

# Gerrit Performance Tuning

How to Properly Tune and Size your Gerrit Backend



# Git Sizing / Performance Tuning - FAQ

- How many servers will I need?
- Which cloning protocols to offer?
- How to set those gazillion gerrit.config options?
- How many CPUs and how much RAM will I need?
- What the heck is pack size?
- How often should you run garbage collection?
- Does it make any difference whether I go with a native Git or JGit based backend?
- How do you handle hundreds of polling CI users without compromising performance for your human end users?
- What about clustering and replication?

“It Depends ...”

One Size does not fit all.



# Responsibility of Gerrit Ops Engineer

## TODAY'S DEVOPS

DEV + OPS

### PART DEV

#### Application Performance

Modern developers use APM tools to decrease latency, have complete visibility into code, database, caches, queues, and third-party service.

#### End User Analytics

A great developer understands end users have the best feedback and analytics play an enormous part of understanding users. Developers are constantly monitoring end user latency and checking performance by devices and browsers

#### Quality Code

Developers need to ensure their deployment and new releases don't implode or degrade the overall performance.

#### Code-Level Errors

When you have a large distributed application it is vital to lower MTTR by finding the root cause of errors and exceptions

### PART OPS

#### Application Availability

The applications need to be up and running and it's Ops responsibility to ensure uptime and SLAS are in order.

#### Application Performance


Classic Ops generally rely on infrastructure metrics - CPU, memory, network and disk I/O etc while modern Ops correlate all of those metrics with application metrics to solve problems 10x faster.

#### End User Complaints

The goal is to know and fix problems before end users complain, reduce the number of support tickets, and eliminate false alerts.

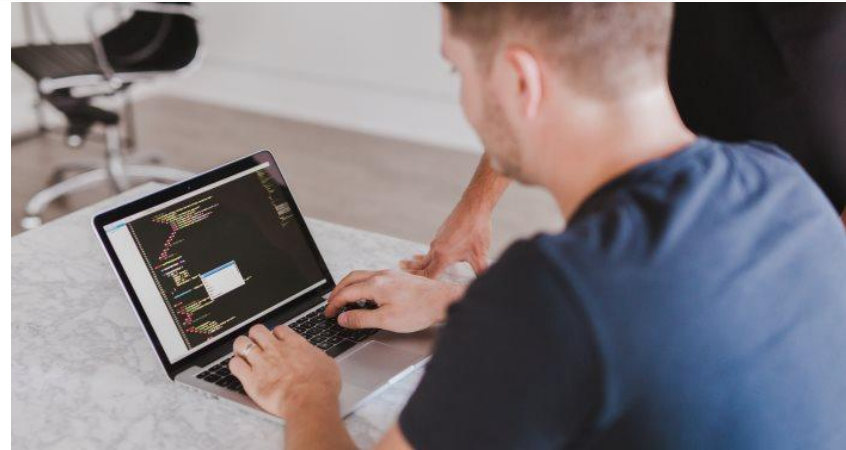
#### Performance Analytics

Automatically generated baselines of the metrics help Ops understand what has changed and where to focus their troubleshooting efforts. Alerts are based upon deviation from observed baselines.



# Typical Ops Engineer in Today's world

- Jack of all trades
- Responsible for dozens of applications
- No Gerrit expert knowledge
- No Java expert knowledge
- Basic Git knowledge
- No access to special HW
- Limited test bed (not same load pattern as on production)
- No access to Gerrit multi master / GFS technology
- No overview about all their user base (tens of thousands of developers in different geographies)

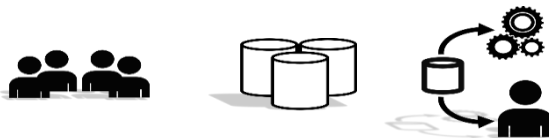


# Challenge

- Tuning advice that is
  - actionable
  - not “one size fits all”
  - targeted at ops people with no expert Gerrit / JVM knowledge
  - only uses easy to measure factors for its recommendations
  - does not require special HW or test beds
  - not depending on proprietary Gerrit extensions/technology
- Keep Motivation up to go through all of this

# Gerrit Performance Tuning in 5 Steps

1. Get your numbers



2. Size your hardware



3. Tune your gerrit.config

```
• File: gerrit.config
  • Section: accounts
  • Section: address:cnx
  • Section: auth
  • Section: cache
  • Section: change
  • Section: changeMerge
  • Section: commentLink
  • Section: container
  • Section: container
  • Section: core
  • Section: database
  • Section: download
  • Section: gc
  • Section: gerrit
  • Section: group
  • Section: groups
  • Section: hooks
  • Section: http
  • Section: httpd
  • Section: index
  • Section: ldap
  • Section: ldap
  • Section: ldap
  • Section: ldap
  • Section: pack
  • Section: plugin
  • Section: repository
  • Section: rules
  • Section: execution
  • Section: sso:band
  • Section: site
  • Section: ssh:alias
  • Section: sshd
  • Section: sshd
  • Section: theme
  • Section: track:tag
  • Section: transfer
  • Section: upload
  • Section: user
```

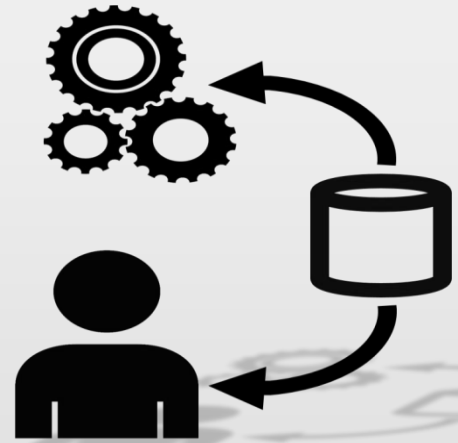
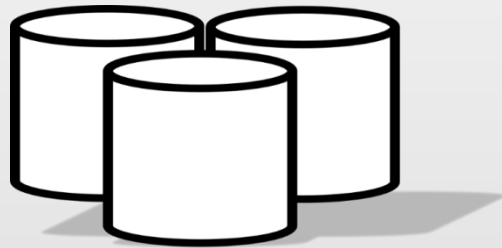
4. Configure Garbage collection



