what gets cached and why is wrapped in a class
	CHEF-754	Chef::Config.cookbook_path is in reverse order of override application
	CHEF-769	knife should be able to specify the config file location
	CHEF-779	rake upload_cookbook shouldn't try to upload cookbooks that don't exist in the local repo
	CHEF-796	Change to mixlib-log breaks Chef::Log.level() usage
	CHEF-815	knife cookbook upload fails when cookbooks are in an SVN repo
	CHEF-840	client admin field should be checkbox not text box
	CHEF-843	FileEdit permission issues

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-850	rake spec in 'chef' project tries to create directory /new/home/adam
	CHEF-881	make :name_attribute work in lwrp
	CHEF-901	If a role is deleted, any nodes that had that role can not be edited or deleted with the webgui
	CHEF-896	file_backup_path include prefix in log output
	CHEF-928	deep_merge issues with empty strings
	CHEF-929	typo in provider/deploy/revision causes rescue from StandardError instead of Chef::Exceptions::FileNotFound
	CHEF-934	chef-server's old gemspec is still in the source tree
	CHEF-980	BULK DELETE (cookbook/node/client/role) defaults to deleting everything
	CHEF-990	bulk deletes are not spec tested for clients, cookbooks, or roles
	CHEF-1432	metadata.rb does not accept symbol as an attribute type
	CHEF-374	drop outdated contrib/
	CHEF-731	Add a screenrc to start a chef environment in screen
	CHEF-825	The WebUI is a little excited about wanting you to change the password and gramatically off
	CHEF-842	duplicate copies of FileEdit: file_edit.rb & fileedit.rb
	CHEF-851	specs leave a /tmp/foo directory lying around

Release Notes - Chef - Version 0.7.16

Release announcement

Version 0.7.16 (6 issues)		
Type	**Key**	**Summary**
	CHEF-812	file backup permissions less secure than file
	CHEF-1337	Chef is too 1337 4 u
	CHEF-686	chef-{server,client,indexer} creating world writable files and directories
	CHEF-859	route provider will incorrectly configures centos/rhel networking
	CHEF-882	slowdown due to debugging in ruby_block provider
	CHEF-923	Chef "Log" resource

Release Notes - Chef - Version 0.7.14

Version 0.7.14 (19 issues)		
Type	**Key**	**Summary**
	CHEF-591	if service doesn't have a status command, process table inspection fails in simple service provider
	CHEF-603	deploy: gems.yml support
	CHEF-604	deploy: sudo / run command handler support
	CHEF-628	Deploy resource removes newest release instead of oldest

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-1337	Chef is too 1337 4 u
	CHEF-602	in deploy provider, callback-defined resources are executed in all subsequent callbacks
	CHEF-640	SCM providers do not set resource updated (notifications)
	CHEF-577	provider.rb doesn't give @definitions a default value of Hash.new, causing epic fail in resource DSL
	CHEF-570	Portage package provider uses wrong regexp in load_current_resource
	CHEF-588	RC is missing debugging info in find_preferred_file
	CHEF-593	deploy resource is not idempotent
	CHEF-614	LWRP undefined local variable or method `new_resource'
	CHEF-619	Mixlib-* gems installed from gemcutter.org have too restrictive permissions
	CHEF-859	route provider will incorrectly configures centos/rhel networking
	CHEF-565	dpkg provider fails at packages with a dash in it, causing packages to be re-installed on every chef run
	CHEF-622	Gem Package resource/provider shouldn't silently ignore the options attribute
	CHEF-621	LWRP dynamic attribute methods are Ruby 1.9 incompatible and cause warnings in 1.8
	CHEF-620	LWRP components should be shown in Web UI
	CHEF-631	Should create LWRP resources/providers for new_cookbook

Release Notes - Chef - Version 0.7.12

Version 0.7.12 (22 issues)		
Type	**Key**	**Summary**
	CHEF-293	Chef breaks on systems with non-English Locales
	CHEF-1337	Chef is too 1337 4 u
	CHEF-544	Service provider fails to set @new_resource.updated
	CHEF-145	Cron resource: add support for setting cron environment variables like MAILTO or PATH
	CHEF-419	Create SCM resource and providers for git & svn
	CHEF-496	add simple service provider to chef
	CHEF-501	Fails to follow notification chains
	CHEF-534	remove execute permissions from javascripts, images, etc.
	CHEF-546	Make couchdb version switcher 0.8 specific, and use the new format for everything else.
	CHEF-559	distro/ should be under 'chef' dir and packaged w/ gem.
	CHEF-562	typo in provider/ifconfig.rb
	CHEF-560	refactor Chef::Provider::Group::Groupadd
	CHEF-569	Remote File causes updates to be sent regardless of idempotency
	CHEF-571	LWRP unit tests
		Lots of files/subdirectories in a remote_directory cause most chef requests to take 11.5 seconds, and the merb process

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-578	goes up to 100% CPU
	CHEF-582	group resource should allow users or members as a parameter
	CHEF-503	cookbooks UI should display the relative path of the template
	CHEF-547	remote_file not computing+comparing checksums before doing work
	CHEF-561	Flexible application layouts for deploy resource and provider
	CHEF-566	Deploy resource/provider callbacks for before_migrate, &etc. should support in-line recipes
	CHEF-580	faster find_preferred_file
	CHEF-568	Increase logging for Remote File status, including checksummation.

Release Notes - Chef - Version 0.7.10

Version 0.7.10 (17 issues)		
Type	**Key**	**Summary**
	CHEF-481	0.7.6 breaks Centos5
	CHEF-492	Attributes aren't quite hash enough to fool variables() method in templates
	CHEF-515	chef-server needs to be compatible with couchdb 0.10.0
	CHEF-1337	Chef is too 1337 4 u
	CHEF-287	chef client logging to file only dumps periodically
	CHEF-362	regression: mixlib-config 1.0.10 breaks log_location= override
	CHEF-449	use the basename of the target path as the default for the template, remote_file and remote_directory sources
	CHEF-476	merb-slices gem not installed with new chef server install on ubuntu 9.0.4
	CHEF-488	When installing a gem_package, the version parameter needs to be quoted
	CHEF-494	Multiple roles' default_attributes are not merged correctly on the node
	CHEF-493	Alias set_unless with "default" in attributes.rb
	CHEF-497	Status page fails to work under ruby 1.8.5
	CHEF-500	server.rb log_location truncation take two
	CHEF-512	Update chef config for pending mixlib-config config_attr_writer fix
	CHEF-509	add man pages, initscripts and other goods from packaging efforts
	CHEF-525	Spec failures in node/attribute_spec.rb
	CHEF-524	Chef::Platform.find_provider_for_node is always used to get an instance of the provider (code duplication)

Release Notes - Chef - Version 0.7.8

Version 0.7.8 (17 issues)		
Type	**Key**	**Summary**
	CHEF-477	Walking an attribute with .each when it is deeply nested and has no defaults or overrides causes an exception

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-484	Method Missing attribute lookups don't honour auto vivify on read.
	CHEF-483	Chef::Node::Attribute.should behave_like_a(Hash)
	CHEF-486	Attributes added from an attribute file can unneccessarily block run list expansion
	CHEF-1337	Chef is too 1337 4 u
	CHEF-482	Nested hashes loaded from JSON don't act like nested hashes
	CHEF-269	remote_file fails if URL contains % characters
	CHEF-319	chef-server should support connecting to an admin-only couchdb server
	CHEF-430	Add support for HTTP Basic Authentication to Chef::REST
	CHEF-432	chef-server-slice views should use slice_url instead of url
	CHEF-431	ChefServerSlice::OpenidServer#index should treat content_type as a symbol
	CHEF-439	service resource - inherited file handle weirdness
	CHEF-474	Switch chef-server syntax highlighting to coderay
	CHEF-472	Output logging is very choppy
	CHEF-475	chef-repo should have a roles directory with chef upgrade from blog post
	CHEF-473	attribute setting issue in 0.7.6
	CHEF-479	Attribute setting issue in 0.7.6 - wrong saved names?

Release Notes - Chef - Version 0.7.6

Version 0.7.6 (30 issues)		
Type	**Key**	**Summary**
	CHEF-411	When a recipe is in the run_list and included, it may be run twice
	CHEF-436	Default attributes from a role should not be written into the role's nodes
	CHEF-465	chef-solo tries to get templates from localhost:4000
	CHEF-467	chef-server web ui doesn't load assets properly with 0.7.5
	CHEF-1337	Chef is too 1337 4 u
	CHEF-410	Be able to have major, minor, revision for metadata version.
	CHEF-87	File specificity (preferred file) is broken by dotfiles
	CHEF-274	Add support for multiple named queues on a single stompserver
	CHEF-357	groupmod error with chef (0.6.2)
	CHEF-371	Link provider will not delete symbolic link if target does not exist
	CHEF-394	Last Check-in is only updated on the first run of chef-client, but not on later runs
	CHEF-395	Splay time should be applied before the first run, not after
	CHEF-409	run_list specified in json doesn't run the file caching step before trying to find recipes
	CHEF-407	CHEF-308 breaks mount provider remote filesystem usage

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-413	why did chef gem get so big?
	CHEF-415	Error message clarity when a registration exists for the same hostname but the local secret doesn't exist / match
	CHEF-416	API roles, search integration features have failing steps
	CHEF-414	add symlink owner/group support
	CHEF-418	Update readme for new integration tests
	CHEF-423	gentoo service provider enable action always runs for long service names
	CHEF-458	File Cache stub missing
	CHEF-463	chef-solo and chef-client should display the version with --version (and optionally, -v)
	CHEF-373	include specs in chef gem
	CHEF-406	typos in specs, fix resulting failures
	CHEF-420	Decouple usage of -s from -i
	CHEF-433	rake install should not copy .git files in addition to .svn
	CHEF-437	if manage_home is set to true but no home is set then home dir becomes -m
	CHEF-448	allow a default URL definition in client.rb
	CHEF-400	Chef::Application#fatal! should log output to STDERR aswell, so errors are more plain
	CHEF-427	Minor cleanup of JSON Attribute editor

Release Notes - Chef - Version 0.7.4

Version 0.7.4 (9 issues)		
Type	**Key**	**Summary**
	CHEF-404	openid consumer only valid for http://authorized_openid_identifier/
	CHEF-1337	Chef is too 1337 4 u
	CHEF-363	we should kick out logged in openids if they become unauthorized
	CHEF-381	Ruby Block Resource
	CHEF-388	rake task new_cookbook generates empty version metadata
	CHEF-402	Weird characters on the bottom of status page
	CHEF-401	After Install Phusion throws a Runtime Error because of the log file
	CHEF-399	Unit test fixes and 100% CPU bug
	CHEF-403	Capitals in authorized_open_id_identifiers

Release Notes - Chef - Version 0.7.2

Version 0.7.2 (19 issues)		
Type	**Key**	**Summary**
	CHEF-367	node run_list is empty when running chef-client with a json file specifying run_list

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-372	chef-client and chef-solo will fail to run ohai if the node_name is explicitly configured.
	CHEF-1337	Chef is too 1337 4 u
	CHEF-355	Test chef::rest, chef::couchdb for remote DoS due to BigDecimal
	CHEF-166	chef-server-slice rake install does not pick up merb-core dependency correctly
	CHEF-299	chef client running under runit does not restart
	CHEF-348	chef-server-slice version isn't tied to chef-server when loaded
	CHEF-359	Using chef-server-slice results in exceptions.rb "superclass mismatch for class Exceptions"
	CHEF-361	Init service provider currently doesn't start a service when the 'restart' action is called if the service is not already running.
	CHEF-368	Chef README in the GitHub repo is outdated!
	CHEF-369	chef-indexer pid and permissions
	CHEF-376	Roles don't save.
	CHEF-386	'rake roles' doesn't update json files from ruby files
	CHEF-383	Cookies have limit of 4K.
	CHEF-396	4k cookie limit with nginx+passenger
	CHEF-360	rake tasks should copy roles
	CHEF-370	prettify /status view of uptime
	CHEF-387	deleting a role generates a 500 error in the web interface
	CHEF-374	drop outdated contrib/

Release Notes - Chef - Version 0.7.0

Version 0.7.0 (59 issues)		
Type	**Key**	**Summary**
	CHEF-330	Chef::Config[:solo] not being set in Solo runs
	CHEF-344	gem_package resources failing to install packages
	CHEF-350	chef-solo json attributes aren't loaded
	CHEF-1337	Chef is too 1337 4 u
	CHEF-345	chef-client doesn't automatically read /etc/chef/client.rb anymore
	CHEF-342	chef-server's config.ru should include the server config file
	CHEF-122	Notifies should be able to send different actions to different resources, perhaps through a hash
	CHEF-151	Refactor application CLI logic to a seperate class so it can be tested
	CHEF-215	Speedup of package provider (at least, for Gentoo)
	CHEF-220	Interface Provider
	CHEF-225	Routing provider needs extended to manage configuration files
	CHEF-232	determine if a recipe has been seen / included on a node

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-242	activesupport conflicts with to_json in node.rb
	CHEF-243	Search results in recipes & chef server return a *flattened* hash. It should return, or be able to return, a non-flattened hash, with all field names intact.
	CHEF-245	chef/provider/user/useradd.rb should only add -m to command line if home directory is being modified
	CHEF-253	chef-server-slice on centos 5.3 incompatibility with merb-core and ruby 1.8.5
	CHEF-259	redhat service provider - correct use of chkconfig
	CHEF-260	yum provider remove_package without version broken
	CHEF-261	yum provider depends on 'json' which is not available for Python 2.4
	CHEF-265	Add support for MacPorts on OS X as a package provider
	CHEF-272	include contrib/ in gems
	CHEF-275	Cookbook Meta-data Generator
	CHEF-276	server.rb log_location truncation
	CHEF-280	recipes evaluated twice
	CHEF-284	Role support
	CHEF-281	Locale isn't set properly during package installation on Ubuntu 8.10 (at least)
	CHEF-297	Failing specs for portage provider
	CHEF-296	limiting openid providers variable consistency
	CHEF-295	chef-indexer throws fatal exception when processing index/remove messages
	CHEF-302	chef-server authentication requirement should link to information on securing openid
	CHEF-306	We should only ship the set of cookbooks needed to build a node to the edge
	CHEF-312	FreeBSD user and group provider broken by CHEF-220
	CHEF-310	yum provider - YumCache - refresh logic
	CHEF-318	yum provider failure /usr/lib/ruby/gems/1.8/gems/chef-0.6.2/lib/chef/provider/package/yum.rb:69:in `load_data': yum failed - #<Process::Status: pid=4726,exited(1)>! (Chef::Exceptions::Package)
	CHEF-320	roles creation fails with an `each' nil:NilClass error
	CHEF-325	cookbook not found error message is not helpful for metadata uptake
	CHEF-323	yum provider - improve failure output
	CHEF-331	user provider - useradd.rb :manage_home confusion
	CHEF-337	Update skeleton repository for Chef 0.7.0
	CHEF-336	Error when no tags set in create REST call
	CHEF-332	Deleting a node via the web-ui pops up an error message, though the deletion is successful
	CHEF-339	log file should be a string, got: #<IO:0x7f0db9a68ad8>
	CHEF-346	Update 0.7.0 for new rake task to test templates and cache tests
	CHEF-354	chef-repo should have config/chef_upgrade.json

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-353	.rake_test_cache gets corrupt
	CHEF-116	Require authentication for the entire Chef Server
	CHEF-131	Add a flag to chef-client to re-read JSON attribute file each time it wakes up (in conjunction with daemonization/interval)
	CHEF-178	remote_directory not supported in chef-solo
	CHEF-226	the group resource doesn't allow you to add or remove users from the group, just explicitly set them.
	CHEF-251	chef-server should run from gems without requiring config file modifications within the gem directories.
	CHEF-268	Config files should be able to be passed into chef-client as http resources
	CHEF-271	rpm packaging - executable examples
	CHEF-288	service resource - supports and custom stop/start/etc commands
	CHEF-329	more useradd home directory fun
	CHEF-327	Package providers shoud have an options attribute to pass verbatin to underlying commands
	CHEF-335	dpkg_package resource does not work
	CHEF-283	"registrations" should be "registration's"
	CHEF-317	Last Check in relative time ago
	CHEF-347	cleanup - chef/provider/remote_directory.rb do_recursive_old

Release Notes - Chef - Version 0.6.2

Version 0.6.2 (7 issues)		
Type	**Key**	**Summary**
	CHEF-255	chef-client doesn't respect interval and stay running in foreground.
	CHEF-1337	Chef is too 1337 4 u
	CHEF-156	chef-client and chef-solo should accept a URL for the -j switch
	CHEF-257	Rake test does not execute tests against the site-cookbooks directory
	CHEF-249	can I has open-uri supported "-j http://blah.com/some.json" for chef-solo/client?
	CHEF-258	gem_package doesn't allow you to point at a custom gem binary outside of your $PATH
	CHEF-277	chef-solo should be able to use cookbook_path from config file

Release Notes - Chef - Version 0.6.0

Version 0.6.0 (46 issues)		
Type	**Key**	**Summary**
	CHEF-159	CookbookLoader#load_cascading_files uses Dir.glob(array) syntax, which is unavailable on RHEL and Centos
	CHEF-162	delayed notifications should coalesce so they don't run multiple times
	CHEF-175	Install instructions on wiki don't work
	CHEF-195	The backup attribute for File needlessly creates a backup then deletes it when set to 0

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-206	chef-server-slice rake install fails
	CHEF-218	Search indexes are broken due to missing route
	CHEF-228	chef-server under passenger: Controller class not found for controller `registrations'
	CHEF-238	Commands with over 4k of output will block forever
	CHEF-1337	Chef is too 1337 4 u
	CHEF-169	Detect and use new view URL's for CouchDB 0.9.0 compatability
	CHEF-222	CouchDB 0.9 compat causes the client to attempt a connection to the CouchDB server (while instantiating a node object)
	CHEF-54	chef-server should be a merb slice
	CHEF-78	Add a mixin for parsing a file and replacing lines
	CHEF-129	Implement pilu's web-app-theme for chef-server
	CHEF-134	User IDs and GIDs with negative numbers cause a type exception
	CHEF-167	chef-server-slice rake install requires sudo
	CHEF-164	Slice routing changes break remote file provider
	CHEF-163	remote_file not working on head, 404 due to missing route
	CHEF-174	directory mode not set correctly
	CHEF-182	Chef::Exception masks ::Exception (see OHAI-79)
	CHEF-188	Merb dependencies for both the server app and slice need to be updated to 1.0.10
	CHEF-192	speed up yum provider
	CHEF-200	fixes from centos packaging attempt
	CHEF-207	chef-server and chef-server-slice should require merb > 1.0, not specific versions
	CHEF-214	On Gentoo, services always support the :status command, so enable it by default
	CHEF-230	remote_file with a url should include a type of hash option to verify the downloaded file is what we expected
	CHEF-227	Delete operation of Search Index is not working
	CHEF-237	Apt provider won't install msttcorefonts
	CHEF-244	Chef server no longer checks openid's against the authorized_openid_identifiers configuration
	CHEF-247	Let chef-server and chef-client compare checksums in bulk rather than one at a time.
	CHEF-148	cookbook naming and routing
	CHEF-153	Allow access to a list of OpenID's via configration.
	CHEF-173	With debug logging level set long running commands do not show their output in real time
	CHEF-171	Package name with a dash (-) in it is not recognised
	CHEF-172	In FreeBSD package provider simplify source parameter "magic" by using PKGNAME in ports Makefile
	CHEF-177	Where multiple ports have the same name allow path to ports to be given
	CHEF-176	Chef status page that displays basic info about each chef managed node

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-186	file delete fails if file does not exist
	CHEF-191	Enable optional CouchDB storage for OpenID associations and nonces
	CHEF-190	apt provider fails on non-English debian installations
	CHEF-210	enterprise linux init scripts and configs
	CHEF-216	Allow execute/script resources to set umask
	CHEF-246	Should be clearer what tags are attached to a node
	CHEF-185	file delete backs up links
	CHEF-198	chef-solo banner
	CHEF-213	Fixing typos in the code

Release Notes - Chef - Version 0.5.2

Version 0.5.2 (27 issues)		
Type	**Key**	**Summary**
	CHEF-37	Chef Solo does not obey the file selection laws
	CHEF-75	Badly behaved children block all IO
	CHEF-1337	Chef is too 1337 4 u
	CHEF-29	Group provider needs to be able to manage group members
	CHEF-44	Chef will block forever reading IO, even on processes that don't play nicely with their filehandles
	CHEF-57	Permission denied when using bash resource and a non-root uid
	CHEF-11	Templates should be cached once
	CHEF-28	Rendering error when editing a node
	CHEF-41	Chef Solo tells lies about being able to --noop
	CHEF-43	service provider lacks action_none
	CHEF-38	Unabled to delete nodes from the Node page
	CHEF-46	route provider for adding and deleting routes
	CHEF-49	Chef should be able to manipulate cron jobs
	CHEF-56	Service resource needs Redhat providers
	CHEF-55	Service resource needs Gentoo providers
	CHEF-58	Support for rc.d services in freebsd
	CHEF-60	Teach chef about the debian platform
	CHEF-62	portage provider should support both fully qualified package names and non prefixed package names
	CHEF-70	Clicking on certain recipes within Chef Server (chef-server 1.0.8.1) Web UI results in error 500
	CHEF-30	Link should be more intuitive

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	CHEF-34	Failing unit test on OS X
	CHEF-36	Default @action for http_request is :create instead of :get
	CHEF-35	http_request should allow a block for the message, which will get evaluated when the request is sent
	CHEF-53	support for freebsd pkg_*
	CHEF-52	Chef traces miserably if ohai fails to provide it with a hostname
	CHEF-66	Create registrations via REST easily
	CHEF-61	Added Cron provider support for gentoo in platform.rb

Release Notes - Chef - Version 0.5.1

Version 0.5.1 (19 issues)		
Type	**Key**	**Summary**
	CHEF-1	Group Support
	CHEF-6	Chef should require Ohai, not Facter
	CHEF-7	Remote File tests failing after Solo updates
	CHEF-9	Chef Client should Daemonize, schedule, and splay
	CHEF-18	Search index does not understand nested hashes
	CHEF-22	templates aren't created / found in the cache and execution aborts
	CHEF-1337	Chef is too 1337 4 u
	CHEF-1489	CLONE -templates aren't created / found in the cache and execution aborts
	CHEF-3	Require chef loads everything
	CHEF-8	Add sugar for a Tag attribute on the nodes
	CHEF-5	Documentation for Service Providers/Resources
	CHEF-16	Search and SearchIndex are only used by the Chef Server, but they live in Chef Client
	CHEF-12	spec_helper causes bogus Constant redefinition
	CHEF-10	Remove Chef::FileStore in favor of Chef::FileCache
	CHEF-26	When you have a resource with the same name, it should inherit the pre-existing resources attributes
	CHEF-25	http_request resource and provider
	CHEF-27	No longer use MD5 anywhere - no sleep till SHA-256
	CHEF-40	Installation Broken
	CHEF-21	Chef::Daemon needs unit tests

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef OmniGraffle & Visio Stencils

This is a set of stencils for use with OmniGraffle and Visio for creating diagrams. This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

Omnigraffle Stencil: **OpscodeChef.gstencil.zip**

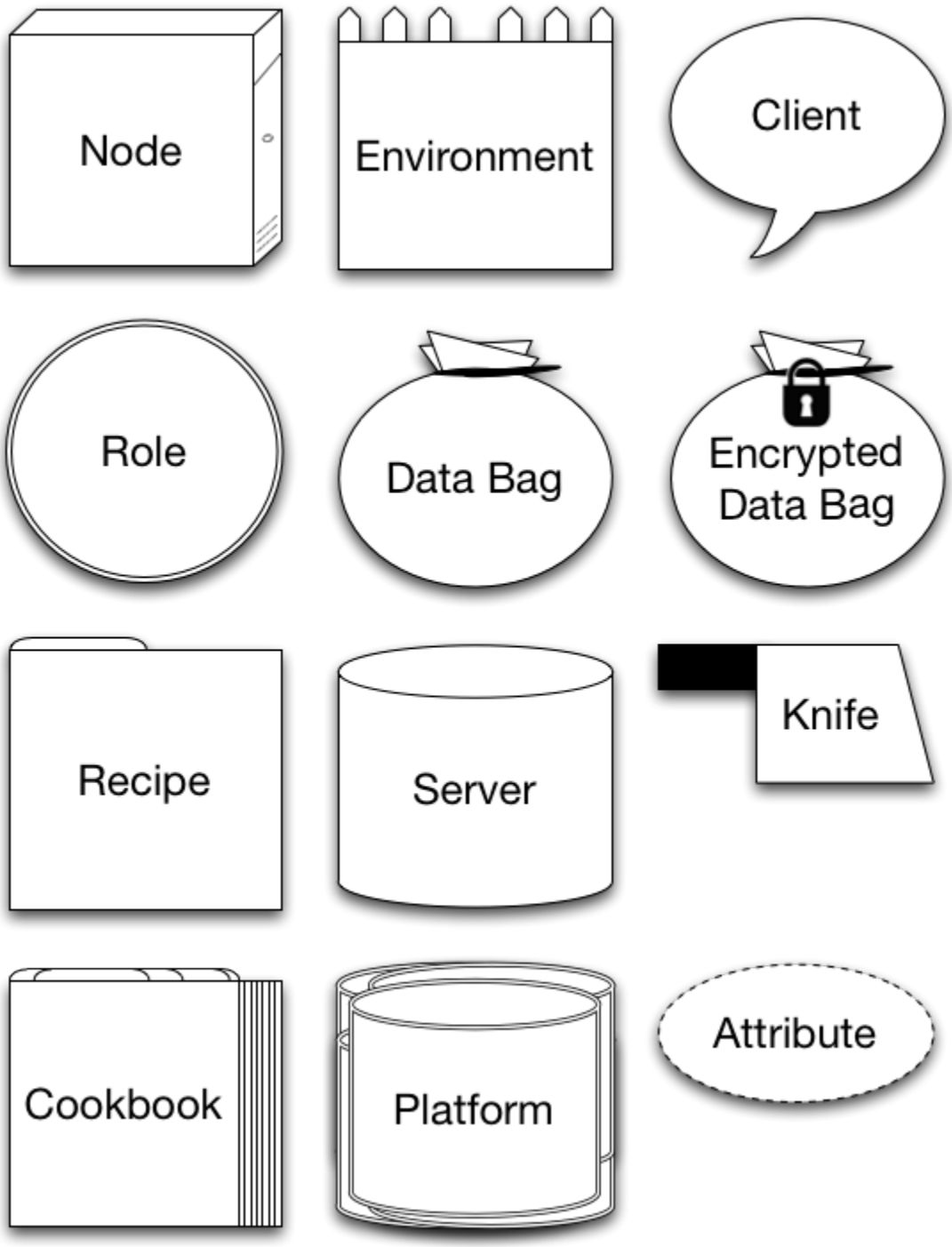
Omnigraffle Visio XML export: **OpscodeChef.vdx**

Omnigraffle SVG export: **OpscodeChef.svg.zip**

Quick Omnigraffle Install

To install, drag the stencil to:
~/Library/Application Support/OmniGraffle/Stencils/
then add to Omnigraffle Stencil Library

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License



Guides

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Walkthrough Guides and Webcasts

Note: These Walkthrough Guides were written for Hosted Chef customers, but should be adaptable for Open Source Chef Server users as well. For assistance, see Support.

- **Build A Rails Stack**
- **Build A Java Web Stack**
- **Nagios Quick Start**
- **Build a LAMP Stack**
- **Build a Django Stack**
- **Automating Zenoss Core**

Includes full-instructions and an accompanying video. Especially for folks just getting going, it's an excellent way to see how a lot of the components of Chef fit together and get a taste of Hosted Chef.



How-To Guides

- **Deploying OpenStack with Chef**
- **Guide to Creating A Cookbook and Writing A Recipe**
- **Working with Git and Cookbooks**
- **How to Proxy Chef Server with Apache**
- **Performance Tune Ohai for Large User Bases**
- **Writing Ohai Plugins**



Tutorials From the Community

- **Managing System Complexity With Chef**
  thanks *Robert Berger*
- **Acceptance Test Driven Infrastructure Development with Cucumber and Chef**
  thanks *Stephen Nelson-Smith*
- **How Yipit Deploys Django With Chef**
  thanks *Zach Smith*
- **Boxgrinder meta appliance using the VMWare ESX/ESXi hypervisor and Opscode Chef**
  thanks *Automation Inc*
- **CentOS 6.1, local VMs and Opscode Chef**
  thanks *Chris Dagdigian*



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Training

- Upcoming Public Training Classes
- Private Training Classes
- Register for Free Open Training Materials

Fast Start Guides



Client Bootstrap Fast Start Guide

Fast Start Guide for Windows

Fast Start Guide for CentOS

E
B
F
G

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Build a Django Stack

Build a Django application stack using Chef cookbooks available from the Cookbooks Community Site and Hosted Chef.



The Walkthrough Guide assumes you followed the Fast Start Guide and have Chef installed and working with the Hosted Chef platform.

At the end of this guide, you'll have four total Ubuntu 10.04 systems running in Amazon EC2.

- 1 haproxy load balancer.
- 2 Django (Gunicorn) application servers.
- 1 MySQL database server.

If you don't already have an account with Amazon AWS, go to Amazon Web Sevices and click "Sign up". You'll need the access and secret access key credentials from the sign-up later.

The Django application used in this guide leverages the Django CMS application. The source code for our Django application can be downloaded on the Opscode GitHub account. This application follows the out of the box setup instructions found on the Django CMS project site. We have also created a pip requirements file for tracking project dependancies.

We're going to reuse a number of cookbooks from the Cookbooks Community Site to build the environment. For example, the source code lives in **git**, so that cookbook will ensure Git is available. The load balancer is **haproxy** because it is very simple to deploy and configure, and we use a recipe that automatically discovers the Django application systems. The heavy lifting is handled by recipes in the **application** and **database** cook books.

The **application** cookbook will perform the following steps:

- create an application specific virtualenv
- install required packages and pips into the project specific virtualenv
- set up the deployment scaffolding
- creates settings_local.py file with the database connection information if required
- performs a revision-based deploy
- install Gunicorn
- create an application specific Gunicorn configuration file
- create a Runit service for managing Gunicorn

Guide Based Upon Ubuntu 10.04 on Amazon AWS EC2 with Chef 0.10.0.

Note: *At this time, the steps described above have only been tested on the identified platform(s). Opscode has not researched and does not support alternative steps that may lead to successful completion on other platforms. Platform(s) supported by this guide may change over time, so please do check back for updates. If you'd like to undertake this guide on an alternate platform, you may desire to turn to open source community resources for support assistance.*

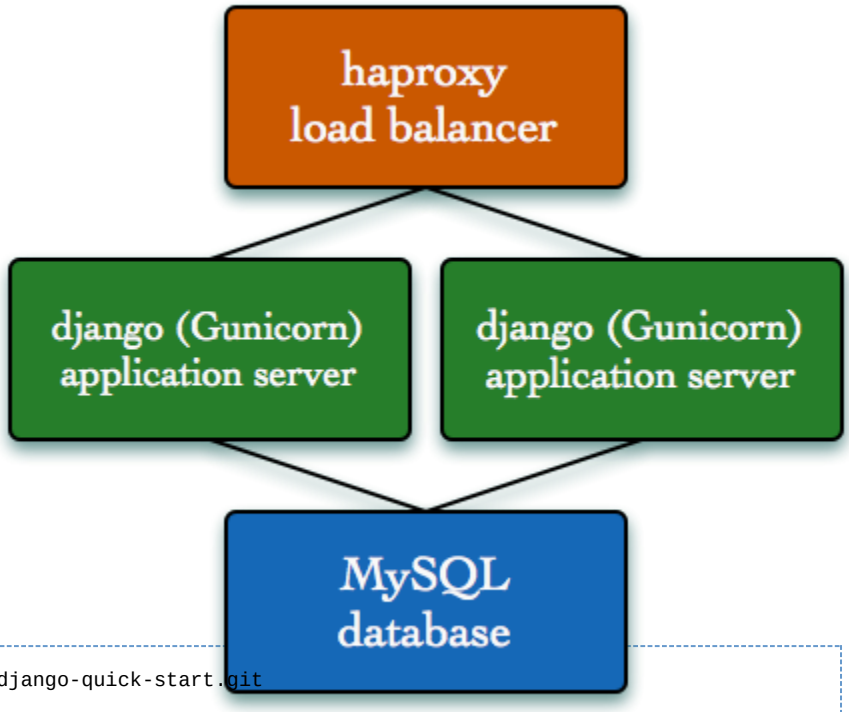


If there are issues with the screencast above, it is also available at blip.tv.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Integrating together with environment roles and the application cookbook, the **database** cookbook looks through the shared **apps** data bag and prepares a mysql database master with an empty database and basic account information for the Django CMS application, leaving it ready for application-specific database bootstrapping steps.

The **django_cms** cookbook does this work and creates an initial superuser for the CMS. This follows the recommended pattern of creating a cookbook named after the application that is being deployed for application-specific setup and configurations.



Environment Setup

First, let's configure the local workstation.

Shell Environment

Obtain the repository used for this guide. It contains all the components required. Use git:

```
git clone git://github.com/opscode/django-quick-start.git
```

Chef and Knife

Ubuntu/Debian users: Install XML2 and XLST development headers on your system:

```
sudo apt-get install libxml2-dev libxslt-dev
```

All Users: You'll need some additional gems for Knife to launch instances in Amazon EC2:

```
sudo gem install knife-ec2
```

As part of the Fast Start Guide, you cloned a chef-repo and copied the Knife configuration file (knife.rb), validation certificate (ORGNAME-validator.pem) and user certificate (USERNAME.pem) to `~/chef-repo/.chef/`. Copy these files to the new django-quick-start repository. You can also re-download the Knife configuration file for your organization with the Hosted Chef Management Console.

```
mkdir ~/django-quick-start/.chef
    cp ~/chef-repo/.chef/knife.rb ~/django-quick-start/.chef
    cp ~/chef-repo/.chef/USERNAME.pem ~/django-quick-start/.chef
    cp ~/chef-repo/.chef/ORGNAME-validator.pem ~/django-quick-start/.chef
```

Add the Amazon AWS credentials to the Knife configuration file.

```
vi ~/django-quick-start/.chef/knife.rb
```

Add the following two lines to the end:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife[:aws_access_key_id] = "replace with the Amazon Access Key ID"
    knife[:aws_secret_access_key] =  "replace with the Amazon Secret Access Key ID"
```

Once the django-quick-start and knife configuration is in place, we'll work from this directory.

```
cd django-quick-start
```

Amazon AWS EC2

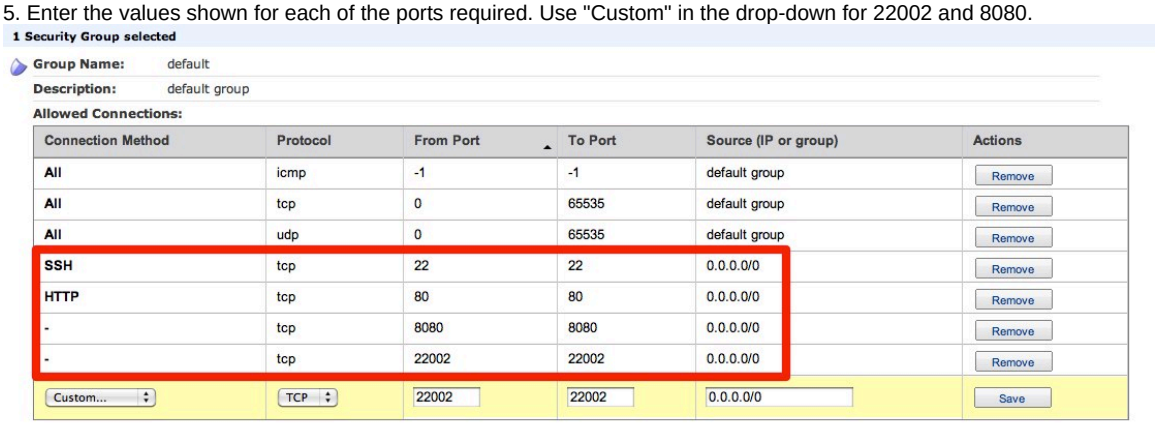
In addition to the credentials, two additional things need to be configured in the AWS account.

Configure the default security group to allow incoming connections for the following ports.

- 22 - ssh
- 80 - haproxy load balancer
- 22002 - haproxy administrative interface
- 8080 - Gunicorn Django application

Add these to the default security group for the account using the AWS Console.

1. Sign into the Amazon AWS Console.
2. Click on the "Amazon EC2" tab at the top.
3. Click on "Security Groups" in the left sidebar of the AWS Console.
4. Select the "Default" group in the main pane.
5. Enter the values shown for each of the ports required. Use "Custom" in the drop-down for 22002 and 8080.



1 Security Group selected

Group Name: default
Description: default group
Allowed Connections:

Connection Method	Protocol	From Port	To Port	Source (IP or group)	Actions
All	icmp	-1	-1	default group	Remove
All	tcp	0	65535	default group	Remove
All	udp	0	65535	default group	Remove
SSH	tcp	22	22	0.0.0.0/0	Remove
HTTP	tcp	80	80	0.0.0.0/0	Remove
-	tcp	8080	8080	0.0.0.0/0	Remove
-	tcp	22002	22002	0.0.0.0/0	Remove
Custom...	TCP	22002	22002	0.0.0.0/0	Save

Create an SSH Key Pair and save the private key in `~/.ssh`.

1. In the AWS Console, click on "Key Pairs" in the left sidebar.
2. Click on "Create Keypair" at the top of the main pane.
3. Give the keypair a name like "django-quick-start".
4. The keypair will be downloaded automatically by the browser and saved to the default Downloads location.
5. Move the django-quick-start.pem file from the default Downloads location to `~/.ssh` and change permissions so that only you can read the file. For example,

```
mv ~/Downloads/django-quick-start.pem ~/.ssh
    chmod 600 ~/.ssh/django-quick-start.pem
```

Acquire Cookbooks

The django-quick-start has all the cookbooks we need for this guide. They were downloaded along with their dependencies from the cookbooks site using Knife. These are in the `cookbooks/` directory.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
apt
    git
    application
    database
    haproxy
```

A single new non-community cookbook was also created for this quick-start. This django_cms cookbook contains a recipe that is used to bootstrap our database and create the initial superuser. This follows the recommended pattern of creating a cookbook named after the application which contains application specific setup and configurations.

```
django_cms
```

Upload all the cookbooks to Hosted Chef.

```
knife cookbook upload -a
```

Server Roles

All the required roles have been created in the django-quick-start repository. They are in the `roles/` directory.

```
base.rb
    django_cms_database_master.rb
    django_cms.rb
    django_cms_load_balancer.rb
```

Upload all the roles to Hosted Chef.

```
rake roles
```

Data Bag Item

The django-quick-start repository contains a data bag item that has all the information required to deploy and configure the Django CMS application from source using the recipes in the **application** and **database** cookbooks.

The data bag name is **apps** and the item name is **django_cms**. Upload this to Hosted Chef.

```
knife data bag create apps
    knife data bag from file apps django_cms.json
```

Decision Time

It is time for you to decide whether you want a single instance running Django CMS, or a few instances as a small infrastructure.

In either case, we're going to use m1.small instances with the 32 bit Ubuntu 10.04 image provided by Canonical. The identifier is **ami-7000f019** fo r the AMI in us-east-1 with instance storage that we will use in this guide. We'll show you the `knife ec2 server create` sub-command to launch instances.

This command will:

- Launch a server on EC2.
- Connect it to Hosted Chef.
- Configure the system with Chef.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

See the appropriate section below for instruction on launching a single instance, or launching the multi-system infrastructure.

Launch Single Instance

Launch the entire stack on a single instance.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S django-quick-start -i ~/.ssh/django-quick-start.pem -x ubuntu \
      -r
'role[base],role[django_cms_database_master],role[django_cms],recipe[django_cms::db_bootstrap],role[d
jango_cms_load_balancer]'
```

Once complete, the instance will be running MySQL and Django CMS under Gunicorn. With only one system, a load balancer is unnecessary.

Launch Multi-instance Infrastructure

We will launch one database server, two application servers and one load balancer. One of the application server instances will include the role for setting up the database as discussed before.

First, launch the database instance.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S django-quick-start -i ~/.ssh/django-quick-start.pem -x ubuntu \
      -r 'role[base],role[django_cms_database_master]'
```

Once the database master is up, launch one node that will set up the database with default data.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S django-quick-start -i ~/.ssh/django-quick-start.pem -x ubuntu \
      -r 'role[base],role[django_cms],recipe[django_cms::db_bootstrap]'
```

Launch the second application instance w/o the django_cms::db_bootstrap recipe.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S django-quick-start -i ~/.ssh/django-quick-start.pem -x ubuntu \
      -r 'role[base],role[django_cms]'
```

Once the second application instance is up, launch the load balancer.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S django-quick-start -i ~/.ssh/django-quick-start.pem -x ubuntu \
      -r 'role[base],role[django_cms_load_balancer]'
```

Once complete, we'll have four instances running in EC2 with MySQL, Django CMS and haproxy up and available to serve traffic.

Verification

Knife will output the fully qualified domain name of the instance when the commands complete. Navigate to the public fully qualified domain name on port 80.

```
http://ec2-xx-xxx-xx-xxx.compute-1.amazonaws.com/
```

The login is admin and the password is djangocms.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

You can access the haproxy admin interface at:

```
http://ec2-xx-xxx-xx-xxx.compute-1.amazonaws.com:22002/
```

Appendix

Database Passwords

The data bag item for dbapp contains default passwords that should certainly be changed to something stronger.

The passwords in the dbapp Data Bag Item are set to the values show below:

```
"mysql_root_password": {
      "_default": "mysql_root"
    },
    "mysql_debian_password": {
      "_default": "mysql_debian"
    },
    "mysql_repl_password": {
      "_default": "mysql_repl"
    },
```

To change the password to something stronger, modify **mysql_root**, **mysql_debian**, **mysql_repl** values. Something like the following secure passwords:

```
vi data_bags/apps/dbapp.json
    "mysql_root_password": {
      "_default": "super_s3cur3_r00t_pw"
    },
    "mysql_debian_password": {
      "_default": "super_s3cur3_d3b1@n_pw"
    },
    "mysql_repl_password": {
      "_default": "super_s3cur3_r3pl_pw"
    },
```

Once the entries are modified, simply load the data bag item from the json file:

```
knife data bag from file apps dbapp.json
```

Non-EC2 Systems

For people not using Amazon EC2, other Cloud computing providers can be used. Supported by knife and fog as of this revision:

- Rackspace Cloud

See the Launch Cloud Instances with Knife for more information about using Knife to launch these instance types.

For people not using cloud at all, but have their own infrastructure and hardware, use the Knife Bootstrap knife command. Note that the run-list specification is slightly different. For the first example of the single instance:

```
knife bootstrap IPADDRESS
     -r
'role[base],role[dbapp_database_master],role[dbapp],recipe[dbapp::db_bootstrap],role[dbapp_load_balan
cer]'
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

See the contextual help for knife bootstrap on the additional options to set for SSH.

```
knife bootstrap --help
```

A Note about EC2 Instances

We used m1.small instances. This is a low performance instance size in EC2 and just fine for testing. Visit the Amazon AWS documentation to learn more about instance sizes.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Build A Java Web Stack

Build a Java Web application stack using Chef cookbooks available from the Cookbooks Community Site and Hosted Chef.

The Walkthrough Guide assumes you followed the Fast Start Guide and have Chef installed and working with the Hosted Chef platform.

At the end of this guide, you'll have four total Ubuntu 10.04 systems running in Amazon EC2.

- 1 haproxy load balancer.
- 2 Tomcat6 web servers.
- 1 MySQL database server.

If you don't already have an account with Amazon AWS, go to Amazon Web Sevices and click "Sign up". You'll need the access and secret access key credentials from the sign-up later.



The Java web application used in this guide is a demo application called **dbapp**. The source code for the WAR (web application archive) can be downloaded on the Opscode GitHub account. The application consists of a servlet that does a simple database query via a JDBC datasource that is retrieved from JNDI and forwards the results onto a JSP page for rendering.

We're going to reuse a number of cookbooks from the Cookbooks Community Site to build the environment.

- The load balancer is **haproxy** because it is very simple to deploy and configure, and we use a recipe that automatically discovers the Tomcat application servers.
- The heavy lifting is handled by recipes in the **application** and **database** cookbooks.

The application cookbook assumes some sort of build process, such as Maven or a Continuous Integration server like Hudson, will create a deployable artifact and make it available for download via HTTP (such as S3). The application cookbook will install Tomcat, download and deploy the dbapp WAR file and create the required Tomcat **context.xml** files (which contains database connection information). The database cookbook installs MySQL, creates any application specific databases and grants application users access to these databases.

We are also following the recommended pattern of creating a cookbook named after the application that is being deployed which contains application specific setup and configurations. In this case, the **dbapp** cookbook contains a recipe that will be used for bootstrapping our database.

Guide Based Upon Ubuntu 10.04 on Amazon AWS EC2 with Chef 0.10.0.

