

Experiment 4

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

Steps:

Step 1: Create an EC2 Amazon Linux instance on AWS. While doing so, make sure that t2.medium is selected as 'instance type' instead of the default t2.micro. This is because t2.medium provides more CPU, memory, and consistent performance, which are crucial for effectively running Kubernetes components and managing cluster workloads.

node i-0b1f12ee5fcadb8ea Running t2.medium Initializing View alarms

aws Services Search [Alt+S] N. Virginia voclabs/user3385469=KULKARNI_ANISH_AMIT @ 6635-3922-7562

Amazon Linux macOS Ubuntu Windows Red Hat SUSE Linux

Amazon Machine Image (AMI)

Amazon Linux 2023 AMI Free tier eligible

ami-0182f373e66f89c85 (64-bit (x86), uefi-preferred) / ami-0b947c5d5516fa06e (64-bit (Arm), uefi)

Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.

Architecture Boot mode AMI ID

64-bit (x86) uefi-preferred ami-0182f373e66f89c85 Verified provider

▼ 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

Additional costs apply for AMIs with pre-installed software

All generations

Compare instance types

```
[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	Size
Installing:				
docker	x86_64	25.0.6-1.amzn2023.0.2	amazonlinux	44 M
Installing dependencies:				
containerd	x86_64	1.7.20-1.amzn2023.0.1	amazonlinux	35 M
iptables-libs	x86_64	1.8.8-3.amzn2023.0.2	amazonlinux	401 k
iptables-nft	x86_64	1.8.8-3.amzn2023.0.2	amazonlinux	183 k
libcgroup	x86_64	3.0-1.amzn2023.0.1	amazonlinux	75 k
libnetfilter_conntrack	x86_64	1.0.8-2.amzn2023.0.2	amazonlinux	58 k
libnftnl	x86_64	1.0.1-19.amzn2023.0.2	amazonlinux	30 k
libnftnl	x86_64	1.2.2-2.amzn2023.0.2	amazonlinux	84 k
pigz	x86_64	2.5-1.amzn2023.0.3	amazonlinux	83 k
runc	x86_64	1.1.13-1.amzn2023.0.1	amazonlinux	3.2 M

```

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      1/10
Verifying      : docker-25.0.6-1.amzn2023.0.2.x86_64      2/10
Verifying      : iptables-libs-1.8.8-3.amzn2023.0.2.x86_64  3/10
Verifying      : iptables-nft-1.8.8-3.amzn2023.0.2.x86_64  4/10
Verifying      : libcgrou-3.0-1.amzn2023.0.1.x86_64      5/10
Verifying      : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64  6/10
Verifying      : libnftnl-1.0.1-19.amzn2023.0.2.x86_64    7/10
Verifying      : libnftnl-1.2.2-2.amzn2023.0.2.x86_64     8/10
Verifying      : pigz-2.5-1.amzn2023.0.3.x86_64           9/10
Verifying      : runc-1.1.13-1.amzn2023.0.1.x86_64        10/10

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

Complete!

```

Step 4: Configure Docker to use systemd for managing cgroups by updating its configuration file, ensuring Docker starts automatically on boot, reloading the systemd configuration, and restarting Docker to apply the changes. Use the following commands to do so:

```

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

```

Step 5: Install kubernetes using the following commands:

```

sudo tee /etc/yum.repos.d/kubernetes.repo <<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
EOF

```

```

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

```

```

sudo yum clean all
sudo yum install -y kubelet kubeadm kubectl --disableexcludes=Kubernetes

```

```

sudo systemctl enable --now kubelet

```

```
aws Services Search [Alt+S] N. Virginia voclabs/user3385469=KULKARNI_ANISH_AMIT @ 6635-3922-7562

[ec2-user@ip-172-31-30-144 ~]$ # Update the Kubernetes repo file and install the required packages
sudo tee /etc/yum.repos.d/kubernetes.repo <<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
EOF

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

# Clean yum cache and install kubelet, kubeadm, and kubectl
sudo yum clean all
sudo yum install -y kubelet kubeadm kubectl --disableexcludes=Kubernetes

# Enable and start kubelet
sudo systemctl enable --now kubelet
[kubernetes]

i-0b1f12ee5fcadb8ea (node)
PublicIPs: 3.88.175.3 PrivateIPs: 172.31.30.144

Running scriptlet: kubectl-1.31.1-150500.1.1.x86_64 9/9
Verifying      : conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64 1/9
Verifying      : libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64 2/9
Verifying      : libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64 3/9
Verifying      : libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64 4/9
Verifying      : cri-tools-1.31.1-150500.1.1.x86_64 5/9
Verifying      : kubeadm-1.31.1-150500.1.1.x86_64 6/9
Verifying      : kubectl-1.31.1-150500.1.1.x86_64 7/9
Verifying      : kubelet-1.31.1-150500.1.1.x86_64 8/9
Verifying      : kubernetes-cni-1.5.1-150500.1.1.x86_64 9/9

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

Complete!
Created symlink /etc/systemd/system/multi-user.target.wants/kubelet.service → /usr/lib/systemd/system/kubelet.service.
```

Step 6: Initialise the kubernetes cluster using the “sudo kubeadm init” command.

