J. RAVIVARMAN

DEVOPS CAPSTONE PROJECT – 1

TASK TO BE PERFORMED:

You have been hired as a Sr. DevOps Engineer in Abode Software. They want to implement DevOps Lifecycle in their company. You have been asked to implement this lifecycle as fast as possible. Abode Software is a product-based company and their product is available on this GitHub link - https://github.com/hshar/website.git

Following are the specifications of the lifecycle:

1. Install the necessary software on the machines using a configuration management tool

2. Git workflow must be implemented

3. Code Build should automatically be triggered once a commit is made to master branch or develop branch.

- If a commit is made to master branch, test and push to prod
- If a commit is made to develop branch, just test the product, do not push to prod

4. The code should be containerized with the help of a Dockerfile. The Dockerfile should be built every time there is a push to GitHub. Use the following pre-built container for your application: hshar/webapp

The code should reside in '/var/www/html'

5. The above tasks should be defined in a Jenkins Pipeline with the following jobs:

- Job1: build
- Job2: test
- Job3: prod

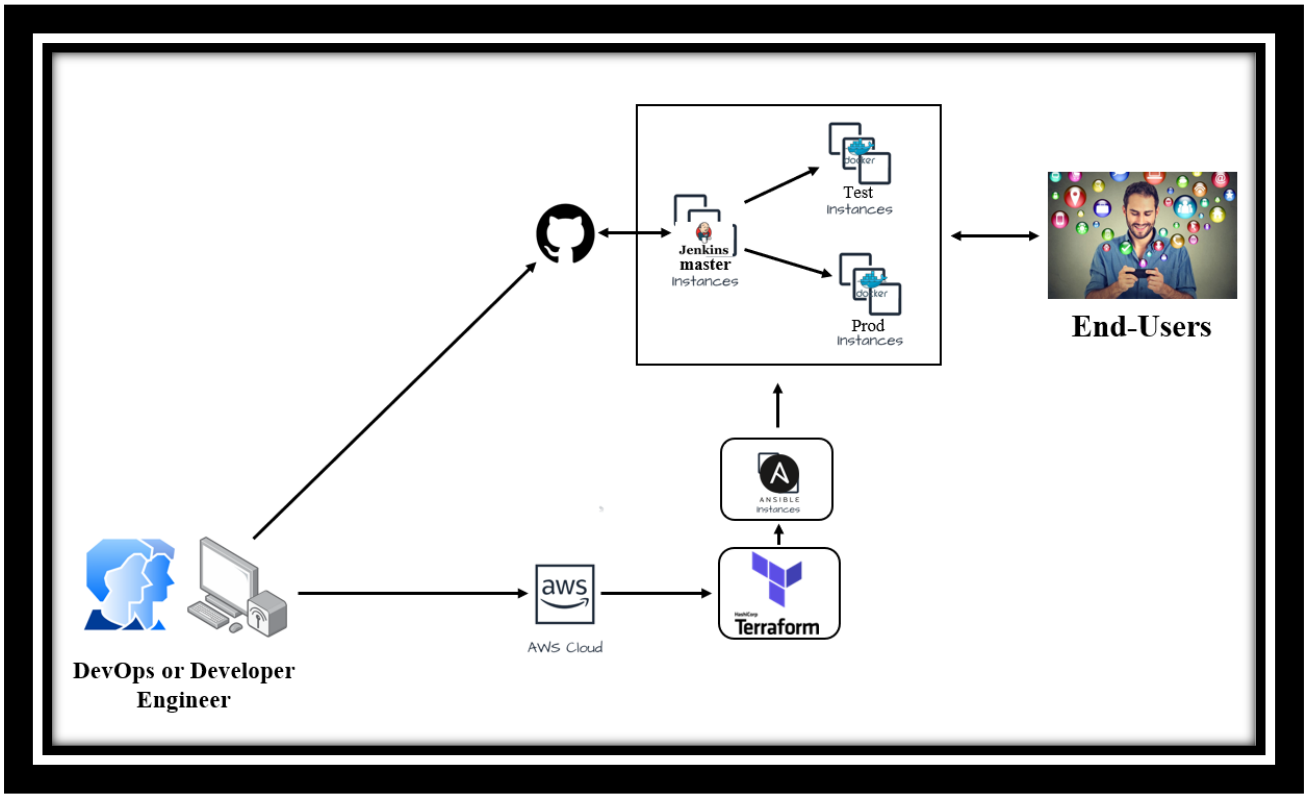
Requirements:

1. No of instances:

- Terraform master
- Ansible master
- Jenkins master
- Jenkins test server
- Jenkins prod server

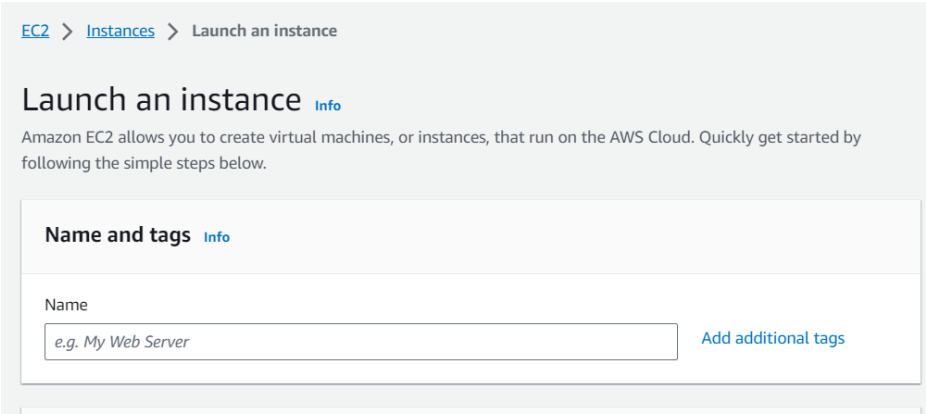
2. Operating-System: Ubuntu 20.04

SOLUTION:

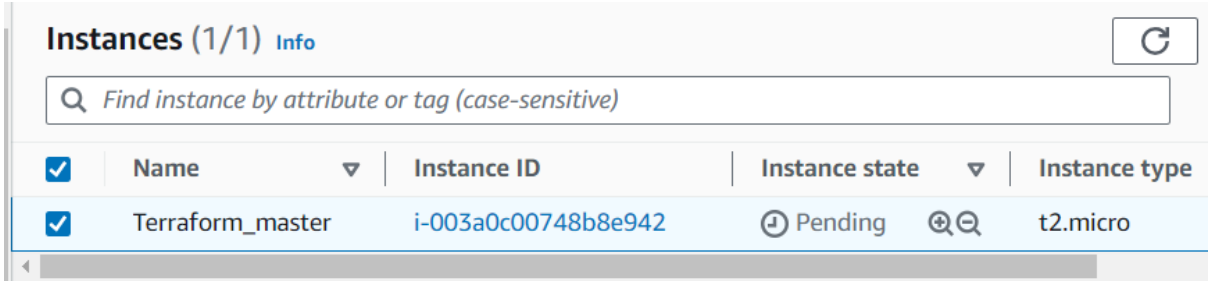


Step-1: Creating a **terraform master** instance:

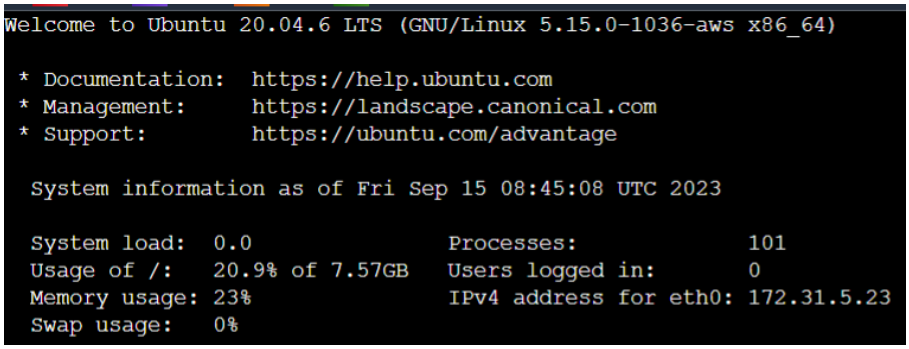
- Login into AWS management console and going EC2 management console for launching terraform master instance.

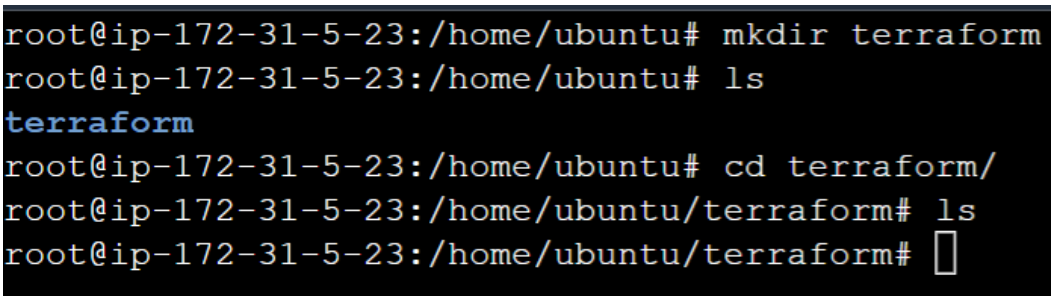


- Here the instance specifications according to our spec needs. I am selecting **os=Ubuntu 20.04 LTS**, **instance type= t2.micro**.



