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

**Theory:**
Container-based microservices architectures have revolutionized how development and operations teams test and deploy modern software. Containers allow companies to scale and deploy applications more efficiently, but they also introduce new challenges, adding complexity by creating a whole new infrastructure ecosystem.
Today, both large and small software companies are deploying thousands of container instances daily. Managing this level of complexity at scale requires advanced tools. Enter Kubernetes.

Originally developed by Google, Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. Kubernetes has quickly become the de facto standard for container orchestration and is the flagship project of the Cloud Native Computing Foundation (CNCF), supported by major players like Google, AWS, Microsoft, IBM, Intel, Cisco, and Red Hat. Kubernetes simplifies the deployment and operation of applications in a microservice architecture by providing an abstraction layer over a group of hosts. This allows development teams to deploy their applications while Kubernetes takes care of key tasks, including:

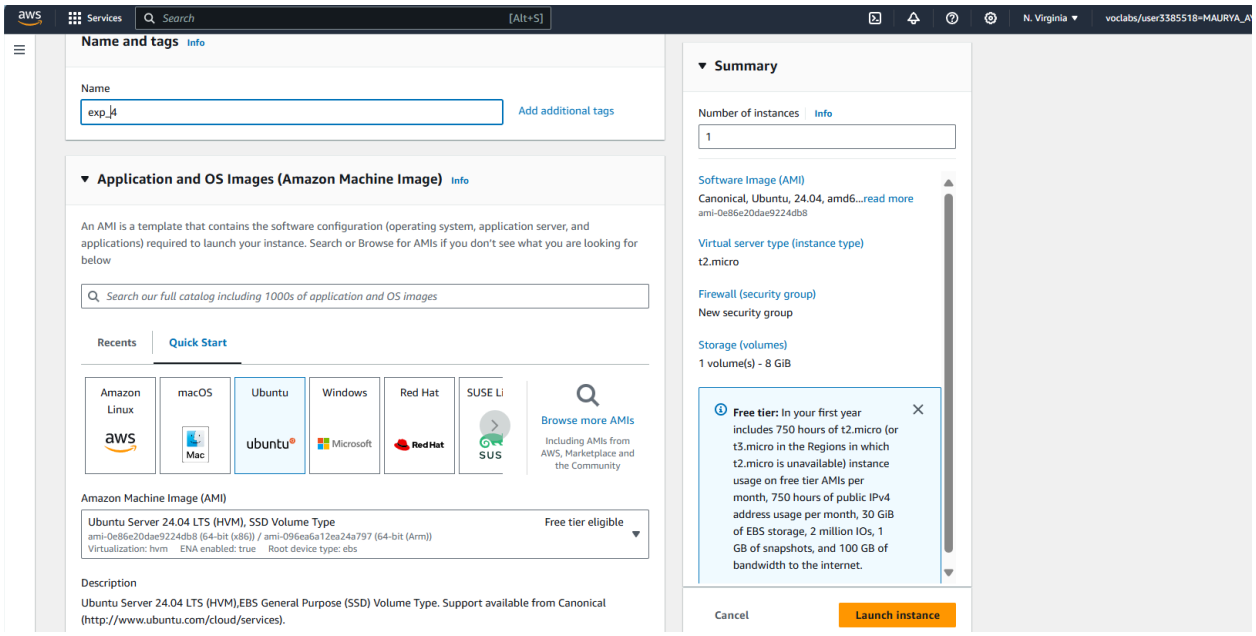
- Managing resource consumption by applications or teams
- Distributing application load evenly across the infrastructure
- Automatically load balancing requests across multiple instances of an application
- Monitoring resource usage to prevent applications from exceeding resource limits and automatically restarting them if needed
- Moving application instances between hosts when resources are low or if a host fails
- Automatically utilizing additional resources when new hosts are added to the cluster
- Facilitating canary deployments and rollbacks with ease

