Project- DevOps

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

Following are the specifications of the lifecycle:

1. Git Workflow has to be implemented

2. Code Build should automatically be triggered once a commit is made to master branch or develop branch. If commit is made to master branch, test and push to prod If commit is made to develop branch, just test the product, do not push to prod

3. The Code should be containerized with the help of a Dockerfile. The Dockerfile should be built every time there is a push to Git-Hub. Use the following pre-built container for your application:
hshar/webapp
The code should reside in '/var/www/html'

4. The above tasks should be defined in a Jenkins Pipeline, with the following Jobs Job
- **1 - Building Website Job**
- **2 - Testing Website Job**
- **3 - Push to Production**

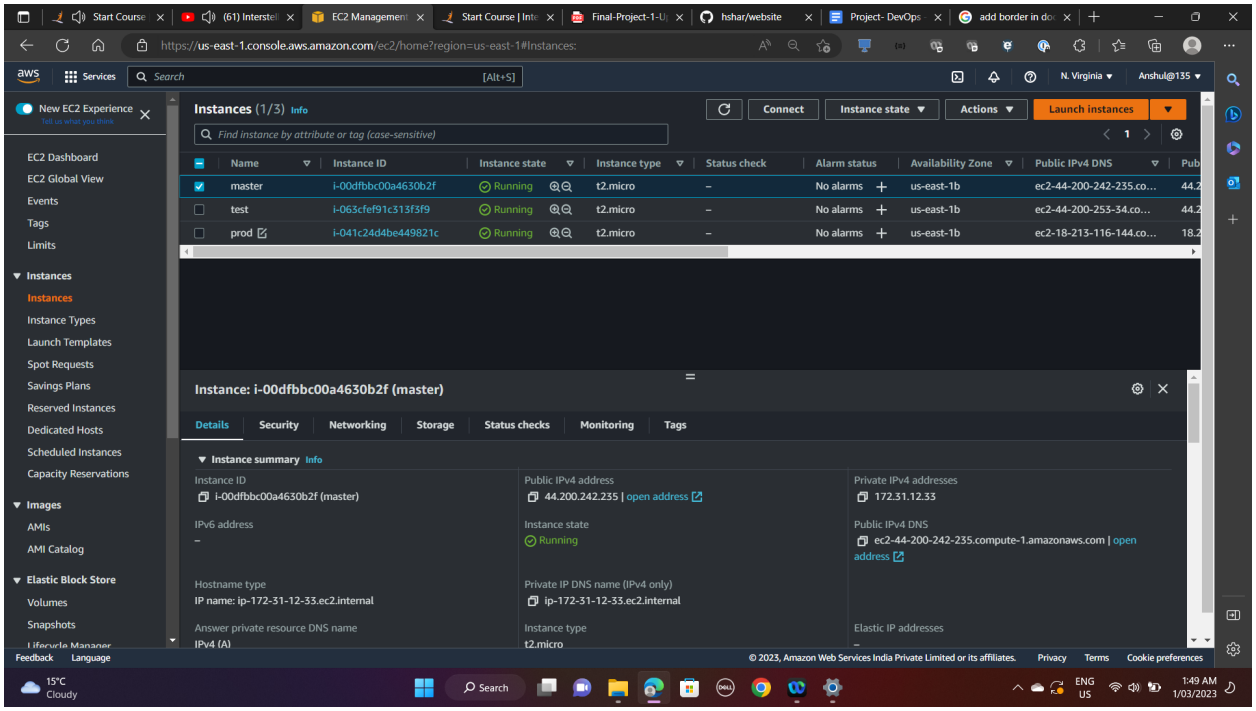
5. Since you are setting up the server for the first time, ensure the following file exists on both Test and Prod server in /home/ubuntu/config-management/status.txt.
This file will be used by a third-party tool. This should basically have the info whether apache is installed on the system or not

The content of this file should be based on whether git is installed or not.

If apache is installed => Apache is Installed on this System"
If apache is not installed => "Apache is not installed on this System"

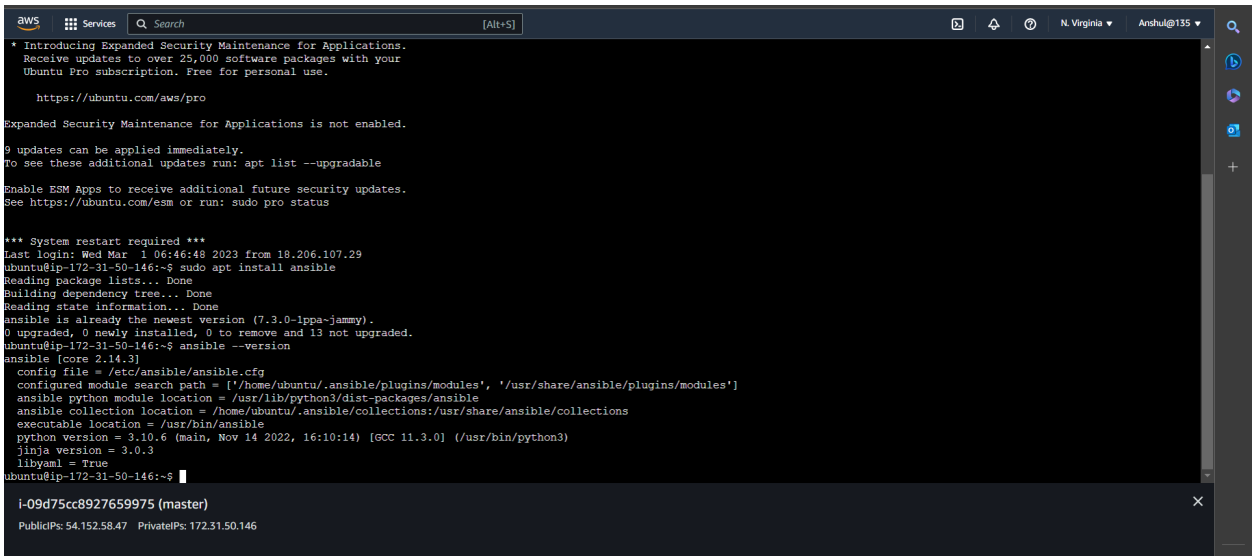
Architectural Advice:
Create 3 servers on AWS "t2.micro"
Server 1 - should have Jenkins Master, Puppet Master and Nagios Installed
Server 2 - Testing Server, Jenkins Slave
Server 3 - Prod Server, Jenkins Slave

Let us create 3 instances first with t2.micro or t2.medium(if you want a smoother operation).



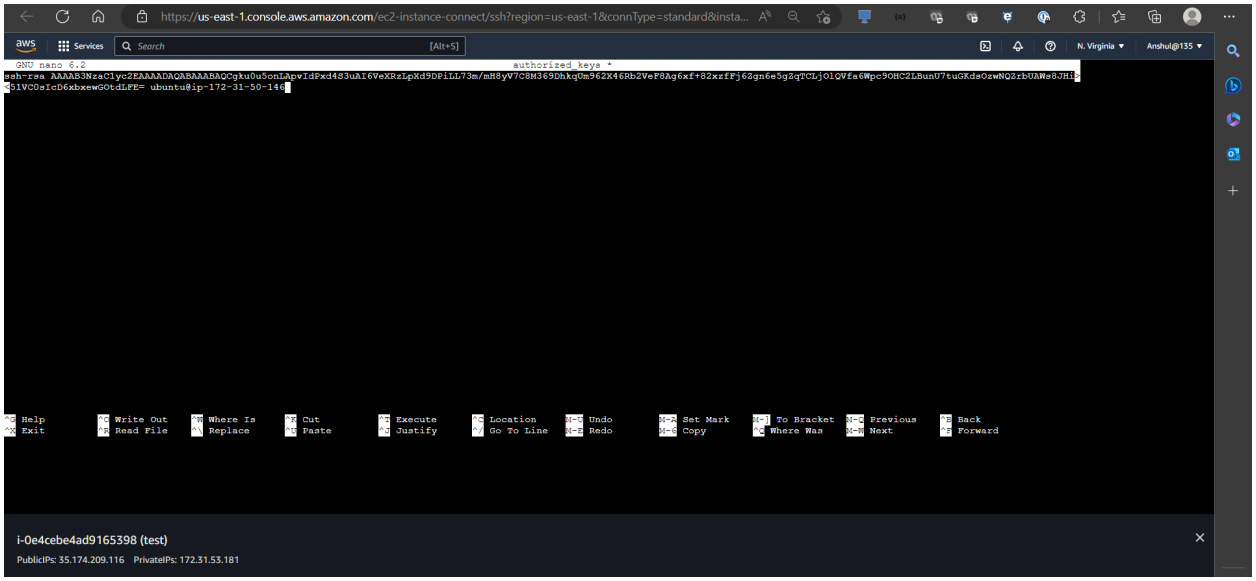
Now let us install Ansible on master referring to the official website.

