

Setting up K8s cluster on ubuntu 22.04 using kubeadm tool

First you need to create ubuntu 22.04 ec2-instance on aws

Ec2-instance configuration for master node

AWS AMI ==> Ubuntu 22.04 LTS

Instance type ==> t2.medium

Storage ==> 25 GB



Amazon Machine Image (AMI)

Ubuntu Server 22.04 LTS (HVM), SSD Volume Type Free tier eligible

ami-0e001c9271cf7f3b9 (64-bit (x86)) / ami-058b428b3b45defec (64-bit (Arm))

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

Description

Canonical, Ubuntu, 22.04 LTS, amd64 jammy image build on 2024-04-11

Architecture

64-bit (x86) ▼

AMI ID

ami-0e001c9271cf7f3b9

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: 0.1064 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](#)

▼ Network settings [Info](#)

[Edit](#)

Network [Info](#)

vpc-001d21e090efcc69a

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-5' with the following rules:

☒ Allow SSH traffic from

Helps you connect to your instance

Anywhere

0.0.0.0/0

☐ 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

Click on edit and select on

VPC - required [Info](#)

vpc-001d21e090efcc69a
172.31.0.0/16

(default) ▼



Subnet [Info](#)

No preference



[Create new subnet](#)

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

Security group name - required

launch-wizard-5

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and _-:/()#,@[]+=&:{}!\$*

Description - required [Info](#)

launch-wizard-5 created 2024-05-23T19:37:55.753Z

Click on drop down and select subnet

▼ **Network settings** [Info](#)

VPC - *required* [Info](#)

vpc-001d21e090efcc69a
172.31.0.0/16

(default) ▼

↻

Subnet [Info](#)

No preference ▲

↻ [Create new subnet](#) [↗](#)

Q |

No preference ✓

subnet-0759ec4448eeb23fa

VPC: vpc-001d21e090efcc69a Owner: 637423429912 Availability Zone: us-east-1e
IP addresses available: 4089 CIDR: 172.31.48.0/20)

subnet-042282dd94f325cd1

VPC: vpc-001d21e090efcc69a Owner: 637423429912 Availability Zone: us-east-1a
IP addresses available: 4091 CIDR: 172.31.16.0/20)

subnet-0b67116a0c2d6fd46

VPC: vpc-001d21e090efcc69a Owner: 637423429912 Availability Zone: us-east-1f
IP addresses available: 4091 CIDR: 172.31.64.0/20)

subnet-0df04889937ffc88a

VPC: vpc-001d21e090efcc69a Owner: 637423429912 Availability Zone: us-east-1d
IP addresses available: 4091 CIDR: 172.31.80.0/20)

subnet-0df4e3e9cba2ae964

VPC: vpc-001d21e090efcc69a Owner: 637423429912 Availability Zone: us-east-1b
IP addresses available: 4091 CIDR: 172.31.32.0/20)

subnet-0f525ec94a97cc910

VPC: vpc-001d21e090efcc69a Owner: 637423429912 Availability Zone: us-east-1c
IP addresses available: 4091 CIDR: 172.31.0.0/20)

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0)

Remove

Note:- You need to remember the subnet you have selected because we will use the same subnet for the worker node as well, so that communication between the master and worker nodes can occur.

After this configure your storage and click on launch instance

▼ **Configure storage** [Info](#) Advanced

1x 30 GiB gp2 Root volume (Not encrypted)

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage

[Add new volume](#)

The selected AMI contains more instance store volumes than the instance allows. Only the first 0 instance store volumes from the AMI will be accessible from the instance

Click refresh to view backup information [Refresh](#)

The tags that you assign determine whether the instance will be backed up by any Data Lifecycle Manager policies.

0 x File systems [Edit](#)

Virtual server type (instance type)
t2.medium

Firewall (security group)
-

Storage (volumes)
1 volume(s) - 30 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 IOs, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

[Cancel](#) [Launch instance](#)

After launching your master node instance, connect to it via SSH. You can also use software like MobaXterm or PuTTY for this.

