

SETTING UP JENKINS SERVER:

Use Digital Ocean to create a new Droplet, it will be used to serve Jenkins app.

1. Create a **new Jenkins user** and provide the necessary **sudo** privileges to the user.

```
[shivs-MacBook-Air:~ Maci$ ssh root@192.241.131.136
root@192.241.131.136's password:
Last failed login: Fri Jun  7 12:15:10 IST 2019 from 122.160.68.189 on ssh:notty
There were 2 failed login attempts since the last successful login.
Last login: Fri Jun  7 11:46:08 2019 from 122.160.68.189
nvm is not compatible with the npm config "prefix" option: currently set to "/usr/local"
Run `npm config delete prefix` or `nvm use --delete-prefix v6.11.2 --silent` to unset it.
[root@demo ~]# adduser jenkins
[root@demo ~]# usermod -a -G wheel jenkins
```

2. Login into the user and install Git.
3. Jenkins is a Java application, so the first step is to install Java.
Run the following command to install Java

```
[jenkins@demo ~]$ sudo yum install java-1.8.0-openjdk-devel
```

Run the following commands to load repository:

```
[jenkins@demo ~]$ curl --silent --location http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo | sudo tee
/etc/yum.repos.d/jenkins.repo
[jenkins@demo ~]$ sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key
Last login: Thu Jun  6 12:01:00 IST 2019 from demo.pnp.otssolutions.com on pts/4
[jenkins@demo ~]$ yum install git
```

Run the following command to install Jenkins:

```
[jenkins@demo ~]$ sudo yum install jenkins
```

After the setup is complete, start the server and check the status:

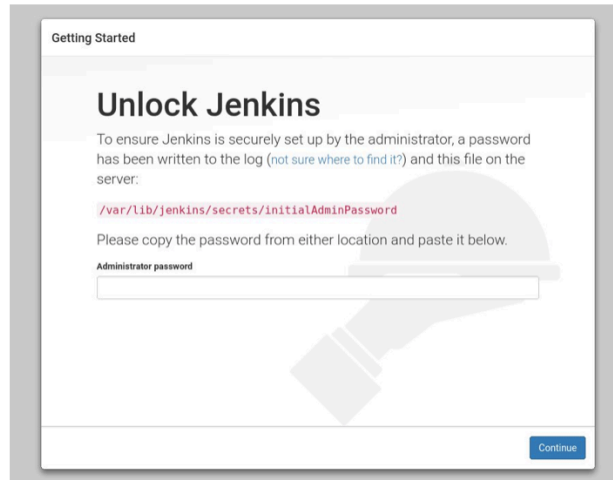
```
[jenkins@demo ~]$ sudo systemctl start jenkins
```

Output should be like this:

```
[jenkins@demo ~]$ sudo systemctl status jenkins
[sudo] password for jenkins:
• jenkins.service - LSB: Jenkins Automation Server
   Loaded: loaded (/etc/rc.d/init.d/jenkins; bad; vendor preset: disabled)
   Active: active (running) since Thu 2019-06-06 12:06:20 IST; 24h ago
     Docs: man:systemd-sysv-generator(8)
  Process: 7529 ExecStop=/etc/rc.d/init.d/jenkins stop (code=exited, status=0/SUCCESS)
  Process: 7543 ExecStart=/etc/rc.d/init.d/jenkins start (code=exited, status=0/SUCCESS)
    Tasks: 52
   Memory: 1.1G
    CGroup: /system.slice/jenkins.service
            └─4099 ssh-agent
              7586 /etc/alternatives/java -Dcom.sun.akuma.Daemon=daemonized -Djava.awt.headless=true -Dma
```

*This setup is performed on macOS Mojave Version 10.14.5
Please find suitable commands from the Internet for other OS.*

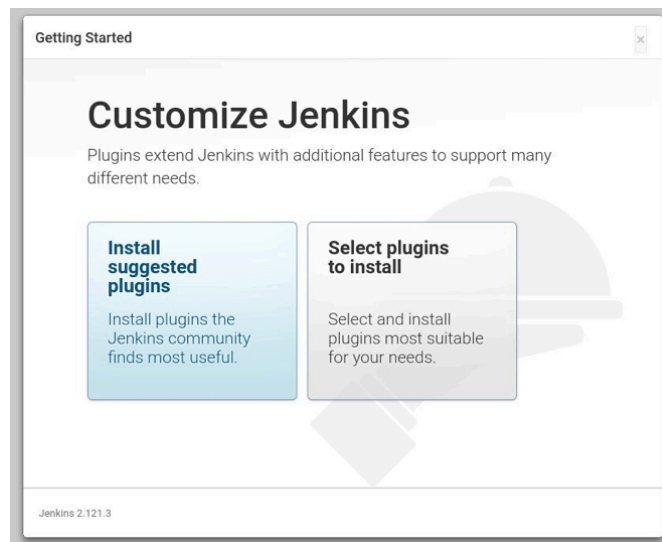
4. Log into the IP: <http://192.241.131.136:8080/>



5. Run the following command and enter the administrator password into Jenkins, change it in future via dashboard for an easy login.

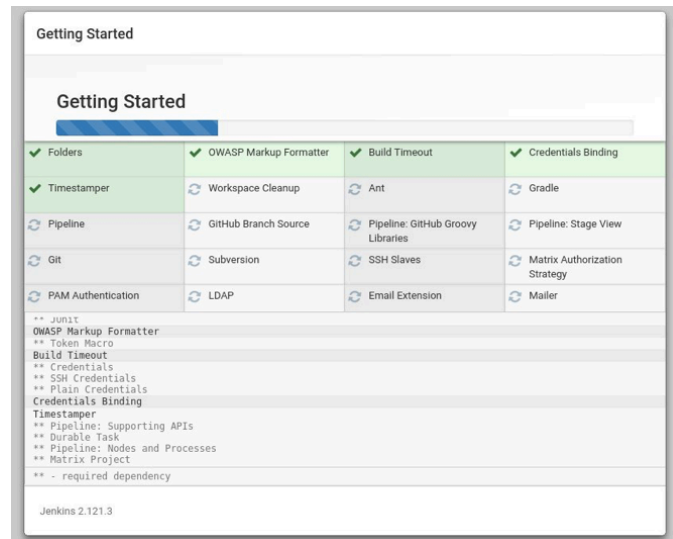
```
[jenkins@demo ~]$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword  
8e5d4345cb3949e88216d3f86482773e
```