Note: *At this time, the steps described above have only been tested on the identified platform(s). Opscode has not researched and does not support alternative steps that may lead to successful completion on other platforms. Platform(s) supported by this guide may change over time, so please do check back for updates. If you'd like to undertake this guide on an alternate platform, you may desire to turn to open source community resources for support assistance.*



If there are issues with the screencast above, it is also available at blip.tv.

Environment Setup

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

First, let's configure the local workstation.

Shell Environment

Obtain the repository used for this guide. It contains all the components required. Use git:

```
git clone git://github.com/opscode/java-quick-start.git
```

Chef and Knife

All Users: You'll need some additional gems for Knife to launch instances in Amazon EC2:

```
sudo gem install knife-ec2
```

As part of the Fast Start Guide, you cloned a chef-repo and copied the Knife configuration file (knife.rb), validation certificate (ORGNAME-validator.pem) and user certificate (USERNAME.pem) to `~/chef-repo/.chef/`. Copy these files to the new java-quick-start repository. You can also re-download the Knife configuration file for your organization with the Hosted Chef Management Console.

```
mkdir ~/java-quick-start/.chef
    cp ~/chef-repo/.chef/knife.rb ~/java-quick-start/.chef
    cp ~/chef-repo/.chef/USERNAME.pem ~/java-quick-start/.chef
    cp ~/chef-repo/.chef/ORGNAME-validator.pem ~/java-quick-start/.chef
```

Add the Amazon AWS credentials to the Knife configuration file.

```
vi ~/java-quick-start/.chef/knife.rb
```

Add the following two lines to the end:

```
knife[:aws_access_key_id] = "replace with the Amazon Access Key ID"
    knife[:aws_secret_access_key] =  "replace with the Amazon Secret Access Key ID"
```

Once the java-quick-start and knife configuration is in place, we'll work from this directory.

```
cd java-quick-start
```

Amazon AWS EC2

In addition to the credentials, two additional things need to be configured in the AWS account.

Configure the default security group to allow incoming connections for the following ports.

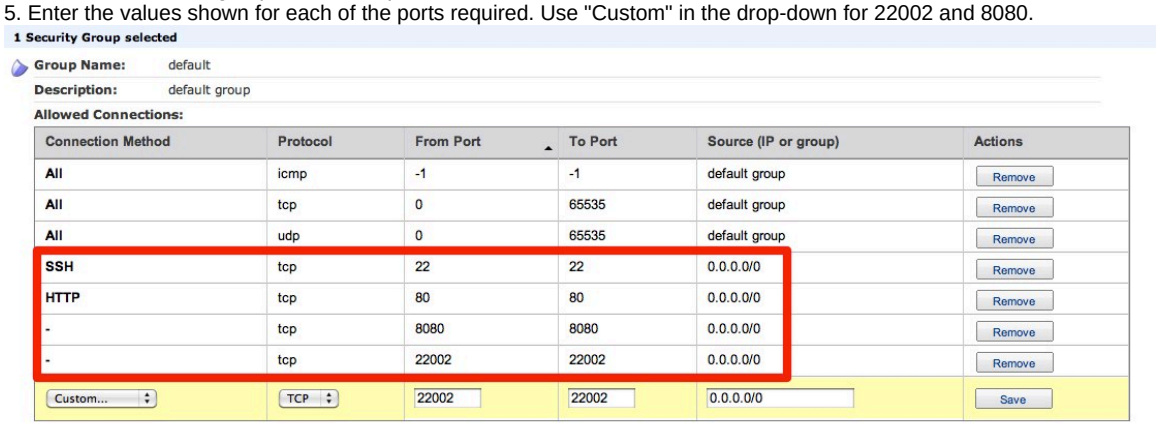
- 22 - ssh
- 80 - haproxy load balancer
- 22002 - haproxy administrative interface
- 8080 - Tomcat servlet engine serving our dbapp application

Add these to the default security group for the account using the AWS Console.

1. Sign into the Amazon AWS Console.
2. Click on the "Amazon EC2" tab at the top.
3. Click on "Security Groups" in the left sidebar of the AWS Console.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

4. Select the "Default" group in the main pane.
5. Enter the values shown for each of the ports required. Use "Custom" in the drop-down for 22002 and 8080.



1 Security Group selected

Group Name: default

Description: default group

Allowed Connections:

Connection Method	Protocol	From Port	To Port	Source (IP or group)	Actions
All	icmp	-1	-1	default group	Remove
All	tcp	0	65535	default group	Remove
All	udp	0	65535	default group	Remove
SSH	tcp	22	22	0.0.0.0/0	Remove
HTTP	tcp	80	80	0.0.0.0/0	Remove
-	tcp	8080	8080	0.0.0.0/0	Remove
-	tcp	22002	22002	0.0.0.0/0	Remove
Custom...	TCP	22002	22002	0.0.0.0/0	Save

Create an SSH Key Pair and save the private key in `~/.ssh`.

1. In the AWS Console, click on "Key Pairs" in the left sidebar.
2. Click on "Create Keypair" at the top of the main pane.
3. Give the keypair a name like "java-quick-start".
4. The keypair will be downloaded automatically by the browser and saved to the default Downloads location.
5. Move the java-quick-start.pem file from the default Downloads location to ***~/.ssh*** and change permissions so that only you can read the file. For example,

```
mv ~/Downloads/java-quick-start.pem ~/.ssh
    chmod 600 ~/.ssh/java-quick-start.pem
```

Acquire Cookbooks

The java-quick-start has all the cookbooks we need for this guide. They were downloaded along with their dependencies from the cookbooks site using Knife. These are in the `cookbooks/` directory.

- apt
- git
- application
- database
- haproxy

A single new non-community cookbook was also created for this quick-start. This **dbapp** cookbook contains a recipe that is used to bootstrap our database. This follows the recommended pattern of creating a cookbook named after the application which contains application specific setup and configurations.

```
dbapp
```

Upload all the cookbooks to Hosted Chef.

```
knife cookbook upload -a
```

Server Roles

All the required roles have been created in the java-quick-start repository. They are in the `roles/` directory.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
base.rb
    dbapp_database_master.rb
    dbapp.rb
    dbapp_load_balancer.rb
```

Upload all the roles to Hosted Chef.

```
rake roles
```

Data Bag Item

The java-quick-start repository contains a data bag item that has all the information required to deploy and configure the Java web application archive (WAR) from S3 using the recipes in the application and database cookbooks.

The data bag name is **apps** and the item name is **dbapp**. Upload this to Hosted Chef.

```
knife data bag create apps
    knife data bag from file apps dbapp.json
```

Decision Time

It is time for you to decide whether you want a single instance running **dbapp**, or a few instances as a small infrastructure.

In either case, we're going to use m1.small instances with the 32 bit Ubuntu 10.04 image provided by Canonical. The identifier is **ami-7000f019** fo r the AMI in us-east-1 with instance storage that we will use in this guide. We'll show you the ***knife ec2 server create*** sub-command to launch instances.

This command will:

- Launch a server on EC2.
- Connect it to Hosted Chef.
- Configure the system with Chef.

See the appropriate section below for instruction on launching a single instance, or launching the multi-system infrastructure.

Launch Single Instance

Launch the entire stack on a single instance.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S java-quick-start -i ~/.ssh/java-quick-start.pem -x ubuntu \
      -r
'role[base],role[dbapp_database_master],role[dbapp],recipe[dbapp::db_bootstrap],role[dbapp_load_balan
cer]'
```

Once complete, the instance will be running MySQL and the Java webapp under Tomcat. With only one system, a load balancer is unnecessary.

Launch Multi-instance Infrastructure

We will launch one database server, two application servers and one load balancer. One of the application server instances will include the role for setting up the database as discussed before.

First, launch the database instance.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S java-quick-start -i ~/.ssh/java-quick-start.pem -x ubuntu \
      -r 'role[base],role[dbapp_database_master]'
```

Once the database master is up, launch one node that will create the database schema and set up the database with default data.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S java-quick-start -i ~/.ssh/java-quick-start.pem -x ubuntu \
      -r 'role[base],role[dbapp],recipe[dbapp::db_bootstrap]'
```

Launch the second application instance w/o the **dbapp::db_bootstrap** recipe.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S java-quick-start -i ~/.ssh/java-quick-start.pem -x ubuntu \
      -r 'role[base],role[dbapp]'
```

Once the second application instance is up, launch the load balancer.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S java-quick-start -i ~/.ssh/java-quick-start.pem -x ubuntu \
      -r 'role[base],role[dbapp_load_balancer]'
```

Once complete, we'll have four instances running in EC2 with MySQL, Tomcat and haproxy up and available to serve traffic.

Verification

Knife will output the fully qualified domain name of the instance when the commands complete. Navigate to the public fully qualified domain name on port 80:

```
http://ec2-xx-xxx-xx-xxx.compute-1.amazonaws.com/
```

You can access the haproxy admin interface at:

```
http://ec2-xx-xxx-xx-xxx.compute-1.amazonaws.com:22002/
```

Appendix

Database Passwords

The data bag item for dbapp contains default passwords that should certainly be changed to something stronger.

The passwords in the dbapp Data Bag Item are set to the values show below:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
"mysql_root_password": {
      "_default": "mysql_root"
    },
    "mysql_debian_password": {
      "_default": "mysql_debian"
    },
    "mysql_repl_password": {
      "_default": "mysql_repl"
    },
```

To change the password to something stronger, modify **mysql_root**, **mysql_debian**, **mysql_repl** values. Something like the following secure passwords:

```
vi data_bags/apps/dbapp.json
    "mysql_root_password": {
      "_default": "super_s3cur3_r00t_pw"
    },
    "mysql_debian_password": {
      "_default": "super_s3cur3_d3b1@n_pw"
    },
    "mysql_repl_password": {
      "_default": "super_s3cur3_r3pl_pw"
    },
```

Once the entries are modified, simply load the data bag item from the json file:

```
knife data bag from file apps dbapp.json
```

Non-EC2 Systems

For people not using Amazon EC2, other Cloud computing providers can be used. Supported by knife and fog as of this revision:

- Rackspace Cloud

See the Launch Cloud Instances with Knife for more information about using Knife to launch these instance types.

For people not using cloud at all, but have their own infrastructure and hardware, use the Knife Bootstrap knife command. Note that the run-list specification is slightly different. For the first example of the single instance:

```
knife bootstrap IPADDRESS
     -r
'role[base],role[dbapp_database_master],role[dbapp],recipe[dbapp::db_bootstrap],role[dbapp_load_balan
cer]'
```

See the contextual help for knife bootstrap on the additional options to set for SSH.

```
knife bootstrap --help
```

A Note about EC2 Instances

We used m1.small instances. This is a low performance instance size in EC2 and just fine for testing. Visit the Amazon AWS documentation to learn more about instance sizes.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Build a LAMP Stack

Build a PHP application stack using Chef cookbooks available from the Cookbooks Community Site and Hosted Chef.



The Walkthrough Guide assumes you followed the Fast Start Guide and have Chef installed and working with the Hosted Chef platform.

At the end of this guide, you'll have four total Ubuntu 10.04 systems running in Amazon EC2.

- 1 haproxy load balancer.
- 2 Apache2 web servers running `mod_php`.
- 1 MySQL database server.

If you don't already have an account with Amazon AWS, go to Amazon Web Sevices and click "Sign up". You'll need the access and secret access key credentials from the sign-up later.

The PHP application used in this guide is MediaWiki 1.18alpha, which powers everyone's favorite encyclopedia Wikipedia.

We're going to reuse a number of cookbooks from the Cookbooks Community Site to build the environment. For example, the source code lives in **git**, so that cookbook will ensure Git is available. The load balancer is **haproxy** because it is very simple to deploy and configure, and we use a recipe that automatically discovers the PHP application systems. The heavy lifting is handled by recipes in the **application** and **database** cookbo oks.

The **application** cookbook will perform the following steps:

- create an application specific virtualenv
- install required packages and pears for the project
- set up the deployment scaffolding
- creates LocalSettings.php file with the database connection information if required
- performs a revision-based deploy
- install Apache2 and mod_php
- create an application specific virtual host configuration file.

We are also following the recommended pattern of creating a cookbook named after the application that is being deployed which contains application specific setup and configurations. In this case, the **mediawiki** cookbook contains a recipe that will be used for bootstrapping our database. It also contains the template that will be used to render our `LocalSettings.php` file.

Guide Based Upon Ubuntu 10.04 on Amazon AWS EC2 with Chef 0.10.0.

Note: *At this time, the steps described above have only been tested on the identified platform(s). Opscode has not researched and does not support alternative steps that may lead to successful completion on other platforms. Platform(s) supported by this guide may change over time, so please do check back for updates. If you'd like to undertake this guide on an alternate platform, you may desire to turn to open source community resources for support assistance.*



If there are issues with the screencast above, it is also available at blip.tv.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Environment Setup

First, let's configure the local workstation.

Shell Environment

Obtain the repository used for this guide. It contains all the components required. Use git:

```
git clone git://github.com/opscode/php-quick-start.git
```

Chef and Knife

Ubuntu/Debian users: Install XML2 and XLST development headers on your system:

```
sudo apt-get install libxml2-dev libxslt-dev
```

All Users: You'll need some additional gems for Knife to launch instances in Amazon EC2:

```
sudo gem install knife-ec2
```

As part of the Fast Start Guide, you cloned a chef-repo and copied the Knife configuration file (knife.rb), validation certificate (ORGNAME-validator.pem) and user certificate (USERNAME.pem) to `~/chef-repo/.chef/`. Copy these files to the new php-quick-start repository. You can also re-download the Knife configuration file for your organization with the Hosted Chef Management Console.

```
mkdir ~/php-quick-start/.chef
    cp ~/chef-repo/.chef/knife.rb ~/php-quick-start/.chef
    cp ~/chef-repo/.chef/USERNAME.pem ~/php-quick-start/.chef
    cp ~/chef-repo/.chef/ORGNAME-validator.pem ~/php-quick-start/.chef
```

Add the Amazon AWS credentials to the Knife configuration file.

```
vi ~/php-quick-start/.chef/knife.rb
```

Add the following two lines to the end:

```
knife[:aws_access_key_id] = "replace with the Amazon Access Key ID"
    knife[:aws_secret_access_key] =  "replace with the Amazon Secret Access Key ID"
```

Once the php-quick-start and knife configuration is in place, we'll work from this directory.

```
cd php-quick-start
```

Amazon AWS EC2

In addition to the credentials, two additional things need to be configured in the AWS account.

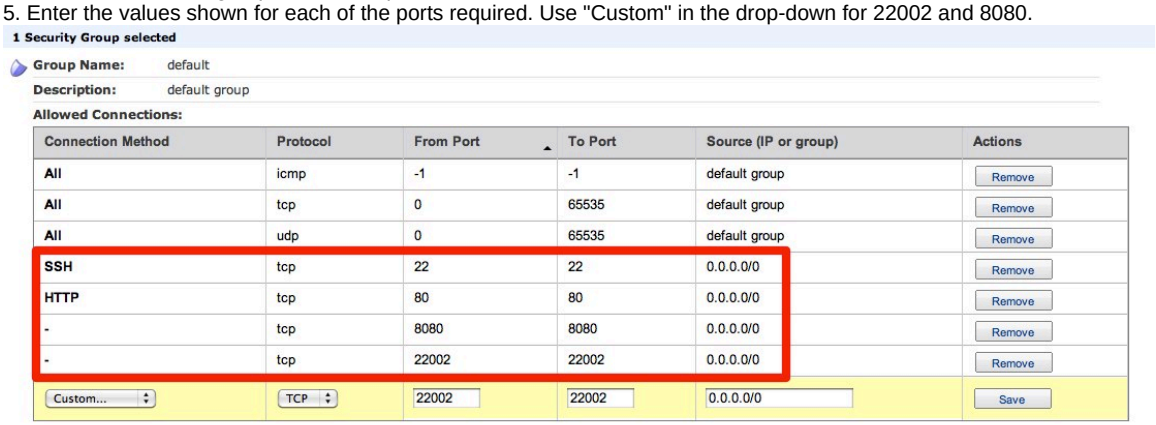
Configure the default security group to allow incoming connections for the following ports.

- 22 - ssh
- 80 - haproxy load balancer
- 22002 - haproxy administrative interface
- 8080 - apache2 web servers running mod_php

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Add these to the default security group for the account using the AWS Console.

1. Sign into the Amazon AWS Console.
2. Click on the "Amazon EC2" tab at the top.
3. Click on "Security Groups" in the left sidebar of the AWS Console.
4. Select the "Default" group in the main pane.
5. Enter the values shown for each of the ports required. Use "Custom" in the drop-down for 22002 and 8080.



1 Security Group selected

Group Name: default

Description: default group

Allowed Connections:

Connection Method	Protocol	From Port	To Port	Source (IP or group)	Actions
All	icmp	-1	-1	default group	Remove
All	tcp	0	65535	default group	Remove
All	udp	0	65535	default group	Remove
SSH	tcp	22	22	0.0.0.0/0	Remove
HTTP	tcp	80	80	0.0.0.0/0	Remove
-	tcp	8080	8080	0.0.0.0/0	Remove
-	tcp	22002	22002	0.0.0.0/0	Remove
Custom...	TCP	22002	22002	0.0.0.0/0	Save

Create an SSH Key Pair and save the private key in `~/.ssh`.

1. In the AWS Console, click on "Key Pairs" in the left sidebar.
2. Click on "Create Keypair" at the top of the main pane.
3. Give the keypair a name like "php-quick-start".
4. The keypair will be downloaded automatically by the browser and saved to the default Downloads location.
5. Move the php-quick-start.pem file from the default Downloads location to `~/.ssh` and change permissions so that only you can read the file. For example,

```
mv ~/Downloads/php-quick-start.pem ~/.ssh
    chmod 600 ~/.ssh/php-quick-start.pem
```

Acquire Cookbooks

The php-quick-start has all the cookbooks we need for this guide. They were downloaded along with their dependencies from the cookbooks site using Knife. These are in the `cookbooks/` directory.

- apt
- git
- application
- database
- haproxy

A single new non-community cookbook was also created for this quick-start. This mediawiki cookbook contains a recipe that is used to bootstrap our database and create the initial superuser. This follows the recommended pattern of creating a cookbook named after the application which contains application specific setup and configurations.

```
mediawiki
```

Upload all the cookbooks to Hosted Chef.

```
knife cookbook upload -a
```

Server Roles

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

All the required roles have been created in the php-quick-start repository. They are in the `roles/` directory.

```
base.rb
    mediawiki_database_master.rb
    mediawiki.rb
    mediawiki_load_balancer.rb
```

Upload all the roles to Hosted Chef.

```
rake roles
```

Data Bag Item

The php-quick-start repository contains a data bag item that has all the information required to deploy and configure the MediaWiki application from source using the recipes in the **application** and **database** cookbooks.

The data bag name is **apps** and the item name is **mediawiki**. Upload this to Hosted Chef.

```
knife data bag create apps
    knife data bag from file apps mediawiki.json
```

Decision Time

It is time for you to decide whether you want a single instance running MediaWiki, or a few instances as a small infrastructure.

In either case, we're going to use m1.small instances with the 32 bit Ubuntu 10.04 image provided by Canonical. The identifier is **ami-7000f019** fo r the AMI in us-east-1 with instance storage that we will use in this guide. We'll show you the `knife ec2 server create` sub-command to launch instances.

This command will:

- Launch a server on EC2.
- Connect it to Hosted Chef.
- Configure the system with Chef.

See the appropriate section below for instruction on launching a single instance, or launching the multi-system infrastructure.

Launch Single Instance

Launch the entire stack on a single instance.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S php-quick-start -i ~/.ssh/php-quick-start.pem -x ubuntu \
      -r
'role[base],role[mediawiki_database_master],role[mediawiki],recipe[mediawiki::db_bootstrap],role[medi
awiki_load_balancer]'
```

Once complete, the instance will be running MySQL and MediaWiki under Apache2 + mod_php. With only one system, a load balancer is unnecessary.

Launch Multi-instance Infrastructure

We will launch one database server, two application servers and one load balancer. One of the application server instances will include the role for running migrations as discussed before.

First, launch the database instance.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S php-quick-start -i ~/.ssh/php-quick-start.pem -x ubuntu \
      -r 'role[base],role[mediawiki_database_master]'
```

Once the database master is up, launch one node that will create the database schema and set up the database with default data.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S php-quick-start -i ~/.ssh/php-quick-start.pem -x ubuntu \
      -r 'role[base],role[mediawiki],recipe[mediawiki::db_bootstrap]'
```

Launch the second application instance w/o the mediawiki::db_bootstrap recipe.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S php-quick-start -i ~/.ssh/php-quick-start.pem -x ubuntu \
      -r 'role[base],role[mediawiki]'
```

Once the second application instance is up, launch the load balancer.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S php-quick-start -i ~/.ssh/php-quick-start.pem -x ubuntu \
      -r 'role[base],role[mediawiki_load_balancer]'
```

Once complete, we'll have four instances running in EC2 with MySQL, MediaWiki and haproxy up and available to serve traffic.

Verification

Knife will output the fully qualified domain name of the instance when the commands complete. Navigate to the public fully qualified domain name on port 80.

```
http://ec2-xx-xxx-xx-xxx.compute-1.amazonaws.com/
```

The login is admin and the password is mediawiki.

You can access the haproxy admin interface at:

```
http://ec2-xx-xxx-xx-xxx.compute-1.amazonaws.com:22002/
```

Appendix

Database Passwords

The data bag item for dbapp contains default passwords that should certainly be changed to something stronger.

The passwords in the dbapp Data Bag Item are set to the values show below:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
"mysql_root_password": {
      "_default": "mysql_root"
    },
    "mysql_debian_password": {
      "_default": "mysql_debian"
    },
    "mysql_repl_password": {
      "_default": "mysql_repl"
    },
```

To change the password to something stronger, modify **mysql_root**, **mysql_debian**, **mysql_repl** values. Something like the following secure passwords:

```
vi data_bags/apps/dbapp.json
    "mysql_root_password": {
      "_default": "super_s3cur3_r00t_pw"
    },
    "mysql_debian_password": {
      "_default": "super_s3cur3_d3b1@n_pw"
    },
    "mysql_repl_password": {
      "_default": "super_s3cur3_r3pl_pw"
    },
```

Once the entries are modified, simply load the data bag item from the json file:

```
knife data bag from file apps dbapp.json
```

Non-EC2 Systems

For people not using Amazon EC2, other Cloud computing providers can be used. Supported by knife and fog as of this revision:

- Rackspace Cloud

See the Launch Cloud Instances with Knife for more information about using Knife to launch these instance types.

For people not using cloud at all, but have their own infrastructure and hardware, use the Knife Bootstrap knife command. Note that the run-list specification is slightly different. For the first example of the single instance:

```
knife bootstrap IPADDRESS
     -r
'role[base],role[dbapp_database_master],role[dbapp],recipe[dbapp::db_bootstrap],role[dbapp_load_balan
cer]'
```

See the contextual help for knife bootstrap on the additional options to set for SSH.

```
knife bootstrap --help
```

A Note about EC2 Instances

We used m1.small instances. This is a low performance instance size in EC2 and just fine for testing. Visit the Amazon AWS documentation to learn more about instance sizes.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Build A Rails Stack

Build a Rails application stack using Chef cookbooks available from the Cookbooks Community Site and Hosted Chef.

At the end of this guide, you'll have four total Ubuntu 10.04 systems running in Amazon EC2.

- 1 haproxy load balancer.
- 2 Ruby on Rails application servers.
- 1 MySQL database server.

If you don't already have an account with Amazon AWS, go to Amazon Web Sevices an d click "Sign up". You'll need the access and secret access key credentials from the sign-up later.



The Ruby on Rails application used in this guide is Radiant CMS.

We're going to reuse a number of cookbooks from the Cookbooks Community Site to build the environment. For example:

- The source code lives in **git**, so that cookbook will ensure Git is available.
- The load balancer is **haproxy** because it is very simple to deploy and configure, and we use a recipe that automatically discovers the Rails application systems.
- The heavy lifting is handled by recipes in the **application** and **database** cookbooks.
- Finally, as we're deploying Radiant, we'll get some help from the **radiant** cookbook.

Guide Based Upon Ubuntu 10.04 on Amazon AWS EC2 with Chef 0.10.0.

Note: *At this time, the steps described above have only been tested on the identified platform(s). Opscode has not researched and does not support alternative steps that may lead to successful completion on other platforms. Platform(s) supported by this guide may change over time, so please do check back for updates. If you'd like to undertake this guide on an alternate platform, you may desire to turn to open source community resources for support assistance.*



If there are issues with the screencast above, it is also available at blip.tv.

Environment Setup

First, let's configure the local workstation.

Shell Environment

Obtain the repository used for this guide. It contains all the components required. Use git:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
git clone git://github.com/opscode/rails-quick-start.git
```

Chef and Knife

Ubuntu/Debian users: Install XML2 and XLST development headers on your system:

```
sudo apt-get install libxml2-dev libxslt-dev
```

All Users: You'll need some additional gems for Knife to launch instances in Amazon EC2:

```
sudo gem install knife-ec2
```

As part of the Fast Start Guide, you cloned a chef-repo and copied the Knife configuration file (knife.rb), validation certificate (ORGNAME-validator.pem) and user certificate (USERNAME.pem) to `~/chef-repo/.chef/`. Copy these files to the new rails-quick-start repository. You can also re-download the Knife configuration file for your organization with the Hosted Chef Management Console.

```
mkdir ~/rails-quick-start/.chef
    cp ~/chef-repo/.chef/knife.rb ~/rails-quick-start/.chef
    cp ~/chef-repo/.chef/USERNAME.pem ~/rails-quick-start/.chef
    cp ~/chef-repo/.chef/ORGNAME-validator.pem ~/rails-quick-start/.chef
```

Add the Amazon AWS credentials to the Knife configuration file.

```
vi ~/rails-quick-start/.chef/knife.rb
```

Add the following two lines to the end:

```
knife[:aws_access_key_id] = "replace with the Amazon Access Key ID"
    knife[:aws_secret_access_key] =  "replace with the Amazon Secret Access Key ID"
```

Once the rails-quick-start and knife configuration is in place, we'll work from this directory.

```
cd rails-quick-start
```

Amazon AWS EC2

In addition to the credentials, two additional things need to be configured in the AWS account.

Configure the default security group to allow incoming connections for the following ports.

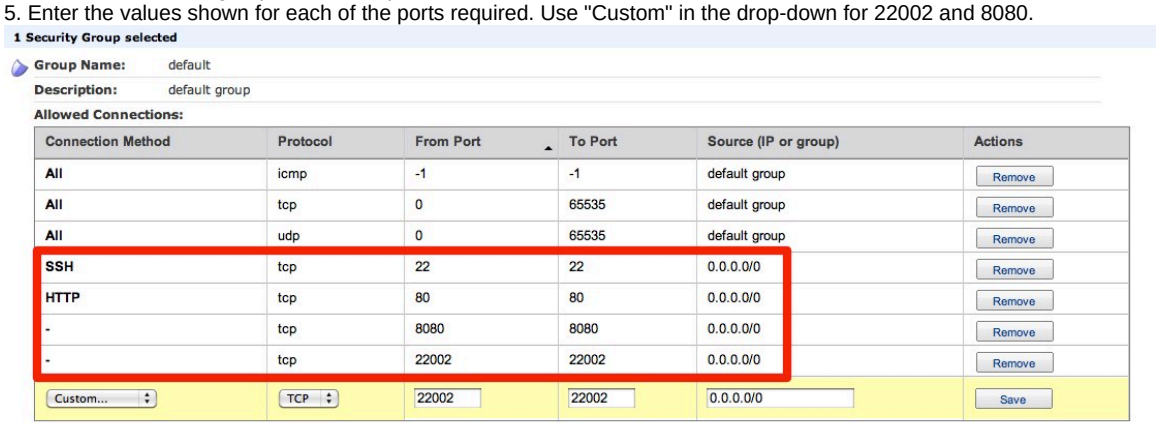
- 22 - ssh
- 80 - haproxy load balancer
- 22002 - haproxy administrative interface
- 8080 - unicorn Rails application

Add these to the default security group for the account using the AWS Console.

1. Sign into the Amazon AWS Console.
2. Click on the "Amazon EC2" tab at the top.
3. Click on "Security Groups" in the left sidebar of the AWS Console.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

4. Select the "Default" group in the main pane.
5. Enter the values shown for each of the ports required. Use "Custom" in the drop-down for 22002 and 8080.



1 Security Group selected

Group Name: default
Description: default group
Allowed Connections:

Connection Method	Protocol	From Port	To Port	Source (IP or group)	Actions
All	icmp	-1	-1	default group	Remove
All	tcp	0	65535	default group	Remove
All	udp	0	65535	default group	Remove
SSH	tcp	22	22	0.0.0.0/0	Remove
HTTP	tcp	80	80	0.0.0.0/0	Remove
-	tcp	8080	8080	0.0.0.0/0	Remove
-	tcp	22002	22002	0.0.0.0/0	Remove
Custom...	TCP	22002	22002	0.0.0.0/0	Save

Create an SSH Key Pair and save the private key in `~/.ssh`.

1. In the AWS Console, click on "Key Pairs" in the left sidebar.
2. Click on "Create Keypair" at the top of the main pane.
3. Give the keypair a name like "rails-quick-start".
4. The keypair will be downloaded automatically by the browser and saved to the default Downloads location.
5. Move the rails-quick-start.pem file from the default Downloads location to ***~/.ssh*** and change permissions so that only you can read the file. For example,

```
mv ~/Downloads/rails-quick-start.pem ~/.ssh
    chmod 600 ~/.ssh/rails-quick-start.pem
```

Acquire Cookbooks

The rails-quick-start has all the cookbooks we need for this guide. They were downloaded along with their dependencies from the cookbooks site using Knife. These are in the `cookbooks/` directory.

- apt
- git
- application
- database
- radiant
- haproxy

Upload all the cookbooks to Hosted Chef.

```
knife cookbook upload -a
```

Server Roles

All the required roles have been created in the rails-quick-start repository. They are in the `roles/` directory.

base.rb
radiant_database_master.rb
radiant.rb
radiant_run_migrations.rb
radiant_load_balancer.rb

Upload all the roles to Hosted Chef.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
rake roles
```

Data Bag Item

The rails-quick-start repository contains a data bag item that has all the information required to deploy and configure the Radiant application from source using the recipes in the **application** and **database** cookbooks.

The data bag name is **apps** and the item name is **radiant**. Upload this to Hosted Chef.

```
knife data bag create apps
    knife data bag from file apps radiant.json
```

Decision Time

It is time for you to decide whether you want a single instance running Radiant, or a few instances as a small infrastructure.

In either case, we're going to use m1.small instances with the 32 bit Ubuntu 10.04 image provided [by Canonical|http://uec-images.ubuntu.com/releases/10.04/release-20101228/. The identifier is **ami-88f504e1** for the AMI in us-east-1 with instance storage that we will use in this guide. We'll show you the **knife ec2 server create** sub-command to launch instances.

This command will:

- Launch a server on EC2.
- Connect it to Hosted Chef.
- Configure the system with Chef.

See the appropriate section below for instruction on launching a single instance, or launching the multi-system infrastructure.

Launch Single Instance

Launch the entire stack on a single instance.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S rails-quick-start -i ~/.ssh/rails-quick-start.pem -x ubuntu -d ubuntu10.04-gems-qs \
      -r
'role[base],role[radiant_database_master],role[radiant],role[radiant_run_migrations],recipe[radiant::
db_bootstrap],role[radiant_load_balancer]'
```

Once complete, the instance will be running MySQL and Radiant under Unicorn. With only one system, a load balancer is unnecessary.

Launch Multi-instance Infrastructure

We will launch one database server, two application servers and one load balancer. One of the application server instances will include the role for running migrations as discussed before.

First, launch the database instance.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S rails-quick-start -i ~/.ssh/rails-quick-start.pem -x ubuntu -d ubuntu10.04-gems-qs \
      -r 'role[base],role[radiant_database_master]'
```

Once the database master is up, launch one node that will run database migration and set up the database with default data.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S rails-quick-start -i ~/.ssh/rails-quick-start.pem -x ubuntu -d ubuntu10.04-gems-qs \
      -r 'role[base],role[radiant],role[radiant_run_migrations],recipe[radiant::db_bootstrap]'
```

Launch the second application instance w/o the ***radiant_run_migrations*** role or ***radiant::db_bootstrap*** recipe.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S rails-quick-start -i ~/.ssh/rails-quick-start.pem -x ubuntu -d ubuntu10.04-gems-qs \
      -r 'role[base],role[radiant]'
```

Once the second application instance is up, launch the load balancer.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
      -S rails-quick-start -i ~/.ssh/rails-quick-start.pem -x ubuntu -d ubuntu10.04-gems-qs \
      -r 'role[base],role[radiant_load_balancer]'
```

Once complete, we'll have four instances running in EC2 with MySQL, Radiant and haproxy up and available to serve traffic.

Verification

Knife will output the fully qualified domain name of the instance when the commands complete. Navigate to the public fully qualified domain name on port 80:

```
http://ec2-xx-xxx-xx-xxx.compute-1.amazonaws.com/
```

The login is admin and the password is radiant.

You can access the haproxy admin interface at:

```
http://ec2-xx-xxx-xx-xxx.compute-1.amazonaws.com:22002/
```

Appendix

Database Passwords

The data bag item for Radiant contains default passwords that should certainly be changed to something stronger.

The passwords in the Radiant Data Bag Item are set to the values show below:

```
"mysql_root_password": {
      "_default": "mysql_root"
    },
    "mysql_debian_password": {
      "_default": "mysql_debian"
    },
    "mysql_repl_password": {
      "_default": "mysql_repl"
    },
```

To change the password to something stronger, modify **mysql_root**, **mysql_debian**, **mysql_repl** values. Something like the following secure passwords:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
vi data_bags/apps/radiant.json
    "mysql_root_password": {
      "_default": "super_s3cur3_r00t_pw"
    },
    "mysql_debian_password": {
      "_default": "super_s3cur3_d3b1@n_pw"
    },
    "mysql_repl_password": {
      "_default": "super_s3cur3_r3pl_pw"
    },
```

Once the entries are modified, simply load the data bag item from the json file:

```
knife data bag from file apps radiant.json
```

Non-EC2 Systems

For people not using Amazon EC2, other Cloud computing providers can be used. Supported by knife and fog as of this revision:

- Rackspace Cloud

See Launch Cloud Instances with Knife for more information about using Knife to launch these instance types.

For people not using cloud at all, but have their own infrastructure and hardware, use the Knife Bootstrap knife command. Note that the run-list specification is slightly different. For the first example of the single instance:

```
knife bootstrap IPADDRESS \
    -r
'role[base],role[radiant_database_master],role[radiant],role[radiant_run_migrations],recipe[radiant::
db_bootstrap],role[radiant_load_balancer]'
```

See the contextual help for knife bootstrap on the additional options to set for SSH.

```
knife bootstrap --help
```

A Note about EC2 Instances

We used m1.small instances. This is a low performance instance size in EC2 and just fine for testing. Visit the Amazon AWS documentation to lea rn more about instance sizes.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Gem Man Pages



Knife Man Pages in Chef 0.10
In Chef 0.10, access to the knife man pages is directly available with `knife help` See `knife help list` for topics.

The Chef gem includes man pages, but the default RubyGem installation location is not in the `$MANPATH`. You can view man pages from any installed gems with `gem-man`.

Install the gem-man gem, then use the man sub-command for gem to view the man pages.

```
% sudo gem install gem-man
Password:
Successfully installed gem-man-0.2.0
1 gem installed
Installing RDoc documentation for gem-man-0.2.0...
% gem man chef
View which manual?
 1. chef-indexer(1)
 2. chef-server-webui(1)
 3. chef-server(1)
 4. chef-solr-indexer(1)
 5. chef-solr(1)
 6. chef-client(8)
 7. chef-solo(8)
 8. chef-solr-rebuild(8)
 9. knife(8)
 10. shef(8)
> 9
```

Operating System packaged versions of Chef include the man pages in the correct location for the OS.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Deploying OpenStack with Chef

Overview



This page is a brief guide to deploying OpenStack with the cookbooks currently available from Matt Ray's GitHub repository (http://github.com/mattray/openstack-cookbooks/tree/cactus). It is currently intended for deploying the point-release codenamed "Cactus", other branches will be added as they are released ("Bexar" is still available as a branch). The intention for the repository is to only support major releases.

This code base is under development, so ask Matt Ray about the current state if these instructions aren't currently working.

Status

Right now the Cactus cookbooks support a single machine with Glance and Nova, but they are under active development. Other components will be added in time and new releases will be added as they are available. There is currently no intention to continue development on previous releases once a new release is available (but they are available and patches are welcome!). The eventual goal is to support all the configuration options available for OpenStack with Chef cookbooks.

All code in the repository is Apache 2 licensed and covered by the Contribution License Agreement. Patches will not be accepted without this requirement.

Requirements

This has been primarily tested with Ubuntu 10.04 (a bit with 10.10) and Chef 0.10 and later.

Hardware Requirements

- OpenStack Compute currently requires x86 hardware with virtualization support (KVM is currently the only hypervisor).
- OpenStack Object Storage should not be deployed with RAID.

OpenStack

What is OpenStack?

> *OpenStack is a collection of open source technologies delivering a massively scalable cloud operating system. OpenStack is currently developing two interrelated projects: OpenStack Compute and OpenStack Object Storage. OpenStack Compute is software to provision and manage large groups of virtual private servers, and OpenStack Object Storage is software for creating redundant, scalable object storage using clusters of commodity servers to store terabytes or even petabytes of data.*

- **OpenStack Compute**, code-named *"Nova"*, provides the software to provision virtual machines on standard hardware on a massive scale.
- **OpenStack Image Registry**, code-named *"Glance"*, provides services for discovering, registering, and retrieving virtual machine images.
- **OpenStack Object Storage**, code-named *"Swift"*, provides the software to store billions of objects distributed across standard hardware.

Code

The cookbooks and configuration files are currently available from Matt Ray's GitHub repository.

(note the "cactus" branch)

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
git clone git://github.com/mattray/openstack-cookbooks.git
```

These were originally forked from the work in Anso Labs' OpenStack-Cookbooks but have diverged greatly since the original fork.

Matt Ray talks Crowbar, Chef, and OpenStack integration for building private clouds

Matt Ray discusses the integration work he is doing with Crowbar (a soon to be open source bare-metal provisioning tool from Dell), Chef, and OpenStack. This toolchain stretches the concept of infrastructure as code all the way from the bios to the provisioning of a private cloud... it's a "datacenter as code".

Watch the Video on the dev2ops blog.



Our friends from the CloundAve Blog put together an excellent overview of OpenStack cloud solution installer, Crowbar - and its' secret sauce: **Opscode Chef, The Secret Sauce Behind Dell Open Source Crowbar**

Automated Deployment of OpenStack With Chef

Watch Opscode team member Matt Ray's Presentation from Velocity 2011 on Deploying OpenStack with Opscode Chef

Cookbooks

Download the following cookbooks:

```
knife cookbook site download apache2 0.99.4
knife cookbook site download apt 1.1.2
knife cookbook site download mysql 1.0.5
knife cookbook site download openssl 1.0.0
knife cookbook site download rabbitmq 1.2.1
```

- **glance**
  - Included in the repository.
- **nova**
  - Included in the repository.
- **swift**
  - Included in the repository, untested.

Once the cookbooks have been all been downloaded and untarred into the cookbooks directory

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife cookbook upload -a
```

Cooking Up OpenStack Chef Recipes

OpenStack team members from Dell also have some Swift and Nova Chef recipes available.

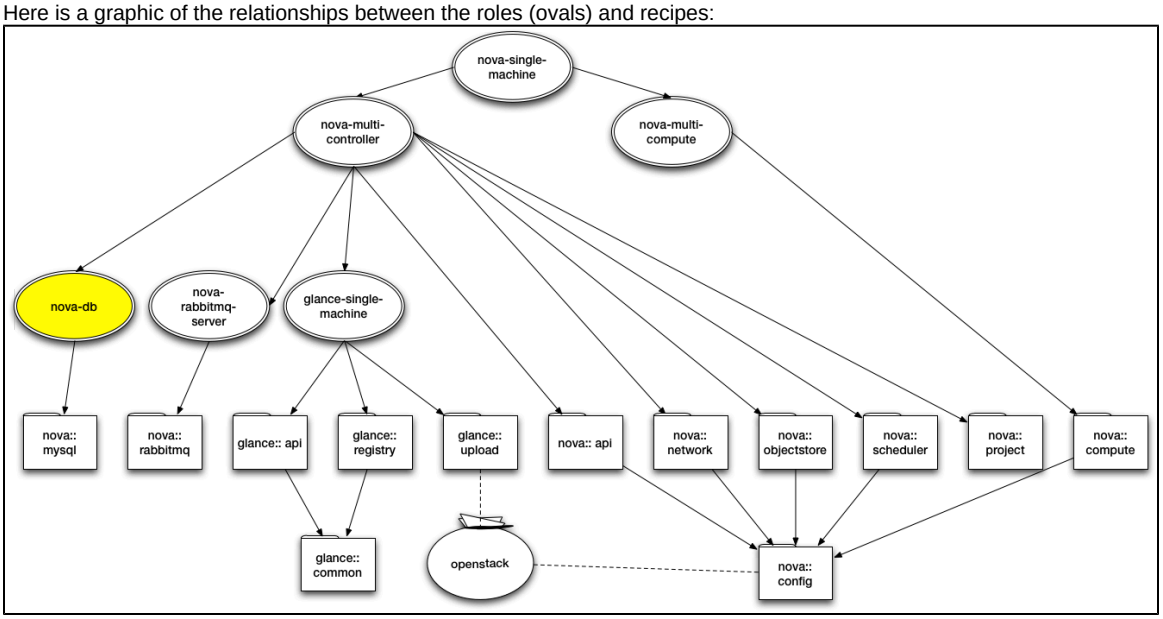
Roles

From the openstack-cookbooks directory

```
rake roles
```

to load all the roles in the repository.

Here is a graphic of the relationships between the roles (ovals) and recipes:



nova-single-machine

Installs everything required to run Nova on a single machine, a special case of multi-node and simply depends on the "nova-multi-controller" and "nova-multi-compute" roles.

nova-multi-controller

- **nova-db**, role providing MySQL configuration
- **nova-rabbitmq-server**, role providing RabbitMQ configuration
- **glance-single-machine,** role installing the Glance service and uploading AMIs
- **nova-api,** recipe providing the nova-api service
- **nova-network,** recipe providing the nova-network service
- **nova-scheduler,** recipe providing the nova-scheduler service
- **nova-objectstore,** recipe providing the nova-objectstore service
- **nova-project,** configures the projects and credentials for the nova installation

nova-multi-compute

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- **nova-compute** components, installing KVM and providing the hypervisor.

Configure Attributes for Deployment

OpenStack is a complicated application with many configurable and interchangeable components. Once you have your repository and have uploaded your cookbooks and roles, you will need to configure and upload the **openstack** data bag. Glance and Nova are configured with the **glance** and **nova** JSON files in the data_bags/openstack/ directory. AMIs to be installed are managed with the **images** JSON file. You may upload these with

```
rake databag:upload_all
```

There are also attributes available for configuration, these may be overridden as necessary. Once these are in place you may apply the roles to nodes. The default installation currently uses MySQL, RabbitMQ and KVM and only deploys Glance on the Nova Controller (so no integration with Swift yet).

Single node

```
knife node run_list add server "role[nova-single-machine]"
```

and run the chef-client on the machine.

Multi node

```
knife node run_list add mycontroller "role[nova-multi-controller]"
knife node run_list add mycomputenodeN "role[nova-multi-compute]"
```

and run the chef-client on the controller and then compute node(s).

Verify your Installation

Once you've deployed your servers, SSH into the controller node (or single box) and

```
sudo su - nova
```

then, as the nova user verify that your services are all running properly:

```
nova@$ nova-manage service list
mycontroller nova-scheduler enabled  :-) 2011-06-23 18:30:45
mycontroller nova-network enabled  :-) 2011-06-23 18:30:48
mycomputenodeN nova-compute enabled  :-) 2011-06-23 18:30:50
```

and list the available images

```
nova@mycontroller:~$ euca-describe-images
IMAGE aki-00000001 None (lucid-server-uec-amd64-vmlinuz-virtual)  available public   kernel
IMAGE ami-00000002 None (lucid-server-uec-amd64.img)  available public   machine aki-00000001
IMAGE aki-00000003 None (maverick-server-uec-amd64-vmlinuz-virtual)  available public   kernel
IMAGE ami-00000004 None (maverick-server-uec-amd64.img)  available public   machine aki-00000003
```

select the ami-00000002 (or whichever you prefer) and enter the following

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
nova@$ euca-run-instances ami-00000002 -k mykey -t m1.tiny
RESERVATION r-mro4sk16 admin default
INSTANCE i-00000001 ami-00000002   scheduling mykey (admin, None) 0  m1.tiny 2011-06-23T22:17:31Z
unknown zone
```

see the progress as your instance starts up

```
nova@$ euca-describe-instances
RESERVATION r-mro4sk16 admin default
INSTANCE i-00000001 ami-00000002 192.168.11.2 192.168.11.2 running mykey (admin, mycontroller) 0
m1.tiny 2011-06-23T22:17:31Z nova
```

once it is running

```
nova@$ ssh -i mykey.priv ubuntu@192.168.11.2
```

and connect to your new OpenStack instance.

knife openstack

OpenStack is supported by the Chef command-line tool knife with Chef 0.10 and the knife openstack plugin. You can install it with "gem install knife-openstack".

