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

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 12:51:00 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
```

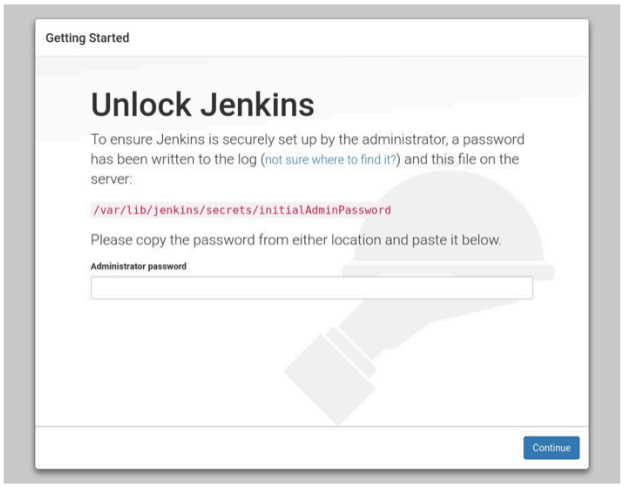
After the setup is complete, start the server and check the status:

```
[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
```

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

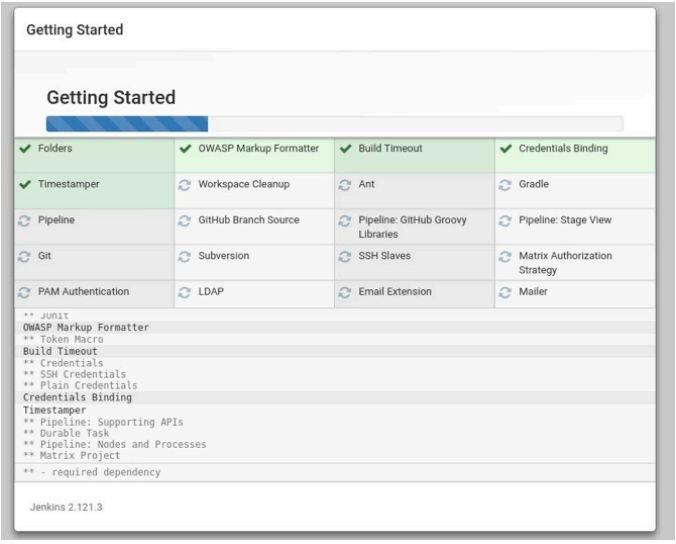


5. Run the following command and enter the administrator password into Jenkins, change it in future via dashboard for an easy login.

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

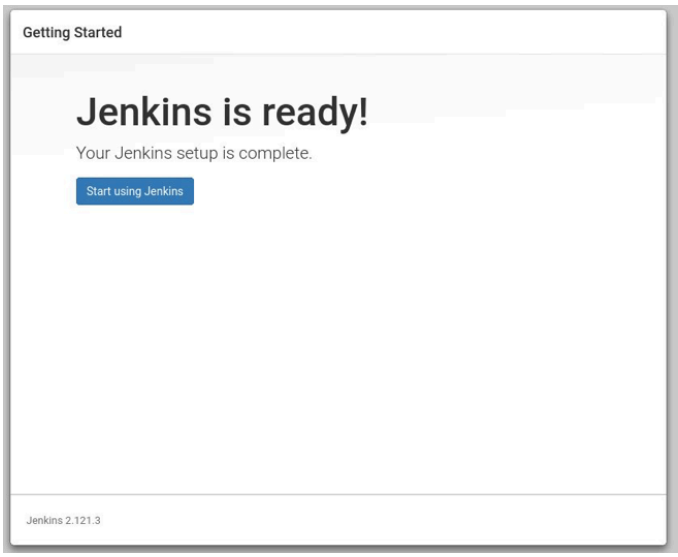
6. Click on Continue and “Install suggested Plugins”.

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



