# Kubernets Essentials

**Rajesh Kumar**

**DevOps Architect**

**@RajeshKumarIN | www.RajeshKumar.xyz**

# What is Kubernetes?

Kubernetes (k8s) is an open source platform for automating container operations such as

deployment,

scheduling and

scalability across a cluster of nodes.

Docker as a low level component used internally by Kubernetes to deploy containers. Not just Docker, but Kubernetes also supports Rocket, another container technology.

- Kubernetes is a mere two years old (as a public open source project) whereas

- Docker Swarm is younger than Kubernetes, and it comes with the backing of the center of the container universe, Docker Inc.

# Why Kubernets

# Limitation of Docker

- Containers are lives only inside the Linux Kernal
  - Containers are kernal's store groups
  - Containers are SELINUX
  - Containers are Kernal's Namespaces

# Kubernets Popularity

- Kubernetes is one of the top projects on GitHub: in the top 0.01 percent in stars and No. 1 in terms of activity.
- While documentation is subpar, Kubernetes has a significant Slack and Stack Overflow community that steps in to answer questions and foster collaboration, with growth that dwarfs that of its rivals.
- More professionals list Kubernetes in their LinkedIn profile than any other comparable offering by a wide margin.
- Perhaps most glaring, data from OpenHub shows Apache Mesos dwindling since its initial release and Docker Swarm starting to slow. In terms of raw community contributions, Kubernetes is exploding, with 1,000-plus contributors and 34,000 commits -- more than four times those of nearest rival Mesos.

# Kubernetes orchestrates

Kubernetes orchestrates your containers so together they are performing a Symphony. This could be anything from, but not limited to, the following:

- automate the deployment and replication of containers,
- scale in or out containers on the fly,
- organise containers in groups and provide load balancing between them,
- easily roll out new versions of application containers,
- provide container resilience, if a container dies it gets replaced, etc..

# In Fact

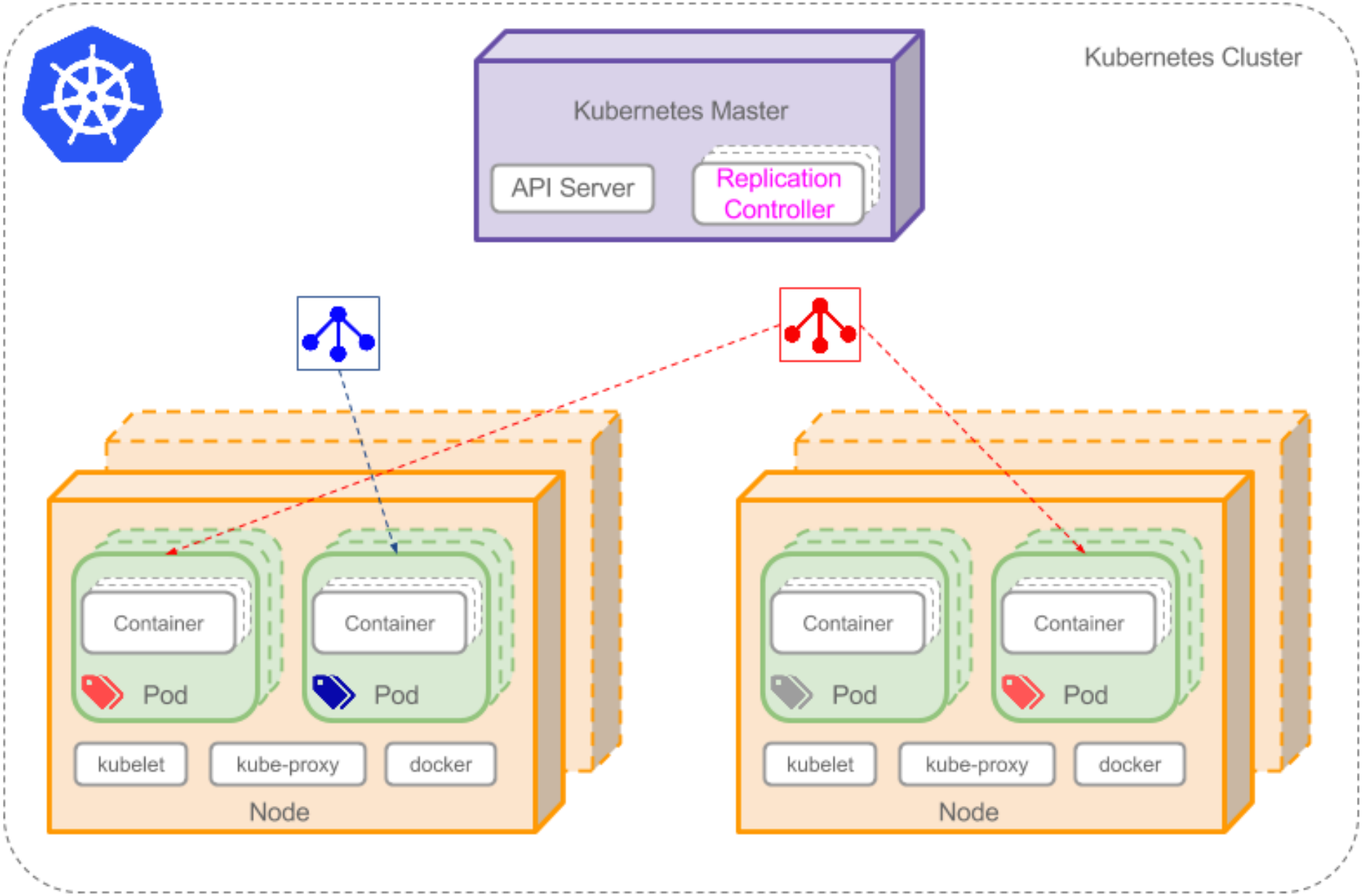
- In fact, with Kubernetes you can deploy a full cluster of multi-tiered containers (frontend, backend, etc…) with a single configuration file and a single command:
  - $ kubectl create -f single-config-file.yaml