You will need to update your .chef/knife.rb with the following:

```
knife[:openstack_access_key_id]     = "Your OpenStack Access Key ID"
knife[:openstack_secret_access_key] = "Your OpenStack Secret Access Key"
knife[:openstack_api_endpoint]      = "http://mycontroller:8773/service/Cloud"
```

You can get the access_key and secret_key values with "knife node show mycontroller -a nova | grep key"

You should now be able to run

```
knife openstack server create -S mykey -x ubuntu -I ami-52f4e131 -f m1.tiny
```

which currently fails to bootstrap, but creates the instance (euca-describe-instances will list it). http://tickets.opscode.com/browse/KNIFE_OPENSTACK-1

Roadmap

This is a non-ordered list of things that are prioritized to work on.

- More fine-grained roles for different deployment environments
  - Pluggable/Modular Roles
    - Databases (MySQL/PostgreSQL/Drizzle/etc.)
    - ObjectStore implementations
    - Network configurations
    - Hypervisors (KVM/Xen/LXC/etc.)
  - Swift integration
- Dashboard
  - simple Django application, recipe already exists
- Swift
  - cookbook exists, untested
  - pluggable/modular roles like Nova

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- Diablo

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Guide to Creating A Cookbook and Writing A Recipe



Introduction

Opscode Team Member Sean OMeara put together a Brief Chef Tutorial on his personal blog. A portion of it is reproduced here, to provide an example...

From concept to release ... the birth of a NTP Cookbook.



You may want to also review the **Cookbook Fast Start Guide**, within the Installation section.

Hello, NTP!

A machine's NTP client is simple to install and configure. Every systems administrator is already familiar with it, which makes it a great example.

Most software available as a native package in a given linux distribution can be managed with a "package, template, service" design pattern. Each of those words refers to a Chef resource, which we pass arguments to.

Step One : Creating an ntp cookbook

documentation reference	How do I create a new Cookbook?

```
$ knife cookbook create ntp
```

This creates a directory structure for the ntp cookbook. You can check it out with ls:

```
$ ls -la cookbooks/ntp/
total 24
drwxr-xr-x  13 someara  staff   442 Mar 14 17:56 .
drwxr-xr-x  36 someara  staff  1224 Mar 15 19:39 ..
-rw-r--r--   1 someara  staff    58 Mar 14 17:56 README.rdoc
drwxr-xr-x   2 someara  staff    68 Mar 14 17:56 attributes
drwxr-xr-x   2 someara  staff    68 Mar 14 17:56 definitions
drwxr-xr-x   3 someara  staff   102 Mar 14 17:56 files
drwxr-xr-x   2 someara  staff    68 Mar 14 17:56 libraries
-rw-r--r--   1 someara  staff   259 Mar 14 17:56 metadata.rb
drwxr-xr-x   2 someara  staff    68 Mar 14 17:56 providers
drwxr-xr-x   4 someara  staff   136 Mar 14 17:56 recipes
drwxr-xr-x   2 someara  staff    68 Mar 14 17:56 resources
drwxr-xr-x   3 someara  staff   102 Mar 14 17:56 templates
```

Step Two : Deciding what to name the recipe

documentation reference	Using Recipes

Recipe names are related to cookbook structure. Putting `recipe[foo::bar]` in a node's run list results in `cookbooks/foo/recipes/bar.rb` being downloaded from chef-server and executed.

There is a special recipe in every cookbook called `default.rb`. It is executed by every recipe in the cookbook. Specifying `recipe[foo::bar]` actually results in `cookbooks/foo/recipes/default.rb`, as well as `cookbooks/foo/recipes/bar.rb` being executed.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Default.rb is a good place to put common stuff when writing cookbooks with multiple recipes, but we're going to keep it simple and just use `default.rb` for everything.

Step Three : Creating a recipe

documentation reference	Recipes	Resources	Just Enough Ruby for Chef

This is where all the fun stuff happens. When using resources, you're writing things in a declarative fashion. Declarative means you can concentrate on the WHAT without having to worry about the HOW. Chef will take care of that for you with something called a resource provider. When installing a package, it will check to see what your operating system is and use the appropriate methodology. For example, on Debian based systems, it will use apt-get, and on Redhat based systems, it will use yum.

```
$ vim cookbooks/ntp/recipes/default.rb

package "ntp" do
    action [:install]
end

template "/etc/ntp.conf" do
    source "ntp.conf.erb"
    variables( :ntp_server => "time.nist.gov" )
end

service "ntpd" do
    action[:enable,:start]
end
```

Chef recipes are evaluated top down (like a normal ruby program), with each resource being ran in the order it appears. Order is important. In the above example, if we were to reverse the order of those three resources, it would first fail to start the service (as the software is not installed yet), then write the configuration file, then finally clobber the file it just wrote by installing the package.

Step Four : Creating the ntp.conf.erb template

documentation reference	Customizing Templates and Files	Templates

```
$ vim cookbooks/ntp/templates/default/ntp.conf.erb

# generated by Chef.
restrict default kod nomodify notrap nopeer noquery
restrict -6 default kod nomodify notrap nopeer noquery
restrict 127.0.0.1
restrict -6 ::1
server <%= @ntp_server %>
server  127.127.1.0     # local clock
driftfile /var/lib/ntp/drift
keys /etc/ntp/keys
```

Step Five : uploading the cookbook to chef-server

documentation reference	How do I upload my cookbooks to the Chef Server?

```
$ knife cookbook upload ntp
```

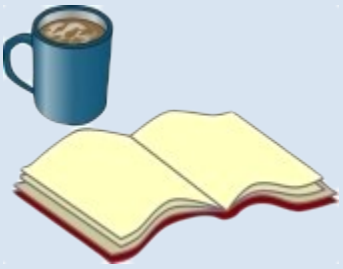
More?

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

For more of the tutorial from Sean, see **A Brief Chef Tutorial (from concentrate)**. For more information on any of the steps above, see the documentation reference listed.

Want Another Example on Writing a Chef Cookbook?

For another guide, Opscode teammate Joshua Timberman has **Guide to Writing Chef Cookbooks** on his personal blog. Check it out for an additional tutorial on building a cookbook for automation of your infrastructure.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Headless Branches for Cookbook Repositories



While Opscode isn't using this method for our own Cookbooks, other people in the community who wish to collaborate and share cookbooks may be interested in this technique.

From the README.markdown in Community Member Tim Dysinger's repository

Instead of one big cookbooks dir to rule them all, you can merge in the branches you would like into your main git-based cookbooks repo.

Create a new cookbook repo

```
cd /my/teh/awesome/projects
mkdir cookbooks
cd cookbooks
git init
touch .gitignore
git add .gitignore
git commit -m "Initial Commit"
```

Add this to your $HOME/.gitconfig

```
[alias]
        headless = !sh -c 'git symbolic-ref HEAD refs/heads/$0 && rm .git/index && git clean -fdx'
```

Create a new headless branch for your own recipe

```
git stash
git headless my-new-recipe
touch .gitignore
git add .gitignore
git commit -m "Initial Commit"
git push origin my-new-recipe
```

Fork a recipe from someone else and work on it

```
git stash
git remote add elsewhere git://github.com/XXX/cookbooks.git
git remote update
git checkout -b milkshake elsewhere/milkshake ; # & work work work
git commit -a -m "Changed some stuff"
git push origin milkshake
```

You can share a branch with multiple related recipes to make it easy for others

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
git stash
git headless my-new-shared-cookbook
touch .gitignore
git add .gitignore
git commit -m "Initial Commit"
git branch ; # list all the recipes
git merge my-1st-recipe
git merge my-2nd-recipe ; # needed by the 1st
git merge my-3rd-recipe ; # needed by the 2nd
git push origin my-new-shared-cookbook
```

You can create a new headless branch for each of your clusters you manage

```
git stash
git headless my-new-cluster-cookbook
touch .gitignore
git add .gitignore
git commit -m "Initial Commit"
git branch ; # list all the recipes
git merge git
git merge apache2
git merge couchdb
git merge erlang
git push mycluster my-new-cluster-cookbook
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

How to Proxy Chef Server with Apache

This procedure describes how to use Apache's mod_proxy to provide an SSL front-end to the Chef Server Merb processes, without adding Passenger to the mix. This is tested on Debian and Ubuntu using the Opscode APT packages, and will require adjustment for other platforms' methods of configuring Apache.

New in Chef 0.8.x
The API and WebUI are separate, and run on separate ports on the Chef Server. This will require setting up multiple virtual hosts. Use two separate CNAMEs in your DNS, for example:

- chef.example.com - API
- chef-www.example.com - WebUI

The instructions below still require modification to reflect these changes, however, the Opscode Chef Cookbook apache-proxy recipe is up to date.

Configure and Start Chef-Server

The Chef Server is a Merb application, and the configuration file is /etc/chef/server.rb. Here's the minimum used in this configuration. Filesystem locations are FHS compliant per Debian Packaging. Adjust the locations for your platform requirements or preferences.

/etc/chef/server.rb

```
log_level          :debug
log_location       "/var/log/chef/server.log"
file_cache_path    "/srv/chef/cache"
ssl_verify_mode    :verify_none
chef_server_url    "http://127.0.0.1:4000"
cookbook_path      [ "/srv/chef/cookbooks", "/srv/chef/site-cookbooks" ]
search_index_path  "/var/chef/search_index"
role_path          "/var/chef/roles"
signing_ca_path    "/var/chef/ca"
couchdb_database   'chef'

validation_client_name "chef-validator"
validation_key   "/etc/chef/validation.pem"
client_key       "/etc/chef/client.pem"
web_ui_client_name "chef-webui"
web_ui_key "/etc/chef/webui.pem"
supportdir = "/srv/chef/support"
solr_jetty_path File.join(supportdir, "solr", "jetty")
solr_heap_size "250M"
solr_data_path File.join(supportdir, "solr", "data")
solr_home_path File.join(supportdir, "solr", "home")
solr_heap_size "256M"

Mixlib::Log::Formatter.show_time = false
```

Once the configuration file is tweaked as required for your environment, start the chef-server. On systems with the Debian/Ubuntu

Install Apache

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Install Apache for your platform. For example, on Debian-based systems, use APT:

```
apt-get install apache2
```

Enable Apache Modules

The following Apache modules need to be enabled:

- proxy
- proxy_http
- proxy_balancer
- ssl
- rewrite
- headers

For Debian-based systems, use the `a2enmod` script.

```
for a2mod in proxy proxy_http proxy_balancer ssl rewrite headers
do
  sudo a2enmod $a2mod
done
```

See your platform's Apache documentation if you're not runnin Debian/Ubuntu.

Create Chef Server Virtual Host

Set up a vhost config file for the Chef Server. This will use the Proxy balancer for the Merb workers running on the localhost. Replace "server_fqdn" with the fully qualified domain name of the server.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

/etc/apache2/sites-available/chef_server.conf

```
<VirtualHost *:443>
  ServerName server_fqdn
  DocumentRoot /usr/share/chef-server-api/public

  <Proxy balancer://chef_server>
    BalancerMember http://127.0.0.1:4000
    Order deny,allow
    Allow from all
  </Proxy>

  LogLevel info
  ErrorLog /var/log/apache2/chef_server-error.log
  CustomLog /var/log/apache2/chef_server-access.log combined

  SSLEngine On
  SSLCertificateFile /etc/chef/certificates/server_fqdn.pem
  SSLCertificateKeyFile /etc/chef/certificates/server_fqdn.pem

  RequestHeader set X_FORWARDED_PROTO 'https'

  RewriteEngine On
  RewriteCond %{DOCUMENT_ROOT}/%{REQUEST_FILENAME} !-f
  RewriteRule ^/(.*)$ balancer://chef_server%{REQUEST_URI} [P,QSA,L]
</VirtualHost>

<VirtualHost *:444>
  ServerName server_fqdn
  DocumentRoot /usr/share/chef-server-webui/public

  <Proxy balancer://chef_server_webui>
    BalancerMember http://127.0.0.1:4040
    Order deny,allow
    Allow from all
  </Proxy>

  LogLevel info
  ErrorLog /var/log/apache2/chef_server-error.log
  CustomLog /var/log/apache2/chef_server-access.log combined

  SSLEngine On
  SSLCertificateFile /etc/chef/certificates/server_fqdn.pem
  SSLCertificateKeyFile /etc/chef/certificates/server_fqdn.pem

  RequestHeader set X_FORWARDED_PROTO 'https'

  RewriteEngine On
  RewriteCond %{DOCUMENT_ROOT}/%{REQUEST_FILENAME} !-f
  RewriteRule ^/(.*)$ balancer://chef_server_webui%{REQUEST_URI} [P,QSA,L]
</VirtualHost>
```

On Debian/Ubuntu systems, use `a2ensite` script to enable the vhost:

```
sudo a2ensite chef_server.conf
```

Add listen Port for the WebUI

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Add `Listen 444` to the Apache config. On Debian/Ubuntu, this is in `/etc/apache2/ports.conf`. Depending on your environment, you may need to enable access to port 444 on your firewall.

```
NameVirtualHost *:80
Listen 80

<IfModule mod_ssl.c>
    Listen 443
    Listen 444
</IfModule>
```



Default Port 80
If you're not using another web service running on port 80, you may wish to set up a mod_rewrite rule to force SSL.

Create SSL Certificates

If you're using the Chef Repository, you can use the `ssl_cert` rake task to create a self-signed certificate.

```
rake ssl_cert FQDN=`hostname -f`
sudo mkdir -p /etc/chef/certificates
sudo cp certificates/`hostname -f`.pem /etc/chef/certificates
```

If you purchased an SSL certificate, you can use that instead, be sure to name it appropriately and change the vhost configuration as required.



Chef Repository
We recommend using the Chef Repository, anyway. The Rakefile provides a number of helpers for maintaining Chef, and it is a convenient way to get started using a Version Control System for your infrastructure's configuration.

(Re)Start Apache

With all the proper bits in place, (re)start Apache.

```
sudo /etc/init.d/apache2 restart
```

Client Configuration

The following configuration is required for clients to connect to the Chef server.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

/etc/chef/client.rb

```
log_level        :info
log_location     STDOUT
ssl_verify_mode  :verify_none
chef_server_url  "https://#{chef_server_fqdn}"
validation_client_name "chef-validator"
file_cache_path  "/srv/chef/cache"
if FileTests.exists?("/etc/chef/validation.pem")
  validation_key   "/etc/chef/validation.pem"
end
client_key       "/etc/chef/client.pem"

pid_file           "/var/run/chef/client.pid"

Mixlib::Log::Formatter.show_time = true
```

Replace server_fqdn with the server's fully qualified domain name.

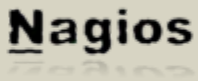
Next Steps

Point your browser at https://server_fqdn:444, where server_fqdn is the server you just configured. You should get the login page.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Nagios Quick Start

Build a Nagios monitoring server using Chef cookbooks available from the Cookbooks Community Site and Hosted Chef.



The Walkthrough Guide assumes you followed the Fast Start Guide and have Chef installed and working with the Hosted Chef platform.

At the end of this guide, you'll have one Ubuntu 10.04 system running Nagios in Amazon EC2.

If you don't already have an account with Amazon AWS, go to Amazon Web Sevices and click "Sign up". You'll need the access and secret access key credentials from the sign-up later.

We are going to reuse a few cookbooks from the Cookbooks Community Site to build the environment. The **nagios** cookbook is required, of course. The Nagios web interface will be set up with **apache2**. Finally, we will need the **apt** cookbook to ensure the package cache is updated.

If you do not already have an account with Amazon AWS, go to Amazon Web Sevices and click "Sign up". You will need the access and secret access key credentials from the sign-up later.

Guide Based Upon Ubuntu 10.04 on Amazon AWS EC2 with Chef 0.10.0.

Note: *At this time, the steps described above have only been tested on the identified platform(s). Opscode has not researched and does not support alternative steps that may lead to successful completion on other platforms. Platform(s) supported by this guide may change over time, so please do check back for updates. If you'd like to undertake this guide on an alternate platform, you may desire to turn to open source community resources for support assistance.*



If there are issues with the screencast above, it is also available at blip.tv.

Environment Setup

First, let's configure the local workstation.

Shell Environment

Obtain the repository used for this guide. It contains all the components required. Use git:

```
git clone git://github.com/opscode/nagios-quick-start.git
```

Chef and Knife

You will need some additional gems for Knife. Fog requires XML2 and XSLT development headers so you'll need to install them for your operating

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

system, for example on Debian and Ubuntu:

```
sudo apt-get install libxml2-dev libxslt-dev
```

All Users: You'll need some additional gems for Knife to launch instances in Amazon EC2:

```
sudo gem install knife-ec2
```

As part of the Fast Start Guide, you cloned a chef-repo and copied the Knife configuration file (knife.rb), validation certificate (ORGNAME-validator.pem) and user certificate (USERNAME.pem) to `~/chef-repo/.chef/`. Copy these files to the new rails-quick-start repository. You can also re-download the Knife configuration file for your organization with the Hosted Chef Management Console.

```
mkdir ~/nagios-quick-start/.chef
    cp ~/chef-repo/.chef/knife.rb ~/nagios-quick-start/.chef
    cp ~/chef-repo/.chef/USERNAME.pem ~/nagios-quick-start/.chef
    cp ~/chef-repo/.chef/ORGNAME-validator.pem ~/nagios-quick-start/.chef
```

Add the Amazon AWS credentials to the Knife configuration file.

```
vi ~/nagios-quick-start/.chef/knife.rb
```

Add the following two lines to the end:

```
knife[:aws_access_key_id] = "replace with the Amazon Access Key ID"
    knife[:aws_secret_access_key] =  "replace with the Amazon Secret Access Key ID"
```

Once the nagios-quick-start and knife configuration is in place, we'll work from this directory.

```
cd nagios-quick-start
```

Amazon AWS EC2

In addition to the credentials, two additional things need to be configured in the AWS account.

Configure the default security group to allow incoming connections for the following ports.

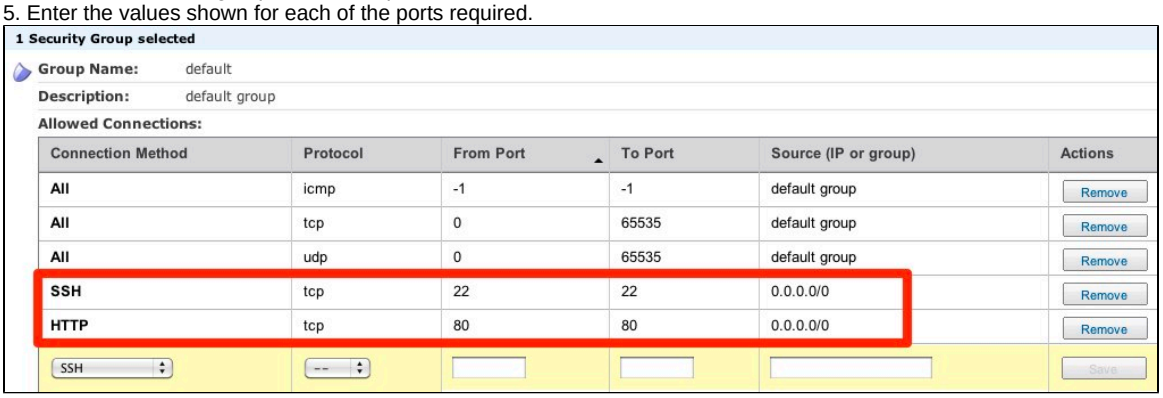
- 22 - SSH
- 80 - Nagios web interface via Apache

Add these to the default security group for the account using the AWS Console.

1. Sign into the Amazon AWS Console.
2. Click on the "Amazon EC2" tab at the top.
3. Click on "Security Groups" in the left sidebar of the AWS Console.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

4. Select the "Default" group in the main pane.
5. Enter the values shown for each of the ports required.



Create an SSH Key Pair and save the private key in `~/.ssh`.

1. In the AWS Console, click on "Key Pairs" in the left sidebar.
2. Click on "Create Keypair" at the top of the main pane.
3. Give the keypair a name like "nagios-quick-start".
4. The keypair will be downloaded automatically by the browser and saved to the default Downloads location.
5. Move the nagios-quick-start.pem file from the default Downloads location to `~/.ssh` and change permissions so that only you can read the file. For example,

```
mv ~/Downloads/nagios-quick-start.pem ~/.ssh
    chmod 600 ~/.ssh/nagios-quick-start.pem
```

Acquire Cookbooks

The nagios-quick-start repository has all the cookbooks we need for this guide. They were downloaded along with their dependencies from the cookbooks site using Knife. These are in the `cookbooks/` directory.

```
apache2
    apt
    nagios
```

Upload all the cookbooks to Hosted Chef.

```
knife cookbook upload -a
```

Server Roles

All the required roles have been created in the nagios-quick-start repository. They are in the `roles/` directory.

```
base.rb
    production.rb
    monitoring.rb
```

Upload all the roles to Hosted Chef.

```
rake roles
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Data Bag Item

The nagios-quick-start repository contains a data bag item that has information about a default user that can log into the Nagios web interface, **nagiosadmin**.

The data bag name is **users** and the item name is **nagiosadmin**. Upload this to Hosted Chef.

```
knife data bag create users
    knife data bag from file users nagiosadmin.json
```

Launch Single Instance

We are going to use an m1.small instance with the 32 bit Ubuntu 10.04 image provided by Canonical. The identifier is **ami-7000f019** for the AMI in us-east-1 with instance storage that we will use in this guide. We'll show you the `knife ec2 server create` sub-command to launch instances.

This command will:

- Launch a server on EC2.
- Connect it to Hosted Chef.
- Configure the system with Chef.

Launch the Nagios monitoring server on a single instance.

```
knife ec2 server create -G default -I ami-7000f019 -f m1.small \
    -S nagios-quick-start -i ~/.ssh/nagios-quick-start.pem -x ubuntu \
    -r 'role[production],role[base],role[monitoring]'
```

Once complete, the instance will be running Nagios.

Verification

Knife will output the fully qualified domain name of the instance when the command completes. You can navigate to the Nagios instance with:

```
http://ec2-xxx-xx-xx-xxx.compute-1.amazonaws.com/
```

The login is nagiosadmin and the password is nagios.

Adding Service Checks

New service checks can be added easily. Update the services.cfg.erb template. If necessary, update the commands.cfg.erb template for an additional command. Then upload the cookbook.

If the check is for all hosts, use **hostgroup_name all**.

```
vi cookbooks/nagios/templates/default/services.cfg.erb
    ...
    define service {
        service_description HTTP Processes
        hostgroup_name      webserver
        check_command       check_http
        use                 default-service
    }
```

If the check is for a certain role, such as **monitoring**, make sure it only gets enabled in the configuration if that role exists. For example:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
vi cookbooks/nagios/templates/default/services.cfg.erb
    ...
    <% unless @service_hosts['webserver'].nil? -%>
    define service {
        service_description HTTP Processes
        hostgroup_name      webserver
        check_command       check_http
        use                 default-service
    }

    <% end -%>
```

If the service check doesn't already exist in the commands.cfg.erb, add it.

```
vi cookbooks/nagios/templates/default/commands.cfg.erb
    ...
    define command {
      command_name    check_http
      command_line    $USER1$/check_http -I $HOSTADDRESS$ -H $HOSTADDRESS$
    }
```

Upload the Nagios cookbook and run chef on the monitoring node.

```
knife cookbook upload nagios
    knife ssh 'role:monitoring' 'sudo chef-client' -x ubuntu -i ~/.ssh/nagios-quick-start.pem
```

Refer to the Nagios Documentation for more information about writing Nagios service check definitions.

Adding NRPE Checks

To add a new NRPE check, create the entry in nrpe.cfg.erb. For example, to add a check for a process named "chef-client":

```
vi cookbooks/nagios/templates/default/nrpe.cfg.erb
    ...
    command[check_chef_client]=/usr/lib/nagios/plugins/check_procs -w 1:2 -c 1:2 -C chef-client
```

Then upload the cookbook and run chef on the client systems, and the plugin will be enabled via NRPE.

```
knife cookbook upload nagios
    knife ssh '*:*' 'sudo chef-client' -x ubuntu -i ~/.ssh/nagios-quick-start.pem
```

Refer to the Nagios Documentation for more information about NRPE.

Adding New Plugin Scripts

If you've found a cool Nagios plugin you'd like to use, you can distribute it to nodes with the cookbook files directory.

```
cp check_something_cool cookbooks/nagios/files/default/plugins
    knife cookbook upload nagios
    knife ssh '*:*' 'sudo chef-client' -x ubuntu -i ~/.ssh/nagios-quick-start.pem
```

Then update the commands.cfg.erb for the new command, and enable a service check by adding an entry in services.cfg.erb, per the sections above.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Refer to the Nagios Documentation for more information about Nagios Plugins.

Appendix

Database Passwords

The data bag item for dbapp contains default passwords that should certainly be changed to something stronger.

The passwords in the dbapp Data Bag Item are set to the values show below:

```
"mysql_root_password": {
      "_default": "mysql_root"
    },
    "mysql_debian_password": {
      "_default": "mysql_debian"
    },
    "mysql_repl_password": {
      "_default": "mysql_repl"
    },
```

To change the password to something stronger, modify **mysql_root**, **mysql_debian**, **mysql_repl** values. Something like the following secure passwords:

```
vi data_bags/apps/dbapp.json
    "mysql_root_password": {
      "_default": "super_s3cur3_r00t_pw"
    },
    "mysql_debian_password": {
      "_default": "super_s3cur3_d3b1@n_pw"
    },
    "mysql_repl_password": {
      "_default": "super_s3cur3_r3pl_pw"
    },
```

Once the entries are modified, simply load the data bag item from the json file:

```
knife data bag from file apps dbapp.json
```

Non-EC2 Systems

For people not using Amazon EC2, other Cloud computing providers can be used. Supported by knife and fog as of this revision:

- Rackspace Cloud

See the Launch Cloud Instances with Knife for more information about using Knife to launch these instance types.

For people not using cloud at all, but have their own infrastructure and hardware, use the Knife Bootstrap knife command. Note that the run-list specification is slightly different. For the first example of the single instance:

```
knife bootstrap IPADDRESS
     -r
'role[base],role[dbapp_database_master],role[dbapp],recipe[dbapp::db_bootstrap],role[dbapp_load_balan
cer]'
```

See the contextual help for knife bootstrap on the additional options to set for SSH.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
knife bootstrap --help
```

A Note about EC2 Instances

We used m1.small instances. This is a low performance instance size in EC2 and just fine for testing. Visit the Amazon AWS documentation to lea rn more about instance sizes.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Performance Tune Ohai for Large User Bases

Overview

Ohai detects information about your operating system, and can therefore benefit from fine-tuning to improve performance when running with large user bases.

Modify nodes.rb to reduce automatic_attrs size

```
# Serialize this object as a hash

    def to_json(*a)
      result = {
        "name" => name,
        "chef_environment" => chef_environment,
        'json_class' => self.class.name,
#        "automatic" => automatic_attrs,

        "normal" => normal_attrs,
        "chef_type" => "node",
        "default" => default_attrs,
        "override" => override_attrs,
        "run_list" => run_list.run_list
      }
      result["_rev"] = couchdb_rev if couchdb_rev
      result.to_json(*a)
end
```

Modify Ohai to read passwords directly

This will be addressed with OHAI-165. Pending that, the following change will be beneficial:

Add to client.rb

```
Ohai::Config[:disabled_plugins] = [ "passwd" ]
```

Update standard bootstrap template

Speeds up chef-client execution, because LDAP servers can return a huge number of passwords.

```
(
cat <<'EOP'
<%= config_content %>
EOP
) > /etc/chef/client.rb

echo "<%= 'Ohai::Config[:disabled_plugins] = [ \"passwd\" ]' %>" >>
/etc/chef/client.rb
```

Disable undesirable Ohai plugins

If you do not want some Ohai plugins to run, you can disable them by doing something like the following:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Modify etc/chef/client.rb

```
Ohai::Config[:disabled_plugins] << 'darwin::system_profiler' << 'darwin::kernel' <<
'darwin::ssh_host_key' << 'network_listeners'
```

You may desire to choose other plugins that the ones in the above example. **See Disabling Ohai Plugins for additional information**.

Thanks!
Thanks to Chef user Mike Hodgson for much of this information!

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Scalability and High Availability

Work in progress on this page.

Right now, this page is just a skeleton to gather information related to medium/large size Opscode Chef deployments. Topics such as Scalability and High Availability will be documented here.

At the moment, there are some threads in the Opscode mailing lists related to these topics:

High Availability for Chef

http://lists.opscode.com/sympa/arc/chef/2011-07/msg00010.html

Performance and Scalability

http://lists.opscode.com/sympa/arc/chef/2011-08/msg00000.html

http://lists.opscode.com/sympa/arc/chef/2012-01/msg00422.html

Links to relevant (Linux) software

Heartbeat
The Linux-HA project maintains a set of building blocks for high availability cluster systems

LVS
Highly scalable and highly available server built on a cluster of real servers, with the load balancer running on the Linux operating system

DRBD
DRBD can be understood as network based raid-1.

PaceMaker
Open Source, High Availability resource manager suitable for both small and large clusters.

Corosync
Implementing high availability within applications.

RabbitMQ HA
High availability with Pacemaker and DRBD

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Working with Git and Cookbooks



Cookbook Development Workflow for Chef Cookbooks

This page describes development workflow for Chef cookbooks using Git as the version control system.

It is not a comprehensive guide to Git, to learn more about Git itself:

- Git Documentation
- Learn Git
- Pro Git

This page will walk through five examples:

- Building a new cookbook and saving it to the local repository.
- Downloading an existing cookbook from the Chef Community Site.
- Modifying an existing cookbook in the local repository for new updates.
- Using Version Control to check the last change made to the local repository.
- Adding a remote repository url to the git directory so you can push the commits to your own remote git repo.

Pre-requisites

The user should already have cloned the skeleton Chef Repository from Opscode's GitHub repository.

git clone Opscode's chef-repo skeleton

```
% git clone git://github.com/opscode/chef-repo.git
Cloning into chef-repo...
remote: Counting objects: 173, done.
remote: Compressing objects: 100% (106/106), done.
remote: Total 173 (delta 62), reused 138 (delta 40)
Receiving objects: 100% (173/173), 27.25 KiB, done.
Resolving deltas: 100% (62/62), done.
```

The remaining examples will be done from the chef-repo directory.

Work from chef-repo

```
% cd chef-repo
% git status
# On branch master
nothing to commit (working directory clean)
```

As this is already a git repository, it's already set up for you to use with Git. However, the remote origin is still Opscode's repository. See the remote repositories section of this article for information on remote repositories in Git.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Git Community Book

This book has been built by dozens of people in the Git community, and is meant to help you learn how to use Git as quickly and easily as possible.



New Cookbook

A cookbook is a directory which contains the recipes and supporting code and assets. Create a new cookbook with knife.

knife cookbook create working_with_git

```
% knife cookbook create working_with_git
** Creating cookbook working_with_git
** Creating README for cookbook: working_with_git
** Creating metadata for cookbook: working_with_git
```

This will create a predefined directory structure of all the various components of a Cookbook.

Pre-filled contents are configurable
The contents of the README and metadata are configurable. See `knife cookbook create --help`.

As it has created files, the git working copy will have uncommitted changes.

Show that we have uncommitted changes

```
% git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       cookbooks/working_with_git/
nothing added to commit but untracked files present (use "git add" to track)
```

Here's what the cookbook directory contains:

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Contents of the cookbook's directory

```
% ls -l cookbooks/working_with_git
total 16
-rw-r--r--  1 jtimberman  staff   58 Mar 24 11:30 README.rdoc
drwxr-xr-x  2 jtimberman  staff   68 Mar 24 11:30 attributes/
drwxr-xr-x  2 jtimberman  staff   68 Mar 24 11:30 definitions/
drwxr-xr-x  3 jtimberman  staff  102 Mar 24 11:30 files/
drwxr-xr-x  2 jtimberman  staff   68 Mar 24 11:30 libraries/
-rw-r--r--  1 jtimberman  staff  262 Mar 24 11:30 metadata.rb
drwxr-xr-x  2 jtimberman  staff   68 Mar 24 11:30 providers/
drwxr-xr-x  3 jtimberman  staff  102 Mar 24 11:30 recipes/
drwxr-xr-x  2 jtimberman  staff   68 Mar 24 11:30 resources/
drwxr-xr-x  3 jtimberman  staff  102 Mar 24 11:30 templates/
```



Why all those directories?
Are all those directories required? No, they are not.

Why do we create all those directories? Simply because we don't know what you want in your cookbook. It is safe to leave them there, or delete the ones you don't need.

For this example, we'll just leave them there.

Next, let's edit the default recipe to do something.

vi cookbooks/working_with_git/recipes/default.rb

```
file "/tmp/working_with_git" do
  content "I'm working with git."
end
```

Now write the changes to the git repository. This is done in two steps. First, we add a list of files with git-add.

Tell git what files will be committed

```
% git add cookbooks/working_with_git
```

We can view the changes that will be written with git-status.

Show what will be committed

```
% git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   cookbooks/working_with_git/README.rdoc
#       new file:   cookbooks/working_with_git/metadata.rb
#       new file:   cookbooks/working_with_git/recipes/default.rb
#
```

Next use git-commit to write the changes to the repository.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Write the commit (changes) to the repository

```
% git commit -m 'create working with git cookbook'
[master 3a63b1e] create working with git cookbook
 3 files changed, 26 insertions(+), 0 deletions(-)
 create mode 100644 cookbooks/working_with_git/README.rdoc
 create mode 100644 cookbooks/working_with_git/metadata.rb
 create mode 100644 cookbooks/working_with_git/recipes/default.rb
```



Congratulations!

You've created a cookbook and written the changes to the local git repository.
To use the cookbook depends on whether you're using Chef Solo or Chef client with a server.

What is metadata.json?

If you are using a Chef Server, after doing the knife cookbook upload, you might notice there's a new file, metadata.json, in the cookbook.

Show generated metadata.json file

```
% git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       cookbooks/working_with_git/metadata.json
nothing added to commit but untracked files present (use "git add" to track)
```

This is the generated JSON file from the upload. This is dynamically generated, and if you modify the metadata.rb, it will change every time the cookbook is uploaded. You can add `metadata.json` to the `.gitignore` file to ignore these generated files, but make sure to always use the metadata.rb file for modifying the cookbook's metadata.

Use .gitignore to ignore metadata.json files

```
% vi .gitignore
metadata.json
% git add .gitignore
% git commit -m 'ignore generated metadata.json files'
[master 8d7d2b3] ignore generated metadata.json files
 1 files changed, 1 insertions(+), 1 deletions(-)
% git status
# On branch master
# Your branch is ahead of 'origin/master' by 2 commits.
#
nothing to commit (working directory clean)
```

Notice that we only added the `.gitignore` file, the metadata.json file is still there, Git just won't show it as uncommitted anymore.

Read more about cookbook Metadata. Read more about gitignore(5)

Download Cookbook

Opscode provides a centralized location for the Chef Community to share and download cookbooks. Knife can be used to retrieve cookbooks from the Community site and automatically integrate them into the local Git repository.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Make sure that the local repository has all changes committed:

Use Git Status to show clean repository

```
% git status
# On branch master
# Your branch is ahead of 'origin/master' by 2 commits.
#
nothing to commit (working directory clean)
```

For this example we're going to download the `chef-client` cookbook and integrate it into the local repository using the "vendor branch" pattern. Output of filenames in the cookbook are truncated; you can view the full output by running the same command.

knife cookbook site install chef-client

```
% knife cookbook site install chef-client
INFO: Downloading chef-client from the cookbooks site at version 0.99.3
INFO: Cookbook saved: /Users/jtimberman/Development/sandbox/chef-repo/cookbooks/chef-client.tar.gz
INFO: Checking out the master branch.
INFO: Checking the status of the vendor branch.
INFO: Creating vendor branch.
INFO: Removing pre-existing version.
INFO: Uncompressing chef-client version 0.99.3.
INFO: Adding changes.
INFO: Committing changes.
INFO: Creating tag chef-vendor-chef-client-0.99.3.
INFO: Checking out the master branch.
INFO: Merging changes from chef-client version 0.99.3.
Updating 8d7d2b3..28cdd4e
Fast-forward
 cookbooks/chef-client/README.md                    |  232 ++++++++++++++++++++
...TRUNCATED ...
 13 files changed, 658 insertions(+), 0 deletions(-)
 create mode 100644 cookbooks/chef-client/README.md
... TRUNCATED ...
INFO: Cookbook chef-client version 0.99.3 successfully vendored!
```

This command does the following:

- Download the cookbook tarball from cookbooks.opscode.com.
- Ensure its on the git master branch.
- Checks for an existing vendor branch, and creates if it doesn't.
- Checks out the vendor branch (chef-vendor-COOKBOOK).
- Removes the existing (old) version.
- Untars the cookbook tarball it downloaded in the first step.
- Adds the cookbook files to the git index and commits.
- Creates a tag for the version downloaded.
- Checks out the master branch again.
- Merges the cookbook into master.

By default, the cookbook's dependencies are also downloaded. The `-D` option can be used to prohibit dependency download. Learn more about cookbook dependencies.



Specify an "Integration" branch
An integration branch is the one that the cookbook gets merged into to integrate it into the repository. By default, this command uses "master" as the integration branch. If you have Chef 0.9.14 or higher, you can specify a different branch with ~~-B BRANCH or~~ -branch BRANCH.

We can now view some changes from this command in the repository. First, a new git-branch(1) is created:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

git branch

```
% git branch
  chef-vendor-chef-client
* master
```

The "*" in front of master indicates it is the currently checked out branch.

Also, a git-tag(1) is created:

git tag

```
% git tag
chef-vendor-chef-client-0.99.3
```

Modify Cookbook

After you have created a new cookbook, or installed a cookbook in the repository with the install command, you may want to make modifications. Edit the files as desired and commit your changes to the Git repository. The filenames and contents changed are not important. Assuming we've editted a recipe and saved changes to the file:

Edit files and save changes

```
% vi cookbooks/working_with_git/recipes/default.rb
```

git status to show uncommitted changes

```
% git status
# On branch master
# Your branch is ahead of 'origin/master' by 3 commits.
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   cookbooks/working_with_git/recipes/default.rb
#
no changes added to commit (use "git add" and/or "git commit -a")
```

git diff to see file content changes

```
% git diff cookbooks/working_with_git/recipes/default.rb
diff --git a/cookbooks/working_with_git/recipes/default.rb
b/cookbooks/working_with_git/recipes/default.rb
index f7ed0a0..534ce13 100644
--- a/cookbooks/working_with_git/recipes/default.rb
+++ b/cookbooks/working_with_git/recipes/default.rb
@@ -10,3 +10,7 @@
 file "/tmp/working_with_git" do
   content "I'm working with git."
 end
+
+file "/tmp/working_with_git_changes" do
+  mode 0755
+end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Add the file to the commit and create the commit.

git add, git commit

```
% git add cookbooks/working_with_git/recipes/default.rb
% git commit -m 'Add another file to the working_with_git recipe.'
[master 7a9aedd] Add another file to the working_with_git recipe.
 1 files changed, 4 insertions(+), 0 deletions(-)
```

Tracking Upstream Changes

One advantage of using the vendor branch pattern described above is it becomes easy to make local changes to a cookbook, and still be able to retrieve upstream improvements and other changes. Once your local changes are committed, you can download a new version again with the same command.

knife cookbook site install

```
% knife cookbook site install chef-client
INFO: Downloading chef-client from the cookbooks site at version 0.99.3
INFO: Cookbook saved: /Users/jtimberman/Development/sandbox/chef-repo/cookbooks/chef-client.tar.gz
INFO: Checking out the master branch.
INFO: Checking the status of the vendor branch.
INFO: Vendor branch found.
INFO: Removing pre-existing version.
INFO: Uncompressing chef-client version 0.99.3.
INFO: Adding changes.
INFO: Committing changes.
WARN: Checking out the master branch.
WARN: No changes from current vendor chef-client
```

Note: there weren't any updates, since we had just downloaded the cookbook of the same version already. If we did make changes **and** there were updates on the Community Site, then they would be merged in.

If there are any conflicts between local changes and the upstream's update, then we would handle those by editing the files, or modifying the local copy in the repository. Read more about branching and merging, "Resolving a Merge" section of the online Git Community Book.

Version Tracking

All git repos have version tracking, so even if you are just committing changes locally you can still go back and see what was changed.

You can use `git log` to see the log of commits. This command would show you the last 3 commits for example:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

git log

```
% git log -n3
commit 319af2c1d14252dda94b3d02259cd8b61c09d435
Author: user <user@email.com>
Date:   Tue Aug 23 14:02:27 2011 -0700

    adding another cookbook

commit 9adb6722384f71f043000ef60147e78c92260cd8
Author: user <user@email.com>
Date:   Mon Aug 22 16:17:13 2011 -0700

    adding in a test cookbook

commit b9c9306d90c394dda3198af7851fe4999bd79e70
Author: user <user@email.com>
Date:   Thu Jul 21 12:25:59 2011 -0700

    new repo testing
```

You could then see more information on a specific commit with `git show`. For example, to see more information on the last commit you could take the commit id and use it in this command:

git show

```
git show 319af2c1d14252dda94b3d02259cd8b61c09d435
```

For more information on the show and log commands, reference the git-show and git-log manual pages. The command `man gittutorial` can also give you further information on how these can be used.

Testing Cookbooks

Testing cookbooks is not directly related to Git, but depending on how you're using Chef determines what you need to do to test changes you've made to cookbooks.

If you're using Chef Solo

You'll need to copy the cookbooks you want to use to a system with Chef Solo, configured to use the cookbooks directory. You'll also need to pass a JSON file to `chef-solo` with the run_list of recipes to use. See the Chef Solo page for information about how to set up a system to run chef-solo.

If you're using Chef Client/Server

Upload the cookbook to the Chef Server with Knife:

knife cookbook upload

```
knife cookbook upload <cookbookname>
```

Then apply the recipe to a node or role and run Chef on the desired system.

Remote Repositories

By default, the remote origin is Opscode's repository. This means that you can make commits to the local repo for version tracking and other features, but not the remote repo so you can access externally.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

If you would like to add a remote origin for your personal git, you will first want to add a new repo to your git server. Once that is done you can add the remote origin with a command like this:

git remote

```
cd ~/chef-repo
git remote add github http://github.com/<username>/<reponame>.git
```

Then, whenever you want to push commits to your remote git you can do it with this command, after using `git add` and `git commit`:

git push to remote

```
git push github master
```

You can use any name in place of `github` in these examples, it is just a name for the remote connection. For more information on remote, read the git-remote manual page.

Here's an example of how your workflow may look, once this is all setup:

1. Create the cookbook directory skeleton with `knife cookbook create`.
2. Edit the files you want to modify in the new cookbook or an existing cookbook.
3. Upload the cookbook to the Chef Server with `knife cookbook upload`.
4. Test that the cookbook changes you made work by running chef-client.
5. Stage the files to commit to your local repository with `git add <filepattern>`.
6. Commit changes to the repository with `git commit`.
7. Push commits to the central git server repository with `git push <remotename> <branchname>`.

Keep in mind that due to how git actually works, we can't predict what the proper behavior for #5 and #6 will be.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Developers



It is truly a community effort

We released Chef on January 15th, 2009. As of November 21, 2011, more than 500 people and 100 different companies have signed up to contribute to Chef. Companies like Engine Yard and Right Scale have integrated it into their business offerings directly.

Many different cookbooks, automating everything from Apache web servers to Xen virtual machines have been published, and put to work in completely different environments.

When you use Chef, you join a community of professionals who have been there before, understand the problems we all face, and are working together to make things better for everyone. The results are far greater than any of us could have possibly achieved alone.

- Get Involved
- How to Contribute
- Approved Contributors
- Release Process
- Release MVPs
- Feature Proposals
- Chef Users List
- Chef Presentations
- AMI List for Developers

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Get Involved

Welcome

The first step in contributing to any Opscode-sponsored project is reviewing and completing the steps outlined on the How to Contribute page.

Next you should make sure you're subscribed to both the 'chef' and 'chef-dev' Mailing Lists which is where a lot of the community discussion happens.



w to help

There are many ways to help with the project.

- Assist other users on the Mailing Lists or on IRC.
- Scratch your own itch and add features
- Improve the code by adding unit tests to uncovered code
- Add wiki documentation, such as how to run and write unit tests
- Review [old] open tickets and ensure they are [still] reproducible
- ... and then provide a fix for that ticket
- Review github pull requests for projects that don't have an associated ticket number and find it, or ask the contributor for one.
- Add small or concise tasks to the list below to help people get started

Specific projects

These items could be completed by anyone. This list shouldn't be work you want someone to do for you, but rather small projects that you think someone new to the project could pick up and help with.

