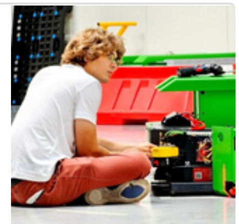
Kubernetes Setup: Install Strategies & Deploy Your Cluster on AWS

Check GitHub for helpful DevOps tools:

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics



Ask Personal AI Document assistant to learn interactively (FASTER)!

1 Download PDF

2 Go to website

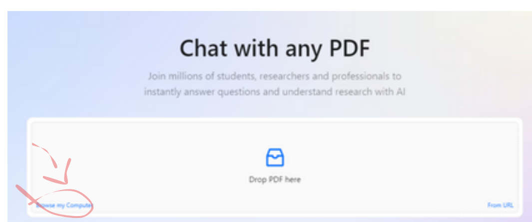
3 Browse file

4 Chat with Document

1 https://github.com/MichaelRobotics/DevOpsTools/blob/main/KubernetesSetup.pdf

Click there to go to ChatPdf website 2

3



Ask questions about document! 4

Complety new to Linux and Networking?

Essential for this PDF is a thorough knowledge of networking. I highly recommend the HTB platform's networking module, which offers extensive information to help build a comprehensive understanding.

HTB - Your Cyber Performance Center

We provide a human-first platform creating and maintaining high performing cybersecurity individuals and organizations.

https://www.hackthebox.com/

What is Kubernetes?

Kubernetes is an open-source platform that automates the deployment, scaling, and management of containerized applications. It helps manage clusters of nodes running containers, ensuring efficient and reliable operation.

How Kubernetes clusters are made?

Kubernetes clusters consist of a control plane and multiple worker nodes. The control plane manages cluster operations, while worker nodes run the actual container workloads.

Why and When use Kubernetes

Kubernetes is ideal for deploying scalable, resilient, and automated containerized applications. It is used when managing multiple containers across different environments is necessary.

Example: Running a microservices-based e-commerce platform that scales up during peak hours.

System Requirements

- RAM: 2 GB per node (1 GB can work for testing but may lead to limited performance)
- 10 GB free storage
- Ubuntu

Kubernetes: Main components & packages

- **kube-apiserver:** Central management component that exposes the Kubernetes API; acts as the front-end for the cluster.
- **etcd:** Distributed key-value store for storing all cluster data, ensuring data consistency across nodes.
- **kube-scheduler:** Assigns pods to available nodes based on resource requirements and policies.
- **kube-controller-manager:** Manages core controllers that handle various functions like node status, replication, and endpoints.
- **kubelet:** Agent that runs on each node, responsible for managing pods and their containers.
- **kube-proxy:** Manages networking on each node, ensuring communication between pods and services within the cluster.

Kubernetes Setup: Different Instalation strategies

Kubernetes (K8s) has become the go-to platform for orchestrating containerized applications, but choosing the right installation strategy depends on your use case

1) POC/Local Installation: KIND, k3s, Minikube

These solutions allow developers to quickly spin up a Kubernetes environment on a local machine without the need for cloud infrastructure.

- **KIND (Kubernetes in Docker):** Runs Kubernetes clusters in Docker containers.
- **k3s:** Lightweight Kubernetes for edge, IoT, and low-resource setups.
- **Minikube:** Creates a local, single-node Kubernetes cluster.

Pros

- **Quick Setup**: Easy install and ideal for fast dev cycles.
- **Lightweight**: Optimized for low-resource use (e.g., laptops, small servers).
- **Great for Learning**: Test Kubernetes features without full-scale deployments.

Cons

- **Limited Scalability**: Not for large-scale production.
- **Resource Constraints**: Limited by local hardware.
- **Networking Challenges**: Difficult to simulate complex setups.

2) Self-Managed Kubernetes: Vagrant/VirtualBox, VMs on Cloud, BareM/VMware

For teams that require more control over their Kubernetes clusters, self-managed setups can provide the flexibility needed to customize and optimize the environment

- **Vagrant/VirtualBox**: Create local VMs for small-scale Kubernetes testing.
- **Cloud VMs**: Manually set up Kubernetes on AWS, Azure, or Google Cloud.
- **Bare Metal/VMware**: Deploy directly on physical servers or virtualized setups.

