

## Experiment No. 3

**Aim:** To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud.

**Steps:**

1. Create 3 EC2 Ubuntu Instances on AWS. (Name 1 as Master, the other 2 as worker-1 and worker-2)

[EC2](#) > [Instances](#) > Launch an instance

## Launch an instance [Info](#)

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

### Name and tags [Info](#)

Name

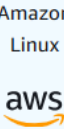
[Add additional tags](#)


### ▼ Application and OS Images (Amazon Machine Image) [Info](#)


An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below


Recents


Quick Start


Amazon Linux


macOS

Ubuntu

Windows

Red Hat

SUSE Li

[Browse more AMIs](#)  
Including AMIs from AWS, Marketplace and

### ▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

[Create new key pair](#)

▼ Network settings

Info

Edit

Network

Info

vpc-0404d393731afabc3

Subnet

Info

No preference (Default subnet in any availability zone)

Auto-assign public IP

Info

Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups)

Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group

Select existing security group

We'll create a new security group called 'launch-wizard-4' with the following rules:

☒ Allow SSH traffic from
 

Anywhere

0.0.0.0/0

Helps you connect to your instance

☒ Allow HTTPS traffic from the internet
 

To set up an endpoint, for example when creating a web server

☒ Allow HTTP traffic from the internet
 

To set up an endpoint, for example when creating a web server

⚠

Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

×

Instances (3)

Info

Last updated less than a minute ago

Refresh

Connect

Instance state ▼

Actions ▼

Launch instances ▼

Find Instance by attribute or tag (case-sensitive)

Running ▼

< 1 > ⚙

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
<input type="checkbox"/>	master	i-0a57da166d1f22307	Running	t2.micro	Initializing	View alarms	us-east-1c	ec2-54-165-196-241.co...	54.165.196.24
<input type="checkbox"/>	worker-2	i-09ef6bb0a00bfc3c	Running	t2.micro	2/2 checks passed	View alarms	us-east-1c	ec2-184-72-206-70.co...	184.72.206.70
<input type="checkbox"/>	worker-1	i-04e9c3add3858c21c	Running	t2.micro	Initializing	View alarms	us-east-1c	ec2-54-211-147-10.co...	54.211.147.10

## 2. SSH into all 3 machines

- Give permissions to the current user to the downloaded pem file using - `chmod 400 <security_filename.pem>`

```

abhin@Abhinavz-Acer MINGW64 ~ (master)
$ cd Downloads/

abhin@Abhinavz-Acer MINGW64 ~/Downloads (master)
$ chmod 400 "serverkey-01.pem"

abhin@Abhinavz-Acer MINGW64 ~/Downloads (master)
$ |

```

```
ssh -i (keyname).pem (username)@(public ipv4 dns address)
```

where keyname is name of the key you created. (server-01.pem). Other details can be found on the Instance dashboard.

```
abhin@Abhinavz-Acer MINGW64 ~/Downloads (master)
$ ssh -i "serverkey-01.pem" ec2-user@ec2-54-165-196-241.compute-1.amazonaws.com
#_
~\##### Amazon Linux 2023
~~~\#####\
~~~~\###|
~~~~\#/_____ https://aws.amazon.com/linux/amazon-linux-2023
~~~~V~' ~->
~~~~
~~~~.-.
~~~~-/
~/m/'
```

### 3. Installation Of Docker on three machines

```
➤ sudo yum install docker -y
```

```
[ec2-user@ip-172-31-90-103 ~]$ sudo yum install docker -y
Last metadata expiration check: 0:13:41 ago on Sat Sep 14 03:42:27 2024.
Dependencies resolved.
=====
Package                Arch      Version                               Repository    Size
=====
Installing:
docker                 x86_64    25.0.6-1.amzn2023.0.2               amazonlinux   44 M
Installing dependencies:
containerd             x86_64    1.7.20-1.amzn2023.0.1               amazonlinux   35 M
iptables-libse         x86_64    1.8.8-3.amzn2023.0.2               amazonlinux   401 k
iptables-nft           x86_64    1.8.8-3.amzn2023.0.2               amazonlinux   183 k
libcgroup              x86_64    3.0-1.amzn2023.0.1                  amazonlinux   75 k
libnetfilter_conntrack x86_64    1.0.8-2.amzn2023.0.2               amazonlinux   58 k
libnfnfnetlink          x86_64    1.0.1-19.amzn2023.0.2              amazonlinux   30 k
libnftnl               x86_64    1.2.2-2.amzn2023.0.2               amazonlinux   84 k
pigz                   x86_64    2.5-1.amzn2023.0.3                  amazonlinux   83 k
runc                   x86_64    1.1.13-1.amzn2023.0.1              amazonlinux   3.2 M
Transaction Summary
=====
Install 10 Packages

