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:

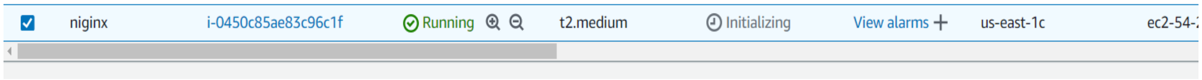
Originally developed by Google, Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications. In fact, Kubernetes has established itself as the defacto standard for container orchestration and is the flagship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

Kubernetes Deployment

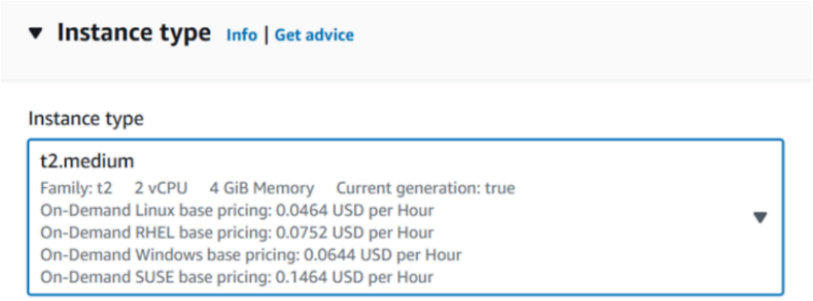
A Kubernetes Deployment is used to tell Kubernetes how to create or modify instances of the pods that hold a containerized application. Deployments can scale the number of replica pods, enable the rollout of updated code in a controlled manner, or roll back to an earlier deployment version if necessary.

Steps:

1. Create an EC2 Ubuntu Instance on AWS.



☑ niginx i-0450c85ae83c96c1f Running t2.medium Initializing View alarms + us-east-1c ec2-54-

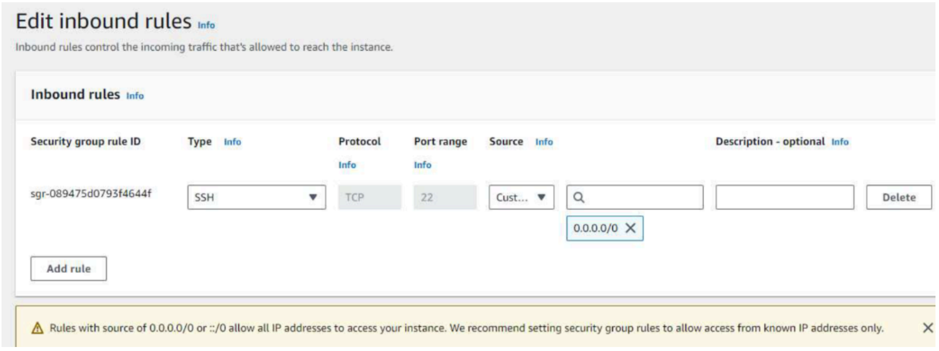


Instance type Info | Get advice

Instance type

t2.medium
Family: t2 2 vCPU 4 GiB Memory Current generation: true
On-Demand Linux base pricing: 0.0464 USD per Hour
On-Demand RHEL base pricing: 0.0752 USD per Hour
On-Demand Windows base pricing: 0.0644 USD per Hour
On-Demand SUSE base pricing: 0.1464 USD per Hour

2. Edit the Security Group Inbound Rules to allow SSH



Edit inbound rules Info

Inbound rules control the incoming traffic that's allowed to reach the instance.

Inbound rules Info

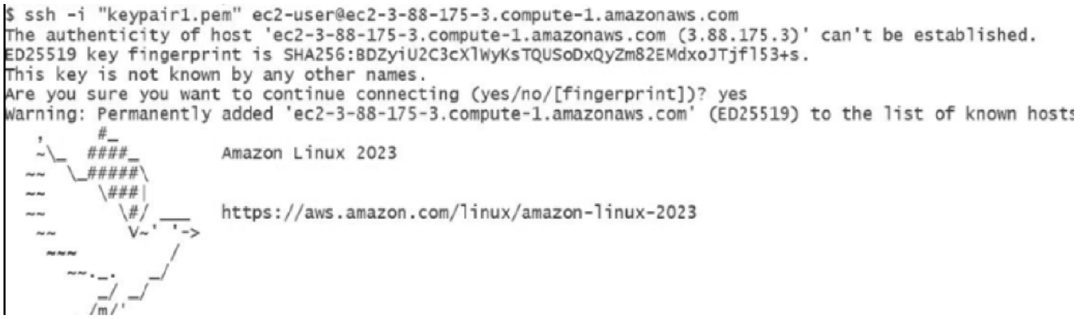
Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info		Description - optional Info	
sgr-089475d0793f4644f	SSH ▼	TCP	22	Cust... ▼	🔍 0.0.0.0/0 ✕		Delete

