J Ravivarman

DevOps Capstone Project – 2

TASKS TO BE DONE:

You are hired as a DevOps Engineer for Analytics Pvt Ltd. This company is a product-based organization which uses Docker for their containerization needs within the company. The final product received a lot of traction in the first few weeks of launch. Now with the increasing demand, the organization needs to have a platform for automating deployment, scaling, and operations of application containers across clusters of hosts. As a DevOps Engineer, you need to implement a DevOps lifecycle such that all the requirements are implemented without any change in the Docker containers in the testing environment. Up until now, this organization used to follow a monolithic architecture with just 2 developers. The product is present on: https://github.com/hshar/website.git

Following are the specifications of the lifecycle:

1. Git workflow should be implemented. Since the company follows a monolithic architecture of development, you need to take care of version control. The release should happen only on the 25th of every month.

2. Code Build should be triggered once the commits are made in the master branch.

3. The code should be containerized with the help of the Dockerfile. The Dockerfile should be built every time if there is a push to GitHub. Create a custom Docker image using a Dockerfile.

4. As per the requirement in the production server, you need to use the Kubernetes cluster and the containerized code from Docker Hub should be deployed with 2 replicas. Create a NodePort service and configure the same for port 30008.

5. Create a Jenkins Pipeline script to accomplish the above task.

6. For configuration management of the infrastructure, you need to deploy the configuration on the servers to install necessary software and configurations.

7. Using Terraform, accomplish the task of infrastructure creation in the AWS cloud provider.

Architectural Advice:

Software's to be installed on the respective machines using configuration management.

- Worker1: Jenkins, Java
- Worker2: Docker, Kubernetes
- Worker3: Java, Docker, Kubernetes
- Worker4: Docker, Kubernetes
- Worker5: Docker, Kubernetes

****

SOLUTION:

PREREQUISITES:

Cloud : AWS Cloud

Server or Vm : AWS EC2 instance

Operating System : Ubuntu 20.04

Network : AWS VPC

Security : AWS IAM

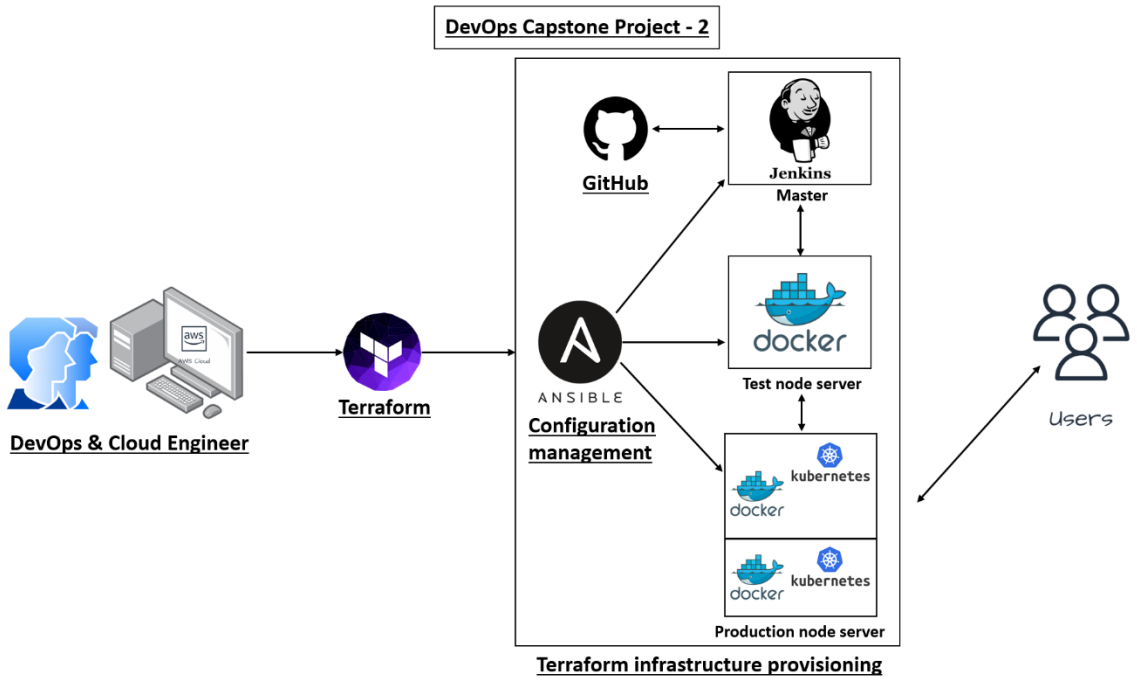
DevOps tools :

- Version Control and Collaboration : **Git and GitHub**
- CI/CD Pipeline : **Jenkins**
- Containerization : **Docker**
- Deployment and Container orchestration : **Kubernetes**
- Infrastructure Provisioning : **Terraform**
- Configuration Management : **Ansible**

Worker nodes:

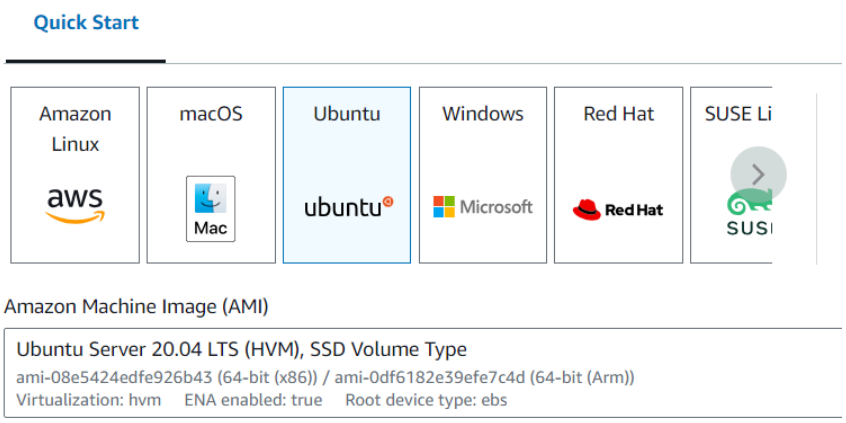
- Terraform master
- Ansible master
- Jenkins master
- Test server
- Prod server – 2

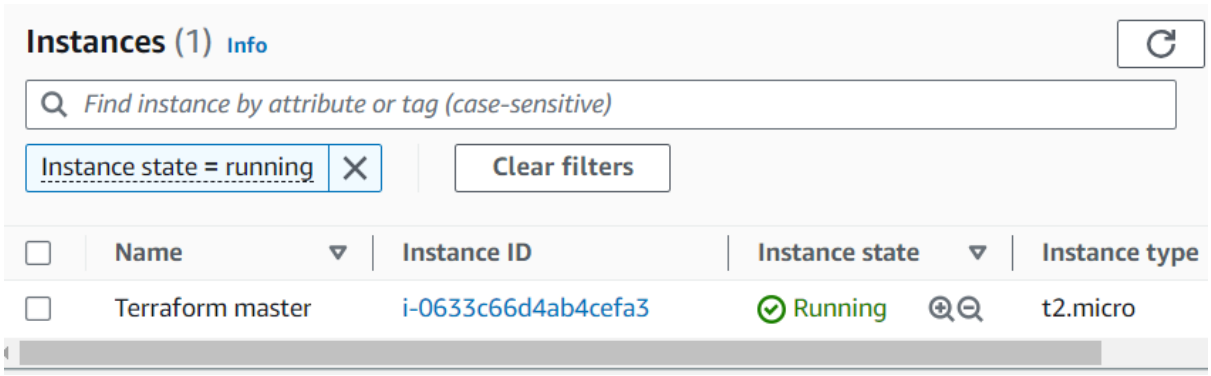
Outline Overview:



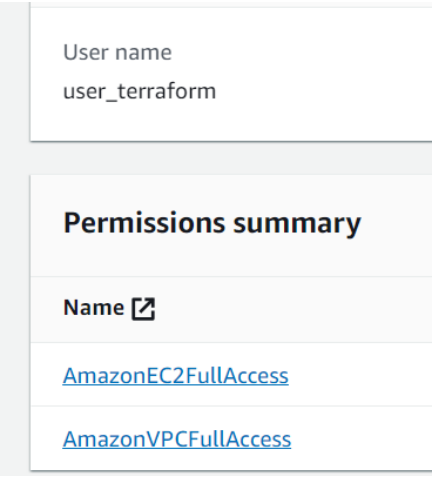
Infrastructure provisioning with Terraform:

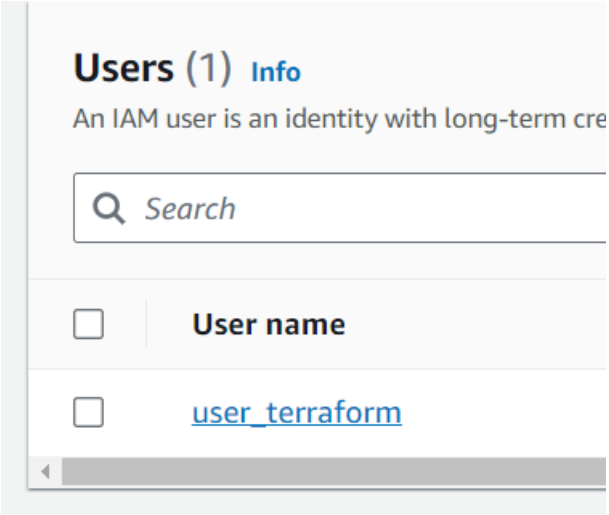
- First creating a instance for **terraform** with os as **Ubuntu 20.04**



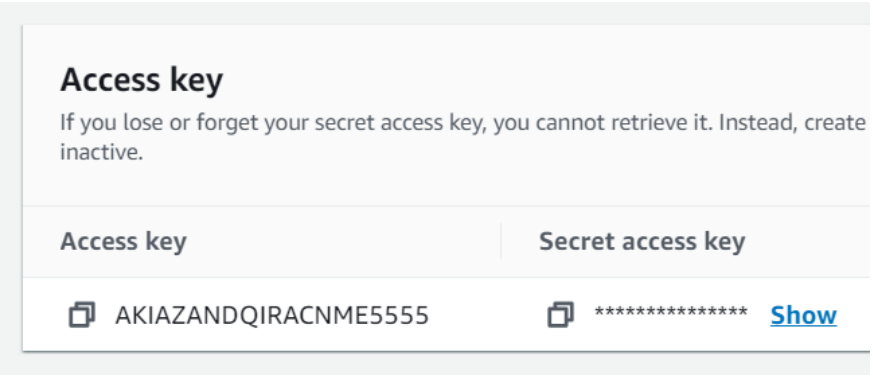


- Instance has been created. Now we need to create **IAM user** for terraform to control our cloud infrastructure for provisioning and destroying infrastructure.
- For the IAM user, I am giving the permissions – EC2 full access and VPC full access in order to create and destroy the resources.





- IAM user had been created. Now we need to generate the access key and secret for the user in order the access the command line features.



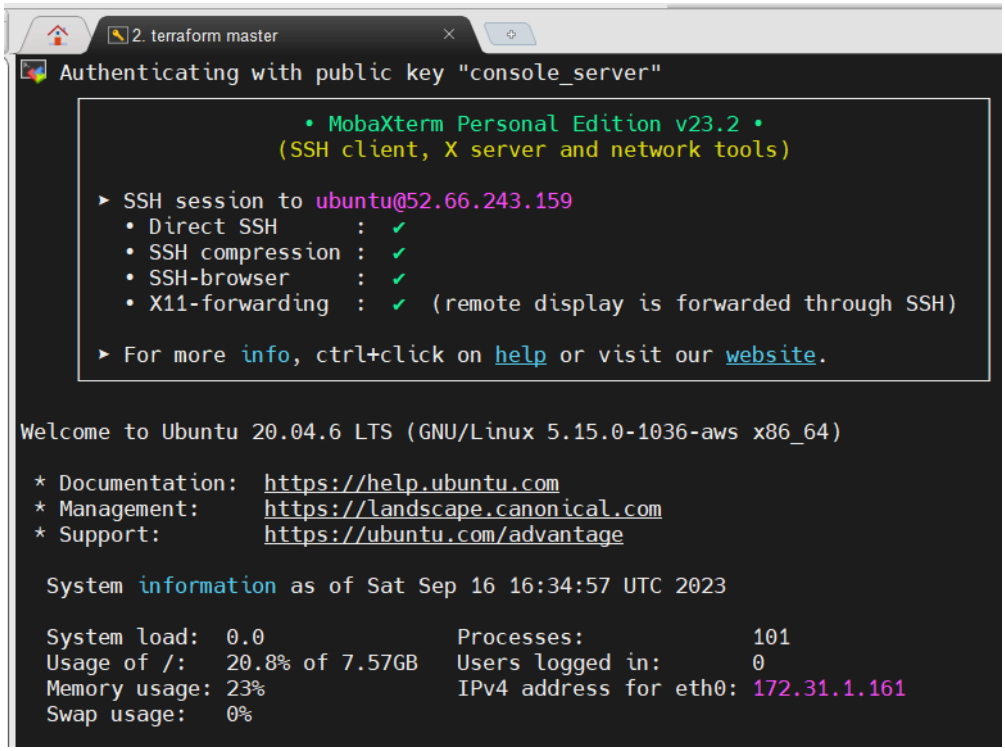
- After creating the access key and secret key. Connect the instance and install terraform on it.

Script to install terraform:

```
#!/bin/bash
#updating the os
apt-get update

#commands to install terraform
```

```
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg -
-dearmor -o /usr/share/keyrings/hashicorp-archive-
keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-
keyring.gpg] https://apt.releases.hashicorp.com
$(lsb_release -cs) main" | sudo tee
/etc/apt/sources.list.d/hashicorp.list

apt-get update
apt-get install -y terraform

#checking the terraform version
terraform --version
```

- I have connected the instance with mobaxterm for execution:



- Creating the script file and executing:

```
ubuntu@ip-172-31-1-161:~$ sudo su
root@ip-172-31-1-161:/home/ubuntu# touch terraform.sh
root@ip-172-31-1-161:/home/ubuntu# vi terraform.sh
root@ip-172-31-1-161:/home/ubuntu# chmod 744 terraform.sh
root@ip-172-31-1-161:/home/ubuntu# ./terraform.sh
```

- Terraform has been installed.

```
Terraform v1.5.7
on linux_amd64
```

- Now we need to provision the infrastructure required for the project with the help of the **terraform**.

Requirements: 4 instances, VPC, security groups, subnets, route table, internet gateway.

