**Kubernetes Assignment**

1. **Kubernetes Installation and Worker Node Join**

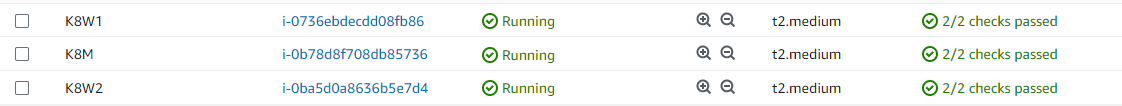
Before installing and setting up Kubernetes, we need to have below pre-requisites.

Prerequisites:

* Launch 3 EC2 Instance with at least t2.medium configuration, one EC2 will be for Master (Control Plane) and other two will be for Worker Nodes.

Pre-requisites are done as below:

* Launching 3 EC2 Instance with at least t2.medium configuration, one EC2 will be for Master (Control Plane) and other two will be for Worker Nodes.



Installation Steps are divided into two parts

* Setting up containerd
* Installation steps for Kubernetes.

Installation Steps are started as below:

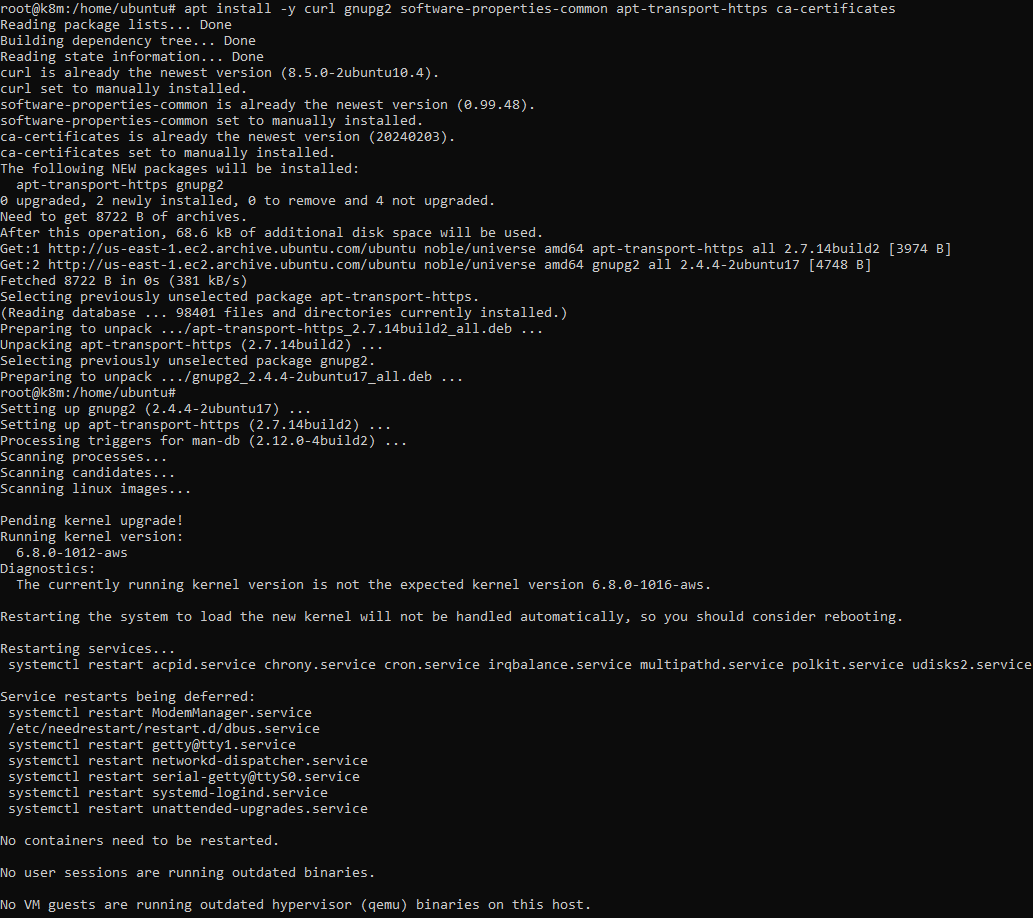
* **Setting up containerd on Master (K8M) EC2 instance**

swapoff -a (This command is used to disable all swap space on the system)



Swap space is an area on the disk that is used when the system's RAM is fully utilized. By turning off swap, you're instructing the system to stop using swap and rely solely on physical RAM.

apt install -y curl gnupg2 software-properties-common apt-transport-https ca-certificates (Command to install gnu package and transport https)



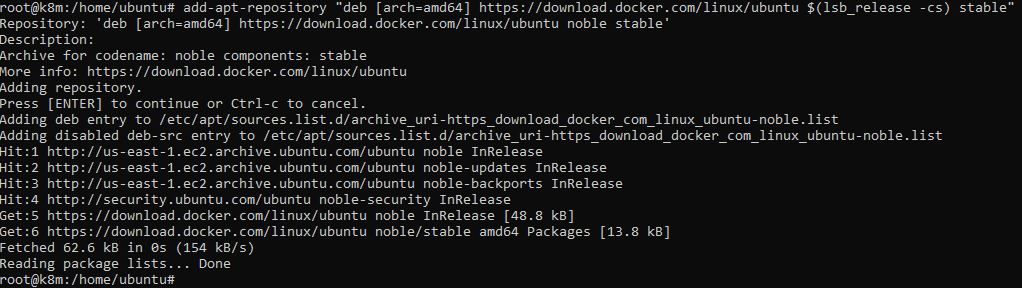
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmour -o /etc/apt/trusted.gpg.d/docker.gpg (Command sequence is used to add Docker's official GPG key to your system's list of trusted keys.)