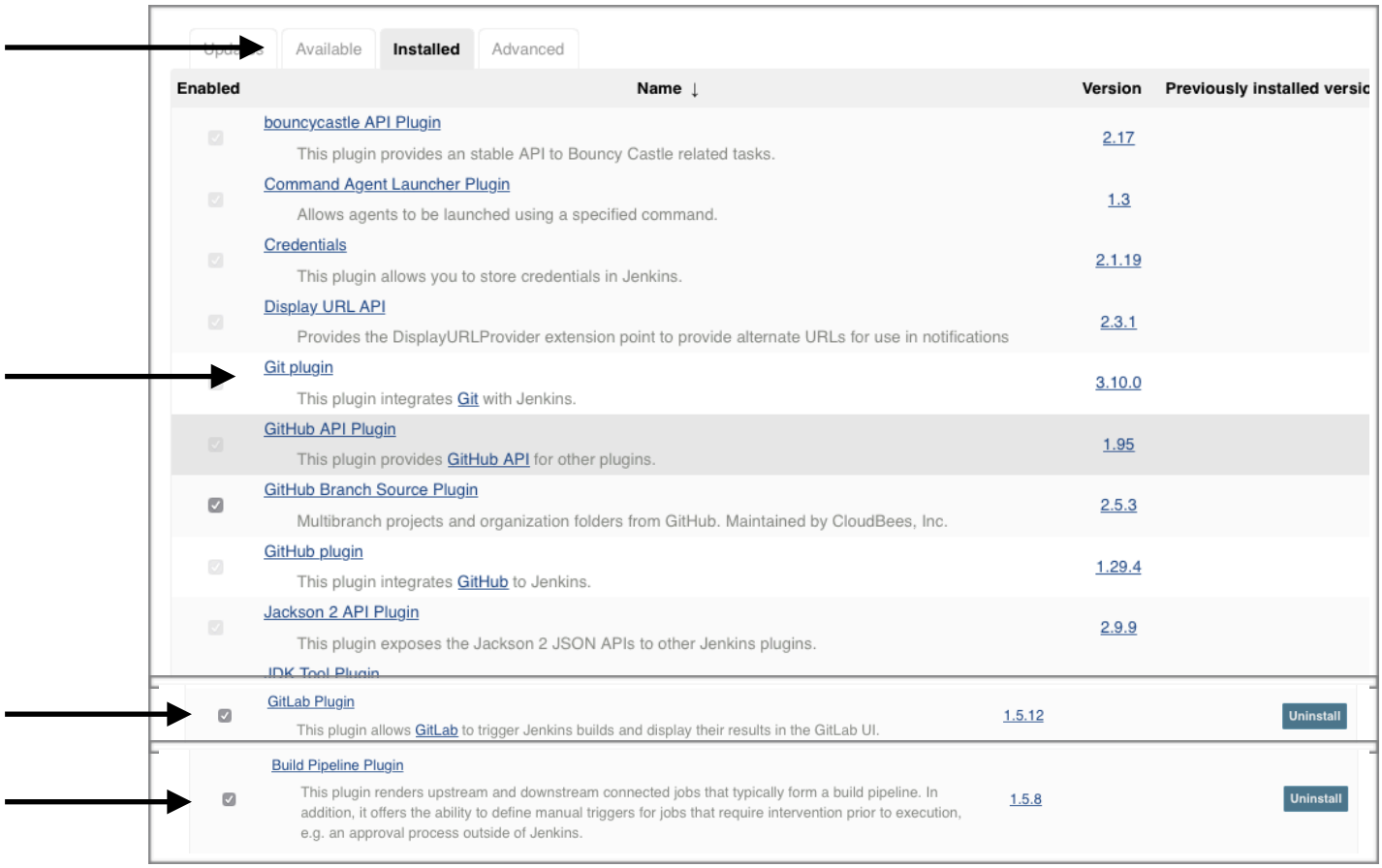
8. Create a new user and Click on Save & Continue.

9. The Page will prompt that Jenkins is ready



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**



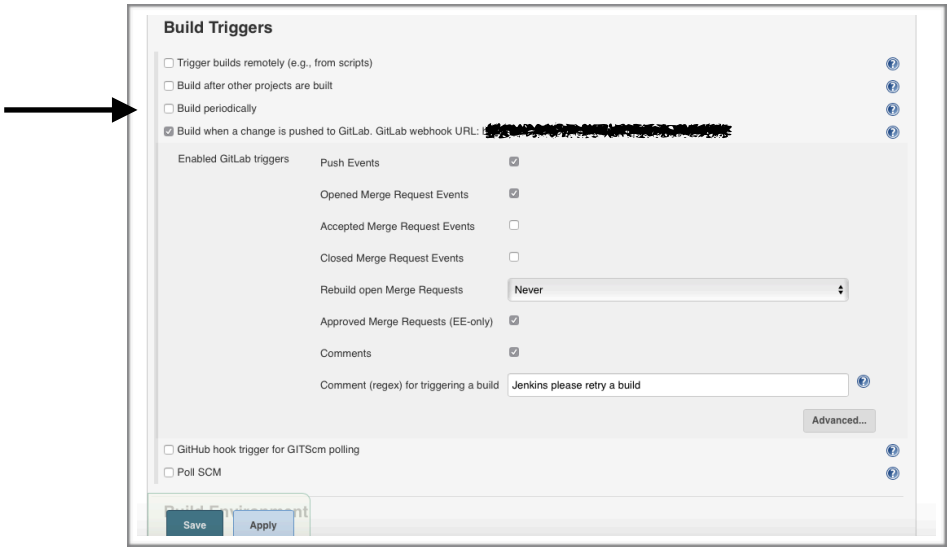
12. Create a new Freestyle job/item.

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/xxxxxxxx"***



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.**

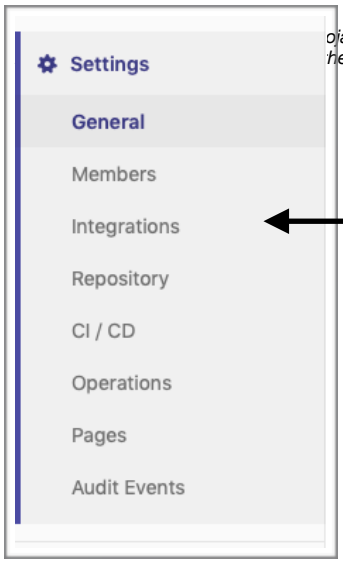
- Click Save

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.

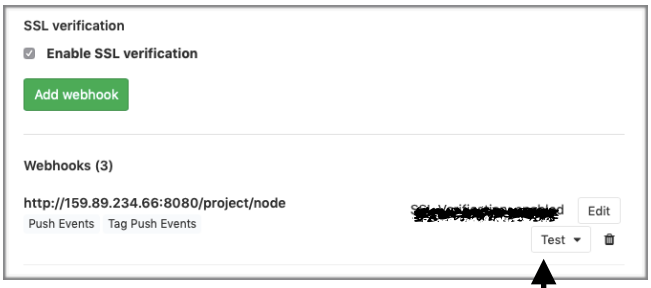
To create a Webhook, go to:

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.

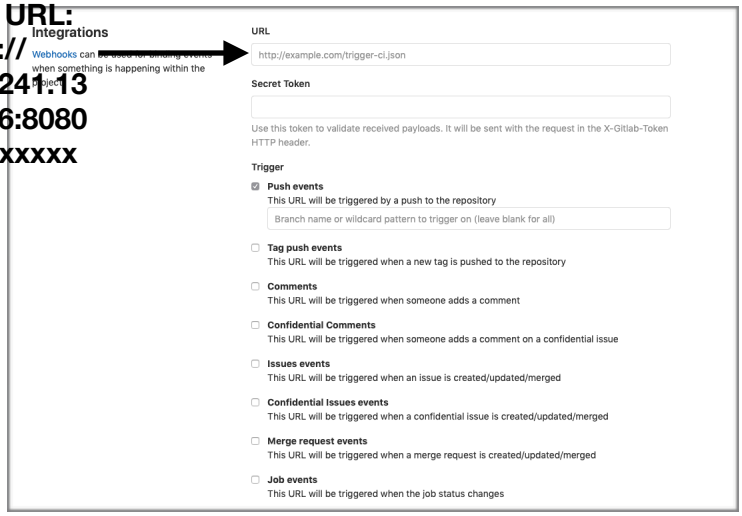


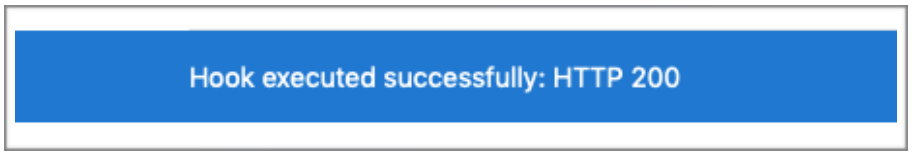
ojave Version 10.14.5
he Internet for other OS.