5. Deal with heavy CI load



# 1. Get Your Numbers



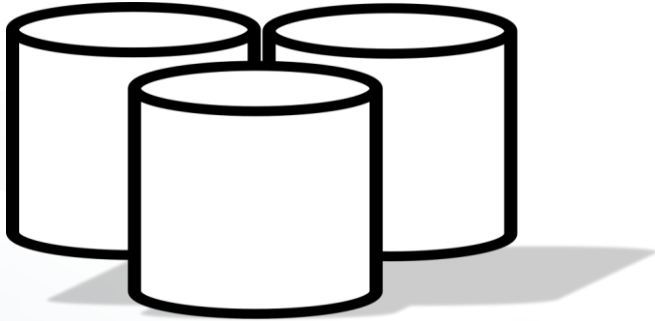
# Step 1: Get your numbers



## Number Of Users

- The number of users is only an indirect factor for Gerrit tuning as most Git operations are done completely offline.
- The more users you have, the more repositories and push/fetch requests you will probably encounter.
- The majority of load is typically caused by build systems (CI). The biggest enterprise instance we have seen has 15k active users.

# Step 1: Get your numbers



## Number Of Repositories

- The number of repositories (Gerrit projects) determines how much disk space you need.
- We have seen instances with more than 10k repositories but would not recommend more than 2500 per server.

# Step 1: Get your numbers

**SSH**

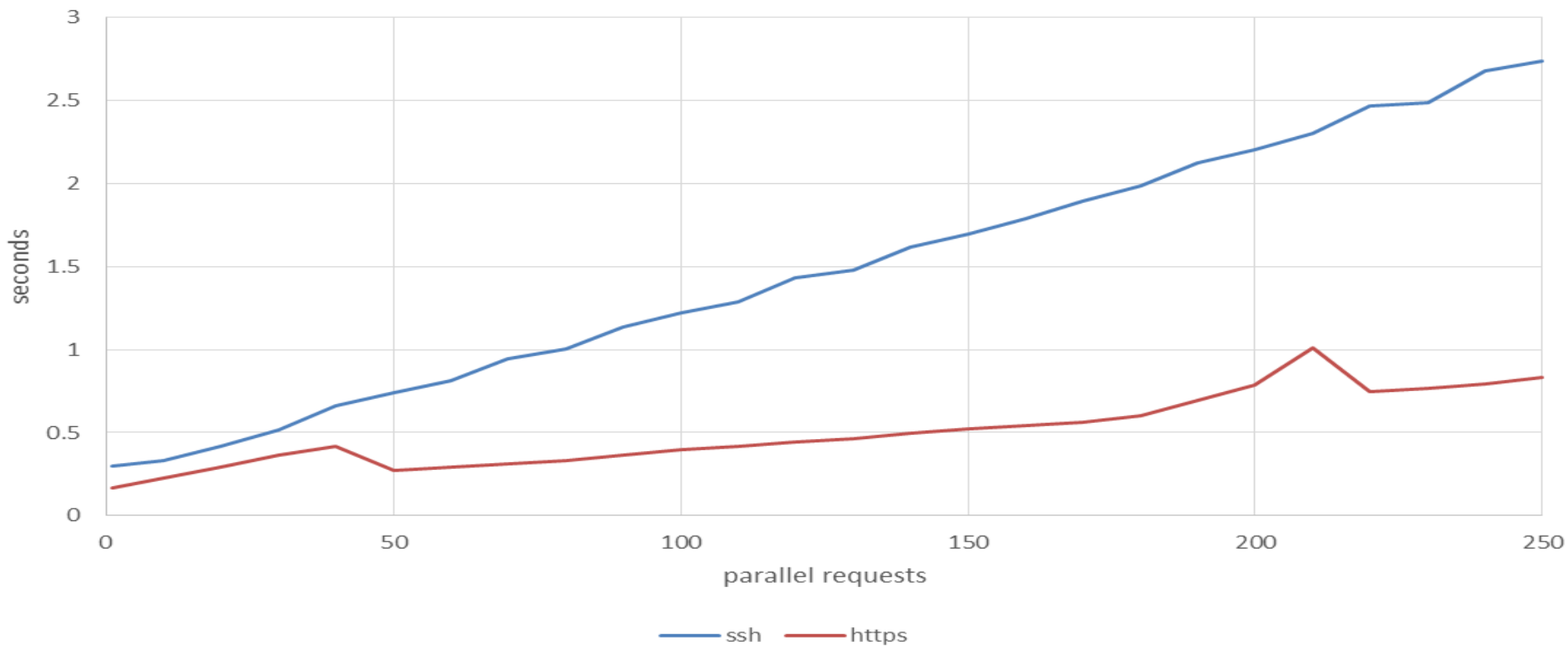
**HTTP**

**Protocol**

- ssh allows you to use public key cryptography which is stronger than passwords
- ssh is recommended for CI users as this allows push based notifications (see step 5).
- http(s) seems to perform better if the majority of the operation time is the connection request itself (not much data transferred, no heavy IO)
- Hybrid approaches are possible

