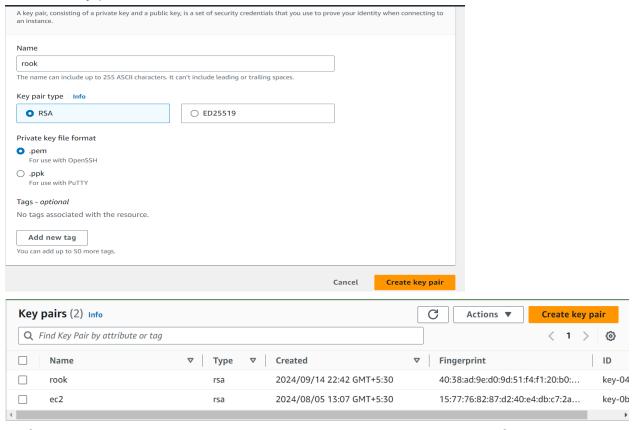
Aim:

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

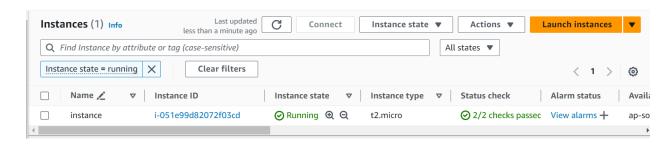
Steps:

1. Create a key pair.

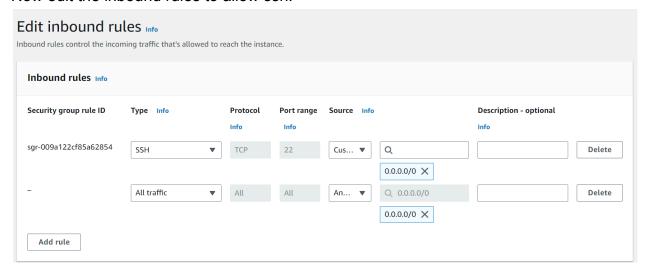


The .pem file will be downloaded on your machine and will be required in the further steps.

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.

```
bhumi@LAPTOP-RVJC2CFS MINGW64 ~/Downloads
$ chmod 400 rook.pem
bhumi@LAPTOP-RVJC2CFS MINGW64 ~/Downloads
$
```

- 5. Now use this command on the terminal: ssh -i <keyname>.pem ubuntu@ 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 and you will find the public ip address there.

6. Docker installation:

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

```
stalling dependencies:
            al download size: 84 M
                                               download size: 84 M
led size: 317 M
advine Pace 1317 M
advine Pace 14 N
ad
                                                       Ing transaction

Ing tr
                                                                                                                                                                                                                                                             iptables-libs-1.8.8-3.amzn2023.0.2.x86_64
libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64
pigz-2.5-1.amzn2023.0.3.x86_64
```

7. Then to configure cgroup in a daemon json file we will run cd /etc/docker cat <<EOF | sudo tee /etc/docker/daemon.json {</p>

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

} EOF

sudo systemctl enable docker sudo systemctl daemon-reload sudo systemctl restart 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"]
}
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.</pre>
```

8. Kubernetes installation:

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

1. Set SELinux to permissive mode: These instructions are for Kubernetes 1.31. Linux in permissive mode (effectively disabling it) | -i 's/^SELINUX=enforcing\$/SELINUX=permissive/' /etc/selinux/config # This overwrites any existing configuration in /etc/yum.repos.d/ cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo</pre> [kubernetes] name=Kubernetes baseurl=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/ gpgcheck=1 gpgkey=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/repodata/repom exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni 3. Install kubelet, kubeadm and kubectl: yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes 4. (Optional) Enable the kubelet service before running kubeadm: sudo systemctl enable --now kubelet

```
ntes
stadata expiration check: 0:00:02 ago on Sat Sep 14 17:47:29 2024.
uncies resolved.
nstalling
 stalling dependencies:
                                                                                                                                                                                                                                                                                                                              45 MB/s | 51 MB
8.0 kB/s | 1.7 kB
             tes
ng GPG key 0x9A296436:
: "isv:kubernetes 0BS Project <isv:kubernetes@build.opensuse.org>"
print: DEIS B144 86CD 3778 9E87 6EIA 2346 54DA 9A29 6436
: https://pkgs.k8s.io/core:/stable:/v1.31/rpm/repodata/repomd.xml.key
orted successfully
transaction check
transaction check
 stalled:
conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64
kubectl-1.31.1-150500.1.1.x86_64
libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64
                                                                                                                                     cri-tools-1.31.1-150500.1.1.x86_64
kubelet-1.31.1-150500.1.1.x86_64
libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64
                                                                                                                                                                                                                                                                             kubeadm-1.31.1-150500.1.1.x86_64
kubernetes-cni-1.5.1-150500.1.1.x86_64
libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64
      lete!
-user@ip-172-31-3-16 docker]$
```

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