- The instance has been created, now we need to connect the instance and download terraform package on it in order terraform to provision our resources. For installing terraform I am creating **a script file**, in that script file that will contain the script to download and install the terraform on our instance and **changing the file permission** of the file for execution purpose. Before the execution of the file making the directory as **terraform** for our working purpose.





```
root@ip-172-31-5-23:/home/ubuntu/terraform# vi terraform.sh
root@ip-172-31-5-23:/home/ubuntu/terraform# chmod 744 terraform.sh
root@ip-172-31-5-23:/home/ubuntu/terraform#
```

- After executing the file. Checking the **terraform version.**

```
root@ip-172-31-5-23:/home/ubuntu/terraform# terraform --version
Terraform v1.5.7
on linux_amd64
root@ip-172-31-5-23:/home/ubuntu/terraform#
```

- Creating a **IAM user** for terraform to access our cloud. Here I am giving 2 permissions – **EC2 full access and VPC full access.**

User details

User name	Console password type	Require password reset
Terraform_user	None	No

Permissions summary

Name	Type	Used as
AmazonEC2FullAccess	AWS managed	Permissions policy
AmazonVPCFullAccess	AWS managed	Permissions policy

- Creating an **access_key and secret_key** credentials for terraform user for security reasons

Retrieve access keys Info

Access key

If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key	Secret access key
AKIAZANDQIRABD4RV2WB	********** Show

- Creating the variables.tf file. Which contains all the variables details.

```
root@ip-172-31-5-23:/home/ubuntu/terraform# vi variable.tf
root@ip-172-31-5-23:/home/ubuntu/terraform# cat variable.tf
variable "reg" {
    description = "The region for creating the resources"
    default = "ap-south-1"
}

variable "ak" {
    description = "Access key of the IAM User"
    default = "AKIAZANDQIRABD4RV2WB"
}

variable "sk" {
    description = "Secret key of the IAM User"
    default = "n5Kn6O9CV3YtaG68CiUaEZk+WmMVAKJQH2SBYAYm"
}

variable "ami_id" {
    description = "Ami_id of the operating system"
    default = "ami-08e5424edfe926b43"
}

variable "keypair" {
    description = "Keypair that should be associated with the instance"
    default = "console_server"
}
root@ip-172-31-5-23:/home/ubuntu/terraform#
```

- Now creating the main terraform file. Which contains all the details for provisioning the infrastructure. Which will create the **VPC** and required components for vpc to assist and **4 instances** will be created. 1for **ansible master** and another one **Jenkins master** and another two instances for **test and prod**.

```
root@ip-172-31-5-23:/home/ubuntu/terraform# touch main.tf
root@ip-172-31-5-23:/home/ubuntu/terraform# ls
main.tf  variable.tf
root@ip-172-31-5-23:/home/ubuntu/terraform#
```

- Now we need to perform **terraform init** command for terraform to download the dependencies required to support the provision for our requirement.

```
root@ip-172-31-5-23:/home/ubuntu/terraform# terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.17.0...
- Installed hashicorp/aws v5.17.0 (signed by HashiCorp)
```

- After that we need to format the terraform files with the help of the command – **terraform fmt**

```
root@ip-172-31-5-23:/home/ubuntu/terraform# terraform fmt
main.tf
variable.tf
root@ip-172-31-5-23:/home/ubuntu/terraform#
```

- Then we need to validate the credentials by using the command – **terraform validate**

```
root@ip-172-31-5-23:/home/ubuntu/terraform# terraform validate
Success! The configuration is valid.

root@ip-172-31-5-23:/home/ubuntu/terraform#
```

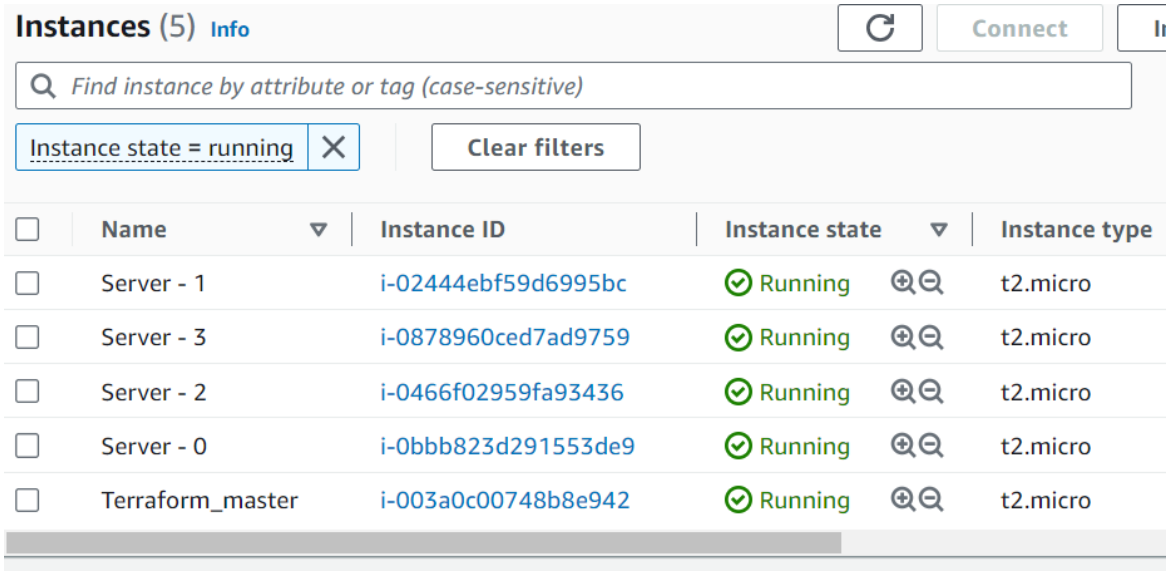
- The credentials have been valid now we need to initiate terraform to plan how to provision the infrastructure requirements.

```
root@ip-172-31-5-23:/home/ubuntu/terraform# terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
  + create

Terraform will perform the following actions:
```

- Finally, we need to perform **terraform apply -auto-approve** command for terraform to provision the infrastructure.

```
Plan: 11 to add, 0 to change, 0 to destroy.
aws_vpc.project1vpc: Creating...
aws_vpc.project1vpc: Creation complete after 1s [id=vpc-06ad6372ff21b6483]
aws_subnet.project1_subnet: Creating...
aws_route_table.project1_route_table: Creating...
aws_security_group.project1_sc: Creating...
aws_internet_gateway.project1_igw: Creating...
aws_internet_gateway.project1_igw: Creation complete after 0s [id=igw-0be72fb3a472a880c]
aws_route_table.project1_route_table: Creation complete after 0s [id=rtb-0b27ba771aa104150]
aws_route.project1_routing: Creating...
aws_subnet.project1_subnet: Creation complete after 0s [id=subnet-081b53993cada8a73]
aws_route_table_association.subnet_association: Creating...
aws_route_table_association.subnet_association: Creation complete after 1s [id=rtbassoc-09395a79da5190808]
aws_route.project1_routing: Creation complete after 1s [id=r-rtb-0b27ba771aa1041501080289494]
aws_security_group.project1_sc: Creation complete after 2s [id=sg-09f48c771648e21c0]
aws_instance.server[0]: Creating...
aws_instance.server[2]: Creating...
aws_instance.server[1]: Creating...
aws_instance.server[3]: Creating...
```

- Resources had been provisioned by terraform now we need to check it on the console.



Instances (5) Info

Connect

Find instance by attribute or tag (case-sensitive)

Instance state = running

Clear filters

Name	Instance ID	Instance state	Instance type
Server - 1	i-02444ebf59d6995bc	Running	t2.micro
Server - 3	i-0878960ced7ad9759	Running	t2.micro
Server - 2	i-0466f02959fa93436	Running	t2.micro
Server - 0	i-0bbb823d291553de9	Running	t2.micro
Terraform_master	i-003a0c00748b8e942	Running	t2.micro

Resources has been provisioned successfully.

- Now we need to rename the resources the instances according to our specifications like ansible master, Jenkins master, Jenkins test, Jenkins prod.
- Connect the all instances by EC2 instance connect for ansible cluster formation to install the necessary software on each instances.
- Other than ansible master nodes we need to make a small change on other servers for ssh connection in order ansible to install the packages.
- For that we need to set up the password for the root user, modify the slight changes on **vi /etc/ssh/sshd_config**
- Once making the changes restart the sshd service with **systemctl restart sshd**
- We need to install ansible on the ansible master instance. For that I am creating the script file with the script to install ansible.

Script file contains:

```
#!/bin/bash
apt-get update
apt-get upgrade -y
sudo apt-add-repository ppa:ansible/ansible
apt-get update
apt-get install -y ansible
```

```
root@ip-10-0-1-102:/home/ubuntu# vi ansible.sh
root@ip-10-0-1-102:/home/ubuntu# chmod 744 ansible.sh
root@ip-10-0-1-102:/home/ubuntu# ./ansible.sh
```

- Now we need to check whether ansible is installed or not by checking the **ansible --version** command

```
root@ip-10-0-1-102:/home/ubuntu# ansible --version
ansible [core 2.12.10]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /root/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.8.10 (default, May 26 2023, 14:05:08) [GCC 9.4.0]
  jinja version = 2.10.1
  libyaml = True
root@ip-10-0-1-102:/home/ubuntu#
```

- Ansible has been successfully installed. Now we need configure the host list on the ansible host file, for installing the packages on node servers like Jenkins master, Jenkins test, Jenkins prod.

```
[jenkins_master]
10.0.1.37

[test_server]
10.0.1.204

[prod_server]
10.0.1.16
```

Here the ip address used are the private ip address because the private ip address are static ip address that will not be changed even we stop the instance.

- Now we need to generate the key for our ansible master in order to connect with other nodes, by using **ssh-keygen** command.

```
root@ip-10-0-1-102:/home/ubuntu# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:jQb4zStFqQo2VCJAjy0+VEdOHB9Swdsu7FQQVmz9YVs root@ip-10-0-1-102
The key's randomart image is:
+---[RSA 3072]----+
|=...+*+*=..      |
| .=o+o+o.+ . o E |
| +.o....B   o +  |
|o..  . B =   o   |
| o+   + S .      |
| ..o . * o       |
|    . + o        |
|       o         |
|                 |
+----[SHA256]-----+
root@ip-10-0-1-102:/home/ubuntu#
```

- Once generating key, we need to copy the key to each instance in order to connect with ansible master by using **ssh-copy-id <user>@<ip_address>**
- Once the key is copied check the connectivity by ansible ping command like **ansible all -m ping**

```
root@ip-10-0-1-102:/home/ubuntu# ansible all -m ping
10.0.1.37 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
10.0.1.204 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
10.0.1.16 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
root@ip-10-0-1-102:/home/ubuntu#
```

- Connectivity has been successfully established between slave nodes. Now we need to create the **ansible playbook** to install necessary software on each node for the project requirements.

