Advanced DevOps Lab
Experiment 4

Aim: To install Kubectl and execute Kubectl commands to manage the Kubernetes cluster and deploy Your First Kubernetes Application.

Theory:

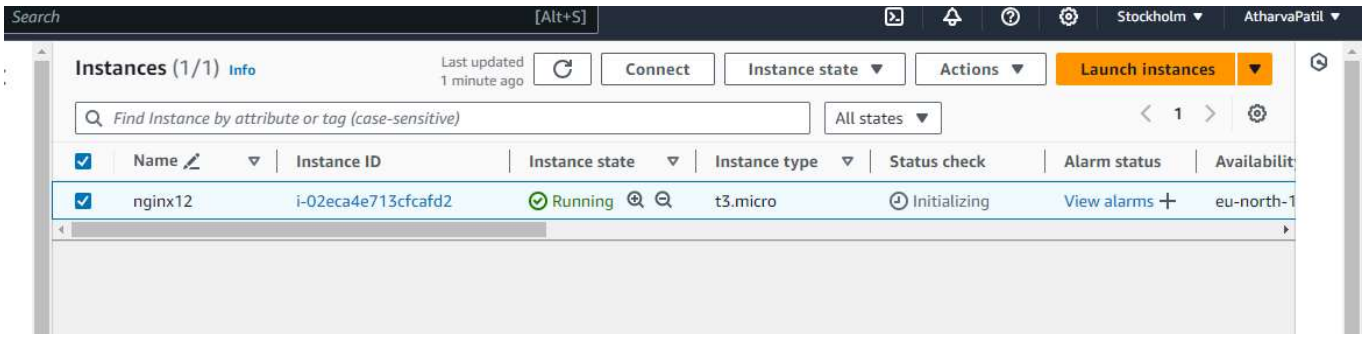
Originally developed by Google, Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. In fact, Kubernetes has established itself as the defacto standard for container orchestration and is the flagship project of the Cloud Native Computing Foundation (CNCF), backed by key players like Google, AWS, Microsoft, IBM, Intel, Cisco, and Red Hat.

Kubernetes Deployment

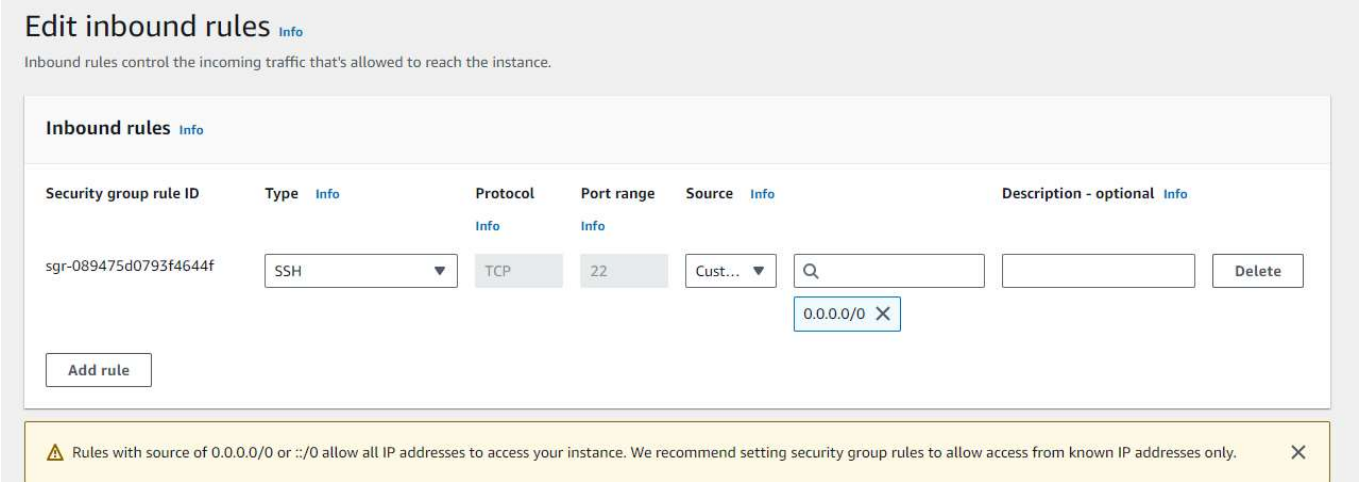
A Kubernetes Deployment is used to tell Kubernetes how to create or modify instances of the pods that hold a containerized application. Deployments can scale the number of replica pods, enable the rollout of updated code in a controlled manner, or roll back to an earlier deployment version if necessary.