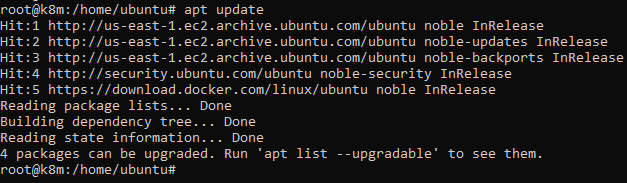
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

(Command adds Docker's APT repository to your system)

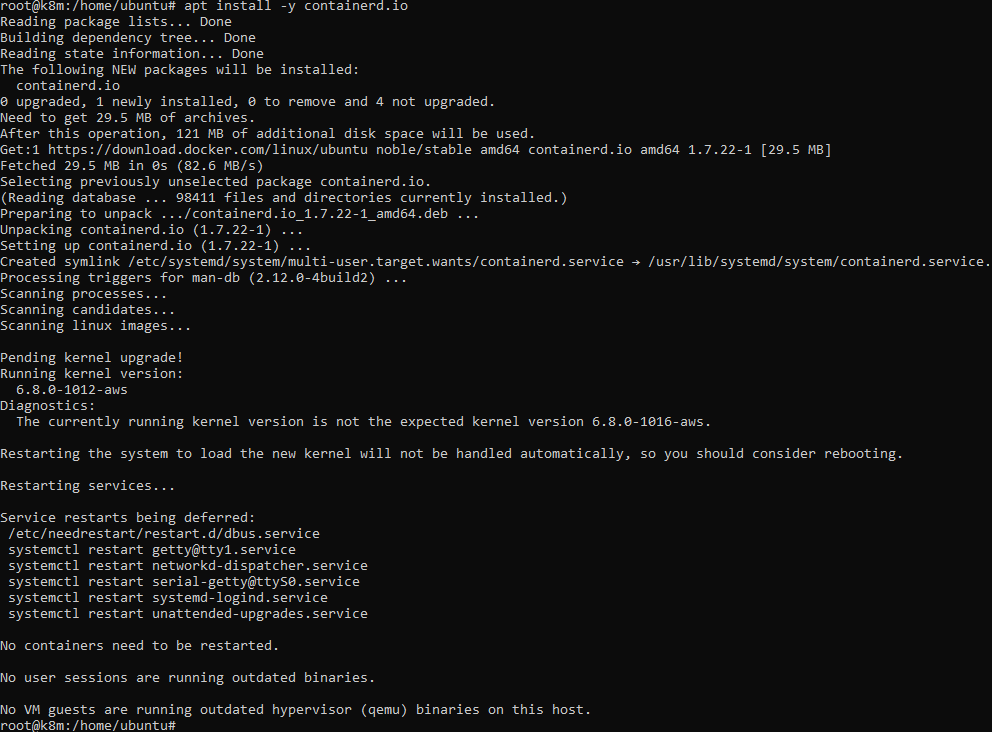
Press Enter to Add it with user input



apt update (command updates the local package database)

****

apt install -y containerd.io(Installs the containerd package)

****

containerd config default | sudo tee /etc/containerd/config.toml >/dev/null 2>&1 (Command is used to generate and save a default configuration file for containerd.)

****

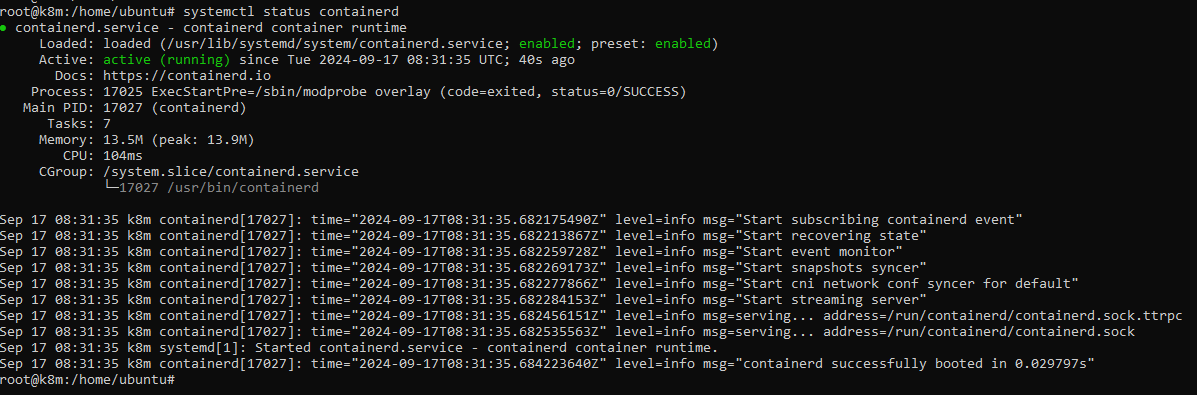
sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' /etc/containerd/config.toml (Command is used to modify the containerd configuration file.)



systemctl restart containerd (Command restarts the containerd service)



systemctl status containerd (Command checks the status of containerd service)



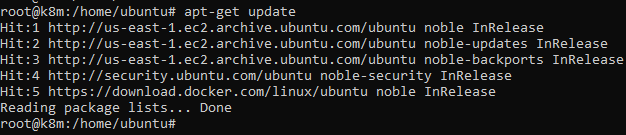
* **Installation steps for setting up Kubernetes Control plane (K8M).**

We need to visit the below website to get the installation steps for latest version and older version also. We are installing the latest version i.e. v1.131

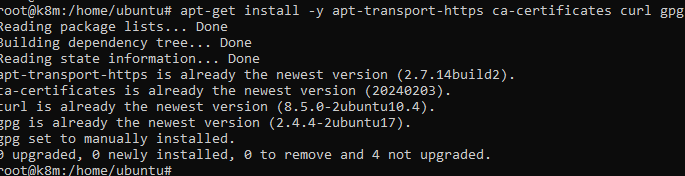
[**https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/**](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/)

**Installation Steps on Control Plane (K8M) are as below:**

apt-get update (Command updates the local package database)

****

apt-get install -y apt-transport-https ca-certificates curl gpg (Already installed before installing containerId)



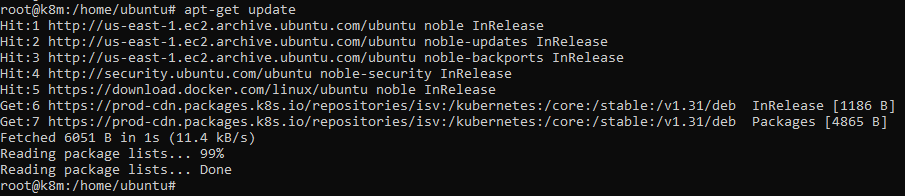
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg (Command fetches the key for the repository)



