Build with Kubernetes
A Self-Learning Edition

Kubernetes

From Pod to Prod

2025

Beginner to Absolute Advance

TABLE OF CONTENTS

5

KUBECONFIG, SERVICES, AND DEPLOYMENTS FILES EXPLAINED

Kubeconfig Files
Service File
Deployment files
HandsOn

6

DEPLOYING YOUR FIRST NODEJS APPLICATION ON KUBERNETES CLUSTER

Demonstration

7

KUBERNETES LABELS, SELECTORS, AND NODE SELECTORS

Labels
Labels-Selectors
Node Selector
HandsOn

8

REPLICATIONCONTROLLER & REPLICASET

ReplicationController
ReplicaSet
HandsOn

9

DEPLOYMENT OBJECT IN KUBERNETES

Use Cases for the Deployment Object
HandsOn

10

KUBERNETES CLUSTER(MASTER+WORKER NODE) USING KUBEADM ON AWS

ReplicationController
ReplicaSet
HandsOn

11 KUBERNETES NETWORKING (SERVICES)

Cluster IP
NodePort
LoadBalancer
ExternalName
HandsOn

12 KUBERNETES ADVANCED NETWORKING: CNI AND CALICO

Alternative of Calico CNI
Key Concepts and Real-time
Example
HandsOn

13 KUBERNETES VOLUMES AND LIVENESSPROBES

EmptyDir
HandsOn
hostPath
HandsOn
Persistent Volume
Persistent Volume Claim(PVC)
HandsOn
LivenessProbe(HealthCheck)
HandsOn

14 KUBERNETES CONFIGMAPS & SECRETS

ConfigMaps
HandsOn
CM from file
CM from the env file
CM from the YAML file
Secrets
HandsOn
Creating Secrets from literal
Secrets from file
Secrets from the env file
CM from the YAML file

15 KUBERNETES JOBS

Use cases
Key-Features
HandsOn

16 KUBERNETES INITCONTAINER

Use cases
HandsOn

17 KUBERNETES POD LIFECYCLE

Types of Pod state

18 KUBERNETES NAMESPACE

When should we consider Kubernetes Namespaces?
HandsOn

19 KUBERNETES RESOURCEQUOTA

Limit & Request
HandsOn

20 KUBERNETES AUTOSCALING

Types of Autoscaling
Key Features of Horizontal Pod AutoScaler
HandsOn

21 MULTI-CLUSTER KUBERNETES WITH HAPROXY

HAproxy
Demonstration

22 KUBERNETES INGRESS

Path-based routing
Host-based routing
HandsOn

23 KUBERNETES STATEFULSETS

Stateful VS Stateless applications
StatefulSets VS Deployment
Key Features StatefulSets
HandsOn

24 KUBERNETES DAEMONSET

DaemonSet Key Features
UseCases
HandsOn

25 KUBERNETES NETWORK POLICIES

Network Policy
Key Features
Use cases of Network Policy
HandsOn

26 KUBERNETES OPERATORS

What are Operators?
Features of Operators:
Custom Resource Definition
Custom Controller
Custom Resource

27 HELM & HELM CHARTS

Helm
Helm Deployment
HandsOn

28 HELM PROJECT

Deploy Flask Application using Helm Chart and many more features

29 AWS ELASTIC KUBERNETES SERVICE(EKS)

What is AWS EKS?
Key Features
AWS EKS Costing
HandsOn

30 AZURE KUBERNETES SERVICE(AKS)

What is AKS?
Key Features
Azure AKS Costing
HandsOn

31 GOOGLE KUBERNETES ENGINE (GKE)

What is GKE?
Key Features
HandsOn

32 END-TO-END DEVSECOPS KUBERNETES PROJECT

Tools
GitHub
AWS (Cloud)
Jenkins
Prometheus, Node Exporter, Grafana
Trivy, Sonarqube, OWASP
Kubernetes

Introduction to Kubernetes

Official Definition of Kubernetes

Kubernetes is an open-source container orchestration engine designed for automating the deployment, scaling, and management of containerized applications. This open-source project is hosted by the Cloud Native Computing Foundation (CNCF).

Understanding of Kubernetes and Docker

To grasp Kubernetes, also known as K8s, it's essential to have a foundation in Docker. In Docker, we deploy our applications inside containers. However, in Kubernetes, we manage containers on a larger scale, often numbering in the thousands or more, depending on the application's traffic.

Visualizing Docker and Kubernetes

In Docker, imagine a ship containing containers.
Now, in Kubernetes, picture that same ship, but this time, it has a steering wheel. Just like a captain operates the ship's wheel to make decisions about its course, Kubernetes acts as the "ship wheel" for managing containers.
Kubernetes is an open-source platform, meaning its source code is freely available for anyone to use, modify, and redistribute.

What are Monolithic Architecture and Microservices Architecture?

Monolithic Architecture:

Imagine a restaurant where everything happens in one big kitchen. This kitchen handles taking orders, cooking food, and serving customers all in a single place.
In this scenario, if the kitchen gets too crowded or if there's a problem with one part of the kitchen, it can affect the entire restaurant's operation.
If the chef is sick, the entire kitchen may come to a halt, impacting the entire dining experience.

Microservices Architecture:

Now, consider a food delivery service like Zomato or Swiggy. Instead of one big kitchen, they have a network of different restaurants, each specializing in a specific type of regional food or cuisine.
When you place an order, it's not prepared in a single kitchen rather, each restaurant (microservice) prepares its own portion of the order. These portions are then assembled and delivered to you.
If one restaurant has an issue, it doesn't necessarily impact the others. For example, if the burger place is busy, it won't affect the rolls restaurant's ability to fulfill orders.

Key Differences:

- Monolithic architecture is like a single kitchen handling all tasks, while microservices
- architecture is like multiple specialized restaurants working together.
- Monoliths are typically easier to set up and manage initially, while microservices offer more flexibility and scalability.
  Monoliths can have a single point of failure, while microservices are more fault-tolerant because a failure in one microservice doesn't necessarily affect the others.

In the end, Kubernetes helps to achieve microservice-based architecture which is good for business aspects, etc.

Why do we need Kubernetes?

After Docker came into the Picture, the deployment of the applications was very easy on the containers because containers are lightweight. But after some time, there were a lot of issues arose such as managing the huge amount of containers in the Production environment where Containers failed leading to huge Business losses. After Kubernetes came, it automates many tasks such as:

- Autoscaling of Containers according to the peak or normal hours.
- Load balancing of multiple containers.
- Automatically deployment of containers to the available nodes in the cluster.
- Self-healing if containers fail.

Kubernetes Origins and Open Source:

Kubernetes was created by Google in 2013 in Golang. Initially, Kubernetes was not open source but in 2014, google introduced Kubernetes open source and donated to CNCF.

Languages Supported by Kubernetes

Kubernetes supports both YAML and JSON for configuration.

Features of Kubernetes

- **AutoScaling** : Kubernetes supports two types of autoscaling horizontal and vertical scaling for large-scale production environments which helps to reduce the downtime of the applications.
- **Auto Healing** : Kubernetes supports auto healing which means if the containers fail or are stopped due to any issues, with the help of Kubernetes components(which will talk in upcoming days), containers will automatically repaired or heal and run again properly.
- **Load Balancing** : With the help of load balancing, Kubernetes distributes the traffic between two or more containers.
- **Platform Independent** : Kubernetes can work on any type of infrastructure whether it's On-premises, Virtual Machines, or any Cloud.
- **Fault Tolerance** : Kubernetes helps to notify nodes or pods failures and create new pods or containers as soon as possible
- **Rollback** : You can switch to the previous version.
- **Health Monitoring of Containers** : Regularly check the health of the monitor and if any container fails, create a new container.
- **Orchestration** : Suppose, three containers are running on different networks (On-premises, Virtual Machines, and On the Cloud). Kubernetes can create one cluster

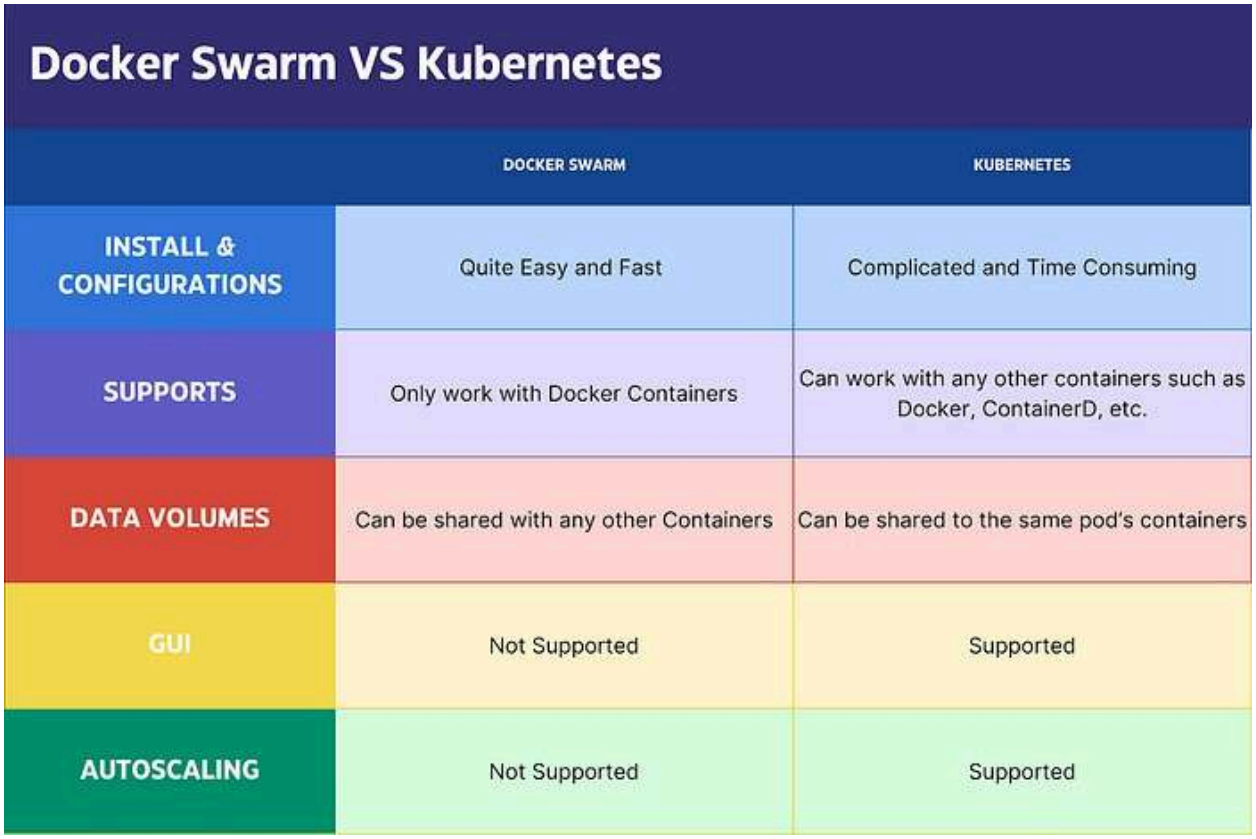
that has all three running containers from different networks.

Alternatives of Kubernetes

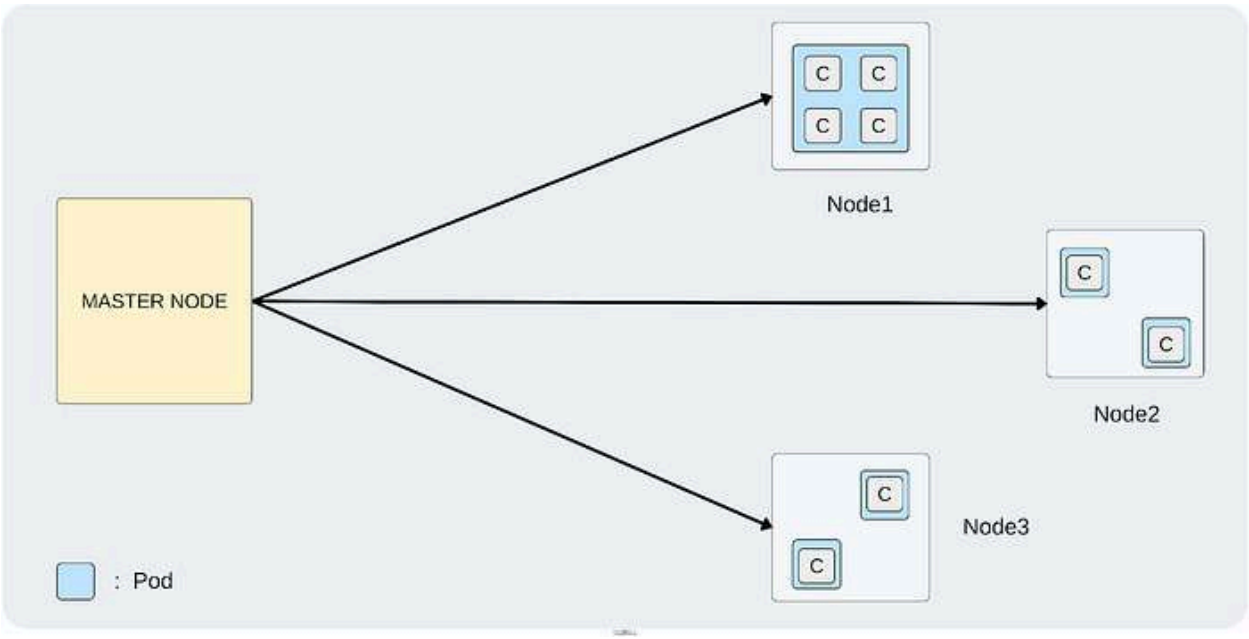
- **Docker Swarm**
- **Apache Mesos**
- **Openshift**
- **Nomad, etc**

We don't need to know the other alternative in depth except Docker Swarm as our main focus is Kubernetes.

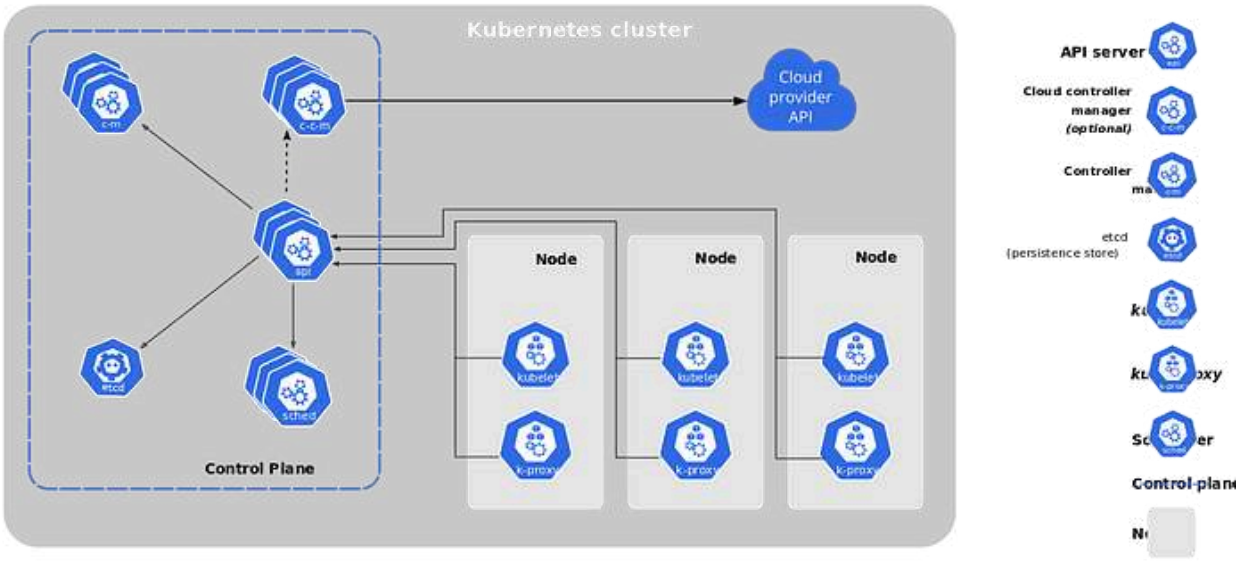
Difference between Docker Swarm and Kubernetes



Master-Slave/ Client-Server Architecture in Kubernetes


C
C
C
C
Node1
MASTER NODE
C
C
Node2
C
C
Node3
: Pod

Kubernetes Architecture- Master Node



Kubernetes Architecture By Kubernetes

Kubernetes follows client-server architecture where the Master Node and Worker node exist which constitutes a 'Kubernetes Cluster'. We can have multiple worker nodes and Master nodes according to the requirement.

Control Plane

The control plane components, including the API server, etcd, scheduler, and controller manager, are typically found on the master node(s) of a Kubernetes cluster. These components are responsible for managing and controlling the cluster as a whol

Master Node

The master node is responsible for the entire Kubernetes cluster and manages all the activities inside the cluster in which master nodes communicate with the worker node to run the applications on the containers smoothly. Master Node has four primary components which help to manage all the things that we have discussed earlier:

1. **API Server:** In Simple terms, after installing the kubectl on the master node developers run the commands to create pods. So, the command will go to the API Server, and then, the API Server forwards it to that component which will help to create the pods. In other words, the API Server is an entry point for any Kubernetes task where the API Server follows the hierarchical approach to implement the things.

2. **Etcd** : Etcd is like a database that stores all the pieces of information of the Master node and Worker node(entire cluster) such as Pods IP, Nodes, networking configs, etc. Etcd stored data in key-value pair. The data comes from the API Server to store in etc.

3. **Controller Manager:** The controller Manager collects the data/information from the API Server of the Kubernetes cluster like the desired state of the cluster and then decides what to do by sending the instructions to the API Server.

4. **Scheduler** : Once the API Server gathers the information from the Controller Manager, the API Server notifies the Scheduler to perform the respective task such as increasing the number of pods, etc. After getting notified, the Scheduler takes action on the provided work.

Let's understand all four components with a real-time example.

Master Node — Mall Management:

- In a shopping mall, you have a management office that takes care of everything. In
- Kubernetes, this is the Master Node.
  The Master Node manages and coordinates all activities in the cluster, just like mall management ensures the mall runs smoothly.

kube-apiserver — Central Control Desk:

- Think of the kube-apiserver as the central control desk of the mall. It's where all
- requests (like store openings or customer inquiries) are directed.
  Just like mall management communicates with stores, kube-apiserver communicates with all Kubernetes components.

etcd — Master Records:

- etcd can be compared to the master records of the mall, containing important information like store locations and hours.

- It’s a key-value store that stores configuration and cluster state data.

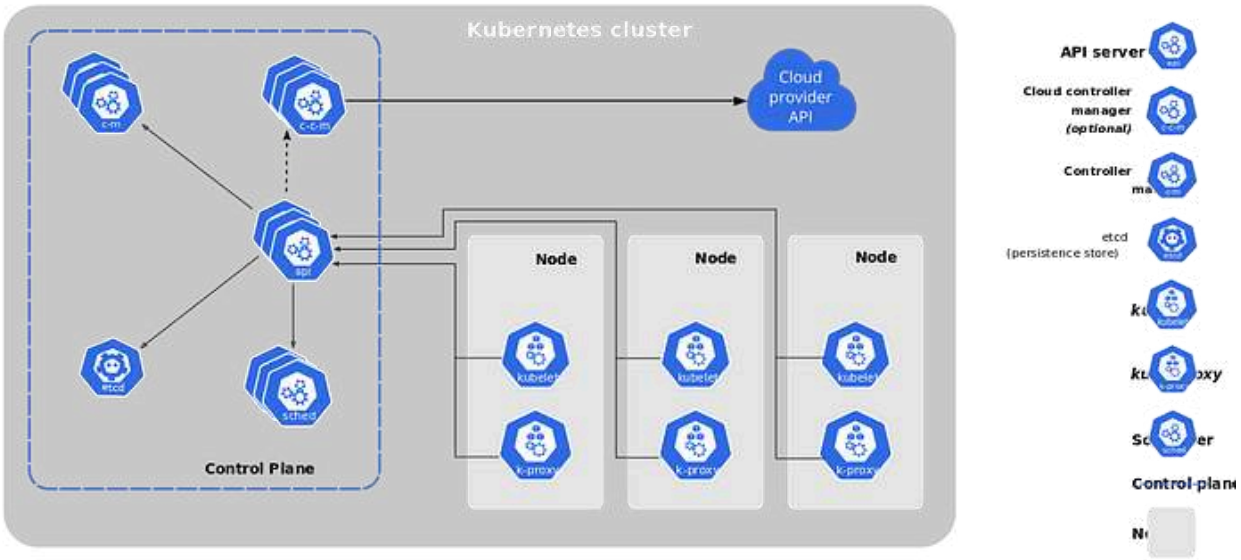
kube-controller-manager — Task Managers:

- Imagine having specialized task managers for different mall departments, like security and maintenance.
- In Kubernetes, the kube-controller-manager handles various tasks, such as ensuring the desired number of Pods are running.

kube-scheduler — Scheduler Manager:

- Think of the kube-scheduler as a manager who decides which employees (Pods) should work where (on which Worker Node).
- It ensures even distribution and efficient resource allocation.

Kubernetes Architecture- Worker Node



Kubernetes Architecture By Kubernetes(Credit)

Worker Node

The Worker Node is the mediator who manages and takes care of the containers and communicates with the Master Node which gives the instructions to assign the resources to the containers scheduled. A Kubernetes cluster can have multiple worker nodes to scale resources as needed.

The Worker Node contains four components that help to manage containers and communicate with the Master Node:

1. **Kubelet** : kubelet is the primary component of the Worker Node which manages the Pods and regularly checks whether the pod is running or not. If pods are not working properly, then kubelet creates a new pod and replaces it with the previous one because the failed pod can't be restarted hence, the IP of the pod might be changed. Also, kubelet gets the details related to pods from the API Server which exists on the Master Node.
2. **Kube-proxy** : kube-proxy contains all the network configuration of the entire cluster such as pod IP, etc. Kube-proxy takes care of the load balancing and routing which comes under networking configuration. Kube-proxy gets the information about pods from the API Server which exists on Master Node.

3. **Pods** : A pod is a very small unit that contains a container or multiple containers where the application is deployed. Pod has a Public or Private IP range that distributes the proper IP to the containers. It's good to have one container under each pod.
4. **Container Engine** : To provide the runtime environment to the container, Container Engine is used. In Kubernetes, the Container engine directly interacts with container runtime which is responsible for creating and managing the containers. There are a lot of Container engines present in the market such as CRI-O, containerd, rkt(rocket), etc. But Docker is one of the most used and trusted Container Engine. So, we will use that in our upcoming day while setting up the Kubernetes cluster.

Let's continue to understand all four components with a real-time example.

Worker Nodes — Storefronts:

Kubelet — Store Managers:

- In each store (Worker Node), you have a store manager (Kubelet) who ensures
- employees (Pods) are working correctly.
  Kubelet communicates with the Master Node and manages the Pods within its store.

kube-proxy — Customer Service Desk:

- kube-proxy acts like a customer service desk in each store. It handles customer inquiries (network requests) and directs them to the right employee (Pod).
- It maintains network rules for load balancing and routing.

Container Runtime — Employee Training:

- In each store, you have employees (Pods) who need training to perform their tasks.
- The container runtime (like Docker) provides the necessary training (runtime environment) for the employees (Pods) to execute their tasks.

Setting up Minikube on Your Machine

Why Set Up Minikube?

Before we begin, you might be wondering why it's essential to set up Minikube. Well, Minikube provides an excellent environment for learning and experimenting with Kubernetes without the need for a full-scale cluster. It's perfect for developers and enthusiasts who want to get hands-on experience with Kubernetes in a controlled environment.

Prerequisites

To follow along with this tutorial, you'll need the following:

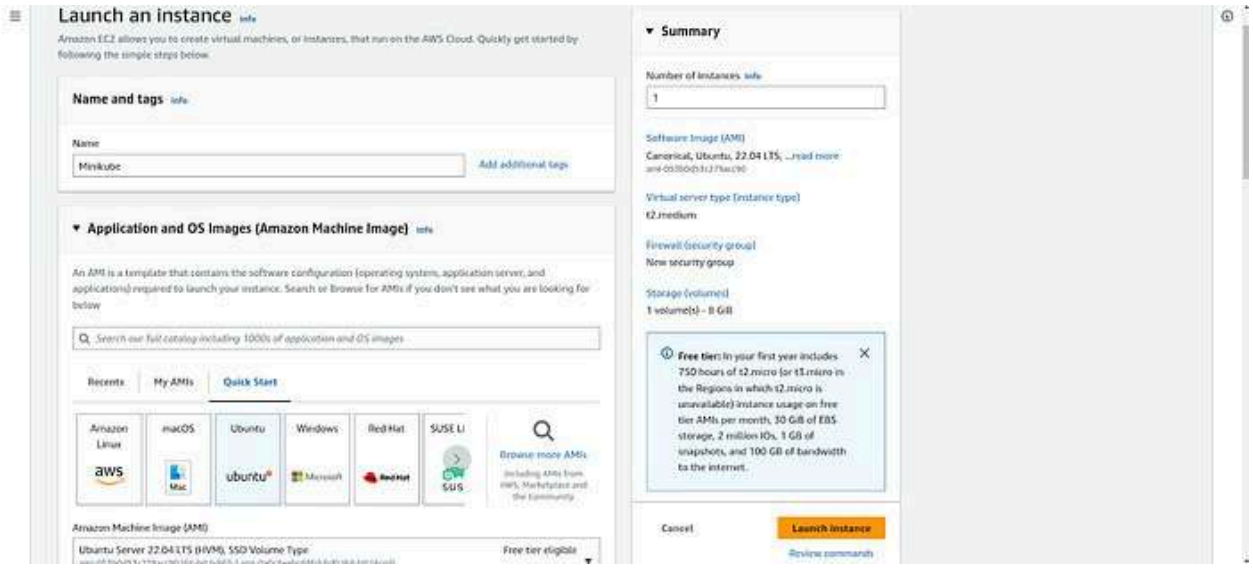
- An AWS account (if you're setting up on an AWS instance).
- Basic knowledge of AWS and Linux terminal commands.

Let's get started!

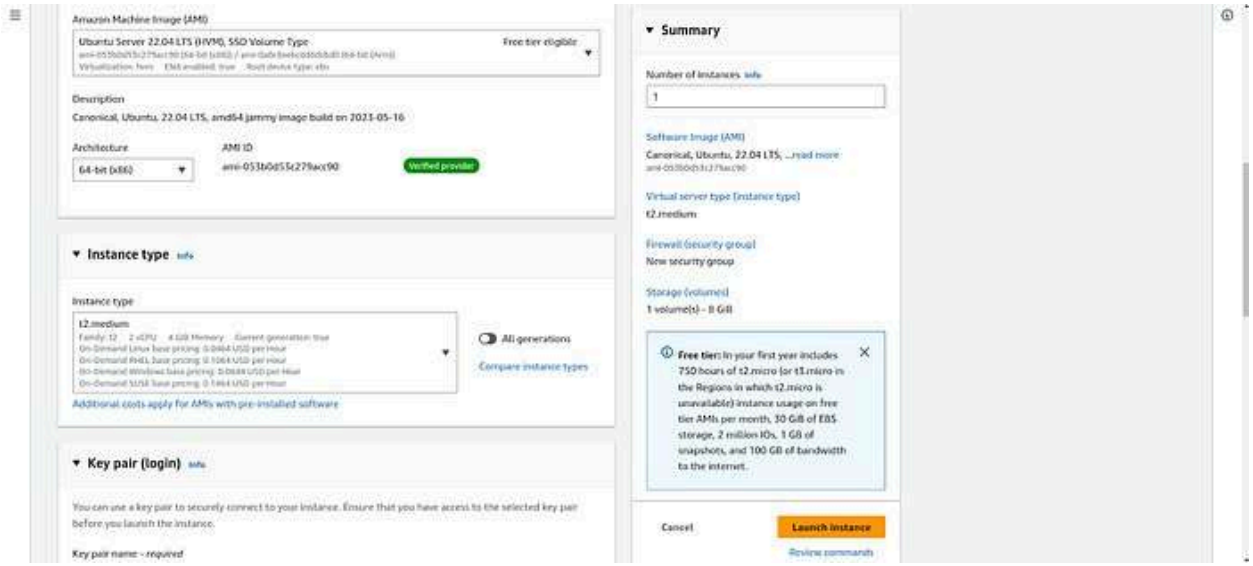
Setting Up Minikube on AWS Instance

Here, I am creating an EC2 Instance to set up minikube on the server. If you are comfortable setting up minikube on your local then feel free to jump on the minikube setup.

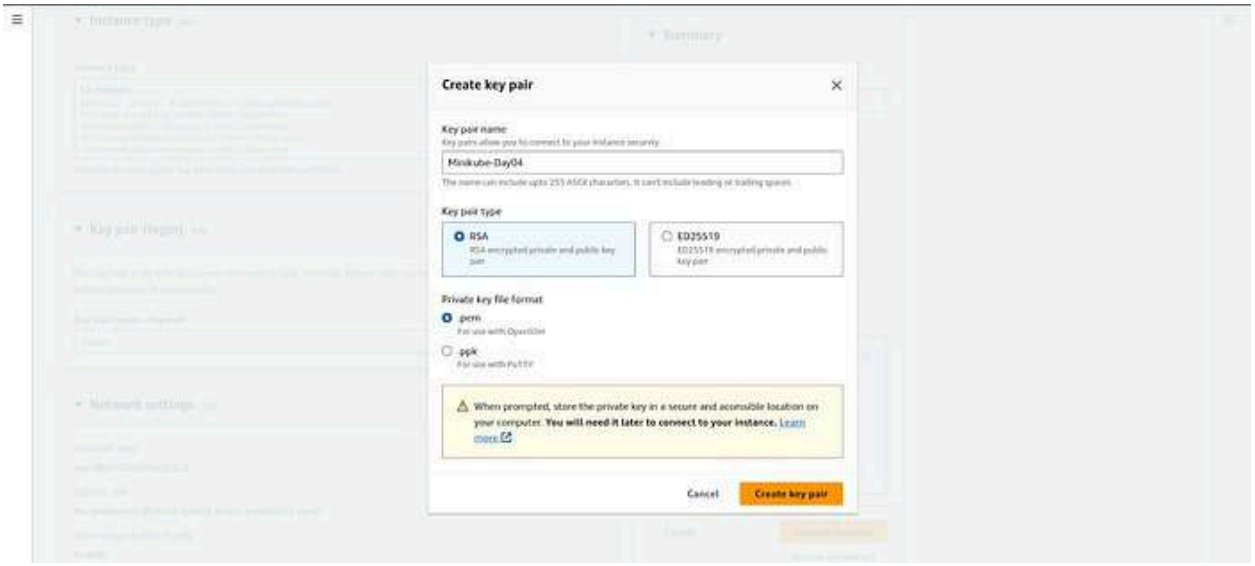
Enter the name of the machine and select the Ubuntu22.04 AMI Image.



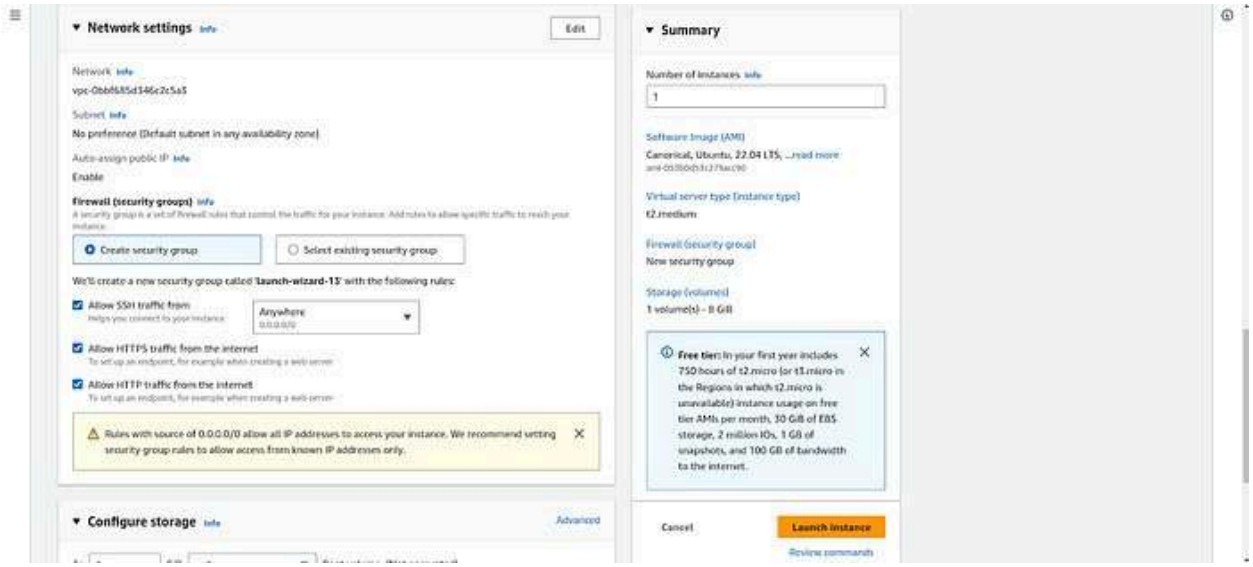
Make sure to select the t2.medium instance type as Master node 2CPU cores which is present in the t2.medium instance type.



Create a new key pair and select the Private key file format according to your OS(For Windows select .ppk or for Linux select .pem).



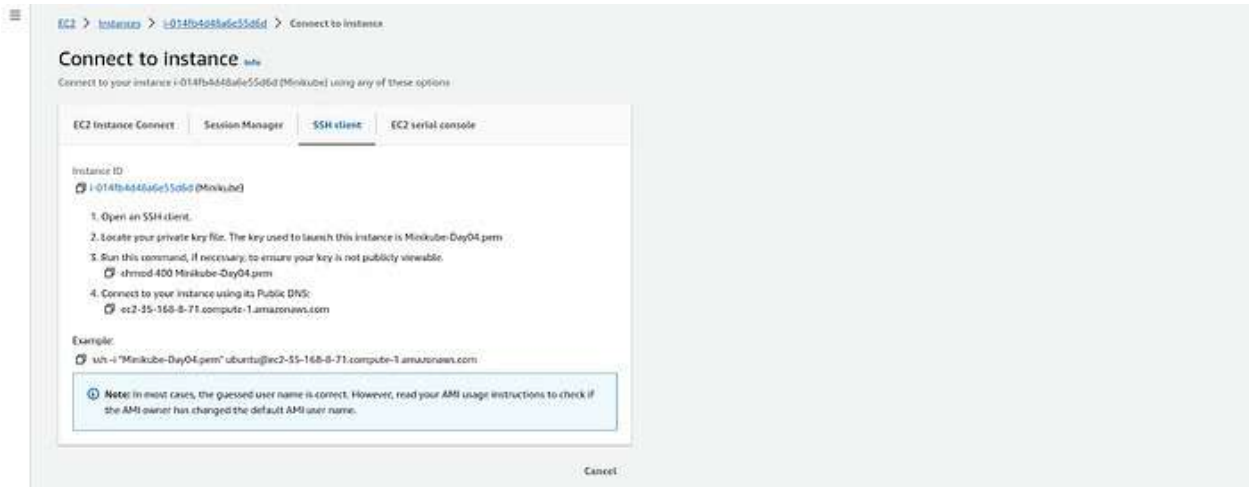
Open port 22 and rest you can leave it.



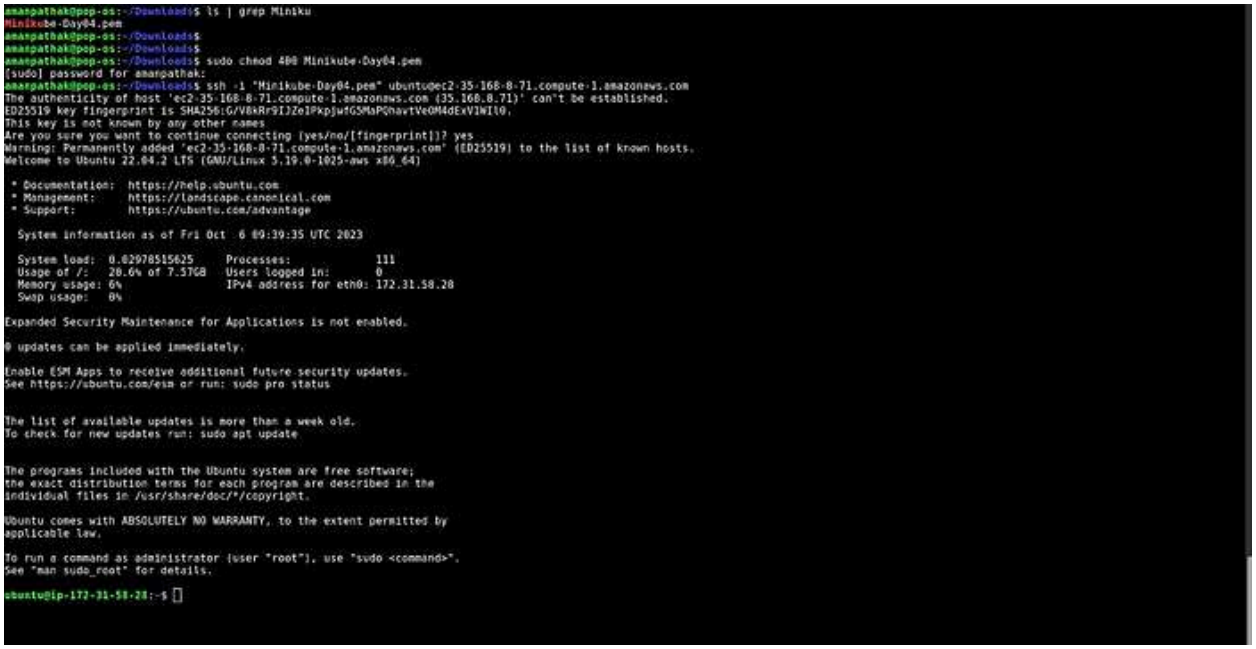
Now, go to your Downloads folder or where you have downloaded your pem file and change the permission by running the command ‘chmod 400 <Pem_file_name>

```
amanpathak@pop-os:~/Downloads$ ls | grep Miniku
Minikube-Day04.pem
amanpathak@pop-os:~/Downloads$
amanpathak@pop-os:~/Downloads$
amanpathak@pop-os:~/Downloads$ sudo chmod 400 Minikube-Day04.pem
[sudo] password for amanpathak:
amanpathak@pop-os:~/Downloads$
```

Now, connect your instance by copying the given command below.



As you can see I logged in to the Instance.



Now, run the following commands to install minikube on your local machine or AWS machine.

```
sudo apt update -y && sudo apt upgrade -y

sudo reboot
```

After 3 to 4 minutes, reconnect with the instance through ssh

```
sudo apt install docker.io

sudo usermod -aG docker $USER && newgrp docker

sudo apt install -y curl wget apt-transport-https

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube

minikube version
```

I have given the line break in the below curl command, Kindly avoid the break and any whitespaces after `. You can refer to the below screenshot.

```
curl -s
https://
tl
```

```
chmod +x kubectl

sudo mv kubectl /usr/local/bin

kubectl version -o yaml

minikube start - vm-driver=docker
```

Now, to verify the installation you can run the given command and if you get the result in the snippet then your installation is completed.

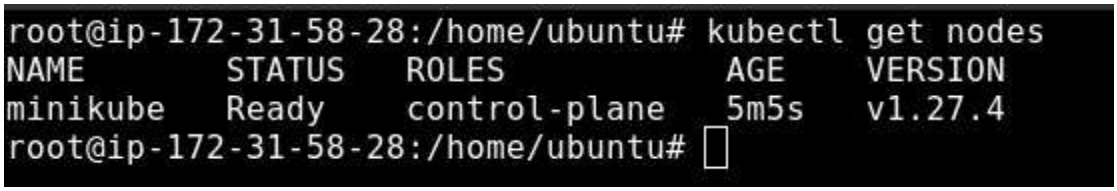
```
minikube status
```

You can also validate your kubectl version by running the command.

```
kubectl version
```

Now, run the given command after 4 to 5 minutes which will show the nodes.

```
kubectl get nodes
```



Creating Your First Pod

To get some hands-on experience, let's create a simple manifest file and deploy a pod on your Minikube cluster. Don't worry if you don't understand everything in the manifest file; we'll cover that in later days.

Create a new file and copy the given content to your file without editing anything. While running the manifest file, if you get any error then it must be related to the indentation of the file. So, check the file again.

```
vim Day04.yml
kind: Pod
apiVersion: v1
metadata:
 name: testpod
spec:
 containers:
  -name: c00
    image: ubuntu
    command: ["/bin/bash", "-c", "while true; do echo Hello-Kubernetes; sleep 5 ; done"]
```

```
  -name: container2
    image: ubuntu
    command: ["/bin/bash", "-c", "while true; do echo Second Container is still running; sleep
3; done"]
 restartPolicy: Never
```

Deploy the Pod:

Run the following command to deploy the pod:

```
kubectl apply -f Day04.yml
```

List Pods:

To list all the pods, use this command:

```
kubectl get pods
```

```
root@ip-172-31-58-28:/home/ubuntu# kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
pod1   2/2     Running   0          35s
root@ip-172-31-58-28:/home/ubuntu#
```

Check Logs:

You can check the logs of the primary container with:

```
kubectl logs -f pod1
```

```
This is Day04 and We are setting up the Minikube
This is Day04 and We are setting up the Minikube
This is Day04 and We are setting up the Minikube
This is Day04 and We are setting up the Minikube
```

To check the logs of the primary container, specify the container name:

```
kubectl logs -f pod1 -c container1
```

```
This is Day04 and We are setting up the Minikube
This is Day04 and We are setting up the Minikube
This is Day04 and We are setting up the Minikube
This is Day04 and We are setting up the Minikube
```

To check the logs of the second container, specify the container name:

kubectl logs -f pod1 -c container2

```
Second Container is still running
Second Container is still running
Second Container is still running
Second Container is still running
```

Delete the Pod:

To delete the pod, use this command:

kubectl delete pod pod1

```
root@ip-172-31-58-28:/home/ubuntu# kubectl delete pod pod1
pod "pod1" deleted
```

To list the IP of the pod, use the below command.

kubectl pod pod1 -c container1 — hostname -i

```
root@ip-172-31-58-28:/home/ubuntu# kubectl exec pod1 -c container1 -- hostname -i
10.244.0.4
root@ip-172-31-58-28:/home/ubuntu# kubectl exec pod1 -c container2 -- hostname -i
10.244.0.4
root@ip-172-31-58-28:/home/ubuntu#
```

To delete the pod by specifying the manifest file name

kubectl delete -f Day04.yml

```
root@ip-172-31-58-28:/home/ubuntu# kubectl delete -f Day04.yml
pod "pod1" deleted
root@ip-172-31-58-28:/home/ubuntu#
```

Kubeconfig, Services, and Deployments Files Explained

Kubeconfig Files

- **Purpose** : Kubeconfig files are used for cluster access and authentication. Kubeconfig defines how ‘kubectl’ or any other Kubernetes clients interact with the Kubernetes cluster.
- **Contents** : The Kubeconfig file contains information about the cluster, user credentials, certificates, and context.
- **Usage** : Kubeconfig files are used by Administrators, developers, or CI/CD systems to authenticate the Kubernetes cluster. They decide who can access and how to access the cluster.

Kubeconfig files can be stored in the user’s home directory (~/.kube/config) or specified using the KUBECONFIG environment variable.

Service File

- **Purpose** : Service files contain all information about networking. The service file defines how networking will be handled on the cluster. Also, the Service file enabled the load balancing option for the applications which is a premium feature of Kubernetes.
- **Contents** : The service file specifies the service’s name, type(ClusterIP, NodePort,
- LoadBalancer, etc[Discuss in Upcoming Blogs]), and selectors to route traffic to pods.
  Usage : Service files are used by developers and administrators to expose and connect applications within the Kubernetes cluster.

Note : Services can also be used for internal communication between Pods within the cluster, not just for exposing applications externally.

Deployment files

- **Purpose** : Deployment files contain all information about the application and define how the application or microservices will be deployed on the Kubernetes cluster. In deployment files, we can define the desired state, pod replicas, update strategies, and pod templates.
- **Contents** : Deployment files define the desired state of a deployment, pod replicas,
- container images, and resource limits.
  Usage : Deployment files are mainly used by developers and administrators to manage the application lifecycle within Kubernetes. They enable declarative application management, scaling, and rolling updates.

Practical Examples

To make things even clearer, let's dive into some practical examples:

Kubeconfig file explained

```
apiVersion: v1
kind: Config
clusters:
-name: my-cluster
 cluster:
   server: https://api.example.com
   certificate-authority-data: <ca-data>
users:
-name: my-user
 user:
   client-certificate-data: <client-cert-data>
   client-key-data: <client-key-data>
contexts:
-name: my-context
 context:
   cluster: my-cluster
   user: my-user
   namespace: my-namespace
current-context: my-context
```

In this example,

- **apiVersion** and **kind** define the resource type.
- **clusters** specifies the clusters with its server and URL and Certificate Authority(CA) data. Here we have to define the server link or Kubernetes API Server of the Kubernetes cluster. So, when we run any command using kubectl then kubectl interacts with the given link or Kubernetes API Server of the Kubernetes cluster.
- **users** specify the users with their client certificate and client key name. So, only authorized users can access the Kubernetes cluster.
  contexts specify the cluster, user, and namespace information that has been defined above. You can create multiple contexts and switch between any different clusters at any time.
- **current-context** specifies that on which cluster the command should run. If you set the current-context one time then you won't have to specify again and again while running the commands.

Service file explained

```
apiVersion: v1
kind: Service
metadata:
 name: my-app-service
spec:
 selector:
```

```
  app: my-app
 ports:
-protocol: TCP
  port: 80
  targetPort: 8080
```

In this example,

- **apiVersion** and **kind** specify the resource type
- **metadata** specify the name of the service
- **spec** specify the desired state of the Service
- **selector** specifies on which pod the configurations will be invoked. If the pod label matches by app value then it will apply the configuration on that pod
- In the **ports** section, **protocol** specifies the network protocol such as TCP, UDP, etc.
- **ports** specifies on which port the service listens for the incoming traffic from the external sources.
  targetports specify on which port the pod is listening.

Example for port and targetports:

Suppose you have a React.js application running inside a Kubernetes Pod, and it's configured to listen on port 3000 within the Pod. However, you want external traffic to reach your application on port 80 because that's the standard HTTP port. To achieve this, you create a Kubernetes Service with a targetPort set to 3000 and a port set to 80. The Service acts as a bridge, directing incoming traffic from port 80 to the application running on port 3000 inside the Pod. This redirection allows users to access your React.js app seamlessly over the standard HTTP port.

Deployment file explained

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: my-app
spec:
 replicas: 3
 selector:
  matchLabels:
   app: my-app
 template:
  metadata:
   labels:
    app: my-app
  spec:
   containers:
   -name: my-app-container
    image: my-app-image:latest
```

In this example,

- **apiVersion** and **kind** define the resource type
- **metadata** specify the details of deployment such as the name of the deployment, and
- labels.
- **spec** defines the desired state of the Deployment.
- **replicas** specify the desired number of pods to maintain.
- The **selector** specifies on which pod the replica configuration should be applied with the
- help of the label of the pod.
- **template** describes the pod template and how deployment will use it to create new pods.
- **containers** will list the containers to run within the pod.
- **name** specifies the name of the container
  image specifies the name that will be used to run the container. The image will be a Docker Image.
  containerport specifies the port at which the container will listen for incoming traffic.

Deploying Your First Nodejs Application on Kubernetes Cluster

---

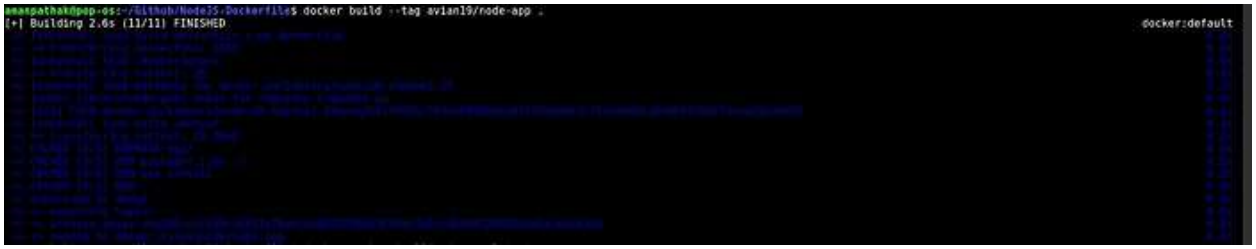
To follow this, you need to install minikube on your local/AWS machine. If you don't know then you can refer to my step-by-step blog which will help you to do it.

Day 04: Setting up Minikube on Your Local Machine or AWS Instance | by Aman Pathak | DevOps.dev

Step 1: Create a Docker Image

Assuming you're already familiar with Docker, let's create a Docker image for your Node.js project. Open your terminal and use the following command to build the image:

```
docker build --tag avian19/node-app .
```



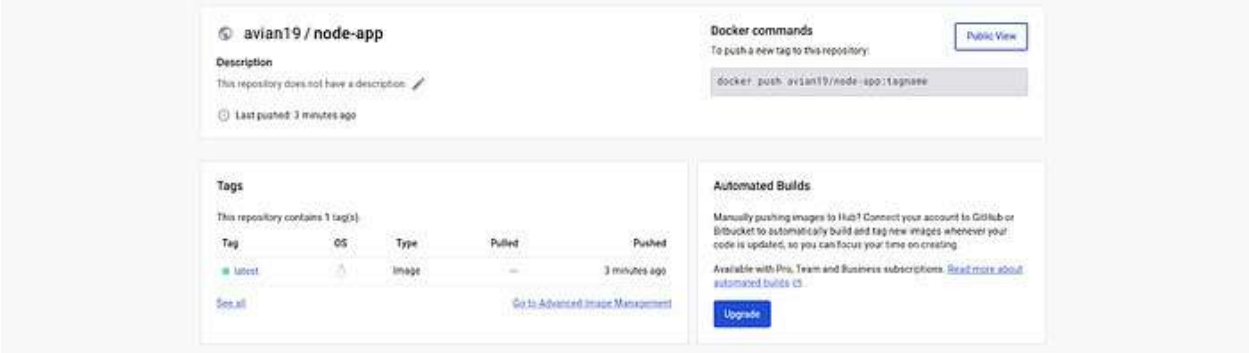
Step 2: Push the Docker Image to Docker Hub

To share your Docker image with your Kubernetes cluster, you can push it to Docker Hub. First, log in to Docker Hub using your terminal:

```
docker login
```

Then, push the Docker image:

You can confirm that your image has been successfully pushed to Docker Hub.



Step 3: Prepare Kubernetes Deployment and Service Files

Create a dedicated directory for your Node.js application's deployment. Inside this directory, add the contents of your deployment.yml and service.yml files.
deployment.yml file

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: node-app-deployment
 labels:
   app: node-app
spec:
 replicas: 2
 selector:
   matchLabels:
     app: node-app
 template:
   metadata:
     labels:
       app: node-app
   spec:
     containers:
     -name: node-container
       image: avian19/node-app:latest
       ports:
       -containerPort: 3000
```

service.yml file

```
apiVersion: v1
kind: Service
metadata:
 name: node-app-service
spec:
 selector:
   app: node-app
 type: LoadBalancer
```

```
ports:
-protocol: TCP
  port: 5000
  targetPort: 3000
  nodePort: 30001
```

```
ubuntu@ip-172-31-49-109:~$ mkdir node-app
ubuntu@ip-172-31-49-109:~$ cd node-app/
ubuntu@ip-172-31-49-109:~/node-app$ sudo vim deployment.yml
ubuntu@ip-172-31-49-109:~/node-app$ sudo vim service.yml
ubuntu@ip-172-31-49-109:~/node-app$
```

Step 4: Deploy Pods

To deploy the pods, use the deployment.yml file with the following command:

```
kubectl apply -f deployment.yml
```

```
ubuntu@ip-172-31-49-109:~/node-app$ kubectl apply -f deployment.yml
deployment.apps/node-app-deployment created
ubuntu@ip-172-31-49-109:~/node-app$ kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
node-app-deployment-99d848784-g568r   1/1     Running   0          13s
node-app-deployment-99d848784-nf9b6   1/1     Running   0          13s
ubuntu@ip-172-31-49-109:~/node-app$
```

Step 5: Deploy Services

Next, deploy the services using the service.yml file:

```
kubectl apply -f service.yml
```

```
ubuntu@ip-172-31-49-109:~/node-app$ kubectl apply -f service.yml
service/node-app-service created
ubuntu@ip-172-31-49-109:~/node-app$ kubectl get svc
NAME               TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes         ClusterIP      10.96.0.1        <none>        443/TCP          85m
node-app-service   LoadBalancer   10.101.148.185   <pending>     5000:30001/TCP   5s
ubuntu@ip-172-31-49-109:~/node-app$
```

Step 6: Validate the Deployment

You can check the status of your deployment by running the following command:

```
kubectl get deployment
```

```
ubuntu@ip-172-31-49-109:~/node-app$ kubectl get deployment
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
node-app-deployment   2/2     2            2           2m36s
ubuntu@ip-172-31-49-109:~/node-app$ .
```

Step 7: Access Your Application

To access your deployed application, use the following command to get the URL:

```
minikube service node-app-service
```

You can now use curl to access the content of your Node.js application through the provided URL.

In nodejs code, you can see that the content is the same at both places.

Kubernetes Labels, Selectors, and Node Selectors

Labels

- Labels are used to organize Kubernetes Objects such as Pods, nodes, etc.
- You can add multiple labels over the Kubernetes Objects.
- Labels are defined in key-value pairs.
- Labels are similar to tags in AWS or Azure where you give a name to filter the
- resources quickly.
  You can add labels like environment, department or anything else according to you.

Labels-Selectors

Once the labels are attached to the Kubernetes objects those objects will be filtered out with the help of labels-Selectors known as Selectors.

The API currently supports two types of label-selectors equality-based and set-based. Label selectors can be made of multiple selectors that are comma-separated.

Node Selector

Node selector means selecting the nodes. If you are not aware of what is nodes. There are two types of nodes Master Nodes and Worker Nodes.

Master Nodes is responsible for the entire Kubernetes Cluster that communicates with the Worker Node and runs the applications on containers smoothly. Master nodes can have multiple Worker Nodes.

Worker Nodes work as a mediator where the nodes communicate with Master nodes and run the applications on the containers smoothly.

So, the use of node selector is choosing the nodes which means on which worker node the command should be applied. This is done by Labels where in the manifest file, we mentioned the node label name. While running the manifest file, master nodes find the node that has the same label and create the pod on that container. Make sure that the node must have the label. If the node doesn't have any label then, the manifest file will jump to the next node.

Label and Label-Selector HandsOn

If you don't understand properly, don't worry. We will do hands-on which will make it easy to understand the concepts of labels, labels-selectors, and node selectors.