  kubectl is a command-line program for interacting with the Kubernetes API.

# NOW LET’S INTRODUCE SOME OF THE KEY CONCEPTS.

# Cluster

A cluster is a group of nodes, they can be physical servers or virtual machines that has the Kubernetes platform installed.



- Looking at the diagram you can spot the following components, I used icons to represent Service & Label:
  - Pods
  - Containers
  - Label(s) (label)
  - Replication Controllers
  - Service (service)
  - Nodes
  - Kubernetes Master

# Pods

- Pods are scheduled to Nodes and contain a group of co-located Containers and Volumes. Containers in the same Pod share the same network namespace and can communicate with each other using localhost.

# Labels

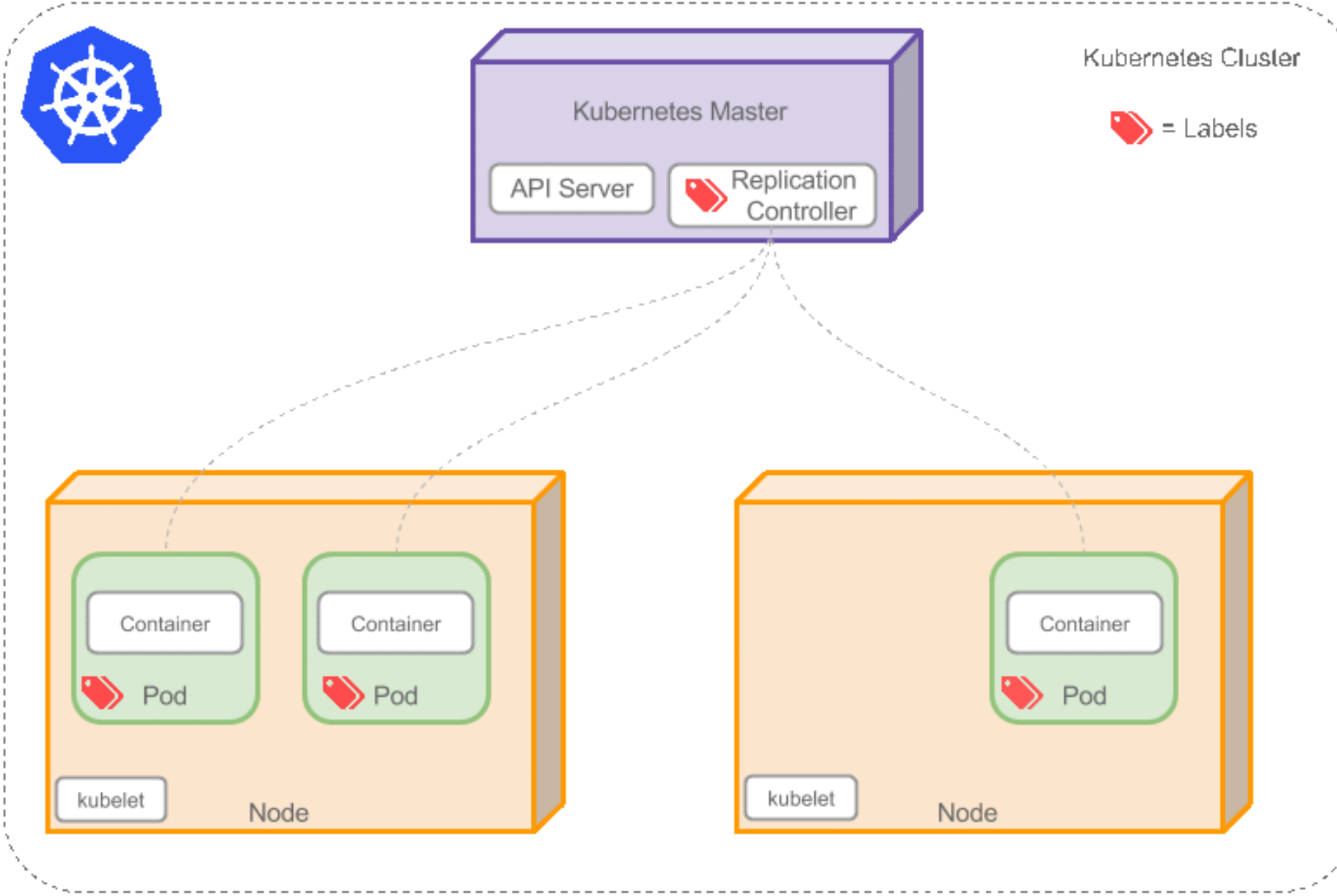
- A Label is a key/value pair attached to Pods and convey user-defined attributes. For example you might create a ‘tier’ and an ‘app’ tags to tag your containers by applying the Labels (tier=frontend, app=myapp) to your frontend Pods and Labels (tier=backend, app=myapp) to backend Pods.
- You can then use Selectors to select Pods with particular Labels and apply Services or Replication Controllers to them.

