DEVOPS CAPSTONE-PROJECT 02

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 implement a DevOps life cycle, 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 life-cycle:

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 Docker file, The Dockerfile should be built every time if there is a push to Git-Hub. 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 accomplishes 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

DEVOPS CAPSTONE-PROJECT 02

Worker5: Docker, Kubernetes

SOLUTIONS: **PREREQUISITES**:

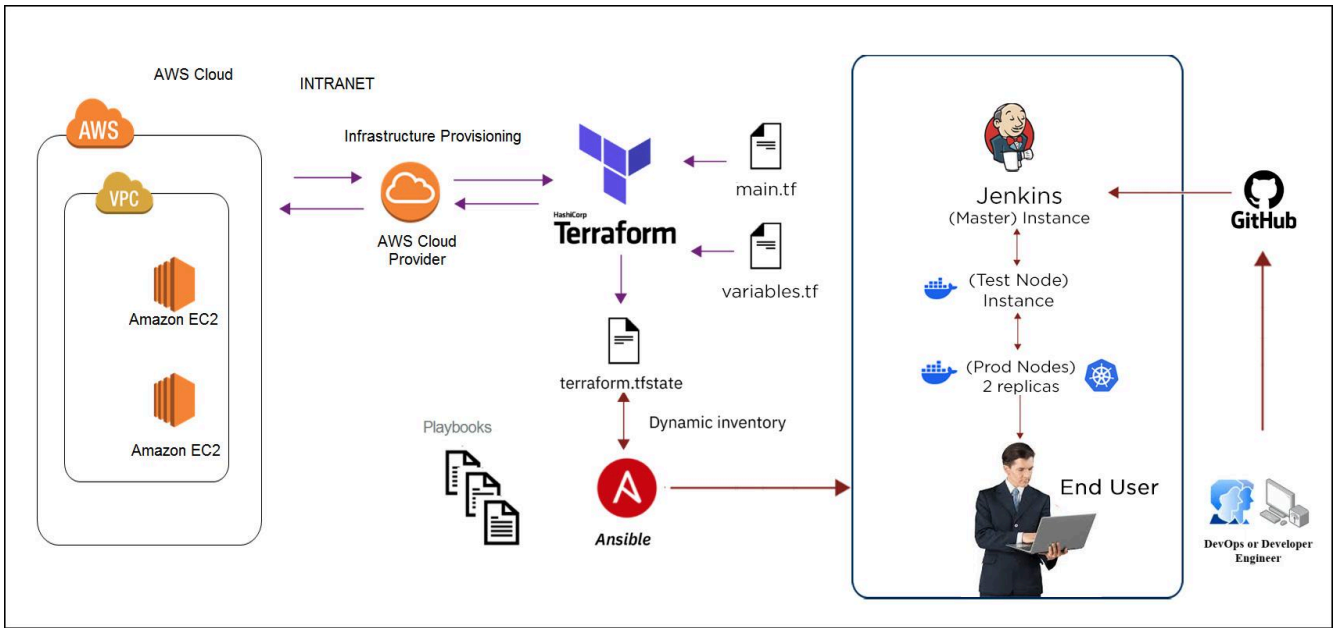
Cloud : AWS Cloud Server **|** **AWS** EC2 instance **|** **Operating System :** Ubuntu 22.04
Network : AWS VPC **| Security :** AWS IAM

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**

DEVOPS CAPSTONE-PROJECT 02

Requirements



Solution - Workflow Diagram

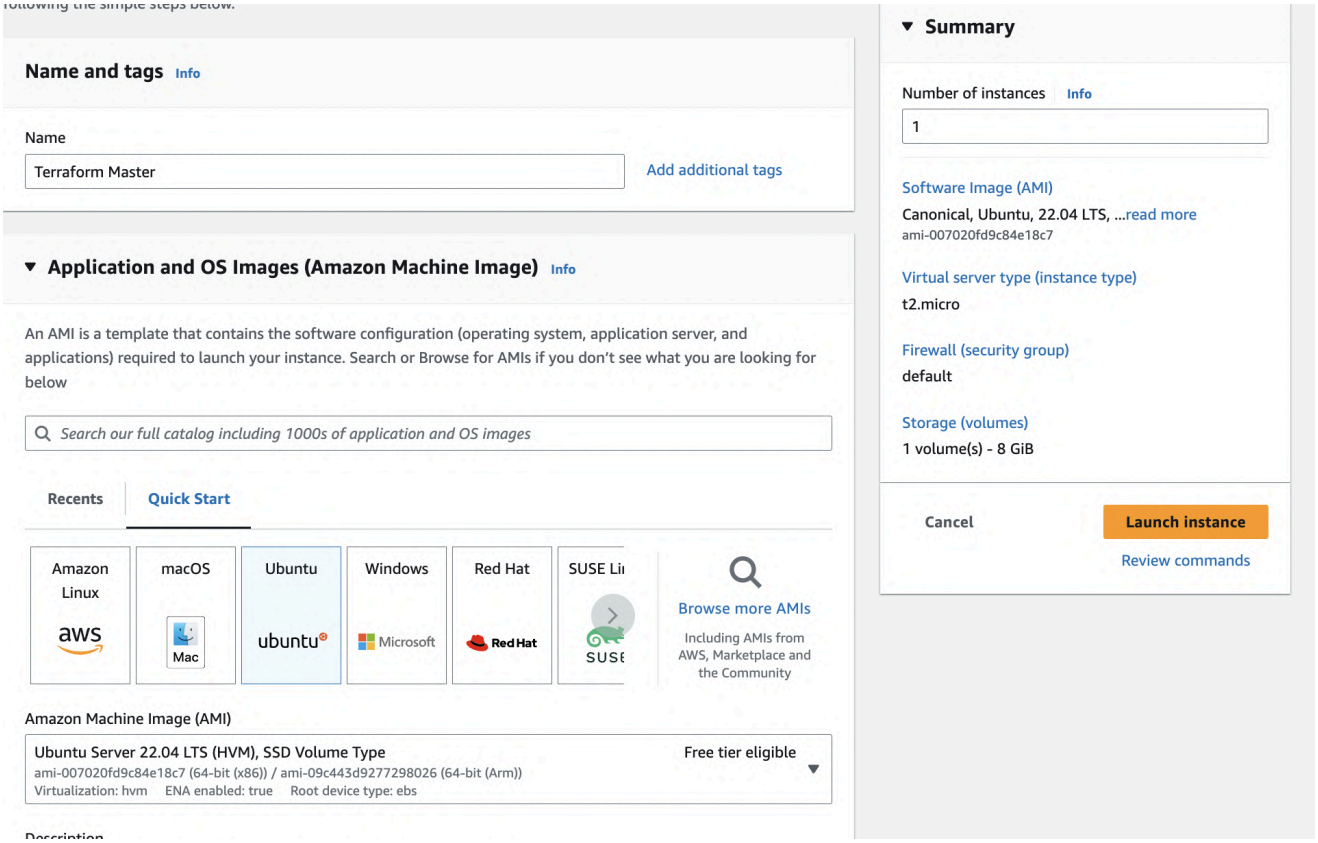
Step-1:

Creating a terraform master instance:

Login into AWS management console and go to EC2 management console for launching terraform master instance.