YAML file

```
apiVersion: v1
kind: Pod
metadata:
 name: day07
 labels:
   env: testing
   department: DevOps
spec:
 containers:
   -name: containers1
     image: ubuntu
     command: ["/bin/bash", "-c", "while true; do echo This is Day07 of 30DaysOfKubernetes;
sleep 5 ; done"]
```

After creating the pods by the following command.
List the pods to see the labels that are attached to the pod by the given command
kubectl get pods — show-labels

```
ubuntu@ip-172-31-63-3:~$ kubectl apply -f pod.yml
pod/day07 created
ubuntu@ip-172-31-63-3:~$ kubectl get pods --show-labels
NAME    READY   STATUS    RESTARTS   AGE   LABELS
day07   1/1     Running   0          13s   department=DevOps,env=testing
ubuntu@ip-172-31-63-3:~$
```

I have created one more manifest file that doesn't have any label as you can see in the below screenshot.

```
ubuntu@ip-172-31-63-3:~$ sudo vim pod2.yml
ubuntu@ip-172-31-63-3:~$ kubectl apply -f pod2.yml
pod/testpod created
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
day07     1/1     Running   0          2m44s
testpod   2/2     Running   0          8s
ubuntu@ip-172-31-63-3:~$ kubectl get pods --show-labels
NAME      READY   STATUS    RESTARTS   AGE     LABELS
day07     1/1     Running   0          2m53s   department=DevOps,env=testing
testpod   2/2     Running   0          17s     <none>
ubuntu@ip-172-31-63-3:~$
```

Now, I want to list those pods that have the label env=testing.

As we have discussed earlier, there are two types of label-selectors equality and set-based.

This is the example of equality based where we used equalsTo(=).

```
kubectl get pods -l env=testing
```

```
ubuntu@ip-172-31-63-3:~$ kubectl get pods -l env=testing
NAME    READY   STATUS    RESTARTS   AGE
day07   1/1     Running   0          4m13s
ubuntu@ip-172-31-63-3:~$
```

Now, here I want to list those pods that don't have the label department=DevOps.

```
kubectl get pods -l department!=DevOps
```

```
ubuntu@ip-172-31-63-3:~$ kubectl get pods -l department!=DevOps
NAME      READY   STATUS    RESTARTS   AGE
testpod   2/2     Running   0          3m53s
ubuntu@ip-172-31-63-3:~$
```

Now, Suppose I forgot to add the label through the declarative(via manifest) method. So, I can add labels after creating the pods as well which is known as the imperative(via command) method.

In the below screenshot, I have added the new label location and given it the value India.

```
kubectl label pods day07 Location=India
```

```
ubuntu@ip-172-31-63-3:~$ kubectl label pods day07 Location=India
pod/day07 labeled
ubuntu@ip-172-31-63-3:~$ kubectl get pods --show-labels
NAME      READY   STATUS    RESTARTS   AGE     LABELS
day07     1/1     Running   0          9m34s   Location=India,department=DevOps,env=testing
testpod   2/2     Running   0          6m58s   <none>
ubuntu@ip-172-31-63-3:~$
```

As we discussed the type of label-selector, let's see the example of a set-based label-selector where we use in, notin, and exists.

In the below screenshot, we are trying to list all those pods that have an env label with value for either testing or development.

```
kubectl get pods -l 'env in (testing,development)'
```

```
ubuntu@ip-172-31-63-3:~$ kubectl get pods -l 'env in (testing,development)'
NAME    READY   STATUS    RESTARTS   AGE
day07   1/1     Running   0          13m
ubuntu@ip-172-31-63-3:~$
```

Here, we are trying to list all those pods that don't have the India or US value for the Location key in the Label.

```
kubectl get pods -l 'Location notin (India, US)'
```

```
ubuntu@ip-172-31-63-3:~$ kubectl get pods -l 'Location notin (India,US)'
NAME      READY   STATUS    RESTARTS   AGE
testpod   2/2     Running   0          12m
ubuntu@ip-172-31-63-3:~$
```

We can also delete the pods as per the label.

Here, We have deleted all those pods that don't have the Chinda value for the location key in the Label.

```
kubectl delete pod -l Location!=China
```

```
ubuntu@ip-172-31-63-3:~$ kubectl delete pod -l Location!=China
pod "day07" deleted
pod "testpod" deleted
ubuntu@ip-172-31-63-3:~$ .
```

Here, We have completed the HandsOn for the label and label-selector. You can explore more yourself.

Let's move it to **node-selector**

nodeSelector HandsOn

As you remember, we have set up minikube on our machine. So, our master and worker node is on the same machine.

To list the nodes on the master node, use the command

```
kubectl get nodes
```

```
ubuntu@ip-172-31-63-3:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   72m   v1.27.4
ubuntu@ip-172-31-63-3:~$
```

```
apiVersion: v1
kind: Pod
metadata:
 name: nodelabels
 labels:
   env: testing
spec:
 containers:
  -name: container1
```

```
    image: ubuntu
    command: ["/bin/bash", "-c", "while true; do echo Node-Selector Example; sleep 5 ;
done"]
 nodeSelector:
  hardware: t2-medium
```

Here, I have created the pod but when I list the pods the status is pending for the created pod.

```
ubuntu@ip-172-31-63-3:~$ kubectl apply -f pod3.yml
pod/nodelabels created
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME         READY   STATUS    RESTARTS   AGE
nodelabels   0/1     Pending   0          5s
ubuntu@ip-172-31-63-3:~$
```

If you observe the manifest file, the master node is looking for that worker node which has the label hardware=t2-medium.

```
  containers:
    - name: container1
      image: ubuntu
      command: ["/bin/bash", "-c", "while true; do echo Node-Selector Example
  nodeSelector:
   hardware: t2-medium
~
```

As you can see there is no label added like hardware=t2-medium

```
ubuntu@ip-172-31-63-3:~$ kubectl get nodes --show-labels
NAME       STATUS   ROLES           AGE   VERSION   LABELS
minikube   Ready    control-plane   81m   v1.27.4   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=minikube,kubernetes.io/os=linux,
minikube.k8s.io/commit=fd7ecd9c4599bef9f04c0986c4a0187f98a4396e,minikube.k8s.io/name=minikube,minikube.k8s.io/primary=true,minikube.k8s.io/updated_at=2023_10_09T09_53_35_0700,minikube.k8s.io
/version=v1.31.2,node-role.kubernetes.io/control-plane=,node.kubernetes.io/exclude-from-external-load-balancers=
ubuntu@ip-172-31-63-3:~$
```

Here we have added the label to the worker node
kubectl label nodes minikube hardware=t2-medium

```
ubuntu@ip-172-31-63-3:~$ kubectl label nodes minikube hardware=t2-medium
node/minikube labeled
ubuntu@ip-172-31-63-3:~$
```

As sooner the label is added, the sooner the pod will be in a running state.

```
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME         READY   STATUS    RESTARTS   AGE
nodelabels   1/1     Running   0          6m31s
ubuntu@ip-172-31-63-3:~$
```

ReplicationController & ReplicaSet

Before Kubernetes, other tools did not provide important and customized features like scaling and replication.

When Kubernetes was introduced, replication and scaling were the premium features that increased the popularity of this container orchestration tool.

Replication means that if the pod's desired state is set to 3 and whenever any pod fails, then with the help of replication, the new pod will be created as soon as possible. This will lead to areduction in the downtime of the application.

Scaling means if the load becomes increases on the application, then Kubernetes increases the number of pods according to the load on the application.

ReplicationController is an object in Kubernetes that was introduced in v1 of Kubernetes which helps to meet the desired state of the Kubernetes cluster from the current state. ReplicationController works on equality-based controllers only.

ReplicaSet is an object in Kubernetes and it is an advanced version of ReplicationController. ReplicaSet works on both equality-based controllers and set-based controllers.

Let’s do some hands-on to get a better understanding of ReplicationController & ReplicaSet.

YML file

```
apiVersion: v1
kind: ReplicationController
metadata:
 name: myreplica
spec:
 replicas: 2
 selector:
   Location: India
 template:
   metadata:
     name: testpod6
     labels:
       Location: India
   spec:
    containers:
      -name: c00
        image: ubuntu
        command: ["/bin/bash", "-c", "while true; do echo ReplicationController Example; sleep
5; done"]
```

Create the replication controller by running the command

kubectl apply -f myrc.yml

```
ubuntu@ip-172-31-63-3:~$ kubectl apply -f myrc.yml
replicationcontroller/myreplica created
ubuntu@ip-172-31-63-3:~$
```

Now, you can see the replication controller that we created earlier and observe the desired state, current state, ready, and age.

```
ubuntu@ip-172-31-63-3:~$ kubectl get rc
NAME        DESIRED   CURRENT   READY   AGE
myreplica   2         2         2       11s
ubuntu@ip-172-31-63-3:~$
```

If you list all the pods, you will see that my replica created two pods that are running.

```
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME              READY   STATUS    RESTARTS   AGE
myreplica-bktvt   1/1     Running   0          54s
myreplica-cmqrh   1/1     Running   0          54s
```

If you try to delete the pods, you will see that the new pod will be created quickly. You can observe through the AGE of both pods.

```
ubuntu@ip-172-31-63-3:~$ kubectl delete pod myreplica-bktvt
pod "myreplica-bktvt" deleted
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME              READY   STATUS    RESTARTS   AGE
myreplica-cmqrh   1/1     Running   0          2m3s
myreplica-xcbxn   1/1     Running   0          37s
ubuntu@ip-172-31-63-3:~$
```

If you want to modify the replicas, you can do that by running the command

kubectl scale --replicas=5 rc -l Location=India

```
ubuntu@ip-172-31-63-3:~$ kubectl get rc
NAME        DESIRED   CURRENT   READY   AGE
myreplica   2         2         2       3m16s
ubuntu@ip-172-31-63-3:~$ kubectl scale --replicas=5 rc -l Location=India
replicationcontroller/myreplica scaled
ubuntu@ip-172-31-63-3:~$ kubectl get rc
NAME        DESIRED   CURRENT   READY   AGE
myreplica   5         5         5       3m32s
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME              READY   STATUS    RESTARTS   AGE
myreplica-bx8dr   1/1     Running   0          15s
myreplica-cmqrh   1/1     Running   0          3m42s
myreplica-tnpsp   1/1     Running   0          15s
myreplica-wrz6p   1/1     Running   0          15s
myreplica-xcbxn   1/1     Running   0          2m16s
ubuntu@ip-172-31-63-3:~$
```

If you try to delete pods, you will see again that the new pod is creating quickly.

```
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME              READY   STATUS    RESTARTS   AGE
myreplica-bx8dr   1/1     Running   0          45s
myreplica-cmqrh   1/1     Running   0          4m12s
myreplica-tnpsp   1/1     Running   0          45s
myreplica-wrz6p   1/1     Running   0          45s
myreplica-xcbxn   1/1     Running   0          2m46s
ubuntu@ip-172-31-63-3:~$ kubectl delete pod myreplica-cmqrh myreplica-xcbxn
pod "myreplica-cmqrh" deleted
pod "myreplica-xcbxn" deleted
myreplica-xcbxnubuntu@ip-172-31-63-3:~$
ubuntu@ip-172-31-63-3:~$
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME              READY   STATUS    RESTARTS   AGE
myreplica-bx8dr   1/1     Running   0          106s
myreplica-q9rgd   1/1     Running   0          46s
myreplica-qwgzv   1/1     Running   0          46s
myreplica-tnpsp   1/1     Running   0          106s
myreplica-wrz6p   1/1     Running   0          106s
ubuntu@ip-172-31-63-3:~$
```

Now, if you want to delete all the pods. You can do it by just deleting the replicationController by the given command.

```
kubectl delete -f myrc.yml
```

```
ubuntu@ip-172-31-63-3:~$ kubectl delete -f myrc.yml
replicationcontroller "myreplica" deleted
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME              READY   STATUS        RESTARTS   AGE
myreplica-bx8dr   1/1     Terminating   0          3m56s
myreplica-q9rgd   1/1     Terminating   0          2m56s
myreplica-qwgzv   1/1     Terminating   0          2m56s
myreplica-tnpsp   1/1     Terminating   0          3m56s
myreplica-wrz6p   1/1     Terminating   0          3m56s
ubuntu@ip-172-31-63-3:~$ kubectl get rc
No resources found in default namespace.
ubuntu@ip-172-31-63-3:~$
```

ReplicaSet HandsOn

YML file

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: myrs
spec:
 replicas: 2
 selector:
   matchExpressions:
    -{key: Location, operator: In, values: [India, US, Russia]}
    -{key: env, operator: NotIn, values: [testing]}
 template:
   metadata:
     name: testpod7
     labels:
       Location: Russia
   spec:
    containers:
     -name: container1
       image: ubuntu
       command: ["/bin/bash", "-c", "while true; do echo ReplicaSet Example; sleep 5 ; done"]
```

Create the replicaSet by running the command

```
kubectl apply -f myrs.yml
```

```
ubuntu@ip-172-31-63-3:~$ kubectl apply -f myrs.yml
replicaset.apps/myrs created
ubuntu@ip-172-31-63-3:~$ kubectl get rs
NAME   DESIRED   CURRENT   READY   AGE
myrs   2         2         2       7s
```

Now, you can see the pods that have been created through replicaSet with labels

```
ubuntu@ip-172-31-63-3:~$ kubectl get pods --show-labels
NAME         READY   STATUS    RESTARTS   AGE   LABELS
myrs-6rxbp   1/1     Running   0          20s   Location=Russia
myrs-hmdzm   1/1     Running   0          20s   Location=Russia
ubuntu@ip-172-31-63-3:~$
```

If you try to delete the pods, you will see that the new pod is created with the same configuration.

```
ubuntu@ip-172-31-63-3:~$ kubectl delete pod myrs-6rxbp
pod "myrs-6rxbp" deleted
```

```
ubuntu@ip-172-31-63-3:~$ kubectl get pods --show-labels
NAME         READY   STATUS    RESTARTS   AGE     LABELS
myrs-hmdzm   1/1     Running   0          2m30s   Location=Russia
myrs-xj4v9   1/1     Running   0          61s     Location=Russia
ubuntu@ip-172-31-63-3:~$
```

If you want to increase or decrease the number of replicas, you can do it by the given command.

```
kubectl scale --replicas=5 rs myrs
```

```
ubuntu@ip-172-31-63-3:~$ kubectl scale --replicas=5 rs myrs
replicaset.apps/myrs scaled
ubuntu@ip-172-31-63-3:~$ kubectl get rs
NAME   DESIRED   CURRENT   READY   AGE
myrs   5         5         5       3m41s
ubuntu@ip-172-31-63-3:~$
```

If you want to delete all the pods, you can do it by deleting the replicaSet with the help of the given command.

```
kubectl delete -f myrs.yml
```

```
ubuntu@ip-172-31-63-3:~$ kubectl delete -f myrs.yml
replicaset.apps "myrs" deleted
ubuntu@ip-172-31-63-3:~$ kubectl get rs
No resources found in default namespace.
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME         READY   STATUS        RESTARTS   AGE
myrs-25znk   1/1     Terminating   0          56s
myrs-c48tv   1/1     Terminating   0          56s
myrs-hmdzm   1/1     Terminating   0          4m31s
myrs-j2w5r   1/1     Terminating   0          56s
myrs-xj4v9   1/1     Terminating   0          3m2s
ubuntu@ip-172-31-63-3:~$
```

Deployment Object in Kubernetes

- Replication Controller and Replica Set don't provide the updates and rollback for the
- application in the cabinets cluster. But in a deployment, the object does.
  The deployment object works as a supervisor for the pods which gives granular control over the pods. Deployment objects can decide how and when the pods should be deployed, rollback or updated.
- In Deployment, we define the desired state and the deployment controller will help to
- achieve the desired state from the current state.
- You can achieve this by declarative(manifest) method only.
- ADeployment provides declarative updates for Pods and ReplicaSets.
  The deployment object supports updates which means if there is any update in the applications that needs to be deployed in the new version then, the deployment object helps to achieve it.
- Deployment object supports rollback which means if the app is crashing in a new
- update then you can easily switch to the previous version by rollback.
- The deployment object doesn't work directly with the pods. Under the deployment object, there will be a replica set or replica controller that manages the pods and helps to manage the desired state.

Use Cases for the Deployment Object

- With the help of deployment, the replica set will be rolled out, which will deploy the pods and check the status in the background whether the rollout has succeeded or not.
- If the pod template spec is modified then, the new replica set will be created with the new desired state and the old replica set will still exist and you can roll back according to the situation.
- You can roll back to the previous deployment if the current state of the deployment is
- not stable.
- You can scale up the deployment to manage the loads.
- You can pause the deployment if you are fixing something in between the deployment and then resume it after fixing it.
  You can clean up those replica sets which are older and no longer needed.

HandsOn

YML file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: thedeployment
spec:
  replicas: 3
  selector:
```

```
  matchLabels:
   name: deploy-pods
 template:
  metadata:
    name: ubuntu-pods
    labels:
      name: deploy-pods
  spec:
   containers:
     -name: container1
       image: ubuntu
       command: ["/bin/bash", "-c", "while true; do echo This is Day09 of
30DaysOfKubernetes; sleep 5; done"]
```

Creating the deployment by running the command ***'kubectl apply -f da-deployment.yml'*** .

If you observe that the replica set has 3 desired states and same as the current state.

Also, all three pods are in ready and running status.

```
ubuntu@ip-172-31-63-3:~$ kubectl apply -f da-deployment.yml
deployment.apps/thedeployment created
ubuntu@ip-172-31-63-3:~$ kubectl get deploy
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
thedeployment   3/3     3            3           7s
ubuntu@ip-172-31-63-3:~$ kubectl get rs
NAME                       DESIRED   CURRENT   READY   AGE
thedeployment-5d7fcd7899   3         3         3       14s
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME                             READY   STATUS    RESTARTS   AGE
thedeployment-5d7fcd7899-lcf6s   1/1     Running   0          20s
thedeployment-5d7fcd7899-pdhdl   1/1     Running   0          20s
thedeployment-5d7fcd7899-vgsw6   1/1     Running   0          20s
ubuntu@ip-172-31-63-3:~$
```

Now, if you try to delete any pod, because of a replica set, the new pod will be created quickly.

```
ubuntu@ip-172-31-63-3:~$ kubectl delete pod thedeployment-5d7fcd7899-pdhdl
pod "thedeployment-5d7fcd7899-pdhdl" deleted
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME                             READY   STATUS    RESTARTS   AGE
thedeployment-5d7fcd7899-lcf6s   1/1     Running   0          2m4s
thedeployment-5d7fcd7899-t2jl7   1/1     Running   0          34s
thedeployment-5d7fcd7899-vgsw6   1/1     Running   0          2m4s
ubuntu@ip-172-31-63-3:~$
```

Here, we have increased the number of replicas set from 3 to 5 by the command

'kubectl scale — replicas=5 deployment <deployment_name>'.

```
ubuntu@ip-172-31-63-3:~$ kubectl get rs
NAME                        DESIRED   CURRENT   READY   AGE
thedeployment-5d7fcd7899    3         3         3       3m52s
ubuntu@ip-172-31-63-3:~$ kubectl scale --replicas=5 deployment thedeployment
deployment.apps/thedeployment scaled
ubuntu@ip-172-31-63-3:~$ kubectl get rs
NAME                        DESIRED   CURRENT   READY   AGE
thedeployment-5d7fcd7899    5         5         5       4m11s
ubuntu@ip-172-31-63-3:~$
```

Here, we are checking the logs for the particular pod.

```
ubuntu@ip-172-31-63-3:~$ kubectl logs -f thedeployment-5d7fcd7899-t2jl7
This is Day09 of 30DaysOfKubernetes
This is Day09 of 30DaysOfKubernetes
This is Day09 of 30DaysOfKubernetes
This is Day09 of 30DaysOfKubernetes
This is Day09 of 30DaysOfKubernetes
This is Day09 of 30DaysOfKubernetes
This is Day09 of 30DaysOfKubernetes
This is Day09 of 30DaysOfKubernetes
```

Here, I made some changes in the previous YML file as you can see in the file. I have updated the image and command for the needed YML file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: thedeployment
spec:
  replicas: 3
  selector:
   matchLabels:
    name: deploy-pods
  template:
    metadata:
      name: ubuntu-pods
      labels:
        name: deploy-pods
    spec:
     containers:
       -name: container1
        image: centos
        command: ["/bin/bash", "-c", "while true; do echo DevOps is a Culture; sleep 5; done"]
```

After updating the file, I have applied the updated file and as you can see, ‘thedeployment’ has 3 replicas which were previously 5. This happened because, in the YML file the replicas are 3.

Also, if you observe that the previous rs is still present with 0 desired and current state.

```
ubuntu@ip-172-31-63-3:~$ kubectl apply -f da-deployment.yml
deployment.apps/thedeployment configured
ubuntu@ip-172-31-63-3:~$ kubectl get deploy
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
thedeployment   3/3     1            3           7m45s
ubuntu@ip-172-31-63-3:~$ kubectl get rs
NAME                      DESIRED   CURRENT   READY   AGE
thedeployment-5d7fcd7899   0         0         0       7m54s
thedeployment-7bcc7b8d89   3         3         3       16s
ubuntu@ip-172-31-63-3:~$
ubuntu@ip-172-31-63-3:~$
ubuntu@ip-172-31-63-3:~$ kubectl get pod
NAME                            READY   STATUS        RESTARTS   AGE
thedeployment-5d7fcd7899-lcf6s   1/1     Terminating   0          8m1s
thedeployment-5d7fcd7899-sj98t   1/1     Terminating   0          3m54s
thedeployment-5d7fcd7899-t2jl7   1/1     Terminating   0          6m31s
thedeployment-5d7fcd7899-v9nbj   1/1     Terminating   0          3m54s
thedeployment-5d7fcd7899-vgsw6   1/1     Terminating   0          8m1s
thedeployment-7bcc7b8d89-5vn29   1/1     Running       0          23s
thedeployment-7bcc7b8d89-j2nsf   1/1     Running       0          12s
thedeployment-7bcc7b8d89-ph7rc   1/1     Running       0          14s
ubuntu@ip-172-31-63-3:~$
```

Now, if you will check the logs of the new pods. You will see the updated command running that we have written in the YML file.

```
ubuntu@ip-172-31-63-3:~$ kubectl logs -f thedeployment-7bcc7b8d89-5vn29
DevOps is a Culture
DevOps is a Culture
DevOps is a Culture
DevOps is a Culture
DevOps is a Culture
DevOps is a Culture
DevOps is a Culture
DevOps is a Culture
DevOps is a Culture
DevOps is a Culture
```

Also, We have updated the image for the OS and you can see that we are getting the expected result for the image.

```
ubuntu@ip-172-31-63-3:~$ kubectl exec thedeployment-7bcc7b8d89-5vn29 -- cat /etc/os-release
NAME="CentOS Linux"
VERSION="8"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="8"
PLATFORM_ID="platform:el8"
PRETTY_NAME="CentOS Linux 8"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:8"
HOME_URL="https://centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"
CENTOS_MANTISBT_PROJECT="CentOS-8"
CENTOS_MANTISBT_PROJECT_VERSION="8"
ubuntu@ip-172-31-63-3:~$
```

Here, we have increased the number of replicas set from 3 to 5 by the command ‘kubectl scale — replicas=5 deployment <deployment_name> ’.

```
ubuntu@ip-172-31-63-3:~$ kubectl scale --replicas=5 deployment thedeployment
deployment.apps/thedeployment scaled
ubuntu@ip-172-31-63-3:~$ kubectl get deploy
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
thedeployment   5/5     5            5           12m
ubuntu@ip-172-31-63-3:~$
ubuntu@ip-172-31-63-3:~$ kubectl get rs
NAME                      DESIRED   CURRENT   READY   AGE
thedeployment-5d7fcd7899   0         0         0       12m
thedeployment-7bcc7b8d89   5         5         5       4m34s
ubuntu@ip-172-31-63-3:~$
ubuntu@ip-172-31-63-3:~$ kubectl pods
error: unknown command "pods" for "kubectl"

Did you mean this?
        logs
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
thedeployment-7bcc7b8d89-5vn29   1/1     Running   0          4m45s
thedeployment-7bcc7b8d89-767j8   1/1     Running   0          29s
thedeployment-7bcc7b8d89-j2nsf   1/1     Running   0          4m34s
thedeployment-7bcc7b8d89-ph7rc   1/1     Running   0          4m36s
thedeployment-7bcc7b8d89-qzpqw   1/1     Running   0          29s
ubuntu@ip-172-31-63-3:~$
```

Now, if you want to switch to the previous deployment you can do it by the given command.

kubectl rollout undo deployment <deployment_name>

If you compare the first kubectl get rs command with the second kubectl get rs, then the desired state shifts to the other deployment.

```
ubuntu@ip-172-31-63-3:~$ kubectl get rs
NAME                      DESIRED   CURRENT   READY   AGE
thedeployment-5d7fcd7899   0         0         0       20m
thedeployment-7bcc7b8d89   5         5         5       12m
ubuntu@ip-172-31-63-3:~$ kubectl rollout undo deployment thedeployment
deployment.apps/thedeployment rolled back
ubuntu@ip-172-31-63-3:~$ kubectl get rs
NAME                      DESIRED   CURRENT   READY   AGE
thedeployment-5d7fcd7899   5         5         3       20m
thedeployment-7bcc7b8d89   1         1         1       13m
ubuntu@ip-172-31-63-3:~$ kubectl get rs
NAME                      DESIRED   CURRENT   READY   AGE
thedeployment-5d7fcd7899   5         5         5       21m
thedeployment-7bcc7b8d89   0         0         0       13m
ubuntu@ip-172-31-63-3:~$
```

Now, if you see the logs of the running pods. You will see that the previous command is running because we have switched to the previous deployment.

```
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
thedeployment-5d7fcd7899-99kz2      1/1     Running   0          61s
thedeployment-5d7fcd7899-gncm9      1/1     Running   0          61s
thedeployment-5d7fcd7899-grhvt      1/1     Running   0          59s
thedeployment-5d7fcd7899-hld4n      1/1     Running   0          61s
thedeployment-5d7fcd7899-rpk86      1/1     Running   0          59s
ubuntu@ip-172-31-63-3:~$ kubectl logs -f thedeployment-5d7fcd7899-grhvt
This is Day09 of 30DaysOfKubernetes
This is Day09 of 30DaysOfKubernetes
This is Day09 of 30DaysOfKubernetes
This is Day09 of 30DaysOfKubernetes
This is Day09 of 30DaysOfKubernetes
This is Day09 of 30DaysOfKubernetes
This is Day09 of 30DaysOfKubernetes
This is Day09 of 30DaysOfKubernetes
This is Day09 of 30DaysOfKubernetes
```

If you see the OS Image, you will understand that in our previous deployment, we used an Ubuntu image which is expected.

```
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
thedeployment-5d7fcd7899-99kz2      1/1     Running   0          2m27s
thedeployment-5d7fcd7899-gncm9      1/1     Running   0          2m27s
thedeployment-5d7fcd7899-grhvt      1/1     Running   0          2m25s
thedeployment-5d7fcd7899-hld4n      1/1     Running   0          2m27s
thedeployment-5d7fcd7899-rpk86      1/1     Running   0          2m25s
ubuntu@ip-172-31-63-3:~$
ubuntu@ip-172-31-63-3:~$ kubectl exec thedeployment-5d7fcd7899-grhvt -- cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.3 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.3 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
ubuntu@ip-172-31-63-3:~$
```

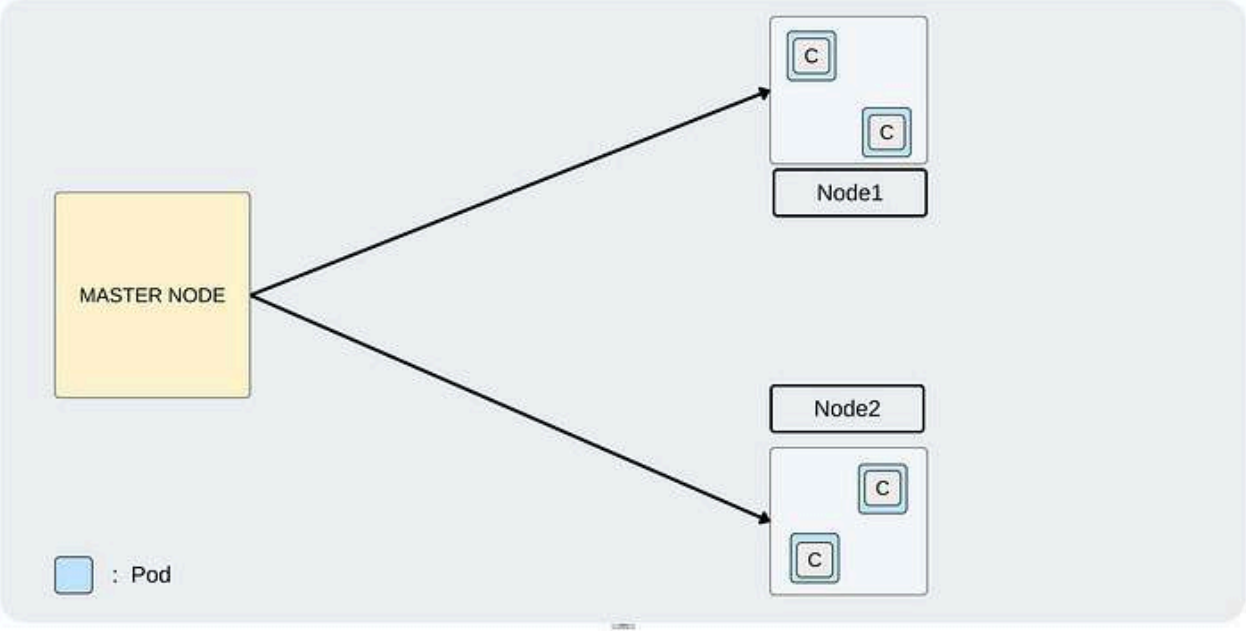
If you want to delete all those running pods and replica sets, then use the given command.

```
kubectl delete -f <deployment_file>
```


ubuntu@ip-172-31-63-3:~$ kubectl delete -f da-deployment.yml
deployment.apps "thedeployment" deleted
ubuntu@ip-172-31-63-3:~$ kubectl get deploy
No resources found in default namespace.
ubuntu@ip-172-31-63-3:~$ kubectl get rs
No resources found in default namespace.
ubuntu@ip-172-31-63-3:~$ kubectl get pods
NAME READY STATUS RESTARTS AGE
thedeployment-5d7fcd7899-99kz2 1/1 Terminating 0 5m40s
thedeployment-5d7fcd7899-gncm9 1/1 Terminating 0 5m40s
thedeployment-5d7fcd7899-grhvt 1/1 Terminating 0 5m38s
thedeployment-5d7fcd7899-hld4n 1/1 Terminating 0 5m40s
thedeployment-5d7fcd7899-rpk86 1/1 Terminating 0 5m38s
ubuntu@ip-172-31-63-3:~$

Kubernetes Cluster(Master+Worker Node) using kubeadm on AWS

Create three EC2 instances with Ubuntu 22.04 and the necessary security group settings.
Configure the instances to prepare for Kubernetes installation.
Install Docker and Kubernetes components on all nodes.
Initialize the Kubernetes cluster on the master node.
Join the worker nodes in the cluster.
Deploy an Nginx application on the cluster for validation.



Step 1:

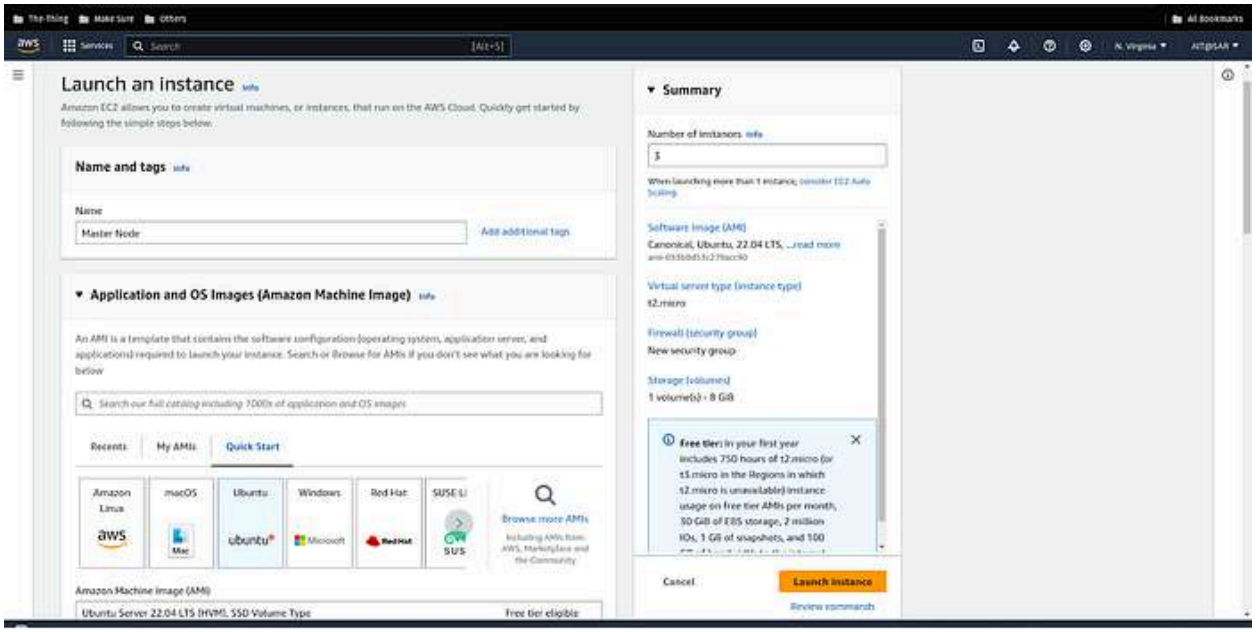
Create three EC2 Instances with the given configuration

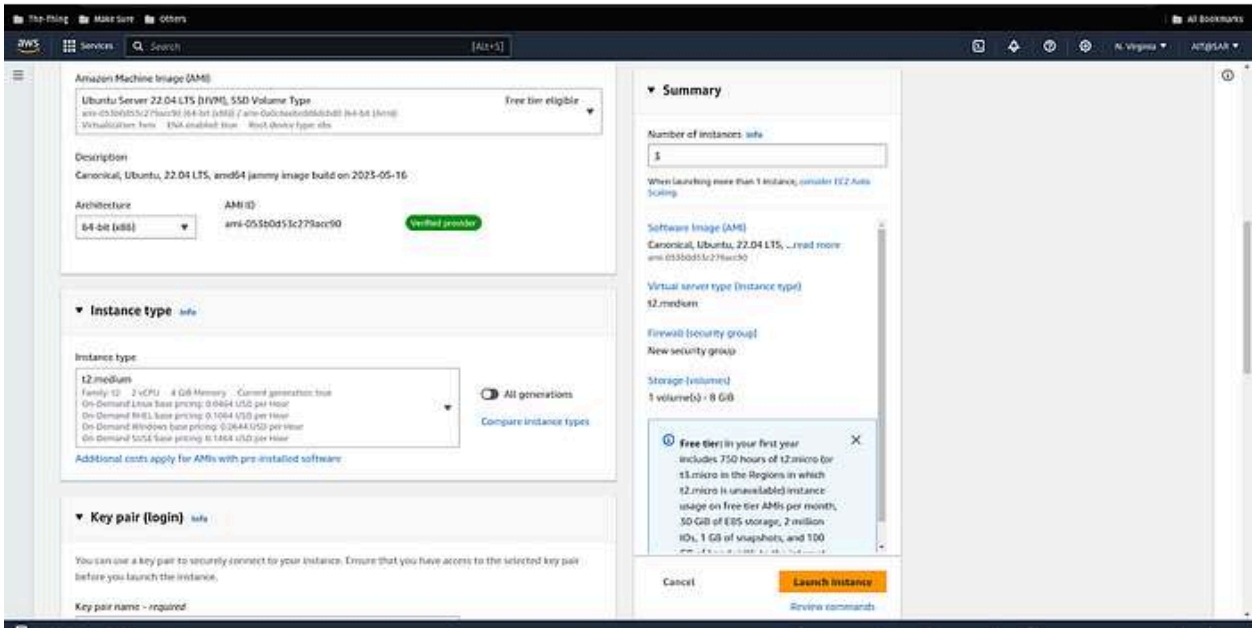
Instance type- t2.medium

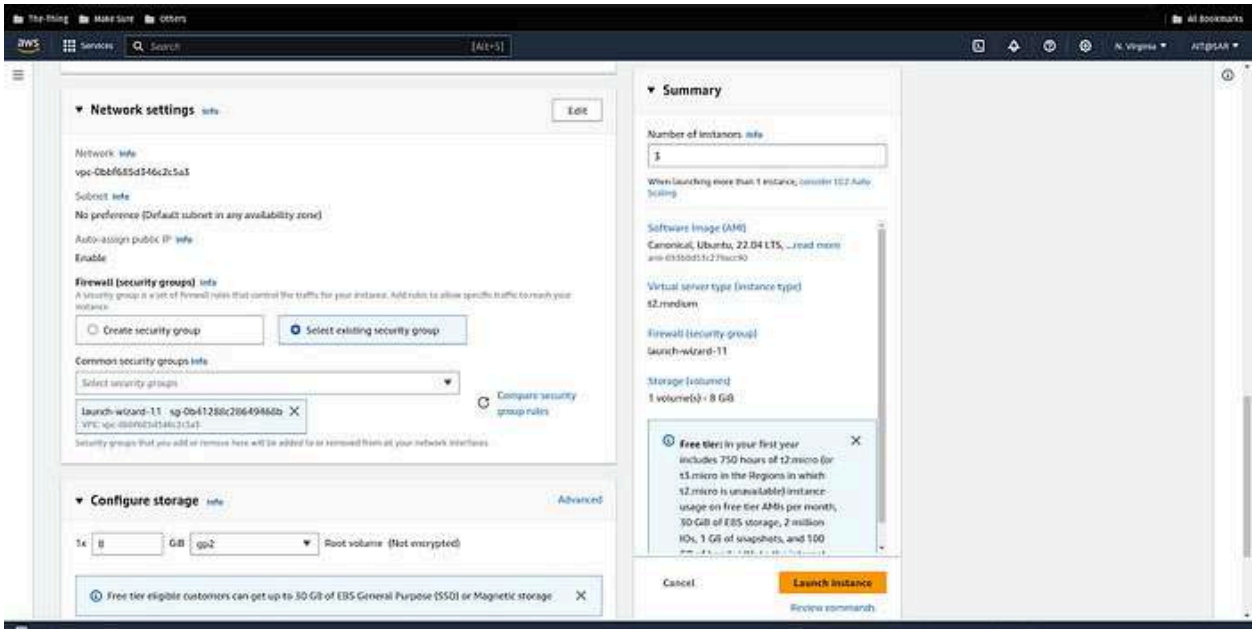
Ubuntu Version- 22.04

Create the keypair so you can connect to the instance using SSH.

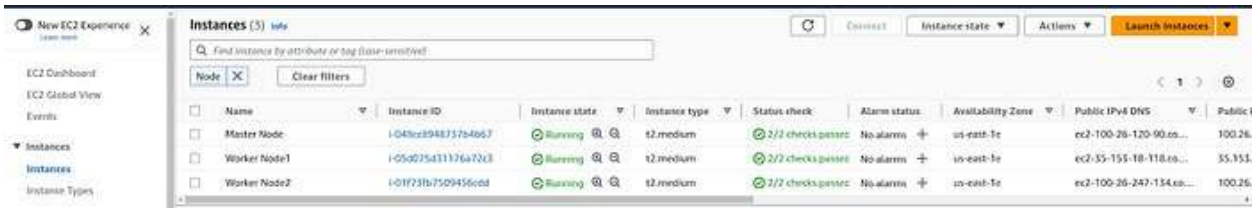
Create the new Security group and once the instances are initialized/created then, make sure to add the Allow All traffic in inbound rule in the attached security group.


Launch an instance
Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.
Name and tags
Name
Master Node
Add additional tags
Application and OS Images (Amazon Machine Image)
An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below
Search our full catalog including 1000s of application and OS images
Recents
My AMIs
Quick Start
Amazon Linux
macOS
Ubuntu
Windows
Red Hat
SUSE Li
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community
Amazon Machine Image (AMI)
Ubuntu Server 22.04 LTS (HVM), SSD Volume Type
Free tier eligible
Summary
Number of instances
1
When launching more than 1 instance, consider EC2 Auto Scaling
Software Image (AMI)
Canonical, Ubuntu, 22.04 LTS, ...read more
Virtual server type (instance type)
t2.micro
Firewall (security group)
New security group
Storage (volumes)
1 volume(s) - 8 GiB
Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million IOs, 1 GB of snapshots, and 100
Cancel
Launch instance
Review commands


Amazon Machine Image (AMI)
Ubuntu Server 22.04 LTS (HVM), SSD Volume Type
Free tier eligible
Description
Canonical, Ubuntu, 22.04 LTS, amd64 jammy image build on 2023-05-16
Architecture
64-bit (x86)
AMI ID
ami-053b0d53c279acc90
Verified provider
Instance type
Instance type
t2.medium
All generations
Compare instance types
Additional costs apply for AMIs with pre-installed software
Key pair (login)
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.
Key pair name - required
Summary
Number of instances
1
When launching more than 1 instance, consider EC2 Auto Scaling
Software Image (AMI)
Canonical, Ubuntu, 22.04 LTS, ...read more
Virtual server type (instance type)
t2.medium
Firewall (security group)
New security group
Storage (volumes)
1 volume(s) - 8 GiB
Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million IOs, 1 GB of snapshots, and 100
Cancel
Launch instance
Review commands



Rename the instances according to you. Currently, I am setting up the One Master and two Workers nodes.



After creating the instances, we have to configure the all instances. Let's do that and follow the steps carefully.

Commands need to run on all Nodes(Master and Worker)

Once we log in the all three instances, run the following command.

Step 2:

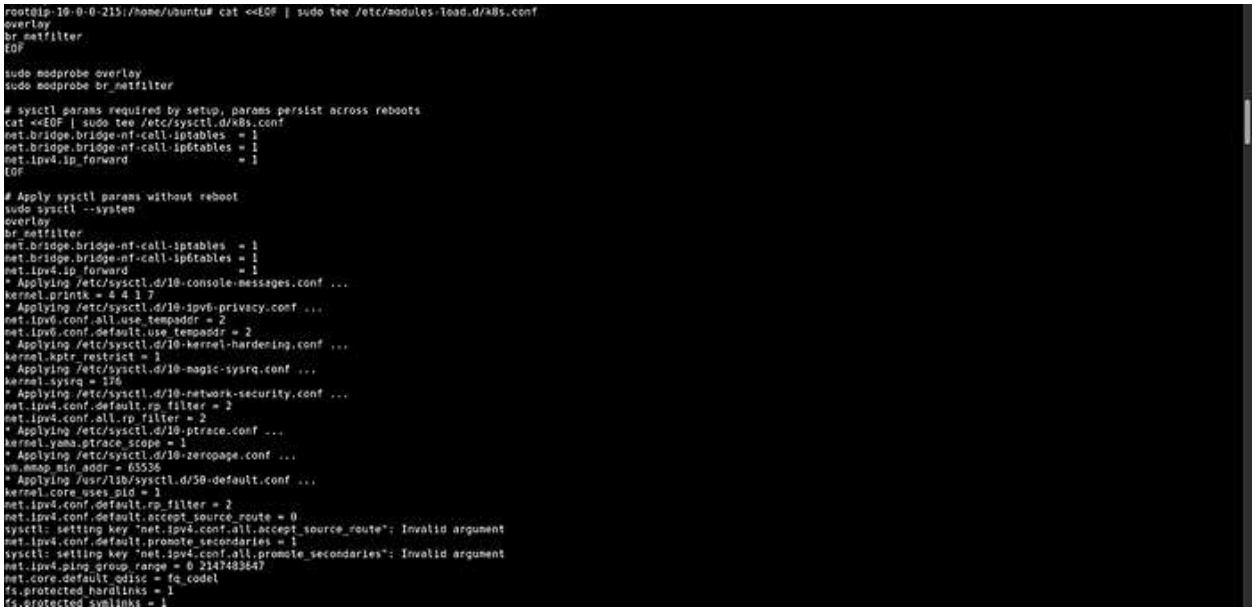
```
sudo su
swapoff -a; sed -i '/swap/d' /etc/fstab
```

```
root@ip-10-0-0-215:/home/ubuntu# sudo swapoff -a
root@ip-10-0-0-215:/home/ubuntu# sudo sed -i '/ swap / s/^/#/' /etc/fstab
```

Step 3:

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

```
sudo modprobe overlay
sudo modprobe br_netfilter
#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
#Apply sysctl params without reboot
sudo sysctl --system
```



Step 4:

```
apt update
```

```
root@ip-10-0-0-215:/home/ubuntu# apt update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [14.1 MB]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe Translation-en [5652 kB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 c-n-f Metadata [286 kB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [217 kB]
Get:9 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse Translation-en [112 kB]
Get:10 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 c-n-f Metadata [8372 B]
Get:11 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1062 kB]
Get:12 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [234 kB]
Get:13 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 c-n-f Metadata [16.0 kB]
Get:14 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [974 kB]
Get:15 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted Translation-en [157 kB]
Get:16 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 c-n-f Metadata [532 B]
Get:17 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [991 kB]
Get:18 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe Translation-en [217 kB]
Get:19 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 c-n-f Metadata [22.0 kB]
Get:20 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 Packages [41.6 kB]
Get:21 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/multiverse Translation-en [9760 B]
Get:22 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 c-n-f Metadata [472 B]
Get:23 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/main amd64 Packages [41.7 kB]
Get:24 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/main Translation-en [10.5 kB]
Get:25 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/main amd64 c-n-f Metadata [388 B]
Get:26 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/restricted amd64 c-n-f Metadata [116 B]
Get:27 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/universe amd64 Packages [24.3 kB]
Get:28 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/universe Translation-en [16.4 kB]
Get:29 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/universe amd64 c-n-f Metadata [640 B]
Get:30 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/multiverse amd64 c-n-f Metadata [116 B]
Get:31 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [856 kB]
Get:32 http://security.ubuntu.com/ubuntu jammy-security/main Translation-en [175 kB]
Get:33 http://security.ubuntu.com/ubuntu jammy-security/main amd64 c-n-f Metadata [11.4 kB]
Get:34 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [953 kB]
Get:35 http://security.ubuntu.com/ubuntu jammy-security/restricted Translation-en [154 kB]
Get:36 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 c-n-f Metadata [532 B]
Get:37 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [789 kB]
Get:38 http://security.ubuntu.com/ubuntu jammy-security/universe Translation-en [145 kB]
Get:39 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 c-n-f Metadata [16.7 kB]
Get:40 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [36.5 kB]
Get:41 http://security.ubuntu.com/ubuntu jammy-security/multiverse Translation-en [7060 B]
Get:42 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 c-n-f Metadata [260 B]
Fetched 27.7 MB in 4s (6777 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
114 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

Step 5:

Install dependencies by running the command

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

```
root@ip-10-0-0-215:/home/ubuntu# sudo apt-get install -y apt-transport-https ca-certificates curl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libcurl4
The following NEW packages will be installed:
  apt-transport-https
The following packages will be upgraded:
  ca-certificates curl libcurl4
3 upgraded, 1 newly installed, 0 to remove and 131 not upgraded.
Need to get 641 kB of archives.
After this operation, 193 kB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 ca-certificates all 20230311ubuntu0.22.04.1 [155 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 apt-transport-https all 2.4.10 [1510 B]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 curl amd64 7.81.0-1ubuntu1.14 [194 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 libcurl4 amd64 7.81.0-1ubuntu1.14 [290 kB]
Fetched 641 kB in 0s (12.5 MB/s)
Preconfiguring packages ...
(Reading database ... 64295 files and directories currently installed.)
Preparing to unpack .../ca-certificates_20230311ubuntu0.22.04.1_all.deb ...
Unpacking ca-certificates (20230311ubuntu0.22.04.1) over (20211016ubuntu0.22.04.1) ...
Selecting previously unselected package apt-transport-https.
Preparing to unpack .../apt-transport-https_2.4.10_all.deb ...
Unpacking apt-transport-https (2.4.10) ...
Preparing to unpack .../curl_7.81.0-1ubuntu1.14_amd64.deb ...
Unpacking curl (7.81.0-1ubuntu1.14) over (7.81.0-1ubuntu1.10) ...
Preparing to unpack .../libcurl4_7.81.0-1ubuntu1.14_amd64.deb ...
Unpacking libcurl4:amd64 (7.81.0-1ubuntu1.14) over (7.81.0-1ubuntu1.10) ...
Setting up apt-transport-https (2.4.10) ...
Setting up ca-certificates (20230311ubuntu0.22.04.1) ...
Updating certificates in /etc/ssl/certs...
rehash: warning: skipping ca-certificates.crt,it does not contain exactly one certificate or CRL
19 added, 6 removed; done.
Setting up libcurl4:amd64 (7.81.0-1ubuntu1.14) ...
Setting up curl (7.81.0-1ubuntu1.14) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for libc-bin (2.35-0ubuntu3.1) ...
Processing triggers for ca-certificates (20230311ubuntu0.22.04.1) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.
```

Step 6:

Fetch the public key from Google to validate the Kubernetes packages once it will be installed.

```
/etc/apt/keyrings/kubernetes-archive-keyring.gpg
```

```
root@ip-10-0-0-215:/home/ubuntu# curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-archive-keyring.gpg
root@ip-10-0-0-215:/home/ubuntu#
root@ip-10-0-0-215:/home/ubuntu#
```

Step 7:

Add the Kubernetes package in the sources.list.d directory

```
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

```
root@ip-10-0-0-215:/home/ubuntu# echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/
kubernetes.list
deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main
```

Step 8:

Update the packages as we have added some keys and packages.

```
apt update
```

```
root@ip-10-0-0-215:/home/ubuntu# apt update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Get:5 https://packages.cloud.google.com/apt kubernetes-xenial InRelease [8993 B]
Get:6 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 Packages [69.9 kB]
Fetched 78.9 kB in 1s (144 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
131 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

Step 9:

Install kubelet, kubeadm, kubectl and kubernets-cni

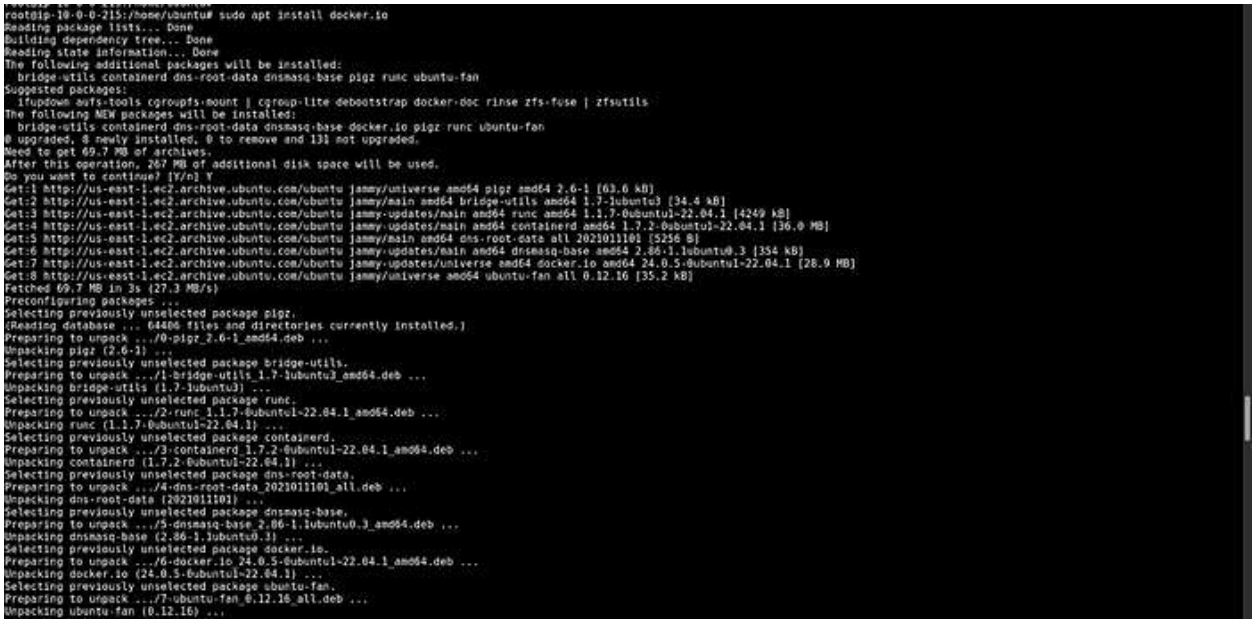
```
apt-get install -y kubelet kubeadm kubectl kubernetes-cni
```

```
root@ip-10-0-0-215:/home/ubuntu# apt-get install -y kubelet kubeadm kubectl kubernetes-cni
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  conntrack cri-tools ebtables socat
The following NEW packages will be installed:
  conntrack cri-tools ebtables kubeadm kubectl kubelet kubernetes-cni socat
0 upgraded, 8 newly installed, 0 to remove and 131 not upgraded.
Need to get 87.1 MB of archives.
After this operation, 336 MB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 conntrack amd64 1:1.4.6-2build2 [33.5 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 ebtables amd64 2.0.11-4build2 [84.9 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 socat amd64 1.7.4.1-3ubuntu4 [349 kB]
Get:4 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 cri-tools amd64 1.26.0-00 [18.9 MB]
Get:5 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 kubernetes-cni amd64 1.2.0-00 [27.6 MB]
Get:6 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 kubelet amd64 1.28.2-00 [19.5 MB]
Get:7 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 kubectl amd64 1.28.2-00 [10.3 MB]
Get:8 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 kubeadm amd64 1.28.2-00 [10.3 MB]
Fetched 87.1 MB in 2s (56.6 MB/s)
Selecting previously unselected package conntrack.
(Reading database ... 64312 files and directories currently installed.)
Preparing to unpack .../0-conntrack_1%3a1.4.6-2build2_amd64.deb ...
Unpacking conntrack (1:1.4.6-2build2) ...
Selecting previously unselected package cri-tools.
Preparing to unpack .../1-cri-tools_1.26.0-00_amd64.deb ...
Unpacking cri-tools (1.26.0-00) ...
Selecting previously unselected package ebtables.
Preparing to unpack .../2-ebtables_2.0.11-4build2_amd64.deb ...
Unpacking ebtables (2.0.11-4build2) ...
Selecting previously unselected package kubernetes-cni.
Preparing to unpack .../3-kubernetes-cni_1.2.0-00_amd64.deb ...
Unpacking kubernetes-cni (1.2.0-00) ...
Selecting previously unselected package socat.
Preparing to unpack .../4-socat_1.7.4.1-3ubuntu4_amd64.deb ...
Unpacking socat (1.7.4.1-3ubuntu4) ...
Selecting previously unselected package kubelet.
Preparing to unpack .../5-kubelet_1.28.2-00_amd64.deb ...
Unpacking kubelet (1.28.2-00) ...
Selecting previously unselected package kubectl.
Preparing to unpack .../6-kubectl_1.28.2-00_amd64.deb ...
Unpacking kubectl (1.28.2-00) ...
Selecting previously unselected package kubeadm.
Preparing to unpack .../7-kubeadm_1.28.2-00_amd64.deb ...
Unpacking kubeadm (1.28.2-00) ...
Setting up conntrack (1:1.4.6-2build2) ...
Setting up kubectl (1.28.2-00) ...
Setting up ebtables (2.0.11-4build2) ...
Setting up socat (1.7.4.1-3ubuntu4) ...
```

Step 10:

This is one of the important dependencies to setting up the Master and Worker nodes. Installing docker.

```
apt install docker.io -y
```



Step 11:

Configuring containerd to ensure compatibility with Kubernetes

```
sudo mkdir /etc/containerd
sudo sh -c "containerd config default > /etc/containerd/config.toml"
sudo sed -i 's/ SystemdCgroup = false/ SystemdCgroup = true/' /etc/containerd/config.toml
```

Step 12:

Restart containerd, kubelet, and enable kubelet so when we reboot our machine the nodes will restart it as well and connect properly.

```
systemctl restart containerd.service
systemctl restart kubelet.service
systemctl enable kubelet.service
```

Now, we have completed the installation of the things that are needed on both nodes (Master and Worker). But in the next steps, we have to configure things only on the Master Node.