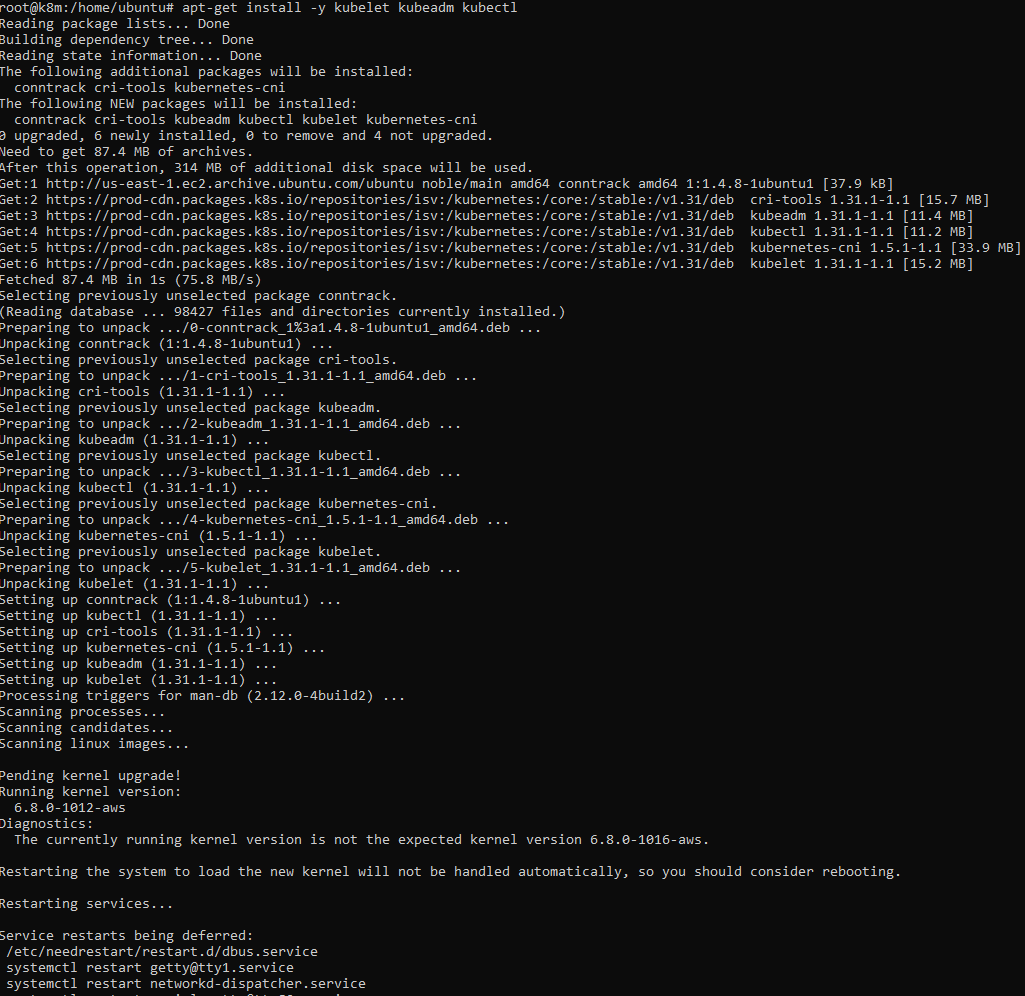
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list (Command overwrites any existing configuration in /etc/apt/sources.list.d/kubernetes.list)



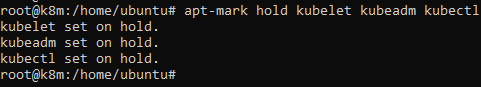
apt-get update (Updates the package index)

****

apt-get install -y kubelet kubeadm kubectl (Command to install kubelet, kubeadm and kubectl)



apt-mark hold kubelet kubeadm kubectl (Command prevents a specific package from being updated due to compatibility issues or other reasons. It ensures that the package remains at its current version until explicitly unheld.)

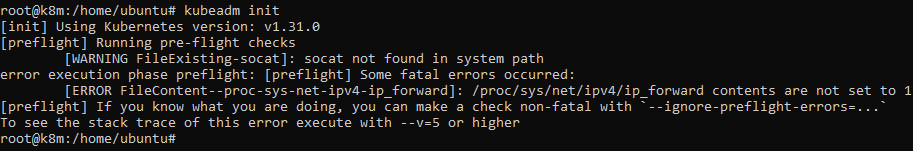


systemctl enable --now kubelet(Enable the kubelet service before running kubeadm)

****

kubeadm init (Command Initializes the Kubernetes setup)

It gives the below error



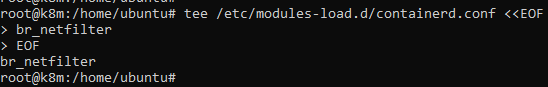
We need to perform the below troubleshooting to initialize the kubeadm by freeing up ram and reinitialise iptables settings

* Configure the Kernel Module ‘br\_netfilter’ in the containerd configuration file.

tee /etc/modules-load.d/containerd.conf <<EOF

br\_netfilter

EOF



* Load the br\_netfilter modules into the running Linux kernel.

modprobe br\_netfilter



* Update Iptables Settings.

**Note:** To ensure packets are properly processed by IP tables during filtering and port forwarding, set the **net.bridge.bridge-nf-call-iptables to ‘1’** in your sysctl configuration file. Otherwise, you may encounter the following error: **[ERROR FileContent–proc-sys-net-ipv4-ip\_forward]: /proc/sys/net/ipv4/ip\_forward contents are not set to 1.** To avoid this, execute the following command.

