Advance DevOps

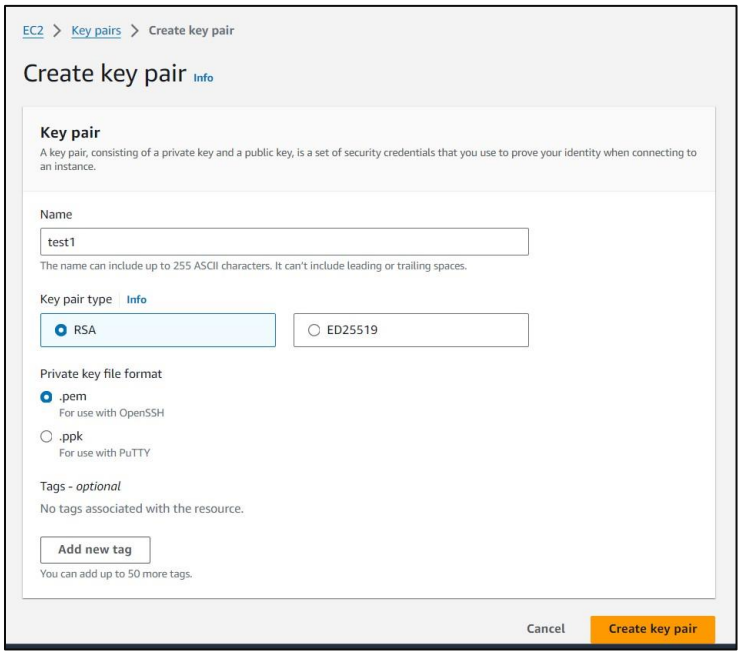
Experiment 4

Aim:

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

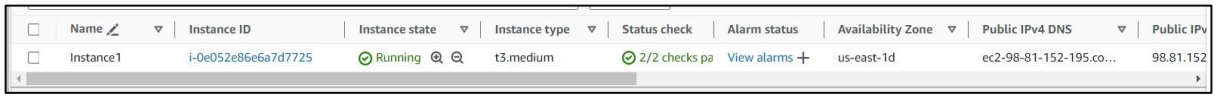
Steps:

1. Create a key pair

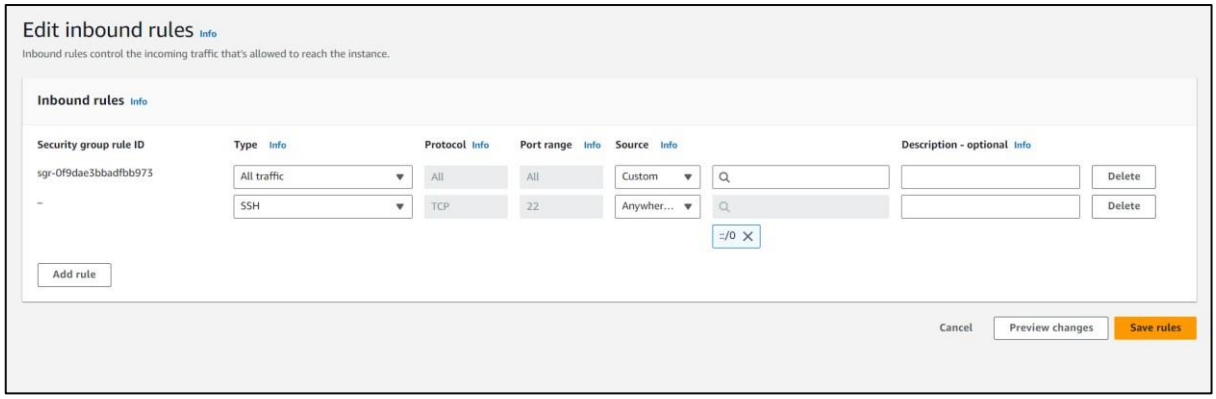


The .pem file will be downloaded on your machine and will be required in the further steps.

