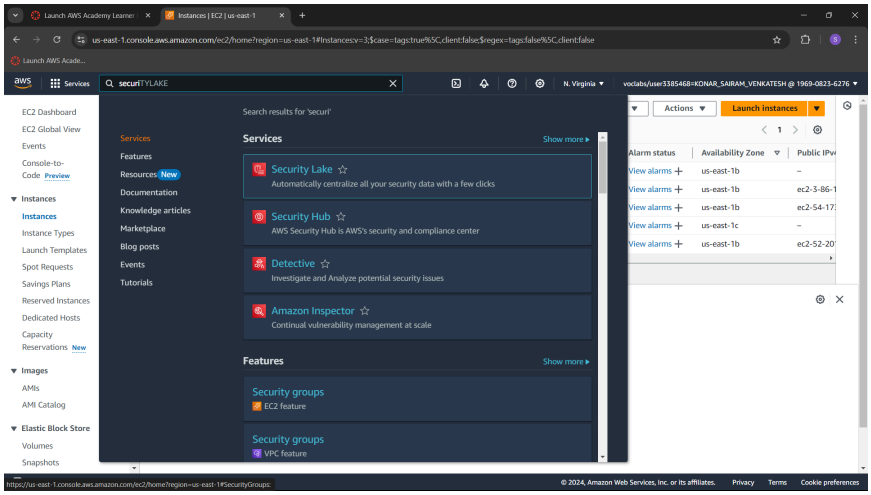
**Sairam Konar D15C 27 2024-25**

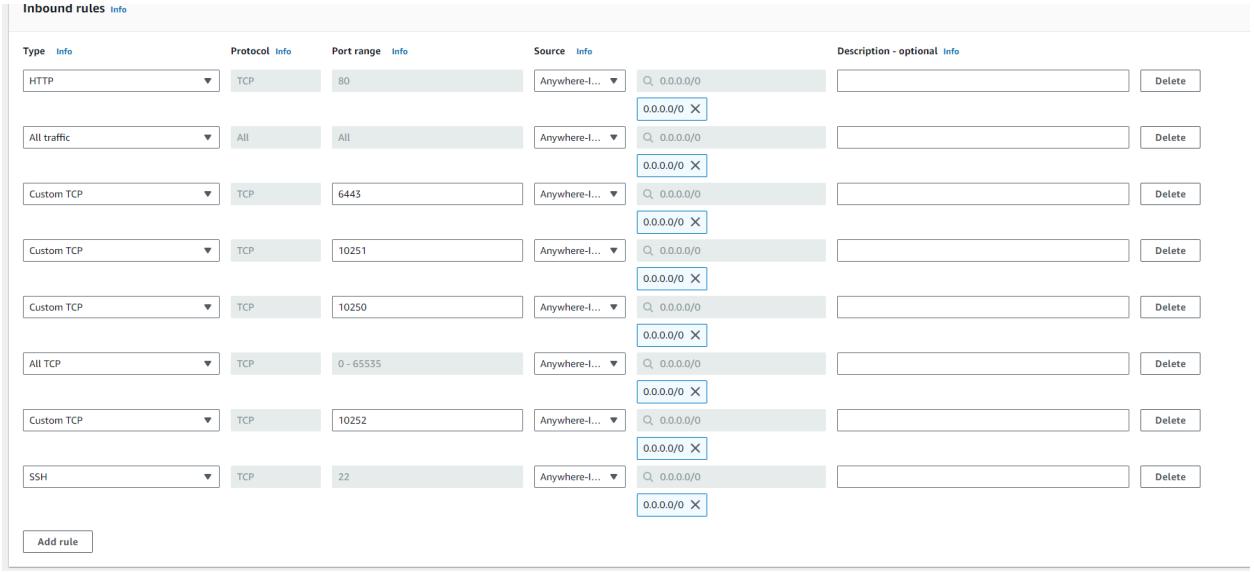
Aim: To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud Platforms.