Here the instance specifications according to our spec needs. I am selecting OS=Ubuntu 22.04 LTS, instance type= t2.micro.

DEVOPS CAPSTONE-PROJECT 02



The instance has been created, now we need to connect the instance and download the terraform package on it in order 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.

nano terraform.sh

```
#!/bin/bash
apt-get update


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
```

To run the vi terraform.sh file using the command

sudo bash terraform.sh

Terraform installed and version can be checked below.

```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@ip-172-31-46-17:/home/ubuntu# terraform --version
Terraform v1.8.2
on linux_amd64
root@ip-172-31-46-17:/home/ubuntu#
```

i-0253c4bc88e1c4856 (Terraform Master)

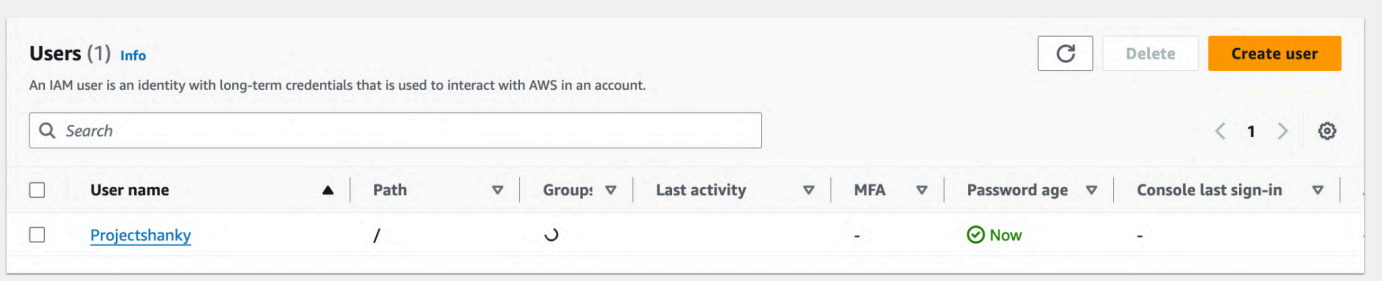
PublicIPs: 3.111.188.238 PrivateIPs: 172.31.46.17

Now, Creating a IAM user for terraform to access our cloud. Here I am giving

2 permissions — EC2 full access and VPC full access.

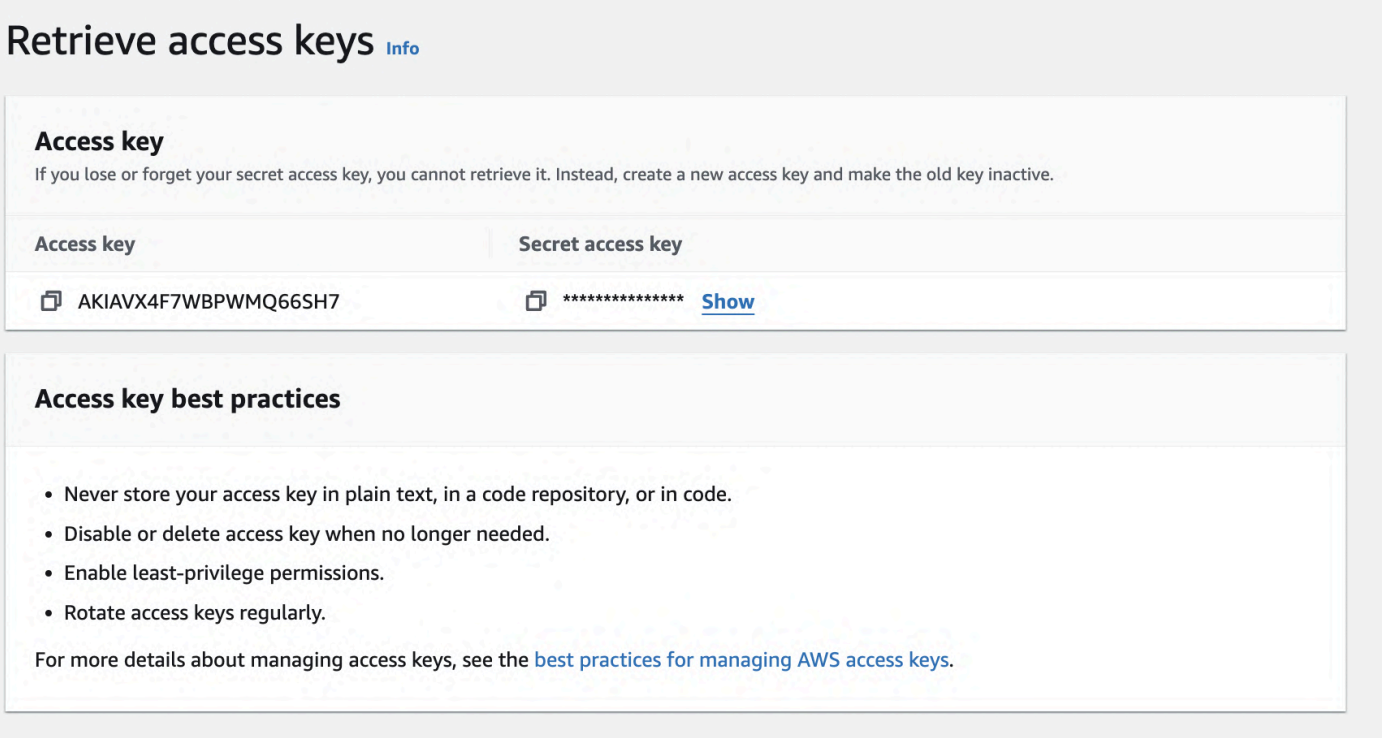
Click->Iam user->create user ->user details

DEVOPS CAPSTONE-PROJECT 02



Note the username & password

Create access key



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

```
  GNU nano 6.2
variable "region" {
  description = "The region for creating the resources"
  default     = "ap-south-1"
}
variable "ak" {
  description = "The access key for the IAM user"
  default     = "wwwAKIAVX4F7WBPUNMTLRNP"
}
variable "sk" {
  description = "The secret key for the IAM user"
  default     = "weremrkjQnGQgseG7Y7S5YUg6EzPhV1BMeCa4Nw2zXyJ"
}
variable "ami" {
  description = "The AMI image of the machine"
  default     = "ami-007020fd9c84e18c7"
}
variable "keypair" {
  description = "The Keypair e of the machine"
  default     = "Capstone.pem"
}
variable "shanky" {
  description = "this is a default Name"
  default     = "shanky"
}
```

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.