# Replication Controllers

- *Do I create Pods manually, what if I want to create a few copies of the same Pod, do I have to create each one individually, can I group Pods into logical groups?*

Replication Controllers ensure the specified number of Pod “replicas” are running at any one time. If you created a Replication Controller for a Pod and specified 3 replicas, it will create 3 Pods and will continuously monitor them.

If one Pod dies then the Replication Controller will replace it to maintain a total count of 3.


Kubernetes Cluster
= Labels
Kubernetes Master
API Server
Replication Controller
Container
Pod
Container
Pod
kubelet
Node
Container
Pod
kubelet
Node

- If the Pod that died comes back then you have 4 Pods, consequently the Replication Controller will terminate one so the total count is 3.

- If you change the number of replicas to 5 on the fly, the Replication Controller will immediately start 2 new Pods so the total count is 5. You can also scale down Pods this way, a handy feature performing rolling updates.

# When creating a Replication Controller you need to specify two things:

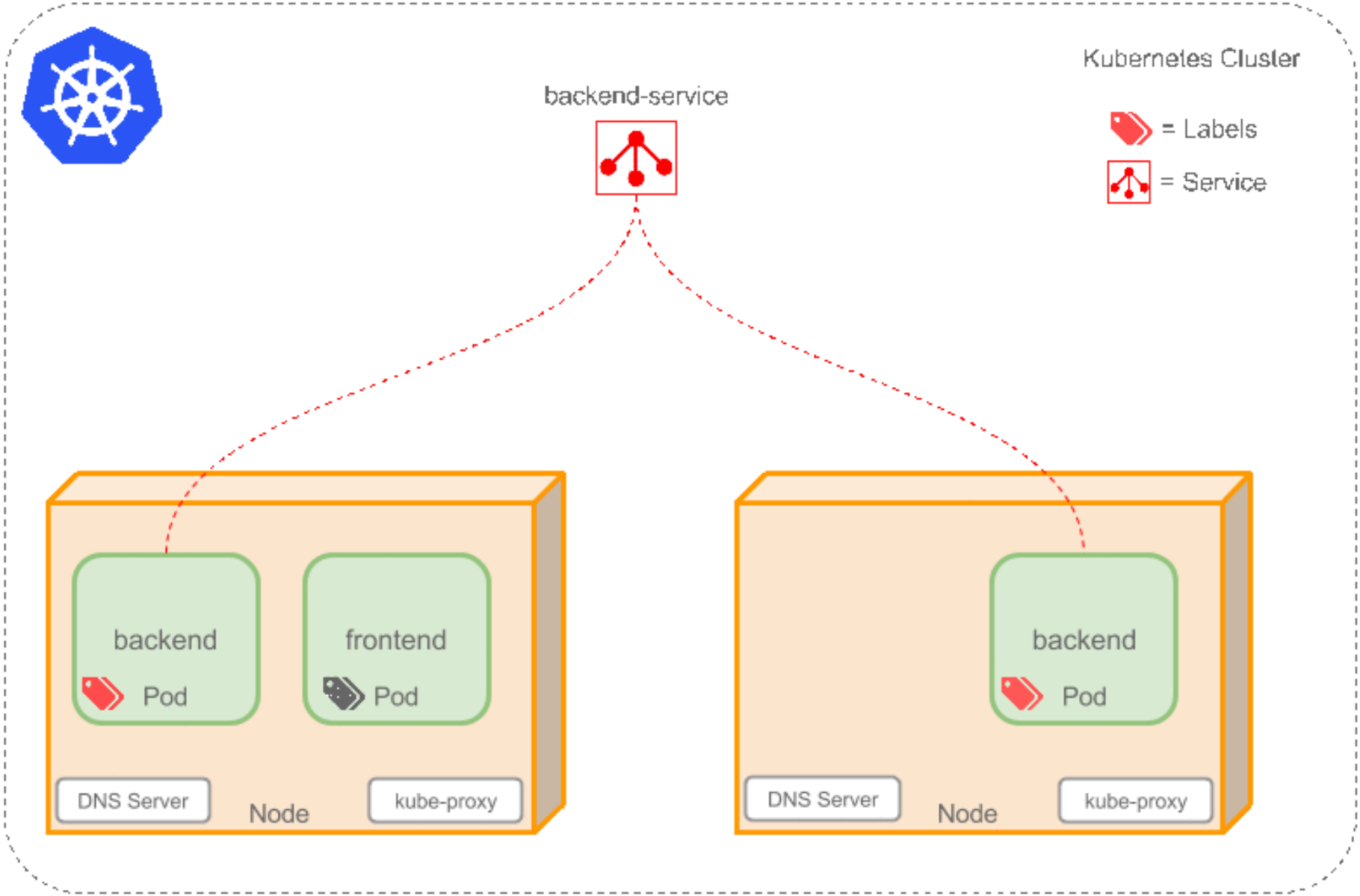
- Pod Template: the template that will be used to create the Pods replicas.
- Labels: the labels for the Pods that this Replication Controller should monitor.

So now you have created a few replicas of a Pod, how do you load balance between them? **Enter Services.**

# Services

*If Pods are ephemeral and their IP address might change if they get restarted how can I reliably reference my backend container from a frontend container?*

- Service is an abstraction that defines a set of Pods and a policy to access them. Services find their group of Pods using Labels. Because Services are abstraction you don’t usually see them in diagrams which makes the concept hard to understand.