We ran all the above mentioned commands on master server and installed Ansible.

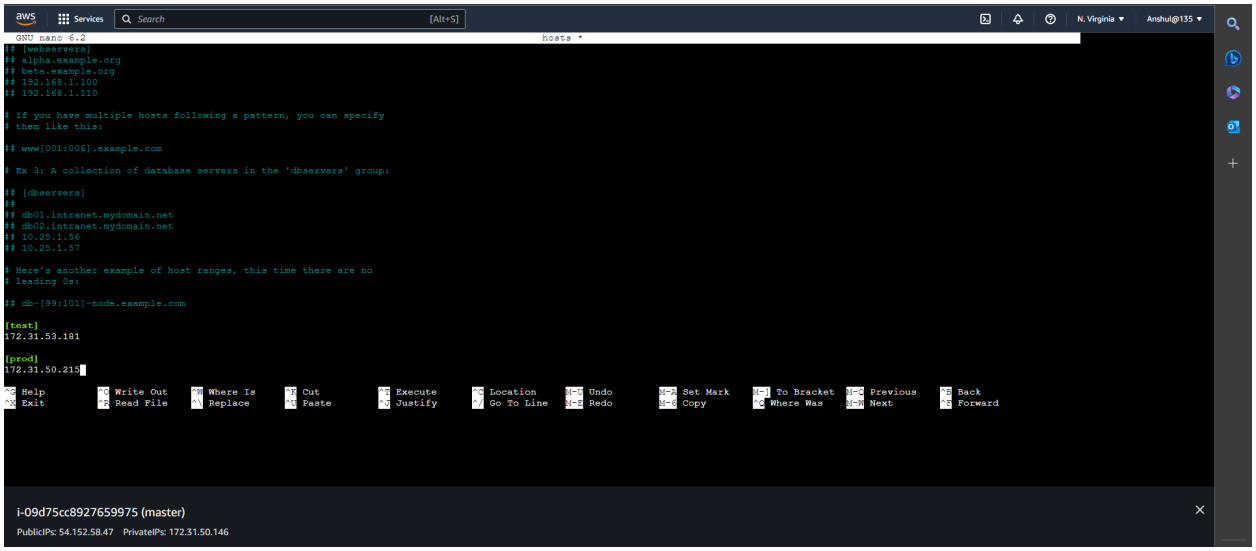


As now ansible is installed in Master, we will make a cluster with test and prod as worker nodes.
For that get the key from master server first using:
ssh key-gen
cd .ssh
Cat /home/ubuntu/.ssh/id_rsa.pub
Copy the key that you get here.

We need to provide these keys to the expected worker nodes.
For this do the following steps in worker nodes:
cd .ssh
sudo nano authorized_keys
And paste the key here



Now we need to let the master server know who its hosts are.
For that do this in master server:
cd /etc/ansible
Sudo nano hosts
And paste the private ip of both the test and prod instances in the following way:



To check if node workers are connected with master: ansible -m ping all

Now that the cluster is made, we need to install:
Jenkins, Java and Docker in master.
Java and docker in Test and Prod servers.
Let us create a shell script file to install jenkins separately then we will call it in playbook YAML file.
nano jenkins.sh
And paste below contents:

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.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
sudo apt-get update
sudo apt-get install jenkins
```

Now let us create a Ansible Playbook:
Sudo nano install.yaml
And paste this:

```
---
- name: Install Jenkins, Docker, and Java on Localhost
  hosts: localhost
  become: true

  tasks:
    - name: Install Java
      apt:
        name: openjdk-11-jdk
        state: present

    - name: Install Docker
      apt:
        name: docker.io
        state: present

    - name: running a script to install jenkins
      script: jenkins.sh

- name: Install Java and Docker on Test and Production Servers
  hosts: test, prod
  become: true

  tasks:
    - name: Install Java
      apt:
        name: openjdk-11-jdk
        state: present