Pros:

- **Full Control**: Customize everything from networking to storage.
- **Choice of Infrastructure**: Flexibility to run on any hardware or cloud.
- **Better Resource Use**: Efficient on bare metal compared to VMs.

Cons:

- **Complex Setup**: Requires expertise; manual scaling and updates.
- **Limited Managed Features**: More effort needed for monitoring and scaling.
- **Scaling Issues**: Manual scaling can be time-consuming and error-prone.

3) Managed Kubernetes on Cloud: EKS, AKS, GKE

Managed Kubernetes services from major cloud providers are ideal for production environments where scalability, reliability, and ease of use are key.

- **EKS (Elastic Kubernetes Service):** Managed k8s service on AWS.
- **AKS (Azure Kubernetes Service):** Managed k8s service on Azure.
- **GKE (Google Kubernetes Engine):** Managed k8s service on Google Cloud.

Pros:

- **Ease of Use**: Simplifies setup; no need to manage control planes.
- **Scalability**: Auto-scaling adjusts to workload demands.
- **High Availability**: Distributed across multiple zones for uptime.

Cons:

- **Cost**: Can be expensive, especially at scale.
- **Limited Customization**: Less control over infrastructure.
- **Vendor Lock-In**: Tied to the cloud provider's ecosystem.
- **Potential Overhead**: Limited visibility due to abstraction.

Kubernetes Setup: Network configuration

When running Kubernetes in an environment with strict network boundaries, such as on-premises datacenter with physical network firewalls or Virtual Networks in Public Cloud, it is useful to be aware of the ports and protocols used by Kubernetes components

Control plane

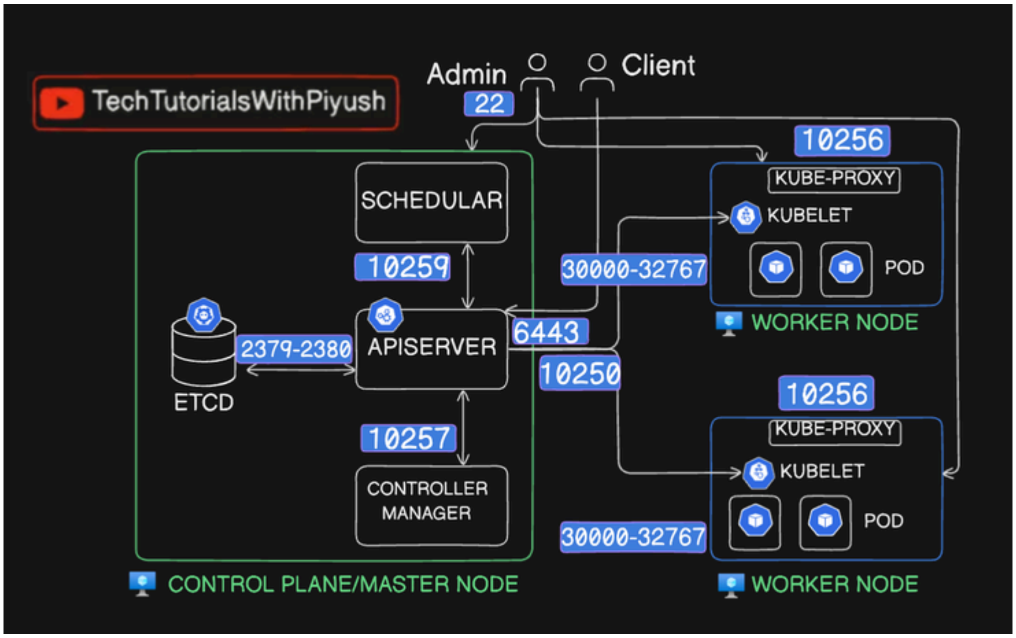
Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	6443	Kubernetes API server	All
TCP	Inbound	2379-2380	etcd server client API	kube-apiserver, etcd
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	10259	kube-scheduler	Self
TCP	Inbound	10257	kube-controller-manager	Self

Worker node(s)

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	10256	kube-proxy	Self, Load balancers
TCP	Inbound	30000-32767	NodePort Services†	All

All default port numbers can be overridden. When custom ports are used those ports need to be open instead of defaults mentioned here.