6. Click on Continue and “Install suggested Plugins”.



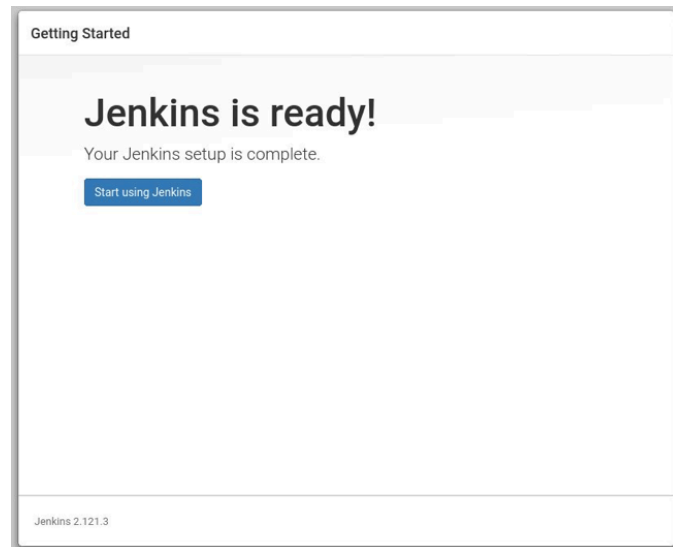
*This setup is performed on macOS Mojave Version 10.14.5
Please find suitable commands from the Internet for other OS.*

7. Following screen will come and the Plugins will get installed.



8. Create a new user and Click on Save & Continue.

9. The Page will prompt that Jenkins is ready

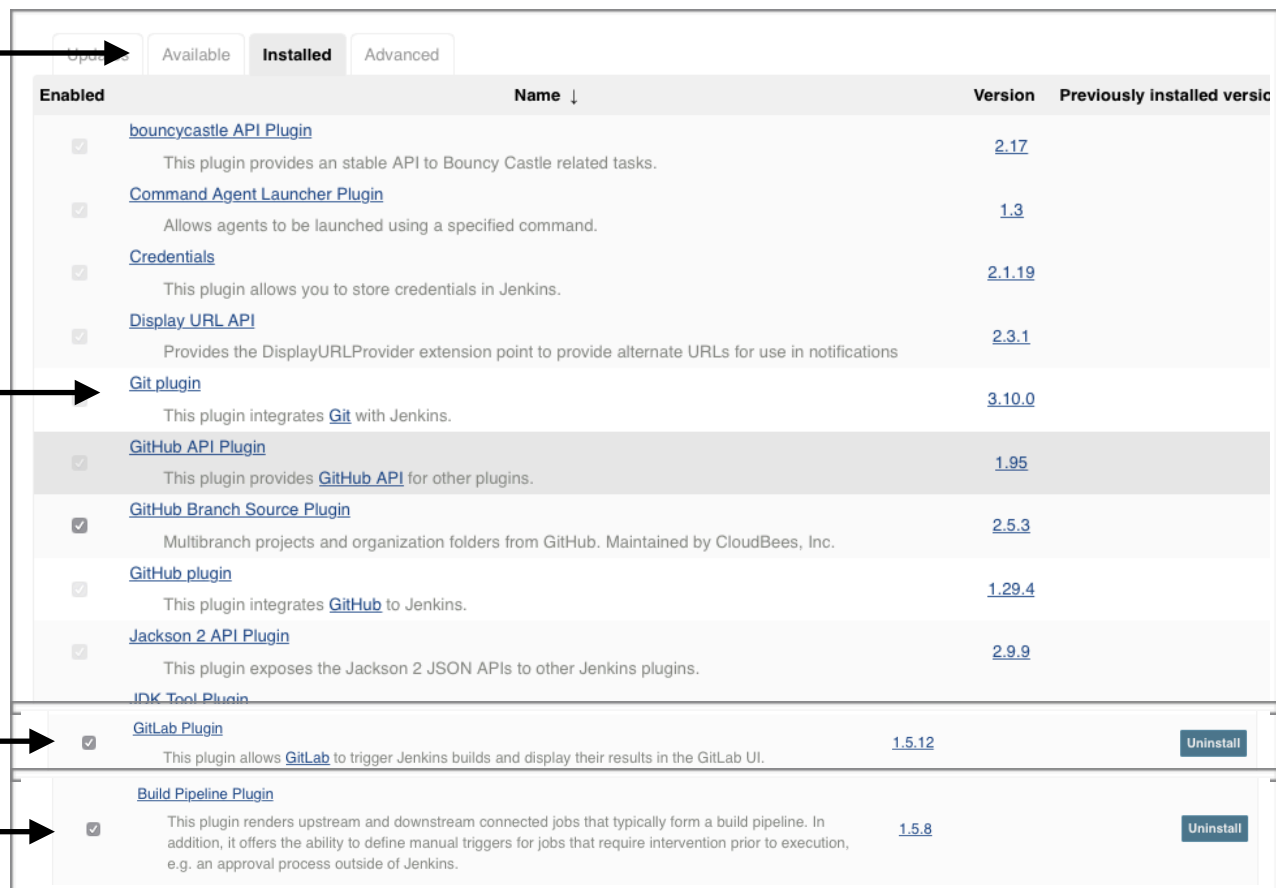


This setup is performed on macOS Mojave Version 10.14.5
Please find suitable commands from the Internet for other OS.

10. The Jenkins dashboard will get loaded.



11. Install the GitHub Plugin, GitLab Plugin, Build Pipeline to integrate the environments.
Go to **Manage Jenkins -> Manage Plugins -> Search for GitHub and GitLab Plugin**
in the available tabs



12. Create a new Freestyle job/item.



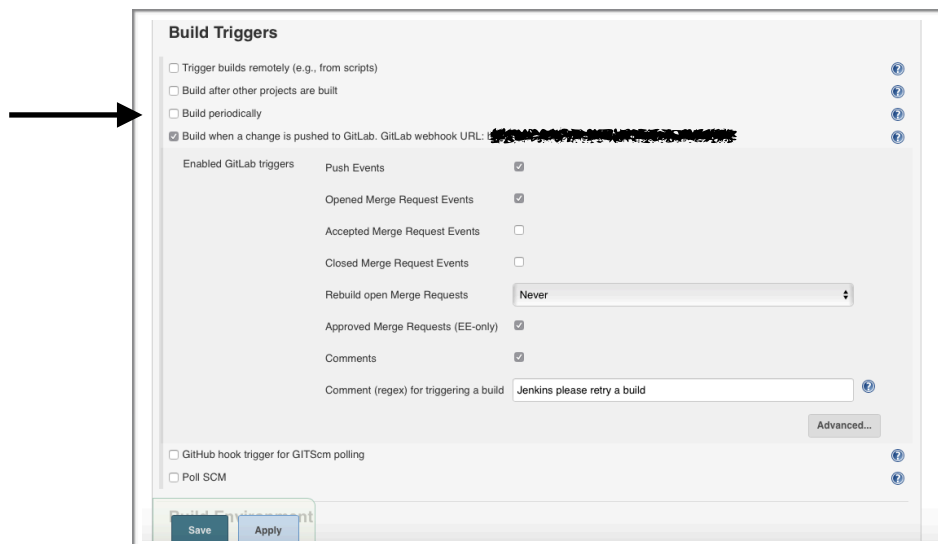
The screenshot shows the Jenkins 'Enter an item name' form. At the top, there is a text input field with a cursor inside. Below the field is a small red asterisk and the text 'Required field'. Below the input field is a section titled 'Freestyle project' with a blue icon of a box and a checkmark. The text below the icon reads: 'This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.'

13. Configure the project.

- Select **Git from Source Code Management**. Enter the required details.

