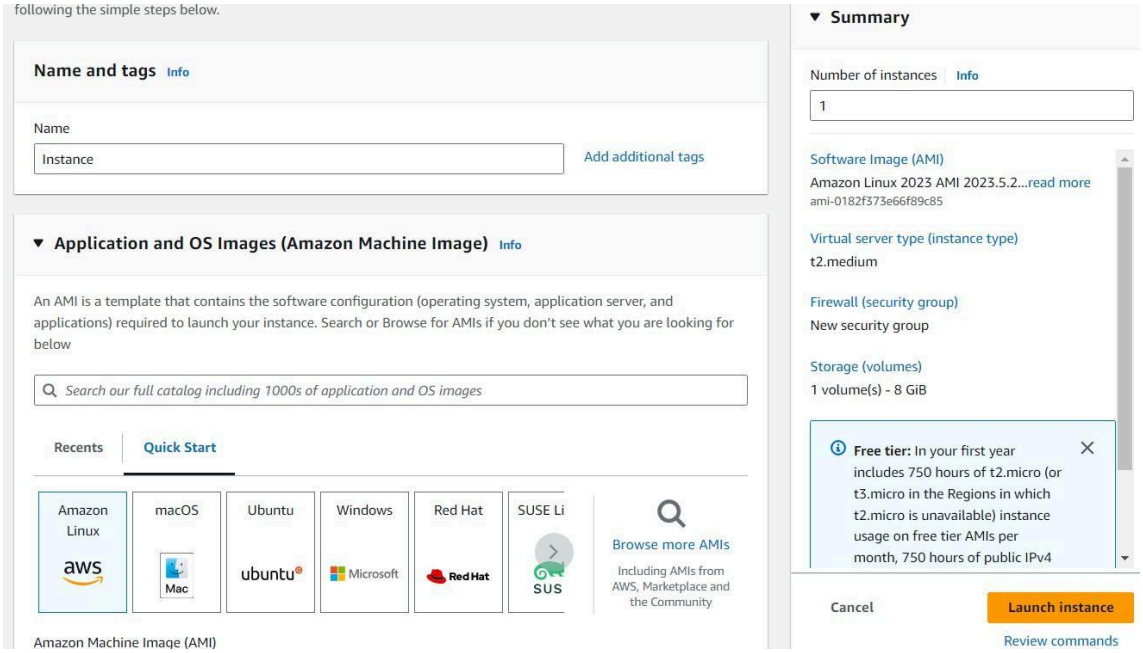
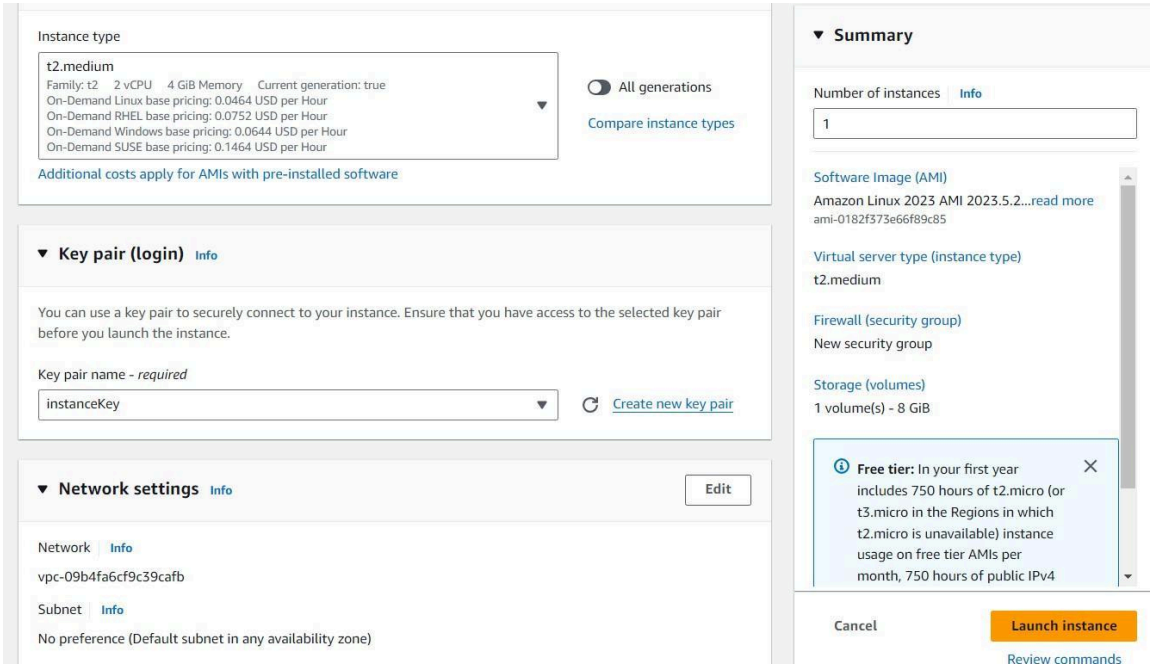
AdvDevops Experiment 4