Visual representation of kubernetes components and how they talk with each other (Shared by TechTutorialsWithPiyush):

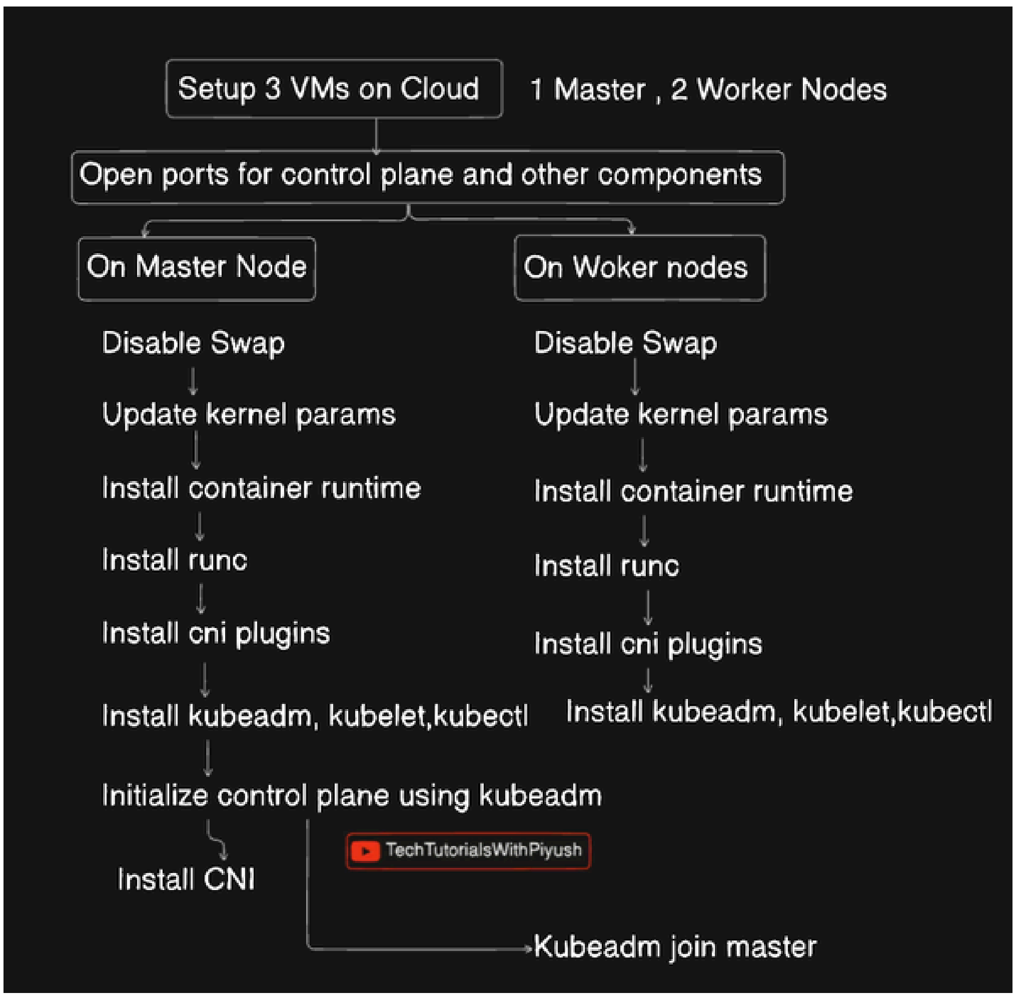


As an admin, you need access to the **API Server**, so ensure port **6443** is open. The kubelet should be accessible by the **API Server** on port **10250**, and **kube-proxy** needs to be reachable by admins on port **10256**. Internally, **etcd** communicates only with the **API Server** over ports **2379-2380**, while the scheduler uses port **10259** and the controller manager uses port **10257**. Applications exposed externally via load balancers will typically listen on **high-end** ports, ranging from **30000** to **32767**. For **SSH** access to the cluster, ensure port **22** is open.

