Are you trying to set up new k8s cluster environment lab using VMs or on any of cloud vendors like GCP, AWS, Microsoft Azure. This could act as a simple reference document for your hands-on exercise.

Document Reference to create k8s cluster against different options:

1. Using Vagrant

Spin up 3 ubuntu VMs using below vagrant file on our local

a.Goto Document create folder k8s .

b. Under k8s folder create vagrantfile with below conente and save it

Vagrant.configure("2") do |config|

config.vm.define "controlplane" do |controlplane|

controlplane.vm.box = "ubuntu/jammy64"

controlplane.vm.network "private\_network", ip: "192.168.32.10"

controlplane.vm.hostname = "controlplane"

controlplane.vm.provider "virtualbox" do |vb|

vb.memory = "2048"

vb.cpus = 2

end

end

config.vm.define "node01" do |node01|

node01.vm.box = "ubuntu/jammy64"

node01.vm.network "private\_network", ip: "192.168.32.11"

node01.vm.hostname = "node01"

node01.vm.provider "virtualbox" do |vb|

vb.memory = "1024"

vb.cpus = 2

end

end

config.vm.define "node02" do |node02|

node02.vm.box = "ubuntu/jammy64"

node02.vm.network "private\_network", ip: "192.168.32.12"

node02.vm.hostname = "node02"

node02.vm.provider "virtualbox" do |vb|

vb.memory = "1024"

vb.cpus = 2

end

end

end

c. Goto CLI and run below commands

$vagrant status

$vagrant up

$vagrant ssh controlplane

d. Inside control plane (master node) perform below steps

Reference: https://kubernetes.io/docs/setup/production-environment/container-runtimes/

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

overlay

br\_netfilter

EOF

    2 sudo modprobe overlay

    3  sudo modprobe br\_netfilter

    4  # sysctl params required by setup, params persist across reboots

    5  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

    6  # Apply sysctl params without reboot

    7  sudo sysctl --system

    8  # Add Docker's official GPG key:

    9  sudo apt-get update

   10  sudo apt-get install ca-certificates curl gnupg

   11  sudo install -m 0755 -d /etc/apt/keyrings

   12  curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

   13  sudo chmod a+r /etc/apt/keyrings/docker.gpg

   14  # Add the repository to Apt sources:

   15  echo   "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \

  $(. /etc/os-release && echo "$VERSION\_CODENAME") stable" |   sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

   16  sudo apt-get update

   17  sudo apt-get install containerd -y

   18  sudo su

* mkdir -p /etc/containerd
* containerd config default > /etc/containerd/config.toml
* # edit this portion in config.toml
* cat

 [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]

  ...

  [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]

    SystemdCgroup = true

* systemctl restart containerd
* systemctl enable containerd

   19  sudo apt-get update

   20  # apt-transport-https may be a dummy package; if so, you can skip that package

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

   22  curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

   23  echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

   24  sudo apt-get update

   25  sudo apt-get install -y kubelet kubeadm kubectl

   26  sudo apt-mark hold kubelet kubeadm kubectl

   27 sudo kubeadm init --apiserver-advertise-address <VM private ip> --pod-network-cidr=10.244.0.0/16

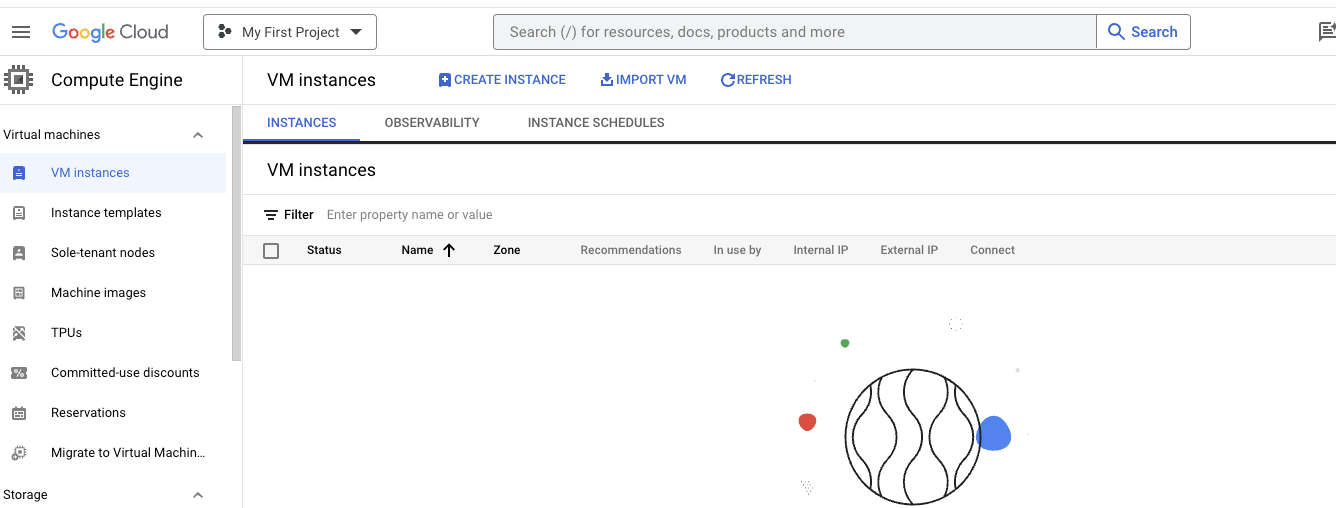
* mkdir -p $HOME/.kube
* sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
* sudo chown $(id -u):$(id -g) $HOME/.kube/config
* kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
* # Now go to other nodes and run the command which will look like the command mentioned below.
* sudo kubeadm join 10.128.0.2:6443 --token 8yq1fw.5w5yjl1ezr1ecbvp \

