

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).

The screenshot shows the AWS EC2 Management console with three instances listed:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Pub
master	i-00dfbb00a4630b2f	Running	t2.micro	-	No alarms	+ us-east-1b	ec2-44-200-242-235.co...	44.2
test	i-063cef91c513f3f9	Running	t2.micro	-	No alarms	+ us-east-1b	ec2-44-200-253-34.co...	44.2
prod	i-041c24d4be449821c	Running	t2.micro	-	No alarms	+ us-east-1b	ec2-18-213-116-144.co...	18.2

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

The screenshot shows the Ansible documentation page for Ubuntu:

Installing Ansible on Ubuntu

Ubuntu builds are available in a PPA here.

To configure the PPA on your system and install Ansible run these commands:

```
$ sudo apt update
$ sudo apt install software-properties-common
$ sudo add-apt-repository -yes --update ppa:ansible/ansible
$ sudo apt install ansible
```

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

```
* Introducing Expanded Security Maintenance for Applications.
Receive updates to over 25,000 software packages with your
Ubuntu Pro subscription. Free for personal use.
https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

9 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

*** System restart required ***
Last login: Wed Mar 1 06:46:48 2023 from 18.206.107.29
ubuntu@ip-172-31-50-146:~$ sudo apt install ansible
Reading package lists... Done
Building dependency tree... Done
Reading status information... Done
ansible is already the newest version (7.3.0-1ppa-jammy).
0 upgraded, 0 newly installed, 0 to remove and 13 not upgraded.
ubuntu@ip-172-31-50-146:~$ ansible --version
ansible [core 2.14.3]
config file = /etc/ansible/ansible.cfg
configured module search path = ['/home/ubuntu/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
ansible python module location = /usr/lib/python3/dist-packages/ansible
ansible collection location = /home/ubuntu/.ansible/collections:/usr/share/ansible/collections
executable location = /usr/bin/ansible
python version = 3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0] (/usr/bin/python3)
jinja version = 3.0.3
libyaml = True
ubuntu@ip-172-31-50-146:~$ 
```

i-09d75cc8927659975 (master)

PublicIPs: 54.152.58.47 PrivateIPs: 172.31.50.146

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.

```
isEcrypt = True
ubuntu18@ip-172-31-50-146:~$ ssh-keygen
Generating RSA public key, 1024 bit...
Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ubuntu/.ssh/id_rsa
Your public key has been saved in /home/ubuntu/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:U7wpdBM9uJ0lR0m8rBabneyfCGC1pFMrzBMK ubuntu18@ip-172-31-50-146
The key's randomart image is:
+---[RSA 1024]---
| . . . .
| . . . .
| . . . .
| . . . .
| . . . .
| . . . .
| . . . .
| . . . .
| . . . .
+----[SHA256]----+
ubuntu18@ip-172-31-50-146:~$ cd .ssh
ubuntu18@ip-172-31-50-146:~/ssh$ ls
authorized_keys  id_rsa  id_rsa.pub
ubuntu18@ip-172-31-50-146:~$ .ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQgCku9nLapwvdPxds3uA16VeKRxZpX9dPfXmH8yV7C8m369DhXm962X46B2vFgA6x+82xzf7g2n6e5g2qTCljO1Qvfa6Wpc3OH21BunU7tuGdsozwnQ2rbUaw8JHid12t9mc7I23LgWNWD5nBl
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQgCku9nLapwvdPxds3uA16VeKRxZpX9dPfXmH8yV7C8m369DhXm962X46B2vFgA6x+82xzf7g2n6e5g2qTCljO1Qvfa6Wpc3OH21BunU7tuGdsozwnQ2rbUaw8JHid12t9mc7I23LgWNWD5nBl
ubuntu18@ip-172-31-50-146:~$ .ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQgCku9nLapwvdPxds3uA16VeKRxZpX9dPfXmH8yV7C8m369DhXm962X46B2vFgA6x+82xzf7g2n6e5g2qTCljO1Qvfa6Wpc3OH21BunU7tuGdsozwnQ2rbUaw8JHid12t9mc7I23LgWNWD5nBl
Fy2dV0WnWzbhWeynpJ66eqp0+6+b3)hydQ01FN9cM2zQhcbRFxuA116x6J0cMoWbG+Ned+VxWHe68LI0t1fqzRcn+Mwzqj0E3yjH01hFdoe9F1EasmlLWshoP02ezx3fywYCE8NxrdBwgLctk8fIKT4xss+draTizxCKnt3pmg5cs7seyJ3M
ubuntu18@ip-172-31-50-146:~$ .ssh$ cat id_rsa
ubuntu18@ip-172-31-50-146:~$ .ssh$
```

We need to provide these keys to the expected worker nodes.

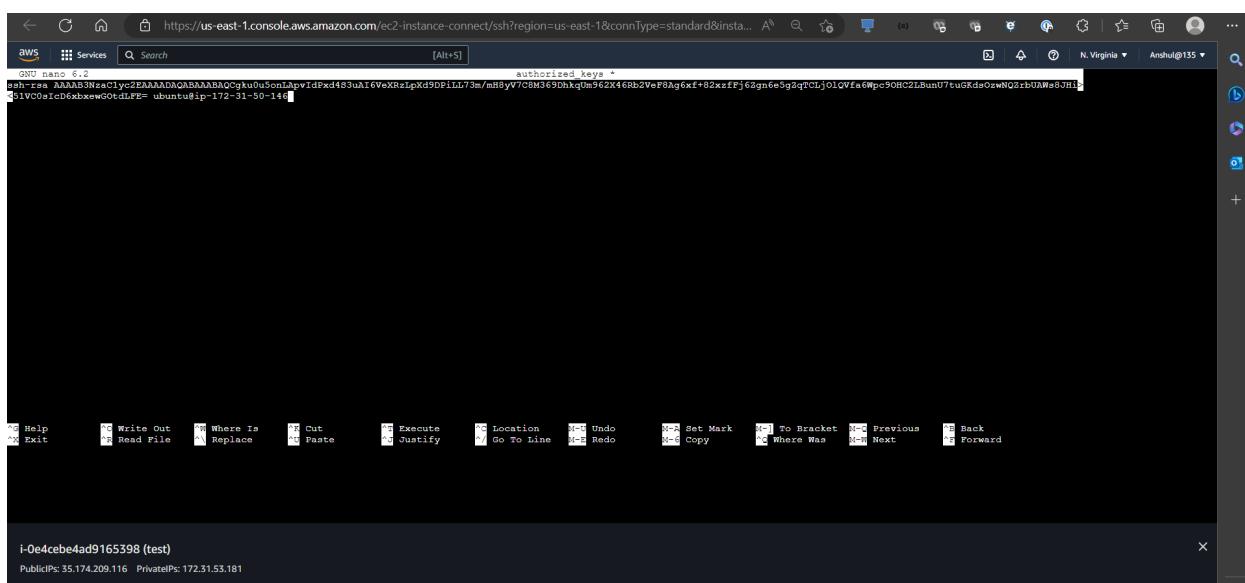
For this do the following steps in worker nodes:

cd .ssh

sudo nano authorized_keys

And paste the key here

```
Reading state information... done
28 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu@ip-172-31-53-181:~$ Ed .ssh
ubuntu@ip-172-31-53-181:~/.ssh$ sudo nano authorized_keys
ubuntu@ip-172-31-53-181:~/.ssh$ [REDACTED]
X
```



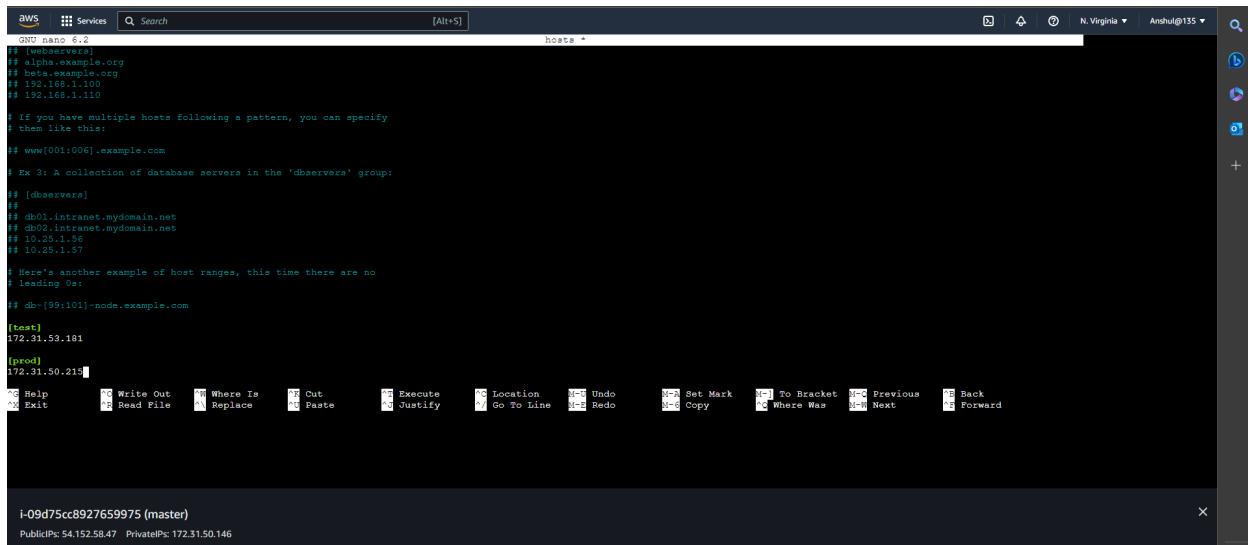
Now we need to let the master server know who its hosts are.

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:



The screenshot shows a terminal window titled 'hosts' with the file content displayed in the nano text editor. The code defines two groups of hosts: 'webservers' and 'db'. The 'webservers' group contains 'alpfa.example.org' and 'beta.example.com' with private IPs '192.168.1.100' and '192.168.1.110'. The 'db' group contains 'www[001:006].example.com' and 'db[99:101]-node.example.com' with private IPs '172.31.53.181' and '172.31.50.215'. The terminal window has a dark theme and includes standard nano keybindings at the bottom.

```
GNU nano 6.2
## (webservers)
## alpfa.example.org
## beta.example.com
## 192.168.1.100
## 192.168.1.110

## If you have multiple hosts following a pattern, you can specify
## them like this:
## www[001:006].example.com

## Ex 3: A collection of database servers in the 'db' group:
## (db)
## db01.intranet.mydomain.net
## db02.intranet.mydomain.net
## 10.25.1.56
## 10.25.1.57

## Here's another example of host ranges, this time there are no
## leading 0s:
## db [99:101]-node.example.com

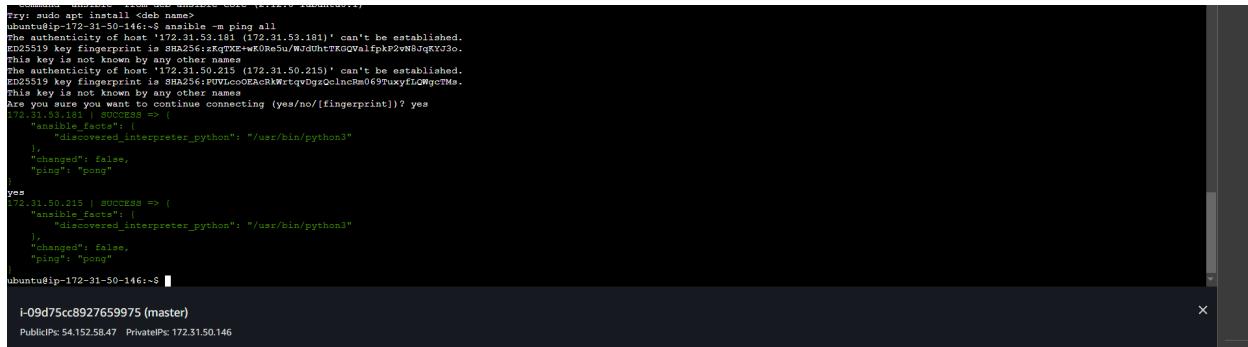
[webservers]
172.31.53.181

[db]
172.31.50.215

## Help          Write Out    Where Is      Cut           Execute      Location      Undo        Set Mark     To Bracket   Where Was    Previous    Back
[X] Exit       [W] Read file [R] Replace [C] Paste [E] Justify [L] Go to Line [U] Undo [S] Copy [B] To Bracket [W] Where Was [P] Previous [F] Back

i-09d75cc8927659975 (master)
PublicIP: 54.152.58.47 PrivateIP: 172.31.50.146
```