Kubernetes Setup: Kubernetes installation steps

1) Intro

Unlike using tools like kind for Kubernetes installation, this approach involves installing all components separately. Setting up Kubernetes manually is a complex process, and it's easy to make mistakes, so proceed carefully. Below is an installation guide provided by TechTutorialsWithPiyush.



2) Setup VM's

Start by creating and configuring the necessary virtual machines for the Kubernetes cluster. Typically, you need at least one master node and multiple worker nodes.

3) Open ports for control plan and other components

Ensure that required ports (e.g., 6443 for API server, 2379-2380 for etcd, 10250 for kubelet, and others) are open for communication between control plane components and nodes.

On Master Node:

4) Disable Swap

Turn off swap to ensure Kubernetes functions properly, as it expects predictable memory management.

5) Update kernel params

Update kernel parameters to optimize networking and enable features like IP forwarding, which are required by Kubernetes.

6) Install container runtime

Install a container runtime (like Docker, containerd, or CRI-O) to run containers.

7)Install runc

Install runc, a lightweight and low-level container runtime, to manage and run container processes.

8) Install cni plugins

Set up Container Network Interface (CNI) plugins to enable networking capabilities for pods within the Kubernetes cluster.

9) Install kubeadm, kubelet, kubectl,

Install kubeadm for initializing and managing the Kubernetes cluster, kubelet as the primary node agent, and kubectl for command-line interaction with the cluster.

10) Initialize control plane using kubeadm

Use kubeadm to set up the control plane components (API server, etcd, controller manager, scheduler).

11) Install CNI

Set up a Container Network Interface (CNI) plugin to handle networking within the cluster.

12) Generate tokens, so workers can join

Create tokens to allow worker nodes to join the cluster securely.

On worker nodes

13) Disable swap

Similar to the master node, disable swap for consistent performance.

14)Update kernel params

Apply the same kernel parameter updates to ensure proper networking and cluster operations.

Kubernetes Setup: AWS VPC & EC2 setup

1) Security Groups

On AWS, you need to create security groups; on Google Cloud, configure firewall rules; and on Azure, set up Network Security Groups (NSGs). These serve the same purpose—allowing or denying traffic on specific ports from designated sources.

ControlPlane security group:

Security group rule...	IP version	Type	Protocol	Port range	Source
sgr-03669ef841440f499	IPv4	Custom TCP	TCP	10248 - 10260	172.31.0.0/16
sgr-054d2072ebcf0d427	IPv4	Custom TCP	TCP	2379 - 2380	172.31.0.0/16
sgr-086c57b21eeaac530	IPv4	SSH	TCP	22	79.184.123.3/32
sgr-0954993a72f2953...	IPv4	Custom TCP	TCP	6443	172.31.0.0/16

Ports 1048-10260

Allow any inboud trafic from pods used by Controll plane components within VPC network (172.31.0.0/16)

Ports 2379-2380

Allow any inboud traffic from ETCD within VPC network (172.31.0.0/16)

Port 22

Allow Admin to ssh into cluster

Port 6443

Allow any inbond traffic from API server within VPC network (172.31.0.0/16)

WorkerNode security group:

Security group rule...	IP version	Type	Protocol	Port range	Source
sgr-0381221d6492aec4c	IPv4	Custom TCP	TCP	30000 - 32767	172.31.0.0/16
sgr-0dfc025d1ddbb5625	IPv4	Custom TCP	TCP	10250	172.31.0.0/16
sgr-0d63cb2ece1117d41	IPv4	SSH	TCP	22	79.184.133.11/32
sgr-029cbeb3b3650e6...	IPv4	Custom TCP	TCP	10256	172.31.0.0/16

Ports 1048-10260

Allow any inboud trafic from KubeProxy within VPC network (172.31.0.0/16)