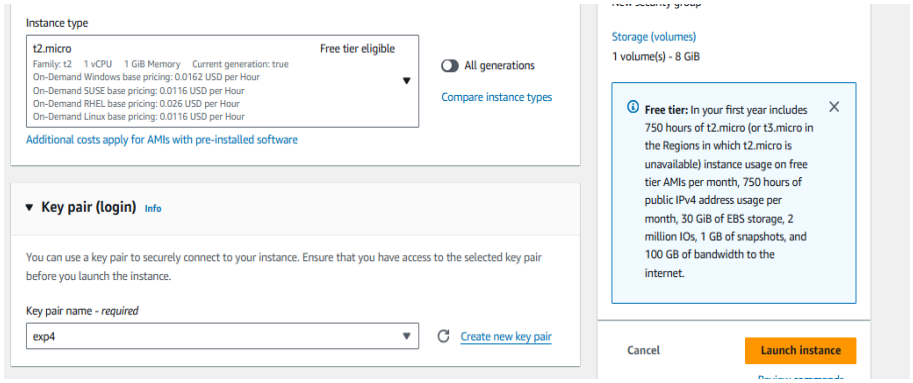
**Necessary Requirements:**
- EC2 Instance: The experiment required launching a t2.medium EC2 instance with 2 CPUs, as Kubernetes demands sufficient resources for effective functioning.

- **Minimum Requirements:**
  - Instance Type: t2.medium
  - CPUs: 2
  - Memory: Adequate for container orchestration.

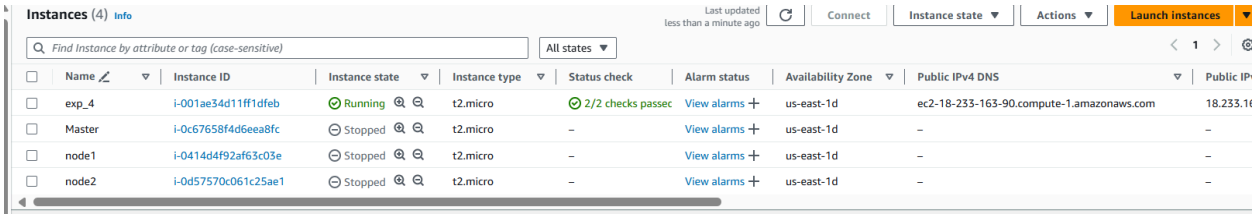
This ensured that the Kubernetes cluster had the necessary resources to function smoothly

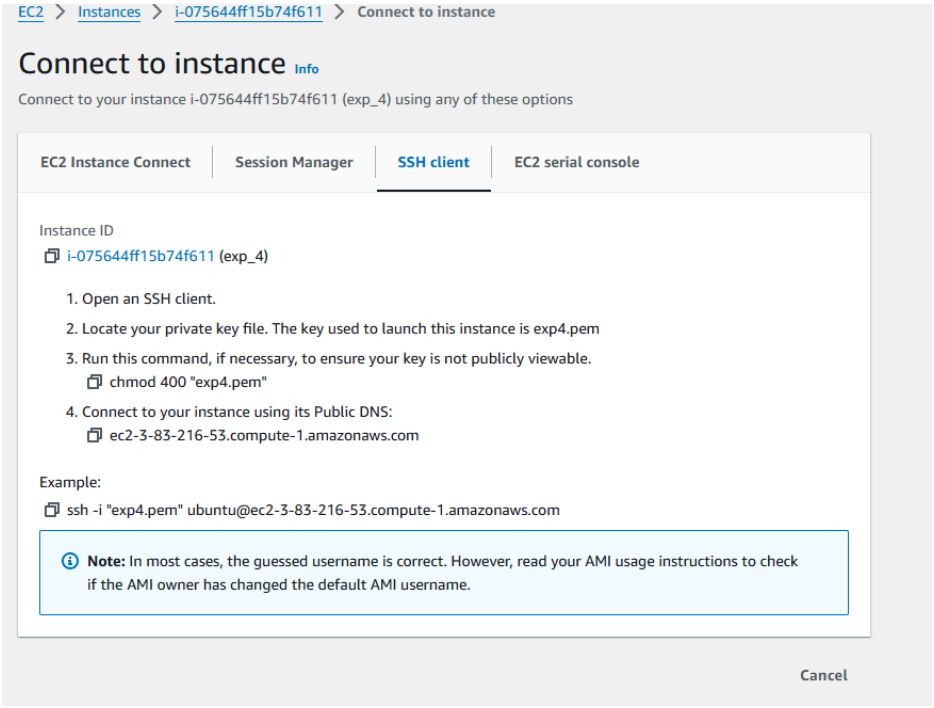
**Step 1:** Log in to your AWS Academy/personal account and launch 3 new Ec2 Instances. Select Ubuntu as AMI and t2.micro (because in academic account only t2.micro is present) as Instance Type and create a key of type RSA with .pem extension and move the downloaded key to the new folder.




