Name : Nidhi Rijhwani
Subject : Adv Devops Exp No 03

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

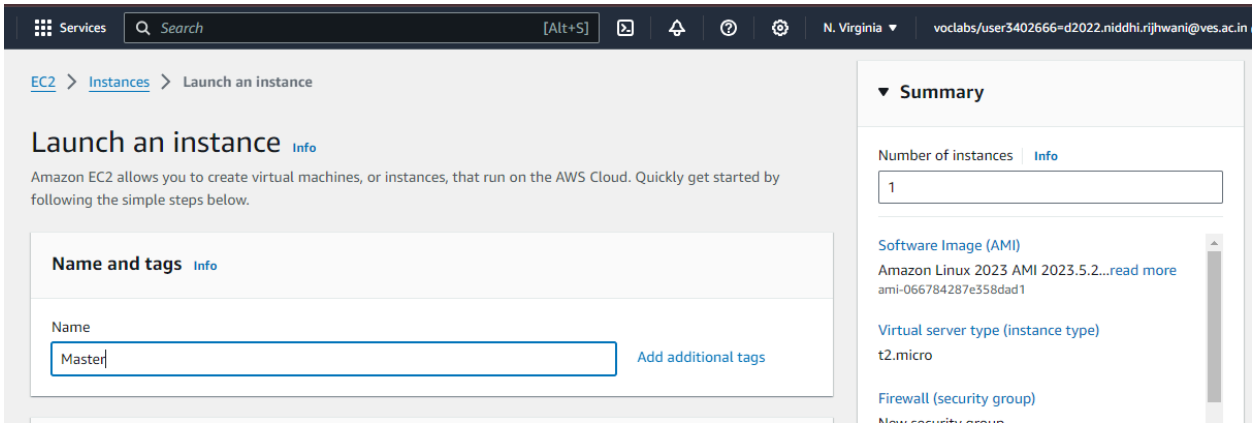
Theory:
Kubernetes is a powerful container orchestration platform that automates the deployment, scaling, and management of containerized applications. Its architecture consists of a Control Plane (including the API server, controller manager, scheduler, etc) that manages the cluster state, and worker nodes that run applications in pods via agents like kubelet and kube-proxy. To spin up a Kubernetes cluster on cloud platforms like AWS, Google Cloud, or Azure, you typically use managed services like EKS, GKE, or AKS, where you can create clusters through their respective management consoles or command-line tools, ensuring seamless scalability and management of your containerized applications.

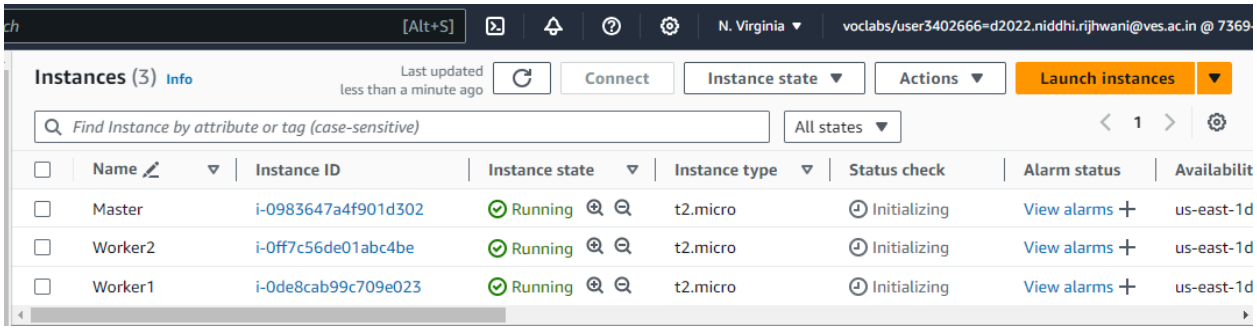
Kubernetes manage the following activities:

- Controlling resource consumption by application or team.
- Evenly spreading application load across a hosting infrastructure.
- Automatically load balancing requests across the different instances of an application.
- Monitoring resource consumption and resource limits to automatically stop applications from consuming too many resources and restarting the applications again.
- Moving an application instance from one host to another if there is a shortage of resources in a host, or if the host dies.