    - name: Install Docker
      apt:
        name: docker.io
        state: present
```

Above playbook decoded:
The first play targets the "localhost" host and installs the required packages. The "become: true" statement is used to escalate privileges to become a superuser (root) before executing the tasks.

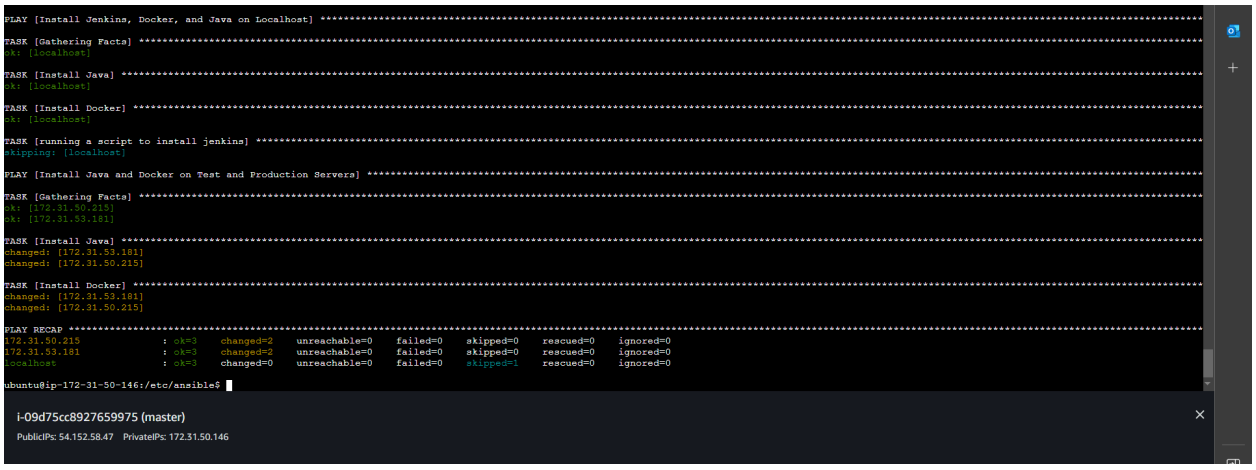
The tasks in the first play install OpenJDK-8-JDK and Docker using the "apt" module, which is a package manager for Debian-based systems like Ubuntu.

The last task in the first play runs a script named "jenkins.sh" to install Jenkins. The script should be present in the same directory as the playbook.

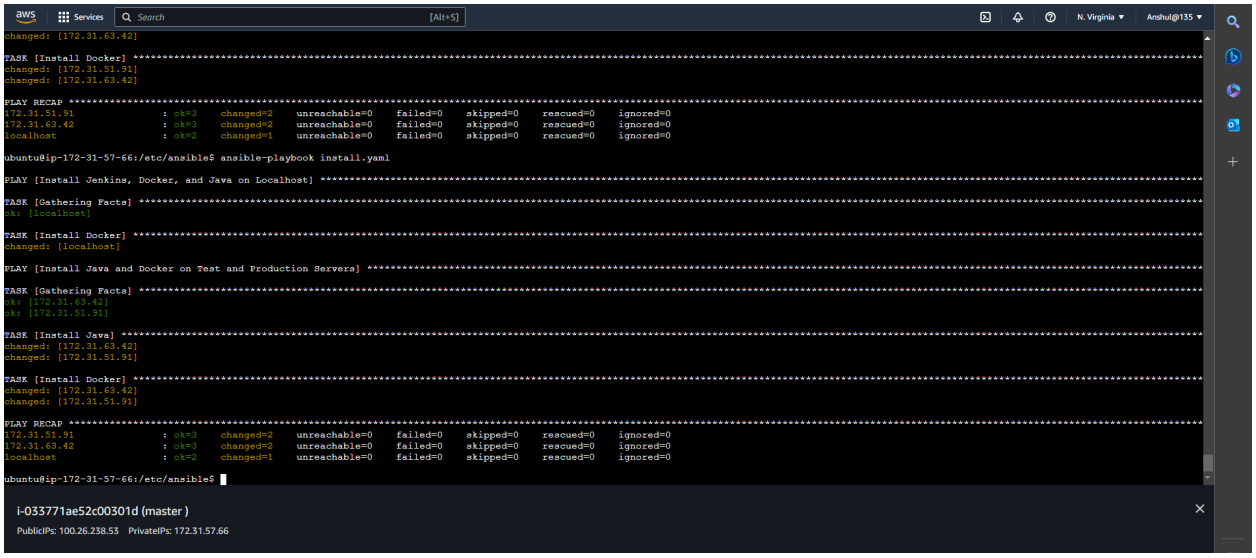
The second play targets the "test" and "prod" hosts and installs OpenJDK-8-JDK and Docker using the "apt" module.

Overall, this playbook automates the installation of Java, Docker, and Jenkins on multiple servers, making it easier to manage and maintain the required software stack. To execute this playbook, use the "ansible-playbook" command and specify the name of the YAML file containing the playbook.

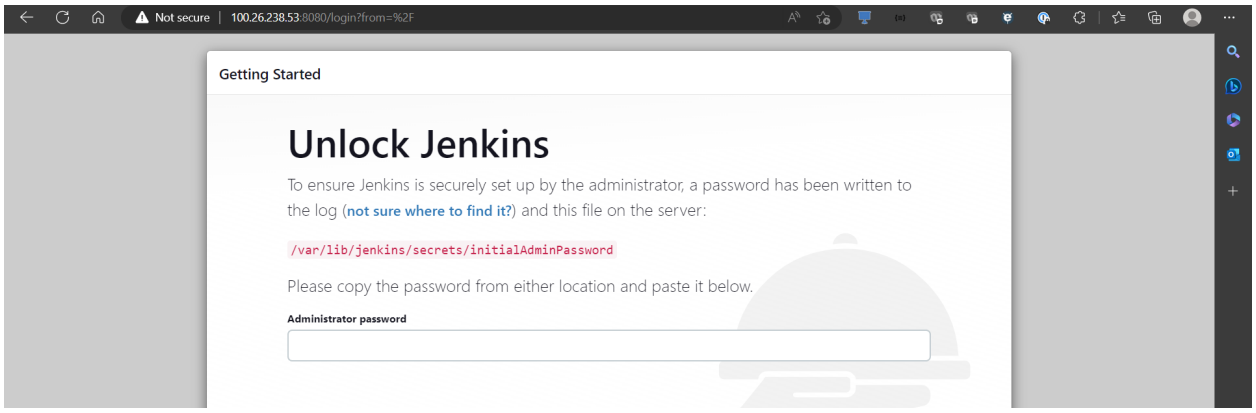
Now let us do the syntax check and dry run of the above YAML file on master server:
Ansible-playbook <filename.yaml> –syntax-check
