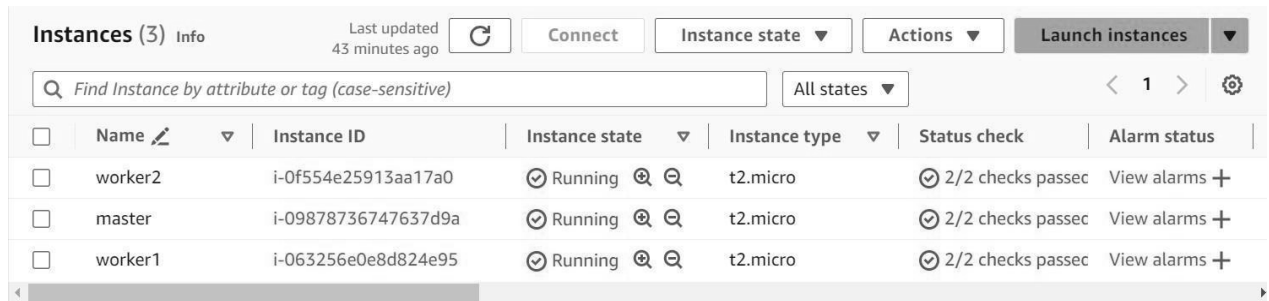


Aim: To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud Platforms.

Steps:

1. We will create 3 EC2 instances. One will be the master node and the other 2 will be slave/worker nodes.



The screenshot displays the AWS Management Console's 'Instances' page. At the top, it shows 'Instances (3)' with an 'Info' link, a refresh button, and a 'Last updated 43 minutes ago' timestamp. There are buttons for 'Connect', 'Instance state', 'Actions', and 'Launch instances'. A search bar is present with the placeholder text 'Find Instance by attribute or tag (case-sensitive)'. Below the search bar, a table lists three instances:

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status
<input type="checkbox"/>	worker2	i-0f554e25913aa17a0	Running	t2.micro	2/2 checks passed	View alarms
<input type="checkbox"/>	master	i-09878736747637d9a	Running	t2.micro	2/2 checks passed	View alarms
<input type="checkbox"/>	worker1	i-063256e0e8d824e95	Running	t2.micro	2/2 checks passed	View alarms

2. After the instances have been created, we will connect them one by one.

Instances (1/3) Info
 Last updated less than a minute ago
 ↻

Connect
Instance state ▼
Actions ▼
Launch instances ▼

All states ▼
< 1 > ⚙

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status
<input type="checkbox"/>	worker2	i-0f554e25913aa17a0	Running	t2.micro	2/2 checks passed	View alarms +
<input checked="" type="checkbox"/>	master	i-09878736747637d9a	Running	t2.micro	2/2 checks passed	View alarms +
<input type="checkbox"/>	worker1	i-063256e0e8d824e95	Running	t2.micro	2/2 checks passed	View alarms +

security group. For increased security, consider restricting access to only the EC2 Instance Connect service IP addresses for your Region: 13.239.158.0/29. [Learn more.](#)

Instance ID

i-09878736747637d9a (master)

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 IP address

3.106.222.144

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.

✕

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

3. Docker installation:

This step has to be performed on all the 3 instances. The following command has to be run:

```
yum install docker -y
```

```
#####  
~\###\ Amazon Linux 2023  
~\###|  
~\V/  
~\V~+--+  
~\---+---+  
~/m/'
```