1 for ansible master **1** Jenkins master **1** test server and **2** prods

nano main.tf

```
#Provider resource details:
provider "aws" {
  region     = var.reg
  access_key = var.ak
  secret_key = var.sk
}

#Creating the Vpc:
resource "aws_vpc" "project2vpc" {
  cidr_block       = "10.0.0.0/16"
  instance_tenancy = "default"

  tags = {
    Name = "project2vpc"
  }
}
#Creating the internet gateway:
resource "aws_internet_gateway" "project2_igw" {
  vpc_id = aws_vpc.project2vpc.id

  tags = {
    Name = "project2_igw"
  }
}
#creating the subnet2:
resource "aws_subnet" "project2_subnet" {
  vpc_id     = aws_vpc.project2vpc.id
  cidr_block = "10.0.1.0/24"

  tags = {
    Name = "project2_subnet"
  }
}
#creating the route table:
resource "aws_route_table" "project2_route_table" {
  vpc_id = aws_vpc.project2vpc.id
  tags = {
    Name = "project2_route_table"
  }
}
#associating route internet gateway in route table:
resource "aws_route" "project2_routing" {
  route_table_id         = aws_route_table.project2_route_table.id
  destination_cidr_block = "0.0.0.0/0" # Route all traffic to the Internet
Gateway
  gateway_id             = aws_internet_gateway.project2_igw.id
}
#creating the subnet association with the route table:
resource "aws_route_table_association" "subnet_association" {
```

```
  subnet_id      = aws_subnet.project2_subnet.id
  route_table_id = aws_route_table.project2_route_table.id
}
#creating the security group:
resource "aws_security_group" "project2_sc" {
  name        = "project2_sc"
  description = "security group for AWS EC2 instances"
  vpc_id      = aws_vpc.project2vpc.id
  # Ingress rules (inbound traffic)
  # Allow SSH (port 22) from anywhere
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Allow HTTP (port 80) from anywhere
  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  # Allow HTTPS (port 443) from anywhere
  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  # Allow CUSTOM TCP (port 8080) from anywhere
  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Allow CUSTOM TCP (port 30008) for NodePort service for accessing from
anywhere
  ingress {
    from_port   = 30008
    to_port     = 30008
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Egress rules (outbound traffic)
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
```

DEVOPS CAPSTONE-PROJECT 02

```
  }

  tags = {
    Name = "project2_sc"
  }
}
#creating the instance:
resource "aws_instance" "server" {
  count                  = "4"
  ami                    = var.ami_id
  instance_type          = "t2.micro"
  subnet_id              = aws_subnet.project2_subnet.id
  key_name               = var.key
  vpc_security_group_ids = [aws_security_group.project2_sc.id]
  associate_public_ip_address = true
  tags = {
    Name = "Server - ${count.index}"
  }
}
```

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

```
Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.46.0...
- Installed hashicorp/aws v5.46.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
root@ip-172-31-46-17:/home/ubuntu#
```

After that we need to format the terraform files with the help of the command

DEVOPS CAPSTONE-PROJECT 02

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

```
root@ip-172-31-46-17:/home/ubuntu# terraform validate
Success! The configuration is valid.

root@ip-172-31-46-17:/home/ubuntu#
```

i-0253c4bc88e1c4856 (Terraform Master)

PublicIPs: 3.111.188.238 PrivateIPs: 172.31.46.17

`terraform plan`

```
  }

  # aws_instance.project1_ec2[3] will be created
  + resource "aws_instance" "project1_ec2" {
      + ami                                  = "ami-007020fd9c84e18c7"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = true
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = "Capstone.pem"
      + monitoring                           = (known after apply)
      + outpost_arn                          = (known after apply)
      + password_data                        = (known after apply)
      + placement_group                      = (known after apply)
      + placement_partition_number           = (known after apply)
      + primary_network_interface_id         = (known after apply)
      + private_dns                          = (known after apply)
      + private_ip                           = (known after apply)
      + public_dns                           = (known after apply)
```

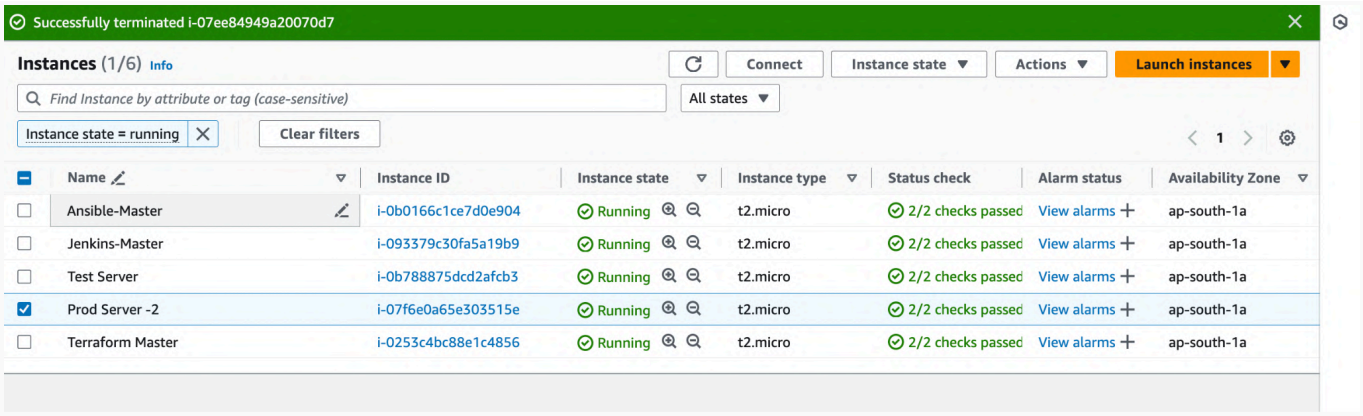
DEVOPS CAPSTONE-PROJECT 02

Created infrastructure. Resources had been provisioned by terraform now we need to check it on the Ec2 console. Resources has been provisioned successfully

```
Plan: 4 to add, 0 to change, 0 to destroy.
aws_instance.ec2server[3]: Creating...
aws_instance.ec2server[0]: Creating...
aws_instance.ec2server[2]: Creating...
aws_instance.ec2server[1]: Creating...
aws_instance.ec2server[3]: Still creating... [10s elapsed]
aws_instance.ec2server[0]: Still creating... [10s elapsed]
aws_instance.ec2server[2]: Still creating... [10s elapsed]
aws_instance.ec2server[1]: Still creating... [10s elapsed]
aws_instance.ec2server[3]: Still creating... [20s elapsed]
aws_instance.ec2server[0]: Still creating... [20s elapsed]
aws_instance.ec2server[2]: Still creating... [20s elapsed]
aws_instance.ec2server[1]: Still creating... [20s elapsed]
aws_instance.ec2server[2]: Creation complete after 21s [id=i-00ceeb1f1a6e2f024]
aws_instance.ec2server[0]: Creation complete after 21s [id=i-07e7d23fe3b41a736]
aws_instance.ec2server[3]: Still creating... [30s elapsed]
aws_instance.ec2server[1]: Still creating... [30s elapsed]
aws_instance.ec2server[1]: Creation complete after 31s [id=i-00b4d5f1de6ff6904]
aws_instance.ec2server[3]: Creation complete after 31s [id=i-075626c03fdfb02fe]

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.
ubuntu@ip-172-31-9-251:~$
```

Now i will rename the resources the instances according to our specifications like Ansible master, Jenkins master, Jenkins test, Jenkins prod.