Prerequisites:

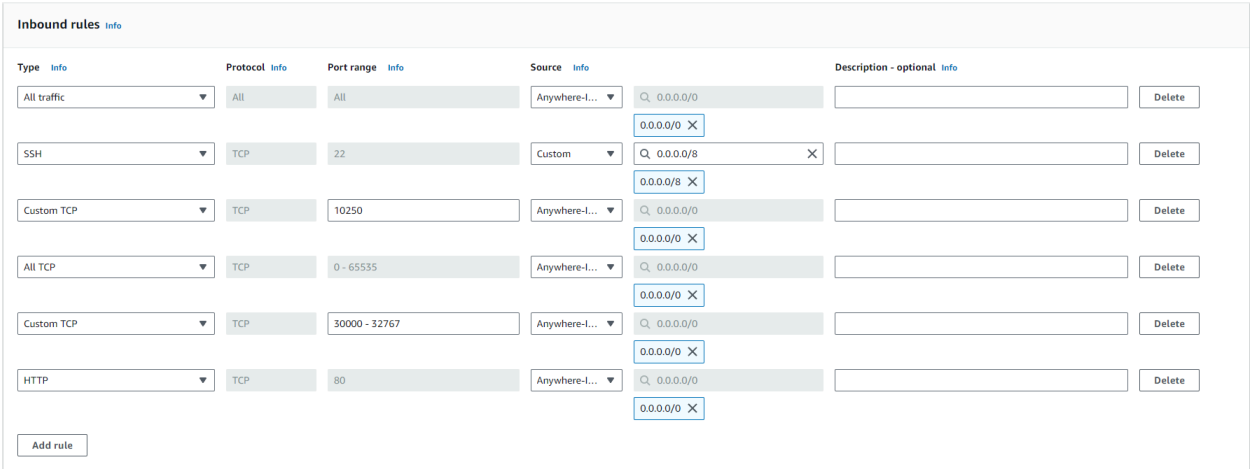
Create 2 security groups by navigating to Search → Security Groups.