2. Now we will create an EC2 Ubuntu instance. Select the key pair which you just created while creating this instance.

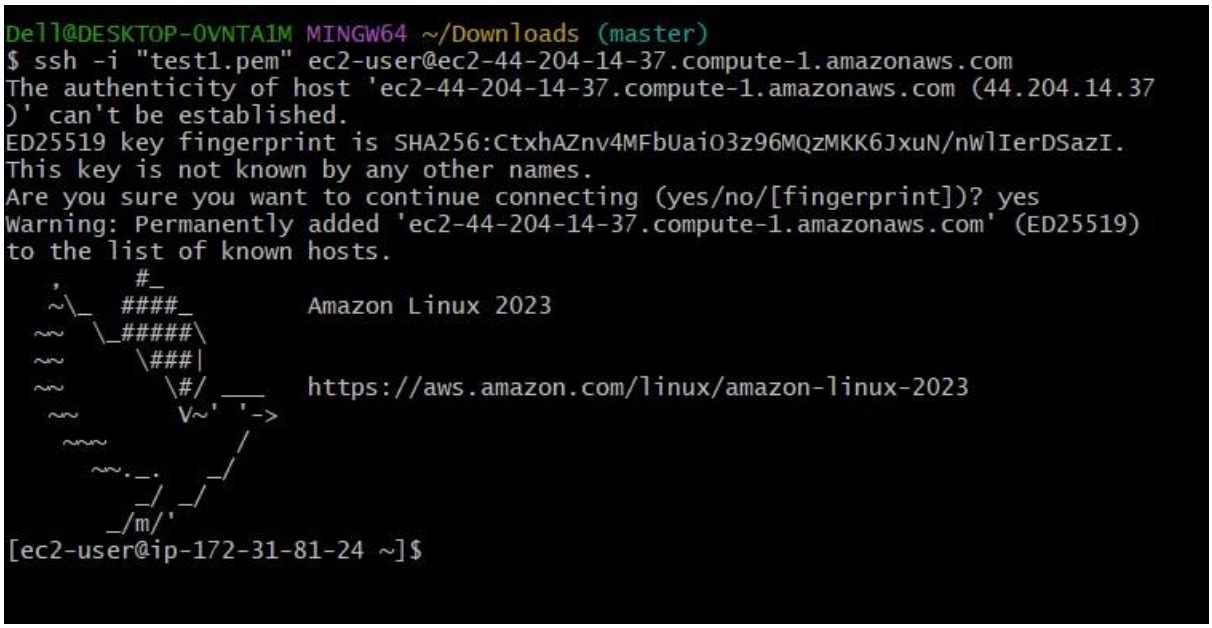


3. Now edit the inbound rules to allow ssh



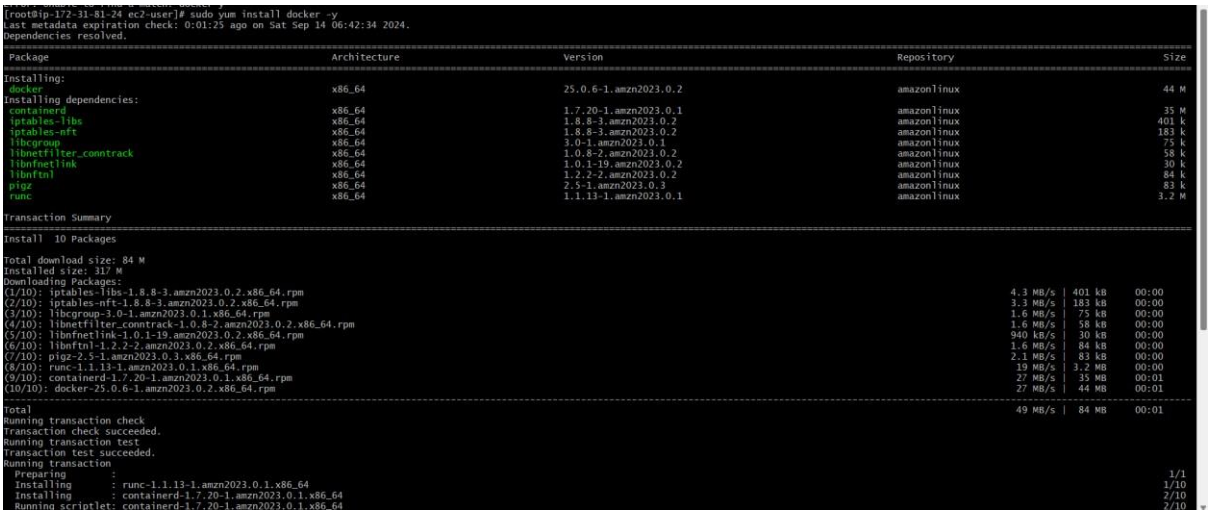
4. Open git bash and go to the directory where pem file is located and use chmod to provide permissions.