```
ubuntu@ip-172-31-11-193:~$ sudo swapoff -a
ubuntu@ip-172-31-11-193:~$ echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /etc/sysctl.conf
net.bridge.bridge-nf-call-iptables=1
ubuntu@ip-172-31-11-193:~$ sudo sysctl -p
net.bridge.bridge-nf-call-iptables = 1
ubuntu@ip-172-31-11-193:~$ |
```

10. Initializing 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 172.31.3.16:6443 --token ekhyop.xkge2agz07jxxqqs \
 --discovery-token-ca-cert-hash sha256:8206263b4e2632eb03dafa4819c7c8505d47b21e8ba8c4901d5802c791c806f7
```

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

```
ubuntu@ip-172-31-11-193:~$ 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

```
ubuntu@ip-172-31-11-193:~$ 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
```

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

```
ubuntu@ip-172-31-11-193:~$ kubectl apply -f https://k8s.io/examples/application/deployment.yamldeployment.apps/nginx-deployment created
```

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

```
ubuntu@ip-172-31-11-193:~$ kubectl get pods
                                     READY
                                                       RESTARTS
NAME
                                             STATUS
                                                                   AGE
nginx-deployment-d556bf558-49tlj
                                     0/1
                                             Pending
                                                       0
                                                                   13s
nginx-deployment-d556bf558-qjqbg
                                     0/1
                                             Pending
                                                       0
                                                                   13s
```

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

```
nginx-deployment-d556bf558-w2pd8 0/1 Pending 0
[ec2-user@ip-172-31-3-16 docker]$ kubectl describe pod nginx |
Name: nginx-deployment-d556bf558-mvnj7
Namespace: default
Priority: 0
Service Account: default
iode: <none>
                                                                                   der ab

<none>

app=nginx

pod-template-hash=d556bf558

<none>

Pending
                                                                                    <none>
ReplicaSet/nginx-deployment-d556bf558
                  trolled By:
                            nx:
nage:
nginx:1.14.2
port:
80/TCP
post Port:
0/TCP
nvironment:
<none>
                   Environment: <none>
Mounts:
/var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-8cms7 (ro)
litions:
          Type Status
PodScheduled False
             Tumes:
kube-api-access-8cms7:

    kube-api-access-8cms7:
    Projected (a volume that contains injected data from multiple sources)

    Type:
    TokenExpirationseconds:

    3607
    kube-root-ca.crt

    ConfigMapOptional:
    (ni)>

    DownwardArd:
    true

    US Class:
    BestEffort

    0de-Selectors:
    <none-</td>

    olde-Selectors:
    node.

    whernets:
    io/uncarrabilationsecute

    node kubernets:
    io/uncarrabilationsecute

                                                                                                                                     3607
kube-root-ca.crt
{nil>
true
BestEffort
{none>
node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
                  Name: nginx-deployment-d556bf558-w2pd8
Namespace: default
Priority: 0
Name: Na
                                                                          <none>
ReplicaSet/nginx-deployment-d556bf558
```

```
Priority: 0
Service Account: default
Node: <none>
.abels: app=ngir
                   app=nginx
pod-template-hash=d556bf558
nnotations:
                   <none>
Pending
                  <none>
ReplicaSet/nginx-deployment-d556bf558
                  nginx:1.14.2
80/TCP
0/TCP
<none>
  Type Status
PodScheduled False
 lumes:
kube-api-access-6f18b:
                              Projected (a volume that contains injected data from multiple sources) 3607 kube-root-ca.crt cnil>
true
BestEffort
   Type:
TokenExpirationSeconds:
ConfigMapName:
ConfigMapOptional:
DownwardAPI:
os Class:
ode-Selectors:
olerations:
                               Age From
           FailedScheduling 57s default-scheduler 0/1 nodes are available: 1 node(s) had untolerated taint {node-role.kubernetes.io/control-plane: }. preemption: 0/1 n
```

Use the below command to remove taints.

kubectl taint nodes -all node-role.kubernetes.io/control-plane-

```
node/ip-172-31-3-16.ap-southeast-2.compute.internal untainted
```

16. Check the pod status.

```
ubuntu@ip-172-31-11-193:~$ kubectl get pods
                                    READY
                                             STATUS
                                                       RESTARTS
                                                                   AGE
nginx-68cf7659df-zx7vp
                                             Running
                                                                   19s
```

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

```
ubuntu@ip-172-31-11-193:~$ kubectl port-forward $POD_NAME 8080:80 Forwarding from 127.0.0.1:8080 -> 80 Forwarding from [::1]:8080 -> 80 Handling connection for 8080
```

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

```
ubuntu@ip-172-31-11-193:~$ curl --head http://127.0.0.1:8080
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Sun, 15 Sep 2024 16:14:46 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
```

Conclusion: In this experiment, we launched an EC2 instance and configured SSH access by updating the inbound rules. Next, we installed Docker and Kubernetes, and adjusted network settings to enable bridging. After completing the setup, we installed the Flannel networking plugin to ensure proper communication within the cluster. Once the cluster was up and running, we successfully deployed an NGINX server and verified its deployment.