

Rhoynar Software Consulting

Document for Devops Setup

Table of Contents

Description.....	3
Technology Requirements.....	3
Configuration Architecture.....	3
Description:.....	3
Workflow Architecture.....	4
Description:.....	5
Physical Machines - Prerequisite.....	6
Installation.....	6
Provisioning using Vagrant	6
Prerequisite:.....	6
Provision Machines.....	7
For Master VM:.....	7
For Agent VM:.....	7
Scripts.....	8
How to run:.....	8
Provision Physical machine	8
Network Configuration.....	8
For Master VM:.....	9
For Agent VM:.....	9
Infrastructure Overview.....	10
Puppet Configuration	10
Custom Configuration for different modules	10
Agent1 - JIRA.....	10
Agent2 - Jenkins.....	11
Agent3 - Gerrit.....	11
Agent4 - Zuul.....	12
Tool Dashboard Configuration.....	12
JIRA.....	12
JIRA Functionality.....	15
Jenkins.....	15
Jenkins Job Functionality.....	17
Gerrit.....	17
Zuul.....	20
Add/Modify Jenkins Job.....	20
Create Project.....	21
Git-Review.....	22
Installation.....	22
Setup.....	22
.gitreview configuration file.....	23
What happens when you submit a change.....	23
Git Workflow.....	24
Post Installation Steps.....	25

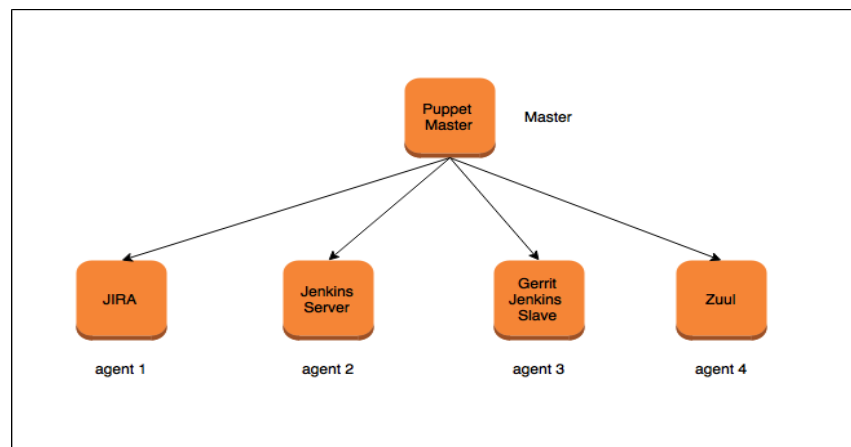
Description

Rhoynar Software Consulting is US based software consulting company which builds most cost effective solutions for software needs. Rhoynar Software Consulting is looking for assistance with designing, implementing and automating scalable, highly available application infrastructure. Rhoynar also wants to build automated system to create consistent application configurations across development and production environments.

Technology Requirements

- Puppet
- Gerrit
- Zuul
- Jenkins
- JIRA
- Git for Revision Control

Configuration Architecture

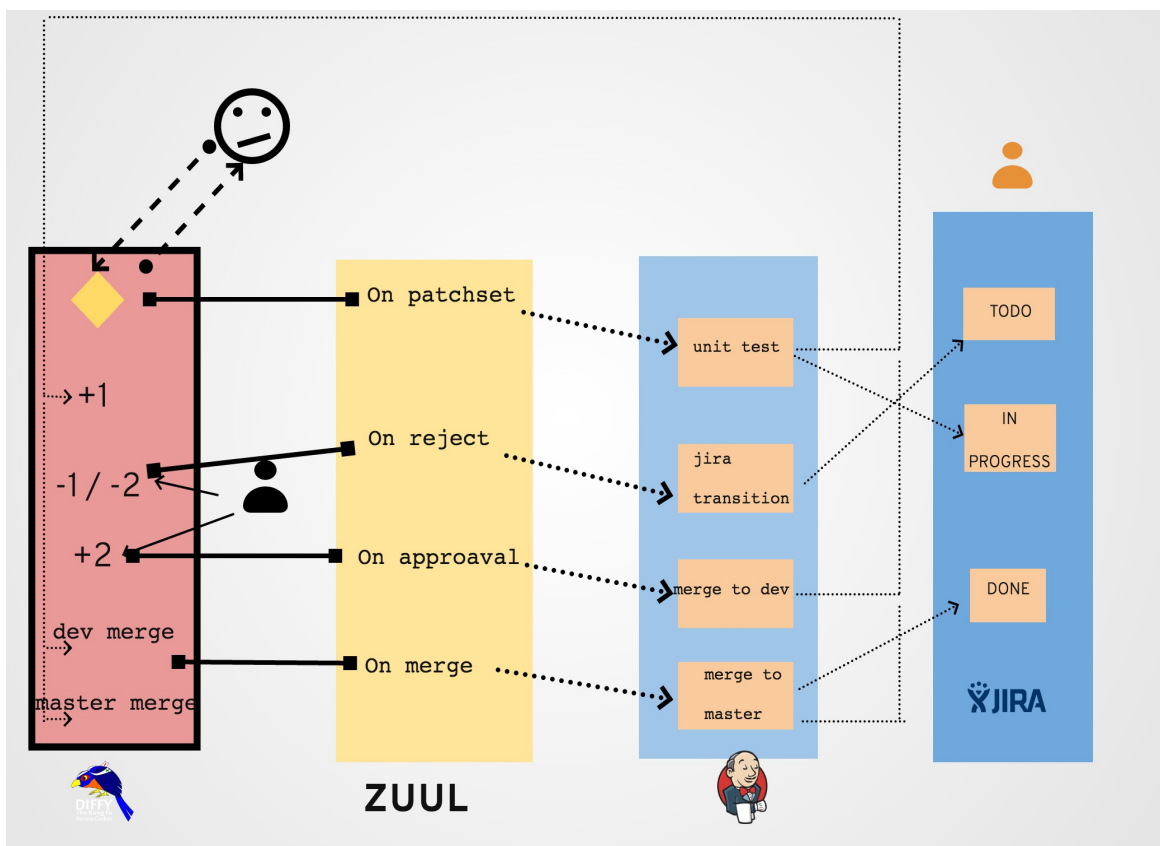


Description:

- **Puppet:** Puppet is a configuration management tool that helps system administrators automate the provisioning, configuration, and management of a server infrastructure.

- **Gerrit:** Gerrit is a web-based code review tool built on top of the git version control system. Gerrit is intended to provide a lightweight framework for reviewing every commit before it is accepted into the code base. Changes are uploaded to Gerrit but don't actually become a part of the project until they've been reviewed and accepted.
- **Jenkins:** Jenkins is a powerful application that allows continuous integration and continuous delivery of projects, regardless of the platform you are working on. The basic functionality of Jenkins is to execute a predefined list of steps. The trigger for this execution can be time or event based. Jenkins also monitors the execution of the steps and allows to stop the process if one of the steps fails. Jenkins can also send out notification about the build success or failure.
- **Zuul:** Zuul is a program that is used to gate the source code repository of a project so that changes are only merged if they pass tests.

Workflow Architecture



Description:

- **On Patchset:** Once jira ticket has been raised developers will clone the project, fix the bug and push it for code review with jira ticket id with commit message. Then zuul will listen on event patchset created and trigger jenkins job for unit test
 - If Successful: Jira issue will be updated to “In Progress” with comment of gerrit link reference and code-review approval will be +1.
 - If Fails: code-review approval will be -1.
- **On Reject:** If code-reviewer change approval to -1/-2 then zuul will listen on event comment added with approval and trigger jenkins job for jira-state-transition
 - Action: Jira issue will be updated to “To Do” with comment of gerrit link reference.
- **On Approval:** If code-reviewer change approval to +2 then zuul will listen on event comment added with approval and trigger jenkins job for merge-to-dev
 - If Successful: Zuul will merge changes to dev
 - If Fails:code-review approval will be -2 and Jira issue will be updated to “To Do” with comment of gerrit link reference and code-review
- **On Merge:**Once changes merged to dev, zuul will listen on event change merged and trigger jenkins job for merge-to-master
 - If Successful: Zuul will merge dev to master and Jira issue will be updated to “Done” with comment of gerrit link reference.
 - If Fails: Manually need to resolve conflict.

Physical Machines - Prerequisite

Five Machines, 1 puppet-master and 4 puppet-agent is required.

Node	Name	Requirements	Purpose
1	Master	OS: Debian Jessie RAM: 2GB+	puppet-master
2	Agent1	OS: Debian Jessie RAM: 8GB+	Jira
3	Agent2	OS: Debian Jessie RAM: 2GB+	Jenkins
4	Agent3	OS: Debian Jessie RAM: 2GB+	Gerrit
5	Agent4	OS: Debian Jessie RAM: 2GB+	Zuul

Installation

Provisioning using Vagrant

Tested with Linux (Debian Jessie)

Note: We can use vagrant setup procedure for Testing Purpose, it is not required if installation is for Physical machines.

Prerequisite:

[Vagrant](#) >= 1.8.5

Download URL's:

- For [Mac](#)
- For [windows](#)

[Virtual Box](#) >= 5.1.2

Download URL's:

- For [Mac](#)
- For [windows](#)

Provision Machines

Here we are provisioning five virtual machines into virtual box using vagrant, follow the below instructions to provisioning.

- Change directory where *Vagrantfile* is placed.
- Ensure that both [puppet-master.sh](#) and [puppet-agent.sh](#) is present in same location where vagrant file is placed.

Make the below necessary changes in Vagrantfile (if required e.g if you want to change ip addresses for nodes)

For Master VM:

```
config.vm.define "master" do |master|
  master.vm.network "private_network", ip: "192.168.35.11"
  master.vm.provision "shell", path: "puppet-master.sh", args: "-m 'master' -H
'192.168.35.11' -a 'agent' -h '192.168.35.12 192.168.35.13 192.168.35.14 192.168.35.15'"
```

Description:

- -m : master host name.
- -H : master host IP.
- -a : agent nodes (Required for autosign certificate and this should be only “**agent**” if you want to use different name for agent we need to add explicitly direct into server)
- -h : agent nodes IP.

For Agent VM:

```
config.vm.define "agent1" do |agent1|
  agent1.vm.network "private_network", ip: "192.168.35.12"
  agent1.vm.provision "shell", path: "puppet-agent.sh", args: "-m 'master' -H '192.168.35.11'
  -a 'agent' -h '192.168.35.12 192.168.35.13 192.168.35.14 192.168.35.15' -i 'agent1'"
end
```

Repeat the same steps in other agents by replacing the agent name.

Scripts

- [puppet-master.sh](#) : for puppet master installation and configuration while provisioning VM.
- [puppet-agent.sh](#) : for puppet agent installation and configuration while provisioning.