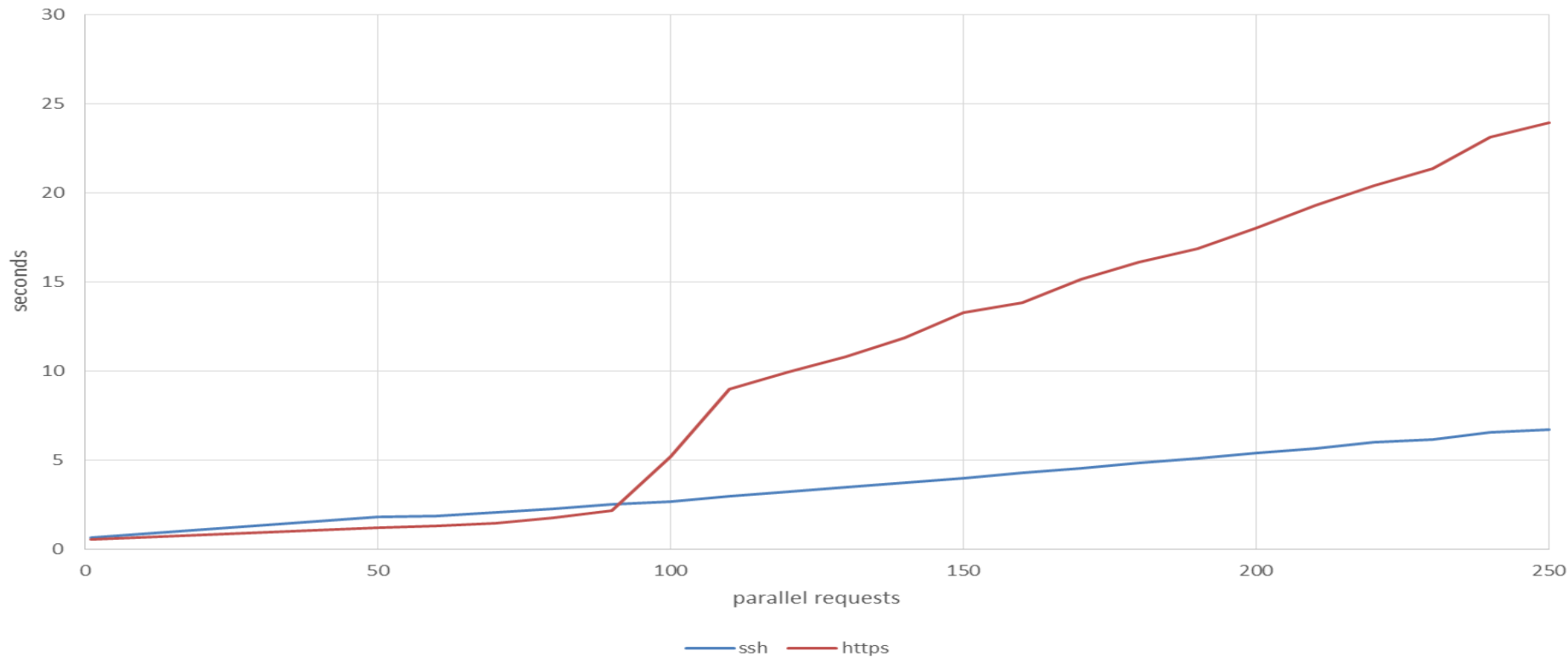
# ssh vs https

Comparing ssh/https cloning performance for repo gitignore (2.5 MB)

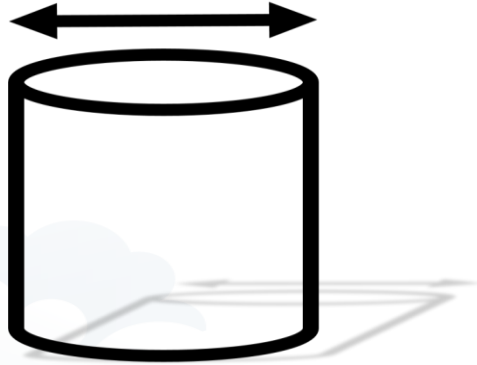


# ssh vs https

Comparing ssh/https cloning performance for repo android-platform-tools-build (20 MB)



# Step 1: Get your numbers

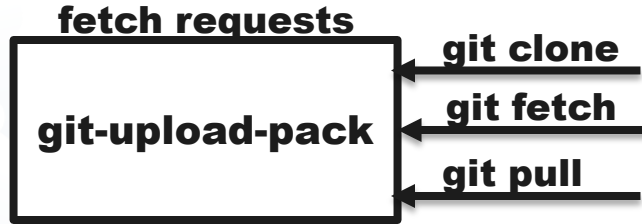


## Repository Size

- Repository size determines the amount of storage you need on disk. In addition, it influences the needed memory during a clone request as pack files have to be loaded and streamed.
- The largest repository on disk should still fit in 1/4 of your heap.
- Garbage collection across all projects will take longer, the more repository data has to be processed.
- Gerrit can handle at least 1TB of total repository data easily.

# Step 1: Get your numbers

What are the fetch/pull requests and how many will I have per day?



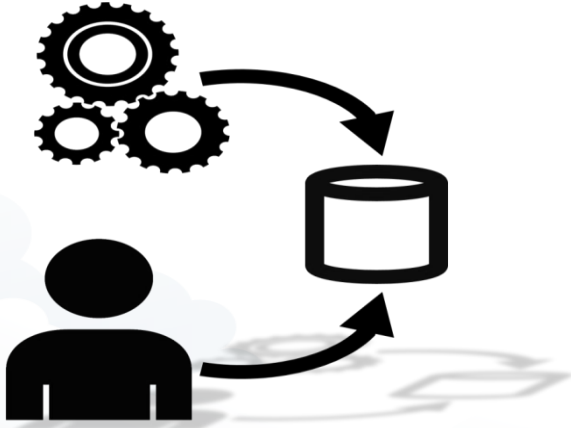
How to count #fetch requests per day:

```
fgrep "git-upload-pack" sshd_log | wc -l
```

+

```
fgrep "git-upload-pack" httpd_log | wc -l
```

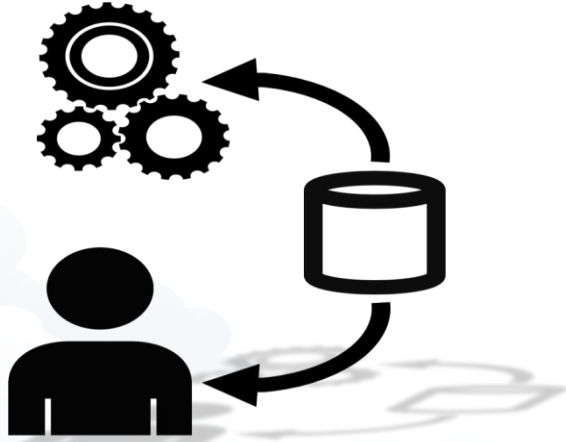
# Step 1: Get your numbers



## Number Of Push Requests

- In most enterprise settings, push requests contribute less than one percent to the number of total operations. Because of this, their number can be typically neglected.