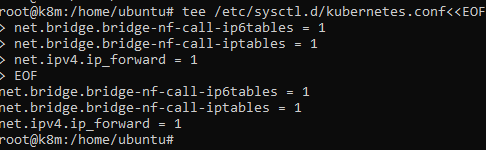
tee /etc/sysctl.d/kubernetes.conf<<EOF

net.bridge.bridge-nf-call-ip6tables = 1

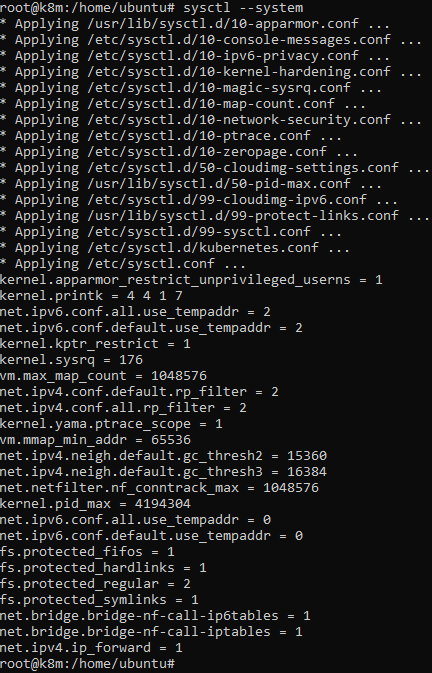
net.bridge.bridge-nf-call-iptables = 1

net.ipv4.ip\_forward = 1

EOF

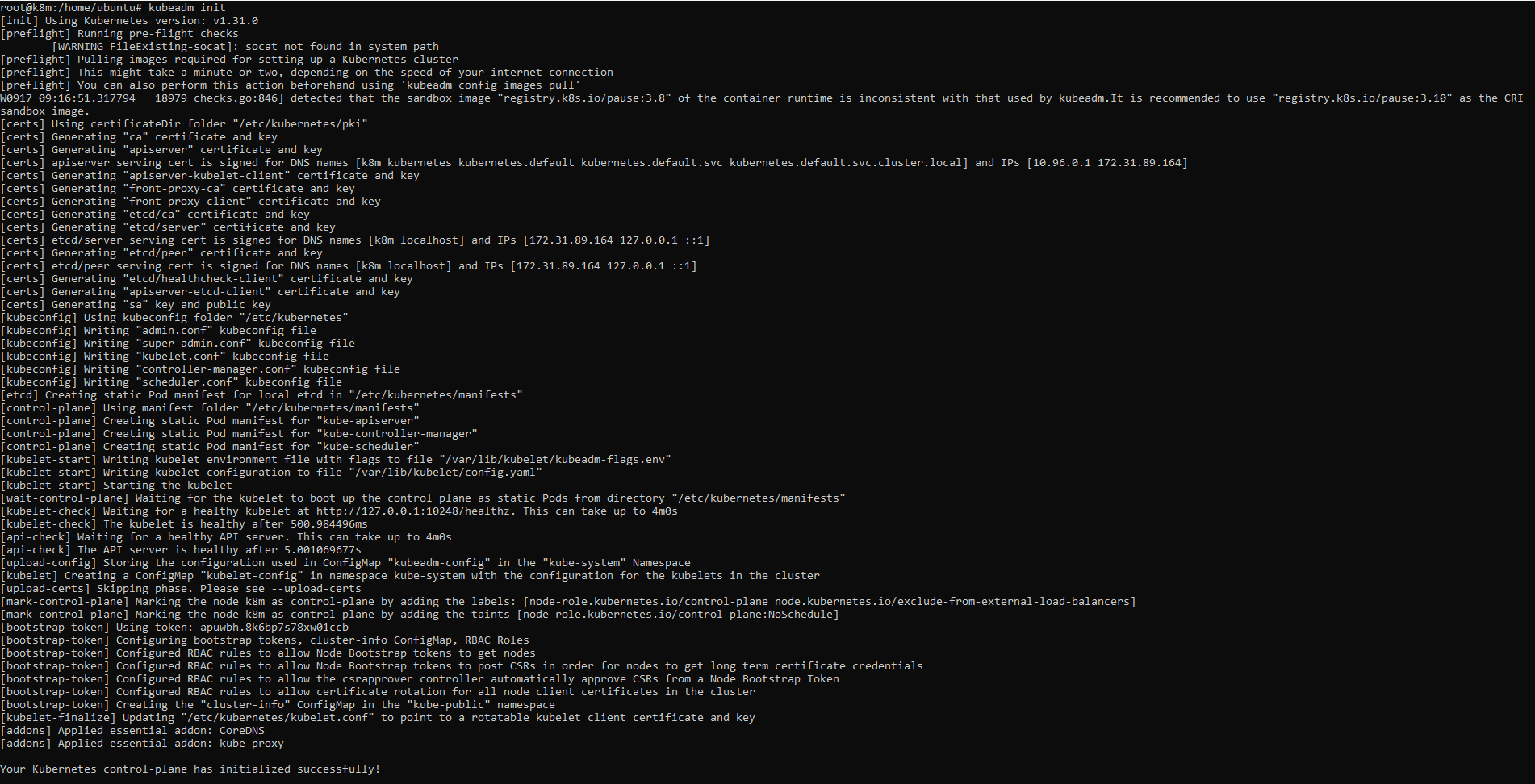


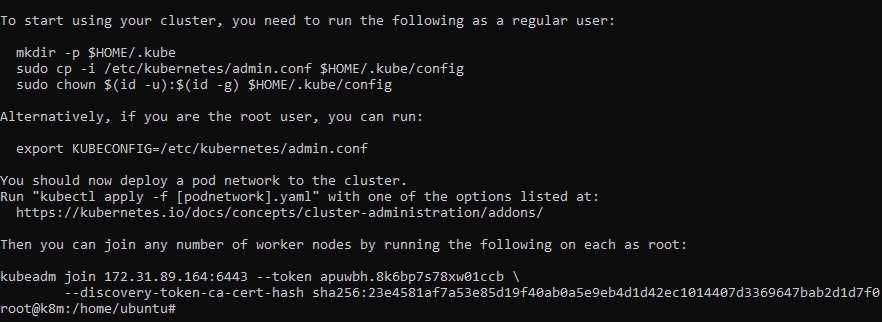
sysctl --system (Command applies kernel settings without reboot)



Once you’ve verified and potentially adjusted the configuration, proceed with reinitializing the Kubernetes cluster. we initialized the kubeadm again and installation was successful.

kubeadm init (Command Initializes the Kubernetes setup)





We can also see the above screenshot that the token has been generated to join the Worker Nodes to control plane. We need to run the command on both the worker nodes.