Now, we begin configuring our master node.

Now, we need to load the kernel modules for Kubernetes: overlay and br_netfilter. To load these modules temporarily, run the following commands:

```
# Enabling kernel modules (overlay and br_netfilter)
```

```
sudo modprobe overlay
```

```
sudo modprobe br_netfilter
```

To load the modules permanently, create a config file in `/etc/modules-load.d`. Run the following command:

```
[root@k8master ~]# vim /etc/modules-load.d/k8s.conf
```

In this file, enter the following two lines:

```
overlay
```

```
br_netfilter
```

To verify the config file, run:

```
[root@k8master ~]# cat /etc/modules-load.d/k8s.conf
```

```
overlay
```

```
br_netfilter
```

Run this command to verify whether modules are loaded or not

```
[root@k8master ~]# lsmod | grep br_
```

```
[root@k8master ~]# lsmod | grep over
```

To create a configuration file for iptables, run the following command in your terminal:

```
[root@k8master ~]# cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
EOF
```

To verify whether rules are added or not run this command

```
[root@k8master ~]# sysctl --system
```

```
[root@k8master ~]# sysctl --system
* Applying /usr/lib/sysctl.d/10-default-yama-scope.conf ...
* Applying /usr/lib/sysctl.d/50-coredump.conf ...
* Applying /usr/lib/sysctl.d/50-default.conf ...
* Applying /usr/lib/sysctl.d/50-libkcap-optmem_max.conf ...
* Applying /usr/lib/sysctl.d/50-pid-max.conf ...
* Applying /usr/lib/sysctl.d/50-redhat.conf ...
* Applying /etc/sysctl.d/99-sysctl.conf ...
* Applying /etc/sysctl.d/k8s.conf ...
* Applying /etc/sysctl.conf ...
kernel.yama.ptrace_scope = 0
kernel.core_pattern = |/usr/lib/systemd/systemd-coredump %P %u %g %s %t %c %h
kernel.core_pipe_limit = 16
fs.suid_dumpable = 2
kernel.sysrq = 16
kernel.core_uses_pid = 1
net.ipv4.conf.default.rp_filter = 2
net.ipv4.conf.ens160.rp_filter = 2
net.ipv4.conf.lo.rp_filter = 2
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.conf.ens160.accept_source_route = 0
net.ipv4.conf.lo.accept_source_route = 0
net.ipv4.conf.default.promote_secondaries = 1
net.ipv4.conf.ens160.promote_secondaries = 1
net.ipv4.conf.lo.promote_secondaries = 1
net.ipv4.ping_group_range = 0 2147483647
net.core.default_qdisc = fq_codel
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
fs.protected_regular = 1
fs.protected_fifos = 1
net.core.optmem_max = 81920
kernel.pid_max = 4194304
kernel.kptr_restrict = 1
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.ens160.rp_filter = 1
net.ipv4.conf.lo.rp_filter = 1
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
```

Disabling Swap Memory

Run this command to disable swap memory

```
[root@k8master ~]# swapoff -a
```

To disable swap permanently, open the `/etc/fstab` file and comment out the swap entry.

```
# Units generated from this file.
#
/dev/mapper/rl-root / xfs defaults 0 0
UUID=63aa319a-771c-4d57-b812-d1c51d5c34a6 /boot xfs defaults 0 0
# /dev/mapper/rl-swap none swap defaults 0 0
```

Now, we need to add ports to our firewall or AWS security group. The Kubernetes master node requires the following ports:

TCP Ports: 6443, 2379, 2380, 10250, 10251, 10252, 10257, 10259, 179

UDP Port: 4789

In my case, I'm using an AWS security group to allow Kubernetes ports.

To do this, navigate to your AWS web console, select your master node, and click on "Security". Then, in the security section, click on the security group associated with your master node.

The screenshot shows the AWS Management Console interface. At the top, the 'Instances' page is displayed with a table of instances. The 'control-plane' instance is selected, and its details are shown below. The 'Security' tab is active, showing the security group 'sg-0304cdd851d6a3f5a (launch-wizard-2)' associated with the instance. Red circles and boxes highlight the instance, the Security tab, and the security group.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
worker-node	i-06f154c208d46f621	Stopped	t2.small	1	View alarms +	us-east-1e	-
control-plane	i-037bb54306b20296a	Stopped	t2.medium	-	View alarms +	us-east-1e	-

i-037bb54306b20296a (control-plane)

Details | Status and alarms New | Monitoring | **Security** | Networking | Storage | Tags

▼ Security details

IAM Role: -

Security groups: **sg-0304cdd851d6a3f5a (launch-wizard-2)**

Owner ID: 637423429912

Launch time: Thu May 23 2024 10:00:00

After selecting the security group, you'll need to edit the inbound rules. To do this, click on "Edit rules".

Details

Security group name launch-wizard-2	Security group ID sg-0304cdd851d6a3f5a	Description launch-wizard-2 created 2024-05-21T12:58:20.223Z	VPC ID vpc-001d21e090efcc69a
Owner 637423429912	Inbound rules count 7 Permission entries	Outbound rules count 1 Permission entry	

Inbound rules | Outbound rules | Tags

Inbound rules (7)

Search

Manage tags **Edit inbound rules**

Add all of these rules

Inbound rules Info

Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info	Description - optional Info
sg-012e54031a10e63dd	SSH	TCP	22	Custom	0.0.0.0
sg-04ea40e024c0a5c2d	All ICMP - IPv4	ICMP	All	Custom	0.0.0.0
sg-055b338419dcf9454	Custom TCP	TCP	10250 - 10260	Custom	0.0.0.0
sg-017ea200bc41991c5	Custom TCP	TCP	2379 - 2380	Custom	0.0.0.0
sg-0f3764f7467e4da4f	Custom UDP	UDP	4789	Custom	0.0.0.0
sg-0b44508ab70f4936b	Custom TCP	TCP	179	Custom	0.0.0.0
sg-00f6f2236a0b43791	Custom TCP	TCP	6443	Custom	0.0.0.0

Add rule

We have completed all the prerequisites for setting up the Kubernetes cluster. Now, we'll proceed with setting up the container runtime and Kubernetes packages such as kubeadm, kubelet, and kubectl.

We'll be using CRI-O as the container runtime. Here are the steps to set it up:

Install dependencies for adding the repositories

apt-get update

apt-get install -y software-properties-common curl

Add the CRI-O repository

```
curl -fsSL https://pkgs.k8s.io/addons:/cri-o:/prerelease:/main/deb/Release.key | gpg --dearmor -o /etc/apt/keyrings/cri-o-apt-keyring.gpg
```

```
echo "deb [signed-by=/etc/apt/keyrings/cri-o-apt-keyring.gpg]  
https://pkgs.k8s.io/addons:/cri-o:/prerelease:/main/deb/ /" | tee /etc/apt/sources.list.d/cri-o.list
```

Add the Kubernetes repository

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

```
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]  
https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /" | tee /etc/apt/sources.list.d/kubernetes.list
```

Install the packages

```
apt-get update  
apt-get install -y cri-o kubelet kubeadm kubectl
```

Now start and enable cri-o

```
systemctl start crio.service  
systemctl status cri-o
```

Run the following command to disable automatic updates for Kubernetes packages kubelet, kubeadm, and kubectl. This prevents version skew between Kubernetes packages.

```
apt-mark hold kubelet kubeadm kubectl
```

Adding user in your master node

```
[root@k8master ~]# useradd kiosk  
[root@k8master ~]# passwd kiosk  
Changing password for user kiosk.  
New password:  
BAD PASSWORD:  
The password is shorter than 8 characters Retype new password:  
passwd: all authentication tokens updated successfully.
```

Giving sudo access to kiosk user

```
[root@k8master ~]# vim /etc/sudoers
```


=====

All configurations for the master are done. Now, we move towards worker node configurations. In my case, only 1 worker node is available. Repeat all of these steps for the worker node.

=====

NOTE:- All the steps are the same for the worker node. The only modification required is adjusting the security group rules. Additionally, you do not need to add the 'kiosk' user in the worker node.

Next, we need to add ports to the firewall for the worker nodes. Kubernetes requires the following ports:

TCP Ports: 179, 10250, 30000-32767

UDP Port: 4789

Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info	
sgr-0e700a55089d2430a	Custom UDP	UDP	4789	Custom	<input type="text" value="0.0.0.0/0"/>
sgr-015b07a4192acb2ec	Custom TCP	TCP	10250	Custom	<input type="text" value="0.0.0.0/0"/>
sgr-01d8db31e31a91639	Custom TCP	TCP	30000 - 32767	Custom	<input type="text" value="0.0.0.0/0"/>
sgr-0de5b14ac1fd40689	SSH	TCP	22	Custom	<input type="text" value="0.0.0.0/0"/>
sgr-0e3dfdb3256ed3da9	Custom TCP	TCP	179	Custom	<input type="text" value="0.0.0.0/0"/>
sgr-0abaae86234e1d276	All ICMP - IPv4	ICMP	All	Custom	<input type="text" value="0.0.0.0/0"/>

[Add rule](#)

With all configurations completed for both the master and worker nodes, it's time to initialize our Kubernetes master node.

To initialize your Kubernetes master node, run the following command on your master node:

[root@k8master ~]# kubeadm init

After completing this command you will get this type of output on your terminal

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 192.168.43.156:6443 --token epiv7r.q6nkchryz72ar9jl \  
--discovery-token-ca-cert-hash sha256:e14c31f6d21c9495b4ddf941d9cacbb63ac7220e  
2d364d8baee624af422d0713
```

Save this output to your notepad file this is very important of do not forget to save this

Before joining worker nodes run this commands on master node as kiosk user

```
[kiosk@k8master ~]$ mkdir -p $HOME/.kube
```

```
[kiosk@k8master ~]$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

```
[kiosk@k8master ~]$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

After this, it's time to run the `kubeadm join` command on your worker node.

Once your worker node successfully joins the master node, you can switch to the master node and run the command `kubectl get nodes` using a normal user account.

Now, we'll configure the CNI plugin for Kubernetes. In my case, I'm using the Calico CNI plugin. Head to your master node and follow these steps.

To set up the Calico plugin, follow these steps on your master node:

Download Calico release v3.26.3 from this link: [Calico Release v3.26.3](https://github.com/projectcalico/calico/releases/download/v3.26.3/release-v3.26.3.tgz), or run the following command with the 'kiosk' user on your master node:

```
[kiosk@k8master ~]$ wget  
https://github.com/projectcalico/calico/releases/download/v3.26.3/release-v3.26.3.tgz
```

After downloading, extract the file using the following command:

```
[kiosk@k8master ~]$ tar -xvf /home/kiosk/release-v3.26.3.tgz
```

Verify that the file is extracted by running:

```
[kiosk@k8master ~]$ ls  
  
release-v3.26.3 release-v3.26.3.tgz
```

Navigate to the extracted directory:

```
[kiosk@k8master ~]$ cd release-v3.26.3/
```

Inside the directory, you'll find 'bin', 'images', and 'manifests'.

Navigate to the 'manifests' directory:

```
[kiosk@k8master release-v3.26.3]$ cd manifests/
```

Verify your location:

```
[kiosk@k8master manifests]$ pwd  
  
/home/kiosk/release-v3.26.3/manifests
```

Finally, apply the Calico YAML manifest file on the master node using the following command. Make sure not to run this command on the worker node

```
[kiosk@k8master manifests]$ kubectl apply -f /home/kiosk/release-v3.26.3/manifests/calico.yaml
```

Run these commands to verify your cluster:

```
[kiosk@k8master]$ kubectl get pods -A
```

```
[kiosk@k8master]$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
k8master.example.com	Ready	master	72m	v1.28.4
worker1.example.com	Ready	<none>	45m	v1.28.4

=====Cluster setup complete. Ready to deploy!=====