5. Now use this command on the terminal: ssh -i <keyname>.pem ubuntu@<public_ip_address> and replace
   - Keyname with the name of your key pair, in our case test1.
   - As we are using amazon Linux instead of ubuntu we will have ec2-user
   - Replace public ip address with its value. Go to your instance and scroll down you will find the public ip address there.



6. Docker installation:

We will be installing docker by using “sudo yum install docker -y”



7. Then to configure cgroup in a daemon json file we will run

cd /etc/docker

cat <<EOF | sudo tee /etc/docker/daemon.json

{

"exec-opts": ["native.cgroupdriver=systemd"]

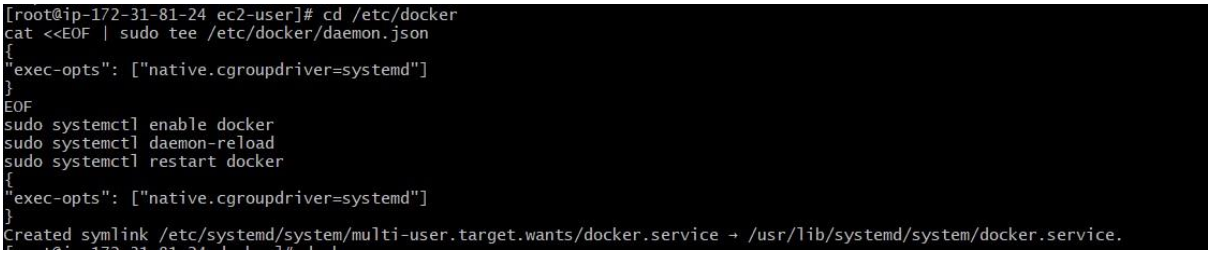
}

EOF

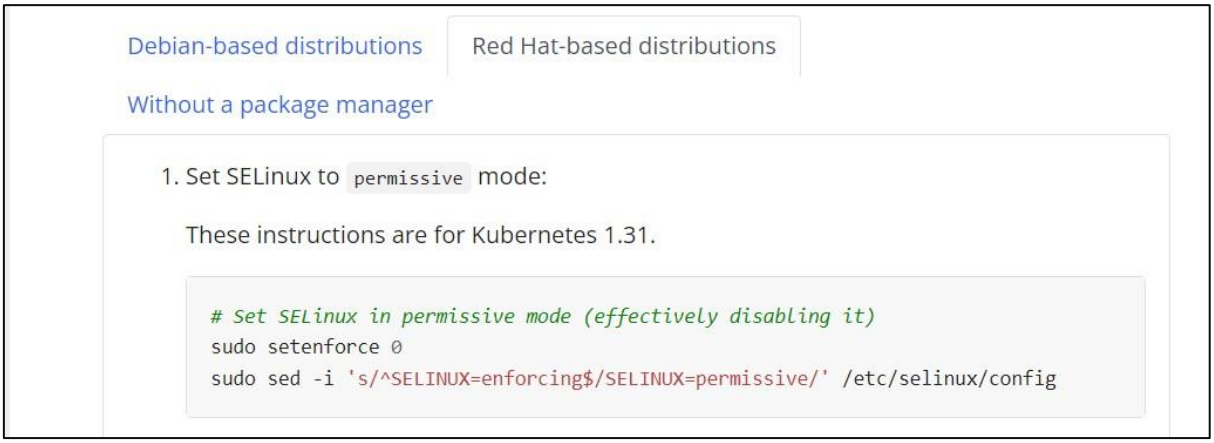
sudo systemctl enable docker sudo

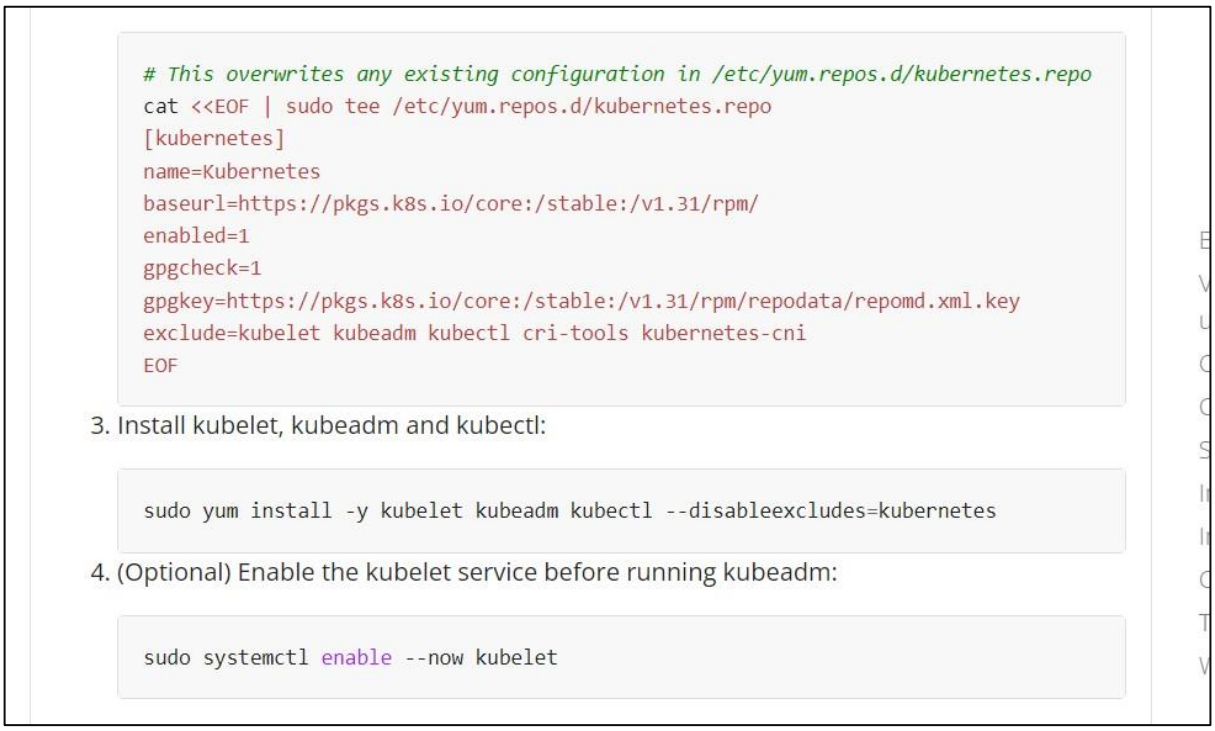
systemctl daemon-reload sudo

systemctl restart docker