- Creating the variables.tf and main.tf file

```
root@ip-172-31-1-161:/home/ubuntu# vi variable.tf
root@ip-172-31-1-161:/home/ubuntu# vi main.tf
root@ip-172-31-1-161:/home/ubuntu# ls
main.tf  terraform.sh  variable.tf
root@ip-172-31-1-161:/home/ubuntu#
```

- Now performing the **terraform init** command for terraform download the necessary dependencies

```
root@ip-172-31-1-161:/home/ubuntu# terraform init

Initializing the backend...

Initializing provider plugins...
```

- Now align the terraform files with the command **terraform fmt** command.

```
root@ip-172-31-1-161:/home/ubuntu# terraform fmt
variable.tf
root@ip-172-31-1-161:/home/ubuntu#
```

- After formatting the terraform files. We need to validate the parameters by using the command **terraform validate** command.

```
root@ip-172-31-1-161:/home/ubuntu# terraform validate
Success! The configuration is valid.

root@ip-172-31-1-161:/home/ubuntu#
```

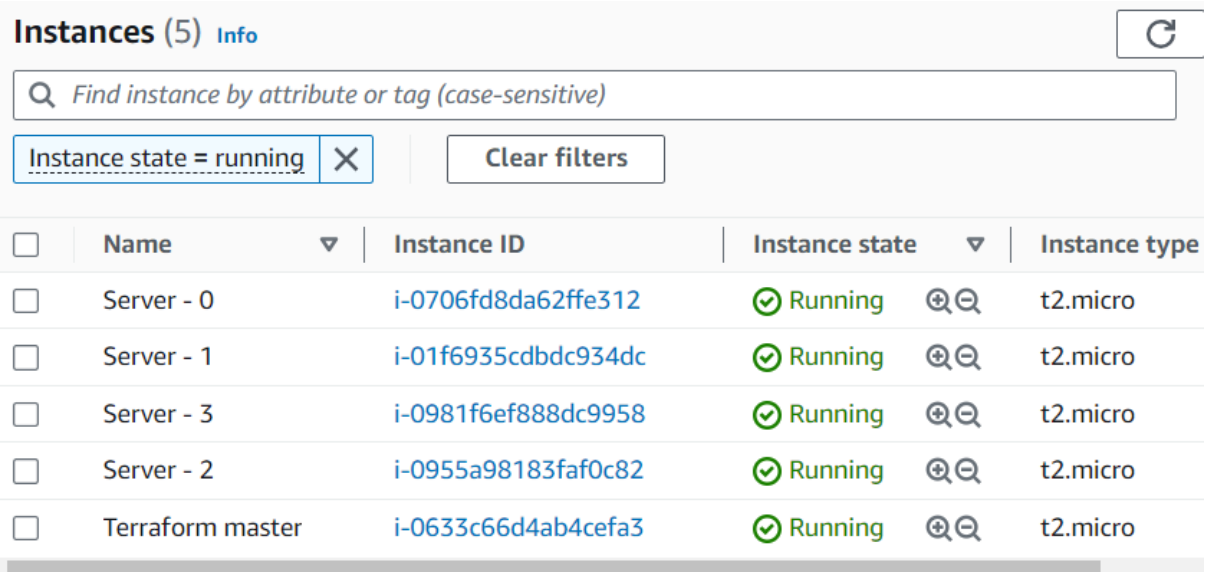
- Now we need to initialize terraform to plan the resource provision by using **terraform plan** command.

```
root@ip-172-31-1-161:/home/ubuntu# 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:
```

- Terraform plan command will show the details about resources. If the resources are okay, go ahead and apply the **terraform apply -auto-approve** command.

```
root@ip-172-31-1-161:/home/ubuntu# terraform apply -auto-approve

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

- Now we need to check on the console to see the resource provisioned by terraform

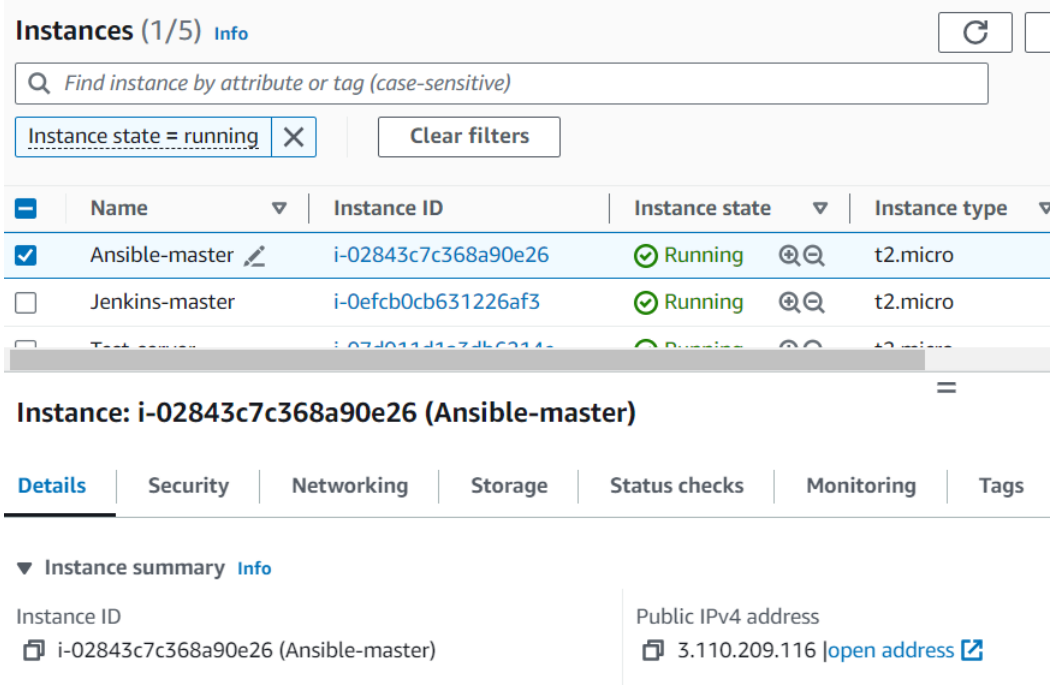


Resources have been provisioned by terraform, rename the instances according to the project requirements.

****

Configuration management by Ansible:

- First, we need to install the ansible on the ansible master instance.
- Connect the instance with mobaxterm:



Script to install ansible:

```
#!/bin/bash

#updating the operating system
apt-get update

#updating the ansible repository to the instance:
sudo apt-add-repository ppa:ansible/ansible

#updating the operating system:
sudo apt-get update

#installing the ansible:
sudo apt-get install -y ansible

#checking the ansible version:
ansible --version
```

```
root@ip-10-0-1-211:/home/ubuntu# vi ansible.sh
root@ip-10-0-1-211:/home/ubuntu# chmod 744 ansible.sh
root@ip-10-0-1-211:/home/ubuntu# ./ansible.sh
```

- Ansible has been installed successfully.

```
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, Mar 13 2023, 10:26:41) [GCC 9.4.0]
  jinja version = 2.10.1
  libyaml = True
```

- Now we need to setup the ansible cluster, in order setup the cluster we need to make a slight change on the slave nodes to connect with master node.

Changes on slave node:

- Change the password for the root account or create the new user for using ansible on it.
- Go to the **vi /etc/ssh/sshd_config**
- Allow for the **port access – 22**
- **Password authentication** to yes
- **Permit root login** to yes
- Finally **restart the sshd service**

Once the changes had been made on the slave nodes. Now we need to update the ip address on ansible master hosts host files.

Location of the host file is **vi /etc/ansible/hosts**

```
#   - You can enter hostnames or ip addresses
#   - A hostname/ip can be a member of multiple groups

# Ex 1: Ungrouped hosts, specify before any group headers:

## green.example.com
## blue.example.com
## 192.168.100.1
## 192.168.100.10
```

- It will display like this, after the updating our configuration details on the host file, it will look like.