Test the Webhook

Add URL: http://192.241.131.136:8080/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)

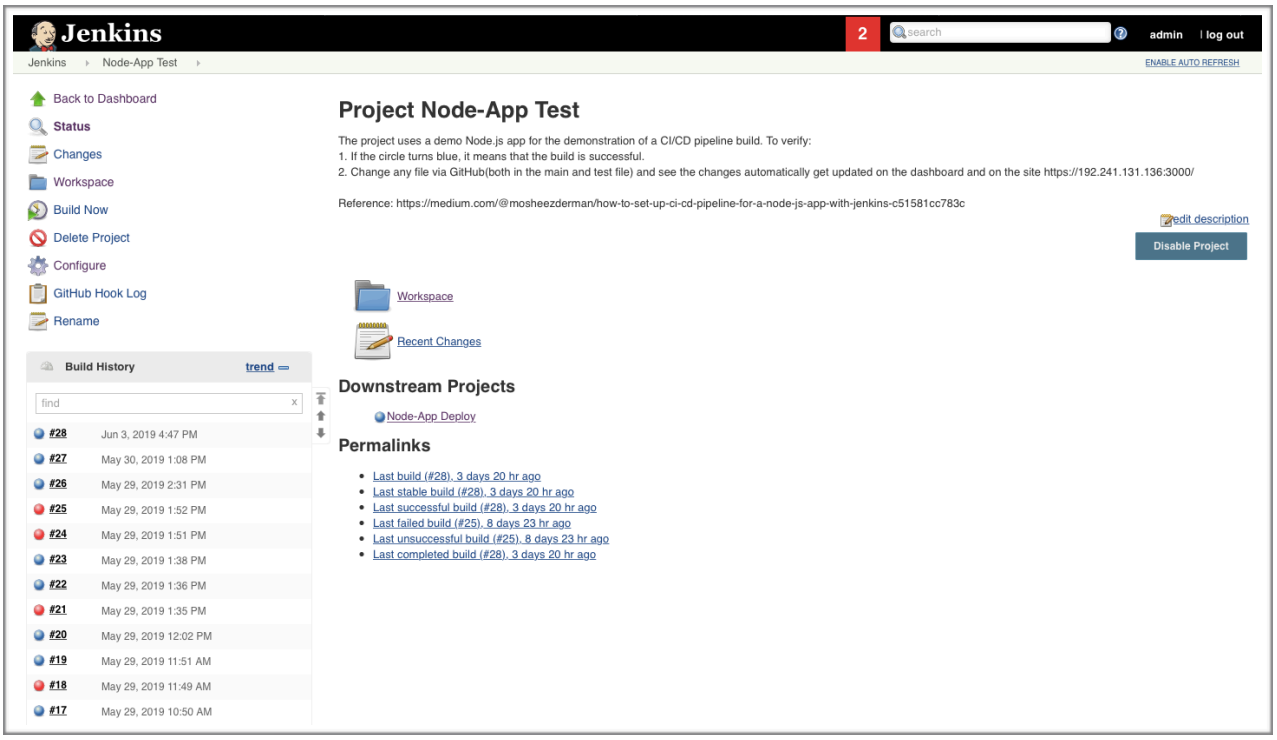
The SMTP server for your host can be found on this link:
https://www.arclab.com/en/kb/email/list-of-smtp-and-pop3-servers-mailserver-list.html

*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.

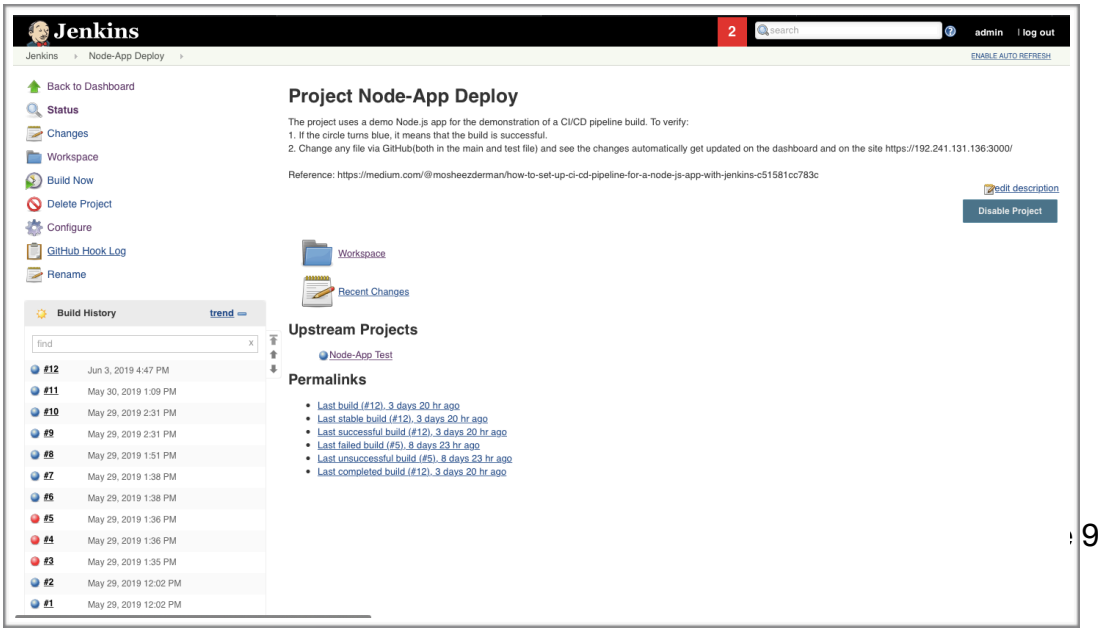
14. A new project will look like this. Build the projects to see the output.



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.

