

## 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.

Adobe 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. On the left, the navigation pane includes 'New EC2 Experience' (selected), 'EC2 Dashboard', 'EC2 Global View', 'Events', 'Tags', 'Limits', 'Instances' (selected), 'Launch Templates', 'Spot Requests', 'Savings Plans', 'Reserved Instances', 'Dedicated Hosts', 'Scheduled Instances', 'Capacity Reservations', 'Images' (selected), 'AMIs', 'AMI Catalog', 'Elastic Block Store', 'Volumes', 'Snapshots', and 'Feedback'. The main area displays 'Instances (1/3) Info' with three entries:

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

Below this, the 'Instance: i-00dfbb00a4630b2f (master)' details are shown:

Details	Security	Networking	Storage	Status checks	Monitoring	Tags
<b>Instance summary</b>	<b>Info</b>					
Instance ID i-00dfbb00a4630b2f (master)	Public IPv4 address 44.200.242.235   open address	Private IP4 addresses 172.31.12.33				
IPv6 address -	Instance state Running	Public IPv4 DNS ec2-44-200-242-235.compute-1.amazonaws.com   open address				
Hostname type IP name: ip-172-31-12-33.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-12-33.ec2.internal	Elastic IP addresses -				
Answer private resource DNS name IPV4 (A)	Instance type t2.micro					

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

The screenshot shows the Ansible documentation page for Ubuntu. The left sidebar has sections: 'Documentation' (selected), 'Installing Ansible on Ubuntu', 'Installing Ansible on Debian', 'Installing Ansible on Windows', 'Configuring Ansible', 'Ansible Porting Guides' (selected), 'USING ANSIBLE', 'Building Ansible inventories', 'Using Ansible command line tools', and 'Using Ansible playbooks'. The main content is titled 'Installing Ansible on Ubuntu' and contains the following text:

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.**

The screenshot shows a terminal window on the AWS Lambda shell. The output shows the following steps:

```
* 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:~$ 
```

At the bottom, it says:

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.

```
l09y1r = r10c
ubuntu@ip-172-31-50-146:~$ ssh-keygen
Generating public/private rsa key pair.
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:U7wpuDmBu9/Kv0lDp4m8rABabnegTFCGIp0FFMzEMk ubuntu@ip-172-31-50-146
The key's randomart image is:
+---[RSA 3072]---+
|...|
|E.|
|.|.
|.|.
|oo .+ S.|
|oo+ o+.|
|+o+ o+.|
|o+o+ o+.|
|o.o o+.|
+---[SHA256]---+
ubuntu@ip-172-31-50-146:~$ cd .ssh
ubuntu@ip-172-31-50-146:~/ssh$ ls
authorized_keys  id_rsa  id_rsa.pub
ubuntu@ip-172-31-50-146:~/ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAQAAADAAAQABACQgku0u5cnLhpvIdPx483uAI6VeXRslpXdp9DPfL73m/mH8yV7C8M369DhkqUm962x46Rb2VeF8Ag6xt+82xxFFj62gn6e5g2qTCJjOLQVfa6Wpc90HC2LBun7tuGKdsOzwNQ2rbUAws8JHid12t9mDc7i22LqWNdw5nB1U4PHI7UW2T9urajn88sYrd1qn3z1k286YNsf8uBg2COdA+g4D24/wd50XBWhdyuM1t4k3e2Ay4ci3sDnS4X5yq3VFLKtpq76ye3Y9fiFLRZE2Eut9twOZNZSMVwhONovyk1zsq/NyqlPjpOKPQt2uNO50iVu/roc5q9jPLdzyd devopsproject
ubuntu@ip-172-31-50-146:~/ssh$ cat /home/ubuntu/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAQAAADAAAQABACQgku0u5cnLhpvIdPx483uAI6VeXRslpXdp9DPfL73m/mH8yV7C8M369DhkqUm962x46Rb2VeF8Ag6xt+82xxFFj62gn6e5g2qTCJjOLQVfa6Wpc90HC2LBun7tuGKdsOzwNQ2rbUAws8JHid12t9mDc7i22LqWNdw5nB1U4PHI7UW2T9urajn88sYrd1qn3z1k286YNsf8uBg2COdA+g4D24/wd50XBWhdyuM1t4k3e2Ay4ci3sDnS4X5yq3VFLKtpq76ye3Y9fiFLRZE2Eut9twOZNZSMVwhONovyk1zsq/NyqlPjpOKPQt2uNO50iVu/roc5q9jPLdzyd devopsproject
ubuntu@ip-172-31-50-146:~/ssh$ i-09d75cc8927659975 (master)
PublicIP: 54.152.58.47 PrivateIP: 172.31.50.146
```