- CHEF-1771 adds support for configuring certain EC2 values to your knife configuration file. It may need to be moved to the KNIFE-EC2 project. Last we knew, it was breaking a number of spec tests and could use some work.
- Someone familiar with debian/ubuntu should try to figure out why the packaging for Chef still sometimes fails when rabbitmq is already configured. The success of this differs on releases, so even just testing a release and documenting what you tried and what the result was would help. Relevant tickets are CHEF-2472 and CHEF-2296. It looks like CHEF-2296 has a fix, it just needs to be written into http://github.com/opscode/opscode-packages.
- CHEF-1367 came up with ideas on how to make remote_file not download a file if it hadn't changed, but the original reporter never completed a CLA so we need to do this work ourselves. Tollef did something similar for the http_request resource in CHEF-1577 so it should be pretty easy to do here as well, although that might eventually mean upfactoring some shared code.
- The community drupal cookbook could stand to have the final installation steps automated with drush. More information in COOK-976.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

How to Contribute

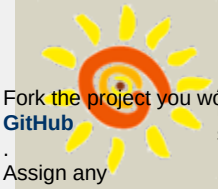
Thank you for contributing to an Opscode Open Source project!



One Time Action

1. Sign up for an account on the **Open Source Ticket Tracking System**.
    a. This will create an identical account for you on the Wiki.
    b. We match the email address from your CLA to this account.
2. Fill out a **Contributor License Agreement**. If your contribution is happening on behalf of a company, they should sign a **Corporate Contributor License Agreement**
    a. Please complete and digitally sign the form, the process and submittal fully managed for you through Echosign. *This will take 3-5 minutes.*
    b. You'll be emailed a copy when we've received it. We'll process it, and send you an introduction email.
    c. You'll then appear as a approved contributor, and be upgraded to a Developer on the Ticket Tracking System: allowing you to modify and resolve tickets.

Individual Contributions



We Welcome GitHub Pull Requests

A one-time CLA submittal - and relating pull requests to trouble ticket(s) - provides for managing submittals across the projects, for fulfilling Apache licensing, and resulting in your successful and appreciated contribution to the community.

1. Fork the project you would like to contribute to on **GitHub**.
2. Assign any ticket(s) you are actively working on to yourself.
3. Provide instructions **on the ticket** which GitHub repository and branch should be pulled from, and **set the ticket status to Resolved** when you are complete.
4. We'll test, verify and merge your changes and then close the ticket.

Opscode Sponsored Open Source Projects

Project	Ticket System	Github
Chef	http://tickets.opscode.com/browse/CHEF	http://github.com/opscode/chef
Ohai	http://tickets.opscode.com/browse/OHAI	http://github.com/opscode/ohai
Cookbooks	http://tickets.opscode.com/browse/COOK	http://github.com/opscode/cookbooks
Knife-EC2	http://tickets.opscode.com/browse/KNIFE_EC2	http://github.com/opscode/knife-ec2
Knife-Rackspace	http://tickets.opscode.com/browse/KNIFE_RACKSPACE	http://github.com/opscode/knife-rackspace
Knife-Windows	http://tickets.opscode.com/browse/KNIFE_WINDOWS	http://github.com/opscode/knife-windows
Knife-OpenStack	http://tickets.opscode.com/browse/KNIFE_OPENSTACK	http://github.com/opscode/knife-openstack

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

There are a number of other smaller projects, such as the mixlib libraries, that we sponsor as dependencies of Chef.
You can find this in the Opscode Github account.

FAQ

Licensing and copyright

FAQ Table of Contents

- Licensing and copyright
  - *Why is your software Apache Licensed?*
  - *How should I update the header file of my contribution to properly credit earlier contributors and recognize copyrights?*
  - *Do I need the agreement of anyone else whose authorship/copyright I come across?*
- CLAs and CCLAs
  - *Why do I need to fill out a CLA?*
  - *Can you accept my contribution without a CLA (I'll sign it later, I promise!)?*
  - *When do I need to have my company sign a CCLA?*
  - *How do I add additional employees to my existing CCLA?*
  - *How do I change the point of contact on my existing CCLA?*
- Contribution Process
  - *Do I have to use GitHub for my fork of the repository?*
  - *Why do I need to have a ticket associated with my patch?*
  - *You use GitHub, why don't you use GitHub Pull Requests?*

Why is your software Apache Licensed?

Opscode uses the Apache License Version 2 because it provides the same level of freedom for our users that we desire for ourselves. Based upon the Apache Licensing FAQ...

It allows you to:

- freely download and use Opscode software, in whole or in part, for personal, company internal, or commercial purposes;
- use Opscode software in packages or distributions that you create.

It forbids you to:

- redistribute any piece of Opscode-originated software without proper attribution;
- use any marks owned by Opscode in any way that might state or imply that Opscode endorses your distribution;
- use any marks owned by Opscode in any way that might state or imply that you created the Opscode software in question.

It requires you to:

- include a copy of the license in any redistribution you may make that includes Opscode software;
- provide clear attribution to Opscode for any distributions that include Opscode software.

It does not require you to:

- include the source of the Opscode software itself, or of any modifications you may have made to it, in any redistribution you may assemble that includes it;
- submit changes that you make to the software back to Opscode (though such feedback is encouraged).



From The Opscode Blog

Why We Chose the Apache License

It is our goal to run a successful, truly open source business.

To that end, we are protecting our own rights by making them explicit in our choice of licensing: you have the same rights to our open source software that we do.

How should I update the header file of my contribution to properly credit earlier contributors and recognize copyrights?

You need to put attribution about the origins of some of the code in the NOTICE file for your app. On an individual header, you can add yourself as an author/copyright holder, placing text in the header file saying that your new work is based on previous work, and referencing the original header below.

If the original file is lacking the license header, please let us know, so we can add one. Even in their absence, they are covered by the Apache 2

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

license.

Do I need the agreement of anyone else whose authorship/copyright I come across?

You can re-use the work without having to get the agreement of the original authors, as long as

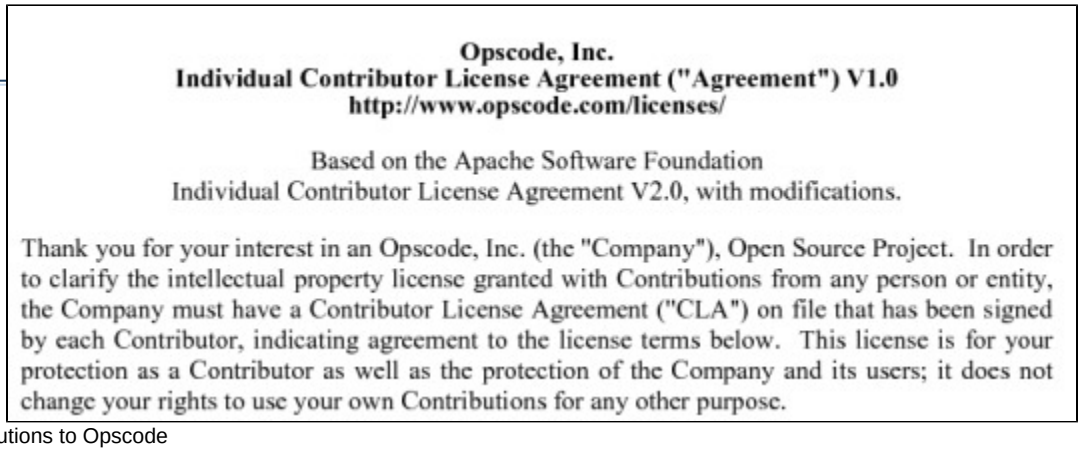
1. you're not going to be changing the licensing terms at all (ie: you maintain and fulfill Apache 2 licensing requirements) **and**
2. you don't modify the fact that the code you are incorporating remains copyrighted by the original authors.

The Apache License grants these rights to those who receive a copy of the software.

CLAs and CCLAs

Why do I need to fill out a CLA?

The CLA (and CCLA) makes everyone's rights clear. It states:



Opscode, Inc.
Individual Contributor License Agreement ("Agreement") V1.0
http://www.opscode.com/licenses/

Based on the Apache Software Foundation
Individual Contributor License Agreement V2.0, with modifications.

Thank you for your interest in an Opscode, Inc. (the "Company"), Open Source Project. In order to clarify the intellectual property license granted with Contributions from any person or entity, the Company must have a Contributor License Agreement ("CLA") on file that has been signed by each Contributor, indicating agreement to the license terms below. This license is for your protection as a Contributor as well as the protection of the Company and its users; it does not change your rights to use your own Contributions for any other purpose.

- You grant copyright license for your contributions to Opscode
- You grant patent license for your contributions to Opscode
- Your contribution is entirely voluntary
- Your work is your original creation
- You are not required to provide support for your contributions

You should read and understand the entire CLA before signing it. *Our description of it is not legally binding.*

The CLA is beneficial to our contributors and users because:

- It ensures that we will always be able to release our projects, free from any individual contributor revoking our rights to distribute their contribution.
  This also means that, if you fork an Opscode project, or utilize it in a commercial product, you know that you are clear of patent and copyright issues.
- It makes clear what is required of our contributors.

The most important thing about the CLA is that it doesn't give Opscode any special rights - it just makes things more explicit.

Can you accept my contribution without a CLA (I'll sign it later, I promise!)?

No. We must have a signed CLA before we can merge your changes to any of our projects.

When do I need to have my company sign a CCLA?

If you are contributing to an Opscode project while doing work on company time, or utilizing company resources, you should have your company sign a CCLA.

How do I add additional employees to my existing CCLA?

The point of contact from your CCLA should email legal@opscode.com with the full name and email address of the individuals account on tickets.opscode.com that they would like added.

How do I change the point of contact on my existing CCLA?

If you need to update the point of contact for the CCLA, the current contact should email legal@opscode.com with the full name and email address of the new contact. If the existing contact is no longer available, please contact us and we will help you out.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Contribution Process



Do I have to use GitHub for my fork of the repository?

Nope - but it makes everyone's lives easier if you do. We'll be happy to pull from any repository you like.

Why do I need to have a ticket associated with my patch?

Because it helps us coordinate the changelog for future releases, and provides for recognizing community participation.

You use GitHub, why don't you use GitHub Pull Requests?

We use JIRA for our open source projects and for our own internal projects: managing all the details of the code base, and fulfilling license requirements for our joint benefit. We also interact regularly with the GitHub issues/pull request system. **You are welcome to open a pull request, and link to that when updating the Jira ticket for a contribution.**

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Apache License



The canonical License
You can get the canonical License at http://apache.org/licenses/LICENSE-2.0.html

Apache License, Version 2.0

Apache License
Version 2.0, January 2004 http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

Table of Contents

- 1. Definitions.
- 2. Grant of Copyright License.
- 3. Grant of Patent License.
- 4. Redistribution.
- 5. Submission of Contributions.
- 6. Trademarks.
- 7. Disclaimer of Warranty.
- 8. Limitation of Liability.
- 9. Accepting Warranty or Additional Liability.

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

1. You must give any other recipients of the Work or Derivative Works a copy of this License; and

2. You must cause any modified files to carry prominent notices stating that You changed the files; and

3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

4. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability.

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

9. Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner] Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Contributor License Agreement

This page formerly held the Contributor License Agreement (CLA).

You can now complete a CLA electronically from the How to Contribute page.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Corporate Contributor License Agreement

This page formerly held the Corporate Contributor License Agreement (CCLA).

You can now complete a CCLA electronically from the How to Contribute page.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Working with Git

Chef Project Development
This page describes the workflow used for working on the Chef and related projects (Ohai, mixlibs, etc) source code. To learn more about Git itself:

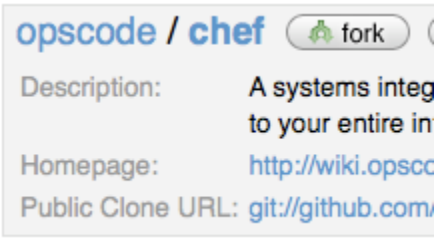
- Git Documentation
- Learn Git
- Pro Git

To learn about working with git and cookbooks:

- Working with Git and Cookbooks

Initial setup of development repository

1. Setup a github account
2. Fork the repositories



3. Clone the repositories locally

```
$ git clone git@github.com:yourgithubusername/chef.git
```

4. Enter the chef directory and add an opscode remote

```
$ cd chef/
$ git remote add opscode git://github.com/opscode/chef.git
```

You'll be able to see if this is successful with git config:

```
$ git config --get-regexp "^remote\.opscode"
remote.opscode.url git://github.com/opscode/chef.git
remote.opscode.fetch +refs/heads/*:refs/remotes/opscode/*
```

5. Adjust your branch to track the opscode master remote branch, by default it'll track your origin remote's master:

```
$ git config --get-regexp "^branch\.master"
branch.master.remote origin
branch.master.merge refs/heads/master
```

Change it with the following:

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
$ git config branch.master.remote opscode
```

Keeping your 'master' up-to-date!

Once all this is done, you'll be able to keep your local master up to date with the simple command:

```
$ git checkout master
$ git pull --rebase
```

Alternatively, you can synchronise your master from any branch with the full fetch/rebase syntax:

```
$ git fetch opscode
$ git rebase opscode/master master
```

Using rebase pull will do a rebase instead of a merge, which will keep a linear history with no unecessary merge commits. It'll also rewind, apply and then reapply your commits at the HEAD.

Use this Rakefile to update chef, ohai and cookbooks repos (edit as needed).

Working on topic branches

So you want to do some work? *Don't put your commits directly in your master branch!* It is important to use a 'topic branch' when working on a large project like Chef. The key to this concept is that each topic branch solves a single and unique problem and should usually be logically organized in the same was as a ticket on the issue tracker. A good example is a branch that adds support for a new init system, or resolves a bug when running under a specific version of CentOS. We prefer that topic branches be named after the bug that they solve so that someone with the same issue can easily find your commits in the 'git log'.

If your topic branch solves multiple bugs, reconsider if your branch is perhaps too broad. What if the person merging the contribution finds an issue with part of the branch but not another? Sometimes when you refactor a large piece of the code-base, you resolve multiple bugs and it is better to put individual issue numbers in the commit messages. One solution doesn't fit all, so use your best judgement.

1. Create an appropriately named tracking branch!

   ```
   $ git checkout --track -b CHEF-XX opscode/master
   ```

   Setting a topic branch up to track opscode/master allows you to easily rebase your commits in preperation for merge.

2. Do work

   ```
   hack
   hack
   ```

3. Commit (see step two if more work remains)

   ```
   $ git status
   $ git commit <filespec>
   ```

4. Rebase your commits against opscode/master
   After your work is finished in the local topic branch, you should rebase you commits against the upstream master. This will temporarily remove your local commits, update the branch from upstream, and then reapply your local commits. You can either do this manually with 'fetch' then 'rebase', or use the 'pull --rebase' shortcut.

   If there are any problems doing so, git will let you know and stop. This is important because it ensures that those that will merge your contribution into the upstream master won't have to resolve differences between your changes and the current branch to include your

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

contribution. If you encounter merge conflicts, you should fix the files as directed and then mark as fixed with 'git add', and then continue rebasing with 'git rebase --continue'. At any stage, you can abort the rebase with 'git rebase --abort'.

Rebase your commits with fetch + rebase

```
$ git fetch opscode
$ git rebase opscode/master CHEF-XX
```

Rebase your commits with the tracking-branch shortcuts

```
$ git pull --rebase
```

5. Push a remote branch

```
$ git push origin CHEF-XX
```

Job's done!

Don't forget to send a pull request and update the ticket. Once your work has been merged by the branch maintainer, it will no longer be necessary to keep the local branch or remote branch, so you can remove them!

1. Sync your local master up:

```
$ git checkout master
$ git pull --rebase
```

Remove your local branch using -d to ensure that it has been merged by upstream. Branch -d will not delete a branch that is not an ancestor of your current head. From the git man page:

```
-d

    Delete a branch. The branch must be fully merged in HEAD.
-D

    Delete a branch irrespective of its merged status.
```

2. Remove your local branch

```
$ git branch -d CHEF-XX
```

3. Remove your remote branch by using the full syntax to 'push', and omitting a source branch.

```
$ git push origin :CHEF-XX
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Approved Contributors

Allowed Contributors

The list of allowed contributors to Opscode projects. Persons listed as associated with a company may also be individual contributors as well.

To get on the list, check out our instructions on how to contribute.

Number	Contributor	Company	Date
1	Adam Jacob	Opscode	
2	Andy Delcambre	Engineyard	1/7/09
3	Arjuna Christensen		1/7/09
4	Artur Bergman	Wikia	
5	Benjamin Black	Opscode	1/10/09
6	Bryan McLellan		1/7/09
7	Dan Walters		3/7/09
8	Edward Muller	Engineyard	1/28/09
9	Ezra Zygmuntowicz	Engineyard	1/7/09
10	Jason Cook	Wikia	
11	Joe Williams		1/17/09
12	Kris Rasmussen	Aptana	2/12/09
13	Lee Jensen	Engineyard	1/24/09
14	Nick Sullivan	Wikia	
15	Paul Nasrat		1/19/09
16	Pawel Rein	Wikia	
17	Przemek Malkowski	Wikia	
18	Sean Cribbs		2/5/09
19	Steve Berryman		1/20/09
20	Steven Parkes	Aptana	2/12/09
21	Thom May		1/21/09
22	Tim Dysinger		1/28/09
23	Michael Hale		2/16/09
24	Mathieu Sauve-Frankel		2/22/09
25	Matthew Landauer		2/25/09
26	John Hampton	CleanOffer	3/2/09

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

27	Nadeem Bitar	CleanOffer	3/2/09
28	James Gartrell		2/6/09
29	Joshua Sierles	37signals	3/4/09
30	Mark Imbriaco	37signals	3/4/09
31	Stephen Haynes	Nomitor	3/9/09
32	Yun Huang Yong	Nomitor	3/9/09
33	David Lee		3/9/09
34	Matthew Kent		3/24/09
35	Dave Myron		4/3/09
36	Miguel Cabeça		8/4/09
37	Jason Jackson		4/9/09
38	Caleb Tennis		4/10/09
39	Michael Lim		4/14/09
40	David Balatero		4/28/09
41	David Grandinetti	We Go To 12	4/30/09
42	Lachlan Cox	Plus2 Pty	5/8/09
43	Scott Likens		4/30/09
44	Andrew Willis		5/25/09
45	Hongli Lai	Phusion	6/22/09
46	Ninh Bui	Phusion	6/22/09
47	Edmund Haselwanter		6/25/09
48	Raphael Simon	RightScale	6/30/09
49	Tony Spataro	RightScale	6/30/09
50	Stéphane Crivisier		6/30/09
51	Matthew Todd	Highgroove Studios	7/1/09
52	Grant Zanetti		2/1/09
53	Peter Woodman		6/22/09
54	Daniel DeLeo		7/10/09
55	Jeppe Madsen		9/13/09
56	Cary Penniman	RightScale	7/20/09
57	J. Chris Anderson		8/7/09
58	Graeme Mathieson	Rubaidh	8/10/09
59	Mark Connell	Rubaidh	8/10/09
60	Jonathan Weiss	Peritor GmbH	8/12/09
61	Mathias Meyer	Peritor GmbH	8/12/09

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

62	Pedro Belo	Heroku	8/13/09
63	Ricardo Chimal Jr.	Heroku	8/13/09
64	Adam Wiggins	Heroku	8/13/09
65	Ryan Tomayko	Heroku	8/13/09
66	Blake Mizerany	Heroku	8/13/09
67	Diego Algorta		
68	Kevin Hunt		8/14/09
69	Sidney Burks		8/20/09
70	Joe Van Dyk		9/1/09
71	Sig Lange		9/1/09
72	Alexander van Zoest		9/1/09
73	Nathan Mueller		9/7/09
74	Roman Heinrich		9/11/09
75	Gábor Vészi		9/20/09
76	Kenneth Kalmer	Internet Exchange	9/22/09
77	Luca Greco		9/22/09
78	Charles Cook	Betfair	9/30/09
79	Mario Giammarco		10/6/09
80	Matthew King		10/14/09
81	James Golick		10/27/09
82	Jörn Berrisch		10/28/09
83	Peter Crossley		10/30/09
84	Eric Hankins	Sojern	11/2/09
85	David McRae	Sojern	11/2/09
86	Dan Fitch	Sojern	11/2/09
87	Ian Meyer		11/8/09
88	John Alberts		11/9/09
89	Lee Marlow		11/11/09
90	Tollef Fog Heen		11/12/09
91	Cuong Chi Nghiem		11/13/09
92	Gordon Thiesfeld		11/18/09
93	Dreamcat4		11/21/09
94	Guy Bolton King		12/3/09
95	Robert Berger	Runa	12/20/09
96	Siva Jagadeesan	Runa	12/20/09

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

97	Ivan Pirlik		12/22/09
98	David Abdemoulaie		12/23/09
99	Alex Soto		12/29/09
100	Bryan Helmkamp		12/30/09
101	Jesse Nelson		1/6/10
102	Seth Chisamore	Opscode	1/11/10
103	Alfredo Deza	MaxMedia	1/11/10
104	N. Alan Johnson Jr.		1/15/10
105	Pavel Valodzka		2/8/10
106	Kyle Maxwell	Quantifind	2/11/10
107	Doug MacEachern	VMware	2/11/10
108	Jan Zimmek		2/14/10
109	Dan Prince	Rackspace	2/26/10
110	Gabe Westmaas	Rackspace	2/26/10
111	Tim Harper		3/8/10
112	Renaud Chaput		3/10/10
113	Daniel Peterson		3/11/10
114	Amit Cohen	Leaway Enterprise	3/16/10
115	Avishai Ish-Shalom	Leaway Enterprise	3/16/10
116	Or Cohen	Leaway Enterprise	3/16/10
117	Jon Swope		3/19/10
118	Jonathan Tron		3/19/10
119	Christopher Peplin	Bueda	3/30/10
120	Trotter Cashion		4/3/10
121	Benjamin Standefer		4/6/10
122	P. Barrett Little		4/7/10
123	John Nixon		4/13/10
124	Bruce Krysiak		4/13/10
125	Akzhan Abdulin		4/14/10
126	Grant Rodgers		4/17/10
127	Wesley Beary		4/22/10
128	Farzad Farid		4/23/10
129	Olivier Raginel		4/26/10
130	Jacques Crocker		5/3/10
131	Pierre Baillet		5/3/10

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

132	Joel Merrick		5/3/10
133	James Sanders		5/3/10
134	John Goulah		5/3/10
135	Toomas Pelberg		5/3/10
136	Ceaser Larry	Divergent Logic	5/3/10
137	Justin Sheehy	Basho Technologies	5/4/10
138	Andrew Gross	Basho Technologies	5/4/10
139	Bryan Fink	Basho Technologies	5/4/10
140	Ben Mabey		5/5/10
141	Christopher Durtschi	Divergent Logic	5/7/10
142	Kevin Carter	Divergent Logic	5/7/10
143	Saimon Moore		5/13/10
144	Troy Davis	Seven Scale	5/13/10
145	Eric Lindvall	Seven Scale	5/13/10
146	Alexey Ivanov		5/14/10
147	Pritesh Mehta		5/19/10
148	Ondrej Kudlik	IglooNET	5/21/10
149	Marek Hulan	IglooNET	5/21/10
150	Chad Woolley		5/22/10
151	Jochen Lillich	Freistil Consulting	5/25/10
152	Marius Ducea	Promet Solutions	5/25/10
153	Eric Butler		5/26/10
154	Sahil Cooner		6/6/10
155	Richard Nicholas	Betfair	6/9/10
156	Dan Slimmon		6/10/10
157	Craig Webster	Picklive	6/16/10
158	Dean Strelau	Mint Digital	6/17/10
159	Kurt Yoder		6/25/10
160	Jim Browne	42 Lines	6/27/10
161	Andrey Sibiryov		7/7/10
162	Anthony Newman	Betfair	7/8/10
163	Thomas Hoover		7/8/10
164	Dylan Egan	Wildfire Interactive	7/9/10
165	Michael Carruthers	Wildfire Interactive	7/11/10
166	Jon Seaberg	RightScale	7/20/10

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

167	Sean O'Meara		7/20/10
168	Lisa Hagemann	Dynamic Network Services	7/21/10
169	Cory von Wallenstein	Dynamic Network Services	7/21/10
170	Michael Leinartas		7/22/10
171	Thomas Bishop		7/23/10
172	Jon Wood		7/29/10
173	Dmitry Vyal		8/4/10
174	Gilles Devaux	PeerPong	8/4/10
175	Chris Pepper		8/5/10
176	Dennis Klein		8/6/10
177	Warwick Poole		8/12/10
178	Ken Ming Ong		8/15/10
179	Ash Berlin		8/16/10
180	Jochen Tuchbreiter	domainfactory GmbH	8/16/10
181	Mat Ellis	Tecnh	8/17/10
182	Michael MacDonald		8/17/10
183	Jorge Luiz deBrito Falcão		8/18/10
184	Jamie Winsor		8/19/10
185	Darrin Eden		8/19/10
186	Jonathan Smith		8/19/10
187	Andrew Fulcher		8/23/10
188	Matthias Marschall		8/25/10
189	Peter Struijk		8/25/10
190	Robert Anthony Postill		8/28/10
191	Joshua Timberman	Opscode	9/6/10
192	Benjamin Rockwood		9/6/10
193	Douglas Knight		9/9/10
194	Andrew Cole	9Summer	9/9/10
195	Dimitri Krassovski	Wixpress	9/13/10
196	Gregory Man	Wixpress	9/13/10
197	Allan Feid		9/17/10
198	Ringo De Smet		9/26/10
199	Tomasz Napierala		9/27/10
200	Jesse Proudman	Blue Box Group	9/29/10
201	Ian Parades	Blue Box Group	9/29/10

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

202	Lee Huffman	Blue Box Group	9/29/10
203	Christopher Horton		10/1/10
204	Jude Sutton	FindsYou Limited	10/6/10
205	James Le Cuirot	FindsYou Limited	10/6/10
206	Richard Pelavin		10/7/10
207	Blake Irvin		10/8/10
208	Jim Van Fleet		10/14/10
209	Laurent Désarmes		10/14/10
210	Jay T. McCanta		10/15/10
211	Eric G. Wolfe		10/20/10
212	Sami Haahtinen		10/21/10
213	Chris Kelly	Highgroove Studios	10/25/10
214	Gerald L. Hevener Jr.		10/25/10
215	Charles Quinn	Highgroove Studios	10/25/10
216	Jonathan Wallace	Highgroove Studios	10/25/10
217	Jason Ardell		10/26/10
218	Sean Carey		10/27/10
219	Pierre-Luc Brunet	ZeStuff	10/28/10
220	Sean Walbran		10/28/10
221	Brian McKelvey	Worlize	10/28/10
222	Jeffrey Hulten	Automated Labs	11/3/10
223	Doug Cole	Estately	11/4/10
224	Ben Bleything	Estately	11/4/10
225	Sebastian Boehm		11/6/10
226	Ches Martin		11/8/10
227	Eric C. Herot		11/8/10
228	Oliver Hankeln		11/10/10
229	David Nolan	Kapoq	11/10/10
230	Frank Louwers	Openminds	11/10/10
231	Bernard Grymonpon	Openminds	11/10/10
232	Bram Gillemon	Openminds	11/10/10
233	Paul Cortens	MobileCause	11/10/10
234	Austin Schneider	MobileCause	11/10/10
235	Kevin Ahrens		11/13/10
236	Stephen Nelson-Smith	Atalanta Systems	11/14/10

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

237	Caleb Groom		11/16/10
238	Michael Ivey	Menue Americas Corporation	11/17/10
239	Wojciech Wnetrzak		11/20/10
240	Dmitriy Tkachenko		11/22/10
241	Filip Tepper		11/24/10
242	Denis Barushev		11/27/10
243	Pedro F. <> Horrillo Guerra		11/28/10
244	James Harton	Sociable Limited	12/1/10
245	Noah Kantrowitz		12/4/10
246	Anthony Burton		12/4/10
247	David Esposito	Nine Summer	12/6/10
248	Mike Lecza	Nine Summer	12/6/10
249	Andrew Cole	Nine Summer	12/6/10
250	Michael Winser	Nine Summer	12/6/10
251	Cory Burke	Nine Summer	12/6/10
252	Charles Duffy	Tippr	12/7/10
253	John Vincent		12/10/10
254	Dustin Currie		12/14/10
255	Mark Sonnabaum		12/14/10
256	Elliot Murphy		12/22/10
257	Laradji Nacer		12/30/10
258	Todd Nine		1/3/11
259	Elijah Wright		1/5/11
260	Anshul Khandelwal		1/13/11
261	Michael Carruthers		1/16/11
262	Vishvananda Ishaya		1/17/11
263	Scott Frazer		1/17/11
264	Eric Hodel		1/21/11
265	Eric Heydrick		1/25/11
266	Andreas Kollegger	Neo Technology	1/27/11
267	Steve Lum		1/31/11
268	Anthony Goddard		2/1/11
269	Roland Moriz	Moriz GmbH	2/2/11
270	Emil Sit	VMWare	2/2/11
271	Ranjib Dey		2/3/11

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

272	Ryan Davis		2/8/11
273	Eric Coleman		2/8/11
274	James Casey		2/9/11
275	Maciej Pasternacki		2/9/11
276	Grzegorz Marszalek		2/11/11
277	James Sulinski	AegisCo	2/14/11
278	Erik Sabowski	AegisCo	2/14/11
279	Maciej Pasternacki	SetJam	2/15/11
280	Steven Dossett	Ning	2/17/11
281	Dane Knecht	Tippr	2/18/11
282	Mark Imbriaco	Heroku	2/28/11
283	Jonathan Matthews	7digital	3/3/11
284	Patrick Collins		3/7/11
285	Jim Hopp	Workday	3/12/11
286	Ken Dove	Workday	3/12/11
287	Philip Reynolds	Workday	3/12/11
288	Victor Zakharyev	Workday	3/12/11
289	Greg Fuller	Workday	3/12/11
290	Michael Callahan	Workday	3/15/11
291	Don Norton	Workday	3/15/11
292	Joe Nuspl	Workday	3/15/11
293	Dan Thom	Workday	3/15/11
294	Rick Cooper	Workday	3/15/11
295	Patrick Debois		3/14/11
296	Michael Guterl		3/15/11
297	Andrew Miklas	PagerDuty	3/17/11
298	Tristan Sloughter		3/22/11
299	Jonathon Ramsey		3/23/11
300	Jesai Langenbach		3/24/11
301	Maciej Pasternacki	Gnowsis	3/25/11
302	Omri Cohen		3/28/11
303	Joseph Sokol-Margolis		3/31/11
304	Alex Tomlins	Unboxed Consulting	4/1/11
305	Holger Just		4/5/11
306	Jake Vanderdray	CustomInk	4/8/11

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

307	Nathen Harvey	CustomInk	4/8/11
308	Padraig O'Sullivan		4/8/11
309	Christian Trabold		4/16/11
310	KC Braunschweig	Edmunds.com	4/19/11
311	Josh Pasqualetto		4/21/11
312	Christopher C. Johnson		4/21/11
313	Christian Paredes		4/22/11
314	Viral Shah		4/24/11
315	Bradley Fritz		4/24/11
316	Nat Lownes		4/25/11
317	Jonathan Tron	TalentBox	4/25/11
318	Joseph Halter	TalentBox	4/25/11
319	Wilson Felipe Nunes Fernandes Pereira		4/27/11
320	Michael Grubb		4/27/11
321	Brandon Konkle		4/28/11
322	Matt Griffin		4/28/11
323	Anh K. Huynh		4/28/11
324	Erik Frey	Wavii	4/29/11
325	Spike Gronim	Wavii	4/29/11
326	Ian MacLeod	Wavii	4/29/11
326	Guido Bartolucci	Wavii	4/29/11
327	Fletcher Nichol		5/3/11
328	James Kane	7digital	5/4/11
329	Paul Richards	7digital	5/4/11
330	Alexis Le-Quoc	Datadog	5/4/11
331	Matthew Singleton	Datadog	5/4/11
332	Carlo Cabanilla	Datadog	5/4/11
333	Olivier Pomel	Datadog	5/4/11
334	Fabrice Ollivier	Datadog	5/4/11
335	Matt Griffin	Viximo	5/10/11
336	Chris Chiodo	Viximo	5/10/11
337	Adam Bell	Viximo	5/10/11
338	Sergio Rubio		5/10/11
339	Elmer Rivera		5/10/11
340	Andrea Campi	ZephirWorks	5/11/11

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

341	Andrea Carlo Granata	ZephirWorks	5/11/11
342	Marco Pierleoni	ZephirWorks	5/11/11
343	Pietro Giorgianni	ZephirWorks	5/11/11
344	Jesse Newland		5/11/11
345	Greg Swallow		5/12/11
346	Scott Jensen	Dell	5/12/11
347	Greg Althaus	Dell	5/12/11
348	Andi Abes	Dell	5/12/11
349	Rob Hirschfeld	Dell	5/12/11
350	Paul Webster	Dell	5/12/11
351	Mitchell Hashimoto		5/12/11
352	Jamie van Dyke		5/17/11
353	Vladimir Kozhukalov		5/18/11
354	Nathan Butler	Newsweek/Daily Beast Company	5/19/11
355	Ken Garland	Newsweek/Daily Beast Company	5/19/11
356	Matt Whiteley	WordStream	5/19/11
357	Michael Yankovski	WordStream	5/19/11
358	Augusto Becciu		5/19/11
359	Greg Albrecht		5/19/11
360	Eric James Buth		5/24/11
361	Dan Porter		5/24/11
362	Adrian Silva	Atalanta Systems	5/25/11
363	Spike Morelli	Atalanta Systems	5/25/11
364	Paul Nicholson		5/27/11
365	Mandi Walls		5/27/11
366	Carl Perry	DreamHost	5/29/11
367	Greg Thornton		5/30/11
368	Joseph Heck		6/1/11
369	Chalres Ray Johnson, Jr.		6/2/11
370	Joseph Anthony Pasquale Holsten		6/5/11
371	John Donagher		6/6/11
372	David Fuhr	Flagbit	6/14/11
373	Jörg Weller	Flagbit	6/14/11
374	Marcel Cary		6/15/11
374	Yedidya "Jay" Feldblum	Applications Online	6/17/11

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

375	Michael Contendo		6/20/11
376	Yogesh Pathade		6/22/11
377	Gavin Sandie		6/25/11
378	Bryan Horstmann-Allen		6/28/11
379	Glenn Pratt		7/5/11
380	Andrew Narkewicz	Versapay	7/5/11
381	Zachary Tomas Stevens		7/11/11
382	Richard Gould		7/11/11
383	Philip Cohen		7/15/11
384	Christopher Michael McClimans		7/18/11
385	Jason J.W. Williams		7/19/11
386	Nuo Yan		7/21/11
387	Ryan Holmes	Edmunds.com	7/22/11
388	Alan Willis	Edmunds.com	7/22/11
389	Jaroslaw miejczak		7/25/11
390	Dimitri Aivaliotis	Every Ware	7/25/11
391	Eric Rochester		7/27/11
392	Alex North-Keys	Tippr	7/29/11
393	Adam Knight	Tippr	7/29/11
394	Eugene Wood	Ask.com	8/1/11
395	Aron Bartling	Ask.com	8/1/11
396	Jack Francis	Ask.com	8/1/11
397	Richard Marshall	Ask.com	8/1/11
398	Mikola Kucharski	Ask.com	8/1/11
400	Rory Mitchell	Ask.com	8/3/11
401	Oskar Stolc	Ask.com	8/3/11
402	Jack (John) Roehrig	Ask.com	8/3/11
403	Paul Stahlke	Ask.com	8/3/11
404	Jorge Mazzei	Ask.com	8/3/11
405	Pakojo Samm	Ask.com	8/3/11
406	David Smith	Ask.com	8/3/11
407	Mike Adolphs		8/5/11
408	Ernad Husremovi	bring.out doo Sarajevo	8/11/11
409	Jasmin Beganovi	bring.out doo Sarajevo	8/11/11
410	Saša Vrani	bring.out doo Sarajevo	8/11/11

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

411	Šator Emir	bring.out doo Sarajevo	8/11/11
412	Jeremy Bingham	Kos Media	8/15/11
413	Michael Taras	Kos Media	8/15/11
414	Tomoyuki Sakurai	reallyenglish.com	8/15/11
415	Mitsuru Yoshida	reallyenglish.com	8/15/11
416	Avishai Ish-Shalom	fewbytes	8/18/11
417	Or Cohen	fewbytes	8/18/11
418	Domenico Delle Side		8/19/11
419	Paul Morton	Business Intelligence Associates	8/19/11
420	Phil Austin	Business Intelligence Associates	8/19/11
421	Andrian Jardan	Tacit Knowledge	8/22/11
422	Vladimir Girnet	Tacit Knowledge	8/22/11
423	Scott Askew	Tacit Knowledge	8/22/11
424	Emmett Finneran		8/26/11
425	Arthur Gautier	Zenexity	8/30/11
426	Edward Middleton	ClassDo	8/30/11
427	Michael Pearson		8/31/11
428	Nikolay Sturm		9/1/11
429	Nathan Lloyd Smith		9/1/11
430	Patrick Connolly	Myplanet	9/2/11
431	Yashar Rassoulli	Myplanet	9/2/11
432	James Walker	Myplanet	9/2/11
433	Chris Read		9/6/11
434	Prashant Srivastava		9/7/11
435	Brad Knowles	ihiji	9/16/11
436	Stuart Rench	ihiji	9/16/11
437	Michael Maniscalco	ihiji	9/16/11
438	Brian Cunnie		9/19/11
439	Joseph F. Reynolds		9/20/11
440	Gabriel McArthur		9/21/11
441	David Keith Hudgins		9/21/11
442	Paul MacDougall		9/23/11
443	Bulat Shakirzyanov		9/23/11
444	Jorge Eduardo Espada		9/28/11
445	Eric Dennis		9/28/11

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

446	Stuart Glenn		9/29/11
447	Bryan Wilson Berry		10/4/11
448	Christopher Sturm		10/5/11
449	John Sumsion		10/12/11
450	Steven Phung		10/13/11
451	Claudio Cesar Sanchez Tejeda		10/14/11
452	Igor Afonov		10/26/11
453	Dan Buettner	Port 80 Production, LLC	10/28/11
454	Robby Grossman		10/31/11
455	Alan Harper		11/1/11
456	Juanje Ojeda		11/1/11
457	Stephane Jourdan	Green Alto	11/2/11
458	Gregory Karekinian	Green Alto	11/2/11
459	Samuel Maftoul	Green Alto	11/2/11
460	Darrin Eden	Heavy Water Software, Inc.	11/4/11
461	Sean Escriva	Heavy Water Software, Inc.	11/4/11
462	Jake Davis	Heavy Water Software, Inc.	11/4/11
463	AJ Christensen	Heavy Water Software, Inc.	11/4/11
464	Michael Weinberg	Heavy Water Software, Inc.	11/4/11
465	Aaron Baer	Heavy Water Software, Inc.	11/4/11
466	Matthew Kanwisher		11/8/11
467	Joshua McKenty		11/9/11
468	Iulian-Corneliu Costan		11/11/11
469	Daniel Oliver		11/14/11
470	Adam Garside		11/15/11
471	Ian Wolfcat Atha	Wealthfront Inc.	11/15/11
472	John Hitchings	Wealthfront Inc.	11/15/11
473	David Fortunato	Wealthfront Inc.	11/15/11
474	Julien Wetterwald	Wealthfront Inc.	11/15/11
475	Kevin Peterson	Wealthfront Inc.	11/15/11
476	Maksim Horbul		11/16/11
477	Roberto Gaiser		11/16/11
478	Robert Di Marco		11/17/11
479	Victor Lowther	Dell	11/17/11
480	Jerry Chen		11/18/11

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

481	Murali Raju		11/18/11
482	Benjamin Smith		11/18/11
483	Aaron Suggs	Kickstarter, Inc.	11/18/11
484	Lance Ivy	Kickstarter, Inc.	11/18/11
485	Cedric Howe	Kickstarter, Inc.	11/18/11
486	Tieg Zaharia	Kickstarter, Inc.	11/18/11
487	Teemu Matilainen	Reaktor Innovation	11/22/11
488	Dave Solbes	Webtrends Inc	11/22/11
489	Grant Hutchins		11/25/11
490	Eric Saxby		11/27/11
491	Gabriel Evans		11/27/11
492	Tim Smith	Webtrends Inc	11/28/11
493	Nathaniel Eliot	Infochimps, Inc	11/28/11
494	Adam Seever	Infochimps, Inc	11/28/11
495	Travis Dempsey	Infochimps, Inc	11/28/11
496	Dhruv Bansal	Infochimps, Inc	11/28/11
497	Andrew Kaczorek	Cycle Computing, LLC	11/29/11
498	Chris Chalfant	Cycle Computing, LLC	11/29/11
499	Dan Harris	Cycle Computing, LLC	11/29/11
500	Ian Alderman	Cycle Computing, LLC	11/29/11
501	Stephen Balukoff		11/30/11
502	Kendrick Martin	Webtrends Inc	12/1/11
503	Adnan Wahab		12/3/11
504	Alex Howells		12/4/11
505	Cameron Johnston		12/5/11
506	Justin Huff		12/8/11
507	Ian Downes	Ubalo, Inc	12/8/11
508	Erik Hollensbe		12/9/11
509	Karel Minarik		12/11/11
510	Adam Greene	SweetSpot Diabetes Care, Inc.	12/12/11
511	Justin Schumacher	SweetSpot Diabetes Care, Inc.	12/12/11
512	Dan Root	SweetSpot Diabetes Care, Inc.	12/12/11
513	Paul Dowman		12/12/11
514	Andrew Le		12/12/11
515	Paul Welch		12/13/11

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

516	Harlan Barnes		12/13/11
517	Philip Kates	Rackspace US, Inc	12/13/11
518	Brandon Philips	Rackspace US, Inc	12/13/11
519	Paul Querna	Rackspace US, Inc	12/13/11
520	Arthur Pirogovski		12/13/11
521	John Scott Sanders, Jr	RideCharge, Inc	12/15/11
522	Jamie Winsor	Riot Games	12/15/11
523	Josiah Kieh	Riot Games	12/15/11
524	Jesse Howarth	Riot Games	12/15/11
525	Michael Matsuuara	Riot Games	12/15/11
526	Cliff Dickerson	Riot Games	12/15/11
527	Philip Gollucci	RideCharge, Inc	12/15/11
528	Radim Marek		12/17/11
529	Hugo Fichter		12/19/11
530	Chris Christensen		12/20/11
531	Chet Luther		12/20/11
532	Mark Luntzel		12/20/11
533	Kevin Karwaski	Fiksu	12/21/11
534	David Calavera		12/22/11
535	Michael Stillwell		12/23/11
536	Aaron Bull Schaefer		12/28/11
537	Max Rabin		1/3/12
538	Michael Bradshaw	WhitePages Inc.	1/3/12
539	Michael Cook	WhitePages Inc.	1/3/12
540	Jack Foy	WhitePages Inc.	1/3/12
541	Devin Ben-Hur	WhitePages Inc.	1/3/12
542	Jeff Bellegarde	WhitePages Inc.	1/3/12
543	John Dyer		1/4/12
544	Sam Marx		1/5/12
545	Praveen Arimbrathodiyil		1/6/12
546	Joshua Buysse		1/6/12
547	Dale Hui		1/6/12
548	Jesse Campbell		1/8/12
549	Roberto Carlos Morano		1/9/12
550	Miah Johnson	CX, Inc.	1/10/12

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

551	Ian Delahorne		1/12/12
552	Mike Javorski	Spoke Software	1/15/12
553	Michael A. Fiedler		1/16/12
554	Luis Bosque		1/16/12
555	Qiming He		1/18/12
556	Hector Castro		1/19/12
557	David King	Xforty Technologies	1/25/12
558	Christian Pearce	Xforty Technologies	1/25/12
559	Andrew Libby	Xforty Technologies	1/25/12
560	Joshua Hou		1/27/12
561	Alice Kaerast		1/28/12
562	Brett Hoerner		1/28/12
563	Jon-Erik Schneiderhan		1/30/12
564	Wes Morgan	Democracy Works, Inc	1/30/12
565	Ernie Brodeur		1/31/12
566	Stephen Figgins		1/31/12
567	Tal Rotbart		2/1/12
568	Benjamin Lindsey		2/1/12
569	Tryn Mirell		2/2/12
570	Hari Krishna Dara		2/7/12
571	Andrew Grangaard		2/8/12
572	Adam Mielke		2/8/12
573	Andrew Allan		2/9/12
574	Antonio Soares de Azevedo Terceiro		2/9/12
575	Istvan Szukacs		2/9/12
576	Brian Parker	Pure Lake Software, Inc.	2/10/12
577	Sean Porter		2/10/12
578	William Carroll		2/12/12
579	Paul Diaconescu	Sveriges Television AB	2/14/12
580	Jonas Eklof	Sveriges Television AB	2/14/12
581	Per Bjorn	Sveriges Television AB	2/14/12
582	Frank Hoffsumer	Sveriges Television AB	2/14/12
583	Samppa Kytomaki	Reaktor Innovation	2/14/12
584	Zuhaib Siddique	Oxygen Cloud, Inc.	2/14/12
585	Andrew Robson	Oxygen Cloud, Inc.	2/14/12

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

586	Aaron Follette	Oxygen Cloud, Inc.	2/14/12
587	Erik Bakker		2/16/12
588	David Golden		2/16/12
589	Jacques Chester	Robojar Pty	2/17/12

Corporate CLAs

The list of Corporate CLAs allowed to contribute to Opscode projects. Only contributions from approved employees of these companies are acceptable.