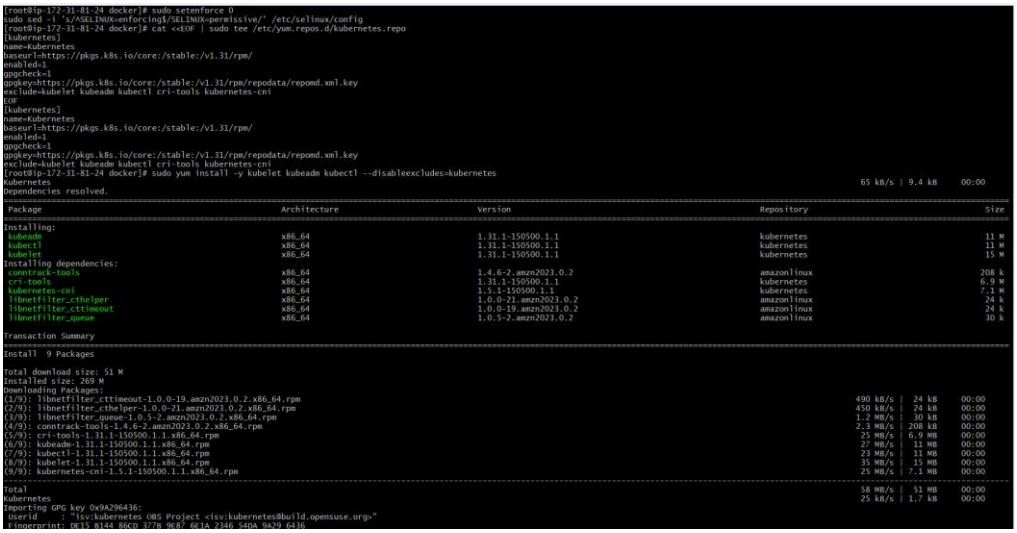


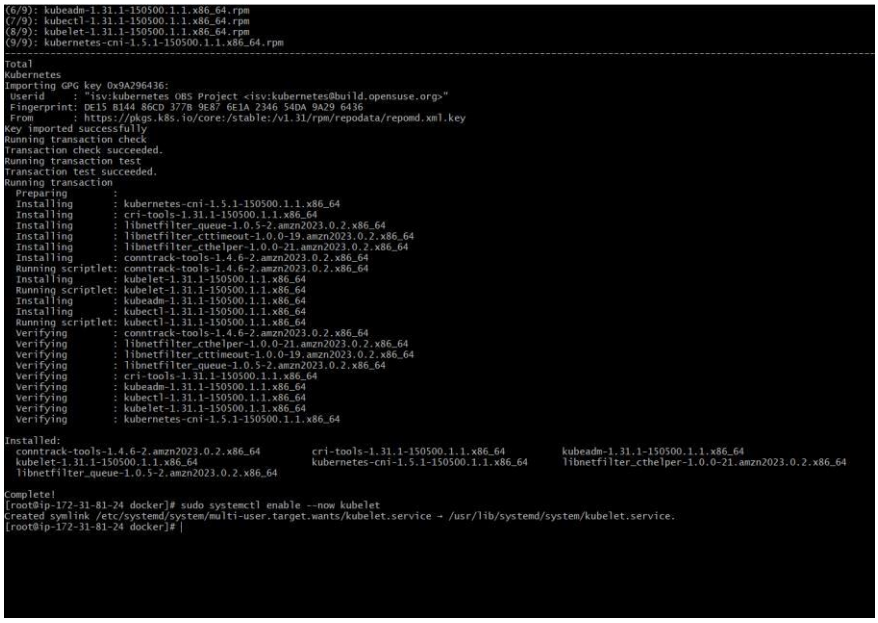
8. Kubernetes installation:
   Search kubeadm installation on your browser and scroll down and select red hat-based distributions.



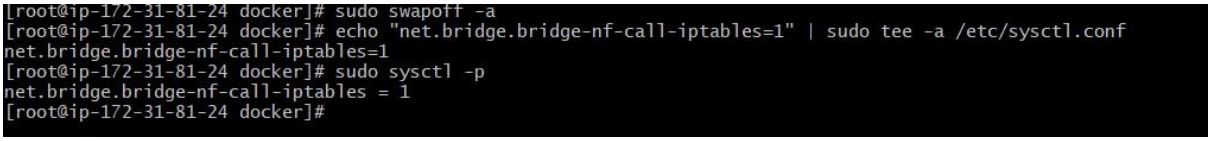


Copy the above given steps and paste in the terminal. This will create a Kubernetes repository, install kubelet,kubeadm and kubectl and also enable the services.