```
[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" of the container runtime is
inconsistent with that used by kubeadm.It is recommended to use "registry.k8s.io/pause:3.10" as the CRI sandbox image.
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [ip-172-31-30-144.ec2.internal kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.31.30.144]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [ip-172-31-30-144.ec2.internal localhost] and IPs [172.31.30.144 127.0.0.1 :

i-0b1f12ee5fcadb8ea (node)
PublicIPs: 3.88.175.3 PrivateIPs: 172.31.30.144
```

```
aws Services Search [Alt+S] N. Virginia voclabs/user3385469=KULKARNI_ANISH_AMIT @ 6635-3922-75

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 172.31.30.144:6443 --token 63vvxh.52ln03e137tw9h5w \
--discovery-token-ca-cert-hash sha256:5eb05d2a34a83ff457f033d4cb3f20b321bbec5e056051d59e17aec1f0ce48ce
[ec2-user@ip-172-31-30-144 ~]$

i-0b1f12ee5fcadb8ea (node)
PublicIPs: 3.88.175.3 PrivateIPs: 172.31.30.144
```

Step 7: 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) $HOME/.kube/config
[ec2-user@ip-172-31-30-144 ~]$

i-0b1f12ee5fcadb8ea (node)
PublicIPs: 3.88.175.3 PrivateIPs: 172.31.30.144
```

Step 8: Deploy the Flannel networking plugin to the Kubernetes cluster using the following command:

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

```
[ec2-user@ip-172-31-30-144 ~]$ 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
[ec2-user@ip-172-31-30-144 ~]$

i-0b1f12ee5fcadb8ea (node)
PublicIPs: 3.88.175.3 PrivateIPs: 172.31.30.144
```

Step 9: Deploy the nginx server on the kubernetes cluster using the following command:

kubectl apply -f <https://k8s.io/examples/application/deployment.yaml>

```
[root@ip-172-31-25-172 docker]# kubectl apply -f https://k8s.io/examples/application/deployment.yaml
deployment.apps/nginx-deployment created
```

Step 10: Execute the “kubectl get pods” command to verify if the deployment was properly created and the pod is working correctly.

```
[root@ip-172-31-25-172 docker]# kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-deployment-d556bf558-bqw22	0/1	Pending	0	66s
nginx-deployment-d556bf558-jxjxq	0/1	Pending	0	66s

Step 11: Here, it is observed that the status of the pods is “Pending”. To convert the status of the pods from “Pending” to “Running”, we must first execute the “kubectl describe pod nginx” command to get detailed information about the nginx pod such as its status, labels, annotations, containers, events, and resource usage.

```
[root@ip-172-31-23-234 docker]# kubectl describe pod nginx
```

```
Name:          nginx-deployment-77d8468669-7hwfr
Namespace:     default
Priority:       0
Service Account: default
Node:          <none>
Labels:        app=nginx
                pod-template-hash=77d8468669
Annotations:   <none>
Status:        Pending
IP:            <none>
```

```
Warning FailedScheduling 61s default-scheduler 0/1 nodes are available: 1 node(s) had untolerated taint {node-role.kubernetes.io/control-plane: }. preemption
: 0/1 nodes are available: 1 Preemption is not helpful for scheduling.
```

Step 12: It is observed that the node has untolerated taints which prevents pods from being scheduled on nodes that have certain conditions or restrictions, which are specified by the taints on those nodes. To fix this, run the following command:
 kubectl taint nodes --all [node-role.kubernetes.io/control-plane:NoSchedule-](#)

Step 13: Execute “kubectl get pods” command again to check if the status of the pods has been converted to “Running”.

Step 14: Forward port 8080 on your local machine to port 80 on the specified pod (\$POD_NAME) using the following command:
 kubectl port-forward \$POD_NAME 8080:80

```
[root@ip-172-31-23-234 docker]# kubectl port-forward nginx-deployment-77d8468669-s77nc 8081:80
```

```
Forwarding from 127.0.0.1:8081 -> 80
Forwarding from [::1]:8081 -> 80
error: lost connection to pod
```

Here, it is observed that an error occurs. This is because there's a disruption or failure in the communication between the local machine and the Kubernetes pod during the kubectl port-forward operation.

Conclusion:

1. In this experiment, we learned how to install Kubectl and execute Kubectl commands to manage the Kubernetes cluster and deploy Your First Kubernetes Application.
2. First, we created an EC2 Amazon Linux instances on AWS and established its connections with a remote server through SSH.

3. Next, we installed and configured docker on all 3 machines.
4. Then, we installed and initialised kubernetes on the machine. We also added the machine to a kubernetes cluster.
5. Then, we deployed the Flannel networking plugin to the Kubernetes cluster using the “kubectl apply -f” command.
6. Then, we deployed the nginx server on the kubernetes cluster. This required the status of the pods to be changed from “Pending” to “Running” which in turn, required the untolateralated taints from the pods to be removed.
7. Then, we tried to forward port 8080 on our local machine to port 80 on the specified pod where an error was encountered due to a disruption or failure in the communication between the local machine and the Kubernetes pod during the kubectl port-forward operation.