Step 2: After creating the instance click on Connect the instance and navigate to SSH Client.





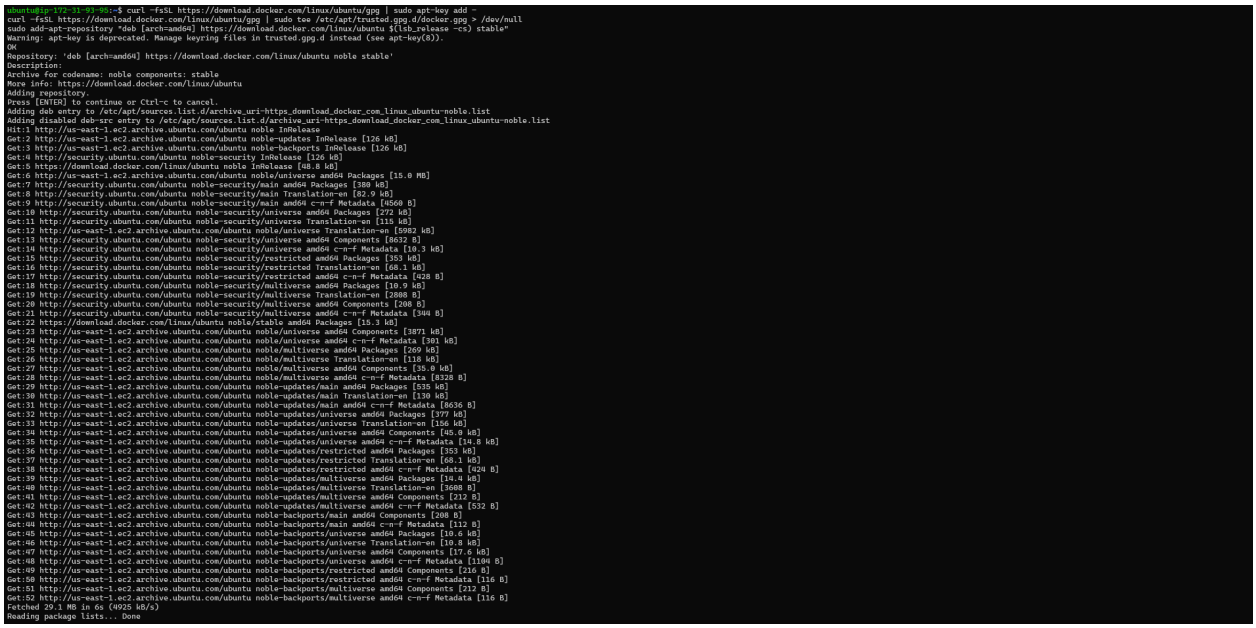

**Step 3:** Now open the folder in the terminal where our .pem key is stored and paste the Example command (starting with ssh -i .....) in the terminal.
ssh -i "<PATH TO FILE>exp3.pem" ubuntu@ec2-3-83-216-53.compute-1.amazonaws.com

```
C:\Users\Ayush Maurya>ssh -i "Downloads/exp4.pem" ubuntu@ec2-3-83-216-53.compute-1.amazonaws.com
The authenticity of host 'ec2-3-83-216-53.compute-1.amazonaws.com (3.83.216.53)' can't be established.
ED25519 key fingerprint is SHA256:jdAspK3Zoikd8XhOvy+g6Ea5WqRKlgbfnr5S66tYTRg.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-83-216-53.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04 LTS (GNU/Linux 6.8.0-1012-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Fri Sep 27 04:30:29 UTC 2024

  System load:  0.03              Processes:             105
  Usage of /:   22.8% of 6.71GB   Users logged in:       0
  Memory usage: 20%               IPv4 address for enX0: 172.31.93.95
```