https://aws.amazon.com/linux/amazon-linux-2023

```
[ec2-user@ip-172-31-12-97 ~]$ sudo su  
[root@ip-172-31-12-97 ec2-user]# yum install docker -y  
Last metadata expiration check: 0:08:33 ago on Sat Sep 14 15:21:32 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-lib s	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

Services Search [Alt+S] Sydney bhumish

Package	Architecture	Version	Repository	Size
libn	x86_64	1.0.8-2.amzn2023.0.2	amazonlinux	58 k
libnfnetlink	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

Transaction Summary

Install 10 Packages

Total download size: 84 M
Installed size: 317 M
Downloading Packages:

Package	Download Size	Uncompressed Size	Estimated Time
(1/10): iptables-lib-1.8.8-3.amzn2023.0.2.x86_64.rpm	3.0 MB/s 401 kB		00:00
(2/10): iptables-nft-1.8.8-3.amzn2023.0.2.x86_64.rpm	6.6 MB/s 183 kB		00:00
(3/10): libcgroup-3.0-1.amzn2023.0.1.x86_64.rpm	1.7 MB/s 75 kB		00:00
(4/10): libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64.rpm	1.6 MB/s 58 kB		00:00
(5/10): libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64.rpm	823 kB/s 30 kB		00:00
(6/10): libnftnl-1.2.2-2.amzn2023.0.2.x86_64.rpm	2.9 MB/s 84 kB		00:00
(7/10): pigz-2.5-1.amzn2023.0.3.x86_64.rpm	2.4 MB/s 83 kB		00:00
(8/10): runc-1.1.13-1.amzn2023.0.1.x86_64.rpm	15 MB/s 3.2 MB		00:00
(9/10): containerd-1.7.20-1.amzn2023.0.1.x86_64.rpm	36 MB/s 35 MB		00:00
(10/10): docker-25.0.6-1.amzn2023.0.2.x86_64.rpm	30 MB/s 44 MB		00:01

Total 56 MB/s | 84 MB 00:01

Run ptables-nft-1.8.8-3.amzn2023.0.2.x86_64 8/10

Installing : libcgroup-3.0-1.amzn2023.0.1.x86_64 9/10

Running scriptlet: docker-25.0.6-1.amzn2023.0.2.x86_64 10/10

Installing : docker-25.0.6-1.amzn2023.0.2.x86_64 10/10

Running scriptlet: docker-25.0.6-1.amzn2023.0.2.x86_64 10/10

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-lib-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 : libcgroup-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 : libnfnetlink-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-lib-1.8.8-3.amzn2023.0.2.x86_64
iptables-nft-1.8.8-3.amzn2023.0.2.x86_64 libcgroup-3.0-1.amzn2023.0.1.x86_64 libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64
libnfnetlink-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!

4. After successfully docker has been installed it has to be started on all machines by using the command “systemctl start docker”

Complete!

```
[root@ip-172-31-12-97 ec2-user]# systemctl start docker
```

- ## 5. 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)
setenforce 0
sed -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
[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
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF
```

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
```

Copy the above given steps and paste in the terminal. This will create a Kubernetes repository, install kubelet, kubeadm and kubectl and also enable the services.

```
[root@ip-172-31-14-85 ec2-user]# 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-172-31-12-97 ec2-user]# yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
Kubernetes
Dependencies resolved.
21 kB/s | 9.4 kB    00:00

Package Architecture Version Repository Size
-----
Installing:
kubeadm x86_64 1.31.1-150500.1.1 kubernetes 11 M
kubectl x86_64 1.31.1-150500.1.1 kubernetes 11 M

kubernetes x86_64 1.31.1-150500.1.1 kubernetes 15 M
Installing dependencies:
conntrack-tools x86_64 1.4.6-2.amzn2023.0.2 amazonlinux 208 k
cri-tools x86_64 1.31.1-150500.1.1 kubernetes 6.9 M
kubernetes-cni x86_64 1.5.1-150500.1.1 kubernetes 7.1 M
libnetfilter_cthelper x86_64 1.0.0-21.amzn2023.0.2 amazonlinux 24 k
libnetfilter_cttimeout x86_64 1.0.0-19.amzn2023.0.2 amazonlinux 24 k
libnetfilter_queue x86_64 1.0.5-2.amzn2023.0.2 amazonlinux 30 k

Transaction Summary
-----
Install 9 Packages

Total download size: 51 M
Installed size: 269 M
Downloading Packages:
(1/9): libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64.rpm 500 kB/s | 24 kB 00:00
(2/9): libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64.rpm 475 kB/s | 24 kB 00:00
(3/9): conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64.rpm 3.6 MB/s | 208 kB 00:00
(4/9): libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64.rpm 1.4 MB/s | 30 kB 00:00
(5/9): kubeadm-1.31.1-150500.1.1.x86_64.rpm 17 MB/s | 11 MB 00:00
(6/9): kubectl-1.31.1-150500.1.1.x86_64.rpm 15 MB/s | 11 MB 00:00
(7/9): cri-tools-1.31.1-150500.1.1.x86_64.rpm 8.0 MB/s | 6.9 MB 00:00
(8/9): kubernetes-cni-1.5.1-150500.1.1.x86_64.rpm 14 MB/s | 7.1 MB 00:00
(9/9): kubelet-1.31.1-150500.1.1.x86_64.rpm 25 MB/s | 15 MB 00:00

Ins libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64 5/9
Installing : conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64 6/9
Running scriptlet: conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64 6/9
Installing : kubelet-1.31.1-150500.1.1.x86_64 7/9
Running scriptlet: kubelet-1.31.1-150500.1.1.x86_64 7/9
Installing : kubeadm-1.31.1-150500.1.1.x86_64 8/9
Installing : kubectl-1.31.1-150500.1.1.x86_64 9/9
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
libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64
```

6. We can check if repository has been created by using yum repolist command.