ansible-playbook <file.yaml> --check



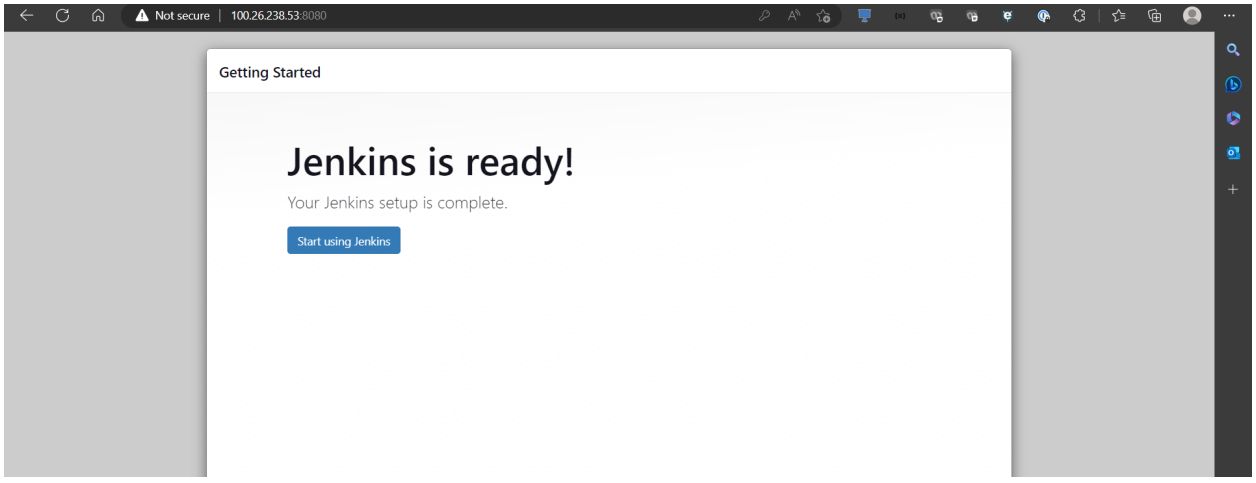
Let us run playbook using: ansible-playbook <filename.yaml>



Let us login to jenkins from master now. Copy paste piblic_ip:8080.
Run:sudo nano `/var/lib/jenkins/secrets/initialAdminPassword`
And paste the password here to login



Now install suggested plugins and create first user.



Now let us clone the git repo mentioned in Project Description.
git clone https://github.com/hshar/website.git

Now we need to create a pipeline through a Docker File.
Let us create a docker file first in the cloned repo.
Cd website
sudo nano dockerfile

And paste these contents here:
FROM ubuntu
RUN apt update
RUN apt install apache2 -y
ADD . /var/www/html
ENTRYPOINT apachectl -D FOREGROUND

Decode:This Dockerfile specifies a Docker image based on the Ubuntu operating system. The RUN command is used to update the package repository with the apt package manager, and then to install the apache2 web server. The -y flag is used to automatically confirm any prompts during the installation process.

The ADD command is used to copy the contents of the current directory (denoted by .) to the /var/www/html directory inside the container. This likely includes the web files that will be served by the Apache web server.