Connect all instances by EC2 instance connect for ansible cluster formation to install the necessary software on each instance.

Configuration Management by Ansible

Other than ansible master nodes we need to make a small change on other servers for ssh connection in order 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 now we connect the Ansible master instance using Ec2 instance connect. For that I am creating the script file with the script to install ansible.

nano ansible.sh

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

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

```
ansible --version
```

DEVOPS CAPSTONE-PROJECT 02

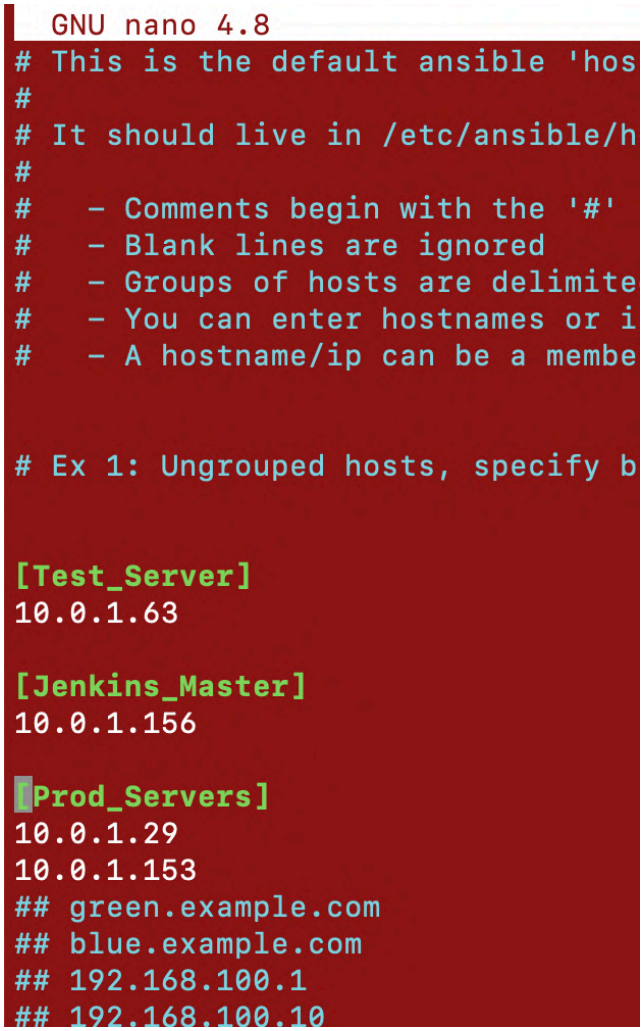
Ansible has been successfully installed. Now we need to setup ansible cluster. For this, we need to make slight changes on the slave nodes to connect with the master node.

Changes on Slave Nodes:

- Change the password for the root account.
- Go the the directory and make some changes:
- Directory: nano /etc/ssh/sshd_config
    - remove the hash near the port number 22
    - under authentication change the PermitRootLogin into yes
    - under password authentication make it has yes
- Finally restart the SSHD service.

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

DEVOPS CAPSTONE-PROJECT 02

1. 

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

DEVOPS CAPSTONE-PROJECT 02

```
root@ip-10-0-1-33:~# 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:dpw5S6AstBYTf/uBzbVyXJZ8X8JLxr64BnqU+694A0c root@ip-10-0-1-33
The key's randomart image is:
+---[RSA 3072]----+
|    .            |
|     o       + . |
|    + . o    . @ o|
|   . = o B  =EB +o|
|    + o S  %o+ o .|
|   . . .  +=*.. . |
|         oo=. .   |
|        . o.+.    |
|         ..+++.   |
+----[SHA256]-----+
root@ip-10-0-1-33:~# sudo nano /etc/ansible/hosts
root@ip-10-0-1-33:~#
```

i-07e7d23fe3b41a736 (Ansible-Master)

PublicIPs: 3.110.168.35 PrivateIPs: 10.0.1.33

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

```
root@ip-10-0-1-18:/home/ubuntu# ansible all -m ping
The authenticity of host '10.0.1.63 (10.0.1.63)' can't be established.
ECDSA key fingerprint is SHA256:Uak0o86DmOCvWbvJQ5NrdSC2Aj4J0IOZfWdkBUapE3s.
The authenticity of host '10.0.1.156 (10.0.1.156)' can't be established.
ECDSA key fingerprint is SHA256:Sf6bQPRi+9LsZ9gU/fwSWmg0z+sTFQVSW5dKz/SPXQA.
The authenticity of host '10.0.1.29 (10.0.1.29)' can't be established.
ECDSA key fingerprint is SHA256:RyorZ1oDWZsnxSiYh/b3irQ7pDSFwZeefT9lpuMXYAw.
The authenticity of host '10.0.1.153 (10.0.1.153)' can't be established.
ECDSA key fingerprint is SHA256:6YrcmiDB12KW3N+kZchKyUw4IFmX8sn8aw3qrFfi0ng.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
10.0.1.63 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
yes
10.0.1.156 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
yes
10.0.1.29 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
yes
10.0.1.153 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
root@ip-10-0-1-18:/home/ubuntu#
```

DEVOPS CAPSTONE-PROJECT 02

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

Yaml file for executing the ansible playbook contains.

nano package.yml

```
- name: To install java and jenkins on jenkins master
  hosts: Jenkins_Master
  become: true
  tasks:
    - name: installing the packages on jenkins-master
      script: jenkins.sh
    - name: updating the task work
      debug:
        msg: "packages had been installed successfully on jenkins-master"
#on test-server node:
- name: To install docker on test-server
  hosts: Test_Server
  become: true
  tasks:
    - name: installing the packages on test-server
      script: test.sh
    - name: updating the task work
      debug:
        msg: "packages had been installed successfully on test-server"
#on prod-server node:
- name: To install k8s on prod-server
  hosts: Prod_Servers
  become: true
  tasks:
    - name: installing the packages on prod-server
      script: prod.sh
    - name: updating the task work
      debug:
        msg: "packages had been installed successfully on prod-server"
#the end:
```

DEVOPS CAPSTONE-PROJECT 02

nano test.sh

```
#!/bin/bash

#updating the os:
apt-get update

#to install java:
apt-get install -y openjdk-11-jre

#installing docker:
apt-get install -y docker.io

#enable docker service:
systemctl enable docker

#starting the docker service:
systemctl start docker

#checking the docker version:
docker -v
```

nano jenkins.sh

```
#!/bin/bash

#updating the os:
apt-get update

#installing the java11:
apt-get install -y openjdk-11-jre

#installing jenkins:
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null

apt-get update
apt-get install -y jenkins

#checking the version:
jenkins --version
```

nano prod.sh

```
#!/bin/bash

#updating the os:
apt-get update

#to install java:
apt-get install -y openjdk-11-jre

#installing docker:
apt-get install -y docker.io

#enable docker service:
systemctl enable docker

#starting the docker service:
systemctl start docker

#installing the kubernetes on it:
apt-get update

apt-get install -y curl apt-transport-https ca-certificates
software-properties-common
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add
-
sudo add-apt-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"
sudo swapoff -a

apt-get update
apt-get install -y kubelet kubeadm kubectl
systemctl enable kubelet.service
systemctl start kubelet
```

Now check the syntax of yml file with the command

The syntax of the yml file can be verified by **ansible playbook package-ym –syntax-check**

Now we need to execute this file for ansible to install the necessary packages on slave nodes.