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

```
l09y1r = r10c
Reading state information... Done
28 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu@ip-172-31-53-181:~$ cd .ssh
ubuntu@ip-172-31-53-181:~/ssh$ sudo nano authorized_keys
ubuntu@ip-172-31-53-181:~/ssh$ i-0e4cebe4ad9165398 (test)
PublicIP: 35.174.209.116 PrivateIP: 172.31.53.181
```

```
l09y1r = r10c
https://us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh?region=us-east-1&connType=standard&insta...
AWS Services Search [Alt+S]
authorized_keys +
GNU nano 6.2
ssh-rsa AAAAB3NzaC1yc2EAAQAAADAAAQABACQgku0u5cnLhpvIdPx483uAI6VeXRslpXdp9DPfL73m/mH8yV7C8M369DhkqUm962x46Rb2VeF8Ag6xt+82xxFFj62gn6e5g2qTCJjOLQVfa6Wpc90HC2LBun7tuGKdsOzwNQ2rbUAws8JHid12t9mDc7i22LqWNdw5nB1U4PHI7UW2T9urajn88sYrd1qn3z1k286YNsf8uBg2COdA+g4D24/wd50XBWhdyuM1t4k3e2Ay4ci3sDnS4X5yq3VFLKtpq76ye3Y9fiFLRZE2Eut9twOZNZSMVwhONovyk1zsq/NyqlPjpOKPQt2uNO50iVu/roc5q9jPLdzyd devopsproject
ubuntu@ip-172-31-50-146:~$ l09y1r = r10c
Help Write Out Where Is Cut Execute Location Undo Set Mark To Bracket Previous Back
Exit Read File Replace Paste Go To Line Redo Copy Where Was Next Forward
l09y1r = r10c
PublicIP: 35.174.209.116 PrivateIP: 172.31.53.181
```

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:

```
aws Services Search [Alt+S] N.Virginia Anshul@135                                                                                                                                                                                                                                                                                                                                                                                                                                           <
```

