

## Project : Capstone II

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

The product is present on: <https://github.com/hshar/website.git>

Following are the specifications of the lifecycle:

1. Git workflow should be implemented. Since the company follows a monolithic architecture of development, you need to take care of version control. The release should happen only on the 25th of every month.
  2. CodeBuild should be triggered once the commits are made in the master branch.
  3. The code should be containerized with the help of the Dockerfile. The Dockerfile should be built every time if there is a push to GitHub. Create a custom Docker image using a Dockerfile.
  4. As per the requirement in the production server, you need to use the Kubernetes cluster and the containerized code from Docker Hub should be deployed with 2 replicas. Create a NodePort service and configure the same for port 30008.
  5. Create a Jenkins Pipeline script to accomplish the above task.
  6. For configuration management of the infrastructure, you need to deploy the configuration on the servers to install necessary software and configurations.
  7. Using Terraform, accomplish the task of infrastructure creation in the AWS cloud provider.
- Architectural Advice: Softwares to be installed on the respective machines using configuration management.

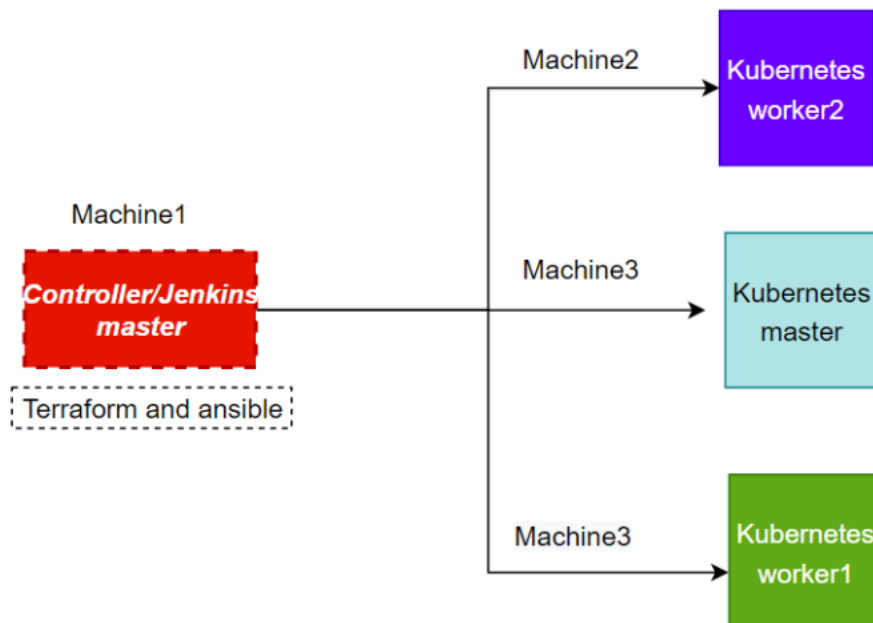
Worker1: Jenkins, Java

Worker2: Docker, Kubernetes

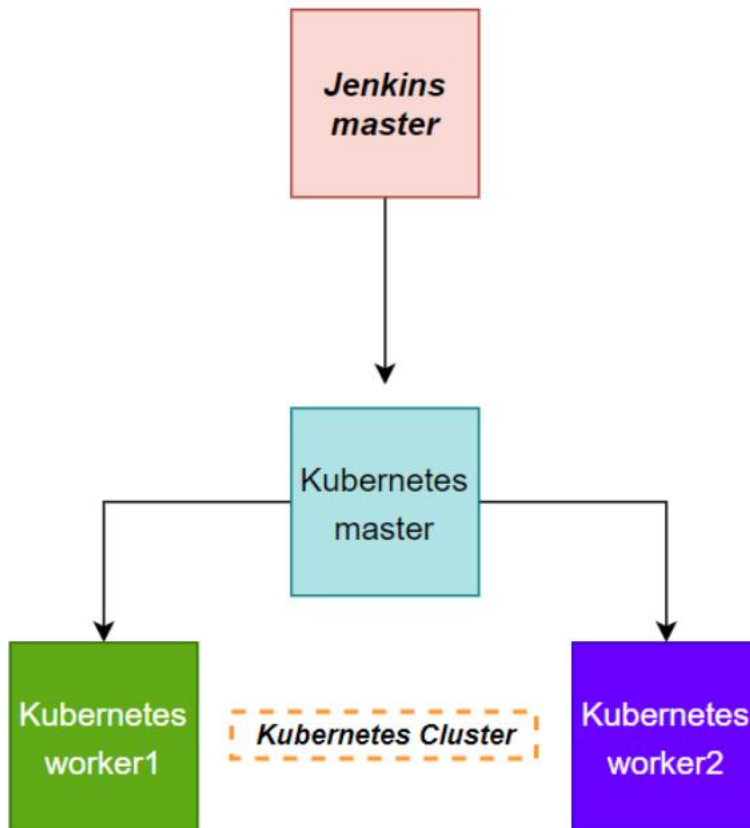
Worker3: Java, Docker, Kubernetes

Worker4: Docker, Kubernetes

## Infrastructure Creation and Configuration Management



***Servers for jenkins and kubernetes configuration***



## Terraform installation

- Created a EC2 instance manually in aws console
- Navigated into the EC2 instance and installed terraform through a shellsript

```
ubuntu@worker1:~$ vi terraforminstall.sh
ubuntu@worker1:~$ bash terraforminstall.sh
```

i-00b8547948a9cfa59 (Worker1)