Finally, the ENTRYPOINT command is used to specify the command that should be executed when the container starts. In this case, it starts the Apache web server using the apachectl command with the -D FOREGROUND option, which starts the server in the foreground and keeps the container running.

Overall, this Dockerfile can be used to create a container that runs the Apache web server and serves the web files located in the /var/www/html directory inside the container.

Let us add this to github and then commit it.

Git add .

Git commit -m “on master dockerfile”

Also create a develop branch which will help later.

git branch develop

Let us push them now. Use:

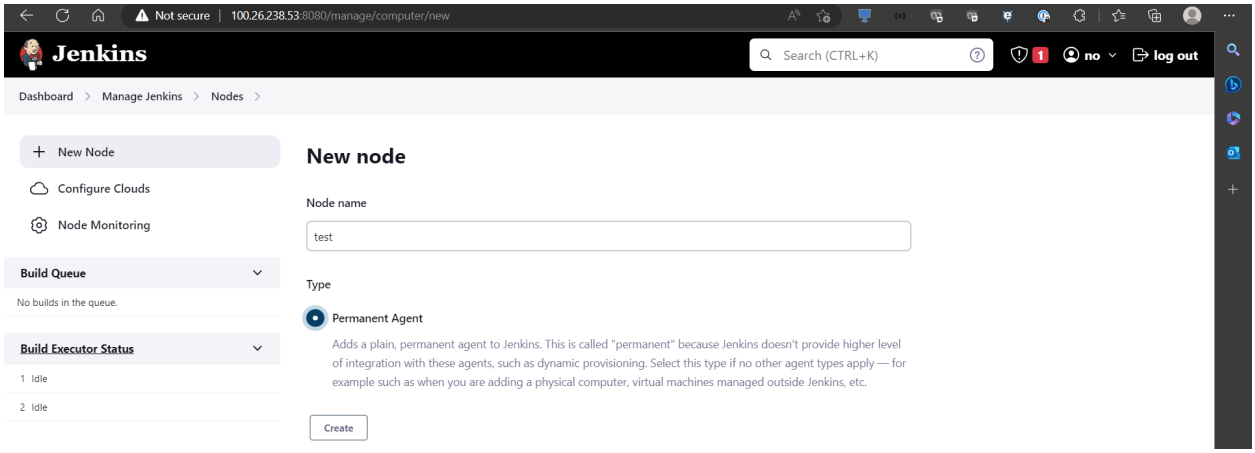
Git push origin develop

And provide necessary git details

Similarly push master branch.

Git checkout master
Git push origin master

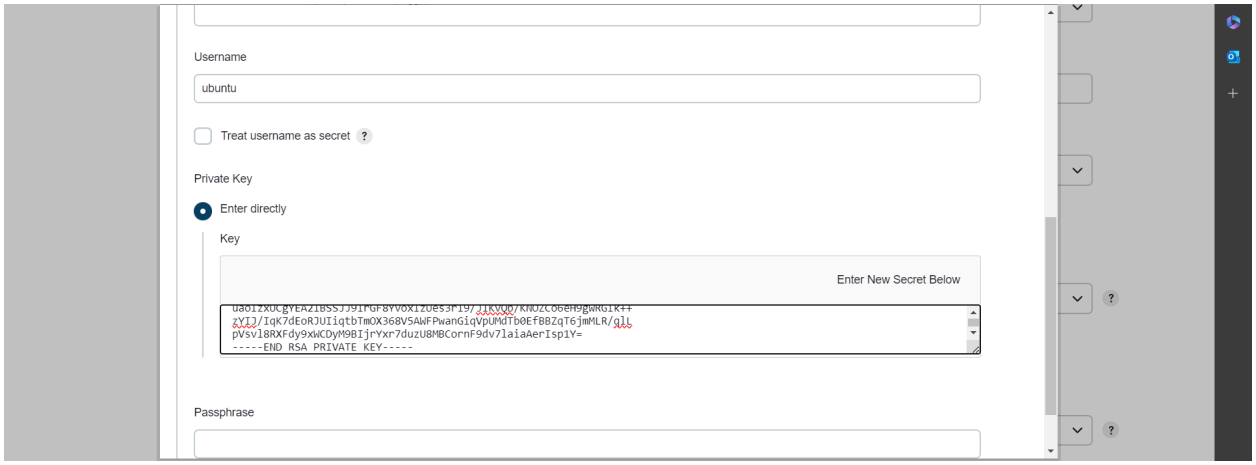
Changes should be reflected in your repo by now.
Now let us go to jenkins and create a pipeline.
Let us connect both the test and prod nodes to jenkins first.
Goto: jenkins dashboard>manage jenkins>manage nodes>add nodes



We are adding Test node first.
Enter our root directory as: /home/ubuntu/jenkins/
And select launch method as: Launch via ssh

Give private ip while filling details.
In kind choose: SSH username with Private Key