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**

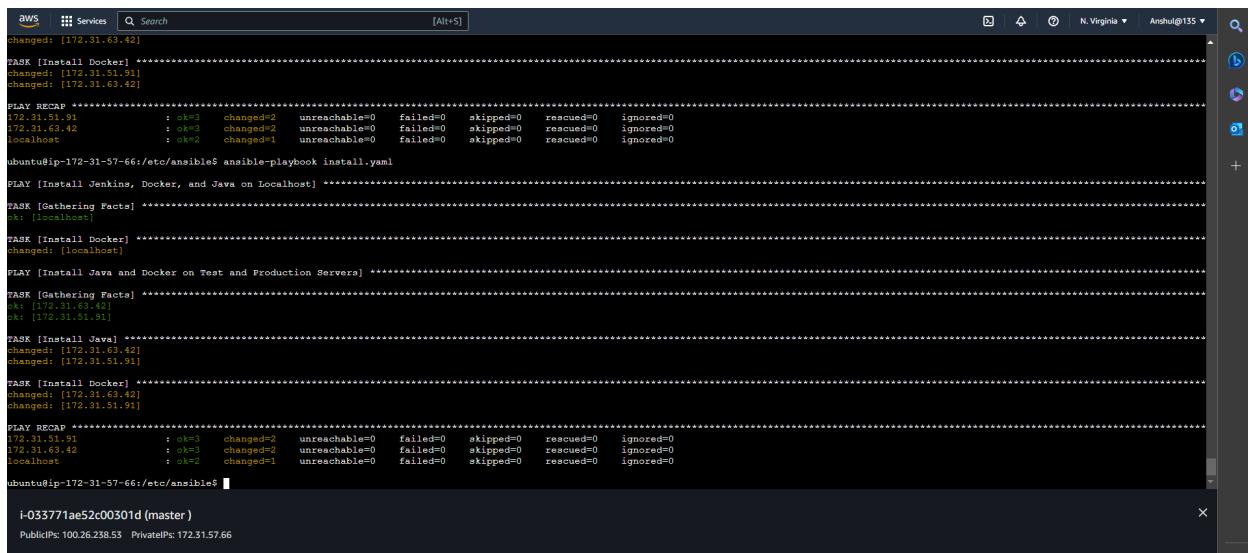
```
[WARNING]: Could not match supplied host pattern; ignoring: master
playbook: install.yaml
ubuntu@ip-172-31-50-146:/etc/ansible$ sudo nano install.yaml
ubuntu@ip-172-31-50-146:/etc/ansible$ sudo nano install.yaml
ubuntu@ip-172-31-50-146:/etc/ansible$ ansible-playbook install.yaml --syntax-check
playbook: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

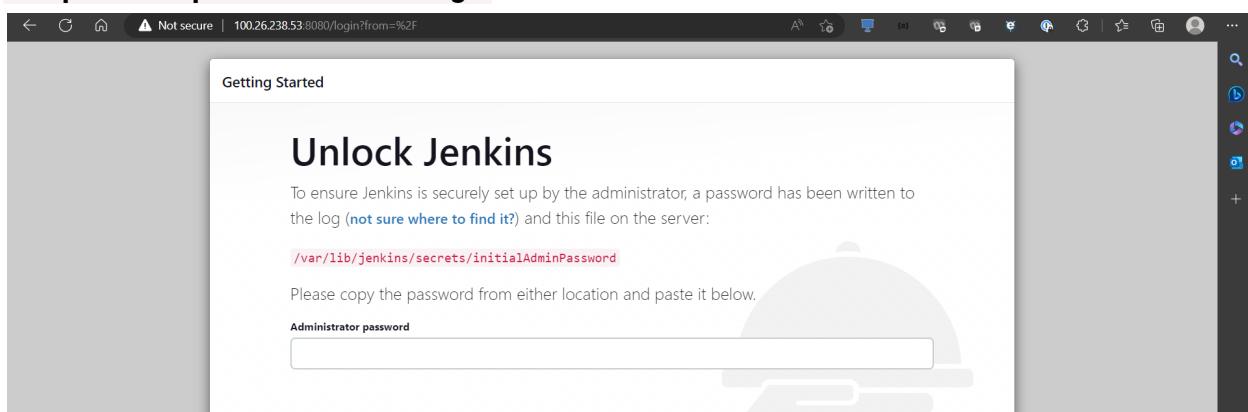