--discovery-token-ca-cert-hash sha256:c48de90c02e04f76c74e0ded53aa97d7669f4a42624338996671c085119cb41a

* kubectl get nodes
* type \_init\_completion
* echo 'source <(kubectl completion bash)' >>~/.bashrc
* kubectl get nodes
* kubectl get nodes -o wide

e. exit . ssh to node1,node2 and perform same steps excluding kubeadm init command alone instead run kubeadm join command from the worker nodes

1. Using Google Cloud Platform (GCP) – k8s cluster creation using 3 VMs (compute Engine)



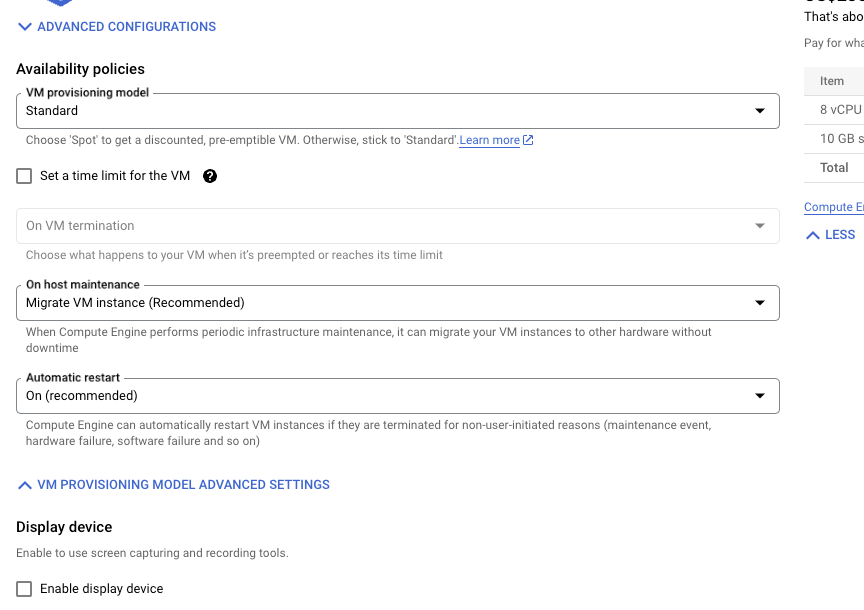
Create Instance ‘controlplane’ with ubuntu image