Since Kubernetes control-plane has initialized successfully, we need to run the below commands to start using the cluster.

mkdir -p $HOME/.kube



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

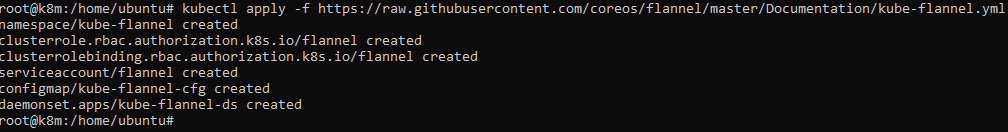


sudo chown $(id -u):$(id -g) $HOME/.kube/config



We also need to ready our cluster, for that we need to assign a flannel network using the kube-flannel.yml file by running the below command.

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



Now we need to setup password less SSH between Worker Node and Control Plane.

* Command runs SSH-Keygen without prompting anything

echo -e "\n" | ssh-keygen -N "" &> /dev/null



* Command to check whether the pub file is created in the below location along with the required contents.

cat /root/.ssh/id\_ed25519.pub

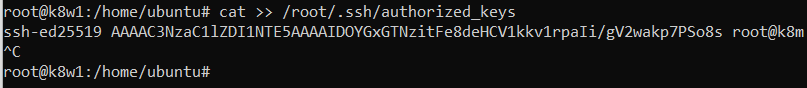


Its shows that the public is created.