```
aws Services Search [Alt+S]
changed: [172.31.63.42]
TASK [Install Docker] ****
changed: [172.31.51.91]
changed: [172.31.63.42]
PLAY RECAP ****
172.31.51.91 : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
172.31.63.42 : ok=3    changed=2    unreachable=0    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
```

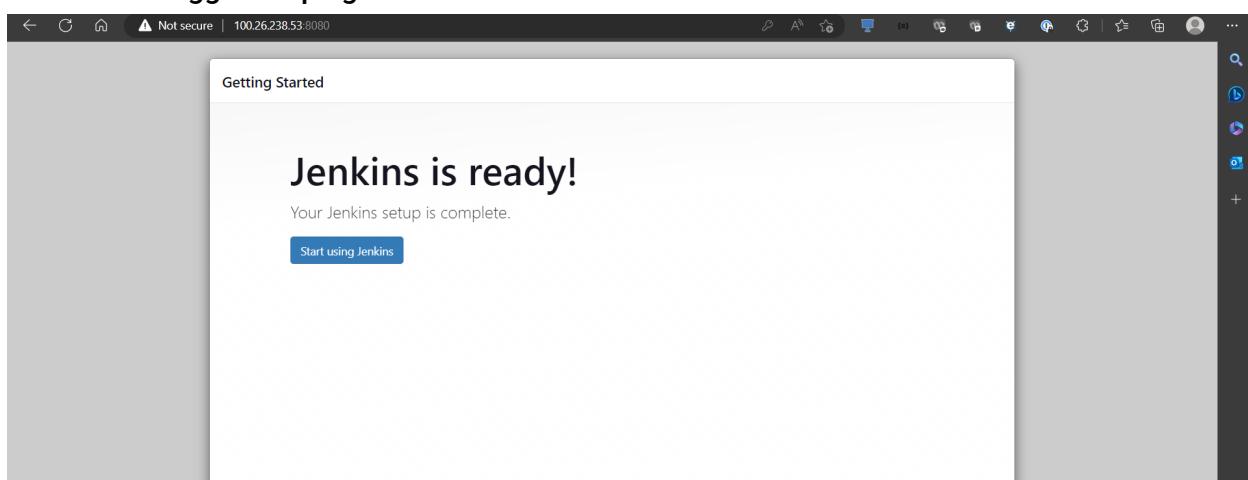
Let us login to jenkins from master now. Copy paste public\_ip:8080.

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  
Read File Replace Undo Redo Copy 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:~/For-devops-project-main$ 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-defined 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:~/For-devops-project-main$  
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': Anshula-repo  
Password for 'https://Anshula-repo@github.com':  
Enumerating objects: 3, done.  
Counting objects: 100% (3/3), 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/Anshula-repo/for-devops-project-main/pull/new/develop  
remote:  
To https://github.com/Anshula-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**

```
[anuhuls@i-033771ae52c00301d ~]$ git push origin master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)
ubuntutest:~/For-DevOpsProject$ git push origin master
Username for 'https://github.com': Anuhuls-repo
Password for 'https://Anuhuls-repo@github.com':
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Anuhuls-repo/for-devops-project/main.git
   883b698..d9e8712 master -> master
ubuntutest:~/For-DevOpsProject$ i-033771ae52c00301d (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**

The screenshot shows the Jenkins 'New node' configuration page. The 'Node name' field contains 'test'. The 'Type' dropdown is set to 'Permanent Agent', with a note explaining it adds a plain, permanent agent to Jenkins. The 'Create' button is at the bottom.

**We are adding Test node first.**

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

**And select launch method as: Launch via ssh**

The screenshot shows the Jenkins 'New node' configuration page. The 'Launch method' dropdown is set to '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: [REDACTED]

**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 >

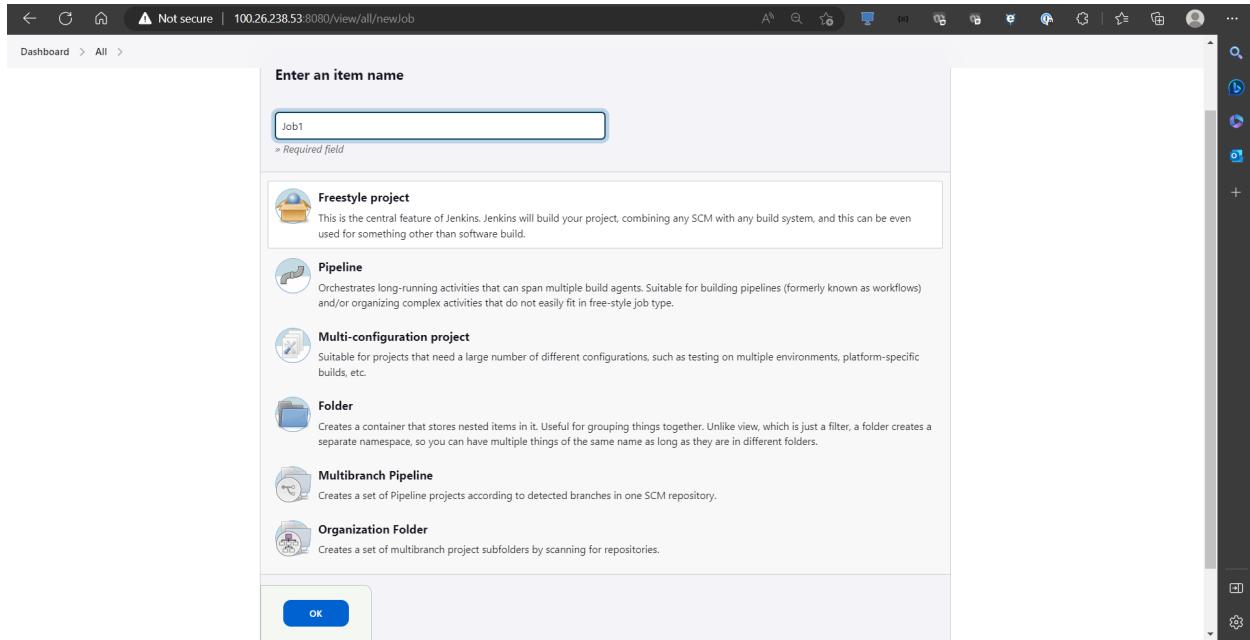
+ New Node      Manage nodes and clouds      Refresh status