9. 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



10. Initializing kubecluster: sudo kubeadm init --pod-network-cidr=10.244.0.0/16

```
[root@ip-172-31-81-24 docker]# sudo kubeadm init --pod-network-cidr=10.244.0.0/16
[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
error execution phase preflight: [preflight] Some fatal errors occurred:
        [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
[preflight] If you know what you are doing, you can make a check non-fatal with `--ignore-preflight-errors=...`
To see the stack trace of this error execute with --v=5 or higher
[root@ip-172-31-81-24 docker]# sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=NumCPU,Mem
[init] Using Kubernetes version: v1.31.0
```

```
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 172.31.81.24:6443 --token 4a91z3.yz6rwmmkf9yncyd2 \
        --discovery-token-ca-cert-hash sha256:3404bd1bcdd9cf90a003673f622d1672acb4c6ce7c15c4738c80a0a1560fe70d
[root@ip-172-31-81-24 docker]#
```

11. The mkdir command that is generated after initialization has to be copy pasted in the terminal.

```
[root@ip-172-31-81-24 docker]#  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

12. Then, add a common networking plugin called flannel:
    kubectl apply -f
    https://raw.githubusercontent.com/coreos/flannel/master/Documentation/ kube-flannel.yml

```
[root@ip-172-31-81-24 docker]# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
namespace/kube-flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
[root@ip-172-31-81-24 docker]#
```

13. 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

```
[root@ip-172-31-81-24 docker]# kubectl apply -f https://k8s.io/examples/application/deployment.yaml
deployment.apps/nginx-deployment created
[root@ip-172-31-81-24 docker]#
```

14. Use kubectl get pods to check if pod is working correctly

```
[root@ip-172-31-81-24 docker]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-d556bf558-8jdlf    0/1     Pending   0          18s
```

15. To change status from pending to running use following command: kubectl describe pod nginx.

```
[root@ip-172-31-16-56 ~]# kubectl describe pod nginx
Name:             nginx-deployment-d556bf558-gw8v8
Namespace:        default
Priority:         0
Service Account:  default
Node:             <none>
Labels:           app=nginx
                  pod-template-hash=d556bf558
Annotations:      <none>
Status:           Pending
IP:
IPs:              <none>
Controlled By:    ReplicaSet/nginx-deployment-d556bf558
Containers:
  nginx:
    Image:        nginx:1.14.2
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:
```

```
Conditions:
  Type           Status
  PodScheduled   False
Volumes:
  kube-api-access-f9k9s:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type     Reason            Age    From               Message
  ----     ------            ----   ----               -------
  Warning  FailedScheduling  114s   default-scheduler  0/1 nodes are available: 1 node(s) had untolerated taint {node-role.kubernetes.io/control-
: }. preemption: 0/1 nodes are available: 1 Preemption is not helpful for scheduling.
  Warning  FailedScheduling  3m18s  default-scheduler  0/1 nodes are available: 1 node(s) had untolerated taint {node-role.kubernetes.io/control-
: }. preemption: 0/1 nodes are available: 1 Preemption is not helpful for scheduling.
```

Use the below command to remove taints

```
[ec2-user@ip-172-31-26-174 ~]$ kubectl taint nodes --all node-role.kubernetes.io
/control-plane-
node/ip-172-31-26-174.ec2.internal untainted
```

16. Check the pod status

```
[ec2-user@ip-172-31-26-174 ~]$ kubectl get pods
NAME    READY   STATUS    RESTARTS      AGE
nginx   1/1     Running   1 (6s ago)   90s
```

17. port forward the deployment to your localhost so that you can view it.

```
[ec2-user@ip-172-31-26-174 ~]$ kubectl port-forward nginx 8081:80
Forwarding from 127.0.0.1:8081 -> 80
Forwarding from [::1]:8081 -> 80
```

18. 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

    If you see your nginx server name and response code is 200 then the deployment was successful.

Conclusion: In this experiment we created an ec2 instance, enabled ssh by editing the inbound rules. After that we installed docker and Kubernetes and configured internet options to allow bridging. Once this setup got completed, we added a common networking plugin called flannel. Once the cluster started running we deployed nginx server on this cluster and verified deployment.