  Jenkins_master = openjdk-11-jre, Jenkins
  Jenkins_test = java, docker
  Jenkins_prod = java, docker

- Script files has been created to for Jenkins master to install the necessary packages. As test and prod server also.

Yaml file for execute the ansible playbook contains:

```
---
- name: To_install_jenkins_on_jenkins_master
  hosts: jenkins_master
  become: true
  tasks:
    - name: To install packages
```

```
      script: jenkins.sh

    - name: To notify the installation
      debug:
        msg: "Package has been installed"

- name: To install docker and java on test server
  hosts: test_server
  become: true
  tasks:
    - name: To install Java and Docker
      script: test.sh

    - name: To notify about the installation!
      debug:
        msg: "Java and Docker has been installed!"

- name: To install docker and java on prod server
  hosts: prod_server
  become: true
  tasks:
    - name: To install Java and Docker
      script: prod.sh

    - name: To notify about the installation!
      debug:
        msg: "Java and Docker has been installed!"
```

- Now check the syntax of yml file with the command – **ansible-playbook package.yml --syntax-check**

```
root@ip-10-0-1-102:/home/ubuntu#
root@ip-10-0-1-102:/home/ubuntu# ls
ansible.sh  jenkins.sh  package.yml  prod.sh  test.sh
root@ip-10-0-1-102:/home/ubuntu# vi package.yml
```

```
root@ip-10-0-1-102:/home/ubuntu# ansible-playbook package.yml --syntax-check

playbook: package.yml
```

- The syntax of the yml file has been verified by ansible-playbook.
- Now we need to execute this file for ansible to install the necessary packages on slave nodes – **ansible-playbook package.yml**

```
root@ip-10-0-1-102:/home/ubuntu# ansible-playbook package.yml

PLAY [To_install_jenkins_on_jenkins_master] ******************

TASK [Gathering Facts] ****************************************
ok: [10.0.1.37]

TASK [To install packages] ************************************
changed: [10.0.1.37]

TASK [To notify the installation] *****************************
ok: [10.0.1.37] => {
    "msg": "Package has been installed"
}
```

- On Jenkins master ansible has done its job. Now we need to check it on Jenkins master instance.

```
root@ip-10-0-1-37:/home/ubuntu# java --version
openjdk 11.0.20.1 2023-08-24
OpenJDK Runtime Environment (build 11.0.20.1+1-post-Ubuntu-0ubuntu120.04)
OpenJDK 64-Bit Server VM (build 11.0.20.1+1-post-Ubuntu-0ubuntu120.04, mixed mode, sharing)
root@ip-10-0-1-37:/home/ubuntu# jenkins --version
2.414.1
root@ip-10-0-1-37:/home/ubuntu#
```

- Packages has been installed successfully on **Jenkins master server**.

```
PLAY [To install docker and java on test server]

TASK [Gathering Facts] ***************************
ok: [10.0.1.204]

TASK [To install Java and Docker] ****************
changed: [10.0.1.204]

TASK [To notify about the installation!] *********
ok: [10.0.1.204] => {
    "msg": "Java and Docker has been installed!"
}
```

- On test server ansible has done its job. Now we need check manually.

```
root@ip-10-0-1-204:/home/ubuntu# java --version
openjdk 11.0.20.1 2023-08-24
OpenJDK Runtime Environment (build 11.0.20.1+1-post-Ubuntu-0ubuntu120.04)
OpenJDK 64-Bit Server VM (build 11.0.20.1+1-post-Ubuntu-0ubuntu120.04, mixed mode, sharing)
root@ip-10-0-1-204:/home/ubuntu# docker -v
Docker version 24.0.5, build 24.0.5-0ubuntu1~20.04.1
root@ip-10-0-1-204:/home/ubuntu# systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2023-09-16 03:17:03 UTC; 6min ago
```

- Docker and java have been installed successfully on **test server**.

```
PLAY [To install docker and java on prod server]

TASK [Gathering Facts] *************************
ok: [10.0.1.16]

TASK [To install Java and Docker] ***************
changed: [10.0.1.16]

TASK [To notify about the installation!] ********
ok: [10.0.1.16] => {
    "msg": "Java and Docker has been installed!"
}
```

- On prod server ansible has done its job. Now we need check manually.

```
root@ip-10-0-1-16:/home/ubuntu# java --version; docker -v; systemctl status docker
openjdk 11.0.20.1 2023-08-24
OpenJDK Runtime Environment (build 11.0.20.1+1-post-Ubuntu-0ubuntu120.04)
OpenJDK 64-Bit Server VM (build 11.0.20.1+1-post-Ubuntu-0ubuntu120.04, mixed mode, sharing)
Docker version 24.0.5, build 24.0.5-0ubuntu1~20.04.1
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2023-09-16 03:18:02 UTC; 8min ago
```

- Docker and java have been installed successfully on **prod server**.

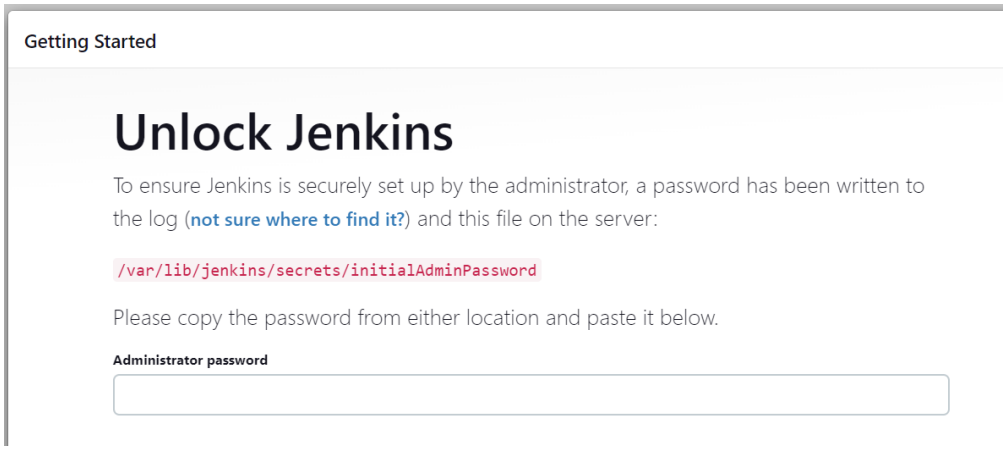
So far, we have installed the necessary packages for this project with the help of **Terraform and Ansible**.

****

Now we need to setup the **Jenkins Master-Slave connection** between Jenkins master and test, prod servers.

<u>Jenkins Master-Slave connection:</u>

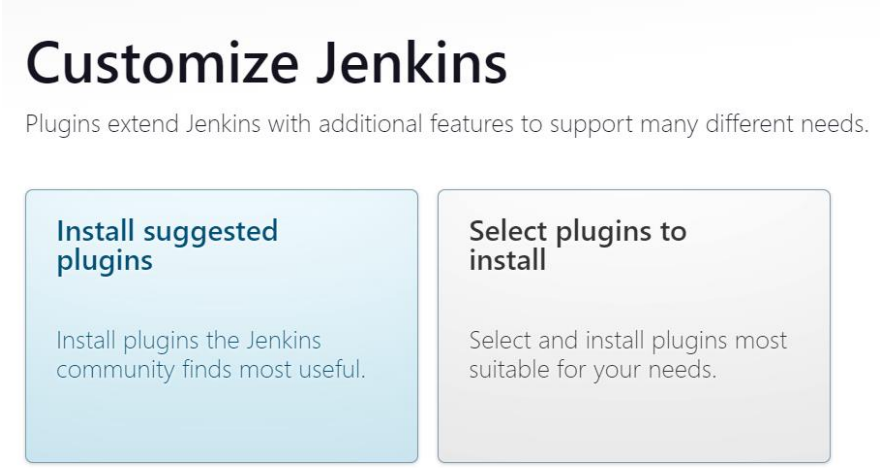
1. we need to setup the Jenkins master:



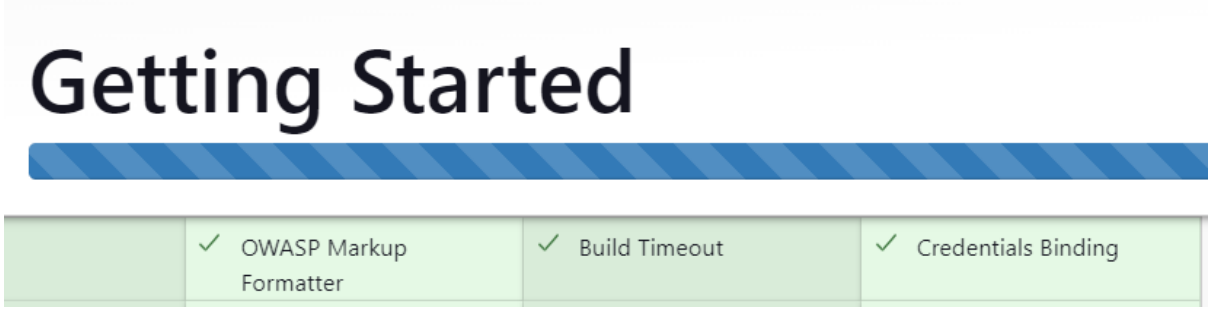
- Take the password by placing the below command on terminal of Jenkins server

```
root@ip-10-0-1-37:/var# cat /var/lib/jenkins/secrets/initialAdminPassword
b1223fa623004724a59295e660a21937
root@ip-10-0-1-37:/var#
```