Total download size: 84 M
Installed size: 317 M
Downloading Packages:
(1/10): iptables-libse-1.8.8-3.amzn2023.0.2.x86_64.rpm | 2.2 MB/s | 401 kB | 00:00
(2/10): iptables-nft-1.8.8-3.amzn2023.0.2.x86_64.rpm | 2.5 MB/s | 183 kB | 00:00
(3/10): libcgroup-3.0-1.amzn2023.0.1.x86_64.rpm | 1.3 MB/s | 75 kB | 00:00
(4/10): libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64.rpm | 1.3 MB/s | 58 kB | 00:00
(5/10): libnfnfnetlink-1.0.1-19.amzn2023.0.2.x86_64.rpm | 938 kB/s | 30 kB | 00:00
(6/10): libnftnl-1.2.2-2.amzn2023.0.2.x86_64.rpm | 1.6 MB/s | 84 kB | 00:00
(7/10): pigz-2.5-1.amzn2023.0.3.x86_64.rpm | 1.7 MB/s | 83 kB | 00:00
(8/10): runc-1.1.13-1.amzn2023.0.1.x86_64.rpm | 21 MB/s | 3.2 MB | 00:00
(9/10): containerd-1.7.20-1.amzn2023.0.1.x86_64.rpm | 31 MB/s | 35 MB | 00:01
(10/10): docker-25.0.6-1.amzn2023.0.2.x86_64.rpm | 28 MB/s | 44 MB | 00:01
-----
Total | 52 MB/s | 84 MB | 00:01
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing :
Installing : runc-1.1.13-1.amzn2023.0.1.x86_64
Installing : containerd-1.7.20-1.amzn2023.0.1.x86_64
Running scriptlet: containerd-1.7.20-1.amzn2023.0.1.x86_64
Installing : pigz-2.5-1.amzn2023.0.3.x86_64
Installing : libnftnl-1.2.2-2.amzn2023.0.2.x86_64
Installing : libnfnfnetlink-1.0.1-19.amzn2023.0.2.x86_64
Installing : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64
Installing : iptables-libse-1.8.8-3.amzn2023.0.2.x86_64
Installing : iptables-nft-1.8.8-3.amzn2023.0.2.x86_64
```

```

Installed:
  containerd-1.7.20-1.amzn2023.0.1.x86_64      docker-25.0.6-1.amzn2023.0.2.x86_64
  libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64  libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64

complete!
[ec2-user@ip-172-31-90-103 ~]$

```

- Configure cgroup in a daemon.json  
(this can be done by creating the file and using **nano** text editor)

```

{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}

```

```

[ec2-user@ip-172-31-90-103 docker]$ sudo nano daemon.json
[ec2-user@ip-172-31-90-103 docker]$ |

```

```

ec2-user@ip-172-31-90-103:/etc/docker
GNU nano 5.8      daemon.json      Modified
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}

```

- Enable and start docker and also load the daemon.json

```

sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker

```

```

[ec2-user@ip-172-31-90-103 docker]$ sudo systemctl enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
[ec2-user@ip-172-31-90-103 docker]$ sudo systemctl daemon-reload
[ec2-user@ip-172-31-90-103 docker]$ sudo systemctl restart docker
[ec2-user@ip-172-31-90-103 docker]$ |

```

- Check if docker is installed

```

[ec2-user@ip-172-31-90-103 docker]$ docker --version
Docker version 25.0.5, build 5dc9bcc
[ec2-user@ip-172-31-90-103 docker]$ |

```

#### 4. Install Kubernetes on all 3 machines

- SELinux needs to be disabled before configuring kubelet

**sudo setenforce 0**

**sudo sed -i 's/^SELINUX=enforcing\$/SELINUX=permissive/' /etc/selinux/config**