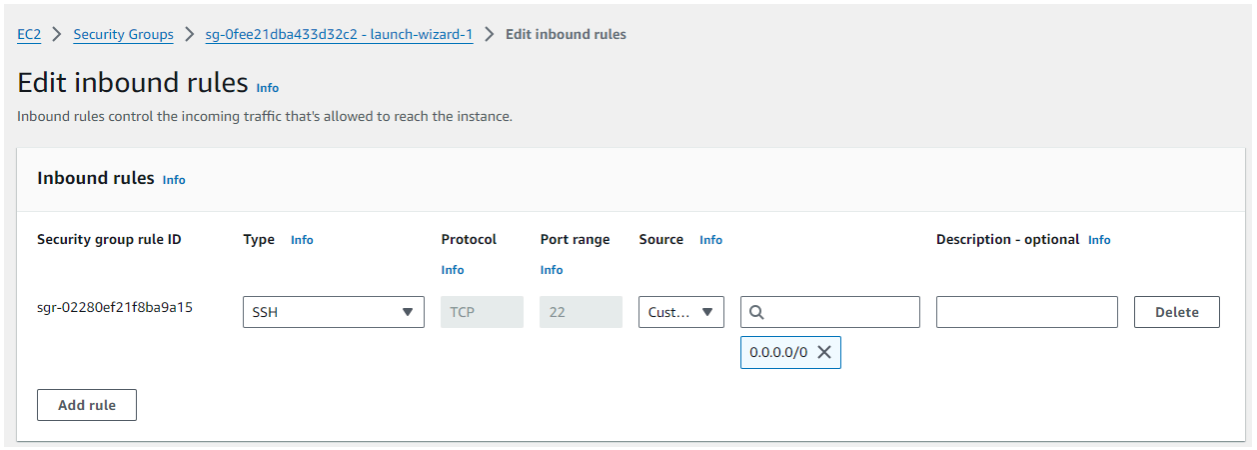
Steps:

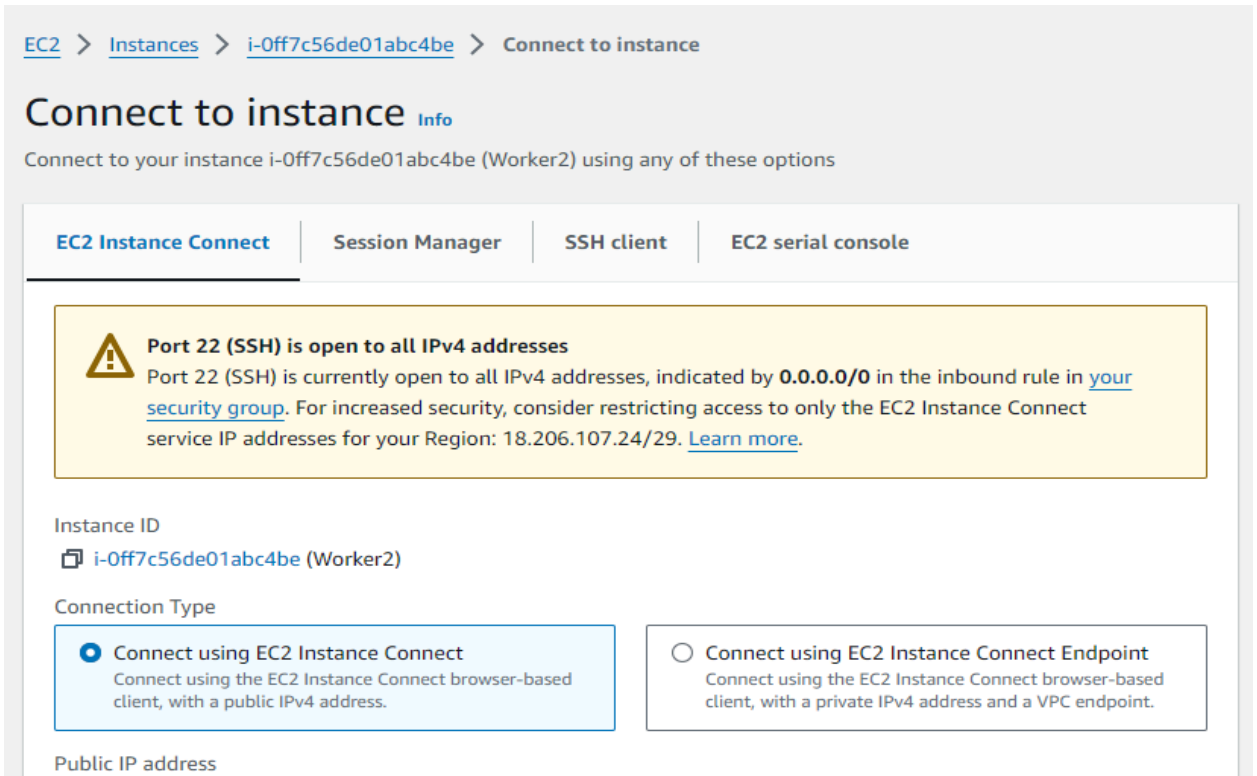
1. **Create 3 EC2 Ubuntu Instances on AWS.**
   (Name 1 as Master, 2nd as Worker1, 3rd as Worker2)





