Deploying PHP Guestbook application with Redis(kubeadm)

Building and deploying a simple, multi-tier web application using kubeadm cluster and containerd which consist of the below components.

1.A single-instance Redis to store guestbook entries

2.Multiple web frontend instances

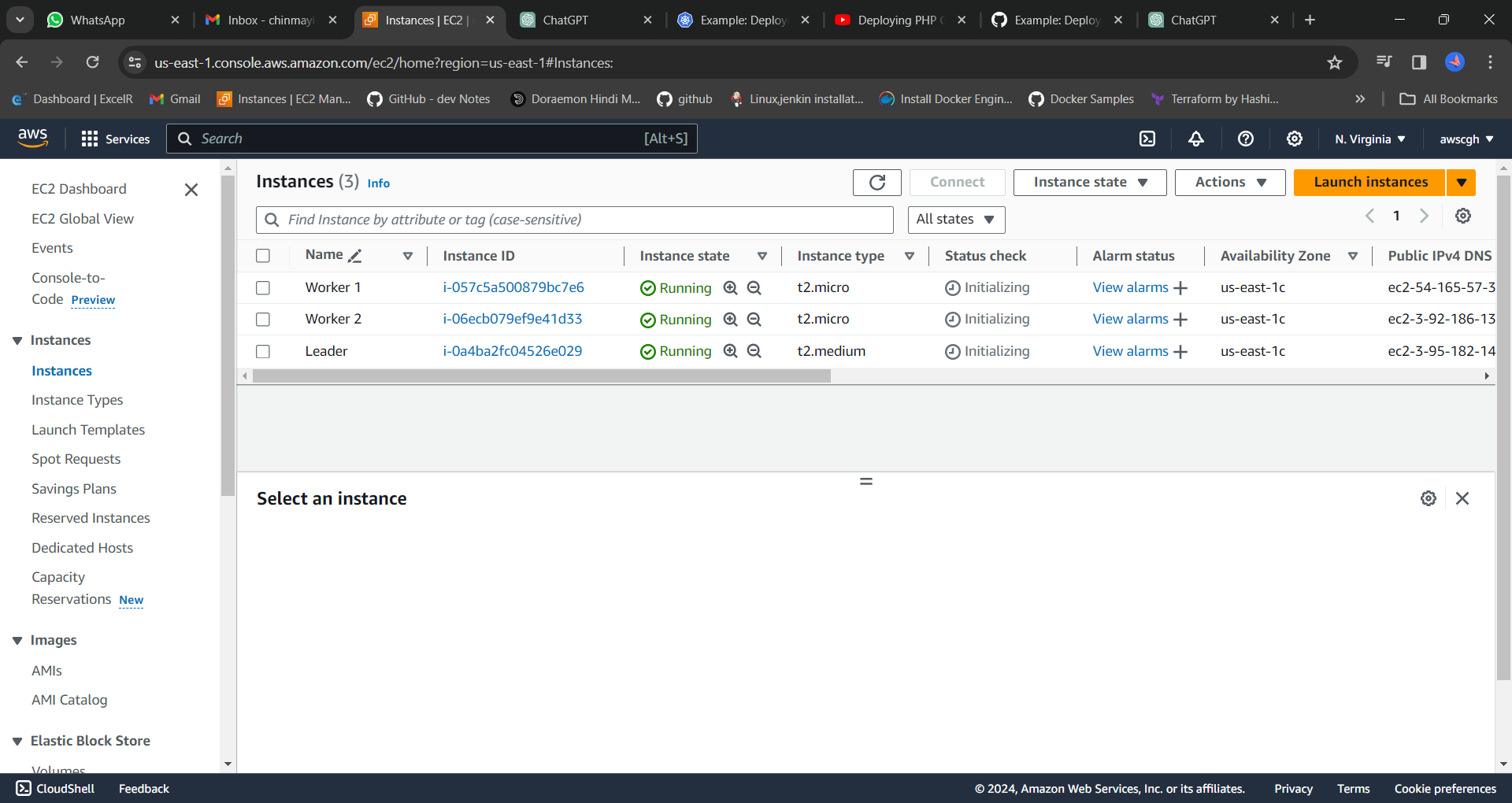
1.Start up a Redis leader.

2.Start up two Redis followers.

3.Start up the guestbook frontend.

4.Expose and view the Frontend Service (Use Kubernetes NodePort Service)

Create 3 Ubuntu instances (22.04) 1 t2.medium (master) , 2 for t2.micro (worker nodes) with all traffic allowed in default vpc



Master:

=======

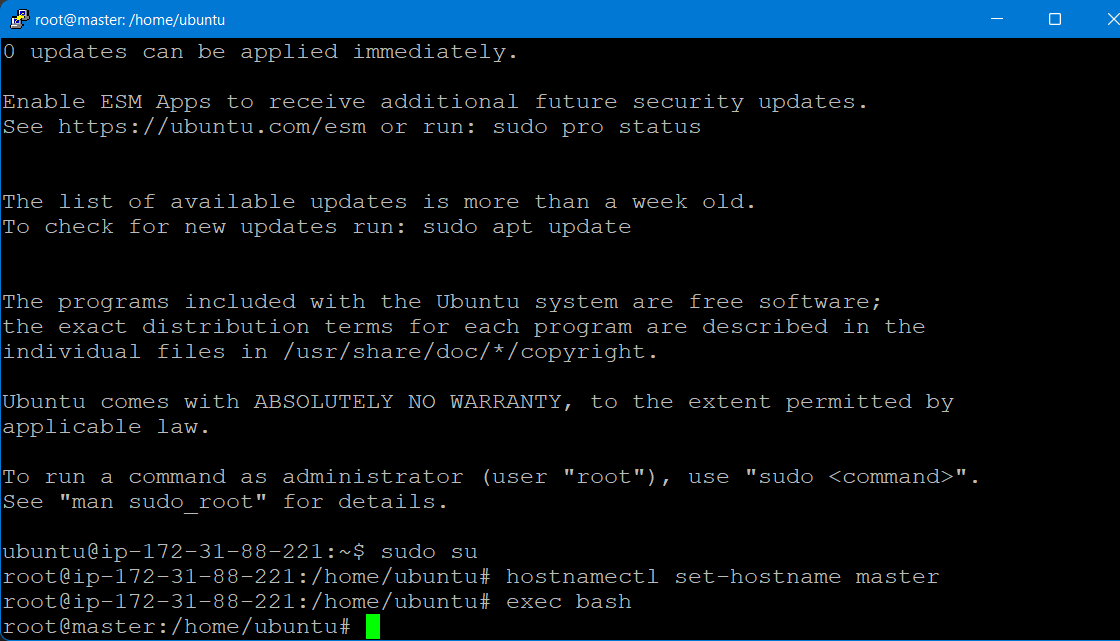
1.Enter the username as ubuntu

2.Switch to root

sudo su -

3.Set the hostname

# hostnamectl set-hostname master

# exec bash

4.Append the below details in /etc/hosts

vi /etc/hosts

Privateips master

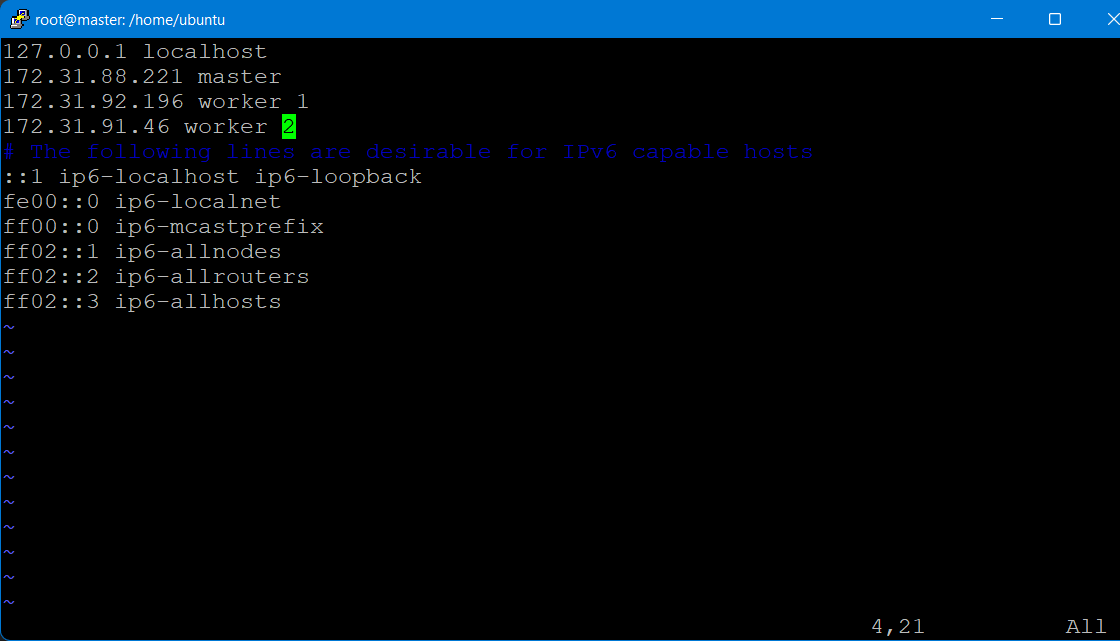
Privateips worker1

Privateip worker2

172.31.88.221 master

172.31.92.196 worker 1

172.31.91.46 worker 2

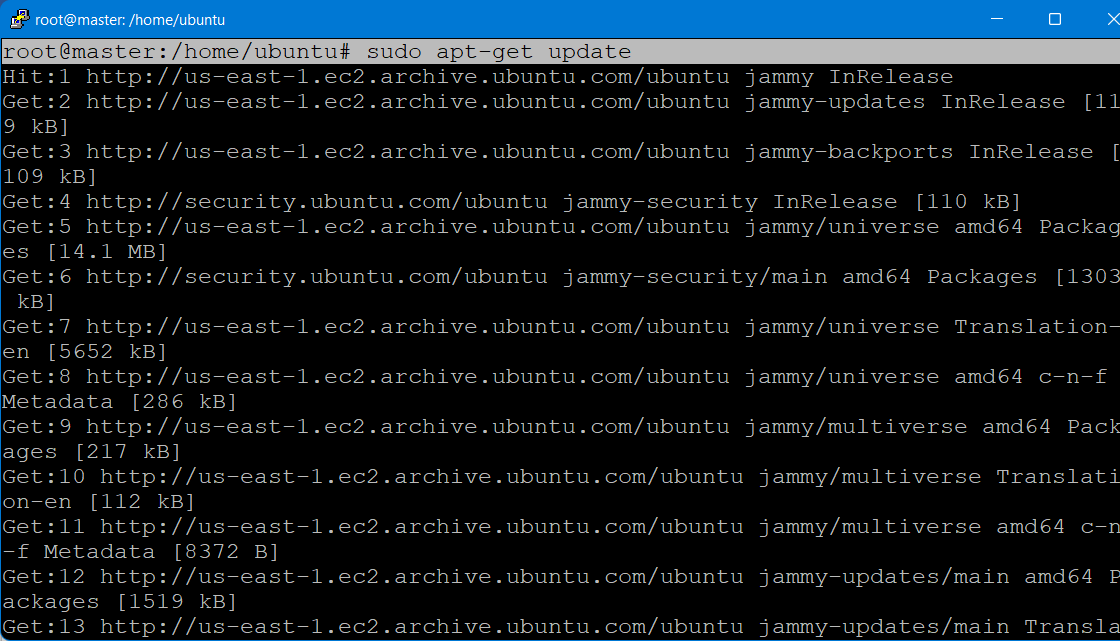


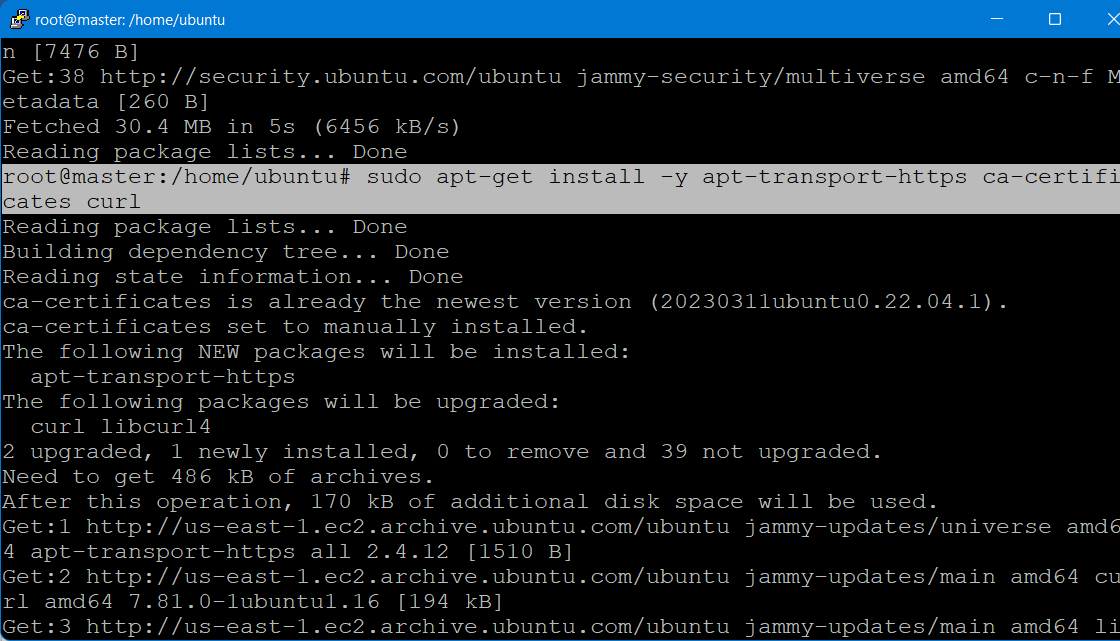
4.Install general dependencies

we need to install packages on our system for the commands we will use later.

sudo apt-get update

sudo apt-get install -y apt-transport-https ca-certificates curl





5.Install containerd

Although we have a few container runtimes to choose from, we’re going with containerd. Before we install containerd, we’ll create its configuration file.

curl -fsSLo containerd-config.toml \

<https://gist.githubusercontent.com/oradwell/31ef858de3ca43addef68ff971f459c2/raw/5099df007eb717a11825c3890a0517892fa12dbf/containerd-config.toml>