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

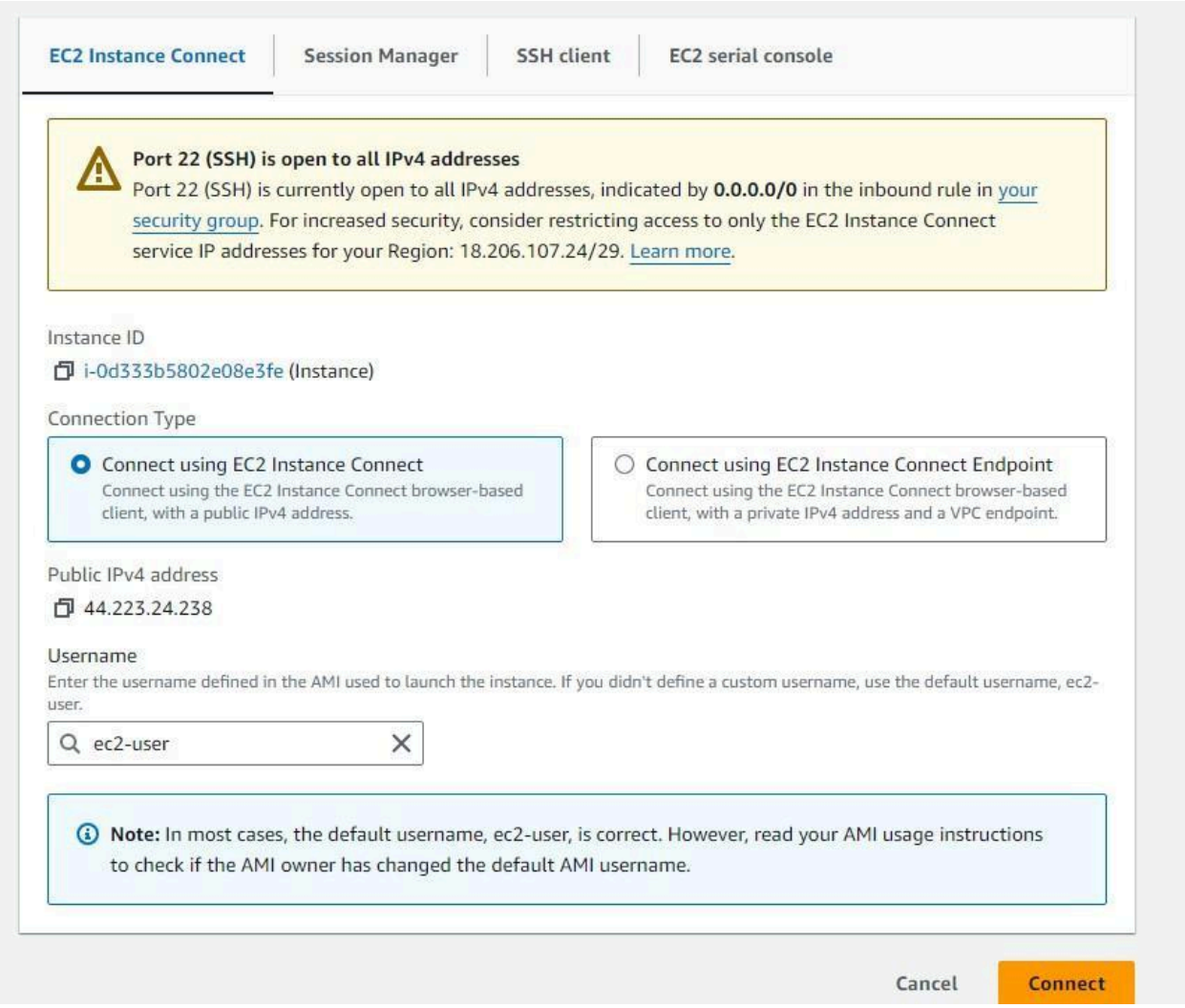
Step 1: Set Up EC2 Instances.



2) Select a key pair, to create one. Click on create.