```
[ec2-user@ip-172-31-90-103 docker]$ sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
[ec2-user@ip-172-31-90-103 docker]$ |
```

- Add Kubernetes using the repo

(this is done by creating **kubernetes.repo** file in **/etc/yum.repos.d** and configuring it using **nano** editor)

**[kubernetes]**

**name=Kubernetes**

**baseurl=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/**

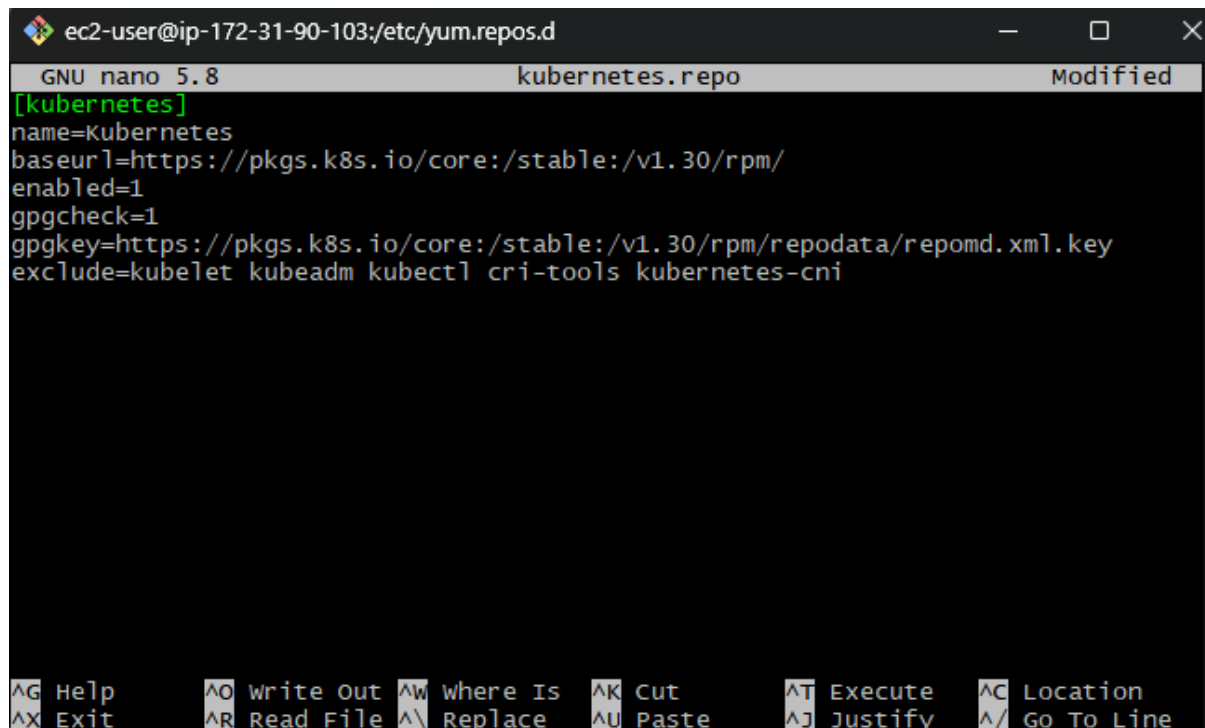
**enabled=1**

**gpgcheck=1**

**gpgkey=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/repodata/repomd.xml.key**

**exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni**

```
[ec2-user@ip-172-31-90-103 docker]$ cd /etc/yum.repos.d/
[ec2-user@ip-172-31-90-103 yum.repos.d]$ ls
amazonlinux.repo  kernel-livepatch.repo
[ec2-user@ip-172-31-90-103 yum.repos.d]$ sudo nano kubernetes.repo
[ec2-user@ip-172-31-90-103 yum.repos.d]$ ls
amazonlinux.repo  kernel-livepatch.repo  kubernetes.repo
[ec2-user@ip-172-31-90-103 yum.repos.d]$ |
```



The screenshot shows a terminal window with the nano text editor open at the file `/etc/yum.repos.d/kubernetes.repo`. The editor's title bar indicates it is GNU nano 5.8. The file content is as follows:

```
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
```

The bottom of the screen shows the nano editor's command shortcuts: `^G Help`, `^O Write Out`, `^W Where Is`, `^K Cut`, `^T Execute`, `^C Location`, `^X Exit`, `^R Read File`, `^U Replace`, `^V Paste`, `^J Justify`, and `^_ Go To Line`.

- Update packages list using **sudo yum update**

```
[ec2-user@ip-172-31-90-103 yum.repos.d]$ sudo yum update
kubernetes                               125 kB/s | 17 kB      00:00
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-172-31-90-103 yum.repos.d]$
```

- Install kubelet kubeadm kubectl

**sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes**

```
[ec2-user@ip-172-31-90-103 yum.repos.d]$ sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
Last metadata expiration check: 0:00:42 ago on Sat Sep 14 04:08:20 2024.
Dependencies resolved.
=====
Package                        Arch      Version                               Repository    Size
=====
Installing:
kubeadm                       x86_64    1.30.5-150500.1.1                   kubernetes    10 M
kubectl                       x86_64    1.30.5-150500.1.1                   kubernetes    10 M
kubelet                       x86_64    1.30.5-150500.1.1                   kubernetes    17 M
Installing dependencies:
contrack-tools                x86_64    1.4.6-2.amzn2023.0.2                 amazonlinux    208 k
cri-tools                     x86_64    1.30.1-150500.1.1                   kubernetes     8.6 M
kubernetes-cni                x86_64    1.4.0-150500.1.1                   kubernetes     6.7 M
libnetfilter_cthelper         x86_64    1.0.0-21.amzn2023.0.2                 amazonlinux     24 k
libnetfilter_cttimeout        x86_64    1.0.0-19.amzn2023.0.2                 amazonlinux     24 k
libnetfilter_queue            x86_64    1.0.5-2.amzn2023.0.2                 amazonlinux     30 k
Transaction Summary
=====
Install 9 Packages

Total download size: 53 M
Installed size: 292 M
Downloading Packages:
(1/9): libnetfilter_cttimeout-1.0.0-19.amzn2023 448 kB/s | 24 kB    00:00
(2/9): libnetfilter_cthelper-1.0.0-21.amzn2023. 409 kB/s | 24 kB    00:00
(3/9): libnetfilter_queue-1.0.5-2.amzn2023.0.2. 1.5 MB/s | 30 kB    00:00
(4/9): contrack-tools-1.4.6-2.amzn2023.0.2.x86 1.8 MB/s | 208 kB    00:00
(5/9): cri-tools-1.30.1-150500.1.1.x86_64.rpm 28 MB/s | 8.6 MB    00:00
(6/9): kubectl-1.30.5-150500.1.1.x86_64.rpm 23 MB/s | 10 MB     00:00
(7/9): kubeadm-1.30.5-150500.1.1.x86_64.rpm 18 MB/s | 10 MB     00:00
(8/9): kubelet-1.30.5-150500.1.1.x86_64.rpm 37 MB/s | 17 MB     00:00
(9/9): kubernetes-cni-1.4.0-150500.1.1.x86_64.r 20 MB/s | 6.7 MB    00:00
-----
Total                                           56 MB/s | 53 MB    00:00
```

```
Installed:
contrack-tools-1.4.6-2.amzn2023.0.2.x86_64
cri-tools-1.30.1-150500.1.1.x86_64
kubeadm-1.30.5-150500.1.1.x86_64
kubectl-1.30.5-150500.1.1.x86_64
kubelet-1.30.5-150500.1.1.x86_64
kubernetes-cni-1.4.0-150500.1.1.x86_64
libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64
libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64
libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64

Complete!
[ec2-user@ip-172-31-90-103 yum.repos.d]$
```

- After installing Kubernetes, we need to configure internet options to allow bridging.

**sudo swapoff -a**

**echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /etc/sysctl.conf**

**sudo sysctl -p**

```
[ec2-user@ip-172-31-90-103 yum.repos.d]$ . sudo swapoff -a
-bash: sudo: No such file or directory
[ec2-user@ip-172-31-90-103 yum.repos.d]$ sudo swapoff -a
[ec2-user@ip-172-31-90-103 yum.repos.d]$ echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /etc/sysctl.conf
net.bridge.bridge-nf-call-iptables=1
[ec2-user@ip-172-31-90-103 yum.repos.d]$ sudo sysctl -p
net.bridge.bridge-nf-call-iptables = 1
[ec2-user@ip-172-31-90-103 yum.repos.d]$ |
```



5. Perform this ONLY on the Master machine Initialize the Kubecluster

**sudo kubeadm init --podnetwork-cidr=10.244.0.0/16**