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

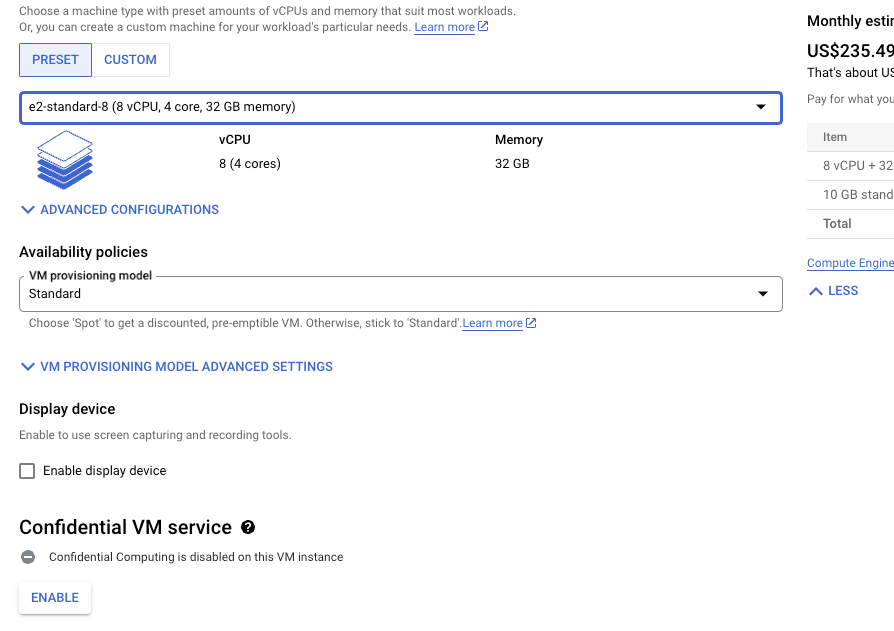
A screenshot of a computer

Description automatically generated

Create node-1,node-2 in similar way

A screenshot of a computer

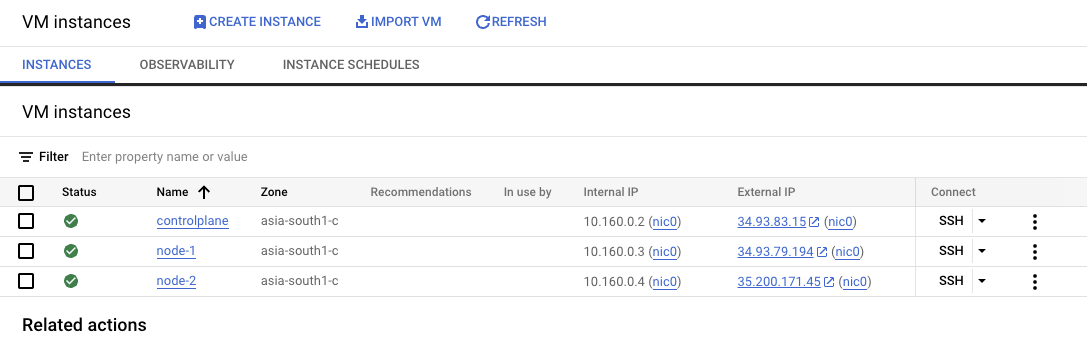
Description automatically generated



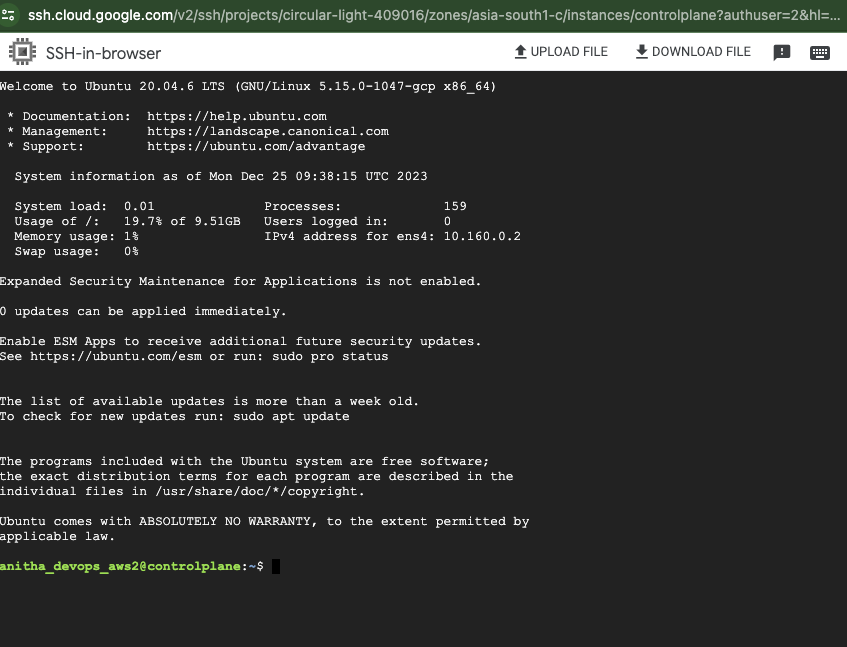
A screenshot of a computer

Description automatically generated

We have 3 VMs up and running in GCP.