How to run:

Use below command for provision VMs in single shot with vagrant (first time it will take time because it will need to download the linux box).

```
vagrant up
```

Once it is finished successfully, check the status using below command

```
vagrant global-status
```

Now we can login the VM, using below command

```
vagrant ssh master
```

we can provide VM id otherwise VM name while login to instance

Provision Physical machine

Network Configuration

Setting up static ip for stable puppet master-agent configuration.

- The majority of network setup can be done via the *interfaces* configuration file at `/etc/network/interfaces`.

- Configuring the interface manually by adding something like this to `/etc/network/interfaces` will set the default ip (network, broadcast and gateway are optional):
- ```

auto eth0
iface eth0 inet static
address 192.168.35.11
netmask 255.255.255.0
gateway 192.168.35.1

```

Note: Your IP configuration may vary according to your Internet settings

- Configure sequentially IP's for Master(192.168.35.11) and Agent(192.168.35.12, 192.168.35.13, 192.168.35.14 & 192.168.35.15) machines respectively.

### For Master VM:

Login to Master Machine and configure puppet-master as follows,

- create executable file named *puppet-master.sh*
- Also you need keep same -a option to “**agent**” only.
- Now run the following command to configure puppet-master.

```
./puppet-master.sh -m 'master' -H '192.168.35.11' -a 'agent' -h '192.168.35.12
192.168.35.13 192.168.35.14 192.168.35.15'
```

- Puppet-Master is configured successfully.

### For Agent VM:

Login to each of the Agent Machine and configure puppet-agent as follows,

- create executable file named *puppet-agent.sh* and add the content from [here](#)
- Now run the following command to configure puppet-agent.

```
./puppet-agent.sh -m 'master' -H '192.168.35.11' -a 'agent' -h '192.168.35.12
192.168.35.13 192.168.35.14 192.168.35.15' -i 'agent1'
```

Note: Replace -i option at the end of command with respect to the agent's

- Puppet-Agent is configured successfully in all agent's.

## Infrastructure Overview

Let us have an overview of our instances/Machines,

| Instance | Name   | Purpose                         | Private IP's  |
|----------|--------|---------------------------------|---------------|
| 1        | Master | Puppet Master will be Installed | 192.168.35.11 |
| 2        | Agent1 | Jira                            | 192.168.35.12 |
| 3        | Agent2 | Jenkins                         | 192.168.35.13 |
| 4        | Agent3 | Gerrit                          | 192.168.35.14 |
| 5        | Agent4 | Zuul                            | 192.168.35.15 |

## Puppet Configuration

After running script on master and all agents puppet will be installed and get configured automatically.

In puppet we need to edit main site.pp into "/etc/puppet/manifests/site.pp", which will consider module distribution to agent, look into below code for configure.

### Custom Configuration for different modules

Let us dig deep in each resource of site.pp, these instructions are only applicable for additional custom configuration for the various CI modules. Use these instructions to suit your organizational needs"

#### Agent1 - JIRA

In this resource part the *Jira class* for installing JIRA and *postgresql class* for db server is configured along with username and password.

```

node 'agent1.example.com'{
 class { 'postgresql::server': } ->

 postgresql::server::db { 'jira':
 user => 'jiraadm',
 password => postgresql_password('jiraadm', 'mypassword'),
 } ->
 file { ['/usr/java/']:
 ensure => 'directory',
 } ->
 java::oracle { 'jdk8' :
 ensure => 'present',
 version => '8',
 java_se => 'jdk',
 } ->
 class { 'jira':
 javahome => '/usr/java/jdk1.8.0_51',
 }
}

```

**Resources for "agent1"**

specifying username and password for postgresql server

## Agent2 - Jenkins

In agent2 resource part Jenkins modules are included along with credentials of JIRA for integration as follows,

```

node 'agent2.example.com'{
 include_jenkins
 include_jenkins::master
 jenkins::plugin { 'gerrit-trigger': }
 jenkins::plugin { 'gearman-plugin': }
 jenkins::plugin { 'workflow-aggregator': }

 class { 'acli':
 version => '5.0.0',
 user => '',
 password => '',
 jira_server => 'http://agent1.example.com:8080',
 }
}

```

**Resources for "agent2"**

Specify the same username and password that specified in Jira Dashboard creation

## Agent3 - Gerrit

Gerrit class is included in agent3 resource part with jenkins slave configuration for integration as shown below,

```

node 'agent3.example.com'{
 class { 'jenkins::slave':
 masterurl => 'http://agent2.example.com:8080',
 ui_user => '',
 ui_pass => '',
 } ->

 class { 'gerrit':
 canonicalweburl => 'http://agent3.example.com:8090/',
 httpd_hostname => 'agent3.example.com',
 httpd_port => '8090',
 }
}