```
[ec2-user@ip-172-31-90-103 yum.repos.d]$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all
I0914 04:12:17.448521 27990 version.go:256] remote version is much newer: v1.31.0; falling back to: stable-1.30
[init] Using Kubernetes version: v1.30.4
[preflight] Running pre-flight checks
        [WARNING NumCPU]: the number of available CPUs 1 is less than the required 2
        [WARNING Mem]: the system RAM (949 MB) is less than the minimum 1700 MB
        [WARNING FileExisting-socat]: socat not found in system path
        [WARNING FileExisting-tc]: tc not found in system path
        [WARNING Service-Kubelet]: kubelet service is not enabled, please run 'systemctl enable kubelet.service'
[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'
W0914 04:12:17.711154 27990 checks.go:844] detected that the sandbox image "registry.k8s.io/pause:3.8" of the container runtime is inconsistent with that used by kubeadm. It is recommended to use "registry.k8s.io/pause:3.9" as the CRI sandbox image.
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
```

```
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 172.31.90.103:6443 --token 0zk8w3.xyegkydsy42vfscm \
--discovery-token-ca-cert-hash sha256:31c672892b19dcb869fc46362d189234128f5bfc302bd41ae8c6078c56173f00
[ec2-user@ip-172-31-90-103 yum.repos.d]$
```

➤ Save the token

```
kubeadm join 172.31.90.103:6443 --token 0zk8w3.xyegkydsy42vfscm \
--discovery-token-ca-cert-hash sha256:31c672892b19dcb869fc46362d189234128f5bfc302bd41ae8c6078c56173f00
```

➤ Copy the mkdir and chown commands from the top and execute them

**mkdir -p \$HOME/.kube**

```
[ec2-user@ip-172-31-90-103 yum.repos.d]$ mkdir -p $HOME/.kube
[ec2-user@ip-172-31-90-103 yum.repos.d]$ |
```

**sudo cp -i /etc/kubernetes/admin.conf \$HOME/.kube/config**

```
[ec2-user@ip-172-31-90-103 yum.repos.d]$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
[ec2-user@ip-172-31-90-103 yum.repos.d]$
```

**sudo chown \$(id -u):\$(id -g) \$HOME/.kube/config**

```
[ec2-user@ip-172-31-90-103 yum.repos.d]$
sudo chown $(id -u):$(id -g) $HOME/.kube/config
[ec2-user@ip-172-31-90-103 yum.repos.d]$
```

- Then, add a common networking plugin called flannel file as mentioned in the code.  
**kubectl apply -f <https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>**

```
[ec2-user@ip-172-31-90-103 yum.repos.d]$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
namespace/kube-flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
[ec2-user@ip-172-31-90-103 yum.repos.d]$
```

## 6. Perform this ONLY on the worker machines

- Paste the below command on all 2 worker machines

```
sudo yum install iproute-tc -y
sudo systemctl enable kubelet
sudo systemctl restart kubelet
```

- Now use the token from earlier to join into worker instances

```
kubeadm join 172.31.90.103:6443 --token 0zk8w3.xyegkydsy42vfscm \
--discovery-token-ca-cert-hash
sha256:31c672892b19dcb869fc46362d189234128f5bfc302bd41ae8c6078c56173f00
```

- **kubectl get nodes** to check whether master and worker nodes are connected successfully

```
[ec2-user@ip-54-211-147-10 docker]$ kubectl get nodes
NAME                                STATUS    ROLES    AGE    VERSION
ip-172-31-90-103.ec2.internal      Ready    control-plane   3m21s   v1.30.5
```

## Conclusion:

An EC2 instance was created on AWS Linux, and Docker, Kubernetes, Kubelet, Kubeadm, and Kubectl were installed. Kubernetes was initialized on the master node, which provided a token for connecting the master and worker nodes. On the slave node, `iproute` was installed, and Kubelet was enabled and restarted. However, there was an issue with joining the slave node to the cluster, resulting in only the master node being listed when running `kubectl get nodes`