Group 1: Master

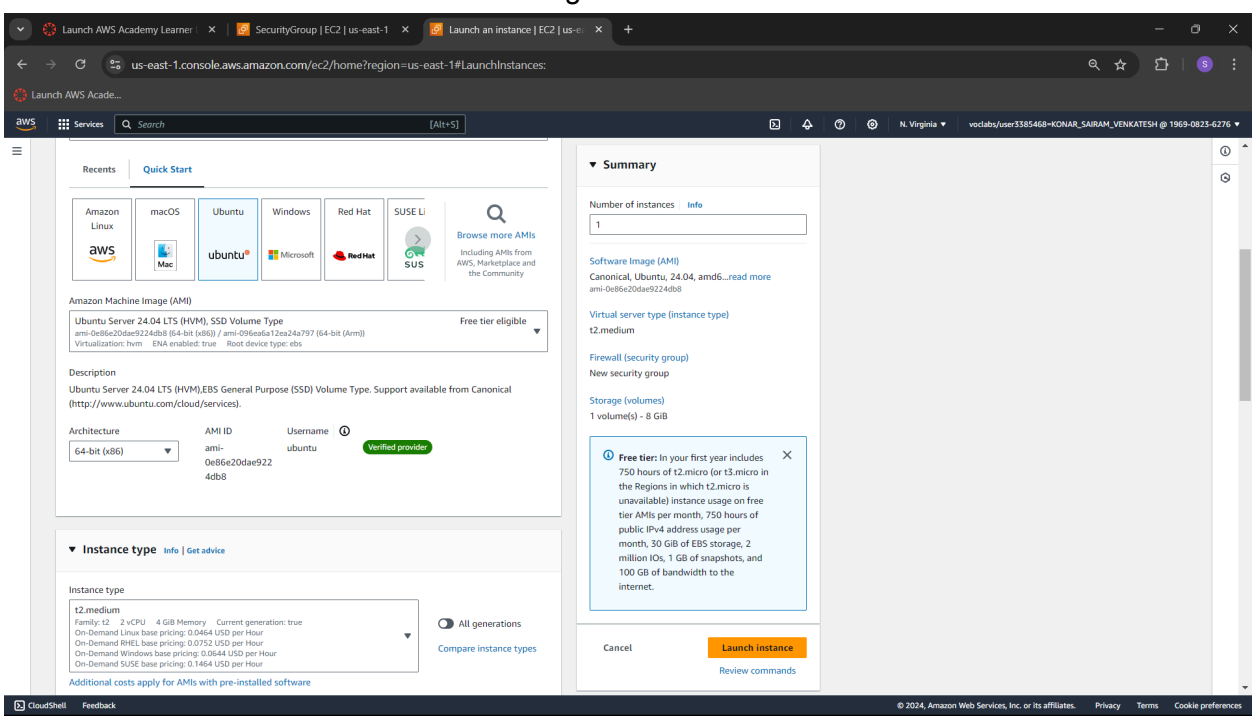


## Group 2: Nodes



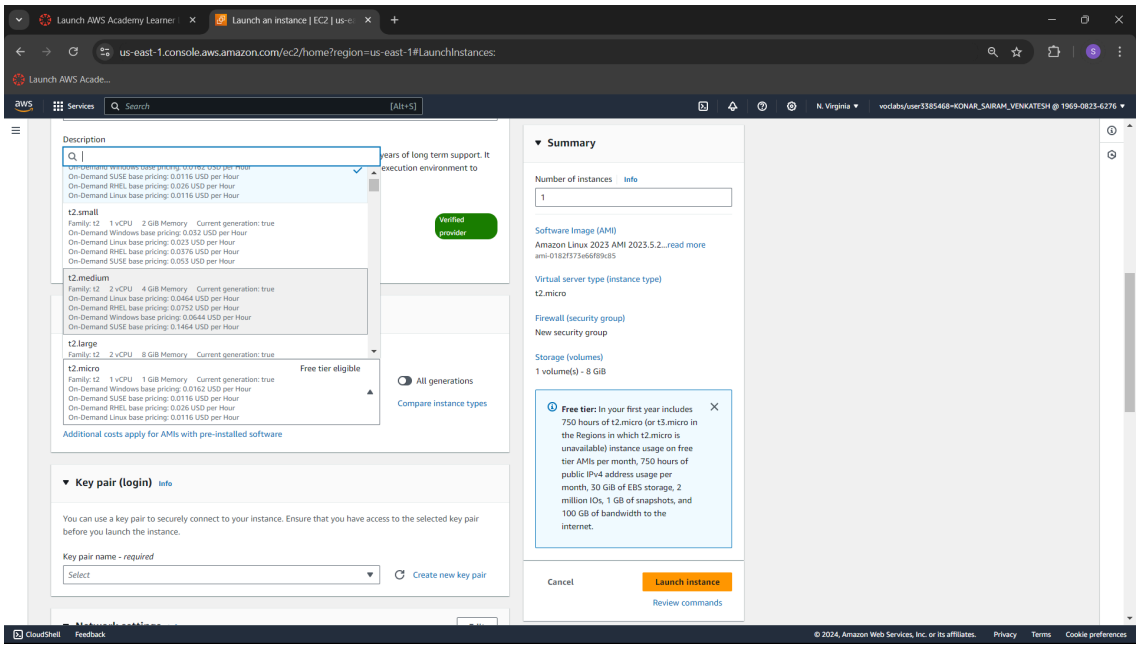

## Step 1: Set Up EC2 Instances.

1) Set up 3 EC2 instances called master, node1, node2
   Select Amazon Linux as the OS image.