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



The screenshot shows a terminal window with the output of the 'ansible -m ping all' command. It shows two hosts: '172.31.53.181' and '172.31.50.215'. Both hosts respond with 'pong', indicating they are connected to the master. The terminal window has a dark theme and includes standard nano keybindings at the bottom.

```
ubuntu@ip-172-31-50-146:~$ ansible -m ping all
The authenticity of host '172.31.53.181' (172.31.53.181) can't be established.
ED25519 key fingerprint is SHA256:sxZqTxw-wfReSu/WzGhtzxGqvAlpkP2wR8JgXv2o.
This key is not known by any other names
The authenticity of host '172.31.50.215' (172.31.50.215) can't be established.
ED25519 key fingerprint is SHA256:FUVlcooEArKWrtyDgQcLnRa069TuxyfCQngCTMs.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
172.31.53.181: SUCCESS!
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
yes
172.31.50.215: SUCCESS!
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
ubuntu@ip-172-31-50-146:~$
```

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`

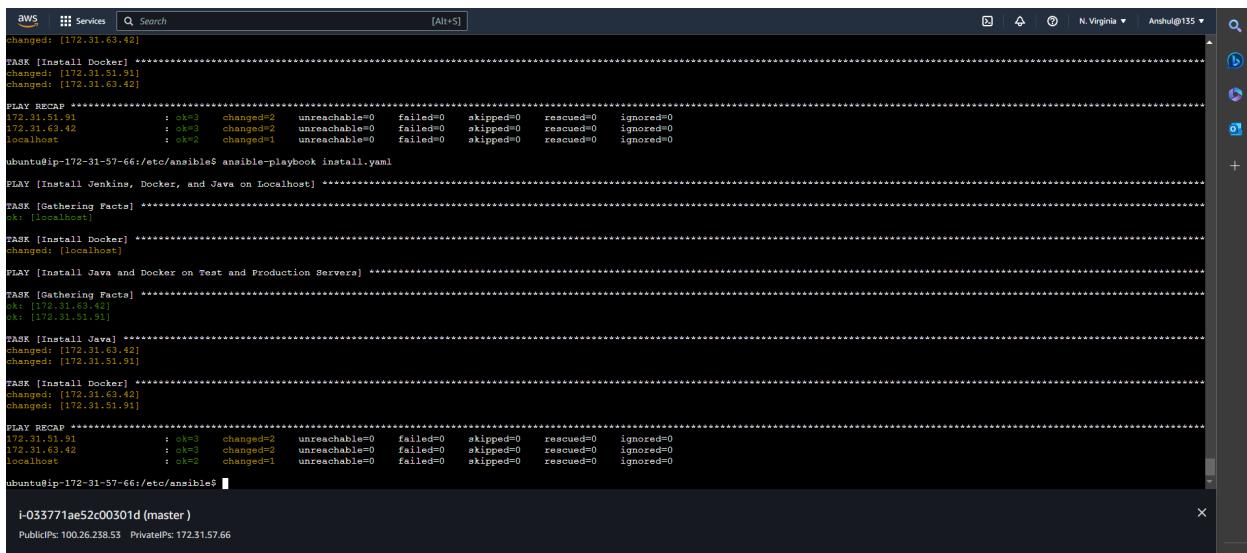
```
[WARNING]: Could not match supplied host pattern; ignoring: master
playbook: install.yaml
ubuntu@ip-172-31-50-146:/etc/ansible$ sudo nano install.yaml
ubuntu@ip-172-31-50-146:/etc/ansible$ sudo nano install.yaml
ubuntu@ip-172-31-50-146:/etc/ansible$ ansible-playbook Anatall.yaml --syntax-check
playbook: install.yaml
ubuntu@ip-172-31-50-146:/etc/ansible$ [REDACTED]

i-09d75cc8927659975 (master)
PublicIP: 54.152.58.47 PrivateIP: 172.31.50.146

PLAY [Install Jenkins, Docker, and Java on Localhost] *****
TASK [Gathering Facts] *****
ok: [localhost]
TASK [Install Java] *****
ok: [localhost]
TASK [Install Docker] *****
ok: [localhost]
TASK [running a script to install Jenkins]
skipping: [localhost]
PLAY [Install Java and Docker on Test and Production Servers] *
TASK [Gathering Facts] *****
ok: [172.31.53.181]
ok: [172.31.50.215]
TASK [Install Java] *****
changed: [172.31.53.181]
changed: [172.31.50.215]
TASK [Install Docker] *****
changed: [172.31.53.181]
changed: [172.31.50.215]
TASK [RECAR] *****
172.31.50.215 : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
172.31.53.181 : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
localhost   : ok=3    changed=0    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
ubuntu@ip-172-31-50-146:/etc/ansible$ [REDACTED]

i-09d75cc8927659975 (master)
PublicIP: 54.152.58.47 PrivateIP: 172.31.50.146
```