Goto the list and select ssh option under controlplane node and perform steps required for k8s cluster installation as in k8s documentation



Steps performed are as below:

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

overlay

br\_netfilter

EOF

2. sudo modprobe overlay

sudo modprobe br\_netfilter

3. *# 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

4. *# Apply sysctl params without reboot*

$ sudo sysctl --system

5. Verify that the br\_netfilter, overlay modules are loaded by running the following commands:

lsmod | grep br\_netfilter

lsmod | grep overlay

6. Verify that the net.bridge.bridge-nf-call-iptables, net.bridge.bridge-nf-call-ip6tables, and net.ipv4.ip\_forward system variables are set to 1 in your sysctl config by running the following command:

$sysctl net.bridge.bridge-nf-call-iptables net.bridge.bridge-nf-call-ip6tables net.ipv4.ip\_forward

7. <https://github.com/containerd/containerd/blob/main/docs/getting-started.md>

Using option 2 : From apt-get or dnf set up containerd runtime

<https://docs.docker.com/engine/install/ubuntu/>

# Add Docker's official GPG key:

sudo apt-get update

sudo apt-get install ca-certificates curl gnupg

sudo install -m 0755 -d /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Add the repository to Apt sources:

$echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \

$(. /etc/os-release && echo "$VERSION\_CODENAME") stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

$sudo apt-get update

$sudo apt-get install containerd -y

$sudo su

$mkdir -p /etc/containerd

$containerd config default > /etc/containerd/config.toml

$ vi /etc/containerd/config.toml

* # edit this portion in config.toml
* cat

 [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]

  ...

  [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]

    SystemdCgroup = true

$ systemctl restart containerd

$ systemctl enable containerd

Install kubeadm

<https://v1-28.docs.kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>

a.Update the apt package index and install packages needed to use the Kubernetes apt repository:

$sudo apt-get update

*# apt-transport-https may be a dummy package; if so, you can skip that package*

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

b.Download the public signing key for the Kubernetes package repositories. The same signing key is used for all repositories so you can disregard the version in the URL:

$curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

c.Add the appropriate Kubernetes apt repository. Please note that this repository have packages only for Kubernetes 1.28; for other Kubernetes minor versions, you need to change the Kubernetes minor version in the URL to match your desired minor version (you should also check that you are reading the documentation for the version of Kubernetes that you plan to install).

*# This overwrites any existing configuration in /etc/apt/sources.list.d/kubernetes.list*

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

e.Update the apt package index, install kubelet, kubeadm and kubectl, and pin their version:

$sudo apt-get update

$sudo apt-get install -y kubelet kubeadm kubectl

$sudo apt-mark hold kubelet kubeadm kubectl

Using kubeadm create cluster

<https://v1-28.docs.kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/>

$ ip route show – to see ip of VM

$sudo kubeadm init --apiserver-advertise-address 10.160.0.2(VMs internal IP) --pod-network-cidr=10.244.0.0/16

Note down join command in command prompt,same to be applied on worker node

kubeadm join 10.160.0.2:6443 --token tcpqd2.slfqllrwwn36yon8 \

--discovery-token-ca-cert-hash sha256:1d007b018abe02e28de066971ff65d772014efd32527bb1b6848b42dfcfa1106

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

You should now deploy a Pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:

/docs/concepts/cluster-administration/addons/

Apply Network Interface – we are using Flannel as below:

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

You can now join any number of machines by running the following on each node

as root:

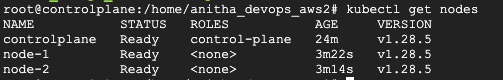
kubeadm join <control-plane-host>:<control-plane-port> --token <token> --discovery-token-ca-cert-hash sha256:<hash>

$kubectl get nodes

<This command will display controlplane/master node with status ‘Ready’>

$kubectl get nodes -o wide

Goto Node 1, Node2 and perform same sequence of steps except kube admin cluster creation – perform kubeadm join



root@controlplane:/home/anitha\_devops\_aws2# kubectl get nodes

NAME STATUS ROLES AGE VERSION

controlplane Ready control-plane 24m v1.28.5

node-1 Ready <none> 3m22s v1.28.5

node-2 Ready <none> 3m14s v1.28.5

root@controlplane:/home/anitha\_devops\_aws2# kubectl get nodes -o wide

NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VERSION CONTAINER-RUNTIME

controlplane Ready control-plane 25m v1.28.5 10.160.0.2 <none> Ubuntu 20.04.6 LTS 5.15.0-1047-gcp containerd://1.7.2

node-1 Ready <none> 3m53s v1.28.5 10.160.0.3 <none> Ubuntu 20.04.6 LTS 5.15.0-1047-gcp containerd://1.7.2

node-2 Ready <none> 3m45s v1.28.5 10.160.0.4 <none> Ubuntu 20.04.6 LTS 5.15.0-1047-gcp containerd://1.7.2

root@controlplane:/home/anitha\_devops\_aws2# kubectl create deploy nginx --image=nginx --replicas=2

deployment.apps/nginx created

root@controlplane:/home/anitha\_devops\_aws2# kubectl get all

NAME READY STATUS RESTARTS AGE

pod/nginx-7854ff8877-cc8fx 0/1 ContainerCreating 0 5s

pod/nginx-7854ff8877-tpr7r 0/1 ContainerCreating 0 5s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