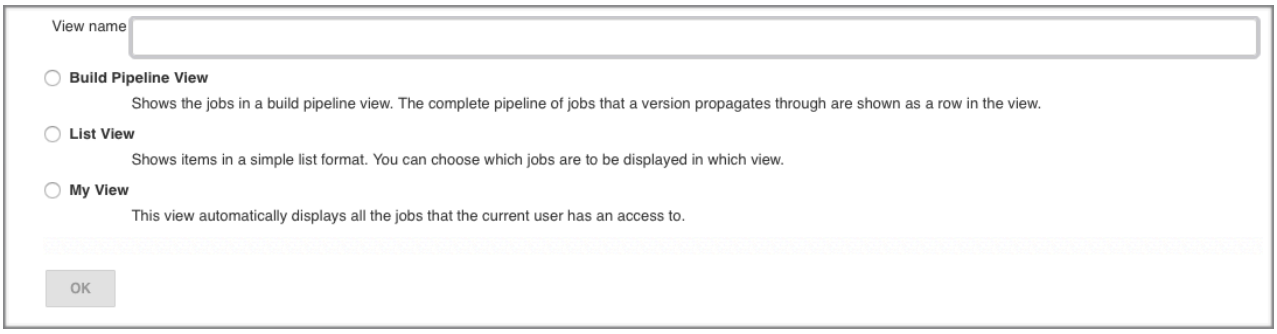
15. Create a new job for deployment something like this.



16.

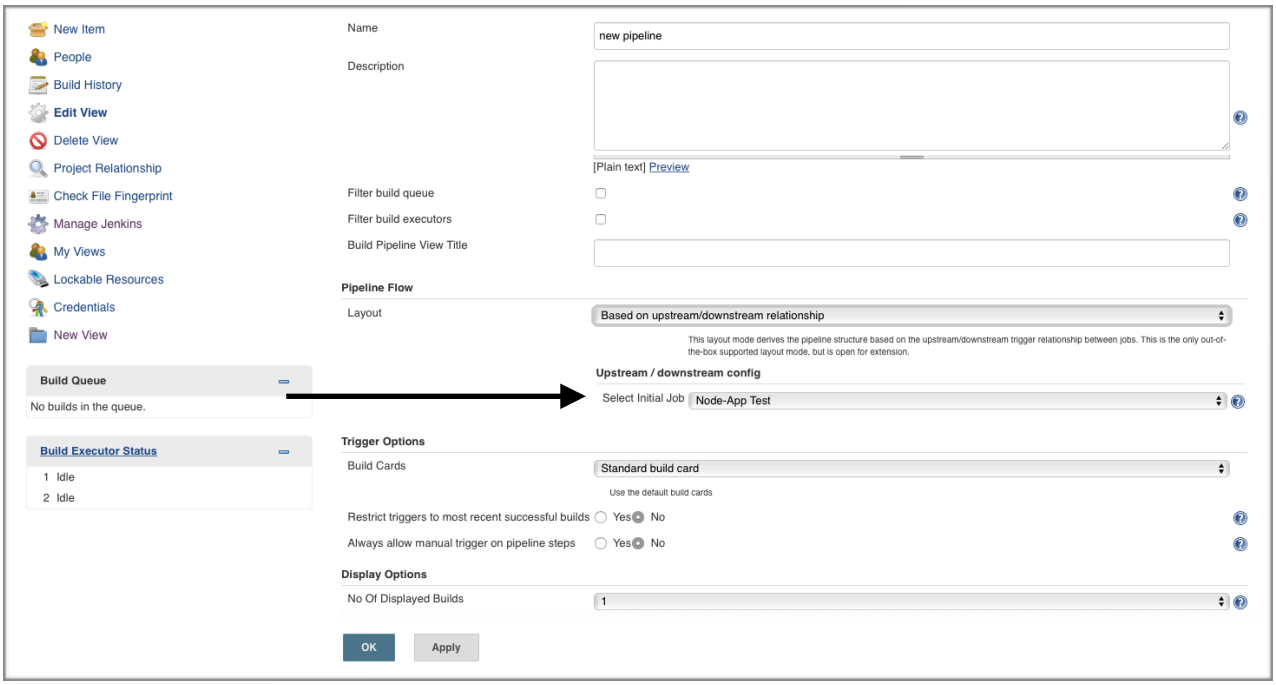
Ensure that 'Build Pipeline' plugin is installed and to Build a Pipeline, follow the below steps: Click on the '**+**' symbol.

17. Select 'Build Pipeline View'

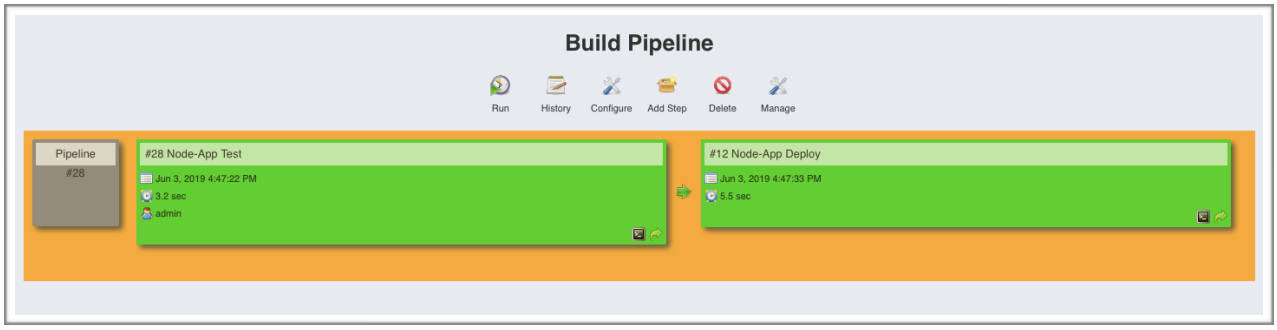


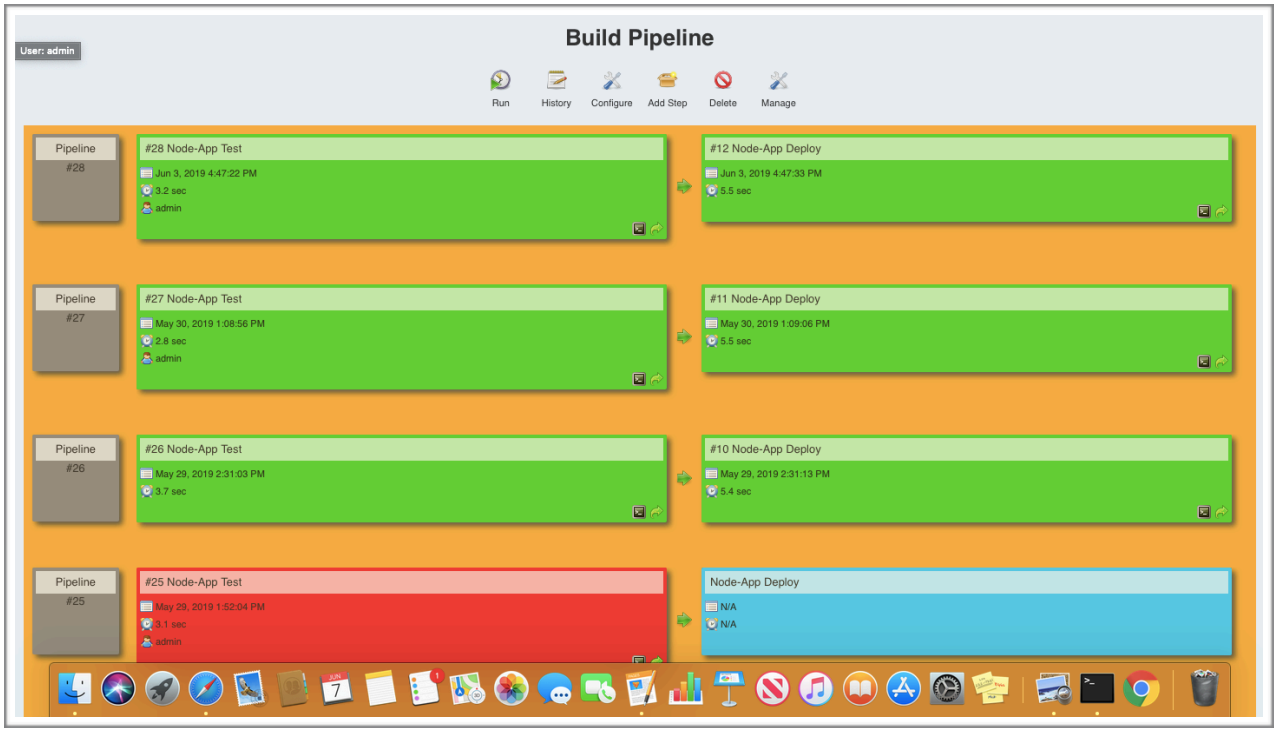
18. Give a new name and select **OK.**

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