Add rule

⚠ Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only. ✕

3. SSH into the machine

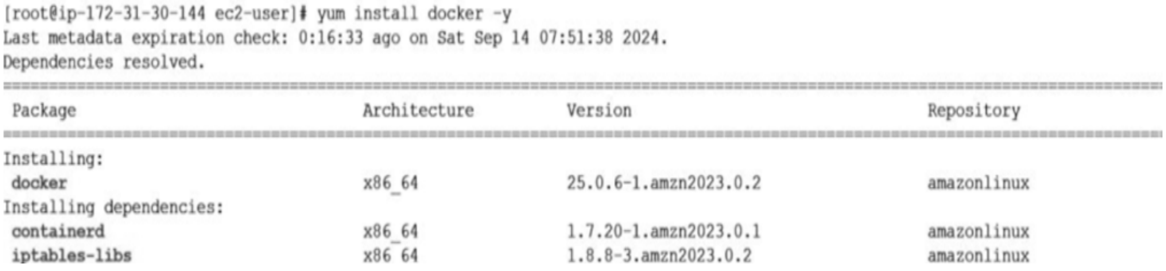
ssh -i <keyname>.pem ubuntu@<public_ip_address>



```
$ ssh -i "keypair1.pem" ec2-user@ec2-3-88-175-3.compute-1.amazonaws.com
The authenticity of host 'ec2-3-88-175-3.compute-1.amazonaws.com (3.88.175.3)' can't be established.
ED25519 key fingerprint is SHA256:BDZyiU2C3cXlWyKsTQUSoDxQyZm82EMdxoJTjf153+s.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-88-175-3.compute-1.amazonaws.com' (ED25519) to the list of known hosts
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
```

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
```



```
[root@ip-172-31-30-144 ec2-user]# yum install docker -y
Last metadata expiration check: 0:16:33 ago on Sat Sep 14 07:51:38 2024.
Dependencies resolved.
=======================================================================================================
 Package                    Architecture        Version                          Repository
=======================================================================================================
Installing:
 docker                     x86_64              25.0.6-1.amzn2023.0.2            amazonlinux
Installing dependencies:
 containerd                 x86_64              1.7.20-1.amzn2023.0.1            amazonlinux
 iptables-libs              x86_64              1.8.8-3.amzn2023.0.2             amazonlinux
```

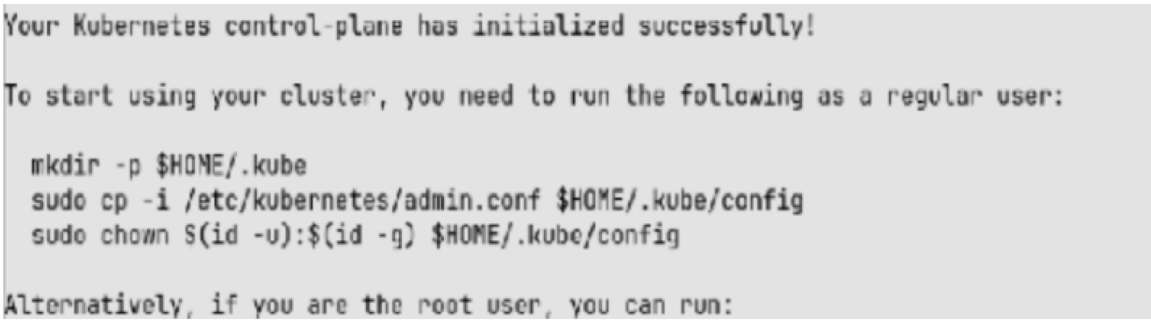
Then, configure cgroup in a daemon.json file.

```
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
```

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
```



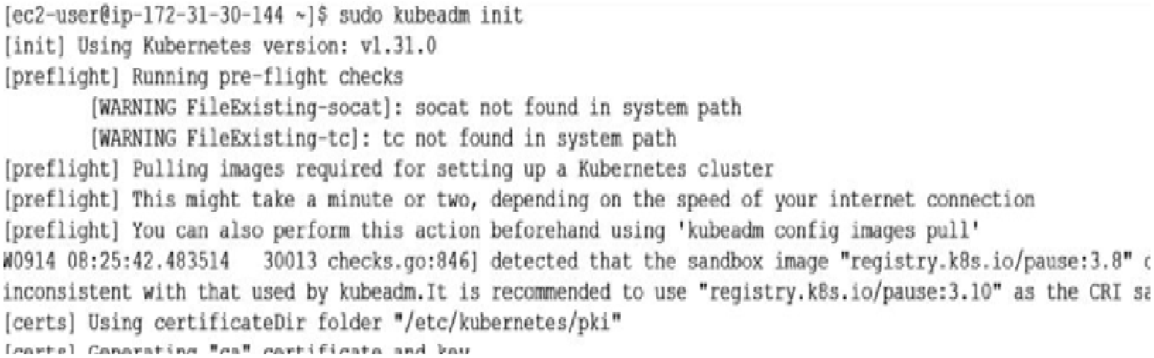
```
Your Kubernetes control-plane has initialized successfully!

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:
```