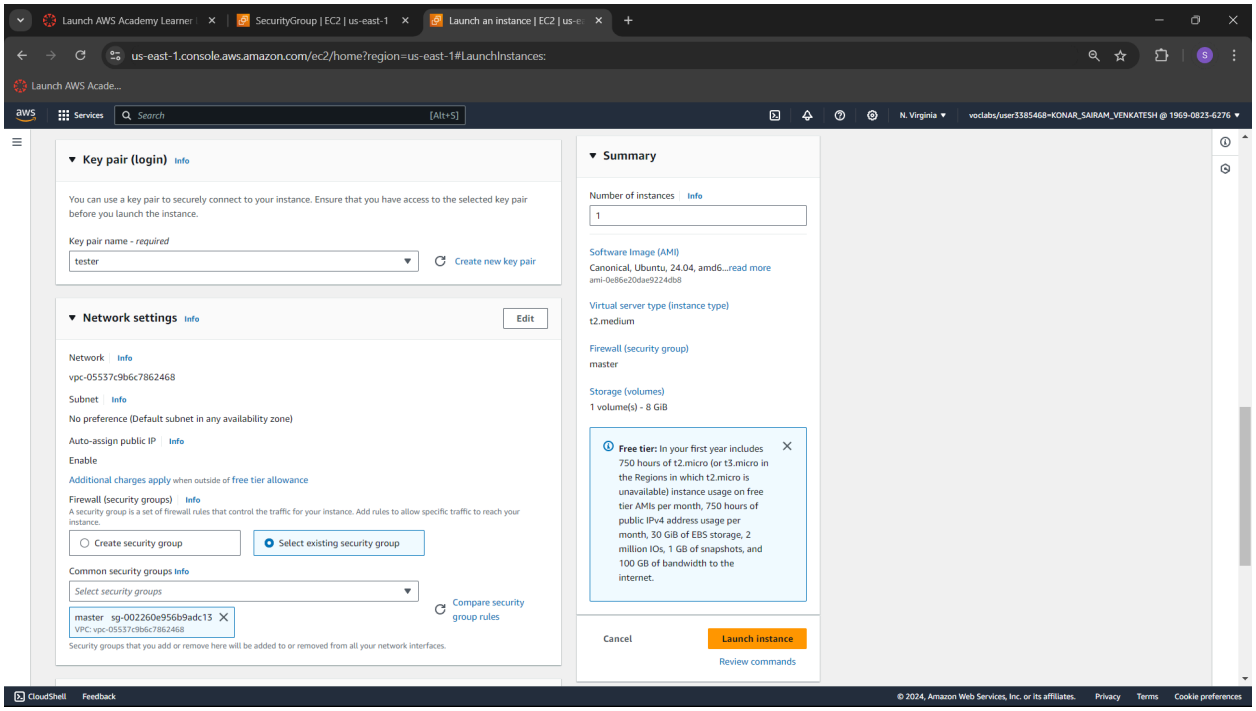



**IMPORTANT:** The default instance type and free one provided by AWS is t2.micro, which provides only 1CPU and 1 GiB of memory. For running Kubernetes, a minimum of 2 CPUs and 2GiB of RAM is required, hence change **t2.micro** to **t2.medium.**

**Sairam Konar D15C 27 2024-25**

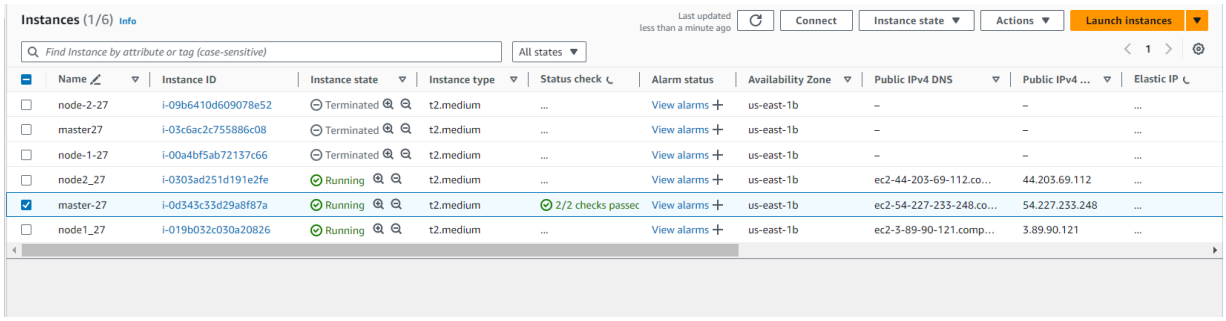



2) Create a key pair as you need the .pem file (private key file) on your system.
3) Click on 'Select existing security group' and select the master group for master instance, node group for both node instances.