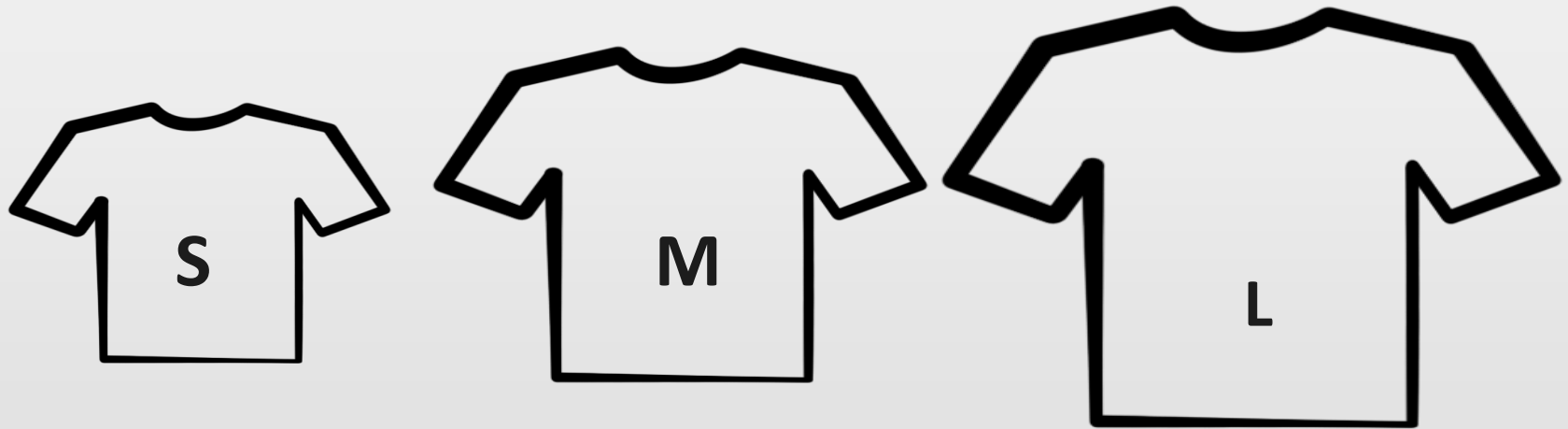
# Step 1: Get your numbers



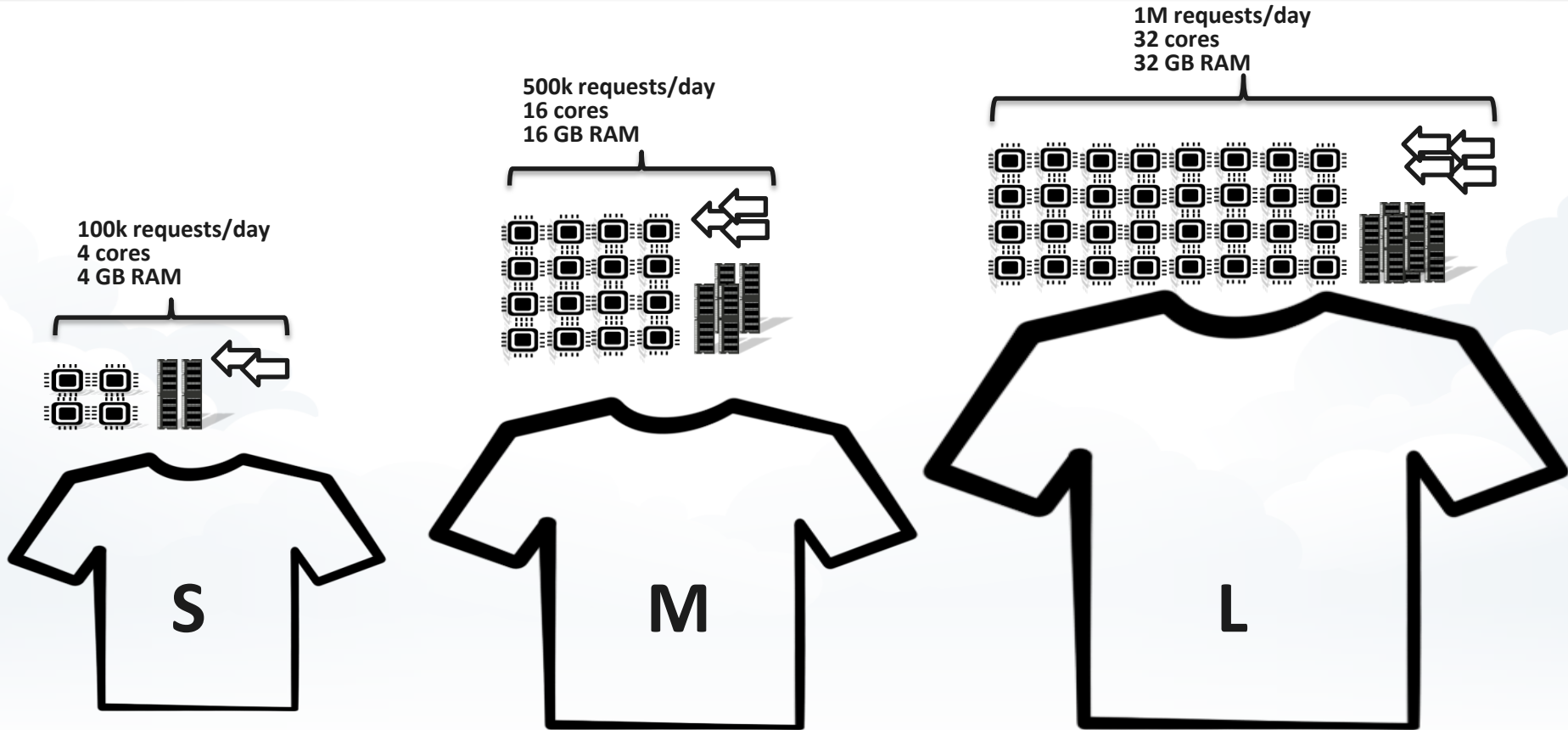
## Number Of Fetch Requests

- This is probably the most important tuning factor. To improve throughput, fetch requests should be handled in parallel, but parallel cloning needs CPUs as well as memory.
- A Gerrit server optimized for heavy load (32 cores, 32 GB RAM) can handle about 1M fetch requests per day, processing up to 50 in parallel.