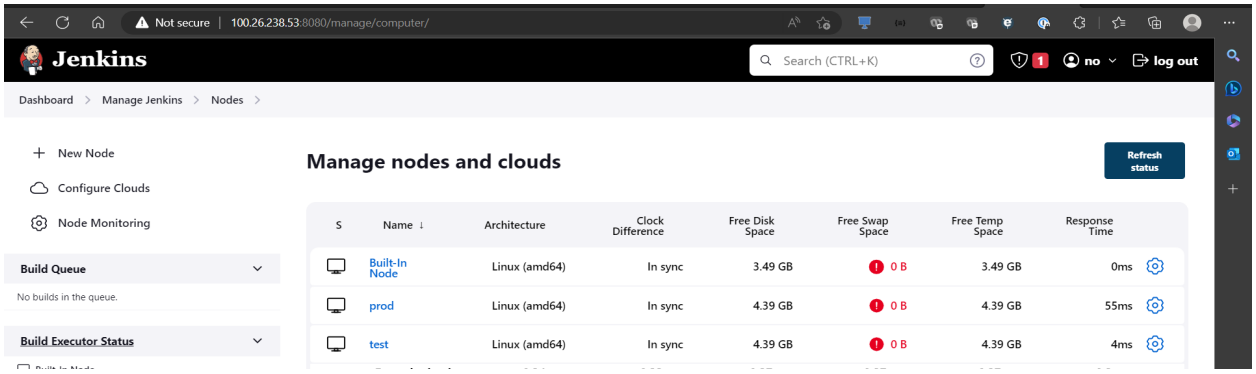
Choose username:ubuntu and paste the keypair details here.



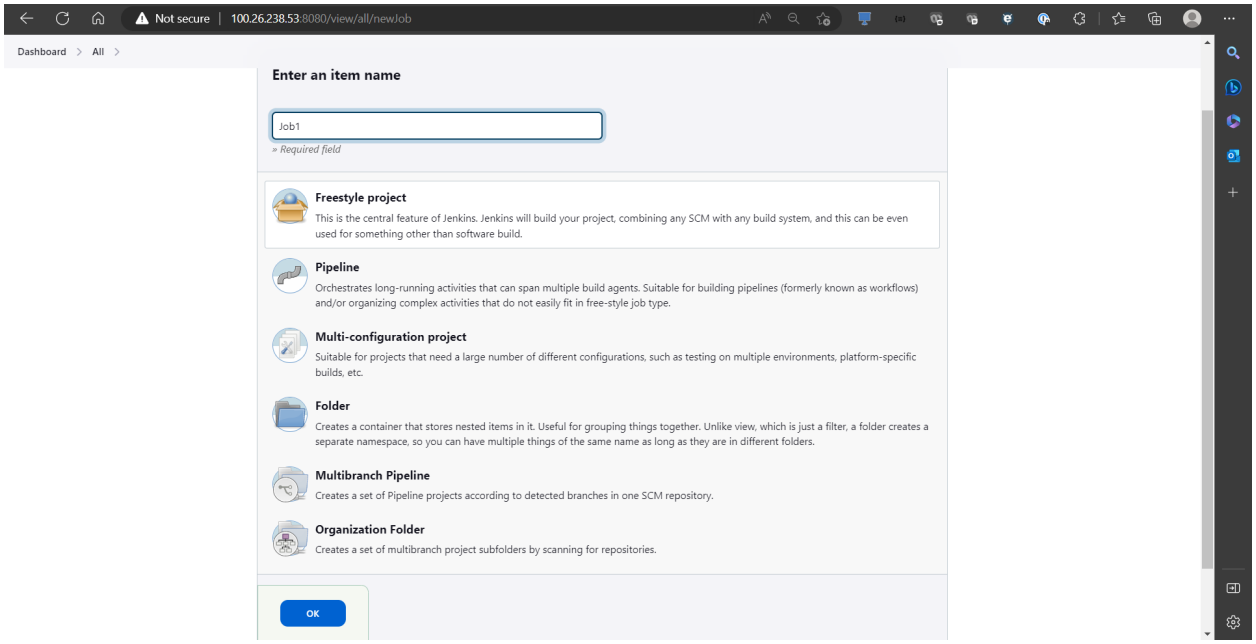
Let host key verification strategy be non verifying and click on save.
Now add Prod node aswell.

Do the same steps to add prod node like we did for test node. Only while providing ip for ssh prod server private ip should be provided.

Our nodes are ready:

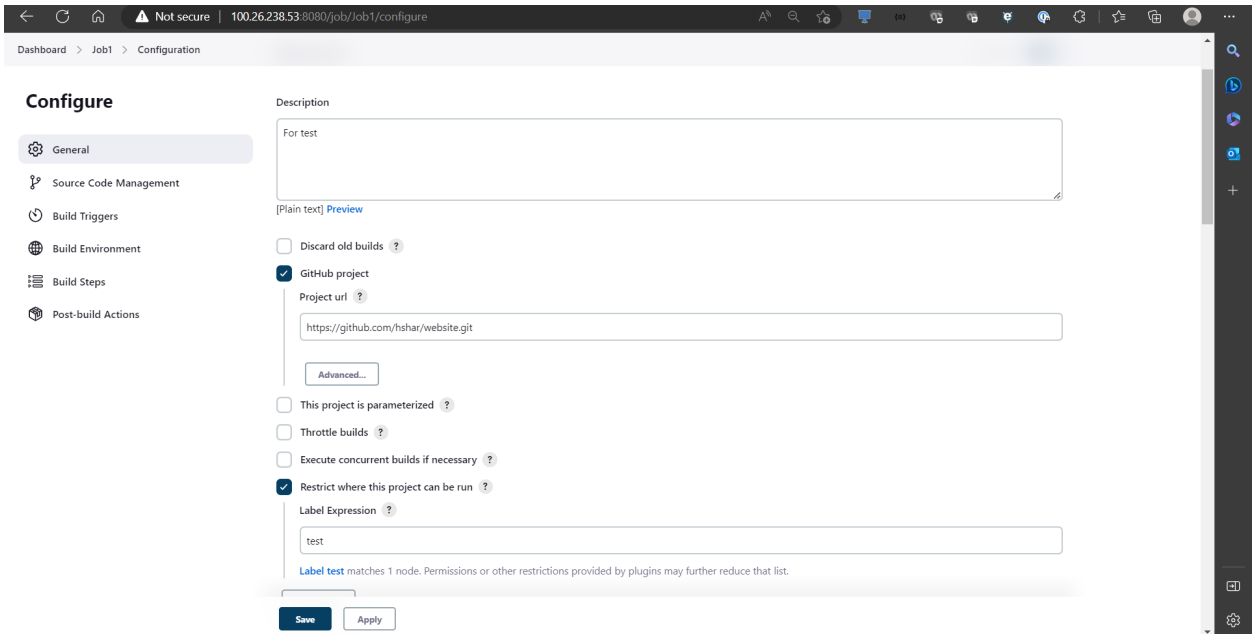


Let us create a Job1 now.
Goto jenkins dashboard create new item with freestyle project. Name it as job1.

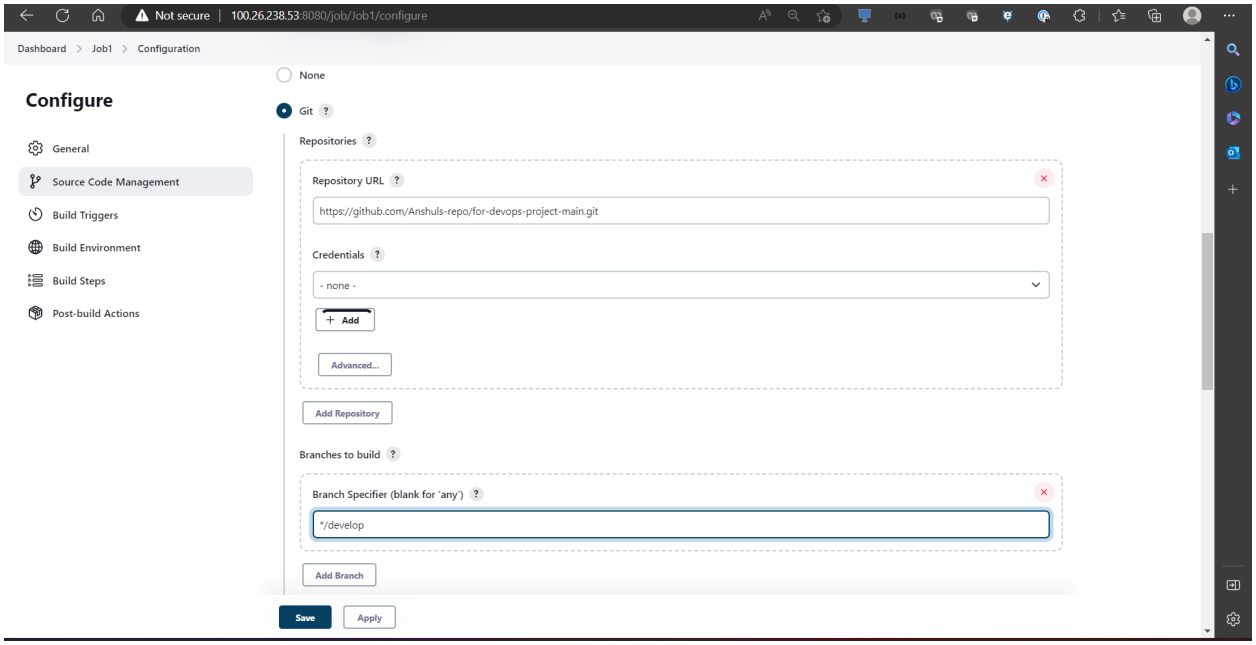


In general, add description according to need then Check Mark Github Project and paste the github link here.

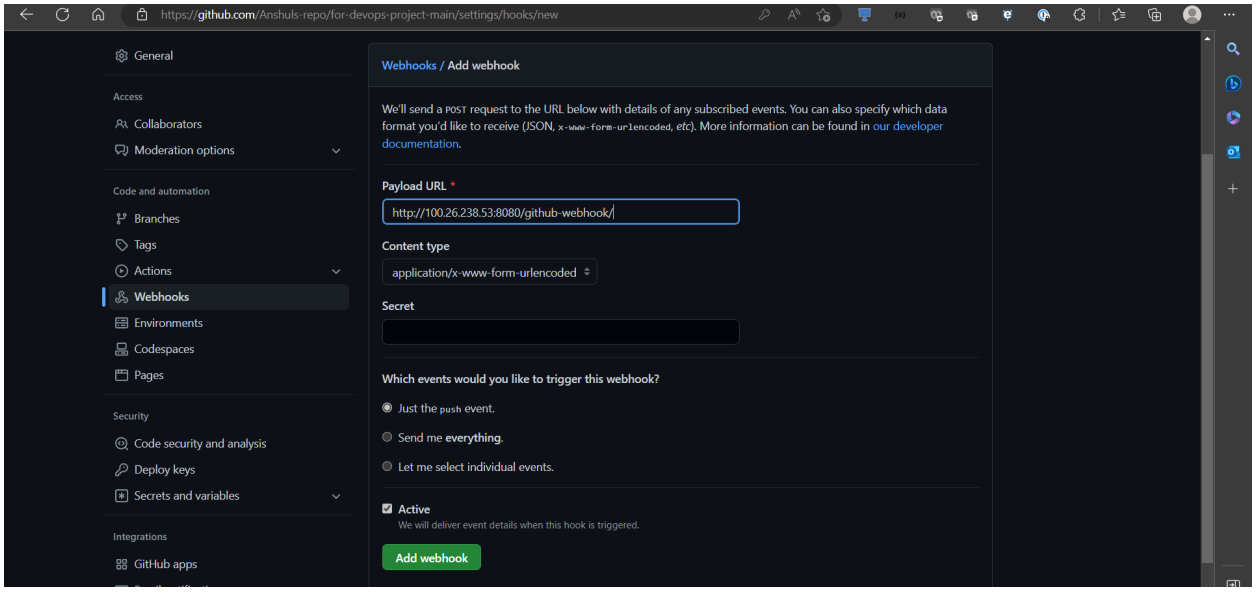
We also wants to run on test node so restrict it to run on test node.



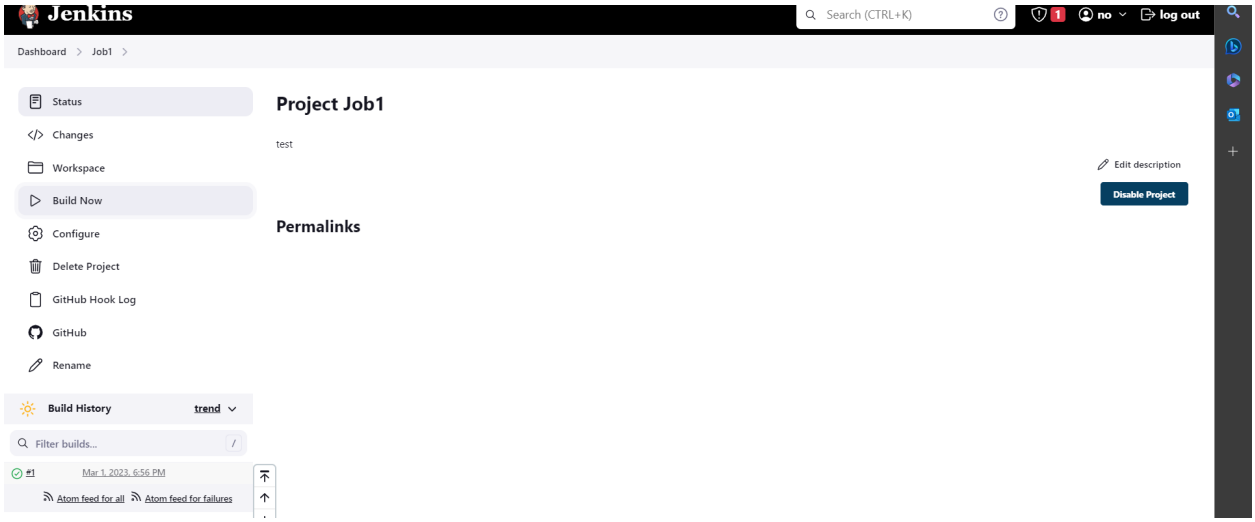
Now in Source Code Management enter your github link and enter branch as develop as we need it to trigger on develop branch changes.



Now in trigger enable GitHub hook trigger for GITScm polling.
Also enable webhook in git. Payload url format:
http://public_ip:8080/github-webhook/



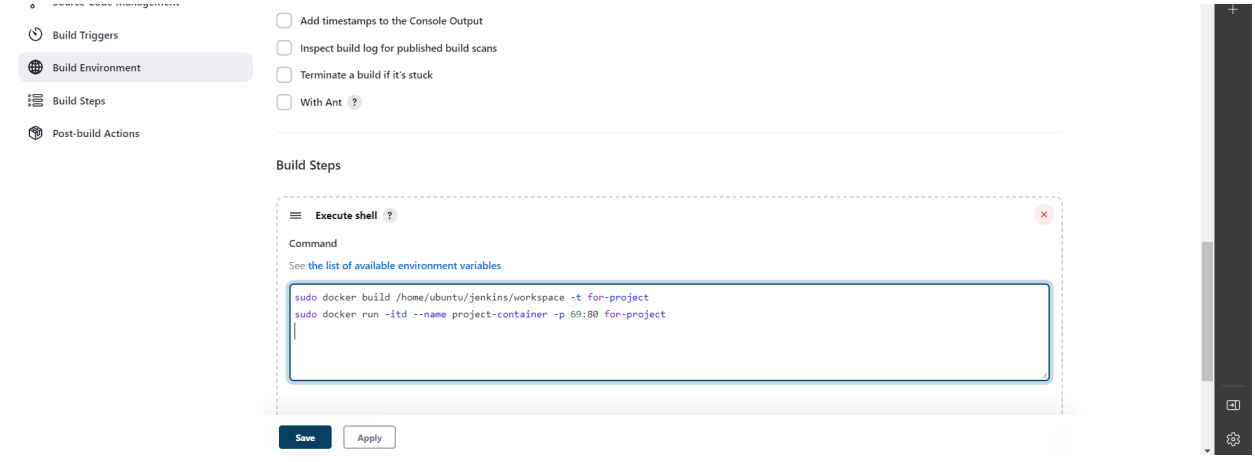
To check, you can run this job to see if it gets reflected in test server. If it does everything is working good yet. Click on Build Now.