`ansible-playbook package.yml`

DEVOPS CAPSTONE-PROJECT 02

```
TASK [updating the task work] ***********************************************************************************************************************************
ok: [10.0.1.156] => {
    "msg": "packages had been installed successfully on jenkins-master"
}

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

TASK [Gathering Facts] ******************************************************************************************************************************************
ok: [10.0.1.63]

TASK [installing the packages on test-server] *******************************************************************************************************************
changed: [10.0.1.63]

TASK [updating the task work] ***********************************************************************************************************************************
ok: [10.0.1.63] => {
    "msg": "packages had been installed successfully on test-server"
}

PLAY [To install k8s on prod-server] ****************************************************************************************************************************

TASK [Gathering Facts] ******************************************************************************************************************************************
ok: [10.0.1.29]
ok: [10.0.1.153]

TASK [installing the packages on prod-server] *******************************************************************************************************************
changed: [10.0.1.29]
changed: [10.0.1.153]

TASK [updating the task work] ***********************************************************************************************************************************
ok: [10.0.1.29] => {
    "msg": "packages had been installed successfully on prod-server"
}
ok: [10.0.1.153] => {
    "msg": "packages had been installed successfully on prod-server"
}

PLAY RECAP ******************************************************************************************************************************************************
10.0.1.153                 : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
10.0.1.156                 : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
10.0.1.29                  : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
10.0.1.63                  : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

root@ip-10-0-1-18:/home/ubuntu#
```

On Jenkins master Slave Node

```
root@ip-10-0-1-156:/home/ubuntu# java --version
openjdk 11.0.22 2024-01-16
OpenJDK Runtime Environment (build 11.0.22+7-post-Ubuntu-0ubuntu220.04.1)
OpenJDK 64-Bit Server VM (build 11.0.22+7-post-Ubuntu-0ubuntu220.04.1, mixed mode, sharing)
root@ip-10-0-1-156:/home/ubuntu# jenkins --v
Running from: /usr/share/java/jenkins.war
webroot: /root/.jenkins/war
Exception in thread "main" java.lang.IllegalArgumentException: Unrecognized option: --v
        at winstone.cmdline.CmdLineParser.parse(CmdLineParser.java:53)
        at winstone.Launcher.getArgsFromCommandLine(Launcher.java:506)
        at winstone.Launcher.main(Launcher.java:468)
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.jav
        at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessor
        at java.base/java.lang.reflect.Method.invoke(Method.java:566)
        at executable.Main.main(Main.java:351)
root@ip-10-0-1-156:/home/ubuntu# jenkins --version
2.452.1
root@ip-10-0-1-156:/home/ubuntu#
```

DEVOPS CAPSTONE-PROJECT 02

```
0-0-1-63:/home/ubuntu# docker -v
rsion 24.0.5, build 24.0.5-0ubuntu1~20.04.1
0-0-1-63:/home/ubuntu# systemctl status docker
service - Docker Application Container Engine
ed: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabl
ve: active (running) since Mon 2024-05-20 06:57:05 UTC; 32s ago
By: ● docker.socket
cs: https://docs.docker.com
ID: 16778 (dockerd)
ks: 7
ry: 34.6M
up: /system.slice/docker.service
    └─16778 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd
```

On 1st Prod Server Node:

```
root@ip-10-0-1-153:/home/ubuntu# docker -v
Docker version 24.0.5, build 24.0.5-0ubuntu1~20.04.1
root@ip-10-0-1-153:/home/ubuntu# systemctl status dokcer
Unit dokcer.service could not be found.
root@ip-10-0-1-153:/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 Mon 2024-05-20 06:58:56 UTC; 56s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 16752 (dockerd)
      Tasks: 7
     Memory: 25.7M
     CGroup: /system.slice/docker.service
             └─16752 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

On 2nd Prod Server Node:

```
root@ip-10-0-1-29:/home/ubuntu# docker -v
Docker version 24.0.5, build 24.0.5-0ubuntu1~20.04.1
root@ip-10-0-1-29:/home/ubuntu# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"30", GitVersio
on:"go1.22.2", Compiler:"gc", Platform:"linux/amd64"}
root@ip-10-0-1-29:/home/ubuntu#
```

i-0ce530d63c180451c (Prod Server)

PublicIPs: 13.233.113.223 PrivateIPs: 10.0.1.29

So far, Infrastructure provision and Software configuration has been done.

DEVOPS CAPSTONE-PROJECT 02

Now we need to two things to setup pipeline

1) Jenkins Master setup pipeline
2) Kubernetes cluster setup for production

First, we will set up the Kubernetes cluster setup in the prod-server node.

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.

```
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet c
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

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
```

```
root@ip-10-0-1-29:/home/ubuntu# kubectl get nodes
NAME           STATUS   ROLES           AGE   VERSION
ip-10-0-1-29   Ready    control-plane   13m   v1.30.1
root@ip-10-0-1-29:/home/ubuntu#
```

i-0ce530d63c180451c (Prod Server)

PublicIPs: 13.233.113.223 PrivateIPs: 10.0.1.29

These steps to be done only on the main production server.

DEVOPS CAPSTONE-PROJECT 02

Copy join command on second prod server node “kubeadm join 10.0.1.29:6443 --token jfupwb.5gua4dqkigte7gb4 \ --discovery-token-ca-cert-hash sha256:01195481352e57b5d75e492c92d1f80ee880ec9bcac5335085c41706bbd831cb ”

```
ubuntu@ip-10-0-1-153:~$ sudo su
root@ip-10-0-1-153:/home/ubuntu# sudo kubeadm join 10.0.1.29:6443 --token ko3ls
9bcac5335085c41706bbd831cb
[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
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/k
[kubelet-start] Starting the kubelet
[kubelet-check] Waiting for a healthy kubelet. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 1.502537262s
[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-10-0-1-153:/home/ubuntu#
```

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/weavedaemonset-k8s.yaml