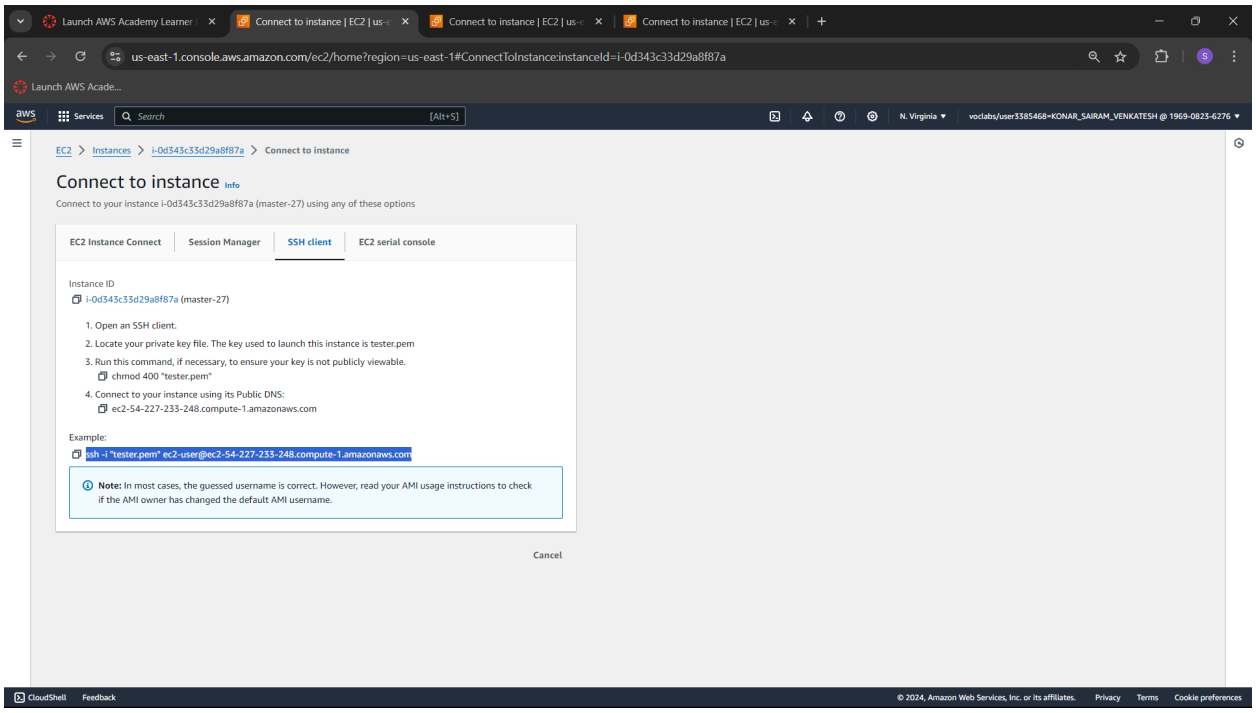


**Sairam Konar D15C 27 2024-25**

4) These are the instances that are created. Click on the Instance ID of master.




5) Click on Connect. This directs you to a connect dashboard. Click on SSH client, you get a SSH command. Use this command to access the instance terminal on your local system.



6) Go to the folder where your private key file (.pem file) is installed. Right click → Open in terminal.

```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\saira\OneDrive\Desktop\AdvDevOps\lab 3 4>
```

Paste the SSH command here and run it.

You might get the error of UNPROTECTED KEY FILE. This is because the .pem access Is with all users of teh system. Run the following commands to change the access to only the current user.

- icacls "C:\Users\saira\OneDrive\Desktop\AdvDevOps\lab 3 4\tester.pem" /inheritance:r
- icacls "C:\Users\saira\OneDrive\Desktop\AdvDevOps\lab 3 4\tester.pem" /grant:r "%USERNAME%:F"

Now rerun the SSH command.

```
PS C:\Users\saira\OneDrive\Desktop\AdvDevOps\lab 3 4> ssh -i "tester.pem" ec
2-user@ec2-54-227-233-248.compute-1.amazonaws.com
The authenticity of host 'ec2-54-227-233-248.compute-1.amazonaws.com (54.227
.233.248)' can't be established.
ED25519 key fingerprint is SHA256:KEZfN8TkgsivFhCjxb34Xb1bs+AOHbm7YKH5AhjCC9
s.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-227-233-248.compute-1.amazonaws.com' (ED2
5519) to the list of known hosts.
       #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-80-245 ~]$
```

7) After doing the above steps, the terminal for all 3 nodes can be seen.

```
ec2-user@ip-172-31-80-245:~
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka
.ms/PSWindows

PS C:\Users\saira\OneDrive\Desktop\AdvDevOps\lab 3 4> ssh -i "tester.pem" ec
2-user@ec2-54-227-233-248.compute-1.amazonaws.com
The authenticity of host 'ec2-54-227-233-248.compute-1.amazonaws.com (54.227
.233.248)' can't be established.
ED25519 key fingerprint is SHA256:KEZfN8TkgsivFhCjxb34Xb1bs+AOHbm7YKH5AhjCC9
s.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-227-233-248.compute-1.amazonaws.com' (ED2
5519) to the list of known hosts.
       #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-80-245 ~]$
```

```
ec2-user@ip-172-31-86-11:~
2-user@ec2-3-89-90-121.compute-1.amazonaws.com
The authenticity of host 'ec2-3-89-90-121.compute-1.amazonaws.com (3.89.90.1
21)' can't be established.
ED25519 key fingerprint is SHA256:THchr7pke1QtAjoA0Gl91aSd+fUCu8b2D/rW3jiZYO
o.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-89-90-121.compute-1.amazonaws.com' (ED2551
9) to the list of known hosts.
       #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-86-11 ~]$
```

```
ec2-user@ip-172-31-95-90:~
2-user@ec2-44-203-69-112.compute-1.amazonaws.com
The authenticity of host 'ec2-44-203-69-112.compute-1.amazonaws.com (44.203.
69.112)' can't be established.
ED25519 key fingerprint is SHA256:+95ilZ7Q193UkcmhX/ErjNsNLCxSUbctHMSyrCSe6E
k.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-44-203-69-112.compute-1.amazonaws.com' (ED25
519) to the list of known hosts.
       #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-95-90 ~]$
```