```
root@ip-172-31-13-197:/home/ubuntu# cat /etc/ansible/hosts
[jenkins-master]
172.31.3.103

[test-server]
172.31.11.163

[prod-server]
172.31.4.88
172.31.13.198
```

- Here I have grouped into three groups, one for Jenkins master server, another one for test-server where docker will be installed, and the finally prod-servers where Kubernetes will be installed.

- Now we need to generate the ssh-key for ansible cluster setup by using the command – **ssh-keygen**

```
root@ip-172-31-13-197:/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:/OP0BlbCwWUJtmEGrjd+j8PpvzhR0NBq7nGMOD1leL4 root@ip-172-31-13-197
The key's randomart image is:
+---[RSA 3072]----+
|        .oB*o.   |
|       . ++++    |
|        ...=     |
|       o  * *    |
|      . S= @     |
|       oooX +    |
|        .==B .   |
|         +**E    |
|          .===.  |
+----[SHA256]-----+
root@ip-172-31-13-197:/home/ubuntu#
```

- Now we need to copy the ssh-key generated on slave nodes by using **ssh-copy-id <user>@<ip_address>**

```
root@ip-172-31-13-197:/home/ubuntu# ssh-copy-id root@172.31.3.103
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
The authenticity of host '172.31.3.103 (172.31.3.103)' can't be established.
ECDSA key fingerprint is SHA256:LMCcDdjO9C5g8N2/v3dyAsWJPLPQAxZgxV9tjgqSaFg.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@172.31.3.103's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@172.31.3.103'"
and check to make sure that only the key(s) you wanted were added.

root@ip-172-31-13-197:/home/ubuntu#
```

The key was copied on Jenkins master slave node

```
root@ip-172-31-13-197:/home/ubuntu# ssh-copy-id root@172.31.11.163
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
The authenticity of host '172.31.11.163 (172.31.11.163)' can't be established.
ECDSA key fingerprint is SHA256:nQWbtN/52s9nx8JQjMUqFNTu0mq4qak7gx++WyjMs3o.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@172.31.11.163's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@172.31.11.163'"
and check to make sure that only the key(s) you wanted were added.

root@ip-172-31-13-197:/home/ubuntu#
```

The key was copied on test-server slave node.

```
root@ip-172-31-13-197:/home/ubuntu# ssh-copy-id root@172.31.4.88
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
The authenticity of host '172.31.4.88 (172.31.4.88)' can't be established.
ECDSA key fingerprint is SHA256:NyIC/Nqp8A8t1URA7dQv09uP6JfBT5awxaTwZJHXEmw.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@172.31.4.88's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@172.31.4.88'"
and check to make sure that only the key(s) you wanted were added.

root@ip-172-31-13-197:/home/ubuntu#
```

```
root@ip-172-31-13-197:/home/ubuntu/ansible# ssh-copy-id root@172.31.13.198
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
The authenticity of host '172.31.13.198 (172.31.13.198)' can't be established.
ECDSA key fingerprint is SHA256:vCC5eKSY1Y6CY/OR8B3EEBZasT5/wJBmlPFJpOl9D90.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@172.31.13.198's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@172.31.13.198'"
and check to make sure that only the key(s) you wanted were added.

root@ip-172-31-13-197:/home/ubuntu/ansible#
```

The key was copied on prod-server slave node.

- Now we need check whether ansible can able to connect with the slave nodes by using **ansible ping** command we ensure that.

```
root@ip-172-31-13-197:/home/ubuntu/ansible# ansible all -m ping
[WARNING]: Invalid characters were found in group names but not replaced, use -vvvv to see details
172.31.3.103 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
172.31.11.163 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
172.31.13.198 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
172.31.4.88 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
root@ip-172-31-13-197:/home/ubuntu/ansible#
```

Ansible cluster had been setuped successfully.

- Now we need to execute the playbook for installing the packages on slave nodes.

Required packages on slave nodes:

- **Jenkins master:** Openjdk-11-jre, Jenkins
- **Test server:** Docker, java
- **Prod server:** Docker, Kubernetes, java

In order to execute the ansible playbook, we need to create the yaml file with necessary script to install packages on slave nodes.

Yaml file contains the script to install the packages will be uploaded the GitHub repository:

- Creating the script files required to install the packages:

```
root@ip-172-31-13-197:/home/ubuntu# mkdir ansible
root@ip-172-31-13-197:/home/ubuntu# cd ansible
root@ip-172-31-13-197:/home/ubuntu/ansible# vi jenkins.sh
root@ip-172-31-13-197:/home/ubuntu/ansible# vi test.sh
root@ip-172-31-13-197:/home/ubuntu/ansible# vi prod.sh
root@ip-172-31-13-197:/home/ubuntu/ansible#
```

- Now checking the syntax of main yml playbook file with the help of the command **ansible-playbook package.yml --syntax-check**

```
root@ip-172-31-13-197:/home/ubuntu/ansible# vi package.yml
root@ip-172-31-13-197:/home/ubuntu/ansible# ansible-playbook package.yml --syntax-check
[WARNING]: Invalid characters were found in group names but not replaced, use -vvvv to see details

playbook: package.yml
```

- Syntax has been verified. Now we need to execute it by using the command - **ansible-playbook package.yml**

Before apply the command checking the slave nodes whether the required packages:

On Jenkins master:

```
root@ip-172-31-3-103:/home/ubuntu# java --version

Command 'java' not found, but can be installed with:

apt install openjdk-11-jre-headless  # version 11.0.19+7~us1-0ubuntu1~20.04.1, or
apt install default-jre              # version 2:1.11-72
apt install openjdk-16-jre-headless  # version 16.0.1+9-1~20.04
apt install openjdk-17-jre-headless  # version 17.0.7+7~us1-0ubuntu1~20.04
apt install openjdk-8-jre-headless   # version 8u372-ga~us1-0ubuntu1~20.04
apt install openjdk-13-jre-headless  # version 13.0.7+5-0ubuntu1~20.04

root@ip-172-31-3-103:/home/ubuntu# jenkin --version
jenkin: command not found
root@ip-172-31-3-103:/home/ubuntu#
```

On test-server:

```
root@ip-172-31-11-163:/home/ubuntu# docker -v

Command 'docker' not found, but can be installed with:

snap install docker     # version 20.10.24, or
apt  install docker.io  # version 20.10.21-0ubuntu1~20.04.2

See 'snap info docker' for additional versions.

root@ip-172-31-11-163:/home/ubuntu#
```

On prod-server:

```
root@ip-172-31-4-88:/home/ubuntu# docker -v

Command 'docker' not found, but can be installed with:

snap install docker     # version 20.10.24, or
apt  install docker.io  # version 20.10.21-0ubuntu1~20.04.2

See 'snap info docker' for additional versions.

root@ip-172-31-4-88:/home/ubuntu# kubeadm

Command 'kubeadm' not found, but can be installed with:

snap install kubeadm

root@ip-172-31-4-88:/home/ubuntu#
```

After applying the command:

On ansible master:

```
root@ip-172-31-13-197:/home/ubuntu/ansible# ansible-playbook package.yml
[WARNING]: Invalid characters were found in group names but not replaced, use -vvvv to see details

PLAY [To install java and jenkins on jenkins master] ***************************************************

TASK [Gathering Facts] *********************************************************************************
ok: [172.31.3.103]

TASK [installing the packages on jenkins-master] ******************************************************
changed: [172.31.3.103]

TASK [updating the task work] **************************************************************************
ok: [172.31.3.103] => {
    "msg": "packages had been installed successfully on jenkins-master"
}
```

On Jenkins master slave node:

```
root@ip-172-31-3-103:/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-172-31-3-103:/home/ubuntu# jenkins --version
2.414.1
root@ip-172-31-3-103:/home/ubuntu#
```

On ansible master:

```
PLAY [To install docker on test-server] ****************************

TASK [Gathering Facts] *********************************************
ok: [172.31.11.163]

TASK [installing the packages on test-server] **********************
changed: [172.31.11.163]

TASK [updating the task work] **************************************
ok: [172.31.11.163] => {
    "msg": "packages had been installed successfully on test-server"
}
```

On test-server slave node:

```
root@ip-172-31-11-163:/home/ubuntu# docker -v
Docker version 24.0.5, build 24.0.5-0ubuntu1~20.04.1
root@ip-172-31-11-163:/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 Sun 2023-09-17 05:53:39 UTC; 10min ago
```

On ansible master:

```
TASK [updating the task work] **************************************
task path: /home/ubuntu/ansible/package.yml:10
ok: [172.31.4.88] => {
    "msg": "packages had been installed successfully on prod-server"
}
ok: [172.31.13.198] => {
    "msg": "packages had been installed successfully on prod-server"
}
```

On prod-servers:

```
root@ip-172-31-4-88:/home/ubuntu# docker -v
Docker version 24.0.5, build 24.0.5-0ubuntu1~20.04.1
root@ip-172-31-4-88:/home/ubuntu# systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
```