```
ip-10-0-1-29   Ready    control-plane   24m   v1.30.1
root@ip-10-0-1-29:/home/ubuntu/.kube# kubeadm token create --print-join-com
kubeadm join 10.0.1.29:6443 --token nl9t92.6ucwnrhbm269uxft --discovery-tok
root@ip-10-0-1-29:/home/ubuntu/.kube# kubectl get nodes
NAME            STATUS   ROLES           AGE   VERSION
ip-10-0-1-153   Ready    <none>          14s   v1.30.1
ip-10-0-1-29    Ready    control-plane   27m   v1.30.1
root@ip-10-0-1-29:/home/ubuntu/.kube# cd ../
root@ip-10-0-1-29:/home/ubuntu# kubectl apply -f https://github.com/weavewo
serviceaccount/weave-net unchanged
clusterrole.rbac.authorization.k8s.io/weave-net unchanged
clusterrolebinding.rbac.authorization.k8s.io/weave-net unchanged
role.rbac.authorization.k8s.io/weave-net unchanged
rolebinding.rbac.authorization.k8s.io/weave-net unchanged
daemonset.apps/weave-net configured
root@ip-10-0-1-29:/home/ubuntu#
```

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

DEVOPS CAPSTONE-PROJECT 02

```
root@ip-10-0-1-29:/home/ubuntu/.kube# kubectl get nodes
NAME            STATUS   ROLES           AGE   VERSION
ip-10-0-1-153   Ready    <none>          14s   v1.30.1
ip-10-0-1-29    Ready    control-plane   27m   v1.30.1
root@ip-10-0-1-29:/home/ubuntu/.kube#
```

i-0ce530d63c180451c (Prod Server)

PublicIPs: 13.233.113.223 PrivateIPs: 10.0.1.29

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

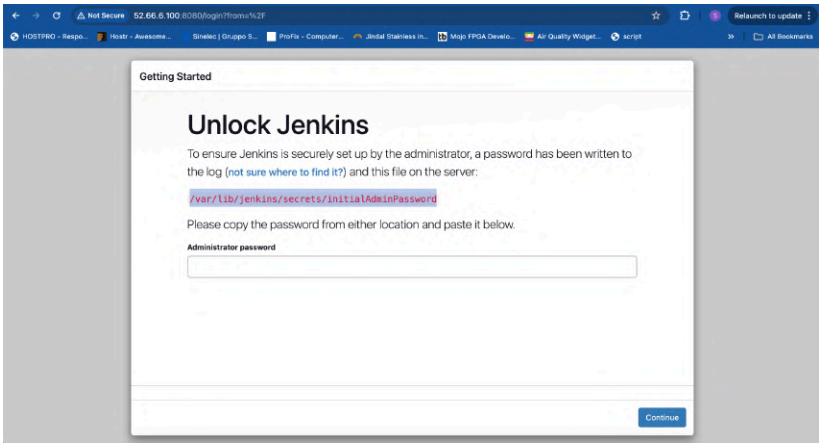
Production server setup has been done successfully.

****

Jenkins Master – slave setup for pipeline execution:

Now copy the jenkins -server <public ip address>:8080 in browser

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

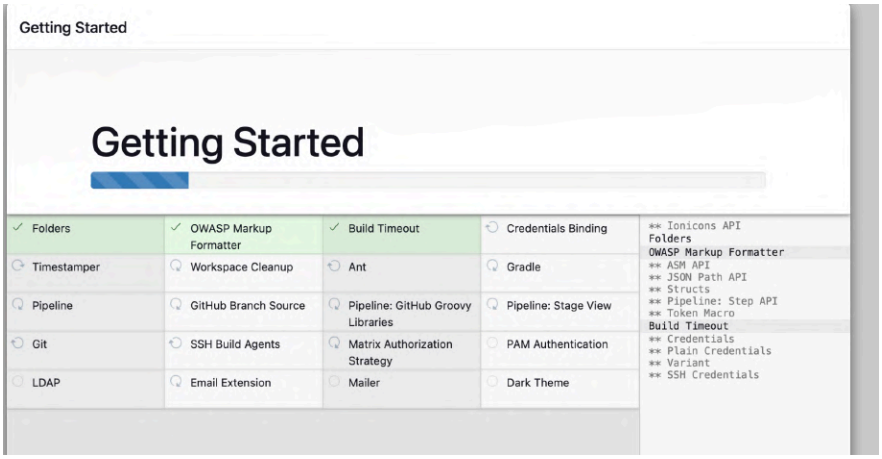


Copy and paste the password and proceed further:

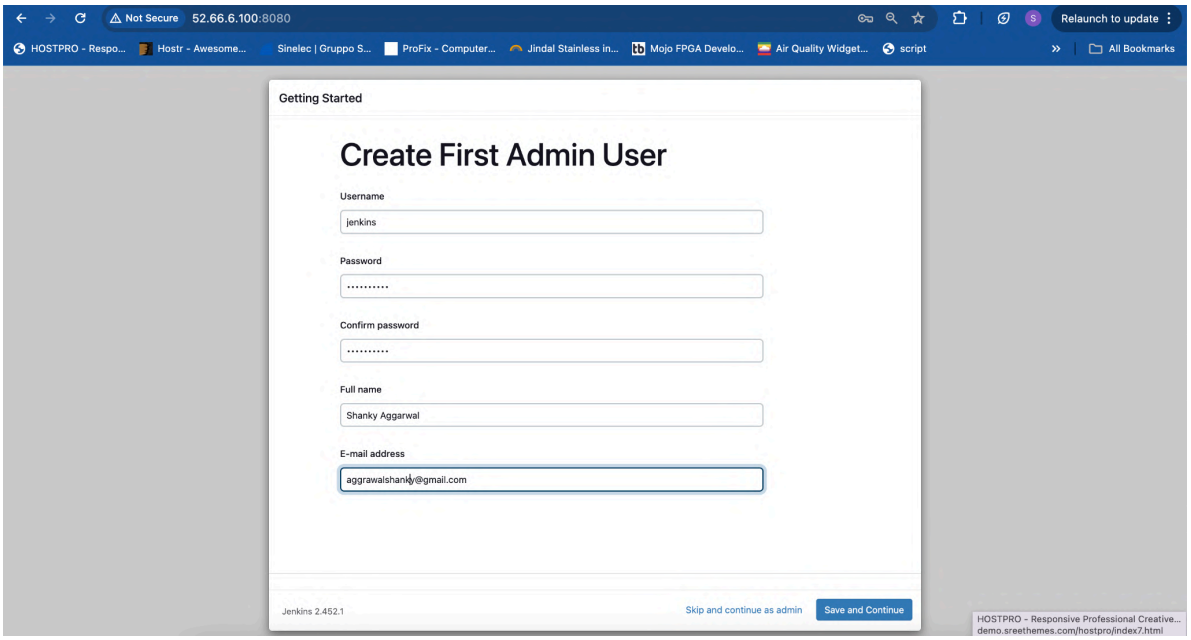
```
root@ip-10-0-1-156:/home/ubuntu# cat /var/lib/jenkins/secrets/initialAdminPassword