Configure Clouds      Node Monitoring

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.**

A screenshot of the Jenkins configuration screen for 'Job1'. The title bar says 'Not secure | 100.26.238.53:8080/job/job1/configure'. The left sidebar shows tabs: General (selected), Source Code Management, Build Triggers, Build Environment, Build Steps, and Post-build Actions. The main area has a 'Configure' heading. Under 'General', there is a 'Description' field with 'For test'. Under 'GitHub project', there is a 'Project url' field with 'https://github.com/hshar/website.git'. Under 'Advanced...', there are several checkboxes: 'Discard old builds', 'GitHub project' (which is checked), 'Project url', 'This project is parameterized', 'Throttle builds', 'Execute concurrent builds if necessary', and 'Restrict where this project can be run' (which is checked). Below these is a 'Label Expression' field with 'test'. A note below it says 'Label test matches 1 node. Permissions or other restrictions provided by plugins may further reduce that list.' At the bottom are 'Save' and 'Apply' buttons.

**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 build configuration page. Under the 'Source Code Management' section, a GitHub repository URL is entered as `https://github.com/Anshuls-repo/for-devops-project-main.git`. The 'Branches to build' field contains the specifier `*/develop`, indicating that the build should trigger on any push to the develop branch. 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/](http://public_ip:8080/github-webhook/)**

The screenshot shows the GitHub repository settings for `for-devops-project-main`. In the 'Webhooks' section, a new webhook is being configured. The 'Payload URL' is set to `http://100.26.238.53:8080/github-webhook/`, and 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 states 'We will deliver event details when this hook is triggered.' The 'Add webhook' button is 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". The "Build Now" button is highlighted in yellow. Other options like "Status", "Changes", "Workspace", "Configure", "Delete Project", "GitHub Hook Log", "GitHub", and "Rename" are also visible. On the right, there are links for "Edit description" and "Disable Project". Below the main menu, there's a "Build History" section showing a single build from March 1, 2023, at 6:56 PM. The log entry is: "Atom feed for all Atom feed for failures".