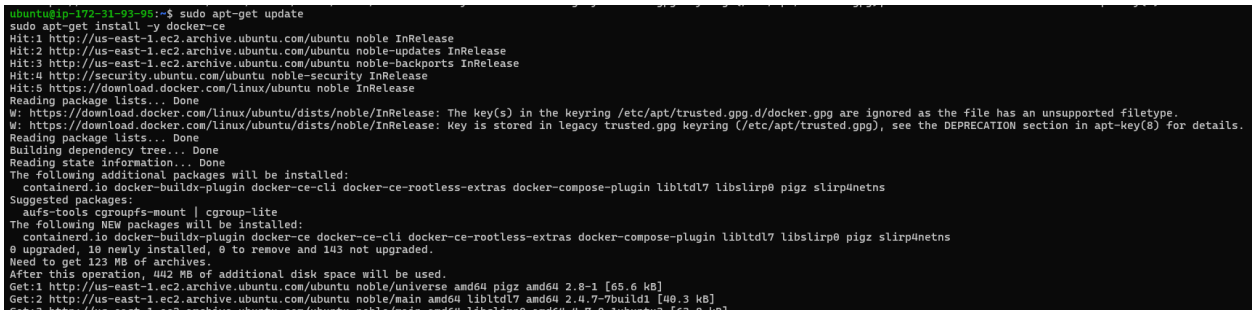
```
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-93-95:~$
```

**Step 4:** Run below commands to install and setup Docker

**curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -**
**curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo tee /etc/apt/trusted.gpg.d/docker.gpg > /dev/null**
**sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"**




**sudo apt-get update**
**sudo apt-get install -y docker-ce**





**sudo mkdir -p /etc/docker**
**cat <<EOF | sudo tee /etc/docker/daemon.json**
**{**

**"exec-opts": ["native.cgroupdriver=systemd"]**
**}**
**EOF**

```
ubuntu@ip-172-31-93-95:~$ sudo mkdir -p /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
EOF
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
ubuntu@ip-172-31-93-95:~$
```

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

```
ubuntu@ip-172-31-93-95:~$ sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
Synchronizing state of docker.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable docker
ubuntu@ip-172-31-93-95:~$ A
```

**Step 5:** Run the below command to install Kubernets.
**curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/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:/v1.31/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list**

```
ubuntu@ip-172-31-93-95:~$ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/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:/v1.31/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /
ubuntu@ip-172-31-93-95:~$
```

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

```
ubuntu@ip-172-31-93-95:~$ sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:5 https://download.docker.com/linux/ubuntu noble InRelease
Get:6 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb  InRelease [1186 B]
Get:7 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb  Packages [4865 B]
Fetched 6051 B in 1s (6278 B/s)
Reading package lists... Done
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: The key(s) in the keyring /etc/apt/trusted.gpg.d/docker.gpg are ignored as the file has an unsupporte
filetype.
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATION section in ap
key(8) for details.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