* Now copy the contents over the Worker Node 1 and Worker Node 2

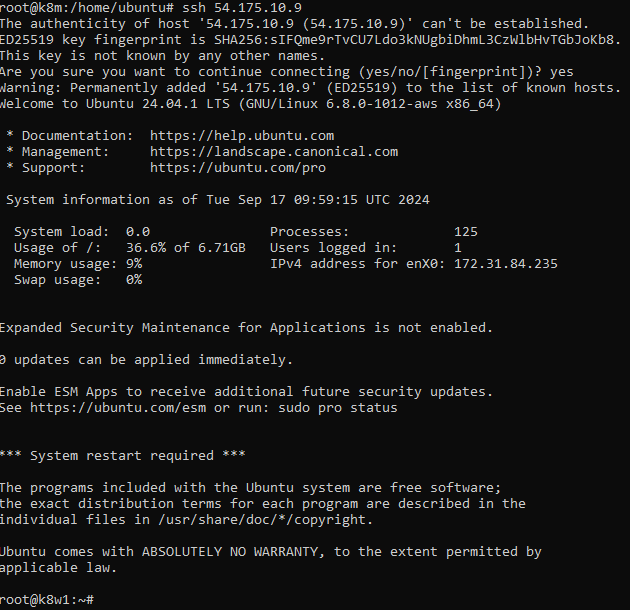
Worker Node 1

cat >> /root/.ssh/authorized\_keys



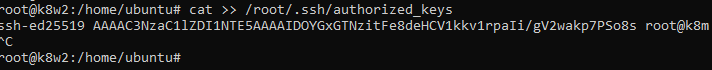
Verify SSH from Control Plane

ssh Public IP of Worker 1



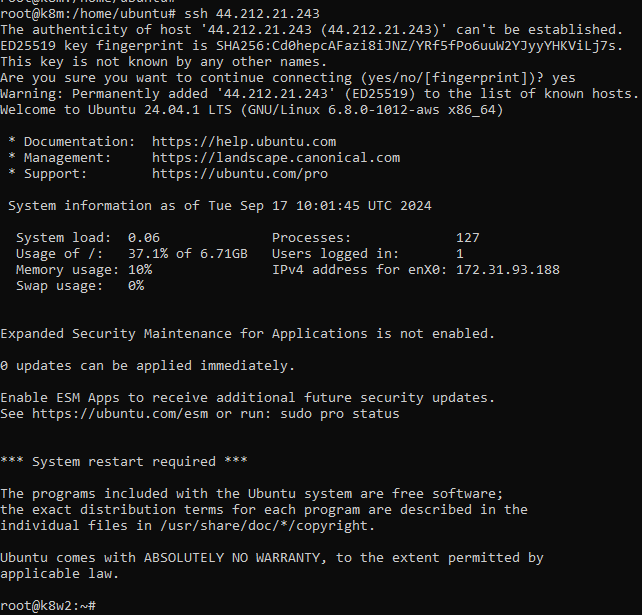
Worker Node 2

cat >> /root/.ssh/authorized\_keys



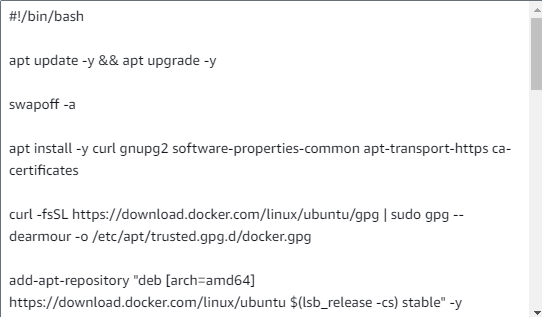
Verify SSH from Control Plane

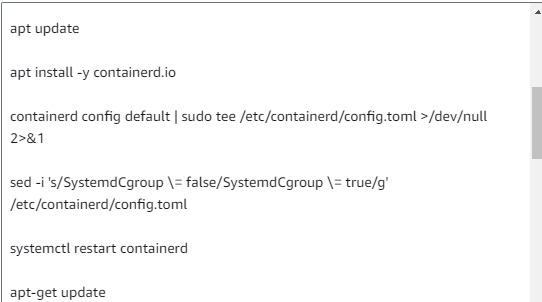
ssh Public IP of Worker 2

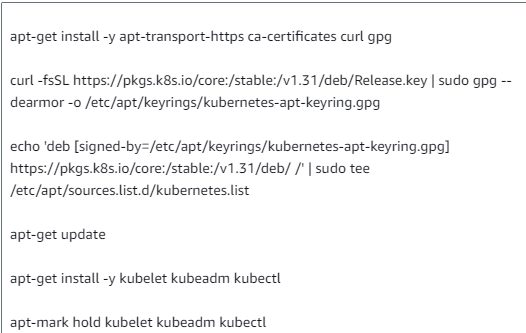


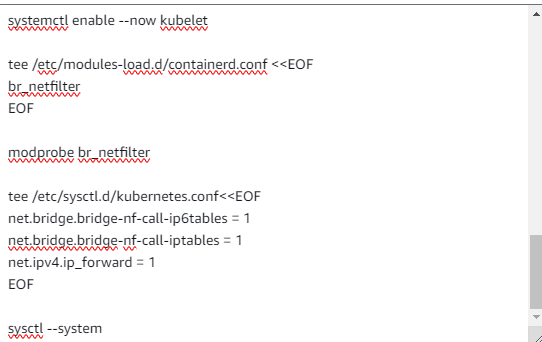
Installation Steps on Worker Node 1 and Worker 2 are as below:

**On Worker Node 1 and Worker Node 2, we have performed the installation of containerd and Kubernetes setup before “kubeadm init” step by putting all the steps performed above in the user data of EC2 instance.**

****

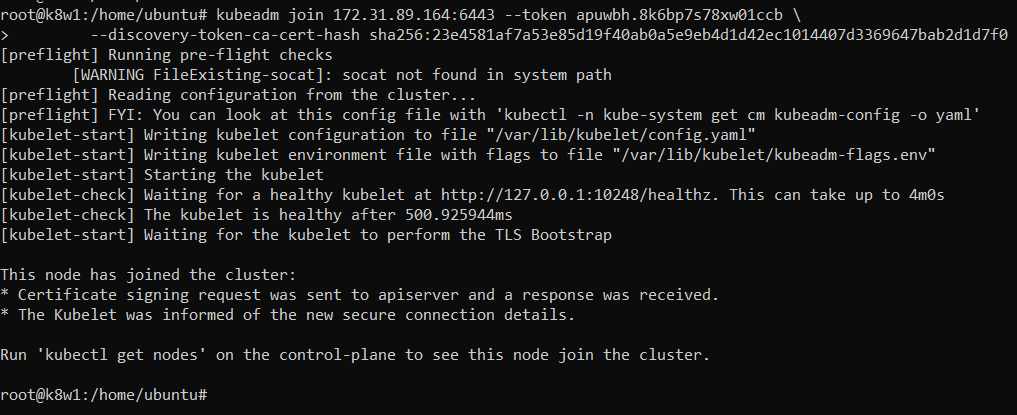
****

****

****

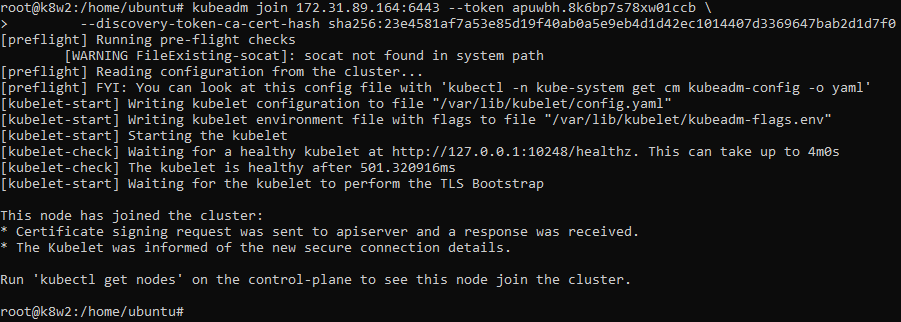
**Now we will perform the step of joining the worker nodes to cluster using the token from Control plane (K8M)**

**Worker Node 1**

****

Successfully joined the Cluster.

**Worker Node 2**

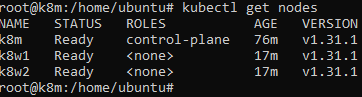
****

Successfully joined the Cluster.

Verify the Nodes are there.