PublicIPs: 3.141.13.91 PrivateIPs: 172.31.26.221

```
sudo apt-get update && sudo apt-get install -y gnupg software-properties-common
wget -O- https://apt.releases.hashicorp.com/gpg | \
gpg --dearmor | \
sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg
gpg --no-default-keyring \
--keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \
--fingerprint
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | \
sudo tee /etc/apt/sources.list.d/hashicorp.list
sudo apt update
sudo apt-get install terraform -y
~
~
~
~
~
~
~
~
"terraforminstall.sh" 12L, 548B
```

## Architecture creation

After installation of terraform created a directory named infracreation

- Navigated inside the directory and created a main.tf file with a hcl script to create three EC2 instances

```
ubuntu@worker1:~$ mkdir infracreation
ubuntu@worker1:~$ cd infracreation
ubuntu@worker1:~/infracreation$ vi main.tf
```

```
provider "aws" {
  secret_key = "Pr7tQthXHgkb3dgT+S1010wfwkQPaDrusDdD8Rf8"
  access_key = "AKIAYJ5MROACCA2BOW5E"
  region = "us-east-2"
  alias = "worker2"
}

provider "aws" {
  secret_key = "Pr7tQthXHgkb3dgT+S1010wfwkQPaDrusDdD8Rf8"
  access_key = "AKIAYJ5MROACCA2BOW5E"
  region = "us-east-2"
  alias = "worker3"
}

provider "aws" {
  secret_key = "Pr7tQthXHgkb3dgT+S1010wfwkQPaDrusDdD8Rf8"
  access_key = "AKIAYJ5MROACCA2BOW5E"
  region = "us-east-2"
  alias = "worker4"
}
```

```
resource "aws_instance" "this2" {
  ami = "ami-05fb0b8c1424f266b"
  instance_type = "t2.medium"
  key_name = "venkatohio"
  provider = aws.worker2
  tags = {
    Name = "worker2"
  }
}
```

```
resource "aws_instance" "this3" {
  ami = "ami-05fb0b8c1424f266b"
  instance_type = "t2.medium"
  key_name = "venkatohio"
  provider = aws.worker3
  tags = {
    Name = "worker3"
  }
}
```

```
resource "aws_instance" "this4" {
  ami = "ami-05fb0b8c1424f266b"
  instance_type = "t2.medium"
  key_name = "venkatohio"
  provider = aws.worker4
  tags = {
    Name = "worker4"
  }
}
```

- Then initialized terraform

```
ubuntu@worker1:~/infrastructure$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.31.0...
- Installed hashicorp/aws v5.31.0 (signed by HashiCorp)

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

Terraform has been successfully initialized!

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

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
ubuntu@worker1:~/infrastructure$
```

- Performed terraform plan

```
ubuntu@worker1:~/infrastructure$ terraform plan

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

Terraform will perform the following actions:

# aws_instance.this2 will be created
+ resource "aws_instance" "this2" {
  + ami              = "ami-05fb0b8c1424f266b"
  + arn              = (known after apply)
  + associate_public_ip_address = (known after apply)
  + availability_zone = (known after apply)
  + cpu_core_count   = (known after apply)
  + cpu_threads_per_core = (known after apply)
  + disable_api_stop  = (known after apply)
  + disable_api_termination = (known after apply)
  + ebs_optimized     = (known after apply)
  + get_password_data = false
  + host_id           = (known after apply)
}
```

- And then applied the changes by terraform apply command

```
aws_instance.this3: Creating...
aws_instance.this4: Creating...
aws_instance.this2: Creating...
aws_instance.this3: Still creating... [10s elapsed]
aws_instance.this4: Still creating... [10s elapsed]
aws_instance.this2: Still creating... [10s elapsed]
aws_instance.this3: Still creating... [20s elapsed]
aws_instance.this4: Still creating... [20s elapsed]
aws_instance.this2: Still creating... [20s elapsed]
aws_instance.this3: Still creating... [30s elapsed]
aws_instance.this4: Still creating... [30s elapsed]
aws_instance.this2: Still creating... [30s elapsed]
aws_instance.this3: Creation complete after 31s [id=i-07edd73dde8245ebf]
aws_instance.this4: Creation complete after 31s [id=i-03b429699eedc391e]
aws_instance.this2: Still creating... [40s elapsed]
aws_instance.this2: Still creating... [50s elapsed]
aws_instance.this2: Creation complete after 52s [id=i-0b31db3012c176c15]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
ubuntu@worker1:~/infrastructure$
```