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

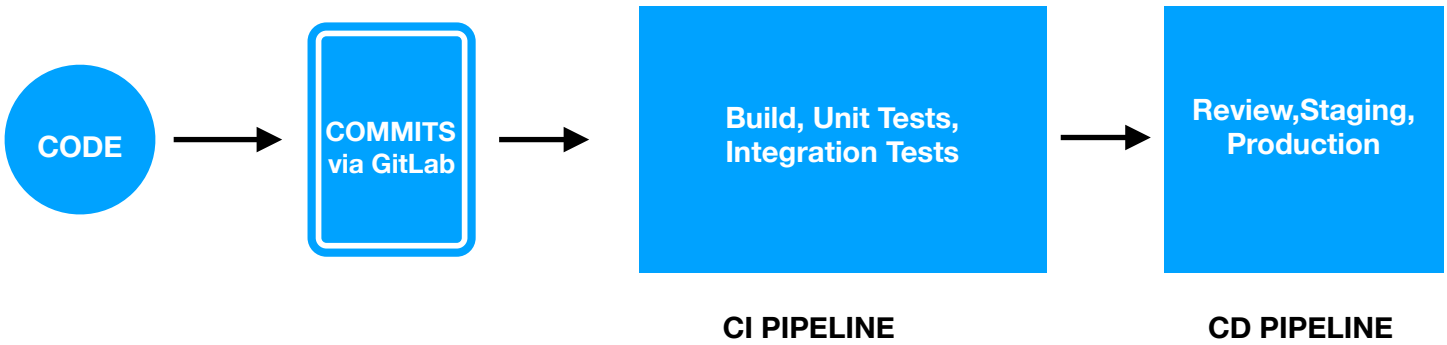
Save the public key here.

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:



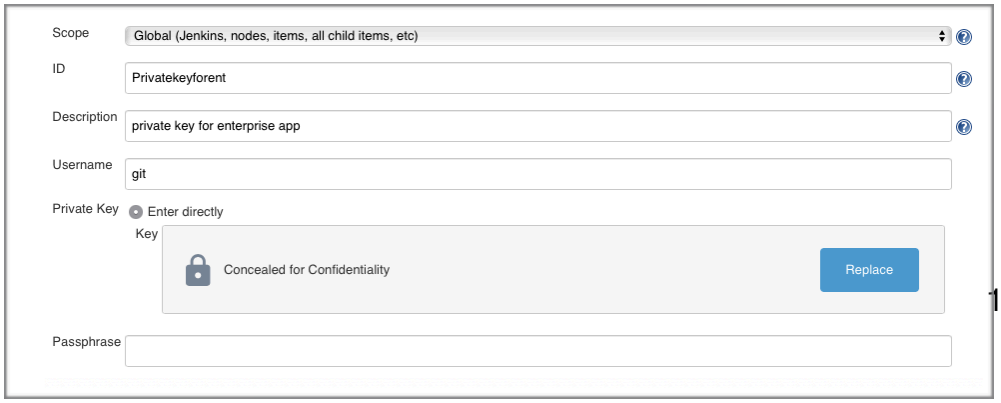
GitLab is a web-based **Git** repository. It focuses on managing software development projects and its files, and tracking their changes from time-to-time. The information is stored in a data structure called repository.

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.