```
root@ip-172-31-4-88:/home/ubuntu# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"28", GitVersion:"v1.28.2"
7b2f", GitTreeState:"clean", BuildDate:"2023-09-13T09:34:32Z", GoVersion:"
root@ip-172-31-4-88:/home/ubuntu#
```

2nd prod servers:

```
root@ip-172-31-13-198:/home/ubuntu# docker -v
Docker version 24.0.5, build 24.0.5-0ubuntu1~20.04.1
root@ip-172-31-13-198:/home/ubuntu# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"28",
eeState:"clean", BuildDate:"2023-09-13T09:34:32Z", GoV
root@ip-172-31-13-198:/home/ubuntu#
```

Now ansible has done its job by installing necessary software's on host nodes:

****

So far infrastructure provisioning is done, and configuration management done.

Now we need to two things to do in order to setup the pipeline:

- **Jenkins master setup for pipeline.**
- **Kubernetes cluster setup for production**

First, we will setup **Kubernetes cluster setup in prod-server node:**

- On the main prod node, we need to init kubeadm by using the command – **kubeadm init --ignore-preflight-errors=all.** This is will initiate this prod server as the k8s master node.

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:
```

- Now we need to follow-up these setups in order setup the **k8s cluster setup.**

```
To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

- These steps to be done only in on main production server.
- After doing these steps on main prod server, we need to generate the token for connecting the second prod server by using the command - **kubeadm token create --print-join-command**

```
root@ip-172-31-4-88:/home/ubuntu# kubeadm token create --print-join-command
kubeadm join 172.31.4.88:6443 --token 944sip.vgjme88z8hxujhsn --discovery-token-ca-cert-hash sha256:be15c3ed60b65d5308345b2
3f5e691c44a03f1ce1541088cfecc468e99882987
root@ip-172-31-4-88:/home/ubuntu#
```

Copy the token and paste it on second prod server node.

```
root@ip-172-31-13-198:/home/ubuntu# kubeadm join 172.31.7.205:6443 --token p1uox7.i2pr6co8ajtvgbkj --discov
ery-token-ca-cert-hash sha256:e4c97ba6d92488f5401335a2d8e49b65bfce5ab93981c5d5e71674e26969f376
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yam
l'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

root@ip-172-31-13-198:/home/ubuntu#
```

This prod 2nd server node has been joined as slave to the prod main server.

- For the network connectivity between k8s cluster we need to install one package - kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml

```
root@ip-172-31-4-88:/home/ubuntu# kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemo
nset-k8s.yaml

serviceaccount/weave-net created
clusterrole.rbac.authorization.k8s.io/weave-net created
clusterrolebinding.rbac.authorization.k8s.io/weave-net created
role.rbac.authorization.k8s.io/weave-net created
rolebinding.rbac.authorization.k8s.io/weave-net created
daemonset.apps/weave-net created
root@ip-172-31-4-88:/home/ubuntu#
root@ip-172-31-4-88:/home/ubuntu#
```

- Now we need to check the cluster setup of production servers: **kubectl get nodes**

```
2. ansible_master    3. jenkins-master    4. test-server
root@ip-172-31-7-205:/home/ubuntu# kubectl get nodes
NAME                STATUS   ROLES           AGE     VERSION
ip-172-31-13-198    Ready    <none>          4m10s   v1.28.2
ip-172-31-7-205     Ready    control-plane   5m43s   v1.28.2
root@ip-172-31-7-205:/home/ubuntu#
```

Production server setup has done successfully.

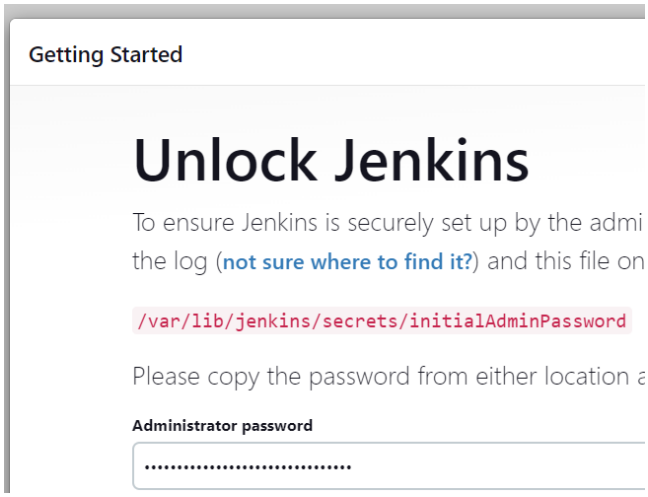
****

Jenkins Master – slave setup for pipeline execution:

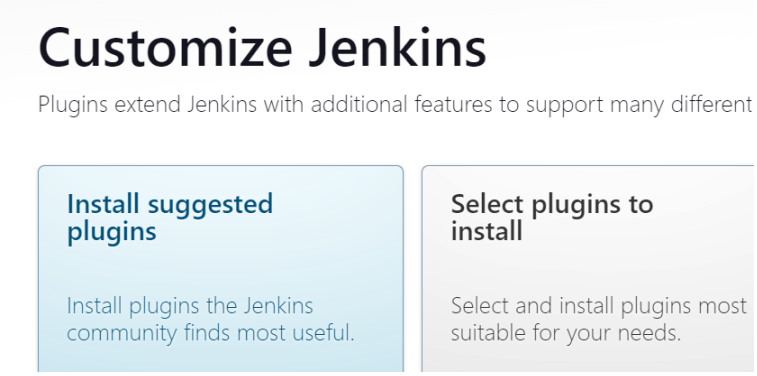
- First, we need to login in Jenkins management console, for that we need to get the password first on Jenkins master server:
  cat /var/lib/jenkins/secrets/initialAdminPassword