```
Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
ubuntu@ip-172-31-93-95:~$
```

**sudo systemctl enable --now kubelet**
**sudo apt-get install -y containerd**

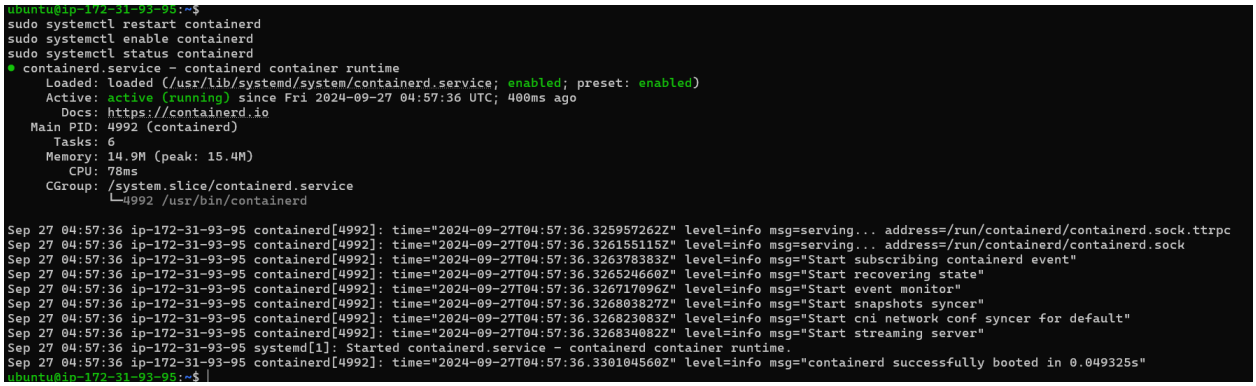
```
ubuntu@ip-172-31-93-95:~$ sudo systemctl enable --now kubelet
sudo apt-get install -y containerd
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7 libslirp0 pigz slirp4netns
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  runc
The following packages will be REMOVED:
  containerd.io docker-ce
The following NEW packages will be installed:
  containerd runc
0 upgraded, 2 newly installed, 2 to remove and 143 not upgraded.
Need to get 47.2 MB of archives.
After this operation, 53.1 MB disk space will be freed.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 runc amd64 1.1.12-0ubuntu3.1 [8599 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 containerd amd64 1.7.12-0ubuntu4.1 [38.6 MB]
Fetched 47.2 MB in 1s (47.0 MB/s)
(Reading database ... 68064 files and directories currently installed.)
Removing docker-ce (5:27.3.1-1~ubuntu.24.04~noble) ...
Removing containerd.io (1.7.22-1) ...
Selecting previously unselected package runc.
(Reading database ... 68044 files and directories currently installed.)
Preparing to unpack .../runc_1.1.12-0ubuntu3.1_amd64.deb ...
Unpacking runc (1.1.12-0ubuntu3.1) ...
```

```
Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-93-95:~$
```

**sudo mkdir -p /etc/containerd**
**sudo containerd config default | sudo tee /etc/containerd/config.toml**

```
ubuntu@ip-172-31-93-95:~$ sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml
disabled_plugins = []
imports = []
oom_score = 0
plugin_dir = ""
required_plugins = []
root = "/var/lib/containerd"
state = "/run/containerd"
temp = ""
version = 2

[cgroup]
  path = ""

[debug]
  address = ""
  format = ""
  gid = 0
  level = ""
  uid = 0

[grpc]
  address = "/run/containerd/containerd.sock"
  gid = 0
  max_recv_message_size = 16777216
  max_send_message_size = 16777216
  tcp_address = ""
  tcp_tls_ca = ""
  tcp_tls_cert = ""
  tcp_tls_key = ""
  uid = 0
```

**sudo systemctl restart containerd**
**sudo systemctl enable containerd**
**sudo systemctl status containerd**



```
ubuntu@ip-172-31-93-95:~$
sudo systemctl restart containerd
sudo systemctl enable containerd
sudo systemctl status containerd
● containerd.service - containerd container runtime
     Loaded: loaded (/usr/lib/systemd/system/containerd.service; enabled; preset: enabled)
     Active: active (running) since Fri 2024-09-27 04:57:36 UTC; 400ms ago
       Docs: https://containerd.io
   Main PID: 4992 (containerd)
      Tasks: 6
     Memory: 14.9M (peak: 15.4M)
        CPU: 78ms
     CGroup: /system.slice/containerd.service
             └─4992 /usr/bin/containerd