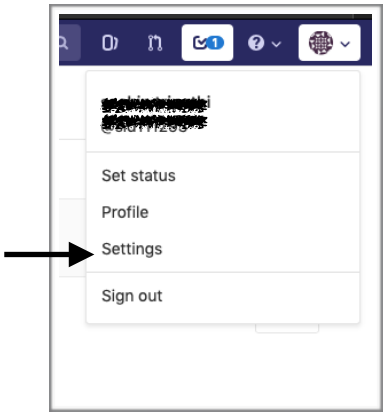
The steps required for setting up any project over GitLab and integrating it with Jenkins are as follows:

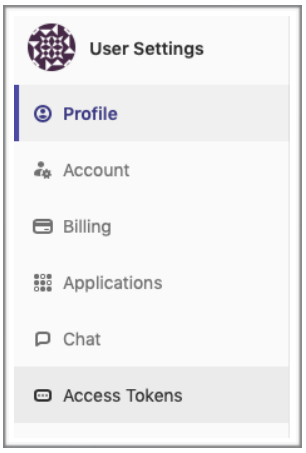
1. Set up the project over **GitLab** and configure SSH over GitLab and Jenkins. **Jenkins -> Credentials -> System -> Right click on Global credentials**



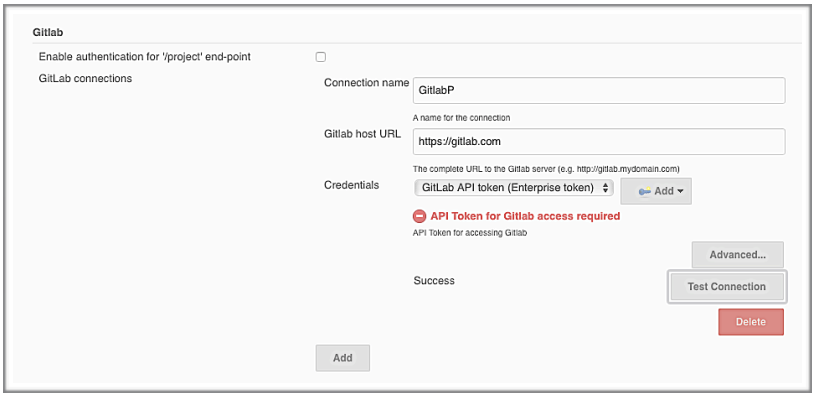
(unrestricted) -> Add Credentials

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

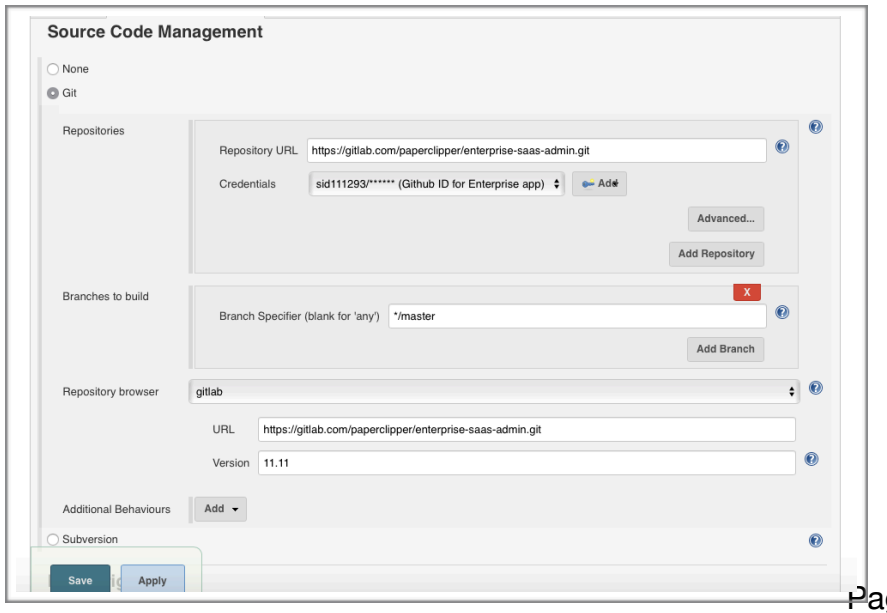
4. Create a new Freestyle job/item.

5. Configure the project.

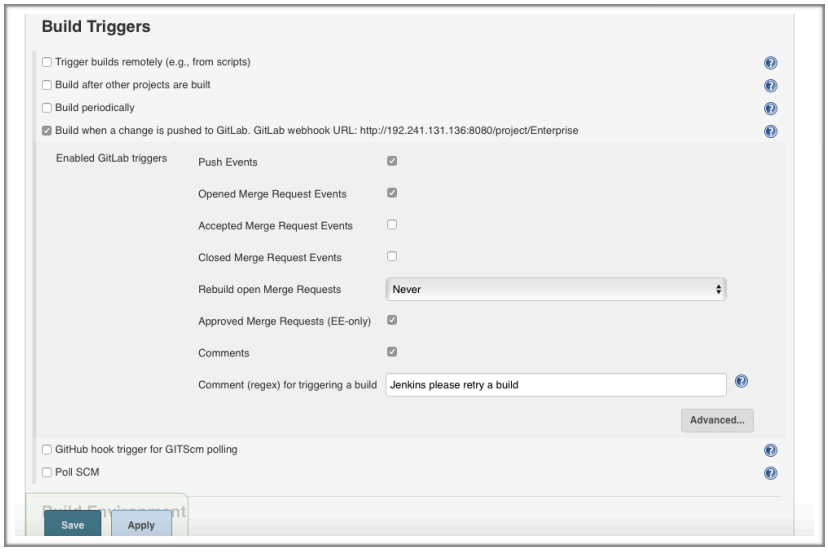
- Select **GitLab Connection**

- 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.



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



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.**

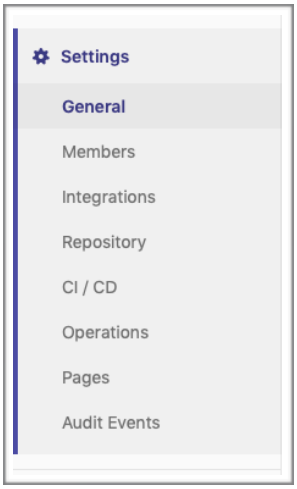
- Click Save

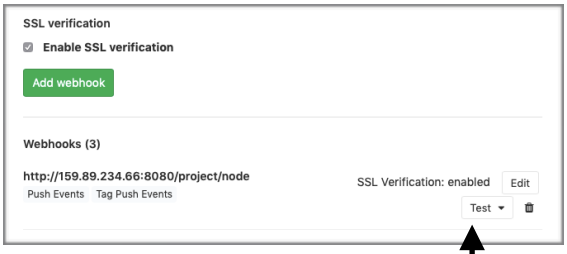
In the previous step, there is also a Webhook URL given for every project,
GitLab Webhooks are basically created to provide other applications with real-time information. A **Webhook** delivers data to other applications as it happens, meaning you get data immediately.

