

Aim: To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud

1. Create 3 EC-2 instances with all running on Amazon Linux as OS with inbound SSH allowed and the proper key

EC2 > Instances > Launch an instance

Launch an instance [Info](#)

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by 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 L
SUS

[Browse more AMIs](#)
Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Amazon Linux 2023 AMI ami-0182f373e66f89c85 (64-bit (x86), uefi-preferred) / ami-0b947c5d5516fa06e (64-bit (Arm), uefi) Virtualization: hvm ENA enabled: true Root device type: ebs	Free tier eligible
---	--------------------

▼ 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 address usage per month, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

Cancel **Launch instance** [Review commands](#)

To efficiently run a Kubernetes cluster, select an instance type of at least t2.medium as Kubernetes recommends at least 2 vCPU to run smoothly.

▼ 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*

kub1 ▼

Create new key pair

▼ Network settings Info Edit

Network Info

vpc-0deb6a82b5be91aae

Subnet Info

No preference (Default subnet in any availability zone)

Auto-assign public IP Info

Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups) Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group

Select existing security group

We'll create a new security group called 'launch-wizard-10' with the following rules:

☒ Allow SSH traffic from

Anywhere
0.0.0.0/0 ▼

Helps you connect to your instance

☒ Allow HTTPS traffic from the internet

To set up an endpoint, for example when creating a web server

☒ Allow HTTP traffic from the internet

To set up an endpoint, for example when creating a web server

⚠ Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

▼ Summary

Number of instances Info

1

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 address usage per month, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

Cancel

Launch instance

[Review commands](#)

In this way create 3 instances namely master, worker-1 and worker-2

Instances (3) [info](#)

Last updated less than a minute ago

Connect

Instance state

Actions

Launch instances

Find Instance by attribute or tag (case-sensitive)

All states

<1>

<input type="checkbox"/>	Name ↗	Instance ID	Instance state ↕	Instance type ↕	Status check ↕	Alarm status	Availability Zone ↕	Public IPv4 DNS ↕	Public IPv4 ... ↕	Elastic IP	IPv6
<input type="checkbox"/>	master	i-0604dbbc26a0a4f01	<div><div>Running</div><div><div></div><div></div></div></div>	t2.medium	<div><div>Initializing</div><div><div></div><div></div></div></div>	View alarms +	us-east-1a	ec2-54-85-79-186.com...	54.85.79.186	-	-
<input type="checkbox"/>	worker-1	i-05548bd7fe0a7292f	<div><div>Running</div><div><div></div><div></div></div></div>	t2.medium	<div><div>Initializing</div><div><div></div><div></div></div></div>	View alarms +	us-east-1a	ec2-54-196-211-209.co...	54.196.211.209	-	-
<input type="checkbox"/>	worker-2	i-007a15dade39c85b0	<div><div>Running</div><div><div></div><div></div></div></div>	t2.medium	<div><div>Initializing</div><div><div></div><div></div></div></div>	View alarms +	us-east-1a	ec2-18-209-62-85.com...	18.209.62.85	-	-

- SSH into all 3 machines each in separate terminal
 - You can do it through the aws console directly


```
[ec2-user@ip-172-31-31-212 ~]$ sudo yum install docker -y
Last metadata expiration check: 0:33:43 ago on Thu Sep 12 13:11:13 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

```

Then, configure cgroup in a daemon.json file by using following 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`

```
[ec2-user@ip-172-31-20-75 ~]$ cd /etc/docker
[ec2-user@ip-172-31-20-75 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
[ec2-user@ip-172-31-20-75 docker]$ ls
daemon.json  kubectl
```

- `sudo systemctl enable docker`
- `sudo systemctl daemon-reload`
- `sudo systemctl restart docker`
- `docker -v`

```
[ec2-user@ip-172-31-31-212 docker]$ sudo systemctl enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
[ec2-user@ip-172-31-31-212 docker]$ sudo systemctl daemon-reload
[ec2-user@ip-172-31-31-212 docker]$ sudo systemctl restart docker
[ec2-user@ip-172-31-31-212 docker]$ docker -v
Docker version 25.0.5, build 5dc9bcc
[ec2-user@ip-172-31-31-212 docker]$
```