Sep 27 04:57:36 ip-172-31-93-95 containerd[4992]: time="2024-09-27T04:57:36.325957262Z" level=info msg=serving... address=/run/containerd/containerd.sock.ttrpc
Sep 27 04:57:36 ip-172-31-93-95 containerd[4992]: time="2024-09-27T04:57:36.326155115Z" level=info msg=serving... address=/run/containerd/containerd.sock
Sep 27 04:57:36 ip-172-31-93-95 containerd[4992]: time="2024-09-27T04:57:36.326378383Z" level=info msg="Start subscribing containerd event"
Sep 27 04:57:36 ip-172-31-93-95 containerd[4992]: time="2024-09-27T04:57:36.326524660Z" level=info msg="Start recovering state"
Sep 27 04:57:36 ip-172-31-93-95 containerd[4992]: time="2024-09-27T04:57:36.326717096Z" level=info msg="Start event monitor"
Sep 27 04:57:36 ip-172-31-93-95 containerd[4992]: time="2024-09-27T04:57:36.326803827Z" level=info msg="Start snapshots syncer"
Sep 27 04:57:36 ip-172-31-93-95 containerd[4992]: time="2024-09-27T04:57:36.326823083Z" level=info msg="Start cni network conf syncer for default"
Sep 27 04:57:36 ip-172-31-93-95 containerd[4992]: time="2024-09-27T04:57:36.326834082Z" level=info msg="Start streaming server"
Sep 27 04:57:36 ip-172-31-93-95 systemd[1]: Started containerd.service - containerd container runtime.
Sep 27 04:57:36 ip-172-31-93-95 containerd[4992]: time="2024-09-27T04:57:36.330104560Z" level=info msg="containerd successfully booted in 0.049325s"
ubuntu@ip-172-31-93-95:~$
```

**sudo apt-get install -y socat**

```
ubuntu@ip-172-31-93-95:~$ sudo apt-get install -y socat
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7 libslirp0 pigz slirp4netns
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  socat
0 upgraded, 1 newly installed, 0 to remove and 143 not upgraded.
Need to get 374 kB of archives.
After this operation, 1649 kB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 socat amd64 1.8.0.0-4build3 [374 kB]
Fetched 374 kB in 0s (11.9 MB/s)
Selecting previously unselected package socat.
(Reading database ... 68108 files and directories currently installed.)
Preparing to unpack .../socat_1.8.0.0-4build3_amd64.deb ...
Unpacking socat (1.8.0.0-4build3) ...
Setting up socat (1.8.0.0-4build3) ...
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-93-95:~$
```

**Step 6:** Initialize the Kubercluster. Now perform this on Master Instance.
**sudo kubeadm init --pod-network-cidr=10.244.0.0/16**

```
ubuntu@ip-172-31-93-95:~$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=NumCPU,Mem
[init] Using Kubernetes version: v1.31.0
[preflight] Running pre-flight checks
        [WARNING NumCPU]: the number of available CPUs 1 is less than the required 2
        [WARNING Mem]: the system RAM (957 MB) is less than the minimum 1700 MB
[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'
W0927 05:00:10.999529    5254 checks.go:846] detected that the sandbox image "registry.k8s.io/pause:3.8" of the container runtime is inconsistent with that used by kubead
m.It is recommended 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-172-31-93-95 kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs
 [10.96.0.1 172.31.93.95]
[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-172-31-93-95 localhost] and IPs [172.31.93.95 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
```

```
  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 172.31.93.95:6443 --token wuhiw8.tqn7cnmejhk5kqey \
        --discovery-token-ca-cert-hash sha256:f9a9d75f6d99fdd71aaf1b049f75e5ece76e902877e420624f4a305cf4125eb7
ubuntu@ip-172-31-93-95:~$
```

**TOKEN**

kubeadm join 172.31.93.95:6443 --token wuhiw8.tqn7cnmejhk5kqey \
--discovery-token-ca-cert-hash sha256:f9a9d75f6d99fdd71aaf1b049f75e5ece76e902877e420624f4a305cf4125eb7

From this command we get token and ca-

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

```
ubuntu@ip-172-31-93-95:~$ mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
ubuntu@ip-172-31-93-95:~$
```

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