Ports 30000 - 32767

Allow any inboud traffic from high end ports within VPC network (172.31.0.0/16)

Port 22

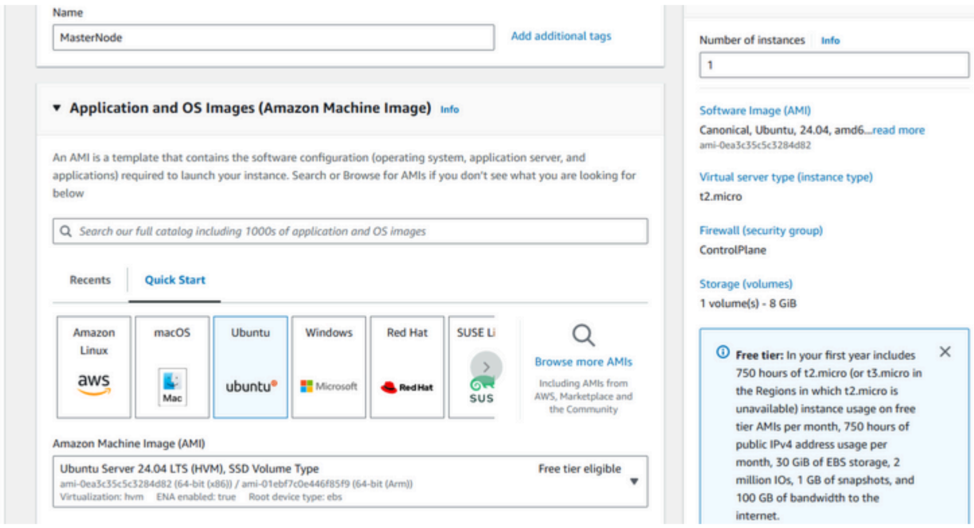
Allow Admin to ssh into cluster

Port 10256

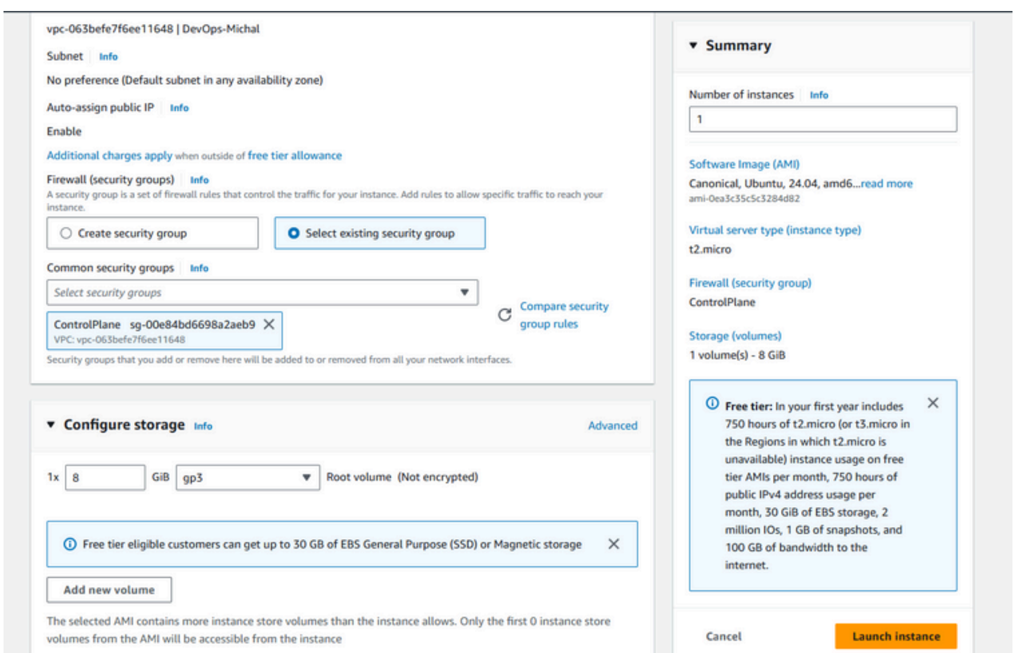
Allow any inbond traffic Kubelet API within VPC network (172.31.0.0/16)

2) Controll Plane VM

A single control plane instance with an Ubuntu system is recommended. A micro instance should be sufficient.