2. Edit the Security Group Inbound Rules to allow SSH





3. **Install Docker**

Add the missing GPG key:

First, manually add the missing GPG key using the following command:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 7EA0A9C3F273FCD8
```

Alternatively, if apt-key is deprecated and not working, you can use gpg to fetch and add the key directly:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | gpg --dearmor | sudo tee /usr/share/keyrings/docker-archive-keyring.gpg > /dev/null
```

Update the Docker repository configuration:

Ensure your Docker repository configuration is using the correct keyring file. The repository configuration should include the signed-by option pointing to the keyring file:

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Update your package index and install Docker:

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Then, configure cgroup in a daemon.json file.

```
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
```

Install Kubernetes on all 3 machines

Download the GPG key and save it to a file:

```
curl -fsSL
https://packages.cloud.google.com/apt/doc/apt-key.gpg | gpg
--dearmor | sudo tee /usr/share/keyrings/cloud-google.gpg >
/dev/null
```

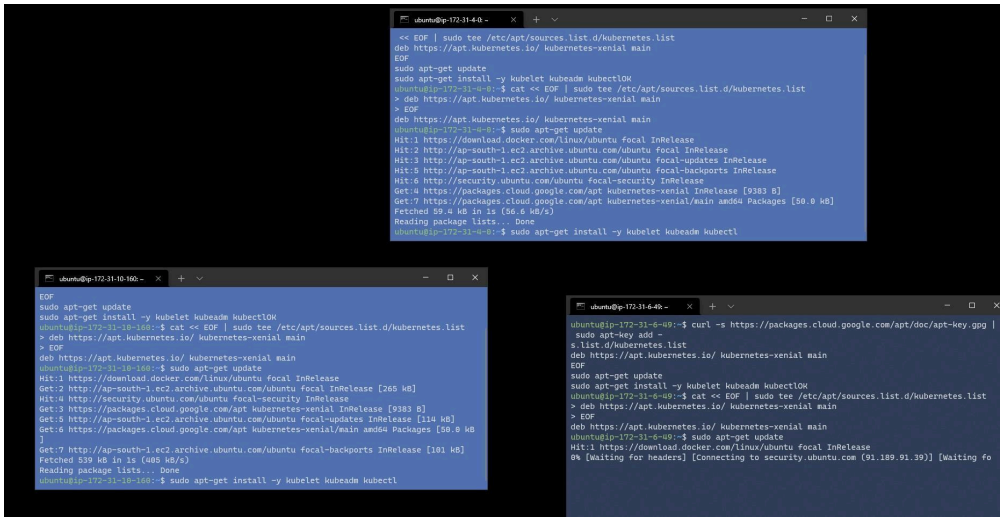
Add the Google Cloud repository to your sources list using the signed-by option:
Create a new file for the Google Cloud repository or update an existing one. For example, you can create a file called `/etc/apt/sources.list.d/google-cloud.list`:

```
echo "deb [arch=amd64
signed-by=/usr/share/keyrings/cloud-google.gpg]
https://packages.cloud.google.com/apt cloud-sdk main" | sudo
tee /etc/apt/sources.list.d/google-cloud.list > /dev/null
```

Update your package index and install the desired packages:

```
sudo apt-get update
sudo apt-get install google-cloud-sdk
```

```
cat << EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
```