## Ansible configuration

- Then installed ansible in the server through a ansibleinstall.sh shell script file

```
GNU nano 6.2                                ansibleinstall.sh
sudo apt update
sudo apt install software-properties-common
sudo apt-add-repository ppa: ansible/ansible
sudo apt install ansible

[ Read 4 lines ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Und
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line M-E Red
```

Ansible is successfully installed

```
ubuntu@worker1:~$ ansible --version
ansible 2.10.8
  config file = None
  configured module search path = ['/home/ubuntu/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
ubuntu@worker1:~$
```

- Ssh key is generated

```
ubuntu@worker1:~$ 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:rSszP1TSJq9shqpp9O+E2D2pfoOyS920KCG421+fwCE ubuntu@worker1
The key's randomart image is:
+----[RSA 3072]-----+
|
|
|          .          |
| .      o.+         |
| o . E o S*.        |
| o.= O +...         |
| ..+.=.@+...        |
| +ooo+B+*o          |
| ..*B==oO=.         |
+----[SHA256]-----+
```

- Ssh private key is copied

```
ubuntu@worker1:~$ sudo cat /home/ubuntu/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaClyc2EAAAADAQABAAQgQCxVi0IcrTODX1Kags427G+1zh5PnV/EICAQDk+aJKqEswZDR4NCQiFPYhPdTNjAsUCs7qR4berZwxAAds7tvZ2o3oiiki20uud0imjN
hULp0okVKwCAtOum66IVw/3IwInmb70vKozc5e+H12Htw3HMBf7Cvc1+xx/QYFE8IXUJ9yIQ5C03BYARqi++aB5r63DP4VGj5Qb67nHEzA9IM9zjIaFILE/7wV1wbyEX6vPCP7Ruitp+
L/6IewxGP0HgClT+ew09rTCCB4jaB9FpZvGwqPI2O0VJ3q7n0pwgHXv8nTWdNngZDTQ8r0ulDTQdQDnxkNA+x10Qq5kX90QBP2dzzeb0uQ0ho9KOh+GSnNeIzV6kwxID+eF7t+Kv5QyZ
HUmSmGJ0sXRPv5ILsMqd00DN53Lcr27Lu0Y6J7cqQgQ1/mXsek1NNUTpmfS3AO81IKtRe0DwRhB1jbx5q1L2L0qRe+RZkfkLoXsgZ7GalzvTs58Let6IDjIh4fscdUHM= ubuntu@work
er1
ubuntu@worker1:~$
```

- Navigated to the server worker2 and opened authorized keys file and pasted ssh key generated in worker 1 to establish connection between the two

Similarly same steps performed in worker 3 and worker4

```

GNU nano 6.2                               ~/.ssh/authorized_keys *
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCFKI/ORqXVvGT4Pdb/6/VH4BmPA9uryj2vGHFYASyrhbo0V+2sln0btYcRcAaD
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgQCxVi0IcrTODX1Kags427G+1Zh5PnV/EICAQDk+aJKqEswZDR4NQiFPYhPdTNjAs

^G Help      ^O Write Out  ^W Where Is   ^R Cut        ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/_ Go To Line M-E Redo

i-0b31db3012c176c15 (worker2)
PublicIPs: 18.116.61.112 PrivateIPs: 172.31.20.162

```

Navigated back to the main server ie., worker1 and created a file named inventory and added private IPs of the worker 2 3 & 4 servers

- Chosen worker3 as a kubernetes master and jenkins slave node
- Remaining two servers as kubernetes slaves

```

GNU nano 6.2                               inventory
[[k8Master]]
worker3 ansible_host=172.31.20.131
[[k8slaves]]
worker2 ansible_host=172.31.20.162
worker4 ansible_host=172.31.19.234

[ Read 5 lines ]

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/_ Go To Line M-

i-00b8547948a9cfa59 (Worker1)
PublicIPs: 3.141.13.91 PrivateIPs: 172.31.26.221

```



- With that ansible cluster setup is completed. Ensured connection between all servers by performing ansible ping

```
ubuntu@worker1:~$ sudo nano inventory
ubuntu@worker1:~$ sudo nano inventory
ubuntu@worker1:~$ ansible -i inventory all -m ping
The authenticity of host '172.31.20.131 (172.31.20.131)' can't be established.
ED25519 key fingerprint is SHA256:xlB5RiGAGABPN0R0DbVSz2KIO59if0V27dyio4nbkU0.
This key is not known by any other names

The authenticity of host '172.31.19.234 (172.31.19.234)' can't be established.
ED25519 key fingerprint is SHA256:Zkel/1vl4HezAbYz0UxAClDr8T0CtPQfBwmuDVsGypA.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? worker2 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
yes
worker3 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
yes
worker4 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
ubuntu@worker1:~$
```

### Configuration deployments

- After that a shell script named localhost.sh is created to install java and jenkins in localhost(worker1)

```
GNU nano 6.2                                localhost.sh
sudo apt update
sudo apt install openjdk-11-jdk -y
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
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 -y
```