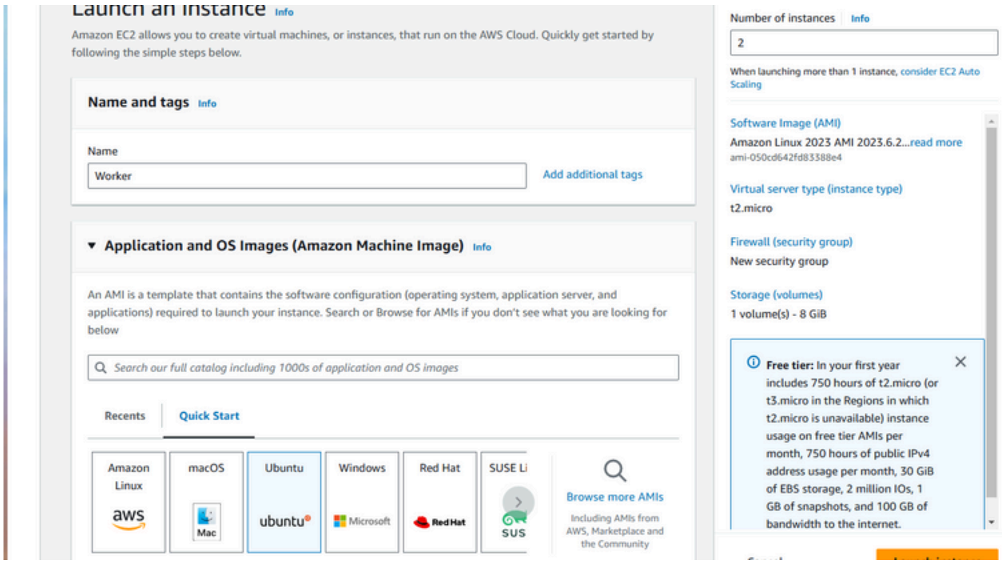


Use the existing ControlPlane security group and an existing SSH key pair to launch the instance.