```

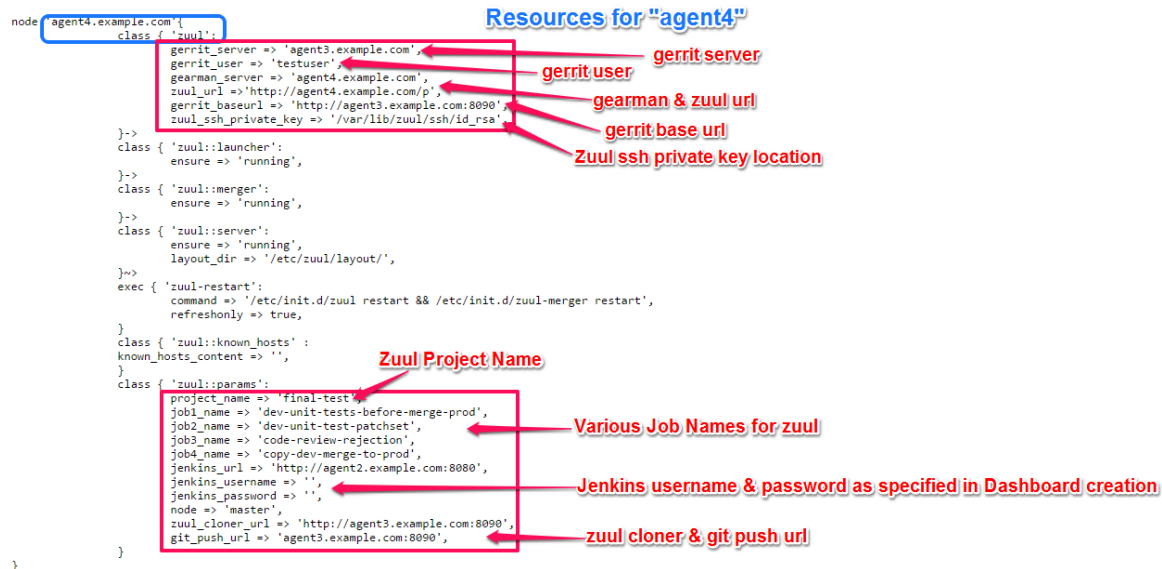
**Resources for "agent3"**

Specify same Jenkins username & password that was specified during Jenkins administration dashboard creation

specify Gerrit url and hostname of gerrit\_httpd

## Agent4 - Zuul

Resource part for agent4 includes Zuul with the following specifications for proper integration,



Now puppet setup and configuring modules with required parameters has been completed then we need to start one by one puppet agent and run it in server level sequentially.

E.g: "puppet agent start && puppet agent -t"

Once jira setup has been completed and installed successfully we need to configure and login using dashboard(See section Module for reference). Then same credentials and keys we need to put into site.pp and required templates for jenkins and further.

Then finally we need to run puppet agent into zuul. Puppet automation installation is complete and puppet master and agents can be stopped

## Tool Dashboard Configuration

Let us explore each modules for detailed understanding,

### JIRA

As per our puppet module JIRA will be installed on *Agent1*

- Once puppet agent installs JIRA visit the url on port 8080 [192.168.35.12:8080](http://192.168.35.12:8080)

Setup JIRA by using the simple GUI as follows

- Choose Application name and Mode

Set Up Application Properties

**Existing data?** You can import your data from another installed or hosted JIRA server instead of completing this setup process.

Application Title

The name of this installation.

Mode ☐ Public  
Anyone can sign up to create issues.

☒ Private  
Only administrators can create new users.

Base URL

The base URL for this installation of JIRA.  
All links created will be prefixed by this URL.

Next

- Choose how do you want to use JIRA

How do you want to use JIRA?

I want to use JIRA for project tracking We'll install JIRA for you.

I want to use JIRA for software development We'll install JIRA and JIRA Agile to enable agile software development workflows and task boards.

I want to use JIRA for IT requests We'll install JIRA and JIRA Service Desk to help make IT request tracking easier for end users and your IT help desk.

Next

- Specify your JIRA License Key

Specify Your License Key

You need a license key to set up JIRA. Enter your license key or generate a new license key below. You need an account at [my.atlassian.com](https://my.atlassian.com) to generate a license key.

☐ I don't have an account ☐ I have an account but no key ☒ I have a JIRA key

Please enter your license key

Server ID **BL9P-022G-9BDZ-LQKP**

Your License Key

Input your key

Next

- Setup Administrator account

Set Up Administrator Account

Enter details for the administrator account. You can add more administrators after setup.

Full name  
rhoynar pvt ltd

Email Address  
admin@rhoynar.com

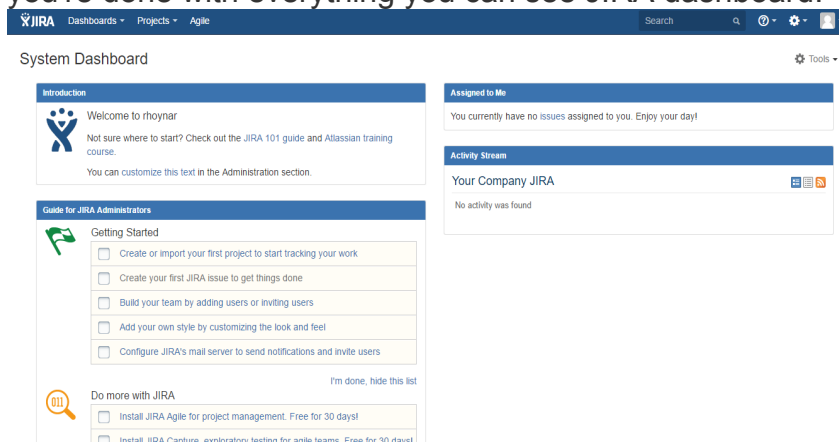
Username  
admin

Password  
\*\*\*\*\*

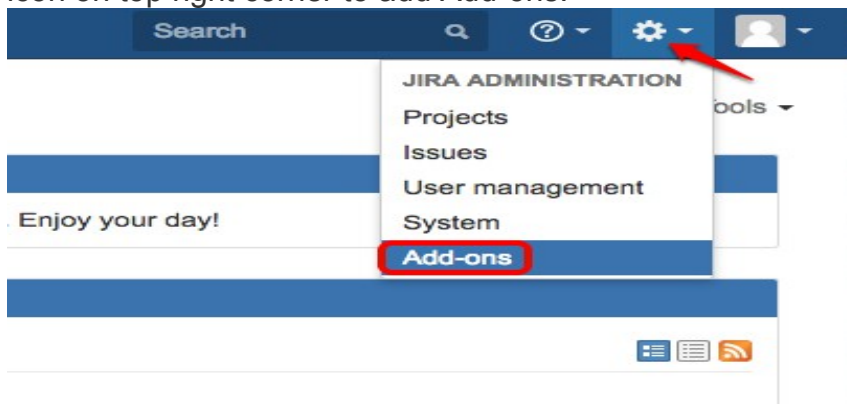
Confirm Password  
\*\*\*\*\*

Next

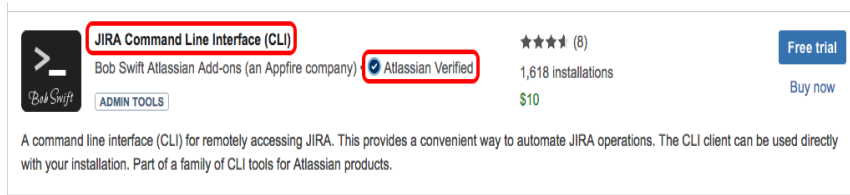
- Configure Email id, Avatar and finish installation with basic intro to JIRA. Once you're done with everything you can see JIRA dashboard.



- Now let us add JIRA CLI Add-ons for interacting with JIRA using CLI, click gear icon on top right corner to add Add-ons.



- From the list of available Add-ons choose JIRA Command Line Interface (CLI) which is Atlassian Verified, to install it.



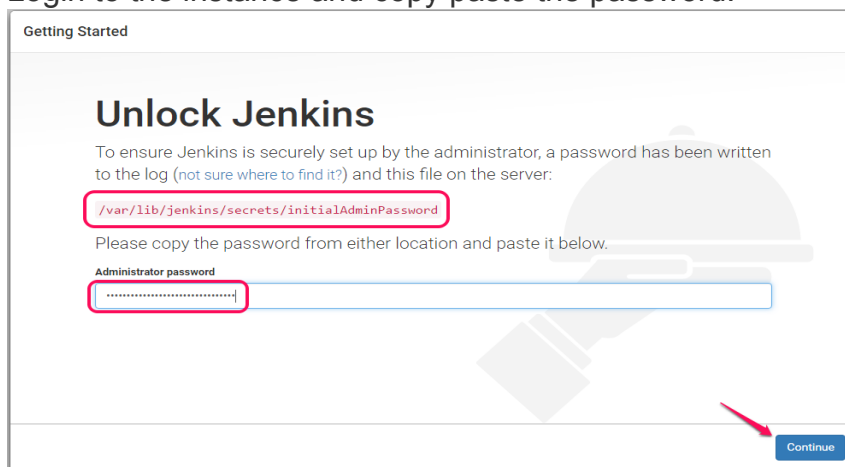
## JIRA Functionality

- we have automated JIRA issues modification as per code-review actions.
- Currently we have added standard workflow transition for jira "To Do – In Progress - Done"
- In case if we need to modify the jobs for transition state. We need to modify puppet jjb template in puppet-master. For modifying jjb follow [instructions](#) given in the Jenkins Job.

## Jenkins

Puppet Module will install Jenkins on Agent2, visit the instance IP with port 8080 to configure Jenkins [192.168.35.12:8080](#)

- Initial password will be automatically generated in the following location `/var/lib/jenkins/secrets/initialAdminPassword`
- Login to the instance and copy paste the password.



- Proceed with installing the basic required plugins for Jenkins.
- Setup new administrator *username* and *password* for administrator account.

### Create First Admin User

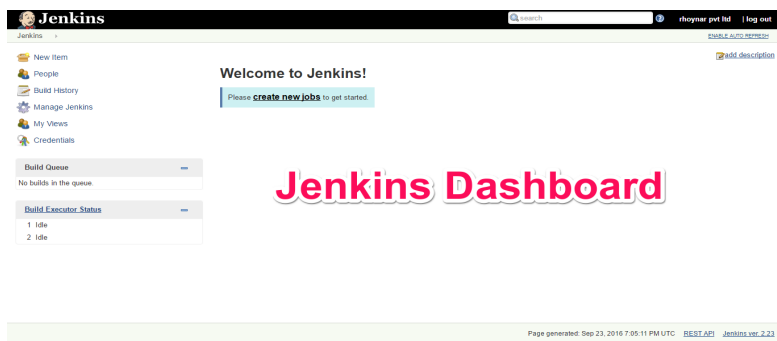
|                   |                   |
|-------------------|-------------------|
| Username:         | rhoynar           |
| Password:         | *****             |
| Confirm password: | *****             |
| Full name:        | rhoynar pvt ltd   |
| E-mail address:   | admin@rhoynar.com |

16 2.23

Continue as admin

Save and Finish

- Now Jenkins is ready.



The screenshot shows the Jenkins dashboard. At the top, there's a header with the Jenkins logo, a search bar, and user information (rhoynar pvt ltd, log out). Below the header, there's a sidebar with navigation links: New Item, People, Build History, Manage Jenkins, My Views, and Credentials. The main content area displays 'Welcome to Jenkins!' with a button to 'create new jobs'. Below this, there's a 'Build Queue' section showing 'No builds in the queue' and a 'Build Executor Status' section showing '1 idle' and '2 idle' executors. A large red text overlay 'Jenkins Dashboard' is centered on the page. At the bottom, there's a footer with the page generation time and version information.

- Let us make connection with Gearman for interaction. Configure Gearman server IP and Listening port to establish connection with Gearman.

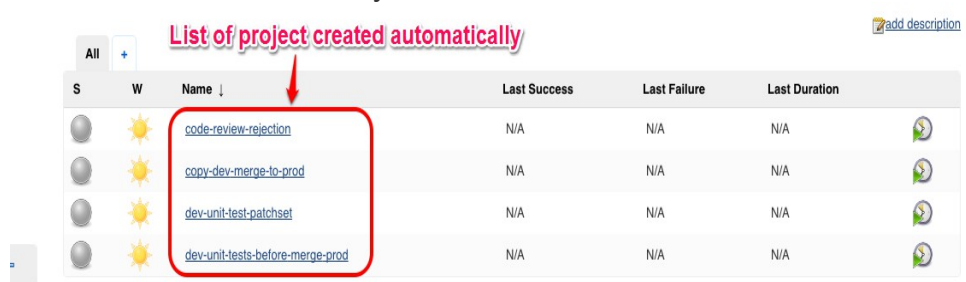
| Gearman Plugin Config |                                                            |
|-----------------------|------------------------------------------------------------|
| Gearman Server Host   | 172.31.48.159 <b>Local IP of Gearman server</b>            |
| Gearman Server Port   | 4730 <b>Listening Port</b>                                 |
| Enable Gearman        | <input checked="" type="checkbox"/> <b>Test Connection</b> |

Select to enable Gearman plugin, Unselect to disable



- After successful connection to Gearman with Jenkins we could see the list of Jobs created automatically in Jenkins Dashboard.

**List of project created automatically**



| S | W | Name ↓                                           | Last Success | Last Failure | Last Duration |
|---|---|--------------------------------------------------|--------------|--------------|---------------|
|   |   | <a href="#">code-review-rejection</a>            | N/A          | N/A          | N/A           |
|   |   | <a href="#">copy-dev-merge-to-prod</a>           | N/A          | N/A          | N/A           |
|   |   | <a href="#">dev-unit-test-patchset</a>           | N/A          | N/A          | N/A           |
|   |   | <a href="#">dev-unit-tests-before-merge-prod</a> | N/A          | N/A          | N/A           |

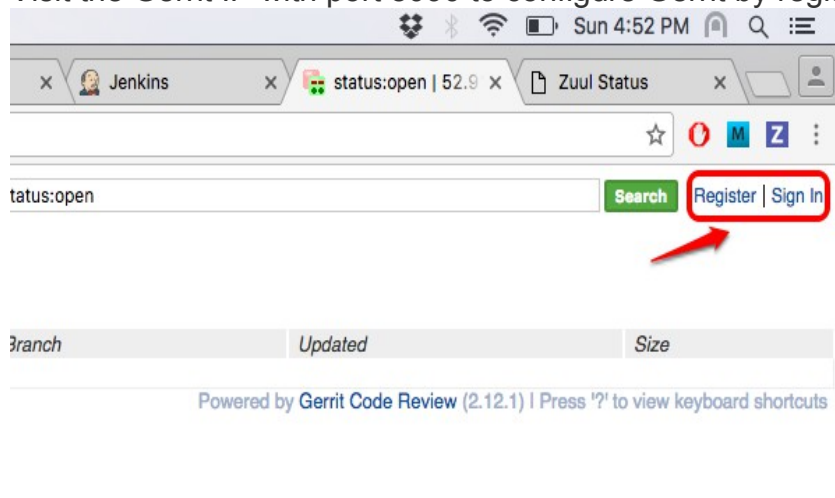
## Jenkins Job Functionality

We have automated Jenkins jobs using puppet through zuul server. Currently we have created 4 jobs, in case if we need to modify or add new jobs to it follow the below instructions

## Gerrit

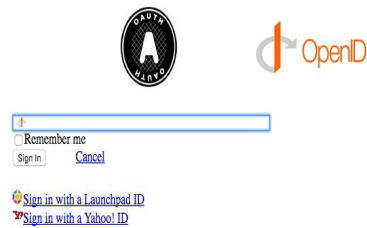
Gerrit is a code review tool which will be installed in Agent3 by puppet module.

- Visit the Gerrit IP with port 8090 to configure Gerrit by registering or signing in.

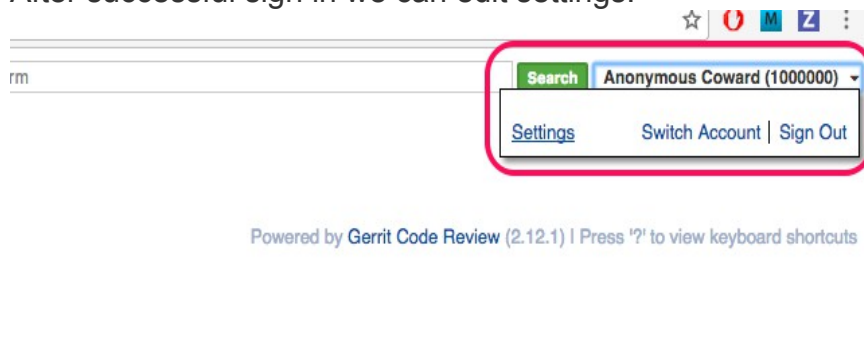


- Sign in to Gerrit with the existing account.