3) Once created, go back to the instances page. Click on the instance id. Then, click on connect.



Step 2: Installation of Docker

1) Use command ‘sudo su’ This allows you to act as the root user of the terminal

```
[ec2-user@ip-172-31-29-96 ~]$ sudo su
[root@ip-172-31-29-96 ec2-user]#
```

2) We can install docker using yum. Use the command 'yum install docker-y'

```
[root@ip-172-31-29-96 ec2-user]# yum install docker -y
Last metadata expiration check: 0:04:49 ago on Sun Sep 15 10:36:23 2024.
Dependencies resolved.
==========================================================================================================
 Package                                          Architecture                      Version
==========================================================================================================
Installing:
 docker                                           x86_64                            25.0.6-1.amzn2023.0.2
Installing dependencies:
 containerd                                       x86_64                            1.7.20-1.amzn2023.0.1
 iptables-libs                                    x86_64                            1.8.8-3.amzn2023.0.2
 iptables-nft                                     x86_64                            1.8.8-3.amzn2023.0.2
 libcgroup                                        x86_64                            3.0-1.amzn2023.0.1
 libnetfilter_conntrack                           x86_64                            1.0.8-2.amzn2023.0.2
 libnfnetlink                                     x86_64                            1.0.1-19.amzn2023.0.2
 libnftnl                                         x86_64                            1.2.2-2.amzn2023.0.2
 pigz                                             x86_64                            2.5-1.amzn2023.0.3
 runc                                             x86_64                            1.1.13-1.amzn2023.0.1

Transaction Summary
==========================================================================================================
Install  10 Packages

Total download size: 84 M
Installed size: 317 M
Downloading Packages:
                                                                      [===              (1/10): containerd

=== Installing      : docker-25.0.6-1.amzn2023.0.2.x86_64 [======================================  Installin
                                                                                         10/10
  Running scriptlet: docker-25.0.6-1.amzn2023.0.2.x86_64
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /usr/lib/systemd/system/docker.socket.

  Verifying        : containerd-1.7.20-1.amzn2023.0.1.x86_64
  Verifying        : docker-25.0.6-1.amzn2023.0.2.x86_64
  Verifying        : iptables-libs-1.8.8-3.amzn2023.0.2.x86_64
  Verifying        : iptables-nft-1.8.8-3.amzn2023.0.2.x86_64
  Verifying        : libcgroup-3.0-1.amzn2023.0.1.x86_64
  Verifying        : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64
  Verifying        : libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64
  Verifying        : libnftnl-1.2.2-2.amzn2023.0.2.x86_64
  Verifying        : pigz-2.5-1.amzn2023.0.3.x86_64
  Verifying        : runc-1.1.13-1.amzn2023.0.1.x86_64

Installed:
  containerd-1.7.20-1.amzn2023.0.1.x86_64   docker-25.0.6-1.amzn2023.0.2.x86_64                 iptables-libs-1
  libcgroup-3.0-1.amzn2023.0.1.x86_64       libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64   libnfnetlink-1.
  pigz-2.5-1.amzn2023.0.3.x86_64            runc-1.1.13-1.amzn2023.0.1.x86_64

Complete!
[root@ip-172-31-29-96 ec2-user]#
```

3) Now, configure a daemon.json file by using the following chain of commands.

- 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"
 }
```

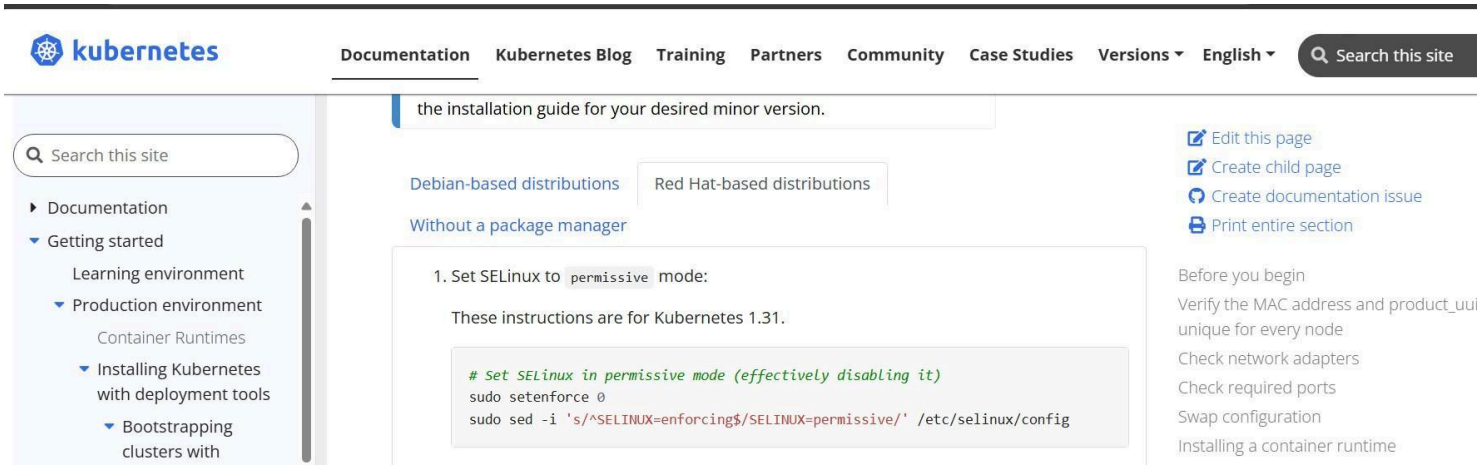
EOF
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker

```
[root@ip-172-31-29-96 docker]# 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
{
"exec-opts": ["native.cgroupdriver=systemd"],
"log-driver": "json-file",
"log-opts": {
"max-size": "100m"
},
"storage-driver": "overlay2"
}
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/
systemd/system/docker.service.
[root@ip-172-31-29-96 docker]#
```

Step 3: Installing Kubernetes

1) For installing kubernetes, we will be using kubeadm, a framework used for creating kubernetes clusters using command line.

https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/



2) Select red hat-based distributions as amazon linux is based on red hat.

sudo setenforce 0
→sets SELinux to permissive mode

sudo sed-i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
→ edits the SELinux configuration file (/etc/selinux/config) to make the change persistent across reboots. If not used, SELinux reverts to enforcing mode after reboot.

Run the following commands:

- sudo setenforce 0
- sudo sed-i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

```
[root@ip-172-31-29-96 docker]# sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

- cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo [kubernetes]
  name=Kubernetes
  baseurl=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/
  enabled=1
  gpgcheck=1
  gpgkey=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/repodata/repomd.xml.key
  exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
  EOF

```
[root@ip-172-31-29-96 docker]# cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
[root@ip-172-31-29-96 docker]#
```

- yum repolist

This command shows the repositories created on the machine.

```
[root@ip-172-31-29-96 docker]# yum repolist
repo id                                                                repo name
amazonlinux                                                            Amazon Linux 2023 repository
kernel-livepatch                                                       Amazon Linux 2023 Kernel Livepatch repository
kubernetes                                                             Kubernetes
[root@ip-172-31-29-96 docker]#
```

Next step is to install kubelet, kubeadm, kubectl

- sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes

```
================================  Installing       : kubectl-1.31.1-150500.1.1.x86_64
                                9/9
  Running scriptlet: kubectl-1.31.1-150500.1.1.x86_64
  Verifying        : conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64
  Verifying        : libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64
  Verifying        : libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64
  Verifying        : libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64
  Verifying        : cri-tools-1.31.1-150500.1.1.x86_64
  Verifying        : kubeadm-1.31.1-150500.1.1.x86_64
  Verifying        : kubectl-1.31.1-150500.1.1.x86_64
  Verifying        : kubelet-1.31.1-150500.1.1.x86_64
  Verifying        : kubernetes-cni-1.5.1-150500.1.1.x86_64

Installed:
  conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64   cri-tools-1.31.1-150500.1.1.x86_64   kubeadm-1.31.1-150500.1.1.x86_64
  kubectl-1.31.1-150500.1.1.x86_64              kubelet-1.31.1-150500.1.1.x86_64     kubernetes-cni-1.5.1-150500.1.1.x86_64
  libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64   libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64   libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64

Complete!
[root@ip-172-31-29-96 docker]#
```

Now, we need to enable the kubelet service. Run the command

- sudo systemctl enable --now kubelet

```
[root@ip-172-31-29-96 docker]# sudo systemctl enable --now kubelet
Created symlink /etc/systemd/system/multi-user.target.wants/kubelet.service → /usr/lib/systemd/syste
m/kubelet.service.
[root@ip-172-31-29-96 docker]#
```

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

```
[root@ip-172-31-29-96 docker]# sudo swapoff -a
echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /etc/sysctl.conf
sudo sysctl -p
net.bridge.bridge-nf-call-iptables=1
net.bridge.bridge-nf-call-iptables = 1
[root@ip-172-31-29-96 docker]#
```