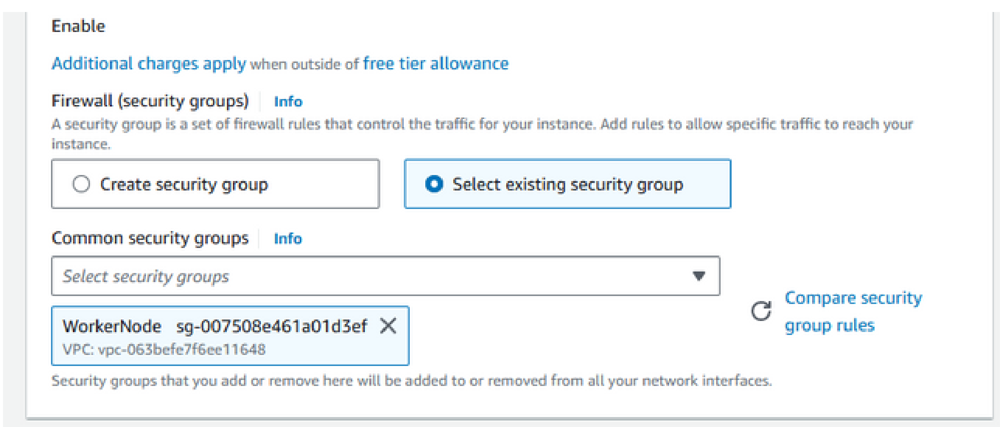


3) Worker Node

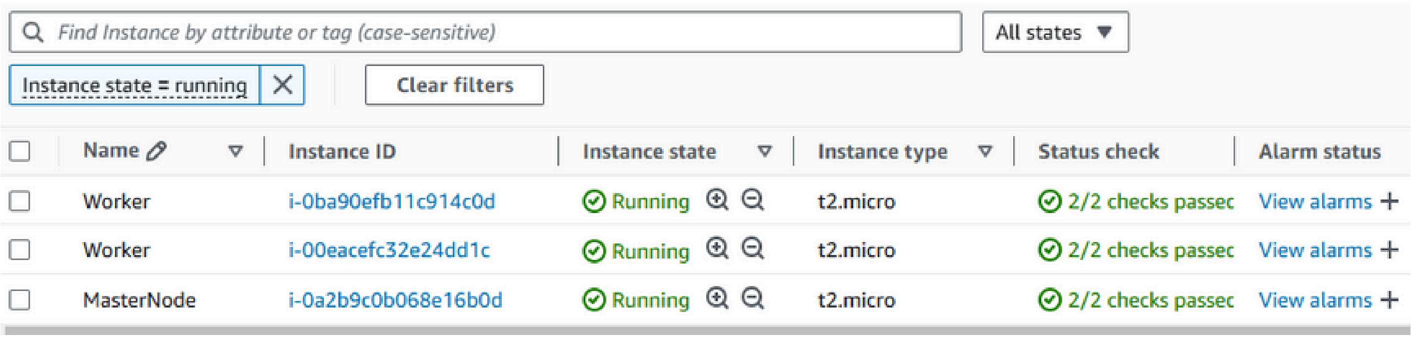
Launch 2 worker node instances with Ubuntu. Micro instances should be sufficient.



Use the created WorkerNode security group and either create a new or use an existing key pair to launch the instances.



4) EC2 state



	Name	Instance ID	Instance state	Instance type	Status check	Alarm status
	Worker	i-0ba90efb11c914c0d	Running	t2.micro	2/2 checks passec	View alarms +
	Worker	i-00eacefc32e24dd1c	Running	t2.micro	2/2 checks passec	View alarms +
	MasterNode	i-0a2b9c0b068e16b0d	Running	t2.micro	2/2 checks passec	View alarms +

Navigate towards EC2 instances Dashboard. It should look alike

Kubernetes Setup: Master Node

1) Disable Swap

```
swapoff -a
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

2) Update Kernel parameters

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter
```

3) sysctl params required by setup, params persist across reboots

```
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
```

4) Apply sysctl params without reboot

```
sudo sysctl --system
```

5) Verify that the br_netfilter, overlay modules are loaded by running the following commands

```
lsmod | grep br_netfilter
lsmod | grep overlay
```

6) Verify that the net.bridge.bridge-nf-call-iptables, net.bridge.bridge-nf-call-ip6tables, and net.ipv4.ip_forward system variables are set to 1 in your sysctl config by running the following command:

```
sysctl net.bridge.bridge-nf-call-iptables net.bridge.bridge-nf-call-ip6tables net.ipv4.ip_forward
```

7) Install container runtime

```
curl -LO https://github.com/containerd/containerd/releases/download/v1.7.14/containerd-1.7.14-linux-amd64.tar.gz
sudo tar Cxzvf /usr/local containerd-1.7.14-linux-amd64.tar.gz
curl -LO https://raw.githubusercontent.com/containerd/containerd/main/containerd.service
sudo mkdir -p /usr/local/lib/systemd/system/
sudo mv containerd.service /usr/local/lib/systemd/system/
sudo mkdir -p /etc/containerd
containerd config default | sudo tee /etc/containerd/config.toml
sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' /etc/containerd/config.toml
sudo systemctl daemon-reload
sudo systemctl enable --now containerd
```

8) Check that containerd service is up and running

```
systemctl status containerd
```

9) Install runc

```
curl -LO https://github.com/opencontainers/runc/releases/download/v1.1.12/runc.amd64
sudo install -m 755 runc.amd64 /usr/local/sbin/runc
```

10) install cni plugin

```
curl -LO https://github.com/containernetworking/plugins/releases/download/v1.5.0/cni-plugins-linux-amd64-v1.5.0.tgz
sudo mkdir -p /opt/cni/bin
sudo tar Cxzvf /opt/cni/bin cni-plugins-linux-amd64-v1.5.0.tgz
```