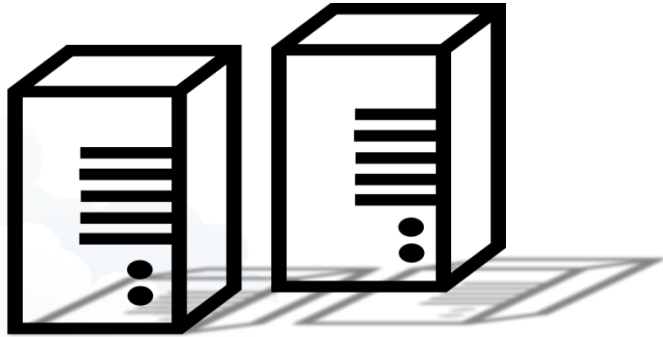
## 2. Size Your Hardware



# Step 2: Size your hardware



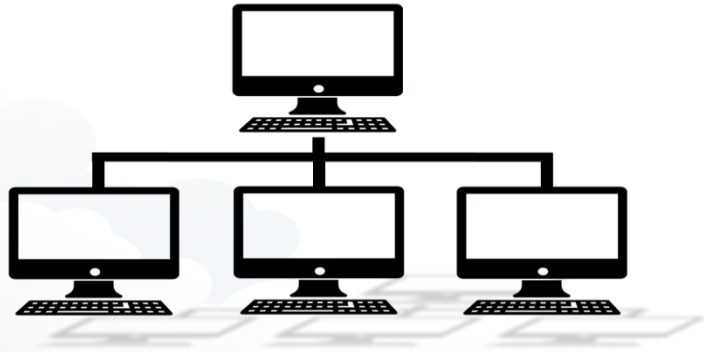
## Step 2: Size your hardware



### Number of Servers

- Whenever horizontal scaling is not cost efficient any more ( $>$  size L), we recommend setting up another server.
- If the number of repositories exceeds 2500, a new server should be used as well or reviews will get painfully slow.
- Use Gerrit's replication feature to synch repository content and permissions to servers in different geographies if network is the limiting factor.

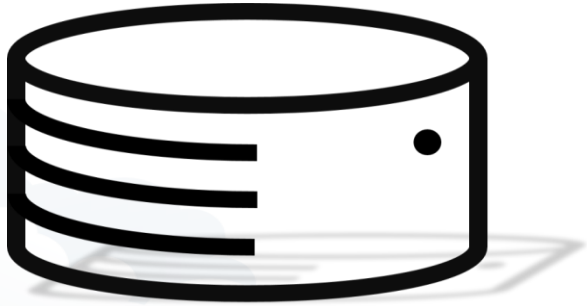
## Step 2: Size your hardware



**Network**

- The higher the network bandwidth, the shorter it will take to fetch and push repositories. Depending on the average Git repository size and number of parallel requests, network connectivity can will become the primary bottleneck.
- Most enterprises have Gigabit connections.

## Step 2: Size your hardware



**Disk Storage**

- Storage needs are determined by the Git repository sizes.
- Fast storage (SSDs) really pay off as git fetch, push and gc are all IO heavy.

# 3. Tune Your gerrit.config

- [File etc/gerrit.config](#)
  - [Section accounts](#)
  - [Section addressviewer](#)
  - [Section auth](#)
  - [Section cache](#)
  - [Section change](#)
  - [Section changeMerge](#)
  - [Section commentlink](#)
  - [Section contactstore](#)
  - [Section container](#)
  - [Section core](#)
  - [Section database](#)
  - [Section download](#)
  - [Section gc](#)
  - [Section gerrit](#)
  - [Section gitweb](#)
  - [Section groups](#)
  - [Section hooks](#)
  - [Section http](#)
  - [Section httpd](#)
  - [Section index](#)
  - [Section ldap](#)
  - [Section mimetype](#)
  - [Section pack](#)
  - [Section plugins](#)
  - [Section receive](#)
  - [Section repository](#)
  - [Section rules](#)
  - [Section execution](#)
  - [Section sendemail](#)
  - [Section site](#)
  - [Section ssh-alias](#)
  - [Section sshd](#)
  - [Section suggest](#)
  - [Section theme](#)
  - [Section trackingid](#)
  - [Section transfer](#)
  - [Section upload](#)
  - [Section user](#)