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  Job1  workspace  
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
```



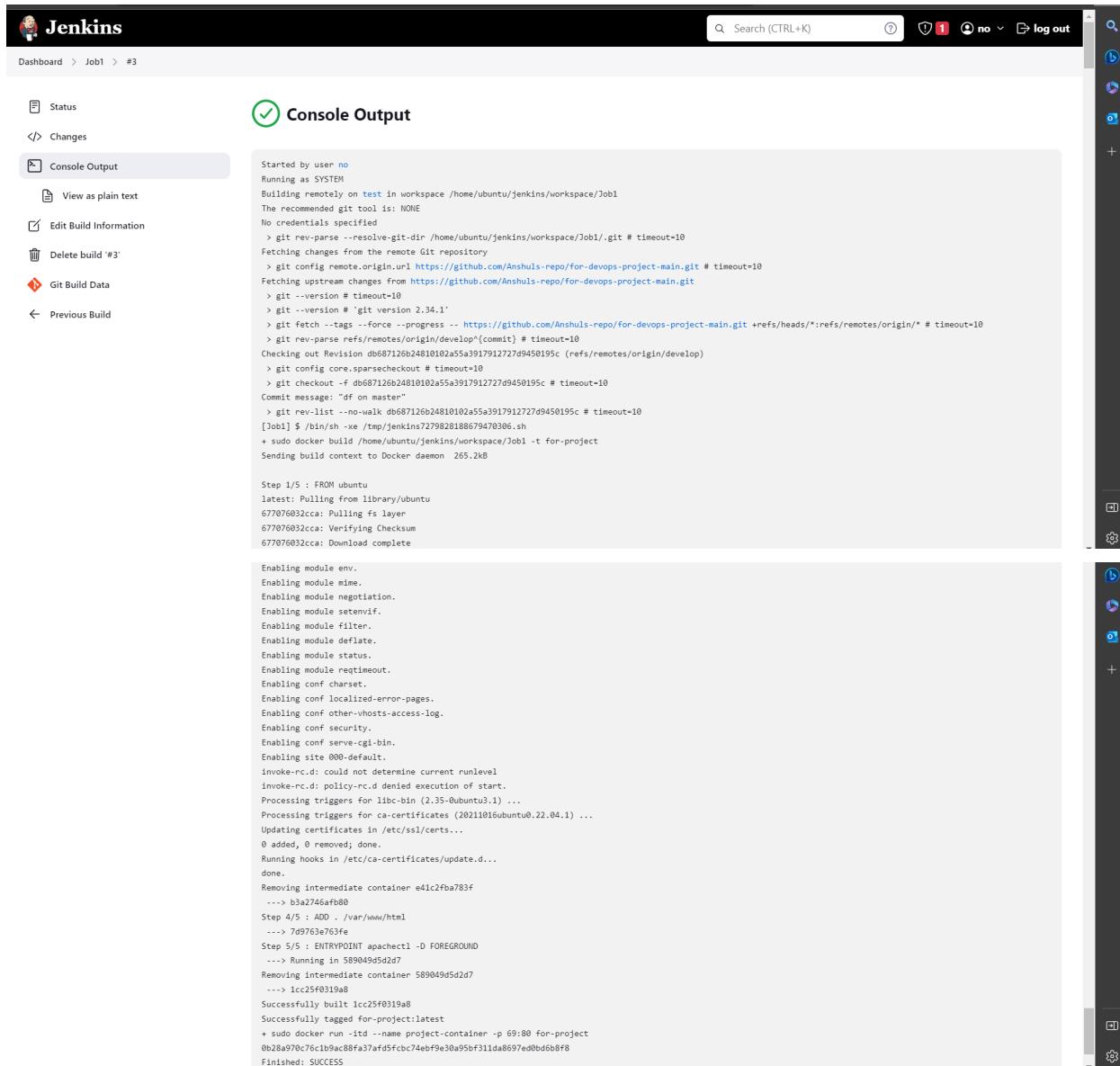
The screenshot shows the Jenkins configuration for "Job1" under the "Build Steps" tab. An "Execute shell" step is selected, and its "Command" field contains the following Docker commands:  