```
[root@ip-172-31-14-85 ec2-user]# yum repolist
repo id repo name
amazonlinux Amazon Linux 2023 repository
kernel-livepatch Amazon Linux 2023 Kernel Livepatch repository
kubernetes Kubernetes
```

7. Now we will be initializing the kubeadm. For that “kubeadm init” command has to be used. It may show errors but those can be ignored by using `--ignore-preflighterrors=all`

```
[root@ip-172-31-14-85 ec2-user]# kubeadm init --ignore-preflight-errors=NumCPU --ignore-preflight-errors=Mem
[init] Using Kubernetes version: v1.31.0
[preflight] Running pre-flight checks
        [WARNING NumCPU]: the number of available CPUs 1 is less than the required 2
        [WARNING Mem]: the system RAM (949 MB) is less than the minimum 1700 MB
        [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 15:50:31.271160 29520 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-172-31-14-85.ap-southeast-2.compute.internal kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.31.14.85]
[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-172-31-14-85.ap-southeast-2.compute.internal localhost] and IPs [172.31.14.85 127.0.0.1 ::1]
```

```
aws Services Search [Alt+S] Sydney bhumishap
85 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [ip-172-31-14-85.ap-southeast-2.compute.internal localhost] and IPs [172.31.14.85 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "super-admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Starting the kubelet
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests"
[kubelet-check] Waiting for a healthy kubelet at http://127.0.0.1:10248/healthz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 518.648244ms
[api-check] Waiting for a healthy API server. This can take up to 4m0s
```

```
aws Services Search [Alt+S] Sydney bhumishap
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests"
[kubelet-check] Waiting for a healthy kubelet at http://127.0.0.1:10248/healthz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 518.648244ms
[api-check] Waiting for a healthy API server. This can take up to 4m0s
[api-check] The API server is healthy after 10.001658622s
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config" in namespace kube-system with the configuration for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node ip-172-31-14-85.ap-southeast-2.compute.internal as control-plane by adding the labels: [node-role.kubernetes.io/control-plane node.kubernetes.io/exclude-from-external-load-balancers]
[mark-control-plane] Marking the node ip-172-31-14-85.ap-southeast-2.compute.internal as control-plane by adding the taints [node-role.kubernetes.io/control-plane:NoSchedule]
[bootstrap-token] Using token: 61ysht.48enn4gmnhof6ex8
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to get nodes
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials
[bootstrap-token] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy
```

Your Kubernetes control-plane has initialized successfully!

8. On successful initialization we need to copy and paste the following commands on the master machine itself:

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
```

9. Next copy and paste the join link in the worker nodes so that the worker nodes can join the cluster.

Then you can join any number of worker nodes by running the following on each as root:

```
kubeadm join 172.31.14.85:6443 --token 6lysht.48enn4gmnhof6ex8 \
--discovery-token-ca-cert-hash sha256:461819c971fe032e04a78e18fde8e28755825e8468d468a2c86d88c52dba4945
```

10. After performing join commands on the worker nodes, we will get following output:

This node has joined the cluster:

- * Certificate signing request was sent to apiserver and a response was received.
- * The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

11. Once again when you run kubectl get nodes you will now see all 3 nodes have joined the cluster.

NAME	STATUS	ROLES	AGE	VERSION
ip-172-31-85-89.ec2.internal	NotReady	control-plane	119s	v1.26.0
ip-172-31-89-46.ec2.internal	NotReady	<none>	19s	v1.26.0
ip-172-31-94-70.ec2.internal	NotReady	<none>	12s	v1.26.0

Conclusion:

This experiment successfully demonstrated the creation of a Kubernetes cluster and the successful addition of all three nodes using various commands. Errors encountered during initialization can be addressed in two ways: 1) by ignoring the errors, or 2) by upgrading the instance type to t3.medium or t3.large if the issues are due to insufficient memory or CPU resources.