On Control Plane, run the below command

kubectl get nodes

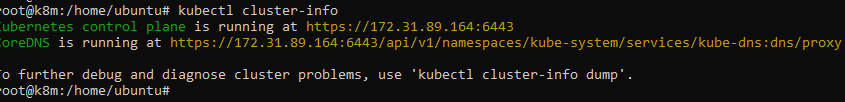


**Kubernetes setup with worker node joined to cluster is completed.**

**2) Kubernetes Commands**

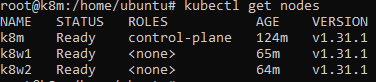
* **View Cluster Info**

kubectl cluster-info

****

* **List All Nodes**

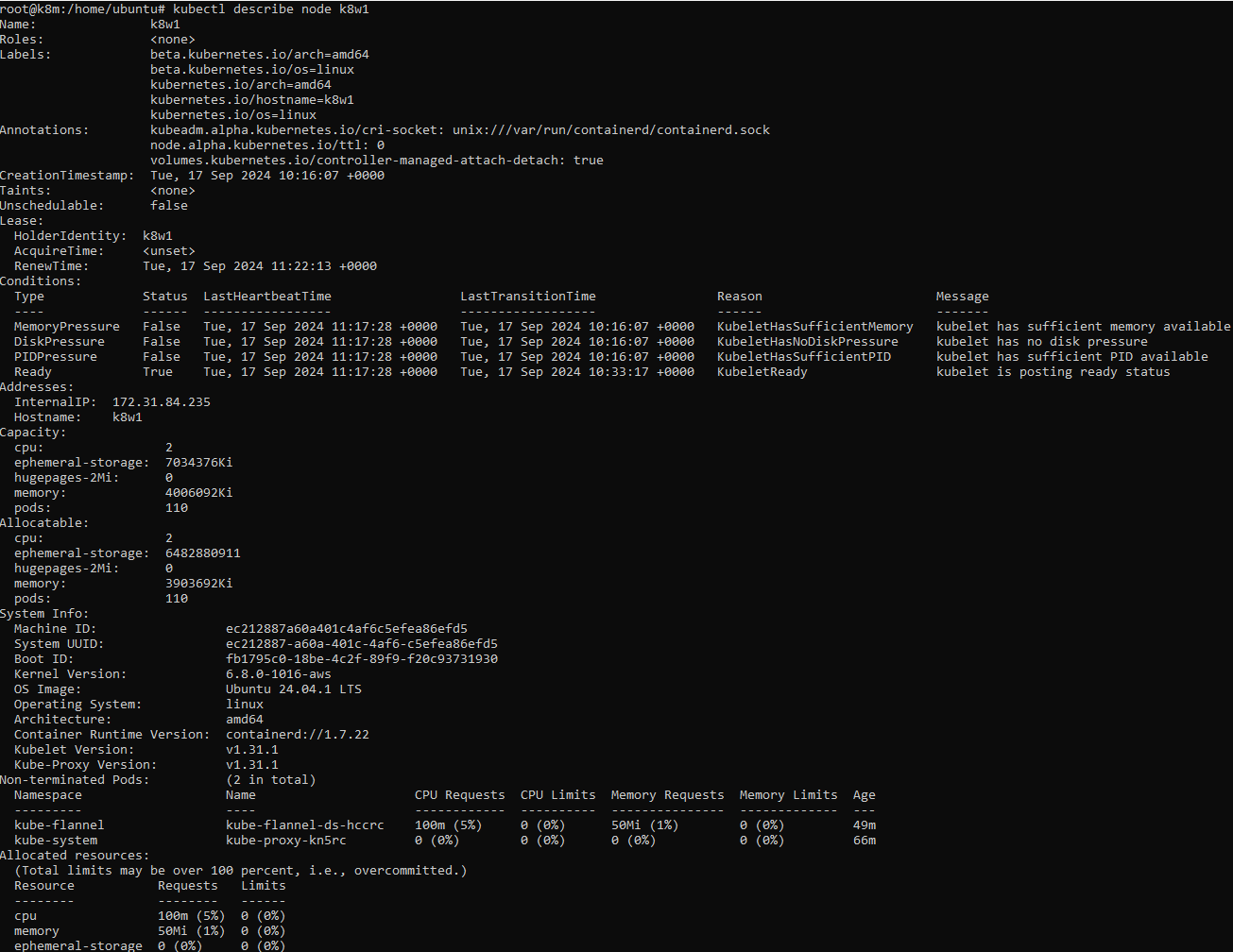
kubectl get nodes

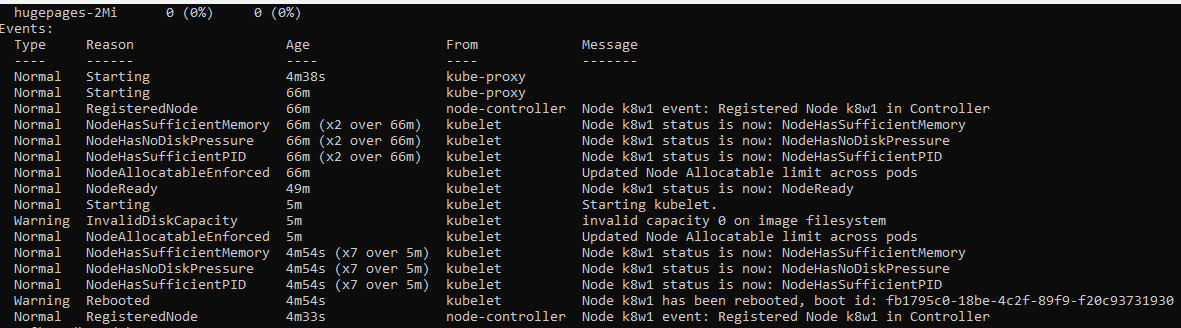
****

* **Describe a Node**

kubectl describe node <node-name>

kubectl describe node k8w1

****

****

* **Get Pod Information**

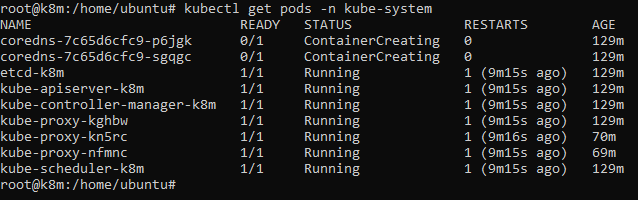
kubectl get pods

****

* **Get Pods in a Specific Namespace**

kubectl get pods -n <namespace>

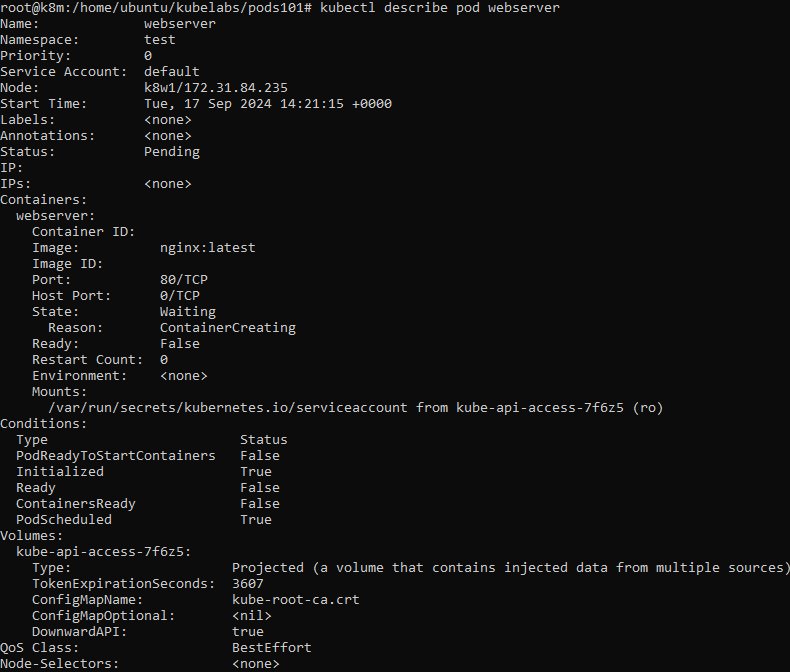
kubectl get pods -n kube-system

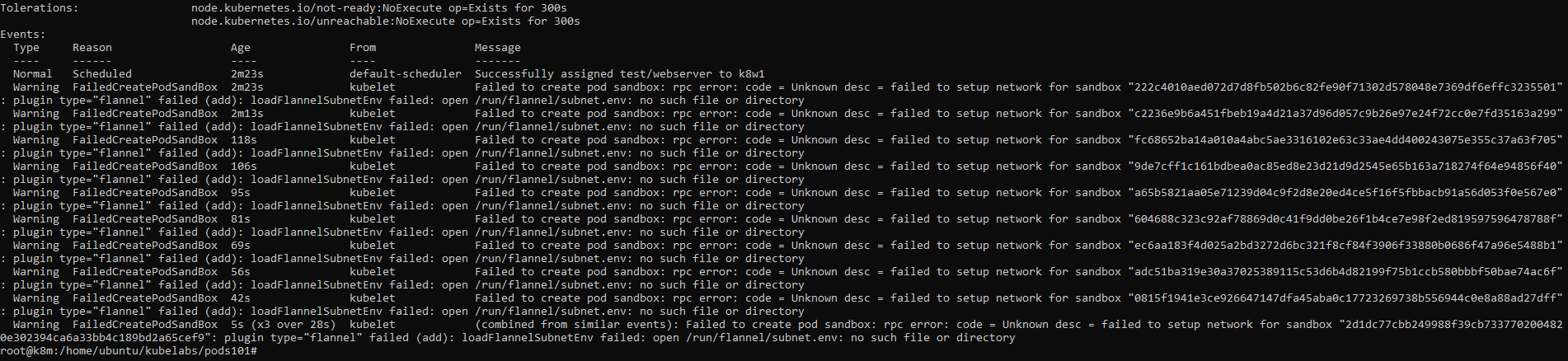
****

* **Describe a Pod**

kubectl describe pod <pod-name>

kubectl describe pod webserver





* **View Pod Logs**

kubectl logs <pod-name>

kubectl logs webserver



* **Create Resources from a YAML File**

kubectl apply -f <file.yaml>

kubectl apply -f pods01.yaml



* **Delete Resources from a YAML File**

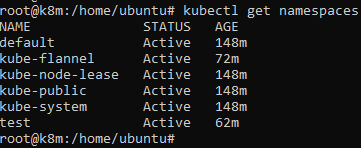
kubectl delete -f <file.yaml>

kubectl delete -f pods01.yaml

****

* **Get Namespaces**

kubectl get namespaces



* **Create a Namespace**

kubectl create namespace <namespace-name>

kubectl create namespace sourabh

****

* **Delete a Namespace**

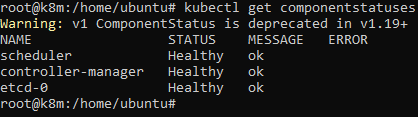
**kubectl delete namespace <namespace-name>**

kubectl delete namespace sourabh

****

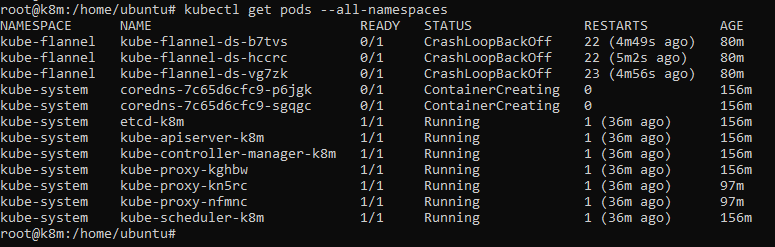
* **Check Cluster Health**

kubectl get componentstatuses

****

* **Display pod information for all namespaces**

kubectl get pods –all-namespaces

****

**3) Creating a Namespace and changing default namespace to your created namespace.**

* Create the yml file for creating the namespace.

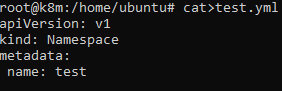