Kubernetes Cluster
backend-service
= Labels
= Service
backend
Pod
frontend
Pod
DNS Server
Node
kube-proxy
backend
Pod
DNS Server
Node
kube-proxy

Now, imagine you have 2 backend Pods and you defined a backend Service named ‘backend-service’ with label selector (tier=backend, app=myapp). Service backend-service will facilitate two key things:

• A cluster-local DNS entry will be created for the Service so your frontend Pod only need to do a DNS lookup for hostname ‘backend-service’ this will resolve to a stable IP address that your frontend application can use.

• So now your frontend has got an IP address for the backend-service, but which one of the 2 backend Pods will it access? The Service will provide transparent load balancing between the 2 backend Pods and forward the request to any one of them (see the animated diagram below). This is done by using a proxy (kube-proxy) that runs on each Node.

- There is a special type of Kubernetes Services called ‘LoadBalancer’, which is used as an external load balancer to balance traffic between a number of Pods.

# Nodes

- A node is a physical or virtual machine that acts as a Kubernetes worker, used to be called Minion. Each node runs the following key Kubernetes components:
  - Kubelet: is the primary node agent.
  - kube-proxy: used by Services to proxy connections to Pods as explained above.
  - Docker or Rocket. The container technology that Kubernetes uses to create containers.

# Kubernetes Master

- The cluster has a Kubernetes Master . The Kubernetes Master provides a unified view into the cluster and has a number of components such the Kubernetes API Server. The API Server provides a REST endpoint that can be used to interact with the cluster. The master also includes the Replication Controllers used to create and replicate Pods.