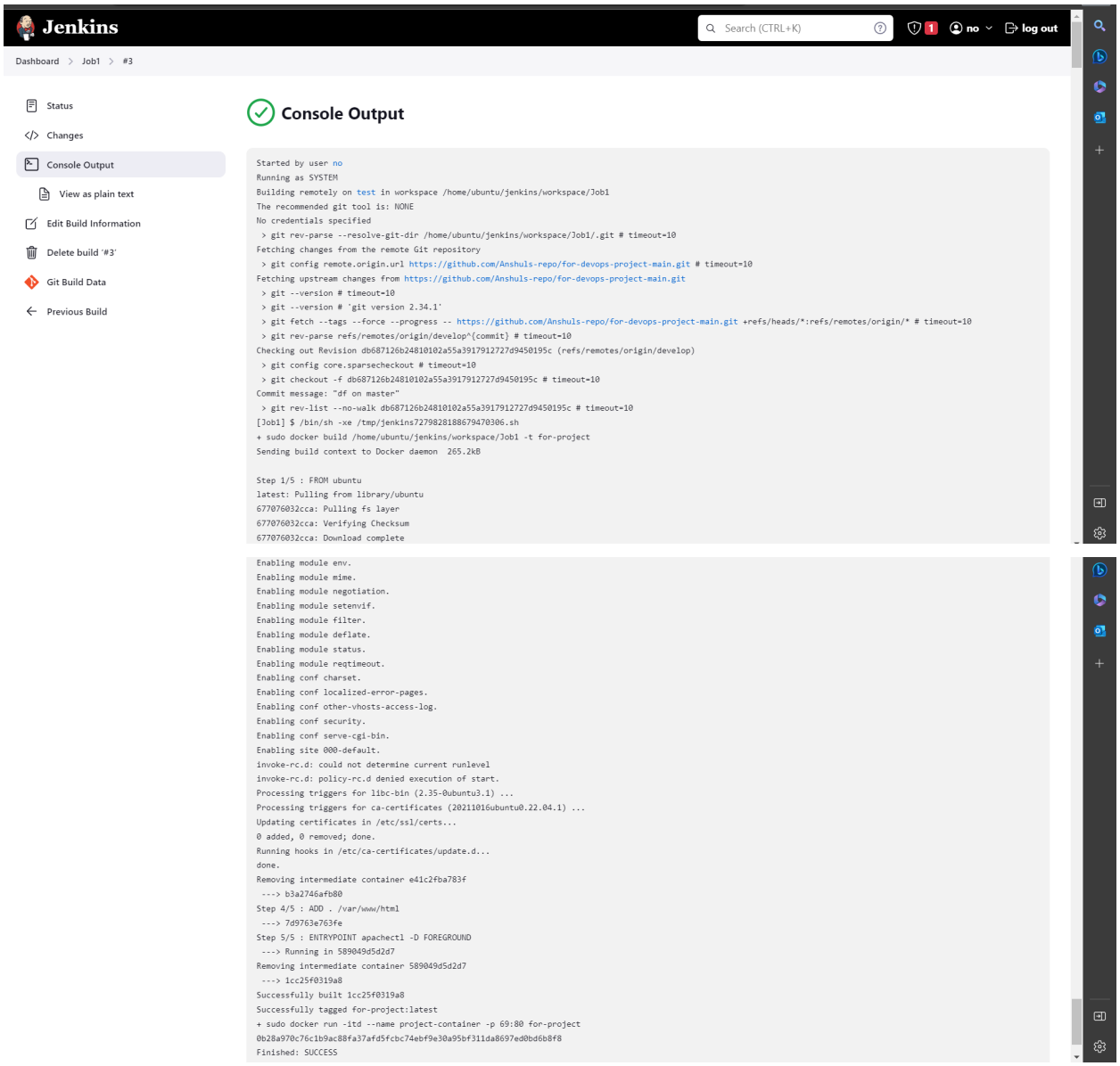
Now in test server:
cd jenkins
cd workspace
ls

You should see Job1 created.

Now goto Configuration>>Build Steps and choose Execute Shell.
We give command to build docker file:
sudo docker build /home/ubuntu/jenkins/workspace/Job1 -t devproject
sudo docker run -itd --name devcontainer -p 81:80 devproject



Apply and save.
Then click on build now.
Job1 build is successful.

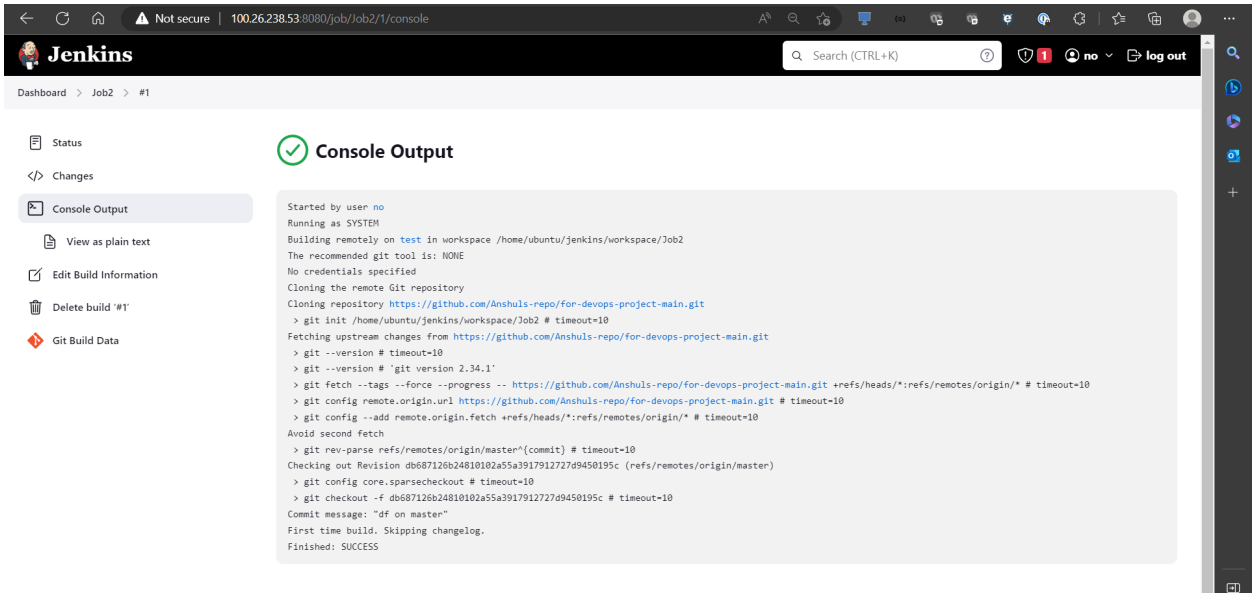


Also you can paste:public_ip_testserver:81

Now let us create a Job2

Do the exact same thing as Job1. Just select branch as Master branch here.

And run the Job2.



Again you will be able to see job2 in workspace in test ec2 like before.

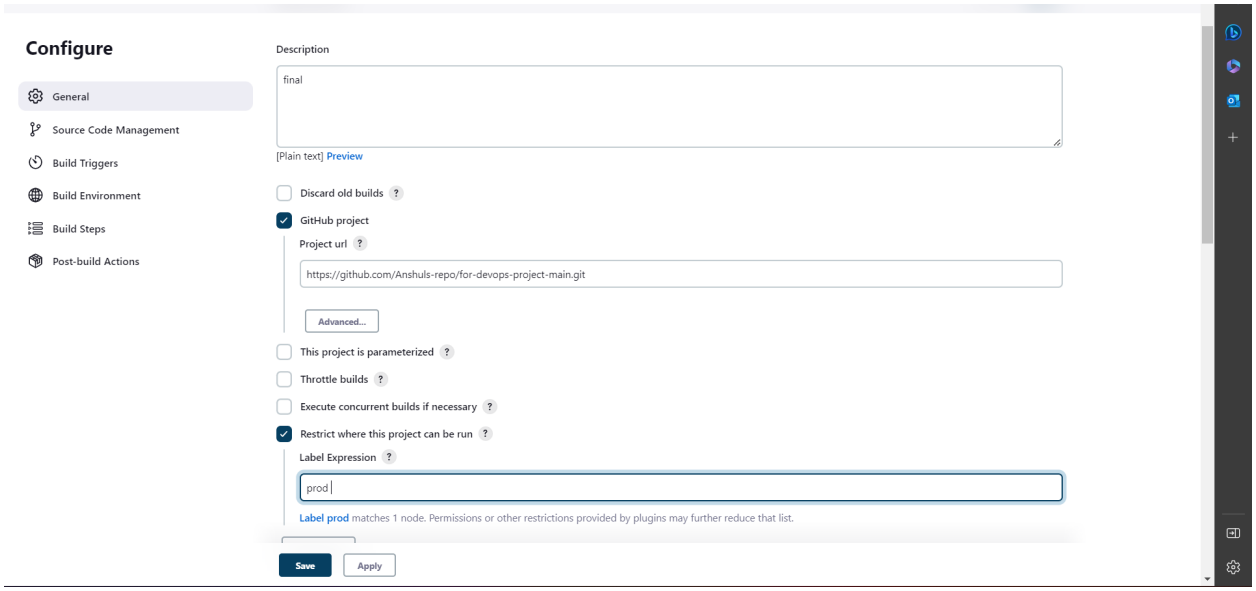
Now goto: configuration>build steps and select Execute shell.
Use commands here:
sudo docker build /home/ubuntu/jenkins/workspace/Job2 -t masterproject
sudo docker run -itd --name devcontainer -p 82:80 masterproject

Copy paste public_ip_testserver:82

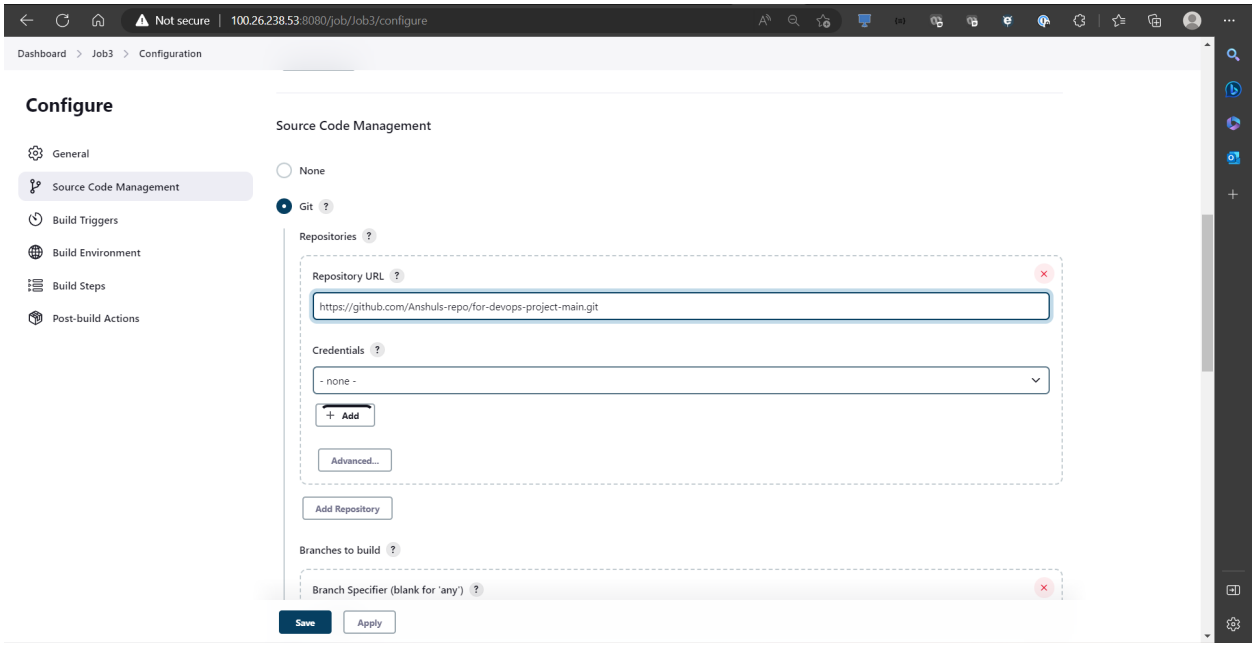
Let add one more command in Execute shell in both the jobs:
sudo docker rm -f $(sudo docker ps -a -q)

So when you run this command, it will forcefully stop and remove all Docker containers that are currently running and stopped on your system.