# Extract the binaries

sudo tar Cxzvf /usr/local containerd-1.6.14-linux-amd64.tar.gz

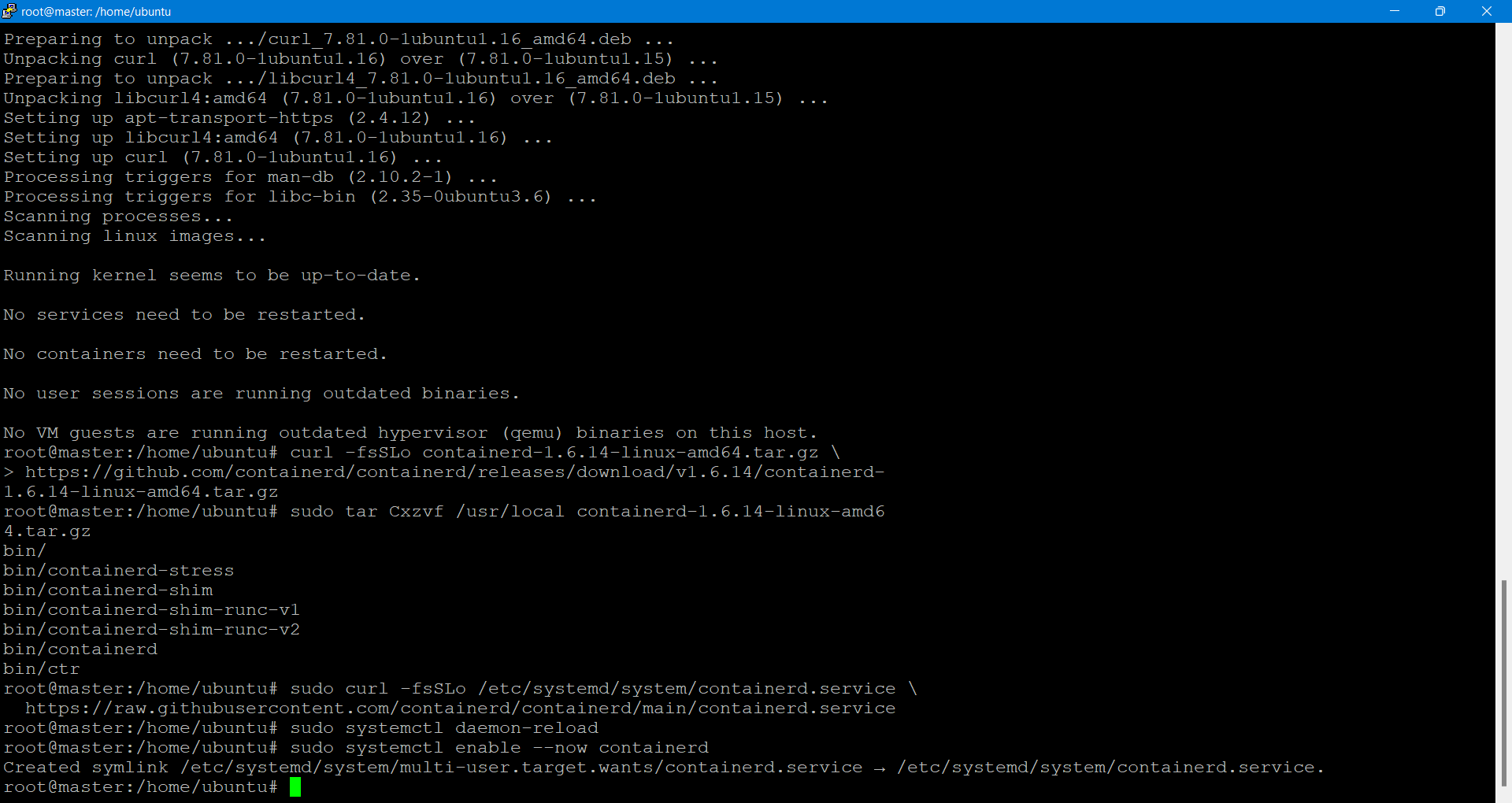
# Install containerd as a service

sudo curl -fsSLo /etc/systemd/system/containerd.service \

https://raw.githubusercontent.com/containerd/containerd/main/containerd.service

sudo systemctl daemon-reload

sudo systemctl enable --now containerd

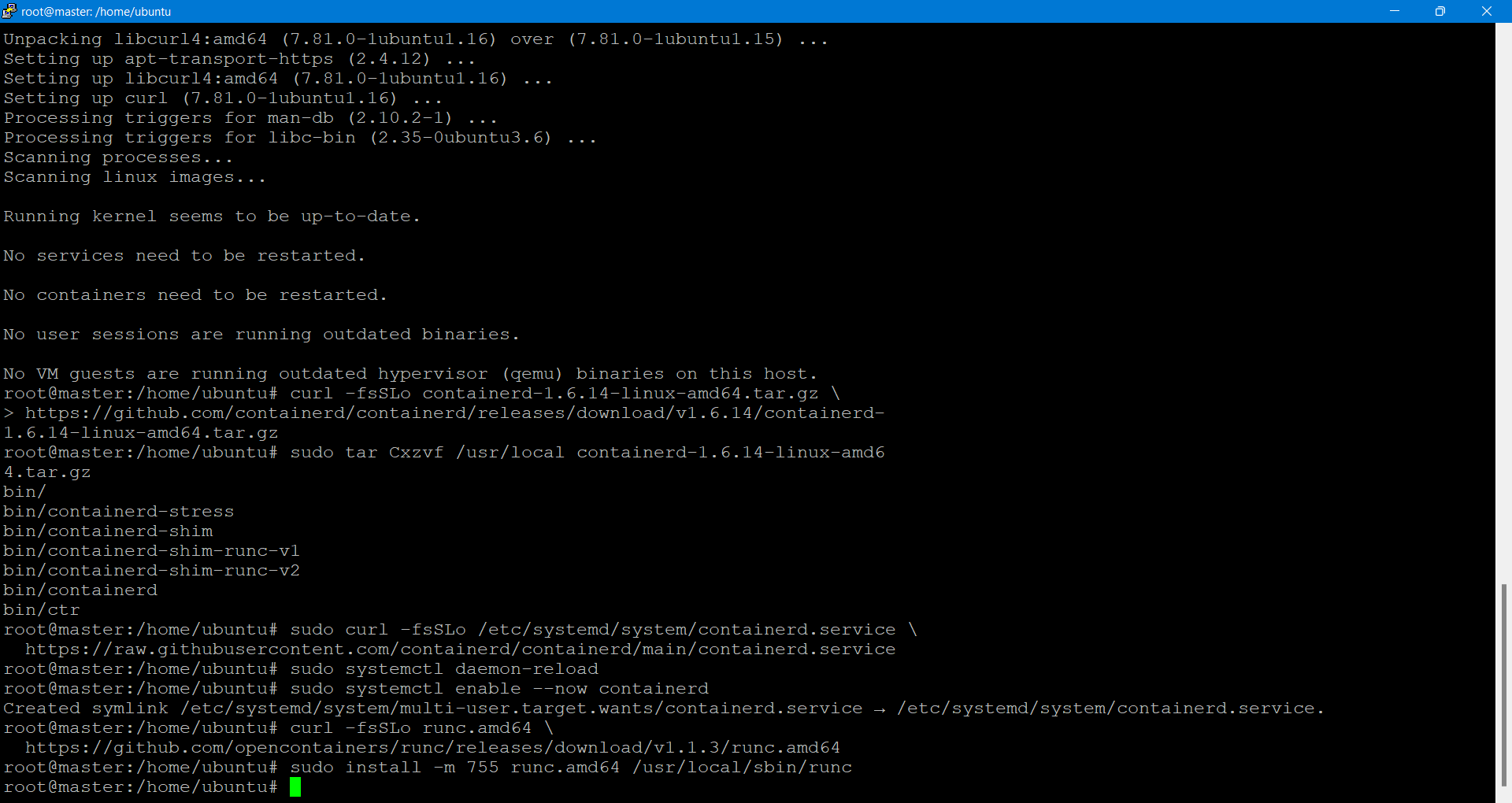


Install runc:

curl -fsSLo runc.amd64 \

https://github.com/opencontainers/runc/releases/download/v1.1.3/runc.amd64

sudo install -m 755 runc.amd64 /usr/local/sbin/runc



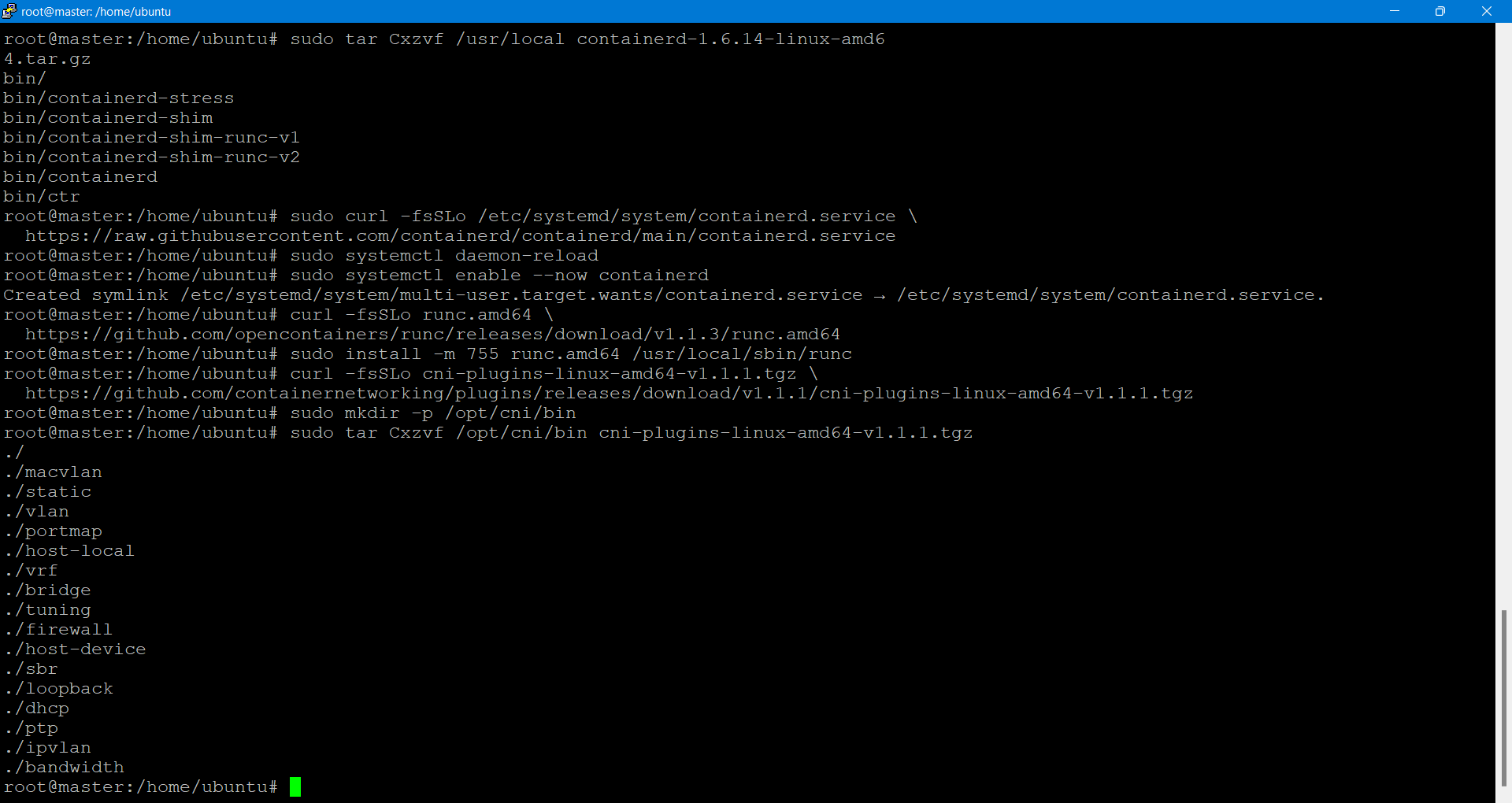
Install CNI network plugins:

curl -fsSLo cni-plugins-linux-amd64-v1.1.1.tgz \

https://github.com/containernetworking/plugins/releases/download/v1.1.1/cni-plugins-linux-amd64-v1.1.1.tgz

sudo mkdir -p /opt/cni/bin

sudo tar Cxzvf /opt/cni/bin cni-plugins-linux-amd64-v1.1.1.tgz



Forward IPv4 and let iptables see bridged network traffic:

You need to enable overlay and br\_netfilter kernel modules. Additionally, you need to allow iptables see bridged network traffic.

cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf

overlay

br\_netfilter

EOF

sudo modprobe -a overlay br\_netfilter

# sysctl params required by setup, params persist across reboots

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf

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

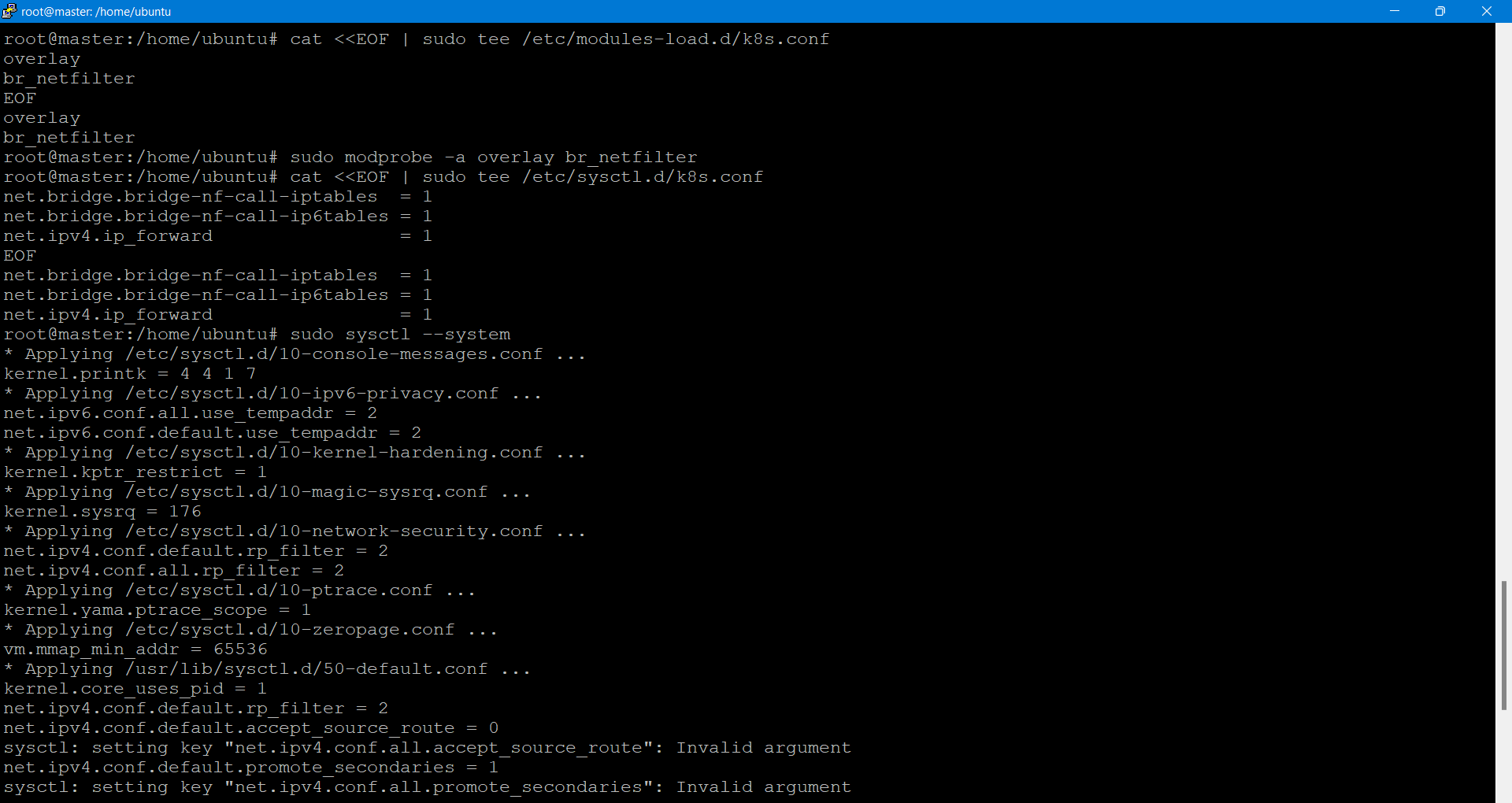
net.bridge.bridge-nf-call-ip6tables = 1

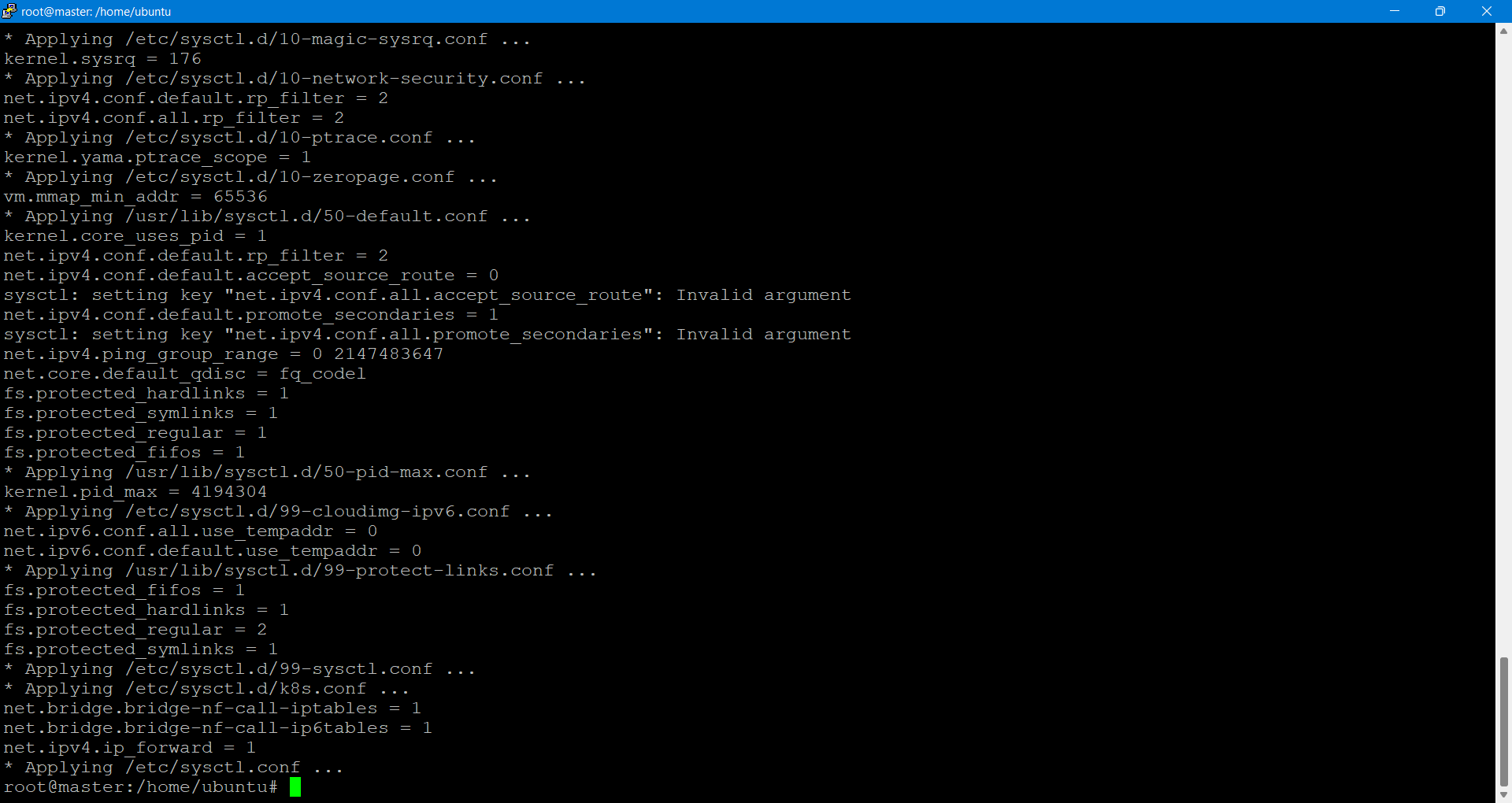
net.ipv4.ip\_forward = 1

EOF

# Apply sysctl params without reboot

sudo sysctl –system





**Install kubeadm, kubelet & kubectl:**

we need to ensure the versions of kubeadm, kubelet and kubectl are compatible.

**# Add Kubernetes GPG key**

sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg \

https://packages.cloud.google.com/apt/doc/apt-key.gpg

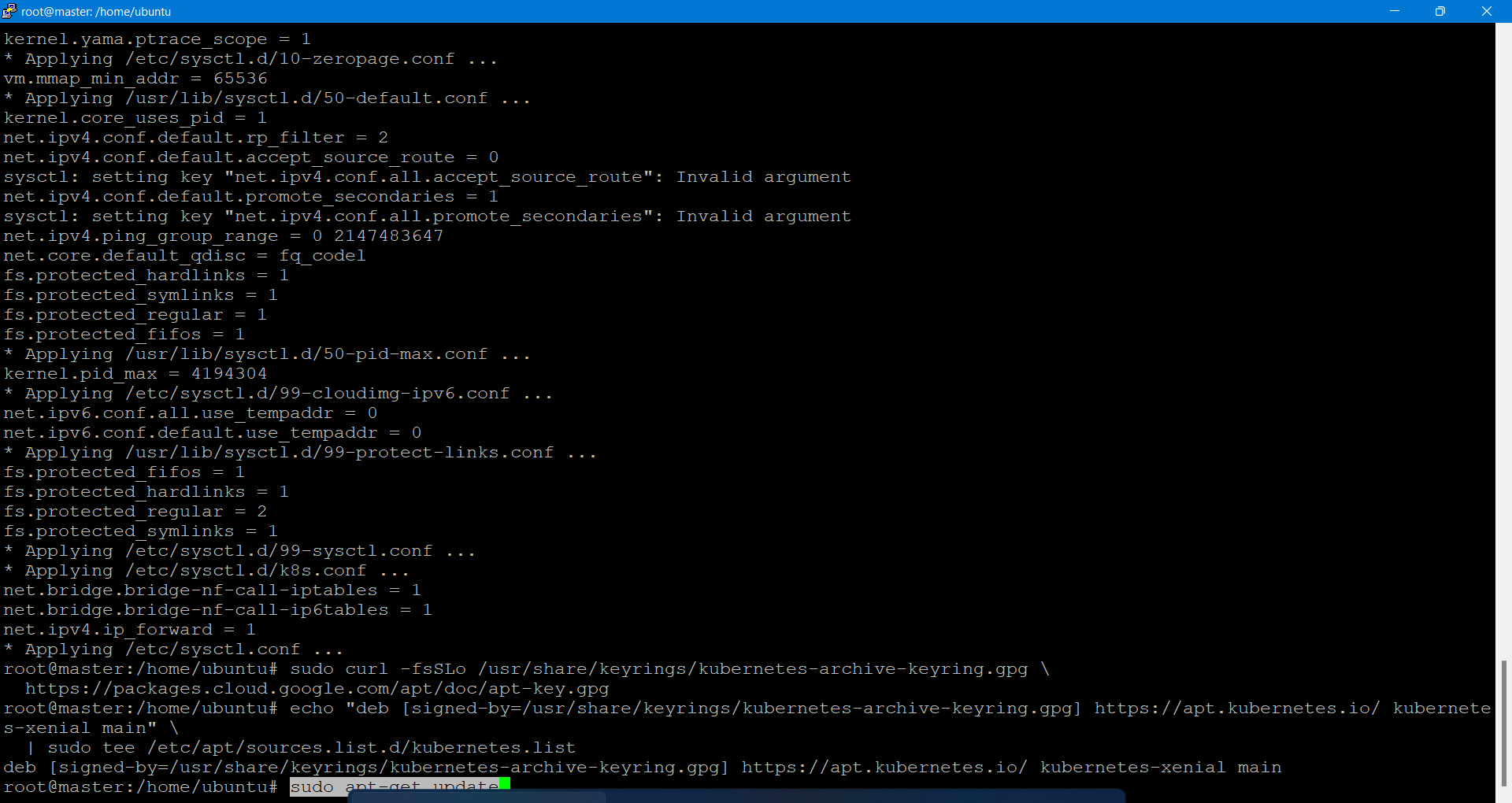
**# Add Kubernetes apt repository**

echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" \

| sudo tee /etc/apt/sources.list.d/kubernetes.list

# Fetch package list

sudo apt-get update



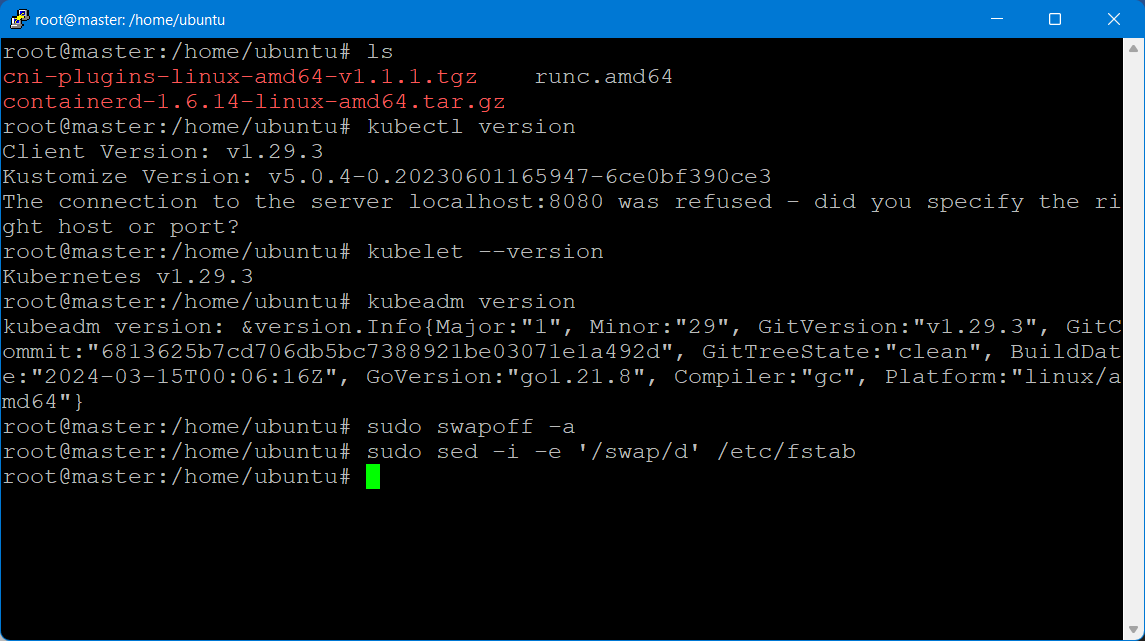
sudo apt-get install -y kubelet kubeadm kubectl

Ensure swap is disabled:

**# Turn off swap**

sudo swapoff -a

sudo sed -i -e '/swap/d' /etc/fstab



**On the worker Nodes:(worker1 and worker2)**

1.Enter the username as ubuntu

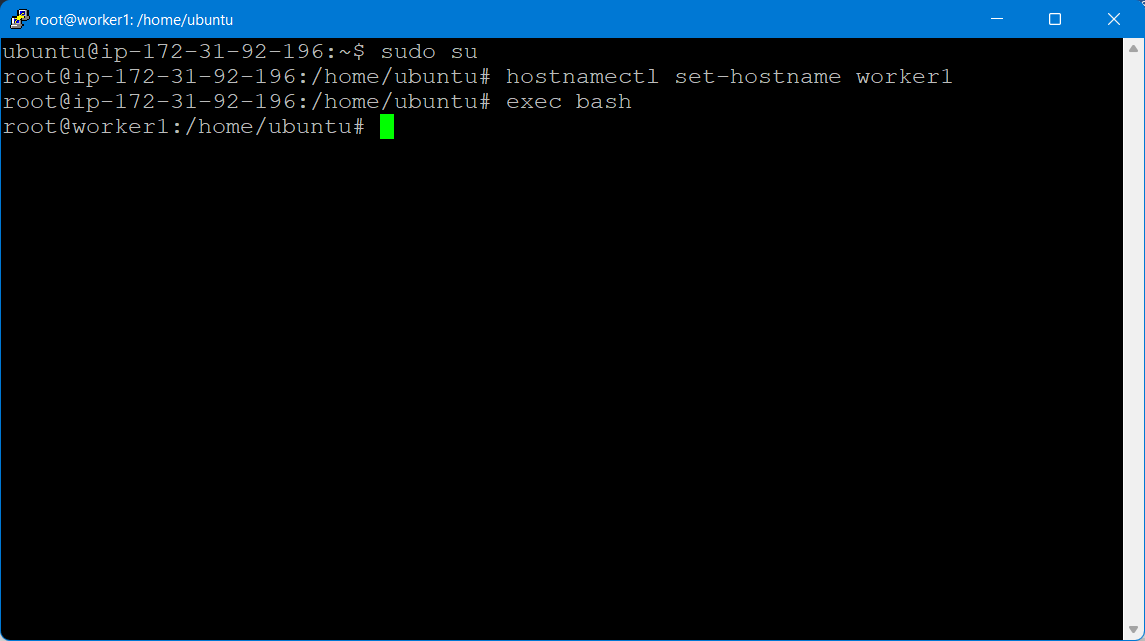
2.Switch to root

sudo su -

3.Set the hostname

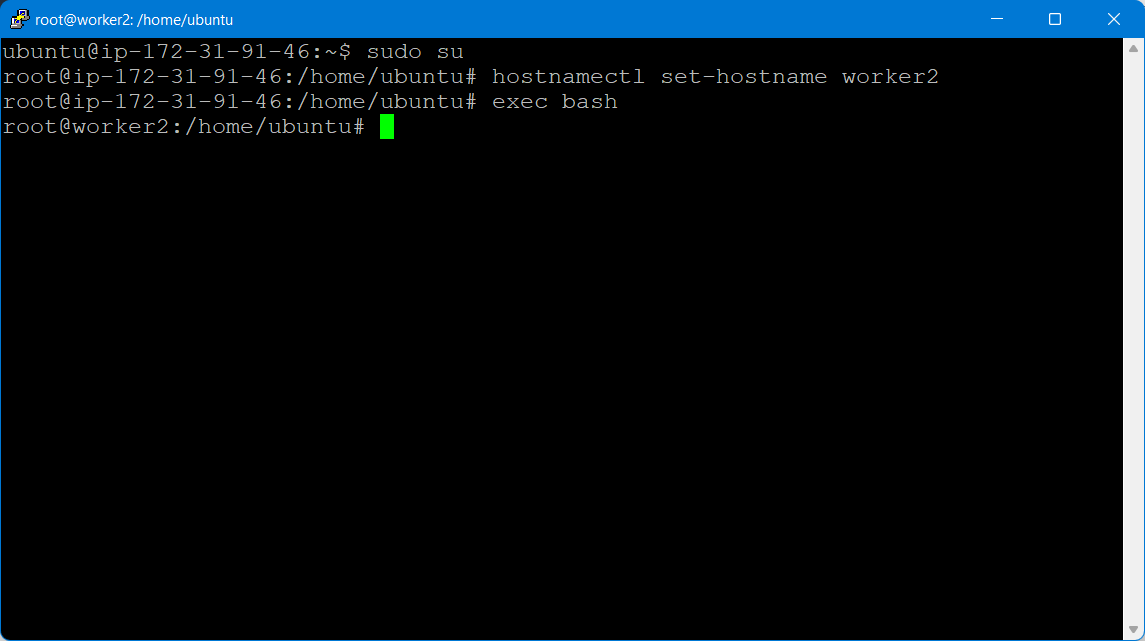
# hostnamectl set-hostname worker1

# exec bash



# hostnamectl set-hostname worker2

# exec bash

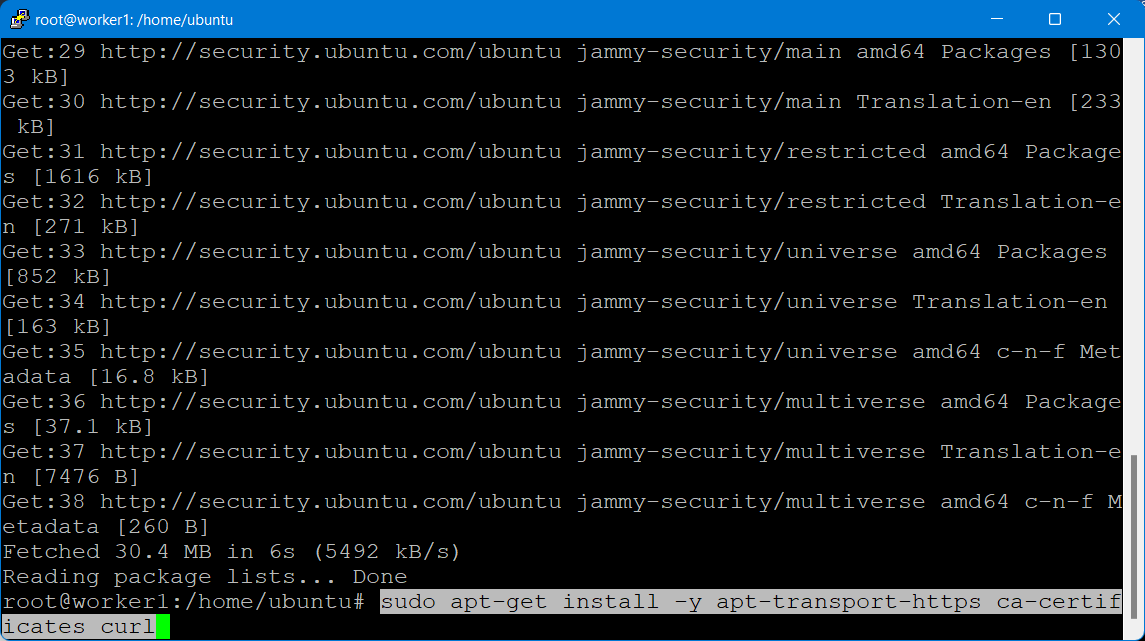


**4.Install general dependencies on both worker**

we need to install packages on your system for the commands we will use later.