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
```

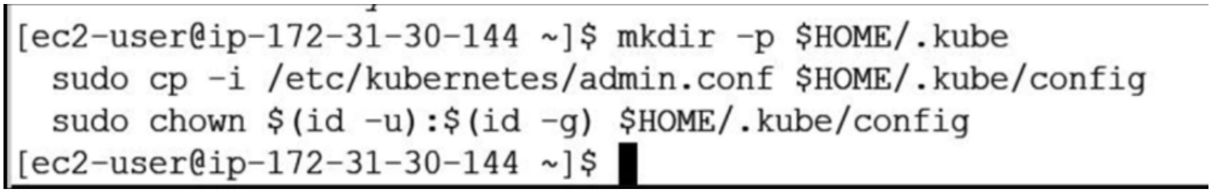
6. Initialize the Kubecluster

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



```
[ec2-user@ip-172-31-30-144 ~]$ sudo kubeadm init
[init] Using Kubernetes version: v1.31.0
[preflight] Running pre-flight checks
        [WARNING FileExisting-socat]: socat not found in system path
        [WARNING FileExisting-tc]: tc not found in system path
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action beforehand using 'kubeadm config images pull'
W0914 08:25:42.483514   30013 checks.go:846] detected that the sandbox image "registry.k8s.io/pause:3.8" c
inconsistent with that used by kubeadm.It is recommended to use "registry.k8s.io/pause:3.10" as the CRI sa
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
```

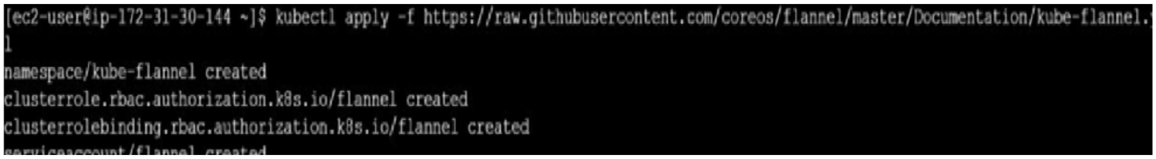
Copy the mkdir and chown commands from the top and execute them



```
[ec2-user@ip-172-31-30-144 ~]$ mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
[ec2-user@ip-172-31-30-144 ~]$
```

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

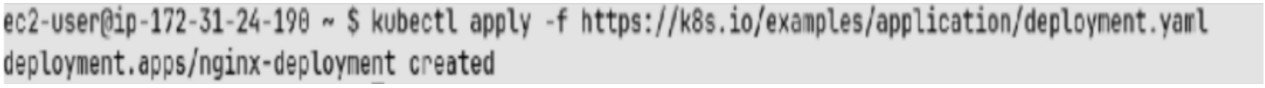
```
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/
k ube-flannel.yml
```



```
[ec2-user@ip-172-31-30-144 ~]$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.
l
namespace/kube-flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
```

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.

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.

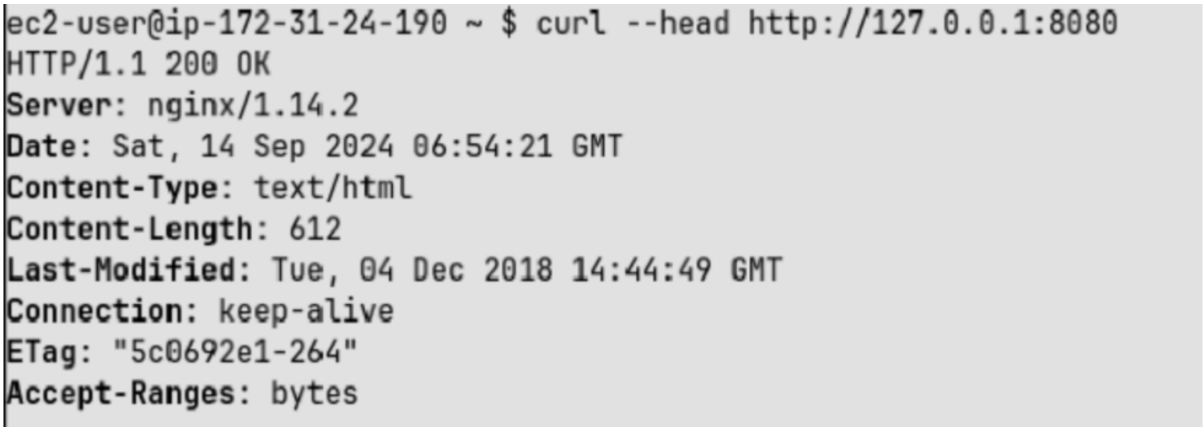
```
kubectl port-forward $POD_NAME 8080:80
```

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: For this experiment, we successfully installed Kubectl, the command-line tool for interacting with Kubernetes clusters, and used it to manage a Kubernetes cluster on AWS. The process involved configuring access to the cluster and performing basic Kubectl commands to verify cluster health and resources. We also deployed our first Kubernetes application, showcasing the simplicity and power of Kubernetes for orchestrating containerized applications.