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

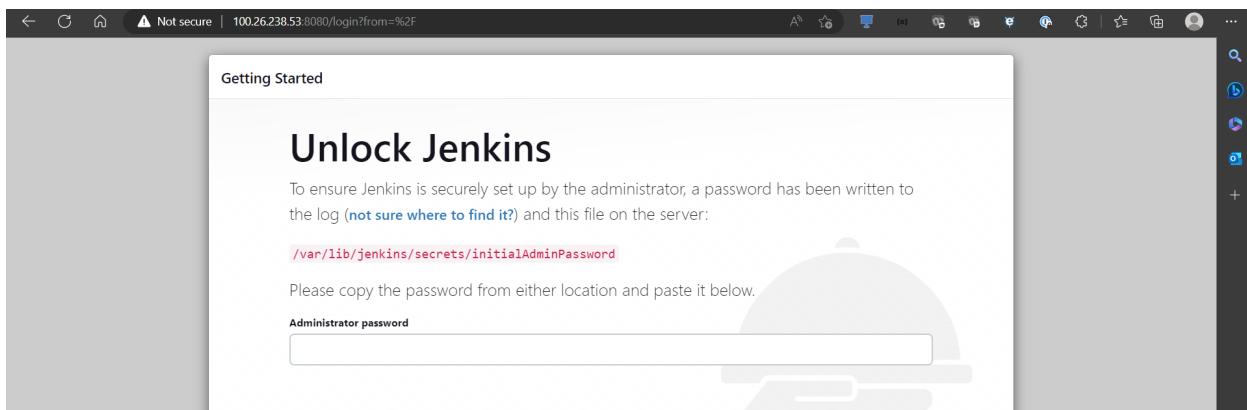


```
aws Services Search [Alt+S]
changed: [172.31.63.42]
TASK [Install Docker] ****
changed: [172.31.51.91]
changed: [172.31.63.42]
PLAY RECAP ****
172.31.51.91 : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
172.31.63.42 : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
localhost   : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
ubuntu@ip-172-31-57-66:/etc/ansible$ ansible-playbook install.yaml
PLAY [Install Jenkins, Docker, and Java on Localhost] ****
TASK [Gathering Facts] ****
skipped: [localhost]
TASK [Install Docker] ****
changed: [localhost]
PLAY [Install Java and Docker on Test and Production Servers] ****
TASK [Gathering Facts] ****
ok: [172.31.51.91]
skipped: [172.31.51.91]
TASK [Install Java] ****
changed: [172.31.63.42]
changed: [172.31.51.91]
TASK [Install Docker] ****
changed: [172.31.63.42]
changed: [172.31.51.91]
PLAY RECAP ****
172.31.51.91 : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
172.31.63.42 : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
localhost   : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
ubuntu@ip-172-31-57-66:/etc/ansible$ i-033771ae52c00301d (master)
PublicIPs: 100.26.238.53 PrivateIPs: 172.31.57.66
```

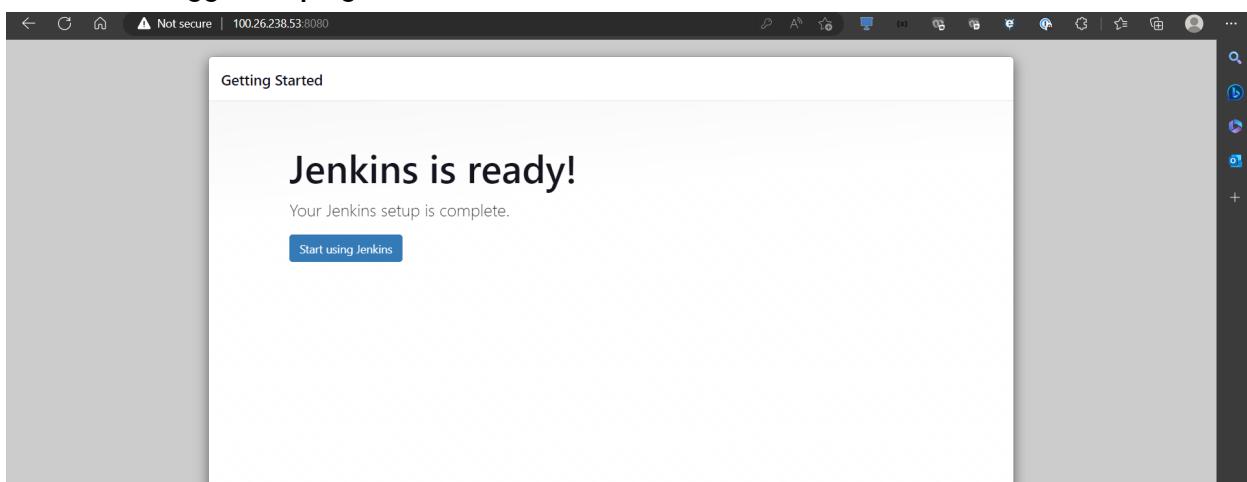
Let us login to jenkins from master now. Copy paste public_ip:8080.

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.

```
[1/2] docker +  
FROM ubuntu  
RUN apt update  
RUN apt install apache2 -y  
ADD . /var/www/html  
ENTRYPOINT apachectl -D FOREGROUND  
  
Help Close Write Out Where Is Cut Paste Execute Justify Location Go To Line Undo Set Mark To Bracket Previous Back Prev Word  
Close Read File Replace Paste Execute Justify Location Go To Line Undo Set Mark To Bracket Previous Back Prev Word  
Where Was Next Forward Next Word  
i-033771ae52c00301d (master) X
```

Let us add this to github and then commit it.

Git add .

Git commit -m "on master dockerfile"

```
Your branch is up to date with 'origin/master'.  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    dockerfile  
nothing added to commit but untracked files present (use "git add" to track)  
ubuntu@ip-172-31-57-66:~/website$ git commit -m "on master dockerfile"  
[master 1bc80ed] to master dockerfile  
  Committer: Ubuntu <ubuntu@ip-172-31-57-66.ec2.internal>  
  Your name and email address were configured automatically based  
  on your user account hostname. Please check that they are accurate.  
  You can override this message by setting them explicitly. Run the  
  following command and follow the instructions in your editor to edit  
  your configuration file:  
    git config --global --edit  
After doing this, you may fix the identity used for this commit with:  
  git commit --amend --reset-author  
  1 file changed, 5 insertions(+)  
  create mode 100644 dockerfile  
ubuntu@ip-172-31-57-66:~/website$  
  
i-033771ae52c00301d (master)  
PublicIPs: 100.26.238.53 PrivateIPs: 172.31.57.66 X
```

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

```
ubuntu@ip-172-31-57-66:~/For-devops-project-main$ git push origin develop  
Username for 'https://github.com': Anshuls-repo  
Password for 'https://Anshuls-repo@github.com':  
Enumerating objects: 100%, done.  
Counting objects: 100% (4/4), done.  
Compressing objects: 100% (3/3), done.  
Writing objects: 100% (3/3), 415.00 KiB/s, done.  
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0  
remote: Create a pull request for 'develop' on GitHub by visiting:  
remote:   https://github.com/Anshuls-repo/for-devops-project-main.git  
remote:  
To https://github.com/Anshuls-repo/for-devops-project-main.git  
 * [new branch]  develop -> develop  
ubuntu@ip-172-31-57-66:~/For-devops-project-main$  
  
i-033771ae52c00301d (master)  
PublicIPs: 100.26.238.53 PrivateIPs: 172.31.57.66 X
```

Similarly push master branch.