4. Install Kubernetes on all 3 machines

SELinux needs to be disabled before configuring kubelet

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

```
[ec2-user@ip-172-31-26-2 docker]$ sudo setenforce 0
[ec2-user@ip-172-31-26-2 docker]$ sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
[ec2-user@ip-172-31-26-2 docker]$
```

Add kubernetes repository (paste in terminal)

```
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.30/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF
```

Type following commands to install set of kubernetes packages:

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

```
[ec2-user@ip-172-31-31-212 docker]$ sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
Last metadata expiration check: 0:00:23 ago on Thu Sep 12 14:09:10 2024.
Dependencies resolved.
```

Package	Architecture	Version
Installing:		
kubeadm	x86_64	1.30.5-150500.1.1
kubectl	x86_64	1.30.5-150500.1.1
kubelet	x86_64	1.30.5-150500.1.1
Installing dependencies:		
conntrack-tools	x86_64	1.4.6-2.amzn2023.0.2
cri-tools	x86_64	1.30.1-150500.1.1
kubernetes-cni	x86_64	1.4.0-150500.1.1
libnetfilter_cthelper	x86_64	1.0.0-21.amzn2023.0.2
libnetfilter_cttimeout	x86_64	1.0.0-19.amzn2023.0.2
libnetfilter_queue	x86_64	1.0.5-2.amzn2023.0.2

```
Transaction Summary
Install 9 Packages

Total download size: 53 M
Installed size: 292 M
Downloading Packages:
(1/9): libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64.rpm
(2/9): libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64.rpm
(3/9): libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64.rpm
(4/9): conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64.rpm
(5/9): kubeadm-1.30.5-150500.1.1.x86_64.rpm
(6/9): kubectl-1.30.5-150500.1.1.x86_64.rpm
(7/9): cri-tools-1.30.1-150500.1.1.x86_64.rpm
(8/9): kubernetes-cni-1.4.0-150500.1.1.x86_64.rpm
(9/9): kubelet-1.30.5-150500.1.1.x86_64.rpm

Total
Kubernetes
Importing GPG key 0x9A296436:
Userid      : "isv:kubernetes OBS Project <isv:kubernetes@build.opensuse.org>"
Fingerprint: DE15 B144 86CD 377B 9E87 6E1A 2346 54DA 9A29 6436
From        : https://pkgs.k8s.io/core/stable/v1.30/rpm/repodata/repomd.xml.key
Key imported successfully
Running transaction check
Transaction check succeeded.
```

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`

5. Perform this ONLY on the Master machine

Initialize kubernetes by typing below command

- `sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all`

```
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

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 172.31.26.2:6443 --token 6cjsz0.8ei243v0zn9k7erg \
--discovery-token-ca-cert-hash sha256:abd917ec30e12c5616bf647a3d174bef3d271e92c30b8f2f7768cfb3181341d4
[ec2-user@ip-172-31-26-2 docker]$
```

Copy the mkdir and chown commands from the top and execute them

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Copy this join link and save it in clipboard (copy from your output as it different for each master instance)

Example :

```
kubeadm join 172.31.20.75:6443 --token 66kg9u.2bc0kze31hrwbzvr \
--discovery-token-ca-cert-hash
sha256:5e478da328b199e17d9b5da68e78bc9a6daab2043b05860552f4c184a7b3cb66
```

Then, add a common networking plugin called flannel file as mentioned in the code.