**PERFORM THE FOLLOWING STEPS ON ALL 3 MACHINES**

Step 2: Installation of Docker

1) Use command

'sudo su'

This allows you to act as the root user of the terminal

```
       _/m/'
[ec2-user@ip-172-31-80-245 ~]$ sudo su
[root@ip-172-31-80-245 ec2-user]#
```

2) We can install docker using YUM(Yellowdog Updater, Modified). Use the command ‘yum install docker -y’

```
[root@ip-172-31-80-245 ec2-user]# yum install docker -y
Last metadata expiration check: 0:05:22 ago on Thu Sep 26 03:53:56 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-libs           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
 libnfnetlink            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
```

```
Installed:
  containerd-1.7.20-1.amzn2023.0.1.x86_64
  docker-25.0.6-1.amzn2023.0.2.x86_64
  iptables-libs-1.8.8-3.amzn2023.0.2.x86_64
  iptables-nft-1.8.8-3.amzn2023.0.2.x86_64
  libcgroup-3.0-1.amzn2023.0.1.x86_64
  libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64
  libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64
  libnftnl-1.2.2-2.amzn2023.0.2.x86_64
  pigz-2.5-1.amzn2023.0.3.x86_64
  runc-1.1.13-1.amzn2023.0.1.x86_64

Complete!
[root@ip-172-31-80-245 ec2-user]#
```

3) Now, configure a daemon.json file by using the following chain of commands.

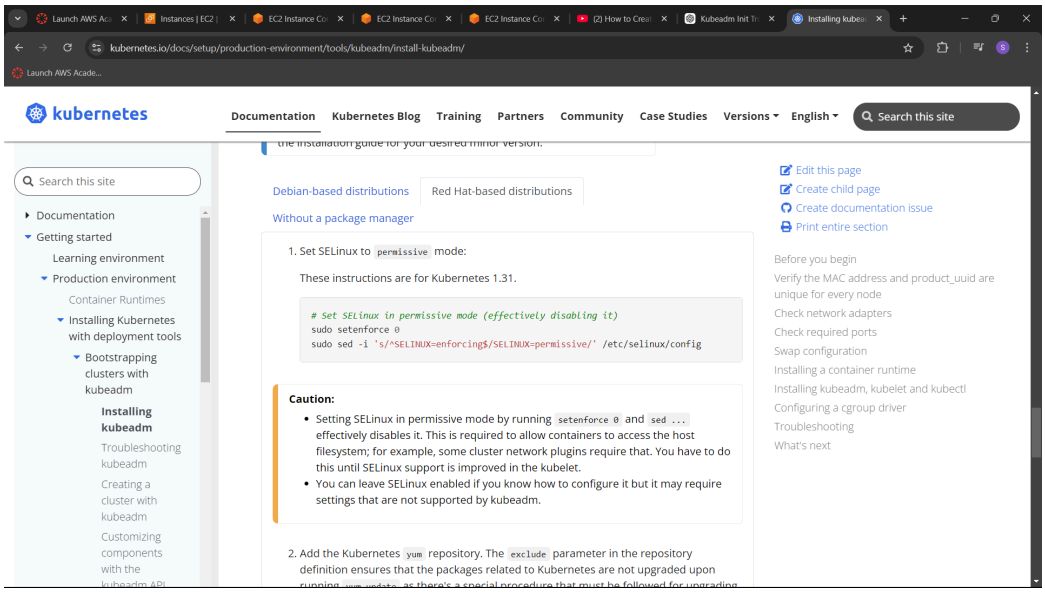
- cd /etc/docker
- cat <<EOF | sudo tee /etc/docker/daemon.json
  {
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
  "max-size": "100m"
  },
  "storage-driver": "overlay2"
  }
  EOF
- sudo systemctl enable docker

- sudo systemctl daemon-reload
- sudo systemctl restart docker

```
[root@ip-172-31-80-245 ec2-user]# cd /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
"exec-opts": ["native.cgroupdriver=systemd"],
"log-driver": "json-file",
"log-opts": {
"max-size": "100m"
},
"storage-driver": "overlay2"
}
EOF
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
{
"exec-opts": ["native.cgroupdriver=systemd"],
"log-driver": "json-file",
"log-opts": {
"max-size": "100m"
},
"storage-driver": "overlay2"
}
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service →
 /usr/lib/systemd/system/docker.service.
[root@ip-172-31-80-245 docker]#
```

## Step 3: Installing Kubernetes

1) For installing kubernetes, we will be using kubeadm, a framework used for creating kubernetes clusters using command line.
   https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/
   The following will be visibe when you visit the website.