Employees get approved by being listed on the schedule A of the Corporate CLA.

Number	Company	Date
1	Opscode	
2	Engine Yard	1/7/09
3	Wikia	
4	Aptana	2/12/09
5	CleanOffer	3/2/09
6	37signals	3/4/09
7	Nomitor	3/9/09
8	We Go To 12	4/30/09
9	Plus2 Pty	5/8/09
10	Phusion	6/22/09
11	Rightscale	6/30/09
12	Rubaidh	7/27/09
13	Peritor GmbH	8/12/09
14	Heroku	8/13/09
15	Internet Exchange	9/22/09
16	Betfair	9/30/09
17	Sojern	11/2/09
18	Runa	12/20/09
19	MaxMedia	1/11/10
20	Quantifind	2/11/10
21	VMware	2/11/10
22	Rackspace	2/26/10
23	Leaway Enterprise	3/16/10
24	Bueda	3/30/10

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

25	Divergent Logic	5/3/10
26	Basho Technologies	5/4/10
27	Seven Scale	5/13/10
28	IglooNET	5/21/10
29	Freistil Consulting	5/25/10
30	Promet Solutions	5/25/10
31	Mint Digital	6/16/10
32	Picklive	6/16/10
33	42 Lines	6/27/10
34	Wildfire Interactive	7/9/10
35	Dynamic Network Services	7/21/10
36	PeerPong	8/4/10
37	domainfactory GmbH	8/16/10
38	Tecnh	8/17/10
39	9Summer	9/9/10
40	Wixpress	9/13/10
41	Blue Box Group	9/29/10
42	FindsYou Limited	10/6/10
43	Highgroove Studios	10/25/10
44	ZeStuff	10/28/10
45	Worlize	10/28/10
46	Automated Labs	11/3/10
47	Estately	11/4/10
48	Kapoq	11/10/10
49	Openminds	11/10/10
50	MobileCause	11/10/10
51	Atalanta Systems	11/14/10
52	Menue Americas	11/17/10
53	Sociable Limited	12/1/10
54	Nine Summer	12/6/10
55	Neo Technology	1/27/11
56	Moriz GmbH	2/2/11
57	AegisCo	2/14/11
58	SetJam	2/15/11
59	Tippr	2/18/11

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

60	Ning	2/24/11
61	Workday	3/12/11
62	7digital	3/3/11
63	PagerDuty	3/17/11
64	Gnowsis	3/25/11
65	Unboxed Consulting	4/1/11
66	CustomInk	4/8/11
67	TalentBox	4/25/11
68	Wavii	4/29/11
69	Datadog	5/4/11
70	Viximo	5/10/11
71	ZephirWorks	5/11/11
72	Dell	5/12/11
73	Newsweek/Daily Beast Company	5/19/11
74	WordStream	5/19/11
75	Flagbit	6/14/11
76	Applications Online	6/17/11
77	Versapay	7/5/11
78	DigiTar	7/19/11
79	DreamHost	7/21/11
80	Edmunds.com	7/22/11
81	Every Ware	7/25/11
82	Ask.com	8/1/11
83	bring.out doo Sarajevo	8/11/11
84	Kos Media	8/15/11
85	reallyenglish.com	8/15/11
86	fewbytes	8/18/11
87	Business Intelligence Associates	8/19/11
88	Tacit Knowledge	8/22/11
89	Zenexity	8/30/11
90	ClassDo	8/30/11
91	Myplanet	9/2/11
92	ihiji	9/16/11
93	Port 80 Productions, LLC	10/28/11
94	Green Alto	11/2/11

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

95	Heavy Water Software, Inc.	11/4/11
96	Wealthfront Inc.	11/15/11
97	Kickstarter, Inc.	11/18/11
98	Webtrends Inc	11/22/11
99	Infochimps, Inc.	11/28/11
100	Cycle Computing, LLC	11/29/11
101	Ubalo, Inc	12/8/11
102	SweetSpot Diabetes Care, Inc.	12/12/11
103	RideCharge, Inc.	12/15/11
104	Riot Games, Inc.	12/15/11
105	Fisku	12/21/11
106	WhitePages Inc.	1/3/12
107	CX, Inc.	1/12/12
108	Spoke Software	1/15/12
109	Xforty Technologies	1/25/12
110	Democracy Works, Inc	1/30/12
111	Pure Lake Software, Inc.	2/10/12
112	Sveriges Television AB	2/14/12
113	Reaktor Innovation	2/14/12
114	Oxygen Cloud, Inc.	2/14/12
115	Robojar Pty	2/17/12

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Release Process



Opscode stewards a number of open-source projects, which includes managing new releases

1. **First, a Ticket is filed under the affected project calling for a new feature or bug fix**
2. **An approved contributor from the community provides the fix, possibly the same individual that filed the ticket**
3. **Opscode triages the ticket and merges it into a development branch**
4. **The feature or bug fix is included in the next release of the project.**

Tickets

The cornerstone of the release process is our ticketing system. All bug fixes and new features are documented under a ticket. This allows a global community to identify, document, reproduce and fix issues.

For the Chef and Ohai projects we follow this process. It differs slightly for others:

1. Ticket is filed describing a bug or desired feature
2. Any further documentation detailing the reproduction of the bug is added
3. A discussion may occur on the mailing list
4. Someone provides a possible fix and the ticket is marked resolved/fixed
5. The ticket is reviewed in one of Opscode's weekly triage meetings. If the fix is accepted for merge, it is marked 'Triaged,' otherwise it may be reopened for further work or rejected for a technical reason that will be specified.
    a. The contributor is validated as an approved contributor
    b. The code is reviewed
6. If the fix includes code, on a regular merge pass against triaged tickets:
    a. Unit tests are run
    b. A merge is completed
    c. fixVersion is set to the version of Chef that this will be included in
    d. The ticket is closed

Contributing

For information on contributing, the process is fully described at How to Contribute.

Versioning

The projects follow a versioning process similar to that documented by Rubygems.

Versions are represented by a series of three integers separated by periods, representing major, minor and point releases respectively.

Chef and Ohai currently use the minor version to indicate changes that may include breaking changes, while other projects use the major version similar to other projects. These types of changes may include a change to the API that is not backward compatible, or a change in the default behavior of a provider.

- An even numbered point version represents an official published release.
- An odd numbered point release means that you're using a development version.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

(While this can help identify that you've installed a project from the source code, it is important to remember that two installations with the same development version may not actually be the same based in development changes having occurred during the time between which the two same numbered development versions were installed.)

If needed, beta and release candidates are indiciated by a suffix of '.beta.x' or '.rc.x' respectively, where x is an integer.

For example, if the last published release was 1.0.0 and the current development release is 1.0.1, we may release:

- 1.0.2.beta.1,
- 1.0.2.beta.2,
- 1.0.2.rc.1, then
- 1.0.2 (final).

This is used because Gems, which is our primary method of releasing the code, consider any version with a letter in it as a pre-release, allowing us to push it to rubygems.org for download, but ensuring it won't be installed accidentally without the use of the '--pre' gem command flag.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Release MVPs

Opscode is quite proud of the Chef community



We usually give a Most Valuable Player (MVP) award for each release to individuals from the community who contributed greatly to that release.

Hall of Fame

These contributors are dependable leaders in the Chef community and earned themselves three MVP awards. This enters them into the hall of fame with our deepest gratitude.

- **Matthew Kent**
- **Doug MacEachern**
- **Tollef Fog Heen**
- **Thom May**
- **Bryan McLellan**



Chef

Release	Date	MVP
Chef 0.10.8	2011-12-15	Bryan Berry
Chef 0.10.6	2011-12-13	Andrea Campi
Chef 0.10.4	2011-08-11	Matthew Kent
Chef 0.10.2	2011-06-29	Daniel Oliver
Chef 0.10.0	2011-05-02	Grace Mollison, Darrin Eden
Chef 0.9.18	2011-06-29	Jesai Langenbach
Chef 0.9.16	2011-04-15	Michael Leinartas

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef 0.9.14	2011-03-04	Gilles Devaux
Chef 0.9.12	2010-10-22	Laurent Désarmes
Chef 0.9.10	2010-10-19	Toomas Pelberg, Tommy Bishop
Chef 0.9.8	2010-08-05	Joe Williams
Chef 0.9.6	2010-07-03	Caleb Tennis
Chef 0.9.4	2010-06-30	Ian Meyer
Chef 0.9.0	2010-06-21	Doug MacEachern
Chef 0.8.16	2010-05-11	Akzhan Abdulin
Chef 0.8.14	2010-05-07	Renaud Chaput
Chef 0.8.10	2010-04-02	Thom May, Tollef Fog Heen
Chef 0.8.8	2010-03-18	Eric Hankins
Chef 0.8.6	2010-03-05	Ian Meyer
Chef 0.8.4	2010-03-02	Tollef Fog Heen
Chef 0.8.2	2010-03-01	Scott M. Likens
Chef 0.7.16	2009-12-22	Bryan McLellan
Chef 0.7.14	2009-10-16	Thom May
Chef 0.7.12	2009-10-06	Diego Algorta
Chef 0.7.10	2009-09-04	Dan DeLeo
Chef 0.7.8	2009-08-13	Jeppe Nejsum Madsen
Chef 0.7.6	2009-08-08	Grant Zanetti
Chef 0.7.4	2009-06-26	Hongli Lai
Chef 0.7.2	2009-06-26	Joshua Sierles
Chef 0.7.0	2009-06-10	Matthew Kent
Chef 0.6.2	2009-04-29	David Balatero
Chef 0.6.0	2009-04-29	Matthew Kent
Chef 0.5.6	2009-03-06	Sean Cribbs
Chef 0.5.4	2009-02-13	Arjuna Christensen
Chef 0.5.2	2009-02-01	Bryan McLellan

Ohai

Release	Date	MVP
Ohai 0.6.10	2011-10-23	Nicolas Szalay, James Brinkerhoff
Ohai 0.6.8	2011-10-05	Bryan W. Berry

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Ohai 0.6.6	2011-10-03	Jason J. W. Williams
Ohai 0.6.4	2011-04-28	Josh Pasqualetto
Ohai 0.6.2	2011-04-14	Doug MacEachern
Ohai 0.6.0	2011-04-13	Kurt Yoder
Ohai 0.5.8	2010-10-19	Toomas Pelberg, Tommy Bishop
Ohai 0.5.6	2010-06-21	Doug MacEachern
Ohai 0.5.4	2010-05-11	Doug MacEachern
Ohai 0.5.0	2010-03-04	Tollef Fog Heen
Ohai 0.4.0	2010-02-28	Mark Giammarco, Jan Zimmek
Ohai 0.3.6	2009-10-16	Thom May
Ohai 0.3.4	2009-10-06	Diego Algorta
Ohai 0.3.2	2009-07-13	James Gartrell
Ohai 0.3.0	2009-06-18	Bryan McLellan
Ohai 0.2.0	2009-03-06	Thom May
Ohai 0.1.4	2009-02-01	Bryan McLellan

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Feature Proposals

This page is a jumping off point for various Chef Feature Proposals.



These are changes or additions to Chef that are a bit more heavy than a single ticket can summarize.

Please add your own!

As the various features are accepted, update the table to include the ticket number(s) where it has been captured and is being pursued.

Feature	*Status*
After hooks	Proposed
Chef 11 Release Checklist	Development
Chef Groups	Proposed - CHEF-506
Configuration Based on Partial Convergence	Proposed
Cookbook Documentation Functional Specification	Proposed
Explicit and Opt-in Filespecificity	Proposed
git resource updates	Proposed
Improved Windows File Security	Proposed
Infrastructure Proposal	Proposed
Machine Tags	Declined
Writing Chef Recipes in various languages	Proposed

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

After hooks

There are certain situations where it would be advantageous to manipulate the Chef::ResourceCollection after it has been compiled - essentially, we need after hooks between the compilation and execution stages of Chef.

This can be done today by simply making the recipes that include your 'after' hooks run at the end of the compilation - but it becomes difficult to manage a number of such hooks. An example of what is possible today:

An after hook in Chef 0.5.4

```
collection.each do |r|
  if r.provider == Chef::Provider::Package::Rubygems
    r.notify :run, resources(:execute => "random command")
  end
end
```

This would make sure that a random command gets notified every time a Gem package is installed. Another, more interesting case is with a package provider such as Yum, that is very costly to run. It would be great to let people distribute code that hooks in to the recipe process, and makes all package installations occur once. What follows is some untested code that would accomplish this:

Package Bundling for Yum

```
package_list = Array.new # The packages we are going to install at once
injection_point = 0      # Where in the resource collection the first package install appears
collection.each_index do |i| # Walk the index
  r = collection[i] # Set the current resource to r
  # if it is a yum package
  if Chef::Platform.find_provider_for_node(node, r) == Chef::Provider::Package::Yum
    # And we are installing without a version
    if r.action == :install && !r.version
      # Store the location, set the action to nothing, and store the name of the package
      injection_point = i unless injection_point == 0
      r.action(:nothing)
      package_list << r.package_name
    end
  end
end

# Create a resource to install all the yum pkgs at once
all_packages = execute "yum bundle" do
  command "yum install #{package_list.join(' ')}"
end
# Insert the new resource
collection.insert(i, all_packages)
```

Long term

It would be great to basically allow any recipe to embedd functionality like this, extending and modifying Chef. We propose the creation of an 'after' hook, that when it appears in any recipe will get executed at the end of the execution cycle. It will be passed the node and the resource collection. So the above would be wrapped:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

After hook version of Package Bundling for Yum

```
after "yum bundle" do |node, collection|
  package_list = Array.new # The packages we are going to install at once
  injection_point = 0      # Where in the resource collection the first package install appears
  collection.each_index do |i| # Walk the index
    r = collection[i] # Set the current resource to r
    # if it is a yum package
    if Chef::Platform.find_provider_for_node(node, r) == Chef::Provider::Package::Yum
      # And we are installing without a version
      if r.action == :install && !r.version
        # Store the location, set the action to nothing, and store the name of the package
        injection_point = i unless injection_point == 0
        r.action(:nothing)
        package_list << r.package_name
      end
    end
  end

  # Create a resource to install all the yum pkgs at once
  all_packages = execute "yum bundle" do
    command "yum install #{package_list.join(' ')}"
  end
  # Insert the new resource
  collection.insert(i, all_packages)
end
```

Note that, in this specific case, there are interesting edge cases. If you specify the version of a package, and we first bundle up a package that depends on it, with yum you will likely not get the behavior you want (you'll get the current version of the dependency, most likely.) Them's the breaks when you manipulate stuff this way. 

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef 11 Release Checklist

A list of small dev tasks that we need to do for Chef 11

- Deprecate Chef::ShellOut compatibility layer - replaced by Mixlib::ShellOut
- Remove Chef::REST#run_request (replaced by api_request and streaming_request)
- Implement platform_family for platform-specific resource Mapping.
- remove cookbook overlays, add support for multiple cookbook versions in different cookbook directories
- merb -> rails3 for webui (opscode esprint-1200)

Proposed

- rename Chef::REST get_rest, put_rest, etc. without the "_rest", replace with aliases to the old names and deprecate those

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Groups

I have been brainstorming on how to build groups functionality into Chef and the best ways to do it. My thinking is that there are two types of groups, mixed groups and configuration groups. Mixed groups are a collection of machines, such as a group of machines that provide a LAMP stack for a website. They are a group because they combine to form a operational unit. Configuration groups are groups that have identical configurations (outside of specific attributes like ip address), such as a group of load balanced web servers. They all have the same configuration and are generally reconfigured as a whole. Additionally what I mean by "configuration" is a combination of roles, cookbooks, recipes. Here's where it gets tricky and fun, mixed groups can contain configuration groups since you may be have a load balanced web server setup for the hypothetical LAMP stack. Hypothetically one could write a cookbook that says "if this machine is in the X configuration group and the Y mixed group do Z". I think this type of hierarchy may lead to some interesting things (inheritance?). Nodes would have attributes stating the groups that it belongs to and there would also be docs in couch stating which the group and its contents whether it be nodes or configurations.

As I mentioned this is still brainstorming so I would be grateful for feedback before I start digging into the code.

(JIRA Ticket: http://tickets.opscode.com/browse/CHEF-498)

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Configuration Based on Partial Convergence



Summary

There are cases in Chef where the configuration of one or more resources depends on the system state or data collected by another resource which is not available until some part of the system's configuration has converged. However, Chef's two phase (compile/converge) execution model, combined with the pass-by-value semantics of Node attributes, makes it unwieldy to dynamically determine configuration values based on data that is not available until the converge phase.

Given that both Chef itself and the Chef community strongly prefer to bring a node to full convergence in a single run, we need to review the use cases for this kind of behavior, as well as the existing patterns/solutions/hacks/workarounds used to achieve it. After a review of this information, we will be in a position to determine if modifications should be made to Chef to enhance support for this behavior, and what those enhancements will look like.

Use Cases Needed

If you need this behavior, please edit this page to describe your use case in the section below

Use Cases

(1) Calling packages in ruby_block above the package resource

Having the capability to have resources cycle thru an arbitrary run-time list is very useful. For example: in a later section of a script, ownership is to bve fixed across a bunch of files, trivial to do using legacy bash, but cant be done easily in Chef (the ability to retrieve the list and manipulate it is also extremely easy in Ruby, but isnt translating to Chef).

Code in a ruby_block is evaluated with other resources during configuration/convergence (2nd phase), but packages are only added to the collection during compilation (1st phase). During compilation (1st Phase) the files are not available from the resource/package evaulation in the (2nd phase). So when calling package (or rpm_package) in the ruby_block above the package resource effectively does not exist, and if it did it would never get executed.

Hacky solution:

```
ruby_block "install_vertica" do
  block do
    Dir.glob("/tmp/vertica/**/*.rpm").each do |pkg|
      run_context = Chef::RunContext.new(node, {})
      p = Chef::Resource::Package.new(pkg, run_context)
      p.source(pkg)
      p.run_action(:install)
    end
  end
  action :nothing
end

log "Install Vertica - Download Vertica"
script "download_vertica" do
  interpreter "bash"
  user "root"
  cwd "/tmp/vertica"
  notifies :create, resources(:ruby_block => "install_vertica"), :immediately
end
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

(2) JIT evaluation

There are times when passing parameters to resources it would be nice if that parameter was not evaluated at compile time but at runtime. This is clearly and most easily in play with not_if/only_if, where the lambda/string that is passed is evaluated just before runtime.

An easy implementation of this would be to allow the passage of lambdas as parameter arguments, such that the evaluation of the lambda is done at runtime. I'm not sure what downside this brings other than complexity, but it can be very valuable in certain situations.

The problem here isn't in triggering one action off of another, but fulfilling the values of that action. For example, what if I had a user creation resource, then a template that needed to fill in a value based on the created UID of that resource?

```
user "blah"

template "/etc/blah" do
  variables(:user_id => SomeLibrary.lookup_the_useruid_from_etc_passwd("blah") )
end
```

Obviously this won't work because the lookup is done at compile time, before the new user is ever created.

However, if I can tuck that into a lambda:

```
template "/etc/blah" do
  variables(:user_id => lambda do SomeLibrary.lookup_the_useruid_from_etc_passwd("blah") end )
end
```

Then I can defer that the lookup isn't evaluated until the value is actually needed, JIT.

It may be valuable to do this in more than template variables, but that's where my example is best seen I think.

(3) Service Discovery

Ganglia wants to know where all its agents are. Its agents want to know where their master is.

Zabbix wants to monitor flume. Flume wants to transport zabbix's logs to the HDFS.

Even when there is no Ouroboros present, this is a persistent source of race conditions. Service A wants to consume Service B, which could be on the same or a different node (staging vs production, master is also a worker, lightweight redis for rails app vs bigass replicated redis for primary data store). Stop/starting a machine on EC2 yields a new IP address, so it's difficult to distinguish a remote resource from an outdated shadow of your former self.

Anyway, the best time for this discovery pass is when the machine has a full sense of its purpose in life and what it has to offer.

It would also mitigate cross-cluster race conditions. Simultaneously launching 30 machines that all want to cross-discover is an interesting affair; a little breathing room between announce and discover would greatly smooth the path to idempotent runs.

(4) Components that might not be present

An hbase node might have one some or all of the hbase master, namenode, datanode, and eight other services. Some of those depend on the same file (in this case, /etc/hadoop/conf/core-site.xml and a few others). When that file is modified, I'd like to restart the daemons. If the file is prepared at the end of convergence, I can know what came before it.

However, the hadoop apt packages start the daemons on install – which means you'd like to have the template file in place **before** anything else runs.

We use a workaround – we always declare a :run_state attribute for daemons, so other components will assume that the service exists if and only if the service is in the 'start' run_state. But that's hacky and wrong.

(4) Competing for Utilization

Going still with HBase/Hadoop – if I'm a typical hadoop compute machine, with only a tasktracker and datanode, I can be agressive in memory usage, largely towards the tasktracker – serving data is boring, it's compute jobs that need ram. If I'm a machine that has an hbase regionserver on it, I want one little wussy tasktracker (one which will be off/asleep for most of its life), a healthy datanode, a highly aggressive regionserver and

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

probably 50+% of ram left over for system caches.

Now you're saying to yourself "self, that sounds like a case for machine-specific tuning". But I'm not talking about an infinite multiplicity of use cases: it's "be a reasonably aggressive compute node when there is no regionserver" or "be a reasonably aggressive database node when there is" or "do it your damn self".

In that case, you have two fairly close-bound components, so it might be fair (if distasteful) to have some coupled logic.

However, consider a machine that specifies both say tokyotyrant and elasticsearch (or their like). Each uses only a single volume, but it uses the hell out of that volume. There's no built-in way to have them peacefully choose distinct volumes on a machine with more than one disk.

More broadly, the machine has a certain set of resources – ram, volumes, ports, uids, etc – that these components would like to discover, claim or use their proportionate share of. Until they know what's out there, they can't be good neighbors and ambitious public servants.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook Documentation Functional Specification

Overview

People want to write Chef cookbooks, but more importantly they want to use Chef cookbooks. To make cookbooks more useful for people, they need to be documented so people can know how to use them.

Disclaimer: This spec is a living document, it's on a wiki! While you can update it with additional requirements, not everyone will rush out and update their documentation.

This spec doesn't describe how to use Chef. It doesn't describe how to program in Ruby, either.

Scenarios - Just why should someone document their cookbooks?

Richard is a system administrator configuring systems to run an uncommon software repository management system. He wants to write a new Chef cookbook to configure this for him, and make sure other people can understand what he's written.

Cheryl is a programmer looking for an existing set of cookbooks to help her deploy her application. She needs a basic web server, load balancer stack, but she doesn't want to write a whole new cookbook. She just needs to understand some cookbooks written by the community so she can drop them into her configuration.

So What Gets Documented?

A cookbook itself has several components. Cheryl and Richard need to know how to use each one. Recipes are a special component and detailed separately.

- Attributes - Describe the attributes in context. If the name isn't obvious, say what it is for.
- Definitions - List the parameters used, what object type the definition will use (String, Array, etc), and provide an example resource using the definition.
- Libraries - The full power of RDoc can be realized here, as libraries are arbitrary Ruby.
- Files and Templates - Describe the files and templates used in the resources that use them.

Recipes

Recipes can become very complex with interdependencies. We need to know the following about each recipe:

- List of resources the recipe creates.
- Attributes required.
- If other recipes are required, mention them.
- If this recipe needs site-specific configuration, mention it.
- Known issues.

Documentation Markup

Using an easy to implement comment markup tool could output documentation in a human readable form, such as html, textile, markdown and other languages. The output could be imported or hosted via the chef-server web interface.

Such a tool would allow Richard to create his documentation easily without having to think about it. Cheryl could view cookbook documentation while she's managing her app servers from her Chef interface.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Explicit and Opt-in Filespecificity

Abstract

File Specificity is a feature of Chef where a Cookbook file or template is dynamically sourced based on the hostname and platform of the system under management. The current implementation of file specificity has the following disadvantages:

- *Users who do not need this feature are required to understand or at least be aware of it in order to create cookbooks:* The directory layouts for templates and cookbook files counterintuitively require the user to place files under a `default/` directory. Before learning about file specificity, users expect to be able to place cookbook files and templates in the root of the `files/` and `templates/` directories, respectively.
- *There is no way for specificity to operate on any property other than hostname and platform:* Many users have homogenous (or reasonably homogenous) environments, but would have a use for specificity based on geography (datacenter A gets a different file), environment (production/qa/dev gets a different file) or another property.

To resolve these deficiencies, file specificity should be *opt-in* and *explicit*. When no specificity is declared, files should be fetched from the root of `files/` or `templates/` as appropriate. When enabled, file specificity should support user-defined specificity rules on a per-resource basis.

Compatibility Concerns

If implemented immediately with no deprecation period, this proposal would break almost every cookbook. Therefore, if implemented, it must be implemented in stages across several major releases of Chef, with backwards compatibility maintained for a reasonable amount of time and clear messaging to users about what is changing, when, and why. If possible, automated tooling should be provided to migrate cookbooks.

Implementation.

Source as Array

The fully explicit syntax would look like this:

```
file "foo" do
  source "host-www.example.com/foo", "ubuntu-10.04/foo", "ubuntu/foo", "default/foo"
end
```

However, helper methods can be provided to make this less onerous:

```
file "foo" do
  source platform_specific("foo")
end
```

Explicit Specificity Option

TODO

Adds a specificity attribute, which takes a code block or a symbol...

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

git resource updates



This is a grouping of three related proposals

(1) Creating a common codepath for the git resource
(2) Adding developer_mode
(3) Adding a local_changes parameter.

Git Resource Change 1: Robustification

The idea here is to put checkout and update onto a common codepath, ensuring that anything that checkout sets up, sync updates (and vice versa). This gives us greater confidence that the code works in both cases, and reduces the test burden. It also makes it easier to add features like developer_mode which would have to be put into both codepaths.

Presently, the git resource works like this:

```
if <destination>/.git exists
  if there is something to sync
    <set remote tracking branch> unless remote = origin
    git fetch <remote>
    git fetch <remote> --tags
    git reset --hard <target revision>
    git submodule init
    git submodule update
  end
  foreach additional_remote
    git remote add <remote> <url>

else
  git clone [-o remote] [--depth depth] repository destination
  git checkout -b deploy <target revision>
  git submodule init
  git submodule update
  foreach additional_remote
    git remote add <remote> <url>
end
```

This model puts setup in the else branch, and maintenance in the if branch. What may not be obvious is that some things that get set in the else branch, don't get updated in the if branch. For example, the branch doesn't get updated (if it gets out of sync, it won't get fixed).

Proposal: One Thing At A Time

The way to think about the resource is, it declares the state it wants to move to. It should be in branch X, it should be at revision Y, it should come from repository Z ...

The proposed flow takes it one piece at a time, first cloning from the repository, then getting on the right branch, then getting on the right revision, then getting the right remotes ...

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
# Ensure we have a working git clone
if <destination>/.git does not exist
  git clone [-o remote] [--depth depth] repository destination
end

# Create branch
if "deploy" branch does not exist
  git branch deploy <target revision>
end

# Check out branch
if current_branch != “deploy”
  git checkout deploy
end

# Update branch to proper revision
if branch is out of sync with target
  git fetch <remote>
  git fetch <remote> --tags
  git reset --hard <target revision>
end

# Set remotes
<set primary remote tracking branch name and url> unless remote = origin
foreach additional_remote
  <set remote tracking branch name and url>
```

This model has the general pattern: if <thing is not what it should be> <make it so>. By breaking it up into fine-grained increments, any individual thing that goes wrong gets fixed, even if other things are in their proper state. (i.e. if the branch changes but nothing else, the branch gets smacked back into shape without changing anything else.)

This model also allows us to have exactly one codepath, so that checkout and sync are both (largely) tested simultaneously.

Git Resource Change 2: `developer_mode`

When setting up git repositories for use by a developer, the current detached “deploy” branch is less than ideal. This proposal adds “developer_mode” to the git resource, which does not create “deploy” branch and instead leaves the user in whatever branch they wanted to use. This would change the above process like so:

```
# Create branch
if <target branch or "deploy"> does not exist
  git branch <target branch or "deploy"> <target revision>
end

# Set upstream
if developer_mode
  git branch --set-upstream <target branch> <remote>/<target branch>
end

# Checkout
git checkout <target branch or "deploy">
```

This is still suboptimal due to the “git reset --hard”, but it gets us a little closer to where we want to be. We address that issue in the third change.

Git Resource Change 3: `update_method`

One issue with the git resource at present is that the policy regarding files that get modified in-place is inflexible, and does not work well for developer mode because the default policy is to overwrite changes. This proposal adds an `update_method` attribute to the git resource, which allows you to specify what should happen with any modified files on your drive. The possible values are:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Value	Git command	Local Modifications	New Untracked Files	Local Commits	In Case Of Conflict	Notes
`:reset_merge`	`git reset --merge`	Keeps changes	Keeps new files	Loses commits	Fails update	
`:reset_hard`	`git reset --hard`	Overwrites changes	Keeps new files	Loses commits	Overwrites local changes	Default
`:reset_clean`	`git reset --hard` `git clean -d`	Overwrites changes	Deletes new files	Loses commits	Overwrites local changes	
`:rebase`	`git rebase`	Keeps changes	Keeps new files	Keeps (rebases) commits	Fails update	Default for `developer_mode`. Does not work with revision or tag.

Unresolved

When do local_changes get updated?
If you have chosen git reset --hard (the default), there is another policy decision that needs to be made. If a file gets modified in-place, when should it get overwritten? Currently we wait until a revision happens on the server. One can imagine people who want their state to converge to the server's state sooner than that. Perhaps a value like local_changes_update=:immediate or :on_version_change

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Improved Windows File Security



Know Your Rights: Windows File Security In Chef

With 0.10.10, we're overhauling the Windows security settings for file and directory resources to work the way Windows works. Specifically, we're adding inheritance and full ACL support, and tweaking the owner, group and mode attributes to do the right thing on Windows. We've made these changes directly to the template, file, remote_file, cookbook_file, directory, and remote_directory resources, so you can start getting the benefit right away by modifying your existing recipes.

Unix security configuration will not change.

ACLs: the "rights" attribute

First off, ACLs. Windows lets you give permissions to multiple users and groups, and the new "rights" attribute gives you control over this powerful feature. It looks like this:

```
file "x.txt" do
  rights :read, "Everyone"
  rights :write, "UBERDOMAIN\Sales"
  rights :full_control, "Frodo Baggins"
end
```

You can add as many (or as few) rights attributes as you like, and Chef will apply them to the file or directory. Here's the syntax:

```
rights <permissions>, <principal>, <options>
```

The `<permissions>` parameter specifies the what rights will be granted to the principal. The possible values are `:read`, `:write`, `:full_control` and `:deny`. *These permissions are cumulative*: if you specify `:write`, it includes `:read`, and `:full_control` includes both `:write` and `:read`. If you specify `:deny`, the user or group in question will have no rights at all to the object in question.

Editor: insert some nice pretty Windows screenshots showing what these mean with the security dialog box.

For those who know the Windows API. `:read` corresponds to `GENERIC_READ` and `GENERIC_EXECUTE`; `:write` corresponds to `GENERIC_WRITE`, `GENERIC_READ` and `GENERIC_EXECUTE`, and :full_control corresponds to `GENERIC_ALL`, allowing a user to change the owner and other metadata about a file.

The `<principal>` parameter is a string specifying a group name or user name. This can be anything you can type in the logon box for Windows: either "`username`", "`domain\username`", or "`username@fully_qualified_domain`". You don't need to tell Chef whether it's a user or group; we'll figure it out when we look it up. *Editor: support for SIDs might be nice. The ability to specify well-known SIDs like Everyone in a language-independent way would be good, too.*

The `<options>` parameter is a hash with several possible advanced rights options. For example, you can add a right to a directory that only applies to its children like this: "`rights :write, "UBERDOMAIN\Sales", :applies_to_self => false`". This table shows all the options:

Key	Possible Values	Meaning
`:applies_to_children`	**`true`**	This right will be inherited by both child directories and files (default)
	`false`	This right will *not* be inherited by any child directories or files.
	`:containers_only`	Child directories (but not files) will inherit this right.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

	`:objects_only`	Child files (but not directories) will inherit this right. (This is recursive.)
`:applies_to_self`	**`true`**	This right will apply to the directory or file resource (default)
	`false`	This right will *not* apply to the directory or file resource, but may apply to any children.
`:one_level_deep`	`true`	Only the first level of children will inherit this right.
	`false`	All children (recursively) will inherit this right.

Other important things to know about rights:

- **Rights are order-independent:** it doesn't matter whether you say `rights :deny, "Sally"` before or after `rights :read, "Sally"`, Sally will be unable to read the document.
- **When you specify rights, they are considered a complete description of all explicit rights on a file**: all existing explicit rights will be removed and the new ones added. (Inherited rights will remain on the file).
- **If you do not specify any rights (or mode) on a file, security will remain untouched:** More specifically, we will not clear out the rights on the file if you fail to specify rights and mode. We figure if you wanted us to mess with it, you would have told us something. *Editor's Note: we need give the user a way to specify that all rights should be cleared.*
- **Changing inheritable rights on a directory with many files can be expensive:** Chef will make sure not to modify rights when nothing has changed, but when it does a large directory can take a while. This is due to inheritance (Windows will propagate the new rights to all children recursively). This is a normal part of running on Windows, but if you need to do this often, careful control over inheritance options can help you get rid of some of the cost.

Inheritance: the `inherits` attribute

By default, any file or directory you create inherits rights from its parent directory. Normally this is exactly what you want; but sometimes you want to carve a little space off from the rest of the world where you control every right. You can tell Chef that your file or directory should *not* inherit rights from its parent by specifying "`inherits false`":

```
directory 'C:\mordor' do
  rights :read, 'MORDOR\Minions'
  rights :full_control, 'MORDOR\Sauron'
end

directory 'C:\mordor\mount_doom' do
  rights :full_control, 'MORDOR\Sauron'
  inherits false # Sauron is the only person who should have any sort of access
end
```

You could have used `:deny` to accomplish the same ends, but unless you deny everyone, something could slip through. It all depends on whether you want that or not.

```
directory 'C:\mordor' do
  rights :read, 'MORDOR\Minions'
  rights :full_control, 'MORDOR\Sauron'
  rights :write, 'SHIRE\Frodo' # Who put that there I didn't put that there
end

directory 'C:\mordor\mount_doom' do
  rights :deny, 'MORDOR\Minions' # Oops, not specific enough
end
```

If you do not specify the `inherits` attribute, Chef will default it to true for new files, but will leave it unchanged for existing files (so if you go modify the file yourself, it will not get flipped back on the next Chef run).

`mode`, `user` and `group`

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

The existing mode, user and group mean roughly the same thing as they did before, with the following modifications:

- **`owner`:** We've added "`owner`", which is the proper Windows term for file ownership. This works like user did before, and now supports any username (including fully qualified "`domain\user`" and "`user@domain`"). If you do not specify this, new files will pick up the current user as the owner and existing files will leave their owner unchanged.
- **`user`:** "`user`" is now an alias to owner for backwards compatibility.*
- **`group`:** This now supports any groupname (including fully qualified "`domain\group`" and "`group@domain`"). If you do not specify this, new files will get your default POSIX group (which you likely don't have), and existing files will have their group unchanged.
- **`mode`:** This is provided for backwards compatibility. This mask is now translated into rights. Values up to 0777 are allowed (no sticky bits), and mean the same thing as the Unix counterpart, with 4 = `GENERIC_READ`, 2 = `GENERIC_WRITE` and 1 = `GENERIC_EXECUTE`. You cannot give `:full_control` with mode (but the owner has full control anyway). mode has no effect when it is not specified. When both mode and rights are specified, the effects are cumulative.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Infrastructure Proposal

Currently, a Chef Server contains nodes, roles, and cookbooks - it's responsible for distributing cookbooks to nodes, allows you to assign roles, and more. This approach works fine when your entire infrastructure is monolithic; as long as every system that uses a given cookbook is ready for that cookbook to be updated, everything is copacetic. The issue arises when this is no longer the case - for example, when you want to have a production, development, or staging environment, where new versions of a cookbook can be used in each. Today, this is only possible by running a distinct chef server instance per environment.

This proposal documents the current thinking around the creation of 'Infrastructures', which solve the above problem.

Infrastructures can be though of as a 'container' for a group of cookbooks, roles, and nodes. Each infrastructure will be given a name, such as 'production', 'development', or 'bootstrapping', along with a long-form description.

What follows is a breakdown of how each component would need to change with the addition of infrastructures, and how they will be exposed through infrastructures.

Components

Cookbooks

We already track cookbook versions. With the addition of infrastructures, we will change the on-disk layout for how the chef server stores cookbooks to be version specific, and to allow multiple versions of the cookbook to be uploaded. For example, when you run:

Uploading cookbooks

```
$ knife cookbook upload apache2
```

Rather than storing the new apache2 cookbook directly at `/var/chef/cookbooks/apache2`, we'll shift to `/var/chef/cookbooks/apache2/VERSION`. The cookbook loader will be extended to understand the existence of multiple cookbook versions.

Each infrastructure will be able to look at the list of cookbooks available on the server, and set itself to using the latest version of the cookbook, a specific version of the cookbook, or to exclude the cookbook altogether. For example, if we had version 1.0.0, 2.0.0, and 3.0.0 of the apache2 cookbook, we can set the production infrastructure at the '2.0.0' version of the cookbook, while letting the development environment always be at the 'latest' version, and making the apcahe2 cookbook unavailable at all within the 'bootstrapping' infrastructure.

Roles

Roles do not currently have a version number applied. In order to support multiple disparate configurations, we'll need to add this ability, or something similar, to roles. Either they gain the ability to be versioned, or they get entirely containerized within an infrastrucutre - and it's the responsibility of the end user to keep the role up to date.

Nodes

A node is 'assigned' to an infrastructure, and it can be in only one at a time.

Table of Contents

- Components
  - Cookbooks
  - Roles
  - Nodes
- API
  - /infrastructures
  - /infrastructures/NAME
- Impacts on Search
- Migration/promotion of cookbooks

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

API

A simple API proposal would look like:

/infrastructures

GET

Returns a list of infrastructures.

GET /infrastructures

```
{
  "production": "https://example.com/infrastructures/production",
  "bootstrapping": "https://example.com/infrastructures/production"
}
```

POST

Creates a new infrastructure. The body of the POST should be as follows:

POST /infrastructures

```
{
  "name": "production",
  "cookbooks": {
    "apache2": "latest",
    "snort": "1.0.1",
    "fatman": "exclude"
  },
  "roles": {
    "webserver": "1.0.0"
  }
}
```

The hash of cookbooks and roles both point to versions. If a cookbook or role is excluded, then we will default to creating the infrastructure with it pinned to 'latest'.

/infrastructures/NAME

GET

Returns the current configuration for this infrastructure. Response looks like:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

POST /infrastructures

```
{
  "name": "production",
  "cookbooks": {
    "apache2": "latest",
    "snort": "1.0.1",
    "fatman": "exclude"
    ....
    "rock": "1.1.1"
  },
  "roles": {
    "webserver": "1.0.0"
  }
}
```

PUT

Updates the role. The body should look like the response from GET.

Impacts on Search

By default, any search request tendered from within a recipe will be scoped to the current infrastructure. For example:

Search with Infrastructures

```
search(:node, "monkey:behind_glass")
```

Would only search within the current node's infrastructure (say, 'production'). If you wanted to search every node known to the chef server, you would do:

Search with Infrastructures - for all nodes

```
# Only in production
search(:node, "monkey:behind_glass", "production")

# Only in production and bootstrap
search(:node, "monkey:behind_glass", [ "production", "bootstrap" ])
```

Migration/promotion of cookbooks

Would be possible through the API - this proposal makes no workflow assumptions, but they would be easily enabled.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Writing Chef Recipes in various languages

While we love Ruby, you should be able to write Chef cookbooks in your language of choice. In order to complete this process, the following changes to Chef are necessary:

Chef::CookbookLoader must understand different languages file types

Currently, the cookbook loader looks for '*.rb' files. It will need to load all the files in the tree that we understand how to parse.

Chef::Cookbook needs to run files for each language

Now that the cookbooks can contain files of different languages, we need to be able to execute them. This process is as follows:

For attributes

We should shell out to the proper interpreter, passing the node object as JSON. We should read the node object back out, replacing the original with the new version.

For libraries

We should include the libraries for the given language as included on the command line (when supported).

For definitions

In order for definitions to work, we need to have a mechanism for re-constituting a Chef::ResourceDefinition object in ruby for each. That will allow us to take the resources generated in any language and utilize the definition for each properly.

For recipes

The languages should send out an interim-format as JSON for the resources to add to the collection. We should then take this format and build resource objects as normal, doing validation. (This differs from the perl-based prototype I have currently, which spits out raw Chef::Resource objects.) This will allow us to keep all the same validation, etc. that exists in the ruby libs.

Perl Prototype

A working Perl prototype exists, that works for just recipes. You can find it on CPAN and the source on GitHub. To give it a shot:

- Install the Chef perl module, and its dependencies. (Moose and JSON::Any)
- Stick the perl cookbook in your chef repository.
- Put your perl recipes in the cookbooks files/default/perl_recipes directory, and watch in amazement as they get executed.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Users List

This list of people / organizations using Chef is to help our community get to know each other better.

Please add an entry for yourself or your organization and include the following:

- Organization/Person
- Usage/implementation: A brief description of what you're using Chef to do.
- IRC: If you use IRC, please include your nickname.
- Twitter: If you use Twitter and don't mind some more followers.

Opscode

Who	We wrote Chef! We eat our own dog food, too  .
How	We run our production infrastructure with Chef, obviously.
IRC nick(s)	holoway, jtimberman, kallistec, skeptomai, jesserobbins, barrysteinglass, btmspox, schisamo, stephendelano, tomascz
Twitter	@opscode, @jtimberman, @kallistec, @adamhjk, @raxmus, @jesserobbins, @skeptomai, @btmspox, @schisamo, @sdelano, @mattray, @timhin, @tmonk42, @nuoyan, @sfalcon, @tomasczt, @metaxis, @sarahnovoty, @stevedanna, @jesbourne, @someara, @kevsmith, @*cwm*, @Marc_Paradise, @mzyk83

ThoughtWorks

Who	A company wholly devoted to the art and science of custom software. We make it, and we make our clients better at it. Our bottom line is to design and deliver software fast and predictably
How	To manage our internal apps and infrastructure, to help our client doing continuous delivery
IRC nick(s)	ranjibd
Twitter	@RanjibDey, @AjeyGore

Runa

Who	SaaS for Online Merchants, allows them to create dynamic promotions based on consumer interests and behavior
How	We're working on our next gen (Chef 0.10.x / Hosted Chef) deployment of Hadoop/HBase, RabbitMQ, Redis, our Clojure based Swarmiji distributed processing cluster. We also use Rails for Merchant and Administrative dashboard. We've been using Chef since 0.8 alpha to deploy our infrastructure on Amazon EC2
IRC nick(s)	rberger, sivajag, rathore
Twitter	@rberger, @sivajag, @amitrathore

Cloudant

Who	A database designed for the web
How	Managing our infrastructure and deploying erlang apps with chef

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

IRC nick(s)	joewilliams
Twitter	@williamsjoe

Vaycay

Who	Vacation Destination Planning
How	Configuration of application, database, and caching servers to Rackspace Cloud.