```
root@ip-172-31-3-103:/home/ubuntu# cat /var/lib/jenkins/secrets/initialAdminPassword
ceda761efc0b47e7849f5a4b03f13a88
root@ip-172-31-3-103:/home/ubuntu#
```

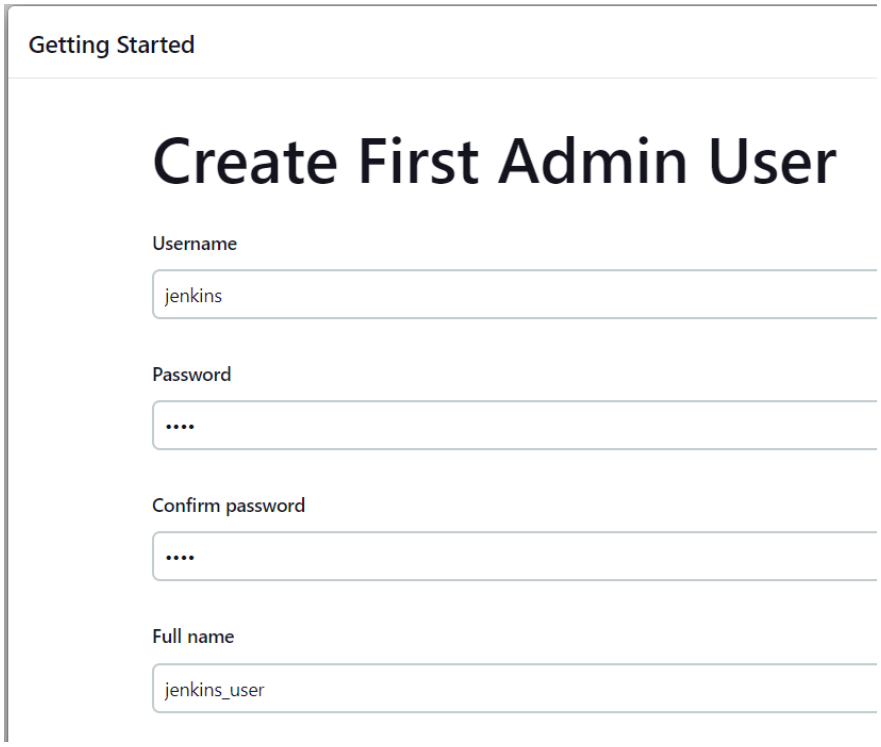
- Copy and paste the password and proceed further:



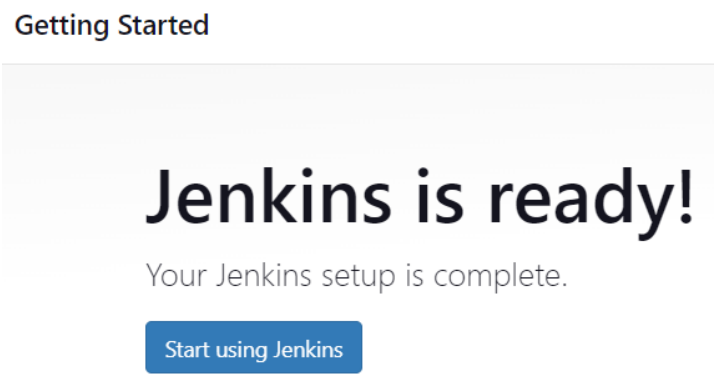
- Then we need to select the plugins installation based on our needs:



- Once the plugins are installed. Then setup the username and password credentials for using Jenkins dashboard, save and continue.

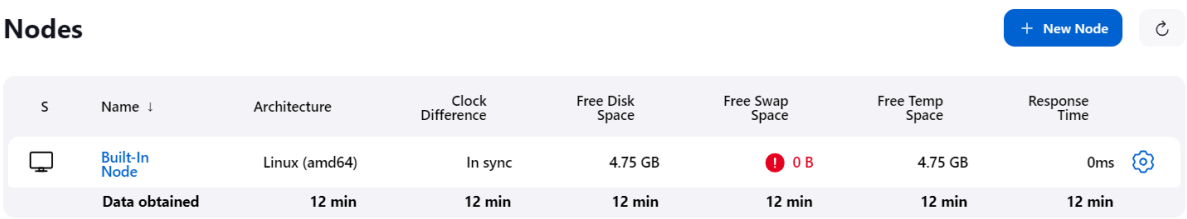


- Then finally **Jenkins will be ready:**

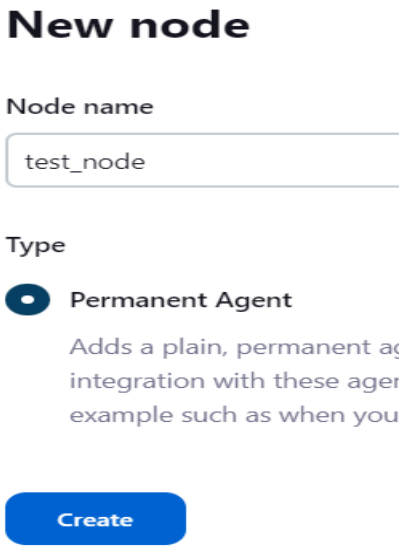


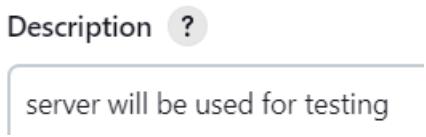
Now we need to setup master-slave connection:

- For master node setup: click **manage Jenkins**, under manage Jenkins click **Nodes**.

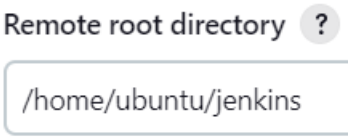


- Click new node on the left side: give the node a **name and select permanent agent.**



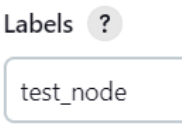


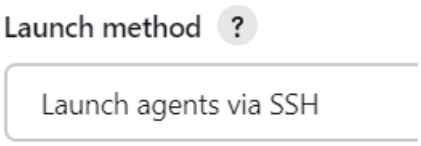
Giving the description according to the project.



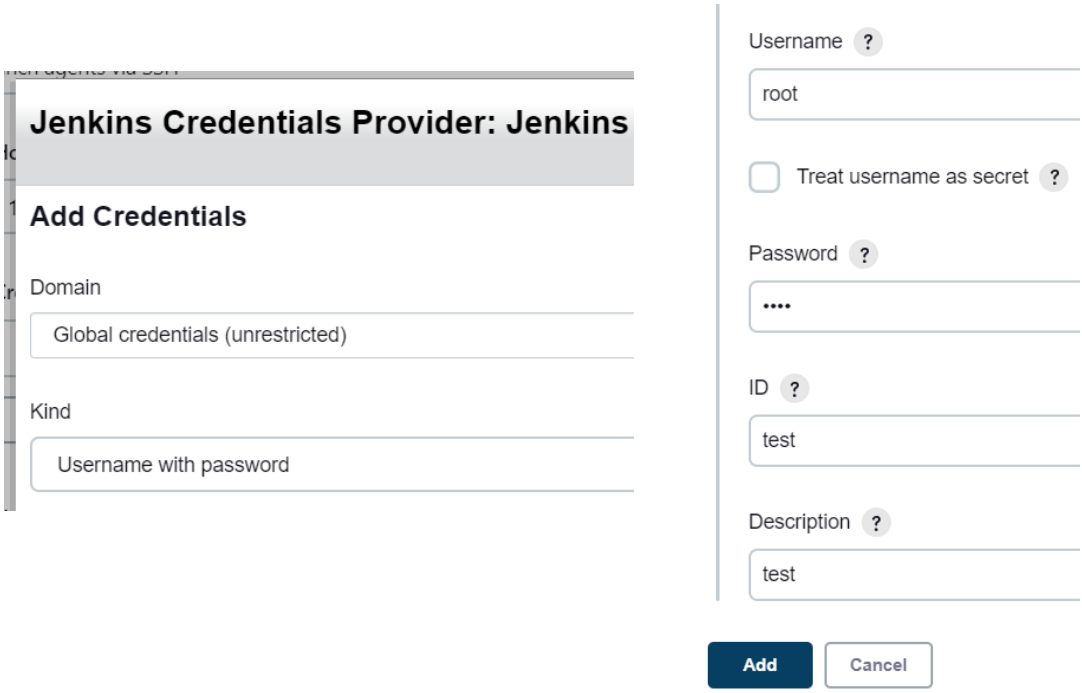
Giving the **remote directory** where we can see the details of jobs execution.

Give the **labels** for node for identification.

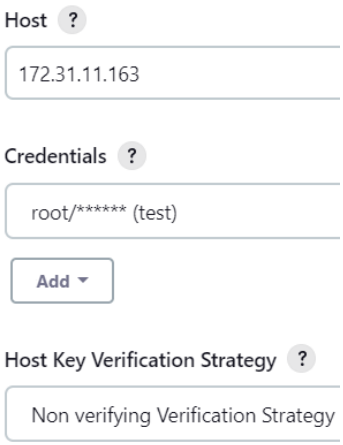




Selecting the **launch method**: here I am selecting the **launch agent via ssh.**



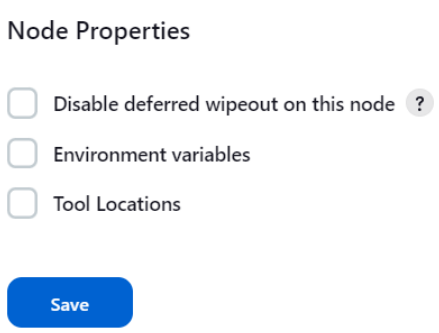
- For ssh connection giving we need to giving the credentials for safety login.



After giving the credentials details now we need to give **ip address of node under host**.

Selecting the credentials which we created just now.

Under **host verification strategy**: selecting **non verifying verification strategy.**



Remaining setting keeps as the default and save it.

- Once we saved under nodes, we can see the **test_node** has been added successfully.

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	4.53 GB	0 B	4.53 GB	0ms
	test_node	Linux (amd64)	In sync	4.70 GB	0 B	4.70 GB	61ms
	Data obtained	**51 ms**	**51 ms**	**51 ms**	**51 ms**	**51 ms**	**46 ms**

Like this we need add **prod main server to Jenkins master** in order Jenkins to control it.

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	4.75 GB	0 B	4.75 GB	0ms
	prod_main_server	Linux (amd64)	In sync	5.22 GB	0 B	5.22 GB	30ms
	test_node	Linux (amd64)	In sync	4.91 GB	0 B	4.91 GB	5ms
	Data obtained	**0.28 sec**	**0.28 sec**	**0.28 sec**	**0.28 sec**	**0.28 sec**	**0.28 sec**

Two nodes were added.

****

Now we need to create the jobs for pipeline execution:

- First, we need to setup the **GitHub hook trigger** for automatically running of the Jenkins pipeline.
- On the **GitHub** repository settings, we can see webhooks on the right-hand side:

Webhooks

Add webhook

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.

- Click add webhook, after that under **payload URL** giving the Jenkins URL along with **github-webhook api.**
- Under the secret give the github secret key, for trigger notifications select according to your preference. Click add webhook.

Payload URL *

http://35.93.137.55:8080/github-webhook/

Content type

application/x-www-form-urlencodec

Secret

Which events would you like to trigger this webhook?

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

- The hook trigger has been setuped on **GitHub** side. Now we need to setup on Jenkins side. Under **manage Jenkins, click system**.

System Configuration

System

Configure global settings and paths.

Under **GitHub**, select the advanced option.

GitHub

GitHub Servers ?

Add GitHub Server

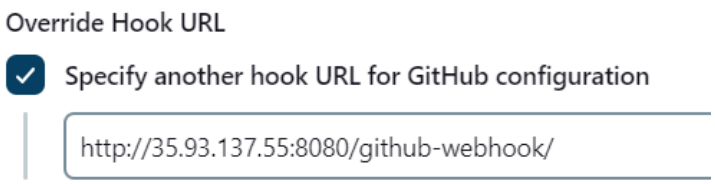
Advanced

Edited

After clicking the advanced option, we can another option called **override hook URL,** click the checkout.

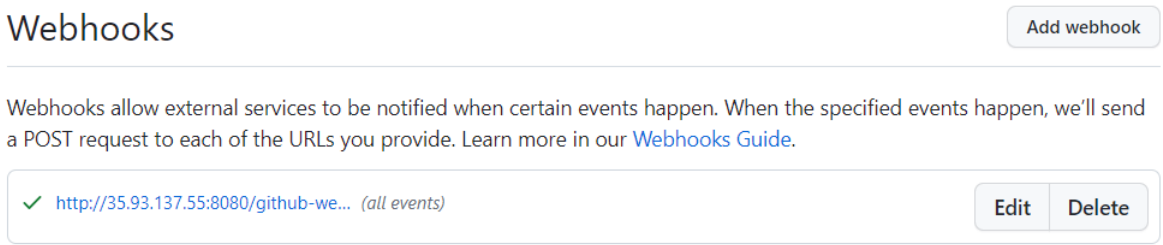
Override Hook URL

☐ Specify another hook URL for GitHub configuration



we can able to the Jenkins URL along with github-webhook api.

Click apply and save at the bottom of the screen.



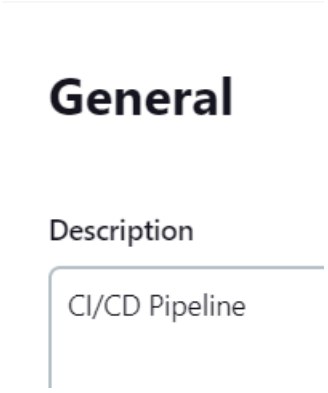
- We can able **GitHub hook trigger** has setuped successfully between GitHub and Jenkins.

Creating Jenkins pipeline jobs:

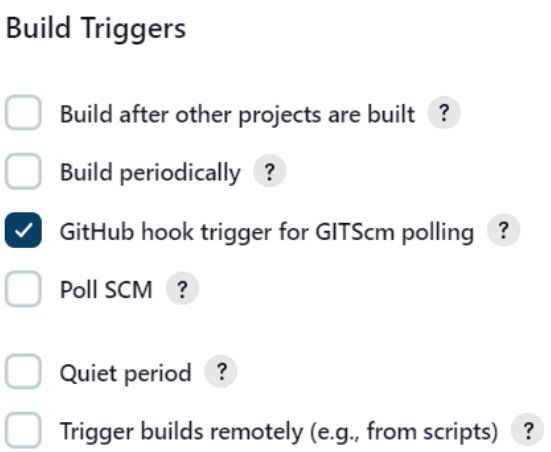
Requirements: 2 jobs

- **1st job** is to run on test server for testing, where docker will build image and run a container. If the application is okay then it will proceed to production server.
- **2nd job** will run on production server, where k8s will take deployment and container orchestration.

1ST Pipeline Job:



Creating a new job, selecting **pipeline type**, giving the description.



Selecting **GitHub webhook trigger** option for trigger purpose.

- Then under pipeline we need to write the **groovy script:**

Groovy script to run the job on test server:

```
pipeline {
    agent { label 'test_node' }

    stages {
        stage('Git cloning') {
            steps {
                git branch: 'main',
                url:
'https://github.com/Ravivarman16/DevOps-Capstone-project-
2.git'
            }
        }

        stage ('Building a dockerimage') {
            steps {
                sh 'docker rm -f $(docker ps -aq)'
                sh 'docker build -t ravivarman46/testimage
.'
                sh 'docker run -d --name container1 -p
8080:80 ravivarman46/testimage'
                sh 'docker push ravivarman46/testimage'
            }
        }
    }
}
```

Click **apply and save.**

Pipeline

Definition

Pipeline script

Script ?

```
pipeline {
    agent { label 'test_node' }

    stages {
        stage('Git cloning') {
            steps {
                git branch: 'main',
```

- Then on the test server we need to login with docker Hub for push the test image to docker hub, so that Kubernetes service on the production server will pull this image for container running and managing.

```
root@ip-172-31-24-246:/home/ubuntu# docker login
Login with your Docker ID to push and pull images
ps://hub.docker.com to create one.
Username: ravivarman46
Password:
WARNING! Your password will be stored unencrypted
Configure a credential helper to remove this warn
https://docs.docker.com/engine/reference/commandl

Login Succeeded
root@ip-172-31-24-246:/home/ubuntu#
```

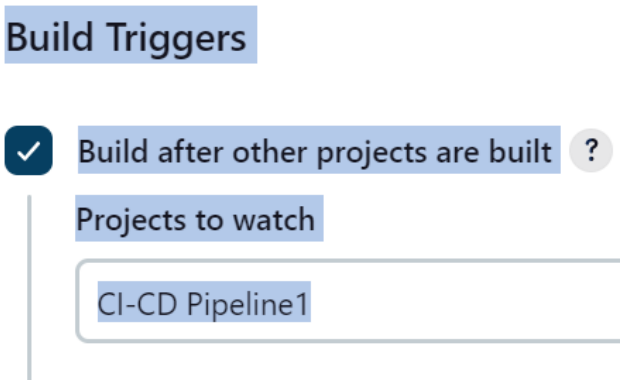
2nd Pipeline Job:

General

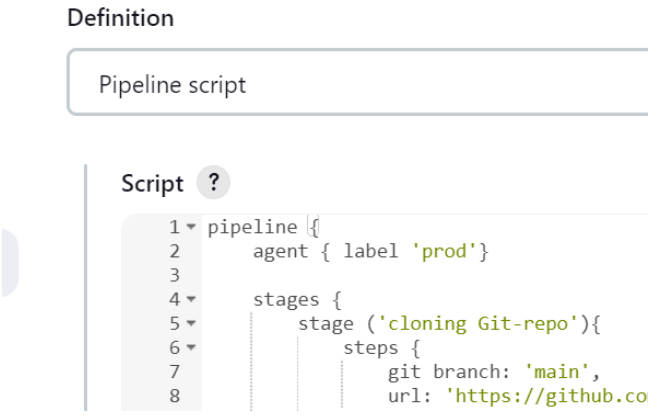
Creating 2nd pipeline job, with **pipeline type**. Under description giving the description according to the project.

Description

production server



Under build triggers we need to **select the first job as the initial job**, so that once the first job executed successfully 2nd pipeline job will execute.



Under pipeline script: we need to write script like this job should be run on prod main server.

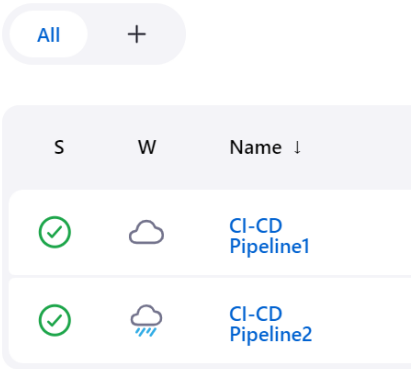
Script:

```
pipeline {
    agent { label 'prod'}
    stages {
        stage ('cloning Git-repo') {
            steps {
                git branch: 'main',
                url: '
https://github.com/Ravivarman16/DevOps-Capstone-project-
2.git '
            }
        }
        stage ('production by k8s') {
            steps {
                sh 'kubectl apply -f production.yml'
```

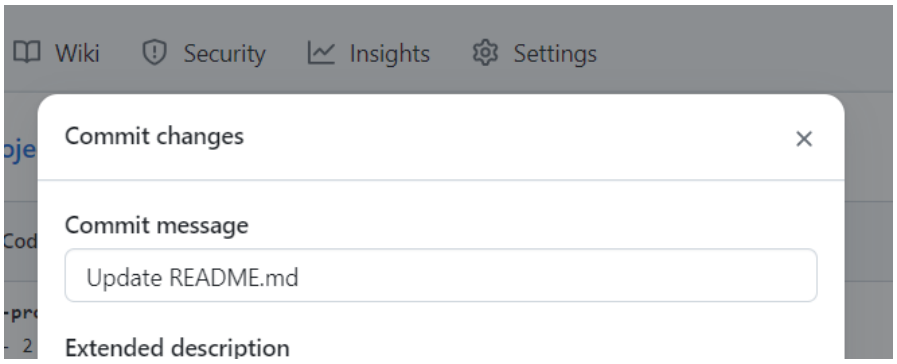
```
                }
            }
        }
}
```

After entering the script **click apply and save.**

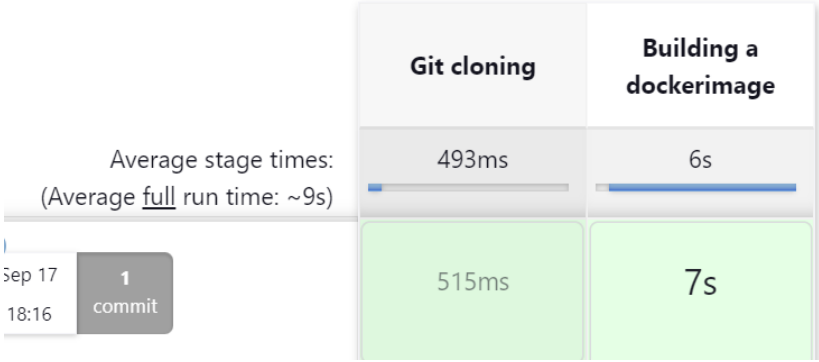
Here we can see **two jobs** were created.



- Now we need to **test the pipeline**, I am making a small change on the **GitHub repository** and commiting it.

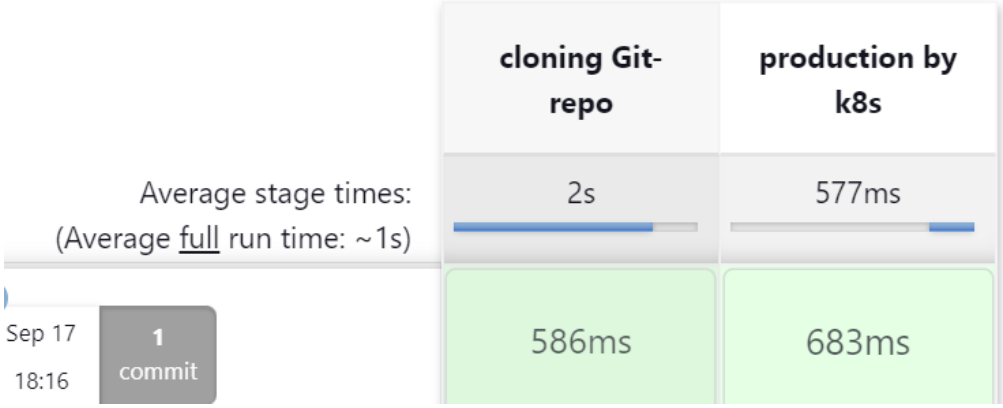


- We can able to see the **1st Pipeline job had been triggered automatically** once I made the changes on the GitHub.



Output of 1st job which is running on the test_server:

- Then we can able to see once **1st is executed successfully automatically the 2nd job is triggered.**



Output from the production server by k8s:

- **Job details on test_server on command line:**

```
root@ip-172-31-24-246:/home/ubuntu# cd jenkins/workspace/
root@ip-172-31-24-246:/home/ubuntu/jenkins/workspace# ls
'CI-CD Pipeline'  'CI-CD Pipeline1'  'CI-CD Pipeline1@tmp'  'CI-CD Pipeline@tmp'
root@ip-172-31-24-246:/home/ubuntu/jenkins/workspace#
```

- **Job details on prod server on command line:**

```
root@ip-172-31-29-177:/home/ubuntu# cd jenkins/
root@ip-172-31-29-177:/home/ubuntu/jenkins# ls
caches  remoting  remoting.jar  workspace
root@ip-172-31-29-177:/home/ubuntu/jenkins# cd workspace/
root@ip-172-31-29-177:/home/ubuntu/jenkins/workspace# ls
'CI-CD Pipeline2'  'CI-CD Pipeline2@tmp'
root@ip-172-31-29-177:/home/ubuntu/jenkins/workspace# kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
dp-pod-6b7f7c476d-f5r9n   1/1     Running   0          13m
dp-pod-6b7f7c476d-gqmgg   1/1     Running   0          13m
dp-pod-6b7f7c476d-jbqpd   1/1     Running   0          13m
root@ip-172-31-29-177:/home/ubuntu/jenkins/workspace# kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP          90m
svc          NodePort    10.107.162.189   <none>        8080:30008/TCP   13m
root@ip-172-31-29-177:/home/ubuntu/jenkins/workspace# kubectl get svc -o wide
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE   SELECTOR
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP          90m   <none>
svc          NodePort    10.107.162.189   <none>        8080:30008/TCP   14m   app=live-app
root@ip-172-31-29-177:/home/ubuntu/jenkins/workspace# kubectl get pods -o wide
NAME                      READY   STATUS    RESTARTS   AGE   IP          NODE
NESS GATES
dp-pod-6b7f7c476d-f5r9n   1/1     Running   0          14m   10.32.0.5   ip-172-31-17-153
>
dp-pod-6b7f7c476d-gqmgg   1/1     Running   0          14m   10.32.0.4   ip-172-31-17-153
>
dp-pod-6b7f7c476d-jbqpd   1/1     Running   0          14m   10.32.0.6   ip-172-31-17-153
>
root@ip-172-31-29-177:/home/ubuntu/jenkins/workspace#
```

CI/CD Pipeline had setup by using devops tools.

GitHub repository for this project:
https://github.com/Ravivarman16/DevOps-Capstone-project-2.git

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