Steps:

1. Create an EC2 Ubuntu Instance on AWS.



2. Edit the Security Group Inbound Rules to allow SSH



3. SSH into the machine

ssh -i <keyname>.pem ubuntu@<public_ip_address>

4. Install Docker

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
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
```

```
Setting up libltdl7:amd64 (2.4.7-7build1) ...
Setting up docker-ce-cli (5:27.2.1-1~ubuntu.24.04~noble) ...
Setting up libslirp0:amd64 (4.7.0-1ubuntu3) ...
Setting up pigz (2.8-1) ...
Setting up docker-ce-rootless-extras (5:27.2.1-1~ubuntu.24.04~noble) ...
Setting up slirp4netns (1.2.1-1build2) ...
Setting up docker-ce (5:27.2.1-1~ubuntu.24.04~noble) ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /usr/lib/systemd/system/docker.socket.
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.2) ...
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-172-31-39-158:~$
```

Then, configure cgroup in a daemon.json file.

```
cd /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"]
}
EOF
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
```

```
ubuntu@ip-172-31-38-181:~$ cd /etc/docker
ubuntu@ip-172-31-38-181:/etc/docker$ cat <<EOF | sudo tee /etc/docker/daemon.json
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
EOF
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
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-172-31-38-181:/etc/docker$
```

5. Install Kubernetes

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo
apt-key add -
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
```

```
ec2-user@ip-172-31-24-190 ~ $ kubectl version
Client Version: v1.31.1
Kustomize Version: v5.4.2
```

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
```

6. Initialize the Kubecluster

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

```
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

Alternatively, if you are the root user, you can run:

  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 172.31.24.190:6443 --token xsbsq1.6ro11sawnvttbsvu \
        --discovery-token-ca-cert-hash sha256:10d2b67f4f4749b51054065a554c74e6a956e4782d9ab4bb79b8591648b3edef
ec2-user@ip-172-31-24-190 ~ $ kubectl get nodes
```

Copy the mkdir and chown commands from the top and execute them

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

Then, add a common networking plugin called flannel as mentioned in the code.

```
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/
k ube-flannel.yml
```

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
```

```
ec2-user@ip-172-31-24-190 ~ $ kubectl apply -f https://k8s.io/examples/application/deployment.yaml
deployment.apps/nginx-deployment created
```

Use ‘kubectl get pods’ to verify if the deployment was properly created and the pod is working correctly.

```
ec2-user@ip-172-31-24-190 ~ $ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-d556bf558-mwd8p    0/1     Pending   0          7s
nginx-deployment-d556bf558-zc25s    0/1     Pending   0          7s
```

Next up, create a name alias for this pod.

```
POD_NAME=$(kubectl get pods -l app=nginx -o
jsonpath="{.items[0].metadata.name}")
```

8. Lastly, port forward the deployment to your localhost so that you can view it.

```
kubectl port-forward $POD_NAME 8080:80
```

9. 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
```

```
ec2-user@ip-172-31-24-190 ~ $ curl --head http://127.0.0.1:8080
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Sat, 14 Sep 2024 06:54:21 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
```

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: After installing docker and kubernetes on the EC2 Ubuntu instance we initialized the Kube cluster. Nginx is a web server software that is used for its low resource usage and high performance. We can use this as a server to host our server and deploy it on the cluster. After deploying we check if the pod is working correctly and lastly port forward the deployment to our local host. Then we check if the server is running, if the server responds with 200, ok then the deployment was successful.