2) Select **red hat-based distributions** as amazon linux is based on red hat.

   sudo setenforce 0
   → sets SELinux to permissive mode

   sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
   → edits the SELinux configuration file (/etc/selinux/config) to make the change persistent across reboots. If not used, SELinux reverts to enforcing mode after reboot.

   Setting SELinux to permissive mode during Kubernetes installation prevents permission-related issues with container runtimes and components that may not function correctly under SELinux's enforcing policies.

   Run the following commands:
   - sudo setenforce 0
   - sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
   - cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
     [kubernetes]
     name=Kubernetes
     baseurl=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/
     enabled=1
     gpgcheck=1
     gpgkey=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/repodata/repomd.xml.key
     exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
     EOF

   This comamnd is a repository script to create a kubernetes repository

```
/usr/lib/systemd/system/docker.service.
[root@ip-172-31-80-245 docker]# sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
[root@ip-172-31-80-245 docker]# cat <<EOF | sudo tee /etc/yum.repos.d/kubern
etes.repo
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
[root@ip-172-31-80-245 docker]#
```

- yum repolist

This command shows the repositories created on the machine.

```
[root@ip-172-31-80-245 docker]# yum repolist
repo id                  repo name
amazonlinux              Amazon Linux 2023 repository
kernel-livepatch         Amazon Linux 2023 Kernel Livepatch repository
kubernetes               Kubernetes
[root@ip-172-31-80-245 docker]#
```

Next step is to install kubelet, kubeadm, kubectl

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

```
[root@ip-172-31-80-245 docker]# sudo yum install -y kubelet kubeadm kubectl
--disableexcludes=kubernetes
Kubernetes                                      65 kB/s | 9.4 kB     00:00
Dependencies resolved.
================================================================================
 Package                 Arch    Version                 Repository    Size
================================================================================
Installing:
 kubeadm                 x86_64  1.31.1-150500.1.1       kubernetes    11 M
 kubectl                 x86_64  1.31.1-150500.1.1       kubernetes    11 M
 kubelet                 x86_64  1.31.1-150500.1.1       kubernetes    15 M
Installing dependencies:
 conntrack-tools         x86_64  1.4.6-2.amzn2023.0.2    amazonlinux  208 k
 cri-tools               x86_64  1.31.1-150500.1.1       kubernetes   6.9 M
 kubernetes-cni          x86_64  1.5.1-150500.1.1        kubernetes   7.1 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
================================================================================
```

```
Installed:
  conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64
  cri-tools-1.31.1-150500.1.1.x86_64
  kubeadm-1.31.1-150500.1.1.x86_64
  kubectl-1.31.1-150500.1.1.x86_64
  kubelet-1.31.1-150500.1.1.x86_64
  kubernetes-cni-1.5.1-150500.1.1.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!
[root@ip-172-31-80-245 docker]#
```

Now, we need to enable the kubelet service. Run the command

- sudo systemctl enable --now kubelet

```
[root@ip-172-31-80-245 docker]# sudo systemctl enable --now kubelet
Created symlink /etc/systemd/system/multi-user.target.wants/kubelet.service
→ /usr/lib/systemd/system/kubelet.service.
[root@ip-172-31-80-245 docker]#
```