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

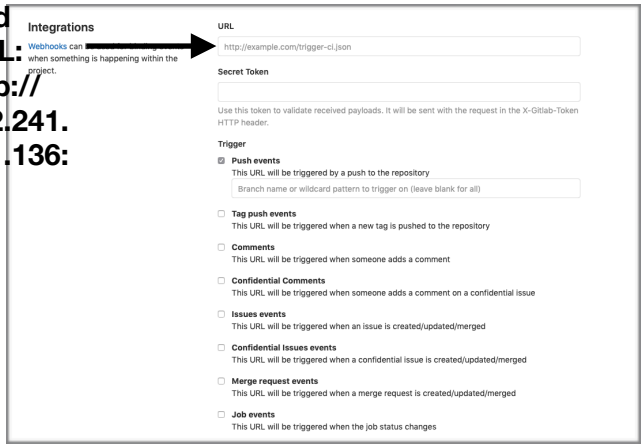
If it gives back an **HTTP 200** response then the Webhook is setup.





Test the Webhook

Add URL: http://192.241.131.136:



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

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.

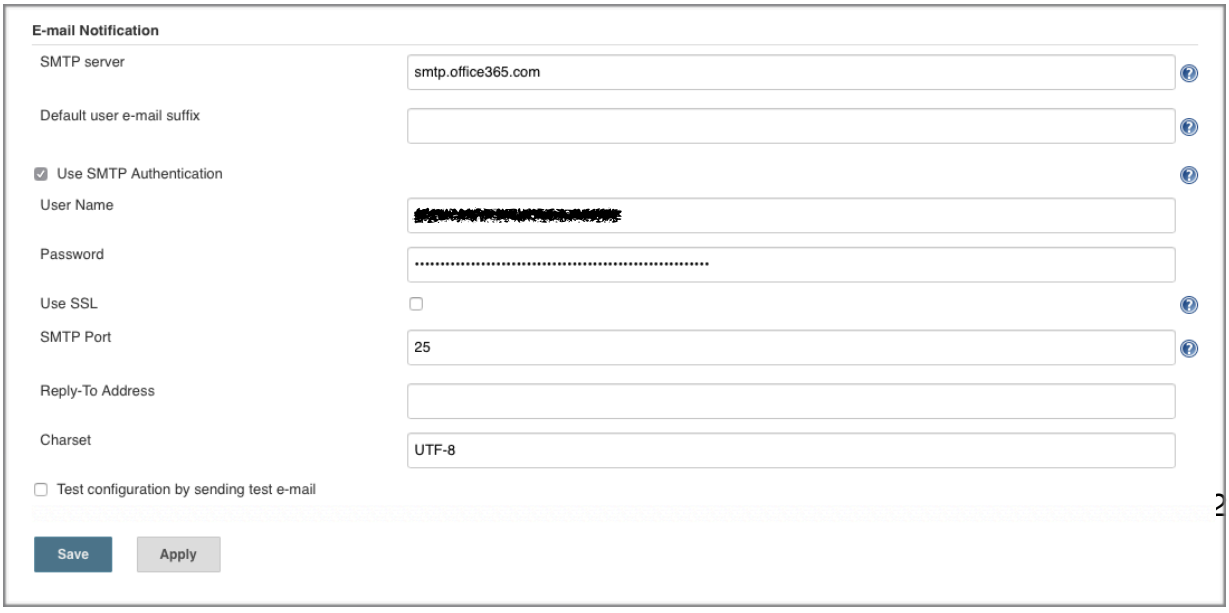
There is also a feature of adding **Email Notifications** about the status of Builds which is available as a Post-Build action.

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

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.

SETUP COMPLETE!

Test the build by making any change via GitLab.
It should show SUCCESS.

GITLAB CODE REVIEW PROCESS:

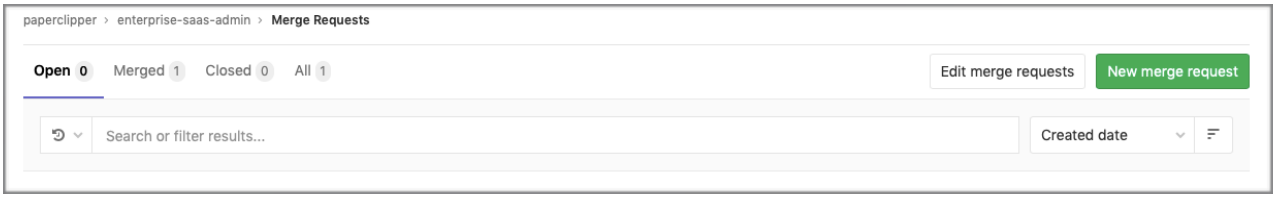
Typical flow of the Code Review Process:

- 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.

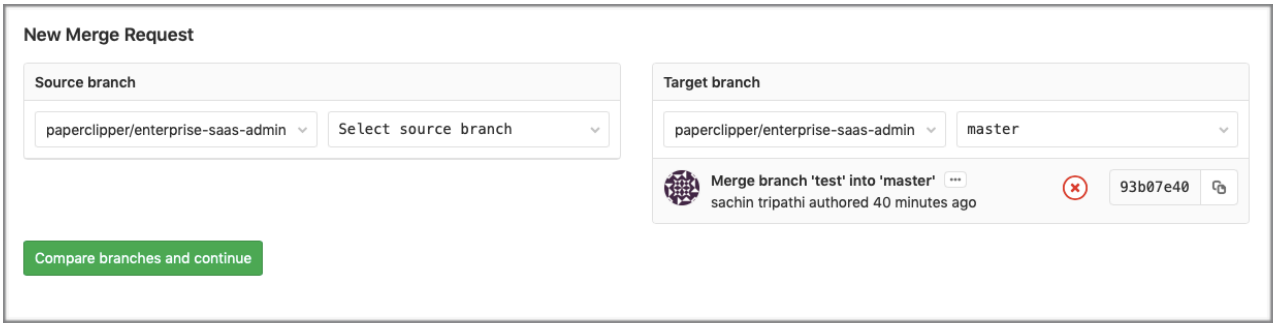
An example is presented below for reference:

1. Select **Merge Requests** from the project tab.

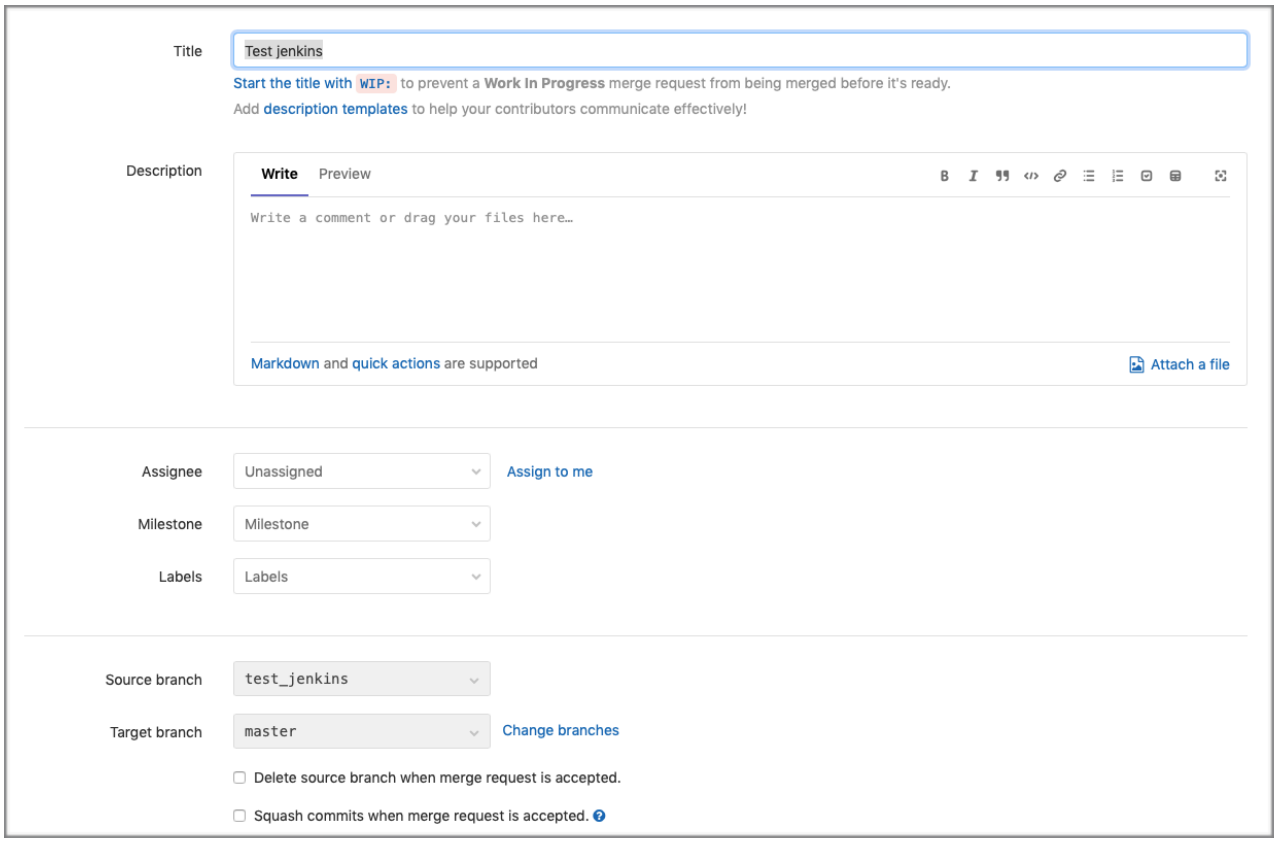
2. Click on **New Merge Request**.



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



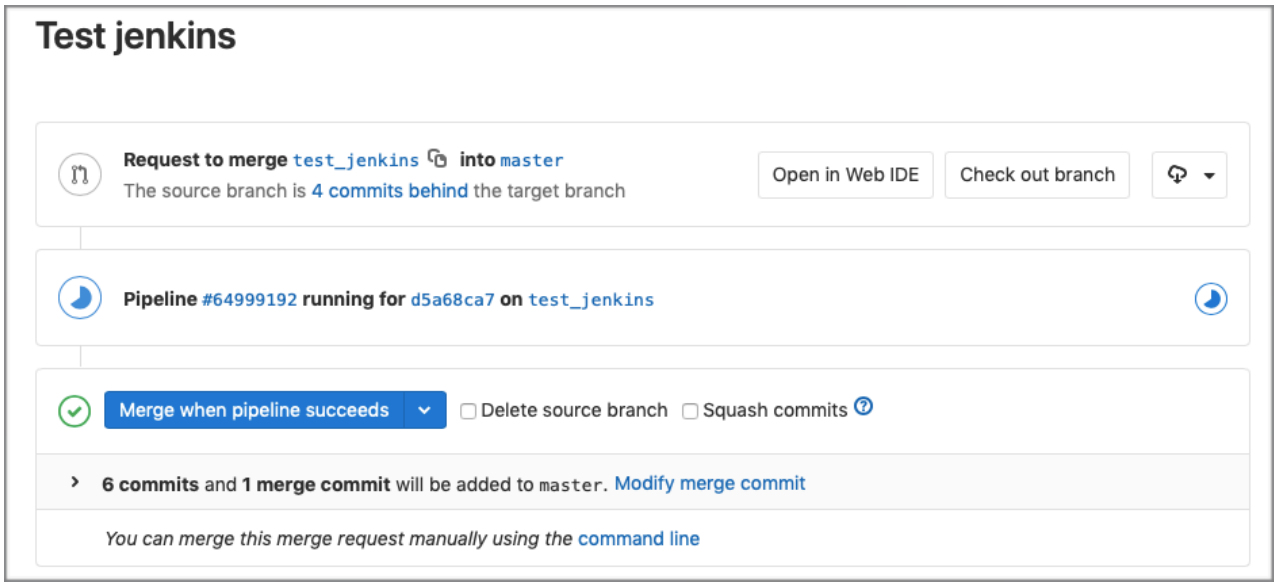
4. Click on **Compare branches and continue**.



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.



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