Only on the Master Node

Step 13:

Initialize the Kubernetes cluster and it will pull some images such as kube-apiserver, kube-controller, and many other important components.

```
root@ip-10-0-0-15:/home/ubuntu# sudo kubeadm config images pull
[config/images] Pulled registry.k8s.io/kube-apiserver:v1.28.2
[config/images] Pulled registry.k8s.io/kube-controller-manager:v1.28.2
[config/images] Pulled registry.k8s.io/kube-scheduler:v1.28.2
[config/images] Pulled registry.k8s.io/kube-proxy:v1.28.2
[config/images] Pulled registry.k8s.io/pause:3.9
[config/images] Pulled registry.k8s.io/etcd:3.5.9-0
[config/images] Pulled registry.k8s.io/coredns/coredns:v1.10.1
```

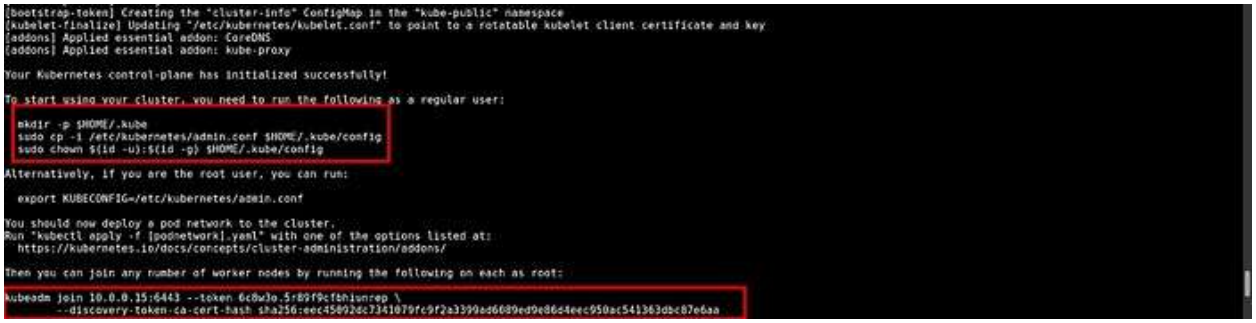
Step 14:

Now, initialize the Kubernetes cluster which will give you the token or command to connect with this Master node from the Worker node. At the end of this command, you will get some commands that need to run and at the bottom, you will get the kubeadm join command that will be run from the Worker Node to connect with the Master Node. I have highlighted the commands in the second next snipped. Please keep the kubeadm join command somewhere in the notepad.

```
kubeadm init
```

```
root@ip-10-0-0-15:/home/ubuntu# kubeadm init
[init] Using Kubernetes version: v1.28.2
[preflight] Running pre-flight checks
[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'
W1012 11:45:04.670039    9276 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 "registry.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 [ip-10-0-0-15 kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 10.0.0.15]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [ip-10-0-0-15 localhost] and IPs [10.0.0.15 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [ip-10-0-0-15 localhost] and IPs [10.0.0.15 127.0.0.1 ::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 7.504239 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 ip-10-0-0-15 as control-plane by adding the labels: [node-role.kubernetes.io/control-plane node.kubernetes.io/exclude-from-external-load-balancers]
[mark-control-plane] Marking the node ip-10-0-0-15 as control-plane by adding the taints [node-role.kubernetes.io/control-plane:NoSchedule]
[bootstrap-token] Using token: 6c8w3o.5r89f9ctph1unrep
[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
```

Keep the kubeadm join command in your notepad or somewhere for later.



Step 15:

As you have to manage the cluster that's why you need to create a .kube file copy it to the given directory and change the ownership of the file.

```
sudo cp -i /etc/kuberne
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Step 16:

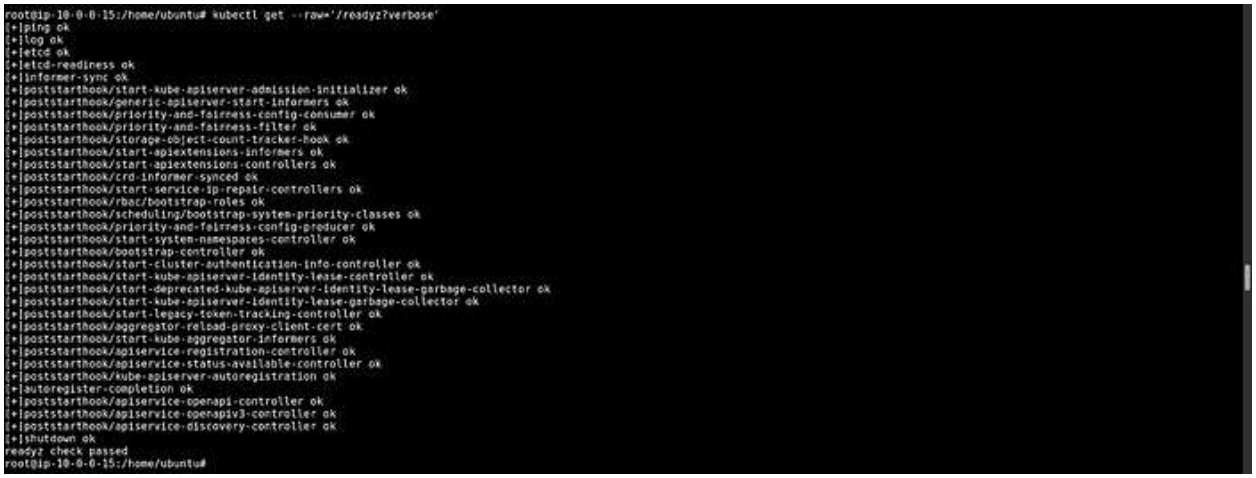
Verify the kubeconfig by kube-system which will list all the pods. If you observe, there are starting twopods that are not ready status because the network plugin is not installed.

```
kubectl get po -n kube-system
```

Step 17:

Verify all the cluster component health statuses

```
kubectl get --raw='/readyz?verbose'
```



Step 18:

Check the cluster-info

kubectl cluster-info

Step 19:

To install the network plugin on the Master node

kubectl apply -f
https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml

Step 20:

Now, If you run the below command, you will observe the two remaining pods are in ready status which means we are ready to bootstrap by our Workers Node or connect to the Master node through the Worker Node.

```
root@ip-10-0-0-15:/home/ubuntu# kubectl get po -n kube-system
NAME                                      READY   STATUS    RESTARTS   AGE
calico-kube-controllers-658d97c59c-pthvg  1/1     Running   0          34s
calico-node-vvkpf                         1/1     Running   0          34s
coredns-5dd5756b68-6ljnb                  1/1     Running   0          2m38s
coredns-5dd5756b68-9xkvz                  1/1     Running   0          2m38s
etcd-ip-10-0-0-15                         1/1     Running   0          2m43s
kube-apiserver-ip-10-0-0-15               1/1     Running   0          2m43s
kube-controller-manager-ip-10-0-0-15      1/1     Running   0          2m43s
kube-proxy-mm4p4                          1/1     Running   0          2m39s
kube-scheduler-ip-10-0-0-15               1/1     Running   0          2m43s
root@ip-10-0-0-15:/home/ubuntu#
```

Step 21:

Now, you have to run the command on Worker Node1. If you remember, I told you to keep the command somewhere in Step 14. Here you have to use your command because your token is different as well as mine.

Worker Node1

```
discovery-token-ca-cert-hash
sha256:eec45092dc7341079[redacted]
```

```
root@ip-10-0-0-215:/home/ubuntu# kubeadm join 10.0.0.15:6443 --token 6c8w3o.5r89f9cfbh1unrep --discovery-token-ca-cert-hash sha256:eec45092dc7341079fc9f2a3399ad6089ed9e86d4eec950ac541363dbc87e6aa
[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@ip-10-0-0-215:/home/ubuntu#
```

Step 22:

Follow the Step 21.

Worker Node2

```
sha256:eec45092dc7341079fc9f2a3399ad6089ed9e86d4eec9
```

```
root@ip-10-0-0-250:/home/ubuntu# kubeadm join 10.0.0.15:6443 --token 6c8w3o.5r89f9cfbh1unrep --discovery-token-ca-cert-hash sha256:eec45092dc7341079fc9f2a3399ad6089ed9e86d4eec950ac541363dbc87e6aa
[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@ip-10-0-0-250:/home/ubuntu#
```

Step 23:

Now, from here all the commands will be run on Master Node only.

If you run the below command you will see that the Worker Nodes are present with their respective Private IPs and it is in Ready. status

```
[redacted]
```

Here, we have completed our Setup of Master and Worker Node. Now let's try to deploy a simple nginx pod on both worker nodes.

Step 23:

Run the command, which includes the deployment file to deploy nginx on both worker Node.

```
kind: Deployment
metadata:


 name: nginx-deployment
spec:
 selector:
   matchLabels:
     app: nginx
 replicas: 2
 template:
   metadata:
     labels:
       app: nginx
   spec:
     containers:
     -name: nginx
       image: nginx:latest
       ports:
       -containerPort: 80
EOF
```

Step 24:

To expose your deployment on NodePort 32000 which means you can access your nginx application on port 32000 through your browser easily.

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
 name: nginx-service
```

```
spec:
 selector:
   app: nginx
 type: NodePort
 ports:
   -port: 80
     targetPort: 80
     nodePort: 32000
EOF
```

Step 25:

Now, check the pods by the below command and you can see that your pods are in running status.

```
kubectl get pods
```

Step 26:

On Worker Node1

You can validate your deployment by copying the Public IP of your WorkerNode1 and adding acolon(:) with port(32000).

On Worker Node2

You can validate your deployment by copying the Public IP of your WorkerNode2 and adding acolon(:) with port(32000).

Kubernetes Networking (Services)

Objectives

By the end of this topic, you will:
Understand the basics of how pods and containers can communicate within the same pod and node.
Explore the critical role of Service objects in Kubernetes networking.
Gain insights into different Service types, including ClusterIP, NodePort, LoadBalancer, and ExternalName.

Things to know about accessing pods or containers in some scenarios:

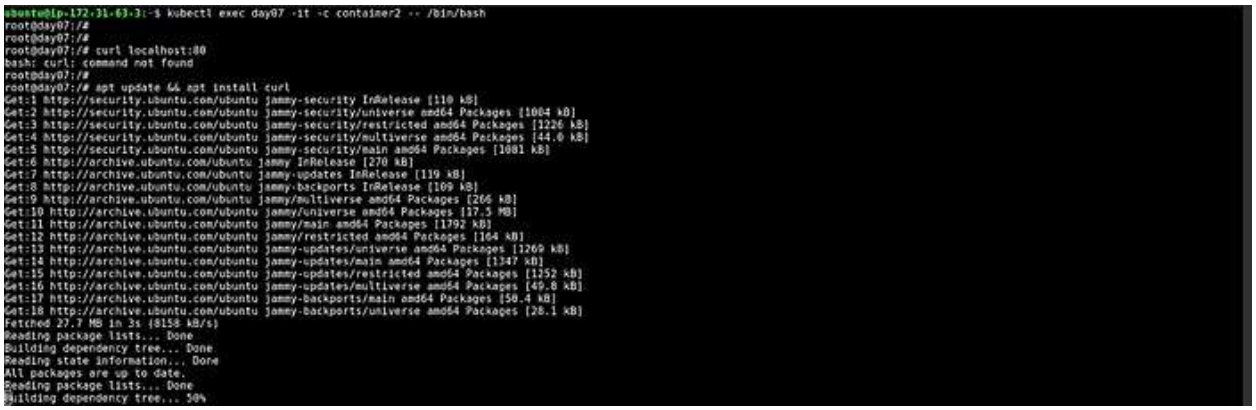
- Multiple containers within the pod access each other through a loopback
- address(localhost).
- The cluster provides communication between multiple pods.
- To access your application from outside of the cluster, you need a Services object in Kubernetes.
  You can also use the Services object to publish services only for access within the cluster.

Access containers within the same pod

```
apiVersion: v1
kind: Pod
metadata:
 name: day10
 labels:
   env: testing
   department: DevOps
spec:
 containers:
  -name: container1
    image: nginx
  -name: container2
    image: ubuntu
    command: ["/bin/bash", "-c", "while true; do echo This is Day07 of 30DaysOfKubernetes;
sleep 5 ; done"]
    ports:
      -containerPort: 80
```

kubectl exec day07 -it -c container2 — /bin/bash

Update the packages and install the curl



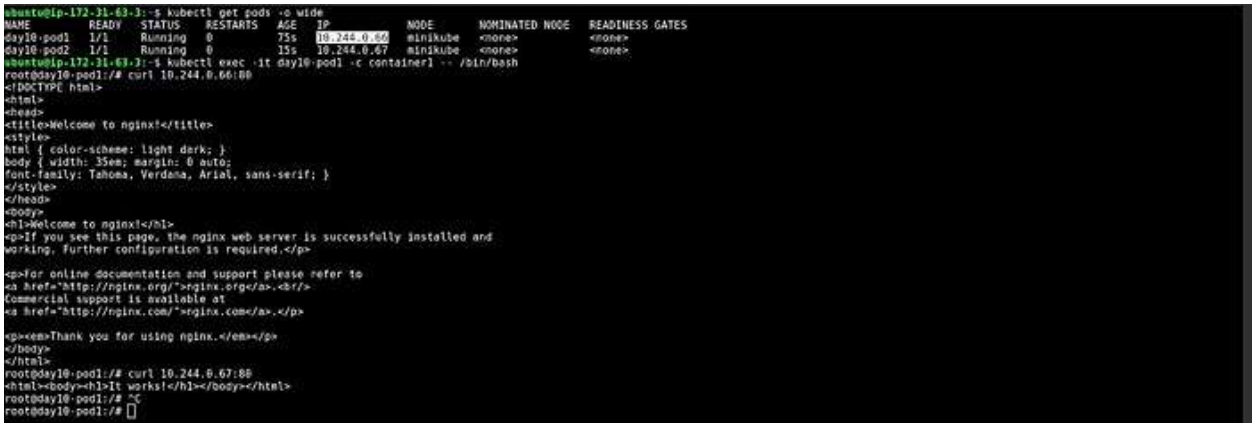
Run the below command.

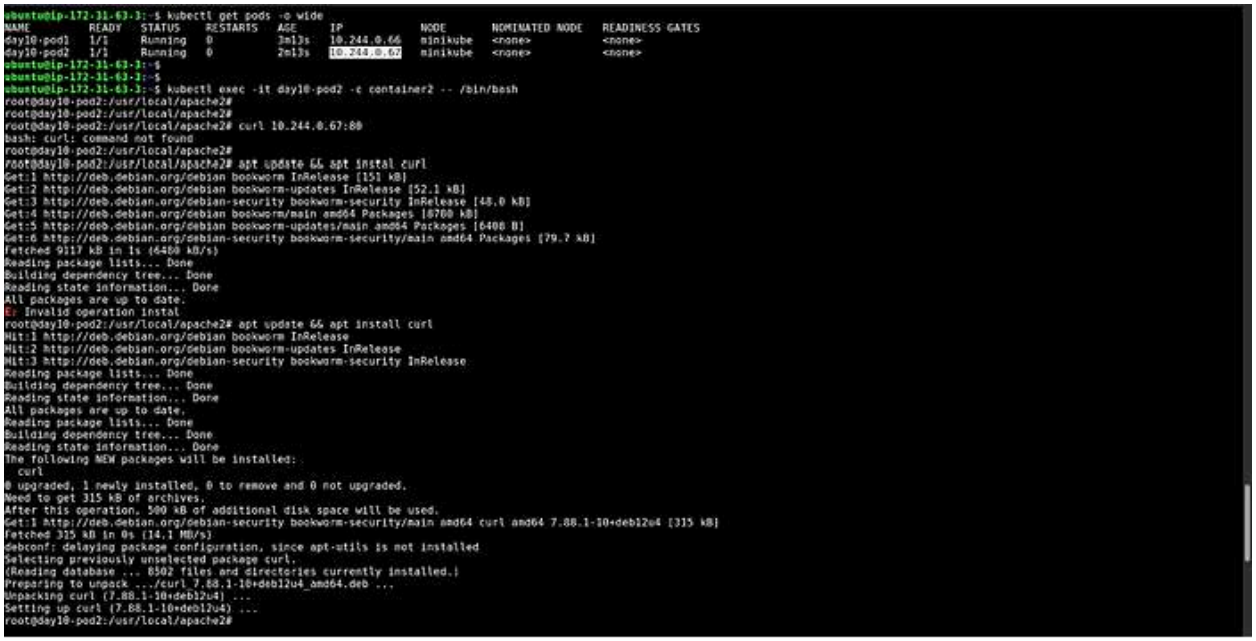
curl localhost:80

Access containers within the same Node.

```
apiVersion: v1
kind: Pod
metadata:
 name: day10-pod1
 labels:
   env: testing
   department: DevOps
spec:
 containers:
   -name: container1
     image: nginx
     ports:
       -containerPort: 80
apiVersion: v1
```

```
kind: Pod
metadata:
 name: day10-pod2
 labels:
   env: testing
   department: DevOps
spec:
 containers:
   -name: container2
     image: httpd
     ports:
       -containerPort: 80
```





Service Object in Kubernetes

To configure Networking for deployed applications on pods and containers, we use Service objects.

There are four main types of Kubernetes Services:

1. **ClusterIP**: As the name suggests, With the help of this Service. Kubernetes expose the service within the cluster which means the service or application won't be accessible outside of the cluster.

You can view the Service YML file and see how to use this service.

```
apiVersion: v1
kind: Service
metadata:
 name: httpd-service
spec:
 ports:
   -port: 80
```

```
    targetPort: 80
 selector:
   name: DevOps
 type: ClusterIP
```

2. **NodePort** : This is the next stage of the ClusterIP where you want to deploy your application or service that should be accessible to the world without any interruption. In this Service, the node port exposes the service or application through the static port on each node's IP.

You can view the Service YML file and see how to use this service.

```
apiVersion: v1
kind: Service
metadata:
 name: httpd-service
spec:
 ports:
   -port: 80
     targetPort: 80
 selector:
   name: DevOps
 type: NodePort
```

3. **LoadBalancer** : Load balancers are used to distribute the traffic between the multiple pods. With the help of this service object, the services will be exposed via the cloud's load balancer.

You can view the Service YML file and see how to use this service.

```
apiVersion: v1
kind: Service
metadata:
 name: svc-lb
spec:
 type: LoadBalancer
 selector:
   tag: DevOps
 ports:
 -name: port-lb
   protocol: TCP
   port: 80
   targetPort: 80
```

4. **ExternalName** : This is a similar object service to ClusterIP but it does have DNS CName instead of Selectors and labels. In other words, services will be mapped to a DNS name.

You can view the Service YML file and see how to use this service.

```
```

```
kind: Service
metadata:
 name: k8-service
 namespace: dev
spec:
 type: ExternalName
 externalName: k8.learning.com
```

Kubernetes Advanced Networking: CNI and Calico

What is CNI?

CNI stands for Container Network Interface. As the name suggests, CNI works on the networking level where CNI takes care of the pods and other things in Kubernetes. CNI ensures how the containers and pods should connect to the network. There are many CNIs available in the market but today we will discuss Calico CNI.

What is Calico CNI?

Calico is an open-source network and network security solution designed for Kubernetes.

Calico is one of the most popular CNIs which used to manage the networking between containers, pods, nodes, or multiple clusters. Calico works only on networking to provide fine-grained control over the containers, pods, nodes, or multiple clusters where Services and

Alternative of Calico CNI

- Flannel: Flannel is a straightforward CNI plugin that uses Layer 3 networking. It is good for small to medium size clusters but the networking policies feature is not good where Calico provides good control over networking policies.
- Weave: Weave is quite a good CNI plugin as compared to Flannel as it provides secure, scalable networking and networking policies as well. But it does not have rich features as Calico does.
- Cilium: Cilium is one of the best CNI plugins which competes with Calico CNI where Cilium provides the best security and observability. Cilium is a good choice for large complex clusters where security is a top concern.
- kube-router: kube-router is a lightweight CNI plugin where that provides good features like serving load balancing and network policies. It is good for small to medium size clusters.

Why you should use Calico CNI over other CNIs(Features)

- **Advanced Networking Policies** : With Calico CNI, we can define fine-grained networking policies over the containers or pods such as which pod can communicate to which other pod and apply rules based on labels, ports, and more. This level of control is not possible through Kubernetes Native Networking.
- **Scalability** : Calico is known for its Scalability where it can handle large clusters with ease and efficiently manage network traffic which makes it suitable for enterprise-level applications with multiple pods.
- **Cross-Cluster Networking** : Calico can be used to connect multiple Kubernetes clusters together, which can be beneficial in hybrid or multi-cluster scenarios.

- **Border Gateway Protocol(BGP) routing** : Calico supports BGP for routing which is quite good if you want to integrate with on-premises data centers or public cloud environments.
- **Security** : Calico supports a very good level of security over the network traffic where Calico encrypts the network traffic so only authorized pods can communicate with the respective pods.

Key Concepts and Real-time Example

- **IP Address Management** : Calico supports managing the IP address for each pod where each pod is assigned to a unique IP address from the cluster's IP address range.
- **Routing and Network Policy** : Calico enables routing for the network traffic between pods. The Network policies can be applied to control traffic between pods. So, you can allow or deny communications between specific pods.
- **Load Balancing** : Calico handles load balancing in which it distributes the traffic between multiple pods.
- **Security and Encryption** : Calico provides security features to protect your Kubernetes clusters. It encrypts the network traffic so that you can ensure only authorized pods can communicate.

Think of Calico as a traffic control system in a city, where every vehicle(pod) gets a unique plate number and license and follows the traffic rules. The Traffic lights(network policies) ensure safe and fully controlled movement. Police officers(security) check for unauthorized actions and keep the movement controlled.

HandsOn

If you want to do HandsOn where you want to install the Calico Network. So you can refer to the Day10 of #30DaysOfKubernetes where we have set up the Master and Worker Node on the AWS EC2 Instance in which I have installed the Calico CNI Plugin.

For now, this is a link to install the Calico and list all the networks, and validate the Calico network

```
kubectl get pods -n kube-system
```

```
root@ip-10-0-0-15:/home/ubuntu# kubectl get po -n kube-system
NAME                                      READY   STATUS    RESTARTS   AGE
coredns-5dd5756b68-6ljnb                  0/1     Pending   0          66s
coredns-5dd5756b68-9xkvz                  0/1     Pending   0          66s
etcd-ip-10-0-0-15                         1/1     Running   0          71s
kube-apiserver-ip-10-0-0-15               1/1     Running   0          71s
kube-controller-manager-ip-10-0-0-15      1/1     Running   0          71s
kube-proxy-mm6p4                          1/1     Running   0          67s
kube-scheduler-ip-10-0-0-15               1/1     Running   0          71s
root@ip-10-0-0-15:/home/ubuntu#
```

```
kubectl apply -f
https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml
```

```
root@ip-10-0-0-15:/home/ubuntu# kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml
poddisruptionbudget.policy/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
serviceaccount/calico-node created
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipreservations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
```

kubectl get pods -n kube-system

```
root@ip-10-0-0-15:/home/ubuntu# kubectl get po -n kube-system
NAME                                       READY   STATUS    RESTARTS   AGE
calico-kube-controllers-658d97c59c-pthvg   1/1     Running   0          34s
calico-node-vvkpf                          1/1     Running   0          34s
coredns-5dd5756b68-6ljnb                   1/1     Running   0          2m38s
coredns-5dd5756b68-9xkvz                   1/1     Running   0          2m38s
etcd-ip-10-0-0-15                          1/1     Running   0          2m43s
kube-apiserver-ip-10-0-0-15                1/1     Running   0          2m43s
kube-controller-manager-ip-10-0-0-15       1/1     Running   0          2m43s
kube-proxy-mm4g4                           1/1     Running   0          2m39s
kube-scheduler-ip-10-0-0-15                1/1     Running   0          2m43s
root@ip-10-0-0-15:/home/ubuntu#
```

Kubernetes Volumes and LivenessProbes

The data is a very important thing for an application. In Kubernetes, data is kept for a short time in the applications in the pods/containers. There is no data persistent things available by default by Kubernetes. To overcome this issue, Kubernetes supports Volumes.

But before going into the types of Volumes. Let's understand some facts about pods and containers' short live data.

- The volumes reside inside the Pod which stores the data of all containers in that pod.
- If the container gets deleted, then the data will persist and it will be available for the new container which was created recently.
  Multiple containers within a pod can share one volume because the volume is attached to the pod.
  If the Pod gets deleted, then the volume will also get deleted which leads to a loss of data for all containers permanently.
  After deleting the pod, the new pod will be created with volume but this time volumes don't have any previous data or any data.

There are some types of Volumes supported by Kubernetes

EmptyDir

- This is one of the basic volume types that we have discussed earlier in the facts. This volume is used to share the volumes between multiple containers within a pod instead of the host machine or any Master/Worker Node.
- EmptyDir volume is created when the pod is created and it exists as long as a pod.
- There is no data available in the EmptyDir volume type when it is created for the first.
- Containers within the pod can access the other containers' data. However, the mount
- path can be different for each container.
  If the Containers get crashed then, the data will still persist and can be accessible by other or newly created containers.

HandsOn

In this snippet, I have created one file in container1 and looking for the same file and content from container2 which is possible.

EmptyDir YML file

```
apiVersion: v1
kind: Pod
metadata:
 name: emptydir
spec:
 containers:
```

```
-name: container1
  image: ubuntu
  command: ["/bin/bash", "-c", "while true; do echo This is Day13 of 30DaysOfKubernetes;
sleep 5 ; done"]
  volumeMounts:
    -name: day13
      mountPath: "/tmp/container1"
-name: container2
  image: centos
  command: ["/bin/bash", "-c", "while true; do echo Chak de INDIA!; sleep 5 ; done"]
  volumeMounts:
    -name: day13
      mountPath: "/tmp/container2"
volumes:
-name: day13
emptyDir: {}
```

In this snippet, I am creating a file in container2 and looking for the file and the same content through container1 which is possible.

2. hostPath

This volume type is the advanced version of the previous volume type EmptyDir.

In EmptyDir, the data is stored in the volumes that reside inside the Pods only where the host machine doesn't have the data of the pods and containers.

hostpath volume type helps to access the data of the pods or container volumes from the host machine.

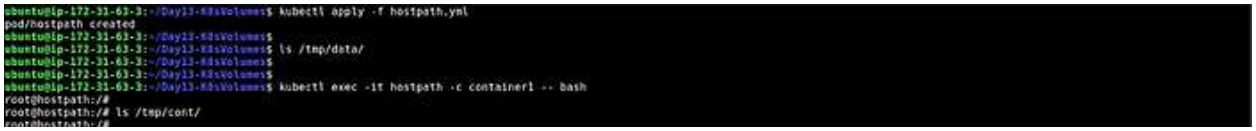
hostpath replicates the data of the volumes on the host machine and if you make the changes from the host machine then the changes will be reflected to the pods volumes(if attached).

HandsOn

In this snippet, once the pod has been created the data directory is also created on the local machine(Minikube Cluster).

hostPath YML file

```
apiVersion: v1
kind: Pod
metadata:
 name: hostpath
spec:
 containers:
 -name: container1
   image: ubuntu
   command: ["/bin/bash", "-c", "while true; do echo This is Day13 of 30DaysOfKubernetes;
sleep 5 ; done"]
   volumeMounts:
   -mountPath: "/tmp/cont"
     name: hp-vm
 volumes:
 -name: hp-vm
   hostPath:
     path: /tmp/data
```



In this snippet, I am creating a txt file inside the pod's mapped directory /tmp/cont which is mapped to the local directory /tmp/data and after that, I am looking for the same file and content on the local machine directory /tmp/data.

3. Persistent Volume

- Persistent Volume is an advanced version of EmptyDir and hostPath volume types.
- Persistent Volume does not store the data over the local server. It stores the data on the cloud or some other place where the data is highly available.

- In previous volume types, if pods get deleted then the data will be deleted as well. But with the help of Persistent Volume, the data can be shared with other pods or other worker node's pods as well after the deletion of pods.
- One Persistent Volume is distributed across the entire Kubernetes Cluster. So that,
- any node or any node's pod can access the data from the volume accordingly.
- With the help of Persistent Volume, the data will be stored on a central location such
- as EBS, Azure Disks, etc.
  Persistent Volumes are the available storage(remember for the next volume type). If you want to use Persistent Volume, then you have to claim that volume with the help of the manifest YAML file.

4. Persistent Volume Claim(PVC)

- To get the Persistent Volume, you have to claim the volume with the help of PVC.
- When you create a PVC, Kubernetes finds the suitable PV to bind them together.
- After a successful bound to the pod, you can mount it as a volume.
- Once a user finishes its work, then the attached volume gets released and will be
- used for recycling such as new pod creation for future usage.
  If the pod is terminating due to some issue, the PV will be released but as you know the new pod will be created quickly then the same PV will be attached to the newly created Pod.

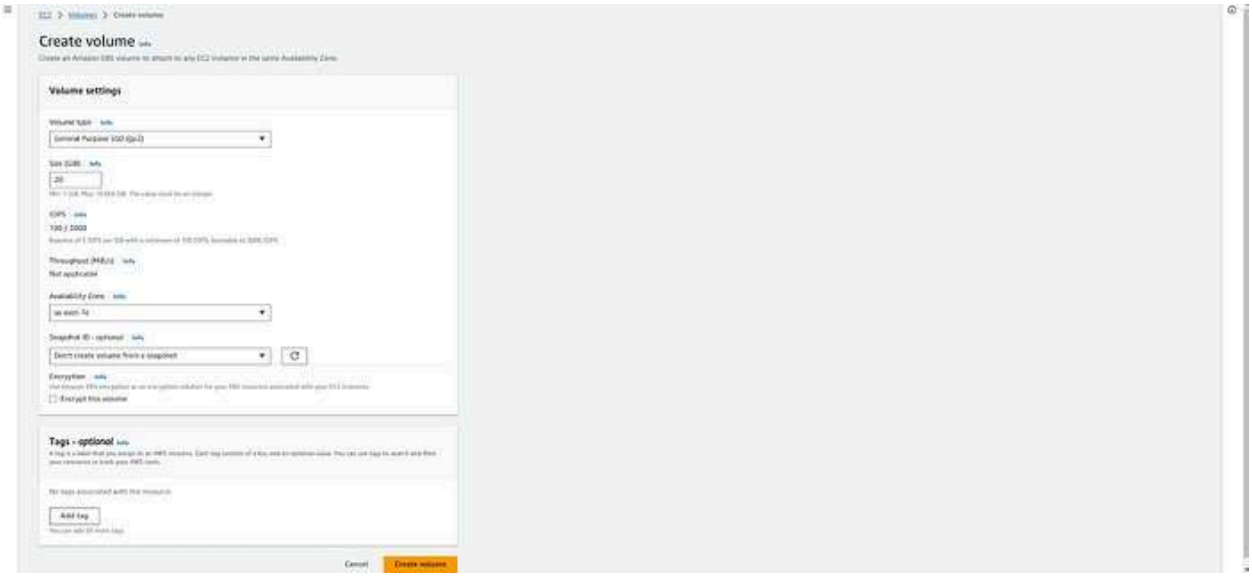
Now, As you know the Persistent Volume will be on Cloud. So, there are some facts and terms and conditions are there for EBS because we are using AWS cloud for our K8 learning. So, let's discuss it as well:

- EBS Volumes keeps the data forever where the emptydir volume did not. If the pods
- get deleted then, the data will still exist in the EBS volume. The nodes on which
- running pods must be on AWS Cloud only(EC2 Instances). Both(EBS Volume & EC2
- Instances) must be in the same region and availability zone. EBS only supports a single EC2 instance mounting a volume

HandsOn

To perform this demo, Create an EBS volume by clicking on 'Create volume'.

Pass the Size for the EBS according to you, and select the Availability zone where your EC2 instance is created, and click on Create volume.



Now, copy the volume ID and paste it into the PV YML file(12th line)

PV YML file

In this snippet, we have created a Persistent Volume where the EBS volume is attached and created 1GB of capacity.

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: myebsvol
spec:
 capacity:
   storage: 1Gi
 accessModes:
  -ReadWriteOnce
 persistentVolumeReclaimPolicy: Recycle
 awsElasticBlockStore:
  volumeID: #Your_VolumeID
  fsType: ext4
```

PVC YML file

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
```

```
 name: myebsvolclaim
spec:
 accessModes:
  -ReadWriteOnce
  resources
   requests:
    storage: 1Gi
```

In this snippet, we have created a Persistent Volume Claim in which PVC requests PV to get 1GB, and as you can see the volume is bound successful.

Deployment YML file

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: pvdeploy
spec:
 replicas: 1
 selector:
  matchLabels:
   app: mypv
 template:
  metadata:
   labels:
    app: mypv
  spec:
   containers:
   -name: shell
    image: centos
    command: ["bin/bash", "-c", "sleep 10000"]
    volumeMounts:
    -name: mypd
      mountPath: "/tmp/persistent"
   volumes:
    -name: mypd
     persistentVolumeClaim:
       claimName: myebsvolclaim
```

In this snippet, we have created a deployment for PV and PVC demonstration.

```
ubuntu@ip-172-31-63-3:~/PV-C$ sudo vim day13-deploy.yml
ubuntu@ip-172-31-63-3:~/PV-C$
ubuntu@ip-172-31-63-3:~/PV-C$
ubuntu@ip-172-31-63-3:~/PV-C$ kubectl apply -f day13-deploy.yml
deployment.apps/pvdeploy created
ubuntu@ip-172-31-63-3:~/PV-C$
ubuntu@ip-172-31-63-3:~/PV-C$
ubuntu@ip-172-31-63-3:~/PV-C$ kubectl get deploy
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
pvdeploy   1/1     1            1           6s
ubuntu@ip-172-31-63-3:~/PV-C$
ubuntu@ip-172-31-63-3:~/PV-C$
ubuntu@ip-172-31-63-3:~/PV-C$ kubectl get rs
NAME                  DESIRED   CURRENT   READY   AGE
pvdeploy-5c885cf554   1         1         1       12s
ubuntu@ip-172-31-63-3:~/PV-C$
ubuntu@ip-172-31-63-3:~/PV-C$
ubuntu@ip-172-31-63-3:~/PV-C$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
hostpath                    1/1     Running   0          91m
pvdeploy-5c885cf554-5gvhl   1/1     Running   0          22s
ubuntu@ip-172-31-63-3:~/PV-C$
```

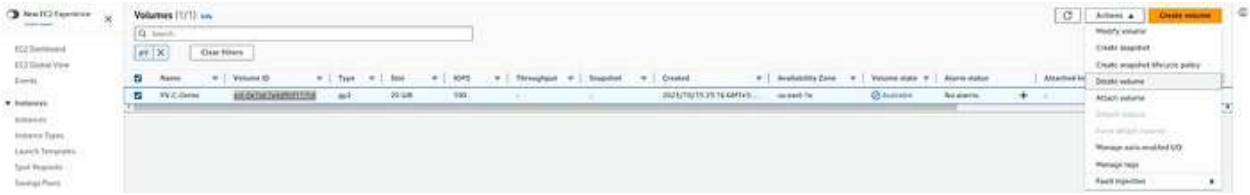
In this snippet, we have logged in to the created container and created one file with some text.

```
ubuntu@ip-172-31-63-3:~/PV-C$ kubectl get pod
NAME                        READY   STATUS    RESTARTS   AGE
pvdeploy-5c885cf554-5gvhl   1/1     Running   0          2m
ubuntu@ip-172-31-63-3:~/PV-C$
ubuntu@ip-172-31-63-3:~/PV-C$
ubuntu@ip-172-31-63-3:~/PV-C$ kubectl exec -it pvdeploy-5c885cf554-5gvhl -c shell -- /bin/bash
[root@pvdeploy-5c885cf554-5gvhl /]#
[root@pvdeploy-5c885cf554-5gvhl /]#
[root@pvdeploy-5c885cf554-5gvhl /]# ls /tmp/
.ICE-unix/      .Test-unix/     .X11-unix/      .XIM-unix/      .font-unix/     ks-script-4luisyla  ks-script-o23i7rc2  ks-script-xdei4wuu  persistent/
[root@pvdeploy-5c885cf554-5gvhl /]# ls /tmp/perssistent/
[root@pvdeploy-5c885cf554-5gvhl /]#
[root@pvdeploy-5c885cf554-5gvhl /]# echo "Data is Presnent in Persistent" >> /tmp/persistent/data.txt
[root@pvdeploy-5c885cf554-5gvhl /]#
[root@pvdeploy-5c885cf554-5gvhl /]# cat /tmp/persistent/data.txt
Data is Presnent in Persistent
[root@pvdeploy-5c885cf554-5gvhl /]#
```

In this snippet, we have deleted the pod, and then because of replicas the new pod was created quickly. Now, we have logged in to the newly created pod and checked for the file that we created in the previous step, and as you can see the file is present which is expected.

```
ubuntu@ip-172-31-63-3:~/PV-C$ kubectl get pod
NAME                        READY   STATUS    RESTARTS   AGE
pvdeploy-5c885cf554-5gvhl   1/1     Running   0          4m28s
ubuntu@ip-172-31-63-3:~/PV-C$
ubuntu@ip-172-31-63-3:~/PV-C$
ubuntu@ip-172-31-63-3:~/PV-C$ kubectl delete pod pvdeploy-5c885cf554-5gvhl
pod "pvdeploy-5c885cf554-5gvhl" deleted
ubuntu@ip-172-31-63-3:~/PV-C$
ubuntu@ip-172-31-63-3:~/PV-C$ kubectl get pod
NAME                        READY   STATUS    RESTARTS   AGE
pvdeploy-5c885cf554-sqtvb   1/1     Running   0          42s
ubuntu@ip-172-31-63-3:~/PV-C$
ubuntu@ip-172-31-63-3:~/PV-C$ kubectl exec -it pvdeploy-5c885cf554-sqtvb -c shell -- /bin/bash
[root@pvdeploy-5c885cf554-sqtvb /]#
[root@pvdeploy-5c885cf554-sqtvb /]#
[root@pvdeploy-5c885cf554-sqtvb /]# ls /tmp/persistent/
data.txt
[root@pvdeploy-5c885cf554-sqtvb /]#
[root@pvdeploy-5c885cf554-sqtvb /]# cat /tmp/persistent/data.txt
Data is Presnent in Persistent
[root@pvdeploy-5c885cf554-sqtvb /]#
```

The demonstration has been completed now feel free to delete the volume.



LivenessProbe (HealthCheck)

- LivenessProbe is a rich feature in Kubernetes that is used to check the health of your
- application.
- Kubernetes by default doesn't check the health check of the applications.
  If you want to use the livenessProve feature and want to check the health of your application, you have to mention it in the manifest file.

- livenessProbe expects a 0 output which means the application is running perfectly. But if there is any other output except 0 then livenessProbe recreates the container and repeats the same process.
- livenessProbe repeats the process after particular seconds or minutes(specified by
- you) to check the health of the application.
  If there is a load balancer attached to multiple pods then, livenessProbe checks the health of the application and if the application's health check status is not healthy then livenessProbe removes the particular pod from the load balancer and recreates the new pod and repeats the same process.

HandsOn

In this snippet, we have created a pod, and if you observe there is a new thing added while describing the pod through the kubectl describe pod command in the bottom which comes under Containers liveness.

livenessProbe YML file

```
apiVersion: v1
kind: Pod
metadata:
 labels:
   test: liveness
 name: mylivenessprobe
spec:
 containers:
 -name: liveness
   image: ubuntu
   args:
   -/bin/sh
   --c
   -touch /tmp/healthy; sleep 1000
   livenessProbe:
     exec:
       command:
       -cat
       -/tmp/healthy
     initialDelaySeconds: 5
     periodSeconds: 5
     timeoutSeconds: 30
```

```
ubuntu@ip-172-31-63-3:~/LivenessProbe$ sudo vim livenessprobe.yml
ubuntu@ip-172-31-63-3:~/LivenessProbe$
ubuntu@ip-172-31-63-3:~/LivenessProbe$
ubuntu@ip-172-31-63-3:~/LivenessProbe$ kubectl apply -f livenessprobe.yml
pod/mylivenessprobe created
ubuntu@ip-172-31-63-3:~/LivenessProbe$
ubuntu@ip-172-31-63-3:~/LivenessProbe$
ubuntu@ip-172-31-63-3:~/LivenessProbe$ kubectl get pods
NAME              READY   STATUS    RESTARTS   AGE
mylivenessprobe   1/1     Running   0          8s
ubuntu@ip-172-31-63-3:~/LivenessProbe$
ubuntu@ip-172-31-63-3:~/LivenessProbe$ kubectl describe pod mylivenessprobe
Name:             mylivenessprobe
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.49.2
Start Time:       Sun, 15 Oct 2023 18:27:53 +0000
Labels:           test=liveness
Annotations:      <none>
Status:           Running
IP:               10.244.0.18
IPs:
  IP:  10.244.0.18
Containers:
  liveness:
    Container ID:  docker://c2d2b9090a182351880bb93ddde24aa7b1124af898bf84d426dd0ebeedd1f783
    Image:         ubuntu
    Image ID:      docker-pullable://ubuntu@sha256:2b7412e6465c3c7fc5bb21d3e6f1917c167358449fecec8176c6e496e5c1f05f
    Port:          <none>
    Host Port:     <none>
    Args:
      /bin/sh
      -c
      touch /tmp/healthy; sleep 1000
    State:          Running
      Started:      Sun, 15 Oct 2023 18:27:54 +0000
    Ready:          True
    Restart Count:  0
    Liveness:       exec [cat /tmp/healthy] delay=5s timeout=30s period=5s #success=1 #failure=3
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-h4fjr (ro)
```

If you see the YML file, there is one condition if the file healthy is not present in /tmp directory then livenessProbe will recreate the container. So, we have deleted that file.

```
ubuntu@ip-172-31-63-3:~/LivenessProbe$ kubectl get pods
NAME              READY   STATUS    RESTARTS   AGE
mylivenessprobe   1/1     Running   0          93s
ubuntu@ip-172-31-63-3:~/LivenessProbe$
ubuntu@ip-172-31-63-3:~/LivenessProbe$ kubectl exec -it mylivenessprobe -c liveness -- /bin/bash
root@mylivenessprobe:/#
root@mylivenessprobe:/#
root@mylivenessprobe:/# cat /tmp/healthy
root@mylivenessprobe:/#
root@mylivenessprobe:/# echo $?
0
root@mylivenessprobe:/#
root@mylivenessprobe:/# rm /tmp/healthy
root@mylivenessprobe:/#
root@mylivenessprobe:/#
root@mylivenessprobe:/# cat /tmp/healthy
cat: /tmp/healthy: No such file or directory
root@mylivenessprobe:/#
root@mylivenessprobe:/#
root@mylivenessprobe:/# echo $?
1
root@mylivenessprobe:/#
```

After deleting the file, if you run kubectl describe pod <pod-name> you will see in the last two lines that the container is failing because the condition is not meeting.

```
Events:
  Type     Reason     Age                 From               Message
  ----     ------     ----                ----               -------
  Normal   Scheduled  3m18s               default-scheduler  Successfully assigned default/mylivenessprobe to minikube
  Normal   Pulling    3m17s               kubelet            Pulling image "ubuntu"
  Normal   Pulled     3m17s               kubelet            Successfully pulled image "ubuntu" in 150.222253ms (151.408942ms including waiting)
  Normal   Created    3m17s               kubelet            Created container liveness
  Normal   Started    3m17s               kubelet            Started container liveness
  Warning  Unhealthy  22s (x3 over 32s)   kubelet            Liveness probe failed: cat: /tmp/healthy: No such file or directory
  Normal   Killing    22s                 kubelet            Container liveness failed liveness probe, will be restarted
```

Kubernetes ConfigMaps & Secrets

ConfigMap

ConfigMap is used to store the configuration data in key-value pairs within Kubernetes. This is one of the ways to decouple the configuration from the application to get rid of hardcoded values. Also, if you observe some important values keep changing according to the environments such as development, testing, production, etc ConfigMap helps to fix this issue to decouple the configurations. ConfigMap stores non-confidential data. ConfigMap can be created through an imperative or declarative way.

- Creating the configMap is the first process which can be done by commands only or aYAML file.
- After creating the configMap, we use the data in the pod by injecting the pods.
- After injecting the pods, if there is any update in the configuration we can modify the configMap, and the changes will be reflected in the injected pod.

Secrets

There are lot of confidential information that needs to be stored on the server such as database usernames, passwords, or API Keys. To keep all the important data secure, Kubernetes has a Secrets feature that encrypts the data. Secrets can store data up to 1MB which would be enough. Secrets can be created via imperative or declarative ways. Secrets are stored in the /tmps directory and can be accessible to pods only.

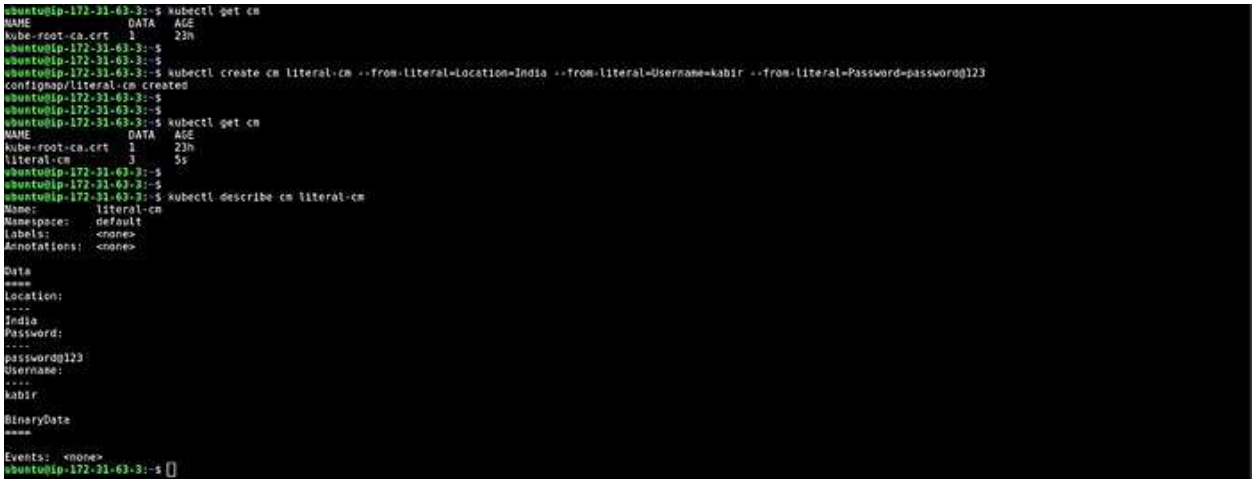
- Creating the Secrets is the first process that can be created by commands or a YAML file.
- After creating the Secrets, applications need to use the credentials or database credentials which will be done by injecting with the pods.

ConfigMap HandsOn

Creating ConfigMap **from literal**

In this snippet, we have created the configMap through — from-literal which means you just need to provide the key value instead of providing the file with key-value pair data.

At the bottom, you can see the data that we have created through the Literal.



CM from file

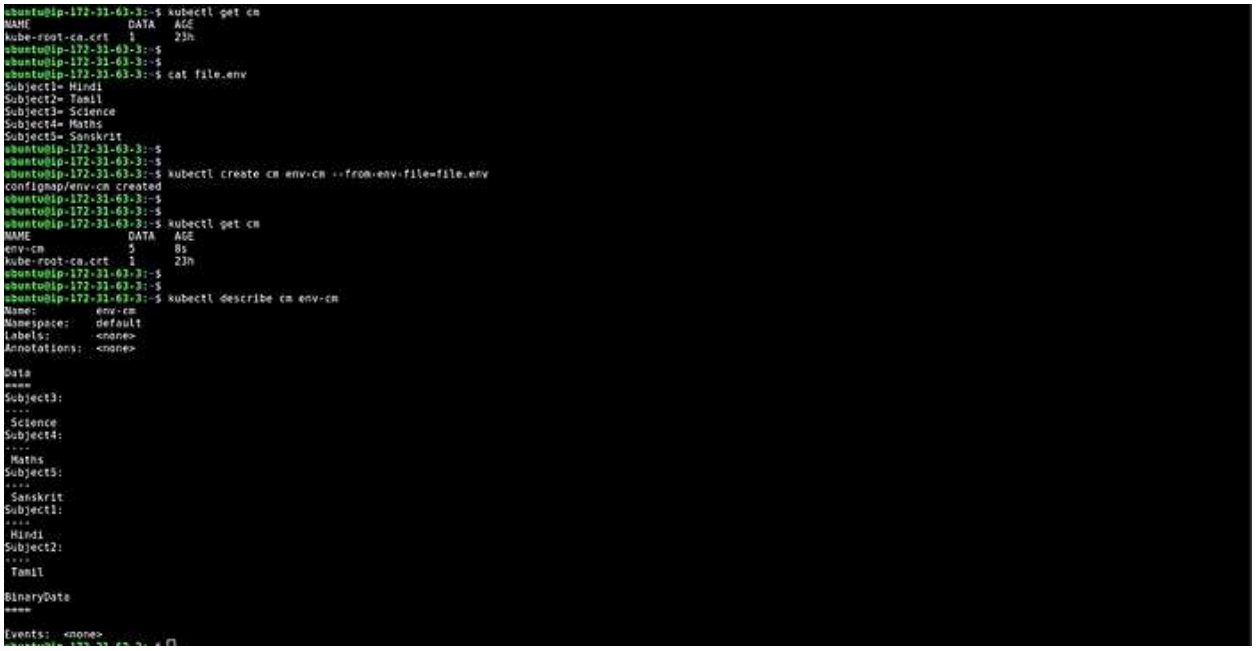
In this snippet, We have created one file first.conf which has some data, and created the configMap with the help of that file.

At the bottom, you can see the data that we have created through the file.

CM from the env file

In this snippet, We have created one environment file first.env which has some data in key-value pairs, and created the configMap with the help of the environment file.

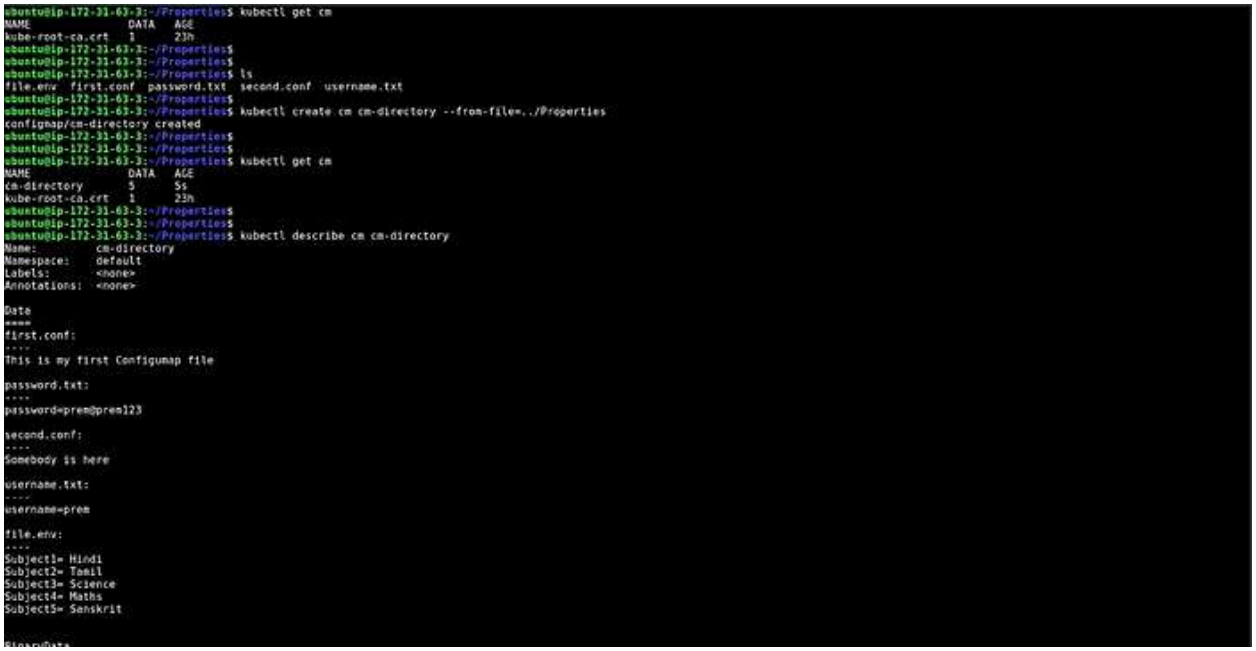
At the bottom, you can see the data that we have created through the env file.



What if you have to create configMap for tons of files?

In this snippet, We have created multiple files in a directory with different extensions that have different types of data and created the configMap for the entire directory.

At the bottom, you can see the data that we have created for the entire directory.

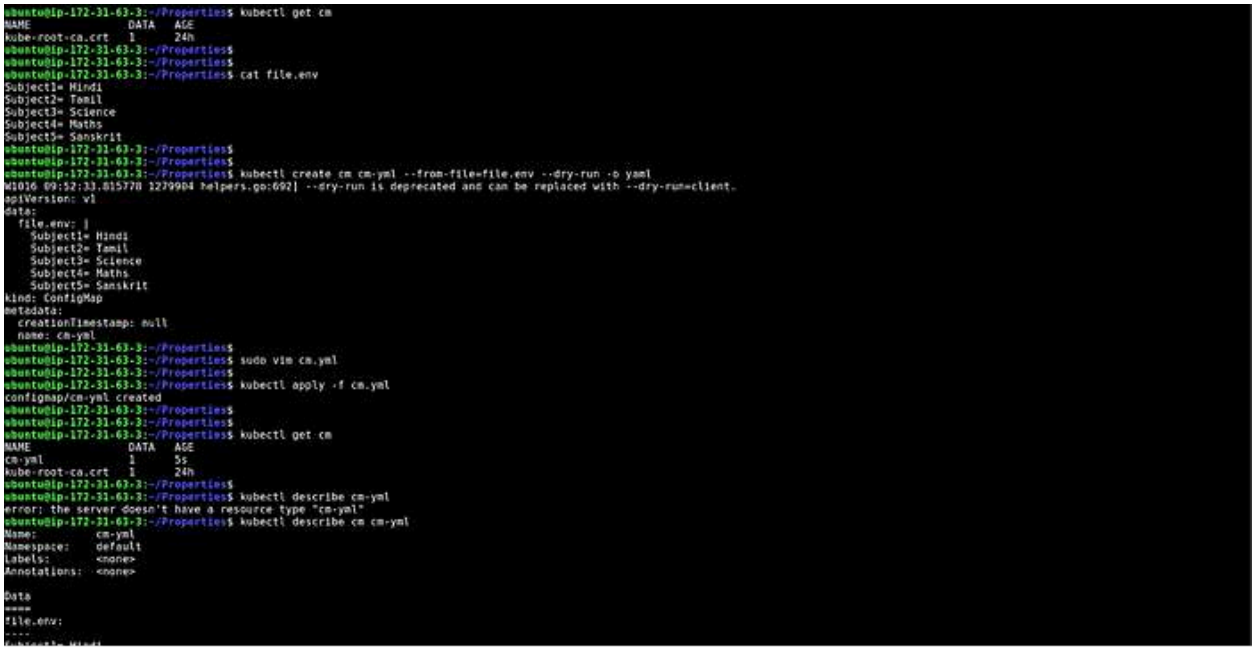


CM from the YAML file

The imperative way is not very good if you have to repeat the same tasks again and again. Now, we will look at how to create configMap through the YAML file.

In this snippet, We have created one file and run the command with --from-file, and in the end, we add -o yaml which generates the YAML file. You can copy that YAML file modify it according to your key-value pairs and apply the file.

At the bottom, you can see the data that we have created through the YAML file.



In the above steps, we have created four types of configMaps but here, we will learn how to use those configMaps by injecting configMaps into the pods.

Injecting CM into the pod with specific key pairs

In this snippet, We have created a YAML file in which we have mentioned the configMap name with the key name which will be added to the pod's environment variable.