Blue State Digital

Who	Web site development, technology development and implementation and strategic campaign and program management services
How	Installing and managing the various daemons and frameworks that make our Linux servers go, and provisioning new servers
Twitter	@bsdwire

Peritor Scalarium

Who	Cloud provisioning and deployment on Amazon EC2
How	Configuring, managing and deploying the instance landscape for customers so they don't have to.
IRC nick(s)	roidrage
Twitter	@scalarium, @peritor, @roidrage, @jweiss

VersaPay

Who	VersaPay offers everything you need to accept credit and debit card payments in person, on the go, on your website, and in the office
How	Configure, manage, implement all aspects of systems infrastructure using Chef
IRC nick(s)	mcarruthers
Twitter	@versapay_tech @versapay @mdcarruthers

Flowdock

Who	Flowdock is a team messenger for agile software developers.
How	Configure, manage and deploy our complex technology stack (involving Ruby on Rails, Scala, Cassandra, ...)
IRC nick(s)	Mutru
Twitter	@flowdock @mutru

Evite

Who	Evite is the premier social-planning website for creating, sending, and managing online invitations.
How	(starting to) configure, manage and deploy our technology stack (involving Python/tornado, nginx, MySQL, MongoDB, ...)
IRC nick(s)	ggheo
Twitter	@griggheo

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Socrata

Who	Socrata provides a turn-key data publishing platform for government agencies, non-profits, and corporations.
How	We use Chef to manage our production and staging environments across multiple datacenters.
IRC nick(s)	natacado
Twitter	@natacado

DevStructure

Who	DevStructure makes smart development environments sold as a service that can figure out what you've done on your server and generate Chef cookbooks (shell scripts and Puppet manifests, too).
How	Our development environments come stocked with out sandbox and blueprint utilities, which are also accessible via the web.
IRC	rcrowley and zenmatt, usually found in #devstructure
Twitter	@devstructure, @rcrowley, and @zenmatt

Reframe It

Who	We wrote Reframe It, a tool that let's you comment on web sites.
How	We're rearchitecting our system, and part of that effort is managing our infrastructure with Chef.
IRC nick(s)	coshx, ajburton, usually available in #reframeit
Twitter	@anthonyburton, @coshx

ZephirWorks

Who	We are a boutique app development and devops shop targeting medium and large companies across Europe.
How	We use Chef to help our customers (mostly in the publishing and telco markets) manage their servers; that includes writing custom cookbooks and certifying stock cookbooks on the different distros. Of course we also use it to manage our own infrastructure.
IRC nick(s)	acampi
Twitter	@andreacampi, @andreacgranata

ShopRun

Who	We are a Tokyo-based SaaS, (the links are Japanese only FTM). ShopRun is a multi-tenant chain-store management service with a 100% uptime SLA.
How	We manage an important chunk of our cloud infrastructure with Chef. We never directly ssh anymore!
Twitter	@shoprun

Aframe

Who	We're a collaborative video platform
How	We run all of our infrastructure with Chef, and we love infrastructure-as-code.
IRC nick(s)	jonlives

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Twitter	@aframehq, @jonlives

DataSift & TweetMeme

Who	We're a realtime social data mining platform
How	All of our development, staging and production servers are entirely managed by Chef. No way we could do it without it!
IRC nick(s)	alexforrow, Gareth / NetworkString
Twitter	@alexforrow , @NetworkString

GSI Helmholtz Centre for Heavy Ion Research

Who	We're a major german public physics research institute
How	We are adopting Infrastructure as Code to ramp up our computing centres including a large multi-application batch computing farm processing data from our local particle accelerator and from LHC.
IRC nick(s)	Reverand

Heavywater Software

Who	We manage infrastructure for small and medium-sized companies using Chef
How	Relentless focus on automation, Hosted Chef, and building awesome teams.
IRC nick(s)	dje, fujin_, slyness, luckymike, webframp,
Twitter	@dje, @fujin_, @slyness, @luckymike, @webframp

Autodesk

Who	We're a provider of 3D design software for a variety of discipliness such as architecture, engineering, and construction.
How	We're adopting Chef to deploy our infrastructure which supports Autodesk Cloud Services.
IRC nick(s)	stphung
Twitter	@stphung

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Presentations

Presentations given by members of our fantastic community

Either upload an attachment and link to it, or link directly to an external URI. Please state when and where the talk was given and a one-line abstract.

Title	**Infrastructure Automation with Opscode Chef**
Who	Joshua Timberman, Adam Jacob, Christopher Brown, Aaron Peterson, Seth Chisamore, Matt Ray
When and Where	06/14/2011 at Velocity Conference
Summary	A overview of managing infrastructure in the cloud - with Chef

Title	**Tasty Infrastructure**
Who	Mark Imbriaco
When and Where	06/16/2009 at Raleigh.rb
Summary	A very high level spin through Chef. Just a few slides, most of the talk was interactive.

Title	**Chef - Evolving with Infrastructure Automation**
Who	Nathaniel Brown
When and Where	10/06/2009 in Hawaii at Aloha on Rails
Summary	Why cooking with Chef can change your business forever
Slides	Chef - Evolving with Infrastructure Automation

Title	**Cooking Security**
Who	Joshua Timberman
When and Where	10/24/2009 at Community SANS Boulder
Summary	Configuration management for Security using Chef.

Title	**Chef 101**
Who	Joshua Timberman

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

When and Where	07/07/2010 via O'Reilly OSCON Preview Webcast
Summary	Introduction to Chef and preview of OSCON 2010 Chef Tutorial

Title	**Automatiser son infrastructure avec Chef**
Who	Olivier Gutknecht
When and Where	11/28/2009 at Paris BlockCamp
Summary	A short (mostly improvised) intro to Chef, in french.

Title	**Automated Infrastructure is on the Menu with Chef**
Who	Joshua Timberman
When and Where	07/20/2010 at OSCON 2010
Summary	3 hour tutorial, with a chef-repo

Title	**Chef in the Cloud**
Who	Joshua Timberman
When and Where	09/27/2010 at Denver/Boulder Cloud Computing Group
Summary	Combination Chef 101 + Automating EC2 with Chef

Title	**Understanding LWRP Development**
Who	Joshua Timberman
When and Where	11/02/2010 at November Chef user meeting in Seattle
Summary	Overview of how LWRPs work and ways to approach writing them

Title	**Introduction to "Chef" framework**
Who	Wojciech Wn?trzak
When and Where	03/12/2010 at Silesian Ruby User Group
Summary	Short introduction in polish language
Slides	Introduction to "Chef" framework

Title	**What Big Data Folks Need to Know About DevOps**
Who	Matt Ray & Flip Kromer
When and Where	01/29/2011 at Data Day Austin

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Summary	Introduction to Devops, Chef and the work being done to automate deploying Hadoop with Cluster Chef.
Slides	What Big Data Folks Need to Know About DevOps

Title	**SCALE: Automated Deployment of OpenStack with Chef framework**
Who	Matt Ray
When and Where	02/26/2011 at Southern California Linux Expo
Summary	Introduction to OpenStack, Chef and the work being done to automate deploying OpenStack.
Slides	SCALE 2011 Deploying OpenStack with Chef

Title	**TXLF: Automated Deployment of OpenStack with Chef framework**
Who	Matt Ray
When and Where	04/02/2011 at Texas Linux Fest
Summary	Introduction to OpenStack, Chef and the work being done to automate deploying OpenStack. Updated from SCALE talk with latest on Crowbar.
Slides	TXLF: Automated Deployment of OpenStack with Chef framework

Title	**Bay Area Chef Users Meetup: Chef 0.10 Overview**
Who	Matt Ray
When and Where	04/26/2011 at Bay Area Chef Users Meetup
Summary	The latest on what is available in the Chef 0.10 release
Slides	Chef 0.10 Overview

Title	**OpenStack Design Summit: Chef Cookbooks for Deploying OpenStack**
Who	Matt Ray
When and Where	04/27/2011 at OpenStack Design Summit
Summary	Presented OpenStack blueprint for using Chef to automate deploying OpenStack.
Slides	Deploying OpenStack with Chef

Title	**Chef, Devops, and You**
Who	Bryan W. Berry
When and Where	11/18/2011 at Food and Agriculture Organization of the United Nations

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Summary	Introduction and tutorial for Chef. Aimed at Developers and Sysadmins looking to write cookbooks. This presentation borrows heavily from Joshua Timberman's Chef 101.
Slides	Chef, Devops, and You

Title	**learnchef**
Who	Warwick Poole
When and Where	11/18/2011 at Harvest HQ
Summary	Introduction to Chef. Lots of awesome diagrams
Slides	Introduction to Chef

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

AMI List for Developers

Overview

This is a list of AMIs for developers, but these are not supported by Opscode. These instances can be useful when developing Chef cookbooks, allowing you to test your cookbook changes on other platforms without having to build a Virtual Machine image. They can also be useful if you are having issues with a particular AMI, to see if the issue occurs on other AMIs as well.

Amazon Machine Images



An Amazon Machine Image (AMI) is a special type of pre-configured operating system and virtual application software which is used to create a virtual machine within the Amazon Elastic Compute Cloud (EC2). It serves as the basic unit of deployment for services delivered using EC2.

These do not have chef-client pre-installed and are meant to be used with the knife-ec2 plugin.

- If you have not used the knife-ec2 plugin before, there is an EC2 Bootstrap Fast Start Guide that can help get you started.
- The bootstrap file listed is the one most likely to work with this AMI, if it does not work or there is not one listed you can try creating a Custom Knife Bootstrap Script.

These are collected and maintained by the community and their compatibility with Chef is not guaranteed.

AMI List



Need Amazon Linux?
There are no AMIs on this list for Amazon Linux. You can find them on Amazon's website. Customers tend to have the most success when using them with the fedora13-gems bootstrap.

AMI Name	OS / Distro / Release	Login Name	Source	Arch	Bootstrap File	Notes
ami-3962a950	Ubuntu 11.10 Oneiric (instance-store)	ubuntu	Alestic	32-bit	ubuntu10.04-gems	You can find more 11.10 AMIs on Canonical's site: https://uec-images.ubuntu.com/releases/11.10/release/
ami-c15994a8	Ubuntu 11.04 Natty (instance-store)	ubuntu	Alestic	32-bit	ubuntu10.04-gems	You can find more 11.04 AMIs on Canonical's site: https://uec-images.ubuntu.com/releases/11.04/release/
ami-9930fdf0	Ubuntu 10.10 Maverick (instance-store)	ubuntu	Alestic	32-bit	ubuntu10.04-gems	You can find more 10.10 AMIs on Canonical's site: http://uec-images.ubuntu.com/releases/10.10/release/
ami-6936fb00	Ubuntu 10.04 LTS Lucid (instance-store)	ubuntu	Alestic	32-bit	ubuntu10.04-gems or ubuntu10.04-apt	You can find more 10.04 AMIs on Canonical's site: http://uec-images.ubuntu.com/releases/10.04/release/
ami-0e7a8267	Ubuntu 8.04 LTS Hardy (instance-store)	ubuntu	Alestic	32-bit		
ami-2d8e4c44	RHEL-5.5 (ebs)			i386		
ami-a95390c0	RHEL-5.5 (ebs)	root		x86_64		This AMI requires a large instance or greater
ami-8d8f4de4	RHEL-5.6 (ebs)	root		i386		

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

ami-8f4083e6	RHEL-5.6 (ebs)	root		x86_64		This AMI requires a large instance or greater
ami-77a2601e	RHEL-5.7 (ebs)	root		i386		
ami-0f0bc866	RHEL-5.7 (ebs)	root		x86_64		This AMI requires a large instance or greater
ami-5550933c	RHEL-6.0 (ebs)	root		i386		
ami-6f63a006	RHEL-6.0 (ebs)	root		x86_64		This AMI requires a large instance or greater
ami-3ddb1954	RHEL-6.1 (ebs)	root		i386	fedora13-gems or centos5-gems	
ami-31d41658	RHEL-6.1 (ebs)	root		x86_64	fedora13-gems or centos5-gems	This AMI requires a large instance or greater
ami-43c2032a	Centos 6.0 (instance-store)	root	Rightscale	32-bit	centos5-gems	https://my.rightscale.com/clouds/ec2/1/images/565694
ami-3fe42456	Centos 5.6 (instance-store)	root	Rightscale	32-bit	centos5-gems	https://my.rightscale.com/clouds/ec2/1/images/585208
ami-f8b35e91	CentoS 5.4 (instance-store)	root	Rightscale	32-bit	centos5-gems	
ami-c041bda9	Debian 6.0.1 Squeeze (instance-store)	root	ec2debian list	32-bit	ubuntu10.04-gems or ubuntu10.04-apt	
ami-dcf615b5	Debian 5.0.3 Lenny (instance-store)	root	Alestic via AMI Catalog	32-bit		

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Community Events

#ChefConf 2012

May 15-17, 2012
SF Airport Marriott Waterfront
Burlingame, California

Conferences & Events Opscode is Attending

Upcoming Opscode events & conferences on Lanyrd

Opscode Official Events

Opscode Chef Fundamentals Training - Los Angeles
Opscode Chef Fundamentals Training - NYC
#ChefConf 2012

OPSCODE
RULE THE CLOUD

See the Opscode Newsroom for more information on upcoming Opscode Events



Food Fight is a bi-weekly podcast for the Chef community.

We bring together the smartest people in the Chef community and the broader DevOps world to discuss the thorniest issues in system administration.

Food Fight is hosted by Community Member Bryan Berry and Opscode Teammate Matt Ray.

Meetup.com Events

Events - Seattle-Opscode-Chef-Meetup-Group RSS

(Events - Seattle-Opscode-Chef-Meetup-Group)
Seattle DevOps Meetup
Events - San-Francisco-DevOps RSS

(Events - San-Francisco-DevOps)
Events - nycdevops RSS
(Events - nycdevops)
Merrill Ferguson and Qing Shou - Meetup.com Architecture

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Events - Seattle DevOps 

(Events - Seattle DevOps)

Opscode Community Summit

Inaugural event, Held on November 29-30, 2011 in Seattle

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

ChefConf



#ChefConf 2012 Call for Proposals

A few days ago, Chef turned 3 years old…

Our community has grown to over 15,000 users, and there are over 500 developers and 100 organizations contributing to and expanding Chef.

In short… we're ready for our own conference!

In a few days we'll be officially announcing #ChefConf 2012, our inaugural community conference. #ChefConf will be held May 15-17th in Burlingame, California. We're really excited about what we're putting together.

Ahead of the formal announcement, we're opening the Call for Proposals now.

This survey is powered by SurveyGizmo's online survey software. Please take my survey now

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Opscode Community Summit 1

The Opscode Community Summit was the inaugural gathering of people passionate about helping Opscode's community, projects and company succeed, and who were willing and able to participate.

Over 110 people participated in 40 sessions over two days!



Attendee Experiences & Reflections

- Rob Hirschfeld - Opscode Summit Recap – taking Chef & DevOps to a whole new level
- Sean Escriva - Day One
- Sean Escriva - Day Two
- Seth Falcon's photos

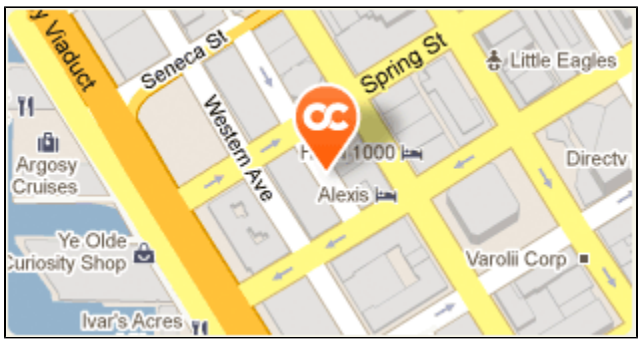
Commemorative T-shirt Ideas

Session Documentation

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Location:

1008 Western Ave on the First Floor
Seattle, WA 98104



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Commemorative T-shirt Ideas

Ideas

Baking Infrastructure (Sean P. Kane)

Front:
Opscode Chef
Inaugural Community Summit
Seattle, WA 2011

Back:
Developing Recipes
Compiling Cookbooks
Baking Infrastructure

Other

Sticky Note agenda

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Participants

- **Aaron Bento, Infogroup**

- **AJ Christensen**

- **Alex Howells, HP Cloud Services**
  Website: http://www.hpcloud.com
  Blog: http://h30529.www3.hp.com/t5/HP-Scaling-the-Cloud-Blog/bg-p/cloudBlog
  Twitter Username: **nixgeek**
  IRC nick: **z**

- **Andrea Campi, ZephirWorks**
  Website: http://zephirworks.com
  Blog: http://blog.zephirworks.com
  Twitter Username: **andreacampi**
  IRC nick: **andreacampi**

- **Andrea Carlo Granata, ZephirWorks**
  Website: http://zephirworks.com
  Blog: http://blog.zephirworks.com
  Twitter Username: **andreacgranata**
  IRC nick: **andreacgranata**

- **Brandon Adams, PivotLink Corp.**
  Website: http://pivotlink.com

- **Brett Weaver, Intuit**
  Website: http://intuit.com

- **Bryan McLellan, Opscode**
  Website: http://www.opscode.com
  Blog: http://blog.loftninjas.org
  Twitter Username: **btmspox**
  IRC nick: **btm**

- **Carl Perry, New Dream Network**
  Website: dreamhost.com
  Blog: blog.dreamhost.com
  Twitter Username: **edolnx**
  IRC nick: **carlp**

- **Chaoran Xie**

- **Charles Duffy, Tippr, Inc.**
  IRC nick: **nDuff**

- **Charles Johnson, SolutionSet**
  Website: http://www.solutionset.com
  IRC nick: **chip**

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- **chris chalfant, Cycle Computing LLC**
  Website: http://www.cyclecomputing.com
  Blog: http://blog.cyclecomputing.com/
  Twitter Username: **@chalfant**

- **Christian Paredes, Seattle Biomedical Research Institute**
  Website: http://seattlebiomed.org
  Blog: http://redbluemagenta.com
  Twitter Username: **redbluemagenta**
  IRC nick: **cparedes**

- **Cuong Nghiem**
  Website: https://github.com/HeSYINUvSBZfxqA

- **Damon Edwards, DTO Solutions**
  Website: http://dtosolutions.com
  Blog: http://dev2ops.org
  Twitter Username: **damonedwards**

- **Dan Prince, Rackspace**
  Website: http://www.rackspace.com
  IRC nick: **dprince**

- **Daniel DeLeo, Opscode**
  Website: http://opscode.com
  Twitter Username: **kallistec**
  IRC nick: **kallistec**

- **David Bamberger, Infogroup**

- **Dmitry Suslov, Grid Dynamics**
  Website: http://www.griddynamics.com/

- **Dr Nic Williams, Engine Yard**
  Website: http://drnicwilliams.com
  Twitter Username: **drnic**

- **Dragos Manolescu, HP Cloud Services**
  Website: http://www.hpcloud.com/
  Blog: http://micro-workflow.com/

- **Eric Wolfe, Marshall University**
  Website: http://opensource.marshall.edu
  Twitter Username: **atomic_penguin**
  IRC nick: **atomic-penguin**

- **Eric Heydrick, Webtrends**
  IRC nick: **heydrick**

- **Ezra Zygmuntowicz, VMWare**
  Website: http://cloudfoundry.org
  Blog: http://brainspl.at

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Twitter Username: **ezmobius**
IRC nick: **ezmobius**

- **Fletcher Nichol**
  Website: http://github.com/fnichol
  Twitter Username: **fnichol**
  IRC nick: **fnichol**

- **Florian Drescher, Cloudtrainings**

- **Gilles Devaux**

- **Grant Rodgers, Urbanspoon**
  Twitter Username: **grantr**
  IRC nick: **grantr**

- **Greg Althaus, Dell**
  Twitter Username: **@galthaus**

- **Greg Albrecht, Splunk**
  Website: http://www.splunk.com/
  Blog: http://helladata.com/
  Twitter Username: **ampledata**
  IRC nick: **ampledata**

- **Harley Alaniz, Lookout Mobile Security**
  Website: http://www.mylookout.com
  Blog: http://blog.mylookout.com/
  Twitter Username: **thehar**
  IRC nick: **thehar**

- **Hernan Alvarez, HP Cloud Services**
  Website: http://www.hpcloud.com
  Blog: http://h30529.www3.hp.com/t5/HP-Scaling-the-Cloud-Blog/bg-p/cloudBlog

---

- **Ian Meyer, Admeld**
  Twitter Username: **ianmeyer**
  IRC nick: **imeyer**

- **James Casey, Simple**
  Website: https://simple.com
  Twitter Username: **jamesc_000**

- **James Sulinski, AegisCo**
  Website: http://www.aegisco.com
  Blog: http://blog.aegisco.com
  Twitter Username: **jsulinski**
  IRC nick: **bemu**

- **Jason Williams**

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- **Jason Cook, Fastly**
  Website: http://fastly.com
  Blog: http://twitter.com/macros
  Twitter Username: **macros**
  IRC nick: **macros**

- **Jason Rohwedder**

- **Jason Ruiz, BIA**

- **Jeremy Bingham, Kos Media, LLC**
  Twitter Username: **captain_tenille**
  IRC nick: **c_t/ct**

- **Jesse Nelson**
  Website: http://f00bar.com
  Blog: http://f00bar.com
  Twitter Username: **spheromak**
  IRC nick: **spheromak**

- **Jesse Robbins, Opscode**
  Website: http://www.opscode.com
  Blog: http://www.opscode.com/blog/
  Twitter Username: **@jesserobbins**
  IRC nick: **jesserobbins**

- **Jim Hopp, Lookout**
  Website: http://www.mylookout.com

- **Joe Willims, Boundary**
  Website: http://Boundary.com

- **John Nestor, Hewlett-Packard Cloud Services**

- **John Willis,**
  Website: http://dtosolutions.com
  Blog: http://dtosolutions.com
  Twitter Username: **botchagalupe**
  IRC nick: **botchagalupe**

- **Jon Lenzer**

- **Joseph Williams, Boundary**
  Website: http://boundary.com
  Twitter Username: **williamsjoe**

- **Joshua Timberman, Opscode**
  Website: http://jtimberman.housepub.org
  Twitter Username: **jtimberman**

- **Justin Kolberg, Sonian Inc.**

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Twitter Username: **amdprophet**

- **Kate Leroux, Urbanspoon**
  Website: http://urbanspoon.com
  Blog: http://mynameiskate.com
  Twitter Username: **kateler**

- **KC Braunschweig, Edmunds.com**
  IRC nick: **kc_edm**

- **Kurt Yoder, CACI**

- **Kyle Burckhard, Marketfish, Inc.**
  Twitter Username: **kyleburckhard**
  IRC nick: **kyleb**

- **Kyle Bader, New Dream Network**
  Website: dreamhost.com
  Blog: blog.dreamhost.com
  Twitter Username: **kyle_bader**
  IRC nick: **kbader**

- **Ledio Ago, Splunk**
  Website: http://splunk.com/

- **Lunch Sponsor Darrin Eden / Heavy Water**
  Website: http://www.heavywater.ca
  Blog: http://www.heavywater.ca

- **Mandi Walls, Admeld**
  Website: http://linuxchick.org
  Blog: http://linuxchick.org
  Twitter Username: **lnxchk**
  IRC nick: **lnxchk**

- **Mathieu Sauve-Frankel, Qualcomm Japan Inc.**
  Website: http://www.qualcomm.com
  Blog: http://kisoku.net
  Twitter Username: **kisoku**
  IRC nick: **msf**

- **Matt Ray, Opscode**
  Website: http://community.opscode.com
  Blog: http://leastresistance.net
  Twitter Username: **mattray**
  IRC nick: **mattray**

- **Matt Olson, Kavi Corporation**
  Website: http://www.kavi.com/

- **Max Martynov, Grid Dynamics**
  Website: http://www.griddynamics.com/

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- **Michael Leinartas**
  Twitter Username: **mleinart**

- **Michael Cook, WhitePages**

- **Mike Tupitsyn**

- **Mitchell Hashimoto, Kiip**
  Twitter Username: **mitchellh**
  IRC nick: **mitchellh**

- **Nathaniel Eliot, InfoChimps, Inc.**
  Website: http://www.infochimps.com
  Blog: http://www.infochimps.com/blog
  Twitter Username: **temujin9**

- **Noah Kantrowitz**

- **Paul DALE, Elemica**
  Website: http://www.elemica.com
  Twitter Username: **_pauldale_**
  IRC nick: **pauldale**

- **Paul Lappas, Piston Cloud Computing**
  Website: http://www.pistoncloud.com

- **Paul Morton, BIA**
  Twitter Username: **mortonpe**

- **Pavel Snagovsky, LimeLIght**

- **Pete Cheslock, Sonian Inc.**
  Website: http://www.sonian.com
  Twitter Username: **petecheslock**

- **Peter Struijk, Urbanspoon**
  Website: http://www.urbanspoon.com

- **Peter Crossley, Webtrends**
  Website: http://www.webtrends.com
  Twitter Username: **pcross616**
  IRC nick: **pcross616**

- **Peter Norton, Knewton inc**
  Website: http://knewton.com

- **Phil Austin, BIA**

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- **Richard Pelavin, Reactor8**

- **Rob Hirschfeld, Dell, Inc.**
  Website: http://dell.com/cloud
  Blog: http://robhirschfeld.com
  Twitter Username: **zehicle**

- **Robert Berger, Runa, Inc**
  Website: http://www.runa.com
  Blog: http://blog.ibd.com
  Twitter Username: **rberger**
  IRC nick: **rberger**

- **Sam Cooper, Sonian Inc.**
  Twitter Username: **sandfish8**

- **sarah novotny**

- **Scott Likens, Engine Yard**
  Website: http://www.engineyard.com
  Blog: http://likens.us

- **Sean Kane, Kavi Corporation**
  Website: http://www.kavi.com/

- **Sean Porter, Sonian**
  Website: http://sonian.com
  Blog: http://portertech.ca
  Twitter Username: **portertech**
  IRC nick: **portertech**

- **Sean Escriva, Heavy Water Software Inc.**
  Website: http://hw-ops.com

- **Sean OMeara**
  Twitter Username: **someara**
  IRC nick: **siezer**

- **Sean Horn, Taleo, Inc.**
  Website: http://taleo.com

- **Seth Chisamore, Opscode, Inc.**
  Twitter Username: **schisamo**
  IRC nick: **schisamo**

- **Soo Choi, Rackspace**
  Website: http://www.rackspace.com/cloud/private_edition/
  Twitter Username: **soochoitweets**

- **Stan Kitsis, Grid Dynamics**

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- **Stephen Nelson-Smith**

- **Stephen Lum**

- **Steve Jang, Hulu**

- **Steven Danna, Opscode, Inc**
  Twitter Username: **SteveDanna**
  IRC nick: **ssd7**

- **Tim Dysinger**
  Twitter Username: **dysinger**
  IRC nick: **dysinger**

- **Todd Bushnell, Toor Security Consulting**
  Twitter Username: **toddmichael**
  IRC nick: **tmb33**

- **Tom Thomas, Opscode**
  Website: http://www.opscode.com
  Blog: http://www.opscode.com/blog/
  Twitter Username: **@tomasczt**
  IRC nick: **tomasczt**

- **Tommy Bishop, Intuit**
  Twitter Username: **thbishop**
  IRC nick: **thbishop**

- **Toufic Boubez, Metafor Software**
  Website: http://metaforsoftware.com
  Twitter Username: **@tboubez**

- **Vish Ishaya, Rackspace**
  Twitter Username: **vish**
  IRC nick: **vishy**

- **William Arnold, Oracle**

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Sessions



The Opscode Community Summit Sessions

Day 1

Day 2

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Closing Session - Decisions, Follow-Ups, and Next Steps

Session Title: Closing Session - Decisions, Follow-Ups, and Next Steps

Convener: Jesse Robbins

Participants: Everyone!

Summary of Decisions, Follow-Ups, and Next Steps:

Ruby & Hacking Chef - Greg / Kurt

- Help others learn

Short Lived Tactical Changes - Andrew

- Ask chef server to tell us delta
- Converge faster / on demand

Cluster Orchestration - Nathan

- Make cookbooks available public
- Clean up documentation

Managed Nodes - Matt Ray

- Discussion on maillist
- Product management follow up on requirements/plan

Crowbar - Rob Hirschfeld

- Doing a meetup to demo crowbar after the summit
- More documentation

Databags - Carl

- Fixing up the encryption
- Being better about handling encryption keys
- Write it up on the wiki

Ticket Triage - Bryan

- Everyone should get on the maillist

Training - Josh Timberman

- Migrate presentations from Keynote to Org|Markdown

Multi Developer Workflows - Erin

- Sharing with his internal team

Erlang Chef Server - Chris Brown

- Write docs on operating the erlang server

Creating a Virtual Hackday - Rob Berger

- Organize a date to hack

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook Style - Micheal

- Write a summary
- Work on a guide document

Cluster Coordination - Peter

- Make a demo

Running Chef Server in Cloud - Andrea 

- none really

Chef Documentation - Steven

- Take feedback and work an outline for chef developer documentation & send to the list

Big Data Clusters with Chef - Eric

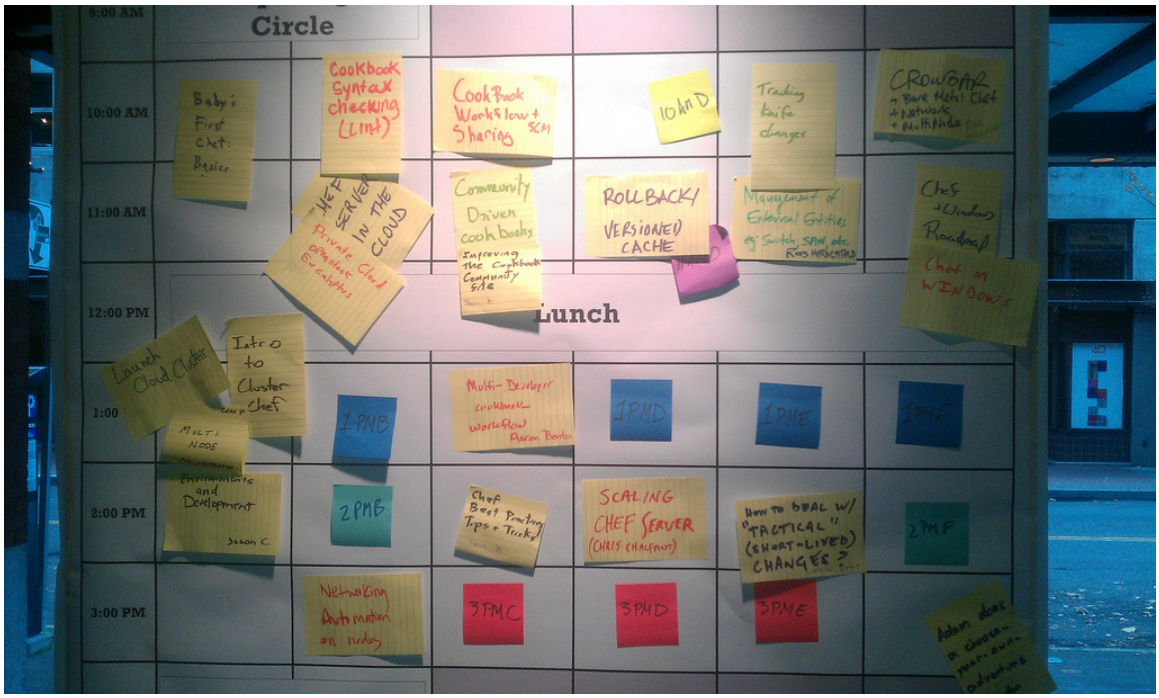
- Look at the public cookbooks
- Contribute Voldemort cookbook work back to the community

Monitoring - Sean Porter

- Document and show how to use chef to setup monitoring

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Day 1



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Baby's First Chef

Session Title: Baby's First Chef

Convener:

Participants:

Summary of discussions: What will we do now? What needs to happen next?

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef & Windows

Session Title:

Convener:

- Paul Morton
- Tim Smith
- Pete Chesloch

Participants:

- Amy Sutedja
- Phil Austin
- Justin Kolberg
- Ash
- Charles Johnson
- Chavran Xie
- Jon Lenzer
- Florian D.
- Seth Chisamore
- John Keiser

Summary of discussions:

Chef installation: the Chef Installer MSI is a highly desired feature that will land soon. People would like to provide base config information at install time: certificate, validator name, etc.

Needed Windows features: we talked for a bit about the biggest pain points, and the things people do most often that could use automation:

- Zipfiles
- Windows patch installs
- Cron
- Domain Switching / Joining, Moving to an Organizational Unit
- DNS Record Updates
- User and Group Memberships on domains
- Power Options, (Control Panel?)
- MSI Installs
- File Shares: easy to do with "net share", a resource may be worthwhile with security attributes though
- Security Auditing: group was like "eh, maybe"

Security mixin: we covered the upcoming rights attributes for files and directories. The fact that :write implies :read seemed right to people. There was a desire to append rights to a file (and not remove them) was there. The ability to clear or remove rights from a file was also desired.

What will we do now? What needs to happen next?

The above features need to get prioritized and written by people.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Best Practices - Tips & Tricks

Session Title: Chef Best Practices - Tips & Tricks

Convener: Sean P. Kane

Participants:
Kurt Yoder

Summary of discussions:

Talked about:

- The creation and use of LWRPs
- Creating Library Cookbook or Ruby Gem
    - The good thing about Ruby is that it is easy to monkey patch. The bad thing about Ruby is that it is easy to monkey patch.
- Chef event/log handler (irc/campfire)
- knife node show -Fj (export whole node to json)
- value_for_platform() - dist and versions
- Continuous Integration
    - pennyworth - https://github.com/heavywater/pennyworth
    - chef-jenkins - https://github.com/fnichol/chef-jenkins
- cookbook style guide would be helpful

What will we do now? What needs to happen next?

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Documentation

Session Title:

Convener: Steven Danna

Participants:

Steven Danna
Dan Deleo
Bryan McLellan
Marc Paradise
Joshua Timberman
Kate Leroux

Summary of discussions:

- Everyone agrees docs have improved over the last year
- Wiki is good in some places, bad in some places
- Needed Documentation
    - Shef!
    - Developer Docs/documentation of internal chef api's
      This affects are ability to make changes to internal APIs, but people are already using them anyway.
    - Ruby DSL functions (related to internal api)
      Examples that include how to translate knife commands into a recipe/knife exec
      script
- List of easy to complete documentation tasks to encourage participation
  Bryan has tried this before with CHEF bugs and didn't have much luck. But maybe things are different now since Opscode has been more responsive to tickets/pull requests in general.
- The long resources page can be intimidating
- Quick Start needs work
    - Currently feels long
    - Lots of Branching
    - Getting started with Shef instead?

What will we do now? What needs to happen next?

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Server In the Cloud

Session Title: Chef Server In the Cloud

Conveners: Andrea Campi, Kyle Burckhand, Jim Hopp

Participants: Andrea Campi, Kyle Burckhand, Jim Hopp, Dan Prince, KC Brunschweng, Grant Rodgers, Kate Leroux, Scott Mccross, Steve Lum, Brandon Adams, Eric Heydrick, Harley Alaniz, Dragos Manolesca

Summary of discussions:

- Spinning up Chef nodes with OpenVPN
- Spinning up new nodes on a cloud platform / autoscaling
- Zephirworks seems to be only ones running chef-server on a PaaS
- Continuous Testing of Chef Cookbooks
    - Local development with Vagrant
    - Jenkins fires up a new custer for every commit on master, then runs a set of tests with cucumber
- Private Repositories for ruby gems, rpms,
    - MANY OF US DO RUN PRIVATE REPOS
- Autoscaling + Chef: Spins up instances on demand
    - Scalr.com - autoscaling by time, load etc

What will we do now? What needs to happen next?

1. Automate the Cloud!!!

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cluster Orchestration

Cluster Orchestration:

Convener: Nathaniel Eliot, Peter Norton, Carl Perry

Participants:

Kyle Bader
Kurt Yoder
Sean P. Kane

Summary of discussions:

- Nathaniel Eliot: Overview of what cluster_chef provides on top of chef
- Discussion on tools for orchestrating large clusters
    - cluster_chef & rundeck
    - dominodes: databag-based locking mechanism (security issue: nodes require write on all databags)
    - chef-solo + push mechanism
    - crowbar: has its own internal orchestration
    - ruote: ruby orchestration library
    - nanite: messaging bus
    - rightscale: has a tool ("remote recipe"?) for invoking single recipes on a target node
- Monitoring tools
    - zabbix - monitoring
    - sensu - monitoring framework from Sonia, using Rabbit MQ
    - nagios - perpetual monitoring standard

- PeterN: use case of launching a cluster, use case for orchestration
    - Standard web app, discussing options for orchestration. Rundeck will be used with cluster_chef
    - Standard stack of rabbitmq, web apps, and nginx needing to be completed in order and asap
    - Learned: rightscale provides a platform that offers orchestration. They are adding chef support (maybe done already? I'm not clear on this)
    - Also there is a non-server-based discovery protocol that can be used for orchestration called dominodes (I'm not sure how this works - in the end I see things like https://github.com/websterclay/chef-dominodes which uses a databag as the lock? I guess dominodes facilitates intra-node communication).
    - Desired use cases:
        - Spin lock (wait_for_foo "foo_name" so that only one node is ever doing "foo_name" in a cluster)
        - event coordination (wait_until_foo "foo_happened" so that when a resource becomes available, nodes that need it can use it)
        - Remote execution (run_then_activate "foo_nodes" where after a node runs an action, either it directly, or through a chef server, can invoke a resource or a chef run on the nodes that comprise "foo_nodes").
    - Takeaway: chef needs to provide orchestration syntax. There are at least 3 use cases. The use cases may be orthogonal, but some API decisions need to be made and these need to be approached. It would be preferred if the chef server could do this instead of just cooperating with an outside infrastructure (in my opinion)

Other stack tools mentioned:

- elastic search
- hadoop
- hbase
- goliath
- unicorn
- apistack
- mongo
- rails
- noah - zookeeper - doozer (doozerd?)
- chef plugin for Run Deck
- mcollective

What will we do now? What needs to happen next?

- Desired feature: ability to run a single recipe (+ dependencies?) on a node, without fouling up run-list or node data

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Community Driven Cookbooks & Improving the Community Site

Session Title:

Conveners: Gilles Devaux & Sean P. Kane

Participants:
Joshua Timberman
Nathaniel Eliot
Sean Horn

Also see: Cookbook Workflow Sharing Summary of discussions:

- One cookbook == one SCM project (one cookbook per repo)
- Less communication channels. One community => one place
- Quality of shared cookbooks matters
    - What metrics?
    - Ticker graph for activity (downloads / version push)
    - SCM introspection seems problematic
- Differentiate Library cookbooks vs. more specifics
    - Guidelines to create library cookbook
    - Noah suggested a way to use library cookbooks while still introducing specific customization (ability to substitute templates)

What will we do now? What needs to happen next?

- (jtimberman) - Split opscode/cookbooks into separate repositories

Opscode Attendees

Barry Steinglass	Community Site Manager
Noah Kantrowitz	Full Time Web Dev
Joshua Timberman	Cookbook Dev

Community Site Should Provide:

- Pointers to Chef Tools (plugins/external)
- Pointers to Chef Repos (apt/yum)
- List of People in Community (profiles)
    - An Opscode/Chef community directory would be great.
- Pointers to Cookbooks/Code (also ticketing & collaboration)
- Discussion (chef/cookbook ideas/collaboration and ticketing)
- Official and Collaborative Documentation/Training (documentation/workflows/best practices/how-tos)

Recommended Tools

- Librarian - https://github.com/applicationsonline/librarian

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook Syntax Checking (Linting)

Session Title:

Convener:

Tim Dysinger

Participants:

Sean Escriva
Jesse Nelson
Nathaniel Eliot

Summary of discussions:

- Round trip to chef server to test cookbooks is expensive
- It is possible to push broken code to the server
- Would be nice to check things for correctness before upload
    - Is our Ruby code valid?
    - Do our resource blocks & attributes validate against their specs?
    - Basic Json syntax checking
    - Do we have at least default attributes for everything?
- Would be nice to be able to run a partial chef run (one cookbook or even one resource)
- Lint checking should be on by default with a flag to turn off
- Don't throw away all edits to a node when the json isn't quit right

What will we do now? What needs to happen next?

Let's create a basic lint check & continually improve it

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook Workflow Sharing

Session Title:

*Conveners: Robert J. Berger & Sean P. Kane

Participants:

Also see: Community Driven Cookbooks & Improving the Community Site Summary of discussions: Notes from R. berger (Not complete, please add any you have)

The process of sharing and using is problematic
More generic way to use lwrp for tweaking cookbooks ala apache configs

- Community site
- Group/ mailing list per cookbook
- Domain area collaboration to create and update related cookbooks
- Feedback in a way that can also go to all the people involved
- Best practices
- Track view source code from github bitbucket etc
- Better search
- Cookbook maturity and other code and history
- Wish list tie into roadmap
- Get better way to download
- Librarian can be the backbone of a new workflow
- Share cookbook development environments along with cookbooks via Vagrant (Ex. freight-cooking)
- Namespace per cookbook
- Network diagram lineage
- Tighter integration with Jira
- Endorsed cookbooks
- Oficially supported cookbooks (by anyone)
- Allow way to tell maintainer of a correction
- Use tickets to track who is working on what new or old cookbook
- irc logs searchable mineable
- Virtual hack days to improve a cookbook at a time
- Better "human-readable" version scheme that allows local version changes that do not conflict with the upstream providers versioning.
    - I.E. Let providers use the first 3 values of a 4 value version number "i.e. 1.3.2.0" and allow end-users to increment the 4th number as much as needed to track there own internal changes, that are not re-shared upstream.

What will we do now? What needs to happen next?

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Crowbar

Session Title:

Convener:
Rob Hirschfeld (@zehicle)

Participants:
About 30+ people

Summary of discussions:

This was held as a pre-req for the Management of External Entitites session

Presented information about Crowbar including demo of install. This information is largely available (including videos) on the Crowbar github site: https://github.com/dellcloudedge/crowbar/wiki

What will we do now? What needs to happen next?

A second session was scheduled on Wednesday @ 9:30 to talk about our cookbook design pattern.

Picture!



(page ref: http://pic.twitter.com/sLzyR0sC)

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

How to apply "tactical" config changes?

How to apply "tactical" config changes?

Convener: Andrew Fort

Participants: Adam, Kyle, Sean Horn, Dave, Matt, Greg, Kate Leroux, apologies to others!

Summary of discussions:

Making tactical changes to a chef managed service.
Approaches;

- Using chef
- Using orchestration (fabric, capistrano, mcollective, noah, doozer, zookeeper, etc)

How does one keep things in their commit history? What about rapidly pushed changes?

Make the change via push then roll the commit anyhow, knowing it will safely integrate later.

Branching strategies for tactical change (hotfixes into master, etc)

Using partial chef runs to apply certain configuration only

Performing roll-forward provisioning via environments/role list changes (major changes)

Using environment names built from and tied to CI (e.g., commit-hash named, or "dev" for latest), so that CI passes can trigger staging updates and green-light revisions for being set to development.

Adam says in response to Andrew's question "Is there a better way than changing BGP routes to move traffic around?": Couple ways to redirect traffic to different segments of the infrastructure

- Case1: You have control of app. To get away from changing BGP routes: Push logic for draining app into app. This circles around dark launching.
- Case2: No control of app: Decouple deploy of app and configuration changes
- Case3: Traditional gates and phases: Log in here, do this, log in there, do that. Done

Do we want to do these things with Chef?

Rolling deployments - When 50% of systems get to a certain state, deploy the next. May move through environments, just add something to role to get to next state. Running on demand Chef runs to roll to a new state with different paths and daemons running there, then write it in a recipe, not roles and environments. Recipe grossness shows the grossness of the system environments.

Adam mentions Mark Burgess (author of cfengine and Principles of Network and System Administration) in question to "why the argument about pull vs push". Pull basically better for reasons of network segmentation, machines receiving timely updates when they appear, etc - but not always; push is almost always used for small changes.

That's the key issue of this session - how one might converge the ideas of chef managed configuration with that which needs to be pushed to machines very rapidly.

One final point Adam mentioned - using polling, if a machine can ask Chef "is there any changes for me" (either via a new TCP/HTTP request or a long-lived TCP connection to a polling-IO HTTP server serving chef), then one may be able to use pull for these types of tactical changes, keeping the workflow the same.

Closing the time delay is the issue: if it takes too long to do things "the right way", they will be done the pragmatic/expedient way.

What will we do now? What needs to happen next?

Consider implementing polling as a solution to reducing the time to making changes the right way.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Management of External Entitites

Session Title:

Convener:
Rob Hirschfeld, Greg Althaus, Matt Ray

Participants:
About 20+ people including Alex Howells (very vocal)
Kurt Yoder
Nathaniel Eliot

Summary of discussions:

External entities are nodes that act on something that the place the code is run does not have direct access to. Sort of like a proxied node.

Use cases:

- switch configuration
- Blade chassis configuration
- SAN config
- manual steps
- non-root control
- external authentication
- SaaS endpoints: AWS's security groups, ELB, Route 53, et. al., and similar offerings

Notes:

- Hack way: put agent into chroot or LXE container. More on this [chef:External Entity Hack and Design]
- Looking for a better way - needs an execution environment
- Challenge - external entities need network access (or serial) from limited locations/paths
- use case - an external database could be managed
- part of the problem is how do you model the state of these external entities? Their state is controlled by the item they proxy for!
- Cookbooks become like services (see [chef:Cookbook As Services Pattern])
- Changes could happen outside of the node's context - this is tricky to manage
- missing - we need a dependency graph for the switch, chassis, SAN etc
    - these are links, not nodes. there is a real dependency between these types of items
    - Crowbar will model them using roles, but that's a hack
- we (Dell) are putting a lot of information as extra attributes (generally roles)
- some things are complex, it helps to treat them as nodes
- Matt Ray will do a session on "Managed Nodes" requirements. He called external entities, managed nodes. Session: [chef:Managed Nodes Part 2]
- Can use by extending Actuate?
- Alex H felt that the Actuator pattern could do a lot of this
- These are not light weight services! The node cannot invoke items that we need to be configured by external entities.
- Directed graphs are needed but are hard to implement in databags

Overall, people saw a lot of uses for this functionality.

What will we do now? What needs to happen next?