**cat>test.yml**

apiVersion: v1

kind: Namespace

metadata:

name: test



* Now we will create the name space using the below command

kubectl apply -f test.yml (we can also use create in place of apply, its same as run and pull in docker)



* Switch Namespaces (Changing the default namespace to our created namespace test) by using the below command.

kubectl config set-context --current --namespace=test



We can see that modification are done successfully.

* To check the current namespace, we will run the below command

kubectl config view --minify | grep namespace



* Verify that default namespace has changed to our namespace.

kubectl get pods



* Namespace has been created and assigned as default namespace.

**4) Create a namespace and allocate resources to that namespace.**

* Create the yaml file for creating the namespace.

**cat>new.yaml**

apiVersion: v1

kind: Namespace

metadata:

name: new

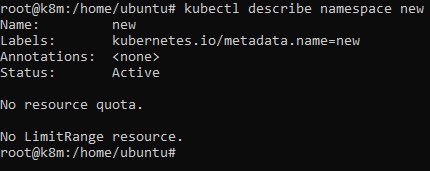
* Creating the namespace using the below command.

kubectl apply -f new.yaml



* Now we will verify if there is any quota or limit assigned wrt to resources on the created namespace “new”

kubectl describe namespace new



Nothing is assigned as of now

* Defining resources to the existing created namespace “new”.

cat > new\_res.yaml

apiVersion: v1

kind: ResourceQuota

metadata:

name: mem-cpu

namespace: new

spec:

hard:

requests.cpu: "1"

requests.memory: 1Gi

limits.cpu: "2"

limits.memory: 2Gi

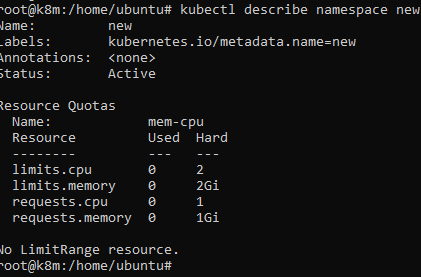
* Creating the resource quota and applying it using the below command.

kubectl apply -f new\_res.yaml



* Verify whether the quota and limits are assigned to the respected namespace “new”

kubectl describe namespace new



* The resource quota and limits have been assigned successfully.

**5) Install an old version of Kubernetes and upgrade it to latest version.**