```
sudo docker build /home/ubuntu/jenkins/workspace -t for-project  
sudo docker run -itd --name project-container -p 69:80 for-project
```

**Apply and save.**

**Then click on build now.**

**Job1 build is successful.**



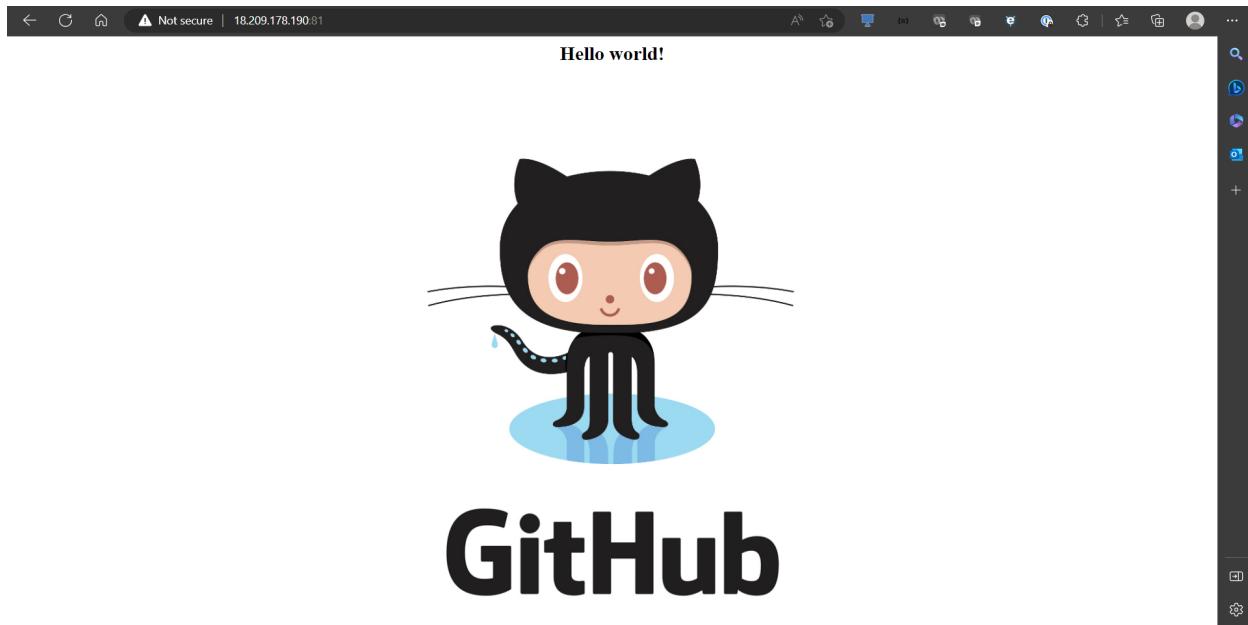
The screenshot shows the Jenkins interface with the job "Job1" successfully completed. The console output tab is selected, displaying the build logs. The logs show the build process starting, cloning from a GitHub repository, fetching upstream changes, and performing a Docker build step. The Docker build step details the creation of a container, pulling the "library/ubuntu" image, and running Apache on port 80. The build concludes with a "SUCCESS" status.

```
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 on 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
--> b3a2746af080
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
0b28a970c76c1b9ac88fa37af5fcfc74ebf9e30a95bf311da8697ed0bd6b8f
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 output. The URL is "100.26.238.53:8080/job/job2/1/console". The page title is "Jenkins". The main content area is titled "Console Output" with a green checkmark icon. It shows the command-line output of a Jenkins build for job "Job2". The log includes commands like "git clone", "git fetch", and "git checkout", along with a commit message and a "Finished: SUCCESS" message. The left sidebar has links for Status, Changes, Console Output (which is selected), View as plain text, Edit Build Information, Delete build #1, and Git Build Data.

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 in the directory "/jenkins/workspace/Job1S", showing files "Dockerfile", "index.html", and "ubuntu8.ip-172-31-51-91~.jenkins/workspace/Job1S". The command "cat Dockerfile" is run, displaying the Dockerfile content which includes "FROM ubuntu:16.04", "RUN apt-get update", "RUN apt install apache2 -y", and "EXPOSE 80". The command "ls" is run again in the workspace directory, showing "Dockerfile", "index.html", and "ubuntu8.ip-172-31-51-91~.jenkins/workspace/Job1S". The command "cd .." is run, followed by "ls" which shows "Job1 Job2". The command "ls" is run again in the workspace directory, showing "Dockerfile", "index.html", and "ubuntu8.ip-172-31-51-91~.jenkins/workspace/Job1S". The command "i-02148e1551d3bd1f2 (test)" is run, followed by "PublicIP: 18.209.178.190 PrivateIP: 172.31.51.91".