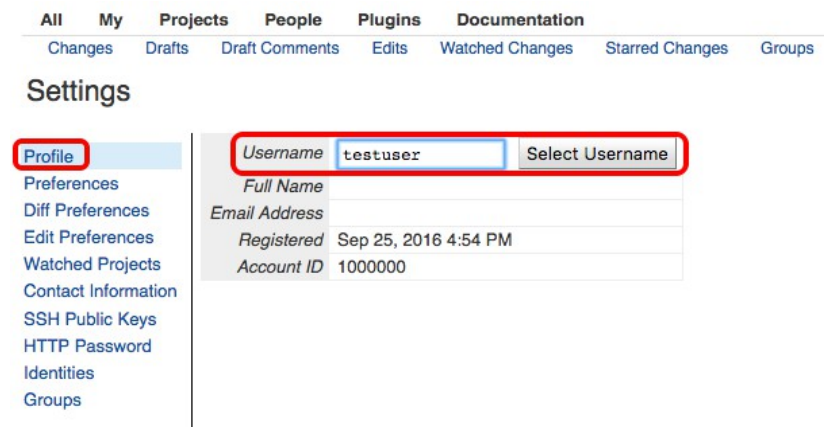
Sign In to Gerrit Code Review at 52.91.212.22



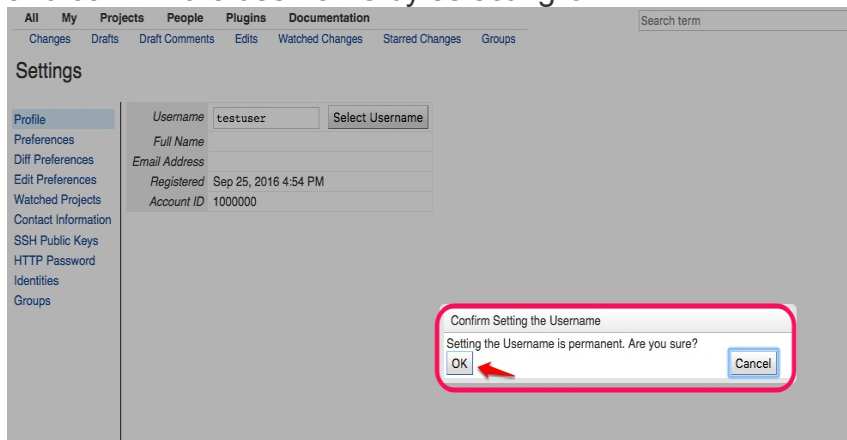
- After successful sign in we can edit settings.



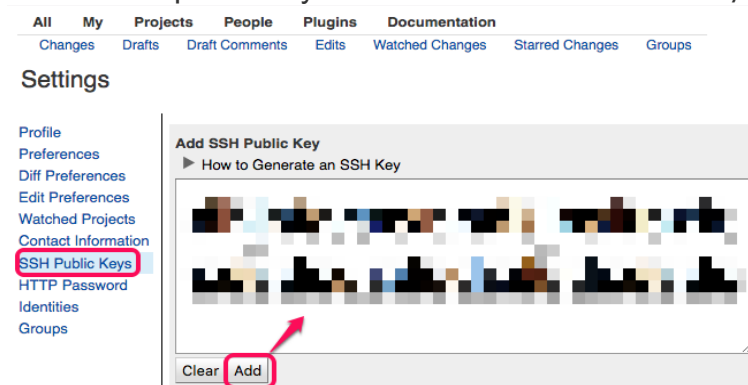
- Under profile select username to setup.



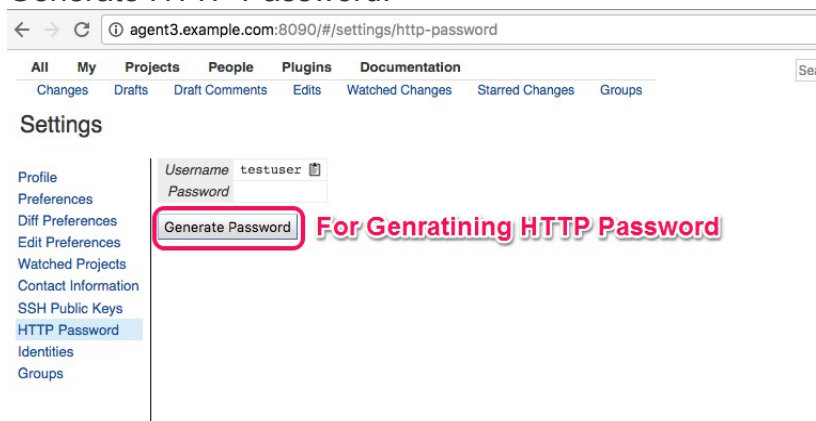
- and confirm the username by selecting OK.



- Add a SSH public key to Gerrit under SSH Public Keys



- Generate HTTP Password.



## Zuul

Puppet module will install Zuul in Agent4. Zuul is a pipeline oriented project gating and automation system.

After installation visit the IP for Zuul Dashboard.

### Zuul Status

Real-time status monitor of Zuul, the pipeline manager between Gerrit and Workers.

Queue lengths: 0 events, 0 results.

Filters  Expand by default: ☐

#### gate

Changes that have been approved by core developers are enqueued in order in this pipeline

0

#### check-reject

Changes that have been rejected by core developers are enqueued in order in this pipeline

0

#### check

This pipeline runs jobs that operate after new patchset created.

0

#### post-merge

This pipeline runs jobs that operate after changes are merged.