11) Install kubeadm, kubelet and kubectl

```
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl gpg
```

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo 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:/v1.29/deb/ /' | sudo tee
/etc/apt/sources.list.d/kubernetes.list
```

```
sudo apt-get update
sudo apt-get install -y kubelet=1.29.6-1.1 kubeadm=1.29.6-1.1 kubectl=1.29.6-1.1 --
allow-downgrades --allow-change-held-packages
sudo apt-mark hold kubelet kubeadm kubectl
```

12) Check kubeadm, kubelet and kubectl versions (to check if installed):

```
kubeadm version
kubelet --version
kubectl version --client
```

13) Configure crictl to work with containerd

```
sudo crictl config runtime-endpoint unix:///var/run/containerd/containerd.sock
```

14) initialize control plane

```
sudo kubeadm init --pod-network-cidr=192.168.0.0/16 --apiserver-advertise-address=172.31.89.68 --node-name master
```

15) Prepare kubeconfig

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

16) Install calico

```
kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.28.0/manifests/tigera-operator.yaml
```

```
curl https://raw.githubusercontent.com/projectcalico/calico/v3.28.0/manifests/custom-resources.yaml -O
```

```
kubectl apply -f custom-resources.yaml
```

Kubernetes Setup: Worker nodes

Perform steps 1-11 on both the nodes

Run the command generated in step 14 on the Master node which is similar to below

```
sudo kubeadm join 172.31.71.210:6443 --token xxxxx --discovery-token-ca-cert-hash sha256:xxx
```

If you forgot to copy the command, you can execute below command on master node to generate the join command again

```
kubeadm token create --print-join-command
```

Kubernetes Setup: Test

run on the master node

```
kubectl get nodes
```

it should return all the 3 nodes in ready status.

Also, make sure all the pods are up and running by using the command as below:

```
kubectl get pods -A
```

common troubleshooting

1) Pods Stuck in Pending State

- Cause: Insufficient resources or node selectors/taints blocking scheduling.
- Troubleshooting: Check kubectl describe pod <pod-name>. Adjust resource limits, node specs, or scale nodes.

2) Failed to Pull Image

- Cause: Wrong image name, permissions issues, or network problems.
- Troubleshooting: Verify image name and registry access. Ensure node connectivity and authentication.

3) CrashLoopBackOff State

- Cause: Container keeps restarting due to errors.
- Troubleshooting: Check logs with kubectl logs <pod-name> -c <container-name>. Verify environment variables and dependencies.

4) Node Not Ready

- Cause: Kubelet issues, network config errors, or resource exhaustion.
- Troubleshooting: Inspect kubectl describe node <node-name> and journalctl -u kubelet. Check resources and port availability.

5) Kubelet Fails to Start

- Cause: Configuration errors or missing dependencies.
- Troubleshooting: Review journalctl -u kubelet. Disable swap (swapoff -a) and set correct kernel parameters.

6) Pods Unable to Communicate Across Nodes

- Cause: CNI plugin issues or firewall restrictions.
- Troubleshooting: Confirm CNI setup via kubectl get pods -n kube-system. Ensure ports for inter-node communication are open.

Learn more about Kubernetes

Check Kubernetes and piyushsachdeva - great docs!

Setup a Multi Node Kubernetes Cluster

kubeadm is a tool to bootstrap the Kubernetes cluster

https://github.com/piyushsachdeva/CKA-2024/tree/main/Resources/Day27

Kubernetes Documentation

This section lists the different ways to set up and run Kubernetes

https://kubernetes.io/docs/setup/

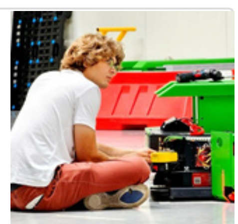
Share, comment, DM and check GitHub for scripts & playbooks created to automate process.

Check my GitHub

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics



PS.

If you need a playbook or bash script to manage KVM on a specific Linux distribution, feel free to ask me in the comments or send a direct message!