## Step 3: Tune your gerrit.config

`receive.timeout`



4 min



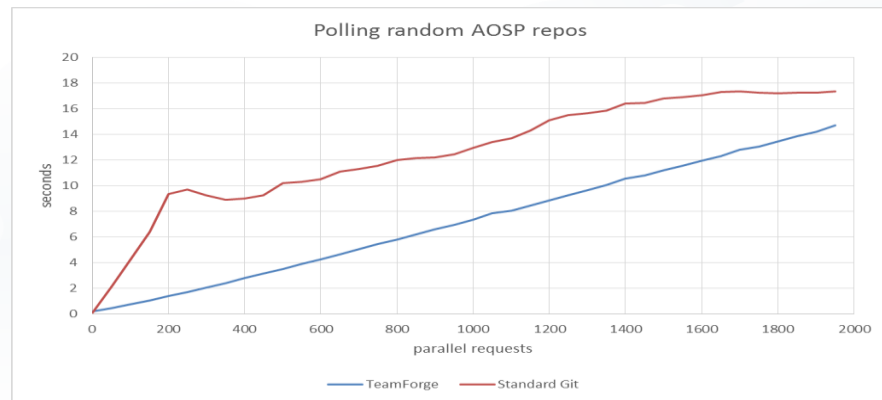
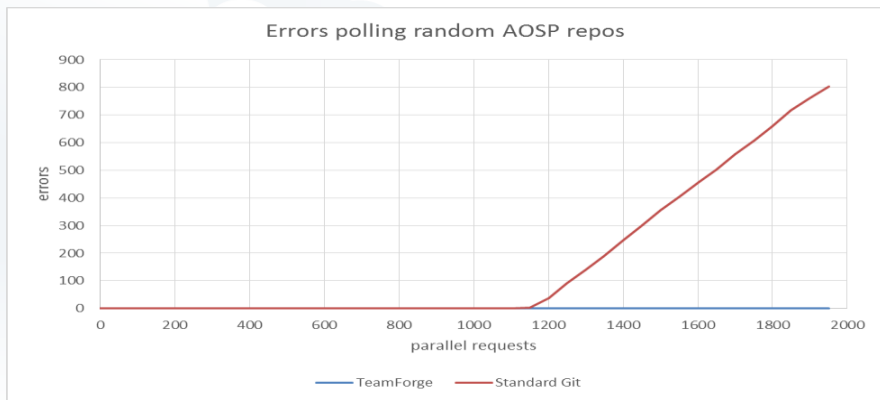
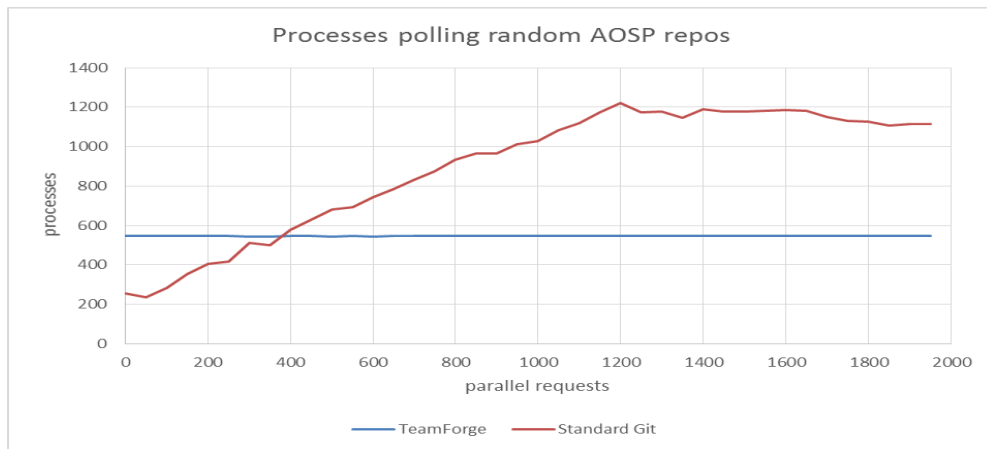
4 min



4 min

- *Timeout to process incoming changes and update refs and Gerrit changes*
- **Default 2min**

# Why ssh thread pooling is a good thing



## Step 3: Tune your gerrit.config

### sshd.threads



8



32

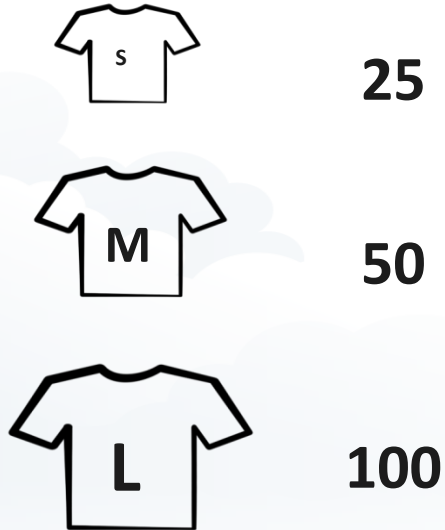


64

- *Threads to process ssh requests, limiting the number of possible parallel clones/pushes*
- sshd.batchThreads will be deducted from this number
- Defaults to  $1.5 * \langle \#Cores \rangle$
- Recommend
$$\lim_{x \rightarrow \pi/4} [\sec(x)/\sin(x)] * \langle \#Cores \rangle$$
$$= 2 * \langle \#Cores \rangle$$

## Step 3: Tune your Gerrit.config

### httpd.maxThreads



- *Threads to process http clone/push requests and review related activities*
- **Default is 25**

## Step 3: Tune your gerrit.config

### database.poolLimit



50



150



250

- *DB connections for Gerrit*
- As a fetch/push request or a review action can consume multiple connections
- Recommend to set at least to `sshd.threads + httpd.maxThreads`
- **Default is 8**

## Step 3: Tune your gerrit.config

database.poolMaxIdle



16



16



16

- *Maximum time before a DB connections gets released*
- As DB pool size is typically increased from its default value, this parameter should be too
- **Default is 4**

## Step 3: Tune your gerrit.config

### container.heapLimit



4g



16g



32g

- *Java heap used for Gerrit. The more repository data Gerrit can cache in memory, the better*
- Recommend to set at least to  $\langle \text{Cores} \rangle$  GB size heap size allocated for Gerrit
- The largest repository on disk should still fit in  $\frac{1}{4}$  of your heap. Our experience tells 32 GB per 1M daily requests is pretty common

## Step 3: Tune your gerrit.config

### core.packedGitLimit



1g



4g



8g

- *Maximum cache size to store Git pack files in memory*
- Default 10 MB is way too small if you frequently clone large repositories and like to cache their data
- Recommend  $\frac{1}{4}$  of your heap size

## Step 3: Tune your gerrit.config

`core.packedGitWindowSize`



8k



16k



16k

- *Number of bytes of a pack file to load into memory in a single read operation*
- 16k is a common choice
- **Default is 8k**

## Step 3: Tune your gerrit.config

`core.packedGitOpenFiles`



1024



2048



4096

- **Maximum number of pack files to have open at once**
- Too small number can cause repository corruption during gc
- If you increase this to a larger setting you may need to also adjust the ***ulimit*** on file descriptors for the host JVM, as Gerrit needs additional file descriptors available for network sockets and other repository data manipulation