sudo apt-get update

sudo apt-get install -y apt-transport-https ca-certificates curl



Install containerd

Although we have a few container runtimes to choose from, we’re going with containerd. Before we install containerd, we’ll create its configuration file.

curl -fsSLo containerd-config.toml \

https://gist.githubusercontent.com/oradwell/31ef858de3ca43addef68ff971f459c2/raw/5099df007eb717a11825c3890a0517892fa12dbf/containerd-config.toml

sudo mkdir /etc/containerd

sudo mv containerd-config.toml /etc/containerd/config.toml

Install containerd from their official GitHub repository as recommended using the following commands:

curl -fsSLo containerd-1.6.14-linux-amd64.tar.gz \

https://github.com/containerd/containerd/releases/download/v1.6.14/containerd-1.6.14-linux-amd64.tar.gz

# Extract the binaries

sudo tar Cxzvf /usr/local containerd-1.6.14-linux-amd64.tar.gz

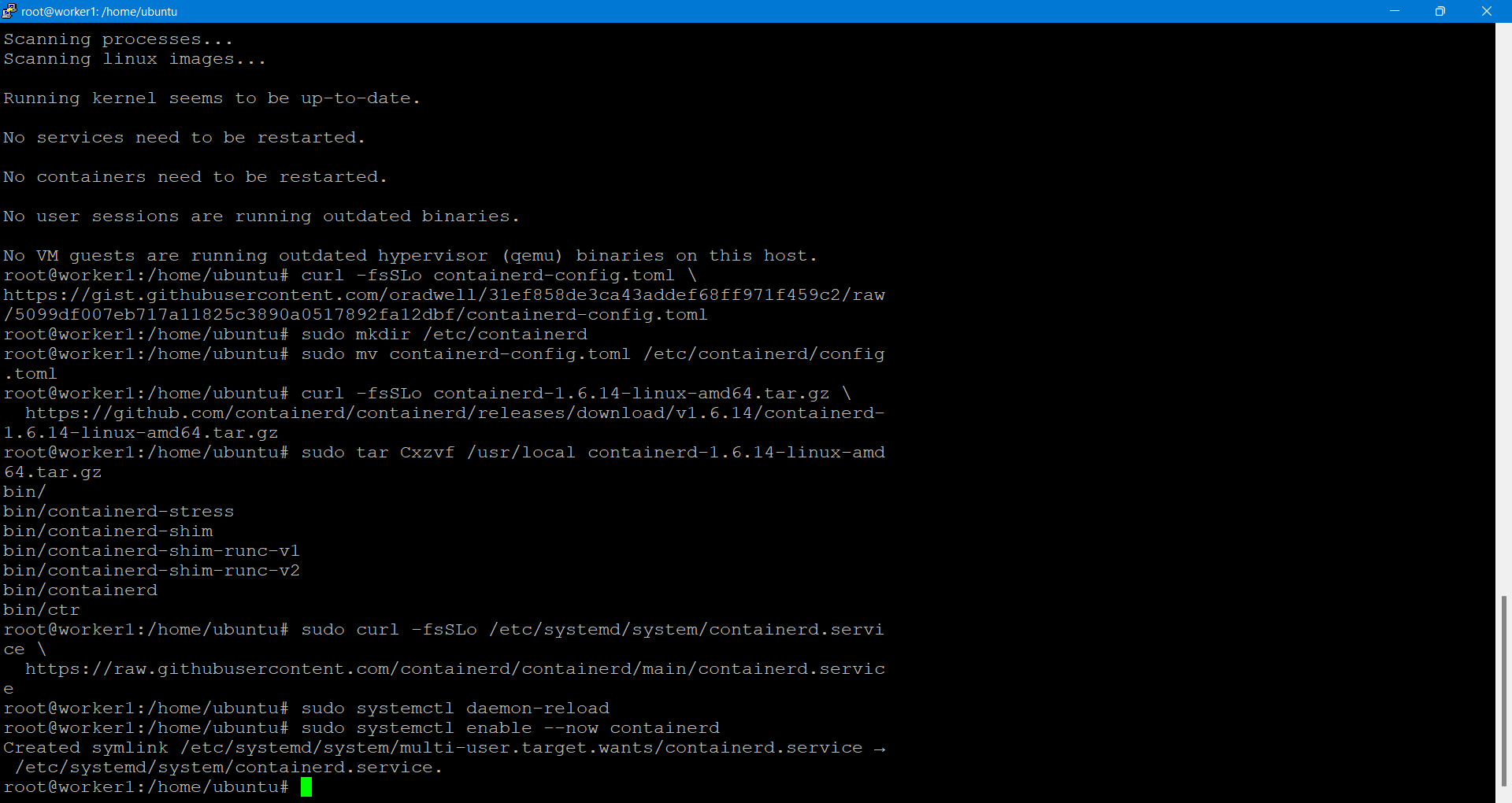
# Install containerd as a service

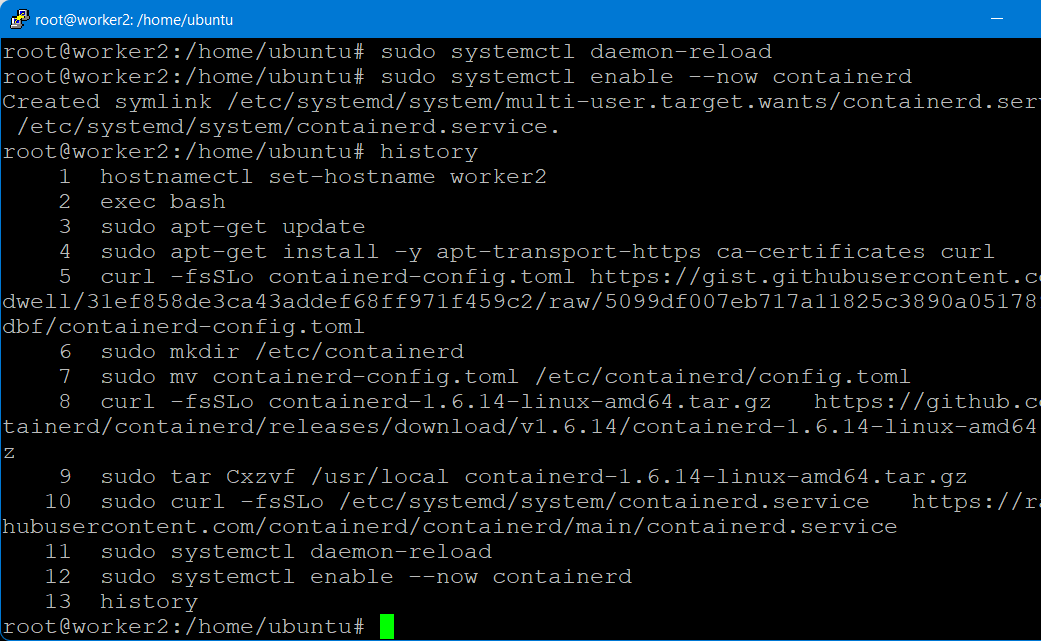
sudo curl -fsSLo /etc/systemd/system/containerd.service \

https://raw.githubusercontent.com/containerd/containerd/main/containerd.service

sudo systemctl daemon-reload

sudo systemctl enable --now containerd





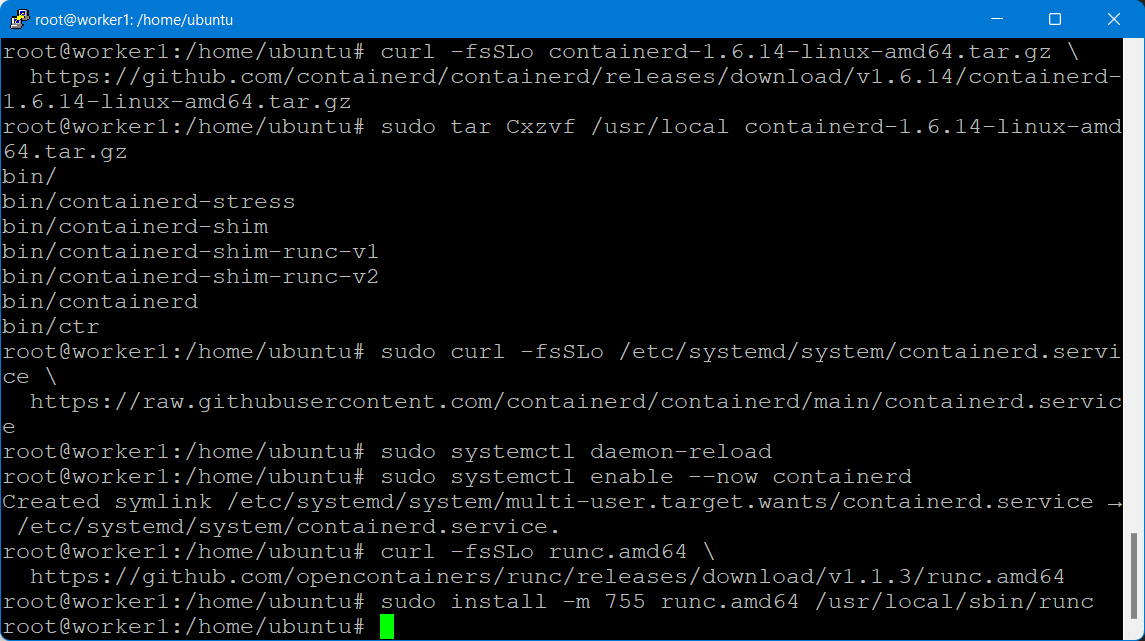
Install runc:

Installing runc from their official GitHub repository is the recommended way.

curl -fsSLo runc.amd64 \

https://github.com/opencontainers/runc/releases/download/v1.1.3/runc.amd64

sudo install -m 755 runc.amd64 /usr/local/sbin/runc



Install CNI network plugins

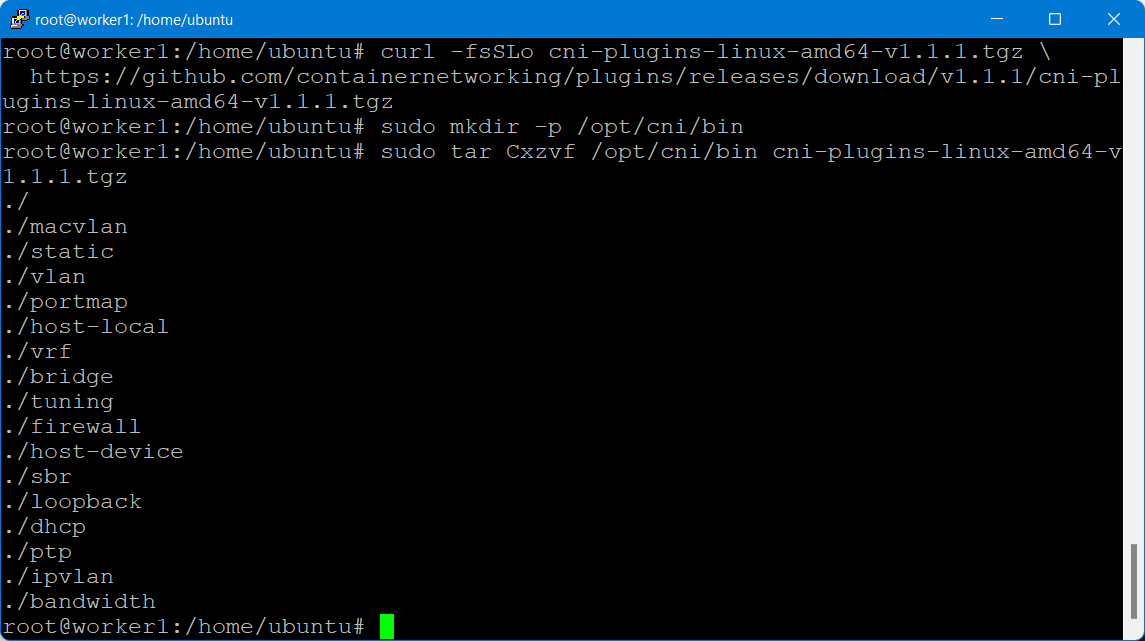
Install Container Network Interface network plugins from their official GitHub repository.

curl -fsSLo cni-plugins-linux-amd64-v1.1.1.tgz \

https://github.com/containernetworking/plugins/releases/download/v1.1.1/cni-plugins-linux-amd64-v1.1.1.tgz

sudo mkdir -p /opt/cni/bin

sudo tar Cxzvf /opt/cni/bin cni-plugins-linux-amd64-v1.1.1.tgz



Forward IPv4 and let iptables see bridged network traffic:

You need to enable overlay and br\_netfilter kernel modules. Additionally, you need to allow iptables see bridged network traffic.

cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf

overlay

br\_netfilter

EOF

sudo modprobe -a overlay br\_netfilter

# sysctl params required by setup, params persist across reboots

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf

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

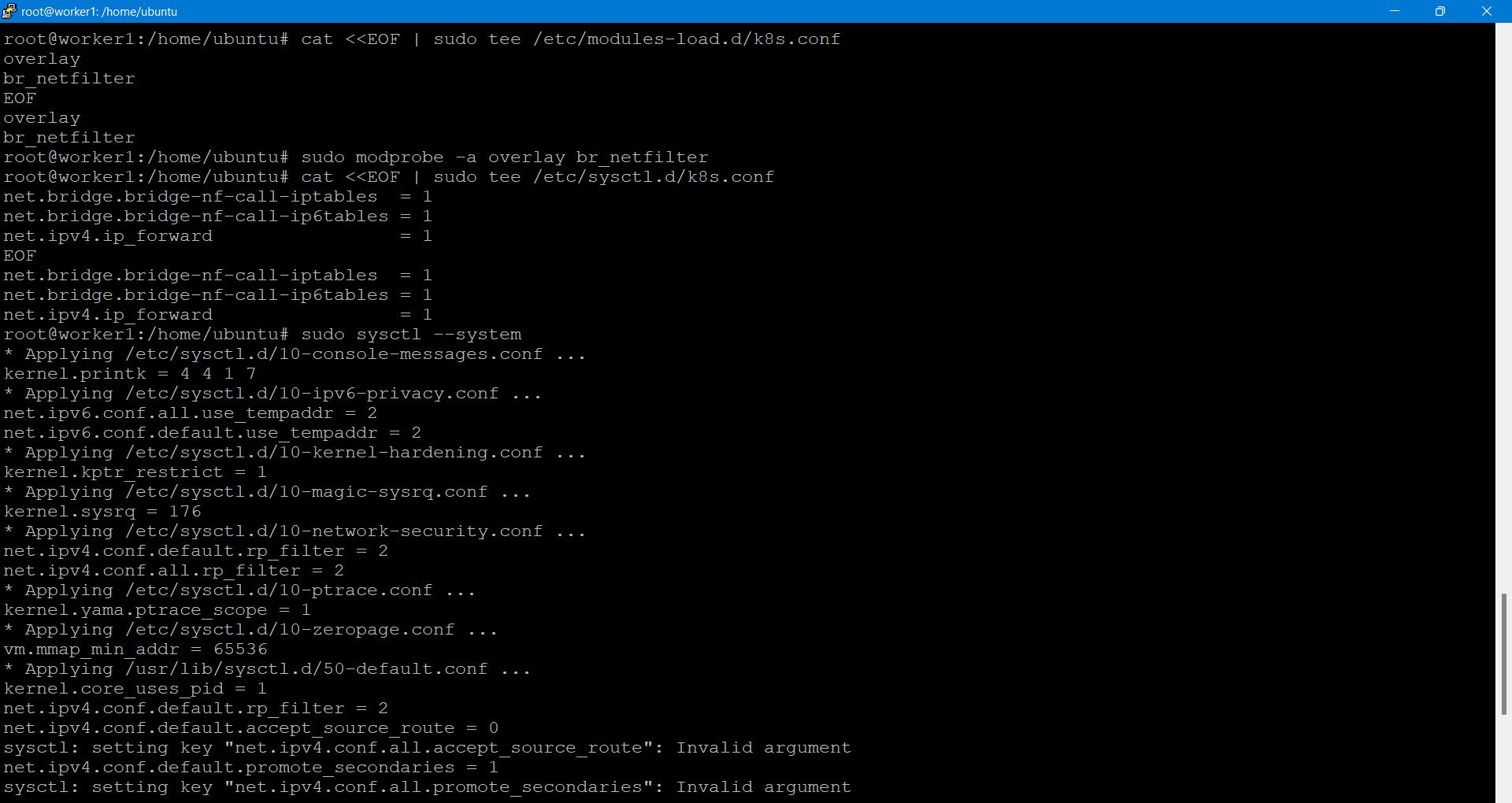
net.bridge.bridge-nf-call-ip6tables = 1

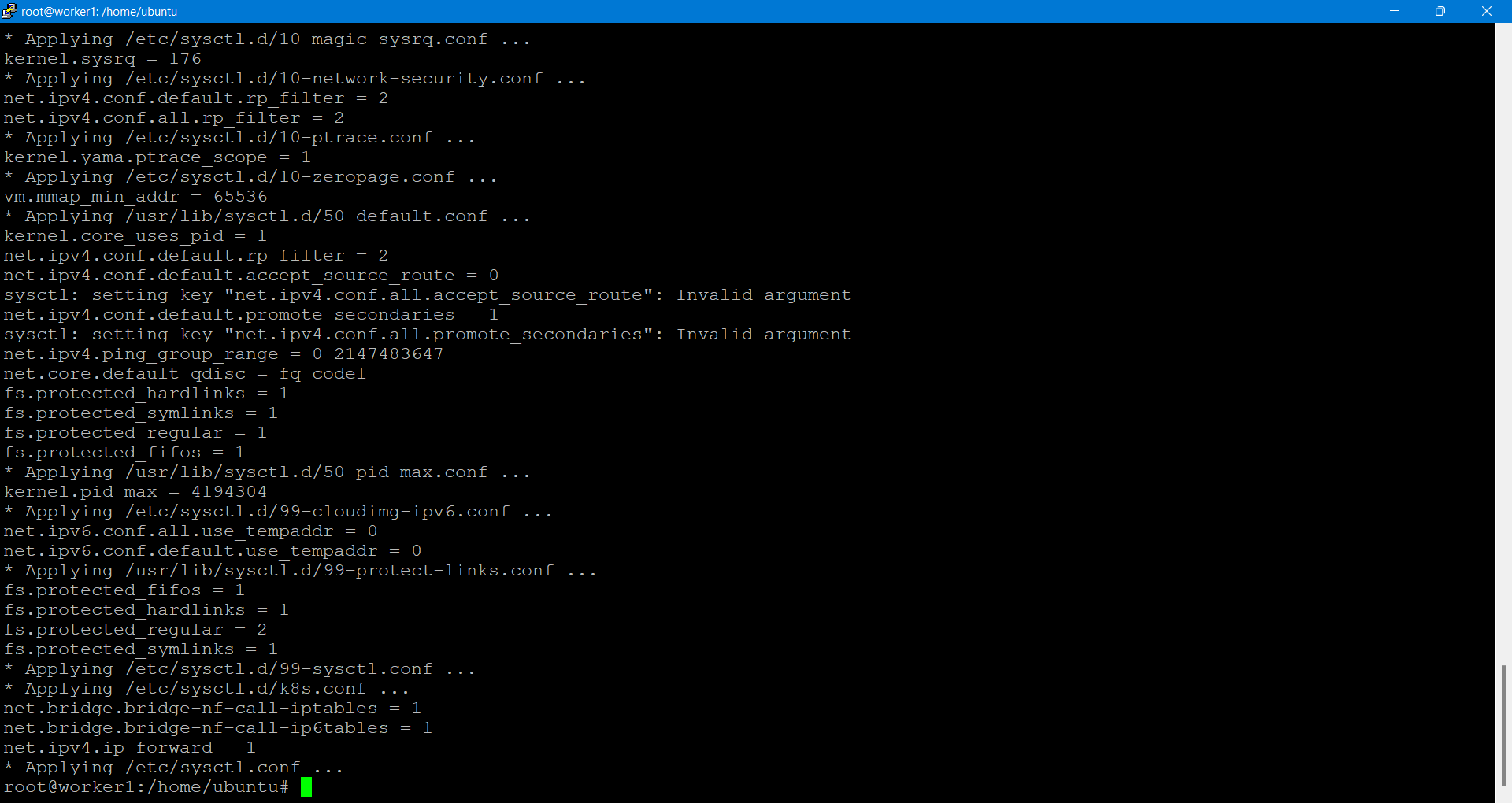
net.ipv4.ip\_forward = 1

EOF

# Apply sysctl params without reboot

sudo sysctl –system





Now Install kubeadm, kubelet &

You need to ensure the versions of kubeadm, kubelet are compatible.

# Add Kubernetes GPG key

sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg \

https://packages.cloud.google.com/apt/doc/apt-key.gpg

# Add Kubernetes apt repository

echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" \

| sudo tee /etc/apt/sources.list.d/kubernetes.list

# Fetch package list

sudo apt-get update

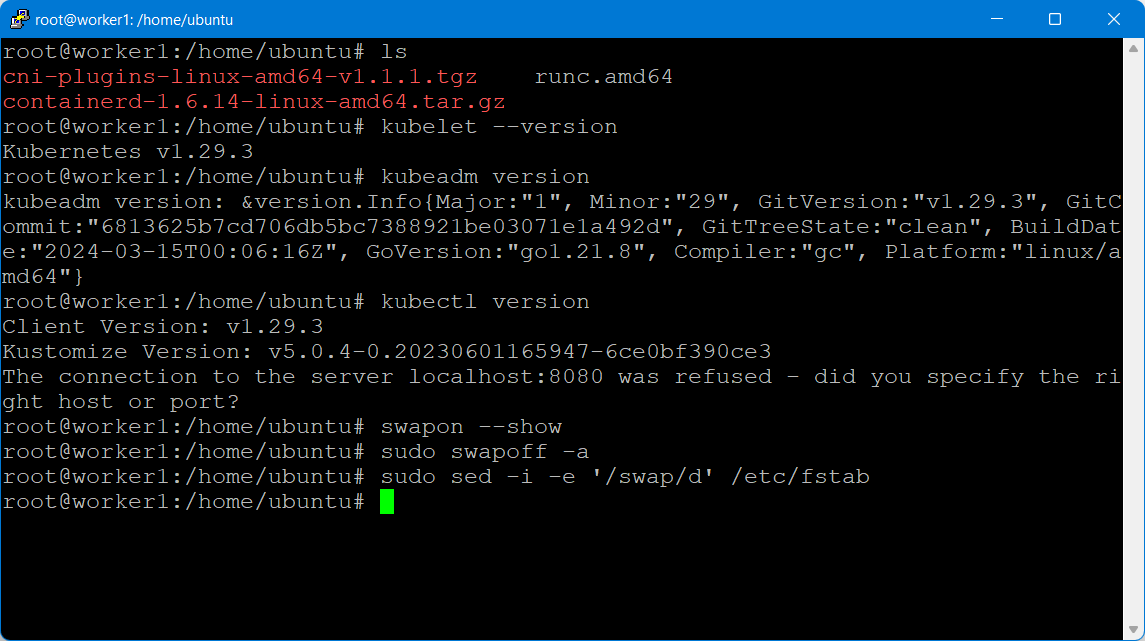
sudo apt-get install -y kubelet kubeadm

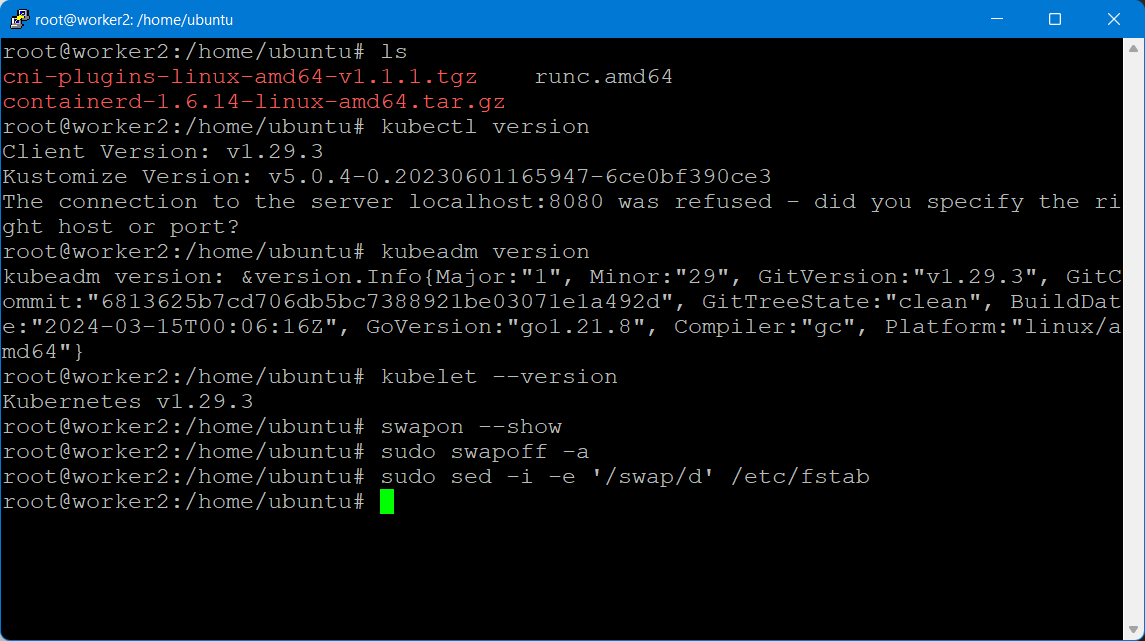
Ensure swap is disabled

# Turn off swap

sudo swapoff -a

sudo sed -i -e '/swap/d' /etc/fstab





**On the Master:**

Create the cluster using kubeadm

kubeadm init

It will create all the master components ..

Copy the token at the end

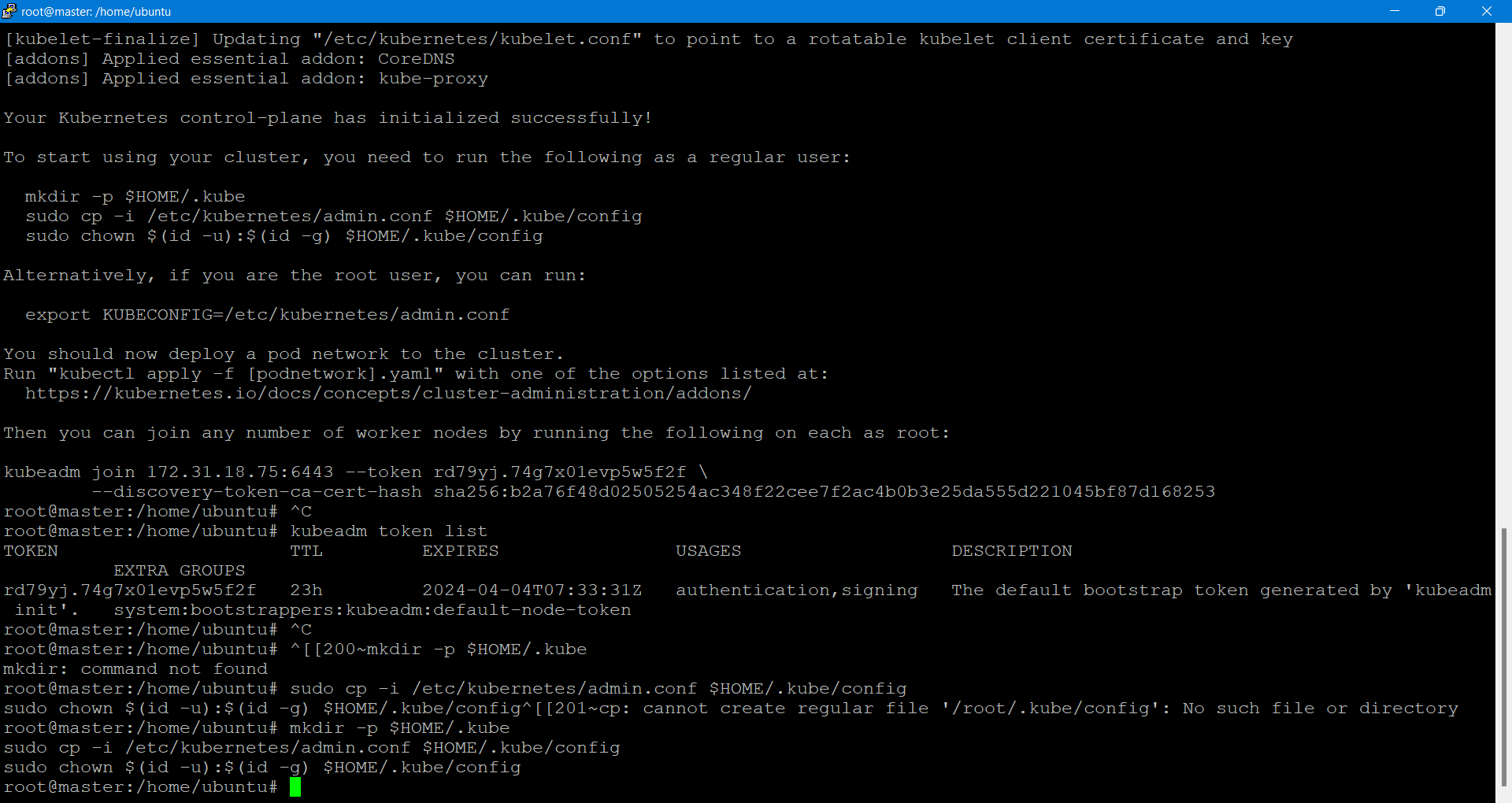
Configure kubectl

To access the cluster, we have to configure kubectl.