e6a0f733b99843a29c2061f33a592799
root@ip-10-0-1-156:/home/ubuntu#
```

i-02aeb75f10905e352 (Jenkins-Server)

DEVOPS CAPSTONE-PROJECT 02

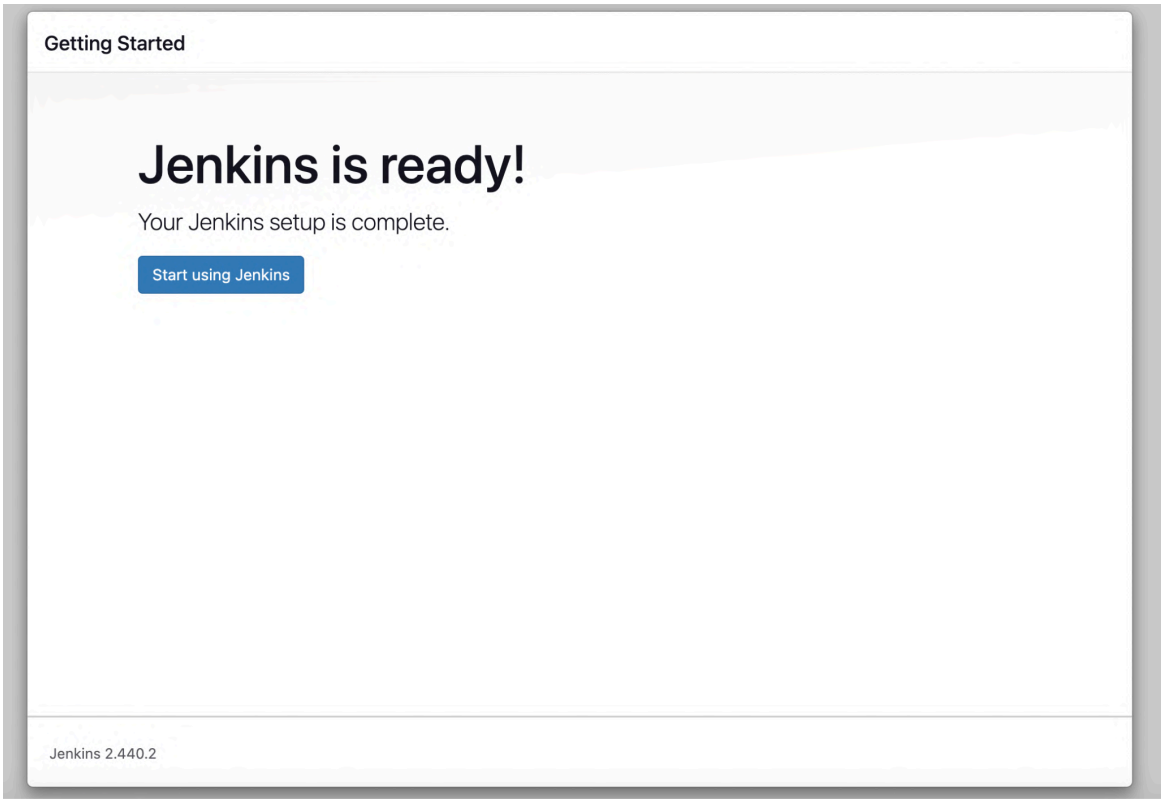


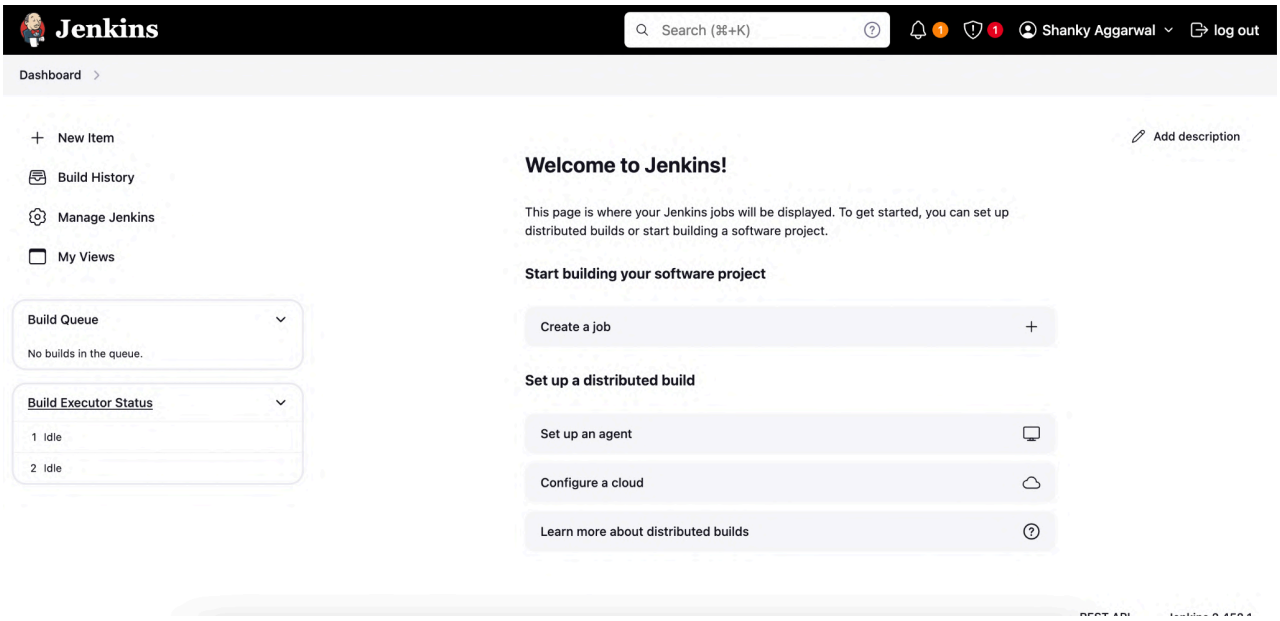
Once the plugins are installed. Then setup the username and password credentials for using Jenkins dashboard, save and continue.



DEVOPS CAPSTONE-PROJECT 02

Now the Jenkins master is ready.





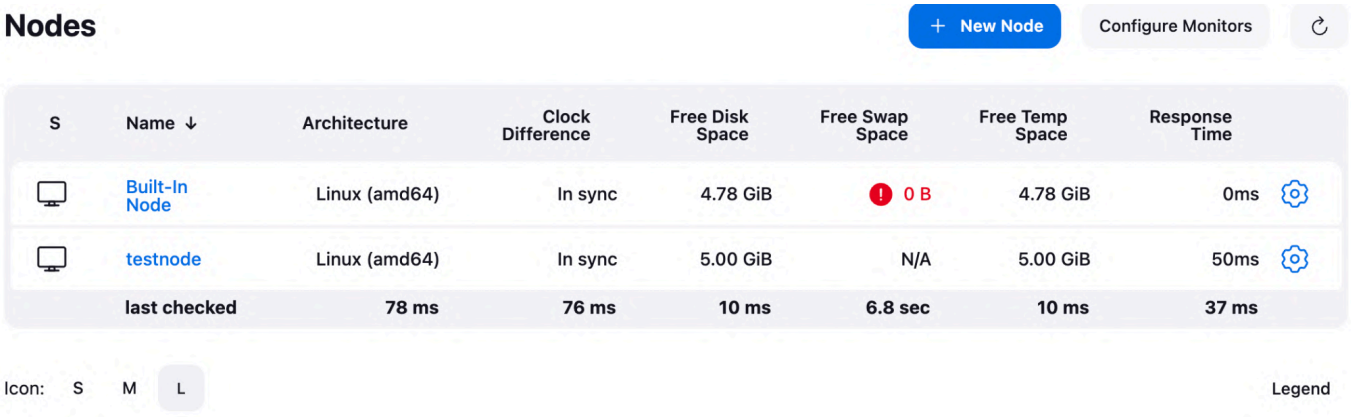
DEVOPS CAPSTONE-PROJECT 02

Now we need to setup master-slave connection:

For master node setup:

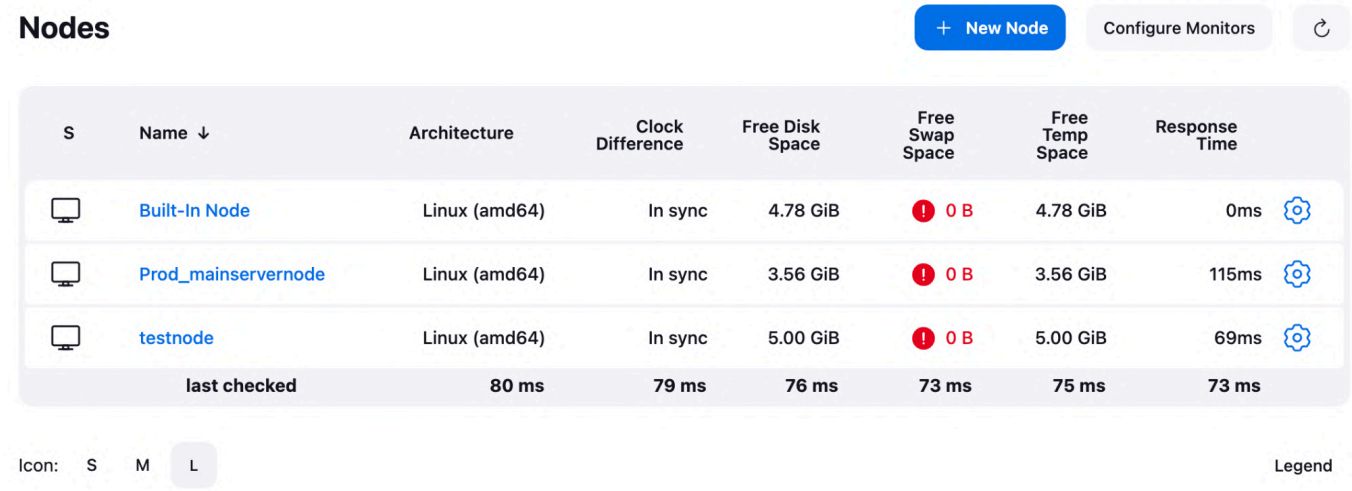
click manage Jenkins, under manage Jenkins click Nodes.

1. On the left-hand side, we are able to see the manage Jenkins option, click that. On the managed Jenkins , we are able to see the nodes, click that.
2. Click the new node on the left side: give the node a name and select a permanent agent.
3. 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



DEVOPS CAPSTONE-PROJECT 02

Like this we need to add the prod main server to Jenkins master in order Jenkins to control it. Two nodes were added.

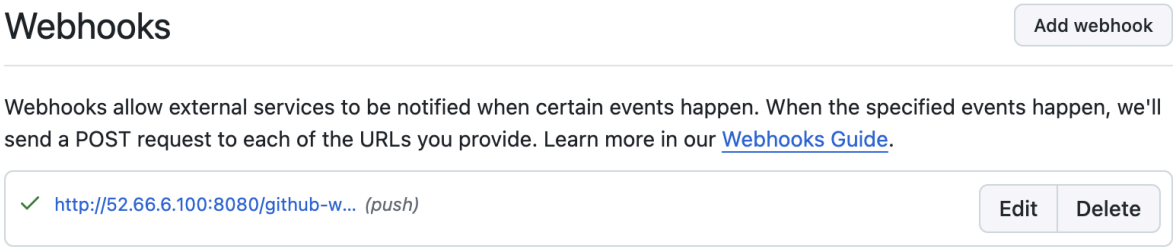


****

Now we need to create the jobs for pipeline execution:

• First, we need to set up the GitHub hook trigger for automatically running the Jenkins pipeline.

• On the GitHub repository settings, we can see webhooks on the right hand side:



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

Creating Jenkins pipeline jobs: Requirements: 2 jobs

DEVOPS CAPSTONE-PROJECT 02

• 1 st 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.
• 2 nd job will run on production server, where k8s will take deployment and container orchestration.

1 ST Pipeline Job:

Creating a new job, selecting pipeline type, giving the description.
1 ST 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:

```
pipeline {
    agent { label 'testnode' }

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

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

2nd Pipeline Job: Creating a new job, selecting pipeline type, giving the description.

```
pipeline {
   agent { label 'prodnode'}
```

DEVOPS CAPSTONE-PROJECT 02

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

Output from the test server.

DEVOPS CAPSTONE-PROJECT 02

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

Output from the production server by k8

Job details on test server on command line:

DEVOPS CAPSTONE-PROJECT 02

Job details on prod server on command line:

```
^Cubuntu@ip-10-0-1-29:~$ kubectl get pods
NAME                      READY   STATUS        RESTARTS   AGE
dp-pod-84d4d88744-g454w   1/1     Running       0          22s
dp-pod-84d4d88744-nb826   1/1     Running       0          22s
dp-pod-84d4d88744-wqwhr   1/1     Running       0          22s
dp-pod-84d4d88744-wwfwn   1/1     Terminating   0          81m
dp-pod-84d4d88744-zbvtg   1/1     Terminating   0          81m
dp-pod-84d4d88744-zgxzw   1/1     Terminating   0          81m
ubuntu@ip-10-0-1-29:~$ kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
dp-pod-84d4d88744-g454w   1/1     Running   0          4m57s
dp-pod-84d4d88744-nb826   1/1     Running   0          4m57s
dp-pod-84d4d88744-wqwhr   1/1     Running   0          4m57s
ubuntu@ip-10-0-1-29:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP          4h22m
svc          NodePort    10.108.195.174   <none>        8080:30008/TCP   106m
ubuntu@ip-10-0-1-29:~$
```

CI/CD Pipeline has been setup by using devops tools.