Continues: Managed Nodes Part 2

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Rollback & Versioned Cache

Session Title: Rollback and Versioned Cache

Convener: Mathieu Sauve-Frankel

Participants:

- Mathieu Sauve-Frankel
- Adam Jacob
- Jesse Nelson
- Seth Falcon
- James Casey
- if you attended this please fill in your name, I'm sorry I forgot

Summary of discussions:

- discussed the feasability of rollback as a general whole, seemed to be consensus that it's really hard
- most of the time you don't need it
- "roll forward bro"
- a few participants agreed it would be nice to have naive rollback
- the use case discussed was "template update triggers a service reload which fails, let's try to restore the old template and notify again before throwing stack"
- discussed throwing all of /var/cache/chef under version control, ie. commit the entire state of the cache after each chef run
- discussion degenerated, msf trolls adam about CHEF-13
- bikeshed about NOOP ensues
- most people want dry-run so we can a text representation about what actions are about to occur and perhaps diffs of what is going to change in the filesystem. a "pre-flight" check.
- we discovered most of the people who want this don't run the chef-client as a daemon, usually resulting in much larger changesets if the client is run infrequently.

What will we do now? What needs to happen next?

have more arguments I suppose...

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Tracking Knife Changes

Session Title: 10AM - E - Tracking Knife Changes

Convener: Kurt Yoder

Participants: Kurt Yoder, Mandi Walls, Dan DeLeo, Christian Paredes, Peter Norton, Andrea Campi, Sean Horn
Steve Lum, Phil Austin, Amy Sutedja, Dave Bamberger

Summary of discussions:

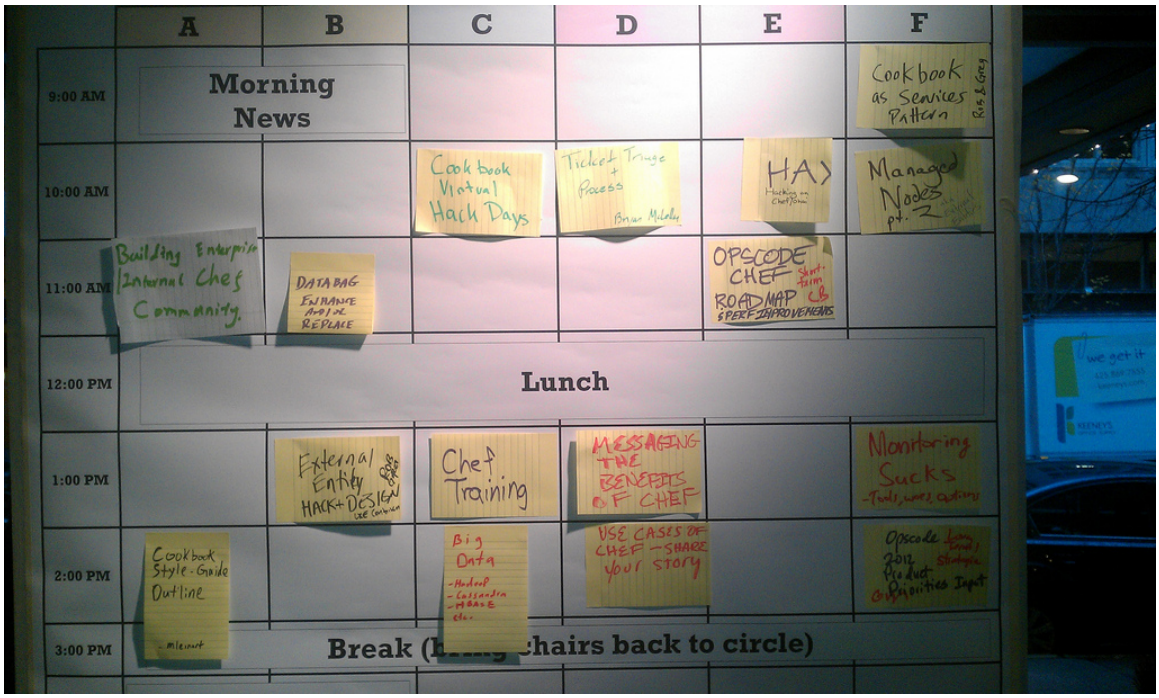
- Upcoming Work by Opscode (reporting)
- Tracking node instantiation
- Using existing hooks in Chef Server
- Restricting Operations by User (ACLS currently on Opscode Hosted Chef & Private Chef)
- Interface for viewing audit information / logs
- kerberos & other Authentication Layers
- Jenkins for Change Control
- API for submitting audit entries

What will we do now? What needs to happen next?

- Audit API on hosted Chef & Open Source Chef
- Simple, baked-in front-end for audit trail

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Day 2



 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Big Data

Session Title: Big Data

Convener: Eric Heydrick

Participants: Kyle Bader, Grant Rodgers, Kate Leroux, Nathaniel Eliot, Sean Horn

Summary of discussions:

We discussed challenges and solutions for managing big data clusters with Chef. Points covered included

- There are good cookbooks for managing Cassandra. https://github.com/riptano/chef/tree/master/cookbooks/cassandra is one of them
- Work on the voldemort cookbook will be shared.
- Cluster Chef has cookbooks for Hadoop but mostly specific to Cluster Chef
- Talked about storage on EC2. Some find ephemeral storage to work well. EBS + RAID0 also works well.
- EBS volumes can be attached w/ IAM restricted keys. Can also have the node make a call to a broker webservice that talks to the EC2 API.
- Cassandra Stress Test, part of the Cassandra distribution, can be used to benchmark performance
- Rhino is an ORM for hbase.
- Talked about monitoring. Ganglia in EC2 requires use of unicast and can be an issue when lots of nodes are in flux.
- Graphite + statsd is useful for trending. Also good reading: http://codeascraft.etsy.com/2011/02/15/measure-anything-measure-everything/

What will we do now? What needs to happen next?

- Investigate tools mentioned during the discussion
- Automate cluster deloyments

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Building Enterprise - Internal Chef Community

Session Title:

Convener:

Participants:

Summary of discussions: What will we do now? What needs to happen next?

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Chef Training

Session Title: Chef Training

Convener: Joshua Timberman

Participants: Approximately 12, attendee sheet was not filled out or collected

Summary of discussions:

This session was a grab bag of topics related to Opscode's Chef Fundamentals Training, and a few topics related to the Advanced Training.

Most of the participants were past the point where they needed fundamentals training themselves, but many wanted to have access to the materials to train others in their own organization.

The current Chef Fundamentals training materials are freely available under Opscode's Open Training program, but the downloads are PDFs, which aren't great for collaborating, or even viewing as a presenter. The source material these are generated from is Keynote, and not everyone who wants to use them has a Mac, let alone Keynote. Opscode's plans to release the training materials in a source format that makes it easier to collaborate and contribute to were well received. This is currently underway, and the plan is to make the materials source format Markdown so it can be easily exported to other formats, such as PDF, or Showoff. An open source repository on GitHub will be created along with tooling.

A fair amount of discussion happened around online training delivery strategies. While Opscode isn't currently offering any online classes, the screencast format is good. 15-30 minute modules with hands on supporting material could be produced fairly easily based on the existing training material.

One of the biggest areas for improvement is a Ruby tutorial, and this will go into the Fundamentals material.

The participants in the discussion didn't seem highly in favor of Chef user / student certification, though an assessment exam / quiz at the end of the fundamentals class might be beneficial for the "Hey look what I learned" aspect. Opscode is still working on plans for a Chef Trainer program which will be somewhat like business partnership for folks who wish to perform Chef Fundamentals Training.

What will we do now? What needs to happen next?

- Chef Fundamentals Training material update (new contents, better strutcure)
- Chef Trainer program
- Release source format on GitHub (requires rewriting in Markdown, plus revision from above)
- Update and make available Advanced Training course.
- *Long Term* Evaluation and selection of online course software/delivery

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook As Services Pattern

Session Title:

Convener:
Rob Hirschfeld

Participants:
Sean P. Kane
Wow...like 60 people

Summary of discussions:

Greg Althaus did session showing how Crowbar uses Roles to track application configuration information
Rob talked about the use case of Crowbar being to use Chef as an installer, needed to keep recipe logic more locked
idea: Crowbar is using Roles as something else, perhaps there are missing object types in Chef.

What will we do now? What needs to happen next?

Next: continue External Entity discussions. See External Entity Hack and Design

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook Style-Guide Outline

Session Title: Cookbook Style Guide Discussion

Convener:
Michael Leinartas

Participants:

- Joshua Timberman
- Noah Kantrowitz
- Eric G. Wolfe
- Mathieu Sauve-Frankel
- Michael Leinartas
- Fletcher Nichol
- Sean P. Kane
- ...and others

Summary of discussions:

The discussion touched on a number of common cookbook patterns, but didn't get to the point where the contents of a "style guide" were discussed or enumerated.

Discussion notes:

Misc criteria for usability of 'library-style' cookbooks

- Easily shareable
- Useful to others
- Environment aware
- LWRPs are very good
- Can make some reasonable assumptions
- Configuration items as attributes
- Ability to easily swap in new templates - usually a good idea

Plugins

- No way to do this right now, but not a very difficult implementation
- Ability to subclass built-in resources and other LWRPs in an LWRP
- Plugins as a way to create hook points/callbacks
- Cookbooks can be more generic and defer specific implementation to plugins (e.g. database replication strategy)
- Related: All resources should have pre/post hooks

@node vs passed in variables in templates?

- Not resolved, some contend that @node should never be used as it's dangerous and has potential for side-effects
- Passed in variables are a way for the cookbook author to define an api into the template

Miscellaneous points discussed

- Attribute references should be strings (node['attribute']) - all other methods are discouraged and deprecated
- Dividing line on what should and should not be an LWRP is fuzzy - this could use some fleshing out by someone
- IP addresses: first interface vs second?, cloud IPs or not? This needs some kindof helper
- Platform case statements are a repeated complaint, no good solution at this point
- Direction on how to properly architect a cookbook, so that internal data is stolred separately from core cookbook files that might be re-shared upstream.

What will we do now? What needs to happen next?

Opscode has begun work on a Cookbook Style Guide Draft

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook Style Guide Draft

General conventions

We follow Ruby community common practices for code formatting.

- http://www.caliban.org/ruby/rubyguide.shtml#style

Highlights:

- 2 spaces for code indentation
- Ruby blocks can be 'do..end' or {..}, use the former for multiline blocks, the latter for single line
- Do not omit parenthesis in method calls except when it is clearer to do so.

Root files

README.md

- Follows the README template
- Attributes that are set in the attributes files in the cookbook don't need to indicate default values in the documentation, as the user can look at the file for the defaults. Do indicate what calculated values are if in recipes.

metadata.rb

- Use the Ruby DSL, the JSON will be automatically generated on cookbook upload.
- For cookbooks that have several dependencies or support several platforms, use an each loop on an array of names.
- Declare dependencies explicitly when used, rather than assuming dependent cookbooks are available.
- Use semantic versioning, http://semver.org/

Recipes

- Follow good Ruby style practices above.

Use string expansion instead of concatenation

```
# do this:
"#{ruby_var} is cool"

# instead of:
ruby_var + " is cool"
```

- Use `Chef::Config[:file_cache_path]` as a temporary location for file downloads when necessary as it will always be present.
- Do not hard code values, it makes cookbooks inflexible and harder to share. Use attributes instead.
- Use libraries if recipes contain a lot of helper code.

Generating Data

- Generate data at the top of the recipe so it can be used later and doesn't clutter the resources.
- Setting attributes that require data from sources like other cookbook's attributes search or data bags should be done in recipes.
- Generate data for `template` resources in the recipe and pass it in

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

with the `variables` parameter.

Resource declaration

- Be explicit for clarity, but leave Chef's defaults if it's obvious what the values should be, for example default actions.

Need specific guidelines for core Chef resources.

Attributes

- Use 'default' to set attributes.
- Start with an attribute space named after the cookbook.
- Do not calculate or derive values in attributes files, use a recipe instead.
- Use strings for accessing node attributes.

Files

- Use with `cookbook_file` resources for binary data/files (gems, .tar.gz, etc).
- Use with `remote_directory` resources for distributing a set of like files.
- Use `files/default` file specificity most of the time.
- File specificity can be useful here, but we recommend against using host specific files, and instead use logic in the recipe as it provides more flexibility.
- Do not use `files` for distributing configuration files as it is inflexible.

Templates

- Use{{template}}resources for rendering configuration files.
- Use templates like views: contain little logic and do not use search or other data retrieval - do that in the recipe.

Other components

Definitions

- Only use definitions for very simple repeated resources that should be named and do not contain a lot of logic.
- We recommend against definitions, use LWRP (resources/providers) instead.

Resources/Providers

- Explicitly set the name attribute.
- Create a default action (not currently possible in the DSL yet, must be a new method)
- overload{{load_current_resource}}in the provider to make the provider idempotent
- Use{{new_resource.updated_by_last_action(true)}}and {{new_resource.updated_by_last_action?}}so notifications work.

Libraries

- Use for adding new methods to the Recipe DSL
- Create other helpers for providers.
- Create heavyweight resources/providers when LWRP is limiting in flexibility or namespacing.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook Virtual Hack Days

Session Title:

Convener:Robert J. Berger

Participants: Fletcher Nichol, Aaron Bento, Michael Leinartas, Shawn OMeara

Summary of discussions:

- Generally Individuals do just enough for their own environment, want to encourage people to work together to improve existing cookbooks or create new ones
- Could have some folks port to different platforms
- Coordinate with the authors of Opscode and other cookbooks to make sure we aren't doing redundant efforts
- Encourage people who have a personal cookbook but need help to get it in shape for public release
- Mix physical and virtual meetups

- Style guid will be helpful
- We could give feedback to styleguide
- Styleguide could be used to judge what cookbooks need updates and what to do
- Create a set of guidelines to decide which cookbooks to update
- Paramerterize / hooks to allow easy ways to use different package managers, from source
- Break out install and configure recipes
- Use LWRP and libraries to impove readability and extensibility
- Supporting more complicated configurations (master/slave, distributed)
- Proper splitting of recipes, LWRP, attributes, data bags

- Can start with mini tutorial from Opscode personnel or other domain or cookbook experts
- How do we do collaborative group efforts

- Start out with group web meeting
- Break up into pairs or small groups to work on specifics using collaborative remote tools or at local meet ups
- Way to allow people to ask for help / review

- Get domain expert who may not be a chef expert

What will we do now? What needs to happen next?

- Create a wiki/ forum thru Opscode
- Create a way to rendezvous with people who want to work on stuff on the wiki/forum
- Wiki page on Opscode that at least points to things
- Draw n community of specific domain experience
- Coordinate with chef and devops meetups
- Coordinate with Jesse and the person from Opscode who will be the Sheppard
- Tie in with contests

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Data Bag - Enhance and or Replace

Session Title: Cookbooks: Enhance and/or Replace

Convener: Carl Perry

Participants:
Kate Leroux
Kurt Yoder
James Casey
Mathieu Sauve-Frankel
Sean P. Kane

Summary of discussions:

The primary topic of discussion was limitations within the data bag system. The basis was there are several things that chef does not and should not support (eg: a full graph database system for tracking dependencies and interconnects for networking). External systems and application are much better suited for some specific needs, but they need a way to put that information into chef.

An idea was proposed to have an "link URL" in which the necessary information could be periodically fetched from an external system, cached for certain period of time, and then merged into the existing JSON structure. However, it was decided that a much simpler approach would be implement a form of deltas and opportunistic locking for all JSON transactions: data bags, roles, and node attributes.

Opportunistic Locking

Currently, all JSON documents stored are given a unique identifier by CouchDB. With the move to a SQL based backend, that ID field will need to be preserved. We discussed just using an incremental integer, but some discussions after made it sound like a last modified timestamp field would be better.

Once this last modified timestamp is in place, whenever a save is done to a JSON document (node attributes, role attributes, or data bag) the last modified timestamp in the PUT must match the last modified timestamp in the currently stored document. If there is a mismatch, then another PUT has occurred and the new contents are invalid. This will result in a failure. If they match, then the server will update the last modified timestamp in the PUT JSON document and then save to the data store.

It was suggested that this change can be implemented now, with little impact to existing tools and will provide a much greater level of data integrity.

JSON Deltas

In some cases, data written by external tools to chef JSON documents is meant as read-only for the chef client. In order to better support these, the idea of JSON deltas was proposed. The idea is to have a new API endpoint where only parts of the JSON document could be submitted. These would be in a dict containing three keys:

- **update** keys which have changes and would need to be replaced
- **create** keys which did not previously exists and would need to be created
- **delete** keys which did previously exist and need to be removed

Under normal conditions a failure of the above statements, such as attempting to update or delete a key which does not already exist, or creating a key which does already exist, would result in a failure. This would be the standard method for a save call within a recipe. However, the save at the end of a chef-client run would have the force flag enabled on the call. This would cause most errors to continue (deletion of a non-existent key is ignored, creation of a key which already exists acts as an update, updating a key which already exists acts as a create). The idea is that the delta API could ignore the last modified timestamp used for opportunistic locking.

Encrypted Data Bag Enhancements

There was also discussion about making encrypted data bags more friendly. The idea is to use the the existing chef client keys for the encrypted data bags as well. In order for this to be as secure as possible, the validation.pem method of creating a client needs to be changed slightly. The client key should be generated on the client and never sent to the server. Instead, a CSR should be sent, and the server should return the signed certificate. This also means fixing the node creation method where a client can submit their own public key to the server (may work in the platform, does not work on open source server).

Mathieu Sauve-Frankel will detail the server side public key escrow system here

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

What will we do now? What needs to happen next?

- Addition of Opportunistic Locking for all JSON objects within chef
- Addition of JSON Delta API for updating all JSON objects within chef
- Modification of chef-client to create private keys on server and send public key or CSR to server when using validation.pem node creation method
- Modification of chef server to allow creation of a node while providing a public key
- Creation of Encrypted Data Bag Public Key Escrow System

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

External Entity Hack and Design

Session Title:

Convener:
Rob Hirschfeld & Greg Althaus

Participants:
Matt Ray, William Arnold, Sean Horn

Summary of discussions:

Continued from 11 AM - E - Management of External Entitites

Dell's priorities on this Network -> SAN -> Chassis
First pass thoughts:

- cookbook: managed node manager
- recipe that sets up the manged node environment (LXE or chroot)
- generate set of chron scripts to run chef-client in the protected environment
- what about using the "full path installer"
- first pass, we are only talking about 3-8 managed nodes per system
- manage node manager that can run multiple of those
- want to be able to poke the managed node
- need ohaithere plug in that builds the node object w/ data that's specific to the managed nodes
- we envision a set of chained recipes (aka barclamps in Crowbar)
- need a disk system to play w/ DAS
- shared configs
- ohai needs to have no plugins (that would be confusing)
- each container would have its own IP address
- this helps w/ use on manage and remote invokation

What will we do now? What needs to happen next?

Greg is writing it - watch Crowbar.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

intro to ruby, and knife+ohai hacking

Session Title:

Convener: Kurt Yoder, Greg Albrecht
Participants:

Summary of discussions:

(there was a pen-written description of this that was turned in, I hope it still makes it into the wiki)
breaking code into libraries
applying unit tests against those libraries

- Always use local Error names (custom exception handlers)
- Ruby unit testing is very similar to Python unit testing
- Lack of intrudcutary documentation on Opscode wiki
- extend @node to more gracefully handle missing nested attributes
- should have a comprehensive document on @node capabilities

- Books:
  - Ruby Kaatas
  - The well grounded Rubyist

- test run : execute the .rb
- ruby-style-guide as an introduction to ruby
- abstracting libraries to makn them more testable
- chef definitions are depricated
- built-in documentatino
- monkey-patching chef patches without having to do a complete upgrade

What will we do now? What needs to happen next?

- beginner ruby documentation in Opscode wiki
- "Ruby on training Wheels"

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Managed Nodes Part 2

Session Title: Managed Nodes Part 2

Convener:
Matt Ray

Participants:
Rob Hirschfeld
Greg Althaus
George Moberly
Chris Brown
Kyle Bader
Carl Perry
Sean P. Kane
Nathaniel Eliot

Summary of discussions:

In order to extend the reach of Chef into managing nodes that cannot run the chef-client, Chef needs to be extended to support the concept of a "Managed Node". Managed Nodes may be network devices (switches, firewalls, load balancers, etc.), network storage, databases and any other device that we may configure via an API or other tools, but we want to treat it as a Node. The Managed Node needs to behave identically to Nodes in many respects, yet have a few additional capabilities exposed. Other Nodes will have to perform these management operations as "Management Providers" without interfering with their own configuration as a Chef Node.

- MANAGED NODES
  - Extension of the Node object within Chef.
  - Appear in searches for Nodes.
  - Have a Management Provider that provides the configuration services for that Managed Node.
  - Have a list of allowed Management Providers to use for configuring the node.
  - The list of Nodes allowed to manage a particular Managed Node are stored on the Managed Node.
  - Resources executed against the Managed Node will be supported by Providers specific to that device.
- MANAGEMENT PROVIDERS
  - Nodes that configure Managed Nodes are referred to as Management Providers.
  - Management Providers have no knowledge that they are providing this service.
  - Security or network topology concerns are addressed by assigning Management Providers to the Managed Nodes.
  - Providers may install additional packages or tools on the Management Provider, but not alter the Management Provider otherwise.
  - "knife ssh" calls will pass through to the Management Provider for the Managed Node.

- ATTRIBUTES
  - node.managed?: boolean for whether or not this is a Managed Node
  - node.manager: the Node that is currently acting as the Management Provider for this node
  - node.allowed_managers: list of Nodes that are allowed to act as Management Providers for this node.

- KNIFE
  - All of the "knife node" operations are available for managing Managed Nodes.
  - "knife search node" will return all Nodes, including Managed Nodes.
  - "knife node managed add NODE PROVIDER" will add a new allowed Management Provider
  - "knife node managed remove NODE PROVIDER" will remove an allowed Management Provider
  - "knife node managed set NODE PROVIDER" will set the Management Provider for the node

- CHEF-CLIENT
  - chef-client currently only supports a single Node, needs to support multiple Managed Nodes.
  - chef-client needs to support Managed Nodes with schedules independent of the Node itself.
  - chef-client needs to be safe for multiple simultaneous runs
  - ohai is currently run as an on-Node application, need to extend it to work remotely or write a new 'ohaithere' application.
  - ohaithere will need to support plugins for supporting multiple remote retrieval methods, yet still return ohai-compatible data

- DISCUSSION
  - What is the best way to manage client keys and access credentials for Managed Nodes? Encrypted data bags?
  - How does the Managed Provider find that it has a Managed Node and start managing it? Check during every chef-client run?
  - Do we want Managed Nodes to have an additional search and knife keyword (ie. 'knife managed' and/or 'search(:managed, "**:**")')

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

?
- Does the Chef Server assign the node.manager from the node.allowed_managers randomly or is always explicit? If random, when do we reassign?
- Do we need to extend the chef-client cookbook to support Managed Nodes?
- knife managedbootstrap managed_node -r 'recipeblah,roleblahblah' -managers "a.foo.com, b.foo.com, c.foo.com"

Rob's notes from the session

Managed Nodes are entities that cannot run chef-client. There are needs to collect information from them (ohai).
Management Provider will be a HWRP (heavy weight resource provider) that does the implementation. For example, the firewall cookbook is a reference implementation.
Terms are subject to change if you have a better suggestion.
Managed nodes are used where the node cannot take actions on their own. A node would make a configration request on the managed node. The managed node would action on that request during its next chef-client request.
We need to treat managed nodes as both a service (actuator / request) and a configuration (node / managed entity). The goal is for the later. In the later case, the node is assumed to have all the information it needs to do the configuration via Chef.

Some configuration may be out of band for setup; however, the node will have all the data that is needed.
This is not just about actions, it is also about search ability.
Use case - have a SAN managed nodes. Cookbook could ask the node for a LUN. Node would be able to discover available storage and create a new LUN.

Implementation discussion - knife & chef-client are not setup to have mulitple entities on the same system. This can create challenges.
Open question - do we need to track relational data between nodes & managed nodes?
Timeframe
- Opscode says this is a high priority to get into the product
- Dell is working in the next quarter on a hack pre-implementation that does this in our context

- Managed Nodes are entities that cannot run chef-client. There are needs to collect information from them (ohai).
- Management Provider will be a HWRP (heavy weight resource provider) that does the implementation. For example, the firewall cookbook is a reference implementation.
- Terms are subject to change if you have a better suggestion.
- Managed nodes are used where the node cannot take actions on their own. A node would make a configration request on the managed node. The managed node would action on that request during its next chef-client request.
- We need to treat managed nodes as both a service (actuator / request) and a configuration (node / managed entity). The goal is for the later. In the later case, the node is assumed to have all the information it needs to do the configuration via Chef.
- Some configuration may be out of band for setup; however, the node will have all the data that is needed.
- This is not just about actions, it is also about search ability.
- Use case - have a SAN managed nodes. Cookbook could ask the node for a LUN. Node would be able to discover available storage and create a new LUN.
- Implementation discussion - knife & chef-client are not setup to have mulitple entities on the same system. This can create challenges.
- Open question - do we need to track relational data between nodes & managed nodes?

Timeframe

- Opscode says this is a high priority to get into the product

- Dell is working in the next quarter on a hack pre-implementation that does this in our context

What will we do now? What needs to happen next?

Discussion of design needs to move to mailing lists and Opscode Product Management will prioritize for development. A number of users have expressed the plan to start implementing stop-gap measures in the meantime.

Talk about External Entity Hack and Design to mode intermediate solution forward.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Messaging The Benefits of Chef

Session Title:

Convener:

Participants:

Summary of discussions: What will we do now? What needs to happen next?

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Monitoring Sucks

Session Title:

Convener:
Sean Porter

Participants:
Jesse Nelson
Kyle Bader
Matt Olson
Kurt Yoder
Sean P. Kane
Nathaniel Eliot
Sean Horn
James Casey

Summary of discussions:

Tool list by category

General Monitoring Platforms

Nagios

- PNP - Graphing plugin for Nagios.
- MKLivestatus - Nagios event broker that exposes Nagios state by way of a socket.
- Thruk - Multi-installation Nagios webui, connects to Nagios servers configured with MKLivestatus event broker.

Collectd
Zennos
Zabbix - InfoChimps is currently integrating this with cluster_chef v3
Cacti
Sensu - uses RabbitMQ and operates at scale
Munin
Ganglia

Graphite - Dynamic addition of arbirtrary metrics. Integrations with Collectd, munin, etc
Statsd - etsy. Metric stats aggregation for dump into Graphite

Cube - Time series visualization
CycleServer
Grill

Logging

Graylog2
LogStash
Logster - etsy.
Splunk

What will we do now? What needs to happen next?

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Opscode Chef Short-Term Roadmap and Performance Improvements

Session Title:

Convener:
Christopher Brown (cb@opscode.com, @skeptomai), Kevin Smith (k@opscode.com, @kevsmith)

Participants:
A cast of thousands... Well, at least a couple dozen and Sean Horn

Slides:
chef_summit_scaling.pdf

Summary of discussions: Rob Hirschfeld's Notes

Chef <3 Erlang

Scaling

- moving to more Erlang on REST APIs, made a 2x in memory use difference
- Erlang bits have realy steady performance

Database performance

- they do not like CouchDB for their use cases
- moving to mySQL, but will allow users to pick

Noop/Dry Run mode

- asking for suggestions

Search improvements

- tune for most common: single attribute & single mode
- considering new syntax for simple search
- subsets of data

Reporting

What will we do now? What needs to happen next?

Q from Sean Horn: Are the search improvements mutually exclusive with retaining all currently available search syntax? Could there be two different modes?
I think there will always be someone who will miss the full power, even when 90% don't need it.
A: (from @skeptomai) Some of the search improvements will dovetail right in with the current implementation and will share and augment syntax in a backward-compatible way.
We are thinking of a new feature as well, that fits many use cases, but would have a different syntax.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Opscode Long-Term (Strategic) Product Priorities Input

Session Title:

Convener:

Participants:

Summary of discussions: Rob Hirschfeld's notes:

Discussion for roadmap

- ITIL integration / Better Error Handling
- scrap node & logs on failed run
- automatic trouble ticket
- Logging & History for audit / history
- Compliance
- RBAC for knife
- Reports
- inventory, audit, compliance
- changed logs
- velocity & consistency

Dell's wish list

- managed nodes

What will we do now? What needs to happen next?

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Ticket Triage and Process

Session Title:

Convener: Bryan McLellan (btm)

Participants: Dan DeLeo (kallistec), Marc Paradise, Joshua Timberman (jtimberman), Tim Smith, Charles J, Brett Weaver, Capen Brinkle, Sean Escriva, Justin Kolberg, Scott Likens, Tommy Bishop, Jeremy Bingham

Summary of discussions:

Everyone expressed being pleased with the improvements in the process over the last year. We talked about the path that different individuals had followed to become involved in the project; how they got there and what information they encountered. Some users found the CLA page, but not the ticket tracker or 'How to Contribute' page, and others found the ticket tracker but not the wiki page by way of search. There were many suggestions for ways to improve the process for new users, which are outlined and captured below:

What will we do now? What needs to happen next?

- Improving the 'Introduction' paragraph on http://tickets.opscode.com - **done**
- Investigate / create a github bot for responding to pull requests
- Have a launchpad/reportbug type plugin for JIRA to search for duplicates when creating bugs
- Posting internal Opscode ticket triage, merge and release processes on the public wiki
- Expand the How to Contribute page for coding information such as style, tests, etc.
- Create an interface to suggest tickets to new contributors by degree of difficulty
- Update the email the user receives after they complete the CLA

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Use Cases of Chef - Share Your Story

Session Title:

Convener:

Participants:
Kurt Yoder

Summary of discussions:

Sharing individual use cases of chef. There were only two of us present/sharing 

What will we do now? What needs to happen next?

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Suggestions for Community Summit

Unfortunately, some community members will not be able to attend the event. This page is for collecting suggestions, particularly focused on how to engage those not present.

Topics

- What is the roadmap for Opscode & Chef?
- What should we do to make Chef more awesome?
- What best practices have emerged & how should we share them?
- How will our community grow while serving experts and welcoming newcomers?
- What should the Opscode Conference & other events in 2012 cover?
    - What should we call the Opscode User Conference?
- How can we make the cookbook site and cookbook community as useful and vibrant as that of rubygems?
    - This is the most important issue facing Chef.
        - The difficulty to find the latest version of an available cookbook and then figure out how its different than dozens of others buried in a repo called cookbooks strewn accross github is daunting
        - The current knife site install being stuck to the one somewhat opaque community cookbook site is stifling
        - Its really hard to make tweaks to cookbooks but still stay in sync with the source of the cookbook, let alone share the tweaks
        - The model of each organization using Chef having one git repo for all cookbooks adds to the difficulty
        - The model of an organization sharing their cookbooks on github as one big repo makes things difficult
        - Some of these difficulties are due to standard ways of doing things with git and github
            - To share via git and github you need to have a cookbook per git repo.
            - If you have dozens of cookbooks in your github account, it overwhelms the github UI
            - Its not really easy to mix and match cookbooks from various sources and still be able to follow upstream changes or push changes upstream
            - Maybe separate cookbook distribution from development (gems vs git repos)?
- Cookbook Management and Use
    - What are anti-patterns for cookbooks?
    - What are best practices for cookbooks?
    - Cookbook testing and how to automate it, esp. for opscode/cookbooks
    - What should the role of opscode/cookbooks be?
- Performance Tuning
- Security
- Building on top of chef, a la crowbar
- How to ensure the Chef server is up to date with version controlled recipes, possibly with different branches or tags for different environments, and with contributions coming from different parts of the organization?
- Professional certification for Chef, like the MCSE, RHCE. Should Chef have it? How can we make it not suck?
- How can organizations that do not use the hosted Chef contribute financially (and otherwise) to the development of Chef?
- How to make Chef-server resilient in the face of cloud providers SLA levels.
- How to manage clusters
- How to making configuration more transparent, make it easier to visualize dependencies
- Auditing/intentions of configuration changes
- Sizing and scaling a chef server/infrastructure. Redundancy, CYA, etc.
- Orchestration of multi-node systems
    - Define functionality of a cluster but render to different target clouds, physical nodes, virtual machines
        - Development, Staging, Production, etc
        - Same Orchestration description / cookbooks should be easy to render to a Vagrant Virtualbox or 10's of nodes on EC2

Event Management

- livestream the event
- Try to facilitate the participation of those not present, i.e. take questions from IRC

> *(Response from Jesse Robbins) The tl;dr here is that real-time remote participation in Open Space is challenging to do in a way that doesn't interfere with and dampen the dynamic and intimate flow of Open Space.*
>
> *If you have never participated in an Open Space, this may sound strange. The Opscode Community Summit isn't a conference. There are no speakers, panelists, or keynotes. There is no pre-set agenda and there is no "audience". There are only fellow participants, a unique and incredible group of people gathered together in one place to work on something that matters to all of us.*
>
> *There are at least two challenges to livestreaming & real-time remote participation:*

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

> *1) The space we create is a safe place for people to openly discuss complex and often confidential topics without worrying that someone will share it, an idea Tim O'Reilly calls the FrieNDA. Livestream ing or recording sessions by default has the potential to change this, and in most cases just doesn't make sense to do as the facilitator. If someone decides to propose a streamed or recorded session, host a call in show, or anything else they are fully empowered to do so.*
>
> *2) The technology required for people to participate in real-time and at scale, which preserves the power and intensity of a face-to-face interaction and doesn't create friction that distracts from the experience... isn't readily available. Many people find they close their laptops/ipads/etc entirely and focus on being present. If people want to host questions on IRC or other means, they are fully empowered to do so.*
>
> *We will provide a wiki for participants to document sessions, issues, actions, conversations, and conclusions. While we don't know what exactly what will happen over the two days of the Summit, there are some things that are guaranteed to happen:*
> *• The issues that are most important to people will get discussed & worked on*
> *• The issues raised will be addressed by the participants best capable of getting something done about them.*
> *• All of the most important ideas, recommendations, discussions, and next steps will be documented in a report which will be published to the entire community.*
> *• Participants will be deeply engaged, connected, and energized.*
>
> *Hope this helps,*
>
> *-Jesse*

- more wifi (I got locked out repeatedly, presumably due to the large number of simultaneous attempted connections)
- ensure each space has whiteboard space to serve as a projection view space (only spaces E & F had it)
- more sound partitioning between spaces, if possible

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Support



Getting Help With Chef

We've tried to make the documentation on this wiki as comprehensive as possible, but inevitably folks need to get help.

Chef User Help Resources

All Chef users have access to multiple help resources beyond the information found here on the Chef Wiki.

Real-time Help

The best way to get help immediately is by joining the #chef channel on irc.freenode.net. Several folks from Opscode are there regularly (easily distinguished by 'operator' status), as well as lots of people in the community who have probably run into your issue or something similar themselves. **Being a Chef user is being part of a vibrant Open Source community!**

- See the **IRC** page for more information.
- **IRC Logs** are maintained as well, which you can conduct a site search against for specific information from previous discussions.

Chef Mailing Lists

Joining the Chef and/or Chef-Dev Mailing Lists can be an excellent way to ask questions, and to interact and participate in discussion on Chef use, gain tips and information on use of Chef, as well as to provide input to product futures.

- Sign up for the Chef Mailing Lists at **http://lists.opscode.com**.

File a Open Source Ticket

If you've found a bug in Chef, Ohai or the Opscode Cookbooks, you'll probably be asked to file a ticket. Our Open Source ticket tracking system is Jira, at **tickets.opscode.com**.

Hosted Chef and Private Chef Directed Support

If you are a paid customer of Hosted Chef or Private Chef, you can get directed help from Opscode by either starting a discussion on **help.opscode.com**, or sending email to **support@opscode.com**. *Be sure to provide as much information (as noted to the right) as you can when creating the ticket.*

Periodic maintenance is announced in advance via the **Opscode Status** site, as are any emergent issues with availability. Hosted Chef customers who use Twitter can also receive status announcements by following the Opscode Status Twitter Account.

Training

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

We offer comprehensive training classes for Chef and Hosted Chef. Classes are conducted by Opscode employees who have years of experience working in the field of distributed configuration management. There is also the Opscode Open Training program, which makes Opscode enterprise training materials freely available to Chef users. See **Opscode training** for more information. *(For a review from a recent Chef Fundamentals attendee, see My Impressions of Opscode Chef Training.)*

For online training - within our **Guides** you'll find a webcast of a training/lab session, a number of walkthrough guides for the installation of common server stacks, how-to guides for specific actions, and tutorials created by community members who were once new to Chef themselves.

Check the FAQs!

1. **General question FAQ** about Chef
2. **Troubleshooting and Technical FAQ** for when disaster strikes your land
3. **Common Errors** encountered with chef-client and knife

Check these locations first, then...

Information Required

First, gather as much information as possible about the issue. The following is needed in almost every case.

- Platform and version.
- How Chef was installed (packages, RubyGems, from source, etc).
- If using RubyGems, how it was installed.
- Debug logs from the client. Run "`chef-client -l debug`".
- Debug logs from the server if you are an Open Source Chef Server user. Set "`log_level :debug`" in `/etc/chef/server.rb`.
- If you custom compiled Ruby (or are using REE).
- Anything else specific about your environment that may be helpful.

Log file locations
Check the server or client config file for the `log_location`.

- /etc/chef/server.rb
- /etc/chef/client.rb

General Communication and Information

Follow the **Opscode Twitter Account** for broader company and product news. Hosted Chef customers can also receive status announcements on Hosted Chef maintenance and availability by following the **Opscode Status Twitter Account**.



twitter

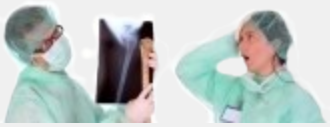
Opscode Blog

Our **Opscode blog** is updated regularly with product information, tips, product releases and previews, training and local Chef User Group meetups, and other information of potential interest. Check out the Blog at Opscode.com, or **subscribe to the RSS Feed**.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Common Errors



Common errors with Chef-Client, Knife and Chef-Server

If you run into an error that is not on this page, please see the Troubleshooting and Technical FAQ and then turn to Support if you need further assistance.

401 Unauthorized (using node's API client)

Note: This error is very similar to the error in the next section. If you see a line that states c:/chef/client.pem is not present, follow the directions in the next section for 401 Unauthorized (using validator API client) instead. NODENAME below could be any name, but will never be ORGANIZATION-validator like the error in the next section.

Error on Hosted Chef:

```
$ sudo chef-client
[Tue, 01 Nov 2011 16:54:29 -0700] INFO: *** Chef 0.10.x ***
[Tue, 01 Nov 2011 16:54:30 -0700] FATAL: Stacktrace dumped to /var/chef/cache/chef-stacktrace.out
[Tue, 01 Nov 2011 16:54:30 -0700] FATAL: Net::HTTPServerException: 401 "Unauthorized"
```

Error on Open Source Chef:

```
[Thu, 29 Sep 2011 10:10:11 -0400] INFO: *** Chef 0.10.x ***
[Thu, 29 Sep 2011 10:10:12 -0400] INFO: HTTP Request Returned 401 Unauthorized: Failed to
authenticate. Ensure that your client key is valid.
[Thu, 29 Sep 2011 10:10:12 -0400] FATAL: Stacktrace dumped to /var/cache/chef/chef-stacktrace.out
[Thu, 29 Sep 2011 10:10:12 -0400] FATAL: Net::HTTPServerException: 401 "Unauthorized"
```

Fix:
This error can typically happen for 1 of 3 reasons:

- **1. Your `client.pem` file is incorrect.**
  This can be fixed by deleting `client.pem` in /etc/chef, deleting the client and node with knife, and re-running chef-client:

  Saving your Node Data
  The following directions involve deleting the data associated with this node. If the node attributes or run_list would be hard to recreate, you may want to save them:

  ```
  knife node show NODENAME -Fj > NODENAME.json
  ```

  ```
  sudo rm /etc/chef/client.rb
  knife client delete NODENAME
  knife node delete NODENAME
  sudo chef-client
  ```

  When chef-client runs, it will register the API client and generate the correct key.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- **2. You are trying to authenticate with a node_name that is different from the one you used on your first chef-client run.**
  This can happen for a number of reasons. For example, if your `client.rb` file does not specify your node name and you have recently changed the systems hostname. Running chef client with debug logging will allow you to see the node name the client is trying to authenticate with:

  ```
  [Wed, 05 Oct 2011 22:05:35 +0000] DEBUG: Signing the request as SOMENODENAMEHERE
  ```

  You can fix this by explicitly setting the node name in the `client.rb` file or with chef-client's `-N` option to match the name originally used to register. Alternatively, you can re-register using the method described above.

- **3. If you are using Hosted Chef, your node name matches another users username on the platform.**
  This issue is documented here: http://tickets.opscode.com/browse/CHEF-2240. If you have unsuccessfully attempted the solutions above, click the Help link from http://manage.opscode.com and request an analyst to verify this is the issue. To rename the node without changing the hostname, follow these steps:
  - Capture you node data by either

    ```
    $ knife node show SOMENODENAMEHERE -Fj > somefile.chef
    ```

    or by

    ```
    $ knife node edit SOMENODENAMEHERE
    ```

    and copy/paste.
  - Delete the node, client and client.pem on the node.

    ```
    $ knife node delete SOMENODENAMEHERE
    ```

    ```
    $ knife client delete SOMENODENAMEHERE
    ```

    - On the node to be renamed:

      ```
      $ sudo rm /etc/chef/client.pem
      ```

  - Run the chef client on the node and specify the new node name:

    ```
    $ sudo chef-client -N SOME_NEW_NODENAMEHERE
    ```

  - Replace your node data
  - Run the chef-client again (and in all future instances) as

    ```
    $ sudo chef-client -N SOME_NEW_NODENAMEHERE
    ```

    or add

    ```
    node_name SOME_NEW_NODENAMEHERE
    ```

    to the /etc/chef/client.rb file.

To rename your node by changing the host name, delete the node,client and client.pem as described above, change the hostname on the host and run the chef-client as normal.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

401 Unauthorized (using validator API client)

Error on Hosted Chef:

```
C:\chef>chef-client -c client.rb
[Tue, 04 Oct 2011 16:07:09 -0700] INFO: *** Chef 0.10.x ***
[Tue, 04 Oct 2011 16:07:11 -0700] INFO: Client key c:/chef/client.pem is not present - registering
[Tue, 04 Oct 2011 16:07:13 -0700] INFO: HTTP Request Returned 401 Unauthorized: Failed to authenticate
as ORGANIZATION-validator. Ensure that your node_name and client key are correct.
[Tue, 04 Oct 2011 16:07:13 -0700] FATAL: Stacktrace dumped to c:/chef/cache/chef-stacktrace.out
[Tue, 04 Oct 2011 16:07:13 -0700] FATAL: Net::HTTPServerException: 401 "Unauthorized"
```

Error on Open Source:

```
[Thu, 14 Jul 2011 11:44:44 +0000] INFO: *** Chef 0.10.x ***
[Thu, 14 Jul 2011 11:44:44 +0000] INFO: Client key /etc/chef/client.pem is not present - registering
[Thu, 14 Jul 2011 11:44:44 +0000] INFO: HTTP Request Returned 401 Unauthorized: Failed to
authenticate. Ensure that your client key is valid.
[Thu, 14 Jul 2011 11:44:44 +0000] FATAL: Stacktrace dumped to /var/cache/chef/chef-stacktrace.out
[Thu, 14 Jul 2011 11:44:44 +0000] FATAL: Net::HTTPServerException: 401 "Unauthorized"
```

Fix:
If you get this error, you most likely need to recreate the validation key. If you lose this file, or perhaps you end up with an empty CouchDB database and no longer have this client in the database, you could get a 401 Unauthorized error when trying to use it.

In Hosted Chef, you can recreate this key by going to http://manage.opscode.com and selecting 'organizations' in the upper right side of the screen. You can then select 'Regenerate validation key' next to the organization you need the key for. You will then want to replace this on the client. Be aware that any other nodes you have will no longer work until you distribute the new validation.pem to them.

If you've also lost your client_key for knife, you can also regenerate this at http://manage.opscode.com by selecting your username in the upper right and then 'get private key'.