3) Firstly, we need toinitialize kubernetes.For This,run the command:

- sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=NumCPU,Mem

```
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 172.31.29.96:6443 --token 0b6cct.1cm4p25mefy05fhl \
        --discovery-token-ca-cert-hash sha256:ae83caa940837900b62231f4f381a06d69b4d25b0207ce5fff9a943e6757b6a8
[root@ip-172-31-29-96 docker]#
```

4) From The Output,we receive the following commands:

- mkdir-p$HOME/.kube
- sudocp-i /etc/kubernetes/admin.conf$HOME/.kube/config
- sudo

chown$(id-u):$(id-g)$HOME/.kube/config Run These

```
[root@ip-172-31-29-96 docker]# mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
[root@ip-172-31-29-96 docker]#
```

Commands

5) Add a common networking plugin flannel usingthiscommand

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

```
[root@ip-172-31-29-96 docker]# kubectl apply -f https://raw.githubusercontent.com/coreos/flan
nel/master/Documentation/kube-fkubectl apply -f https://raw.githubusercontent.com/coreos/flan
nel/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-172-31-29-96 docker]#
```

Step3:Deploy nginx server

1) Now that the cluster is set,applythedeployment fileofnginxusingthiscommand

- kubectl apply-fhttps://k8s.io/examples/pods/simple-pod.yaml

```
[root@ip-172-31-29-96 docker]# kubectl apply -f https://k8s.io/examples/pods/simple-pod.yaml
pod/nginx created
[root@ip-172-31-29-96 docker]#
```

2) Use The Command

- kubectl get pods

To Get the list of pods in cluster.

```
[root@ip-172-31-29-96 docker]# kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   0/1     Pending   0          44s
[root@ip-172-31-29-96 docker]#
```

This output shows that thepodisina‘PENDING’state, change it to RUNNING state,runtuh following commands.

- kubectl describe pod nginx:Provides Details About Your Pod

This command is used to get details about the pod and potential issues with the pod

```
[root@ip-172-31-29-96 docker]# kubectl describe pod nginx
Name:             nginx
Namespace:        default
Priority:         0
Service Account:  default
Node:             <none>
Labels:           <none>
Annotations:      <none>
Status:           Pending
IP:
IPs:              <none>
Containers:
  nginx:
    Image:        nginx:1.14.2
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
```

```
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  2m43s  default-scheduler  0/1 nodes are available: 1 node(s) had
 untolerated taint {node-role.kubernetes.io/control-plane: }. preemption: 0/1 nodes are avail
able: 1 Preemption is not helpful for scheduling.
  Warning  FailedScheduling  1s     default-scheduler  0/1 nodes are available: 1 node(s) had
 untolerated taint {node-role.kubernetes.io/control-plane: }. preemption: 0/1 nodes are avail
able: 1 Preemption is not helpful for scheduling.
[root@ip-172-31-29-96 docker]#
```

3) From this output, we get to know that the node has some untolerated taint. To remove this, use

- kubectl taintnodes --allnode-role.kubernetes.io/control-plane:NoSchedule-

```
[root@ip-172-31-29-96 docker]# kubectl taint nodes --all node-role.kubernetes.io/control-plane:NoSche
dule-                          kubectl taint nodes --all node-role.kubernetes.io/control-plane:NoSche
dule-
node/ip-172-31-29-96.ec2.internal untainted
```

4) Now, we check the status of the pod by running “kubectl get pods” again

```
[root@ip-172-31-29-96 docker]# kubectl get pods
NAME    READY   STATUS    RESTARTS      AGE
nginx   1/1     Running   2 (68s ago)   9m54s
[root@ip-172-31-29-96 docker]#
```

5) Now, change the port to which you want to host your server on using command

- kubectl port-forward nginx <port number you want to host on>:80

```
[root@ip-172-31-29-96 docker]# kubectl port-forward pod/nginx 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```

6) To check whether the deployment was successful, run the command

- curl--head http://127.0.0.1:<port number given by you>

If the terminal returns a status code of 200, it means that the deployment is successful.

Conclusion:

1. EC2 Instance Configuration Issue: The security settings for EC2 might be wrong, stopping the Kubernetes nodes from talking to each other or making services unreachable.

2. Docker Installation Issue: .Docker might not install or run properly because of wrong setup instructions, missing files, or permission problems.

3. Nginx Deployment Issues: The Nginx server might not start because of network problems in Kubernetes or restrictions on the control plane that stop the pod from running.