- Similarly another shellscript worker3.sh is created to install java docker and kubernetes in worker3

```

GNU nano 6.2 worker3.sh
sudo apt update
sudo apt install openjdk-11-jdk -y
sudo apt update
sudo apt install docker.io -y
sudo apt update -y

sudo apt install curl apt-transport-https -y
curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --dearmor -o /etc/apt/trusted.gpg.d/k8s.gpg
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt update -y
sudo apt install kubelet kubeadm kubectl -y
sudo apt-mark hold kubelet kubeadm kubectl

sudo swapoff -a
sudo sed -i 's/^.*$/\1/g' /etc/fstab
sudo tee /etc/modules-load.d/k8s.conf <<EOF

[ Read 46 lines ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-R Set Mark  M-J To Bracket
^X Exit      ^S Read File  ^R Replace    ^U Paste      ^J Justify    ^_ Go To Line  M-E Redo      M-C Copy      ^G Where Was

```

- Finally last shellscript is created to install docker and kubernetes on worker2 and worker4

```

GNU nano 6.2 slaves.sh *
sudo apt update
sudo apt install docker.io -y
sudo apt update -y

sudo apt install curl apt-transport-https -y
curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --dearmor -o /etc/apt/trusted.gpg.d/k8s.gpg
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt update -y
sudo apt install kubelet kubeadm kubectl -y
sudo apt-mark hold kubelet kubeadm kubectl

sudo swapoff -a
sudo sed -i 's/^.*$/\1/g' /etc/fstab
sudo tee /etc/modules-load.d/k8s.conf <<EOF
overlay

```

- An ansible playbook is created to execute these shellscripts in their respective servers/hosts

```

---
- hosts: localhost
  name: installing jenkins and java
  become: yes
  tasks:
    - name: executing localhost.sh
      script: localhost.sh
- hosts: k8sMaster
  name: installing java docker and kubernetes
  become: yes
  tasks:
    - name: executing worker3.sh
      script: worker3.sh
- hosts: k8slaves
  name: installing docker and kubernetes
  become: yes
  tasks:
    - name: executing slaves.sh
      script: slaves.sh
~
"playbook.yml" 19L, 427B

```

- The playbook is executed successfully resulting the installations in the target host servers

```

    "No VM guests are running outdated hypervisor (qemu) binaries on this host.",
    "kubelet set on hold.",
    "kubeadm set on hold.",
    "kubectl set on hold.",
    "net.bridge.bridge-nf-call-iptables=1",
    "",
    "net.bridge.bridge-nf-call-iptables = 1",
    "Synchronizing state of docker.service with SysV service script with /lib/systemd/systemd-sysv-install.",
    "Executing: /lib/systemd/systemd-sysv-install enable docker"
  ]
}
META: ran handlers
META: ran handlers

PLAY RECAP *****
localhost      : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
worker2        : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
worker3        : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
worker4        : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
ubuntu@worker1:~$

```

## Kubernetes cluster configuration

Navigated to the kubernetes master and initialised kubeadm

```

ubuntu@ip-172-31-20-131:~$ sudo kubeadm init --apiserver-advertise-address=172.31.20.131 --pod-network-cidr=10.244.0.0/16
I0107 06:18:34.581405 50266 version.go:256] remote version is much newer: v1.29.0; falling back to: stable-1.28
[init] Using Kubernetes version: v1.28.5
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'

```

Generated kubeadm token is copied

```

Alternatively, if you are the root user, you can run:

export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 172.31.20.131:6443 --token fe9ej1.gd40acezkaqa0st9 \
--discovery-token-ca-cert-hash sha256:7428b54a21823b234b8af394bdb4863597695b345ae2e4655f5773731f1ff59d
ubuntu@ip-172-31-20-131:~$

```

kubeadm join 172.31.20.131:6443 --token fe9ej1.gd40acezkaqa0st9 \

--discovery-token-ca-cert-hash

sha256:7428b54a21823b234b8af394bdb4863597695b345ae2e4655f5773731f1ff59d

- Kubeadm token is ran in the kubernetes slaves joining them to the cluster
- Kubernetes cluster is successfully configured

```

ubuntu@ip-172-31-20-131:~$ kubectl get nodes
NAME                 STATUS    ROLES    AGE   VERSION
ip-172-31-19-234     Ready    <none>   3m59s v1.28.2
ip-172-31-20-131     Ready    control-plane   26m   v1.28.2
ip-172-31-20-162     Ready    <none>   3m56s v1.28.2
ubuntu@ip-172-31-20-131:~$

```

i-07edd73dde8245ebf (worker3)

PublicIPs: 13.58.198.22 PrivateIPs: 172.31.20.131

## Github configuration

- Then navigated to the git repo provided and forked to my github account

The screenshot shows the GitHub repository page for 'Capstone-Project-II' by Venkat-Narayan-07. The repository is a fork of 'hshar/website'. The 'master' branch is selected, showing 1 branch and 0 tags. The file list includes 'images' (final, 6 years ago) and 'index.html' (modified, 6 years ago). The right sidebar shows the repository's activity, including 0 stars, 0 watching, and 2.2k forks. The 'About' section is empty.

- Created a new branch named develop

The screenshot shows the 'Branches' page for the 'Capstone-Project-II' repository. The 'Overview' tab is selected. The 'Default' section shows the 'master' branch as the default. The 'Your branches' section shows the 'develop' branch, which is now active. The 'Active branches' section is empty.

- Then created a new file named dockerfile to copy the entire repo and create a custom image out of it

The screenshot shows the GitHub code editor for the 'Dockerfile' in the 'Capstone-Project-II' repository. The file is being edited in the 'master' branch. The code content is as follows:

```
1 FROM ubuntu
2 RUN apt update
3 RUN apt-get install apache2 -y
4 ENTRYPOINT apache2ctl -D FOREGROUND
5 ADD . /var/www/html
```

- Similarly created a new file named hshar-deployment.yml to deploy hshar webpage from the image created by dockerfile with 3 replicas and create and attach nodeport service on port 30008

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: hshar-deployment
5    labels:
6      web: hshar
7  spec:
8    replicas: 3
9    selector:
10     matchLabels:
11       web: hshar
12   template:
13     metadata:
14       labels:
15         web: hshar
16     spec:
17       containers:
18         - name: hshar
19           image: hshar
20           ports:
21             - containerPort: 80
22   ---
23   apiVersion: v1
24   kind: Service
25   metadata:
26     name: hshar-service
27   spec:
28     type: NodePort
29     selector:
30       web: hshar
31     ports:
32       - port: 80
33         nodePort: 30008

```

- With that our git repo is ready

**Capstone-Project-II** Public

Pin
 Watch 0
 Fork 2.2k
 Star 0

forked from [hshar/website](#)

master
 2 Branches
 0 Tags
 
 Add file
 Code

This branch is 2 commits ahead of [hshar/website:master](#).
 Contribute
 Sync fork

File	Commit Message	Time Ago	Commits
images	final	6 years ago	
Dockerfile	Create Dockerfile	7 minutes ago	
hshar-deployment.yml	Create hshar-deployment.yml	3 minutes ago	
index.html	modified	6 years ago	

**About**

No description, website, or topics provided.

Activity

0 stars

0 watching

2.2k forks

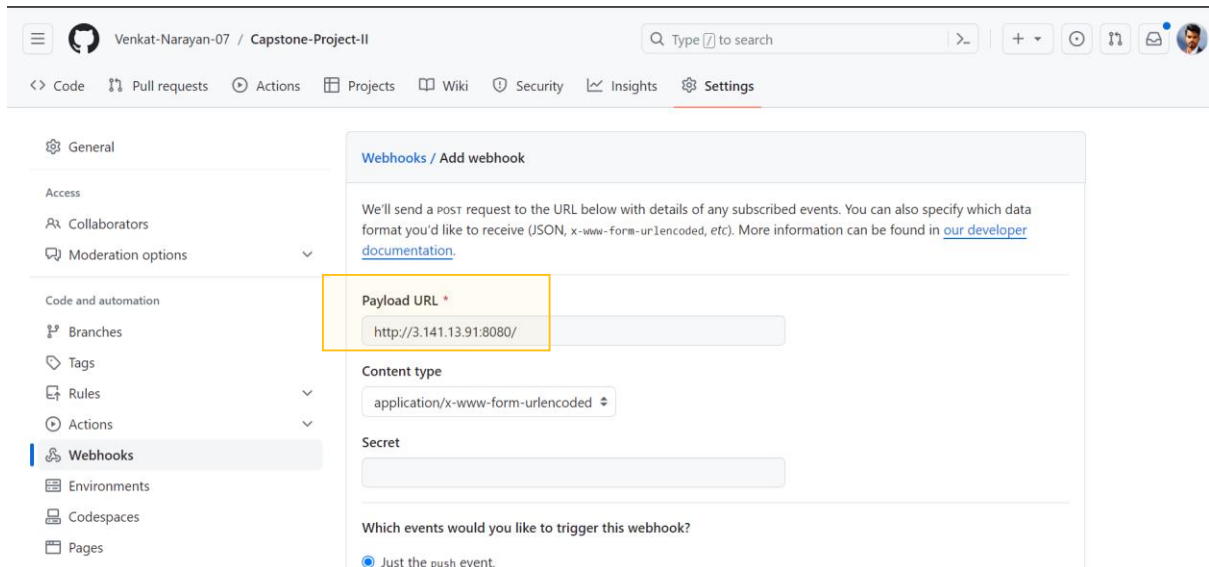
**Releases**

No releases published

[Create a new release](#)

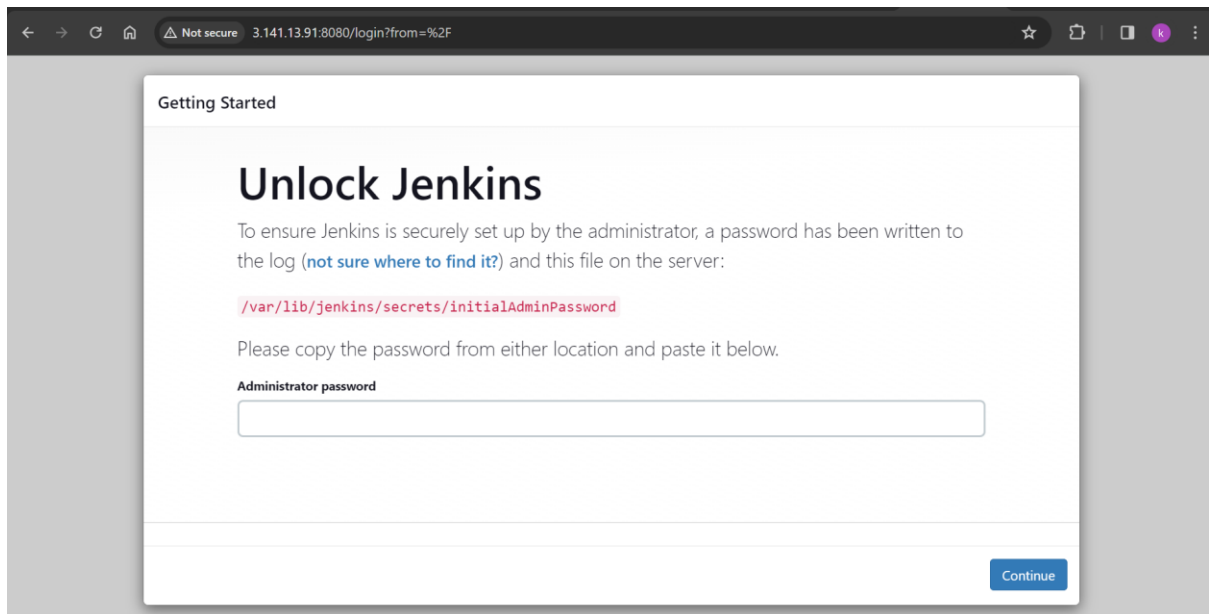
**Packages**

- Finally in the repo settings , a webhook is created to get triggered whenever there is a push to the repo



### Jenkins configuration

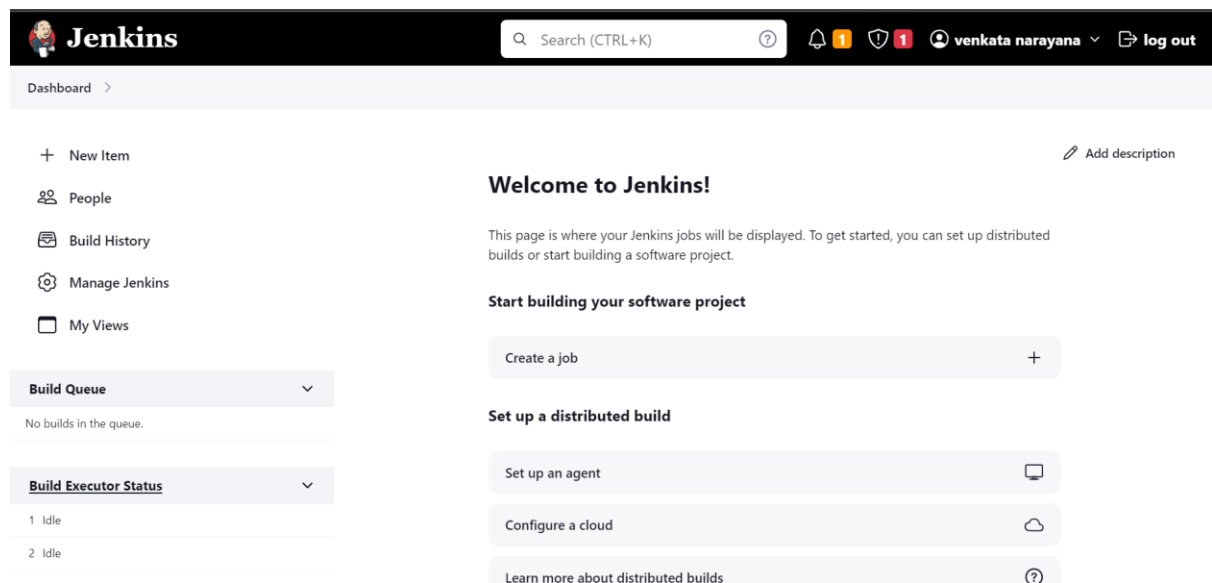
- Afterthat navigated to the jenkins dashboard and signed in



```
ubuntu@worker1:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
1104584b404c4f5998b0fa1460455790
ubuntu@worker1:~$
```

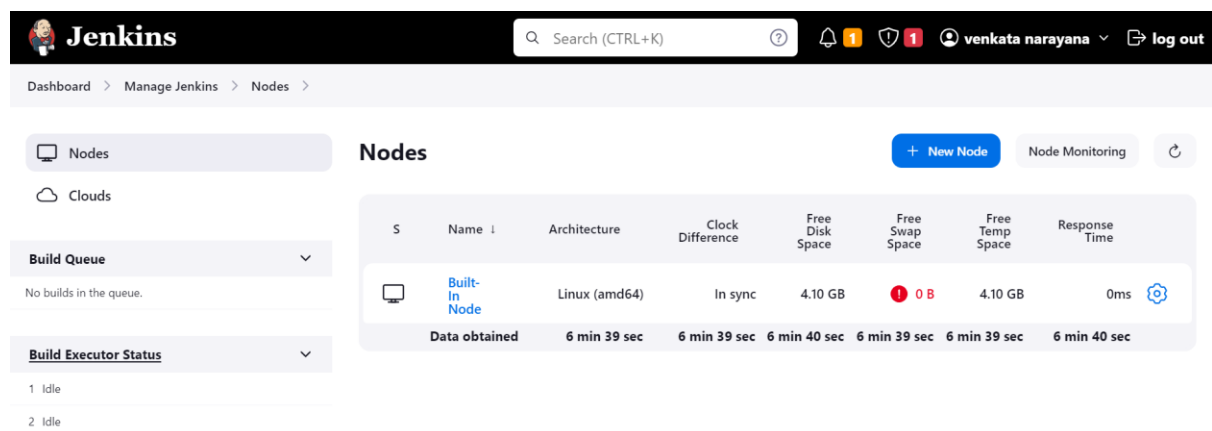
i-00b8547948a9cfa59 (Worker1)

PublicIPs: 3.141.13.91 PrivateIPs: 172.31.26.221




The screenshot shows the Jenkins dashboard. The top navigation bar includes the Jenkins logo, a search bar, and user information for 'venkata narayana'. The left sidebar contains links for 'New Item', 'People', 'Build History', 'Manage Jenkins', and 'My Views'. The main content area displays a 'Welcome to Jenkins!' message and options to 'Start building your software project' (Create a job) and 'Set up a distributed build' (Set up an agent, Configure a cloud, Learn more about distributed builds). On the left, there are sections for 'Build Queue' (No builds in the queue) and 'Build Executor Status' (2 Idle executors).

- In the jenkins dashboard navigated to manage jenkins and nodes and added a new node



The screenshot shows the 'Nodes' page in the Jenkins dashboard. The top navigation bar is the same as the previous screenshot. The left sidebar shows 'Nodes' and 'Clouds' in the main menu, and 'Build Queue' and 'Build Executor Status' in the sub-menu. The main content area is titled 'Nodes' and features a '+ New Node' button and a 'Node Monitoring' link. Below this is a table listing the nodes:

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	 Built-In Node	Linux (amd64)	In sync	4.10 GB	1.0 B	4.10 GB	0ms
Data obtained				6 min 39 sec	6 min 39 sec	6 min 39 sec	6 min 40 sec

- Named the node as kubernetes master and provided private IP of the worker3 server

Dashboard > Manage Jenkins > Nodes > New node

## New node

Node name

kubernetes-master

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



Name ?

kubernetes-master

Description ?

Plain text [Preview](#)

Number of executors ?

1

Remote root directory ?

/home/ubuntu/jenkins

Labels ?

Usage ?

Use this node as much as possible



Launch method ?

Launch agents via SSH



Host ?

172.31.20.131

Credentials ?

ubuntu



+ Add

Host Key Verification Strategy ?

Non verifying Verification Strategy



Advanced



Availability ?

Keep this agent online as much as possible



#### Node Properties



Disable deferred wipeout on this node ?

Save

- Worker3 is successfully added as a Jenkins node

Dashboard > Manage Jenkins > Nodes >

Nodes

Clouds

Build Queue

No builds in the queue.

Build Executor Status

Built-In Node

1 Idle

2 Idle

kubernetes-master

1 Idle

### Nodes

+ New Node Node Monitoring ↻

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	3.88 GB	1 0 B	3.88 GB	0ms
	kubernetes-master	Linux (amd64)	In sync	3.16 GB	1 0 B	3.16 GB	34ms
	Data obtained	0.27 sec	0.26 sec	0.23 sec	0.2 sec	0.23 sec	0.24 sec

- Then created a job1

Dashboard > All >

Enter an item name

job1

» Required field

**Freestyle project**

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

**Pipeline**

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

**Multi-configuration project**

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

**Folder**

A container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a concrete namespace, so you can have multiple things of the same name as long as they are in different folders.

OK

- Job1 is configured to the worker3 and git repo and master branch is specified in source code management.
- Enabled webhook to trigger build whenever there is a push to master branch
- And configured the job to run shell scripts post building to create a docker image out of repo and deploy the hshar-deployment.yml to kubernetes cluster

## Configure

General

Source Code Management

Build Triggers

Build Environment

Build Steps

Post-build Actions

## General

Enabled 

## Description

Dockerfile should be built every time if there is a push to GitHub.  
Create a custom Docker image using a Dockerfile.  
As per the requirement in the production server, need to use the  
Kubernetes cluster and the containerized code from Docker Hub should be  
deployed with 2 replicas. Create a NodePort service and configure the  
same for port 30008

Plain text [Preview](#)☐ Discard old builds ?☐ GitHub project☐ This project is parameterized ?☐ Throttle builds ?☐ Execute concurrent builds if necessary ?☒ Restrict where this project can be run ?

Label Expression ?

kubernetes-master

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

Advanced ▾

## Source Code Management

☐ None☒ Git ?

## Configure

General

Source Code Management

Build Triggers

Build Environment

Build Steps

Post-build Actions

## Repositories ?

Repository URL ?

https://github.com/Venkat-Narayan-07/Capstone-Project-II.git

Credentials ?

ubuntu

+ Add ▾

Advanced ▾

Add Repository

## Configure

General

Source Code Management

Build Triggers

Build Environment

Build Steps

Post-build Actions

## Branches to build ?

Branch Specifier (blank for 'any') ?

\*/master

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add ▾

### Configure

- General
- Source Code Management
- Build Triggers**
- Build Environment
- Build Steps
- Post-build Actions

#### Build Triggers

- ☐ Trigger builds remotely (e.g., from scripts) ?
- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ **GitHub hook trigger for GITScm polling ?**
- ☐ Poll SCM ?

#### Build Environment

- ☐ Delete workspace before build starts
- ☐ Use secret text(s) or file(s) ?
- ☐ Add timestamps to the Console Output
- ☐ Inspect build log for published build scans
- ☐ Terminate a build if it's stuck

### Configure

- General
- Source Code Management
- Build Triggers
- Build Environment
- Build Steps**
- Post-build Actions

#### Build Steps

##### Execute shell ?

Command

See [the list of available environment variables](#)