Copy these keys over to your node. Your node should be able to register now, as long as there is not still a client by the same name. If there is, you will need to delete that client first. Note that on Opscode Hosted Chef, you currently will need to delete the node as well, because the default permissions only allow the client that created the node to modify it.

```
$ knife client list
  ORGANIZATION-validator
  ip-10-204-150-209.ec2.internal
  new_ubuntu
  ubuntu
$ knife client delete ip-10-204-150-209.ec2.internal
Do you really want to delete ip-10-204-150-209.ec2.internal? (Y/N) y
Deleted client[ip-10-204-150-209.ec2.internal]
$ knife node delete ip-10-204-150-209.ec2.internal
Do you really want to delete ip-10-204-150-209.ec2.internal? (Y/N) y
Deleted node[ip-10-204-150-209.ec2.internal]
```

On a Chef Server, you can recreate the key by following these steps. First you'll want to remove your validation key on the server, which is typically stored at /etc/chef/validation.pem. Afterwards you can restart the chef-server to create a new key pair on both the disk and in the database:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
$ ls -l /etc/chef/validation.pem
-rw-r--r-- 1 root root 1676 2011-07-14 11:44 /etc/chef/validation.pem
$ sudo rm /etc/chef/validation.pem
$ sudo /etc/init.d/chef-server restart
 * Restarting chef-server
 ~ Killing pid 10783 with INT
 ~ In 12051
   ...done.
$ ls -l /etc/chef/validation.pem
-rw------- 1 chef chef 1679 2011-07-14 11:55 /etc/chef/validation.pem
```

The same process works with the webui key pair, which knife uses as the default “admin” key to create initial knife clients:

```
$ ls -l /etc/chef/webui.pem
-rw------- 1 chef chef 1675 2011-07-14 11:31 /etc/chef/webui.pem
$ sudo rm /etc/chef/webui.pem
$ sudo /etc/init.d/chef-server restart
 * Restarting chef-server
 ~ Killing pid 12051 with INT
 ~ In 12091
   ...done.
$ ls -l /etc/chef/webui.pem
-rw------- 1 chef chef 1675 2011-07-14 11:57 /etc/chef/webui.pem
```

If you’ve also lost your client key for your knife client, you will need to create another one. Use a new client name unless you’re sure that the server does not still contain a registration for the previous client. After creating the new client, you can delete the old one from the server using ‘knife client delete MY_OLD_CLIENT’ by replacing MY_OLD_CLIENT with the name of the former client.

```
$ sudo knife configure --initial
Overwrite /home/ubuntu/.chef/knife.rb? (Y/N) y
Please enter the chef server URL: [http://ip-10-204-150-209.ec2.internal:4000]
Please enter a clientname for the new client: [ubuntu] new_ubuntu
Please enter the existing admin clientname: [chef-webui]
Please enter the location of the existing admin client's private key: [/etc/chef/webui.pem]
Please enter the validation clientname: [chef-validator]
Please enter the location of the validation key: [/etc/chef/validation.pem]
Please enter the path to a chef repository (or leave blank):
Creating initial API user...
Created client[new_ubuntu]
Configuration file written to /home/ubuntu/.chef/knife.rb
```

Copy these keys over to your node. Your node should be able to register now, as long as there is not still a client by the same name. If there is, you will need to delete that client first.

```
$ knife client list
  chef-validator
  chef-webui
  ip-10-204-150-209.ec2.internal
  new_ubuntu
  ubuntu
$ knife client delete ip-10-204-150-209.ec2.internal
Do you really want to delete ip-10-204-150-209.ec2.internal? (Y/N) y
Deleted client[ip-10-204-150-209.ec2.internal]
```

You should now be able to run the chef-client command without errors.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

401 Unauthorized (Failed to Authenticate as "NODENAME")

Error on Hosted Chef:

```
chef-client -j run_list.json
[Fri, 03 Jun 2011 14:44:33 -0700] INFO: Client key /etc/chef/client.pem is not present - registering
[Fri, 03 Jun 2011 14:44:38 -0700] WARN: HTTP Request Returned 401 Unauthorized: Failed to authenticate
as NODENAME. Ensure that your node_name and client key are correct.
```

There is an open Hosted Chef bug that user names and client names cannot be the same.

> *Creating a new client with an id that clashes with an id in the User namespace results in a successful client create.*
>
> *Trying to use this client, however, results in failed authentication because the client tries to authenticate in the user namespace.*
>
> *The system currently does not differentiate between the User and Client namespaces during authentication, and this needs to happen.*

Future activities to implement name spacing will address this, but pending this bug being fixed, if your proposed NODENAME is already an existing User name - the authentication will fail for this reason. To validate that this is the cause, search the Opscode Community Site Users to see if there is an existing user with the same name as what you attempted to name the node. Changing the NODENAME to being unique will address this error.

If your node name is unique, and there is no overlapping user name, please provide support with the debug output of the chef-client run and we'll determine and address the cause.

401 Unauthorized (Please synchronize the clock)

Error on Hosted Chef:

```
$ sudo chef-client
[Tue, 01 Nov 2011 16:55:23 -0700] INFO: *** Chef 0.10.x ***
[Tue, 01 Nov 2011 16:55:23 -0700] INFO: Client key /etc/chef/client.pem is not present - registering
[Tue, 01 Nov 2011 16:55:24 -0700] INFO: HTTP Request Returned 401 Unauthorized: Failed to authenticate
as ORGANIZATION-validator. Synchronize the clock on your host.
[Tue, 01 Nov 2011 16:55:24 -0700] FATAL: Stacktrace dumped to /var/chef/cache/chef-stacktrace.out
[Tue, 01 Nov 2011 16:55:24 -0700] FATAL: Net::HTTPServerException: 401 "Unauthorized"
```

Error on Open Source:

```
$ sudo chef-client
[Tue, 01 Nov 2011 17:01:11 -0700] INFO: *** Chef 0.10.x ***
[Tue, 01 Nov 2011 17:01:12 -0700] INFO: Client key /home/ubuntu/chef-repo/client.pem is not present -
registering
[Tue, 01 Nov 2011 17:01:12 -0700] INFO: HTTP Request Returned 401 Unauthorized: Failed to
authenticate. Please synchronize the clock on your client
[Tue, 01 Nov 2011 17:01:12 -0700] FATAL: Stacktrace dumped to /var/chef/cache/chef-stacktrace.out
[Tue, 01 Nov 2011 17:01:12 -0700] FATAL: Net::HTTPServerException: 401 "Unauthorized"
```

Fix:
Your system clock has drifted from the actual time by more than 15 minutes. This can be fixed by syncing your clock with an NTP server. You'll need to rerun chef-client after synching NTP.

403 Forbidden

Error on Hosted Chef:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
$ sudo chef-client
[Mon, 30 Jan 2012 19:58:51 -0800] INFO: *** Chef 0.10.x ***
[Mon, 30 Jan 2012 19:58:53 -0800] FATAL: Stacktrace dumped to /var/chef/cache/chef-stacktrace.out
[Mon, 30 Jan 2012 19:58:53 -0800] FATAL: Net::HTTPServerException: 403 "Forbidden"
```

Fix:
On Hosted Chef, this indicates an issue with the permissions that are setup at http://manage.opscode.com. You can get more information on the specific permission to check by re-running chef-client with `-l debug` at the end of the command to get the debug logs. Once it is done running, you'll need to scroll up in the logs until you find the last HTTP request. There are two different types of permissions issues, object specific and global permissions.

This is an example of a 403 error on a specific object:

```
[Mon, 30 Jan 2012 20:06:43 -0800] DEBUG: Sending HTTP Request via PUT to
api.opscode.com:443/organizations/ORGNAME/nodes/NODENAME
[Mon, 30 Jan 2012 20:06:43 -0800] ERROR: Running exception handlers
[Mon, 30 Jan 2012 20:06:43 -0800] FATAL: Saving node information to
/var/chef/cache/failed-run-data.json
[Mon, 30 Jan 2012 20:06:43 -0800] ERROR: Exception handlers complete
[Mon, 30 Jan 2012 20:06:43 -0800] DEBUG: Re-raising exception: Net::HTTPServerException - 403
"Forbidden"
```

What we are specifically looking for is the type of http request, in this case PUT, and where it tried to PUT to, in this case a specific node/object. Because this points to a specific node instead of the nodes group, this will be an issue with the object permissions. If this pointed to the nodes group, this would be an issue with the global permissions on nodes instead. These errors do not just apply to nodes, you may see these errors for any section of the management console such as Roles, Cookbooks or Environments.

Creating Clients with Knife
Are you seeing the 403 error when doing a PUT request to /clients/NODENAME whenever you start up a new node? If you are creating the client with knife first, remember to copy the key it provides to `/etc/chef/client.pem` or chef-client will get a 403 error when trying to create it.

To fix the object permissions, you'll want to follow these steps:
1. Go to http://manage.opscode.com and click on nodes in this case, or wherever you see the request failing
2. For object specific permissions, click on the specific object that showed up in the error, in this case NODENAME
3. Click on the permissions sub-tab. Which permission it needs, depends on the type of request that failed:
GET - Under the client section, make sure it has the READ permission checked next to the NODENAME
PUT - Under the client section, make sure it has the UPDATE permission checked next to the NODENAME
DELETE - Under the client section, make sure it has the DELETE permission checked next to the NODENAME
4. Check the checkboxes needed and click on 'update permissions'.

If this is an issue with the global permissions the error will look a bit different. **This is an example of a 403 error on the global permissions:**

```
[Mon, 30 Jan 2012 20:16:22 -0800] DEBUG: Sending HTTP Request via POST to
api.opscode.com:443/organizations/ORGNAME/nodes
[Mon, 30 Jan 2012 20:16:23 -0800] FATAL: Stacktrace dumped to /var/chef/cache/chef-stacktrace.out
[Mon, 30 Jan 2012 20:16:23 -0800] DEBUG: Net::HTTPServerException: 403 "Forbidden"
```

The difference is that here it is referring to the entire nodes group, instead of a specific object/node.

To fix the global permissions you'd want to follow these steps:
1. Go to http://manage.opscode.com and click on nodes in this case, or wherever you see the request failing
2. Click on the permissions sub-tab. Which permission it needs, depends on which request that failed:
GET - Under the group section, make sure it has the LIST permission checked next to the clients group
POST - Under the group section, make sure it has the CREATE permission checked next to the clients group
3. Check the checkboxes needed and click on 'update permissions'.

More information on the API requests and associated errors can also be found on the Server API page.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Commit or stash your changes before importing cookbooks

This isn't really an error, but can be confusing to new users. When you try to install a cookbook with changes that have not been committed to git you will get this error:

```
$ knife cookbook site install getting-started
Installing getting-started to /home/jes/chef-repo/.chef/../cookbooks
ERROR: You have uncommitted changes to your cookbook repo (/home/jes/chef-repo/.chef/../cookbooks):
 M cookbooks/getting-started/recipes/default.rb
?? .chef/
?? log
Commit or stash your changes before importing cookbooks
```

Fix:
You can solve this by committing your changes. For example, this would commit all new changes with the message "updates".

```
$ git commit -a -m "updates"
```

Alternatively, if you do not want to commit the changes, you can save the changes for later (reverting the branch to the state of the latest commit) by running

```
$ git stash save
```

Afterwards you can re-enter the command to install the cookbook.

No such file or directory - /etc/chef/validation.pem

Error:

```
C:\> chef-client -c c:\chef\client.rb
[Fri, 23 Sep 2011 17:37:16 -0700] INFO: *** Chef 0.10.x ***
[Fri, 23 Sep 2011 17:37:18 -0700] INFO: Client key /etc/chef/client.pem is not
resent - registering
[Fri, 23 Sep 2011 17:37:18 -0700] WARN: Failed to read the private key /etc/che
/validation.pem: #<Errno::ENOENT: No such file or directory - /etc/chef/validat
on.pem>
[Fri, 23 Sep 2011 17:37:18 -0700] FATAL: Stacktrace dumped to C:/var/chef/cache
chef-stacktrace.out
[Fri, 23 Sep 2011 17:37:18 -0700] FATAL: Chef::Exceptions::PrivateKeyMissing: I
cannot read /etc/chef/validation.pem, which you told me to use to sign requests
```

Fix:
Make sure your validation.pem (for Chef server Users) or ORGANIZATION-validator.pem (for Hosted Chef Users) is downloaded. Edit your client.rb to point to the location of your validator pem.

Can not find config file

Error:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
c:\chef>chef-client
[Fri, 30 Sep 2011 13:51:15 -0700] WARN: ****************************************
*
[Fri, 30 Sep 2011 13:51:15 -0700] WARN: Can not find config file: /etc/chef/clie
nt.rb, using defaults.
[Fri, 30 Sep 2011 13:51:15 -0700] WARN: No such file or directory - /etc/chef/cl
ient.rb
... output truncated ...
[Fri, 30 Sep 2011 13:51:20 -0700] FATAL: Chef::Exceptions::PrivateKeyMissing: I
cannot read /etc/chef/validation.pem, which you told me to use to sign requests!
```

Fix:
This error is related to bug CHEF-2317 on Windows, and can also happen if you are not in your chef directory on Linux and Mac. You can work around it by supplying the full path to your client.rb. For example, if your client.rb was in C:\chef you could use this command:

```
chef-client -c C:\chef\client.rb
```

Can't convert Array into String

Error:

```
$ knife bootstrap IP --distro ubuntu -x USERNAME
Bootstrapping Chef on IP
ERROR: TypeError: can't convert Array into String
```

Fix:
There are a few things that can cause this error. Check the following to resolve this:

1. Make sure you are running this command from your chef-repo directory which has the .chef file in it. Knife will need direct access to the .chef file, which contains it's config.

2. If you are using the `--distro` switch, confirm the bootstrap file exists in one of the directories it checks:

- bootstrap directory in the installed Chef Knife library (Location varies depending how you installed Chef).
- bootstrap directory in the $PWD/.chef, e.g. in ~/chef-repo/.chef.
- bootstrap directory in the users `$HOME/.chef`.

In the example above, the bootstrap file could be located at `~/chef-repo/.chef/bootstrap/ubuntu.erb`. More information on this is on the Knife Bootstrap and the Custom Knife Bootstrap Script pages.

3. Check the knife.rb file for any relative paths and try using the full absolute path instead.

Windows Specific

These errors below are common on Windows systems and may not apply to Linux/Mac.

Knife cookbook site install

Error:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
C:\chef>knife cookbook site install getting-started
Installing getting-started to c:\chef\cookbooks
Checking out the master branch.
Pristine copy branch (chef-vendor-getting-started) exists, switching to it.
... output truncated ...
---- End output of rm -r c:\chef\cookbooks/getting-started ----
Ran rm -r c:\chef\cookbooks/getting-started returned 1
```

Fix:
Knife cookbook site install currently doesn't work due to bug CHEF-2250. You can work around this by using `knife cookbook site download` and then untaring the file:

```
C:\> cd %HOMEPATH%\chef-repo\cookbooks
C:\Users\username\chef-repo\cookbooks> knife cookbook site download getting-started
C:\Users\username\chef-repo\cookbooks> tar zxvf getting-started-0.2.0-tar.gz
C:\Users\username\chef-repo\cookbooks> del getting-started-0.2.0-tar.gz
```

Cannot find a version for node

Error:

```
C:\>chef-client -c c:\chef\client.rb
[Fri, 23 Sep 2011 17:43:20 -0700] INFO: *** Chef 0.10.x ***
[Fri, 23 Sep 2011 17:43:25 -0700] FATAL: Stacktrace dumped to c:/chef/cache/chef
-stacktrace.out
[Fri, 23 Sep 2011 17:43:25 -0700] FATAL: ArgumentError: Cannot find a version fo
r node[windows-xp.hsd1.wa.comcast.net.]
```

Fix:
This is fixed in Ohai versions 0.6.8 and above. You can update to it with:

```
gem update ohai
```

You can confirm you're at the correct version with this command:

```
C:\> ohai --version
Ohai: 0.6.8
```

Alternatively, you can downgrade your Ruby to 1.8.7 to get around this error.

No such file to load – ruby-wmi

Error:

```
C:\chef>chef-client -c c:\chef\client.rb
C:/Ruby192/lib/ruby/site_ruby/1.9.1/rubygems/custom_require.rb:36:in `require':
no such file to load -- ruby-wmi (LoadError)
```

Fix:
Install the Ruby Gems needed for Windows:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
C:\> gem install win32-open3 ruby-wmi windows-api windows-pr --no-rdoc --no-ri --verbose
C:\> gem install rdp-ruby-wmi
```

Chef Server Specific

These errors are specific to using the Open Source Chef Server, and are not something you will see when using Hosted Chef.

Starting rabbitmq-server: FAILED

This error only occurs in Ubuntu versions 11.04 and below when installing `chef-server`. It is a `rabbitmq-server` issue that is related to this Rabbitmq bug: https://bugs.launchpad.net/ubuntu/+source/rabbitmq-server/+bug/653405

Error:

```
$ sudo apt-get install chef chef-server
[sudo] password for ubuntu:
Reading package lists... Done
Building dependency tree
... output truncated ...
Adding group `rabbitmq' (GID 123) ...
Done.
Adding system user `rabbitmq' (UID 115) ...
Adding new user `rabbitmq' (UID 115) with group `rabbitmq' ...
Not creating home directory `/var/lib/rabbitmq'.
Starting rabbitmq-server: FAILED - check /var/log/rabbitmq/startup_log, _err
rabbitmq-server.
invoke-rc.d: initscript rabbitmq-server, action "start" failed.
... output truncated ...
Processing triggers for libc-bin ...
ldconfig deferred processing now taking place
Errors were encountered while processing:
 rabbitmq-server
 chef-solr
 chef-expander
 chef-server-api
 chef-server
 chef-server-webui
E: Sub-process /usr/bin/dpkg returned an error code (1)
```

Fix:
This error can occur when the hostname is not properly resolving, or when the machine has not been rebooted after changing the hostname. You can test this by running the command `ping $HOSTNAME` if it responds with `unknown host` or the old hostname, follow these directions to get Chef Server installed:

1. Edit `/etc/hostname` and `/etc/hosts` so the hostname in `/etc/hostname` resolves to 127.0.1.1 or 127.0.0.1 in `/etc/hosts`.
2. Reboot the machine.
3. Remove rabbitmq-server, chef, and chef-server:

```
sudo apt-get remove rabbitmq-server chef chef-server
```

4. Remove this rabbitmq file so it gets properly installed next time:

```
sudo rm -R /var/lib/rabbitmq
```

5. Re-install Chef and Chef Server:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
sudo apt-get install chef chef-server
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Cookbook Support



Opscode maintains the cookbooks we share on the Opscode Community Cookbooks Site in the same manner as our other Open Source projects.

We provide assistance and guidance for our cookbooks to the Open Source Chef community through the Open Source Community Support Channels, including through Chef Mailing Lists, IRC, and through directed support for Hosted Chef and Private Chef customers.

Opscode Cookbooks Testing

Our full support for Opscode Cookbooks includes testing for functionality and validating usability for all platforms documented below.

Opscode Hosted Chef and Opscode Private Chef customers can also open support requests through our support site or by emailing support@opscode.com. Directed support for cookbooks rarely rises to a high severity level based in the Service Level Agreement for escalated response, as cookbook issues can be usually be resolved by modifying the local copy as a workaround. We fully support our cookbooks, and commit to doing so - endeavoring to respond to all Hosted Chef customer questions ahead of our service level commitments.

What cookbooks are supported?

All cookbooks that Opscode released under the "opscode" user on the Opscode Community site are supported.

(The number of cookbooks on the screenshot may differ from the actual number)



Cookbook source code is available on in the Opscode Cookbooks GitHub Repository. Cookbooks in GitHub may be in the midst of a development cycle, *we therefore recommend users download cookbooks from the Chef Community site to ensure they are receiving the latest released version*.

We will provide our best effort in response to any questions that arise in use of non-Opscode contributed cookbooks, but **Opscode does not test or directly support cookbooks that other members of the community share on the Opscode Community site.** Issues with non-Opscode provided and supported cookbooks would need to be addressed with the provider, or directly by the user through modifying the cookbook for their specific environment.

How do I get cookbook support?

If the interaction in Open Source Support Resources results in a bug determination, all users may open tickets in the Open Source Bug Ticket Tracking System. To do so, use the COOK project, specify the cookbook name as the component, and include as much information about your environment as you can.

Opscode Hosted Chef and Opscode Private Chef customers may open a directed support ticket at support site or by emailing support@opscode.com to request support for a cookbook. Such tickets are typically

Help with Using the Community Cookbook Site

Cookbook Site Help detail is available on the Wiki. Check Cookbook Site Help first, the resolution for issue with its use may be known and readily available.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

classified as Severity 3 or 4, and will receive a response from Opscode
consistent with that classification. Questions or issues submitted through Support may result in a bug ticket being opened in the Open Source Bug Ticket Tracking System, depending upon the particular.

Whether the ticket is opened through Support or through the Open Source Bug Ticket System, please provide as much detail about the cookbook as possible.

- Node platform and version
- Version of Chef
- Log and stack trace output (if any)
- Cookbook name and version
- The failing resource code from the recipe

The more information provided, the more likely we can respond with a proper solution in a timely manner. Often, we can provide a workaround that can be implemented immediately.

What distributions and releases do we support?

Declaring specific distribution support across the board is difficult. Every platform has differing opinions and implementations of the software our cookbooks install and configure.

Any cookbook that Opscode releases will indicate the platforms it supports. For supported platforms: Opscode personnel have converged a node of that platform by following the instructions in the cookbook's documentation, and validated the functionality provided by the cookbook is implemented.

Cookbooks are documented with a README file in the root of the cookbook, which is distributed in the downloadable archive file. For example:

```
% knife cookbook site download apache2
Downloading apache2 from the cookbooks site at version 1.0.2 to
/Users/jtimberman/Downloads/apache2-1.0.2.tar.gz
Cookbook saved: /Users/jtimberman/Downloads/apache2-1.0.2.tar.gz

% tar -zxvf apache2-1.0.2.tar.gz
...

% ls -1 apache2
README.md          <---- Here it is!
attributes/
definitions/
files/
metadata.json
metadata.rb
recipes/
templates/
```

***We focus** cookbook supportability on the following platforms and versions*

- Ubuntu (10.04, 11.04, 11.10)
- Debian (5.0, 6.0)
- RHEL / CentOS (5.x, 6.x)
- Fedora 14+
- Mac OS X (10.5+)
- Windows 7
- Windows Server 2003 R2, 2008 R2

As time progresses, older versions of distributions may be removed from support in the cookbook without warning.

Platform versions not explicitly stated above may (or may not) function as expected

Opscode has simply not *tested and validated functionality* on all possible platforms beyond those listed above. The Opscode cookbooks may require modifications for full functionality on alternate platforms, and on those platforms we cannot guarantee any particular functionality or clean exit.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Also, some cookbooks may not be relevant for a particular platform. (For example, you can add "`recipe[iis]`" to an Ubuntu node, and it will fail to converge.) *It is up to the customer to understand the purpose of the cookbooks they're adding to their nodes.* (i.e.: "This stuff runs as root!")

As part of ongoing maintenance of the Cookbooks, Opscode may add functionality for other platforms to existing cookbooks. The supported platforms will always be included in the cookbook README and metadata. *It is up to the customer to read the cookbook's documentation and note the supported platforms listed in the README and the cookbook metadata.* If a customer or user of a cookbook has a modification that would provide additional functionality, or extend the cookbook to an alternate platform, we'd be happy to included it - please see the next section for how to contribute that change.

How do I contribute to Opscode Cookbooks?

Opscode Cookbooks are maintained in the same manner as Chef and our other Open Source products. Details are available in the Developers section of this Wiki, including the How to Contribute section which details the process for making an open source contribution.

Also please read the README.md file in the root directory of the http://github.com/opscode/cookbooks repository.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

IRC



The secret of Open Source is that all the good things happen on IRC.

Chef is no exception - you can find us in **`irc.freenode.net #chef`** and **`irc.freenode.net #chef-hacking`**, or use the embedded web client to the right!

IRC Logs are maintained as well, which you can conduct a site search against for specific information from discussions on days gone by.

Your browser doesn't support IFRAME, visit http://webchat.freenode.net/?channels=chef

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Mailing Lists



Mailing Lists

All members of the Opscode community can always post to the **Chef Mailing Lists** for interaction and assistance.

Joining the Chef and/or Chef-Dev Mailing Lists can be an excellent way to hear and participate in discussion on Chef use, gain tips and information on use of Chef, and to provide input to product futures.

Sign up for the Chef Mailing Lists at **http://lists.opscode.com**.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Troubleshooting and Technical FAQ



This is a general troubleshooting and technical FAQ document.

Common Errors has common error messages for Chef-Client, Knife and Chef-Server - and their solutions. For an "About Chef" FAQ, see Chef-FAQ

What is the current version of Chef? Do I need to upgrade?

- The current version of Chef is 0.10.8
- The current version of Ohai is 0.6.10

Private Chef users will have migrations to new versions addressed through a jointly developed migration plan and upgrade implementation. Please follow up with your Private Chef administrator for additional information.

Hosted Chef will always be enabled with the current version, and with functionality sufficient to support use of pre-release versions. When new versions of Chef and the Opscode API are released, they will be immediately available to you on Hosted Chef – data migrations and backwards compatibility taken care of by Opscode. You can upgrade to new release chef-client versions and ohai clients as convenient for you, in order to gain the advantage of new releases functionality.

Open Source Chef users should upgrade their chef-server(s) and chef-clients to the newest version that provides the functionality and bug fixes they require. Another driver for upgrades would be availability of support resources from Opscode and the community. Release Notes has a listing of all of the current supportable versions of Chef. *Versions older than 0.9.0 should be considered obsolete at this point.*

What is an FQDN and why do I need one?

An FQDN is a "Fully Qualified Domain Name." By default, Chef Client uses the node's FQDN to uniquely identify node names to the server if the node name was not explicitly set.

When using Passenger or Apache Proxy configurations for the Chef Server, the FQDN is used for the SSL certificate CN - canonical name.

Do you have a video, or offer training on Chef?

We have both! At **Guides** you'll find a number of "Walkthrough Guides" that take you step by step through the installation of 6 different common server stacks including:

- Rails,
- Java,
- LAMP,
- and others.

With detailed documentation and an accompanying screencast, you can go at your own pace and learn how to use Chef for your environment – all while setting up multiple server stacks you can put to use.

There are also "How To Guides" available on:

- Writing a Recipe and Creating a Chef Cookbook,
- Working with Git and Cookbooks,
- How to Proxy Chef Server with Apache,
- Deploying OpenStack with Chef,

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

- and others.

For a broader review of Chef, there's a Webcast of "Why Chef" which is aimed at users new to Chef. And, if you'd like, following registration, we'll be happy to send you the Open Training Materials for free.



Why do I need RubyGems installed from source?

You don't *have* to install RubyGems from source. However, Opscode recommends it when installing Chef from RubyGems. Installing from source ensures consistency across platforms, as we can reasonably expect it to install in the same locations. RubyGems doesn't follow Filesystem Hierarchy Standard on Linux, to which Debian and Ubuntu adhere, which may cause pathing problems that are sometimes difficult to track down.

For example on Debian and Ubuntu, using the RubyGems distro package doesn't install binary programs to `/usr/bin`, instead they live in `/var/lib/gems/1.8/bin`, which isn't in the default user `$PATH`. This causes confusion for some users, as when they install Chef they can't run `chef-client`, `chef-solo` or even `chef-server`. Thus we recommend installing RubyGems from source when installing Chef from RubyGems on these distributions.

What if I have multiple versions of the same Gem?

Having multiple versions of the same gem can prove problematic. Ruby can have unpredictable issues and responses in that scenario, which can affect your environment and the success of Chef runs.

There are some knife-plugins that require a specific gem version, if that is the case, it will be specified in the plugins gemspec. To remove a specific gem `sudo gem uninstall GEMNAME -v VERSION#`. To remove all older versions of any gems, run `sudo gem cleanup`. It will leave only the newest versions of installed gems.

Where can I find Chef's logs?

Chef logs to standard out (STDOUT) for all its programs `chef-client`/`chef-solo`, `chef-indexer`, `chef-server`. This can be changed by adjusting the configuration value of "log_location" in the appropriate configuration file. For example,

/etc/chef/client.rb

```
log_location "/var/log/chef/client.log
```

runit logs

When installing Chef via the bootstrap recipe, Chef is configured to use the runit init system. In this case your logs are located in `/etc/sv/PROGRAM/log/main/current`

How can I get more log information?

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

By default, Chef's programs have `error` level logging. To get more information, turn on `info` level. This can be done for a single execution of the program with `-l` or by changing the `verbose` value to `:info` (the colon is significant). For example,

```
sudo chef-client -l
```

/etc/chef/client.rb

```
verbose :info
```

You can also setup Chef to have `debug` level logging. This can be done with two executions of the program with `-V` or by changing the verbose value to `:debug` (The colon is significant). For example,

```
sudo chef-client -l debug
```

/etc/chef/client.rb

```
log_level :debug
```

If you are requesting help on IRC or the Mailing List, you may be asked to provide debug log output.

Connection refused for localhost:5984, what is listening on 5984?

By default, CouchDB listens on port 5984. You need to make sure it is installed and running for your Chef Server. The best way to make sure it is running is to make an http request:

```
$ curl http://localhost:5984/
{"couchdb":"Welcome","version":"0.8.0-incubating"}
```

Why doesn't Chef work on XYZ platform?

Chef detects the platform with Ohai, and looks up providers to use for various resources in the platform.rb file. If a platform is not defined in this file with specific providers, the `:default` values are used for each resource. Not all resources need a platform-specific provider. Chef allows you to easily create new providers within your cookbooks, but we also appreciate new contributions to Chef.

How do I delete a node with an empty name?

In some situations a node might get saved to the Chef Server with an empty name. When viewed in the Node List in the WebUI, the name is completely blank. This can be fixed by removing the `node_` document from the CouchDB. You can do this in Futon, the CouchDB webui, or by interacting with the CouchDB API with a tool such as curl.

Futon - CouchDB WebUI

- Navigate to http://localhost:5984/_utils on the chef server with your web browser. Set up an SSH tunnel if necessary.

  ```
  ssh -L 5984:localhost:5984 chef.example.com
  ```

- Click on 'chef' in the database list.
- Click on the document 'node_' with ID: node_.
- Click 'Delete Document'.

Refresh the Chef Server nodes list and the node will be gone.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

CouchDB API via Curl

Removing the node via the CouchDB API with a tool like curl is very straightforward.

- Find the revision of the blank node_ document, otherwise CouchDB will display a conflict error.

```
% curl http://localhost:5984/chef/node_
{"_id":"node_","_rev":"5-3246d1733ae522bd637ee8f437b99df4","name":"","attributes":{},"json_clas
s":"Chef::Node","run_list":["recipe[]"],"chef_type":"node"}
```

- Delete the node.

```
% curl -X DELETE http://localhost:5984/chef/node_?rev=5-3246d1733ae522bd637ee8f437b99df4
{"ok":true,"id":"node_","rev":"6-78b988c185c7b50fe8b74ba5a1b03f68"}
```

- The node will now show as deleted.

```
% curl http://localhost:5984/chef/node_
{"error":"not_found","reason":"deleted"}
```

Also, if you refresh the Chef Server WebUI nodes list and the node will be gone.

Why is my Chef run crashing on a require statement for a gem that should have been installed via Chef?

Recipes are evaluated during the node Convergence phase of the Chef run, which happens in two phases. The first phase is 'compile' phase where ruby code (like `require 'sys-filesystem'`) is evaluated, and the resources are added to the resource collection. The second is the 'execute' phase where the actions are taken on the resources in the collection. (See Anatomy of a Chef Run for more information).

In order to use a gem in the same run that the gem is installed, you have to create that resource in the compile phase. To install the gem in the compile phase, use this:

```
r = gem_package "sys-filesystem" do
  action :nothing
end
r.run_action(:install)
Gem.clear_paths
require "sys/filesystem"
```

See our Opscode blog post for additional information.

Is there a recommended approach for managing individual credentials and knife.rb when multiple developers will be using the same chef-repo?

We recommend checking the knife.rb into your chef-repo (CHEF_REPO/.chef/knife.rb) and then having user specific (or sensitive) values dynamically filled using a series of environment variables. Since the knife.rb file is just Ruby this is pretty easy to do. For example:

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
current_dir = File.dirname(__FILE__)
    user = ENV['OPSCODE_USER'] || ENV['USER']
    node_name                user
    client_key               "#{ENV['HOME']}/.chef/#{user}.pem"
    validation_client_name   "#{ENV['ORGNAME']}-validator"
    validation_key           "#{ENV['HOME']}/.chef/#{ENV['ORGNAME']}-validator.pem"
    chef_server_url          "https://api.opscode.com/organizations/#{ENV['ORGNAME']}"
    cache_type               'BasicFile'
    cache_options( :path => "#{ENV['HOME']}/.chef/checksums" )
    cookbook_path            ["#{current_dir}/../cookbooks"]
    cookbook_copyright "Your Company, Inc."
    cookbook_license "apachev2"
    cookbook_email "cookbooks@yourcompany.com"

    # all your credentials are belong to us

    # Amazon AWS
    knife[:aws_access_key_id] = "#{ENV['AWS_ACCESS_KEY_ID']}"
    knife[:aws_secret_access_key] = "#{ENV['AWS_SECRET_ACCESS_KEY']}"

    # Rackspace Cloud
    knife[:rackspace_api_username] = "#{ENV['RACKSPACE_USERNAME']}"
    knife[:rackspace_api_key] = "#{ENV['RACKSPACE_API_KEY']}"
```

This approach allows you to control who has a copy of the org validation.pem but allows you to share your chef code with the company as a whole (yeah devops!). Some companies even have multiple organizations and switching between them is as easy as cd-ing into a different chef repository!

May I use RVM to install Ruby and/or RubyGems that the chef-client runs under?

The recommended method to install chef as a client is as a RubyGem with RubyGems installed from source, or from our APT repository on Ubuntu and Debian.

We don't recommend using RVM to install Ruby/RubyGems that the chef-client runs under, because that can cause system PATH and Ruby loadpath errors. The knife bootstrap subcommand makes installing Chef much easier with the templates that it ships with.

Many however do use RVM to install Ruby on the workstations where they use knife, but that is a separate matter than running Chef on client nodes. ***Note:*** If you're installing Chef under RVM on your workstation for using knife and managing the chef repository (chef-repo), you don't need to use sudo, and probably shouldn't as it may result in pathing differences to install the gems. The problem will be sudo scrubbing the environment variables rvm uses to keep track of which ruby and rubygems installation you're using. For example, Opscode star Dan Deleo's rvm installation is using the following:

```
rvm_path=/Users/ddeleo/.rvm
    rvm_selfcontained=1
    rvm_version=1.1.0
    RUBY_VERSION=ruby-1.9.2-p0
    GEM_HOME=/Users/ddeleo/.rvm/gems/ruby-1.9.2-p0
    GEM_PATH=/Users/ddeleo/.rvm/gems/ruby-1.9.2-p0:/Users/ddeleo/.rvm/gems/***@global
    MY_RUBY_HOME=/Users/ddeleo/.rvm/rubies/ruby-1.9.2-p0
    IRBRC=/Users/ddeleo/.rvm/rubies/ruby-1.9.2-p0/.irbrc
    rvm_ruby_string=ruby-1.9.2-p0'
```

If you run `sudo env` you can see that these variables were "scrubbed" by sudo. In order to work around the problem, try running the command as `sudo -E chef-client`. If that doesn't work, you might need to run a shell as root with the rvm environment, by running `sudo -E bash` and then just `chef-client`. Or, you should be able to simply do `gem install chef` as your normal user without sudo and it will install the gems under your default ruby in rvm.

Alternatively, if you'd like to run chef client on a machine who's Rubies are under the control of RVM, you'll need to accept installing the chef gem in the global gemset. It'll be available in all your gemsets. The caveat here is you might find your self with gem version conflicts.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
rvm gemset use global
    gem install chef
    rvm gemset use <your-other-gemset>
    sudo -Es 'bash -lc chef-client'
```

That should work in place of the two step `sudo -E bash` then `chef-client` method put forth above, but again, you may find that you are dealing with gem version conflicts.

What if an Opscode service appears to be unavailable? For instance, what if I'm using Hosted Chef and I am unable to acces it?

Hosted Chef is built from the ground up by Opscode to be fully multi-tenant, horizontally scalable, and highly available. Opscode has several external monitors of it, and all Opscode web service functions and services, enabling rapid response to any emergent availability issues.

Occasionally, periodic maintenance is necessary to maintain efficient functionality and performance for all services. Any planned maintenance is announced in advance via the Opscode Status site. Emergent issues in Hosted Chef availability, or that of other Opscode services such as Cookbooks, Open Source Tickets or this Chef Wiki will also be posted at Opscode Status.

Opscode customers who use **Twitter** can also receive availability status announcements by following Opscode Status on Twitter. Please also follow Opscode on Twitter for company and product news and information. If you are having issues in accessing any shared Opscode services, and find no announcement at Opscode Status or through following Opscode Status on Twitter, please send an email to support@opscode.com describing the issue you are experiencing.

Note: Help on issues specific to running your own Chef Server environment and use of Chef should be addressed through Support Resources.

How can I ensure a notified action is processed, regardless of the full chef-client run completing?

Delayed notifications do not run after an error by design. An example supporting this behavior is the case where you are making changes to the web server config but something fails. A restart would bring the service down whereas fixing the problem and re-running chef-client successfully will not.

The general recommendation to alleviate this problem during development is to develop in an environment where you can occasionally as a matter of practice "start from scratch". Much more difficult to detect errors can happen with this same "broken first run" problem.

In particular cases, you may want to ensure the action gets taken immediately, which will ensure the rebuild happens regardless of what happens later in the run.

```
notifies :run, "execute[RECIPE]", :immediately
```

The Resources Notification Option page has more detail and additional examples.

How do I create a recipe that will only execute an action on the first run of chef-client?

Creating a "First Run Only" Resource provides information on how to accomplish this - however it is an approach that should be used sparingly. Most resources are idempotent by construction and do not require the methods discussed. Whenever possible, it is best to use an idempotent resource.

How do I extend a base cookbook to account for the use of the IUS RPMs?

We would suggest just modifying the package names in your copy of the COOKBOOK::package recipe. Our cookbooks are meant to be a starting point, and we expect community members to modify then and personalize for their infrastructure. That's why we have build in tools like `knife cookbook site install` which allow easy upstream tracking.

That said, there are approaches to add logic to change the package names based on the YUM sources. The following examples use the PHP Cookbook.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
pkgs = value_for_platform(
  [ "centos", "redhat", "fedora" ] => {
    "default" => %w{ php53 php53-devel php53-cli php-pear }
  },
  [ "debian", "ubuntu" ] => {
    "default" => %w{ php5-cgi php5 php5-dev php5-cli php-pear }
  },
  "default" => %w{ php5-cgi php5 php5-dev php5-cli php-pear }
)

pkgs.each do |pkg|
  pkg += 'u' if ::File.exists?('/etc/yum.repos.d/ius.repo')
  package pkg do
    action :install
  end
end

template "#{node['php']['conf_dir']}/php.ini" do
  source "php.ini.erb"
  owner "root"
  group "root"
  mode "0644"
end
```

The main thing to take note of in the above example is the new line in the `pkgs.each` block. This basically searches for the existence of the `ius` yum source and then appends the u to the package name.

Alternatively:

```
packages = %w{ php53-common php53 php53-devel php53-cli php-pear }

if File.exists?("/etc/yum.repos.d/ius.repo")
    packages = %w{ php53u-common php53u php53u-devel php53u-cli php53u-pear }
end

pkgs = value_for_platform(
  [ "centos", "redhat", "fedora" ] => {
    "default" => packages
  },
  [ "debian", "ubuntu" ] => {
    "default" => %w{ php5-cgi php5 php5-dev php5-cli php5-common php-pear }
  },
  "default" => %w{ php5-cgi php5 php5-dev php5-cli php5-common php-pear }
)

pkgs.each do |pkg|
  package pkg do
    action :install
  end
end
```

How can I run the chef-client periodically?

If you want, you can run the Chef Client as a persistent daemon. To do this, make your startup script for the chef client execute something like:

```
$ chef-client -i 3600 -s 600
```

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

The `-i` option provides an *Interval* - it's how often the Chef client will attempt to wake up and Converge this node. The `-s` option is the *Splay* - a random piece of time added to the interval, which helps avoid the thundering herd problem.

The chef-client cookbook available on the community site provides an easy way to manage the chef-client configuration with chef itself! *See Chef Client for additional information.*

What is the best way to manage software that needs regular updates or patches?

First, you will want to determine the type of change that needs to be made regularly.

- If the change is to a behavior, such as a patch, you will want to control this with a recipe or role.
- If the change is to data, such as with DNS or the current software version, you will want to control this with a data bag or attribute.

The best way to apply regular patches is to specify the actual version in the package that is run over time, and change that version to a new one after testing it. If you do this frequently you can data-drive this change by dropping an attribute into the role that includes this cookbook. When a new version comes out:

1. drop the version string into that role's development environment list and test it, then
2. update the version on the role's production environment run_list.

This way you are only changing a role. It is possible to apply patches with 'action :upgrade' but this will control when it is updated and not necessarily what version it is updated to.

The best way to apply data updates like DNS is to separate that data into a data bag and then render from the data bag into a zone file via the recipe and chef template.

After applying the update you could run something like this:

```
knife ssh "role:nameserver" "sudo chef-client"
```

To select the nameservers in your fleet, with appropriate additional command line args for authentication, parallelism, etc. More information on the knife ssh command can be found in the Knife Wiki Article.

Any tuning recommendations for those with a large LDAP user base?

As Ohai gathers information about your operating system while it is running, you can make some modifications to the chef client configuration to increase performance when dealing with large installations or large user bases.

1. Modify nodes.rb to reduce automatic_attrs size

```
# Serialize this object as a hash

        def to_json(*a)
          result = {
            "name" => name,
            "chef_environment" => chef_environment,
            'json_class' => self.class.name,
    #        "automatic" => automatic_attrs,

            "normal" => normal_attrs,
            "chef_type" => "node",
            "default" => default_attrs,
            "override" => override_attrs,
            "run_list" => run_list.run_list
          }
          result["_rev"] = couchdb_rev if couchdb_rev
          result.to_json(*a)
    end
```

2> Modify Ohai to read passwords directly

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

```
This will be addressed with [Ohai-165](http://tickets.opscode.com/browse/OHAI-165).  Pending that,
adding the following to the client.rb file would be beneficial.

    Ohai::Config[:disabled_plugins] = [ "passwd" ]
```

3> Update standard bootstrap template

```
This speeds up chef-client execution, because LDAP servers can return a huge number of passwords.

    (
    cat <<'EOP'
    <%= config_content %>
    EOP
    ) > /etc/chef/client.rb

    echo "<%= 'Ohai::Config[:disabled_plugins] = [ \"passwd\" ]' %>" >>
    /etc/chef/client.rb
```

4> Disable undesirable Ohai plugins

If you do not want some Ohai plugins to run, you can disable them by doing something like the following within /etc/chef/client.rb:

```
Ohai::Config[:disabled_plugins] << 'darwin::system_profiler' << 'darwin::kernel' <<
'darwin::ssh_host_key' << 'network_listeners'
```

(You may desire to choose other plugins that the ones in the above example.) See Disabling Ohai Plugins for additional information.

I have one-off nodes in my environment and need to manage them uniquely. How do I do this with Chef?

This is a common question from people that follow a pattern of naming schemes for hostnames and/or fully-qualified domain names. Because the Chef Server API provides a search interface to rich metadata about Chef nodes, we do not recommend treating anything as a "Unique and Beautiful Snowflake." Instead, we recommend that nodes be treated as ephemeral, and that the discovered attributes (by Ohai) and roles (assigned in the run-list) describe the node.

If a node is truly unique, give it a role that describes its functionality, and use node attributes set in the role as required to be used in its recipes that affect the required "one-off" configuration. *That way if the node goes away for some reason (deleted by accident, hardware fails, etc), its functionality can be replaced easily by applying the role to another node.*

How do I recreate my validator client when using Hosted Chef?

If you accidentally delete your validator client, you can recreate it using the following directions. In these directions, ORGNAME should be replaced by your organizations short name.

1. In the management console, create a new client with the name ORGNAME-validator (https://manage.opscode.com/clients/new). Save the private key of this client in a safe place.
2. In the management console, go to the Clients page.
3. Click "Edit Permissions" (https://manage.opscode.com/clients/_acl)
4. Check both LIST and CREATE for ORGNAME-validator
5. Click "Save"

The private key of the client you created in step 1 is now your organization's validator key.

Are there common errors and error messages that I should be aware of?

Common Errors contains errors that arise with Chef Client and Knife and their suggested fixes. If your issue is neither on this page, nor on Com

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

mon Errors, please return to Support to seek assistance.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

User Environment PATH Sanity

As mentioned on the Installation page, you need to have a sane PATH set up.

Neither Chef nor Ohai hardcodes paths when running system commands, because we don't want to make assumptions about where programs might live for every single platform. For most Unix and Linux systems, the following should cover everything that we'll need to run for Providers or Ohai Plugins.

$PATH in Bourne shell derivatives

```
export PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
```

A Note About Sudoers

This comment on OHAI-87 is worth repeating here, and may be applicable on other systems that do an env_reset in /etc/sudoers.

The main issue is the default /etc/sudoers on CentOS does an env_reset, and if the user's PATH is reset to "/usr/bin:/bin". To fix for the local user, add "/usr/sbin:/sbin" to the user's PATH, and run sudo with -E.

```
echo $PATH
/usr/local/bin:/usr/local/sbin:/bin:/sbin:/usr/bin:/usr/sbin
sudo -E chef-client
```

Chef/Ohai work fine when executed by a default root shell on CentOS. Also note that Opscode's sudo cookbook will overwrite sudoers without the env_reset, so once the user's PATH is set, sudo won't strip it on execution.



Chef 0.10.0 Client Enforces Path Sanity

As of Chef 0.10.0, chef-client will enforce path sanity by default. If you don't want this, you can turn it off with enforce_path_sanity false in your config file.

 This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License