

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.

following the simple steps below.

Name and tags [Info](#)

Name

[Add additional tags](#)

▼ Application and OS Images (Amazon Machine Image) [Info](#)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Recents

Quick Start

Amazon Linux

aws

macOS

Mac

Ubuntu

ubuntu

Windows

Microsoft

Red Hat

Red Hat

SUSE Li

SUS

[Browse more AMIs](#)

Including AMIs from AWS, Marketplace and the Community

▼ Summary

Number of instances [Info](#)

Software Image (AMI)

Amazon Linux 2023 AMI 2023.5.2...[read more](#)
ami-0182f373e66f89c85

Virtual server type (instance type)

t2.medium

Firewall (security group)

New security group

Storage (volumes)

1 volume(s) - 8 GiB

[Free tier](#): In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 750 hours of public IPv4

Cancel

Launch instance

[Review commands](#)

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

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

☒ All generations

[Compare instance types](#)

Additional costs apply for AMIs with pre-installed software

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

[Create new key pair](#)

▼ Network settings [Info](#)

Network [Info](#)

vpc-09b4fa6cf9c39cafb

Subnet [Info](#)

No preference (Default subnet in any availability zone)

▼ Summary

Number of instances [Info](#)

Software Image (AMI)

Amazon Linux 2023 AMI 2023.5.2...[read more](#)
ami-0182f373e66f89c85

Virtual server type (instance type)

t2.medium

Firewall (security group)

New security group

Storage (volumes)

1 volume(s) - 8 GiB

[Free tier](#): In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 750 hours of public IPv4

Cancel

Launch instance

[Review commands](#)


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


EC2 Instance Connect

Session Manager

SSH client

EC2 serial console


**Port 22 (SSH) is open to all IPv4 addresses**
Port 22 (SSH) is currently open to all IPv4 addresses, indicated by **0.0.0.0/0** in the inbound rule in [your security group](#). For increased security, consider restricting access to only the EC2 Instance Connect service IP addresses for your Region: 18.206.107.24/29. [Learn more.](#)

Instance ID
 i-0d333b5802e08e3fe (Instance)

Connection Type

☒ **Connect using EC2 Instance Connect**
Connect using the EC2 Instance Connect browser-based client, with a public IPv4 address.


☐ **Connect using EC2 Instance Connect Endpoint**
Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

Public IPv4 address
 44.223.24.238

Username
Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.

Q ec2-user

X

 **Note:** In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

Cancel

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
libnftnl	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:
```

```

=====
=== Installing : docker-25.0.6-1.amzn2023.0.2.x86_64 [===== Installing
                                           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      : libnftnl-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  libnftnl-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/>

The screenshot shows the Kubernetes documentation website. The header includes the Kubernetes logo and navigation links: Documentation, Kubernetes Blog, Training, Partners, Community, Case Studies, Versions, and English. A search bar is on the right. The left sidebar shows a navigation menu with 'Getting started' expanded, leading to 'Production environment', which includes 'Container Runtimes', 'Installing Kubernetes with deployment tools', and 'Bootstrapping clusters with'. The main content area is titled 'the installation guide for your desired minor version.' and has tabs for 'Debian-based distributions' and 'Red Hat-based distributions'. The 'Without a package manager' section is selected, showing step 1: 'Set SELinux to permissive mode:'. It includes instructions for Kubernetes 1.31 and a terminal snippet: '# Set SELinux in permissive mode (effectively disabling it)', 'sudo setenforce 0', and 'sudo sed -i 's/^SELINUX=enforcing\$/SELINUX=permissive/' /etc/selinux/config'. On the right, there are links to 'Edit this page', 'Create child page', 'Create documentation issue', and 'Print entire section', followed by a 'Before you begin' section with links to 'Verify the MAC address and product\_uid', 'unique for every node', 'Check network adapters', 'Check required ports', 'Swap configuration', and 'Installing a container runtime'.

**kubernetes**

Documentation Kubernetes Blog Training Partners Community Case Studies Versions English Search this site

the installation guide for your desired minor version.

Debian-based distributions Red Hat-based distributions

Without a package manager

1. Set SELinux to **permissive** mode:

These instructions are for Kubernetes 1.31.

```
Set SELinux in permissive mode (effectively disabling it)
sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

Edit this page  
Create child page  
Create documentation issue  
Print entire section

Before you begin  
Verify the MAC address and product\_uid  
unique for every node  
Check network adapters  
Check required ports  
Swap configuration  
Installing a container runtime

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 : kubelet-1.31.1-150500.1.1.x86_64
9/9
Running scriptlet: kubelet-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/system/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 to initialize 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
- sudo cp -i /etc/kubernetes/admin.conf \$HOME/.kube/config
- sudo chown \$(id -u):\$(id -g) \$HOME/.kube/config

Run These Commands

```

[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]#

```

5) Add a common networking plugin flannel using this command

- 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/flannel/master/Documentation/kube-flannel.yml
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-172-31-29-96 docker]#

```

Step3: Deploy nginx server

1) Now that the cluster is set, apply the deployment file of nginx using this command

- kubectl apply -f <https://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 the pod is in a 'PENDING' state, change it to RUNNING state, run the 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: <none>
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

- `kubectltaintnodes --allnode-role.kubernetes.io/control-plane:NoSchedule-`

```
[root@ip-172-31-29-96 docker]# kubectltaint nodes --all node-role.kubernetes.io/control-plane:NoSchedule-
kubectltaint nodes --all node-role.kubernetes.io/control-plane:NoSchedule-
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.