```
apiVersion: v1
kind: Pod
metadata:
 name: firstpod
spec:
 containers:
  -image: coolgourav147/nginx-custom
    name: firstcontainer
    imagePullPolicy: Never
```

```
    env:
     -name: valuefromenv
       valueFrom:
         configMapKeyRef:
           key: Subject2
           name: cm-from-env
```

```
ubuntu@ip-172-31-63-3:~/Properties$ kubectl get cm
NAME               DATA   AGE
kube-root-ca.crt   1      24h
ubuntu@ip-172-31-63-3:~/Properties$ kubectl create cm cm-from-env --from-env-file=file.env
configmap/cm-from-env created
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubect get cm
kubect: command not found
ubuntu@ip-172-31-63-3:~/Properties$ kubectl get cm
NAME               DATA   AGE
cm-from-env        5      11s
kube-root-ca.crt   1      24h
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl describe cm cm-from-env
Name:         cm-from-env
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
Subject1:
----
 Hindi
Subject2:
----
 Tamil
Subject3:
----
 Science
Subject4:
----
 Maths
Subject5:
----
 Sanskrit

BinaryData
====

Events:  <none>
ubuntu@ip-172-31-63-3:~/Properties$ sudo vim cm-pod.yml
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl apply -f cm-pod.yml
pod/cm-pod created
```

At the bottom, you can see the configuration in the pod's environment.

```
ubuntu@ip-172-31-63-3:~/Properties$ kubectl apply -f cm-pod.yml
pod/firstpod created
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl get pods
NAME       READY   STATUS    RESTARTS   AGE
firstpod   1/1     Running   0          5s
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl exec -it firstpod -- env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=firstpod
TERM=xterm
valuefromenv= Tamil
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_PORT=443
NGINX_VERSION=1.17.6
NJS_VERSION=0.3.7
PKG_RELEASE=1~buster
HOME=/root
ubuntu@ip-172-31-63-3:~/Properties$
```

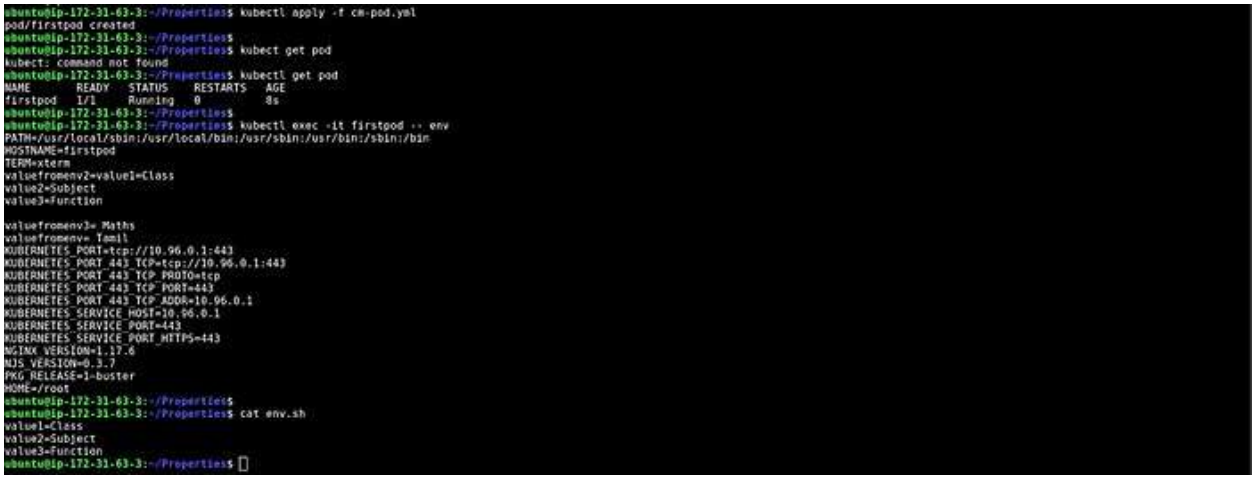
Injecting multiple CMs with specific and multiple values

In this snippet, we have added multiple key pairs from different files.

At the bottom, you can see the data that is from different files.

```
apiVersion: v1
kind: Pod
metadata:
 name: firstpod
spec:
 containers:
   -image: coolgourav147/nginx-custom
    name: firstcontainer
    imagePullPolicy: Never
```

```
    env:
     -name: valuefromenv
       valueFrom:
         configMapKeyRef:
           key: Subject2
           name: cm-from-env
     -name: valuefromenv2
       valueFrom:
         configMapKeyRef:
           key:  env.sh
           name: cm2
     -name: valuefromenv3
       valueFrom:
         configMapKeyRef:
           key: Subject4
           name: cm-from-env
```



Injecting the created environment file single cms and getting all the value

```
apiVersion: v1
kind: Pod
metadata:
 name: firstpod
spec:
 containers:
   -image: coolgourav147/nginx-custom
     name: firstcontainer
     imagePullPolicy: Never
     envFrom:
      -configMapRef:
          name: cm-from-env
```

```
ubuntu@ip-172-31-63-3:~/Properties$ sudo vim cm-pod2.yml
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl get cm
NAME               DATA   AGE
cm-from-env        5      76m
cm2                1      15m
kube-root-ca.crt   1      25h
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl apply -f cm-pod2.yml
pod/firstpod created
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl get pods
NAME       READY   STATUS    RESTARTS   AGE
firstpod   1/1     Running   0          5s
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl exec -it firstpod -- env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=firstpod
TERM=xterm
Subject2= Tamil
Subject3= Science
Subject4= Maths
Subject5= Sanskrit
Subject1= Hindi
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT=tcp://10.96.0.1:443
NGINX_VERSION=1.17.6
NJS_VERSION=0.3.7
PKG_RELEASE=1~buster
HOME=/root
ubuntu@ip-172-31-63-3:~/Properties$
```

Injecting cm in the pod with the entire proper file

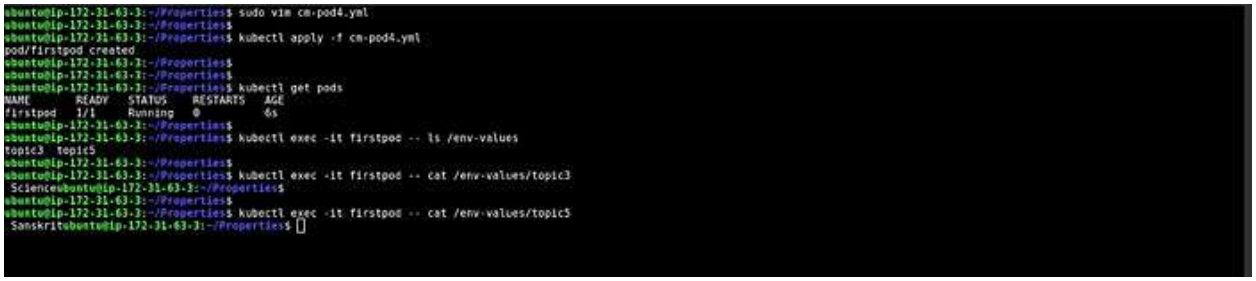
```
apiVersion: v1
kind: Pod
metadata:
 name: firstpod
spec:
 containers:
   -image: coolgourav147/nginx-custom
     name: firstcontainer
     imagePullPolicy: Never
     volumeMounts:
     -name: test
       mountPath: "/env-values"
       readOnly: true
 volumes:
   -name: test
     configMap:
          name: cm-from-env
```

```
ubuntu@ip-172-31-63-3:~/Properties$ sudo vim cm-pod3.yml
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl apply -f cm-pod3.yml
pod/firstpod created
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl get pods
NAME       READY   STATUS    RESTARTS   AGE
firstpod   1/1     Running   0          3s
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl exec -it first -- ls /env-values
Error from server (NotFound): pods "first" not found
ubuntu@ip-172-31-63-3:~/Properties$ kubectl exec -it firstpod -- ls /env-values
Subject1  Subject2  Subject3  Subject4  Subject5
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl exec -it firstpod -- cat /env-values/Subject4
 Mathsubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl exec -it firstpod -- cat /env-values/Subject2
 Tamilubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$
```

Injecting CM and creating a file in the pod with the selected key pairs

```
apiVersion: v1
kind: Pod
metadata:
 name: firstpod
spec:
 containers:
   -image: coolgourav147/nginx-custom
```

```
    name: firstcontainer
    imagePullPolicy: Never
    volumeMounts:
    -name: test
      mountPath: "/env-values"
      readOnly: true
volumes:
  -name: test
    configMap:
         name: cm-from-env
         items:
          -key: Subject3
            path: "topic3"
          -key: Subject5
            path: "topic5"
```

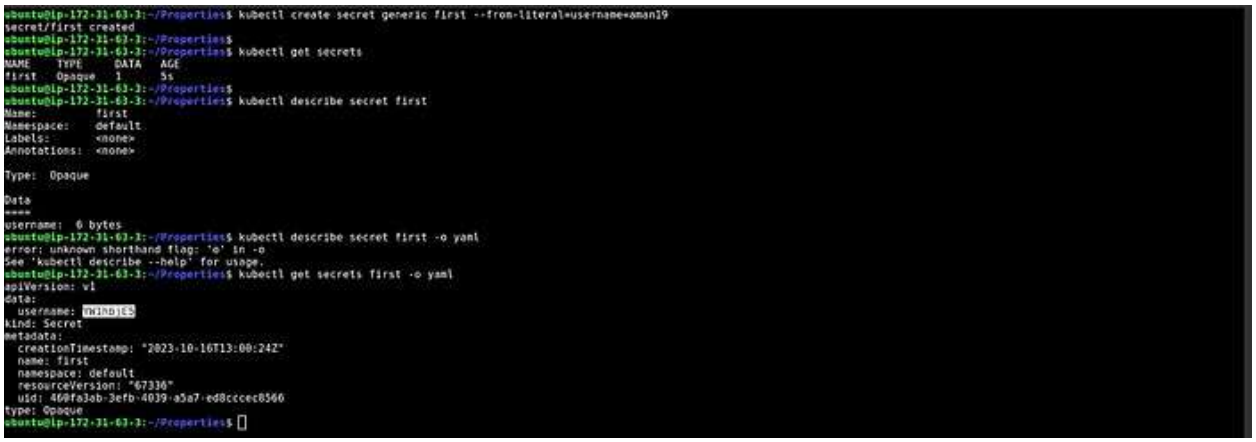


Secrets HandsOn

Creating Secrets from literal

In this snippet, we have created the Secrets through — from-literal which means you just need to provide the key value instead of providing the file with key-value pair data.

At the bottom, you can see the key and encrypted value because Kubernetes encrypts the secrets.



Secrets from file

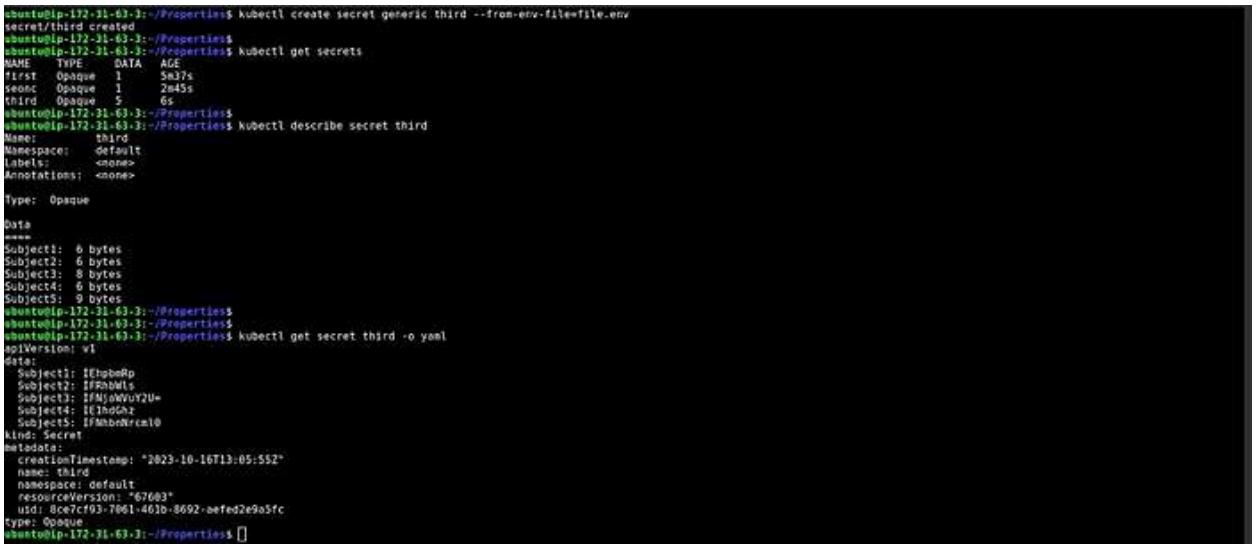
In this snippet, We have created one file first.conf which has some data, and created the Secrets with the help of that file.

At the bottom, you can see the encrypted data..

Secrets from the env file

In this snippet, We have created one environment file first.env which has some data in key-value pairs, and created the Secrets with the help of the environment file.

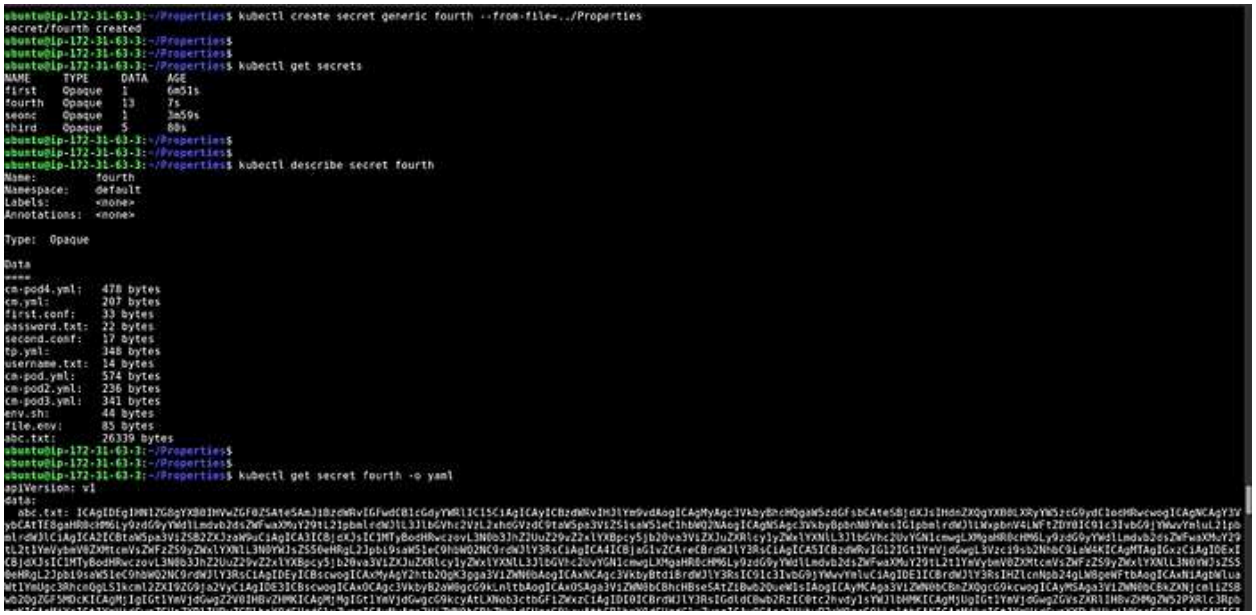
At the bottom, you can see the encrypted data.



What if you have to create Secrets for tons of files?

In this snippet, We have created multiple files in a directory with a different extension that has different types of data and created the Secrets for the entire directory.

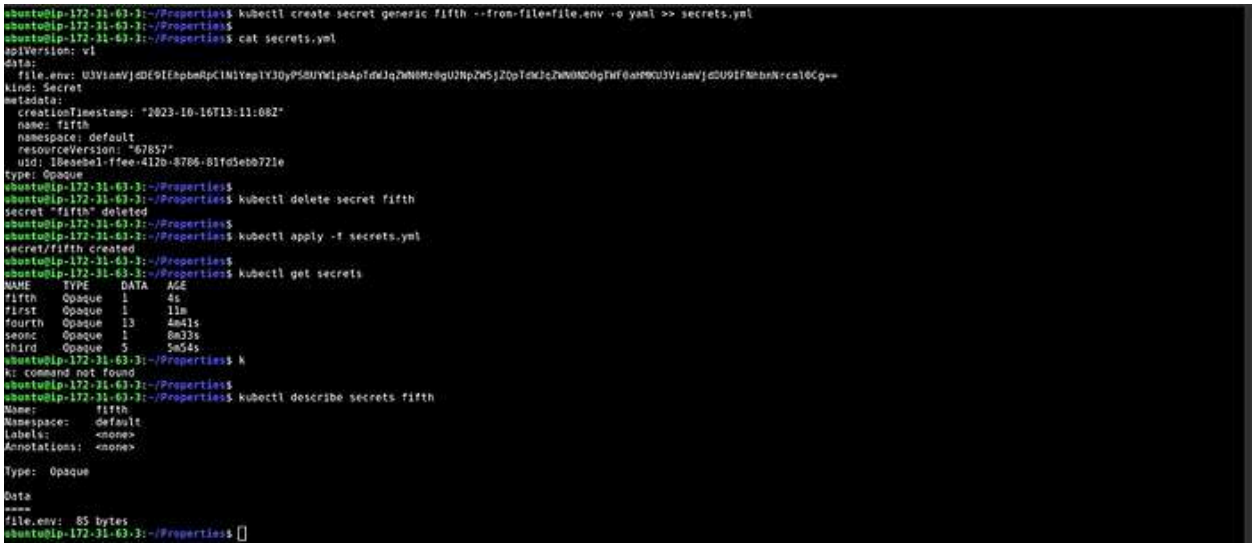
At the bottom, you can see the encrypted data.



CM from the YAML file

In this snippet, We have created one file and run the command with — from-file, and in the end, we add -o yaml which generates the YAML file. You can copy that YAML file modify it according to your key-value pairs and apply the file.

At the bottom, you can see the encrypted data.



In the above steps, we have created four types of Secrets but here, we will learn how to use those Secrets by injecting Secrets to the pods.

Injecting Secret with a pod for a particular key pairs

In this snippet, We have created a YAML file in which we have mentioned the Secrets name with the key name which will be added to the pod's environment variable.

```
kind: Pod
metadata:
 name: secret-pod
spec:
 containers:
  -image: coolgourav147/nginx-custom
    name: firstcontainer
    imagePullPolicy: Never
    env:
     -name: the-variable
       valueFrom:
         secretKeyRef:
           key: Subject1
           name: third
```

Injecting the created environment file single cms and getting all the value

```
apiVersion: v1
kind: Pod
metadata:
 name: secret-pod
spec:
 containers:
  -image: coolgourav147/nginx-custom
    name: firstcontainer
    imagePullPolicy: Never
    volumeMounts:
     -name: test
       mountPath: "/secrets-values"
 volumes:
  -name: test
    secret:
        secretName: seonc
```

```
ubuntu@ip-172-31-63-3:~/Properties$ sudo vim secrets3.yml
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl apply -f secrets3.yml
pod/secret-pod created
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl get pods
NAME         READY   STATUS    RESTARTS   AGE
secret-pod   1/1     Running   0          4s
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl exec -it secret-pod -- ls /secret-value
ls: cannot access '/secret-value': No such file or directory
command terminated with exit code 2
ubuntu@ip-172-31-63-3:~/Properties$ kubectl exec -it secret-pod -- ls /secret-values
ls: cannot access '/secret-values': No such file or directory
command terminated with exit code 2
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl exec -it secret-pod -- bash
root@secret-pod:/#
root@secret-pod:/#
root@secret-pod:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  secrets-values  srv  sys  tmp  usr  var
root@secret-pod:/# cd secrets-values/
root@secret-pod:/secrets-values# ls
file.env
root@secret-pod:/secrets-values# cat file.env
Subject1= Hindi
Subject2= Tamil
Subject3= Science
Subject4= Maths
Subject5= Sanskrit
```

Injecting Secrets and creating a file in the pod with the selected key pairs

```
apiVersion: v1
kind: Pod
metadata:
 name: secret-pod
spec:
 containers:
  -image: coolgourav147/nginx-custom
    name: firstcontainer
    imagePullPolicy: Never
    volumeMounts:
      -name: test
       mountPath: "/secrets-values"
 volumes:
  -name: test
    secret:
         secretName: third
         items:
          -key: Subject3
            path: "topic3"
          -key: Subject5
            path: "topic5"
```

```
ubuntu@ip-172-31-63-3:~/Properties$ kubectl apply -f secret4.yml
pod/secret-pod created
ubuntu@ip-172-31-63-3:~/Properties$ kubectl get pods
NAME         READY   STATUS    RESTARTS   AGE
secret-pod   1/1     Running   0          7s
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl exec -it secret-pod -- ls secrets-values/
topic3  topic5
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl exec -it secret-pod -- cat secrets-values/topic3
 Scienceubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$
ubuntu@ip-172-31-63-3:~/Properties$ kubectl exec -it secret-pod -- cat secrets-values/topic5
 Sanskritubuntu@ip-172-31-63-3:~/Properties$
```

Kubernetes Jobs

Kubernetes Jobs is a resource that is used to achieve a particular work such as a backup script and, once the work is completed the pod will be deleted.

Use cases:

- Database backup script needs to run
- Running batch processes
- Running the task on the scheduled interval
- Log Rotation

Key-Features:

- **One-time Execution** : If you have a task that needs to be executed one time whether
- it's succeed or fail then the job will be finished.
- **Parallelism** : If you want to run multiple pods at the same time.
- **Scheduling** : If you want to schedule a specific number of pods after a specific time.
  Restart Policy : You can specify whether the Job should restart if fails.

Let's do some **hands-on work to get** abetter understanding of Kubernetes Jobs.

Work completed and pod deleted

```
apiVersion: batch/v1
kind: Job
metadata:
 name: testjob
spec:
 template:
   metadata:
     name: testjob
   spec:
     containers:
       -image: ubuntu
         name: container1
         command: ["bin/bash", "-c", "sudo apt update; sleep 30"]
     restartPolicy: Never
```

Create and run the pods simultaneously and delete once the work is completed.

```
apiVersion: batch/v1
kind: Job
metadata:
 name: testjob
spec:
 parallelism: 3 # Create 3 pods and run simultaneously
 activeDeadlineSeconds: 10 # Pods will terminate after 40 secs(10+30(command sleep
time))
 template:
   metadata:
     name: testjob
   spec:
     containers:
       -image: ubuntu
         name: container1
         command: ["bin/bash", "-c", "sudo apt update; sleep 30"]
     restartPolicy: Never
```

Scheduling a pod after each minute

```
apiVersion: batch/v1
kind: CronJob
metadata:
 name: testjob
spec:
 schedule: "* * * * *"
 jobTemplate:
```

```
  spec:
    template:
      spec:
        containers:
          -image: ubuntu
            name: container1
            command: ["bin/bash", "-c", "sudo apt update; sleep 30"]
        restartPolicy: Never
```

Kubernetes InitContainer

- Init Containers are those containers that run before the main or app container for the particular work.
- Init Containers motive for completion which means that the given work needs to be completed.
- If a pod fails due to Init Containers then, Kubernetes restarts the init Container until it will succeed.

Use cases:

- To Install the dependencies before running the application on the main container
- Clone a git repository into the volume
- Generate configuration files dynamically
- Database Configuration

Let's do some **hands-on** to get a better understanding of Init Containers.

In this snippet, we are creating a new pod in which we have created two containers and the first container is initcontainer.

```
apiVersion: v1
kind: Pod
metadata:
 name: initcontainer
spec:
 initContainers:
 -name: container1
   image: ubuntu
   command: ["bin/bash", "-c", "echo We are at 16 Days of 30DaysOfKubernetes >
/tmp/xchange/testfile; sleep 15"]
   volumeMounts:
     -name: xchange
       mountPath: "/tmp/xchange"
 containers:
 -name: container2
   image: ubuntu
   command: ["bin/bash", "-c", "while true; do echo `cat /tmp/data/testfile`; sleep 10; done"]
   volumeMounts:
     -name: xchange
       mountPath: /tmp/data
 volumes:
 -name: xchange
   emptyDir: {}
```

```
ubuntu@ip-172-31-63-3:~/Jobs$ sudo vim init.yml
ubuntu@ip-172-31-63-3:~/Jobs$
ubuntu@ip-172-31-63-3:~/Jobs$ kubectl apply -f init.yml --dry-run=client
pod/initcontainer created (dry run)
ubuntu@ip-172-31-63-3:~/Jobs$
ubuntu@ip-172-31-63-3:~/Jobs$ kubectl apply -f init.yml
pod/initcontainer created
ubuntu@ip-172-31-63-3:~/Jobs$
ubuntu@ip-172-31-63-3:~/Jobs$ kubectl get pods
NAME            READY   STATUS     RESTARTS   AGE
initcontainer   0/1     Init:0/1   0          7s
ubuntu@ip-172-31-63-3:~/Jobs$
ubuntu@ip-172-31-63-3:~/Jobs$ kubectl get pods
NAME            READY   STATUS     RESTARTS   AGE
initcontainer   0/1     Init:0/1   0          15s
ubuntu@ip-172-31-63-3:~/Jobs$ kubectl describe pod initcontainer
Name:             initcontainer
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.49.2
Start Time:       Tue, 17 Oct 2023 09:26:14 +0000
Labels:           <none>
Annotations:      <none>
Status:           Running
IP:               10.244.0.67
IPs:
  IP:  10.244.0.67
Init Containers:
  container1:
    Container ID:  docker://c9a509eb610e1f4e162b7d950ef8f37751200bdd1fd1238c7f461031a7ac20f6
    Image:         ubuntu
    Image ID:      docker-pullable://ubuntu@sha256:2b7412e6465c3c7fc5bb21d3e6f1917c167358449fecac8176c6e496e5c1f05f
    Port:          <none>
    Host Port:     <none>
    Command:
      bin/bash
      -c
      echo We are at 16 Days of 30DaysOfKubernetes > /tmp/xchange/testfile; sleep 15
    State:          Terminated
      Reason:       Completed
      Exit Code:    0
      Started:      Tue, 17 Oct 2023 09:26:15 +0000
      Finished:     Tue, 17 Oct 2023 09:26:30 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /tmp/xchange from xchange (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-dzknz (ro)
Containers:
```

In this snippet, you will see the initcontainer is configured and then the main application container is running.

```
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  36s   default-scheduler  Successfully assigned default/initcontainer to minikube
  Normal  Pulling    35s   kubelet            Pulling image "ubuntu"
  Normal  Pulled     35s   kubelet            Successfully pulled image "ubuntu" in 160.622359ms (160.646389ms including waiting)
  Normal  Created    35s   kubelet            Created container container1
  Normal  Started    35s   kubelet            Started container container1
  Normal  Pulling    19s   kubelet            Pulling image "ubuntu"
  Normal  Pulled     19s   kubelet            Successfully pulled image "ubuntu" in 190.310697ms (190.332802ms including waiting)
  Normal  Created    19s   kubelet            Created container container2
  Normal  Started    19s   kubelet            Started container container2
ubuntu@ip-172-31-63-3:~/Jobs$
ubuntu@ip-172-31-63-3:~/Jobs$ kubectl get pods
NAME            READY   STATUS    RESTARTS   AGE
initcontainer   1/1     Running   0          46s
ubuntu@ip-172-31-63-3:~/Jobs$
ubuntu@ip-172-31-63-3:~/Jobs$ kubectl logs -f pod/initcontainer
Defaulted container "container2" out of: container2, container1 (init)
We are at 16 Days of 30DaysOfKubernetes
We are at 16 Days of 30DaysOfKubernetes
We are at 16 Days of 30DaysOfKubernetes
We are at 16 Days of 30DaysOfKubernetes
We are at 16 Days of 30DaysOfKubernetes
We are at 16 Days of 30DaysOfKubernetes
We are at 16 Days of 30DaysOfKubernetes
We are at 16 Days of 30DaysOfKubernetes
We are at 16 Days of 30DaysOfKubernetes
^C
ubuntu@ip-172-31-63-3:~/Jobs$
ubuntu@ip-172-31-63-3:~/Jobs$ kubectl exec -it initcontainer -c container2 -- cat /tmp/data/testfile
We are at 16 Days of 30DaysOfKubernetes
ubuntu@ip-172-31-63-3:~/Jobs$
```

Kubernetes Pod Lifecycle

There are multiple types of states for the Pod lifecycle that we will discuss here:

1. **Pending**: When a pod is created it has to go through the pending status in which Master nodes allocate the nodes to where to create the pod. The pods will remain in the pending state until all the necessary resources are allocated such as CPU, memory, and storage.
2. **Running** : Once the pod has been scheduled to a node, it comes into the Running Status. Once the pod comes into a running state, the containers within the pods will be creating and doing the tasks that have been provided in the manifest file.
3. **Succeeded** : Once the pods have completed their task then, the pods come in the Succeeded state and then terminates.
4. **Failed** : Once the pods intend to create but due to some issues the pods are not creating and showing the Failed state leads to issues with the configurations which need to be addressed by the creator of the file.
5. **CrashLoopBackOff** : This is the advanced state of a Failed state where the container is crashing and restarts. To fix this issue, the creator of the file needs to check the manifest file.
6. **Unknown** : In some cases, the Kubernetes may lose the connection with the nodes to create the pods that show the unknown status of the particular pod.
7. **Termination** : When a pod is no longer available it comes in the termination process. Once the pod is deleted, it can not restart again the same pods and is removed from the entire Kubernetes cluster.

There are some conditions that come under while creating Pods:

- **Initialized**: This condition shows whether all the init containers have started successfully
- or not. If the status is false it means the init containers have not started.
- **Ready**: This condition shows the pod is ready to use.
- **ContainersReady**: As the name suggests, if the containers are ready within a pod it will show True in the status.
  PodScheduled: This condition shows that the pod has been scheduled on the node.

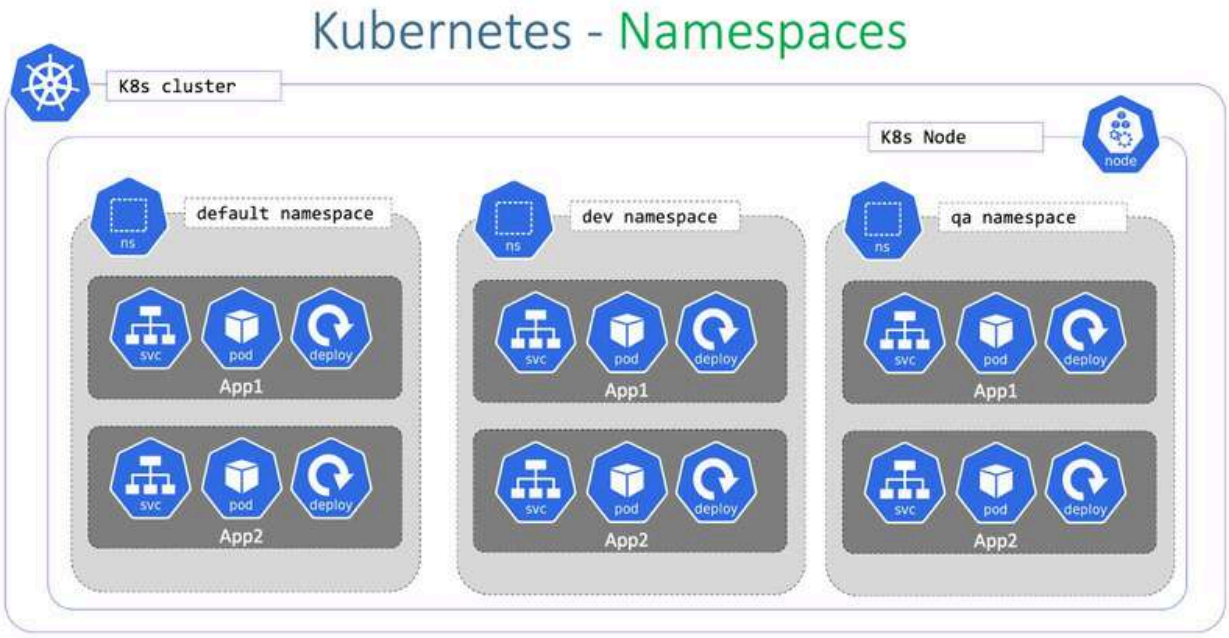
If you create any pod and describe that pod by running the command kubectl describe pod <pod-name>. You will see the status like this, the pod is scheduled but other things are not because it is in progress.

```
Conditions:
  Type              Status
  Initialized       False
  Ready             False
  ContainersReady   False
  PodScheduled      True
```

After some seconds, if you describe again the same pod. You will see that everything is perfect and configured properly.

```
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
```

Kubernetes Namespace



About Namespace

- Anamespace is a logical entity that is used to organize the Kubernetes Cluster into
- virtual sub-clusters.
- The namespace is used when an organization shares the same Kubernetes cluster for
- multiple projects.
  There can be any number of namespaces can be created inside the Kubernetes cluster. Nodes and Kubernetes Volumes do not come under the namespaces and are visible to every namespace.

Pre-existed Kubernetes namespaces

- **default** : As the name suggests, whenever we create any Kubernetes object such as pod, replicas, etc it will create in the default namespace. If you want to create the particular pod in the different namespace you have to create the namespace and while creating the pod you have to mention the namespace(Will do in handsOn).
- **kube-system** : This namespace contains the Kubernetes components such as kube-controller-manager, kube-scheduler, kube-dns or other controllers. This

namespace must be avoided to create pods or other objects if you want to keep the Kubernetes cluster stable.

- **kube-public** : This namespace is used to share non-sensitive information that can be viewed by any of the members who are part of the Kubernetes cluster.

When should we consider Kubernetes Namespaces?

- **Isolation**: When there are multiple numbers of projects running then we can consider
- making namespaces and put the projects accordingly in the namespaces.
  Organization: If there is only one Kubernetes Cluster which has different environments to keep isolated. If something happens to the particular environment then it won't affect the other environments.
- **Permission** : If some objects are confidential and must need access to the particular persons then, Kubernetes provides RBAC roles as well which we can use in the namespace. It means that only authorized users can access the objects within the namespaces.

HandsOn

If you observe the first command, while running the command 'kubectl get pods' you are getting 'No resources found in default namespace' which means that we are trying to list the pods from the default namespace.

If you observe the second command 'kubectl get namespaces' which is used to list all the namespaces.

The default one is created for pods and others for Kubernetes itself which we don't use for ourselves.

```
ubuntu@ip-172-31-63-3:~$ kubectl get pods
No resources found in default namespace.
ubuntu@ip-172-31-63-3:~$
ubuntu@ip-172-31-63-3:~$
ubuntu@ip-172-31-63-3:~$ kubect get namespaces
kubect: command not found
ubuntu@ip-172-31-63-3:~$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   2d23h
kube-node-lease   Active   2d23h
kube-public       Active   2d23h
kube-system       Active   2d23h
ubuntu@ip-172-31-63-3:~$
```

In this snippet, we have created one new namespace named tech-mahindra and you can validate whether the namespace is created or not by running 'kubectl get namespaces' command.

```
apiVersion: v1
kind: Namespace
metadata:
  name: tech-mahindra
  labels:
    name: tech-mahindra
```

```
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$ sudo vim  new-ns.yml
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$ kubectl apply -f new-ns.yml
namespace/tech-mahindra created
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   2d23h
kube-node-lease   Active   2d23h
kube-public       Active   2d23h
kube-system       Active   2d23h
tech-mahindra     Active   9s
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$
```

In this snippet, we have created one pod in a namespace that we created in the previous step(tech-mahindra). If you want to list the pods from the other namespace except the default namespace then you have to mention the namespace name by following -n <namespace-name>.

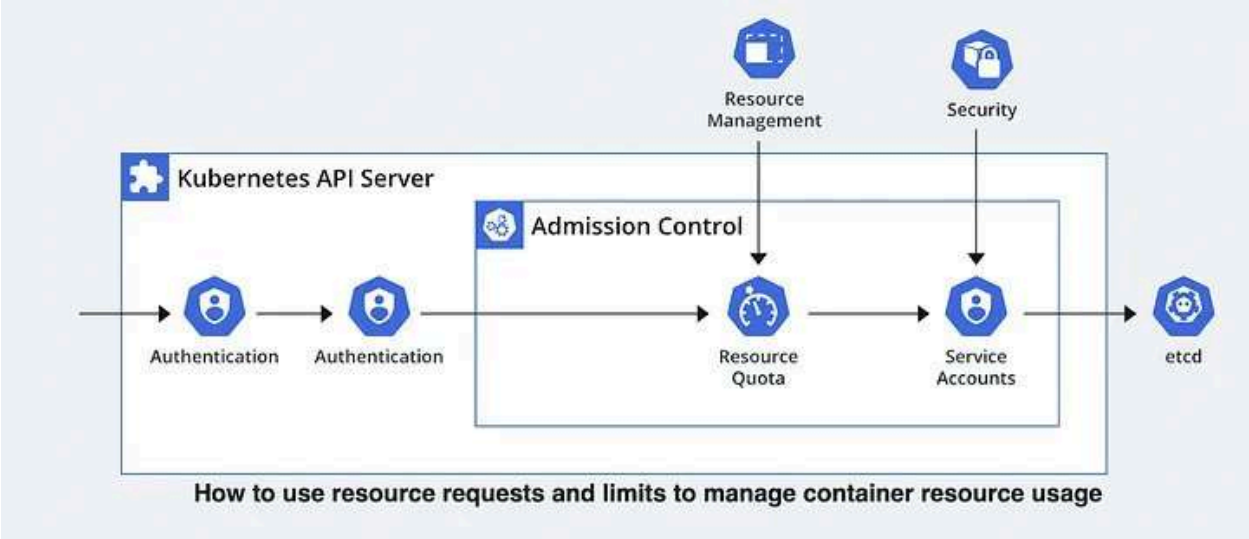
```
apiVersion: v1
kind: Pod
metadata:
  name: ns-demo-pod
spec:
  containers:
  -name: container1
    image: ubuntu
    command: ["bin/bash", "-c", "while true; do echo We are on 18th Day of
30DaysOfKubernetes; sleep 30; done"]
  restartPolicy: Never
```

```
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$ sudo vim ns-pod.yml
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$ kubectl apply -f ns-pod.yml -n tech-mahindra
pod/ns-demo-pod created
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$ kubectl get pods
No resources found in default namespace.
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$ kubectl get pods -n tech-mahindra
NAME          READY   STATUS    RESTARTS   AGE
ns-demo-pod   1/1     Running   0          16s
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$
```

If you want to delete the pod from the different namespace except default then you have to mention the namespace otherwise it will throw the error.

If you want to set your namespace as the default namespace, you can use the command ‘kubectl config set-context $(kubectl config current-context) — namespace <namespace-name>’’ and if you want to see the default namespace use the command ‘kubectl config view | grep namespace’

Kubernetes ResourceQuota



How to use resource requests and limits to manage container resource usage

ResourceQuota is one of the rich features of Kubernetes that helps to manage and distribute resources according to the requirements.

Suppose in an Organization, two teams share the same Kubernetes Cluster where the first team needs more CPUs and Memory because of heavy workload tasks. Now, in this situation, if the CPUs and other resources go to another team then it might increase the chance of failure. To resolve this issue, we can allocate particular CPU cores and Memory to every project.

- Apod in Kubernetes will run with no limits on CPU and memory by default.
- You can specify the RAM, Memory, or CPUs for each container and pod.
- The scheduler decides which node will create pods, if the node has enough CPU
- resources available then, the node will place the pods.
- CPU is specified in units of cores and memory is specified in units of bytes.
  As you know, the Kubernetes cluster can be divided into namespaces and if a container is created in a namespace that has a default CPU limit and container does not specify the CPU limit then the container will have the default CPI limit.
- Anamespace can be assigned to ResourceQuota objects, this will help to limit the amount of usage to the objects within the namespaces. You can limit the computer (CPU), Memory, and Storage.
- Restrictions that a resource-quotas imposes on namespaces

- Every container that is running on the namespace must have its own CPU limit.
- The total amount of CPU used by all the containers in the namespace should not exceed aspecified limit.

There are two types of constraints that need to be mentioned while using ResourceQuota:

- **Limit**: Limit specifies that the container, pod, or namespace will have the limit resources
- where if the objects will exceed the limit then, the object won't create.
  Request: The request specifies that the container, pod, or namespace needs a particular amount of resources such as CPU and memory. But if the request is greater than the limit then, Kubernetes won't allow the creation of pods or containers.

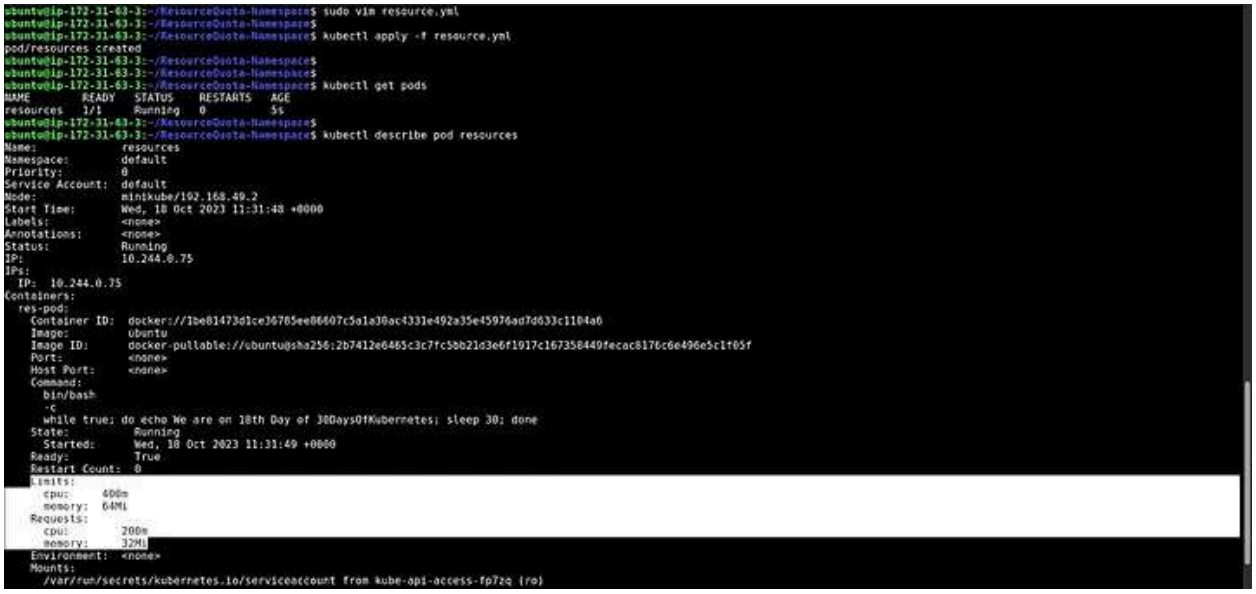
Now, there are some conditions or principles for requests and limit which needs to be understood.

Let's understand with hands-on and theoretical.

1. If the requests and limits are given in the manifest file, it works accordingly.
2. If the requests are given but the limit is not provided then, the default limit will be used.
3. If the requests are not provided but the limit is provided then, the requests will be equal to the limit.

When booth requests and limits have been mentioned for a pod then it will create the pod according to the provided resources

```
apiVersion: v1
kind: Pod
metadata:
  name: resources
spec:
  containers:
  -image: ubuntu
  name: res-pod
  command: ["bin/bash", "-c", "while true; do echo We are on 18th Day of
30DaysOfKubernetes; sleep 30; done"]
     resources
      requests:
        memory: "32Mi"
        cpu: "200m"
      limits:
        memory: "64Mi"
        cpu: "400m"
```



Creating ResourceQuota

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: res-quota
spec:
  hard:
    limits.cpu: "200m"
    requests.cpu: "150m"
    limits.memory: "38Mi"
    requests.memory: "12Mi"
```

Creating deployment but the pod is not creating due to out of limit

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rq-deployments
spec:
  replicas: 4
  selector:
    matchLabels:
      objtype: rq-deployments
  template:
```

```
  metadata:
    name: rq-pod
    labels:
      objtype: rq-deployments
  spec:
    containers:
    -name: rq-cont
      image: ubuntu
      command: ["bin/bash", "-c", "while true; do echo We are on 18th Day of
30DaysOfKubernetes; sleep 30; done"]
       resources
        requests:
          cpu: "50m"
```

```
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$ sudo vim rq-pod.yml
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$ kubectl apply -f rq-pod.yml
deployment.apps/rq-deployments created
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$ kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
resources   1/1     Running   0          10m
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$ kubectl get deploy
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
rq-deployments   0/4     0            0           16s
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$ kubectl get rs
NAME                       DESIRED   CURRENT   READY   AGE
rq-deployments-549c9455b5   4         0         0       22s
ubuntu@ip-172-31-63-3:~/ResourceQuota-Namespace$ kubectl describe rs rq-deployments-549c9455b5
Name:           rq-deployments-549c9455b5
Namespace:      default
Selector:       objtype=rq-deployments,pod-template-hash=549c9455b5
Labels:         objtype=rq-deployments
                pod-template-hash=549c9455b5
Annotations:    deployment.kubernetes.io/desired-replicas: 4
                deployment.kubernetes.io/max-replicas: 5
                deployment.kubernetes.io/revision: 1
Controlled By:  Deployment/rq-deployments
Replicas:       0 current / 4 desired
Pods Status:    0 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  objtype=rq-deployments
           pod-template-hash=549c9455b5
  Containers:
   rq-cont:
    Image:      ubuntu
    Port:       <none>
    Host Port:  <none>
    Command:
      bin/bash
      -c
      while true; do echo We are on 18th Day of 30DaysOfKubernetes; sleep 30; done
    Requests:
      cpu:        50m
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type             Status  Reason
```

Error logs for the above work

```
  Type     Reason        Age              From                   Message
  ----     ------        ----             ----                   -------
  Warning  FailedCreate  30s              replicaset-controller  Error creating: pods "rq-deployments-549c9455b5-gscwn" is forbidden: failed quota: res-quota: must specify limits.cpu for: r
q-cont; limits.memory for: rq-cont; requests.memory for: rq-cont
  Warning  FailedCreate  30s              replicaset-controller  Error creating: pods "rq-deployments-549c9455b5-ls8sh" is forbidden: failed quota: res-quota: must specify limits.cpu for: r
q-cont; limits.memory for: rq-cont; requests.memory for: rq-cont
  Warning  FailedCreate  30s              replicaset-controller  Error creating: pods "rq-deployments-549c9455b5-9vvwb" is forbidden: failed quota: res-quota: must specify limits.cpu for: r
q-cont; limits.memory for: rq-cont; requests.memory for: rq-cont
  Warning  FailedCreate  30s              replicaset-controller  Error creating: pods "rq-deployments-549c9455b5-dklrj" is forbidden: failed quota: res-quota: must specify limits.cpu for: r
q-cont; limits.memory for: rq-cont; requests.memory for: rq-cont
  Warning  FailedCreate  30s              replicaset-controller  Error creating: pods "rq-deployments-549c9455b5-45dp2" is forbidden: failed quota: res-quota: must specify limits.cpu for: r
q-cont; limits.memory for: rq-cont; requests.memory for: rq-cont
  Warning  FailedCreate  30s              replicaset-controller  Error creating: pods "rq-deployments-549c9455b5-l98pq" is forbidden: failed quota: res-quota: must specify limits.cpu for: r
q-cont; limits.memory for: rq-cont; requests.memory for: rq-cont
  Warning  FailedCreate  29s              replicaset-controller  Error creating: pods "rq-deployments-549c9455b5-jhqmb" is forbidden: failed quota: res-quota: must specify limits.cpu for: r
q-cont; limits.memory for: rq-cont; requests.memory for: rq-cont
  Warning  FailedCreate  29s              replicaset-controller  Error creating: pods "rq-deployments-549c9455b5-8ckdv" is forbidden: failed quota: res-quota: must specify limits.cpu for: r
q-cont; limits.memory for: rq-cont; requests.memory for: rq-cont
  Warning  FailedCreate  28s              replicaset-controller  Error creating: pods "rq-deployments-549c9455b5-vjbkj" is forbidden: failed quota: res-quota: must specify limits.cpu for: r
q-cont; limits.memory for: rq-cont; requests.memory for: rq-cont
  Warning  FailedCreate  5s (x4 over 27s)  replicaset-controller  (combined from similar events): Error creating: pods "rq-deployments-549c9455b5-dvbqn" is forbidden: failed quota: res-quota
```

When I did not provide requests

In this snippet, we have created a limit range where we have set the default value for CPU as 1 and from that 1 cpu we are requesting 0.5 CPU only to create a container, if the container needs more than 0.5 CPU then due to the limit range it won't happen

```
apiVersion: v1
```

```
kind: LimitRange
metadata:
  name: cpulimitrange
spec:
  limits:
  default:
  cpu: 1
    defaultRequest:
      cpu: 0.5
    type: Container
```

When I did not provide the limit


```
apiVersion: v1
kind: Pod
metadata:
  name: no-request-demo
spec:
  containers:
  -name: container1
    image: ubuntu
    resources:
      limits:
        cpu: "2"
```

When I did not provide limits


```
apiVersion: v1
kind: Pod
metadata:
  name: default-cpu-demo-3
spec:
  containers:
  -name: default-cpu-demo-3-ctr
    image:  ngin
    resources:
    requests:
       cpu: "0.75"
```

Kubernetes AutoScaling

Kubernetes supports autoscaling. If you don't know about autoscaling, let me explain you in a simple way. As you know, to run the application we need CPU and memory. Sometimes, there will be a chance where the CPU gets loaded, and this might fail the server or affect the application. Now, we can't afford the downtime of the applications. To get rid of these, we need to increase the number of servers or increase the capacity of servers.

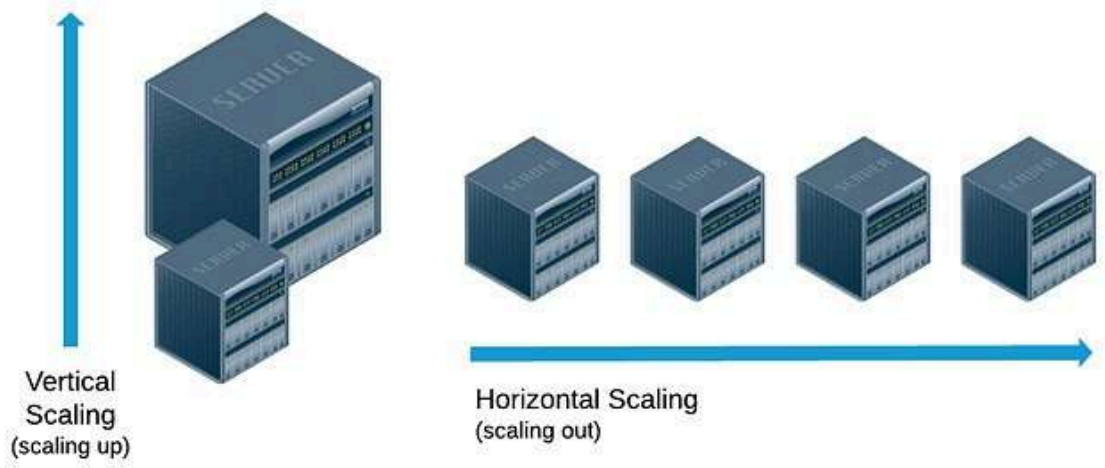
Let's understand with a real-time example

There are some OTT platforms such as Netflix or Hotstar. If any web show or movie is coming on the platform the audience is eagerly waiting for that. Then, the OTT platform can't handle the lot of users that might crash the application. This will lead to a loss of business and the OTT platform can't afford this business loss. Now, they have two options to solve this.

- First, The Platform knows that they need a particular amount of servers such as 100. So, they can buy those servers forever but in this situation, when the load decreases then the other servers will become unused. Now, if the server is unused, still they have paid for those servers which is not a cost-effective method. Second, The Platform
- doesn't know when the load will increase. So, they have one option which is autoscaling in which when the CPU utilization crosses a particular number, it creates new servers. So, the platform can handle loads easily which is very cost effective as well.

Types of Autoscaling

- **Horizontal Pod AutoScaling** : In this type of scaling, the number of servers will increase according to CPU utilization. In this, you define the minimum number of servers, maximum number of servers, and CPU utilization. If the CPU utilization crosses more than 50% then, it will add the one server automatically.
- **Vertical Pod AutoScaling** : In this type of scaling, the server will remain the same in numbers but the server's configuration will increase such as from 4GB RAM to 8GM RAM, and the same with the other configurations. But this is not cost-effective and business-effective. So, we use Horizontal Pod AutoScaling.



Key Features of Horizontal Pod AutoScaler

- By default, Kubernetes does not provide AutoScaling. If you need AutoScaling, then you have to create hpa(Horizontal Pod AutoScaling) and vpa(Vertical Pod AutoScaling) objects.
- Kubernetes supports the autoscaling of the pods based on Observed CPU
- Utilization.
- Scaling only supports Scalable objects like Controller, deployment, or ReplicaSet.
- HPA is implemented as a Kubernetes API resource and a controller.
  The Controller periodically adjusts the number of replicas in a replication controller or deployment to match the observed average CPU utilization to the target specified in the manifest file or command.

We will not cover Vertical Scaling, but you must be aware of it. To get a better understanding of VPA. Refer this:
Kubernetes VPA

Let's do some hands-on for Horizontal Pod AutoScaler.

We can perform HPA through two types:
- **Imperative** way: In this way, you will create hpa object by command only.
- **Declarative** way: In this way, you will create a proper manifest file and then create an object by applying the manifest file.

HandsOn

To perform HPA, we need one Kubernetes component which is a metrics server. To download this, use the below command.

```
curl -LO https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

```
ubuntu@ip-172-31-63-3:~$ curl -LO https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100  4186  100  4186    0     0  12614      0 --:--:-- --:--:-- --:--:-- 12614
ubuntu@ip-172-31-63-3:~$
ubuntu@ip-172-31-63-3:~$
ubuntu@ip-172-31-63-3:~$ ls | grep comp
components.yaml
ubuntu@ip-172-31-63-3:~$
```

Now, edit the components.yml file
add
hostNetwork: true
under spec: line and,
add
---kubelet-insecure-tls
under the metric-resolutions line, then save the file.

```
131
132     spec:
133       hostNetwork: true
134       containers:
135       - args:
136         - --cert-dir=/tmp
137         - --secure-port=4443
138         - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
139         - --kubelet-use-node-status-port
140         - --metric-resolution=15s
141         - --kubelet-insecure-tls
142         image: registry.k8s.io/metrics-server/metrics-server:v0.6.4
143         imagePullPolicy: IfNotPresent
```

Now, you have to apply this by the below command
kubectl apply -f components.yml
To validate, run the command below
kubectl get pods -n kube-system
If the metrics-server pod is running then, you are ready to scale.

```
ubuntu@ip-172-31-63-3:~$ kubectl get pods -n kube-system
NAME                               READY   STATUS    RESTARTS        AGE
coredns-5d78c9869d-rd7dr           1/1     Running   4 (3d20h ago)   7d2h
etcd-minikube                      1/1     Running   4 (3d20h ago)   7d2h
kube-apiserver-minikube            1/1     Running   4 (3d20h ago)   7d2h
kube-controller-manager-minikube   1/1     Running   4 (3d20h ago)   7d2h
kube-proxy-r8jfx                   1/1     Running   4 (3d20h ago)   7d2h
kube-scheduler-minikube            1/1     Running   4 (3d20h ago)   7d2h
metrics-server-6b4694c468-rpxrk    1/1     Running   0               61s
storage-provisioner                1/1     Running   9 (7m29s ago)   7d2h
ubuntu@ip-172-31-63-3:~$
```

Through Imperative way
Create the deployment by the below file

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: thdeploy
spec:
  replicas: 1
  selector:
   matchLabels:
    name: deployment
  template:
    metadata:
      name: testpod8
      labels:
        name: deployment
    spec:
     containers:
       -name: container1
         image: nginx
         ports:
         -containerPort: 80
```

```
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
```

Apply the deployment file

```
kubectl apply -f <file_name.yml>
```

Now, I have opened two windows to show you whether the pod is scaling or not.
Run the watch ‘kubectl get all’ command to see the pod creation.
On the other window, we are creating an hpa object through command.

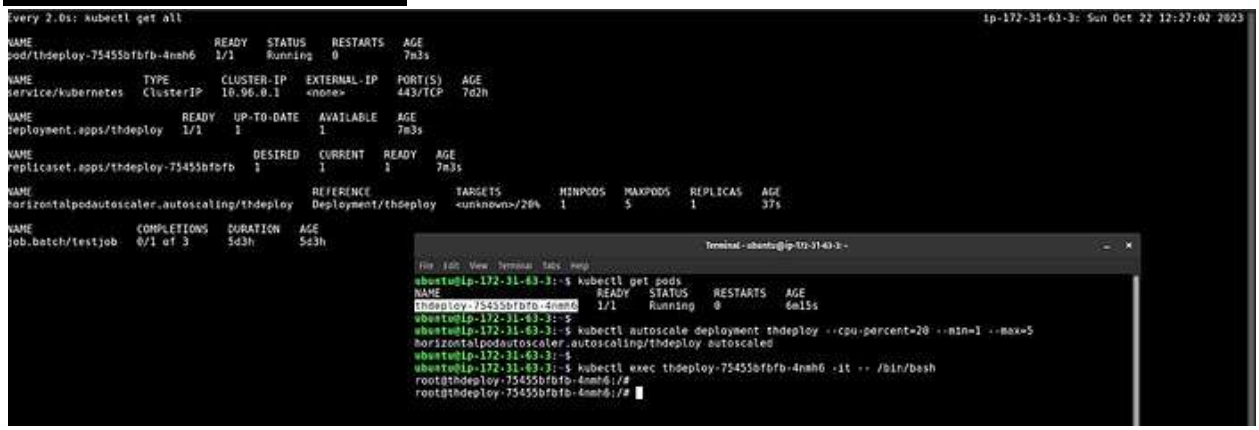
```
min=1 — max=5
```

Now on the other window, go into the container using the below command:

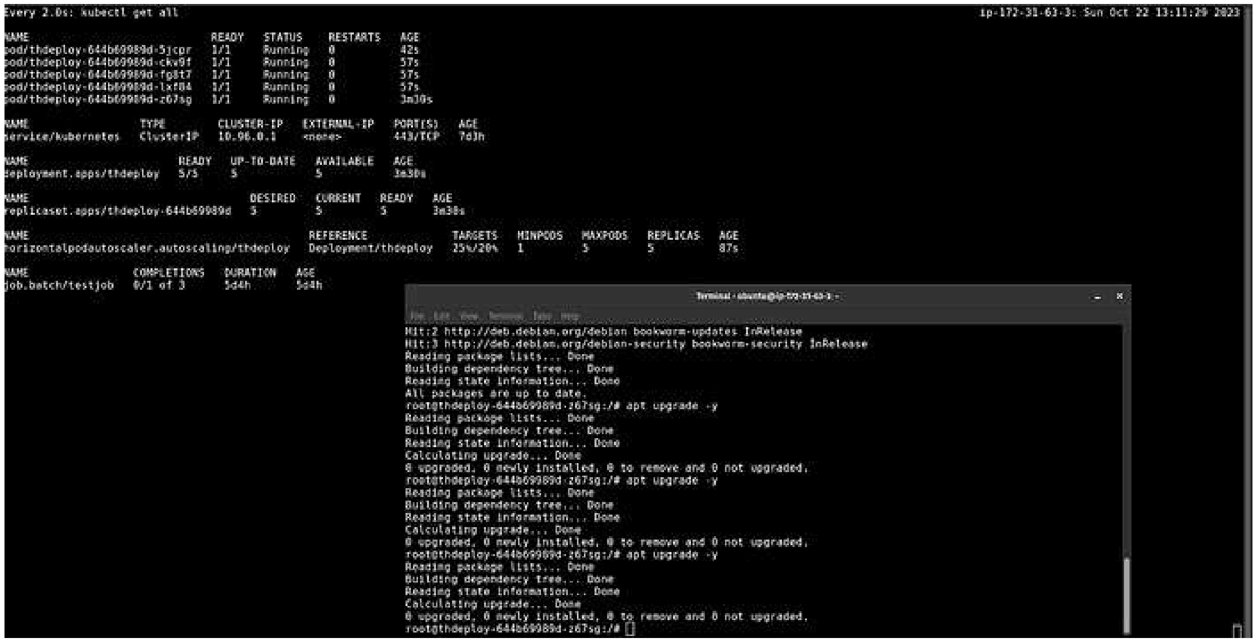
```
kubectl exec <po
```

Now run the below command inside the container and see the magic.

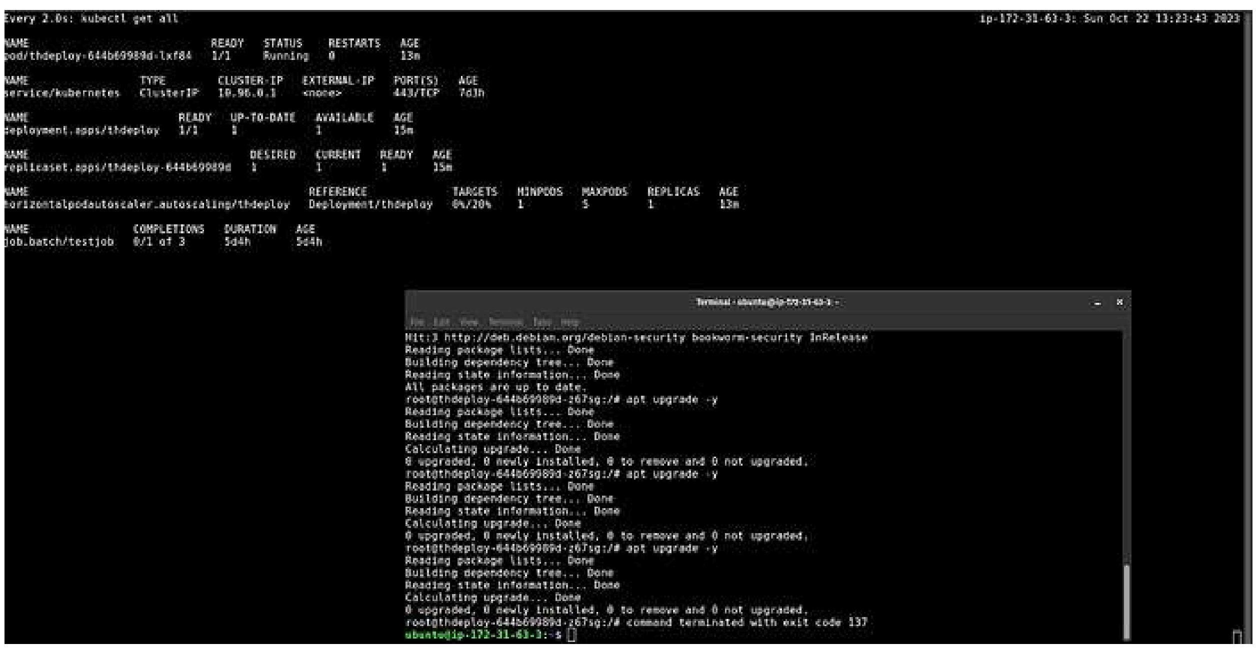
```
while true; do apt update; done
```



In this scenario, you can observe that the pod count has increased, reaching its maximum value of 5. If you intend to downscale, simply halt the command currently executing within the container. Afterward, Kubernetes will automatically delete the pods within approximately 5 minutes. This delay is designed to allow Kubernetes to assess the system load, if the load surges once more, Kubernetes will need to scale up the number of pods. This phase is commonly referred to as the “cooldown period.



Here, you can see the pod number again comes to 1 because there is no load on the container.



Through declarative way

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
 name: demo-hpa
spec:
 scaleTargetRef:
```

```
  apiVersion: apps/v1
  kind: Deployment
  name: thdeploy
minReplicas: 1
maxReplicas: 5
metrics:
  -type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 20
```

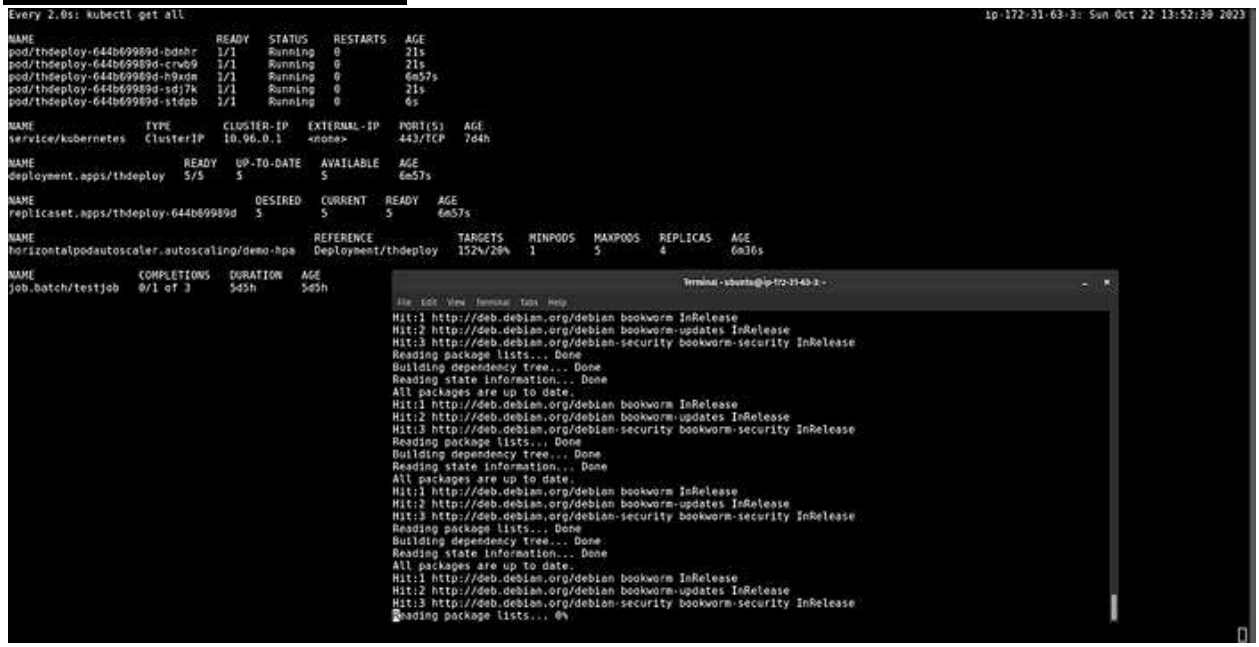
Create a file for the hpa object and copy the above content in the file.

```
kubectl apply -f <hpa.yml>
```

The hpa object has been created. Now, you can perform the above steps to increase the load on the container to see the pod scaling.
I have opened two windows to show you whether the pod is scaling or not.
Run the watch ‘kubectl get all’ command to see the pod creation.
Now on the other window, go into the container using the below command:

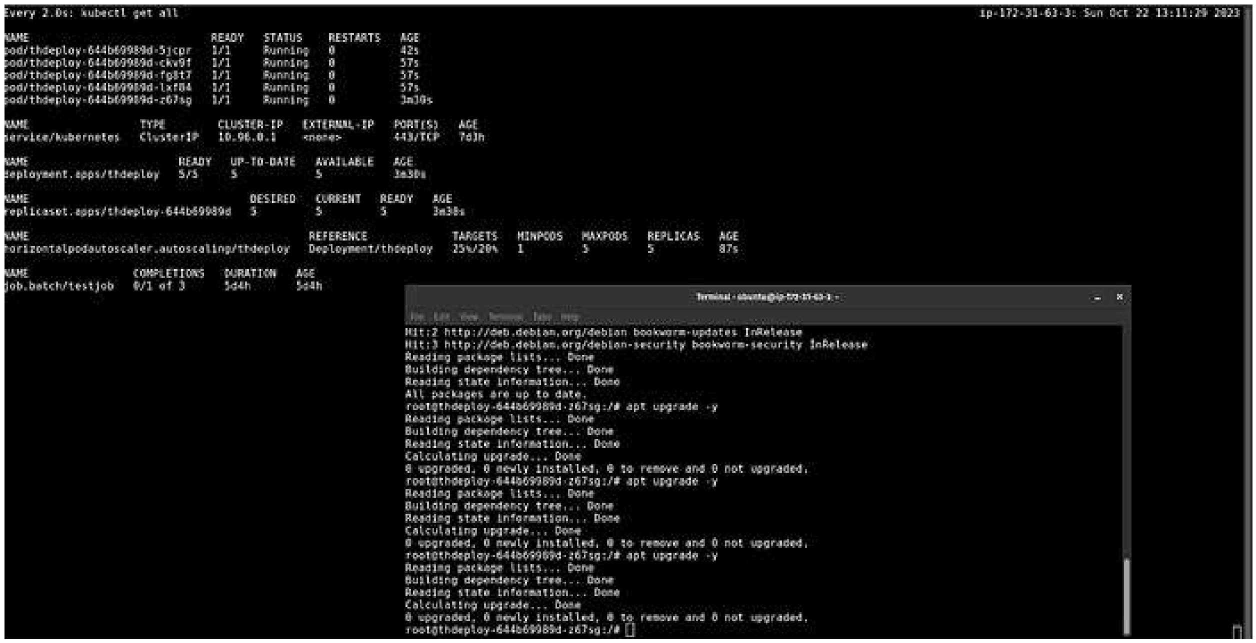
Now run the below command inside the container and see the magic.

```
while true; do apt update; done
```

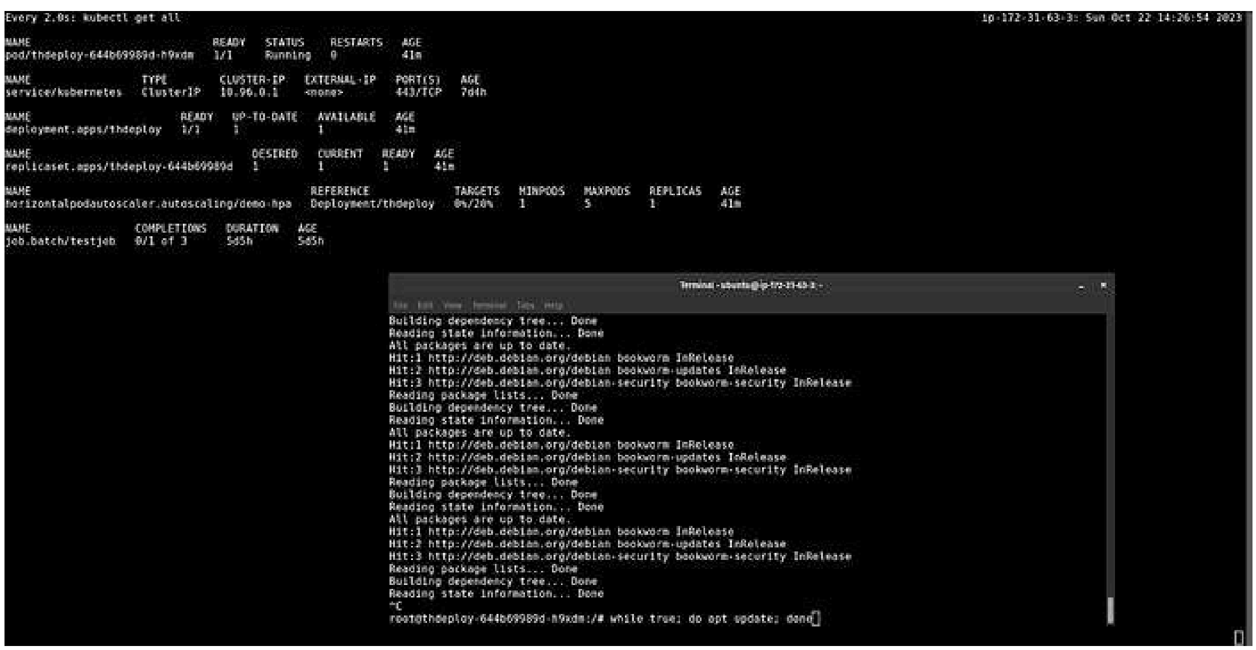


In this scenario, you can observe that the pod count has increased, reaching its maximum value of 5. If you intend to downscale, simply halt the command currently executing within the container. Afterward, Kubernetes will automatically delete the pods within approximately 5 minutes. This delay is designed to allow Kubernetes to assess

the system load, if the load surges once more, Kubernetes will need to scale up the number of pods. This phase is commonly referred to as the “cooldown period.



Here, you can see the pod number again comes to 1 because there is no load on the container.



Multi-Cluster Kubernetes with HAProxy

Why Multi Cluster

Multi-cluster Kubernetes setups are beneficial for various reasons, including high availability, disaster recovery, and geographical distribution. Having multiple clusters can ensure that if one cluster fails, your application remains available in other clusters. It also helps distribute workloads geographically, improving latency for users in different regions.

HAproxy

HAProxy is used as a load balancer to distribute traffic across multiple Kubernetes clusters. It plays a crucial role in maintaining high availability by redirecting traffic to available clusters. In the provided setup, it acts as an entry point, routing requests to the appropriate Kubernetes cluster.

I have added the details for all Five servers. So, you will be able to understand the high overview of all the servers

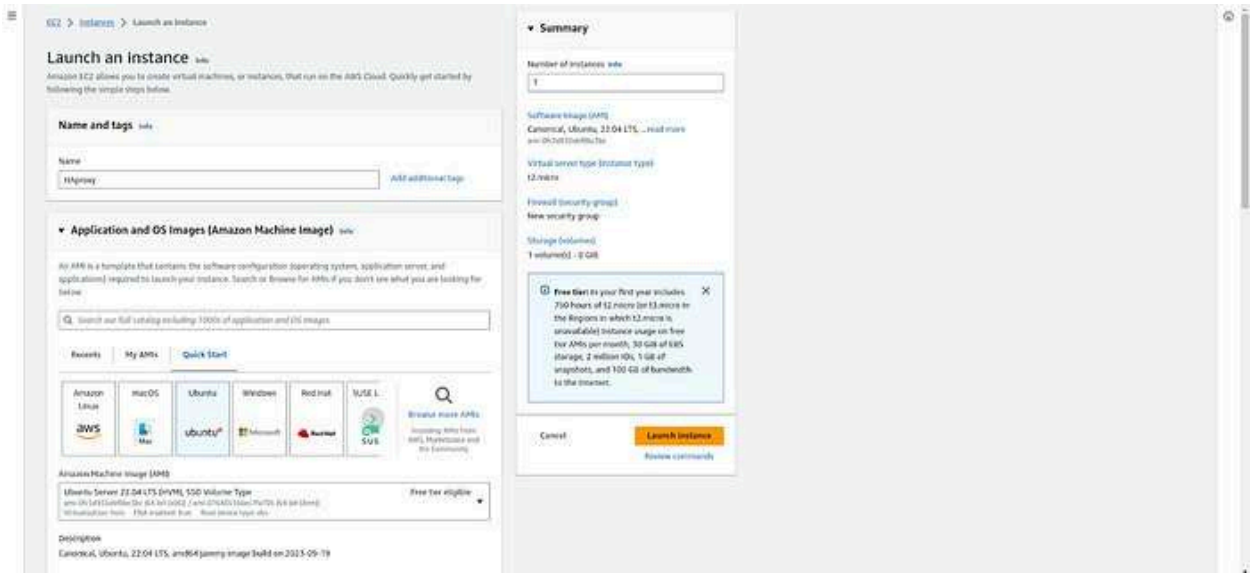
Role	HostName	IP	OS	RAM	CPU
HAproxy	lb.node.com	172.31.22.132	Ubuntu 22.04	1GB	1
Master	k8master1.node.com	172.31.23.243	Ubuntu 22.04	4GB	2
Master	k8master2.node.com	172.31.28.74	Ubuntu 22.04	4GB	2
Worker	k8worker1.node.com	172.31.31.111	Ubuntu 22.04	1GB	1
Worker	k8worker2.node.com	172.31.22.133	Ubuntu 22.04	1GB	1

We have to set up five servers where two will be Master Nodes, the other two will be Worker Nodes, and the last one HAproxy.

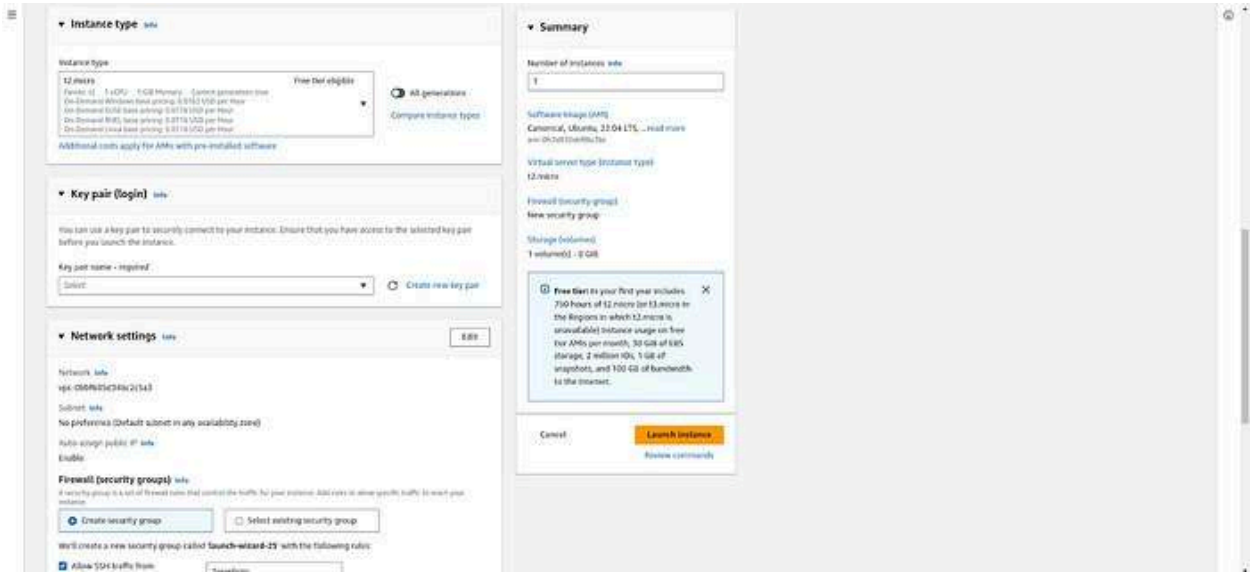
Create HAproxy Server(EC2 Instance)

Click on **Launch instances**

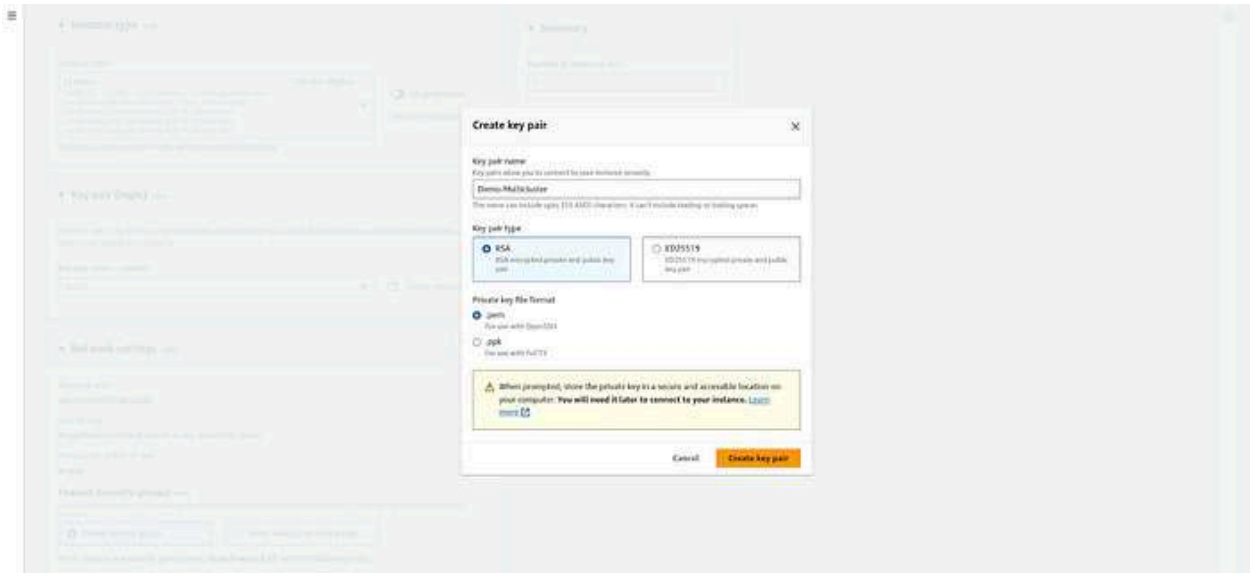
Enter the name of the instance and select Ubuntu 22.04(must).



The instance type will be t2.micro and click on Create new key pair for this demo.

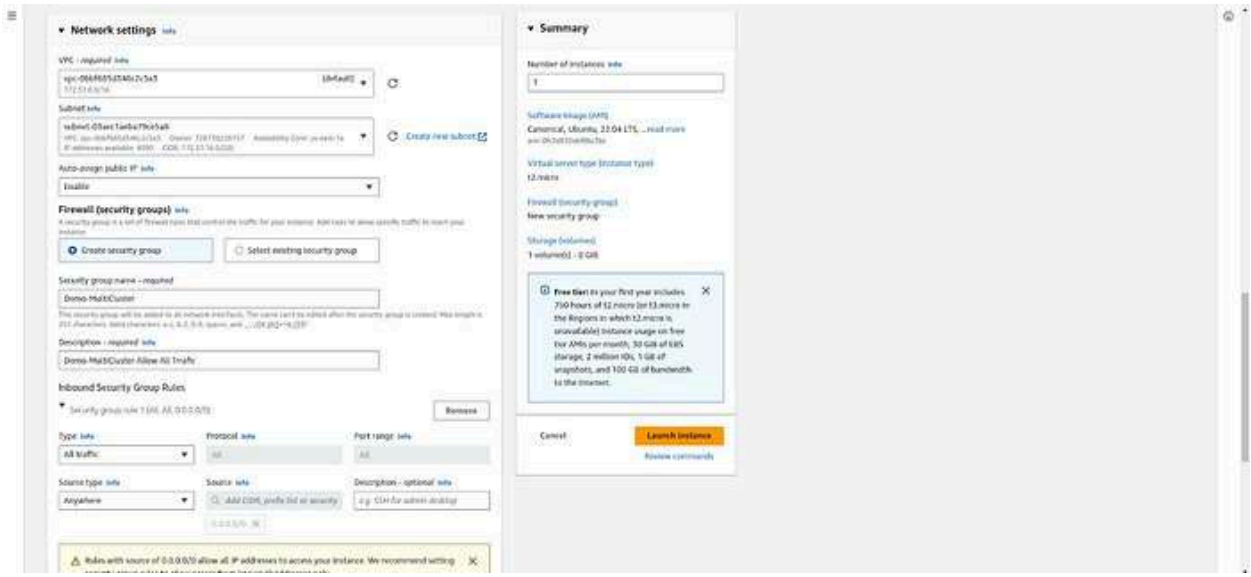


Enter the name, keep the things as it is, and click on Create key pair.



Select the default vpc and select the subnet from the availability zone us-east-1a and create the new security group with All Traffic type where the Source type will be Anywhere.

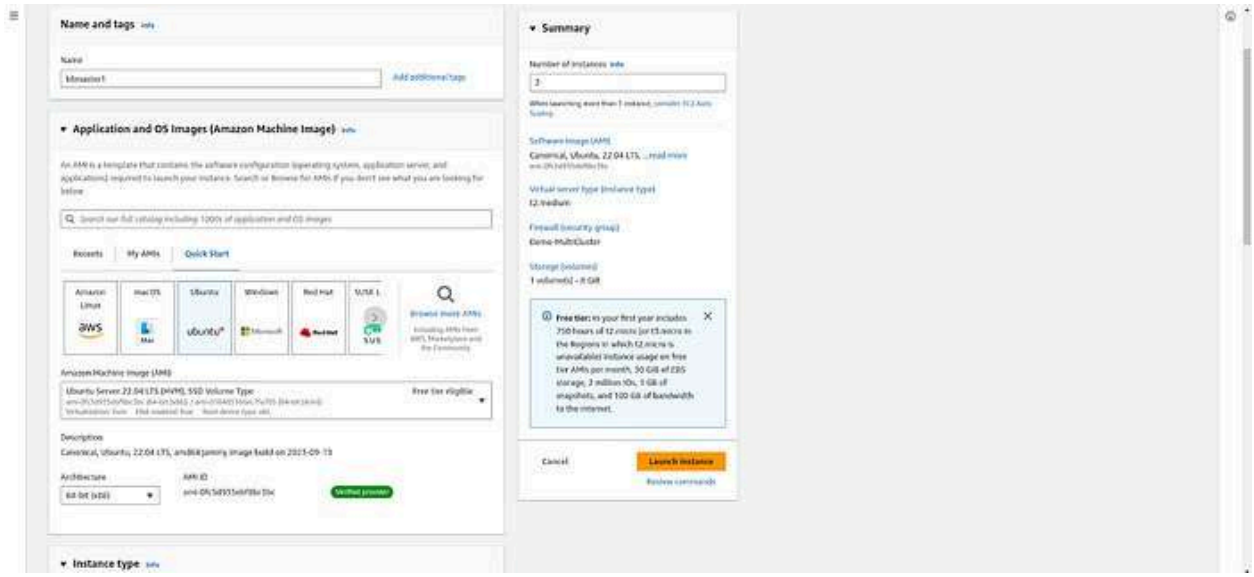
Here we have configured everything for HAproxy, So click on Launch Instance.



Creating Master Nodes(EC2 Instance)

Here, we have to set up the two Master Nodes.

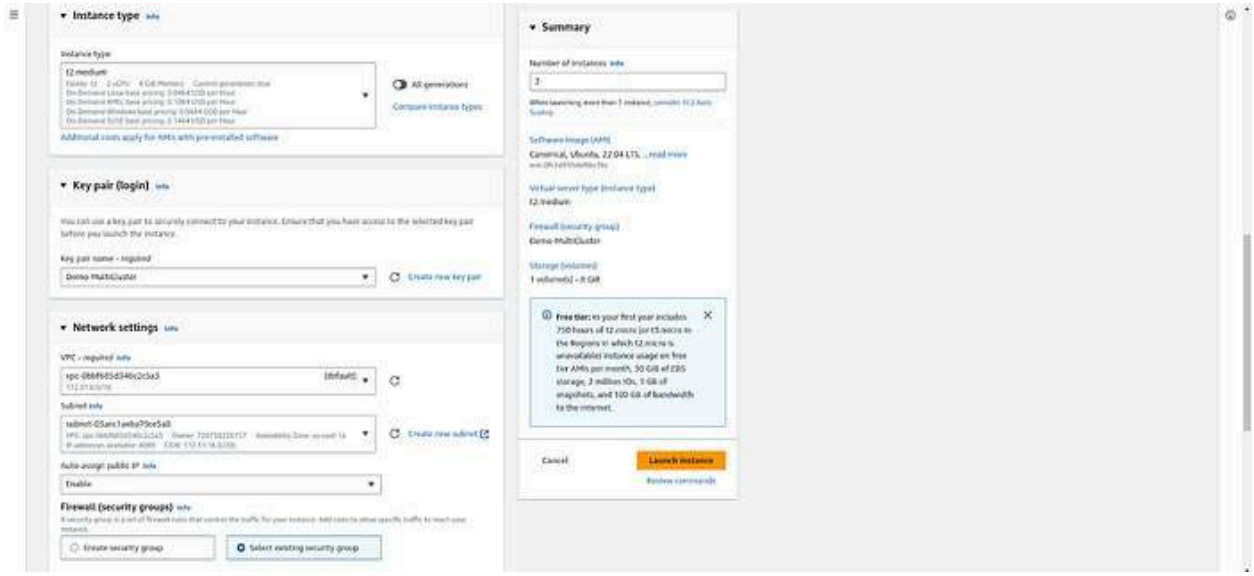
Enter the name of the instance and select Ubuntu 22.04(must) and in the right number of instances will be 2. So that, we will save us some time.



The master node needs 2CPU that will get in the t2.medium instance type.

Provide the same key that we have provided for the HAproxy server.

Select the same VPC and same Subnet that we have provided for the HAproxy server.

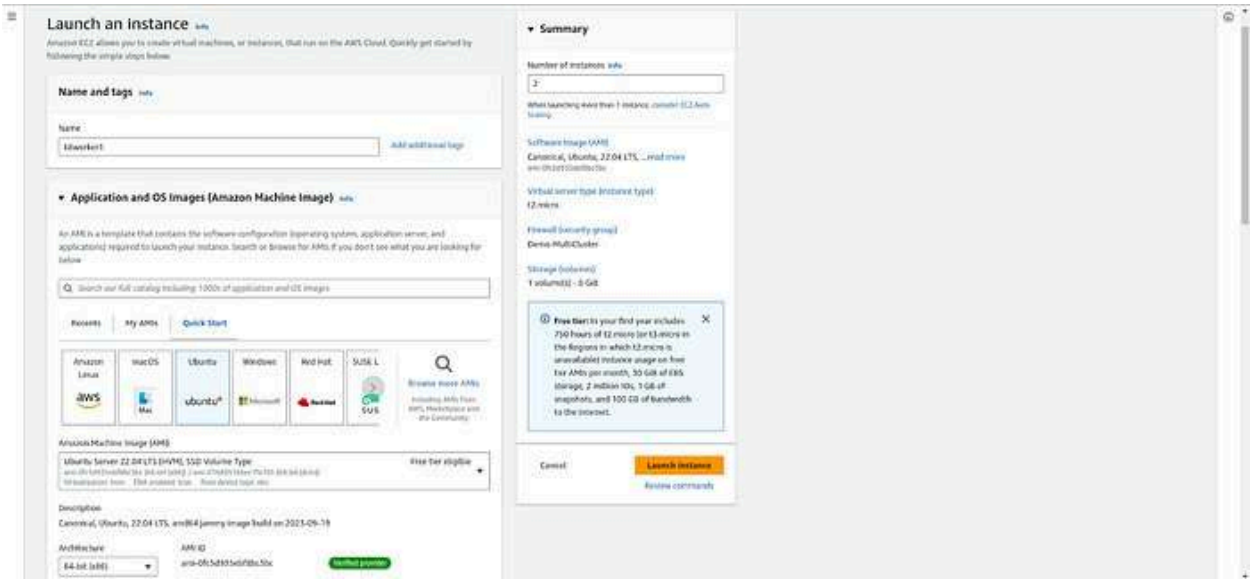


Select the same Security Group that we have created for the HAproxy server.

Creating Worker Nodes(EC2 Instance)

Here, we have to set up the two Worker Nodes.

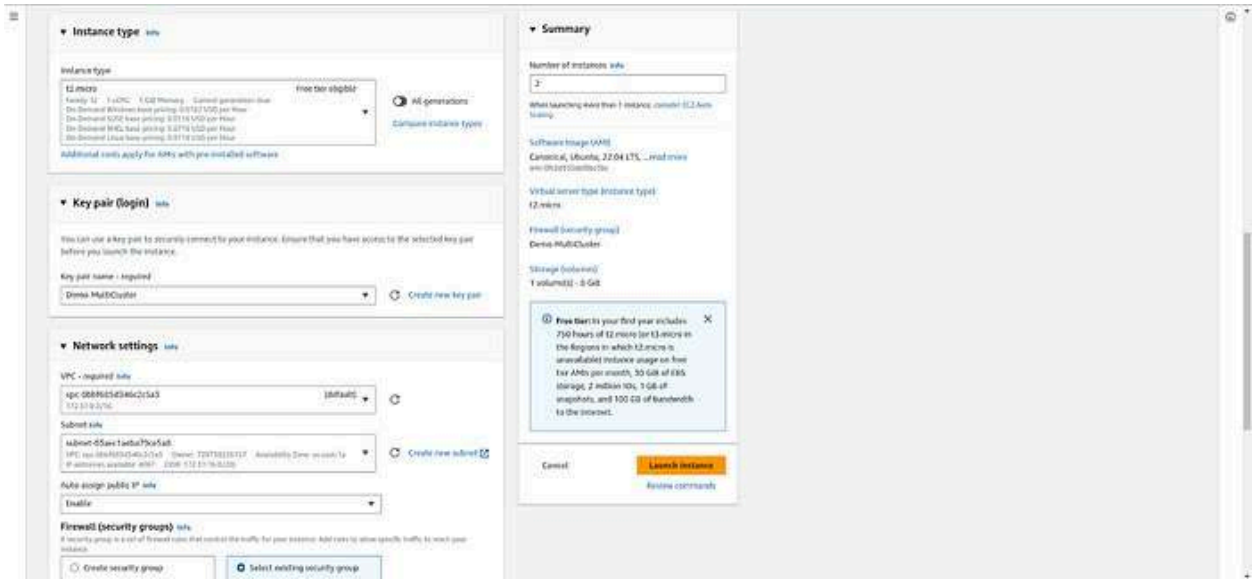
Enter the name of the instance and select Ubuntu 22.04(must) and in the right number of instances will be 2. So that, we will save us some time.



The master node doesn't need 2CPU that's why the instance type will be the same t2.micro

Provide the same key that we have provided for the HAproxy server.

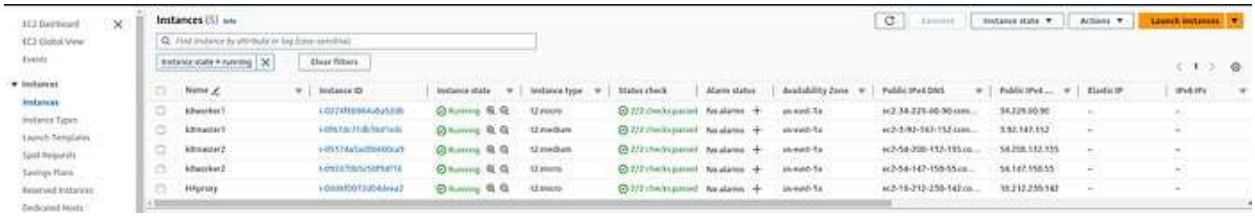
Select the same VPC and same Subnet that we have provided for the HAproxy server.



Select the same Security Group that we have created for the HAproxy server.

Now, both the master and worker nodes' names will be the same. So, you can modify the name of each by masternode1 and masternode2, the same for the worker node.

This is the total of five servers that we have created.



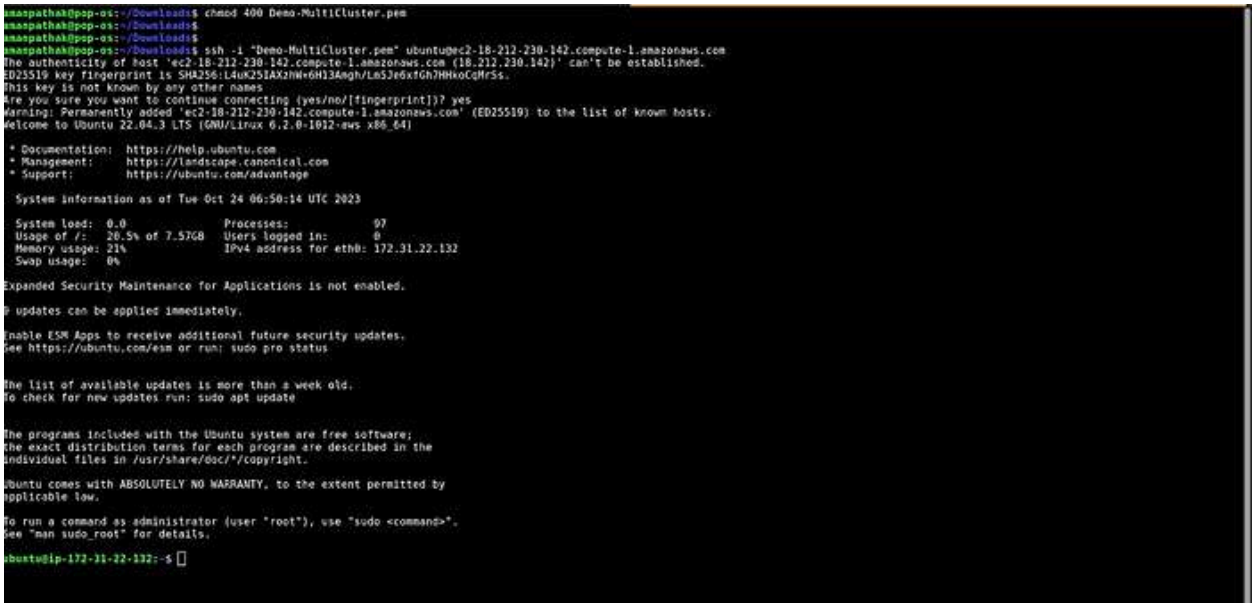
Now, we have to do the configurations in all the servers. Let's do this and start with the HAproxy server.

On HAproxy Server

Before doing SSH, modify the permission of the PEM file that we will use to do SSH.

```
sudo su
chmod 400 Demo-MultiCluster.pem
```

Now, use the command to SSH into the HAproxy server.



To become a root user run the below command

```
sudo su
```

Now, update the package and install haproxy which will help us to set our Kubernetes multi-cluster

```
apt update && apt install -y haproxy
```

Here, we have to set the backend and frontend to set up Kubernetes Multi-Cluster.

Open the file haproxy.cfg and add the code snippets according to your Private IPs

```
vim /etc/haproxy/haproxy.cfg
```

Remember, in frontend block HAproxy Private IP needs to be present.

In the backend block, Both Master Node IP needs to be present.

```
frontend kubernetes-frontend
  bind 172.31.22.132:6443
  mode tcp
  option tcplog
  default_backend kubernetes-backend

backend kubernetes-backend
  mode tcp
  option tcp-check
  balance roundrobin
  server kmaster1 172.31.23.243:6443 check fall 3 rise 2
  server kmaster2 172.31.28.74:6443 check fall 3 rise 2
```

```
frontend kubernetes-frontend
    bind 172.31.22.132:6443
    mode tcp
    option tcplog
    default_backend kubernetes-backend

backend kubernetes-backend
    mode tcp
    option tcp-check
    balance roundrobin
    server kmaster1 172.31.23.243:6443 check fall 3 rise 2
    server kmaster2 172.31.28.74:6443 check fall 3 rise 2
```

Once you add the frontend and backend, restart the haproxy service
`systemctl restart haproxy`

Now, check the status of whether the haproxy service is running or not

`systemctl status haproxy`

If you look at some bottom lines, the kmaster1 and kmaster2 nodes are down which is correct. But this indicates that the frontend and backend code are reflected.

```
root@ip-172-31-22-132:/home/ubuntu# sudo vim /etc/haproxy/haproxy.cfg
root@ip-172-31-22-132:/home/ubuntu#
root@ip-172-31-22-132:/home/ubuntu# systemctl restart haproxy
root@ip-172-31-22-132:/home/ubuntu#
root@ip-172-31-22-132:/home/ubuntu#
root@ip-172-31-22-132:/home/ubuntu# systemctl status haproxy
● haproxy.service - HAProxy Load Balancer
     Loaded: loaded (/lib/systemd/system/haproxy.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2023-10-24 06:55:44 UTC; 7s ago
       Docs: man:haproxy(1)
             file:/usr/share/doc/haproxy/configuration.txt.gz
    Process: 2013 ExecStartPre=/usr/sbin/haproxy -Ws -f $CONFIG -c -q $EXTRAOPTS (code=exited, status=0/SUCCESS)
   Main PID: 2015 (haproxy)
      Tasks: 2 (limit: 1121)
     Memory: 71.1M
        CPU: 109ms
     CGroup: /system.slice/haproxy.service
             ├─2015 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -S /run/haproxy-master.sock
             └─2017 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -S /run/haproxy-master.sock

Oct 24 06:55:44 ip-172-31-22-132 systemd[1]: Starting HAProxy Load Balancer...
Oct 24 06:55:44 ip-172-31-22-132 haproxy[2015]: [NOTICE]   (2015) : New worker #1 (2017) forked
Oct 24 06:55:44 ip-172-31-22-132 systemd[1]: Started HAProxy Load Balancer.
Oct 24 06:55:45 ip-172-31-22-132 haproxy[2017]: [WARNING]  (2017) : Server kubernetes-backend/kmaster1 is DOWN, reason: Layer4 connection problem, info: "Connection refused at initial conne>
Oct 24 06:55:45 ip-172-31-22-132 haproxy[2017]: [WARNING]  (2017) : Server kubernetes-backend/kmaster2 is DOWN, reason: Layer4 connection problem, info: "Connection refused at initial conne>
Oct 24 06:55:45 ip-172-31-22-132 haproxy[2017]: [NOTICE]   (2017) : haproxy version is 2.4.22-0ubuntu0.22.04.1
Oct 24 06:55:45 ip-172-31-22-132 haproxy[2017]: [NOTICE]   (2017) : path to executable is /usr/sbin/haproxy
Oct 24 06:55:45 ip-172-31-22-132 haproxy[2017]: [ALERT]    (2017) : backend 'kubernetes-backend' has no server available!
lines 1-22/22 (END)
```

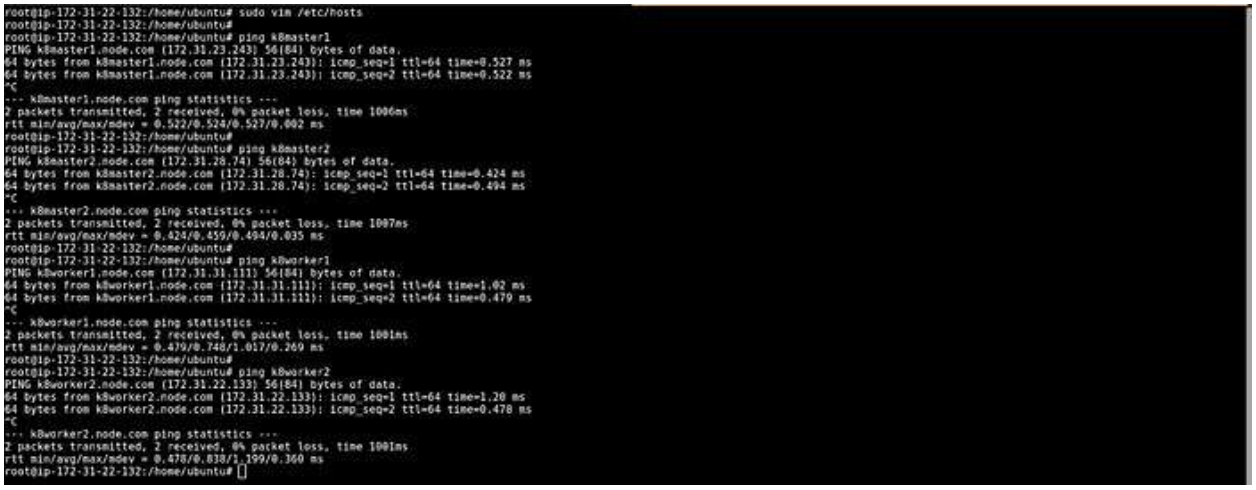
Now, add the hostnames in the /etc/hosts files with all five servers' Private IPs like below

`vim /etc/hosts`

```
172.31.23.243 k8master1.node.com node.com k8master1
172.31.28.74 k8master2.node.com node.com k8master2
172.31.31.111 k8worker1.node.com node.com k8worker1
172.31.22.133 k8worker2.node.com node.com k8worker2
172.31.22.132 lb.node.com node.com lb
```

```
127.0.0.1 localhost

172.31.23.243 k8master1.node.com node.com k8master1
172.31.28.74 k8master2.node.com node.com k8master2
172.31.31.111 k8worker1.node.com node.com k8worker1
172.31.22.133 k8worker2.node.com node.com k8worker2
172.31.22.132 lb.node.com node.com lb

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

Now, try to ping all four servers(Master+Worker) from HAproxy. If your machine is receiving the packets then we are good to go for the next step which is configuring the Master Nodes



On Master Nodes

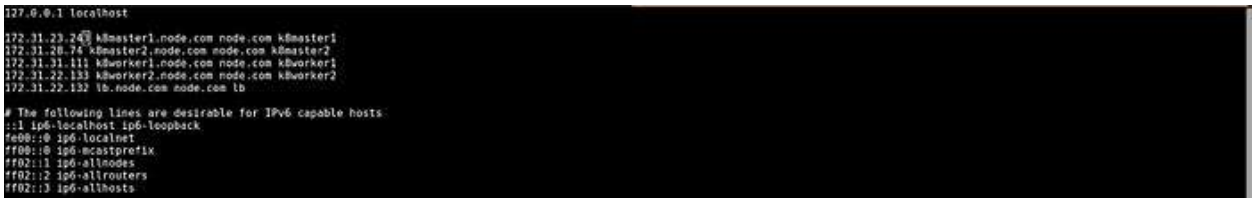
I have provided you the snippets of One Master Nodes only. But I configured both Master nodes. So, make sure to configure each and everything simultaneously on both Master Nodes.

Login to your both Master Nodes

Once you log into both machines, run the command that is necessary for both Master Nodes

Now, add the hostnames in the /etc/hosts files with all five servers' Private IPs like below

```
172.31.31.111 k8worker1.node.com node.com k8worker1
172.31.22.133 k8worker2.node.com node.com k8worker2
172.31.22.132 lb.node.com node.com lb
```



After closing the host's file, run the below commands.

```
sudo su
ufw disable
reboot
```

Now, log in again to your both machines after 2 to 3 minutes.

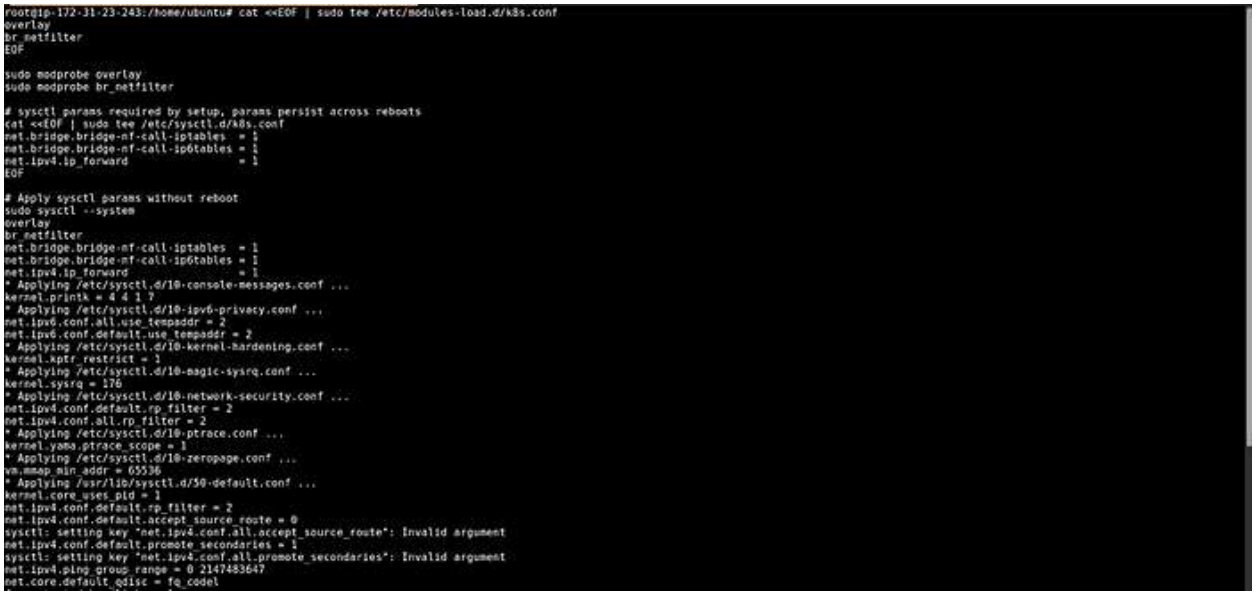
Run the below commands

```
sudo su
swapoff -a; sed -i '/swap/d' /etc/fstab
```

```
ubuntu@ip-172-31-23-243:~$ sudo su
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu# swapoff -a; sed -i '/swap/d' /etc/fstab
root@ip-172-31-23-243:/home/ubuntu#
```

Run the below commands

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
sudo modprobe overlay
sudo modprobe br_netfilter
#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
#Apply sysctl params without reboot
sudo sysctl - system
```



Install some dependencies packages and add the Kubernetes package

```
sudo apt-get install -y apt-transport-https ca-certificates curl
curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg - dearmor -o
/etc/apt/keyrings/kubernetes-archive-keyring.gpg
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg]
https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list
```

```
root@ip-172-31-23-243:/home/ubuntu# sudo apt-get install -y apt-transport-https ca-certificates curl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'apt' instead of 'apt-transport-https'
apt is already the newest version (2.4.10).
apt set to manually installed.
ca-certificates is already the newest version (20230311ubuntu0.22.04.1).
ca-certificates set to manually installed.
curl is already the newest version (7.81.0-1ubuntu1.13).
curl set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu# curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-archive-keyring.gpg
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu# echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list
.d/kubernetes.list
deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu#
```

As we have added the gpg keys, we need to run the update command
`apt update`

```
root@ip-172-31-23-243:/home/ubuntu# apt update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:5 https://packages.cloud.google.com/apt kubernetes-xenial InRelease [8993 B]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [14.1 MB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe Translation-en [5652 kB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 c-n-f Metadata [286 kB]
Get:9 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [217 kB]
Get:10 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse Translation-en [112 kB]
Get:11 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 c-n-f Metadata [8372 B]
Get:12 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1103 kB]
Get:13 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [896 kB]
Get:14 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [239 kB]
Get:15 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 c-n-f Metadata [16.0 kB]
Get:16 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [1036 kB]
Get:17 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted Translation-en [167 kB]
Get:18 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 c-n-f Metadata [536 B]
Get:19 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [995 kB]
Get:20 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe Translation-en [218 kB]
Get:21 http://security.ubuntu.com/ubuntu jammy-security/main Translation-en [180 kB]
Get:22 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 c-n-f Metadata [22.0 kB]
Get:23 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 Packages [41.6 kB]
Get:24 http://security.ubuntu.com/ubuntu jammy-security/main amd64 c-n-f Metadata [11.4 kB]
Get:25 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [1016 kB]
```

Now, we have to install docker on our both master nodes
`apt install docker.io -y`

```
root@ip-172-31-23-243:/home/ubuntu# apt install docker.io -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io pigz runc ubuntu-fan
0 upgraded, 8 newly installed, 0 to remove and 31 not upgraded.
Need to get 69.7 MB of archives.
After this operation, 267 MB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 pigz amd64 2.6-1 [63.6 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 bridge-utils amd64 1.7-1ubuntu3 [34.4 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 runc amd64 1.1.7-0ubuntu1~22.04.1 [4249 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 containerd amd64 1.7.2-0ubuntu1~22.04.1 [36.0 MB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 dns-root-data all 2021011101 [5256 B]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 dnsmasq-base amd64 2.86-1.1ubuntu0.3 [354 kB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 docker.io amd64 24.0.5-0ubuntu1~22.04.1 [28.9 MB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 ubuntu-fan all 0.12.16 [35.2 kB]
Fetched 69.7 MB in 2s (37.6 MB/s)
Preconfiguring packages ...
Selecting previously unselected package pigz.
(Reading database ... 64726 files and directories currently installed.)
Preparing to unpack .../0-pigz_2.6-1_amd64.deb ...
```

Do some configurations for containerd service
`mkdir /etc/containerd`
`sh -c "containerd config default > /etc/containerd/config.toml"`
`sed -i 's/ SystemdCgroup = false/ SystemdCgroup = true/' /etc/containerd/config.toml`
`systemctl restart containerd.service`

```
root@ip-172-31-23-243:/home/ubuntu# mkdir /etc/containerd
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu# sh -c "containerd config default > /etc/containerd/config.toml"
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu# sed -i 's/ SystemdCgroup = false/ SystemdCgroup = true/' /etc/containerd/config.toml
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu# systemctl restart containerd.service
root@ip-172-31-23-243:/home/ubuntu#
```

Now, we will install our kubelet, kubeadm, and kubectl services on the Master node
`apt-get install -y kubelet kubeadm kubectl kubernetes-cni`

Now, restart the kubelet service, and don't forget to enable the kubelet service. So that, if any master node will reboot then we don't need to start the kubelet service.

```
sudo systemctl restart kubelet.service
sudo systemctl enable kubelet.service
```

Only on Master Node1

This command must need to run on Master Node1

We have to init the kubeadm and provide the endpoint which will be the haproxy server Private IP and in the end provide the Master Node1 IP only.

```
kubeadm init — control-plane-endpoint="<hap-private-ip:6443" — upload-certs —
apiserver-advertise-address=<master-node1-private-ip>
```

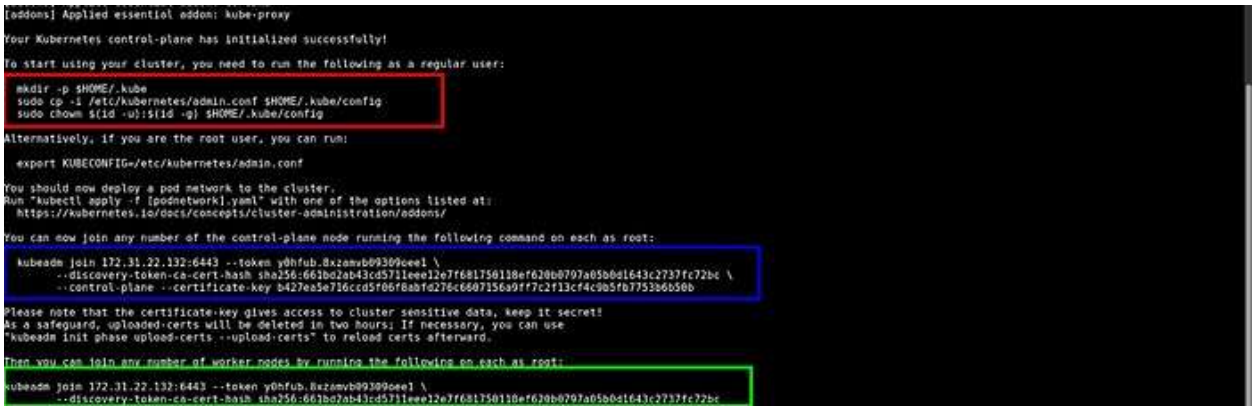
```
apiserver-advertise-address=172.31.28.74
```

Once you run the above command, scroll down.

Once you scroll down, you will see Kubernetes control plan has initialized successfully which means Master node1 joined the HAproxy server. Now, we have to initialize Kubernetes Master Node2 as well. Follow the below steps:

- Copy the Red highlighted commands, Blue highlighted commands and Green
- highlighted commands and paste them into your notepad.
  Now, run the Red highlighted commands on the Master node1 itself.

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

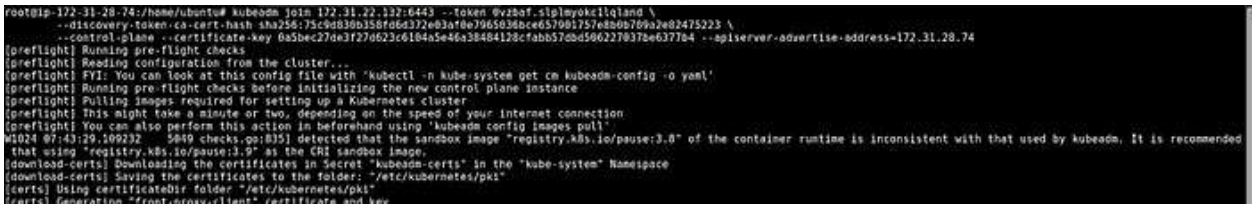


On Master Node2

Now, we need to do one more thing so that Master Node2 joins the HAproxy server as well. Follow the steps:

- We have to use the Blue highlighted command, but we need to add one more thing with the command, refer to the below(add —apiserver-advertise-address=<PrivateIP-of-MasterNode2>

```
kubeadm join 172.31.22.132:6443 - token 0vzbaf.slplmyokc1lqland \
-discovery-token-ca-cert-hash
sha256:75c9d830b358fd6d372e03af0e7965036bce657901757e8b0b789a2e82475223 \
-control-plane - certificate-key
0a5bec27de3f27d623c6104a5e46a38484128cfabb57dbd5062270370e6377b4 -
apiserver-advertise-address=172.31.28.74
```



Once you followed the above steps, you can run the below commands on Master Node2

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

Now, if you run the command ‘kubectl get nodes’ on Master Node1 to see the nodes. You will get both nodes but they are not in ready status because we did not configure the network. We will configure that once the Worker Nodes are configured.

Note: Copy the Green highlighted command, which we will use to connect with Worker Nodes

Now, if you run the command ‘kubectl get nodes’ on Master Node2 to see the nodes. You will get both nodes but they are not in ready status because we did not configure the network. We will configure that once the Worker Nodes are configured.

Note: Copy the Green highlighted command, which we will use to connect with Worker Nodes

On Both Worker Nodes

Now, Let’s configure our Worker Nodes.

I have provided you the snippets of One Master Nodes only. But I configured both Worker nodes. So, make sure to configure each and everything simultaneously on both Worker Nodes.

Login to your both Worker Nodes

Once you log into your both machines, run the command that is necessary for both Master Nodes

Now, add the hostnames in the /etc/hosts files with all five servers' Private IPs like below

```
172.31.23.243
172.31.28.74 k8master2.node.com node.com k8master2
172.31.31.111 k8worker1.node.com node.com k8worker1
172.31.22.133 k8worker2.node.com node.com k8worker2
172.31.22.132 lb.node.com node.com lb
```

After closing the host's file, run the below commands.

```
sudo su
ufw disable
```

reboot

Now, log in again to both machines after 2 to 3 minutes.

Run the below commands
sudo su
swapoff -a; sed -i '/swap/d' /etc/fstab

Run the below commands:
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
sudo modprobe overlay
sudo modprobe br_netfilter
#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
#Apply sysctl params without reboot
sudo sysctl - system

Install some dependencies packages and add the Kubernetes package
sudo apt-get install -y apt-transport-https ca-certificates curl
curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg - dearmor -o /etc/apt/keyrings/kubernetes-archive-keyring.gpg
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

```
root@ip-172-31-22-133:/home/ubuntu# sudo apt-get install -y apt-transport-https ca-certificates curl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'apt' instead of 'apt-transport-https'
apt is already the newest version (2.4.10).
apt set to manually installed.
ca-certificates is already the newest version (20230311ubuntu0.22.04.1).
ca-certificates set to manually installed.
curl is already the newest version (7.81.0-1ubuntu1.13).
curl set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@ip-172-31-22-133:/home/ubuntu#
root@ip-172-31-22-133:/home/ubuntu#
root@ip-172-31-22-133:/home/ubuntu# curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-archive-keyring.gpg
root@ip-172-31-22-133:/home/ubuntu#
root@ip-172-31-22-133:/home/ubuntu#
root@ip-172-31-22-133:/home/ubuntu# echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list
.d/kubernetes.list
deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main
root@ip-172-31-22-133:/home/ubuntu#
```

As we have added the gpg keys, we need to run the update command
`apt update`

```
root@ip-172-31-22-133:/home/ubuntu# apt update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:5 https://packages.cloud.google.com/apt kubernetes-xenial InRelease [8993 B]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [14.1 MB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe Translation-en [5652 kB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 c-n-f Metadata [286 kB]
Get:9 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [217 kB]
Get:10 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse Translation-en [112 kB]
Get:11 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 c-n-f Metadata [8372 B]
Get:12 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1103 kB]
Get:13 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [239 kB]
Get:14 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 c-n-f Metadata [16.0 kB]
Get:15 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [1036 kB]
Get:16 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted Translation-en [167 kB]
Get:17 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 c-n-f Metadata [536 B]
Get:18 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [995 kB]
Get:19 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe Translation-en [218 kB]
Get:20 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 c-n-f Metadata [22.0 kB]
Get:21 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 Packages [41.6 kB]
```

Now, we have to install docker on our both worker nodes

`apt install docker.io -y`

```
root@ip-172-31-22-133:/home/ubuntu# apt install docker.io -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io pigz runc ubuntu-fan
0 upgraded, 8 newly installed, 0 to remove and 31 not upgraded.
Need to get 69.7 MB of archives.
After this operation, 267 MB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 pigz amd64 2.6-1 [63.6 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 bridge-utils amd64 1.7-1ubuntu3 [34.4 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 runc amd64 1.1.7-0ubuntu1~22.04.1 [4249 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 containerd amd64 1.7.2-0ubuntu1~22.04.1 [36.0 MB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 dns-root-data all 2021011101 [5256 B]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 dnsmasq-base amd64 2.86-1.1ubuntu0.3 [354 kB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 docker.io amd64 24.0.5-0ubuntu1~22.04.1 [28.9 MB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 ubuntu-fan all 0.12.16 [35.2 kB]
Fetched 69.7 MB in 2s (41.2 MB/s)
```

Do some configurations for containerd service
`mkdir /etc/containerd`
`sh -c "containerd config default > /etc/containerd/config.toml"`
`sed -i 's/ SystemdCgroup = false/ SystemdCgroup = true/' /etc/containerd/config.toml`
`systemctl restart containerd.service`

```
root@ip-172-31-22-133:/home/ubuntu# mkdir /etc/containerd
root@ip-172-31-22-133:/home/ubuntu#
root@ip-172-31-22-133:/home/ubuntu#
root@ip-172-31-22-133:/home/ubuntu# sh -c "containerd config default > /etc/containerd/config.toml"
root@ip-172-31-22-133:/home/ubuntu#
root@ip-172-31-22-133:/home/ubuntu# sed -i 's/ SystemdCgroup = false/ SystemdCgroup = true/' /etc/containerd/config.toml
root@ip-172-31-22-133:/home/ubuntu#
root@ip-172-31-22-133:/home/ubuntu# systemctl restart containerd.service
root@ip-172-31-22-133:/home/ubuntu#
```

Now, we will install our kubelet, kubeadm, and kubectl services on the Worker node
`apt-get install -y kubelet kubeadm kubectl kubernetes-cni`

```
root@ip-172-31-22-133:/home/ubuntu# apt-get install -y kubelet kubeadm kubectl kubernetes-cni
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  conntrack cri-tools ebtables socat
The following NEW packages will be installed:
  conntrack cri-tools ebtables kubeadm kubectl kubelet kubernetes-cni socat
0 upgraded, 8 newly installed, 0 to remove and 31 not upgraded.
Need to get 87.1 MB of archives.
After this operation, 336 MB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 conntrack amd64 1:1.4.6-2build2 [33.5 kB]
```

Now, restart the kubelet service, and don't forget to enable the kubelet service. So that, if any worker node will reboot then we don't need to start the kubelet service.

```
sudo systemctl restart kubelet.service
sudo systemctl enable kubelet.service
```

```
root@ip-172-31-23-243:/home/ubuntu# sudo systemctl restart kubelet.service
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu# sudo systemctl enable kubelet.service
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu#
```

If you remember, I told you to copy the Green highlighted command.
Paste that command, on both Worker Node1 and Worker Node2.

Once you do that, you will see the output like the below snippet.

```
root@ip-172-31-22-133:/home/ubuntu#
kubeadm join 172.31.22.132:6443 --token 0vzbaf.slplmyokc1lqland \
        --discovery-token-ca-cert-hash sha256:75c9d830b358fd6d372e03af0e79650360ce65790175 7e8b0b789a2e02475223
[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@ip-172-31-22-133:/home/ubuntu#
```

Run on any Master Node

Let's validate whether both Worker Nodes are joined in the Kubernetes Cluster by running the below command.

```
kubectl get nodes
```

If you can see four servers then, Congratulations you did 99% work.

```
root@ip-172-31-23-243:/home/ubuntu# kubectl get nodes
NAME              STATUS     ROLES           AGE     VERSION
ip-172-31-22-133  NotReady   <none>          33s     v1.28.2
ip-172-31-23-243  NotReady   control-plane   9m33s   v1.28.2
ip-172-31-28-74   NotReady   control-plane   5m19s   v1.28.2
ip-172-31-31-111  NotReady   <none>          26s     v1.28.2
root@ip-172-31-23-243:/home/ubuntu#
```

As you know, our all nodes are not in ready status because of network components.

Run the below command to add the Calico networking components in the Kubernetes Cluster.

```
kubectl apply -f
https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml
```

```
root@ip-172-31-23-243:/home/ubuntu# kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml
poddisruptionbudget.policy/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
serviceaccount/calico-node created
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipreservations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
root@ip-172-31-23-243:/home/ubuntu#
```

After 2 to 3 minutes, if you run the command ‘kubectl get nodes’. You will see that all nodes are getting ready.

```
root@ip-172-31-23-243:/home/ubuntu# kubectl get nodes
NAME               STATUS     ROLES           AGE     VERSION
ip-172-31-22-133   Ready      <none>          95s     v1.28.2
ip-172-31-23-243   NotReady   control-plane   10m     v1.28.2
ip-172-31-28-74    NotReady   control-plane   6m21s   v1.28.2
ip-172-31-31-111   Ready      <none>          88s     v1.28.2
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu# kubectl get nodes
NAME               STATUS   ROLES           AGE     VERSION
ip-172-31-22-133   Ready    <none>          2m3s    v1.28.2
ip-172-31-23-243   Ready    control-plane   11m     v1.28.2
ip-172-31-28-74    Ready    control-plane   6m49s   v1.28.2
ip-172-31-31-111   Ready    <none>          116s    v1.28.2
root@ip-172-31-23-243:/home/ubuntu#
```

Let’s deploy the Nginx Container on Worker Node1 and the Apache Container on Worker Node2

To achieve this, you have to perform the commands on Master Nodes only.

Add a label on both worker nodes

For WorkerNode1

```
kubectl label nodes <WorkerNode1-Private-IP> mynode=node1
```

For WorkerNode2

```
kubectl label nodes <WorkerNode2-Private-IP> mynode=node2
```

You can also validate whether the labels are added to both Worker Nodes or not by running the below command

```
kubectl get nodes --show-labels
```

```
root@ip-172-31-23-243:/home/ubuntu# kubectl get nodes
NAME               STATUS   ROLES           AGE     VERSION
ip-172-31-22-133   Ready    <none>          3m59s   v1.28.2
ip-172-31-23-243   Ready    control-plane   12m     v1.28.2
ip-172-31-28-74    Ready    control-plane   8m45s   v1.28.2
ip-172-31-31-111   Ready    <none>          3m52s   v1.28.2
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu# kubectl label nodes ip-172-31-22-133 mynode=node1
node/ip-172-31-22-133 labeled
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu# kubectl label nodes ip-172-31-31-111 mynode=node2
node/ip-172-31-31-111 labeled
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu# kubectl get nodes --show-labels
NAME               STATUS   ROLES           AGE     VERSION   LABELS
ip-172-31-22-133   Ready    <none>          5m42s   v1.28.2   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=ip-172-31-22-133,kuber
netes.io/os=linux,mynode=node1
ip-172-31-23-243   Ready    control-plane   14m     v1.28.2   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=ip-172-31-23-243,kuber
netes.io/os=linux,node-role.kubernetes.io/control-plane=,node.kubernetes.io/exclude-from-external-load-balancers=
ip-172-31-28-74    Ready    control-plane   10m     v1.28.2   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=ip-172-31-28-74,kubern
etes.io/os=linux,node-role.kubernetes.io/control-plane=,node.kubernetes.io/exclude-from-external-load-balancers=
ip-172-31-31-111   Ready    <none>          5m35s   v1.28.2   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=ip-172-31-31-111,kuber
netes.io/os=linux,mynode=node2
root@ip-172-31-23-243:/home/ubuntu#
```

Let’s create two Container on both different Worker nodes from different Master nodes

I am creating an Nginx Container on Worker Node1 from Master node1

Here is the deployment YML file

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
spec:
 replicas: 1
 selector:
   matchLabels:
     app: nginx
 template:
   metadata:
     labels:
       app: nginx
   spec:
     nodeSelector:
       mynode: node1 # This deploys the container on node1
     containers:
     -name: nginx
image: nginx:latest
ports:
-containerPort: 80
```

Here is the service YML file

```
apiVersion: v1
kind: Service
metadata:
 name: nginx-service
spec:
 selector:
   app: nginx
 ports:
 -protocol: TCP
   port: 80
   targetPort: 80
 type: LoadBalancer # Use LoadBalancer for external access
```

Apply both files by the below commands

```
kubectl apply -f deployment.yml
kubectl apply -f service.yml
```

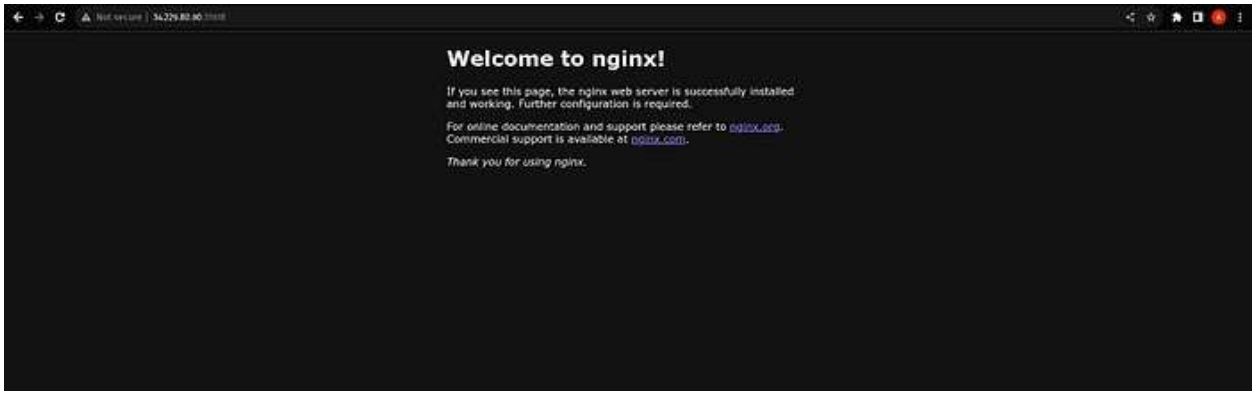
Validate whether the deployment is complete or not by running the below commands

```
kubectl get deploy
kubectl get pods
kubectl get svc
```

Now, to check whether the application is running or not from outside of the cluster. Copy the worker node1 public ip and then use the port that is showing when you run the ‘kubectl get svc’ command that i have highlighted in the snippet.

```
root@ip-172-31-23-243:/home/ubuntu# mkdir Node1
root@ip-172-31-23-243:/home/ubuntu#
root@ip-172-31-23-243:/home/ubuntu# cd Node1/
root@ip-172-31-23-243:/home/ubuntu/Node1#
root@ip-172-31-23-243:/home/ubuntu/Node1# sudo vim deployment.yml
root@ip-172-31-23-243:/home/ubuntu/Node1#
root@ip-172-31-23-243:/home/ubuntu/Node1# sudo vim svc.yml
root@ip-172-31-23-243:/home/ubuntu/Node1#
root@ip-172-31-23-243:/home/ubuntu/Node1# kubectl apply -f deployment.yml
deployment.apps/nginx-deployment created
root@ip-172-31-23-243:/home/ubuntu/Node1#
root@ip-172-31-23-243:/home/ubuntu/Node1# kubectl apply -f svc.yml
error: error parsing svc.yml: error converting YAML to JSON: yaml: line 2: mapping values are not allowed in this context
root@ip-172-31-23-243:/home/ubuntu/Node1# sudo vim svc.yml
root@ip-172-31-23-243:/home/ubuntu/Node1# kubectl apply -f svc.yml
service/nginx-service created
root@ip-172-31-23-243:/home/ubuntu/Node1#
root@ip-172-31-23-243:/home/ubuntu/Node1# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-6997f4bfb9-gqmvc   1/1     Running   0          58s
root@ip-172-31-23-243:/home/ubuntu/Node1#
root@ip-172-31-23-243:/home/ubuntu/Node1# kubectl get deploy
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   1/1     1            1           68s
root@ip-172-31-23-243:/home/ubuntu/Node1#
root@ip-172-31-23-243:/home/ubuntu/Node1# kubectl get svc
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP      10.96.0.1        <none>        443/TCP        20m
nginx-service   LoadBalancer   10.110.213.172   <pending>     80:31974/TCP   52s
root@ip-172-31-23-243:/home/ubuntu/Node1#
```

Here, you can see our nginx container is perfectly running outside of the Cluster.



The second Container from the Second Master Node on the Second Worker Node

I am creating Apache Container on Worker Node2 from Master node2

Here is the deployment YML file

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: apache-deployment
spec:
 replicas: 1
 selector:
   matchLabels:
     app: apache
 template:
   metadata:
     labels:
       app: apache
   spec:
     nodeSelector:
       mynode: node2 # This deploys the container on node1
     containers:
```

```
    -name: apache
image: httpd:latest
ports:
-containerPort: 80
```

Here is the service YML file

```
apiVersion: v1
kind: Service
metadata:
 name: apache-service
spec:
 selector:
   app: apache
 ports:
 -protocol: TCP
   port: 80
   targetPort: 80
 type: LoadBalancer # Use LoadBalancer for external access
```

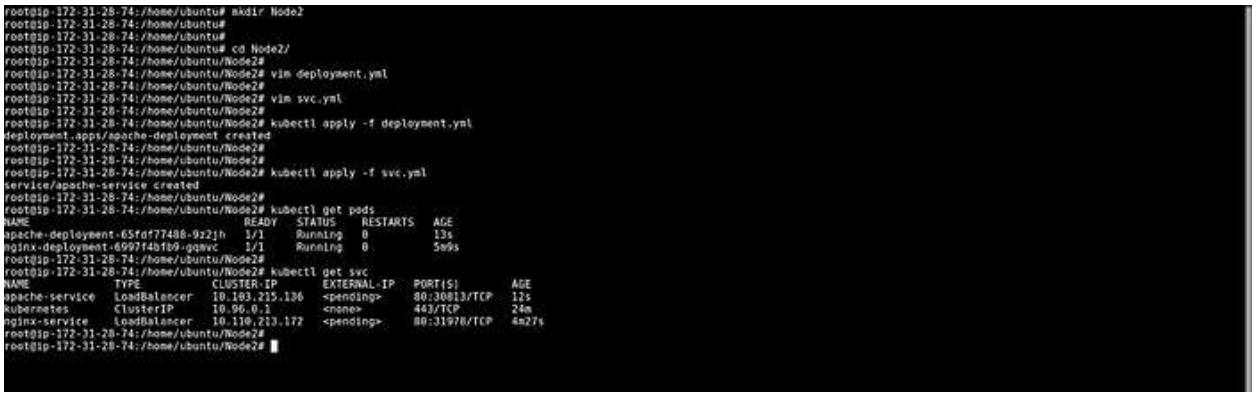
Apply both files by the below commands

```
kubectl apply -f deployment.yml
kubectl apply -f service.yml
```

Validate whether the deployment is complete or not by running the below commands

```
kubectl get deplov
kubectl get pods
kubectl get svc
```

Now, to check whether the application is running or not from outside of the cluster. Copy the worker node2 public IP and then use the port that is showing when you run the ‘kubectl get svc’ command corresponding to port 80.



Here, you can see our Apache container is perfectly running outside of the Cluster.

Kubernetes Ingress

In the world of Kubernetes, Ingress is your ticket to managing external traffic to services within the cluster. Before we dive into the details, let's recap what we've learned so far. Before Ingress, the Service provides a Load balancer, which is used to distribute the traffic between multiple applications or pods.

Ingress helps to expose the HTTP and HTTPS routes from outside of the cluster.

Ingress enables Path-based and Host-based routing.

Ingress supports Load balancing and SSL termination.

Simple Definition/Explanation

Kubernetes Ingress is like a cop for your applications that are running on your Kubernetes cluster. It redirects the incoming requests to the right services based on the Web URL or path in the address.

Ingress provides the encryption feature and helps to balance the load of the applications.

In simple words, Ingress is like a receptionist who provides the correct path for the hotel room to the visitor or person.

Why do we use Ingress because the load balancer supports the same thing?

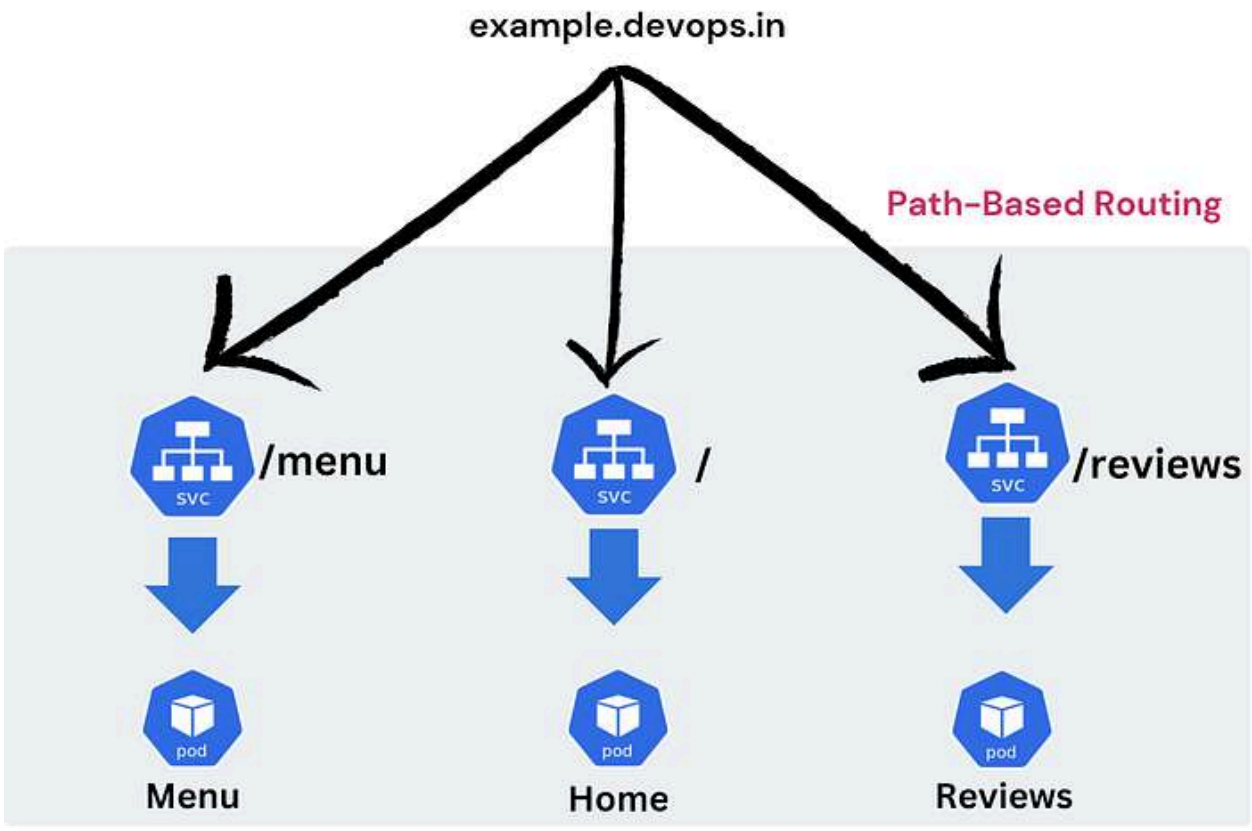
Ingress is used to manage the external traffic to the services within the cluster which provides features like host-based routing, path-based routing, SSL termination, and more. Where a Load balancer is used to manage the traffic but the load balancer does not provide the fine-grained access control like Ingress.

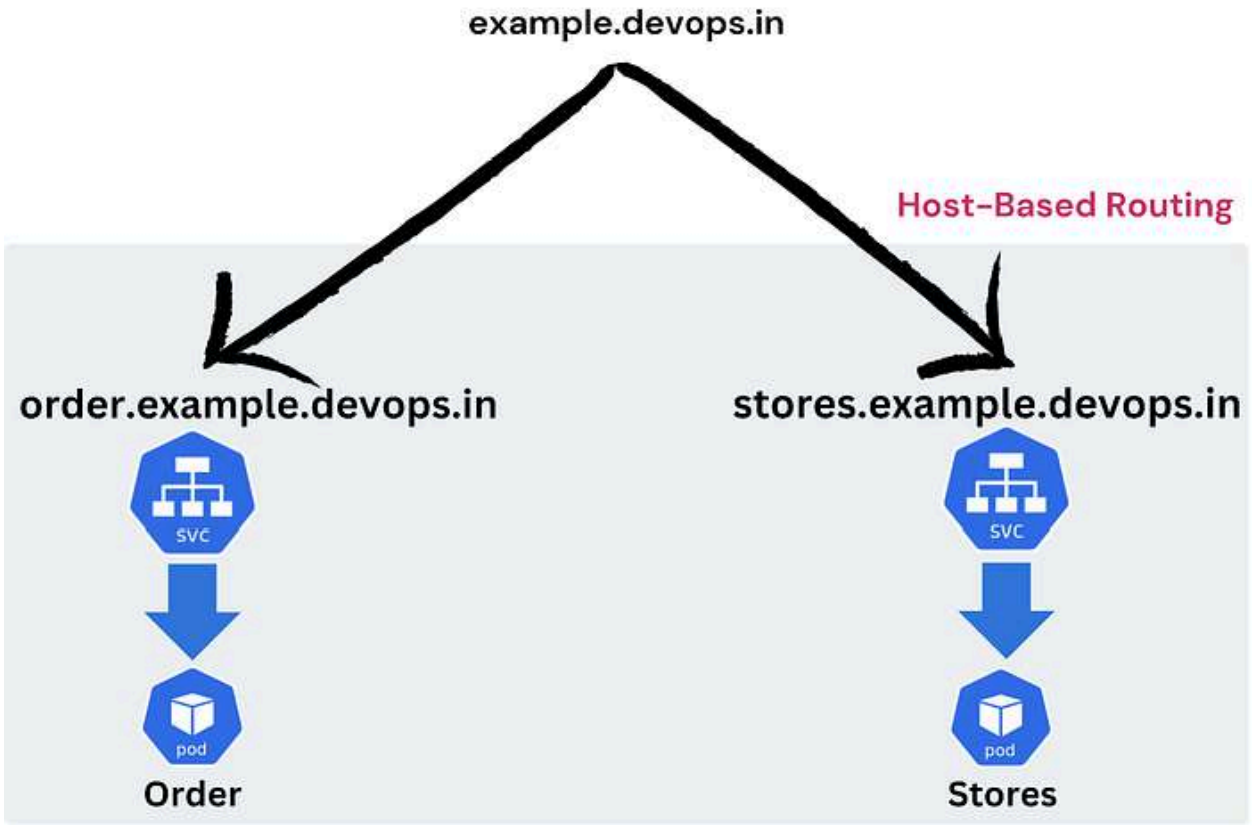
Example:

Suppose you have multiple Kubernetes services running on your cluster and each service serves a different application such as example.com/app1 and example.com/app2. With the help of Ingress, you can achieve this. However, the Load Balancer routes the traffic based on the ports and can't handle the URL-based routing.

There are two types of Routing in Ingress:

- **Path-based routing:** Path-based routing directs traffic to the different services based on the path such as example.com/app1.
- **Host-based routing:** Host-based routing directs traffic to the different services based on the Website's URL such as demo.example.com.


example.devops.in
Path-Based Routing
SVC
/menu
SVC
/
SVC
/reviews
pod
Menu
pod
Home
pod
Reviews


example.devops.in
Host-Based Routing
order.example.devops.in
stores.example.devops.in
SVC
SVC
pod
Order
pod
Stores

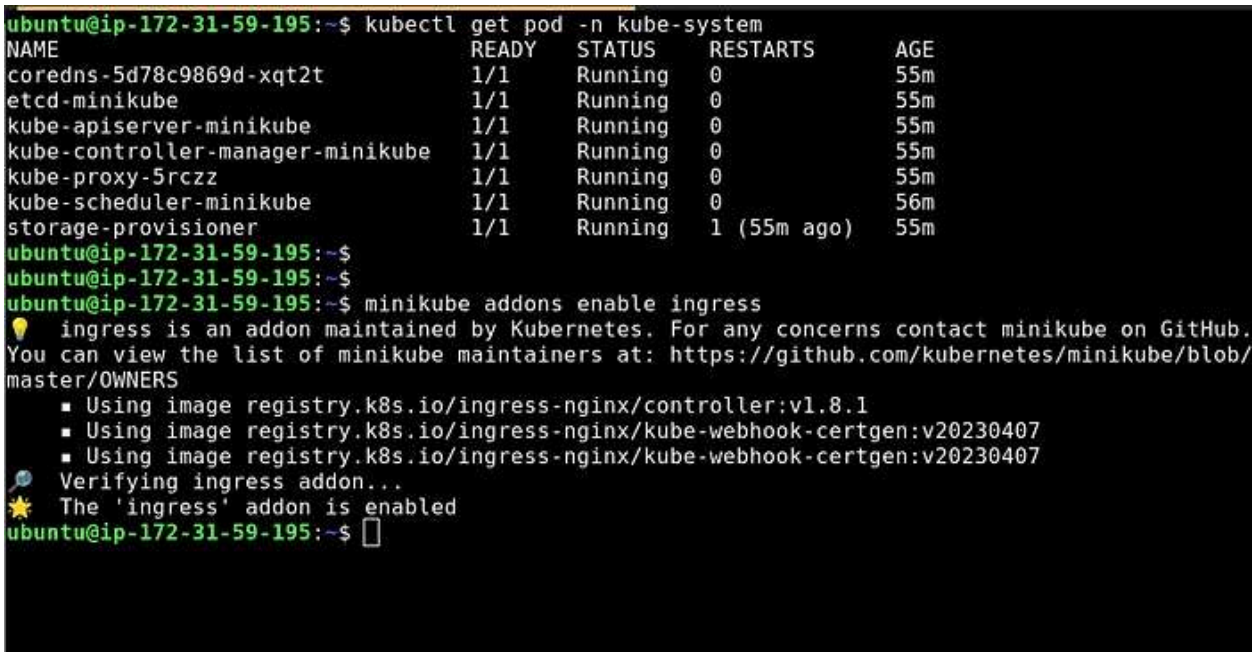
To implement Ingress, we have to deploy Ingress Controllers. We can use any Ingress Controllers according to our requirements.

Hands-On

Here, we will use the nginx ingress controller.

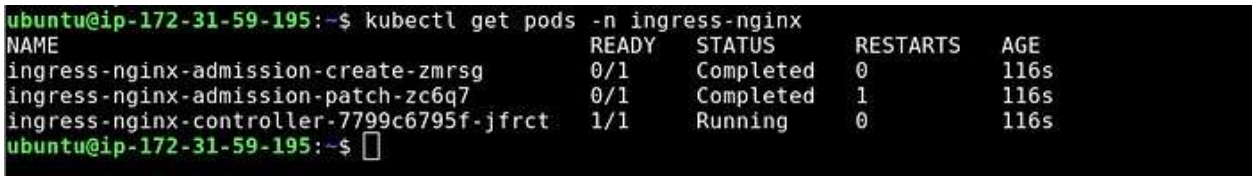
To install it, use the command.

```
minikube addons enable ingress
```



Validate whether the controller is deployed or not

```
kubectl get pods -n ingress-nginx
```



Now, let's do some hands-on for Path-based routing.

Deploy home page

deployment1.yml file

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
spec:
 replicas: 1
 selector:
```

```
  matchLabels:
    app: nginx
 template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
    -name: nginx
      image: avian19/choco-shop-home
      ports:
      -containerPort: 80
service1.yml file
apiVersion: v1
kind: Service
metadata:
 name: nginx-service
spec:
 selector:
  app: nginx
 ports:
 -protocol: TCP
  port: 80
 type: NodePort
```

```
kubectl apply -f deployment1.yml
kubectl apply -f service1.yml
```

Once you created the deployment and service, now we have to create the ingress for path-based routing. As we want to direct the requests to the default path use the below YAML file.

ingress.yml

```
kind: Ingress
metadata:
 name: ingress-deployment
 annotations:
  nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
 rules:
  -host: example.devops.in
    http:
      paths:
        -path: /
          pathType: Prefix
          backend:
            service:
              name: nginx-service
              port:
```

```
              number: 80
```

```
kubectl apply -f ingress.yml
```

Add the IP ADDRESS that you got in the above snippet from ingress-deployment(192.168.49.2).

```
vim /etc/hosts
```

If you do curl from the terminal then you can able to see the content of your application.

If you map the DNS name with ingress Private IP, then you can able to see the content of your application from the browser.

Now I have one more module whose name is Menu that I want to deploy on the other services.

To do that, Create a deployment file and a service file.

deploy2.yml file

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment2
spec:
 replicas: 1
 selector:
   matchLabels:
     app: nginx2
 template:
   metadata:
```

```
    labels
      app: nginx2
  spec:
    containers:
    -name: nginx2
      image: avian19/choco-shop-menu
      ports:
      -containerPort: 80
```

service2.yml file

```
apiVersion: v1
kind: Service
metadata:
 name: nginx-service2
spec:
 selector:
   app: nginx2
 ports:
 -protocol: TCP
   port: 80
 type: NodePort
```

```
kubectl apply -f deploy2.yml
kubectl apply -f service2.yml
```

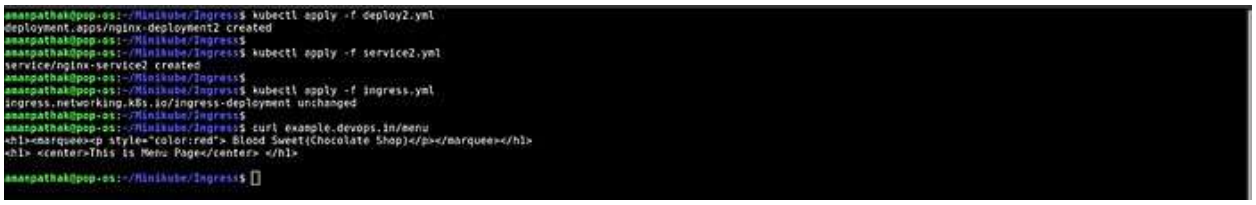
Now, Here we have to modify our ingress file as we have added a new service which has a new application. To avoid confusion, just remove the previous content from the ingress.yml file copy and paste the entire content in the ingress.yml file, and apply the updated configurations.

Updated ingress.yml file

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: ingress-deployment
 annotations:
   nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
 rules:
  -host: example.devops.in
    http:
      paths:
        -path: /
          pathType: Prefix
          backend:
            service:
              name: nginx-service
              port:
                number: 80
        -path: /menu
          pathType: Prefix
          backend:
```

```
            service:
              name: nginx-service2
              port:
                number: 80
```

```
kubectl apply -f ingress.yml
```



Now, we can access our application on the /menu path.

Now I have one more module which name is Reviews that I want to deploy on other services.

To do that, Create a deployment file and a service file.

deploy3.yml file

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment3
spec:
 replicas: 1
 selector:
   matchLabels:
     app: nginx3
 template:
   metadata:
     labels:
       app: nginx3
   spec:
     containers:
     -name: nginx3
       image: avian19/choco-shop-reviews
       ports:
       -containerPort: 80
```

service3.yml file

```
apiVersion: v1
```

```
kind: Service
metadata:
 name: nginx-service3
spec:
 selector:
   app: nginx3
 ports:
 -protocol: TCP
   port: 80
 type: NodePort
```

```
kubectl apply -f deploy3.yml
kubectl apply -f service3.yml
```

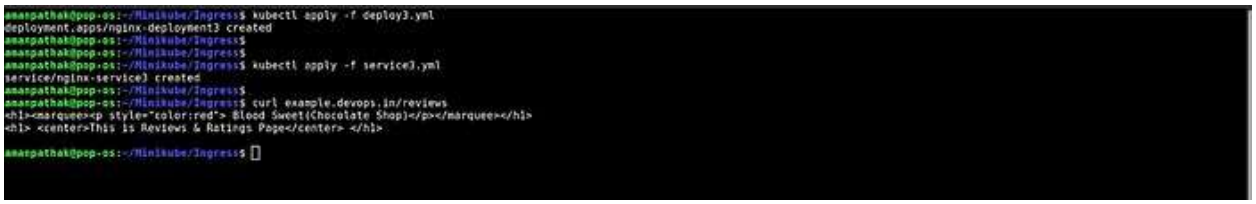
Now, Here we have to modify our ingress file as we have added a new service which has a new application. To avoid confusion, just remove the previous content from the ingress.yml file copy and paste the entire content in the ingress.yml file, and apply the updated configurations.

Updated ingress.yml file

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: ingress-deployment
 annotations:
   nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
 rules:
   -host: example.devops.in
     http:
       paths:
         -path: /
           pathType: Prefix
           backend:
             service:
               name: nginx-service
               port:
                 number: 80
         -path: /menu
           pathType: Prefix
           backend:
             service:
               name: nginx-service2
               port:
                 number: 80
         -path: /reviews
           pathType: Prefix
           backend:
             service:
```

```
            name: nginx-service3
            port:
              number: 80
```

```
kubectl apply -f ingress.yml
```



Now, we can access our application on the /reviews path.

Host-based Routing

Now, we have completed our hands-on for Path-based Routing.

I want to create one more application to order anything from different hosts. Let's do that.

Deploy the applications and services.

deploy4.yml file

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment4
spec:
 replicas: 1
 selector:
   matchLabels:
     app: nginx4
 template:
   metadata:
     labels:
       app: nginx4
   spec:
     containers:
     -name: nginx4
       image: avian19/choco-shop-order
       ports:
       -containerPort: 80
```

service4.yml file

```
apiVersion: v1
kind: Service
metadata:
 name: nginx-service4
spec:
 selector:
   app: nginx4
 ports:
 -protocol: TCP
   port: 80
 type: NodePort
```

```
kubectl apply -f deploy4.yml
kubectl apply -f service4.yml
```

Now, Here we have to modify our ingress file as we have added a new service which has a new application. To avoid confusion, just remove the previous content from the ingress.yml file copy and paste the entire content into the ingress.yml file, and apply the updated configurations.

Updated ingress.yml file

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: ingress-deployment
 annotations:
   nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
 rules:
   -host: example.devops.in
     http:
       paths:
         -path: /
           pathType: Prefix
           backend:
             service:
               name: nginx-service
               port:
                 number: 80
         -path: /menu
           pathType: Prefix
           backend:
             service:
               name: nginx-service2
               port:
                 number: 80
         -path: /reviews
           pathType: Prefix
           backend:
             service:
```

```
            name: nginx-service3
            port:
              number: 8
#Host Based Routing
 -host: example2.devops.in
  http:
    paths:
      -path: /
        pathType: Prefix
        backend:
          service:
            name: nginx-service4
            port:
              number: 80
```

```
kubectl apply -f ingress.yml
```

Now, if you try to curl from the terminal then you will be able to get the content. But if you try from the browser then you won't be able to get the content.

Now, we have added a new host in the ingress file. To get the content on our local host, we need to add this host in the /etc/hosts file.

Now check on the browser by hitting the new hostname.

Kubernetes StatefulSets

What are the Stateful applications?

Stateful applications are those applications that contain the previous states or data of the applications. If the applications restart or move to other environments then, the data and the state of the applications still exist.

Examples of Stateful applications includes PostgreSQL, MySQL, messages queues. These applications ensures that the data, application reliability, and state of the applications will be taken care of by the StatefulSets Object in Kubernetes.

Difference between Stateful and Stateless applications

Let's understand the Stateful and Stateless applications with real-time examples.

Stateful applications

- "Remember when we used to play GTA Vice City during our teen years? Completing
- that game required hours because we were in school at the time."
- "Once I completed a mission, I saved the game. So, whenever I wanted to continue, I just went to the save game section and resumed from where I left off."
  "At that moment, we didn't think about how it worked, but now we understand that there were sessions that helped us save the game data, which was stored in our root folder."

Stateless applications

- "If you used old keypad mobiles, you might have used a calculator application."
- "Your father asked you to perform operations like addition and subtraction."
- "By mistake, you tapped the red button, which took you back to the home screen."
- "Now, you can't retrieve the numbers you were entering. This means there is no session involved."

StatefulSets VS Deployment

Kubernetes has rich features like StatefulSets and deployment. But Statefulsets eliminates the previous states and data stored problems.
Let's understand both.

StatefulSets

- **Identity and Stable Network hostnames** : StatefulSets are used for those applications that require stable network identity and hostnames. Whenever the pod is created, it gets a unique name, an ordinal index appended to its name. For example, the pod name looks like web-1, web-2, web-3, and so on.
- **Order deployment and Scaling:** StatefulSets deploy the pods in sequential order. If you observe in deployment, the multiple pods in replicas are created at one time. But In StatefulSets, each pod will be created once the previous pod is created. If pods

are deleted, the newest created pod will be deleted first which means, that to delete the pod StatefulSets follows the reverse order.

- **Data Persistence** : StatefulSets are used for those applications that require data persistence such as databases. StatefulSets allow to attachment and mount of the permanent volumes to the disk. So, if any pod is rescheduled or restarted it will have the all data.
- **Headless Services** : StatefulSets have one more rich feature which is Headless Services. StatefulSets can be associated with the headless services that provide the DNS for each pod's hostnames. This will help to communicate with the specific pods.

Deployment

- **Scalability and Rolling updates** : Deployments are often used for stateless
- applications. Deployments provide the replicas and no downtime feature.
- **No Stable Hostnames** : Deployments do not provide the feature of stable hostnames which means, that if the pod is created then it will have a randomly generated name. **No Data Persistence:** As deployment objects are often used for stateless applications So, pods are stateless too which means it does not provide data persistence. Whenever a pod is replaced, the data will be lost of the previous pod.
- **Load Balancing:** Deployment works with Services objects which provides Load Balancing. It distributes the traffic between multiple pods to make the application highly available.

Key Features and Facts of StatefulSets:

1. StatefulSets provides the Order pod creation which provides the unique name to each pod in a predictable order such as web-1, web-2, and so on.
2. StatefulSets provides the Stable Network Identity which makes it easy to connect with the pods.
3. StatefulSets provides the Data Persistence which stores the data in the databases and whenever the pod is restarted or rescheduled, the pods get the same persistent data.
4. StatefulSets provides the PV and PVC to provide the persistent storage to StatefulSets pods.
5. StatefulSets provides the Backup and Recovery features which are very crucial for maintaining data integrity with StatefulSets.

Hands-On

Open the two terminals

We have to see how the pods are created. To do that, run the below command on terminal1

```
kubectl get pods -w -l app=nginx
```

Create a service file and StatefulSet file copy the below content in the respective file and apply it.

service.yml

```
apiVersion: v1
kind: Service
metadata:
 name: nginx
 labels:
   app: nginx
spec:
 ports:
 -port: 80
   name: web
 clusterIP: None
 selector:
   app: nginx
StatefulSets.yml
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: web
spec:
 serviceName: "nginx"
 replicas: 2
 selector:
   matchLabels:
     app: nginx
 template:
   metadata:
     labels:
       app: nginx
   spec:
     containers:
     -name: nginx
       image: registry.k8s.io/nginx-slim:0.8
       ports:
       -containerPort: 80
         name: web
       volumeMounts:
       -name: www
         mountPath: /usr/share/nginx/html
 volumeClaimTemplates:
 -metadata:
     name: www
   spec:
     accessModes: [ "ReadWriteOnce"
     resources:

         storage: 1Gi
```

```
amarpathak@pop-os:~/Minikube/StatefulSets$ vim service.yml
amarpathak@pop-os:~/Minikube/StatefulSets$
amarpathak@pop-os:~/Minikube/StatefulSets$ vim StatefulSet.yml
amarpathak@pop-os:~/Minikube/StatefulSets$
amarpathak@pop-os:~/Minikube/StatefulSets$ kubectl apply -f service.yml
service/nginx created
amarpathak@pop-os:~/Minikube/StatefulSets$
amarpathak@pop-os:~/Minikube/StatefulSets$ kubectl apply -f StatefulSet.yml
statefulset.apps/web created
amarpathak@pop-os:~/Minikube/StatefulSets$
```

The Pods are creating sequential, observations in the first command output.

```
amarpathak@pop-os:~/Minikube/StatefulSets$ kubectl get pods -w -l app=nginx
NAME    READY   STATUS              RESTARTS   AGE
web-0   0/1     Pending             0          0s
web-0   0/1     Pending             0          0s
web-0   0/1     Pending             0          1s
web-0   0/1     ContainerCreating   0          2s
web-0   1/1     Running             0          13s
web-1   0/1     Pending             0          0s
web-1   0/1     Pending             0          0s
web-1   0/1     Pending             0          1s
web-1   0/1     ContainerCreating   0          2s
web-1   1/1     Running             0          2s
^Camarpathak@pop-os:~/Minikube/StatefulSets$
amarpathak@pop-os:~/Minikube/StatefulSets$
amarpathak@pop-os:~/Minikube/StatefulSets$ kubectl get service nginx
NAME    TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
nginx   ClusterIP   None         <none>        80/TCP    2m3s
amarpathak@pop-os:~/Minikube/StatefulSets$
amarpathak@pop-os:~/Minikube/StatefulSets$
amarpathak@pop-os:~/Minikube/StatefulSets$ kubectl get statefulset web
NAME   READY   AGE
web    2/2     2m3s
amarpathak@pop-os:~/Minikube/StatefulSets$
amarpathak@pop-os:~/Minikube/StatefulSets$
```

As we have created pods from statefulset, So the pod's names have sticky and unique names.

```
amarpathak@pop-os:~/Minikube/StatefulSets$ kubectl get pods -l app=nginx
NAME    READY   STATUS    RESTARTS   AGE
web-0   1/1     Running   0          3m59s
web-1   1/1     Running   0          3m46s
amarpathak@pop-os:~/Minikube/StatefulSets$
```

If you run the below command, you will see that both pods have stable hostnames on their ordinal index.

```
for i in 0 1; do kubectl exec "web-$i" -- sh -c 'hostname'; done
```

```
amarpathak@pop-os:~/Minikube/StatefulSets$ for i in 0 1; do kubectl exec "web-$i" -- sh -c 'hostname'; done
web-0
web-1
amarpathak@pop-os:~/Minikube/StatefulSets$
```

Now, I want to check the pod's dns addresses. Run the below command

```
kubectl run -i --tty --image busybox:1.28 dns-test --restart=Never --rm
```

```
amarpathak@pop-os:~/Minikube/StatefulSets$ kubectl run -i --tty --image busybox:1.28 dns-test --restart=Never --rm
```

Once you enter the container, run the below command.

```
nslookup web-0.nginx
```

Now, you can check the dns address.

```
amarpathak@pop-os:~/Minikube/StatefulSets$ kubectl run -i --tty --image busybox:1.28 dns-test --restart=Never --rm
If you don't see a command prompt, try pressing enter.
/ #
/ # nslookup web-0.nginx
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      web-0.nginx
Address 1: 10.244.0.61 web-0.nginx.default.svc.cluster.local
/ #
```

Open two terminals, run the below command on terminal1

```
kubectl get pod -w -l app=nginx
```

As you can see both pods are running.

Now, run the below command to delete both pods on terminal 2.

```
kubectl delete pod -l app=nginx
```

As you know, we have set up the replicas=2. So, if we delete any pod it will create a new pod to meet with the desired state of the replicas.

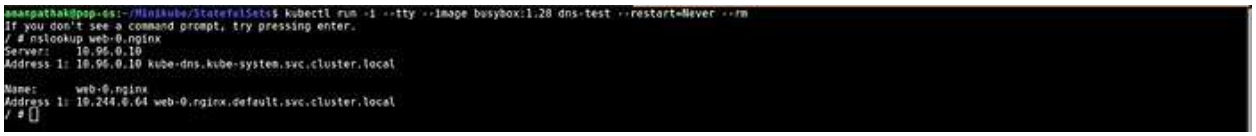
But there is one more thing to notice here, that the pod is deleting sequentially.

If you run the last command, you will see that both pods have been created again.

If you run the below command, you will see the same hostnames that were present for previous delete pods.

```
for i in 0 1; do kubectl exec web-$i — sh -c 'hostname'; done
```

Now, if you log in to the container. You will be able to see the same dns address but the IP might have changed.



Run the command to get a list of all the persistent volume claims that are attached to the app nginx.

```
ananpathak@pop-os:~/Minikube/StatefulSets$ kubectl get pvc -l app=nginx
NAME        STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
www-web-0   Bound    pvc-976c12ac-2c88-4a60-a347-3e3134538609   1Gi        RWO            standard       26m
www-web-1   Bound    pvc-55aaa2ff-048f-4329-8c28-603bd87da117   1Gi        RWO            standard       26m
ananpathak@pop-os:~/Minikube/StatefulSets$
```

As we have mounted our persistent volume to the path /usr/share/nginx/html in the Stateful.yml file. So, the path will be backed by the Persistent Volume.

Now, to check the hostnames of both pods run the below command.

```
for i in 0 1; do kubectl exec "web-$i" -- sh -c 'echo "$(hostname)" > /usr/share/nginx/html/index.html'; done
```

```
ananpathak@pop-os:~/Minikube/StatefulSets$ for i in 0 1; do kubectl exec "web-$i" -- sh -c 'echo "$(hostname)" > /usr/share/nginx/html/index.html'; done
ananpathak@pop-os:~/Minikube/StatefulSets$
ananpathak@pop-os:~/Minikube/StatefulSets$ for i in 0 1; do kubectl exec -i -t "web-$i" -- curl http://localhost/; done
web-0
web-1
ananpathak@pop-os:~/Minikube/StatefulSets$
```

Delete both pods

```
kubectl delete pod -l app=nginx
```

```
ananpathak@pop-os:~/Minikube/StatefulSets$ kubectl delete pod -l app=nginx
pod "web-0" deleted
pod "web-1" deleted
ananpathak@pop-os:~/Minikube/StatefulSets$
```

Validate the pods' deletion and creation on the other window

```
kubectl get pod -w -l app=nginx
```

```
ananpathak@pop-os:~/Minikube/StatefulSets$ kubectl get pod -w -l app=nginx
NAME    READY   STATUS              RESTARTS   AGE
web-0   1/1     Running             0          26m
web-1   1/1     Running             0          26m
web-0   1/1     Terminating         0          27m
web-1   1/1     Terminating         0          27m
web-0   0/1     Terminating         0          27m
web-1   0/1     Terminating         0          27m
web-0   0/1     Terminating         0          27m
web-0   0/1     Terminating         0          27m
web-0   0/1     Terminating         0          27m
web-1   0/1     Terminating         0          27m
web-0   0/1     Pending             0          0s
web-1   0/1     Terminating         0          27m
web-1   0/1     Terminating         0          27m
web-0   0/1     Pending             0          0s
web-0   0/1     ContainerCreating   0          0s
web-0   1/1     Running             0          2s
web-1   0/1     Pending             0          0s
web-1   0/1     Pending             0          0s
web-1   0/1     ContainerCreating   0          0s
web-1   1/1     Running             0          1s
^Cananpathak@pop-os:~/Minikube/StatefulSets$
```

Now, validate the web servers whether the hostname is the same or not

```
for i in 0 1; do kubectl exec -i -t “web-$i” — curl http://localhost/; done
```

```
ananpathak@pop-os:~/Minikube/StatefulSets$ for i in 0 1; do kubectl exec -i -t "web-$i" -- curl http://localhost/; done
web-0
web-1
ananpathak@pop-os:~/Minikube/StatefulSets$
```

Now, let’s scale up the StatefulSets. You can use the kubectl scale or kubectl patch to scale up or scale down the replicas.

```
kubectl scale sts web — replicas=5
```

```
anangathak@pop-os:~/Minikube/StatefulSets$ kubectl scale sts web --replicas=5
statefulset.apps/web scaled
anangathak@pop-os:~/Minikube/StatefulSets$ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
web-0   1/1     Running   0          65m
web-1   1/1     Running   0          65m
web-2   1/1     Running   0          14s
web-3   1/1     Running   0          11s
web-4   1/1     Running   0          8s
anangathak@pop-os:~/Minikube/StatefulSets$
```

If you see the persistent volume claim, it will be increased as pods are created.

kubectl get pvc -l app=nginx

```
anangathak@pop-os:~/Minikube/StatefulSets$ kubectl get pvc -l app=nginx
NAME        STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
www-web-0   Bound    pvc-976c12ac-2c88-4a60-a347-3e3134538609   1Gi        RWO            standard       115m
www-web-1   Bound    pvc-55aaa2ff-048f-4329-8c28-6038d87da117   1Gi        RWO            standard       115m
www-web-2   Bound    pvc-3bbced08-3cb3-4234-9d79-c208b1c57095   1Gi        RWO            standard       7m14s
www-web-3   Bound    pvc-ec6c0879-3e61-4ab9-a14b-b6afce1e24a0   1Gi        RWO            standard       7m11s
www-web-4   Bound    pvc-d1b75023-4628-4f42-83cf-e9023496ee99   1Gi        RWO            standard       7m8s
anangathak@pop-os:~/Minikube/StatefulSets$
```

Now, scale down the number of replicas through the kubectl patch

kubectl patch sts web -p '{"spec":{"replicas":3}}'

```
anangathak@pop-os:~/Minikube/StatefulSets$ kubectl patch sts web -p '{"spec":{"replicas":3}}'
statefulset.apps/web patched
anangathak@pop-os:~/Minikube/StatefulSets$
anangathak@pop-os:~/Minikube/StatefulSets$ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
web-0   1/1     Running   0          75m
web-1   1/1     Running   0          75m
web-2   1/1     Running   0          9m45s
```

If you list the Persistent volume claims, you will see all 5 PVCs present. This is because StatefulSets assumes that you deleted the pods by mistake.

```
anangathak@pop-os:~/Minikube/StatefulSets$ kubectl get pvc -l app=nginx
NAME        STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
www-web-0   Bound    pvc-976c12ac-2c88-4a60-a347-3e3134538609   1Gi        RWO            standard       118m
www-web-1   Bound    pvc-55aaa2ff-048f-4329-8c28-6038d87da117   1Gi        RWO            standard       118m
www-web-2   Bound    pvc-3bbced08-3cb3-4234-9d79-c208b1c57095   1Gi        RWO            standard       10m
www-web-3   Bound    pvc-ec6c0879-3e61-4ab9-a14b-b6afce1e24a0   1Gi        RWO            standard       9m58s
www-web-4   Bound    pvc-d1b75023-4628-4f42-83cf-e9023496ee99   1Gi        RWO            standard       9m55s
anangathak@pop-os:~/Minikube/StatefulSets$
```

Now, to rollback use the below command

kubectl patch statefulset web -p '{"spec":{"updateStrategy":{"type":"RollingUpdate"}}}'

The below command will change the image of the container.

kubectl patch statefulset web — type='json' -p='[{"op": "replace", "path": "/spec/template/spec/containers/0/image", "value":"gcr.io/google_containers/nginx-slim:0.8"}]'

```
anangathak@pop-os:~/Minikube/StatefulSets$ kubectl patch statefulset web -p '{"spec":{"updateStrategy":{"type":"RollingUpdate"}}}'
statefulset.apps/web patched (no change)
anangathak@pop-os:~/Minikube/StatefulSets$
anangathak@pop-os:~/Minikube/StatefulSets$ kubectl patch statefulset web --type='json' -p='[{"op": "replace", "path": "/spec/template/spec/containers/0/image", "value":"gcr.io/google_contain
ers/nginx-slim:0.8"}]'
statefulset.apps/web patched
```

Use the below command to check the container image

for p in 0 1 2; do kubectl get pod "web-$p" — template '{{range $i, $c := .spec.containers}}{{$c.image}}{{end}}'; echo; done

```
anangathak@pop-os:~/Minikube/StatefulSets$ for p in 0 1 2; do kubectl get pod "web-$p" --template '{{range $i, $c := .spec.containers}}{{$c.image}}{{end}}'; echo; done
gcr.io/google_containers/nginx-slim:0.8
gcr.io/google_containers/nginx-slim:0.8
gcr.io/google_containers/nginx-slim:0.8
anangathak@pop-os:~/Minikube/StatefulSets$
```

If you want to delete StatefulSets then you have two options Non-Cascading deletion and Cascading deletion.

In Non Cascading deletion, the StatefulSet's Pods are not deleted when StatefulSets is deleted.

In Cascading deletion, both the StatefulSet's Pod and StatefulSets are deleted.

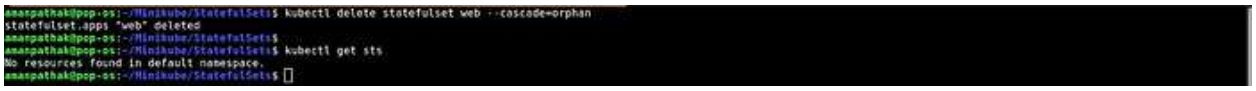
Non-Cascading deleted

Open the two terminals and run the below command on the first terminal.

```
kubectl get pods -w -l app=nginx
```

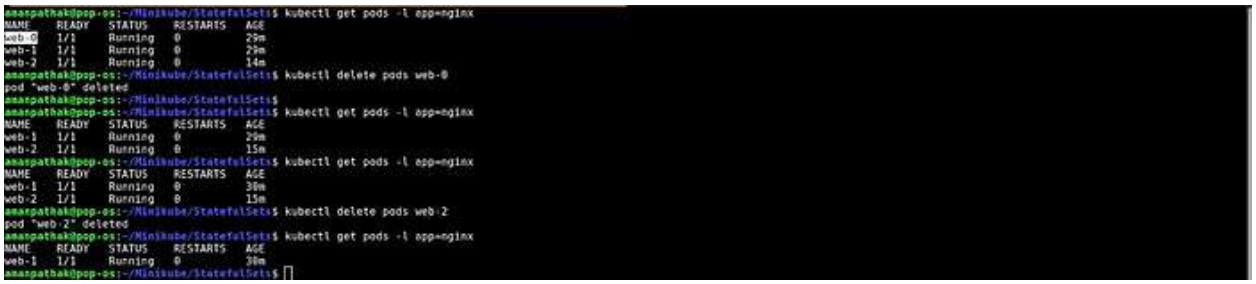
On the second terminal, this command will delete only StatefulSets only, not the StatefulSets pods.

```
kubectl delete statefulset web --cascade=orphan
```



If you go to the first terminal, you will see the pods still exist.

If you try to delete any pod, you will observe that the deleted pods are not relaunching.



Now, we have to perform the StatefulSets deletion through the Cascading method.

To do that, apply the both service and StatefulSets files.

If you see both pods are running

Check the hostname of both pods by running the below command

```
for i in 0 1; do kubectl exec -i -t “web-$i” — curl http://localhost/; done
```

Cascading deleted

In this method, the StatefulSets will be deleted with StatefulSet’s Pod.

```
kubectl delete statefulset web
```

Kubernetes DaemonSet

"A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created." — Kubernetes DaemonSet Official Definition Suppose you have to deploy the same pod on all nodes for monitoring. DaemonSet ensures that a specific pod runs on all nodes within the cluster.

Key Features

1. **DaemonSets Ensure Uniformly** : DaemonSets ensures that adesignated pod which is used for logs and monitoring will be deployed on every node within the cluster.
2. **Perfect for Infrastructure Services** : DaemonSets are great for services that need to run on every node like networking, storage, or security agents.
3. **Scaling Automatically** : If you add more nodes, then the DaemonSets Pods will be added to new nodes automatically.
4. **Stable Hostnames** : DaemonSets provides stable hostnames which means on each node, the pod's names remain the same if the pods is rescheduled or restarted which makes it easy to reference them across the cluster.

UseCases

1. **Monitoring and Logging:** With the help of DaemonSets, we can deploy monitoring agents or log collectors to gather the information on the node. Tools like Prometheus, beats, ElasticSearch, etc can be deployed using DaemonSets to ensure complete coverage across the cluster.
2. **Security Agents:** We can deploy Security Agents like intrusion detection systems (IDS) or anti-malware software on every node to protect the cluster from threats.
3. **Custom Network Policies** : We can deploy the custom network policies or firewall rules on each node to control communication and security at the node level.
4. **Operating System Updates** : We can deploy the updates or patches at one time with the help of DaemonSets.
5. **Storage and Data Management:** Ensuring that each node has access to particular storage resources, such as local storage or network-attached storage(NAS). DaemonSets can manage storage plugins or agents to provide consistent access.

HandsOn

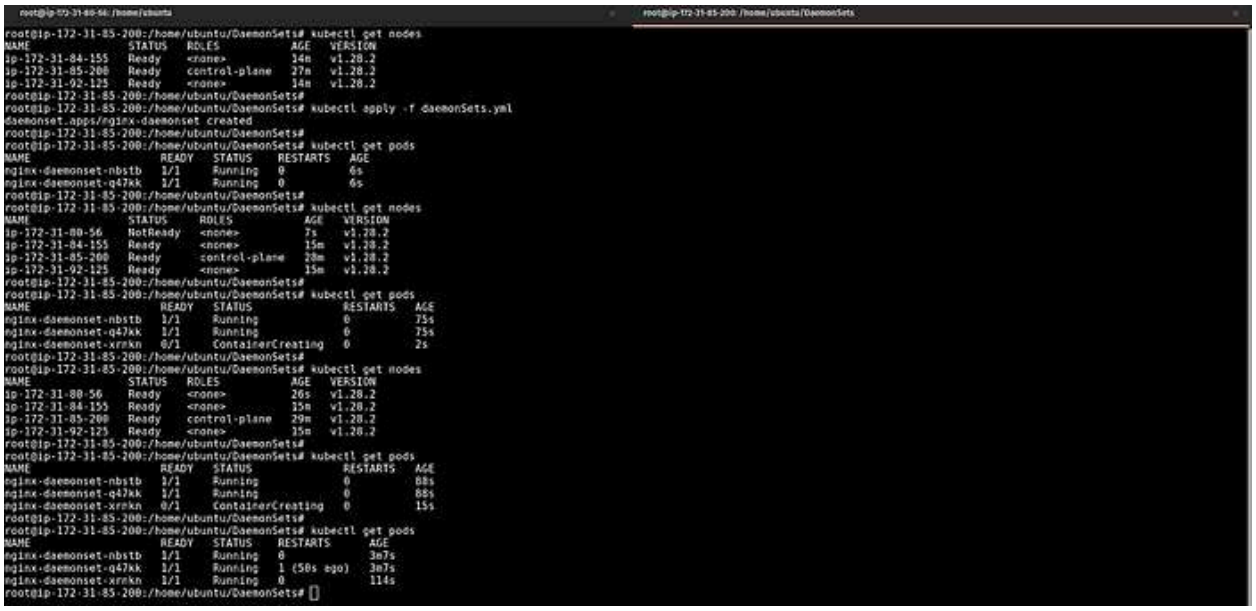
Initially, I created three machines of which one is a Master Node and the rest of two are Worker Nodes.

If you don't know how to set up the multi-worker nodes using kubeadm then kindly refer to my blog Day10- Setting up a Kubernetes Cluster(Master+Worker Node) using kubeadm on AWS EC2 Instances(Ubuntu 22.04) | by Aman Pathak | DevOps.dev which will take maximum 15 minutes

Let's see our Object first:

- Currently, I have three machines Only(1 Master + 2 Worker Nodes)
- We will deploy the DaemonSet Pod on two Worker Nodes.
- Once the Pods will be running on both Worker Nodes. We will create a new machine
- as Worker Node 3.
- Worker Node3 will join the Kubernetes cluster.
  The DaemonSet Pod will be running automatically without any intervention from our side.

This is the Entire Proof of our Hands-On. You just need to take a look. Further Step by Step, we will look into the next steps.



As of now, we have only One Master Node(control plane) and Two Worker Nodes.

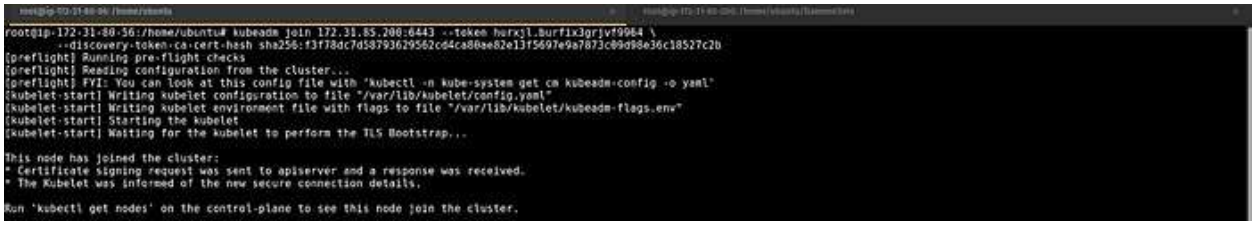
DaemonSet YAML file

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
 name: nginx-daemonset
spec:
 selector:
    matchLabels
     name: nginx
 template:
   metadata:
     labels:
       name: nginx
```

```
  spec:
    containers:
    -name: nginx
      image: nginx:latest
```

I have deployed my daemonSet YAML file which is deployed to both Worker Nodes.

Here, I have run the command to join the Kubernetes Cluster for Worker Node3.



After joining, you can see the node is getting ready to be part of the cluster.

Here, you can see that Worker Node3 is in ready status.

If you list all the pods, you will see there is one more started in which the container is creating without running any command.

Here, you can see the Pod is in running status.

Kubernetes Network Policies

What is Network Policy?

By default, a pod can communicate with any other pods whether it's present in any namespaces. But if you want to secure your pod by providing access to only known pods or authorized pods then Kubernetes has the richest feature known as Network Policy. Network Policy will help you to protect your pod by accessing only authorized pods. So, this way the pod's security will be enhanced.

Network Policy allows us to define the rules to communicate between the pods. With the help of Networking Policy, Kubernetes provides fine-grained controls over what traffic is allowed or denied which leads to enhancing the security and isolation of your applications.

Key Features:

Policy Rules: Network Policies consist of a set of rules in which you define how the traffic is allowed or denied. You can specify these rules by pod labels, namespaces, or particular IPs.

Pod Selectors: If you want to apply the Network Policy to the particular pod then you can use Pod Selector which will select the particular pod and apply the Network Policy on that pod.

Ingress and Egress: Network Policies allow you to define the Ingress and Egress rules. Ingress means incoming traffic on the pod from the outside whereas Egress means outgoing traffic to the internet(anywhere) from the pod itself.

Namespaces: If you want to apply your Network Policy to the group of pod which is present in the particular namespace then you can namespaceSelector which will help you to invoke the Network Policy on all pods within the particular namespace.

Priority: Network Policy also provides the priority feature in which you define the priority of the rules which helps you to get fine-grained control over the traffic rules of your application.

Use cases of Network Policy:

- **Isolation** : You can invoke the Network Policies to isolate different application components inside the Cluster. For example, you have an application with frontend and backend so you will create a namespace and apply the network policy on that namespace to make the frontend and backend application secure.
- **Microservices** : Network Policies help to make microservices architecture more secure and prevent unauthorized communications between different microservices.
- **Compliance** : For Compliance reasons, you have to follow the protocol in which only authorized pods can communicate with each other. Network policies help you to achieve this.

- **Multi-tenancy** : Network Policies make sure that it will not interfere with different tenants. So, the flow of the multi-tenant cluster will be working smoothly if anything fails on the particular tenant.
- **Application testing** : Suppose, if you are testing something of an application then, Network Policy helps to control its access to production services, reducing the risks of unintended interactions.

Hands-On Demo:

To perform the hands-on Network Policy, Kubernetes must have network plugins like Calico or Cilium. By default noop network plugin is installed that does not provide advanced or rich features of Kubernetes Networking.

In the below steps, we will install the Cilum networking plugin to perform our demo. Without any advanced networking plugin, we can't able to perform a demo.

minikube start --network-plugin=cni

curl -LO https://github.com/cilium/cilium-cli/releases/latest/download/cilium-linux-amd64.tar.gz

sudo tar xzvfC cilium-linux-amd64.tar.gz /usr/local/bin

rm cilium-linux-amd64.tar.gz

cilium install

To validate whether your networking pod is running or not, run the below command.

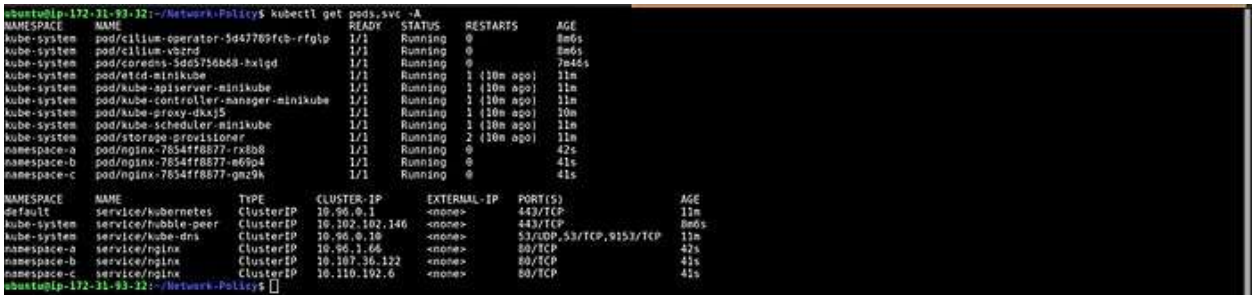
kubectl get pods — namespace=kube-system -l k8s-app=cilium

Create three namespaces and deploy the nginx pod on those namespaces with service where we have to expose port 80

```
kubectl create namespace namespace-a
kubectl create deployment nginx - image=nginx - namespace namespace-a
kubectl expose deployment nginx - port=80 - namespace namespace-a
kubectl create namespace namespace-b
kubectl create deployment nginx - image=nginx - namespace namespace-b
kubectl expose deployment nginx - port=80 - namespace namespace-b
kubectl create namespace namespace-c
kubectl create deployment nginx - image=nginx - namespace namespace-c
kubectl expose deployment nginx - port=80 - namespace namespace-c
```

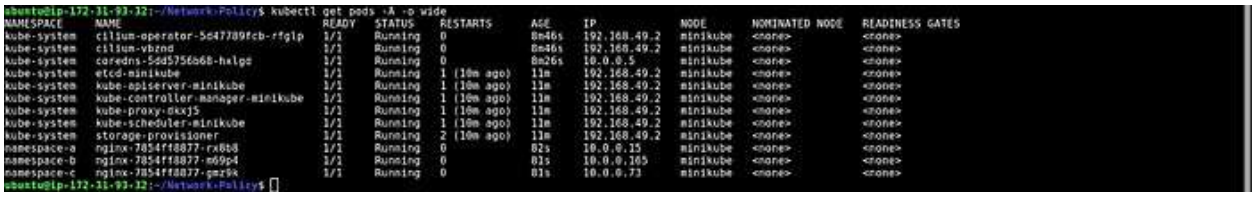
Check whether the pods are running or not of all three namespaces.

```
kubectl get pods -A
```



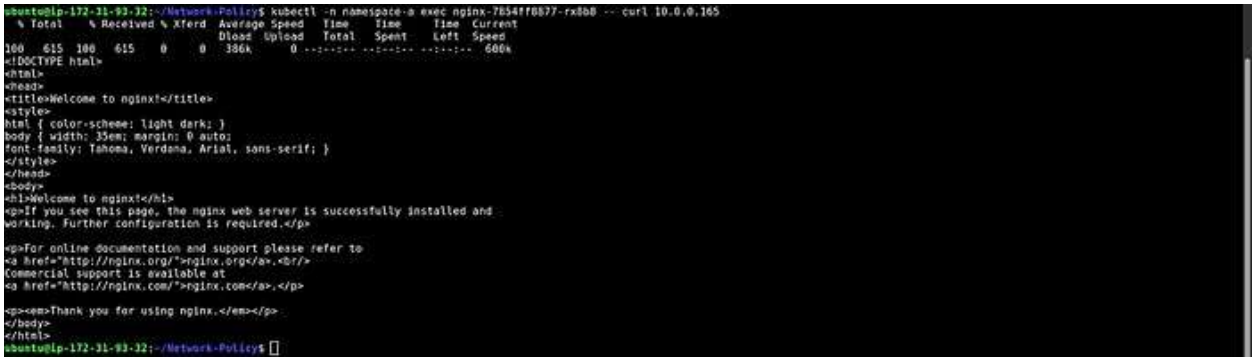
List the Private IPs of all three pods running from all three namespaces.

```
kubectl get pods -A -o wide
```

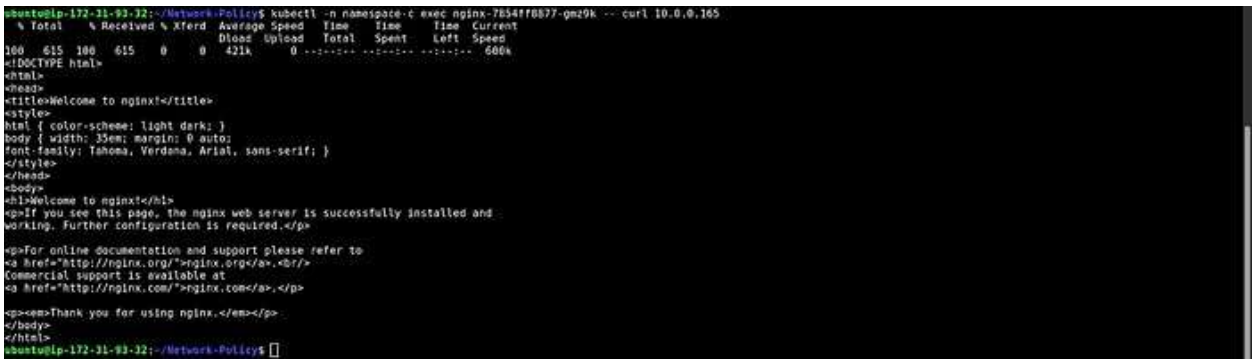


Now, try to access the pod of namespace-b from the namespace-a pod

```
kubectl -n namespace-c exec <namespace-c_pod_name> - curl
<namespace-a_pod_private_ip>
kubectl -n namespace-c exec nginx-77b4fdf86c-v4qdd - curl 10.244.0.106
```



Now, try to access the pod of namespace-b from the namespace-c pod



As you saw in the above two steps, different namespace pods are able to access the other namespace's pods which is not good DevOps practice. Let's first try to implement where the namespace-b pod can't be accessible to any other pod.

Deny All

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: deny-all-traffic
 namespace: namespace-b
spec:
 podSelector: {}
 policyTypes:
 -Ingress
 -Egress
```

Deploy the NetworkPolicy to deny all access for namespace-b

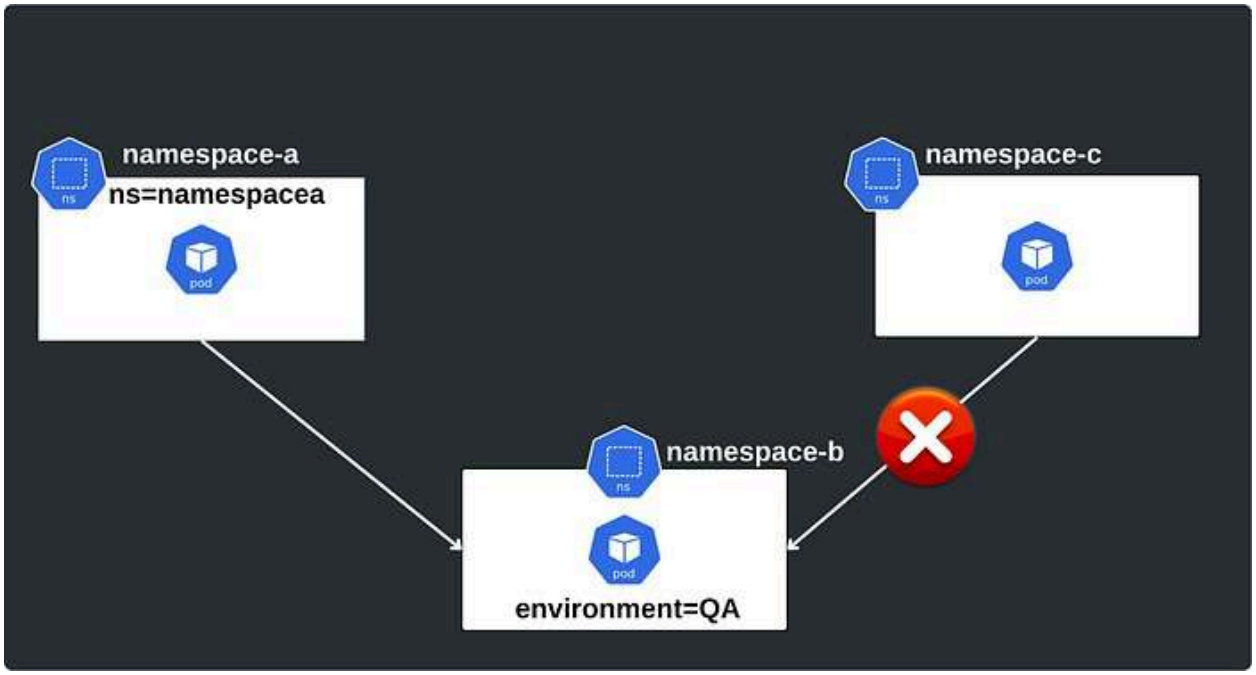
After deploying the network policy, now try to access the namespace-b pod from both namespace-a and namespace-c pod and you will see in the below snippet that you can't access the namespace-b pod which is expected.

```
ubuntu@ip-172-31-93-32:~/Network-Policy$ vim Deny-All.yml
ubuntu@ip-172-31-93-32:~/Network-Policy$
ubuntu@ip-172-31-93-32:~/Network-Policy$ kubectl apply -f Deny-All.yml
networkpolicy.networking.k8s.io/deny-all-traffic created
ubuntu@ip-172-31-93-32:~/Network-Policy$
ubuntu@ip-172-31-93-32:~/Network-Policy$ kubectl -n namespace-c exec nginx-7854ff8877-gmz9k -- curl 10.0.0.165
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:00:06 --:--:--     0^C
ubuntu@ip-172-31-93-32:~/Network-Policy$
ubuntu@ip-172-31-93-32:~/Network-Policy$
ubuntu@ip-172-31-93-32:~/Network-Policy$ kubectl -n namespace-a exec nginx-7854ff8877-rx8b8 -- curl 10.0.0.165
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:00:49 --:--:--     0^C
ubuntu@ip-172-31-93-32:~/Network-Policy$
```

Once you delete the Network Policy, then try to access the namespace-b pod from the other pod and you will see that you can access the namespace-b pod.

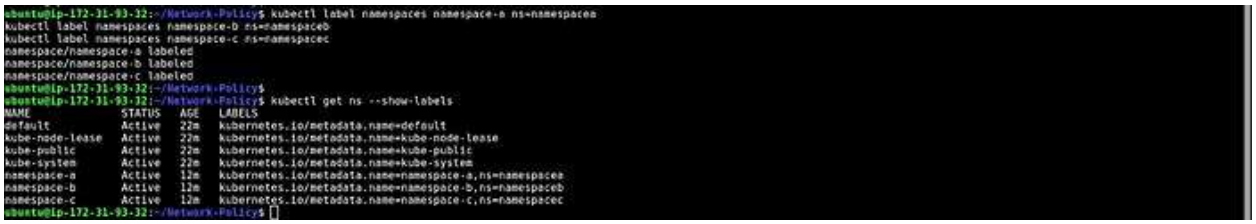
```
ubuntu@ip-172-31-93-32:~/Network-Policy$ kubectl delete -f Deny-All.yml
networkpolicy.networking.k8s.io "deny-all-traffic" deleted
ubuntu@ip-172-31-93-32:~/Network-Policy$
ubuntu@ip-172-31-93-32:~/Network-Policy$ kubectl -n namespace-a exec nginx-7854ff8877-rx8b8 -- curl 10.0.0.165
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   615  100   615    0     0   545k      0 --:--:-- --:--:-- --:--:--  600k
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
ubuntu@ip-172-31-93-32:~/Network-Policy$
```

Now, let's try to impose accessibility like the one below snippet where the namespace-c pod can only access the namespace-b pod. If namespace-a tries to access namespace-b pod then our goal is to prevent the access.



Add the label to all three K8s namespaces
namespaces namespace-a ns=namespacea kubectl lab
namespaces namespace-b ns=namespaceb

kubectl label namespaces namespace-c ns=namespacec



Now, Our motive is that only namespace-a can be able to access the namespace-b pod whereas the namespace-c can't be able to access the namespace-b pod. Let's implement the Network Policy.

Add the label environment=QA to the namespace-b pod because there can be multiple pods running inside one namespace. So, if you have to give access to the particular pod instead of all.

kubectl get pods - namespace namespace-b
kubectl label - namespace namespace-b pod nginx-7854ff8877-m69p4 environment=QA
kubectl get pods - namespace namespace-b - show-labels

Now, try to access namespace-b pod from namespace-a where it should be accessible.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: nginx-ingress
 namespace: namespace-b
spec:
 podSelector:
  matchLabels:
    environment: QA
 policyTypes:
 -Ingress
 ingress:
 -from:
  -namespaceSelector:
      matchLabels:
        ns: namespacea
```

kubectl apply -f Allow-namsepacea.yml
kubectl -n namespace-a exec nginx-7854ff8877-rx8b8 - curl 10.0.0.165

```
ubuntu@ip-172-31-93-32:~/Network-Policy$ kubectl apply -f Allow-namespaceb.yml
networkpolicy.networking.k8s.io/nginx-ingress created
ubuntu@ip-172-31-93-32:~/Network-Policy$
ubuntu@ip-172-31-93-32:~/Network-Policy$ kubectl -n namespace-a exec nginx-7854ff8877-rx8b8 -- curl 10.0.0.165
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   615  100   615    0     0   337k      0 --:--:-- --:--:-- --:--:--  600k
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
ubuntu@ip-172-31-93-32:~/Network-Policy$
```

Now, Let’s try to access namespace-b pod from namespace-c.
kubectl -n namespace-c exec nginx-7854ff8877-gmz9k — curl 10.0.0.165

As you can see in the below snippet, namespace-c pod could not able to access namespace-b pod which is expected.

```
ubuntu@ip-172-31-93-32:~/Network-Policy$ kubectl -n namespace-c exec nginx-7854ff8877-gmz9k -- curl 10.0.0.165
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:00:05 --:--:--     0
```

Kubernetes Operators

What are Operators?

Kubernetes Operator is a method for packaging or bundling, deploying, and managing the application by extending the functionality of the Kubernetes API.

The above definition is One and a half line but if you deep dive into that line you will see there are a lot of things which is followed to write this definition. Let's try to understand it.

There are two types of applications Stateless and Stateful Applications.

Stateless applications are those applications where the data or persisting data is not the priority. So, if the pods get restarted then it will lose the data but it won't bother us because we already knew that our application is Stateless.

But Stateful applications are those applications where the data is very important and we have to keep our data persistent whether its in Persistent Volume or somewhere else. But whenever the pod is replaced or restarted then the pod will lose the data which is not what we want. To get rid of the data loss, we will use Stateful Applications.

First of all, this is only one usage of the Operators. There will be more usages of the Kubernetes Operators which, we will discuss later in this blog.

Kubernetes Operators have rich features that help to deploy stateless, stateful, complex, or custom applications.

Features of Operators:

1. **Custom Resource Definitions(CRD)** : With the help of Operators, you can define your CRD to extend the capability of your Kubernetes for a particular application. We will discuss CRD in a detailed way later.

   Example : Prometheus is one of the finest Operators, in which there is a Custom Resource known as Prometheus which allows users to define the monitor configuration declaratively. You can specify the details like alerting rules, service monitors, etc using the custom resource.

2. **Custom Controllers** : There is no use of Custom resource if the Custom Controller is not present. Operators implement custom controllers to watch and reconcile(correct) the state of the custom resource and ensure that the desired state matches the actual state.

   Example : In Native Kubernetes, etcd is one of the components that take care of the current and desired state thing and other things like stability of the Kubernetes Cluster and node failures, etc.

3. **Automated Operations** : Operators automate routine tasks which makes it easier to manage the complex applications over their lifecycle.

Example : In the MongoDB operator, if the users increase the number of replicas in the custom resource then, the Operator automatically adjusts the replicas to seamless scaling.

4. **Operational Policies** : Operators consistently enforce the operational policies to keep the environment secure across instances of application.
   Example : The Vault operators enforce the security policies over HashiCorp Vault which ensures that the sensitive data will be secured.

5. **Rolling Updates and Upgrades** : Operators manage the rolling updates and upgrades of the application. So that, there will be no downtime.
   Example : CockroachDB operator handles the rolling updates on the CockroachDB cluster one node at a time to follow the no downtime protocol during the upgrading hours..

6. **Integrate with Ecosystem tools** : Operators integrate with other Kubernetes tools to provide better functionality for the application.
   Example : Prometheus Operator can be integrated with Grafana to provide complete monitoring.

7. **Stateful Applications:** Operators help to manage complex or stateful applications by handling tasks like data persistent, disaster recovery, scaling, etc.
   Example : Apache Kafka Operator manages Kafka clusters which automates tasks such as topic creation, partition reassignment, etc.
   Kubernetes is one of the best container orchestration tools because of the Operator feature. If you observe, to leverage the rich feature you have to install operators like to get the benefit of the HPA(Horizontal Pod Autoscaler) feature you need to install an operator for it, and the same for Network Policy, etc.
   So, without Operators, the life of a Kubernetes DevOps guy is not very easy.

Now this is enough to get a basic understanding of Kubernetes Operators. But there are some more things that you need to know.

1. Custom Resource Definition

CRD is indeed one of the powerful features of Kubernetes, acting like a superpower that lets you define and use custom resources tailored to your specific needs. Let’s put it in a scenario:

Imagine you have a fantastic application ready to roll, but Kubernetes, as amazing as it is, might not have all the necessary tools and features to handle the uniqueness of your application. Here’s where CRD steps in as your superhero sidekick.

In straightforward terms, CRD allows you to create your very own resource types in Kubernetes. It’s like getting a custom tool for your specific job. But, of course, there’s a twist —your CRD needs the Kubernetes seal of approval. Think of it as a passport check; once

your CRD gets the nod from Kubernetes, it becomes a certified member of the Kubernetes family.

Now, here's the exciting part. Once you've crafted your CRD to match your application's needs, you can share it with the world! Just like posting your creation on a hub for others to benefit. In the Kubernetes world, this hub is known as operatorshub.io. It's a place where your CRD can shine, offering its capabilities to others who might have similar challenges.

So, in a nutshell, CRD empowers you to extend Kubernetes by creating your custom resources, and once validated, you can share your creations on operatorshub.io, contributing to the Kubernetes ecosystem and helping others tackle their unique challenges. It's like giving your application its very own set of superpowers within the Kubernetes universe!

Custom Resource Definition can be created in YAML

2. Custom Controller

Without a Custom Controller, there is no benefit to using Custom Resource Definition. Custom Controller is a dedicated created for the CRD. The main work of the Custom Controller is to meet the desired state of the custom resources with the actual state such as replicates. Also, Custom Controllers are like a watcher who watches and keep an eye on the custom resource. So, if there is any misshappening occurs then Custom Controllers fix it or take the necessary steps as part of the auto healing.

Custom Controller can be created in many languages like Go, Python, etc and there is one component client-go that is dedicated to Golang which has all the necessary tools for working with Kubernetes.

3. Custom Resource

Once the Custom Controller is deployed, you have to think or you have already prepared the roadmap for how many namespaces you have to deploy your Custom Resource. It can be deployed on multiple worker nodes as per the requirement. Custom Resource is the last step in the process of creating and deploying the Custom Resources where Custom Controller is the second and CRD is the first step.

Let's go through a sample **Custom Resource Definition**

Ascenario could be managing a custom application called "AwesomeApp" that needs to maintain a specific number of replicas based on a defined metric. Here's how the CRD might look:

```
metadata:
  name: awesomeapps.app.example.com
spec:
  group: app.example.com
  names:
    kind: AwesomeApp
    listKind: AwesomeAppList
```

```
  plural: awesomeapps
  singular: awesomeapp
 scope: Namespaced
 versions:
  -name: v1
    served: true
    storage: true
 additionalPrinterColumns:
  -name: Replicas
    type: integer
    JSONPath: .spec.replicas
```

In this CRD, we define a custom resource named “AwesomeApp” belonging to the group “app.example.com.” It has a field for the number of replicas.

Let’s go through a sample Custom Controller:

Now, let’s create a simple Custom Controller in Go that watches for changes to our custom resource and takes action accordingly.

```
// File: main.go

import (
"context"
"flag"
"fmt"
"time"
"k8s.io/client-go/kubernetes"
"k8s.io/client-go/tools/cache"
"k8s.io/client-go/tools/clientcmd"
"k8s.io/client-go/util/homedir"
"k8s.io/client-go/util/wait"
"k8s.io/client-go/util/workqueue"
)
func main() {
kubeconfig, _ := getKubeConfig()
config, err := clientcmd.BuildConfigFromFlags("", kubeconfig)
if err != nil {
 panic(err.Error())
}
clientset, err := kubernetes.NewForConfig(config)
if err != nil {
 panic(err.Error())
}
informer := cache.NewSharedInformer(
 cache.NewListWatchFromClient(
  clientset.AppsV1().RESTClient(),
  "awesomeapps",
  "namespace-name",
  cache.ResourceEventHandlerFuncs{
   AddFunc: func(obj interface{}) {
```

```
    fmt.Println("AwesomeApp created:", obj
   // Logic to handle the creation of AwesomeApp
  },
  UpdateFunc: func(oldObj, newObj interface{}) {
  fmt.Println("AwesomeApp updated:", newObj)
  // Logic to handle the update of AwesomeApp
  },
  DeleteFunc: func(obj interface{}) {
   fmt.Println("AwesomeApp deleted:", obj)
   // Logic to handle the deletion of AwesomeApp
  },
 },
),
 &v1.AwesomeApp{},
 0, // no resync
 cache.ResourceEventHandlerFuncs{},
)
stopCh := make(chan struct{})
defer close(stopCh)
go informer.Run(stopCh)
// Run forever
select {}
}
func getKubeconfig() (*string, error) {
home := homedir.HomeDir()
kubeconfig := flag.String("kubeconfig", home+"/.kube/config", "absolute path to the
kubeconfig file")
flag.Parse()
return kubeconfig, nil
}
```

In this simplified example:

- The controller watches for changes in AwesomeApp resources.
- When an AwesomeApp resource is created, updated, or deleted, the corresponding handler functions are invoked.
  You would extend this controller with your custom logic to manage the AwesomeApp replicas based on the specified metric.

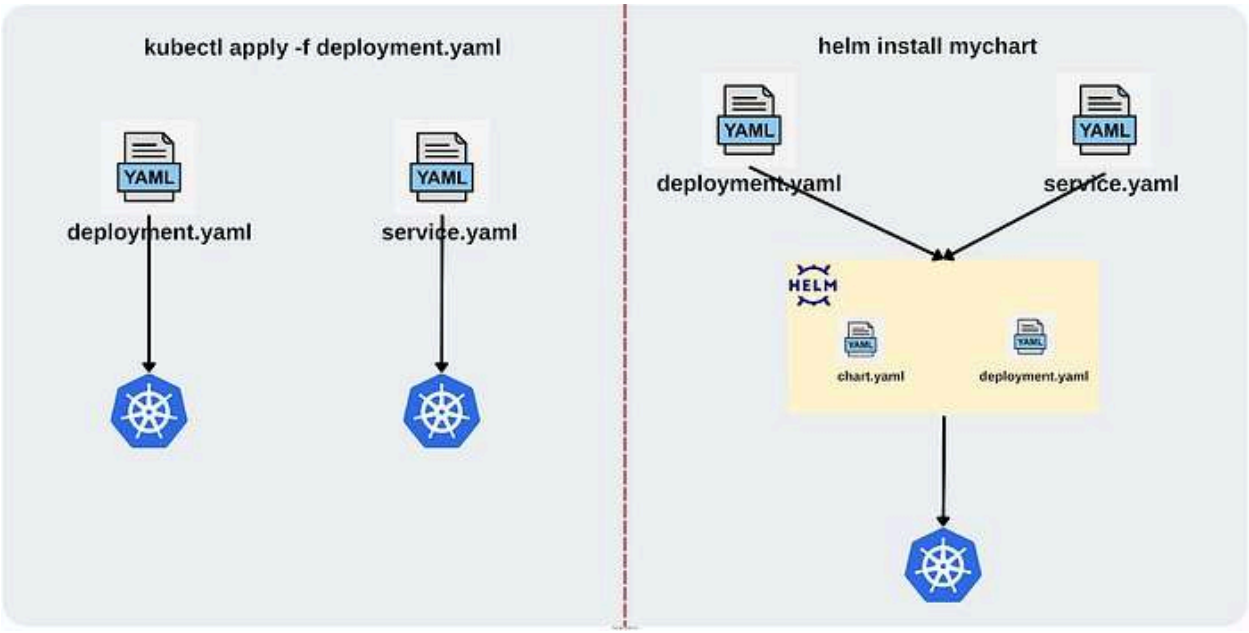
Helm & Helm Charts

Helm

Helm is a Kubernetes package manager in which the multiple numbers of YAML files such as the backend, and frontend come under one roof(helm) and deploy using helm.

Let’s understand with the help of a simple example.

Suppose you have an application where frontend, backend, and database code needs to deploy on Kubernetes. Now, the task becomes hectic if you have to deploy frontend and backend codes because your application will be Stateful. For frontend, backend, and database you will have to create different YAML files and deploy them too but it will be complicated to manage. So, Helm is an open-source package manager that will help you to automate the deployment of applications for Kubernetes in the simplest way.



Let’s understand again with the help of the above architecture.

Normal deployment

As you know, to deploy your code you need to write a minimum of two YAML files which is deployment and service file. Those files will be deployed with the help of the kubectl command. These files act differently from each other but you know that the files are dependent on each other.

Helm deployment

In the help deployment, all the YAML files related to the application deployment will be in the helm chart. So, if you want to deploy your application you don't need to deploy each YAML file one by one. Instead, you can just write one command helm install <your-chart-name> and it will deploy your entire application in one go. Helm is the Package manager for the Kubernetes which helps to make the deployment simple.

Benefits

- Because of the helm, the time will be saved.
- It makes the automation more smoothly
- Reduced complexity of deployments.
- Once files can be implemented in different environments. You just need to change
- the values according to the environmental requirements.
- Better scalability.
- Perform rollback at any time.
- Effective for applying security updates.
  Increase the speed of deployments and many more.

Hands-On

Install Helm

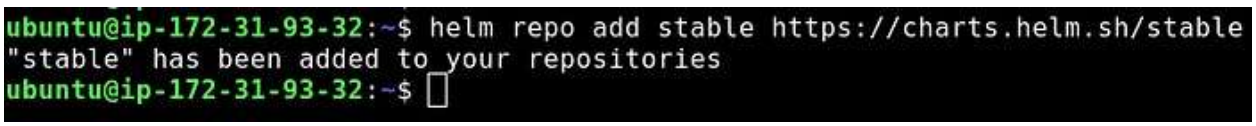
```
curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
helm version
```

To list the all repositories

```
helm list
```

Create a stable repository using the helm

```
helm repo add stable https://charts.helm.sh/stable
```



List the repo, again

helm repo list

```
ubuntu@ip-172-31-93-32:~$ helm repo list
NAME    URL
stable  https://charts.helm.sh/stable
```

To remove the repo

helm repo remove stable

```
ubuntu@ip-172-31-93-32:~$ helm repo remove stable
"stable" has been removed from your repositories
ubuntu@ip-172-31-93-32:~$
ubuntu@ip-172-31-93-32:~$ helm repo list
Error: no repositories to show
ubuntu@ip-172-31-93-32:~$
```

To create our own repo

helm create my-repo

```
ubuntu@ip-172-31-93-32:~$ helm create my-repo
Creating my-repo
ubuntu@ip-172-31-93-32:~$
ubuntu@ip-172-31-93-32:~$ ls my-repo/
Chart.yaml  charts  templates  values.yaml
ubuntu@ip-172-31-93-32:~$
```

To see the files inside the created repo

```
ubuntu@ip-172-31-93-32:~$ tree my-repo/
my-repo/
├── Chart.yaml
├── charts
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

3 directories, 10 files
```

Let’s do a simple demo where we will host the nginx page on Kubernetes using the helm chart.

Create a chart using the command

helm create helloworld

```
amanpathak@pop-os:~/Learning/Kubernetes/Helm$ helm create helloworld
Creating helloworld
amanpathak@pop-os:~/Learning/Kubernetes/Helm$
amanpathak@pop-os:~/Learning/Kubernetes/Helm$ ls
helloworld
amanpathak@pop-os:~/Learning/Kubernetes/Helm$
```

The file structure should look like the below snippet.

```
amanpathak@pop-os:~/Learning/Kubernetes/Helm$ tree helloworld/
helloworld/
├── charts
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   ├── service.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

3 directories, 10 files
amanpathak@pop-os:~/Learning/Kubernetes/Helm$
```

Go to the Helm Chart directory helloworld and edit the values.yaml file

cd helloworld

vim values.yaml

```
amanpathak@pop-os:~/Learning/Kubernetes/Helm$ cd helloworld/
amanpathak@pop-os:~/Learning/Kubernetes/Helm/helloworld$
amanpathak@pop-os:~/Learning/Kubernetes/Helm/helloworld$ vim values.yaml
amanpathak@pop-os:~/Learning/Kubernetes/Helm/helloworld$
```

Replace the ClusterIP with NodePort

```
41
42 service:
43   type: NodePort
44   port: 80
45
```

Now deploy your helm chart

To do that, you have to be present in the directory where the chart is present.

In my case, my helloworld chart is in the Helm directory. So, I have to be in the Helm directory and run the below command to install the Helm Chart.

```
helm install thehelloworld helloworld
```

```
helm install <custom-chart-release-name> <given-chart-name>
```

```
amanpathak@pop-os:~/Learning/Kubernetes/Helm$ helm install thehelloworld helloworld
NAME: thehelloworld
LAST DEPLOYED: Fri Nov 24 18:49:46 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services thehelloworld)
  export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
  echo http://$NODE_IP:$NODE_PORT
amanpathak@pop-os:~/Learning/Kubernetes/Helm$
```

Now, check the services whether your application is running or not

```
kubectl get svc
```

```
amanpathak@pop-os:~/Learning/Kubernetes/Helm$ kubectl get svc
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes     ClusterIP   10.96.0.1        <none>        443/TCP        42d
thehelloworld  NodePort    10.111.180.153   <none>        80:30738/TCP   16m
amanpathak@pop-os:~/Learning/Kubernetes/Helm$
```

Now, access your application via browser by copying the port number(in my case 30738) with minikube private IP in starting.

To get the IP, you can simply run the command on the terminal minikube ip and use it to view the content.

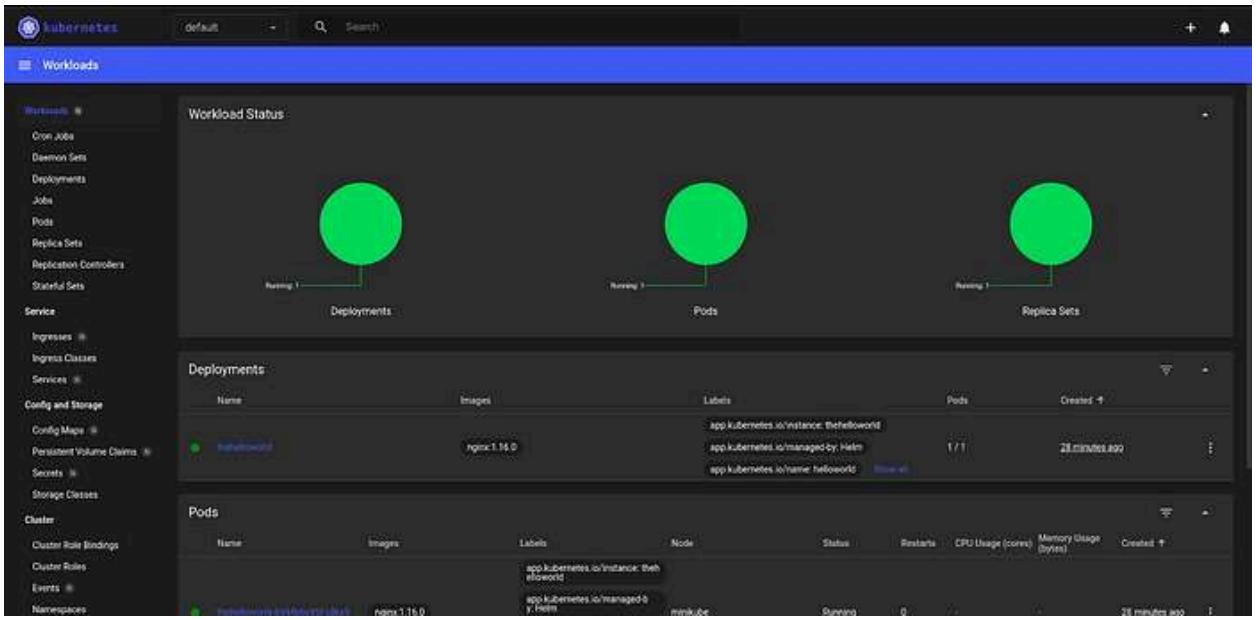
As you can see in the below snippet, our application is successfully deployed with the help of Helm Chart.

If you want to see the minikube dashboard GUI then run the below command on the terminal.

```
minikube dashboard
```

```
amanpathak@pop-os:~/Learning/Kubernetes/Helm$ minikube dashboard
🤔  Verifying dashboard health ...
🚀  Launching proxy ...
🤔  Verifying proxy health ...
🎉  Opening http://127.0.0.1:39115/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
Opening in existing browser session.
```

Once you run the command, a new tab will open in your browser that will look like the below snippet.



Now, if you want to uninstall your deployment. You can simply run the below command.

helm uninstall thehelloworld

```
amanpathak@pop-os:~/Learning/Kubernetes/Helm$ helm uninstall thehelloworld
release "thehelloworld" uninstalled
amanpathak@pop-os:~/Learning/Kubernetes/Helm$
```

Once you uninstall the deployment you will see nothing on your K8s dashboard because the deployment has been deleted.

Demo of Helm Cheat Sheets

Create the helm chart

helm create helloworld

```
amanpathak@pop-os:~/Learning/Kubernetes/Helm$ helm create helloworld
Creating helloworld
amanpathak@pop-os:~/Learning/Kubernetes/Helm$
```

Create the release and deploy the helm chart

```
helm install thehelloworld helloworld
```

Now, if you want 2 replicas instead of 1. So you will make the changes in the values.yaml file and replace the replica 2 with 1. But you have to redeploy the changes which will be done by the below command after doing the changes.

```
helm upgrade thehelloworld helloworld
```

Now, you think that you did something wrong or unexpected and you want to switch to the previous deployment. To do that, first of all, you need to know that to which revision number you have to switch.