## PERFORM THE FOLLOWING ON ONLY THE MASTER MACHINE

1) Firstly, we need to initialize kubernetes. For this, run the command:
   - kubeadm init

```
[root@ip-172-31-21-124 ec2-user]# kubeadm init
[init] Using Kubernetes version: v1.31.0
[preflight] Running pre-flight checks
        [WARNING FileExisting-socat]: socat not found in system path
        [WARNING FileExisting-tc]: tc not found in system path
[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 beforehand using 'kubeadm config images pull'
W0913 17:19:05.142482   28614 checks.go:846] detected that the sandbox image "registry.k8s.io/pause:3.8" of the container runtime is inconsistent with that used by kubeadm.It is rec
ommended to use "registry.k8s.io/pause:3.10" as the CRI sandbox image.
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [ip-172-31-21-124.ec2.internal kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and
IPs [10.96.0.1 172.31.21.124]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [ip-172-31-21-124.ec2.internal localhost] and IPs [172.31.21.124 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [ip-172-31-21-124.ec2.internal localhost] and IPs [172.31.21.124 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "super-admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
```

```
[api-check] Waiting for a healthy API server. This can take up to 4m0s
[api-check] The API server is healthy after 5.002506688s
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config" in namespace kube-system with the configuration for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node ip-172-31-21-124.ec2.internal as control-plane by adding the labels: [node-role.kubernetes.io/control-plane node.kubernetes.io/exclude-from-ext
ernal-load-balancers]
[mark-control-plane] Marking the node ip-172-31-21-124.ec2.internal as control-plane by adding the taints [node-role.kubernetes.io/control-plane:NoSchedule]
[bootstrap-token] Using token: 76rlql.qpqk9pmv8t9osgq4
[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 credentials
[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 172.31.21.124:6443 --token 76rlql.qpqk9pmv8t9osgq4 \
        --discovery-token-ca-cert-hash sha256:a03f677cdc93d202c5f89e2370eb7231d212211bac6f816cddcb547b7f00a4f2
[root@ip-172-31-21-124 ec2-user]#
```

From the output, we will receive a command that is used to link the nodes to the master. Copy it and save it somewhere local.

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

kubeadm join 172.31.80.245:6443 --token g7dyjj.r62w19oc05n7k4kc \
        --discovery-token-ca-cert-hash sha256:2975de445064ffc56887b7b0ef79d7
```

2) From the output, we receive the following commands:

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

   Run these commands.

```
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
```

3) To check whether nodes are connected, run the command
   - kubectl get nodes

   This output shows only master is connected right now.

```
[root@ip-172-31-80-245 docker]# kubectl get nodes
NAME                              STATUS     ROLES           AGE    VERSION
ip-172-31-80-245.ec2.internal     NotReady   control-plane   112s   v1.31.1
[root@ip-172-31-80-245 docker]#
```

## PERFORM THE FOLLOWING ONLY ON THE NODE MACHINES

Use the command that you had copied before and use them on the node machines.

```
root@ip-172-31-86-11:/etc/do

em get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/conf
ig.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/li
b/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-check] Waiting for a healthy kubelet at http://127.0.0.1:10248/heal
thz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 501.57413ms
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was recei
ved.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the clust
er.

[root@ip-172-31-86-11 docker]#
```

```
root@ip-172-31-95-90:/etc/dc

em get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/conf
ig.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/li
b/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-check] Waiting for a healthy kubelet at http://127.0.0.1:10248/heal
thz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 1.001297846s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was recei
ved.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the clust
er.

[root@ip-172-31-95-90 docker]#
```

**NOW GO BACK TO THE MASTER MACHINE AND RERUN 'kubectl get nodes'**

```
[root@ip-172-31-80-245 docker]# kubectl get nodes
NAME                              STATUS     ROLES           AGE     VERSION
ip-172-31-80-245.ec2.internal     NotReady   control-plane   2m18s   v1.31.1
ip-172-31-86-11.ec2.internal      NotReady   <none>          9s      v1.31.1
ip-172-31-95-90.ec2.internal      NotReady   <none>          2s      v1.31.1
[root@ip-172-31-80-245 docker]#
```

As we see, the status of the nodes are <NOT READY>. To change it, we need to install a network CNI plugin.

**Use the following command only on the master machine:**:

kubectl apply -f

https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

Now run 'kubectl get nodes' again.

```
[root@ip-172-31-80-245 docker]# kubectl get nodes
NAME                              STATUS   ROLES           AGE     VERSION
ip-172-31-80-245.ec2.internal     Ready    control-plane   9m33s   v1.31.1
ip-172-31-86-11.ec2.internal      Ready    <none>          7m24s   v1.31.1
ip-172-31-95-90.ec2.internal      Ready    <none>          7m17s   v1.31.1
```

## Conclusion:

In this experiment, we have learned how to create kubernetes clusters on a linux terminal, how the ssh command works on a local terminal and what requirements are necessary to create kubernetes clusters. We have used many command inline tools of Kubernetes like kubecmd, kubectl to set up the clusters and work with docker container to perform the connection of the nodes with master machine.