*Note: If you enter the HTTPS URL then use GitLab ID credentials else if SSH URL is used then the SSH Credential should be used.

- Select the **“Build when a change is pushed to GitLab. GitLab Webhook URL: http://192.241.131.136:8080/xxxxxxx”**



The screenshot shows the Jenkins 'Build Triggers' configuration page. A black arrow points to the 'Build when a change is pushed to GitLab' option, which is selected. The 'GitLab webhook URL' is set to 'http://192.241.131.136:8080/xxxxxxx'. Below this, there is a section for 'Enabled GitLab triggers' with a table of options:

Enabled GitLab triggers	Push Events
Push Events	<input checked="" type="checkbox"/>
Opened Merge Request Events	<input checked="" type="checkbox"/>
Accepted Merge Request Events	<input type="checkbox"/>
Closed Merge Request Events	<input type="checkbox"/>
Rebuild open Merge Requests	<input type="text" value="Never"/>
Approved Merge Requests (EE-only)	<input checked="" type="checkbox"/>
Comments	<input checked="" type="checkbox"/>
Comment (regex) for triggering a build	<input type="text" value="Jenkins please retry a build"/>

At the bottom of the page, there are buttons for 'Save' and 'Apply', and an 'Advanced...' link.

*This setup is performed on macOS Mojave Version 10.14.5
Please find suitable commands from the Internet for other OS.*

Now, the Build steps can be added. These are the commands that will be used to test your project.

Go to **Source Code Management -> Build -> Select Execute Shell** from the dropdown.



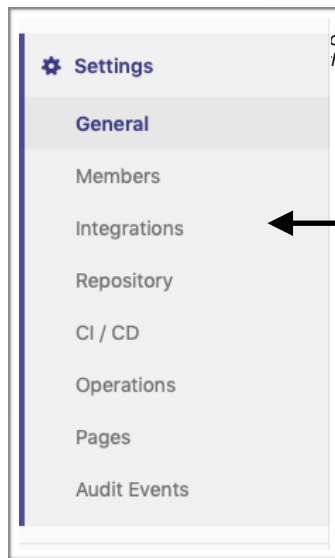
- Click Save

In the previous step, there is also a Webhook URL given for every project, GitLab Webhooks are basically created to provide other applications with real-time information. A **Webhook** delivers data to other applications as it happens, meaning you get data immediately.

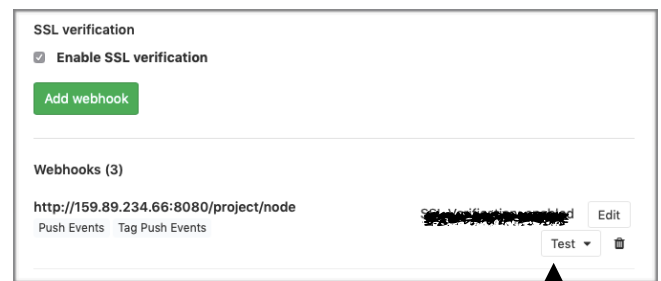
To create a Webhook, go to:

Settings -> Integrations -> Copy and Paste the URL generated in previous step -> Select the triggers for which you want the updates-> Add Webhook -> Test it

If it gives back an **HTTP 200** response then the Webhook is setup.

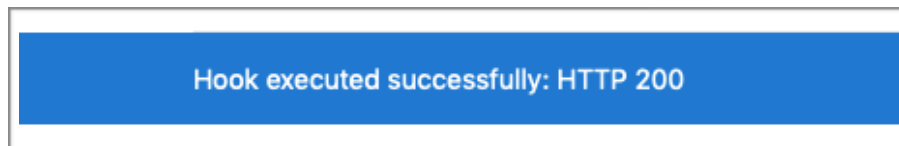
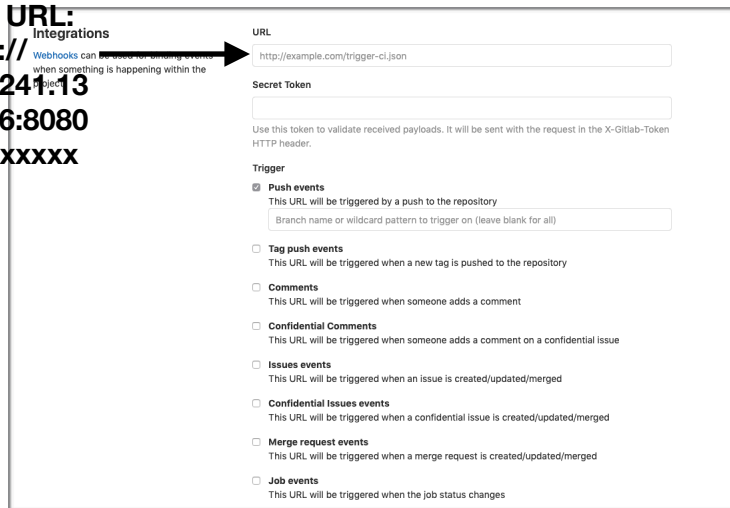


Java Version 10.14.5
the Internet for other OS.



Test the Webhook

Add URL:
http://
192.241.13
1.136:8080
/xxxxxxxx



***Note: Do not enable SSL verification**

Now if commit is pushed to the repository, you should see the Jenkins job start running. Once the job completes, you should see the status next to the commit in GitLab.

There is also a feature of adding **Email Notifications** about the status of Builds which is available as a Post-Build action.

- Select **Email Notification** from the drop-down.

Add the list of recipients, select the favorable options for receiving notifications.

Now go to
Jenkins -> Manage Jenkins -> Email Notification -> Fill in SMTP server, and enable SMTP authentication

Fill in **Username and Password (Credentials of the sender)**

Fill in SMTP Port (25 or 587)

Test the configuration by sending a mail to any recipient, the build will start automatically will **fail** once so that the recipient could receive the mail.

Also fill in the System Admin Email address (Optional)

The SMTP server for your host can be found on this link:

<https://www.arclab.com/en/kb/email/list-of-smtp-and-pop3-servers-mailserver-list.html>

*Note if the **Test Configuration** shows **ERROR**

Add this

“-Dmail.smtp.starttls.enable=“true””

to

“JENKINS_JAVA_OPTIONS=“-Djava.awt.headless=true”

in your **/etc/sysconfig/jenkins** file.

This setup is performed on macOS Mojave Version 10.14.5
Please find suitable commands from the Internet for other OS.

14. A new project will look like this. Build the projects to see the output.

The screenshot shows the Jenkins web interface for a project named 'Node-App Test'. The top navigation bar includes the Jenkins logo, a search bar, and user links for 'admin' and 'log out'. The left sidebar contains a 'Build History' table and a list of project actions: 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build Now', 'Delete Project', 'Configure', 'GitHub Hook Log', and 'Rename'. The main content area displays the project name, a description of the CI/CD pipeline, and a reference link. Below this, there are links for 'Workspace' and 'Recent Changes'. The 'Build History' table lists builds from #17 to #28, showing their status (blue for success, red for failure) and timestamps. The 'Downstream Projects' section lists 'Node-App Deploy'. The 'Permalinks' section provides links to various build states like 'Last build', 'Last stable build', 'Last successful build', 'Last failed build', 'Last unsuccessful build', and 'Last completed build'.