Git checkout master

Git push origin master

```
[Anshul's-MacBook-Pro:~/Desktop/for-devops-project-main] i-033771ae52c00301d$ git push origin master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)
ubuntutip:172.31.57.66:~/Desktop/for-devops-project-main] i-033771ae52c00301d$ git push origin master
Username for 'https://github.com': Anshuls-repo
Password for 'https://Anshuls-repo@github.com':
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Anshuls-repo/for-devops-project-main.git
 883baf9..d9e8f12 master -> master
ubuntutip:172.31.57.66:~/Desktop/for-devops-project-main] i-033771ae52c00301d (master)
```

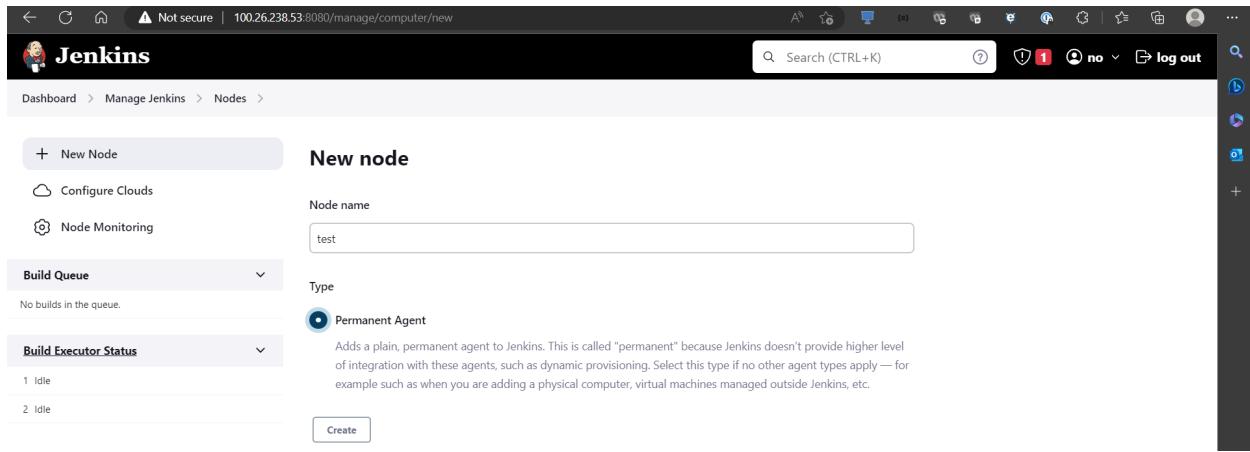
PublicIPs: 100.26.238.53 PrivateIPs: 172.31.57.66

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



New node

Node name: test

Type: Permanent Agent

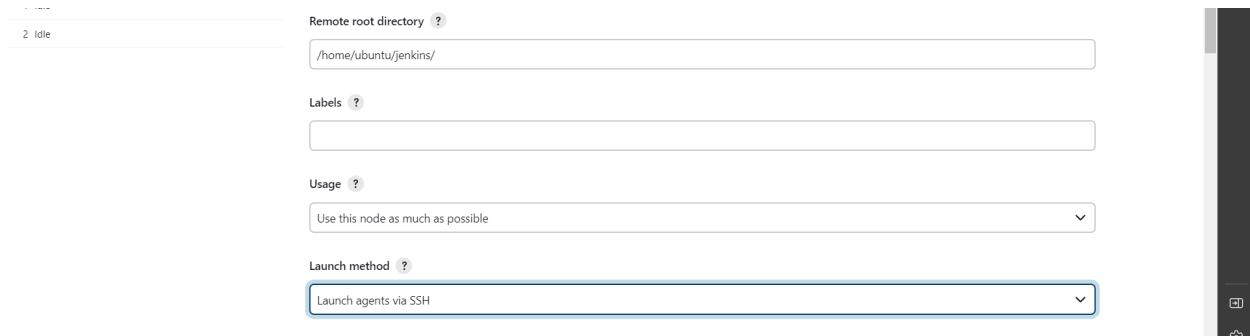
Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.

Create

We are adding Test node first.

Enter our root directory as: /home/ubuntu/jenkins/

And select launch method as: Launch via ssh



Remote root directory: /home/ubuntu/jenkins/

Labels:

Usage: Use this node as much as possible

Launch method: Launch agents via SSH

Give private ip while filling details.

In kind choose: SSH username with Private Key

Jenkins Credentials Provider: Jenkins

Add Credentials

Domain: Global credentials (unrestricted)

Kind: SSH Username with private key

Scope: Global (Jenkins, nodes, items, all child items, etc)

ID: test

Description:

Choose username:ubuntu and paste the keypair details here.

Username: ubuntu

Treat username as secret

Private Key:

Enter directly

Key:

```
-----BEGIN RSA PRIVATE KEY-----
[REDACTED]
-----END RSA PRIVATE KEY-----
```

Passphrase:

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:

Not secure | 100.26.238.53:8080/manage/computer/

Jenkins

Dashboard > Manage Jenkins > Nodes >

+ New Node Manage nodes and clouds Refresh status

Configure Clouds Node Monitoring

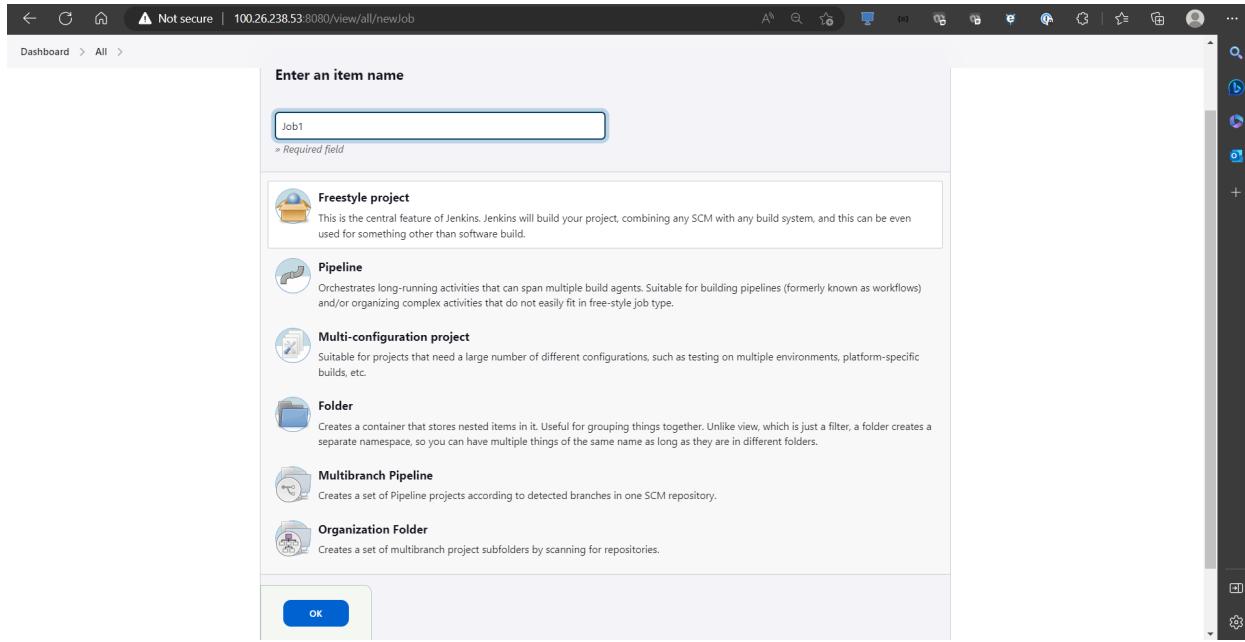
Build Queue Build Executor Status

No builds in the queue.

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
1	Built-In Node	Linux (amd64)	In sync	3.49 GB	0 B	3.49 GB	0ms
2	prod	Linux (amd64)	In sync	4.39 GB	0 B	4.39 GB	55ms
3	test	Linux (amd64)	In sync	4.39 GB	0 B	4.39 GB	4ms

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.

Configure

Description

For test

General

Source Code Management

Build Triggers

Build Environment

Build Steps

Post-build Actions

Discard old builds ?

GitHub project

Project url ?

https://github.com/hshar/website.git

This project is parameterized ?

Throttle builds ?

Execute concurrent builds if necessary ?

Restrict where this project can be run ?

Label Expression ?

test

Label test matches 1 node. Permissions or other restrictions provided by plugins may further reduce that list.

Save Apply

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

The screenshot shows a configuration page for a job named 'Job1'. Under the 'Source Code Management' section, the 'Repository URL' is set to 'https://github.com/Anshuls-repo/for-devops-project-main.git'. The 'Branches to build' section has a 'Branch Specifier' set to '/develop'. The 'Save' and 'Apply' buttons are visible at the bottom.

Now in trigger enable GitHub hook trigger for GITScm polling.

Also enable webhook in git. Payload url format:

http://public_ip:8080/github-webhook/

The screenshot shows the GitHub settings page for a repository. In the 'Webhooks' section, a new webhook is being configured. The 'Payload URL' is set to 'http://100.26.238.53:8080/github-webhook/'. The 'Content type' is set to 'application/x-www-form-urlencoded'. The 'Secret' field is empty. Under 'Which events would you like to trigger this webhook?', the 'Just the push event.' option is selected. A checkbox for 'Active' is checked, and a note says 'We will deliver event details when this hook is triggered.' The 'Add webhook' button is visible at the bottom.

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.

The screenshot shows the Jenkins interface for a project named 'Job1'. On the left, there's a sidebar with options like Status, Changes, Workspace, and Build Now (which is highlighted). In the main area, there's a 'Project Job1' header, a 'test' workspace section, and a 'Permalinks' section with links for Atom feed. On the right, there are buttons for Edit description and Disable Project.

Now in test server:

```
cd jenkins  
cd workspace  
ls
```

You should see Job1 created.

```
17 of these updates are standard security updates.  
To see these additional updates run: apt list --upgradable  
Enable ESM Apps to receive additional future security updates.  
See https://ubuntu.com/esm or run: sudo pro status  
Last login: Wed Mar 1 17:28:40 2023 from 18.206.107.28  
ubuntu@ip-172-31-51-91:~$ ls  
jenkins  
ubuntu@ip-172-31-51-91:~$ cd jenkins  
ubuntu@ip-172-31-51-91:~/jenkins$ ls  
jenkins  
ubuntu@ip-172-31-51-91:~/jenkins$ cd workspace  
ubuntu@ip-172-31-51-91:~/jenkins/workspace$ ls  
Job1  
ubuntu@ip-172-31-51-91:~/jenkins/workspace$  
i-02148c1551d5bd1f2 (test)  
PublicIPs: 18.209.178.190 PrivateIPs: 172.31.51.91
```

Now goto Configuration>>Build Steps and choose Execute Shell.

We give command to build docker file:

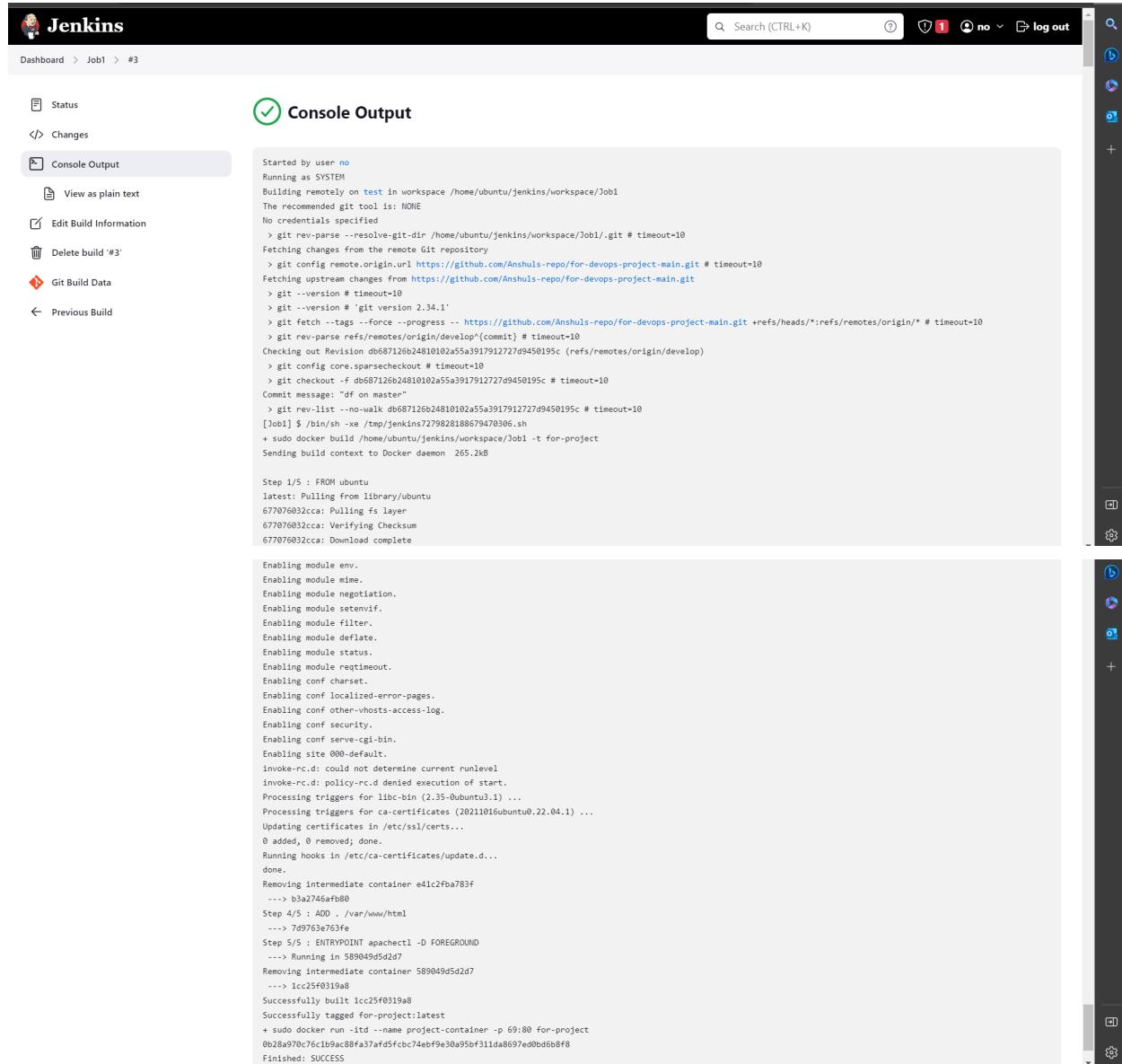
```
sudo docker build /home/ubuntu/jenkins/workspace/Job1 -t devproject  
sudo docker run -itd --name devcontainer -p 81:80 devproject
```

The screenshot shows the Jenkins configuration for 'Job1'. Under 'Build Environment', 'Execute shell' is selected. The 'Command' field contains the Docker build and run commands shown above. At the bottom, there are 'Save' and 'Apply' buttons.

Apply and save.

Then click on build now.

Job1 build is successful.



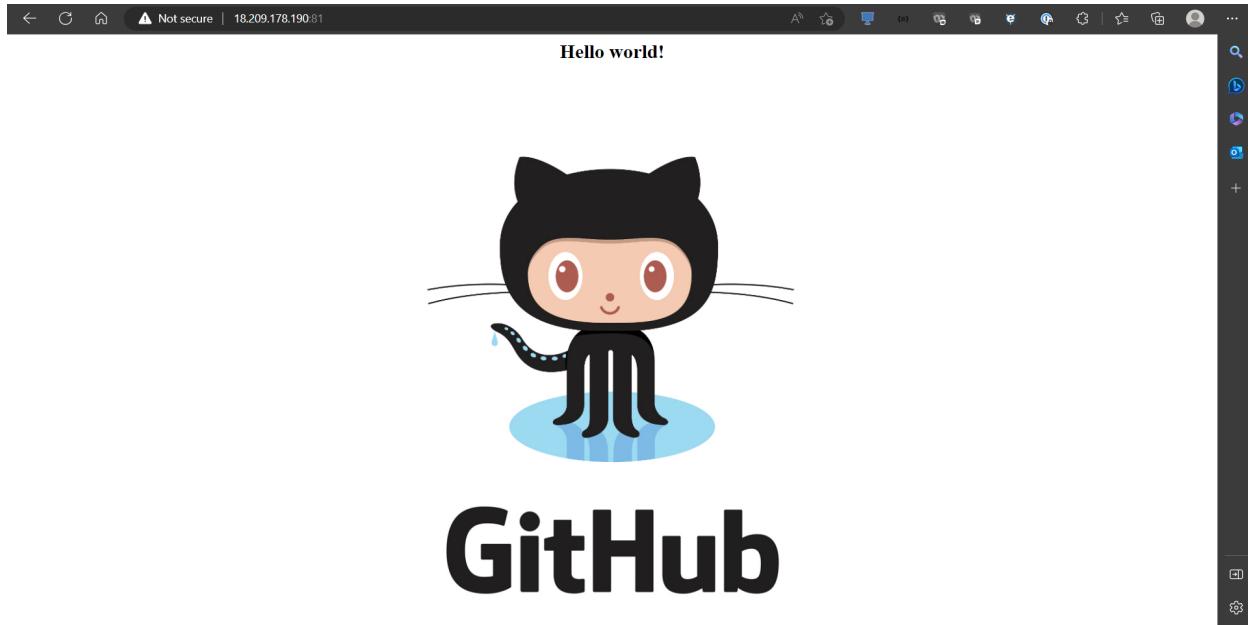
The screenshot shows the Jenkins interface with the 'Console Output' tab selected. The log output is as follows:

```
Started by user no
Running as SYSTEM
Building remotely on test in workspace /home/ubuntu/jenkins/workspace/Job1
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /home/ubuntu/jenkins/workspace/Job1/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/Anshuls-repo/for-devops-project-main.git # timeout=10
Fetching upstream changes from https://github.com/Anshuls-repo/for-devops-project-main.git
> git -version # timeout=10
> git fetch --tags -force -progress -- https://github.com/Anshuls-repo/for-devops-project-main.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/develop^{commit} # timeout=10
Checking out Revision db687126b24810102a55a3917912727d9450195c (refs/remotes/origin/develop)
> git config core.sparsecheckout # timeout=10
> git checkout -f db687126b24810102a55a3917912727d9450195c # timeout=10
Commit message: "df -h master"
> git rev-list --no-walk db687126b24810102a55a3917912727d9450195c # timeout=10
[Job1] $ /bin/sh -c /tmp/jenkins7279828188679470306.sh
+ sudo docker build /home/ubuntu/jenkins/workspace/Job1 -t for-project
Sending build context to Docker daemon 265.2kB

Step 1/5 : FROM ubuntu
latest: Pulling from library/ubuntu
677076032cca: Pulling fs layer
677076032cca: Verifying Checksum
677076032cca: Download complete

Enabling module env.
Enabling module mime.
Enabling module negotiation.
Enabling module setenvif.
Enabling module filter.
Enabling module deflate.
Enabling module status.
Enabling module reqtimeout.
Enabling conf charset.
Enabling conf localized-error-pages.
Enabling conf other-vhosts-access-log.
Enabling conf security.
Enabling conf serve-cgi-bin.
Enabling site 000-default.
invoke-rc.d: could not determine current runlevel
invoke-rc.d: policy-rc.d denied execution of start.
Processing triggers for libc-bin (2.35-0ubuntu3.1) ...
Processing triggers for ca-certificates (20211016ubuntu0.22.04.1) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
Removing intermediate container e41c2fba783f
--> b3a2746af0b0
Step 4/5 : ADD . /var/www/html
--> 7d9763e7679fe
Step 5/5 : ENTRYPOINT apachectl -D FOREGROUND
--> Running in 589049d5d2d7
Removing intermediate container 589049d5d2d7
--> 1cc25f0319a8
Successfully built 1cc25f0319a8
Successfully tagged for-project:latest
+ sudo docker run -itd -n name-project-container -p 69:80 for-project
0b28a970c76c1b9ac88fa37af5fcfc74ebf9e30a95bf311da8697ed0bd6b8f8
Finished: SUCCESS
```