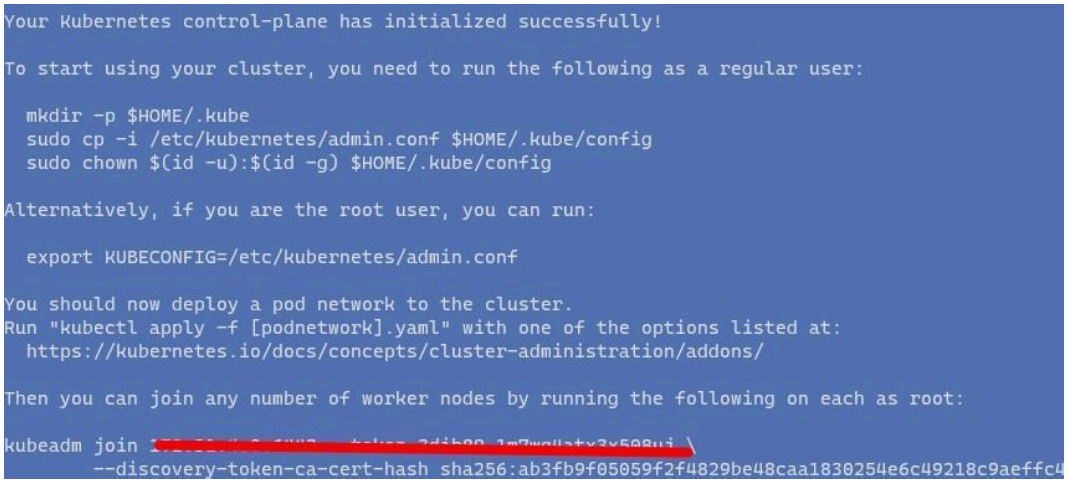


After installing Kubernetes, we need to configure internet options to allow bridging.

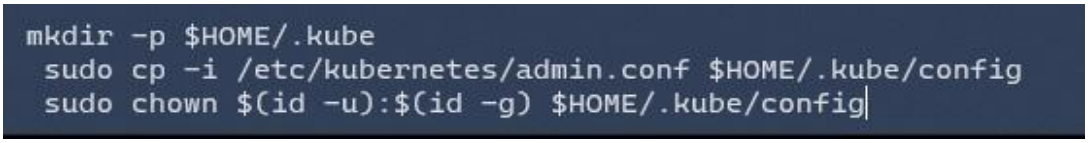
sudo swapoff -a
echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /etc/sysctl.conf
sudo sysctl -p

4. **Perform this ONLY on the Master Machine**

Initialize the Kubecluster
sudo kubeadm init –pod-network-cider=10.244.0.0/16 –ignore-preflight-errors=all



Copy the join command and keep it in a notepad, we'll need it later.
Copy the mkdir and chown commands from the top and execute them.



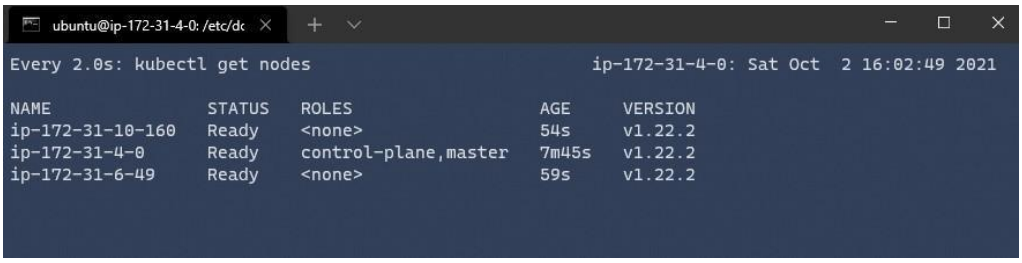
Then, add a common networking plugin called flamel file as mentioned in the code.
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

Check the created pod using this command.
Now, keep a watch on all nodes using the following command

- watch kubectl get nodes

5. **Perform this ONLY on the worker machines**

sudo kubeadm join <ip> --token <token> \ --discovery-token-ca-cert-hash <hash>
Now, notice the changes on the master terminal



That's it, we now have a Kubernetes cluster running across 3 AWS EC2 Instances. This cluster can be used to further deploy applications and their loads being distributed across these machines.

Conclusion:

In this experiment, we learned how to install Kubernetes, create a Kubernetes Cluster in AWS EC2 instances and get them up and running.