Build Number	Status	Timestamp
#28	Success	Jun 3, 2019 4:47 PM
#27	Success	May 30, 2019 1:08 PM
#26	Success	May 29, 2019 2:31 PM
#25	Success	May 29, 2019 1:52 PM
#24	Success	May 29, 2019 1:51 PM
#23	Success	May 29, 2019 1:38 PM
#22	Success	May 29, 2019 1:36 PM
#21	Success	May 29, 2019 1:35 PM
#20	Success	May 29, 2019 12:02 PM
#19	Success	May 29, 2019 11:51 AM
#18	Success	May 29, 2019 11:49 AM
#17	Success	May 29, 2019 10:50 AM

You can also create the test files and deploy files and build them as separate projects to create a Pipeline for the same.

Here the Upstream project will be the test project and the Downstream project will be the deployment of the app, both of them are interlinked such that only if the test is passed the deploy will happen.

15. Create a new job for deployment something like this.

The screenshot shows the Jenkins web interface for a project named 'Node-App Deploy'. The top navigation bar includes the Jenkins logo, a search bar, and user links for 'admin' and 'log out'. The left sidebar contains a 'Build History' table and a list of project actions: 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build Now', 'Delete Project', 'Configure', 'GitHub Hook Log', and 'Rename'. The main content area displays the project name, a description of the CI/CD pipeline, and a reference link. Below this, there are links for 'Workspace' and 'Recent Changes'. The 'Build History' table lists builds from #1 to #12, showing their status (blue for success, red for failure) and timestamps. The 'Upstream Projects' section lists 'Node-App Test'. The 'Permalinks' section provides links to various build states like 'Last build', 'Last stable build', 'Last successful build', 'Last failed build', 'Last unsuccessful build', and 'Last completed build'.

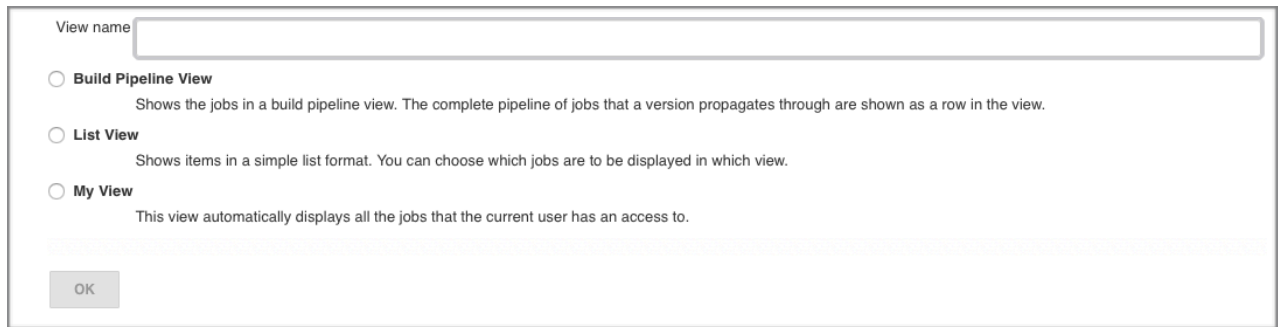
Build Number	Status	Timestamp
#12	Success	Jun 3, 2019 4:47 PM
#11	Success	May 30, 2019 1:09 PM
#10	Success	May 29, 2019 2:31 PM
#9	Success	May 29, 2019 2:31 PM
#8	Success	May 29, 2019 1:51 PM
#7	Success	May 29, 2019 1:38 PM
#6	Success	May 29, 2019 1:38 PM
#5	Success	May 29, 2019 1:36 PM
#4	Success	May 29, 2019 1:36 PM
#3	Success	May 29, 2019 1:35 PM
#2	Success	May 29, 2019 12:02 PM
#1	Success	May 29, 2019 12:02 PM

16.

This setup is performed on macOS Mojave Version 10.14.5
Please find suitable commands from the Internet for other OS.

Ensure that 'Build Pipeline' plugin is installed and to Build a Pipeline, follow the below steps: Click on the '+' symbol.

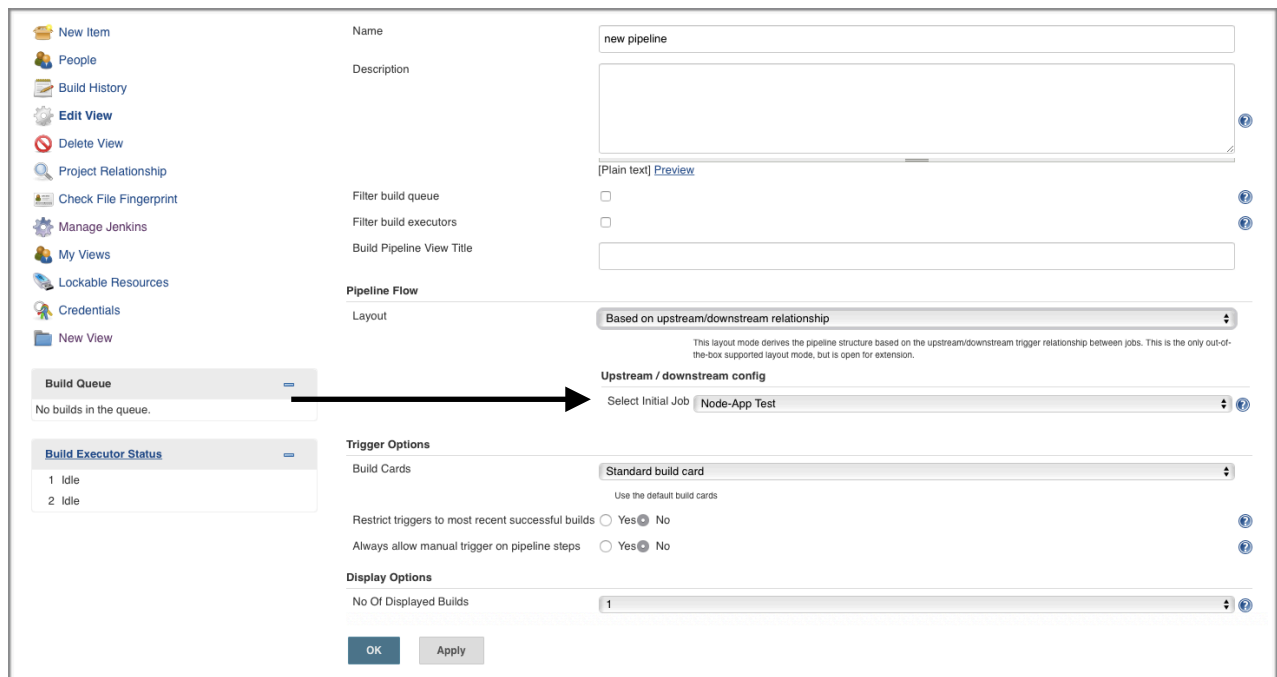
17. Select 'Build Pipeline View'



A configuration dialog for selecting a view. It features a 'View name' text input field at the top. Below it are three radio button options: 'Build Pipeline View', 'List View', and 'My View'. Each option has a descriptive text block below it. 'Build Pipeline View' states: 'Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.' 'List View' states: 'Shows items in a simple list format. You can choose which jobs are to be displayed in which view.' 'My View' states: 'This view automatically displays all the jobs that the current user has an access to.' At the bottom left is an 'OK' button.

18. Give a new name and select **OK**.

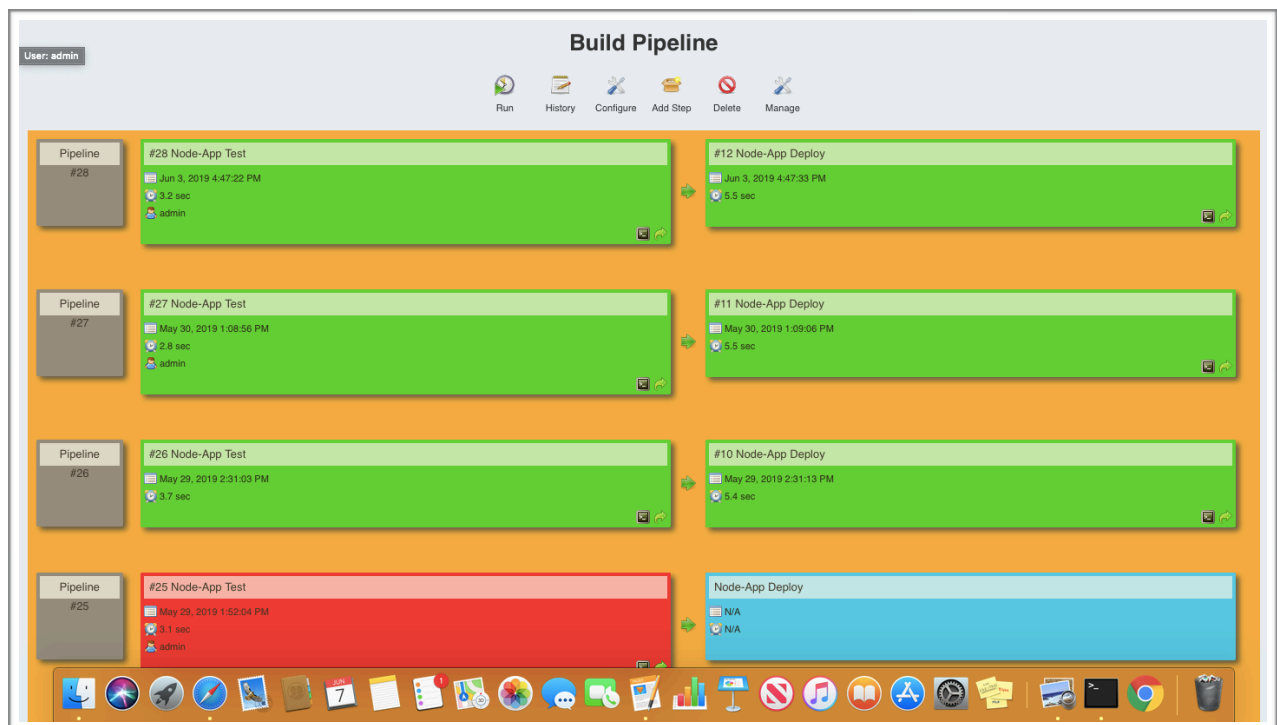
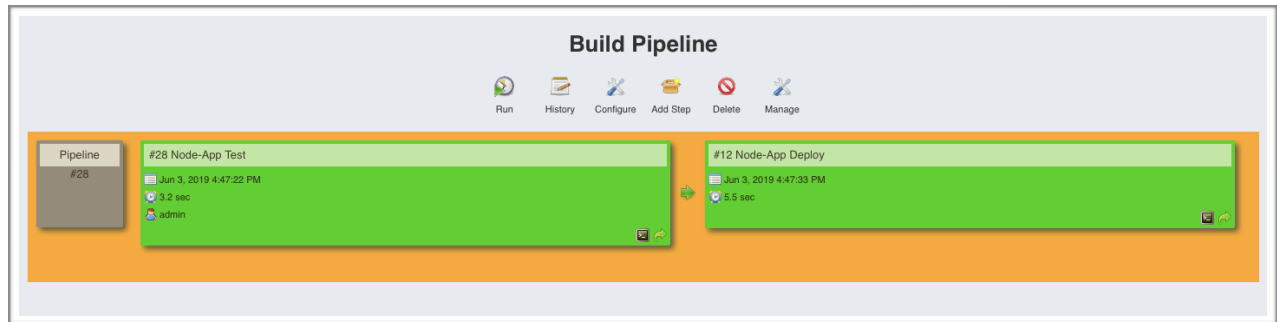
19. Select the **Initial Job** and hit **OK**.



A screenshot of the 'Build Pipeline' configuration page. On the left is a sidebar with navigation links: 'New Item', 'People', 'Build History', 'Edit View', 'Delete View', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', 'My Views', 'Lockable Resources', 'Credentials', and 'New View'. The main area is divided into sections: 'Name' (with input 'new pipeline'), 'Description' (with a text area), 'Filter build queue' (checkbox), 'Filter build executors' (checkbox), 'Build Pipeline View Title' (input), 'Pipeline Flow' (dropdown set to 'Based on upstream/downstream relationship'), 'Upstream / downstream config' (dropdown set to 'Node-App Test'), 'Trigger Options' (including 'Build Cards' set to 'Standard build card', 'Restrict triggers to most recent successful builds' set to 'No', and 'Always allow manual trigger on pipeline steps' set to 'No'), and 'Display Options' (dropdown set to '1'). An arrow points from the 'Build Queue' section on the left to the 'Upstream / downstream config' dropdown. At the bottom are 'OK' and 'Apply' buttons.

*This setup is performed on macOS Mojave Version 10.14.5
Please find suitable commands from the Internet for other OS.*

20. A new pipeline gets created, the pipeline can be configure to as many builds as required by the user e.g. :



As shown in the above figure the Node-App-Deploy only gets executed when the the Node-App-Test else it will not be executed.

JENKINS IS SETUP!!!

*Note: This project is hosted on GitHub and not GitLab. GitLab's setup can be found below.

Note: The SSH configuration between GitLab/GitHub with Jenkins must be setup before doing any project.

The steps are given below:

From your root ID generate SSH keys using the following command:

```
ssh-keygen -t rsa
```

Save the generated keys on your preferable location and print the public key.

```
cat ~/<location of the folder .ssh>/id_rsa.pub
```

Save this public key into your GitLab/GitHub account. Print the private key.

```
cat ~/<location of the folder .ssh>/id_rsa
```

Save the private key into your Jenkins account.

Jenkins -> Credentials -> System ->Add Credential -> Select SSH with private key from the drop-down and enter the copied private key

This method can be verified by executing any Git command, if it still asks for password that means the setup is not complete.