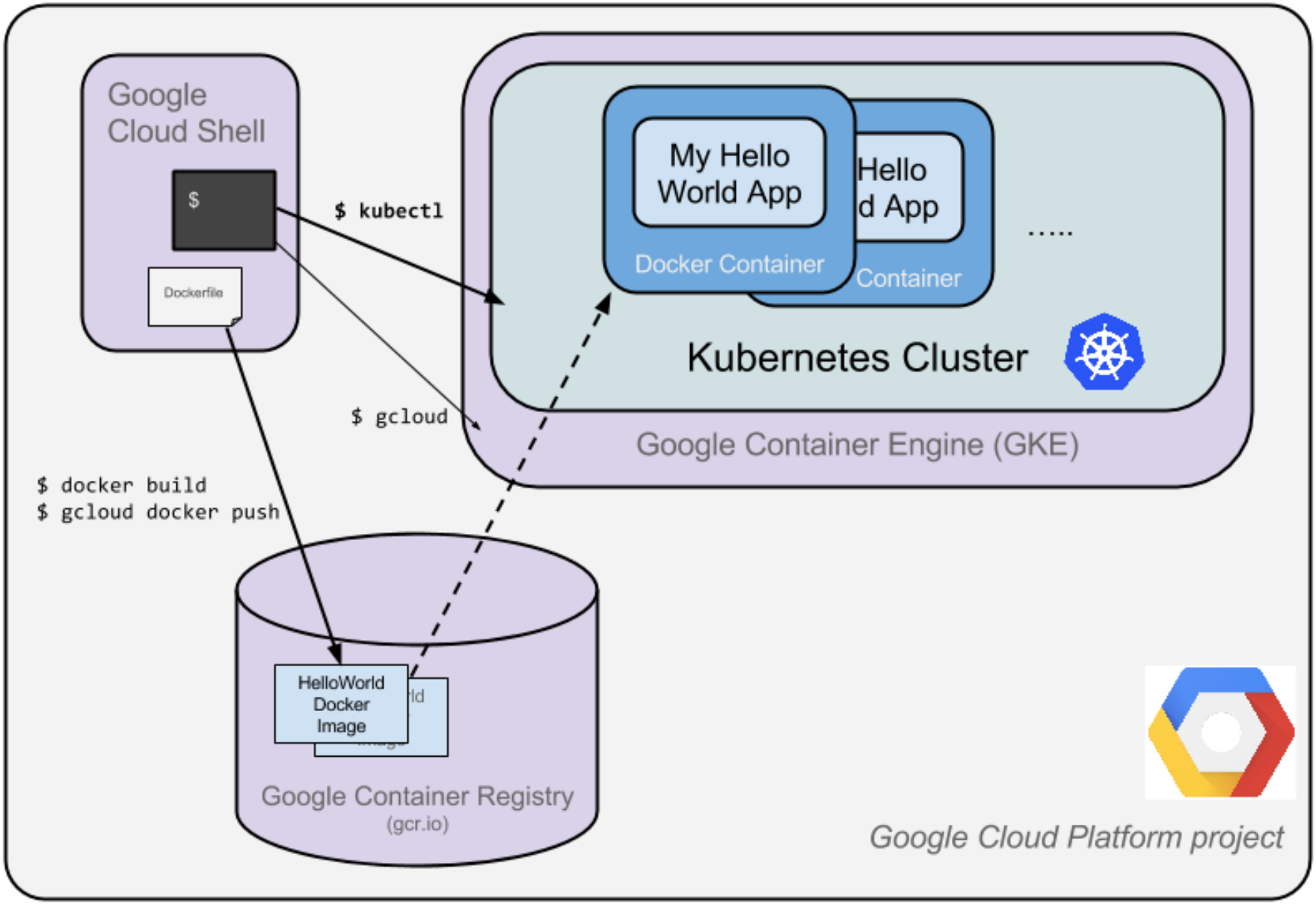
# *Minion*

- A slave that runs tasks as delegated by the user and Kubernetes master.

# *kubecfg*

- the command line config tool

# LAB -1 - BASICS


Google
Cloud Shell
$
Dockerfile
$ kubectl
$ gcloud
$ docker build
$ gcloud docker push
My Hello
World App
Docker Container
Hello
d App
Container
.....
Kubernetes Cluster
Google Container Engine (GKE)
HelloWorld
Docker
Image
Google Container Registry
(gcr.io)
Google Cloud Platform project

# LAB 1
# https://goo.gl/ZN67CC

# LAB 2

# https://goo.gl/Lr4xRx

# Good reading

- http://omerio.com/2015/12/18/learn-the-kubernetes-key-concepts-in-10-minutes/
- https://www.youtube.com/watch?v=9Wzw84Q-8yc
- https://blog.risingstack.com/moving-node-js-from-paas-to-kubernetes-tutorial/
- https://www.ctl.io/developers/blog/post/what-is-kubernetes-and-how-to-use-it/