mkdir -p $HOME/.kube

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

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



>View node information on the master. The nodes will not be ready.

kubectl get nodes

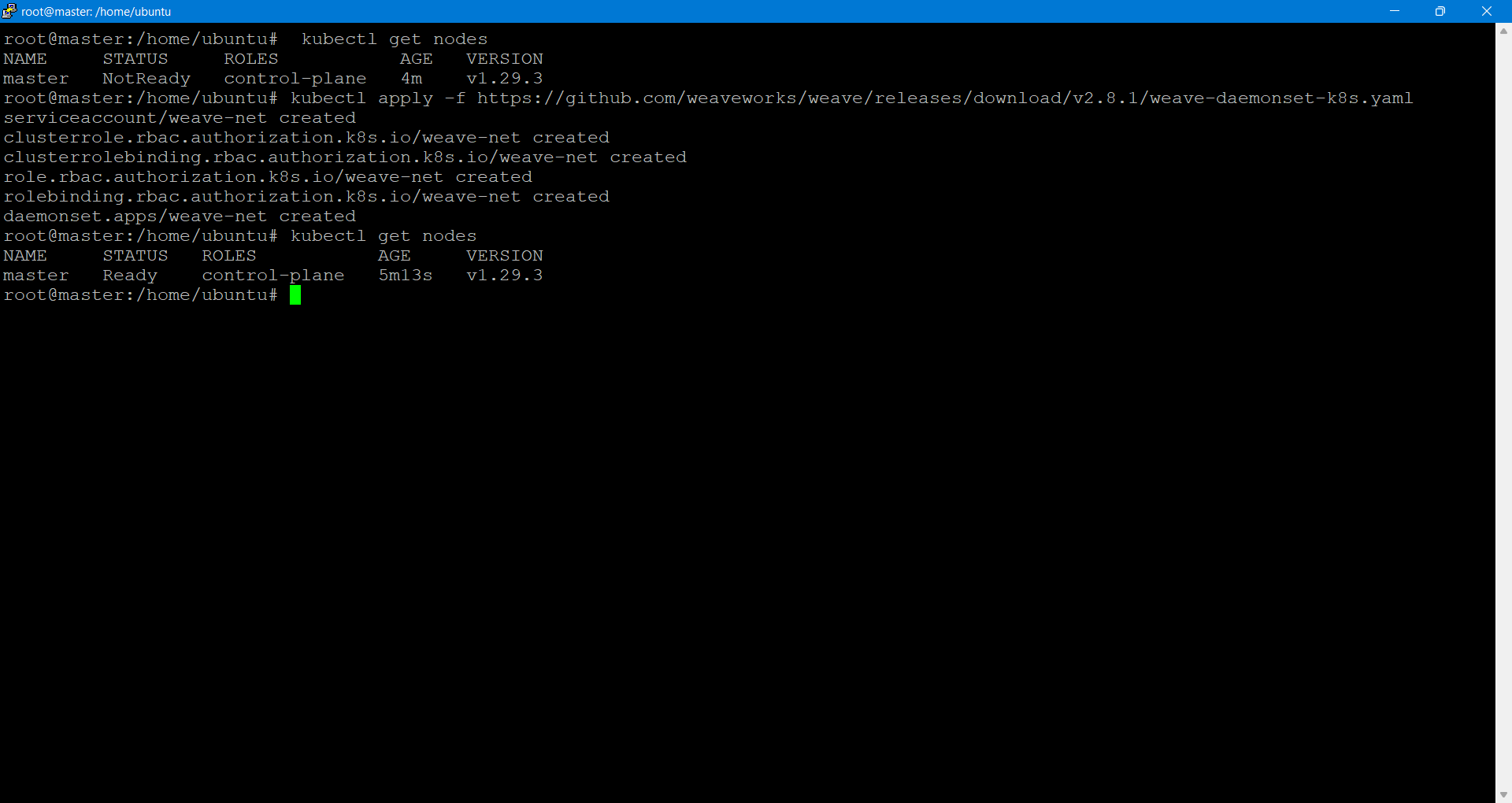
> Let us add the Network CNI on the master

Apply weave CNI (Container Network Interface) as shown below:

# kubectl apply -f <https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml>

>View nodes to see that they are ready

kubectl get nodes

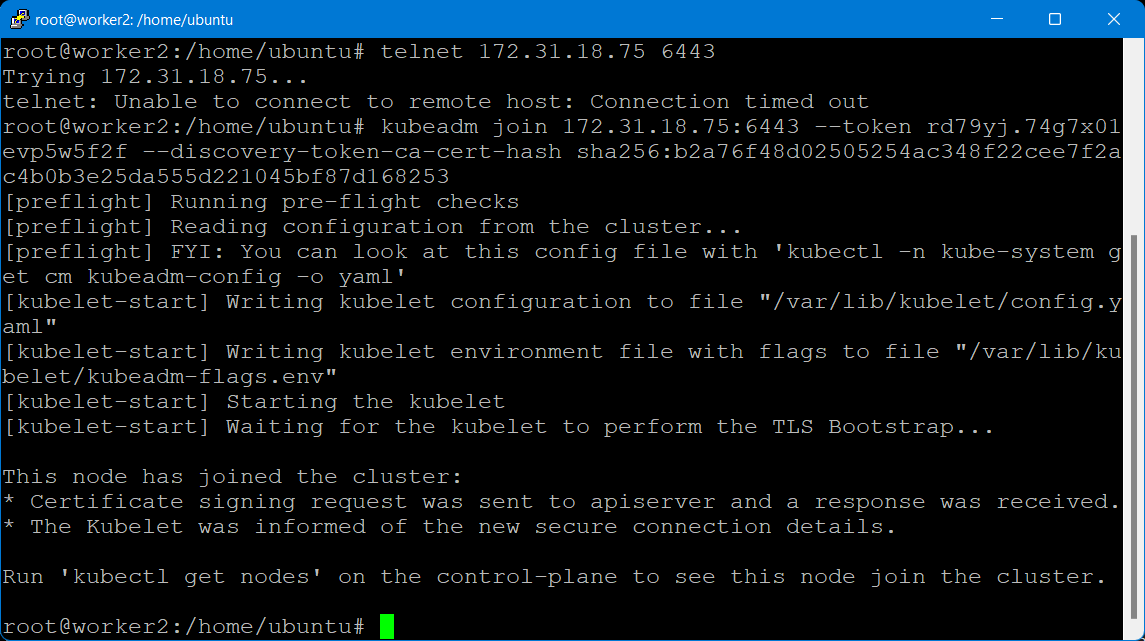


On the worker nodes:

Run the token to join to the master

kubeadm join 172.31.18.75:6443 --token rd79yj.74g7x01evp5w5f2f \

--discovery-token-ca-cert-hash sha256:b2a76f48d02505254ac348f22cee7f2ac4b0b3e25da555d221045bf87d168253



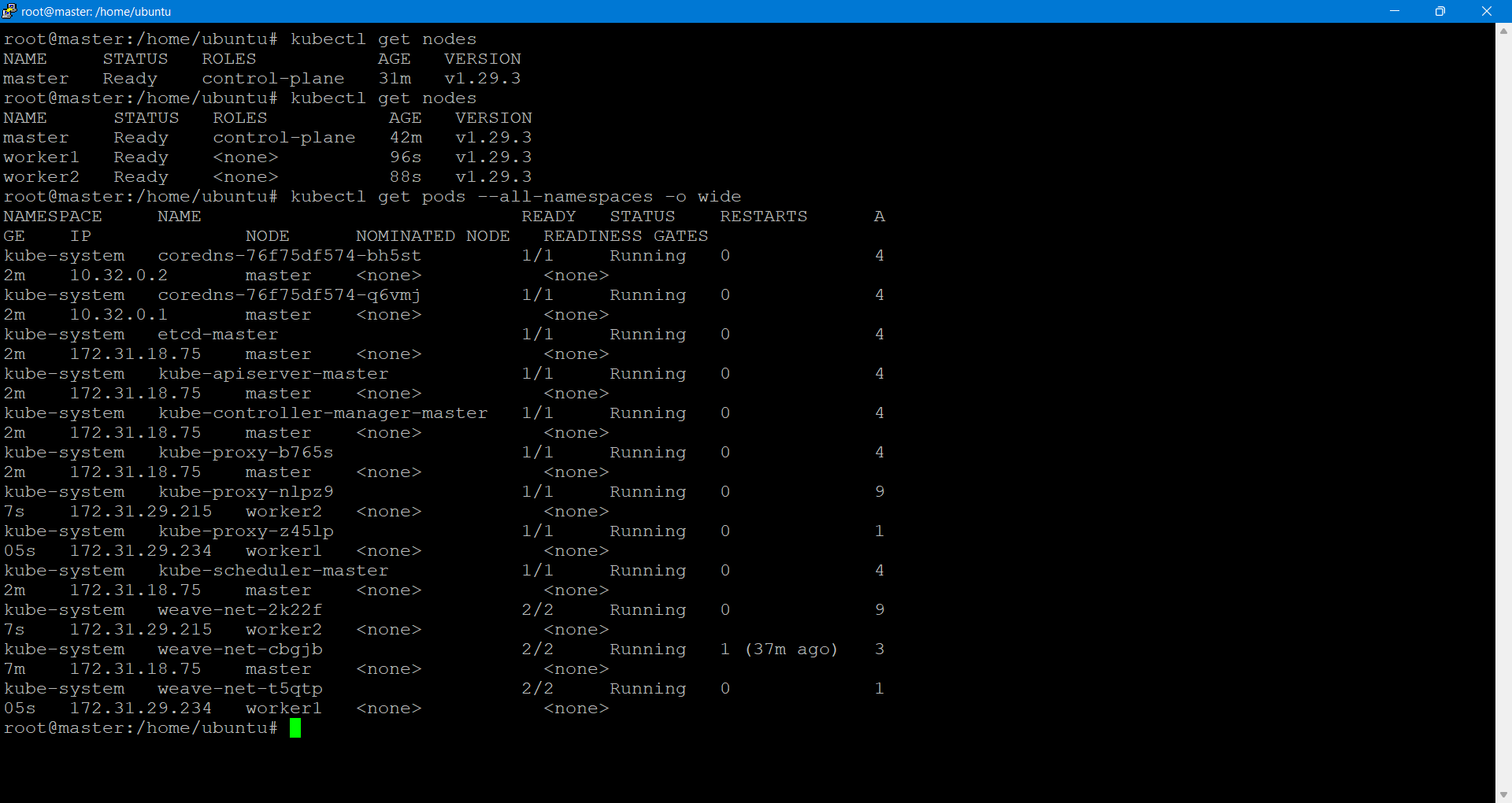
On the master:

View nodes to see that they are ready

kubectl get nodes

Check if all the management pods has been created:

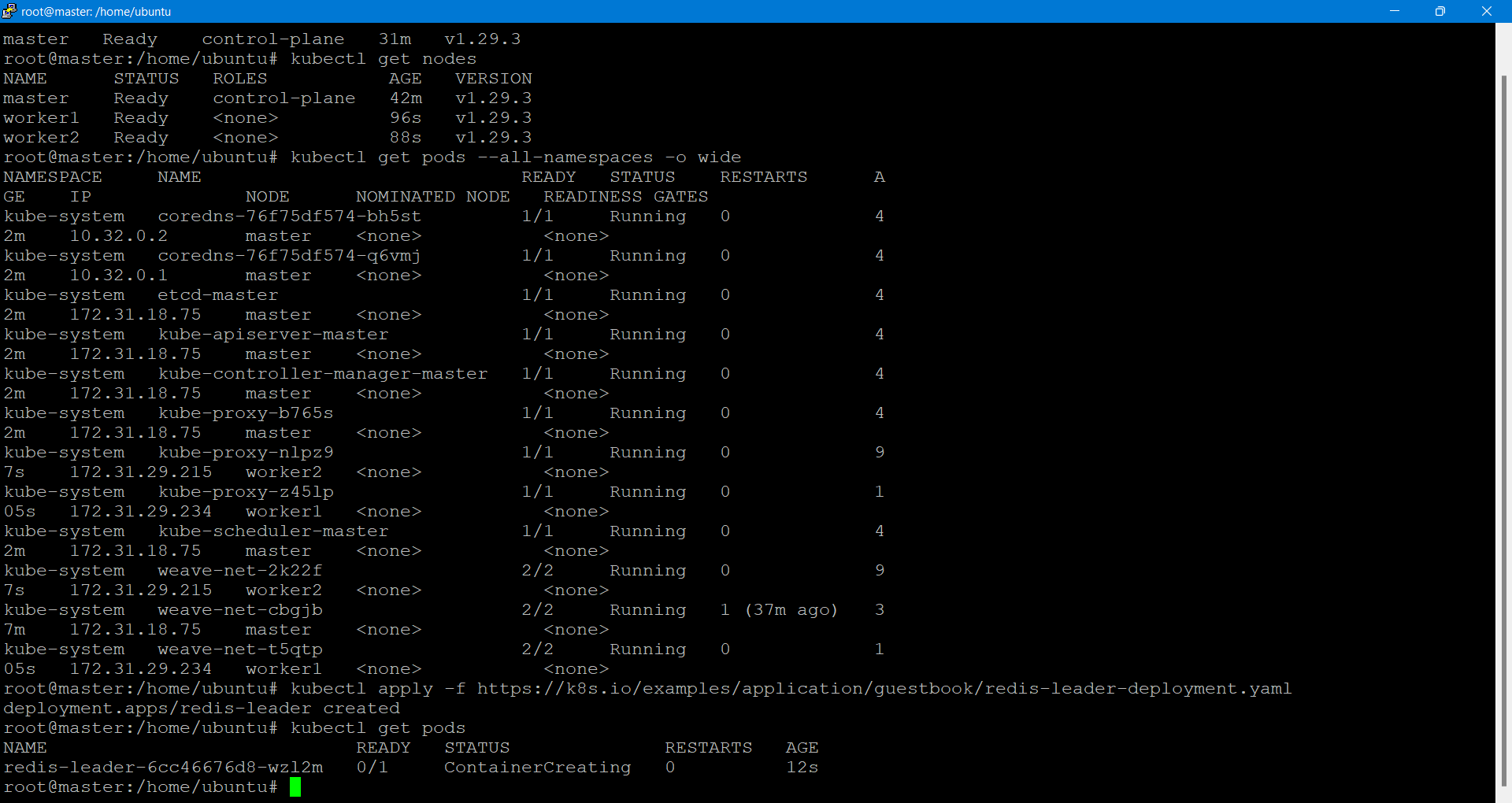
kubectl get pods --all-namespaces -o wide



**Start up the Redis Database**

The guestbook application uses Redis to store its data.