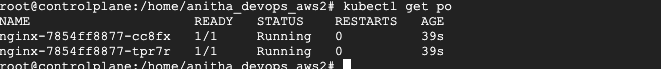
service/kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 25m

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/nginx 0/2 2 0 5s

NAME DESIRED CURRENT READY AGE

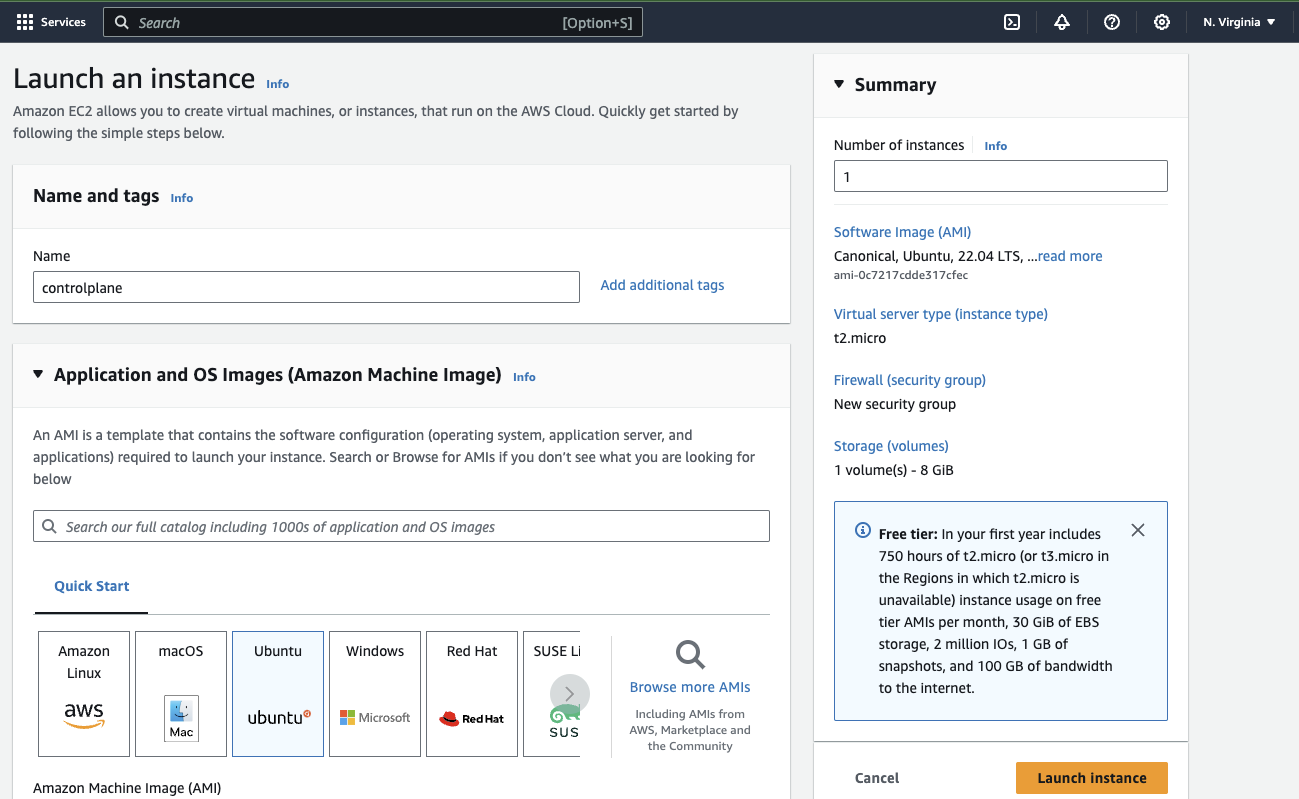
replicaset.apps/nginx-7854ff8877 2 2 0 5s



STOP all 3 VMs and terminate to avoid billing

3.Using Amazon webservices cloud – AWS :

* 1. create 3 EC2 machine with ubuntu OS and perform same steps as performed under GCP nodes



A screenshot of a computer

Description automatically generated

3 VMs created under N.Virginia Region us-east-1a Availability Zone

A screenshot of a computer

Description automatically generated

[ERROR NumCPU]: the number of available CPUs 1 is less than the required 2

[ERROR Mem]: the system RAM (949 MB) is less than the minimum 1700 MB

1. Using Microsoft Azure

Create 2 Machines under Microsoft Azure and perform same above steps as performed under GCP