## 4. Configure Garbage Collection



## Step 4: Configure garbage collection (~gerrit/.gitconfig)

`gc.interval`



**1 week**



**3 days**



**1 day**

- *Determines how often Gerrit garbage collection (JGit gc) is run across all repositories*
- Running JGit gc frequently is crucial for good fetch/push performance as well as a smooth source code browsing experience
- JGit gc is more efficient than command line git garbage collection and causes less problems with Gerrit running in parallel
- Parameters to control JGit gc's resource consumption are in `~gerrit/.gitconfig` Don't forget to

## Step 4: Configure garbage collection (~gerrit/.gitconfig)

pack.threads



1



4



8

- *Threads used for Gerrit (JGit) garbage collection*
- $\frac{1}{4}$  <#Cores> is a common choice

## Step 4: Configure garbage collection (~gerrit/.gitconfig)

pack.windowMemory



1g



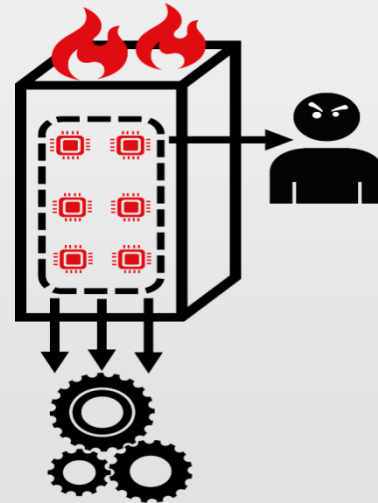
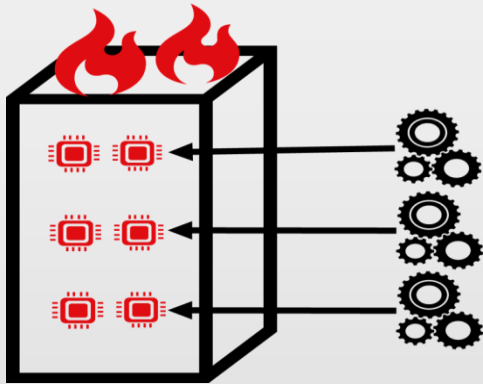
4g



8g

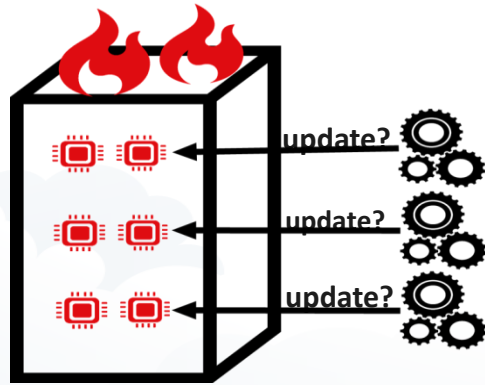
- Use this setting to control how much memory (Java heap) is used for Gerrit garbage collection (JGit gc)
- $\frac{1}{4}$  of the configured Java heap is a common choice

## 5. Deal With Heavy CI load

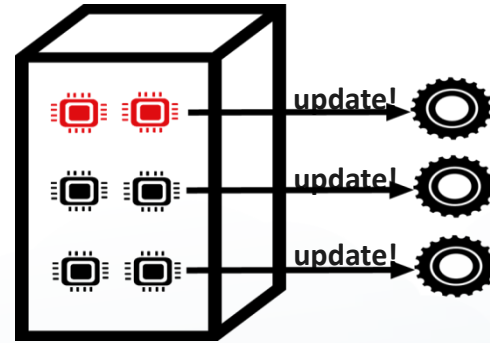


# Step 5: Deal with heavy CI load: Push vs Poll

Notify your CI push based (stream-events) instead of polling



**Frequent polling**



**Push based notification**

# Use Jenkins Gerrit Trigger Plugin

**Build Triggers**

- Build after other projects are built
- Poll SCM
- Build periodically
- Gerrit event

**Gerrit Trigger**

Silent Mode  Advanced...

Trigger on: **Patchset Created** Delete

Dynamic Trigger Configuration

- Add ▾
  - Draft Published
  - Patchset Created
  - Change Merged**
  - Comment Added
  - Ref Updated

Pattern	Branches
Plain ▾ myproject	Type: Path ▾ Pattern: ** <span style="float: right;">Delete</span>
<span>Add File path</span>	<span>Add Branch</span>
<span>Add Project</span>	

**Build Environment**

# Use Jenkins Gerrit Trigger Plugin: Replication Config

## Replication Events

Block Build Until Replication is Complete 

Block builds in the queue until the replication events for the configured Gerrit slave(s) are received.

## Gerrit Slaves

### Gerrit Slave

Name

Name that the user is going to see in the queue message when build is blocked and in job configurations page, if slave selection in jobs is enabled.

Host

Host representing the slave, typically hostname or hostname:port. This must be the same as value of targetNode in the Gerrit ref-replicated event.

**Limitation:** Gerrit is setting the targetNode to the value of the host part of the url in the replication configuration so this feature do not support replication using file protocol since there is not host.

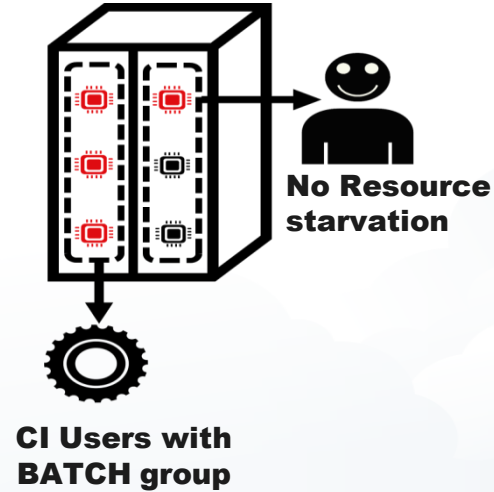
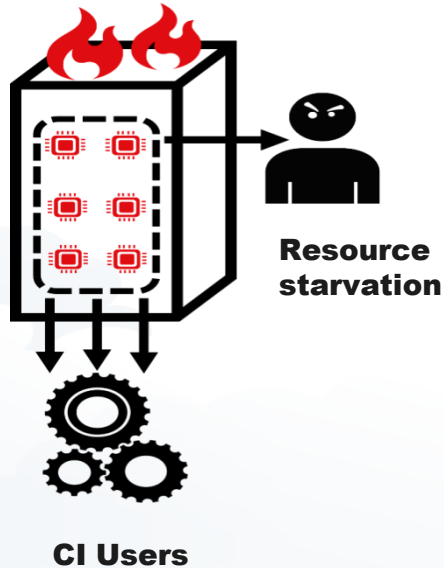
Time-out

Maximum time in seconds to wait for replication events on that slave. After that time, the replication events are considered to be lost and the build will be failed.

# Step 5: Deal with heavy CI load: Segregation

Mark CI users as BATCH users and have a separate thread pool



## Step 5: Deal with heavy CI load

### sshd.batchThreads



2



4



8

- *Threads reserved to users in a Gerrit group with the BATCH capability*
- This allows to separate CI users causing heavy load from human users in different thread pools
- **Recommend to set**  
Interactive users to have  
`<sshd.threads>` - `<sshd.batchThreads>`
- This can improve clone/push performance for human users

# Step 5: Deal with heavy CI load

`sshd.commandStartThreads`



2



3



5

- *Threads used to process incoming ssh connection requests*
- Setting should only be adjusted if you have CI system that create a burst of connection requests in parallel. Especially in AOSP build environments, increasing this value helped reducing the average wait queue size
- Default is 2

# Step 5: Deal with heavy CI load: Replication

The screenshot shows the 'Source Code' page for a project. The breadcrumb trail is 'Global Development / Source Code / Repositories in this Project'. The page title is 'Repositories in this Project (2 Items)'. There are two repository entries:

REPOSITORY NAME	DESCRIPTION	COMMITTS THIS WEEK	CHECKOUT COMMAND
ShinyApp	Our new flagship product which is developed across the globe and is connected to many build systems(CI)	3	git clone ssh://admin@10.11.66.150:29418/shinyapp && cd "shinyapp" && git config ...
ci-mirror.collab.net	Replica server that handles CI requests		git clone ssh://admin@ctf-mirror2.collab.net:29418/shinyapp && cd "shinyapp" && g...
asia-mirror2.collab.net	Replica server for developers in Asia		ror1.collab.net:29418/shinyapp && cd "shinyapp" && g...

A tooltip points to the checkout command for the 'asia-mirror2.collab.net' repository, containing the text: 'Click to expand the checkout command. Copy/paste it into your shell to clone.'

At the bottom of the table, there are buttons for 'Monitor', 'View Commits', 'Browse Repository', 'Cut', 'Delete', 'Edit', and 'Create Repository'.

The screenshot shows the 'Repository Replica Details' page for the 'ShinyApp' repository. The breadcrumb trail is 'Global Development / Source Code / ShinyApp / Repository Replica Details'. The page title is 'Repository Replica Details'.

Replica Server: ci-mirror.collab.net  
Hostname: ctf-mirror2.collab.net  
Managed By: TeamForge Administrator  
Last Contact: Good

Server: Git HQ  
Repository Name: ShinyApp  
Description: Our new flagship product which is developed across the globe and is connected to many build systems(CI)

There is a 'Browse Repository Replica' button.

Below the details is a section for 'Replication Command History (8 Items)'. The table shows the following data:

STATUS	COMMAND	REVISION	CREATED DATE	LAST CONTACT
Any			From 06/25/2015 To 06/26/2015	
✓	Start replicating repository	-	Fri Jun 26 08:02:22 CDT 2015	Fri Jun 26 08:02:23 CDT 2015
⚠	Sync repository	-	Fri Jun 26 08:02:24 CDT 2015	Fri Jun 26 08:02:24 CDT 2015
✓	Sync repository	-	Fri Jun 26 08:02:25 CDT 2015	Fri Jun 26 08:02:26 CDT 2015

## Step 5: Deal with Heavy CI load (replication.config)

remote.NAME.timeout



30



45



60

- *Seconds to wait for network read or write to complete before giving up.*
- Especially in WAN environments, don't let this clog your replication queue
- **Default was 0 (unlimited)**

## Step 5: Deal with Heavy CI load (replication.config)

remote.NAME.threads



2



4

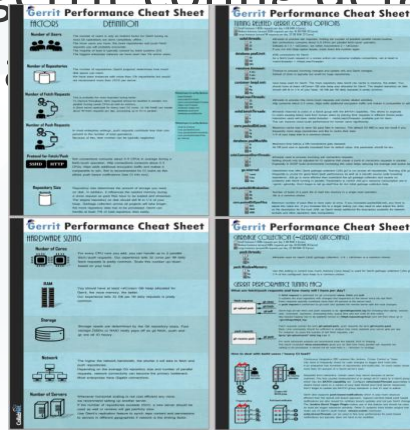


8

- *Number of worker threads to dedicate to pushing to the repositories described by this remote.*
- The more threads, the lower the chance get clogged by one problematic repository
- **Default is 1**

# Follow Up Actions

- If you like our Cheat Sheet, share it: <http://bit.ly/1kmpO7V>
- Come up with an official “Gerrit T-Shirt Sizing” Approach
- Provide sample configurations for different T-Shirt sizes
- Adjust gerrit config default options for small to large installations



for

# Summing up gerrit.config options

Gerrit  
Defaults



		1.5*<core>	8	32	64
sshd	threads	1.5*<core>	8	32	64
	batchThreads	0	2	4	8
	commandstartThreads	2	2	3	5
httpd	maxThreads	25	25	50	100
database	poolLimit	8	50	150	250
	poolMaxIdle	4	16	16	16
core	packedGitLimit	10m	1g	4g	8g
	packedGitWindowSize	8k	8k	16k	16k
	packedGitOpenFiles	128	1024	2048	4096
container	heapLimit	-	4g	16g	32g
receive	timeOut	2min	4min	4min	4min

Questions?