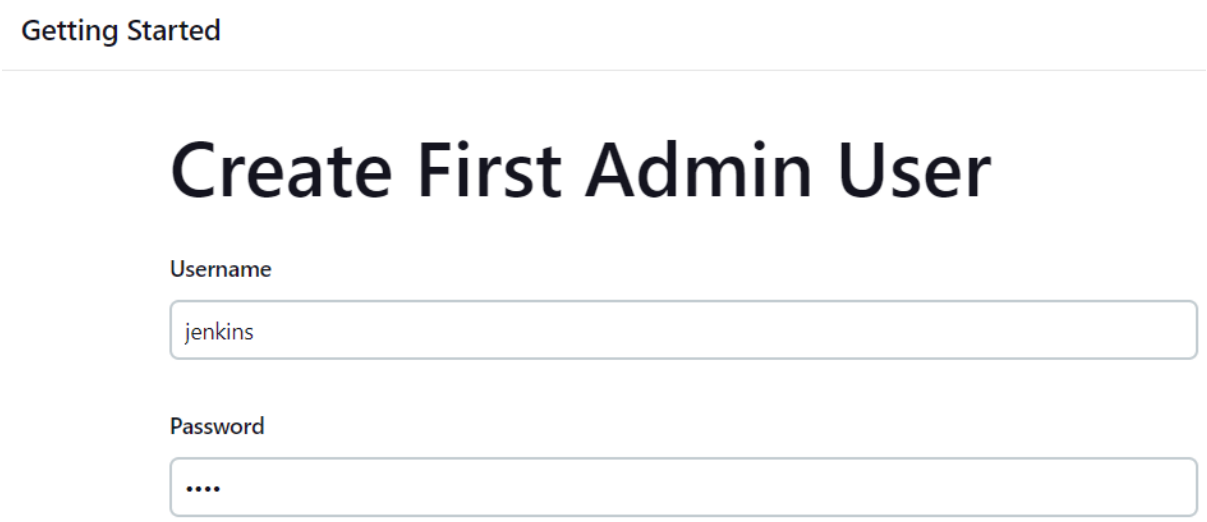
Place the password and proceed further.



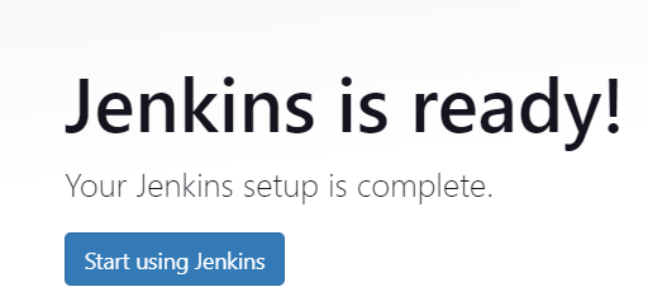
- Here selects the install suggested plugins option. It will install the necessary plugins.



- Once the plugins installed next it will ask to fill the sign in details:

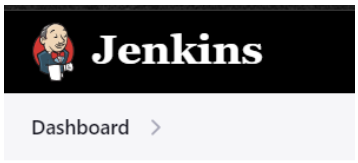


- Give the details and proceed further.

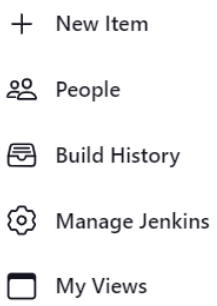


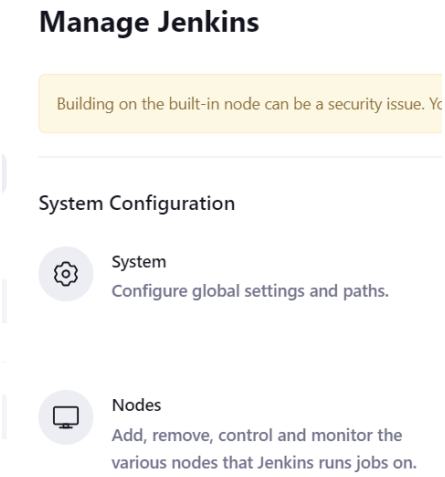
- Now the **Jenkins master** is ready.

<u>Master-slave setup:</u>



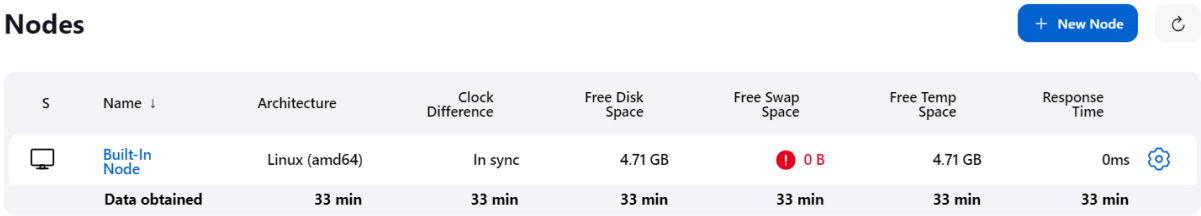
- On the right-hand side, we can able to see **manage Jenkins** option, click that.

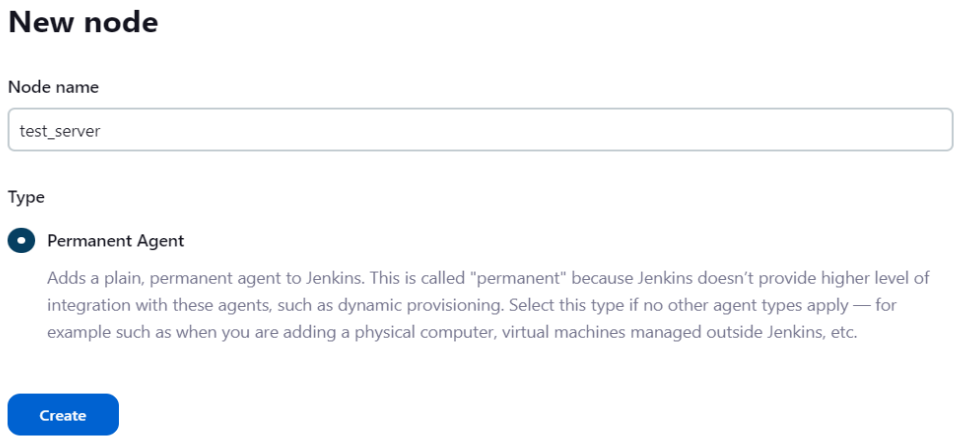




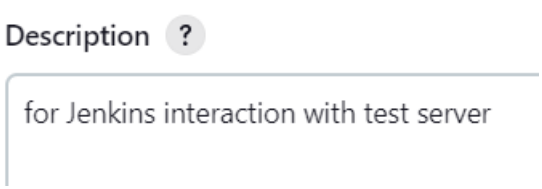
On the **manage Jenkins** under **system configuration**, we can able to see the **nodes**, click that.

Then click **new node** option on the right side





- Giving the node name as the **test_server** selecting permanent agent type, click create.



give the description according to project.

Remote root directory ?

/home/ubuntu/jenkins

give the remote root directory of the node.

Labels ?

test_server

Launch method ?

Launch agents via SSH

under launch method select the – **launch agents via SSH.**

And under the **host** give the Jenkins test server **private_ip_address.**

Host ?

10.0.1.204

Add Credentials

Domain

Global credentials (unrestricted)

Kind

Username with password

then we need to add the credentials for securing the test node server agent. Here I selected the **username with password** kind.

I am giving my user details and finally adding it.

Username ?

root

Treat username as secret ?

Password ?

••••

ID ?

for-test_server

Description ?

test_server_user

Add Cancel

refresh it, select the credentials details.

Credentials ?

root/****** (test_server_user)

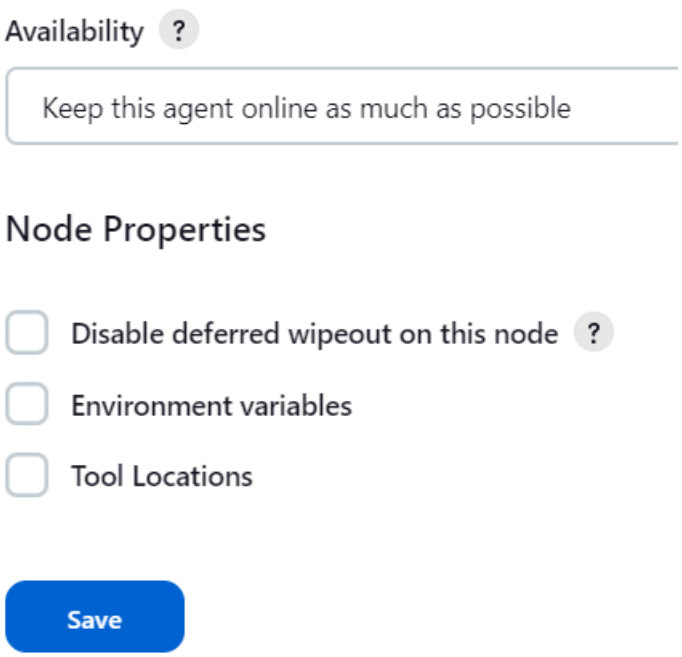
Add

under the host verification strategy – select the non-verifying verification strategy.

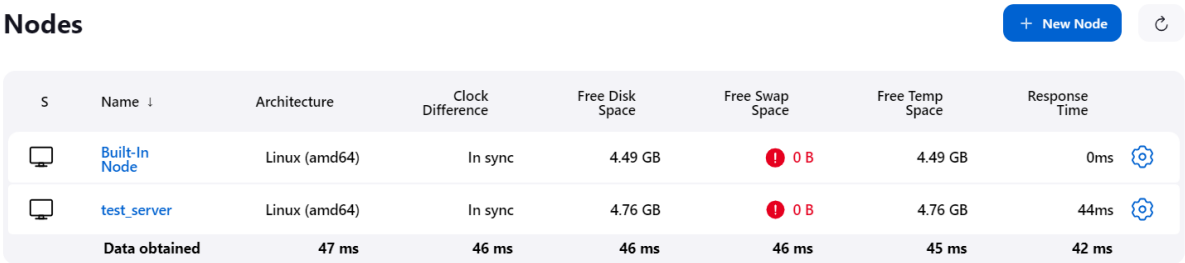
Host Key Verification Strategy ?