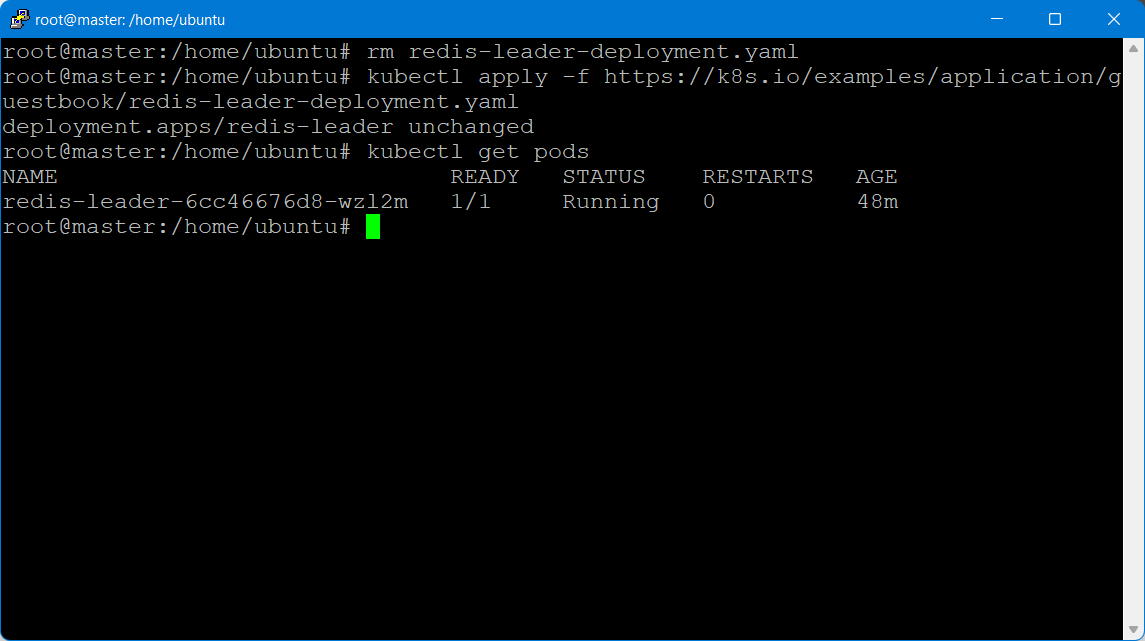
**1.Creating the Redis Leader**



Apply the Redis Deployment from the redis-leader-deployment.yaml file:

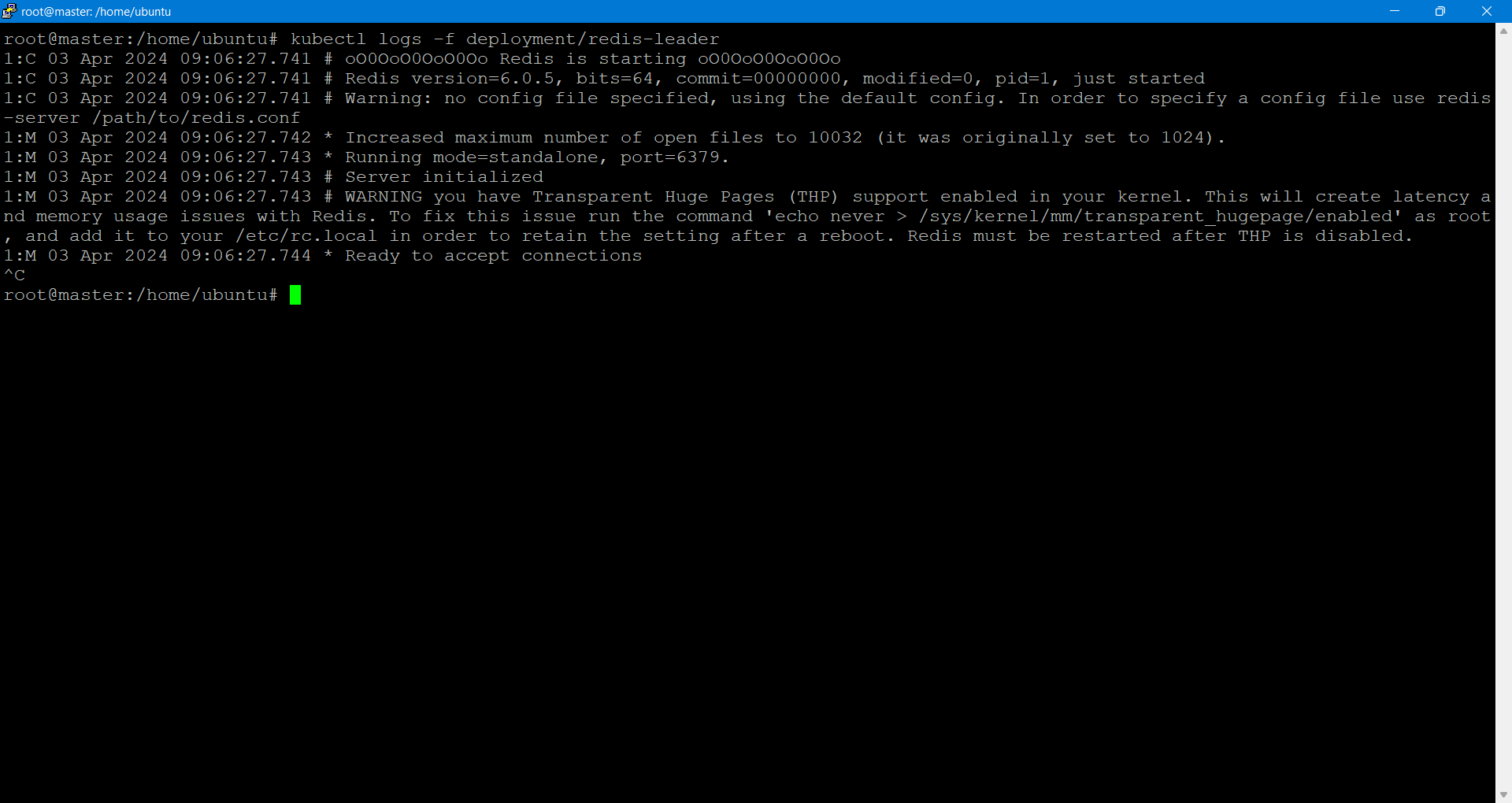
kubectl apply -f redis-leader-deployment.yaml

kubectl get pods

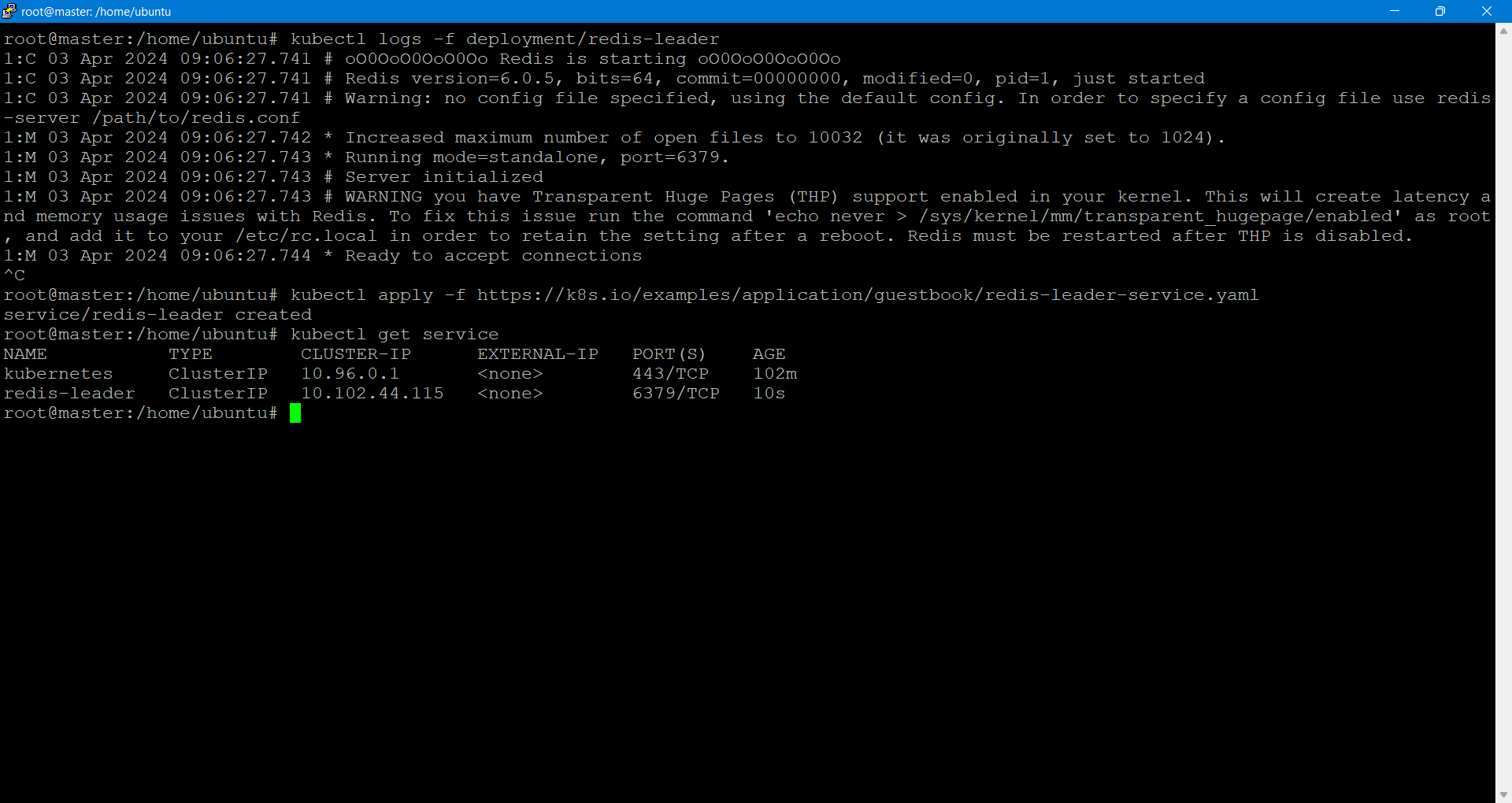


Run the following command to view the logs from the Redis leader Pod:

kubectl logs -f deployment/redis-leader



Creating the Redis leader Service



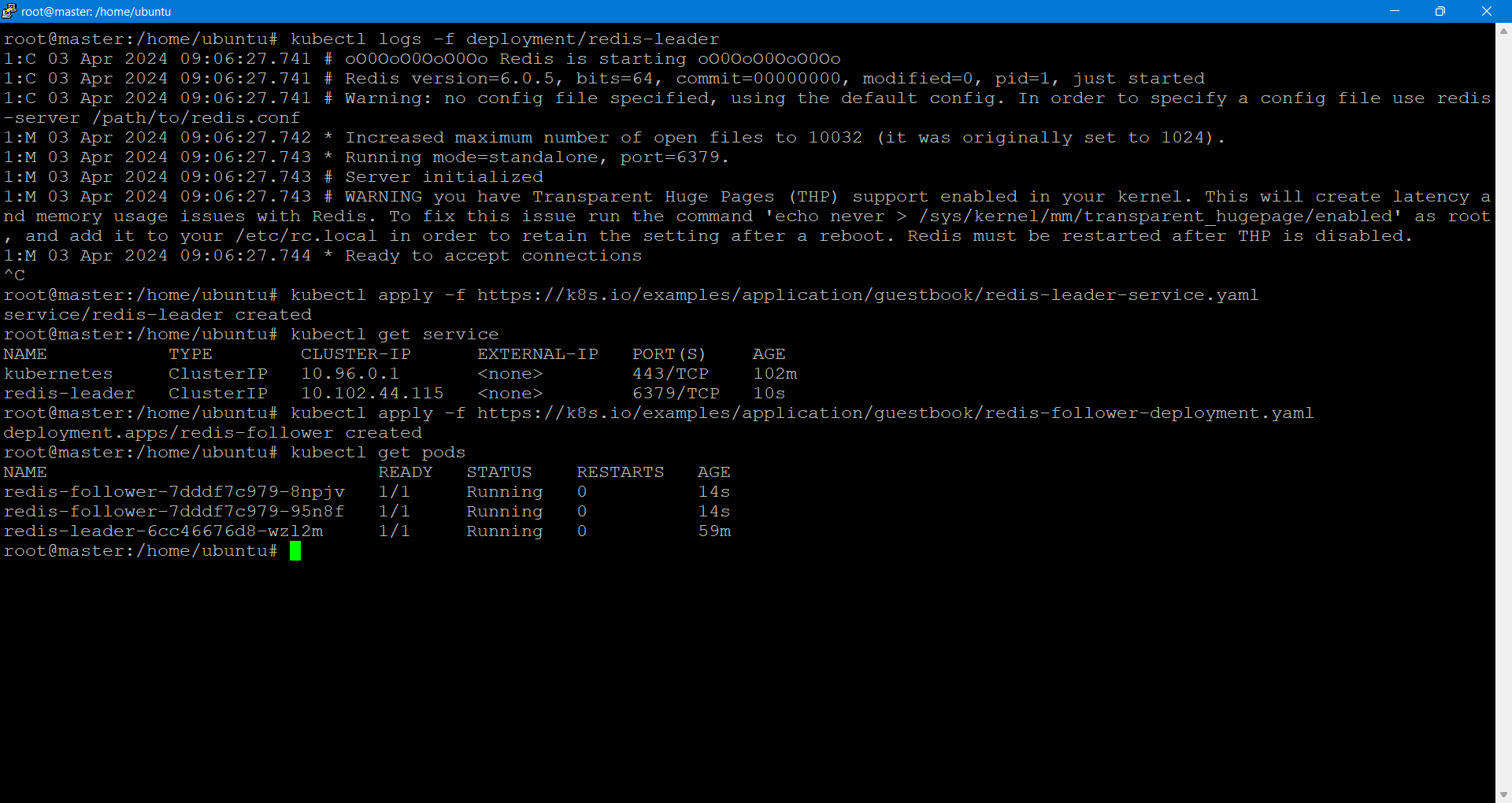
**2.Set up Redis followers**

Apply the Redis Deployment from the following redis-follower-deployment.yaml file:

kubectl apply -f https://k8s.io/examples/application/guestbook/redis-f

Verify that the two Redis follower replicas are running by querying the list of Pods:

kubectl get pods

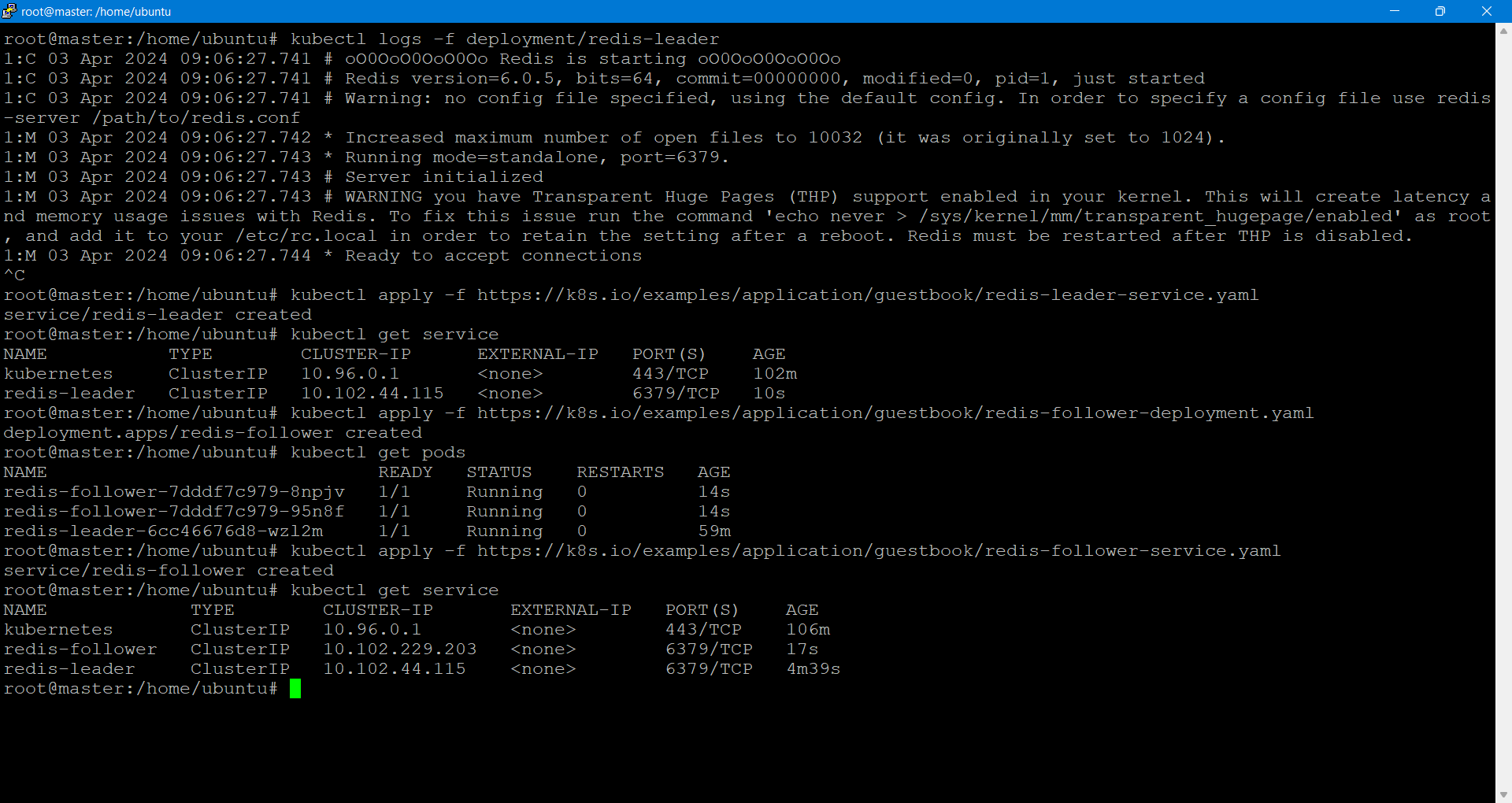


Creating the Redis follower service  
Apply the Redis Service from the following redis-follower-service.yaml file:

kubectl apply -f https://k8s.io/examples/application/guestbook/redis-fol

Query the list of Services to verify that the Redis Service is running:

kubectl get service



**3.Set up and Expose the Guestbook Frontend**

Creating the Guestbook Frontend Deployment

Apply the frontend Deployment from the frontend-deployment.yaml file:

Vi frontend-deployment.yaml

apiVersion: v1

kind: Service

metadata:

name: frontend

labels:

app: guestbook

tier: frontend

spec:

# if your cluster supports it, uncomment the following to automatically create

# an external load-balanced IP for the frontend service.

# type: LoadBalancer

type: NodePort

ports:

# the port that this service should serve on

- port: 80

selector:

app: guestbook

tier: frontend

>kubectl apply -f [frontend-deployment.yaml](https://k8s.io/examples/application/guestbook/frontend-deployment.yaml)

Query the list of Pods to verify that the three frontend replicas are running:

kubectl get pods -l app=guestbook -l tier=frontend

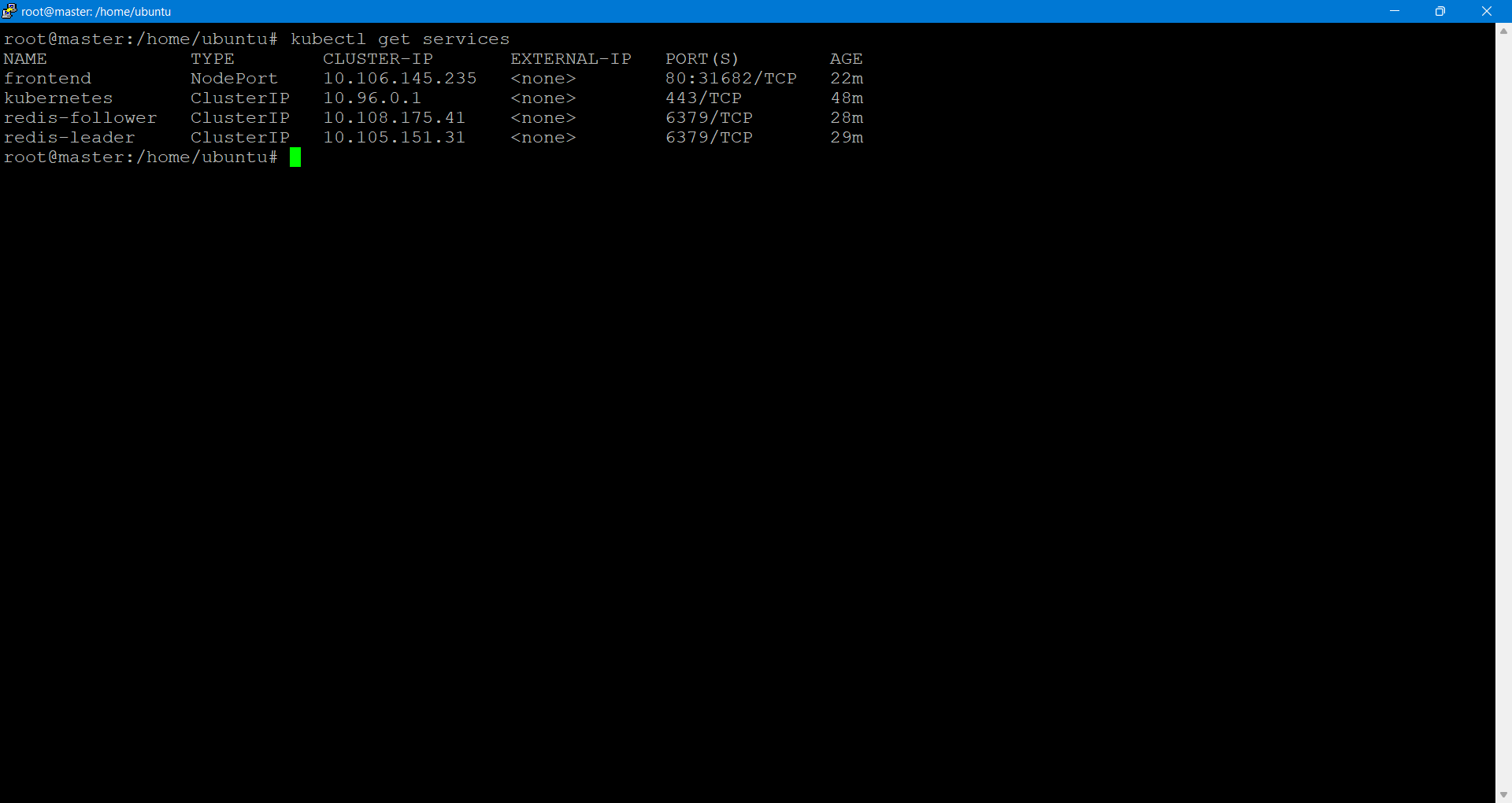
**-Creating the Frontend Service**

Apply the frontend Service from the frontend-service.yaml file:

kubectl apply -f <https://k8s.io/examples/application/guestbook/frontend-service.yaml>

Query the list of Services to verify that the frontend Service is running:

kubectl get services

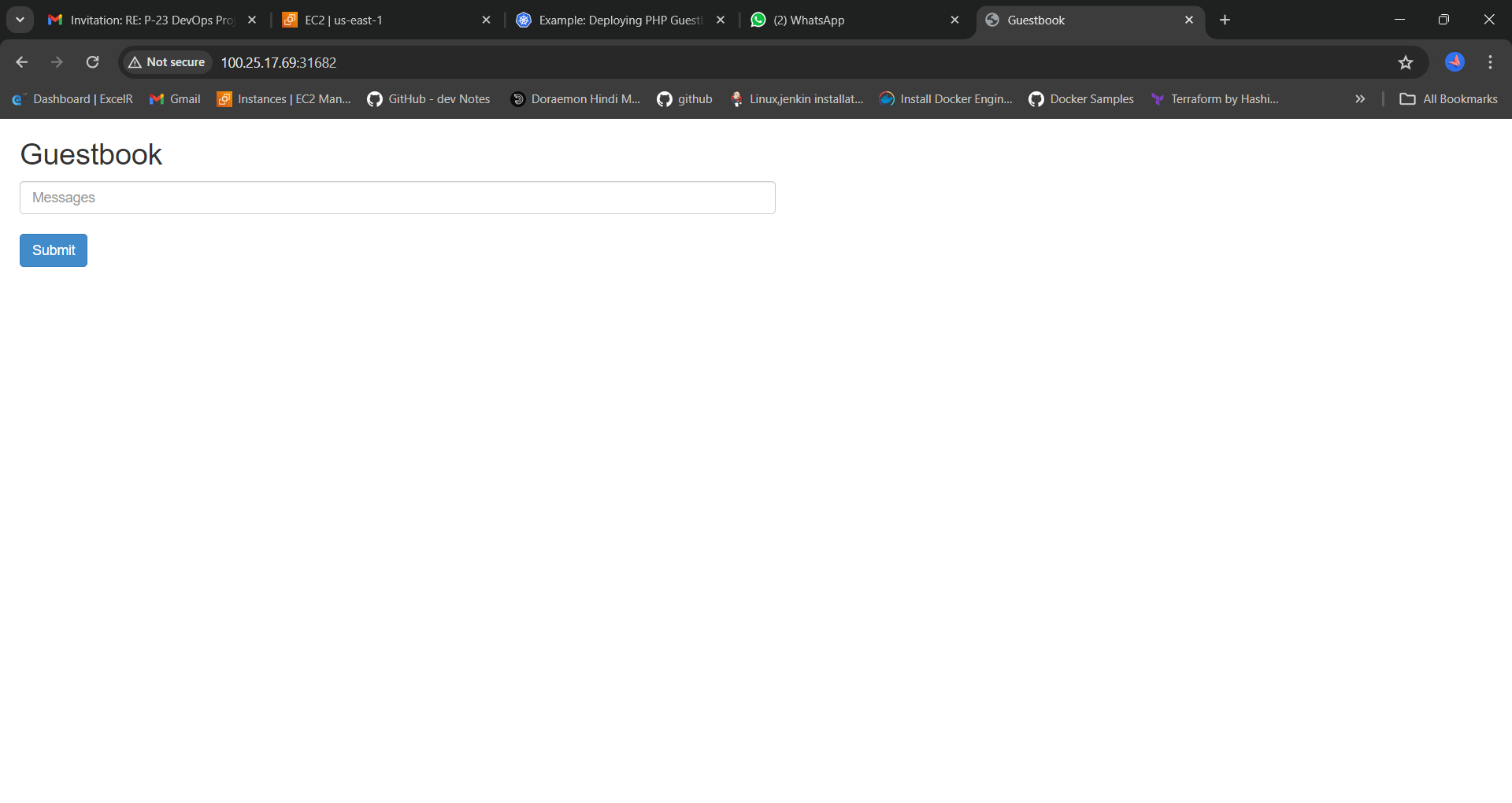


**4.Expose and view the Frontend Service (Use Kubernetes NodePort Service)**

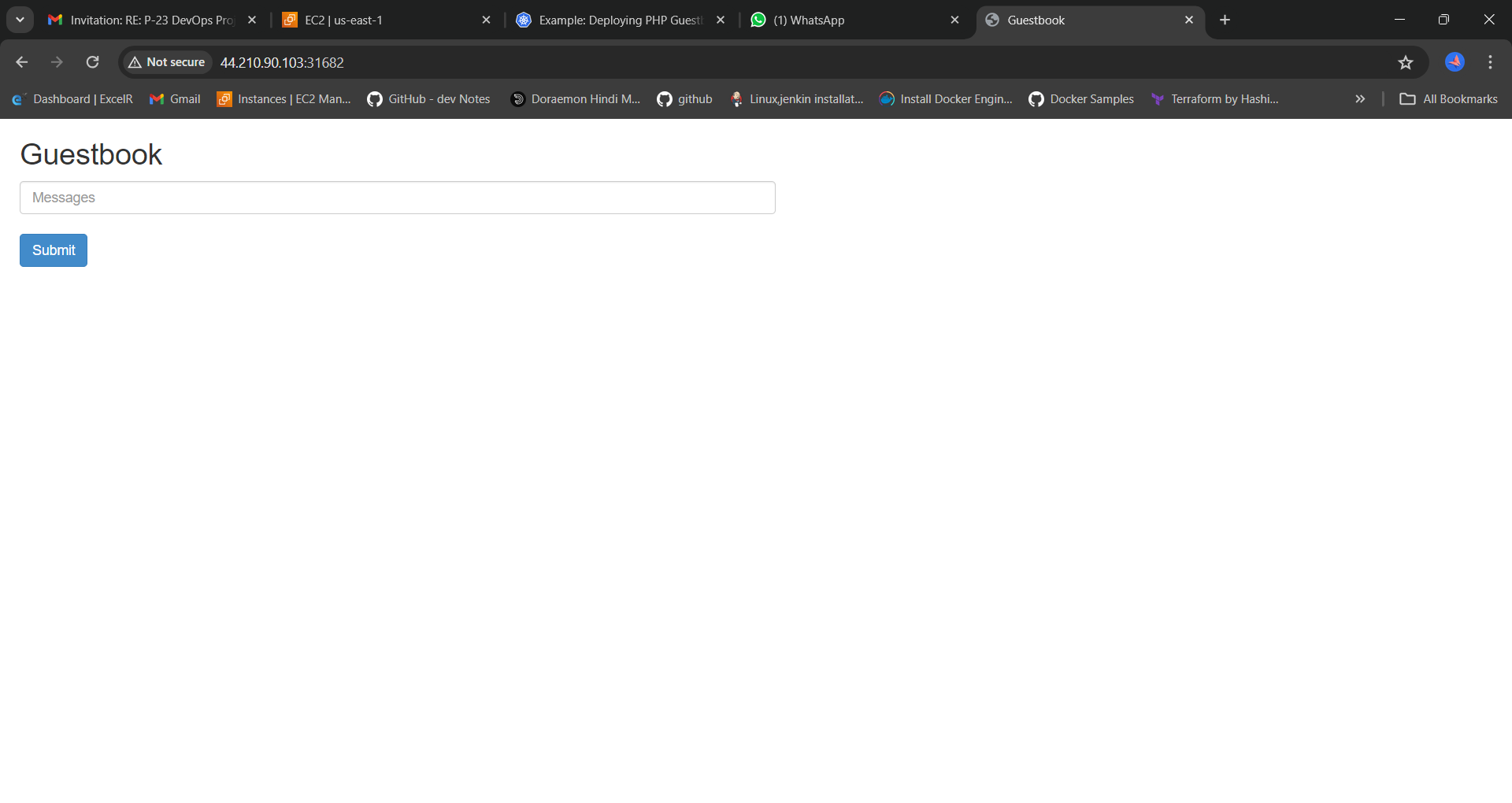
Viewing the Frontend Service by using publicip:nodeport

Here it is 100.25.17.69:31862

Master.



Node1:



Node2:

