Project Report

on

Kubernetes cluster with High Availability and Scalability of web server.

Submitted in partial fulfillment for the award of

Post Graduate Diploma in High Performance Computing System Administration *from* ***C-DAC ACTS (Pune)***

Guidedby: Ms.Tejaswini Apate

Presented by :

Miss Pranjali Patil (*230940127009****)***

Miss Sushmita Diwakar (*230940127051****)***

Miss Sukanya Mane (*230940127005****)***

Mr. Trishna Dhruw (*230940127053****)***

Mr. Tarun Shori (*230940127052****)***

Centre of Development of Advanced Computing (C-DAC), Pune

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Miss Pranjali Patil

Miss Sushmita Diwakar

Miss Sukanya Mane

Mr. Trishna Dhruw

Mr. Tarun Shori

have successfully completed their project on

Kubernetes cluster with High Availability and scalability of web server

Under the Guidance of Ms. Tejaswini apate

Project Guide ***Project Supervisor***

HOD ACTS
Mr.

ACKNOWLEDGEMENT

This project " Kubernetes cluster with High Availability and scalability of web server" was a great learning experience for us and we are submitting this work to Advanced Computing Training School (CDAC ACTS).

We all are very glad to mention the name of Ms.Tejaswini apate for his valuable guidance to work on this project. Overcome various obstacles and intricacies during the course of project work.

We are highly grateful to HPC tech team (ACTS training Centre, C- DAC), For his valuable guidance and support whenever necessary while doing this course Post Graduate Diploma in High Performance Computing System Administration (PG- DHPCSA) Through CDAC ACTS , Pune.

Our most heartfelt thank goes to Ms. Swati salunkhe (Course Coordinator, PG- DHPCSA) who gave all the required support and kind coordination to provide all the necessities like required hardware, internet facility and extra Lab hours to complete the project and throughout the course up to the last day here in C-DAC ACTS, Pune.

From:

Miss Pranjali Patil (*230940127009****)***

Miss Sushmita Diwakar (*230940127051****)***

Miss Sukanya Mane (*230940127005****)***

Mr. Trishna Dhruw (*230940127053****)***

Mr. Tarun Shori (*230940127052****)***

TABLE OF CONTENTS

Abstract

This project focuses on deploying and managing a highly available web service on a Kubernetes cluster hosted on the AWS platform. The web service is built using the Apache HTTP Server (httpd) and is deployed using Kubernetes Deployments, Services, Persistent Volumes, and Persistent Volume Claims. The project aims to demonstrate the high availability and scalability features provided by Kubernetes in a real-world scenario.

The key components of the project include:

1. Setting up an AWS Kubernetes cluster: The project starts by provisioning an AWS Kubernetes cluster using tools self-managed Kubernetes on AWS EC2 instances.

2. Deploying the web service: The Apache HTTP Server (httpd) is containerized and deployed as a Kubernetes Deployment. Multiple replicas of the httpd Deployment are created to ensure high availability.

3. Configuring Persistent Volumes: Persistent Volumes (PVs) and Persistent Volume Claims (PVCs) are used to provide persistent storage for the web service. PVs are configured to store data on AWS EBS volumes or other suitable storage solutions.

4. Scaling the web service: The project demonstrates how to scale the number of replicas of the httpd Deployment dynamically based on demand. This showcases Kubernetes' ability to handle increased traffic and ensure consistent performance.

6. Testing high availability: To test the high availability of the web service, the project includes scripts to simulate node failures in the Kubernetes cluster. The scripts drain and uncordon nodes to observe how Kubernetes manages pod scheduling and maintains service availability.

Overall, this project serves as a practical demonstration of deploying and managing a highly available web service on Kubernetes using AWS infrastructure. It highlights the benefits of using Kubernetes for container orchestration and showcases its capabilities in ensuring service availability, scalability, and resilience.

Introduction

In today's era of cloud-native applications, ensuring high availability and scalability of web services is crucial for businesses to meet the demands of their users while maintaining reliability and performance. Kubernetes, as a powerful container orchestration platform, provides robust solutions for deploying, managing, and scaling applications in a cloud-native environment.

This This project entails the creation of a robust Kubernetes cluster deployed on AWS instances. Leveraging Docker containers within Kubernetes pods, the Apache Web server is orchestrated to ensure both high availability and scalability. Through meticulous configuration and management, this deployment architecture promises to elevate the reliability and efficiency of web services while maximizing resource utilization.

Use Cases

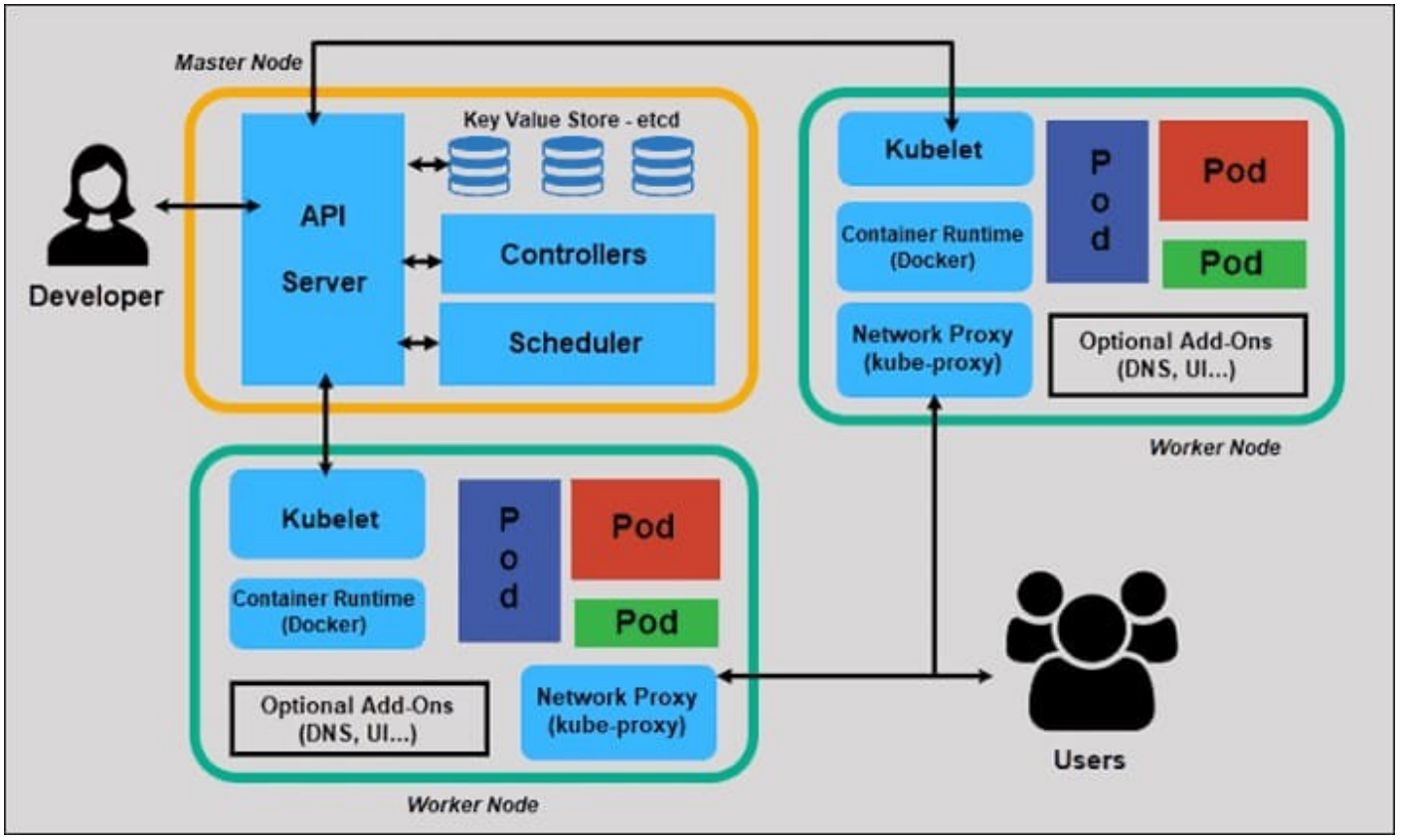
1. Implement High Availability:
   - Configured Kubernetes Pod Replication Controllers or Deployments to ensure redundancy.

2. Enable Scaling:
   - Implement Kubernetes Horizontal Pod Autoscaler (HPA) to automatically scale the web server based on demand.
   - Adjust HPA settings for CPU or memory utilization thresholds.

3. Configure Load Balancing:

   - Used Kubernetes Services to expose the web server, automatically creating an internal load balancer.

Workflow


Master Node
Key Value Store - etcd
API Server
Controllers
Scheduler
Developer
Kubelet
Container Runtime (Docker)
Network Proxy (kube-proxy)
Pod
Pod
Pod
Optional Add-Ons (DNS, UI...)
Worker Node
Kubelet
Container Runtime (Docker)
Network Proxy (kube-proxy)
Pod
Pod
Pod
Optional Add-Ons (DNS, UI...)
Worker Node
Users

System Requirements

For all the Nodes :

- **RAM: 4 GB**
- **Storage: 15 GB**
- **Processors: 2 cores**
- **OS : AWS Linux**

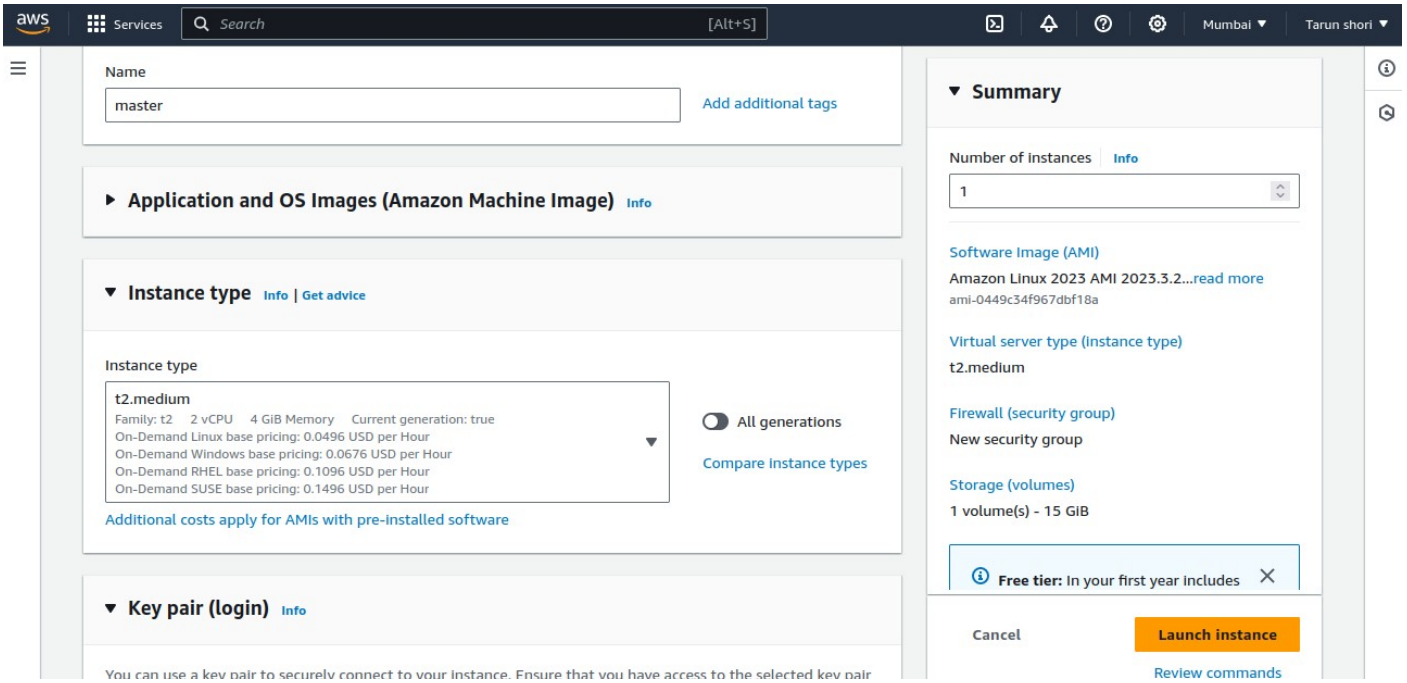
Software Requirements

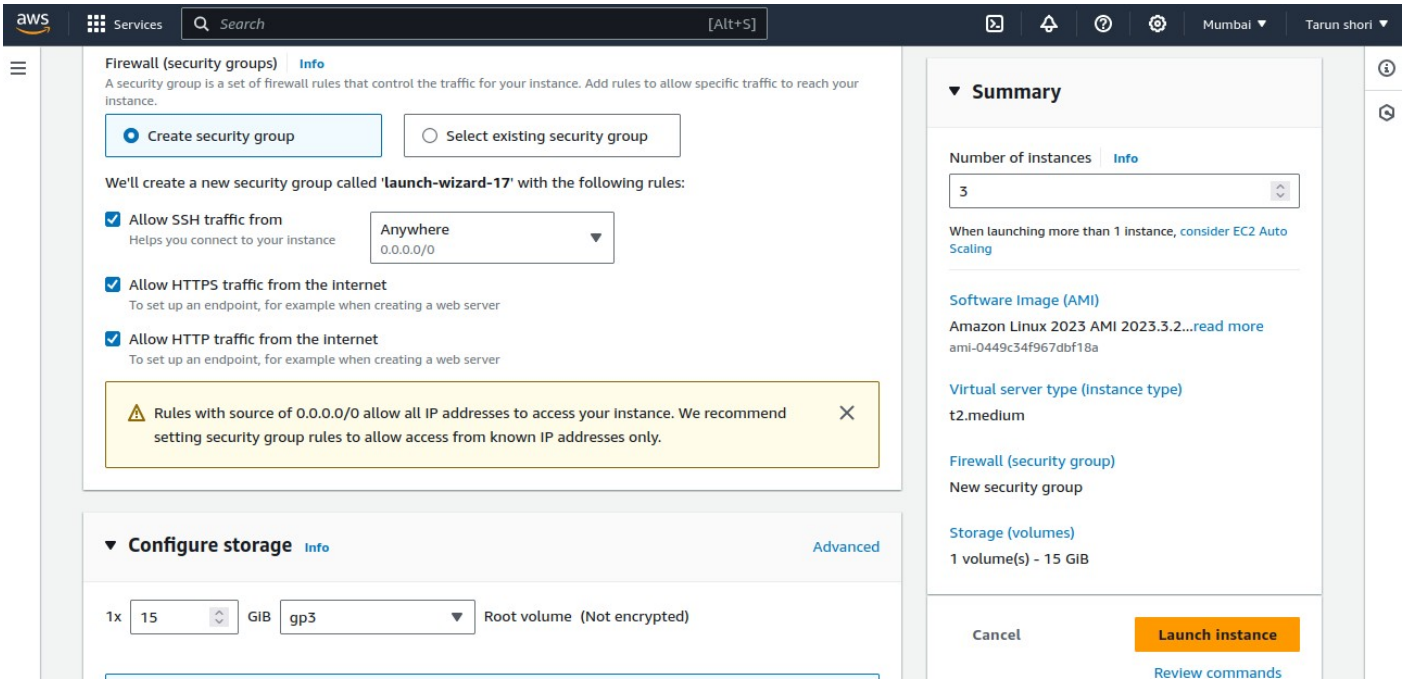
- **Kubernetes Tool - Kubeadm**
- **Container Runtime – Containerd**
- **Networking Plugin- Calico**
- **Repository – Docker hub**
- **Web server -Apache2 – Httpd**
- **Cloud Provider - AWS**

Setting up the AWS Instances

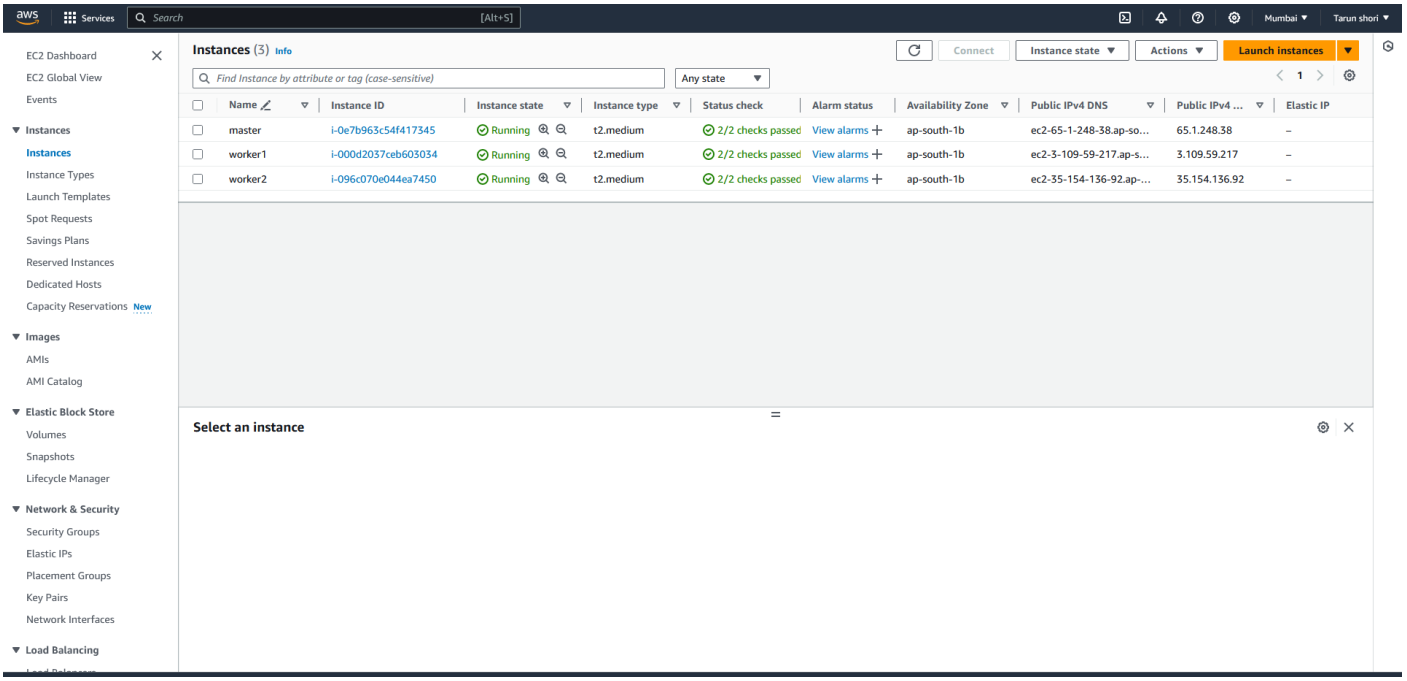
- create a Ec2 instance for master and worker nodes

Step 1: configure all the necessary configurations for Ec2 instances



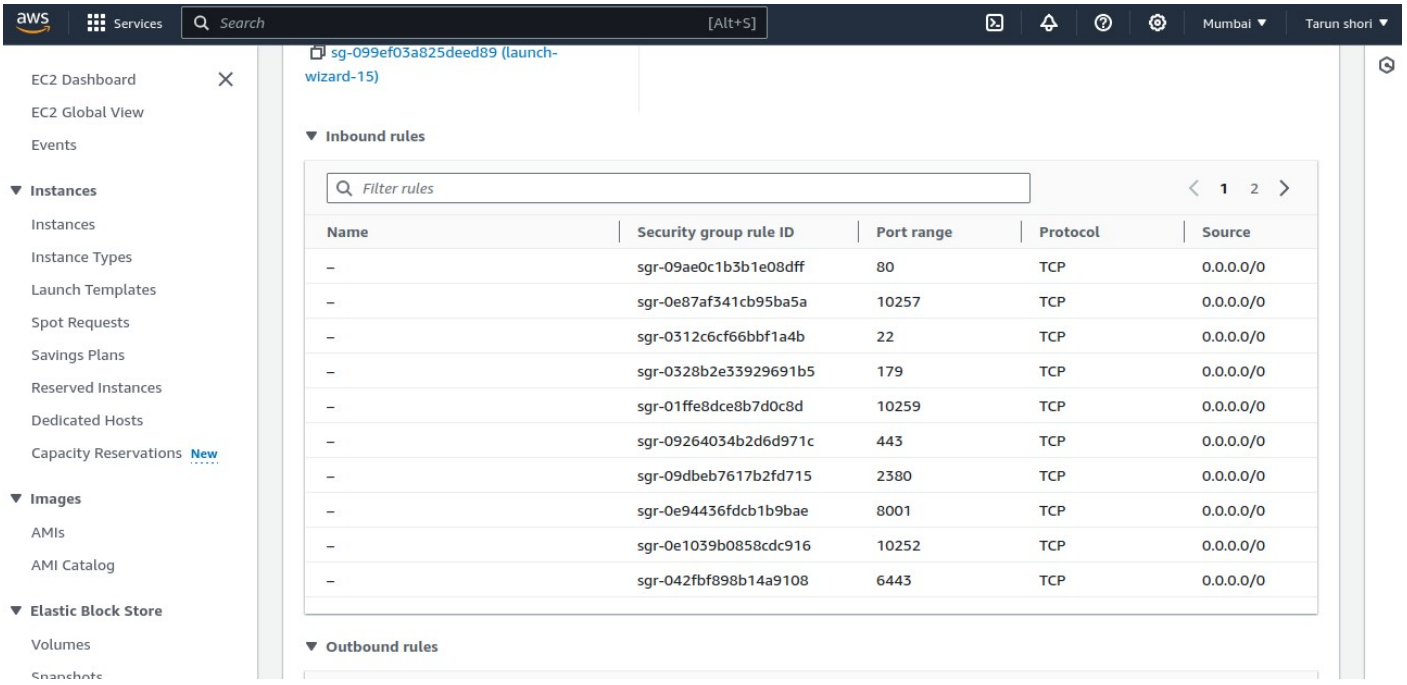


Step 2: Final step is to confirm the configuration and click Launch instance

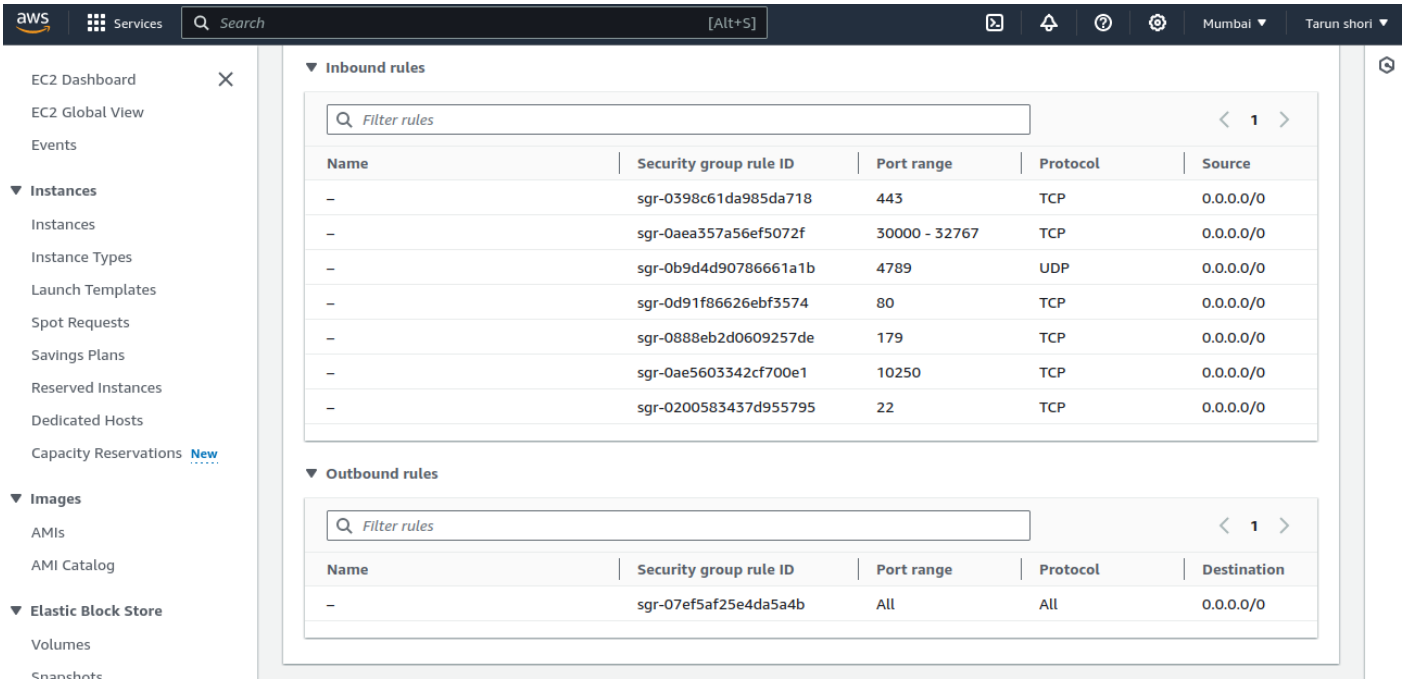


Step 3: Create Security group for instances for master and worker nodes.

For master node :



For worker nodes :



Connect the instances via SSH

```
onnection to ec2-13-127-92-200.ap-south-1.compute.amazonaws.com closed.
S C:\Users\dhpcsa\Downloads>
S C:\Users\dhpcsa\Downloads> ssh -i "master-k8s.pem" ec2-user@ec2-65-1-248-38.ap-south-1.compute.amazonaws.com
he authenticity of host 'ec2-65-1-248-38.ap-south-1.compute.amazonaws.com (65.1.248.38)' can't be established.
D25519 key fingerprint is SHA256:Iitv4X6CSpHykt2fMVlQ/av3aFERIegPUzsaXV+sXvk.
his key is not known by any other names
re you sure you want to continue connecting (yes/no/[fingerprint])? yes
arning: Permanently added 'ec2-65-1-248-38.ap-south-1.compute.amazonaws.com' (ED25519) to the list of known hosts.
       #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
ec2-user@ip-172-31-13-236 ~]$
ec2-user@ip-172-31-13-236 ~]$
ec2-user@ip-172-31-13-236 ~]$
```

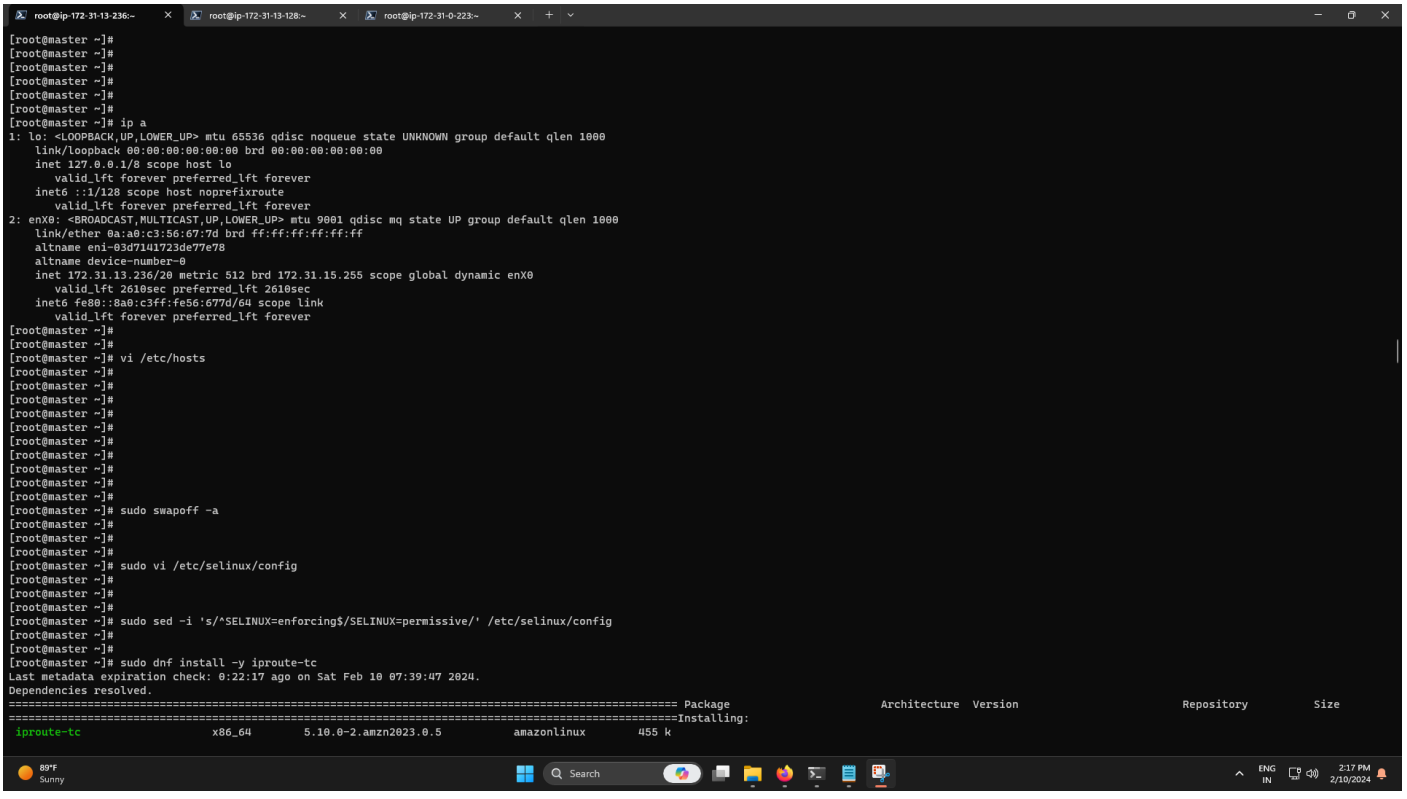
NOTE: Steps 1 to 6 should be applied to both the Master and the worker node.

Step 1) Disable swap space

For best performance, Kubernetes requires that swap is disabled on the host system. This is because memory swapping can significantly lead to instability and performance degradation.

To disable swap space, run the command:

$ sudo swapoff -a



To make the changes persistent, edit the /etc/fstab file and remove or comment out the line with the swap entry and save the changes.

$ sudo sed -i '/swap/d' /etc/fstab

Step 2) Disable SELinux

Additionally, we need to disable SELinux and set it to ‘permissive’ in order to allow smooth communication between the nodes and the pods.

To achieve this, open the SELinux configuration file.

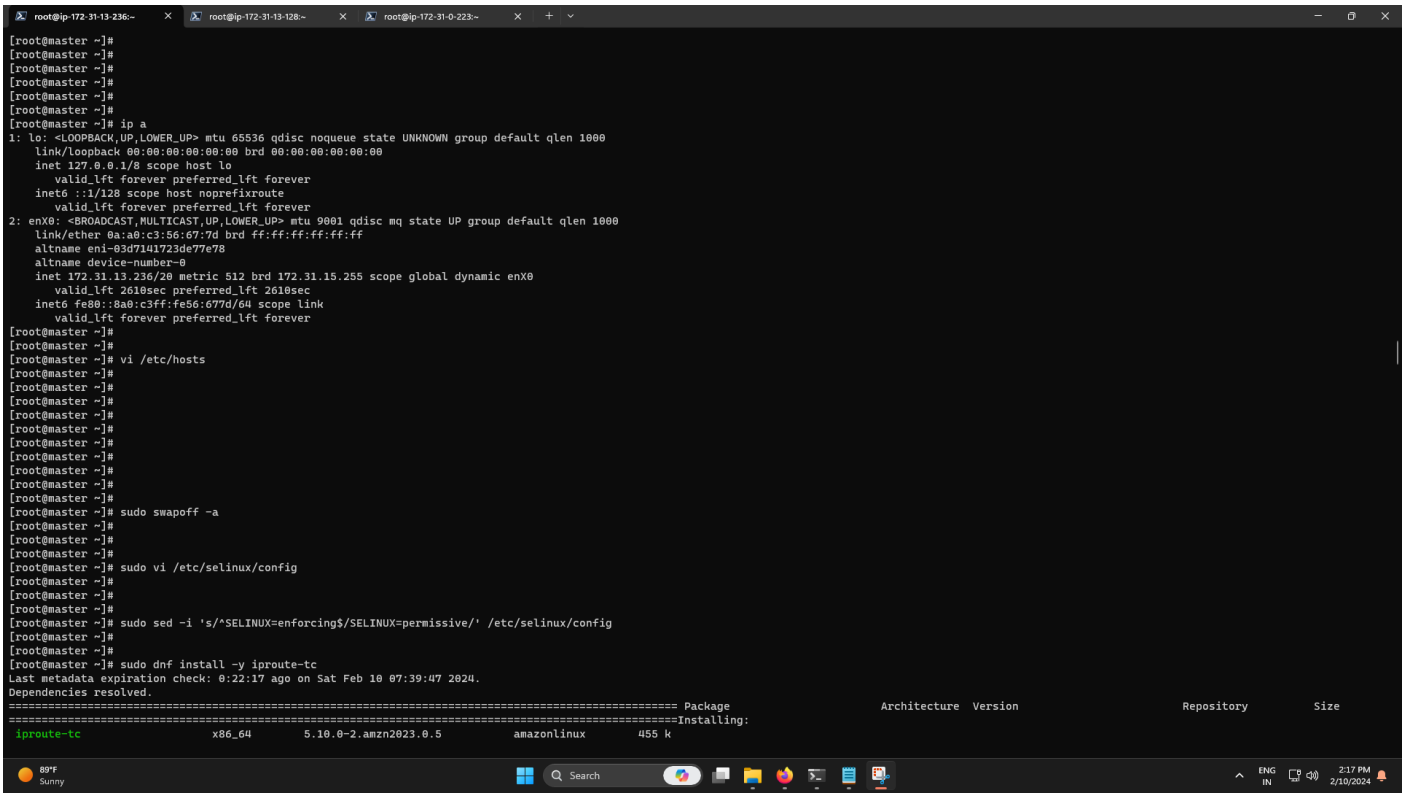
$ sudo vi /etc/selinux/config

Change the SELINUX value from enforcing to permissive.

SELINUX=permissive

Alternatively, you use the sed command as follows.

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



Step 3) Configure networking in master and worker node

Some additional network configuration is required for your master and worker nodes to communicate effectively. On each node, edit the /etc/hosts file.

$ sudo vi /etc/hosts

Next, update the entries as shown

172.31.13.236 master // For the Master node

172.31.13.128 worker1 // For the Worker node

172.31.0.223 worker2 // For the Worker node

```
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost6 localhost6.localdomain6



172.31.13.236 master
172.31.13.128 worker1
172.31.0.223 worker2
~
~
~
```

Save and exit the configuration file. Next, install the traffic control utility package:

$ sudo dnf install -y iproute-tc

```
[root@master ~]#
[root@master ~]#
[root@master ~]# sudo dnf install -y iproute-tc
Last metadata expiration check: 0:22:17 ago on Sat Feb 10 07:39:47 2024.
Dependencies resolved.
================================================================================ Package                    Architecture  Version
================================================================================Installing:
 iproute-tc                 x86_64       5.10.0-2.amzn2023.0.5          amazonlinux       455 k
Installing dependencies:
 iptables-libs              x86_64       1.8.8-3.amzn2023.0.2           amazonlinux       401 k
 libnetfilter_conntrack     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

Transaction Summary
================================================================================Install  4 Packages

Total download size: 943 k
Installed size: 3.0 M
Downloading Packages:
(1/4): libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64.rpm        1.0 MB/s |  58 kB     00:00
(2/4): libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64.rpm                 491 kB/s |  30 kB     00:00
(3/4): iptables-libs-1.8.8-3.amzn2023.0.2.x86_64.rpm                 5.0 MB/s | 401 kB     00:00
(4/4): iproute-tc-5.10.0-2.amzn2023.0.5.x86_64.rpm                   8.1 MB/s | 455 kB     00:00
--------------------------------------------------------------------------------Total
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                             1/1
  Installing       : libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64                   1/4
  Installing       : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64          2/4
  Installing       : iptables-libs-1.8.8-3.amzn2023.0.2.x86_64                   3/4
  Installing       : iproute-tc-5.10.0-2.amzn2023.0.5.x86_64                     4/4
  Running scriptlet: iproute-tc-5.10.0-2.amzn2023.0.5.x86_64                     4/4
  Verifying        : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64          1/4
  Verifying        : libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64                   2/4
  Verifying        : iptables-libs-1.8.8-3.amzn2023.0.2.x86_64                   3/4
  Verifying        : iproute-tc-5.10.0-2.amzn2023.0.5.x86_64                     4/4

Installed:
```

For seamless communication between the Master and worker node, configure the firewall and allow some pertinent ports and services as outlined below.

On Master node, allow following ports,

$ sudo firewall-cmd --permanent –add-port=6443/tcp

$ sudo firewall-cmd --permanent –add-port=2379-2380/tcp

$ sudo firewall-cmd --permanent –add-port=10250/tcp

$ sudo firewall-cmd --permanent –add-port=10251/tcp

$ sudo firewall-cmd --permanent –add-port=10252/tcp

$ sudo firewall-cmd --reload

```
sudo: firewall-cmd: command not found
[root@master ~]# yum install firewalld
Last metadata expiration check: 0:26:02 ago on Sat Feb 10 07:39:47 2024.
Dependencies resolved.
================================================================================ Package                         Architecture
==========================================================================================Installing:
 firewalld                      noarch       1.2.3-1.amzn2023             amazonlinux       452 k
Installing dependencies:
 firewalld-filesystem           noarch       1.2.3-1.amzn2023             amazonlinux        11 k
 gobject-introspection          x86_64       1.73.0-2.amzn2023.0.3        amazonlinux       255 k
 ipset                          x86_64       7.11-1.amzn2023.0.3          amazonlinux        40 k
 ipset-libs                     x86_64       7.11-1.amzn2023.0.3          amazonlinux        67 k
 iptables-nft                   x86_64       1.8.8-3.amzn2023.0.2         amazonlinux       183 k
 libnftnl                       x86_64       1.2.2-2.amzn2023.0.2         amazonlinux        84 k
 nftables                       x86_64       1:1.0.4-3.amzn2023.0.2       amazonlinux       400 k
 python3-firewall               noarch       1.2.3-1.amzn2023             amazonlinux       357 k
 python3-gobject-base           x86_64       3.42.2-2.amzn2023.0.3        amazonlinux       178 k
 python3-gobject-base-noarch    noarch       3.42.2-2.amzn2023.0.3        amazonlinux       154 k
 python3-nftables               x86_64       1:1.0.4-3.amzn2023.0.2       amazonlinux        18 k
Installing weak dependencies:
 libcap-ng-python3              x86_64       0.8.2-4.amzn2023.0.2         amazonlinux        30 k

Transaction Summary
==========================================================================================Install  13 Packages
```

```
Complete!
[root@master ~]# systemctl start firewalld
[root@master ~]# systemctl enable firewalld
[root@master ~]#
[root@master ~]#
[root@master ~]#
[root@master ~]#  sudo firewall-cmd --permanent --add-port=6443/tcp
 sudo firewall-cmd --permanent --add-port=2379-2380/tcp
 sudo firewall-cmd --permanent --add-port=10250/tcp
 sudo firewall-cmd --permanent --add-port=10251/tcp
 sudo firewall-cmd --permanent --add-port=10252/tcp
 sudo firewall-cmd --reload
success
success
success
success
success
success
[root@master ~]#
[root@master ~]#
```

On Worker node, allow following ports,

$ sudo firewall-cmd --permanent –add-port=10250/tcp

$ sudo firewall-cmd --permanent --add-port=30000-32767/tcp

$ sudo firewall-cmd --reload

```
[root@worker1 ~]# sudo firewall-cmd --permanent --add-port={179,10250,30000-32767}/tcp
sudo firewall-cmd --permanent --add-port=4789/udp
sudo firewall-cmd --reload
Warning: ALREADY_ENABLED: 10250:tcp
Warning: ALREADY_ENABLED: 30000-32767:tcp
success
success
success
[root@worker1 ~]#
[root@worker1 ~]#
```

```
[root@worker2 ~]#
[root@worker2 ~]#
[root@worker2 ~]# sudo firewall-cmd --permanent --add-port={179,10250,30000-32767}/tcp
sudo firewall-cmd --permanent --add-port=4789/udp
sudo firewall-cmd --reload
Warning: ALREADY_ENABLED: 10250:tcp
Warning: ALREADY_ENABLED: 30000-32767:tcp
success
success
success
[root@worker2 ~]#
[root@worker2 ~]#
[root@worker2 ~]# firewall-cmd --list-all
public
  target: default
  icmp-block-inversion: no
  interfaces:
  sources:
  services: dhcpv6-client mdns ssh
  ports: 10250/tcp 30000-32767/tcp 179/tcp 4789/udp
  protocols:
  forward: yes
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
[root@worker2 ~]#
```

Step 5) Install Containerd container runtime

Kubernetes requires a container runtime for pods to run. Kubernetes 1.23 and later versions require that you install a container runtime that confirms with the Container Runtime Interface.

In this guide, we will install Containerd which is a high-level container runtime. To do so, we need to enable two crucial kernel modules – overlay and br_netfilter modules.

To achieve this, we need to configure the prerequisites as follows:

First, create a modules configuration file for Kubernetes:

$ sudo tee /etc/modules-load.d/containerd.conf <<EOF

overlay

br_netfilter

EOF

Then load both modules using the modprobe command.

$ sudo modprobe overlay

$ sudo modprobe br_netfilter

Next, configure the required sysctl parameters as follows:

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-iptables = 1

net.ipv4.ip_forward = 1

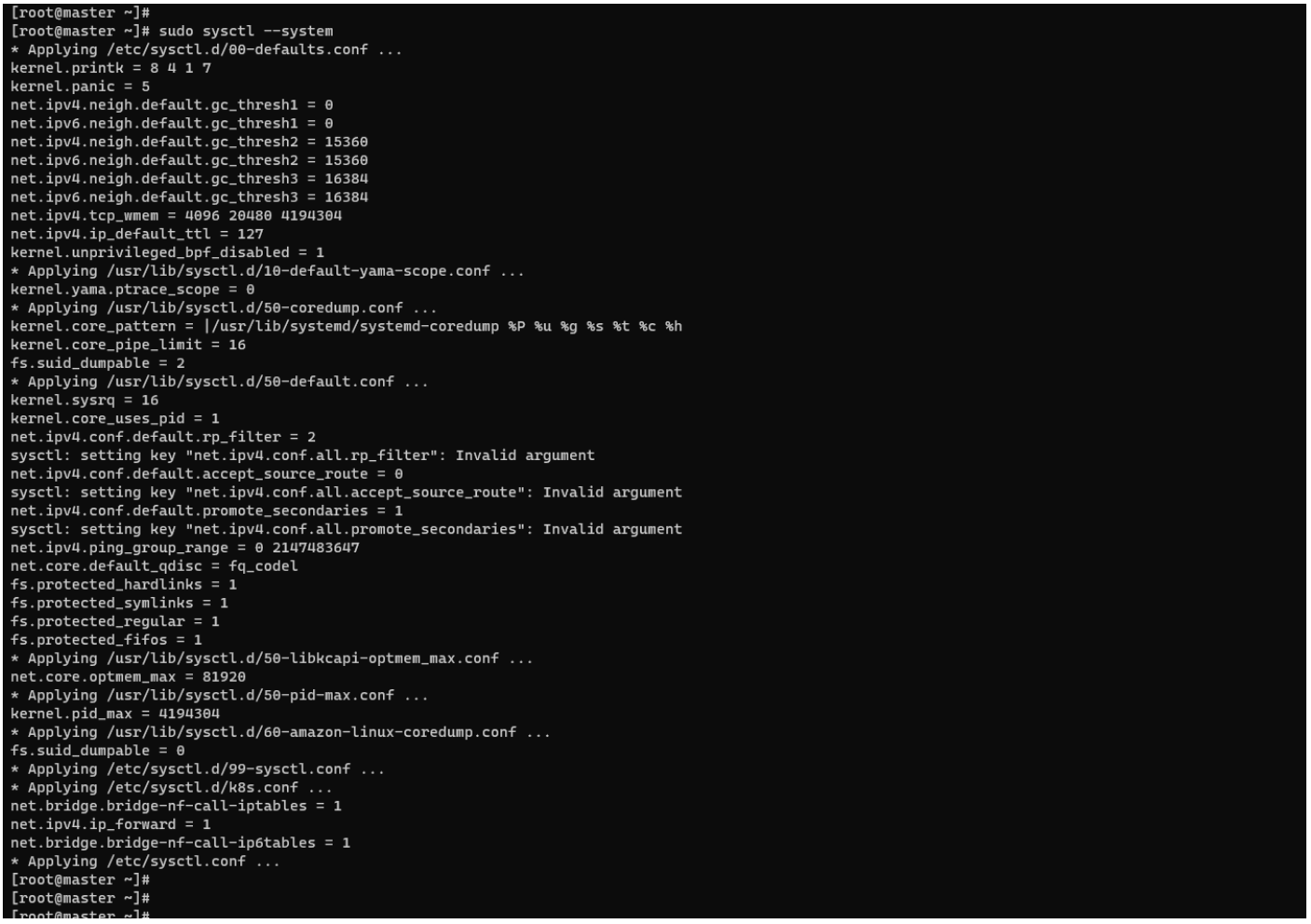
net.bridge.bridge-nf-call-ip6tables = 1

EOF

```
br_netfilter
[root@master ~]#
[root@master ~]#
[root@master ~]#
[root@master ~]# sudo modprobe overlay
sudo modprobe br_netfilter
[root@master ~]#
[root@master ~]# sudo modprobe overlay
[root@master ~]# sudo modprobe br_netfilter
[root@master ~]#
[root@master ~]#
[root@master ~]# cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
[root@master ~]#
[root@master ~]#
[root@master ~]#
[root@master ~]# sudo sysctl --system
* Applying /etc/sysctl.d/00-defaults.conf ...
kernel.printk = 8 4 1 7
kernel.panic = 5
net.ipv4.neigh.default.gc_thresh1 = 0
net.ipv6.neigh.default.gc_thresh1 = 0
net.ipv4.neigh.default.gc_thresh2 = 15360
net.ipv6.neigh.default.gc_thresh2 = 15360
net.ipv4.neigh.default.gc_thresh3 = 16384
net.ipv6.neigh.default.gc_thresh3 = 16384
net.ipv4.tcp_wmem = 4096 20480 4194304
net.ipv4.ip_default_ttl = 127
kernel.unprivileged_bpf_disabled = 1
* Applying /usr/lib/sysctl.d/10-default-yama-scope.conf ...
kernel.yama.ptrace_scope = 0
* Applying /usr/lib/sysctl.d/50-coredump.conf ...
kernel.core_pattern = |/usr/lib/systemd/systemd-coredump %P %u %g %s %t %c %h
kernel.core_pipe_limit = 16
fs.suid_dumpable = 2
* Applying /usr/lib/sysctl.d/50-default.conf ...
kernel.sysrq = 16
kernel.core_uses_pid = 1
net.ipv4.conf.default.rp_filter = 2
sysctl: setting key "net.ipv4.conf.all.rp_filter": Invalid argument
net.ipv4.conf.default.accept_source_route = 0
sysctl: setting key "net.ipv4.conf.all.accept_source_route": Invalid argument
net.ipv4.conf.default.promote_secondaries = 1
sysctl: setting key "net.ipv4.conf.all.promote_secondaries": Invalid argument
```

Save the changes and exit. To confirm the changes have been applied, run the command:

$ sudo sysctl –system



Install containerd runtime

$ sudo dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

$ sudo yum install containerd.io -y

$ sudo containerd config default | sudo tee /etc/containerd/config.toml >/dev/null 2>&1

$ sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' /etc/containerd/config.toml

Start and enable the containerd service and also check the status of the service.

```
[root@master ~]#
[root@master ~]#
[root@master ~]# sudo containerd config default | sudo tee /etc/containerd/config.toml >/dev/null 2>&1[root@master ~]#
[root@master ~]#
[root@master ~]#
[root@master ~]# sudo containerd config default | sudo tee /etc/containerd/config.toml >/dev/null 2>&1[root@master ~]#
[root@master ~]#
[root@master ~]# sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' /etc/containerd/config.toml
[root@master ~]#
[root@master ~]#
[root@master ~]#
```

```
[root@master ~]#
[root@master ~]#
[root@master ~]#
[root@master ~]# sudo sed -i '/swap/d' /etc/fstab
[root@master ~]#
[root@master ~]#
[root@master ~]#
[root@master ~]# sudo sed -i '/swap/d' /etc/fstab
[root@master ~]#
[root@master ~]#
[root@master ~]#
[root@master ~]# sudo dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
Adding repo from: https://download.docker.com/linux/centos/docker-ce.repo
[root@master ~]#
[root@master ~]#
[root@master ~]#
[root@master ~]# sudo yum install containerd -y
Docker CE Stable - x86_64                                       355  B/s | 397  B     00:01
Errors during downloading metadata for repository 'docker-ce-stable':
  - Status code: 404 for https://download.docker.com/linux/centos/2023.3.20240205/x86_64/stable/repodata/repomd.xml (IP: 108.159.46.105)
Error: Failed to download metadata for repo 'docker-ce-stable': Cannot download repomd.xml: Cannot download repodata/repomd.xml: All mirrors were tried
Ignoring repositories: docker-ce-stable
Last metadata expiration check: 0:39:27 ago on Sat Feb 10 07:39:47 2024.
Dependencies resolved.
================================================================================ Package             Architecture     Version
================================================================================Installing:
 containerd          x86_64          1.7.11-1.amzn2023.0.1             amazonlinux           35 M
Installing dependencies:
 runc                x86_64          1.1.11-1.amzn2023.0.1             amazonlinux          3.0 M

Transaction Summary
================================================================================Install  2 Packages

Total download size: 38 M
Installed size: 137 M
Downloading Packages:
(1/2): runc-1.1.11-1.amzn2023.0.1.x86_64.rpm                    27 MB/s | 3.0 MB     00:00
(2/2): containerd-1.7.11-1.amzn2023.0.1.x86_64.rpm              52 MB/s |  35 MB     00:00
```

$ sudo systemctl start containerd

$ sudo systemctl enable containerd

$ sudo systemctl status containerd

```
[root@master ~]#
[root@master ~]# sudo systemctl restart containerd
[root@master ~]#
[root@master ~]#
[root@master ~]# sudo systemctl enable containerd
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.service → /usr/lib/systemd/system/containerd.service.
[root@master ~]#
[root@master ~]#
[root@master ~]#
```

Step 6) Install Kubernetes Packages

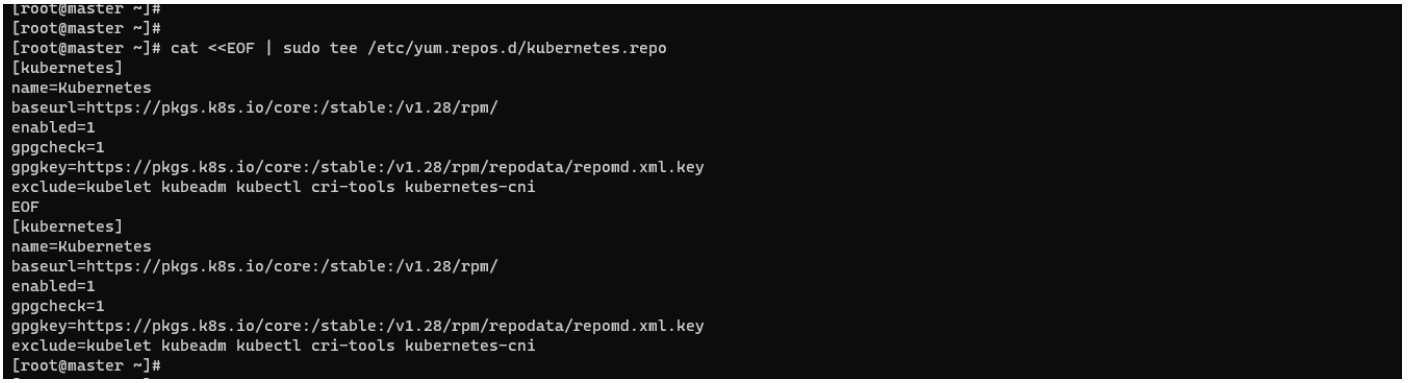
With everything required for Kubernetes to work installed, let us go ahead and install Kubernetes packages like kubelet, kubeadm and kubectl. Create a Kubernetes repository file.

$ sudo vi /etc/yum.repos.d/kubernetes.repo

And add the following lines.

[kubernetes]

name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kubelet kubeadm kubectl



Finally, install k8s package as follows.

$ sudo dnf install -y kubelet kubeadm kubectl disableexcludes=kubernetes

Once installed, be sure to enable and start Kubelet service.

$ sudo systemctl enable kubelet

$ sudo systemctl start kubelet

At this juncture, we are all set to install Kubernetes cluster.

```
[root@master ~]#
[root@master ~]#
[root@master ~]# sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
Docker CE Stable - x86_64                                       359  B/s | 397  B     00:01
Errors during downloading metadata for repository 'docker-ce-stable':
  - Status code: 404 for https://download.docker.com/linux/centos/2023.3.20240205/x86_64/stable/repodata/repomd.xml (IP: 108.159.46.83)
Error: Failed to download metadata for repo 'docker-ce-stable': Cannot download repomd.xml: Cannot download repodata/repomd.xml: All mirrors were tried
Kubernetes                                                       28 kB/s |  18 kB     00:00
Ignoring repositories: docker-ce-stable
Dependencies resolved.
================================================================================ Package                     Architecture  Version
================================================================================Installing:
 kubeadm                        x86_64       1.28.6-150500.1.1              kubernetes         9.7 M
 kubectl                        x86_64       1.28.6-150500.1.1              kubernetes          10 M
 kubelet                        x86_64       1.28.6-150500.1.1              kubernetes          19 M
Installing dependencies:
 conntrack-tools                x86_64       1.4.6-2.amzn2023.0.2           amazonlinux        208 k
 cri-tools                      x86_64       1.28.0-150500.1.1              kubernetes         8.1 M
 kubernetes-cni                 x86_64       1.2.0-150500.2.1               kubernetes         6.2 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
 socat                          x86_64       1.7.4.2-1.amzn2023.0.2         amazonlinux        303 k

Transaction Summary
================================================================================Install  10 Packages

Total download size: 53 M
Installed size: 292 M
Downloading Packages:
(1/10): socat-1.7.4.2-1.amzn2023.0.2.x86_64.rpm                 4.7 MB/s | 303 kB     00:00
(2/10): libnetfilter_cttimeout-1.0.0-19.amzn2023.0.2.x86_64.rpm 351 kB/s |  24 kB     00:00
(3/10): conntrack-tools-1.4.6-2.amzn2023.0.2.x86_64.rpm         2.8 MB/s | 208 kB     00:00
(4/10): libnetfilter_queue-1.0.5-2.amzn2023.0.2.x86_64.rpm      2.2 MB/s |  30 kB     00:00
(5/10): libnetfilter_cthelper-1.0.0-21.amzn2023.0.2.x86_64.rpm  1.6 MB/s |  24 kB     00:00
(6/10): kubectl-1.28.6-150500.1.1.x86_64.rpm                     19 MB/s |  10 MB     00:00
(7/10): cri-tools-1.28.0-150500.1.1.x86_64.rpm                   13 MB/s | 8.1 MB     00:00
(8/10): kubeadm-1.28.6-150500.1.1.x86_64.rpm                     14 MB/s | 9.7 MB     00:00
(9/10): kubelet-1.28.6-150500.1.1.x86_64.rpm                     30 MB/s |  19 MB     00:00
(10/10): kubernetes-cni-1.2.0-150500.2.1.x86_64.rpm             9.8 MB/s | 6.2 MB     00:00
--------------------------------------------------------------------------------Total                                                       3
Kubernetes                                                       5.2 kB/s | 1.7 kB     00:00
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:/v1.28/rpm/repodata/repomd.xml.key
Key imported successfully
Running transaction check
Transaction check succeeded.
```

```
Complete!
[root@master ~]# sudo systemctl enable --now kubelet
Created symlink /etc/systemd/system/multi-user.target.wants/kubelet.service → /usr/lib/systemd/system/kubelet.service.
[root@master ~]#
[root@master ~]#
[root@master ~]#
[root@master ~]#
[root@master ~]# sudo systemctl enable kubelet
[root@master ~]#
[root@master ~]#
[root@master ~]#
[root@master ~]#  sudo systemctl start kubelet
[root@master ~]#
[root@master ~]#
[root@master ~]#
[root@master ~]#
```

Step 7) Create a Kubernetes cluster

We are going to initialize a Kubernetes cluster using the kubeadm command as follows. This initializes a control plane in the master node.

```
$ sudo kubeadm init --control-plane-endpoint=master
```

Once the control plane is created, you will be required to carry out some additional commands to start using the cluster.

```
[root@master ~]#
[root@master ~]# sudo kubeadm init --control-plane-endpoint=master
I0210 08:27:05.496381   28641 version.go:256] remote version is much newer: v1.29.1; falling back to: stable-1.28
[init] Using Kubernetes version: v1.28.6
[preflight] Running pre-flight checks
       [WARNING Firewalld]: firewalld is active, please ensure ports [6443 10250] are open or your cluster may not function correctly
[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 in beforehand using 'kubeadm config images pull'
W0210 08:27:20.014625   28641 checks.go:835] 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 that using "regist
ry.k8s.io/pause:3.9" 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 [kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local master] and IPs [10.96.0.1 172.31.13.236]
[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 [localhost master] and IPs [172.31.13.236 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [localhost master] and IPs [172.31.13.236 127.0.0.1 ::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 "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". This can take up to 4m0s
[apiclient] All control plane components are healthy after 8.503217 seconds
[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 master 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 master as control-plane by adding the taints [node-role.kubernetes.io/control-plane:NoSchedule]
[bootstrap-token] Using token: j1xv31.6jb6v67dzwxctyf3
[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
```

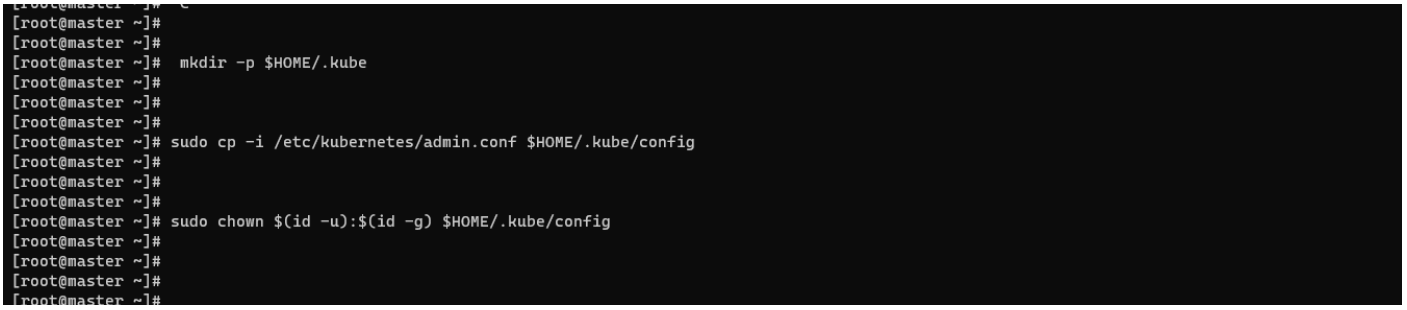
```
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to get nodes
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate 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!

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/

You can now join any number of control-plane nodes by copying certificate authorities
and service account keys on each node and then running the following as root:

  kubeadm join master:6443 --token j1xv31.6jb6v67dzwxctyf3 \
        --discovery-token-ca-cert-hash sha256:0521e8914103f4979b41f5ec7a56ea511d0b14ad452467af10dd2c8304f7fc8c \
        --control-plane

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

kubeadm join master:6443 --token j1xv31.6jb6v67dzwxctyf3 \
        --discovery-token-ca-cert-hash sha256:0521e8914103f4979b41f5ec7a56ea511d0b14ad452467af10dd2c8304f7fc8c
[root@master ~]#
[root@master ~]#
```

Therefore, run the commands, sequentially.

$ mkdir -p $HOME/.kube

$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

$ sudo chown $(id -u):$(id -g) $HOME/.kube/config



Step 8) Install Calico Pod Network Add-on

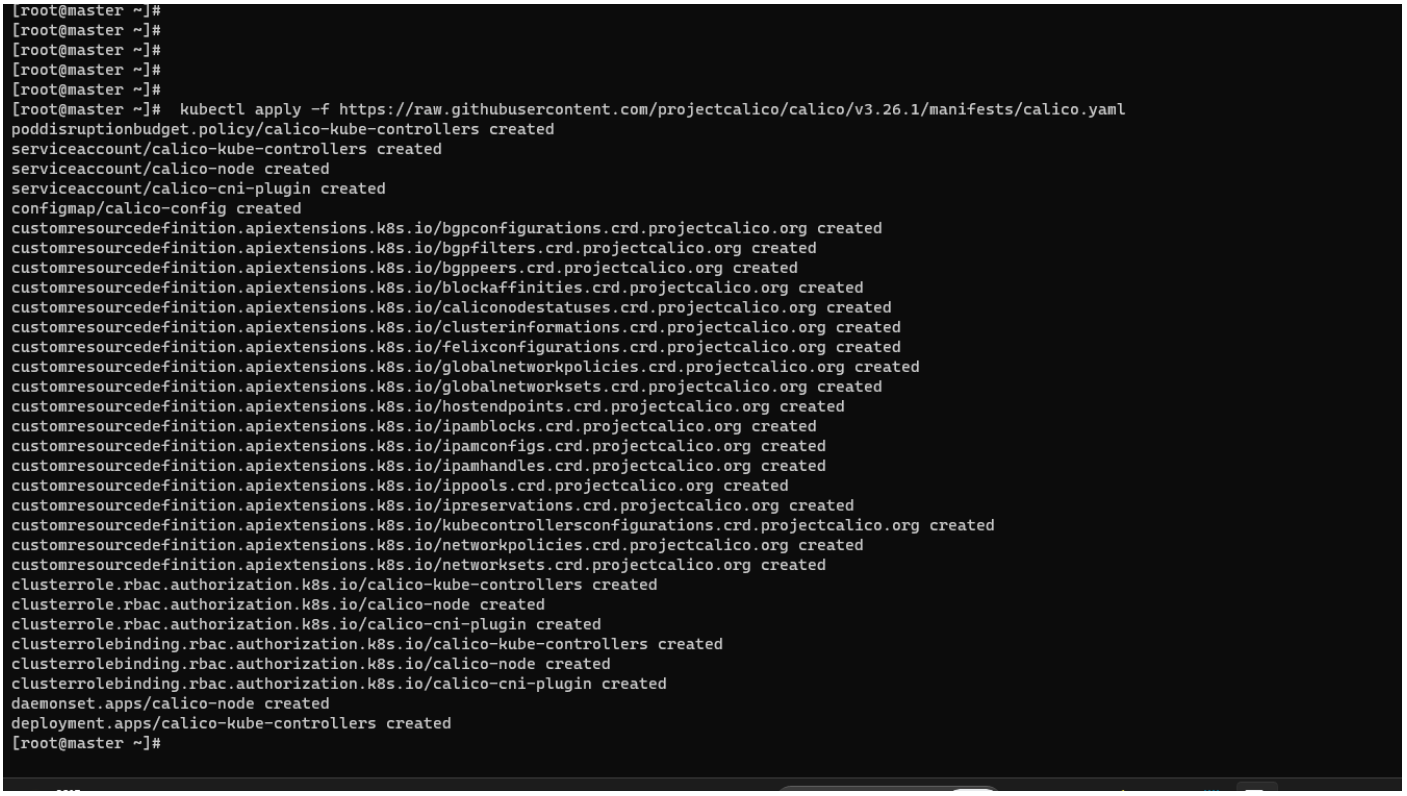
The next step is to install Calico CNI (Container Network Interface). It is an opensource project used to provide container networking and security. After Installing Calico CNI, nodes state will change to Ready state, DNS service inside the cluster would be functional and containers can start communicating with each other.

Calico provides scalability, high performance, and interoperability with existing Kubernetes workloads. It can be deployed on-premises and on popular cloud technologies such as AWS .

To install Calico CNI, run the following command from the master node

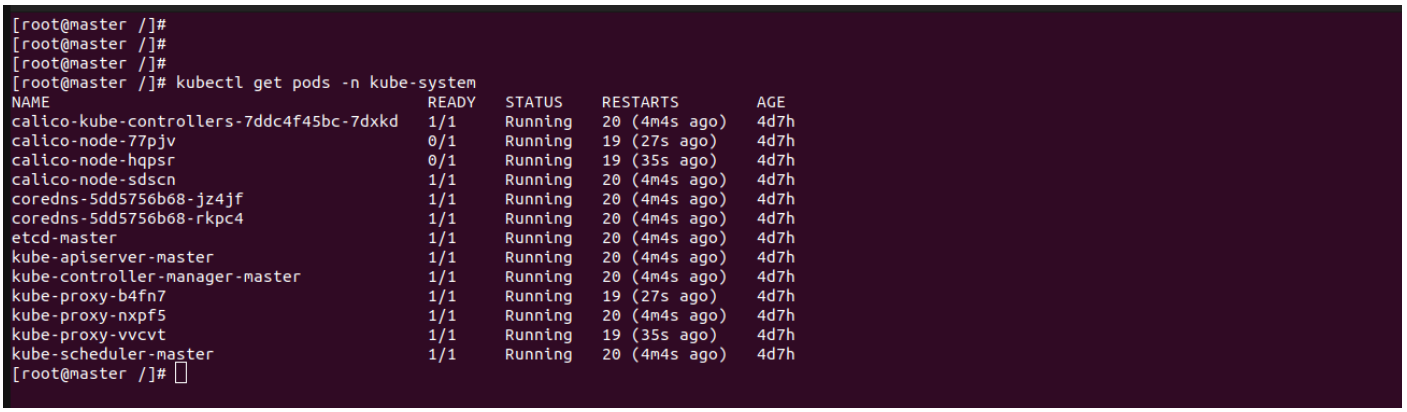
$ kubectl apply -f

https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/calico.yaml



To confirm if the pods have started, run the command:

$ kubectl get pods -n kube-system



Step 9) Adding worker node to the cluster

To add the worker node to the Kubernetes cluster, follow step 1 up until Step 6. Once you are done, run the command generated by the master node for joining a worker node to the cluster. In our case, this will be:

$ sudo kubeadm join master:6443 --token cqb8vy.iicmmqrb1m8u9cob --discovery-token-ca-cert-hash

sha256:79748a56f603e6cc57f67bf90b7db5aebe090107d540d6cc8a8f65b785de7543

If all goes well, you should get the notification that the node has joined the cluster. Repeat the same procedure for other nodes in case you have multiple worker nodes

On worker1:

```
[root@worker1 ~]#
[root@worker1 ~]#
[root@worker1 ~]# kubeadm join master:6443 --token j1xv31.6jb6v67dzwxctyf3 \
        --discovery-token-ca-cert-hash sha256:0521e8914103f4979b41f5ec7a56ea511d0b14ad452467af10dd2c8304f7fc8c
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

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.

[root@worker1 ~]#
```

On Worker2 :

```
[root@worker2 ~]#
[root@worker2 ~]#
[root@worker2 ~]# kubeadm join master:6443 --token j1xv31.6jb6v67dzwxctyf3 \
        --discovery-token-ca-cert-hash sha256:0521e8914103f4979b41f5ec7a56ea511d0b14ad452467af10dd2c8304f7fc8c
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

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.

[root@worker2 ~]#
```

Now, head back to the master node and, once again, verify the nodes in your cluster. This time around, the worker node will appear in the list on nodes in the cluster,

In addition, you can retrieve more information using the -o wide options.

$ kubectl get nodes -o wide

```
[root@master /]#
[root@master /]#
[root@master /]# kubectl get nodes -o wide
NAME      STATUS   ROLES           AGE    VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE           KERNEL-VERSION                 CONTAINE
R-RUNTIME
master    Ready    control-plane   4d7h   v1.28.6   172.31.13.236   <none>        Amazon Linux 2023   6.1.75-99.163.amzn2023.x86_64   containe
rd://1.7.11
worker1   Ready    <none>          4d7h   v1.28.6   172.31.13.128   <none>        Amazon Linux 2023   6.1.75-99.163.amzn2023.x86_64   containe
rd://1.7.11
worker2   Ready    <none>          4d7h   v1.28.6   172.31.0.223    <none>        Amazon Linux 2023   6.1.75-99.163.amzn2023.x86_64   containe
rd://1.7.11
[root@master /]#
[root@master /]#
```

The above output confirms that the master node is ready. Additionally, you can check the pod namespaces:

$ kubectl get pods –all-namespaces

```
[root@master /]#
[root@master /]# kubectl get pods --all-namespaces
NAMESPACE              NAME                                       READY   STATUS    RESTARTS       AGE
kube-system            calico-kube-controllers-7ddc4f45bc-7dxkd   1/1     Running   20 (24m ago)   4d7h
kube-system            calico-node-77pjv                          1/1     Running   19 (21m ago)   4d7h
kube-system            calico-node-hqpsr                          1/1     Running   19 (21m ago)   4d7h
kube-system            calico-node-sdscn                          1/1     Running   20 (24m ago)   4d7h
kube-system            coredns-5dd5756b68-jz4jf                   1/1     Running   20 (24m ago)   4d8h
kube-system            coredns-5dd5756b68-rkpc4                   1/1     Running   20 (24m ago)   4d8h
kube-system            etcd-master                                1/1     Running   20 (24m ago)   4d8h
kube-system            kube-apiserver-master                      1/1     Running   20 (24m ago)   4d8h
kube-system            kube-controller-manager-master             1/1     Running   20 (24m ago)   4d8h
kube-system            kube-proxy-b4fn7                           1/1     Running   19 (21m ago)   4d7h
kube-system            kube-proxy-nxpf5                           1/1     Running   20 (24m ago)   4d8h
kube-system            kube-proxy-vvcvt                           1/1     Running   19 (21m ago)   4d7h
kube-system            kube-scheduler-master                      1/1     Running   20 (24m ago)   4d8h
```

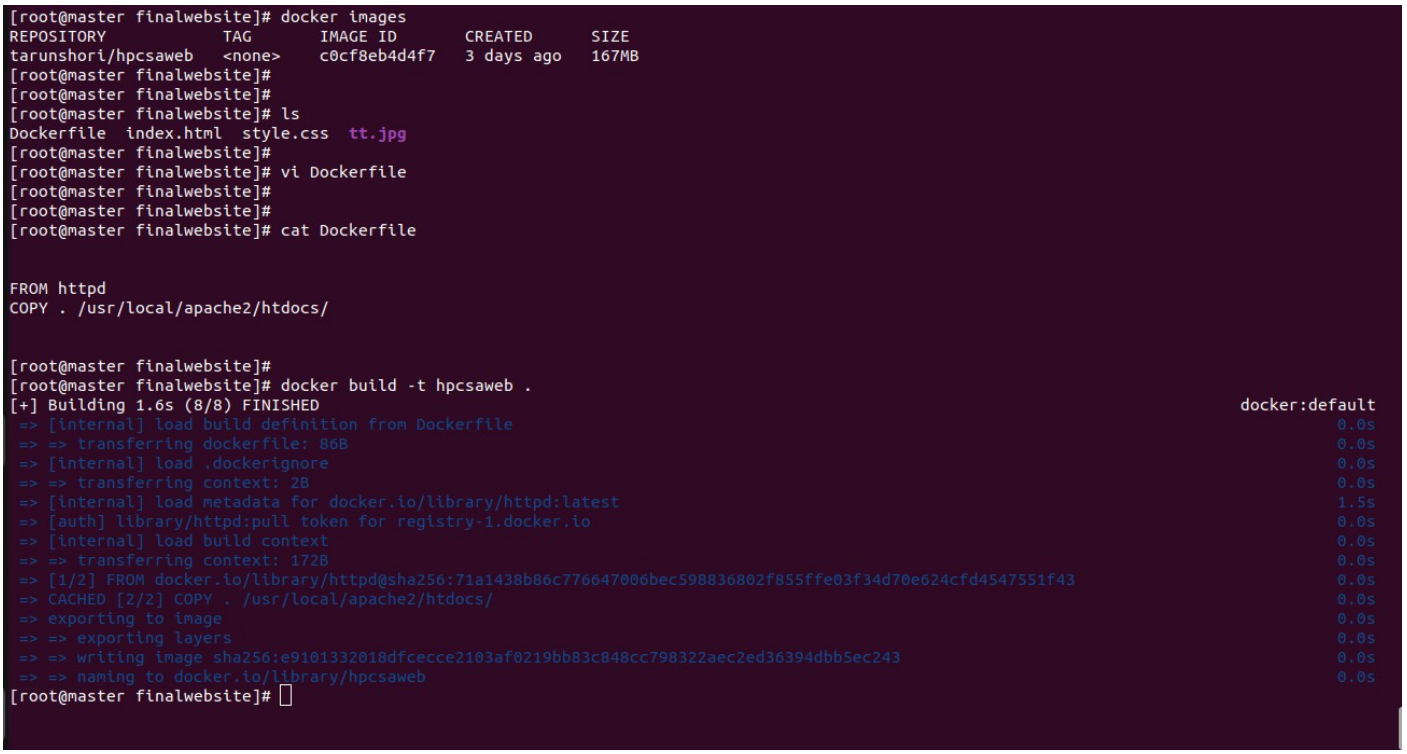
$ kubectl get nodes

```
[root@master /]# kubectl get nodes
NAME      STATUS   ROLES           AGE    VERSION
master    Ready    control-plane   4d7h   v1.28.6
worker1   Ready    <none>          4d7h   v1.28.6
worker2   Ready    <none>          4d7h   v1.28.6
[root@master /]#
```

Create Docker Image And Push To Docker Hub

1.Create a Dockerfile: Create a Dockerfile in our project directory with the necessary instructions to build your Docker image.

2.Build the Docker image: navigate to the directory containing your Dockerfile and execute the following command to build the Docker image.

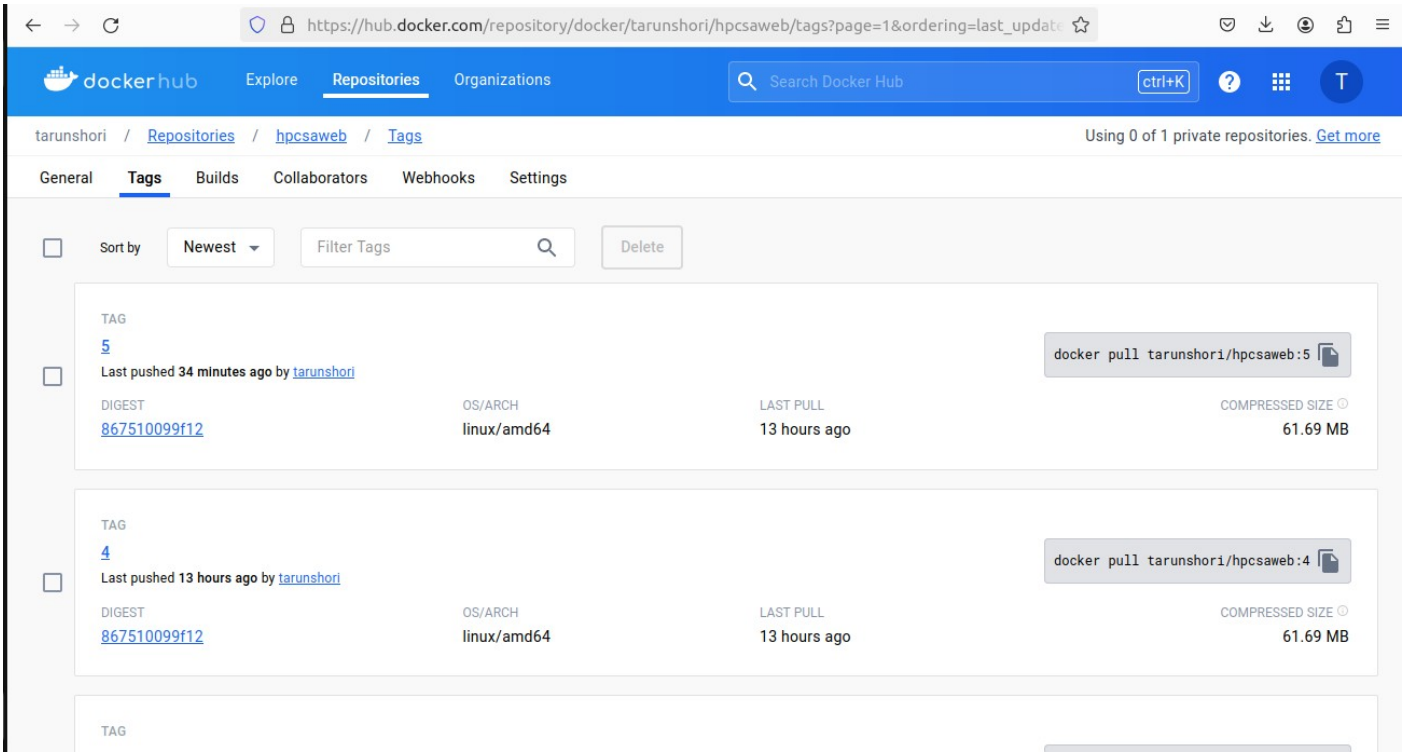


3.Tag the Docker image: After building the Docker image, need to tag it with the appropriate version.

4.Push the Docker image to Docker Hub: Log in to Docker Hub account using the docker login command, then push the tagged image to Docker Hub using the docker push command.

```
[root@master finalwebsite]# docker images
REPOSITORY            TAG       IMAGE ID       CREATED        SIZE
hpcsaweb              latest    e9101332018d   12 hours ago   168MB
tarunshori/hpcsaweb   <none>    c0cf8eb4d4f7   3 days ago     167MB
[root@master finalwebsite]#
[root@master finalwebsite]# docker tag hpcsaweb tarunshori/hpcsaweb:5
[root@master finalwebsite]#
[root@master finalwebsite]# docker images
REPOSITORY            TAG       IMAGE ID       CREATED        SIZE
hpcsaweb              latest    e9101332018d   12 hours ago   168MB
tarunshori/hpcsaweb   5         e9101332018d   12 hours ago   168MB
tarunshori/hpcsaweb   <none>    c0cf8eb4d4f7   3 days ago     167MB
[root@master finalwebsite]#
[root@master finalwebsite]# docker push tarunshori/hpcsaweb:5
The push refers to repository [docker.io/tarunshori/hpcsaweb]
188e1442e17f: Layer already exists
83e84e8d7bd1: Layer already exists
ee9a39d3b67e: Layer already exists
70d67450158d: Layer already exists
5f70bf18a086: Layer already exists
8f562cbc866f: Layer already exists
ceb365432eec: Layer already exists
5: digest: sha256:867510099f12c48c9932c451671d5daa372b8eb7767c0d69b564056d43045dd9 size: 1782
[root@master finalwebsite]#
```

5.Verify on Docker Hub: After pushing the image, verify that it's available on Docker Hub by visiting repository's page on the Docker Hub website.



Create PersistentVolume And PersistentVolumeClaim

1.create PersistentVolume:

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: web-pv
spec:
  capacity:
    storage: 4Gi
  accessModes:
    - ReadWriteMany
  local:
    path: /data/
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - worker1
          - worker2
  storageClassName: standard
  volumeMode: Filesystem
```

This YAML manifest defines a PersistentVolume (PV) named `web-pv` within the `project-website` namespace. Here's a breakdown of its components:

1. apiVersion: v1
   - Specifies the Kubernetes API version being used for this resource.

2. kind: PersistentVolume
   - Indicates that this resource is a PersistentVolume.

3. metadata:
   - name: web-pv
     - Provides a name for the PersistentVolume.
   - namespace: project-website
     - Specifies the namespace in which the PersistentVolume resides.

4. spec:
   - capacity:
     - storage: 4Gi
     - Specifies the storage capacity of the PersistentVolume, which is 4 gigabytes.

5. accessModes:

   - ReadWriteMany
     - Specifies that the volume can be mounted as read-write by multiple nodes simultaneously.

- local:
  - path: /data/
    - Specifies the path on the local node where the volume is located.

6. Node Affinity:
   -Specifies that the PV is bound to nodes with the hostname "worker1" or "worker2".

7. volumeMode: Filesystem
   - Indicates that the PersistentVolume will be formatted with a filesystem.
8. storageClassName: standard
   - Specifies the StorageClass to use for provisioning the PersistentVolume.

This PersistentVolume manifest defines a volume named `web-pv` with a capacity of 4 gigabytes, using the standard StorageClass. It allows multiple nodes to mount the volume as read-write. The volume is hosted locally at the directory `/data/` on the node.

2.create PersistentVolumeClaim :

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: web-pvc
  namespace: project-website
spec:
  volumeName: web-pv
  storageClassName: standard
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 2Gi
~
~
```

This YAML manifest defines a PersistentVolumeClaim (PVC) named `web-pvc` within the `project-website` namespace. Here's a breakdown of its components:

1. apiVersion: v1
   - Specifies the Kubernetes API version being used for this resource.

2. kind: PersistentVolumeClaim
   - Indicates that this resource is a PersistentVolumeClaim.

3. metadata:
   - name: web-pvc
     - Provides a name for the PersistentVolumeClaim.
   - namespace: project-website
     - Specifies the namespace in which the PersistentVolumeClaim resides.

4. spec:
   - volumeName: web-pv
     - Specifies the name of the PersistentVolume to claim (`web-pv` in this case).
   - storageClassName: standard
     - Specifies the StorageClass to use for provisioning the PersistentVolumeClaim.
   - accessModes:
     - - ReadWriteMany
       - Specifies that the volume can be mounted as read-write by multiple nodes simultaneously.
   - resources:
     - requests:
       - storage: 2Gi
         - Specifies the amount of storage requested for the PersistentVolumeClaim, which is 2 gigabytes.

This PersistentVolumeClaim manifest requests a volume named `web-pv` with a capacity of 2 gigabytes, using the standard StorageClass. It allows multiple nodes to mount the volume as read-write.

Create Deployment and Service

run an application by creating a Kubernetes Deployment,Services,Horizontal Pod Autoscaler and other object. we describe all configurations in a YAML file.

1. Create a **Deployment** based on the YAML file:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-web
  namespace: project-website
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-web
  template:
    metadata:
      labels:
        app: my-web
    spec:
      containers:
      - name: web-container
        image: docker.io/tarunshori/hpcsaweb
        ports:
        - containerPort: 80
        volumeMounts:
        - name: data-volume
          mountPath: /data/
      volumes:
      - name: data-volume
        persistentVolumeClaim:
          claimName: web-pvc  # Reference to the PVC
```

```
        livenessProbe:                # Added livenessProbe
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 15
          periodSeconds: 10
        readinessProbe:               # Added readinessProbe
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 10
```

1. Name: `my-web`
   - Identifies the deployment within the Kubernetes cluster.

2. Namespace: `project-website`
   - Specifies the namespace where the deployment resides.

3. Replicas: `3`
   - Indicates the desired number of replica pods for the deployment.

4. Selector:
   - `matchLabels: app=my-web`
     - Specifies the labels used to match the pods controlled by this deployment.

5. Template:
   - Defines the pod template for the deployment.

6. Containers:
   - `web-container`
     - Name of the container within the pod.
   - `docker.io/tarunshori/hpcsaweb`
     - Docker image used to create the container.
   - Ports:
     - `containerPort: 80`
       - Exposes port 80 within the container.

7. Liveness Probe:

- livenessProbe:
  - This section indicates the beginning of the livenessProbe
- configuration.httpGet: Specifies that an HTTP GET :
  -request will be used for probing the container.
- Path: /
  - Defines the path to which the HTTP GET request will be sent. In this case, it's the root path /.
-port: 80:
  -Specifies the port on which the HTTP GET request will be sent. Port 80 is commonly used for HTTP traffic.

- InitialDelaySeconds: 15
  - Specifies the number of seconds to wait after the container starts before the first liveness probe is performed. This delay allows the container to initialize.

- periodSeconds: 10
  - Specifies the interval between successive liveness probe executions. In this case, a new probe will be executed every 10 seconds.

8. Readiness Probe:

- readinessProbe :
  - This section indicates the beginning of the readiness Probe configuration.
- httpGet:
  - Specifies that an HTTP GET request will be used for probing the container's readiness.
- path: /:
  -Defines the path to which the HTTP GET request will be sent. Here, it's the root path /.
- port: 80:
  - Specifies the port on which the HTTP GET request will be sent. Again, port 80 is used for HTTP traffic.

- initialDelaySeconds: 5
  -Specifies the number of seconds to wait after the container starts before the first readiness probe is performed. This allows the container some time to become ready to serve traffic.
- periodSeconds: 10:
  -Specifies the interval between successive readiness probe executions. In this case, a new probe will be executed every 10 seconds.

9. Volumes:
   - `data-volume`
     - Name of the volume.
     - Persistent Volume Claim: `web-pvc`
       - Reference to the PersistentVolumeClaim used by the volume.
     - Mount Path: `/data/`
       - Specifies the mount path within the container.

2. Create a **Update Strategy** based on the YAML file:

```yaml
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 1
    maxUnavailable: 1

# You can specify the number of old ReplicaSets to retain for rollback
revisionHistoryLimit: 2
```

1. Update Strategy:
   - `type: RollingUpdate`
     - Defines the update strategy for the deployment.
   - `rollingUpdate:`
     - Specifies the parameters for rolling updates.
   - `maxSurge: 1`
       - Maximum number of additional pods that can be created during an update.
   - `maxUnavailable: 1`
       - Maximum number of pods that can be unavailable during an update.

2. Revision History Limit:`2`
   - Specifies the number of old ReplicaSets to retain for rollback purposes.

3. Create a **Horizontal Pod Autoscaler** based on the YAML file:

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: web-autoscaler
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-web
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 70
```

1. Name: `web-autoscaler`
- Identifies the HorizontalPodAutoscaler within the Kubernetes cluster.

2. Scale Target Reference:
   - `apiVersion: apps/v1`
   - `kind: Deployment`
   - `name: my-web`
     - Specifies the deployment to scale based on CPU utilization.

3. Minimum Replicas: `3`
   - Specifies the minimum number of pods to maintain.

4. Maximum Replicas: `10`
   - Specifies the maximum number of pods to scale up to.

5. Target CPU Utilization:`70%`
   - Sets the target CPU utilization percentage to trigger scaling.

4. Create a **Service** based on the YAML file:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: web-service
  namespace: project-website
spec:
  type: NodePort
  selector:
    app: my-web
  ports:
    - protocol: TCP
      port: 80
```

1. Name:`web-service`
   - Identifies the service within the Kubernetes cluster.

2. Namespace: `project-website`
   - Specifies the namespace where the service resides.

3. Type: `NodePort`
  - Exposes the service on a static port on each node's IP.

4. Selector:
   - `app: my-web`
     - Routes traffic to pods with the label `app=my-web`.

5. Ports:
   - `protocol: TCP`
     - Specifies the protocol used for the port.
   - `port: 80`
     - Exposes port 80 within the cluster.

5. Create **Pod anti-affinity** on the Deployment YAML file:

```yaml
spec:
  affinity:                          # Added pod anti-affinity
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: app
            operator: In
            values:
            - my-web
        topologyKey: "kubernetes.io/hostname"
```

1. Label Selector: Ensure that the label selector (`app: my-web`) matches the labels applied to your pods correctly. If the label is not applied or does not match, the anti-affinity rule won't have any effect.

2. Topology Key: The topologyKey specifies the key of the node label to use for anti-affinity. In this case, it's set to "kubernetes.io/hostname", which means Kubernetes will consider the hostname of nodes when applying the anti-affinity rule. Make sure that your nodes have unique hostnames for this rule to work effectively.

3. Cluster Size: In a small cluster with only a few nodes, applying pod anti-affinity may limit the scheduling options for Kubernetes, potentially affecting the distribution of pods across nodes. Consider the impact on resource utilization and scheduling efficiency before applying anti-affinity rules in such environments.

Apply YAML Manifest For The "My-Web"

1. Apply YAML Manifest :

```
$ kubectl apply -f web-pv.yaml

$ kubectl apply -f web-pvc.yaml

$ kubectl get pv -n=project-website

$ kubectl get pvc -n=project-website
```

```
[root@master new-yaml-files]#
[root@master new-yaml-files]# kubectl apply -f web-pv.yaml
persistentvolume/web-pv created
[root@master new-yaml-files]#
[root@master new-yaml-files]#
[root@master new-yaml-files]#
[root@master new-yaml-files]#
[root@master new-yaml-files]# kubectl apply -f web-pvc.yaml
persistentvolumeclaim/web-pvc created
[root@master new-yaml-files]#
[root@master new-yaml-files]#
[root@master new-yaml-files]#
[root@master new-yaml-files]#
[root@master new-yaml-files]#
[root@master new-yaml-files]#
[root@master new-yaml-files]# kubectl get pv -n=project-website
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                     STORAGECLASS   REASON   AGE
host-pv   4Gi        RWX            Retain           Bound    project-website/host-pvc   standard               20h
web-pv    4Gi        RWX            Retain           Bound    project-website/web-pvc    standard               34s
[root@master new-yaml-files]#
[root@master new-yaml-files]#
[root@master new-yaml-files]# kubectl get pvc -n=project-website
NAME       STATUS   VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
host-pvc   Bound    host-pv   4Gi        RWX            standard       20h
web-pvc    Bound    web-pv    4Gi        RWX            standard       31s
[root@master new-yaml-files]#
[root@master new-yaml-files]#
[root@master new-yaml-files]#
[root@master new-yaml-files]#
```

$ kubectl apply -f deploy-service-all.yaml

```
[root@master new-yaml-files]# ls
deploy-service-all.yaml  web-pv.yaml  web-pvc.yaml
[root@master new-yaml-files]#
[root@master new-yaml-files]# kubectl apply -f deploy-service-all.yaml
deployment.apps/my-web created
horizontalpodautoscaler.autoscaling/web-autoscaler unchanged
service/web-service created
```

2. Display information about the Deployment:

```
[root@master new-yaml-files]#
[root@master new-yaml-files]# kubectl get deploy -n=project-website
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
my-web   0/3     3            0           12s
[root@master new-yaml-files]#
```

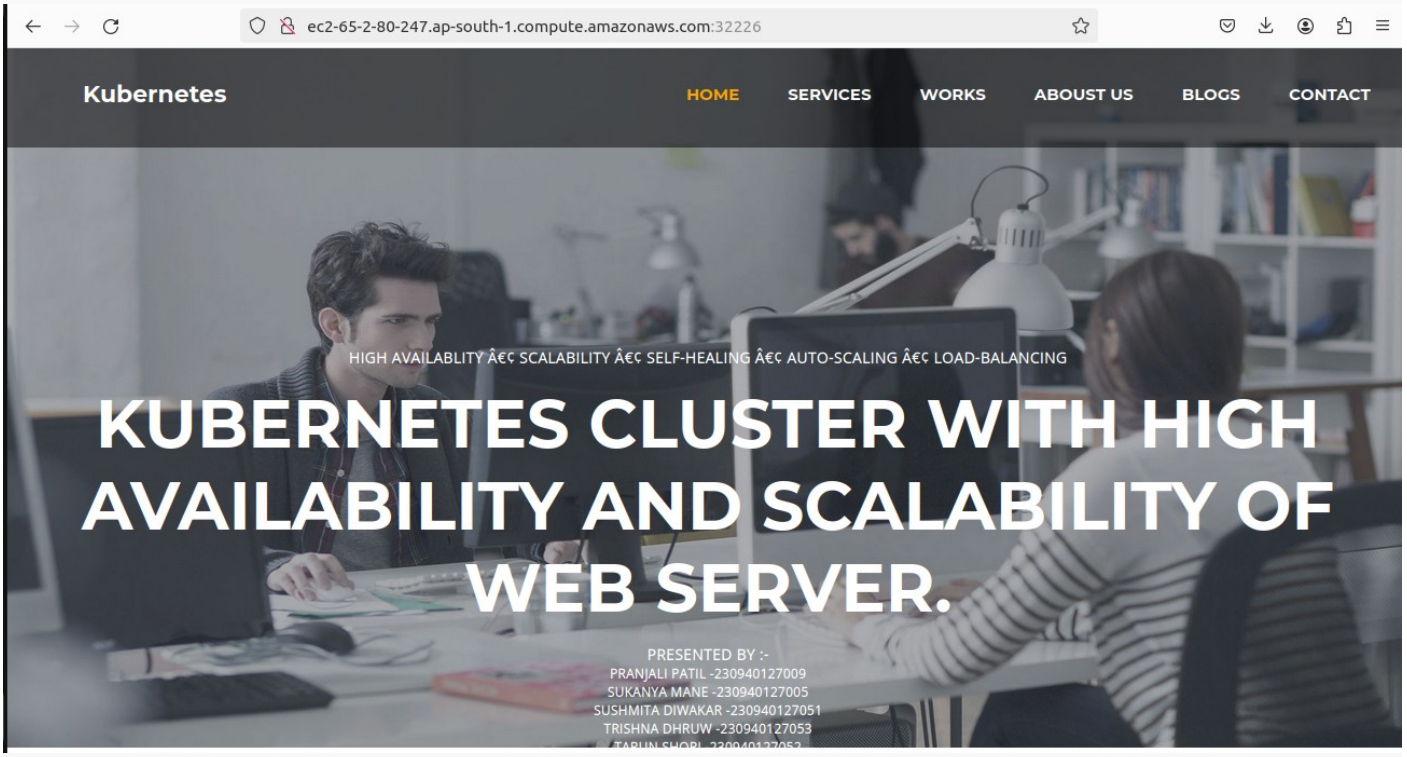
3. Display information about the Service:

```
[root@master new-yaml-files]# kubectl get svc -n=project-website
NAME          TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
web-service   NodePort   10.96.2.98    <none>        80:32226/TCP   43s
[root@master new-yaml-files]#
```

4. List the Pods created by the deployment :

```
[root@master new-yaml-files]# kubectl get pods -o wide -n=project-website
NAME                  READY   STATUS    RESTARTS   AGE   IP               NODE      NOMINATED NODE   READINESS GATES
my-web-7f67bfb967-47rgc   1/1     Running   0          19s   192.168.235.138   worker1   <none>           <none>
my-web-7f67bfb967-4jc7l   1/1     Running   0          16s   192.168.189.108   worker2   <none>           <none>
my-web-7f67bfb967-bqnb4   1/1     Running   0          19s   192.168.189.106   worker2   <none>           <none>
[root@master new-yaml-files]#
```

5. Access web-server from outside of the cluster using Node Port :



Kubernetes Dashboard Setup

The dashboard is a web-based Kubernetes user interface. we can use the dashboard to deploy containerized applications to a Kubernetes cluster, troubleshoot your containerized application, and manage the cluster resources. You can use the dashboard to get an overview of the applications running on your cluster, as well as for creating or modifying individual Kubernetes resources (such as Deployments, Jobs, DaemonSets, etc). For example, you can scale a Deployment, initiate a rolling update, restart a POD, or deploy new applications using a deploy wizard.

1. The Dashboard UI is not deployed by default. To deploy it, run the following command:

Create a new Directory for Dashboard and create a yaml file for ServiceAccount, ClusterRoleBinding,Role, Deployment and Service.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: kubernetes-dashboard
  name: dashboard-secret-updater
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "update", "patch"]

---

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dashboard-secret-updater-binding
  namespace: kubernetes-dashboard
subjects:
- kind: ServiceAccount
  name: default
  namespace: kube-system
roleRef:
  kind: Role
  name: dashboard-secret-updater
  apiGroup: rbac.authorization.k8s.io

---
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubernetes-dashboard
  namespace: kube-system
  labels:
    k8s-app: kubernetes-dashboard
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: kubernetes-dashboard
  template:
    metadata:
      labels:
        k8s-app: kubernetes-dashboard
    spec:
      containers:
      - name: kubernetes-dashboard
        image: kubernetesui/dashboard:v2.4.0
        ports:
        - containerPort: 8443
          protocol: TCP
        args:
          - --auto-generate-certificates
          - --namespace=kubernetes-dashboard
        volumeMounts:
        - name: my-service-account-token
          mountPath: /cert
          readOnly: true
      volumes:
      - name: my-service-account-token
        secret:
          secretName: my-service-account-token

---
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  type: NodePort
  ports:
    - port: 443
      targetPort: 8443
      nodePort: 32000   # Choose a NodePort number (range: 30000-32767)
  selector:
    k8s-app: kubernetes-dashboard
```

This YAML file describes a set of Kubernetes resources for deploying the Kubernetes Dashboard application with RBAC (Role-Based Access Control) configuration. Let's break down each section:

1. Role:

- apiVersion: rbac.authorization.k8s.io/v1

- kind: Role

- metadata: Specifies metadata for the Role.

- namespace: The namespace where the Role is defined (`kubernetes-dashboard`).

- name: The name of the Role (`dashboard-secret-updater`).

- rules: Defines permissions/rules for the Role.

- Grants permission to access, update, and patch secrets within the same namespace (`kubernetes-dashboard`).

2. RoleBinding:

- apiVersion: rbac.authorization.k8s.io/v1

- kind: RoleBinding

- metadata: Specifies metadata for the RoleBinding.

- name: The name of the RoleBinding (`dashboard-secret-updater-binding`).

- namespace: The namespace where the RoleBinding is defined (`kubernetes-dashboard`).

- subjects: Defines the subjects (users, groups, or service accounts) that the RoleBinding applies to.

- In this case, it applies to the `default` service account in the `kube-system` namespace.

- roleRef: Specifies the Role reference.

- kind: Specifies the kind of resource being referenced (`Role`).

- name: Specifies the name of the Role (`dashboard-secret-updater`).

- apiGroup: Specifies the API group (`rbac.authorization.k8s.io`).

3. Deployment:

- apiVersion: apps/v1

- kind: Deployment

- metadata: Specifies metadata for the Deployment.

- name: The name of the Deployment (`kubernetes-dashboard`).

- namespace: The namespace where the Deployment is defined (`kube-system`).

- labels: Labels to identify the Deployment (`k8s-app: kubernetes-dashboard`).

- spec: Specifies the desired state for the Deployment.

- replicas: Number of desired replicas (1).

- selector: Specifies how the Deployment finds which Pods to manage.

- template: Defines the Pod template for the Deployment.

  - metadata: Metadata for the Pod template.

  - spec: Specification for the Pod template.

    - containers: List of containers to run in the Pod.

      - Specifies the container named `kubernetes-dashboard`.

        - image: Docker image to use (`kubernetesui/dashboard:v2.4.0`).

          - ports: Specifies the port mapping for the container (`containerPort: 8443`).

        - args: Arguments to the container.

        - volumeMounts: Mounts a volume into the container.

        - volumes: Defines a volume for the Pod (`my-service-account-token`).

4. Service:

- apiVersion: v1

- kind: Service

- metadata: Specifies metadata for the Service.

  - name: The name of the Service (`kubernetes-dashboard`).

  - namespace: The namespace where the Service is defined (`kube-system`).

- spec: Specifies the desired state for the Service.

  - type: The type of Service (`NodePort`).

  - ports: Specifies the port mapping for the Service.

    - Exposes port 443 on the Service and targets port 8443 on Pods.

    - Specifies a NodePort number (32000) for accessing the Service externally.

- selector: Selects the Pods to which the Service applies based on labels.

This YAML file defines RBAC rules, a Deployment for running the Kubernetes Dashboard application, and a Service for exposing it externally.

2. Then apply yaml file :

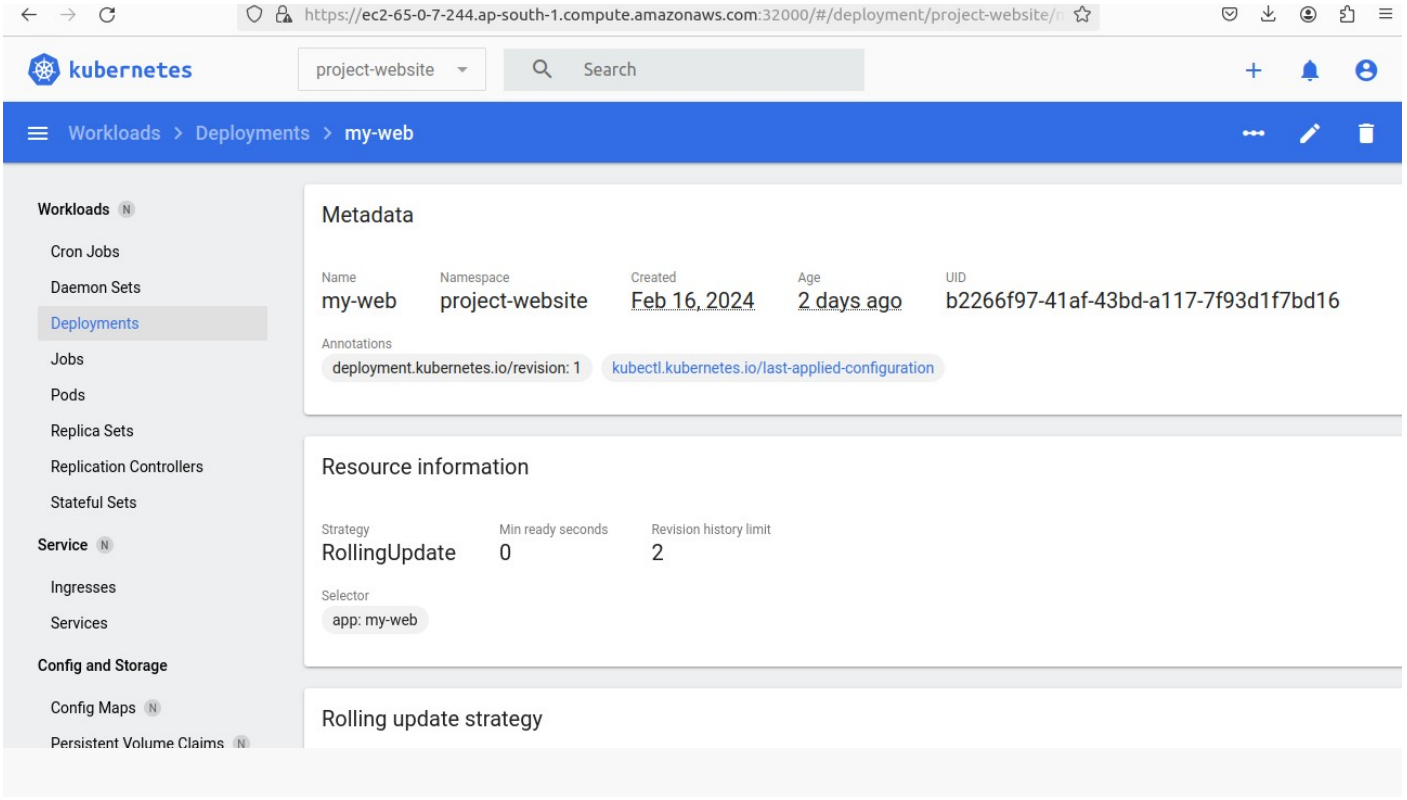
$ kubectl apply -f sa-dashboard.yaml

3. Get the token for authentication :

$ kubectl get secrets -n kube-system

$ kubectl describe secret my-service-account-token -n kube-system

4. Copy the token and paste in the kubernetes dashboard url with Nodeport.



Validation

1. Verify **High Availability** of Pods if worker node is Failed : ReplicaSets or Deployments to ensure that

multiple replicas of your application pods are running across multiple nodes in the cluster. ReplicaSets

manage the lifecycle of replicated pods and ensure that a specified number of pod replicas are running at all times.

```
[root@master /]# kubectl get nodes
NAME      STATUS   ROLES           AGE     VERSION
master    Ready    control-plane   4d13h   v1.28.6
worker1   Ready    <none>          4d13h   v1.28.6
worker2   Ready    <none>          4d13h   v1.28.6
[root@master /]#
[root@master /]# kubectl get pods -o wide -n=project-website
NAME                     READY   STATUS    RESTARTS   AGE    IP               NODE      NOMINATED NODE   READINESS GATES
my-web-7f67bfb967-2djqw   1/1     Running   0          91s    192.168.235.144   worker1   <none>           <none>
my-web-7f67bfb967-47rgc   1/1     Running   0          68m    192.168.235.138   worker1   <none>           <none>
my-web-7f67bfb967-vlpdr   1/1     Running   0          91s    192.168.235.137   worker1   <none>           <none>
[root@master /]#
[root@master /]# ./drain_node.sh worker1
Draining node worker1...
Flag --delete-local-data has been deprecated, This option is deprecated and will be deleted. Use --delete-emptydir-data.
node/worker1 cordoned
Warning: ignoring DaemonSet-managed Pods: kube-system/calico-node-hqpsr, kube-system/kube-proxy-vvcvt
evicting pod project-website/my-web-7f67bfb967-vlpdr
evicting pod kubernetes-dashboard/dashboard-metrics-scraper-84d944dc8f-fq66q
evicting pod project-website/my-web-7f67bfb967-2djqw
evicting pod project-website/my-web-7f67bfb967-47rgc
evicting pod kubernetes-dashboard/kubernetes-dashboard-585d8f955-rx9rj
pod/dashboard-metrics-scraper-84d944dc8f-fq66q evicted
pod/kubernetes-dashboard-585d8f955-rx9rj evicted
pod/my-web-7f67bfb967-vlpdr evicted
pod/my-web-7f67bfb967-47rgc evicted
pod/my-web-7f67bfb967-2djqw evicted
node/worker1 drained
Node worker1 drained successfully.
[root@master /]#
[root@master /]# kubectl get nodes
NAME      STATUS                     ROLES           AGE     VERSION
master    Ready                      control-plane   4d13h   v1.28.6
worker1   Ready,SchedulingDisabled   <none>          4d13h   v1.28.6
worker2   Ready                      <none>          4d13h   v1.28.6
[root@master /]#
```

```
[root@master /]# kubectl get pods -o wide -n=project-website
NAME                     READY   STATUS    RESTARTS   AGE    IP               NODE      NOMINATED NODE   READINESS GATES
my-web-7f67bfb967-5rhrh   1/1     Running   0          104s   192.168.189.116   worker2   <none>           <none>
my-web-7f67bfb967-9w95t   1/1     Running   0          104s   192.168.189.107   worker2   <none>           <none>
my-web-7f67bfb967-kmdgb   1/1     Running   0          104s   192.168.189.112   worker2   <none>           <none>
[root@master /]#
[root@master /]# kubectl get nodes
NAME      STATUS                     ROLES           AGE     VERSION
master    Ready                      control-plane   4d13h   v1.28.6
worker1   Ready,SchedulingDisabled   <none>          4d13h   v1.28.6
worker2   Ready                      <none>          4d13h   v1.28.6
[root@master /]# ./uncordon_node.sh worker1
Uncordoning node worker1...
node/worker1 uncordoned
Node worker1 uncordoned successfully.
[root@master /]#
[root@master /]# kubectl get nodes
NAME      STATUS   ROLES           AGE     VERSION
master    Ready    control-plane   4d13h   v1.28.6
worker1   Ready    <none>          4d13h   v1.28.6
worker2   Ready    <none>          4d13h   v1.28.6
[root@master /]# kubectl get pods -o wide -n=project-website
NAME                     READY   STATUS    RESTARTS   AGE     IP               NODE      NOMINATED NODE   READINESS GATES
my-web-7f67bfb967-5rhrh   1/1     Running   0          2m33s   192.168.189.116   worker2   <none>           <none>
my-web-7f67bfb967-9w95t   1/1     Running   0          2m33s   192.168.189.107   worker2   <none>           <none>
my-web-7f67bfb967-kmdgb   1/1     Running   0          2m33s   192.168.189.112   worker2   <none>           <none>
[root@master /]# kubectl get deploy -o wide -n=project-website
NAME     READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS      IMAGES                              SELECTOR
my-web   3/3     3            3           74m   web-container   docker.io/tarunshori/hpcsaweb:3   app=my-web
[root@master /]#
```

2. Verify **Scalability** of Pods : **Horizontal Pod Autoscaling (HPA)**: Horizontal Pod Autoscaling to

automatically scale the number of pods in a deployment based on observed CPU or custom metrics.

- HorizontalPodAutoscaler (HPA) resource that specifies the deployment to autoscale and the desired target CPU utilization or custom metric threshold.
- Configured HPA to scale within specified minimum and maximum replica counts.
- Monitor the HPA's behavior as load increases or decreases to ensure that pods are scaled up or down accordingly.

3. verify **Manual Scaling**: Experiment with manually scaling the number of replicas in your deployment

using the kubectl scale command. Verify that pods are added or removed as expected and that the

application remains responsive during the scaling process.

```
[root@master /]# kubectl get pods -o wide -n=project-website
NAME                    READY   STATUS    RESTARTS   AGE   IP               NODE      NOMINATED NODE   READINESS GATES
my-web-7f67bfb967-5rhrh   1/1     Running   0          26m   192.168.189.116   worker2   <none>           <none>
my-web-7f67bfb967-9w95t   1/1     Running   0          26m   192.168.189.107   worker2   <none>           <none>
my-web-7f67bfb967-kmdgb   1/1     Running   0          26m   192.168.189.112   worker2   <none>           <none>
[root@master /]#
[root@master /]# kubectl scale deploy my-web --replicas=5 -n=project-website
deployment.apps/my-web scaled
[root@master /]#
[root@master /]# kubectl get pods -o wide -n=project-website
NAME                    READY   STATUS              RESTARTS   AGE   IP               NODE      NOMINATED NODE   READINESS GATES
my-web-7f67bfb967-5rhrh   1/1     Running             0          26m   192.168.189.116   worker2   <none>           <none>
my-web-7f67bfb967-9w95t   1/1     Running             0          26m   192.168.189.107   worker2   <none>           <none>
my-web-7f67bfb967-kmdgb   1/1     Running             0          26m   192.168.189.112   worker2   <none>           <none>
my-web-7f67bfb967-twrl5   0/1     ContainerCreating   0          3s    <none>            worker1   <none>           <none>
my-web-7f67bfb967-v6dl8   1/1     Running             0          3s    192.168.235.145   worker1   <none>           <none>
[root@master /]# kubectl get pods -o wide -n=project-website
NAME                    READY   STATUS    RESTARTS   AGE   IP               NODE      NOMINATED NODE   READINESS GATES
my-web-7f67bfb967-5rhrh   1/1     Running   0          26m   192.168.189.116   worker2   <none>           <none>
my-web-7f67bfb967-9w95t   1/1     Running   0          26m   192.168.189.107   worker2   <none>           <none>
my-web-7f67bfb967-kmdgb   1/1     Running   0          26m   192.168.189.112   worker2   <none>           <none>
my-web-7f67bfb967-twrl5   1/1     Running   0          7s    192.168.235.143   worker1   <none>           <none>
my-web-7f67bfb967-v6dl8   1/1     Running   0          7s    192.168.235.145   worker1   <none>           <none>
[root@master /]#
```

4.Testing anti-affinity:test affinity and anti-affinity rules thoroughly to ensure they have the desired effect without causing unintended consequences. Monitor pod scheduling and cluster behavior after applying these rules to verify their effectiveness.

```
[root@master new-yaml-files]#
[root@master new-yaml-files]#
[root@master new-yaml-files]#
[root@master new-yaml-files]# kubectl get pods -o wide -n=project-website
NAME                    READY   STATUS    RESTARTS   AGE     IP               NODE      NOMINATED NODE   READINESS GATES
my-web-59bdc5c498-f6jts   1/1     Running   0          8m41s   192.168.235.152   worker1   <none>           <none>
my-web-59bdc5c498-rxl5q   1/1     Running   0          8m41s   192.168.189.120   worker2   <none>           <none>
my-web-6d55f5cff9-gvdjc   0/1     Pending   0          76s     <none>            <none>    <none>           <none>
my-web-6d55f5cff9-gxb5l   0/1     Pending   0          76s     <none>            <none>    <none>           <none>
[root@master new-yaml-files]#
[root@master new-yaml-files]#
[root@master new-yaml-files]#
```

References & Bibliography

1. Kubernetes Documentation

   https://kubernetes.io/docs/home/

2. Containerd Documentation

   https://containerd.io/docs/

3. AWS Cloud Documentation

   https://docs.aws.amazon.com/

4. Calico Documentation

   https://docs.tigera.io/

Project Link

Github: https://github.com/tarun-code/KUBERNETES-CLUSTER-PROJECT

Limitations

1. Vendor lock-in: While Kubernetes itself is open-source and portable, using it on AWS may tie you to AWS-specific services and features, making it harder to migrate to another cloud provider in the future.

2. Service limitations: Some AWS services may not be fully compatible or optimized for Kubernetes. For example, AWS Load Balancers may require additional configuration to work seamlessly with Kubernetes services.

3. Regional availability: Not all AWS services and features are available in every AWS region, which could impact your Kubernetes deployment's flexibility and availability

4. Effectiveness: Pod anti-affinity may not be as effective in a two-node cluster compared to larger clusters with more nodes. With only two nodes available, there are fewer options for Kubernetes to schedule pods away from each other while still maintaining resource balance.

Conclusion

deploying a Kubernetes cluster on AWS with a focus on achieving high availability, load balancing, and scaling of web server applications offers numerous benefits but also presents certain challenges and limitations. Throughout this project, we have explored the following key points:

1. Benefits: Kubernetes provides a powerful platform for automating the deployment, scaling, and management of containerized applications. By leveraging AWS infrastructure, we can take advantage of scalable compute resources, managed services, and global reach to build resilient and efficient Kubernetes clusters.

2. Challenges : Deploying and managing a Kubernetes cluster on AWS involves addressing various challenges, including complexity, cost, resource management, networking considerations, and security concerns. It requires expertise in Kubernetes administration, AWS services, infrastructure management, and DevOps practices.

3. Best Practices: Adhering to best practices for Kubernetes cluster design, architecture, configuration, and operation is essential for ensuring reliability, performance, and security. This includes proper resource allocation, networking setup, security hardening, monitoring, logging, and disaster recovery planning.

4. Continuous Improvement: Deploying a Kubernetes cluster on AWS is an iterative process that requires continuous monitoring, optimization, and refinement. By regularly reviewing cluster performance, analyzing metrics, identifying bottlenecks, and implementing optimizations, we can enhance the efficiency and resilience of our Kubernetes deployments over time.

In conclusion, deploying a Kubernetes cluster on AWS for high availability, load balancing, and scaling of web server applications is a complex but rewarding endeavor. By addressing challenges, leveraging best practices, and embracing automation and scalability principles, organizations can build robust and resilient Kubernetes environments that meet the demands of modern cloud-native applications.