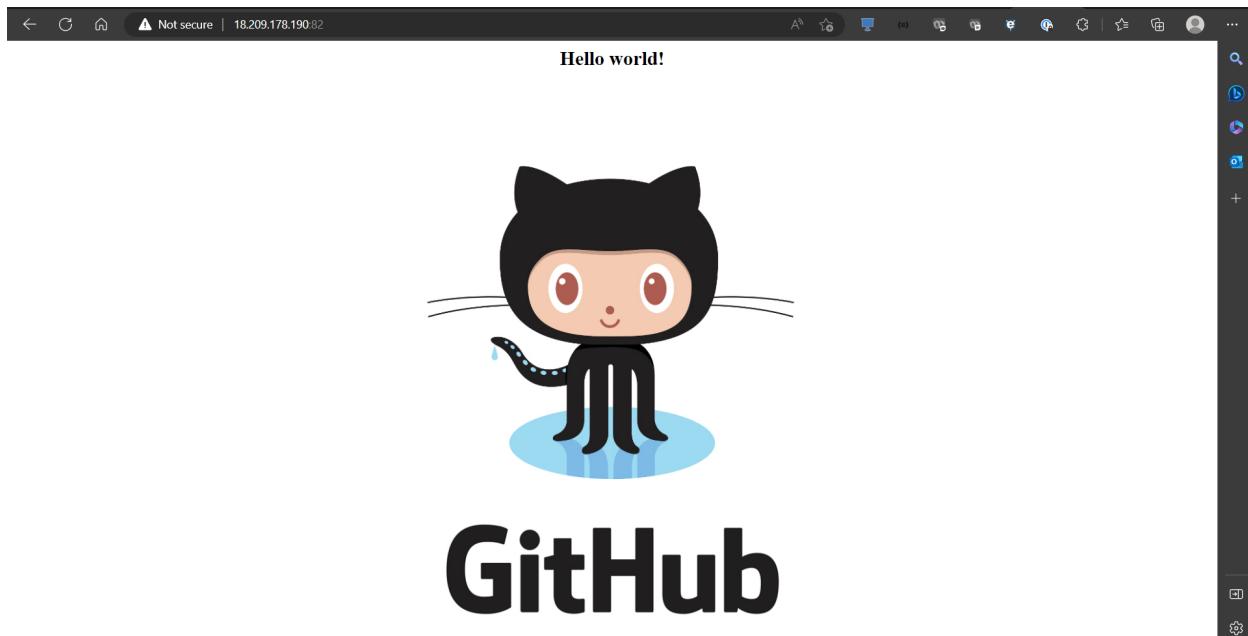
**Now goto: configuration>build steps and select Execute shell.**

**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 job named 'Job3'. Under the 'General' tab, the 'Description' field contains 'final'. The 'Source Code Management' tab is selected, showing a 'GitHub project' configuration. The 'Project url' is set to 'https://github.com/Anshuls-repo/for-devops-project-main.git'. In the 'Label Expression' field, the value 'prod' is entered, which is highlighted with a blue border. Below the label expression, a note states 'Label prod matches 1 node. Permissions or other restrictions provided by plugins may further reduce that list.' At the bottom of the screen 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, the 'Git' option is selected. The 'Repository URL' field contains 'https://github.com/Anshuls-repo/for-devops-project-main.git'. The 'Branches to build' field is empty, containing 'Branch Specifier (blank for "any")'. At the bottom of the screen 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 for a job named 'Job3'. The left sidebar has tabs for General, Source Code Management, Build Triggers, Build Environment, Build Steps (which is selected), and Post-build Actions. The main content area is titled 'Build Steps' and contains a section for 'Execute shell'. It shows a command box with the following Docker run command:

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

Below the command box are 'Advanced...' and 'Add build step' buttons. The 'Post-build Actions' section at the bottom has an 'Add post-build action' button. At the bottom right are 'Save' and 'Apply' buttons.

**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:

**Build Steps**

**Post-build Actions**

**Build other projects** ? x

Projects to build

① No such project 'Job'. Did you mean 'Job1'?

Trigger only if build is stable

Trigger even if the build is unstable

Trigger even if the build fails

[Add post-build action ▾](#)

**Overview:**