This error can be rectified by creating a new remote and adding your branch into it.

```
git remote add <name of the remote> <repository URL>
```

and then execute all commands with this remote e.g. :

```
git pull <name of the remote> <name of the branch>
```

The SSH configuration must be done between users on two different servers. Generate the keys and save the public key created from the Jenkins User account into the other account in a new file in .ssh called 'authorized_keys'

```
vim ~/<location of .ssh folder>/authorized_keys
```

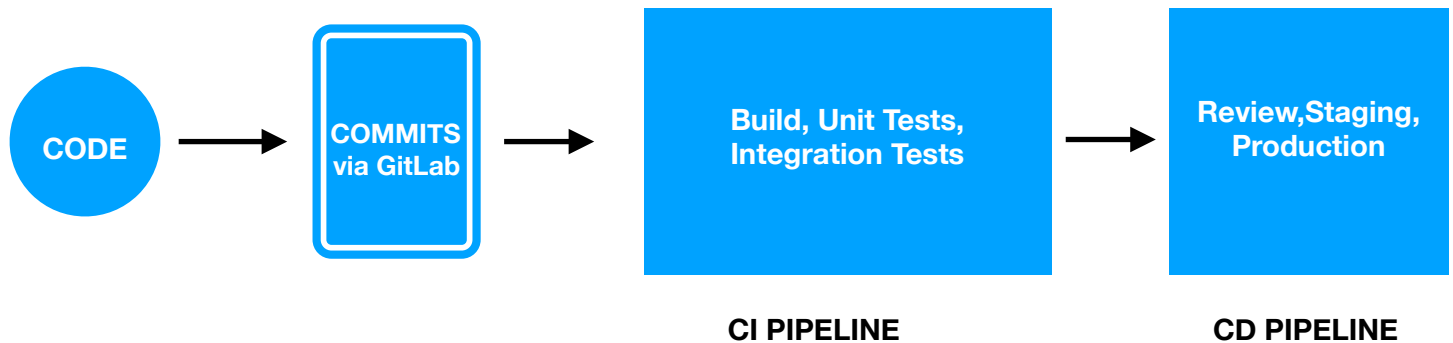
Save the public key here.

```
chmod 700 ~/.ssh  
chmod 640 ~/.ssh/*
```

You must be able to login into the other user without password.

```
eval `ssh-agent`  
ssh-add  
ssh root@JENKINS.SERVER.IP  
su - jenkins  
ssh <username>@NODE.SERVER.IP  
SSH CONFIGURATION IS COMPLETED!!
```

GITLAB CI/CD Basics:



GitLab is a web-based **Git** repository. It focuses on managing software development projects and its files, and tracking their changes from time-to-time. The information is stored in a data structure called repository.

GitHub and GitLab deliver the same functionalities, only difference is GitLab offers usage by dev teams apart from users.

It also offers CI/CD pipelining functions. CI/CD is short for Continuous Integration and Continuous Deployment. It enables teams to build, test and release software at a faster rate. It removes human interaction and automates the process till the deployment stage. Jenkins, the open-source tool can also be used for the same, thus offering integration with GitLab to run the project over specified servers and generate a pipeline for it.

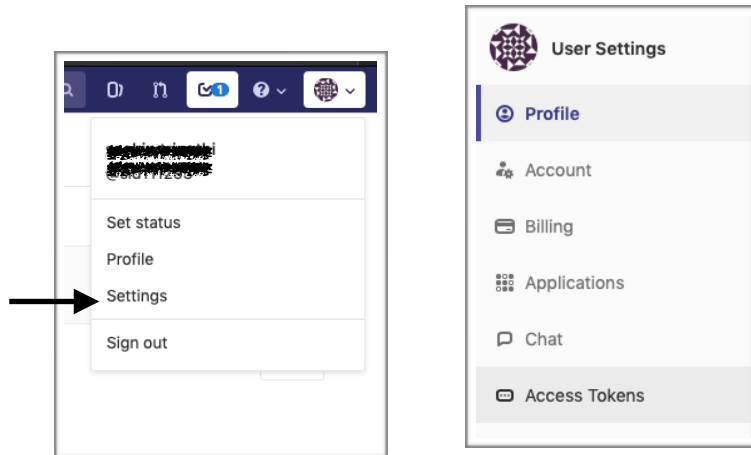
The steps required for setting up any project over GitLab and integrating it with Jenkins are as follows:

1. Set up the project over **GitLab** and configure SSH over GitLab and Jenkins.
Jenkins -> Credentials -> System -> Right click on Global credentials

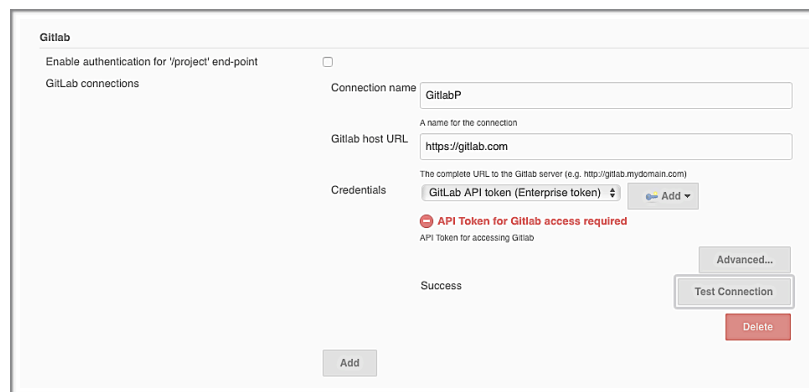
The screenshot shows the Jenkins 'Add Credentials' dialog box. The 'Scope' is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'ID' is 'Privatekeyforent'. The 'Description' is 'private key for enterprise app'. The 'Username' is 'git'. The 'Private Key' section is selected with the radio button 'Enter directly'. The 'Key' field is masked with a lock icon and the text 'Concealed for Confidentiality'. There is a 'Replace' button next to it. The 'Passphrase' field is empty.

(unrestricted) -> Add Credentials

Select SSH Private key from the drop down and enter the ID, Username & private key and enter the description.



2. Generate an Access Token.



****Ensure that Jenkins GitLab Plugin and Jenkins Git Plugin are installed.***

3. Add this token to Jenkins.

Jenkins -> Credentials -> System -> Right click on Global credentials (unrestricted) -> Add Credentials

Select GitLab API Token from the drop down and enter Access Token as the ID, and enter the description.

4. Now connection between Jenkins and GitLab can be established.

Go to Jenkins -> Manage Jenkins -> Configure System -> Gitlab

and enter the required details as shown below:

Select the **API Token** from drop down for the Credentials field and test the connection.

It will output “**Success**”

*Note: If there is no particular Host URL, use <https://gitlab.com>

4. Create a new Freestyle job/item.

5. Configure the project.

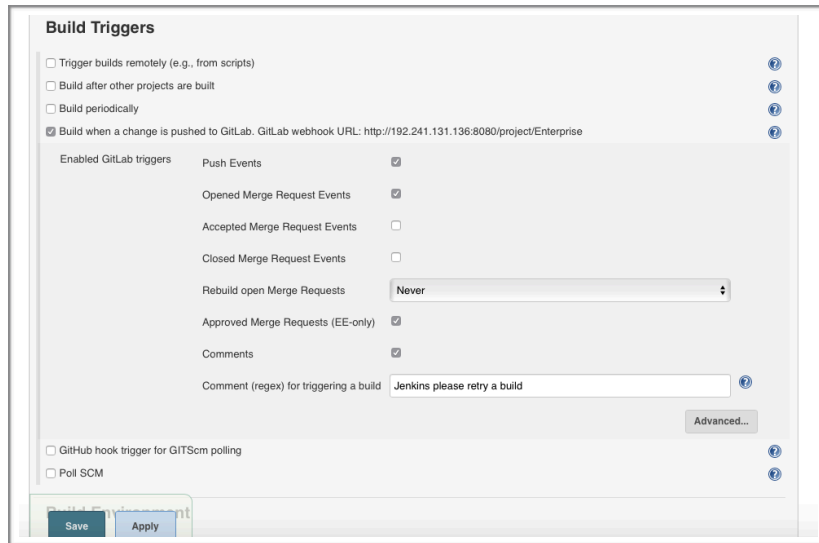
- Select **GitLab Connection**

- Select **Git from Source Code Management**

*Note: If you enter the HTTPS URL then use GitLab ID credentials else if SSH URL is used then the SSH Credential should be used.

*This setup is performed on macOS Mojave Version 10.14.5
Please find suitable commands from the Internet for other OS.*

- Select Build Triggers and the **“Build when a change is pushed to GitLab. GitLab Webhook URL: http://192.241.131.136:8080/xxxxxxx”**



The screenshot shows the 'Build Triggers' configuration page in Jenkins. The 'Build when a change is pushed to GitLab' option is selected, and the 'GitLab webhook URL' is set to 'http://192.241.131.136:8080/project/Enterprise'. Under 'Enabled GitLab triggers', the following options are checked: 'Push Events', 'Opened Merge Request Events', 'Approved Merge Requests (EE-only)', and 'Comments'. The 'Rebuild open Merge Requests' dropdown is set to 'Never'. The 'Comment (regex) for triggering a build' field contains 'Jenkins please retry a build'. At the bottom, there are 'Save' and 'Apply' buttons.

Now, the Build steps can be added. These are the commands that will used to test your project.

Go to **Source Code Management -> Build -> Select Execute Shell** from the drop-down.



The screenshot shows the 'Build' configuration page in Jenkins. The 'Execute shell' step is selected, and the 'Command' field contains the following text: `cd spec`
`npm test`. Below the command field, there is a link to 'See the list of available environment variables'. At the bottom left, there is an 'Add build step' button. At the bottom right, there is an 'Advanced...' button.

- Click Save

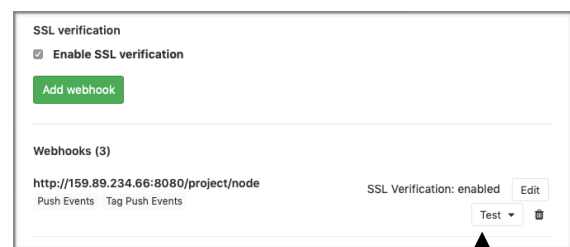
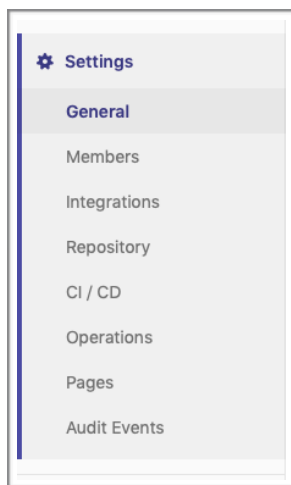
This setup is performed on macOS Mojave Version 10.14.5
Please find suitable commands from the Internet for other OS.

In the previous step, there is also a Webhook URL given for every project, GitLab Webhooks are basically created to provide other applications with real-time information. A **Webhook** delivers data to other applications as it happens, meaning you get data immediately.

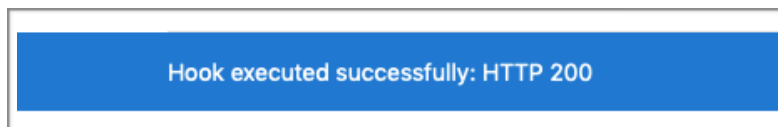
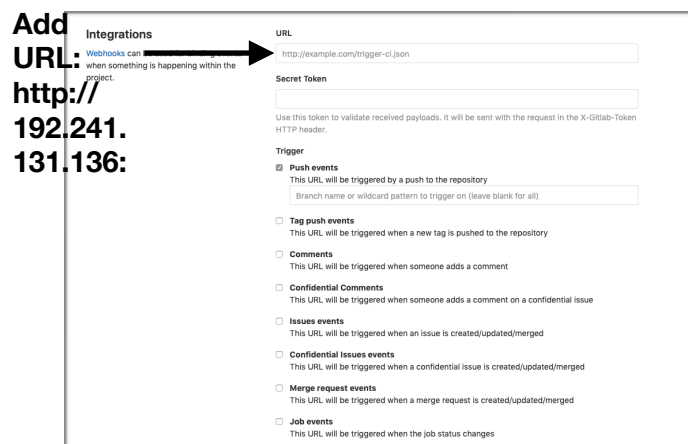
To create a Webhook, go to:

Settings -> Integrations -> Copy and Paste the URL generated in previous step -> Select the triggers for which you want the updates-> Add Webhook -> Test it

If it gives back an **HTTP 200** response then the Webhook is setup.



**Test the
Webhook**



***Note: Do not enable
SSL verification**

*This setup is performed on macOS Mojave Version 10.14.5
Please find suitable commands from the Internet for other OS.*

Now if a commit is pushed to the repository, you should see the Jenkins job start running. Once the job completes, you should see the status next to the commit in GitLab.

There is also a feature of adding **Email Notifications** about the status of Builds which is available as a Post-Build action.

- Select **Email Notification** from the drop-down.



The screenshot shows the 'Post-build Actions' configuration in Jenkins. The 'E-mail Notification' action is expanded, showing a text input for 'Recipients' with the value 'tandlum.suhash@otssolutions.com'. Below this, there is a description: 'Whitespace-separated list of recipient addresses. May reference build parameters like \$PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.' There are two checkboxes: 'Send e-mail for every unstable build' (checked) and 'Send separate e-mails to individuals who broke the build' (unchecked). Below the email notification section, the 'Publish build status to GitLab' action is visible. At the bottom left, there is a dropdown menu labeled 'Add post-build action'.

Add the list of recipients, select the favorable options for receiving notifications.