Non verifying Verification Strategy

Advanced

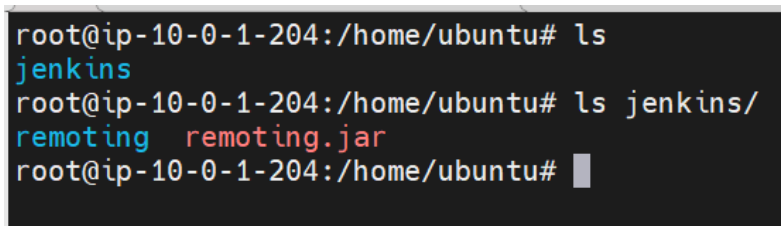


then keep of the rest of the setting default and save it.



- We can able to see the test_server node has been created successfully.
- **How can able to say that the node is created or added successfully?**

we can able to see the **check difference represents in sync**. And we can confirm manually, by checking on the test_server:



This confirms that the node is successfully added to this Jenkins master.

Like this we need to setup the prod node also.

Nodes

+ New Node

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	4.49 GB	0 B	4.49 GB	0ms
	prod_node	Linux (amd64)	In sync	4.75 GB	0 B	4.75 GB	20ms
	test_server	Linux (amd64)	In sync	4.76 GB	0 B	4.76 GB	14ms
	Data obtained	82 ms	79 ms	74 ms	71 ms	64 ms	61 ms

As the per the task requirements: **two nodes** were setup successfully – **test and prod.**

****

Now we need to create the Jenkins jobs according to the project:

<u>Requirements:</u>

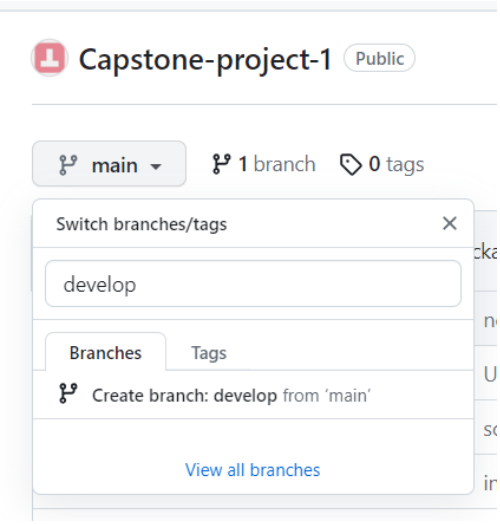
- If a commit is made to master branch, test and push to prod.
- If a commit is made to develop branch, just test the product, do not push to prod.
- The code should be containerized with the help of a Dockerfile. The Dockerfile should be built every time there is a push to GitHub

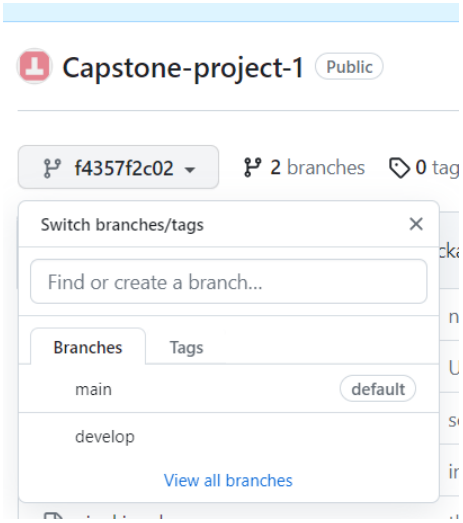
<u>SOLUTION:</u>

- Before creating the jobs in Jenkins, first I need to create the **develop branch** and setup the **GitHub-webhook trigger for the repository**.

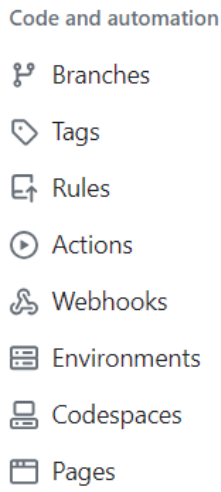
GitHub repository: https://github.com/Ravivarman16/Capstone-project-1.git

- Creating a new branch – develop as per the task.



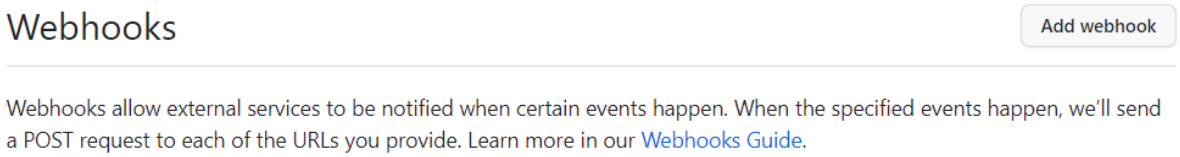


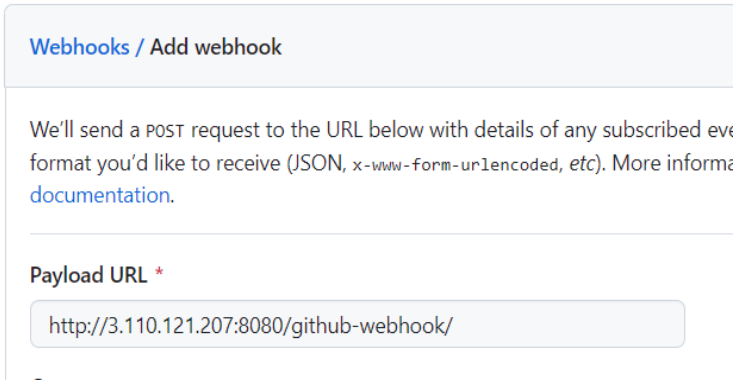
- The branch has created. Now we need to setup the **hook trigger from GitHub to Jenkins master.**
- Under the **settings of the repository**, if we click it on the right-hand side, we can able to see **webhook.**



- Select the webhook option.

- After clicking we can able to see the **add webhook** option click that.





- Under the **payload URL**, we need to give the **Jenkins URL** with the option named **github-webhook** to trigger automatically when ever some one pushes to the repository, it will immediately take the source to Jenkins and run the desired job specified to it.

Secret

Under secret I am giving my github secret.

Which events would you like to trigger this webhook?

- Just the push event.
- Send me **everything**.
- Let me select individual events.

- After giving the secret, then what kinds notifications to be triggered to Jenkins. Here I am selecting **send me everything** option.

Active

We will deliver event details when this hook is triggered.

Add webhook

- Click **add webhook**.

- Webhook has been setup on github side but not on Jenkins side, now we need to setup on Jenkins side.

On Jenkins master dashboard:

System Configuration

System

Configure global settings and paths.

On the Jenkins master dashboard, click the manage Jenkins option. Under manage Jenkins option, click **system**.

GitHub

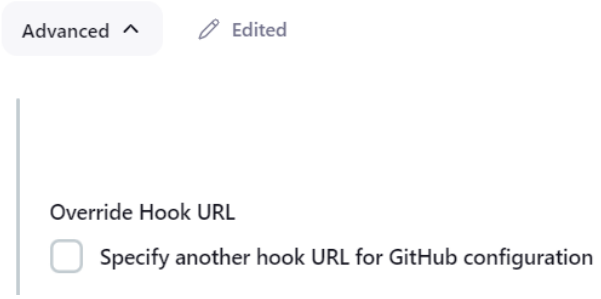
GitHub Servers ?

Add GitHub Server

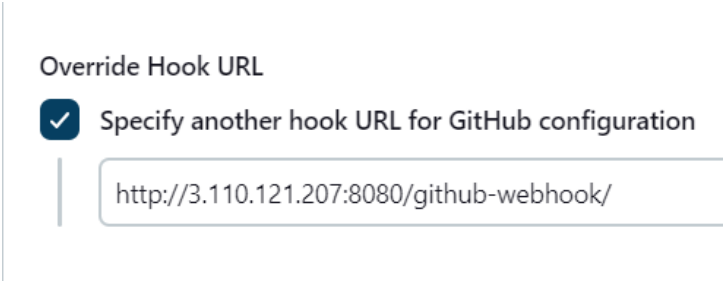
Advanced

Edited

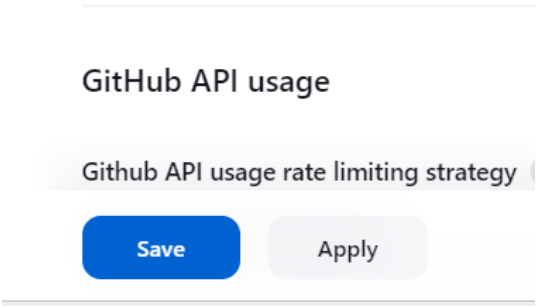
Once clicking that, scroll down we can able to see GitHub will be there, click the **advanced option**.



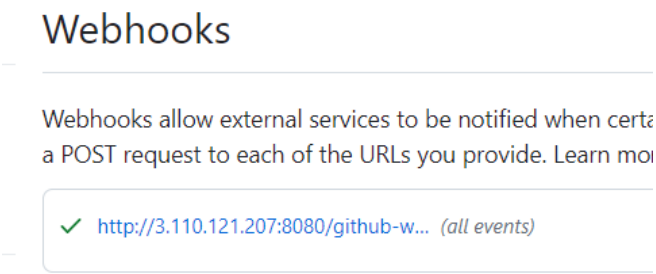
After clicking the advanced option, we can see the **Override Hook URL**, click that checkout box option.



Once we click the checkout box, we can able to see the URL that we give on **GitHub webhook** under the **playload URL.** Just click the apply and save option on the bottom of the screen.



Once clicking the apply and save option. Come back to the github repository webhook page, we can able to see the **green tick symbol** represents the connection was established between **GitHub and Jenkins**.



****

Creating Jenkins jobs:

Requirements:

- If a commit is made to master branch, test and push to prod
- If a commit is made to develop branch, just test the product, do not push to prod

If the commit is made to **master branch** job should be **tested on test node** and **pushed to prod node for uprunning**.