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.

A screenshot of a Jenkins job console. The URL is "100.26.238.53:8080/job/job2/1/console". The left sidebar shows options like Status, Changes, Console Output (which is selected), View as plain text, Edit Build Information, Delete build #1, and Git Build Data. The main area is titled "Console Output" and shows the following log output:

```
Started by user no
Running as SYSTEM
Building remotely on test in workspace /home/ubuntu/jenkins/workspace/Job2
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
  Cloning repository https://github.com/Anshuls-repo/for-devops-project-main.git
    > git init /home/ubuntu/jenkins/workspace/Job2 # timeout=10
    Fetching upstream changes from https://github.com/Anshuls-repo/for-devops-project-main.git
      > git --version # timeout=10
      > git -v
      > git fetch --tags -force -progress -- https://github.com/Anshuls-repo/for-devops-project-main.git +refs/heads/*:refs/remotes/origin/* # timeout=10
      > git config remote.origin.url https://github.com/Anshuls-repo/for-devops-project-main.git # timeout=10
      > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
      Avoid second fetch
      > git rev-parse refs/remotes/origin/master^(commit) # timeout=10
      Checking out Revision db687126b24810102a55a3917912727d9450195c (refs/remotes/origin/master)
      > git config core.sparsecheckout # timeout=10
      > git checkout -f db687126b24810102a55a3917912727d9450195c # timeout=10
      Commit message: "df on master"
      First time build. Skipping changelog.
      Finished: SUCCESS
```

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

A screenshot of a terminal window. The command "ls" is run, showing files "Dockerfile" and "index.html". The command "cat Dockerfile" is run, displaying the Dockerfile content:

```
FROM ubuntu:latest
RUN apt update
RUN apt install apache2 -y
ADD ./var/www/html
ENTRYPOINT apache2 -D FOREGROUND
```

The command "ls" is run again, showing "Dockerfile" and "index.html". The command "cd .." is run, followed by "ls", which shows "Job1" and "Job2". Finally, "i-02148e1551d3bd1f2 (test)" is displayed at the bottom.

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.

The screenshot shows the 'Configure' screen for a new Jenkins job named 'Job3'. Under the 'General' tab, the 'Description' field contains 'final'. The 'Source Code Management' section is set to 'GitHub project' with the 'Project url' set to 'https://github.com/Anshuls-repo/for-devops-project-main.git'. A 'Label Expression' field contains 'prod', which is highlighted with a blue border. Below the configuration are 'Save' and 'Apply' buttons.

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

The screenshot shows the 'Configure' screen for 'Job3' under the 'Source Code Management' tab. It is set to 'Git' with the 'Repository URL' field containing 'https://github.com/Anshuls-repo/for-devops-project-main.git'. The 'Branches to build' field has 'Branch Specifier (blank for "any")' selected. Below the configuration are 'Save' and 'Apply' buttons.

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

The screenshot shows the Jenkins job configuration interface. Under the 'Build Steps' tab, there is one step defined: 'Execute shell'. The command entered is:

```
sudo docker build . -t finalproject
sudo docker run -itd --name finalcontainer -p 80:80 finalprodproject
```

Below this, there is an 'Advanced...' button and a 'Post-build Actions' section which is currently empty.

Click build now.

The screenshot shows the Jenkins job console output for build #2. The output shows the execution of the build steps, including cloning the repository, installing Apache2, and running the Docker container. The output ends with the message:

```
Finalized: success
```

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

The screenshot shows the Jenkins job configuration interface under the 'Post-build Actions' tab. A new action is being added: 'Build other projects'. In the 'Projects to build' field, 'Job3' is selected. Below this, there are three trigger options: 'Trigger only if build is stable' (selected), 'Trigger even if the build is unstable', and 'Trigger even if the build fails'.

Overview:

