

## Capstone Project 2

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: Java, Docker, Kubernetes**

**Worker3: Docker, Kubernetes**

**Worker4: Docker, Kubernetes**

## 1. Create an Instance Manually on AWS

The screenshot shows two sequential steps of the AWS Launch an instance wizard.

**Step 1: Summary**

Number of instances: 1

Software Image (AMI): Canonical, Ubuntu, 24.04, amd64

Virtual server type (instance type): t2.micro

Storage (volumes): 1 volume(s) - 8 GiB

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free

**Step 2: Instance type**

Instance type: t2.micro

Family: t2

1 vCPU

1 GiB Memory

Current generation: true

Free tier eligible

On-Demand Windows base pricing: 0.0162 USD per Hour

On-Demand Ubuntu Pro base pricing: 0.0134 USD per Hour

On-Demand SUSE base pricing: 0.0116 USD per Hour

On-Demand Linux base pricing: 0.0116 USD per Hour

All generations

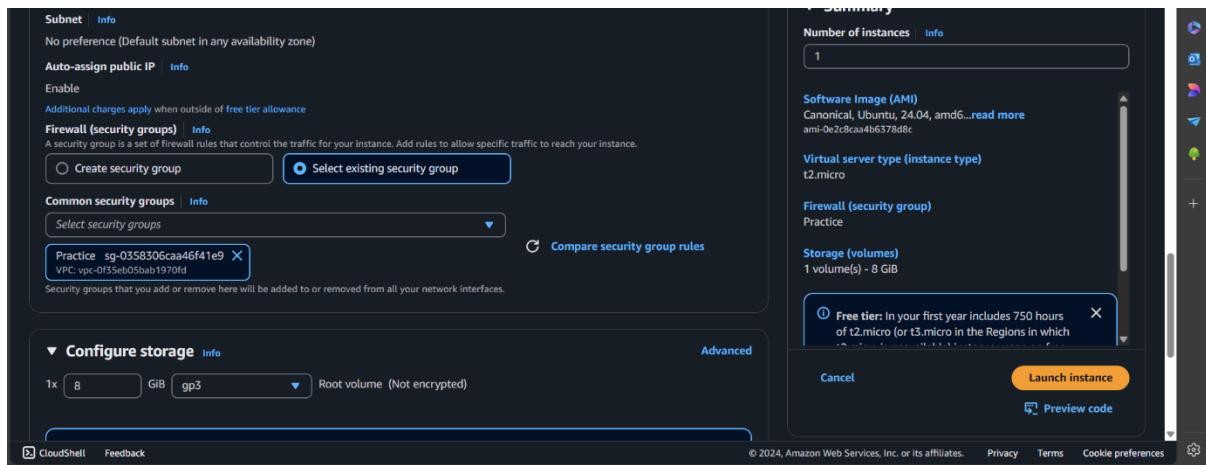
Compare instance types

**Key pair (login)**

Key pair name - required: Demo-1

Create new key pair

**Network settings**



## 2. Install Terraform on Instance [Machine-1 (Main)]

```
ubuntu@Machine-1:~$ sudo nano install.sh
```

#Enter this Command

```
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg
```

```
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list
```

```
sudo apt update && sudo apt install terraform
```

```
terraform -version
```

### Installing the CLI

```
ubuntu@Machine-1:~$ bash install.sh
```

```
ubuntu@Machine-1:~$ sudo apt install unzip -y
```

```

ubuntu@Machine-1:~$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
ubuntu@Machine-1:~$ unzip awscliv2.zip
ubuntu@Machine-1:~$ sudo ./aws/install
ubuntu@Machine-1:~$ /usr/local/bin/aws --version
ubuntu@Machine-1:~$ aws configure

```

Enter Access Key and secrete key

### 3. Run “Terraform Script” to Create Other Three Instances

```

ubuntu@Machine-1:~$ vi main.tf
provider "aws" {
    region = "us-east-1"
}

resource "aws_instance" "instance-1" {
    ami           = "ami-005fc0f236362e99f"
    key_name      = "Demo-1"
    instance_type = "t2.medium"
    tags = {
        Name = "Machine-2"
    }
}

resource "aws_instance" "instance-2" {
    ami           = "ami-005fc0f236362e99f"
    key_name      = "Demo-1"
    instance_type = "t2.medium"
    tags = {
        Name = "Machine-3"
    }
}

resource "aws_instance" "instance-3" {
    ami           = "ami-005fc0f236362e99f"
    key_name      = "Demo-1"
    instance_type = "t2.medium"
    tags = {
        Name = "Machine-4"
    }
}

"main.tf" 30L, 576B

```

The screenshot shows the AWS CloudWatch Instances page. The left sidebar includes links for Dashboard, EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images (AMIs, AMI Catalog), and Elastic Block Store (Volumes, Snapshots, Lifecycle Manager). The main content area displays a table of instances:

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
Machine-3	i-02d6a74a742a96868	Running	t2.medium	Initializing	<a href="#">View alarms +</a>	us-east-1d	
Machine-2	i-0a948e18169269ce0	Running	t2.medium	Initializing	<a href="#">View alarms +</a>	us-east-1d	
Machine-4	i-09f9854f4cc22e4a9	Running	t2.medium	Initializing	<a href="#">View alarms +</a>	us-east-1d	
Machine-1	i-0a014fd8ed676cd9	Running	t2.micro	2/2 checks passed	<a href="#">View alarms +</a>	us-east-1d	

Below the table, the details for instance i-0a948e18169269ce0 (Machine-2) are shown. The Public IPv4 address is 3.226.247.20, and the Private IPv4 address is 172.31.11.3. The instance state is Running.

## Commands

terraform init

terraform plan

terraform apply

## 4. Install “Ansible” on Machine 1 (main)

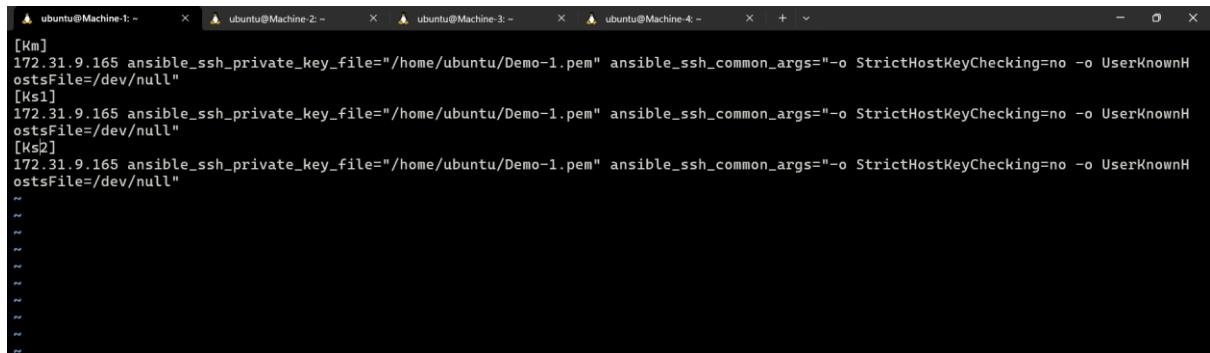
```
ubuntu@Machine-1:~$ sudo nano ansible.sh
```

```
GNU nano 7.2
sudo apt-add-repository ppa:ansible/ansible
sudo apt update
sudo apt install ansible -y
```

```
ubuntu@Machine-1:~$ bash ansible.sh |
```

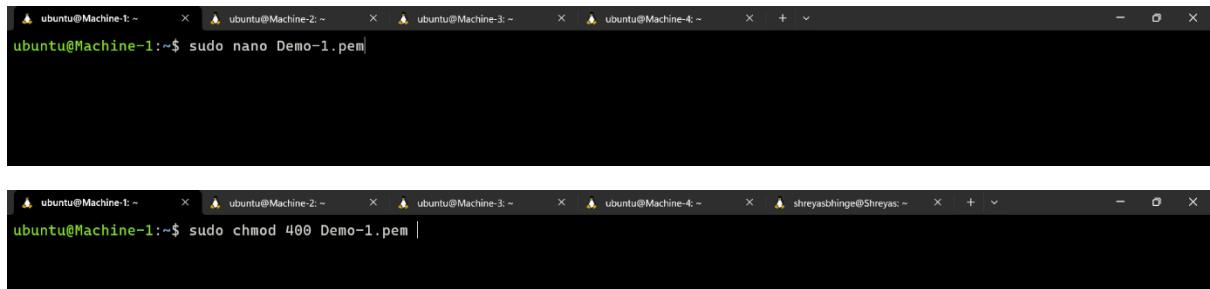
Paste the slaves & master Public IP Addresses here:

```
ubuntu@Machine-1:~$ vi inventory|
```



```
[Km]
172.31.9.165 ansible_ssh_private_key_file="/home/ubuntu/Demo-1.pem" ansible_ssh_common_args="-o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null"
[Ks1]
172.31.9.165 ansible_ssh_private_key_file="/home/ubuntu/Demo-1.pem" ansible_ssh_common_args="-o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null"
[Ks2]
172.31.9.165 ansible_ssh_private_key_file="/home/ubuntu/Demo-1.pem" ansible_ssh_common_args="-o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null"
~
~
~
~
~
~
```

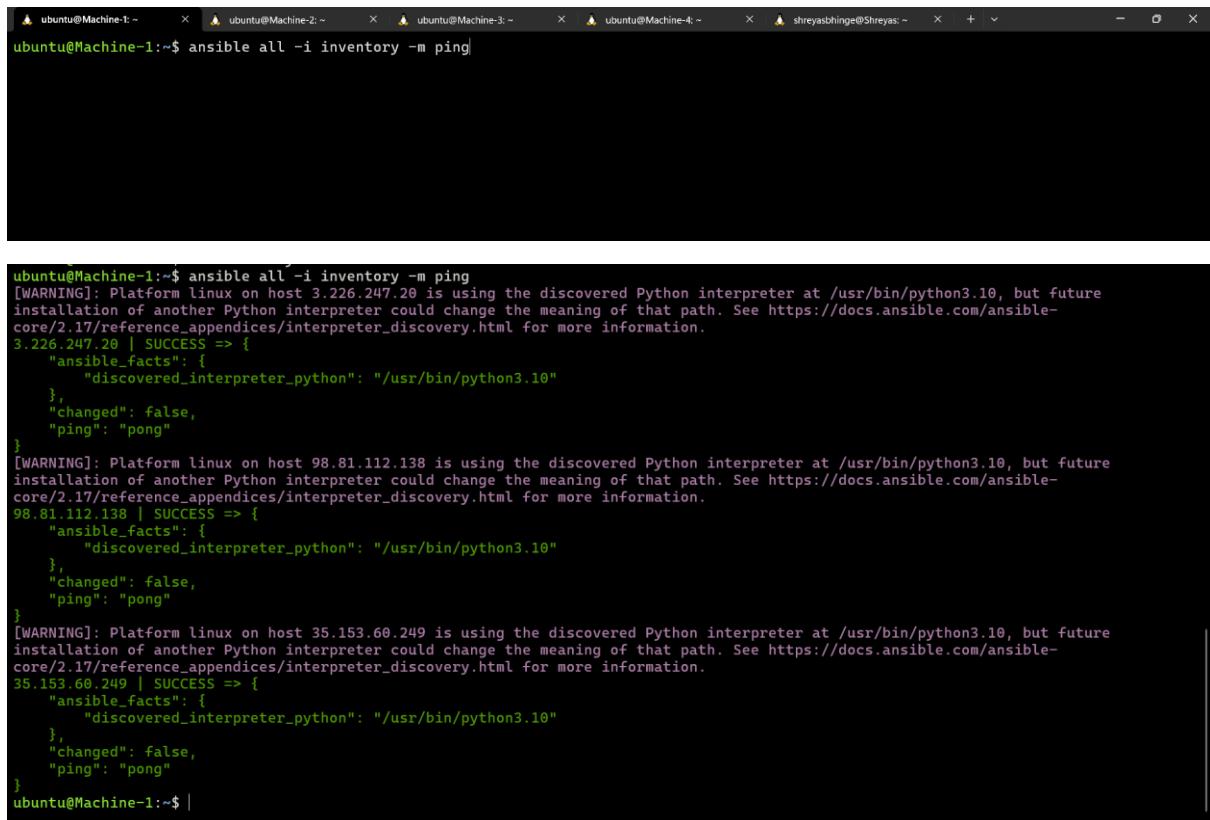
Add private key to home directory



```
ubuntu@Machine-1:~$ sudo nano Demo-1.pem
```

```
ubuntu@Machine-1:~$ sudo chmod 400 Demo-1.pem |
```

Now, we will ping all machines using the below-given command:



```
ubuntu@Machine-1:~$ ansible all -i inventory -m ping
```

```
[WARNING]: Platform linux on host 3.226.247.20 is using the discovered Python interpreter at /usr/bin/python3.10, but future installation of another Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter_discovery.html for more information.
3.226.247.20 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.10"
    },
    "changed": false,
    "ping": "pong"
}
[WARNING]: Platform linux on host 98.81.112.138 is using the discovered Python interpreter at /usr/bin/python3.10, but future installation of another Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter_discovery.html for more information.
98.81.112.138 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.10"
    },
    "changed": false,
    "ping": "pong"
}
[WARNING]: Platform linux on host 35.153.60.249 is using the discovered Python interpreter at /usr/bin/python3.10, but future installation of another Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter_discovery.html for more information.
35.153.60.249 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.10"
    },
    "changed": false,
    "ping": "pong"
}
```

Create the Playbooks to Run these Scripts to Install the Much Needed Tools

## Create Three Scripts for Installing Required Tools on Machines

## Worker1: Jenkins, Java

## Worker2: Java, Docker, Kubernetes

## Worker3: Docker, Kubernetes

Worker4: Docker, Kubernetes

```
ubuntu@Machine-1:~$ sudo nano script1.sh
```

```
GNU nano 7.2                                         script1.sh *
```

```
sudo apt update
sudo apt install openjdk-17-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 fontconfig openjdk-17-jre -y
sudo apt-get install jenkins -y
```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo  
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-A Set Mark  
M-6 Copy

```
ubuntu@Machine-1:~$ sudo nano script2.sh
ubuntu@Machine-1:~$ vi playbook.yaml
ubuntu@Machine-1:~$ sudo nano script3.sh
```

```
GNU nano 7.2                                         script2.sh *
```

```
sudo apt update -y
sudo apt install openjdk-17-jdk -y
sudo apt-get update -y
sudo apt install apt-transport-https curl -y

## Install containerd
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/docker.list &> /dev/null
sudo apt-get update -y
sudo apt-get install containerd.io -y

## Create containerd configuration
sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml
sudo sed -i -e 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml
sudo systemctl restart containerd

## Install Kubernetes
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list &> /dev/null
sudo apt-get update -y
sudo apt-get install -y kubelet kubeadm kubectl
```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo  
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-A Set Mark  
M-6 Copy

```
ubuntu@Machine-1:~$ sudo nano script2.sh
ubuntu@Machine-1:~$ vi playbook.yaml
ubuntu@Machine-1:~$ sudo nano script3.sh
```

```

GNU nano 7.2
script3.sh *

sudo apt update -y
sudo apt install apt-transport-https curl -y

## Install containerd
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/docker.list >> /dev/null
sudo apt-get update -y
sudo apt-get install containerd.io -y

## Create containerd configuration
sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml
sudo sed -i -e 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml
sudo systemctl restart containerd

## Install Kubernetes
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list >> /dev/null
sudo apt-get update -y
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
sudo systemctl enable --now kubelet

```

File menu: G Help, A Write Out, W Where Is, R Cut, T Execute, C Location, U Undo, M-A Set Mark  
Edit menu: X Exit, R Read File, \ Replace, U Paste, J Justify, Y Go To Line, E Redo, M-6 Copy

## Change permissions of all scripts

```

ubuntu@Machine-1:~$ sudo chmod +r /home/ubuntu/script1.sh
ubuntu@Machine-1:~$ sudo chown ubuntu:ubuntu /home/ubuntu/script1.sh
ubuntu@Machine-1:~$ 

ubuntu@Machine-1:~$ sudo chmod +r /home/ubuntu/script2.sh
ubuntu@Machine-1:~$ sudo chown ubuntu:ubuntu /home/ubuntu/script2.sh
ubuntu@Machine-1:~$ 

ubuntu@Machine-1:~$ sudo chmod +r /home/ubuntu/script3.sh
ubuntu@Machine-1:~$ sudo chown ubuntu:ubuntu /home/ubuntu/script3.sh
ubuntu@Machine-1:~$ 

```

## Check the syntax using the below-given command:

```

ubuntu@Machine-1:~$ ansible-playbook -i inventory playbook.yaml --syntax-check
playbook: playbook.yaml
ubuntu@Machine-1:~$ 

```

## Now, we will run the “play.yaml” using the below-given command

```

ubuntu@Machine-1:~$ ansible-playbook -i inventory playbook.yaml |

```

```

PLAY [execute the script on Km] ****
TASK [Gathering Facts] ****
[WARNING]: Platform linux on host 3.226.247.20 is using the discovered Python interpreter at /usr/bin/python3.10, but future
installation of another Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-
core/2.17/reference_appendices/interpreter_discovery.html for more information.
ok: [3.226.247.20]

TASK [installing java k8s docker] ****
changed: [3.226.247.20]

PLAY [execute the script on Ks1 Ks2] ****
TASK [Gathering Facts] ****
[WARNING]: Platform linux on host 98.81.112.138 is using the discovered Python interpreter at /usr/bin/python3.10, but future
installation of another Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-
core/2.17/reference_appendices/interpreter_discovery.html for more information.
ok: [98.81.112.138]

[WARNING]: Platform linux on host 35.153.60.249 is using the discovered Python interpreter at /usr/bin/python3.10, but future
installation of another Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-
core/2.17/reference_appendices/interpreter_discovery.html for more information.
ok: [35.153.60.249]

TASK [installing k8s and docker] ****
changed: [35.153.60.249]
changed: [98.81.112.138]

PLAY RECAP ****
3.226.247.20 : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
35.153.60.249 : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
98.81.112.138 : ok=2    changed=1    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@Machine-1:~$

```

## Configure Kubernetes on Master (Machine-2)

```

ubuntu@Machine-2:~$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16

```

Paste the below-given commands in the “Master” node:

```

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

```

```

[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to get nodes
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate cr
eentials
[bootstrap-token] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

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 172.31.11.3:6443 --token hw9v5p.6jarlosxgl5da7m \
    --discovery-token-ca-cert-hash sha256:28cc7a884f0a46438fb55b2a7e1cfb662dc0331420f5e5deedbccef11177086a
ubuntu@Machine-2:~$ mkdir -p $HOME/.kube
ubuntu@Machine-2:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
ubuntu@Machine-2:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
ubuntu@Machine-2:~|

```

## In K8s Slaves (Machine-3 and Machine-4}

### Machine-3

```
ubuntu@Machine-1:~ x ubuntu@Machine-2:~ x ubuntu@Machine-3:~ x ubuntu@Machine-4:~ x shreyashbingle@Shreyas:~ x + - o x
ubuntu@Machine-3:~$ sudo kubeadm reset pre-flight checks
```

```
ubuntu@Machine-1:~ x ubuntu@Machine-2:~ x root@Machine-3:/home/uba x ubuntu@Machine-4:~ x + - o x
ubuntu@Machine-3:~$ sudo kubeadm reset pre-flight checks
W1225 06:39:27.608054    3690 preflight.go:56] [reset] WARNING: Changes made to this host by 'kubeadm init' or 'kubeadm join' will be reverted.
[reset] Are you sure you want to proceed? [y/N]: y
[preflight] Running pre-flight checks
W1225 06:39:31.346529    3690 removeetcdmember.go:106] [reset] No kubeadm config, using etcd pod spec to get data directory
[reset] Deleted contents of the etcd data directory: /var/lib/etcd
[reset] Stopping the kubelet service
[reset] Unmounting mounted directories in "/var/lib/kubelet"
[reset] Deleting contents of directories: [/etc/kubernetes/manifests /var/lib/kubelet /etc/kubernetes/pki]
[reset] Deleting files: [/etc/kubernetes/admin.conf /etc/kubernetes/super-admin.conf /etc/kubernetes/kubelet.conf /etc/kubernetes/bootstrap-kubelet.conf /etc/kubernetes/controller-manager.conf /etc/kubernetes/scheduler.conf]

The reset process does not clean CNI configuration. To do so, you must remove /etc/cni/net.d

The reset process does not reset or clean up iptables rules or IPVS tables.
If you wish to reset iptables, you must do so manually by using the "iptables" command.

If your cluster was setup to utilize IPVS, run ipvsadm --clear (or similar)
to reset your system's IPVS tables.

The reset process does not clean your kubeconfig files and you must remove them manually.
Please, check the contents of the ${HOME}/.kube/config file.
ubuntu@Machine-3:~$ sudo su
root@Machine-3:/home/ubuntu# |
```

Copy Token from Machine-2 and paste above –v=5 in machine3

```
ubuntu@Machine-3:~$ sudo su
root@Machine-3:/home/ubuntu# kubeadm join 172.31.11.3:6443 --token hw9v5p.6jarlosxxgl5da7m \
--discovery-token-ca-cert-hash sha256:28cc7a884f0a46438fb55b2a7e1cfb662dc0331420f5e5dee1dbcef11177086a --v=5|
```

**Same steps to follow with Machine-4**

**Install the “Calico Network” to run the cluster using the below-given command in Machine-2 (Km):**

**kubectl apply -f <https://raw.githubusercontent.com/flannel-io/flannel/master/Documentation/kube-flannel.yml>**

```
ubuntu@Machine-2:~$ kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml
namespace/kube-flannel created
serviceaccount/flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
```

**All the deployments have been successfully created. Now, run the below-given command to get all the nodes:**

```
ubuntu@Machine-1:~ x ubuntu@Machine-2:~ x root@Machine-3:/home/uba x root@Machine-4:/home/uba x + - o x
ubuntu@Machine-2:~$ kubectl get node
NAME      STATUS    ROLES   AGE     VERSION
machine-2  NotReady control-plane 3m44s  v1.30.8
machine-3  NotReady <none>    99s    v1.30.8
machine-4  NotReady <none>    16s    v1.30.8
ubuntu@Machine-2:~$ |
```

## (In Machine-2) i.e Master

The error indicates that containerd.io conflicts with another package (containerd) on your system, or that there are held packages causing dependency resolution issues. Here's how to resolve it step by step:

### 1. Check for Held Packages

Held packages can prevent updates or installations. Check for held packages using:

bash

Copy code

**dpkg --get-selections | grep hold**

If any packages are held, unhold them:

bash

Copy code

**sudo apt-mark unhold <package-name>**

```
ubuntu@Machine-2:~$ kubectl get node
NAME     STATUS   ROLES      AGE     VERSION
machine-2 Ready    control-plane   10m    v1.30.8
machine-3 Ready    <none>    8m28s   v1.30.8
machine-4 Ready    <none>    7m5s    v1.30.8
ubuntu@Machine-2:~$ dpkg --get-selections | grep hold
kubeadm          hold
kubectl          hold
kubelet          hold
ubuntu@Machine-2:~$ sudo apt-mark unhold kubeadm
Canceled hold on kubeadm.
ubuntu@Machine-2:~$ sudo apt-mark unhold kubectl
Canceled hold on kubectl.
ubuntu@Machine-2:~$ sudo apt-mark unhold kubelet
Canceled hold on kubelet.
ubuntu@Machine-2:~$ |
```

### 2. Remove Conflicting Packages

Since containerd.io conflicts with containerd, you may need to remove containerd:

bash

Copy code

**sudo apt-get remove --purge containerd**

```
ubuntu@Machine-1:~ -> ubuntu@Machine-2:~ -> root@Machine-3:/home/ubur <-> root@Machine-4:/home/ubur - + ->
ubuntu@Machine-2:~$ sudo apt-get remove --purge containerd
```

### 3. Clean Up and Update

Clean up and refresh the package manager

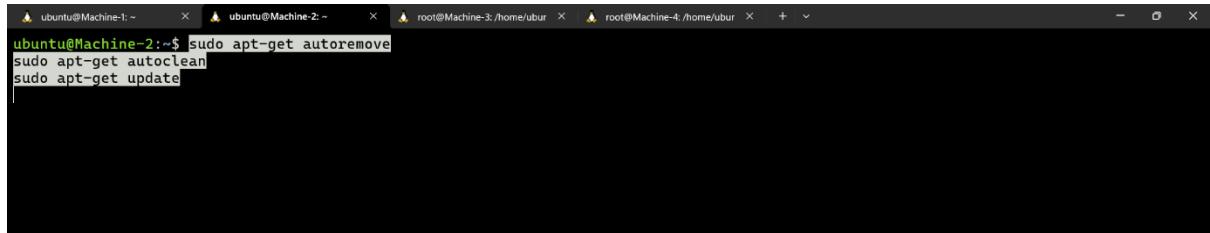
bash

Copy code

**sudo apt-get autoremove**

**sudo apt-get autoclean**

**sudo apt-get update**



```
ubuntu@Machine-2:~$ sudo apt-get autoremove
sudo apt-get autoclean
sudo apt-get update
```

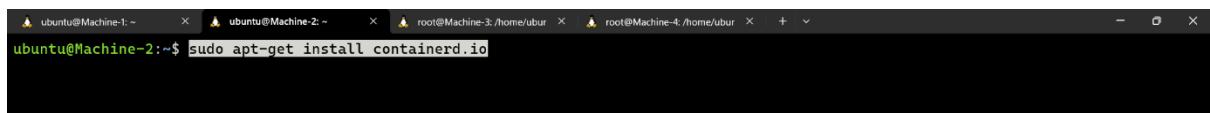
#### 4. Install containerd.io

Try installing containerd.io again:

bash

Copy code

**sudo apt-get install containerd.io**



```
ubuntu@Machine-2:~$ sudo apt-get install containerd.io
```

#### Check Docker and Containerd Versions

If you're using Docker, ensure the versions of Docker and containerd.io are compatible.

Update Docker if necessary:

bash

Copy code

**sudo apt-get install docker-ce docker-ce-cli**

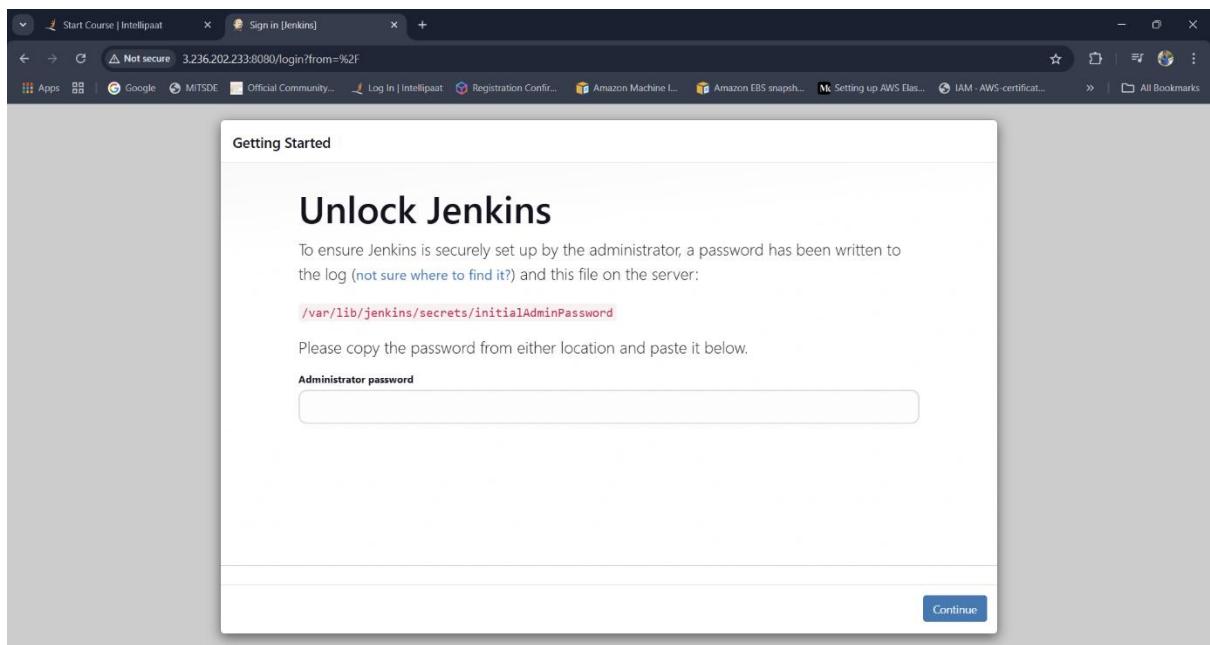
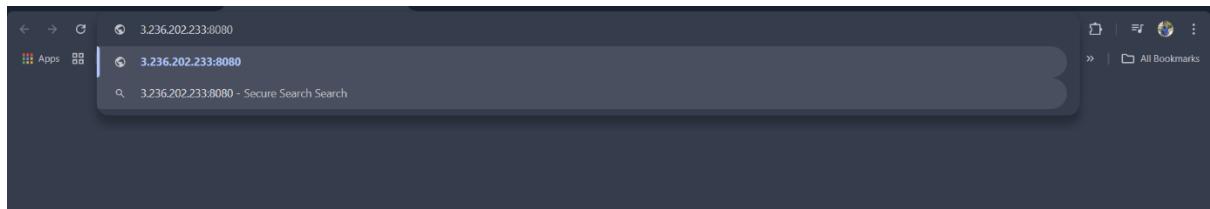


```
ubuntu@Machine-2:~$ sudo apt-get install docker-ce docker-ce-cli
```

**Copy same steps to Machine-3 and Machine-4**

**Configure Jenkins Setup Properly Here on Machine-1**

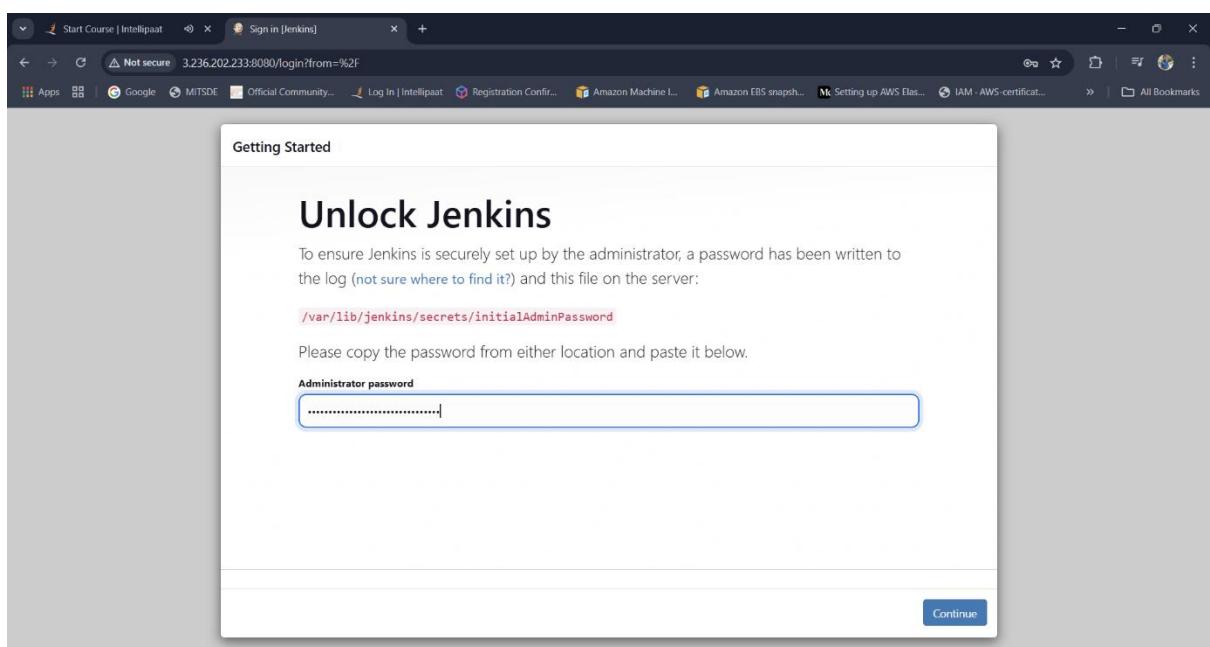
**Copy Public Ip of Machine-1**



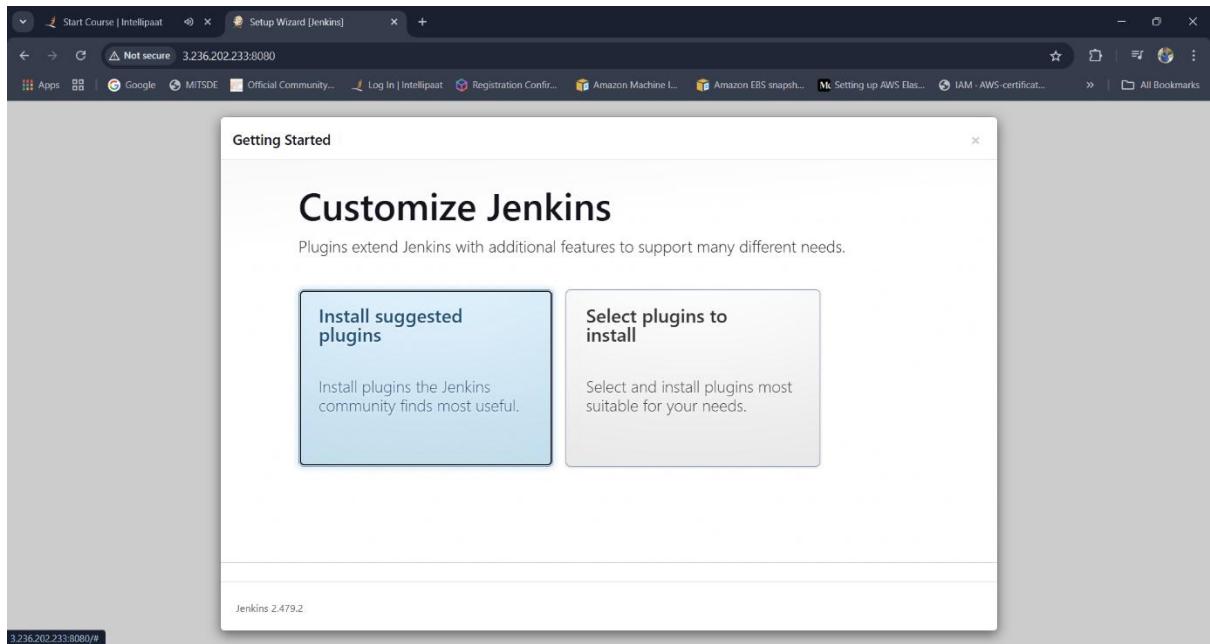
**Go to the “Machine-1 (Main)” & paste the below-given command with “sudo cat”**

```
| ~$ ubuntu@Machine-1:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

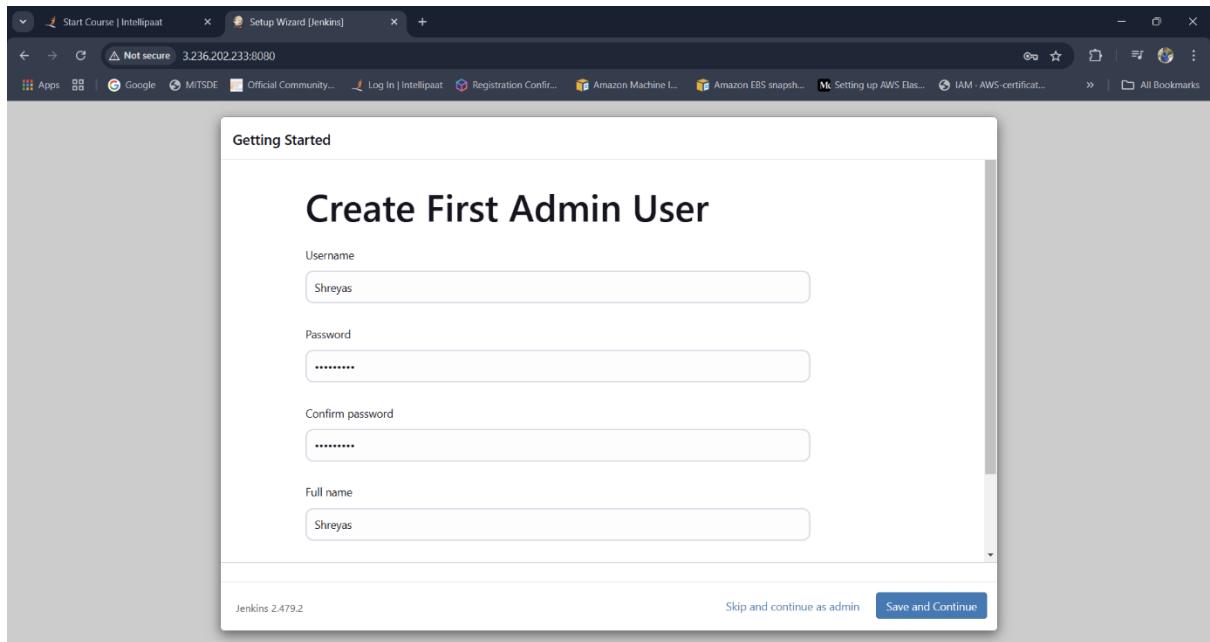
**Paste the token in the “Administrator Password” section & click on “Continue”.**



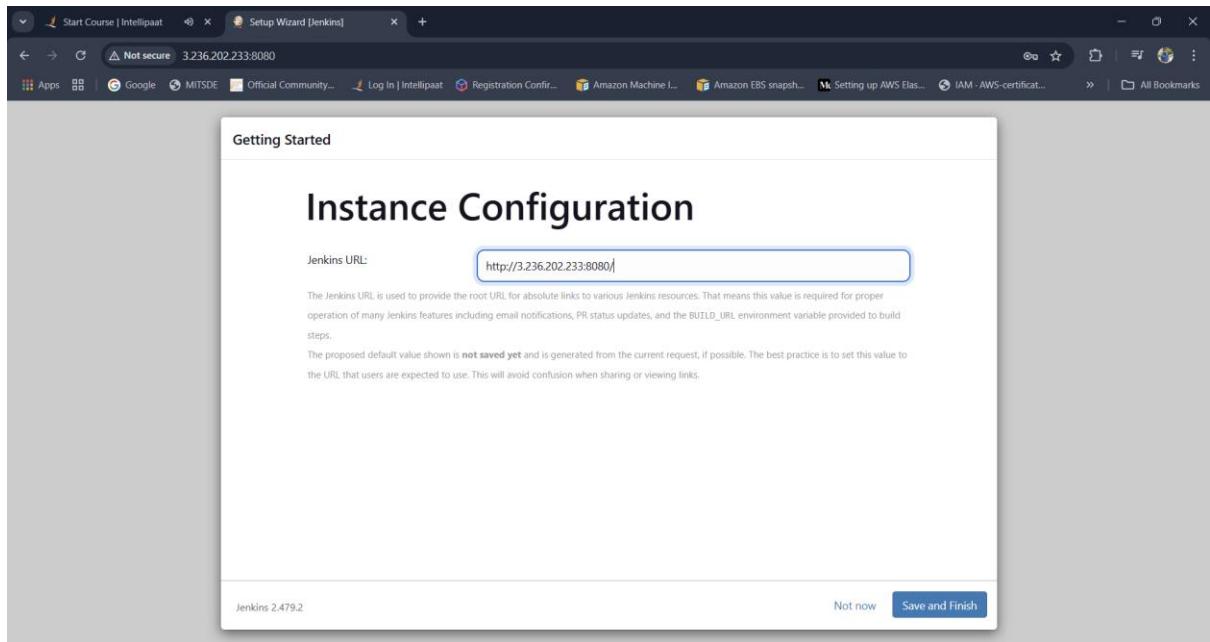
**Click on “Install Suggested Plugins”.**



**Create the user by filling the below-given details:**



**Click on “Save and Finish”**



## Add “Kubernetes Master (Machine-2)” as a Node Here

Click on “Set up an agent”.

### Add new Node

The screenshot shows the Jenkins dashboard with the "Nodes" tab selected. The main table lists one node: "Built-In Node" (Architecture: Linux (amd64), Last checked: 31 min ago). The table includes columns for Name, Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. A "New Node" button is visible at the top right of the table area.

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	2.10 GiB	0 B	2.10 GiB	0ms
	last checked	31 min	31 min	31 min	31 min	31 min	31 min

choose “Node name” as “Km” & “Type” as “Permanent Agent”.

New node

Node name

Km

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

REST API Jenkins 2.479.2

## Choose the following options here

Name ? Km

Description ?

Plain text Preview

Number of executors ? 1

Remote root directory ? /home/ubuntu/jenkins

Save

Start Course | Intellipaat Jenkins

Not secure 3.236.202.233:8080/manage/computer/createlnem

Dashboard > Manage Jenkins > Nodes >

Label ? Km

Usage ? Use this node as much as possible

Launch method ? Launch agents via SSH

Host ? 172.31.11.3

Credentials ? - none - Jenkins Credentials Provider Jenkins cannot be found

Host Key Verification Strategy ?

Save

This screenshot shows the Jenkins Node creation form. It includes fields for Label (Km), Usage (Use this node as much as possible), Launch method (Launch agents via SSH), Host (172.31.11.3), and Credentials (- none -). A Jenkins Credentials Provider button is present, but it shows an error message: 'cannot be found'. There is also a Host Key Verification Strategy section.

Start Course | Intellipaat Jenkins

Not secure 3.236.202.233:8080/manage/computer/createlnem

Dashboard > Manage Jenkins > Nodes >

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 PEM

Description

Save

This screenshot shows the Jenkins Credentials Provider configuration dialog. It displays a 'Add Credentials' form with the following details: Domain set to 'Global credentials (unrestricted)', Kind set to 'SSH Username with private key', Scope set to 'Global (Jenkins, nodes, items, all child items, etc)', and ID set to 'PEM'. There is also a Description field and a Save button at the bottom.

Jenkins Credentials Provider: Jenkins

Username: ubuntu

Treat username as secret

Private Key:

Enter directly

Key:

```
-----BEGIN RSA PRIVATE KEY-----
4am9AogALby05FtYp4c1m2dXHw41HeMho/ypaJha/tMVj92mK0-1o/8jHhNxo0J
a9MKdI6GwIXqTIncbjvbaR2svQ4vHMLLev3ebmcToKSx/zENDct5jpJQLLm
jrdkeyybEjyu/AQ7ul8GP4k+UjxAv41X6YIwD21I21LkxCJK7Q=
-----END RSA PRIVATE KEY-----
```

Passphrase:

**Save**

- none -  
ubuntu

**Choose “Host Key Verification Strategy” as “Non verifying Verification Strategy”.**

**Click on “Save”.**

Host Key Verification Strategy: Non verifying Verification Strategy

Availability: Keep this agent online as much as possible

Node Properties:

- Disable deferred wipeout on this node
- Disk Space Monitoring Thresholds
- Environment variables
- Tool Locations

**Save**

REST API Jenkins 2.479.2

**“Km” has been successfully added as a “Node”.**

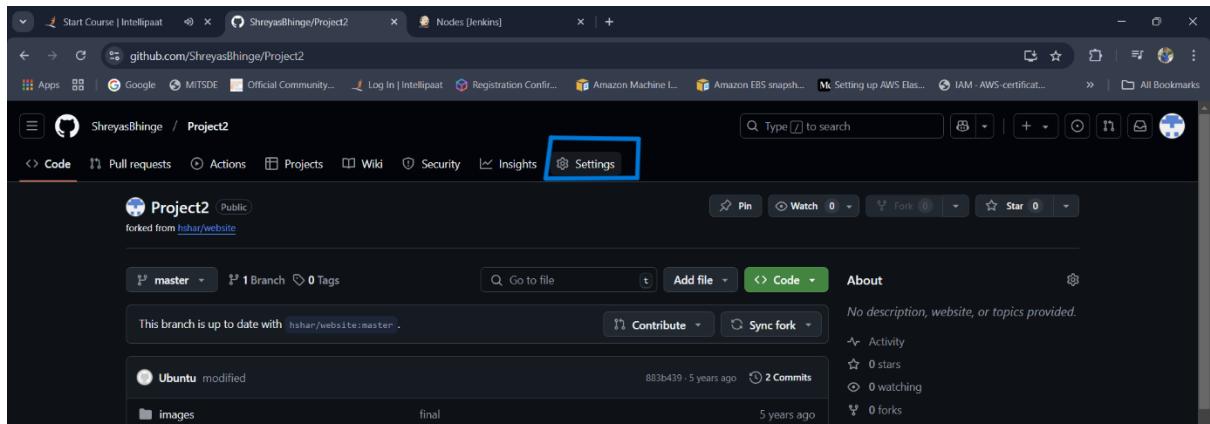
The screenshot shows the Jenkins 'Nodes' page. At the top, there's a search bar and user information. Below it, a sidebar has sections for 'Build Queue' (empty) and 'Build Executor Status' (two entries: 'Built-In Node' and 'Km'). The main area is titled 'Nodes' and contains a table with columns: S, Name, Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. It lists two nodes: 'Built-In Node' (Linux (amd64), In sync, 1.94 GiB free disk, 0 B swap, 1.94 GiB temp, 0ms response) and 'Km' (Linux (amd64), In sync, 3.29 GiB free disk, 0 B swap, 3.29 GiB temp, 27ms response). A legend at the bottom right defines icons for S, M, and L.

## Fork the Repository in the GitHub Account

The screenshot shows the GitHub repository 'hshar/website'. The repository page includes a 'Code' tab, a commit history (master branch, 1 commit, 0 tags), and sections for 'About', 'Releases', 'Packages', and 'Deployments'. On the right, there's a sidebar with repository statistics: Activity (27 stars, 4 watching, 2.9k forks), a 'Report repository' link, and a 'Fork your own copy of hshar/website' button.

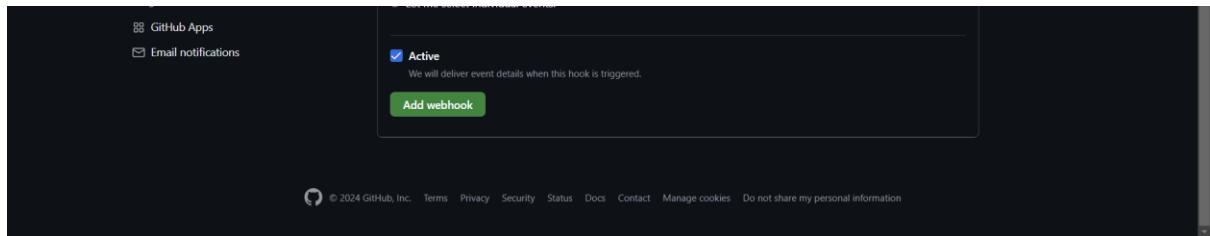
The screenshot shows the GitHub 'Create a new fork' form. It asks for the owner ('ShreyasBhinge') and repository name ('Project2'). A note says 'Project2 is available.' Below, it says 'By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.' There's a 'Description (optional)' field, a checked checkbox for 'Copy the master branch only', and a note about contributing back. At the bottom, a note says 'You are creating a fork in your personal account.' and a green 'Create fork' button.

## Create webhook



This screenshot shows the 'General' settings page for 'Project2'. The 'Webhooks' section is highlighted with a blue box. Other sections visible include General, Access, Collaborators, Moderation options, Code and automation, Branches, Tags, Rules, Actions, Environments, Codespaces, and Pages. The 'Default branch' is set to 'master'. The 'Social preview' section allows for uploading a custom image for social media sharing.

This screenshot shows the 'Add webhook' page for 'Project2'. The 'Webhooks / Add webhook' section is highlighted with a blue box. It includes fields for 'Payload URL' (set to http://3.236.202.233:8080/github-webhook/), 'Content type' (set to application/x-www-form-urlencoded), and a 'Secret' field. Under 'SSL verification', the 'Enable SSL verification' option is selected. The 'Which events would you like to trigger this webhook?' section shows 'Just the push event' selected.



## Create a Docker file in Given GitHub Repository

The image consists of three vertically stacked screenshots of a GitHub repository named 'Project2'.

- Screenshot 1:** Shows the repository overview. It has 1 branch and 0 tags. A tooltip over the 'Add file' button indicates '+ Create new file' and 'Upload files'. The repository is up-to-date with the 'master' branch of the forked repository 'hshar/website'.
- Screenshot 2:** Shows the 'Code' tab where a new Dockerfile is being created. The code is as follows:

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

- Screenshot 3:** Shows the commit dialog. The 'Commit message' field contains 'Create Dockerfile'. The 'Extended description' field is empty. The 'Commit directly to the master branch' radio button is selected. There are 'Cancel' and 'Commit changes...' buttons at the bottom.

## Create a Pipeline to Automate the Tasks

Click on “Create a job”.

Dashboard > New Item

Enter an item name  
JOB-1

Select an item type

**Freestyle project**  
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

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

**Multibranch Pipeline**  
Creates a set of Pipeline projects according to detected branches in one SCM repository.

OK

Throttle builds ?

**Build Triggers**

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

**Advanced Project Options**

Save Apply

## Add DockerHub credential to Credentials

0/1

Security

Credentials Configure credentials

Credential Providers Configure the credential providers and types

Users Create/delete/modify users that can log in to this Jenkins.

Status Information  
3.236.202.233:8080/manage/credentials

Click on global

The screenshot shows the Jenkins 'Credentials' management interface. At the top, there's a navigation bar with tabs like 'Dashboard', 'Manage Jenkins', and 'Credentials'. Below the navigation is a search bar and a user profile. The main content area is titled 'Credentials' and contains a table with columns: ID, Name, and Domain. One row is visible, showing 'ID' as 'PEM', 'Name' as 'ubuntu', and 'Domain' as '(global)'. Below the table, there's a section titled 'Stores scoped to Jenkins' with a similar table structure, showing a single entry for 'System' under 'Domains'. The bottom of the page includes a URL bar ('3.236.202.233:8080/manage/credentials/store/system/domain/\_/'), a REST API link, and a Jenkins version number ('Jenkins 2.479.2').

**Click on Add credential**

The screenshot shows the 'Global credentials (unrestricted)' page in Jenkins. The URL in the address bar is '3.236.202.233:8080/manage/credentials/store/system/domain/\_/'. The page title is 'Global credentials (unrestricted)'. It features a table with columns: ID, Name, Kind, and Description. One row is present, with 'ID' as 'PEM', 'Name' as 'ubuntu', 'Kind' as 'SSH Username with private key', and a 'Description' column containing a key icon. Below the table, there are icons for sorting by ID, Name, Kind, and Description, and a link to 'Add Credentials'.

**Enter your dockerhub user name and password**

The screenshot shows the 'Global credentials (unrestricted)' page in Jenkins. The URL in the address bar is '3.236.202.233:8080/manage/credentials/store/system/domain/\_/'. The page title is 'Global credentials (unrestricted)'. It features a table with columns: ID, Name, Kind, and Description. Two rows are present: one for 'ubuntu' (Kind: SSH Username with private key) and one for 'shreyas3103' (Kind: Username with password). Both rows have a key icon in the 'Description' column. Below the table, there are icons for sorting by ID, Name, Kind, and Description, and a link to 'Add Credentials'.

**Go to the “Pipeline” section & choose the “Hello World” script here.**

**Edit the script add you github url and docker credential**

The screenshot shows the Jenkins Pipeline configuration page for a job named 'JOB-1'. The 'Pipeline' tab is selected under 'Definition'. The pipeline script is as follows:

```
1 ~ pipeline {  
2 ~   agent none  
3 ~   environment {  
4 ~     DOCKERHUB_CREDENTIALS = credentials("84896bc2-9dbe-45c6-9cd5-74766b9ae3d2")  
5 ~   }  
6 ~   stages {  
7 ~     stage('git') {  
8 ~       agent{  
9 ~         label "K8n"  
10 ~       }  
11 ~       steps {  
12 ~         script {  
13 ~           git 'https://github.com/ShreyasBhinge/Project2.git'  
14 ~         }  
15 ~       }  
16 ~     }  
17 ~ }
```

Below the script, there is a checkbox for 'Use Groovy Sandbox' which is checked. At the bottom are 'Save' and 'Apply' buttons.

## Apply the script

The screenshot shows the Jenkins Job-1 dashboard. The 'Status' tab is selected. The pipeline is named 'Job-1'. Below the pipeline name, it says 'Job-1'. On the right, there is an 'Edit description' link. The 'Builds' section shows a single build entry from today at 07:37, indicated by a green checkmark icon. The URL for the configuration page is shown at the bottom of the dashboard.

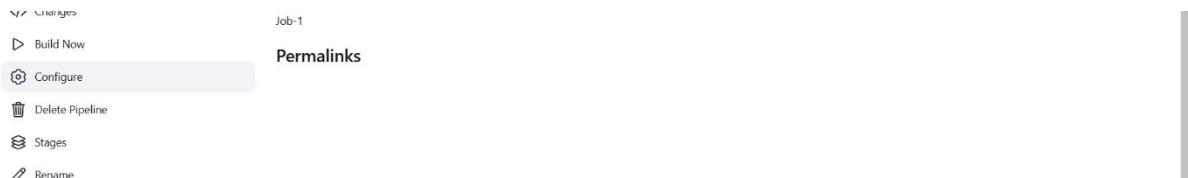
Confirm it

```

ubuntu@Machine-1:~          ubuntu@Machine-2:~/jenkin  ubuntu@Machine-3:~          ubuntu@Machine-4:~          shreyasbhinge@Shreyas:~
jenkins
ubuntu@Machine-2:~$ cd jenkins/
ubuntu@Machine-2:~/jenkins$ ls
remoting  remoting.jar  workspace
ubuntu@Machine-2:~/jenkins$ cd workspace/
ubuntu@Machine-2:~/jenkins/workspace$ ls
JOB-1
ubuntu@Machine-2:~/jenkins/workspace$ cat JOB-1/
cat: JOB-1/: Is a directory
ubuntu@Machine-2:~/jenkins/workspace$ cd JOB-1/
ubuntu@Machine-2:~/jenkins/workspace/JOB-1$ ls
Dockerfile  images  index.html
ubuntu@Machine-2:~/jenkins/workspace/JOB-1$ |

```

## Again Configure



Now, we will push the “Docker Hub Image” using the pipeline code.

```

1 pipeline {
2   agent none
3   environment {
4     DOCKERHUB_CREDENTIALS = credentials("84896bc2-9dbe-45c6-9cd5-74766b9ae3d2")
5   }
6
7   stages {
8     stage('git') {
9       agent{
10         label "Km"
11       }
12       steps {
13         script {
14           git 'https://github.com/ShreyasBhinge/Project2.git'
15         }
16       }
17     }
18   }
19   stages {
20     stage('docker') {
21       agent{
22         label "Km"
23       }
24       steps {
25         script {
26           sh 'sudo docker build . -t shreyas3103/project2'
27           sh 'sudo docker login -u ${DOCKERHUB_CREDENTIALS_USR} -p ${DOCKERHUB_CREDENTIALS_PSN}'
28           sh 'sudo docker push shreyas3103/project2'
29         }
30       }
31     }
32   }
33 }

```

## Apply it



**Now, we will create the “deployment.yaml” & “service.yaml” file to deploy the website using the “Kubernetes” tool.**

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-deployment
  template:
    metadata:
      labels:
        app: my-deployment
    spec:
      containers:
        - name: my-deployment
          image: shreyas3103/project2
      ports:
        - containerPort: 80
  
```

## Service.yaml

```

1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: nginx-service
5  spec:
6    type: NodePort
7    selector:
8      app: nginx
9    ports:
10   - protocol: TCP
11     port: 80
12     targetPort: 80
13     nodePort: 30008

```

## Add Kubernetes stage in pipeline below docker

```

stage('K8s') {
    agent {
        label 'Kubernetes-Master'
    }
    steps {
        sh 'kubectl apply -f deployment.yaml'
        sh 'kubectl apply -f service.yaml'
    }
}
}
}
}
}

```

General

Advanced Project Options

Pipeline

```

28
29
30 }
31 stage('kubernetes') {
32     agent {
33         label "K8s"
34     }
35     steps {
36         script {
37             sh 'kubectl apply -f deployment.yaml'
38             sh 'kubectl apply -f service.yaml'
39         }
40     }
41 }
42 }
43

```

Use Groovy Sandbox ?

Pipeline Syntax

Save Apply

**Now apply**

The screenshot shows a CI/CD pipeline interface with a sidebar containing options like Configure, Delete Pipeline, Stages, Rename, Pipeline Syntax, and GitHub Hook Log. Below this is a 'Builds' section with a filter and a log entry for build #7 at 08:08. The main area displays terminal logs from four machines (Machine-1 to Machine-4) and a host machine (Shreyas). The logs show commands like 'kubectl get svc' and 'sudo docker images' being run.

```

ubuntu@Machine-2:~$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
kubernetes     ClusterIP  10.96.0.1   <none>        443/TCP       91m
nginx-service  NodePort   10.96.118.251 <none>        80:30008/TCP  88s
ubuntu@Machine-2:~$ |
```

```

ubuntu@Machine-1:~$ | ubuntu@Machine-2:~$ | ubuntu@Machine-3:~$ | ubuntu@Machine-4:~$ | shreyashb hinge@Shreyas:~$ |
```

```

ubuntu@Machine-2:~$ sudo docker images
REPOSITORY          TAG      IMAGE ID      CREATED             SIZE
shreyas3103/project2  latest   2e802969de5a  About a minute ago  230MB
shreyas3103/project2  <none>   123e8225133b  7 minutes ago    230MB
shreyas3103/project2  <none>   79ce8e8f881f  10 minutes ago   230MB
shreyas3103/project2  <none>   098cb79386a5  17 minutes ago   230MB
ubuntu@Machine-2:~$ |
```

## Pipeline script

```

pipeline {
    agent none
    environment {
        DOCKERHUB_CREDENTIALS = credentials("cb26ac19-f954-40ca-a12e-d790594bcc7")
    }
    stages {
        stage('git') {
            agent {
                label "Km"
            }
            steps {
                script {

```

```
        git'https://github.com/Sameer-8080/website.git'

    }

}

}

stage('docker') {

    agent {

        label "K8-Master"

    }

    steps {

        script {

            sh 'sudo docker build shreyas3103/project2'

            sh 'sudo docker login -u ${DOCKERHUB_CREDENTIALS_USR} -p
${DOCKERHUB_CREDENTIALS_PSW}'

            sh 'sudo docker push shreyas3103/project2'

        }

    }

}

stage('kubernetes') {

    agent {

        label "K8-Master"

    }

    steps {

        script {

            sh 'kubectl delete deploy nginx-deployment'

            sh 'kubectl apply -f deployment.yaml'

            sh 'kubectl delete service my-service'

            sh 'kubectl apply -f service.yaml'

        }

    }

}

}
```

}

}