```
sudo docker build /home/ubuntu/jenkins/workspace/job1 -t hshar  
kubectl apply -f hshar-deployment.yml
```

Advanced ▾

Add build step ▾

**Save** Apply

- Saved the job1 to the jenkins dashboard
- Then manually ran builtnow to test the job.

Dashboard > job1 >

- Status**
- Changes
- Workspace
- Build Now
- Configure
- Delete Project
- GitHub Hook Log
- Rename

**Build History** trend ▾

## job1

Dockerfile should be built every time if there is a push to GitHub. Create a custom Docker image using a Dockerfile. As per the requirement in the production server, need to use the Kubernetes cluster and the containerized code from Docker Hub should be deployed with 2 replicas. Create a NodePort service and configure the same for port 30008

Edit description

Disable Project

### Permalinks

- Job1 is successfully completed build operation

Dashboard > job1 > #4

Status

Changes

Console Output

Edit Build Information

Delete build '#4'

Git Build Data

Previous Build

✓ #4 (Jan 7, 2024, 9:55:58 AM)

Add description

Started 36 min ago

Took 1.5 sec on kubernetes-master

No changes.

Started by user [venkata narayana](#)

git

Revision: 02b74cfc3914ddd2488d0fde495d8e14dce1151b  
Repository: <https://github.com/Venkat-Narayan-07/Capstone-Project-II.git>  
• refs/remotes/origin/master

Dashboard > job1 > #4 > Console Output

Status

Changes

Console Output

View as plain text

Edit Build Information

Delete build '#4'

Git Build Data

Previous Build

✓ Console Output

```

Started by user venkata narayana
Running as SYSTEM
Building remotely on kubernetes-master in workspace /home/ubuntu/jenkins/workspace/job1
The recommended git tool is: NONE
using credential ffc2bc49-5a47-4856-82ca-4de20fcb7ba2
> 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/Venkat-Narayan-07/Capstone-Project-II.git # timeout=10
Fetching upstream changes from https://github.com/Venkat-Narayan-07/Capstone-Project-II.git
> git --version # timeout=10
> git --version # 'git version 2.34.1'
using GIT_SSH to set credentials
Verifying host key using known hosts file
You're using 'Known hosts file' strategy to verify ssh host keys, but your known_hosts file does not exist, please
go to 'Manage Jenkins' -> 'Security' -> 'Git Host Key Verification Configuration' and configure host key
verification.
> git fetch --tags --force --progress -- https://github.com/Venkat-Narayan-07/Capstone-Project-II.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
#4 DONE 0.0s

#5 [internal] load build context
#5 transferring context: 4.29kB done
#5 DONE 0.0s

#6 [2/4] RUN apt update
#6 CACHED

#7 [3/4] RUN apt-get install apache2 -y
#7 CACHED

#8 [4/4] ADD . /var/www/html
#8 CACHED

#9 exporting to image
#9 exporting layers done
#9 writing image sha256:d90f760ee1557aeab8179ca9b44139008ef98f79b2daddfb1f8805bb6ea43f1e done
#9 naming to docker.io/library/hshar done
#9 DONE 0.0s

+ kubectl apply -f hshar-deployment.yml
deployment.apps/hshar-deployment created
service/hshar-service created
Finished: SUCCESS

```

- Whenever there is a push made to the master branch , jenkins job gets triggered and automatically containerize the code and deploy replicas of the webpage using docker and kubernetes respectively