Here it is useful because when you want to change container that is supposed to run on that same port, we will not need to do it manually.

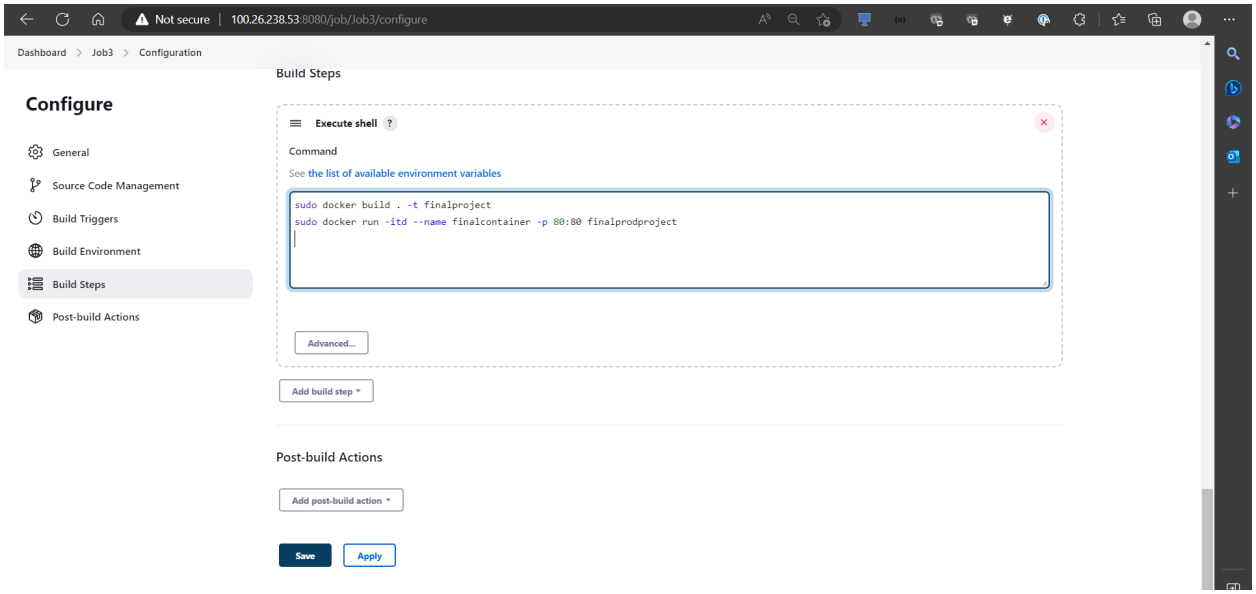
Let us move forward. Create job3 that will be run on prod worker node:
goto:dashboard>>new item>>Job3(freestyle)
In general give the github link and restrict it to prod node.



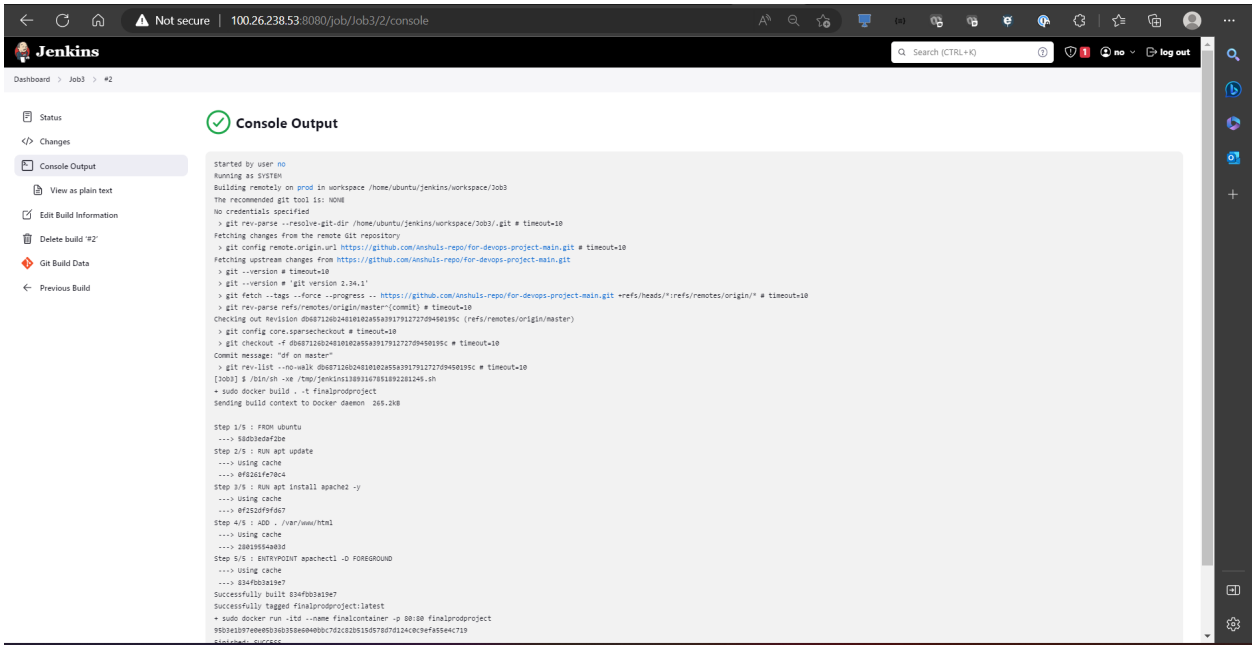
In Source Code management give your github repo name and branch of your choice.



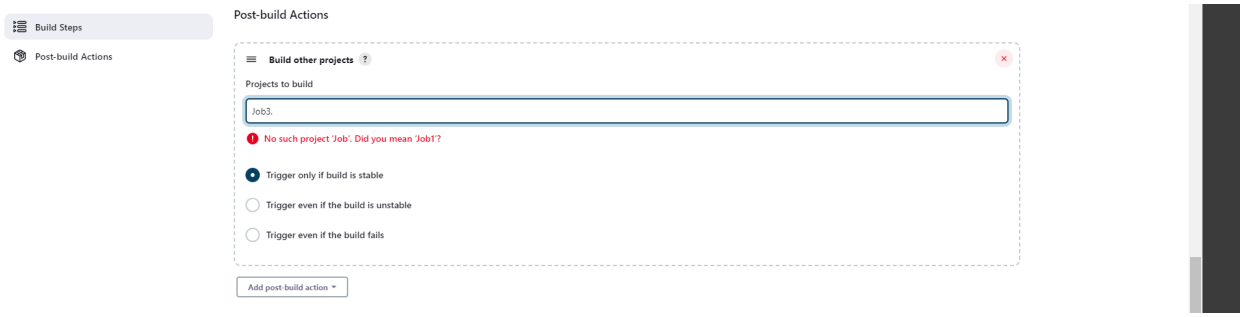
Do not choose Webhook as build trigger here, as build trigger for prod should be successful running on test.
For now add these Execute shell Commands in build steps:
sudo docker rm -f $(sudo docker ps -a -q)
sudo docker build . -t finalprodproject
sudo docker run -itd --name finalcontainer -p 80:80 finalprodproject



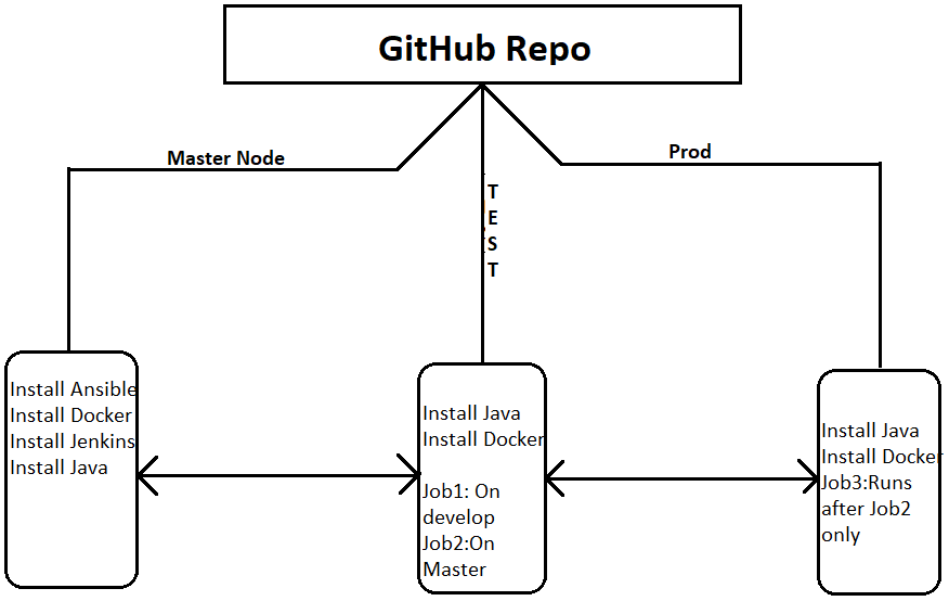
Click build now.



Now that it is running fine without dependencies. Let us give dependency to job2. Add this in Post Build Options of job2:



Overview:


GitHub Repo
Master Node
TEST
Prod
Install Ansible
Install Docker
Install Jenkins
Install Java
Install Java
Install Docker
Job1: On develop
Job2:On Master
Install Java
Install Docker
Job3:Runs after Job2 only