Command:

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

```
[ec2-user@ip-172-31-26-2 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
```

6. Perform this ONLY on the worker machines

Paste the below command on all 2 worker machines

- `sudo yum install iproute-tc -y`
- `sudo systemctl enable kubelet`
- `sudo systemctl restart kubelet`

```
ec2-user@ip-172-31-31-212 docker]$ sudo yum install iproute-tc -y
Last metadata expiration check: 0:15:14 ago on Thu Sep 12 14:09:10 2024.
Dependencies resolved.
=====================================================================================================================================
Package                               Architecture           Version                 Repository              Size
=====================================================================================================================================
Installing:
iproute-tc                           x86_64                 5.10.0-2.amzn2023.0.5   amazonlinux              455 k
Transaction Summary
=====================================================================================================================================
Install 1 Package
Total download size: 455 k
Installed size: 928 k
Downloading Packages:
iproute-tc-5.10.0-2.amzn2023.0.5.x86_64.rpm                                           4.0 MB/s | 455 kB  00:00
-----
Total                                                                                   2.8 MB/s | 455 kB  00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing      :
  Installing     : iproute-tc-5.10.0-2.amzn2023.0.5.x86_64                        1/1
  Running scriptlet: iproute-tc-5.10.0-2.amzn2023.0.5.x86_64                    1/1
  Verifying      : iproute-tc-5.10.0-2.amzn2023.0.5.x86_64                      1/1
Installed:
  iproute-tc-5.10.0-2.amzn2023.0.5.x86_64
Complete!
```

```
ec2-user@ip-172-31-31-212 docker]$
[ec2-user@ip-172-31-17-184 docker]$ sudo systemctl enable kubelet
Created symlink /etc/systemd/system/multi-user.target.wants/kubelet.service → /usr/lib/systemd/system/kubelet.service.
[ec2-user@ip-172-31-17-184 docker]$ sudo systemctl restart kubelet
[ec2-user@ip-172-31-17-184 docker]$
```

Now paste the hash that you copied in these worker note to connect to master cluster

- `kubeadm join 172.31.20.75:6443 --token 66kg9u.2bc0kze3lhrwbzvr \`
`--discovery-token-ca-cert-hash`
`sha256:5e478da328b199e17d9b5da68e78bc9a6daab2043b05860552f4c184a7b3cb66`

Now we can see in the master/control node of Kubernetes that worker nodes are connected by this command

- **watch kubectl get nodes**
(in the master node instance)

Errors faced during the execution :

1. In the end kubelet might not respond or the connectivity of nodes to master might not happen
2. You can see this error


```
[ec2-user@ip-172-31-20-75 docker]$ kubectl get nodes
E0914 06:14:55.956919 3650 memcache.go:265] couldn't get current server API group list: Get "https://172.31.20.75:6443/api?timeout=32s":
connection refused
E0914 06:14:55.957758 3650 memcache.go:265] couldn't get current server API group list: Get "https://172.31.20.75:6443/api?timeout=32s":
connection refused
E0914 06:14:55.959507 3650 memcache.go:265] couldn't get current server API group list: Get "https://172.31.20.75:6443/api?timeout=32s":
connection refused
E0914 06:14:55.960160 3650 memcache.go:265] couldn't get current server API group list: Get "https://172.31.20.75:6443/api?timeout=32s":
connection refused
E0914 06:14:55.961526 3650 memcache.go:265] couldn't get current server API group list: Get "https://172.31.20.75:6443/api?timeout=32s":
connection refused
E0914 06:14:55.962881 3650 memcache.go:265] couldn't get current server API group list: Get "https://172.31.20.75:6443/api?timeout=32s":
connection refused
```

3. Try to restart the kubelet from worker instance and try the commands again

Conclusion :

In this experiment, we aimed to deploy Kubernetes in Docker, connecting a master node with two worker nodes. We encountered several challenges, starting with SSH inbound rule misconfigurations, which were resolved by enabling the correct rules for secure access. It became clear that using **t2.medium** or **t3** instances was essential for sufficient resources to run Kubernetes smoothly. Despite these adjustments, the worker nodes failed to join the cluster. Although the master node was ready, the issue seemed to stem from improper configuration of the worker nodes' kubelet or a networking problem, such as the worker nodes being unable to reach the master node's API server. This could be due to incorrect firewall settings, missing API server certificates, or failure in configuring the kubeadm join process on the worker nodes.