0

Zuul version: 2.5.1.dev20

Last reconfigured: Sat Sep 24 2016 06:02:53 GMT+0530 (IST)

## Add/Modify Jenkins Job

Zuul creates Jenkins job according to the puppet configuration.

Edit the following file in Puppet-Master for making changes with respect to job or project.

- `/etc/puppet/modules/zuul/templates/jjb.yaml.erb`
- `/etc/puppet/modules/zuul/manifests/params.pp`

Finally add the modified entries in main site.pp

- `/etc/puppet/manifests/site.pp`

Once all is done, then pull changes from puppet server to zuul server(agent 4) using “puppet agent -t” command.

# Create Project

- Create a new project under Projects section.

All My **Projects** People Plugins Documentation

List [Create New Project](#)

Create Project **For creating a new project**

Project Name:

Rights Inherit From:  [Browse](#)

☐ Create initial empty commit

☐ Only serve as parent for other projects

[Create Project](#)

| Parent Suggestion | Project Name | Project Description                     |
|-------------------|--------------|-----------------------------------------|
| ►                 | All-Projects | Access inherited by all other projects. |

- Create dev branch

All My **Projects** People Plugins Documentation

List General **Branches** Tags Access Dashboards [Create New Project](#)

Project CI-Project

Filter

| Branch Name                  | Revision                                 |                                 |
|------------------------------|------------------------------------------|---------------------------------|
| HEAD                         | master                                   | <a href="#">[icon]</a> (gitweb) |
| refs/meta/config             | ae3da60215049a68c6c619443a7a041484d79566 | (gitweb)                        |
| <input type="checkbox"/> dev | 558cc4ed5bd9517a8e88a2645183ec5313925441 | (gitweb)                        |
| master                       | 558cc4ed5bd9517a8e88a2645183ec5313925441 | (gitweb)                        |

[Delete](#)

Branch Name:

Initial Revision:

[Create Branch](#)

- Clone test-project from agent3 for testuser and choose FALSE for Allow content merges

Project test-project

Clone | Clone with commit-msg hook | anonymous http | http | ssh

git clone http://testuser@agent3.example.com:8090/a/test-project

Description

Project Options

State: Active

Submit Type: Merge if Necessary

Allow content merges: FALSE (selected)

Create a new change for every commit not in the target branch

Require Change-Id in commit message:

Maximum Git object size limit:

Contributor Agreements

Require signed-off-by in commit message: INHERIT (false)

Save Changes

Project Commands

Commands: Run GC Create Change Edit Config

## Git-Review

We use git-review tool for submitting git branches to gerrit for review.

### Installation

We use pip to install git-review

```
pip install git-review
```

### Setup

By default, git-review will look for a remote named 'gerrit' for working with Gerrit. If the remote exists, git-review will submit the current branch to HEAD:refs/for/master at that remote.

If the Gerrit remote does not exist, git-review looks for a file called .gitreview at the root of the repository with information about the gerrit remote. Assuming that file is present,

git-review should be able to automatically configure your repository the first time it is run.

The name of the Gerrit remote is configurable; see the configuration section below.

#### **.gitreview configuration file**

Example .gitreview file (used to upload for git-review itself):

```
[gerrit]
host=review.openstack.org
port=29418
project=openstack-infra/git-review.git
defaultbranch=master
```

Note:

Required values: host, project

Optional values: port (default: 29418), defaultbranch (default: master), defaultremote (default: gerrit).

#### **What happens when you submit a change**

When you submit a change, git review does the following things:

It looks up which branch to push to (production or whatever) in the .gitreview file. If it can't find this information, it pushes to master

it figures out what "topic" to put on the revision (you can set the topic manually with -t)

if you're resubmitting a downloaded change, it will reuse the tag of the original change

if your commit summary contains a bug number like bug 12345, the tag will be bug/12345

otherwise, the tag will be the name of your local branch

it rebases your change against the HEAD of the branch you're pushing to (use -R to skip this)

if you are submitting more than one change at once, or submitting a change that's based on another unmerged change, it will ask you whether you really meant to do that (use -y to skip this)

it pushes the change for review

For more detailed informaton visit [openstack-docs](#)

## Git Workflow

### Cloning Project:

First we need to clone project to local directory to modify it.

e.g: `git clone http://agent3.example.com:8090/Project-Name`

### Change Branch to Dev:

After cloning project to local directory we need to change the branch to dev.

e.g: `git checkout dev` or `git checkout -b dev`

### Pull changes from dev:

Once you changed to dev branch you need to pull recent changes for dev branch.

e.g: `git pull origin dev`

### Modify your code:

Then you can start with your work of modifying the code.

### Add changes to git:

Once you done with all your changes you need to add changes to git.

e.g: `git add file-name` or `git add`

### Commit your changes:



Then commit your code, this time we need to take care, while committing code we need to follow the below given standard format

```
git commit -m "JIRA_ISSUE_ISSUE-ID"
```

e.g: `git commit -m "JIRA_ISSUE_TEST-1"`

Push your changes:

Once you committed your changes push code for code-review, we can push code with 2 ways either `git push` or `git-review`

e.g: `git push origin HEAD:refs/for/dev`

```
git-review
```

## Post Installation Steps

- Once the setup is configured successfully we can stop the puppet master-slave process.
- It is *not mandatory* to stop the process.
- To stop puppet-master login to master machine and run the following command.

```
service puppetmaster stop
```

- Similarly to stop puppet-agent login to each agent machine and run the following command.

```
service puppet stop
```