```
ubuntu@ip-172-31-93-95:~$ 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
ubuntu@ip-172-31-93-95:~$
```

**Step 7:** Now that the cluster is up and running, we can deploy our nginx server on this cluster.Apply this deployment file using this command to create a deployment
**kubectl apply -f https://k8s.io/examples/application/deployment.yaml**

```
ubuntu@ip-172-31-93-95:~$ kubectl apply -f https://k8s.io/examples/application/deployment.yaml
deployment.apps/nginx-deployment created
ubuntu@ip-172-31-93-95:~$
```

**kubectl get pods**

```
ubuntu@ip-172-31-93-95:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-d556bf558-wpb7n    0/1     Pending   0          30s
nginx-deployment-d556bf558-wvkzl    0/1     Pending   0          30s
ubuntu@ip-172-31-93-95:~$
```

**POD_NAME=$(kubectl get pods -l app=nginx -o jsonpath="{.items[0].metadata.name}")**
**kubectl port-forward $POD_NAME 8080:80**

```
ubuntu@ip-172-31-93-95:~$ POD_NAME=$(kubectl get pods -l app=nginx -o jsonpath="{.items[0].metadata.name}")
kubectl port-forward $POD_NAME 8080:80
error: unable to forward port because pod is not running. Current status=Pending
ubuntu@ip-172-31-93-95:~$ POD_NAME=$(kubectl get pods -l app=nginx -o jsonpath="{.items[0].metadata.name}")
ubuntu@ip-172-31-93-95:~$ kubectl port-forward $POD_NAME 8080:80
error: unable to forward port because pod is not running. Current status=Pending
ubuntu@ip-172-31-93-95:~$
```

**Note :** We have faced an error as pod status is pending so make it running run below commands
then again run above 2 commands.

**kubectl taint nodes --all node-role.kubernetes.io/control-plane-node/ip-172-31-20-171 untainted**
**kubectl get nodes**

```
ubuntu@ip-172-31-93-95:~$ kubectl taint nodes --all node-role.kubernetes.io/control-plane-node/ip-172-31-20-171 untainted
kubectl get nodes
error: at least one taint update is required
NAME              STATUS   ROLES           AGE   VERSION
ip-172-31-93-95   Ready    control-plane   20m   v1.31.1
ubuntu@ip-172-31-93-95:~$
```

**kubectl get pods**

```
ubuntu@ip-172-31-93-95:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-d556bf558-wpb7n    0/1     Pending   0          19m
nginx-deployment-d556bf558-wvkzl    0/1     Pending   0          19m
ubuntu@ip-172-31-93-95:~$
```

**POD_NAME=$(kubectl get pods -l app=nginx -o jsonpath="{.items[0].metadata.name}")**
**kubectl port-forward $POD_NAME 8080:80**

```
ubuntu@ip-172-31-93-95:~$ POD_NAME=$(kubectl get pods -l app=nginx -o jsonpath="{.items[0].metadata.name}")
kubectl port-forward $POD_NAME 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```

**Step 8:** Verify your deployment
Open up a new terminal and ssh to your EC2 instance.
Then, use this curl command to check if the Nginx server is running.
**curl --head http://127.0.0.1:8080**

```
ubuntu@ip-172-31-93-95:~$ curl --head http://127.0.0.1:8080
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Fri, 27 Sep 2024 06:06:41 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 04 Dec 2018 14:44:49 GMT
Connection: keep-alive
ETag: "5c0692e1-264"
Accept-Ranges: bytes

ubuntu@ip-172-31-93-95:~$
```

If the response is 200 OK and you can see the Nginx server name, your deployment was successful.
We have successfully deployed our Nginx server on our EC2 instance.

**Conclusion:** We successfully set up a Kubernetes cluster on AWS EC2, addressing issues related to component setup and residual configurations. We ensured proper cleanup of previous Kubernetes files and mounts, verified the kubelet service, and applied Flannel for networking. Finally, we resolved connectivity issues, and after a thorough review of logs and configuration, we deployed and exposed an NGINX server using Kubernetes services, preparing the cluster for efficient traffic management and scaling.