Now go to

Jenkins -> Manage Jenkins -> Email Notification -> Fill in SMTP server, and enable SMTP authentication

Fill in **Username and Password (Credentials of the sender)**



The screenshot shows the 'E-mail Notification' configuration page in Jenkins. The 'SMTP server' is set to 'smtp.office365.com'. The 'Default user e-mail suffix' is empty. The 'Use SMTP Authentication' checkbox is checked. The 'User Name' and 'Password' fields are filled with redacted text. The 'Use SSL' checkbox is unchecked. The 'SMTP Port' is set to '25'. The 'Reply-To Address' is empty. The 'Charset' is set to 'UTF-8'. The 'Test configuration by sending test e-mail' checkbox is unchecked. At the bottom, there are 'Save' and 'Apply' buttons.

*This setup is performed on macOS Mojave Version 10.14.5
Please find suitable commands from the Internet for other OS.*

Fill in SMTP Port (25 or 587)

Test the configuration by sending a mail to any recipient, the build will start automatically will **fail** once so that the recipient could receive the mail.

Also fill in the System Admin Email address (Optional)



The screenshot shows a Jenkins configuration window titled "Jenkins Location". It contains two input fields: "Jenkins URL" with the value "http://192.241.131.136:8080/" and "System Admin e-mail address" with the value "palak.arora@otssolutions.com". Both fields have a help icon (question mark) to their right.

The SMTP server for your host can be found on this link:

<https://www.arclab.com/en/kb/email/list-of-smtp-and-pop3-servers-mailserver-list.html>

*Note if the **Test Configuration** shows **ERROR**

Add this

“-Dmail.smtp.starttls.enable=“true””

to

“JENKINS_JAVA_OPTIONS=“-Djava.awt.headless=true”

in your **/etc/sysconfig/jenkins** file.

SETUP COMPLETE!

Test the build by making any change via GitLab.

It should show **SUCCESS**.

GITLAB CODE REVIEW PROCESS:

Typical flow of the Code Review Process:

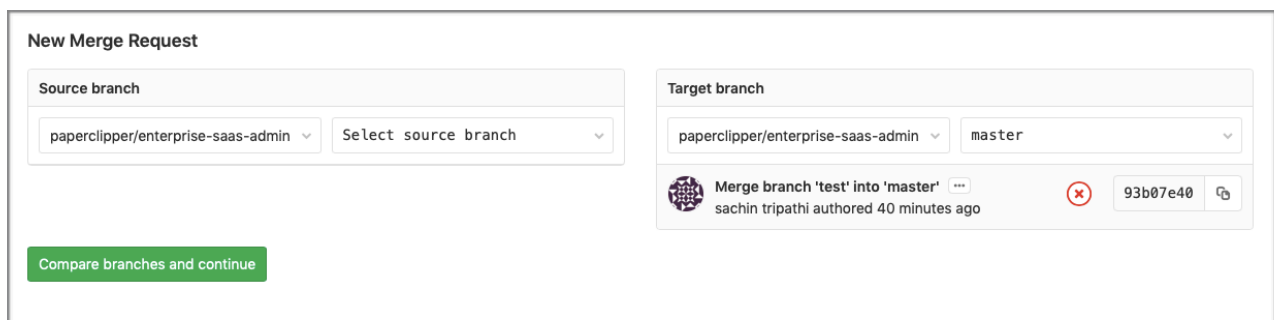
- A developer makes a change in their feature branch and tests it. When they're happy they push, and make a merge request.
- The developer assigns the merge request to a reviewer, who looks at it and makes line and design level comments as appropriate. When the reviewer is finished, they assign it back to the author.
- The author addresses the comments. This stage can go around for a while, but once both are happy, one assigns to a final reviewer who can merge.
- The final reviewer follows the same process again. The author again addresses any comments, either by changing the code or by responding with their own comments.
- Once the final reviewer is happy and the build is green, they will merge.

An example is presented below for reference:

1. Select **Merge Requests** from the project tab.
2. Click on **New Merge Request**.



3. Select **Source Branch** and **Target Branch**.



4. Click on **Compare branches and continue**.

This setup is performed on macOS Mojave Version 10.14.5
Please find suitable commands from the Internet for other OS.

Title

Start the title with **WIP:** to prevent a Work In Progress merge request from being merged before it's ready.
Add [description templates](#) to help your contributors communicate effectively!

Description

Write Preview

Write a comment or drag your files here...

Markdown and quick actions are supported [Attach a file](#)

Assignee [Assign to me](#)

Milestone

Labels

Source branch

Target branch [Change branches](#)

☐ Delete source branch when merge request is accepted.

☐ Squash commits when merge request is accepted. [?](#)

5. Fill in the details such as **Description, Assignee, Milestone etc.**

6. Click on **Submit Merge Request.**

7. If the assignee finds it suitable he/she can directly merge it or create an issue, the notification hence will be passed to the user who requested the merge.

Test jenkins

Request to merge `test_jenkins` **into** `master`
The source branch is **4 commits behind** the target branch

[Open in Web IDE](#) [Check out branch](#)

Pipeline #64999192 running for `d5a68ca7` on `test_jenkins`

Merge when pipeline succeeds

☐ Delete source branch ☐ Squash commits [?](#)

6 commits and **1 merge commit** will be added to master. [Modify merge commit](#)

You can merge this merge request manually using the [command line](#)

This setup is performed on macOS Mojave Version 10.14.5
Please find suitable commands from the Internet for other OS.

8. Now the merge request can also be seen in the Console Output of Jenkins Server.

Changes

Console Output

View as plain text

Edit Build Information

Delete build '#70'

Polling Log

Git Build Data

No Tags

Previous Build

Next Build

```
Triggered by GitLab Merge Request #2: paperclipper/test_jenkins => master
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/Enterprise
using credential GithubIDforent
> /usr/bin/git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> /usr/bin/git config remote.origin.url https://gitlab.com/paperclipper/enterprise-saas-admin.git # timeout=10
Fetching upstream changes from https://gitlab.com/paperclipper/enterprise-saas-admin.git
> /usr/bin/git --version # timeout=10
using GIT_ASKPASS to set credentials Github ID for Enterprise app
> /usr/bin/git fetch --tags --progress https://gitlab.com/paperclipper/enterprise-saas-admin.git +refs/heads/*:refs/remotes/origin/*
skipping resolution of commit d5a68ca709c980652a123f0032ec98bd2e4e9fb7, since it originates from another repository
> /usr/bin/git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> /usr/bin/git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 93b07e4023157f4978c9bbe353c7dd79b173e28b (refs/remotes/origin/master)
> /usr/bin/git config core.sparsecheckout # timeout=10
> /usr/bin/git checkout -f 93b07e4023157f4978c9bbe353c7dd79b173e28b
Commit message: 'Merge branch 'test' into 'master''
> /usr/bin/git rev-list --no-walk 93b07e4023157f4978c9bbe353c7dd79b173e28b # timeout=10
[Enterprise] $ /bin/sh -xe /tmp/jenkins1396458327425688147.sh
+ cd spec
+ npm test

> paperclipper@1.0.0 test /var/lib/jenkins/workspace/Enterprise
> ./node_modules/jasmine/bin/jasmine.js

Randomized with seed 34031
Started
Swagger: skipping unknown type "json".
-----
checking logs.
-----
(node:16248) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new
parser, pass option { useNewUrlParser: true } to MongoClient.connect.
Web server listening at: http://localhost:8900
```