If the commit is made to **develop branch**, the job should be **tested on test node** and it should not be pushed to prod node.

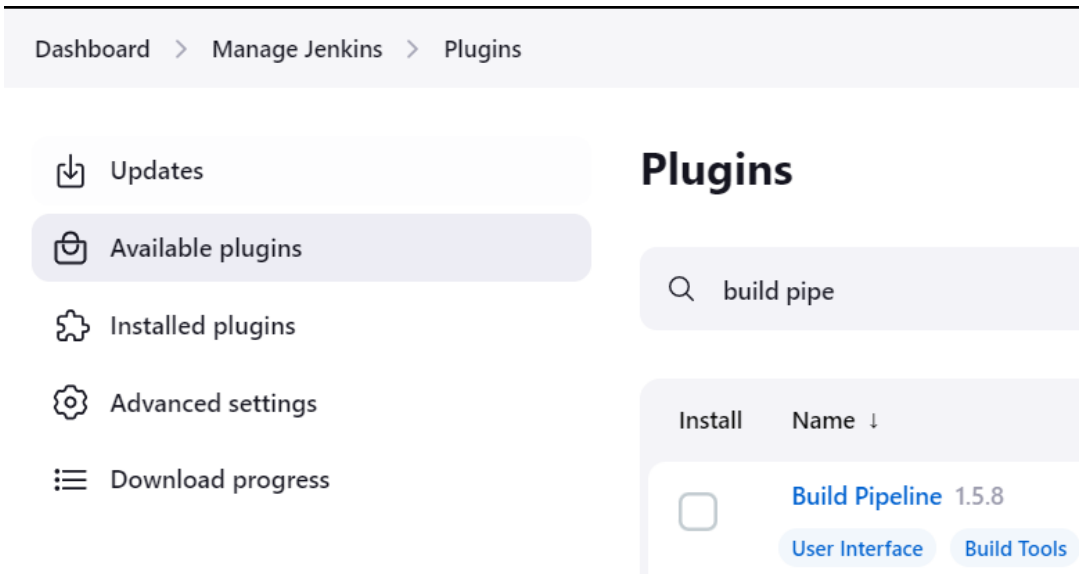
So, I am creating:

- **Master test job**
- **Master prod job**
- **Develop test job**

Required plugins: **Build Pipeline**

Installing the build pipeline plugin:

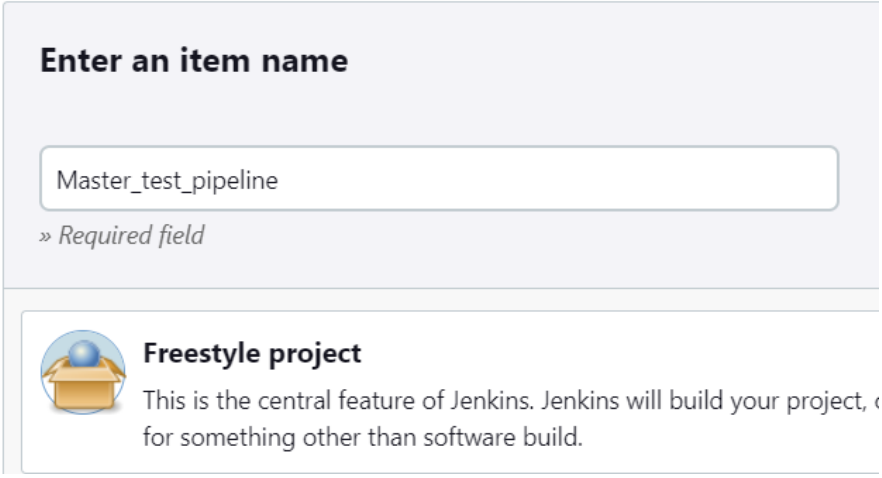
- Under **Jenkins dashboard** click the **manage Jenkins**. Under the manage Jenkins option click the **plugins**.



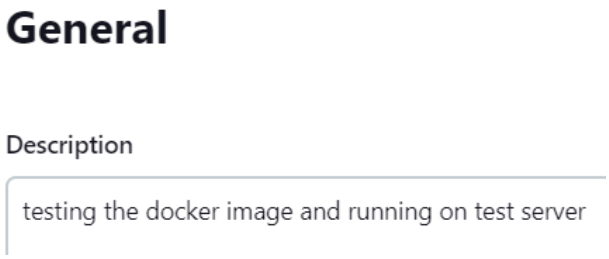
- After clicking the plugins. Click the available plugins search Build Pipeline, select the check box, click the option install without restart.
- This plugin will give the pipeline overview for free style project jobs.

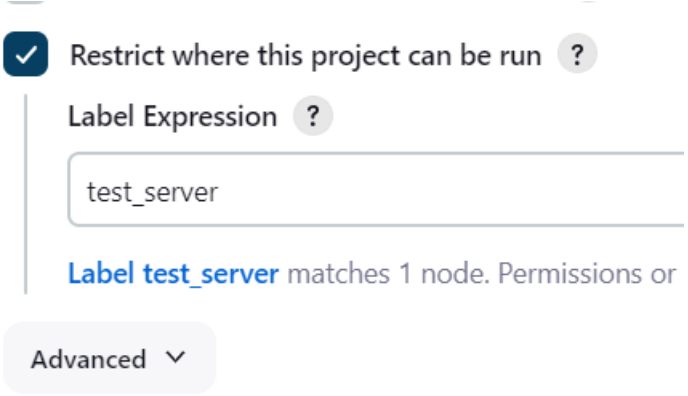
Master test job:

- On the Jenkins dashboard click the new. Give the name for the pipeline job and here I am selecting freestyle project.

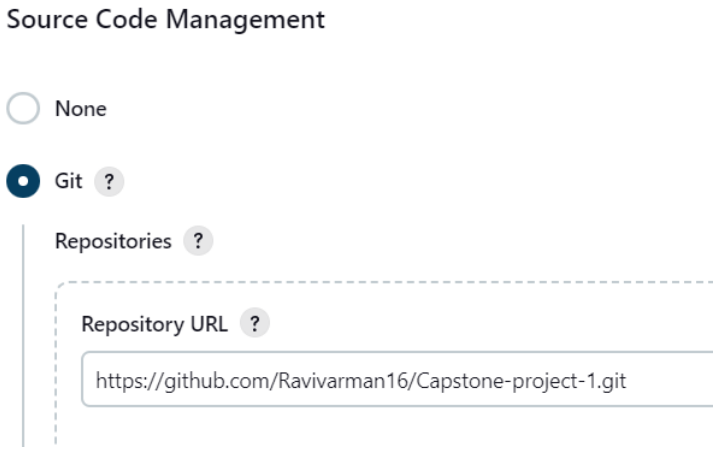


Giving the description for the job.

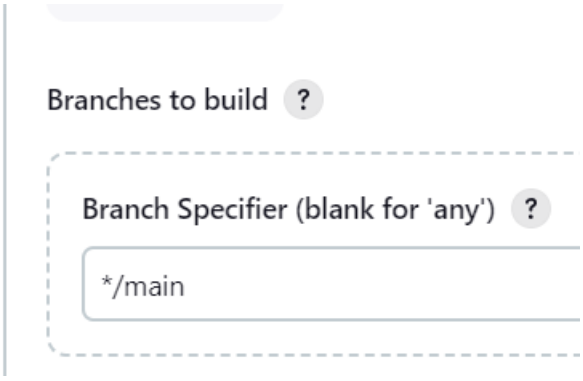




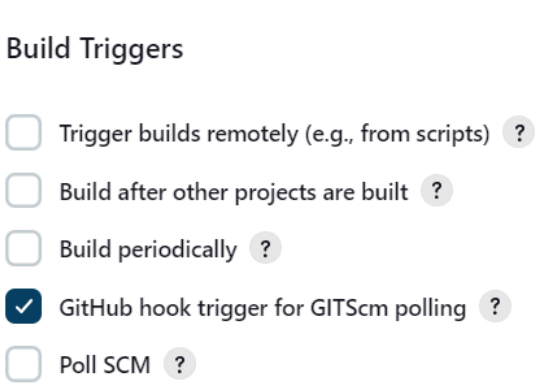
Under the general option there will be option – **restrict where this project can be run**, under give the test_server name because on this server code should build the docker image and check how it's working.



Under the **source code management** select the git and giving the repository our project where the code resides.



According to the test giving the **main branch.**



Under the **build triggers**: selecting the **GitHub hook trigger** option because this pushes the code to Jenkins whenever new commit it made to the repository.

Build Steps

Execute shell ?

Command

See the list of available environment variables

```
docker rm -f $(docker ps -aq)
docker build -t testimage .
docker run -d --name capstone-project -p 8080:80 testimage
```

Under the **build steps**, select the option **execute shell**: under the give the script to build the image and run a container from the image.

- Finally apply and save the project.

Now we need to make the second job master prod job:

Once code is tested on master_test_job immediately the code should be pushed master prod job for the final implement.

Enter an item name

Master_prod_job

» Required field

Freestyle project

This is the central feature of Jenkins. Jenkin
for something other than software build.

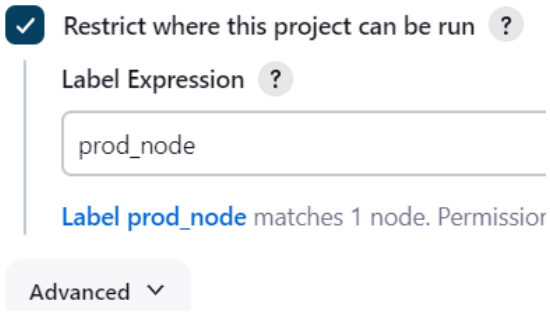
Naming this job as the **Master_prod_job** and selecting the **free style project.**

General

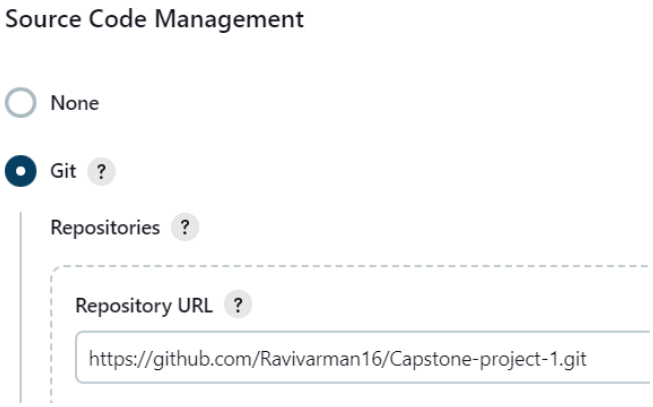
Description

finally implementation

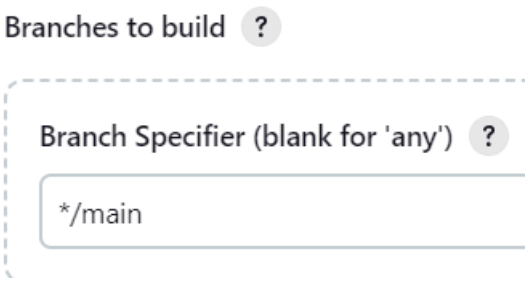
Under the description giving the description as per the project.



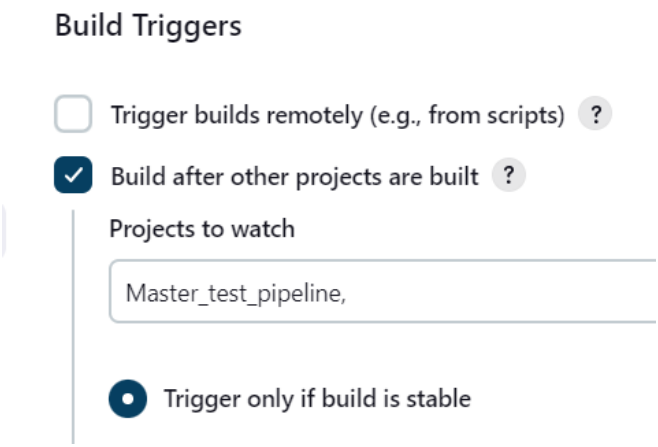
Under the **general**: **click the restrict where this project can be run – under label expression** give the prod_node name.



Under the **source code management** giving the github repository of the project where the code resides.



Under the branches giving the **main branch** as per the task requirements.

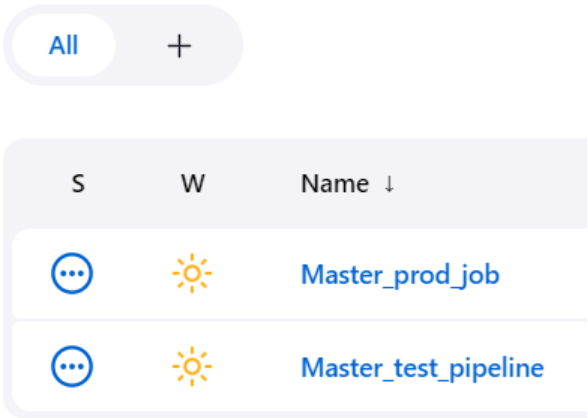


Under the **build Triggers option:** selecting the option- build after other projects are build. Selecting the before job for trigger purpose.

Under triggering: **selecting trigger only if build is stable**.

Under the **build steps**: select the option **execute shell**. And give the script or command according to the task.

And click apply and save it option.

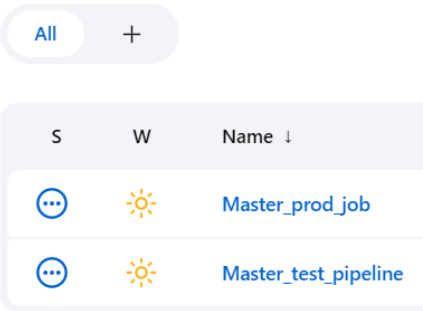


The jobs have been created for the first requirement:

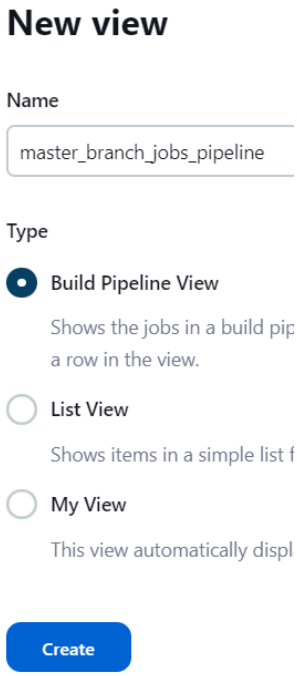
Once the commit is made to **master branch**, immediately the **master_test_pipeline** job will trigger and test it. Once the test is stable on master_test_pipeline **Master_prod_job** will run and implement the code at live.

To make the pipeline view clear we can create the **build pipeline view** with the help the **plugin – build pipeline**.

Steps to create the pipeline view:

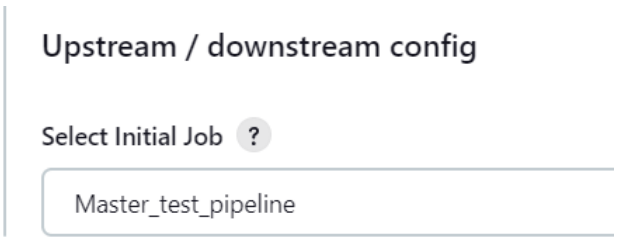


Select the + **symbol** at the top.



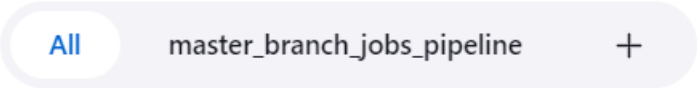
Naming the view as the – **master_branch_jobs_pipeline**

Selecting the **build pipeline view** and click the create option.



Under **the upstream/downstream config option:** selecting the initial job.

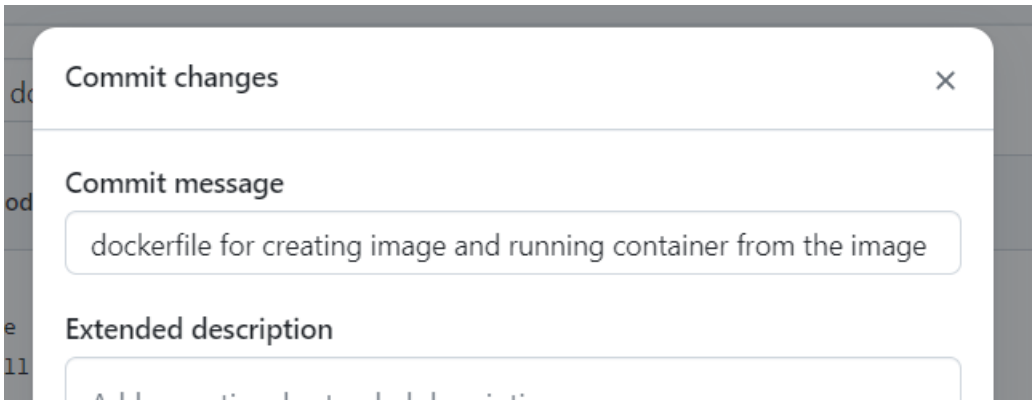
Keep the remaining options as the default and click apply.



The **pipeline view** has been created successfully.

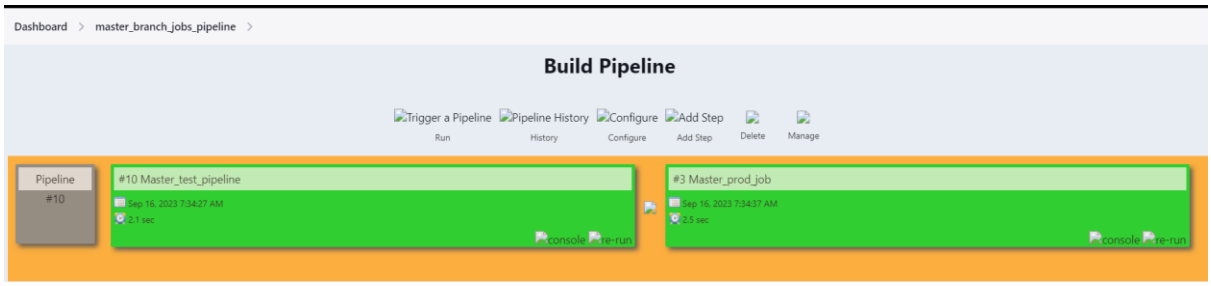
The pipeline has been set for the task-1:

Now we need to check it. I am making a small change on **GitHub** repository to see the **automatic trigger and build of CI/CD pipeline**.



I am making a change on the dockerfile and commiting it.

We can able to see the pipeline is trigger automatically.



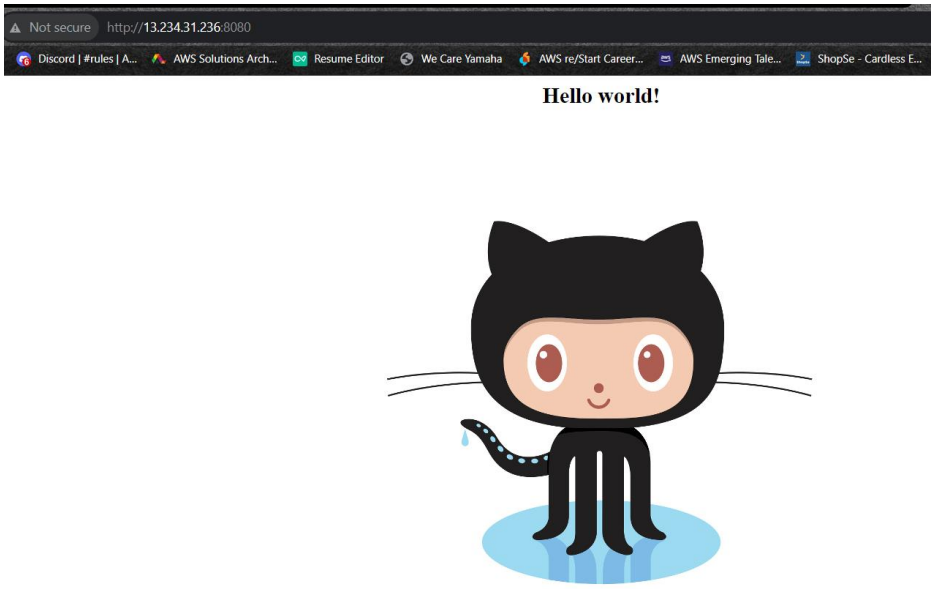
Copying and pasting the ip address of Jenkins_test and Jenkins_prod server

Ip address of Jenkins test server: **13.234.31.236**

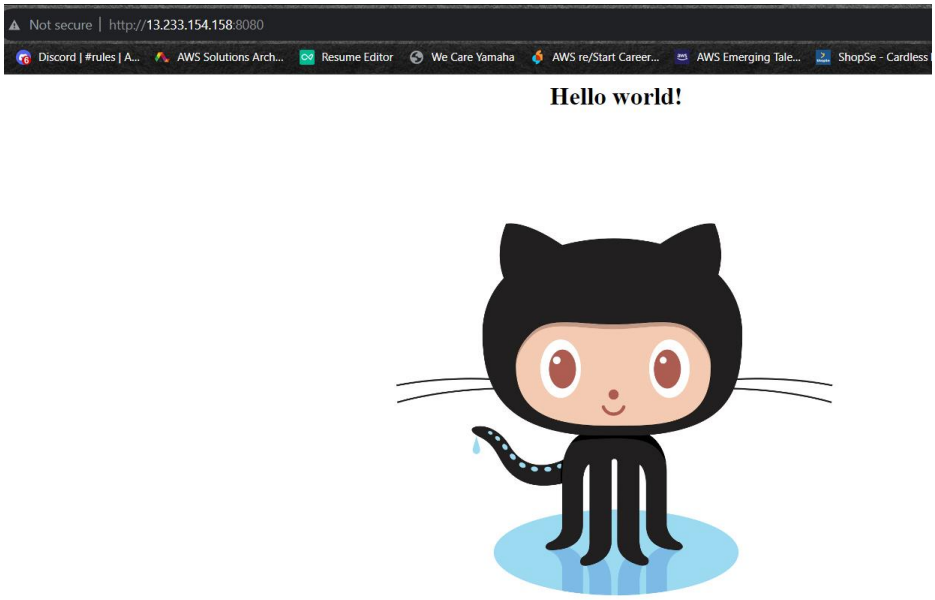
Ip address of Jenkins prod server: **13.233.154.158**

Checking on the browser:

Jenkins test server:



Jenkins prod server:

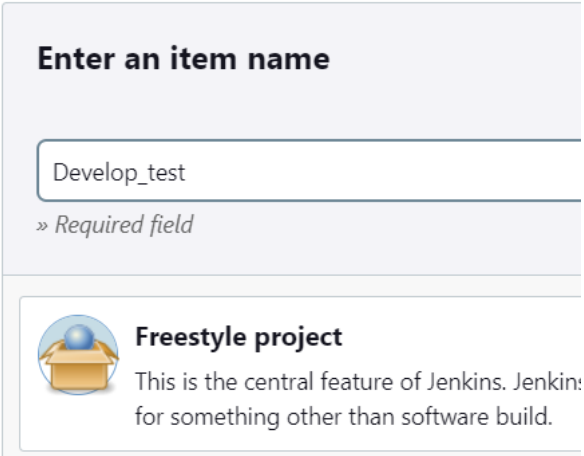


Task-1 has been executed perfectly.

Task-2 requirements:

- If a commit is made to develop branch, just test the product, do not push to prod.

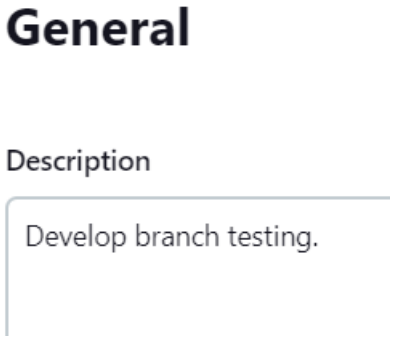
For this requirement creating a single job:

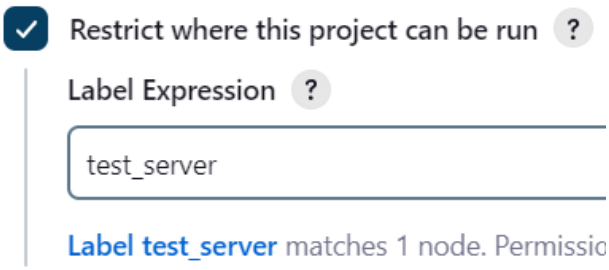


Creating a new job. Naming this job as **develop_test.**

And select the **freestyle project** and create it.

Giving the description for the job





Selecting the job to be run on **test_server node**.

Source Code Management

None

Git ?

Repositories ?

Repository URL ?

https://github.com/Ravivarman16/Capstone-project-1.git

Source code management giving GitHub repository link URL.

Branches to build ?

Branch Specifier (blank for 'any') ?

*/develop

Selecting the develop branch as per the task.

Build Triggers

- [ ] Trigger builds remotely (e.g., from scripts) ?
- [ ] Build after other projects are built ?
- [ ] Build periodically ?
- [x] GitHub hook trigger for GITScm polling ?
- [ ] Poll SCM ?

Under **build triggers** selecting the option github hook trigger option.

Build Steps

Execute shell ?

Command

See the list of available environment variables

```
docker rm -f $(docker ps -aq)
docker build -t developimage .
docker run -d --name capstone-develop-image -p 81:80 developimage
```

under the **Build steps:** select the option **execute shell**. Give the necessary commands as per the task.

Click apply and save it.

All | master_branch_jobs_pipeline | +

S	W	Name ↓
✓	☁	Develop_test
✓	☼	Master_prod_job
✓	☼	Master_test_pipeline

The job has been created. Now we need to test the job.

Commit changes ×

Commit message

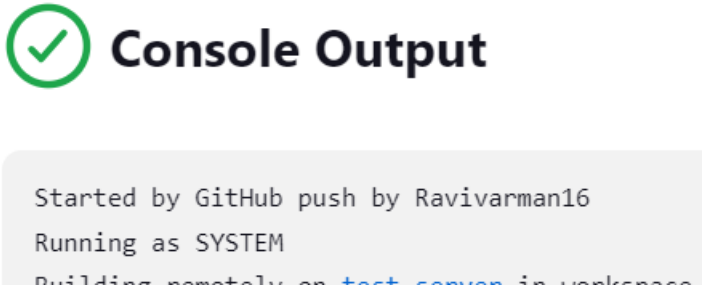
dockerfile from develop branch

Extended description

Add an optional extended description..

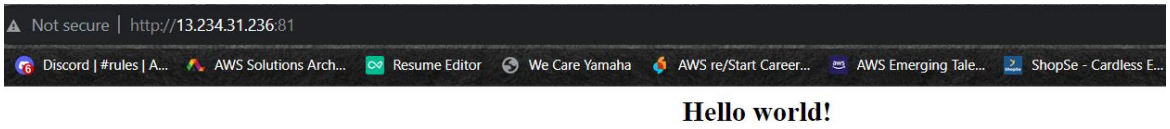
I am a small change on github repository especially on the develop branch.

Making the change and commit it.



On the Jenkins master dashboard, we can able to see the job has automatically triggered once I made a change from the developer branch. And executed perfectly.

Copying the public ip address of the test server and pasting it on the browser with port 81



The job has been executed. The task-2 has been fulfilled.

Task-2 CI/CD Pipeline has been setuped.

Checking the test_node and prod_node server:

Test_node:

```
root@ip-10-0-1-204:/home/ubuntu# ls
jenkins
root@ip-10-0-1-204:/home/ubuntu# cd jenkins/
root@ip-10-0-1-204:/home/ubuntu/jenkins# ls
remoting  remoting.jar  workspace
root@ip-10-0-1-204:/home/ubuntu/jenkins# cd workspace/
root@ip-10-0-1-204:/home/ubuntu/jenkins/workspace# ;s
bash: syntax error near unexpected token `;'
root@ip-10-0-1-204:/home/ubuntu/jenkins/workspace# ls
Develop_test  Master_test_pipeline
root@ip-10-0-1-204:/home/ubuntu/jenkins/workspace# ls Master_test_pipeline/
README.md  ansible.sh  dockerfile  images  index.html  jenkins.sh  main.tf  package.yml  prod.sh  terraform.sh  test.sh  variable.tf
root@ip-10-0-1-204:/home/ubuntu/jenkins/workspace# ls Develop_test/
README.md  ansible.sh  dockerfile  images  index.html  jenkins.sh  main.tf  package.yml  prod.sh  terraform.sh  test.sh  variable.tf
root@ip-10-0-1-204:/home/ubuntu/jenkins/workspace#
```

Prod_node:

```
root@ip-10-0-1-16:/home/ubuntu# ls
jenkins
root@ip-10-0-1-16:/home/ubuntu# cd jenkins/
root@ip-10-0-1-16:/home/ubuntu/jenkins# ls
remoting  remoting.jar  workspace
root@ip-10-0-1-16:/home/ubuntu/jenkins# cd workspace/
root@ip-10-0-1-16:/home/ubuntu/jenkins/workspace# ls
Master_prod_job
root@ip-10-0-1-16:/home/ubuntu/jenkins/workspace# ls Master_prod_job/
README.md  ansible.sh  dockerfile  images  index.html  jenkins.sh  main.tf  package.yml  prod.sh  terraform.sh  test.sh  variable.tf
root@ip-10-0-1-16:/home/ubuntu/jenkins/workspace#
```

Capstone Project -1 tasks had been executed.

GitHub Repository link: https://github.com/Ravivarman16/Capstone-project-1.git

********** THE END**********