To check the current revision use the below command.

```
helm list -a
```

Now, I want to switch to the 1 Revision number. So, I will roll back the changes by the below command.

```
helm rollback thehelloworld 1
```

If you want to dry-run the chart before installation use the below command.

```
helm install thehelloworld — debug — dry-run helloworld
```

```
amanpathak@pop-os:~/Learning/Kubernetes/Helm$ helm install thehelloworld --debug --dry-run helloworld
install.go:214: [debug] Original chart version: ""
install.go:231: [debug] CHART PATH: /home/amanpathak/Learning/Kubernetes/Helm/helloworld

NAME: thehelloworld
LAST DEPLOYED: Fri Nov 24 19:56:04 2023
NAMESPACE: default
STATUS: pending-install
REVISION: 1
USER-SUPPLIED VALUES:
{}

COMPUTED VALUES:
affinity: {}
autoscaling:
  enabled: false
  maxReplicas: 100
  minReplicas: 1
  targetCPUUtilizationPercentage: 80
fullnameOverride: ""
image:
  pullPolicy: IfNotPresent
  repository: nginx
  tag: ""
imagePullSecrets: []
ingress:
  annotations: {}
  className: ""
  enabled: false
  hosts:
  - host: chart-example.local
    paths:
    - path: /
      pathType: ImplementationSpecific
  tls: []
nameOverride: ""
nodeSelector: {}
podAnnotations: {}
podLabels: {}
podSecurityContext: {}
replicaCount: 2
resources: {}
securityContext: {}
service:
  port: 80
  type: ClusterIP
serviceAccount:
```

If you want to validate your YAML files within the helm chart then, use the below command.

helm template helloworld

```
amanpathak@pop-os:~/Learning/Kubernetes/Helm$ helm template helloworld
---
# Source: helloworld/templates/serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: release-name-helloworld
  labels:
    helm.sh/chart: helloworld-0.1.0
    app.kubernetes.io/name: helloworld
    app.kubernetes.io/instance: release-name
    app.kubernetes.io/version: "1.16.0"
    app.kubernetes.io/managed-by: Helm
automountServiceAccountToken: true
---
# Source: helloworld/templates/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: release-name-helloworld
  labels:
    helm.sh/chart: helloworld-0.1.0
    app.kubernetes.io/name: helloworld
    app.kubernetes.io/instance: release-name
    app.kubernetes.io/version: "1.16.0"
    app.kubernetes.io/managed-by: Helm
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: http
      protocol: TCP
      name: http
  selector:
    app.kubernetes.io/name: helloworld
    app.kubernetes.io/instance: release-name
---
# Source: helloworld/templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: release-name-helloworld
  labels:
    helm.sh/chart: helloworld-0.1.0
    app.kubernetes.io/name: helloworld
    app.kubernetes.io/instance: release-name
    app.kubernetes.io/version: "1.16.0"
```

If you want to validate your Charts then use the below command.

helm lint helloworld

```
amanpathak@pop-os:~/Learning/Kubernetes/Helm$ helm lint helloworld
==> Linting helloworld
[INFO] Chart.yaml: icon is recommended

1 chart(s) linted, 0 chart(s) failed
amanpathak@pop-os:~/Learning/Kubernetes/Helm$
```

If you want to delete or release the deployment then, use the below command.

```
helm uninstall thehelloworld
```

Deploy Flask Application using Helm Chart and many more features

---

Docker Project of Python Flask Repo- https://github.com/AmanPathak-DevOps/Docker-Projects/tree/master/Python-Project-for-Helm-K8s

Kubernetes Manifest file Repo- https://github.com/AmanPathak-DevOps/Kubernetes-files

In the Previous Chapter, we have covered some theory and basic hands-on. But today, we will deep dive and do more hands-on.

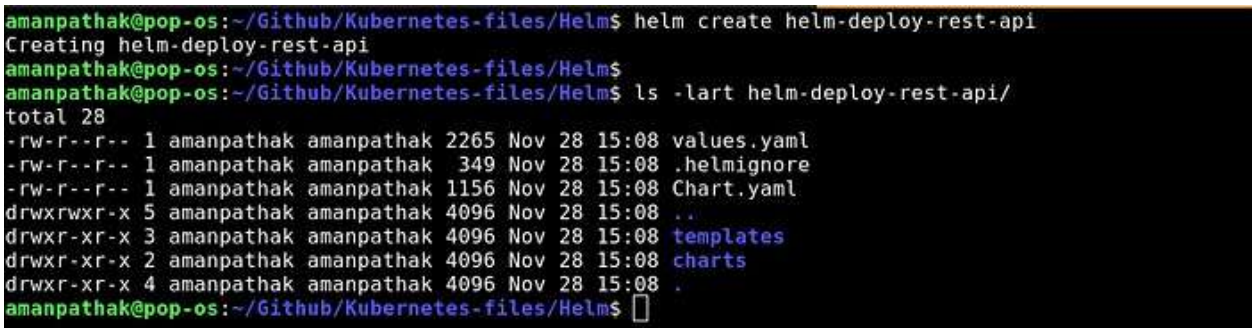
The topics that we will cover here:

- ✅ Deploy Python Flask Application using Helm Chart
- ✅ What is Helmfile?
- ✅ Demo of Helmfile(Deploy HelmCharts using declarative method)
- ✅ Test Cases for your Helm Chart

Demonstration

Create a helm chart for the Python application by using the below command

```
helm create helm-deploy-rest-api
ls -lart Helm-Deploy-Rest-API
```



Comment Out the appVersion in the Chart.yaml file

```
vim helm-deploy-rest-api/Chart.yaml
```

```
14
15 # This is the chart version. This version number should be incremented each time you make change
16 # to the chart and its templates, including the app version.
17 # Versions are expected to follow Semantic Versioning (https://semver.org/)
18 version: 0.1.0
19
20 # This is the version number of the application being deployed. This version number should be
21 # incremented each time you make changes to the application. Versions are not expected to
22 # follow Semantic Versioning. They should reflect the version the application is using.
23 # It is recommended to use it with quotes.
24 #appVersion: "1.16.0"
```

Replace the image repository with nginx

vim helm-deploy-rest-api/values.yaml

```
 3 # Declare variables to be passed into your templates.
 4
 5 replicaCount: 1
 6
 7 image:
 8   repository: avian19/python-flask-application:latest
 9   pullPolicy: IfNotPresent
10   # Overrides the image tag whose default is the chart appVersion.
11   tag: ""
12
13 imagePullSecrets: []
14 nameOverride: ""
```

Replace the service type from ClusterIP to NodePort in the same values.yaml file

```
38   # readOnlyRootFilesystem: true
39   # runAsNonRoot: true
40   # runAsUser: 1000
41
42 service:
43   type: NodePort
44   port: 80
45
46 ingress:
47   enabled: false
48   className: ""
49   annotations: {}
50     # kubernetes.io/ingress.class: nginx
```

vim helm-deploy-rest-api/templates/deployment.yaml

Remove the appversion set and write only the image repository name

```
34         - name: {{ .Chart.Name }}
35           securityContext:
36             {{- toYaml .Values.securityContext | nindent 12 }}
37           image: "{{ .Values.image.repository }}"
38           imagePullPolicy: {{ .Values.image.pullPolicy }}
39           ports:
40             - name: http
```

In the same deployment.yaml file.

Modify the port to 9001 according to our application

```
38          imagePullPolicy: {{ .Values.image.pullPolicy }}
39          ports:
40            - name: http
41              containerPort: 9001
42              protocol: TCP
43          livenessProbe:
44            httpGet:
```

Now in the same deployment.yaml file comment out all the liveness probe and readiness probe function

```
43              #livenessProbe:
44            #httpGet:
45              #path: /
46                # port: http
47                #readinessProbe:
48            # httpGet:
49              #path: /
50                #    port: http
```

Now, install your helm chart

```
helm install pythonhelm helm-deploy-rest-api/
```

```
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$ helm install pythonhelm helm-deploy-rest-api/
NAME: pythonhelm
LAST DEPLOYED: Tue Nov 28 15:12:59 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services pythonhelm-helm-deploy-rest-api)
  export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
  echo http://$NODE_IP:$NODE_PORT
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$
```

Now, check whether the pod is running or not

```
kubectl get pods
```

```
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$ kubectl get pods
NAME                                          READY   STATUS    RESTARTS   AGE
pythonhelm-helm-deploy-rest-api-5fc585d4-c5k8n   1/1     Running   0          53m
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$
```

Check the service to get the port number

```
kubectl get svc
```

```
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$ kubectl get svc
NAME                            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
kubernetes                      ClusterIP   10.96.0.1      <none>        443/TCP        45d
pythonhelm-helm-deploy-rest-api   NodePort    10.98.22.103   <none>        80:30454/TCP   55m
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$
```

Now, get the minikube ip using the minikube ip command and paste it on the browser to get the content of the application

Use the ‘/main’ because this is our path where content is present. Otherwise, you will get errors on different paths.

You can uninstall the release for the helm chart

```
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$ helm uninstall pythonhelm
release "pythonhelm" uninstalled
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$
```

Helmfile

Earlier, we used to deploy helm charts in an imperative way. But if you want to deploy helm charts using a declarative way then we will use Helmfile. Helmfile also helps to deploy multiple charts in one go.

Demo of Helmfile

Install the file

```
wget https://github.com/roboll/helmfile/releases/download/v0.144.0/helmfile_linux_amd64
```

Rename the file name to helmfile

mv helmfile_linux_amd64 helmfile

```
amanpathak@pop-os:~/Downloads$ mv helmfile_linux_amd64 helmfile
```

Change the permissions of the helmfile

```
chmod 777 helmfile
```

```
amanpathak@pop-os:~/Downloads$ chmod 777 helmfile
amanpathak@pop-os:~/Downloads$
```

Now, move the helmfile to the bin folder

```
mv helmfile /usr/local/bin
```

```
amanpathak@pop-os:~/Downloads$ sudo mv helmfile /usr/local/bin
```

Validate the version by the command

```
helmfile — version
```

```
amanpathak@pop-os:~/Downloads$ helmfile --version
helmfile version v0.144.0
```

Now, We will try to deploy our previous Python application using helmfile.

```
releases:
 -name: pythonpr
   chart: ./helm-deploy-rest-api
   installed: true
```

```
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$ cat helmfile.yaml
---
releases:

  - name: pythonpr
    chart: ./helm-deploy-rest-api
    installed: true
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$
```

Once you run the command helmfile sync, your chart will be deployed.

```
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$ helmfile sync
Building dependency release=pythonpr, chart=helm-deploy-rest-api
Affected releases are:
  pythonpr (helm-deploy-rest-api) UPDATED

Upgrading release=pythonpr, chart=helm-deploy-rest-api
Release "pythonpr" does not exist. Installing it now.
NAME: pythonpr
LAST DEPLOYED: Tue Nov 28 17:35:28 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services pythonpr-helm-deploy-rest-api)
  export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
  echo http://$NODE_IP:$NODE_PORT

Listing releases matching ^pythonpr$
pythonpr        default         1               2023-11-28 17:35:28.520188194 +0530 IST deployed        helm-deploy-rest-api-0.1.0


UPDATED RELEASES:
NAME       CHART                VERSION
pythonpr   helm-deploy-rest-api     0.1.0
```

If you want to uninstall the release then, go to the yaml file replace the true with false in the installed line, and run helm sync.

```
---
releases:

  - name: pythonpr
    chart: ./helm-deploy-rest-api
    installed: false
~
```

As we run the command to uninstall the deployment the pod is terminating now.

```
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$ helmfile sync
Listing releases matching ^pythonpr$
pythonpr        default         1               2023-11-28 17:35:28.520188194 +0530 IST deployed        helm-deploy-rest-api-0.1.0

Affected releases are:
  pythonpr (./helm-deploy-rest-api) DELETED

Deleting pythonpr
release "pythonpr" uninstalled


DELETED RELEASES:
NAME
pythonpr
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$ kubectl get pods
NAME                                      READY   STATUS        RESTARTS   AGE
pythonpr-helm-deploy-rest-api-7ff6676c86-g9ldt   1/1     Terminating   0          6m23s
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$
```

Helmfile using Git repository

Suppose your charts are present on the Git repository and you want to install them. So, Helm provides you with a feature in which you don't need to clone the repo manually to deploy the chart. Instead of this, you just have to provide the git repository URL in the helmfile and helm will deploy the charts automatically.

Demo

To leverage this feature, you need to install one plugin for it. To install it just copy and paste the below command.

```
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$ helm plugin install https://github.com/aslafy-z/helm-git --version 0.15.1
Installed plugin: helm-git
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$
```

Now, add your repo accordingly in the yaml file

```
---
repositories:
  -name: helm-python
    url:
git+https://github.com/AmanPathak-DevOps/Kubernetes-files@Helm?ref=master&sparse=0
releases:
  -name: pythonpr
    chart: ./helm-deploy-rest-api
    installed: false
```

```
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$ cat helmfile.yaml
---
repositories:
  - name: helm-python
    url: git+https://github.com/AmanPathak-DevOps/Kubernetes-files@Helm?ref=master&sparse=0

releases:

  - name: pythonpr
    chart: ./helm-deploy-rest-api
    installed: true
```

Now, run the helmfile using the below command

helmfile sync

```
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$ vim helmfile.yaml
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$ helmfile sync
Adding repo helm-python git+https://github.com/AmanPathak-DevOps/Kubernetes-files@Helm?ref=master&sparse=0
"helm-python" has been added to your repositories

Building dependency release=pythonpr, chart=helm-deploy-rest-api
Affected releases are:
  pythonpr (helm-deploy-rest-api) UPDATED

Upgrading release=pythonpr, chart=helm-deploy-rest-api
Release "pythonpr" does not exist. Installing it now.
NAME: pythonpr
LAST DEPLOYED: Tue Nov 28 17:57:30 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services pythonpr-helm-deploy-rest-api)
  export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
  echo http://$NODE_IP:$NODE_PORT

Listing releases matching ^pythonpr$
pythonpr        default         1               2023-11-28 17:57:30.618835916 +0530 IST deployed        helm-deploy-rest-api-0.1.0


UPDATED RELEASES:
NAME       CHART                VERSION
pythonpr   helm-deploy-rest-api     0.1.0

amanpathak@pop-os:~/Github/Kubernetes-files/Helm$ kubectl get pods
NAME                                      READY   STATUS    RESTARTS   AGE
pythonpr-helm-deploy-rest-api-7ff6676c86-74kcs   1/1     Running   0          5s
amanpathak@pop-os:~/Github/Kubernetes-files/Helm$
```

Install multiple charts using helmfile

You just need to add the charts in the previous helmfile below

```
---
repositories:
  -name: helm-python
    url:
git+https://github.com/AmanPathak-DevOps/Kubernetes-files@Helm?ref=master&sparse=0
releases:
  -name: pythonpr
    chart: ./helm-deploy-rest-api
    installed: true
  -name: helloworld
    chart: ./helloworld
    installed: true
```

```
anaspathak@pop-os:~/Github/Kubernetes-files/Helm$ cat helmfile.yaml
---
repositories:
  - name: helm-python
    url: git+https://github.com/AmanPathak-DevOps/Kubernetes-files@Helm?ref=master&sparse=0

releases:

  - name: pythonpr
    chart: ./helm-deploy-rest-api
    installed: true

  - name: helloworld
    chart: ./helloworld
    installed: true
```

Now, run the command to install the charts

helmfile sync

```
anaspathak@pop-os:~/Github/Kubernetes-files/Helm$ helmfile sync
Adding repo helm-python git+https://github.com/AmanPathak-DevOps/Kubernetes-files@Helm?ref=master&sparse=0
"helm-python" has been added to your repositories

Building dependency release=helloworld, chart=helloworld
Building dependency release=pythonpr, chart=helm-deploy-rest-api
Affected releases are:
  helloworld (helloworld) UPDATED
  pythonpr (helm-deploy-rest-api) UPDATED

Upgrading release=helloworld, chart=helloworld
Upgrading release=pythonpr, chart=helm-deploy-rest-api
Release "pythonpr" does not exist. Installing it now.
NAME: pythonpr
LAST DEPLOYED: Tue Nov 28 18:35:18 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services pythonpr-helm-deploy-rest-api)
  export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
  echo http://$NODE_IP:$NODE_PORT

Listing releases matching "pythonpr$
Release "helloworld" does not exist. Installing it now.
NAME: helloworld
LAST DEPLOYED: Tue Nov 28 18:35:18 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=helloworld,app.kubernetes.io/instance=helloworld" -o jsonpath="{.items[0].metadata.name}")
  export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT

Listing releases matching "helloworld$
pythonpr        default         1               2023-11-28 18:35:18.002815238 +0530 IST deployed        helm-deploy-rest-api-0.1.0

helloworld      default         1               2023-11-28 18:35:18.002356284 +0530 IST deployed        helloworld-0.1.0        1.16.0


UPDATED RELEASES:
NAME         CHART                VERSION
pythonpr     helm-deploy-rest-api     0.1.0
helloworld   helloworld               0.1.0
```

Test your helm chart

Once I deploy the chart, I want to test the chart whether it's working or not.

So, you can define your test cases in the test-connection.yaml file which is presented in the tests folder of the chart itself.

```
anaspathak@pop-os:~/Github/Kubernetes-files/Helm$ tree helm-deploy-rest-api/
helm-deploy-rest-api/
├── charts
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   ├── service.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

3 directories, 10 files
anaspathak@pop-os:~/Github/Kubernetes-files/Helm$
```

As we have deployed our charts in the previous demo. So, we will test those charts.

To test the particular chart use the below command.

helm test <chart>

As you can see in the below snippet. Our helloworld chart test is succeeded.

AWS Elastic Kubernetes Service(EKS)

What is AWS EKS?

AWS EKS(Elastic Kubernetes Service) is a managed service that eliminates things like installation of the Kubernetes and maintaining the Kubernetes cluster.

Some basic benefits like you can focus on deployment for the applications. You don't need to think about the availability of your cluster, AWS will take care of those things.

Key features of EKS:

- **Manage Control Plane** : In EKS, the Control Plane will be managed by AWS itself.
- **Configure Node Groups:** In EKS, You can add multiple Worker Nodes according to
- the requirements in no time.
- **Cluster Scaling** : AWS will take care of the Cluster scaling according to your
- requirements whether it's upscaling or downscaling.
- **High Availability** : AWS provides high availability of your Kubernetes cluster.
  Security : AWS enhances the security by integrating IAM service with EKS.
  Networking : AWS provides better control to manage the networking stuff for your Kubernetes cluster.

If you want to read more features in a detailed way refer to the following link:

Managed Kubernetes Service - Amazon EKS Features

AWS EKS Costing

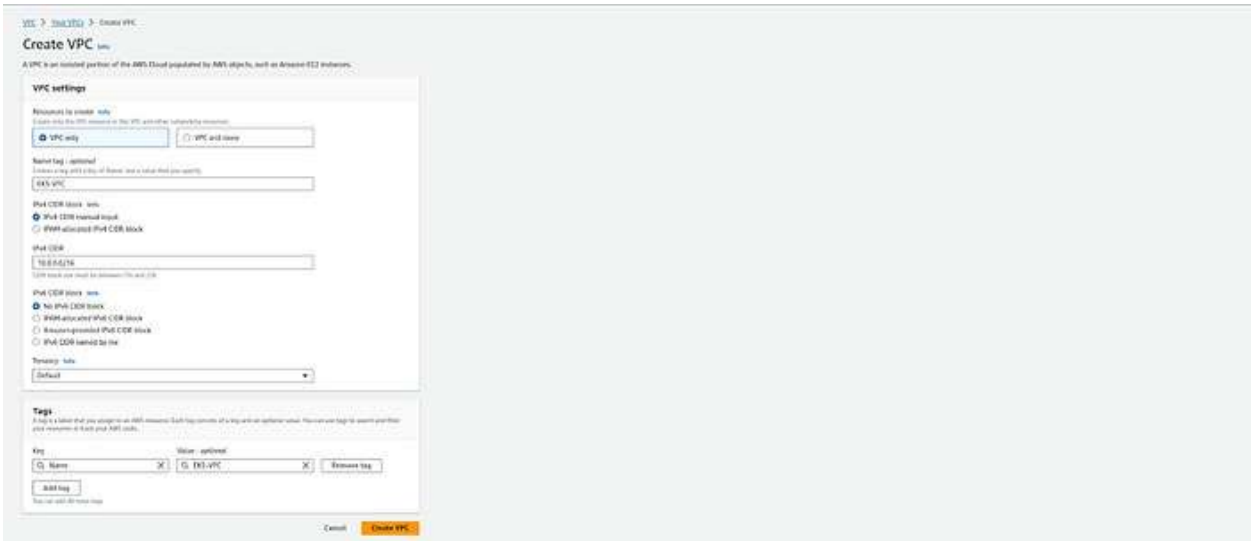
AWS will cost you 0.10$ per hour for each cluster. If you create EC2 for Node Groups then it will cost you separately according to the instance type and the same with ECS Fargate(depends on vCPU and memory resources).

🛠️

Let's Dive into the Demo!

To create EKS, we need to configure VPC and other networking things. If you are not a beginner in the cloud feel free to skip the network configuration part. But if you are new to EKS or the AWS Cloud, I would say to follow each step. So, it will help you to get a better understanding of each service that is related to AWS EKS.

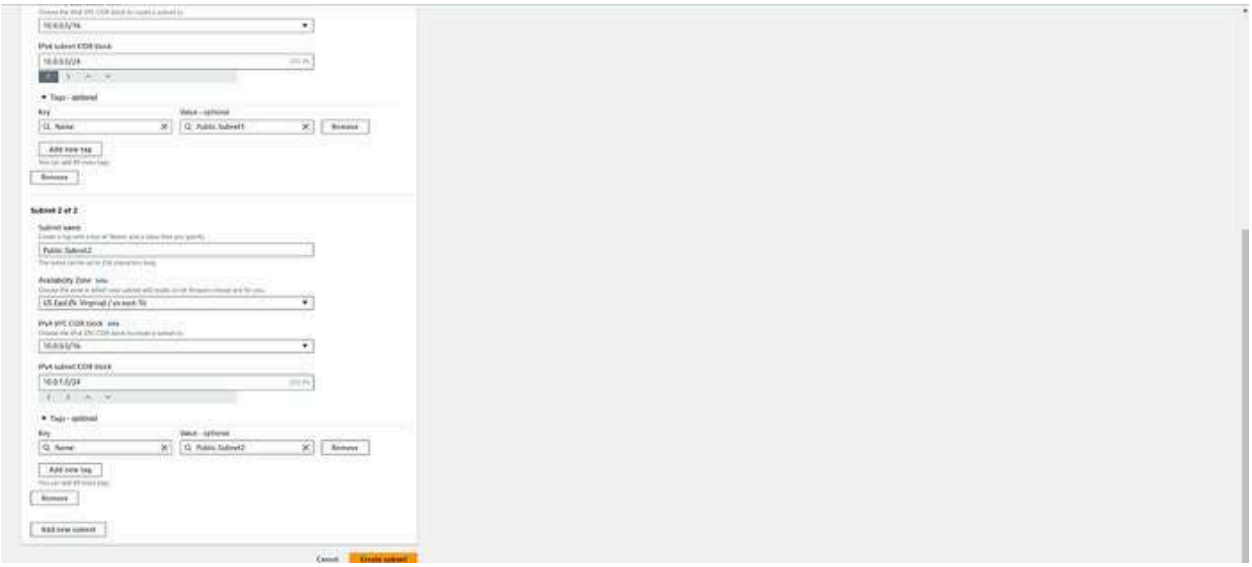
Create VPC and select the desired IPv4 CIDR.



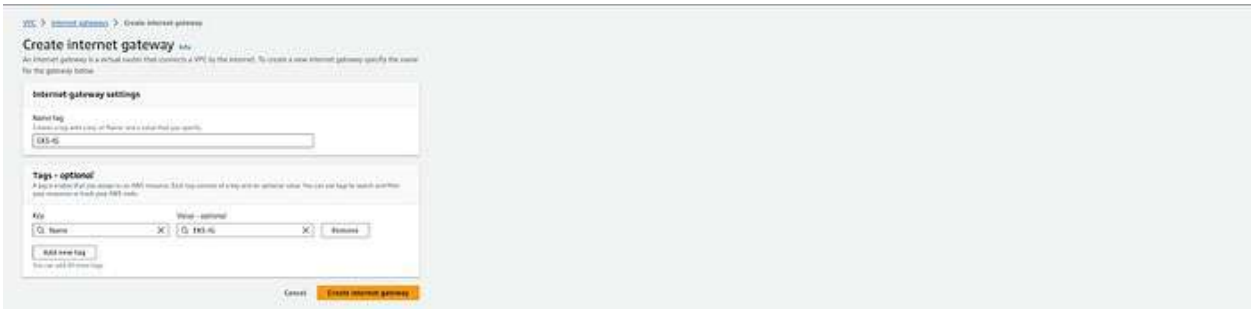
We need to create at least two Public Subnets to ensure high availability.

Public-Subnet1

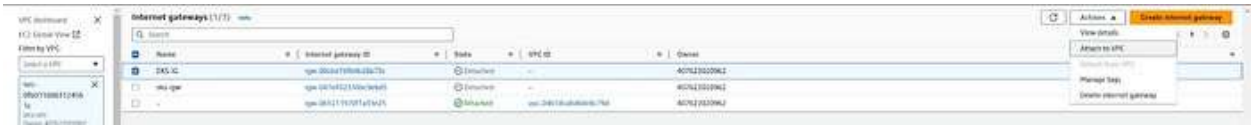
Public Subnet2



We need an internet connection for our clusters and worker nodes. To do that, create an Internet Gateway.

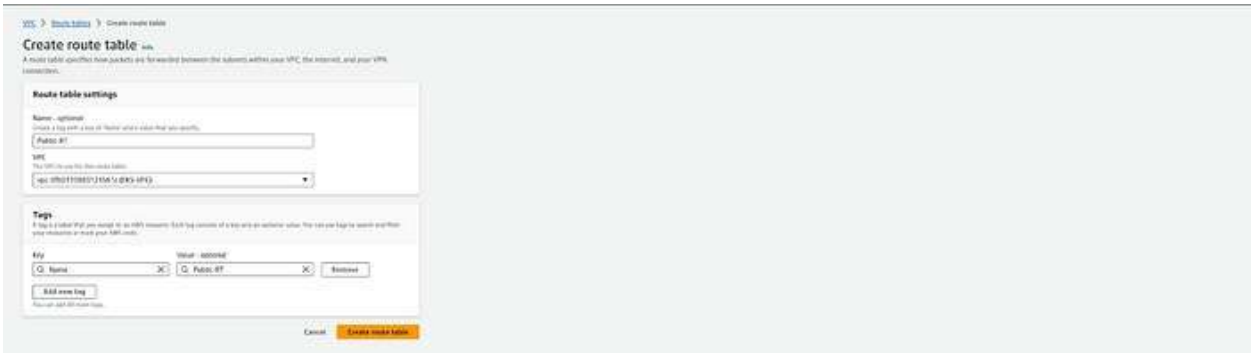


Now, attach the above Internet Gateway to the VPC that we created in the earlier step.



We need to create a route table as well for the internet access for each subnet.

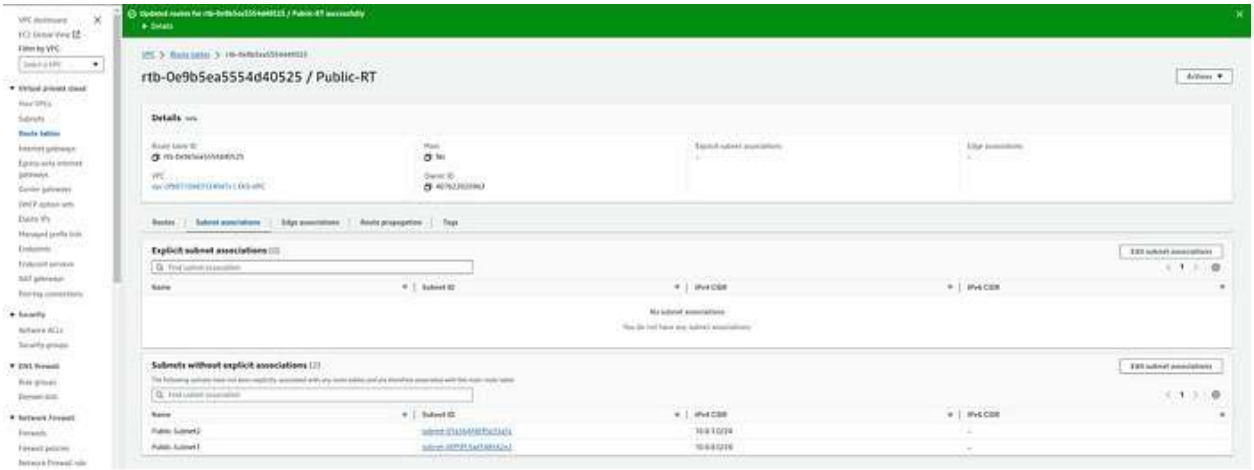
Public Route table



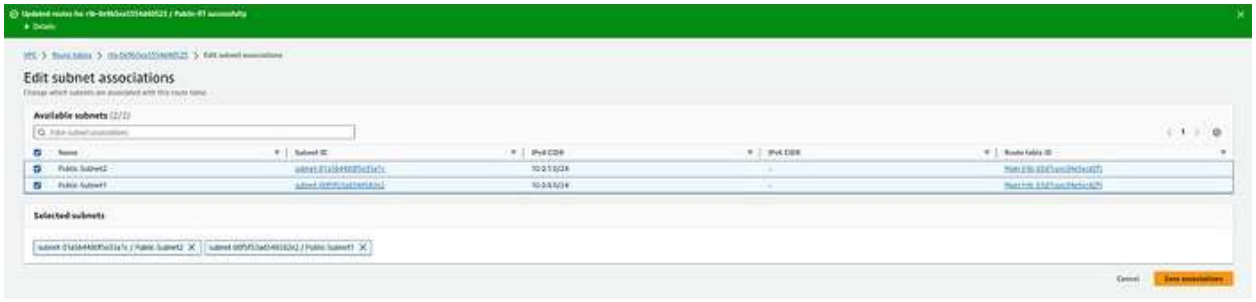
Select the Internet Gateway in the Target.

Once you add routes, then you have to add subnets for which purpose we are creating a Public Route table.

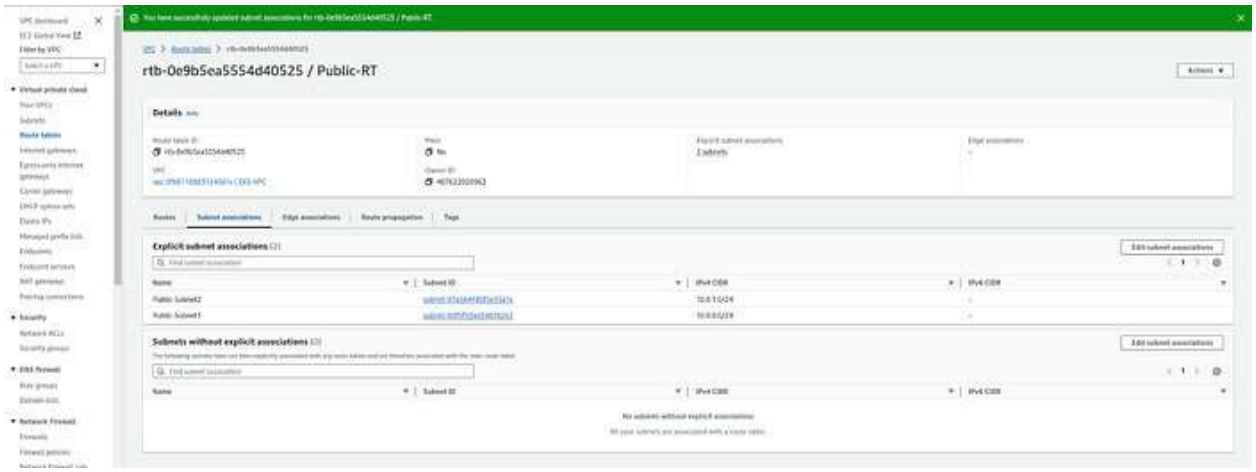
Click on Edit subnet associations.



Select both subnets and click on Save associations.



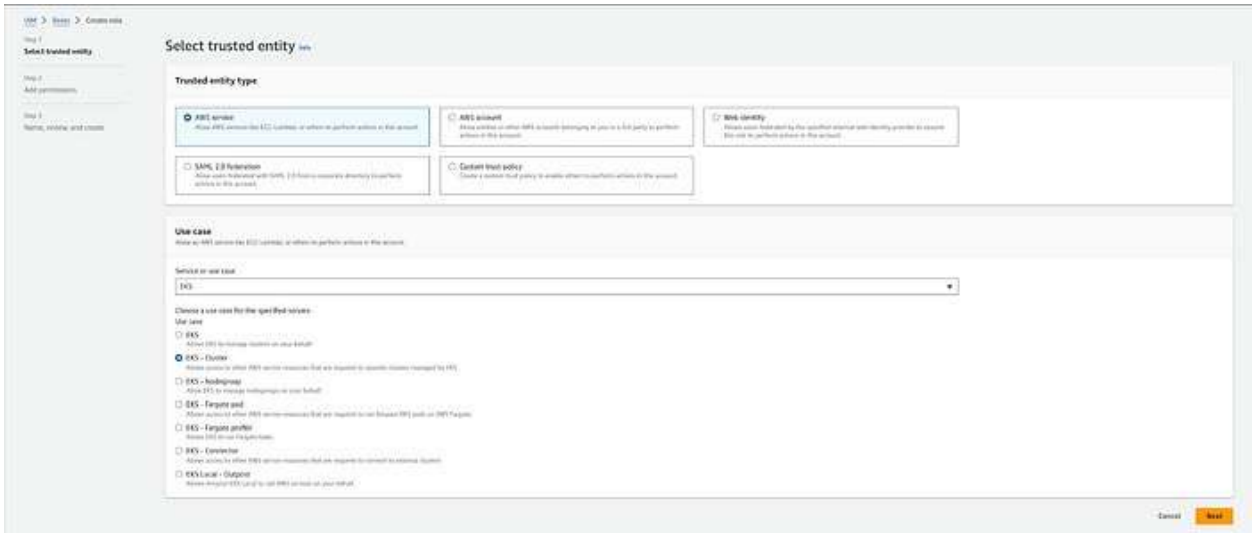
Once you associate the subnets. You will see your subnets look like the below snippets.



Now, the EKS Cluster needs some access to the AWS Services like ec2, kms, and load balancer.

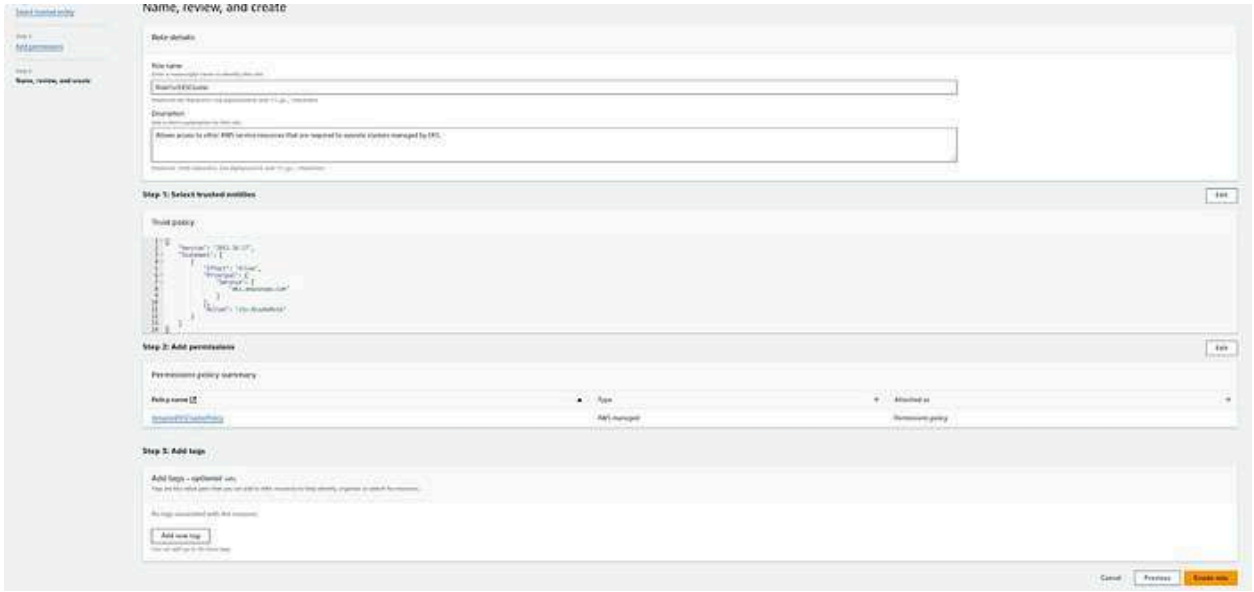
To do that, we will create an IAM Role and Policy for the EKS Cluster

Click on AWS service as a Trusted entity type and select the EKS as Usecase and in the below options, choose EKS-Cluster.



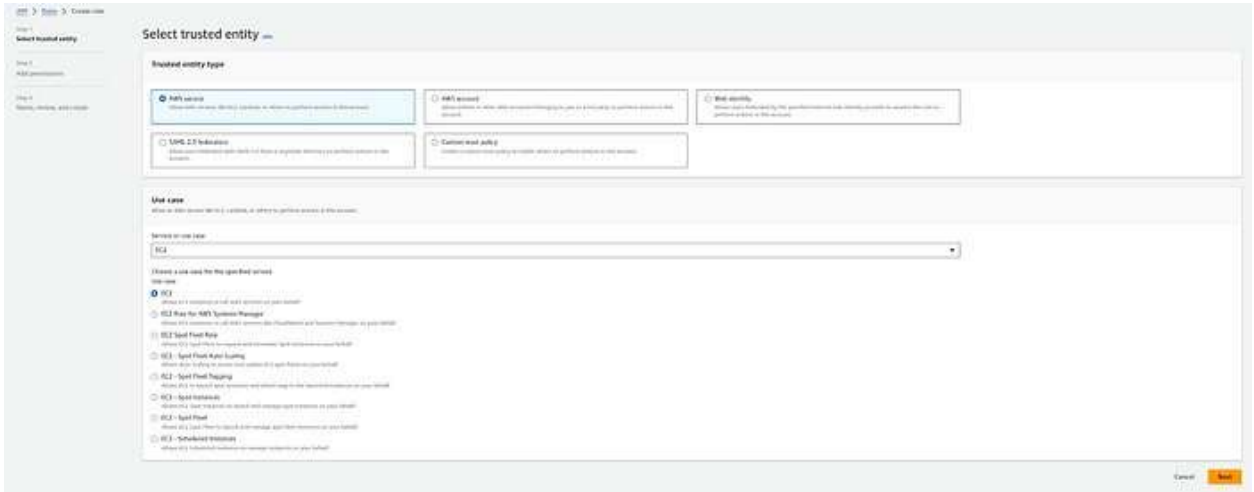
Click on Next.

Provide the Role name



Once we created the roles for the EKS Cluster. Now, we have to create a role for the Worker Nodes which is also a necessary part.

Click on AWS service as a Trusted entity type and select the EC2 as Usecase and in the below options, choose EC2.

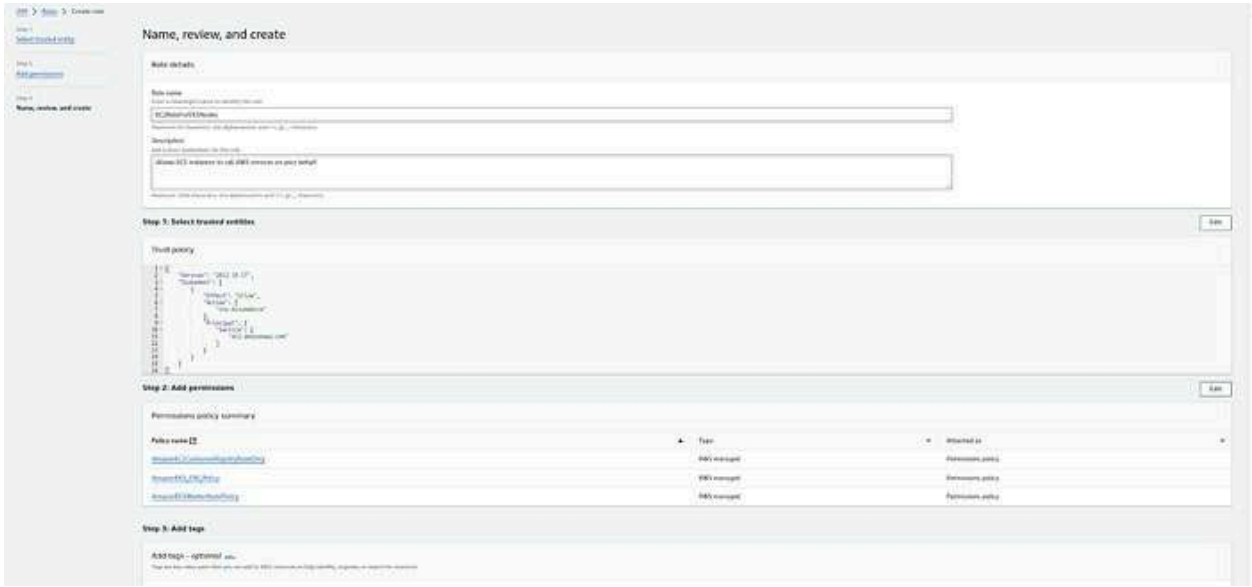


When you will get a popup to add the Policy for the Worker Nodes.

Select the below three policies for our Worker Nodes.

AmazonEC2ContainerRegistryReadOnly , AmazonEKS_CNI_Policy , and AmazonEKSWorkerNodePolicy

Provide the name of the Role and click on next.

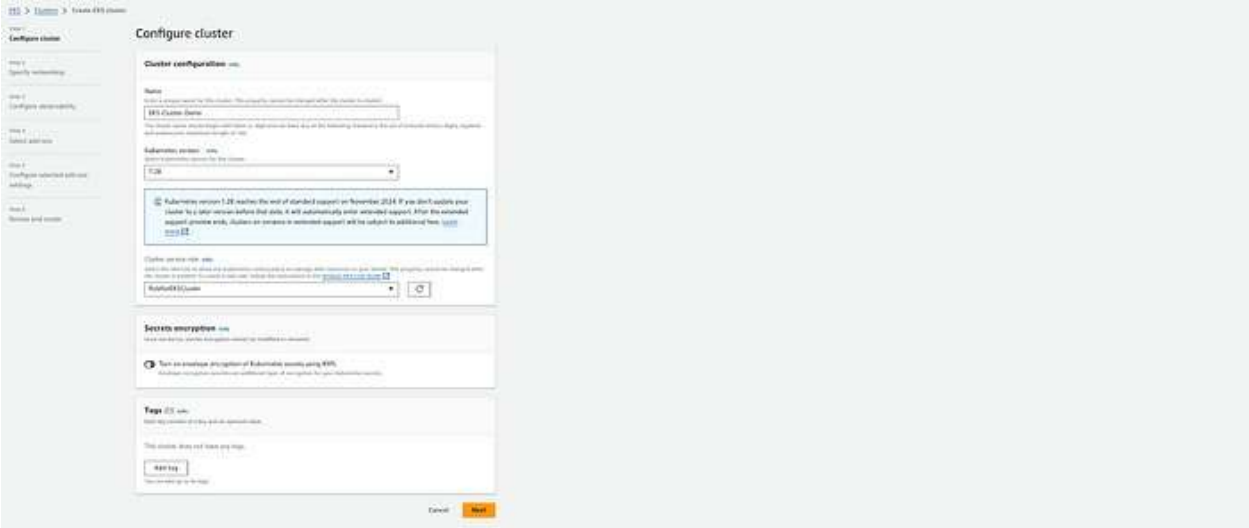


Now, Prerequisites are completed. Let’s create the EKS

Navigate to the AWS EKS and click on Add cluster.

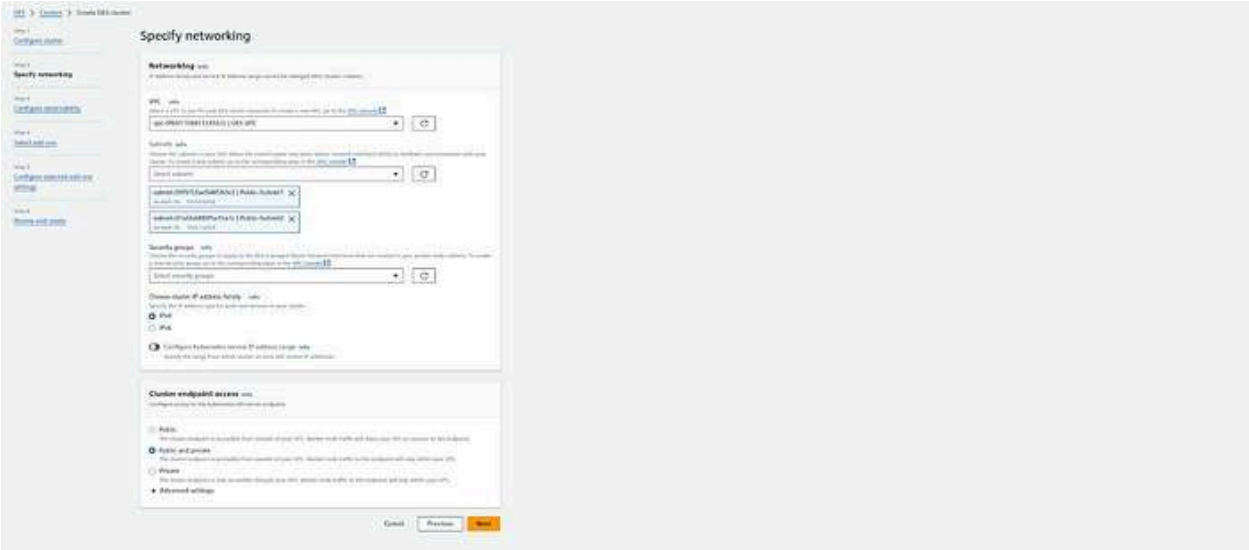
Select Create.

Provide the name of your EKS Cluster then select the Cluster role that we have created for EKS Cluster and rest of the things will be as they as and click on Next.

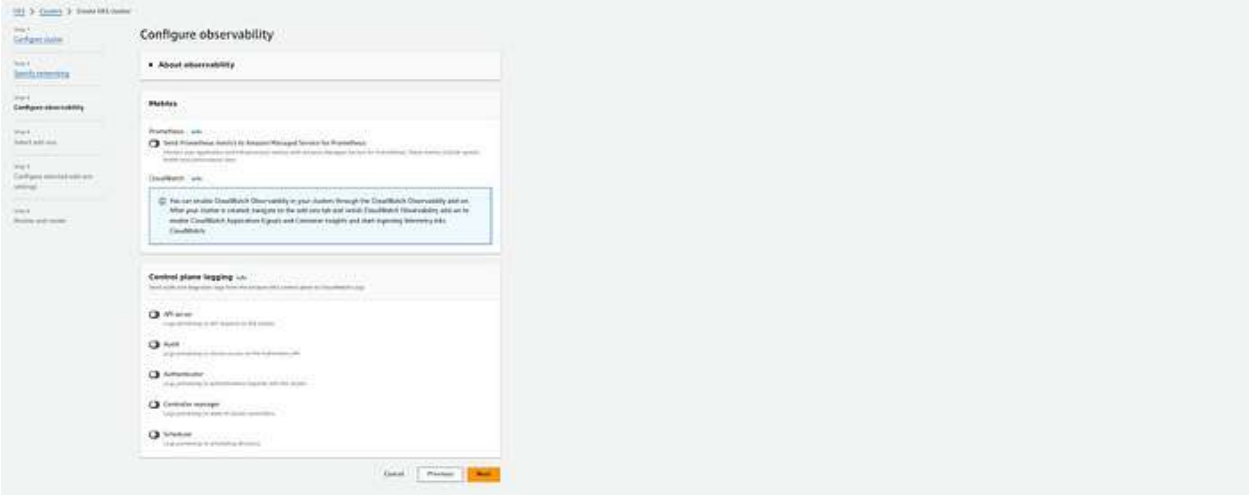


In the network configuration,

Select the vpc that we created earlier with both subnets. Apart from that, others will be as it is, and click on Next.

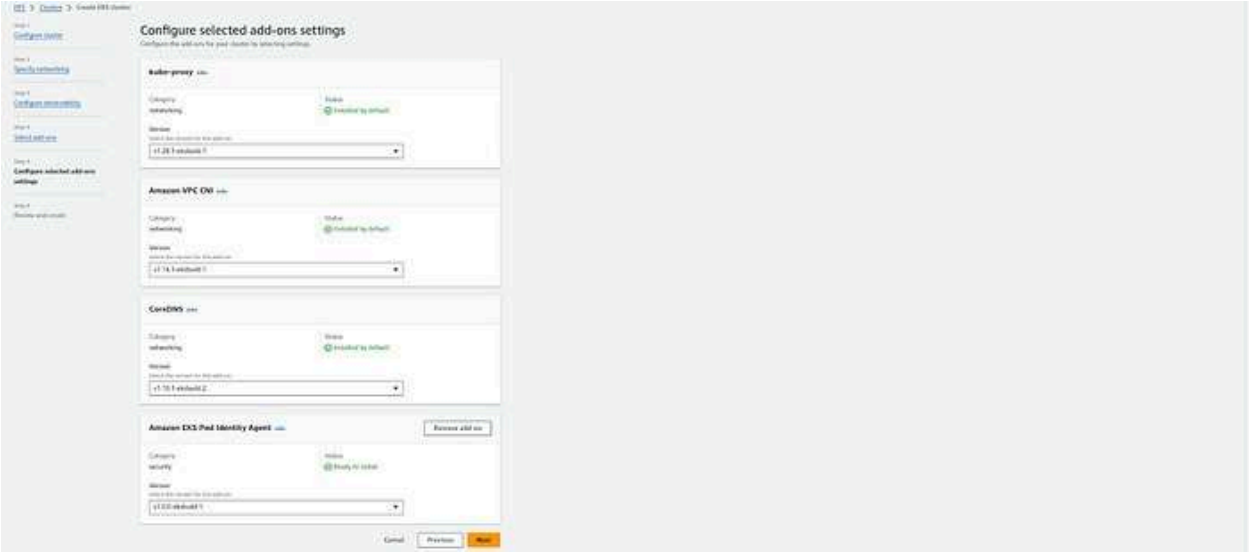


Keep the default things as it is and click on Next.

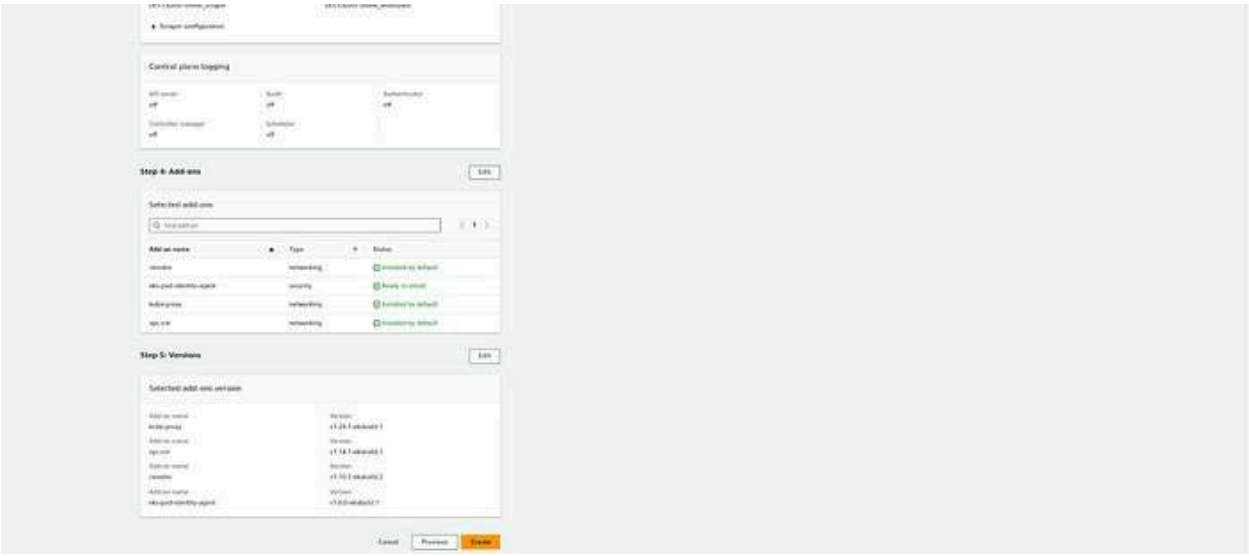


Keep the default things as it is and click on Next.

Keep the default things as it is and click on Next.



Keep the default things as it is and click on Create.



After clicking on Create, AWS will take around 4 to 5 minutes to create the EKS. Meanwhile, let's install Kubectl to work on AWS EKS.

```
curl -O
https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.3/2023-11-14/bin/linux/amd64/kubectl
```

```
amanpathak@pop-os:~$ curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.3/2023-11-14/bin/linux/amd64/kubectl
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 47.5M  100 47.5M    0     0  5843k      0  0:00:09  0:00:09 --:--:-- 7830k
amanpathak@pop-os:~$
```

```
sha256sum -c kubectl.sha256
```

```
amanpathak@pop-os:~$ sha256sum -c kubectl.sha256
kubectl: OK
amanpathak@pop-os:~$
```

```
openssl sha1 -sha256 kubectl
```

```
amanpathak@pop-os:~$ openssl sha1 -sha256 kubectl
SHA2-256(kubectl)= 3b9ffe2effbfc12a30b12739126f069fe8a7f13625e71ccb82c33ad1ea8f8092
amanpathak@pop-os:~$
```

```
chmod +x ./kubectl
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export
PATH=$HOME/bin:$PATH
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
```

```
amanpathak@pop-os:~$ chmod +x ./kubectl
amanpathak@pop-os:~$ mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
amanpathak@pop-os:~$ echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
amanpathak@pop-os:~$
```

```
kubectl version --client
```

```
amanpathak@pop-os:~$ kubectl version --client
Client Version: v1.28.3-eks-e71965b
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
amanpathak@pop-os:~$
```

Install eksctl on the local machine (Optional)

```
ARCH=amd64
PLATFORM=$(uname -s)_$ARCH
curl -sLO
"https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$PLATFORM.tar.gz"
curl -sL "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_checksums.txt" |
grep $PLATFORM | sha256sum - check
tar -xzf eksctl_$PLATFORM.tar.gz -C /tmp && rm eksctl_$PLATFORM.tar.gz
sudo mv /tmp/eksctl /usr/local/bin
```

```
ubuntu@ip-172-31-82-215:~$ ARCH=amd64
PLATFORM=$(uname -s)_$ARCH
curl -sLO "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$PLATFORM.tar.gz"
curl -sL "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_checksums.txt" | grep $PLATFORM | sha256sum --check
tar -xzf eksctl_$PLATFORM.tar.gz -C /tmp && rm eksctl_$PLATFORM.tar.gz
sudo mv /tmp/eksctl /usr/local/bin
eksctl_Linux_amd64.tar.gz: OK
ubuntu@ip-172-31-82-215:~$
ubuntu@ip-172-31-82-215:~$ eksctl version
0.165.0
ubuntu@ip-172-31-82-215:~$
```

Now, check the status of EKS whether it is Active or Not.

Once the status of EKS is Active, run the below command.

```
aws eks update-kubeconfig — region us-east-1 — name EKS-Cluster-Demo
```

If you are getting the error which is showing below in the snippet. Then, don’t worry. Let’s solve it in the next step.

```
ubuntu@ip-172-31-82-215:~$ aws eks update-kubeconfig --region us-east-1 --name EKS-Cluster-Demo
Added new context arn:aws:eks:us-east-1:407622020962:cluster/EKS-Cluster-Demo to /home/ubuntu/.kube/config
ubuntu@ip-172-31-82-215:~$
ubuntu@ip-172-31-82-215:~$ kubectl get nodes
error: exec plugin: invalid apiVersion "client.authentication.k8s.io/v1alpha1"
ubuntu@ip-172-31-82-215:~$
```

Replace the ‘alpha’ with ‘beta’ like below snippet

```
16 - name: arn:aws:eks:us-east-1:407622020962:cluster/EKS-Cluster-Demo
17   user:
18     exec:
19       apiVersion: client.authentication.k8s.io/v1beta1
20       args:
21       - --region
```

Now, run the command again to update the config

```
aws eks update-kubeconfig — region us-east-1 — name EKS-Cluster-Demo
```

```
amanpathak@pop-os:~$ aws eks update-kubeconfig --region us-east-1 --name EKS-Cluster-Demo
Added new context arn:aws:eks:us-east-1:407622020962:cluster/EKS-Cluster-Demo to /home/amanpathak/.kube/config
amanpathak@pop-os:~$
```

It’s working

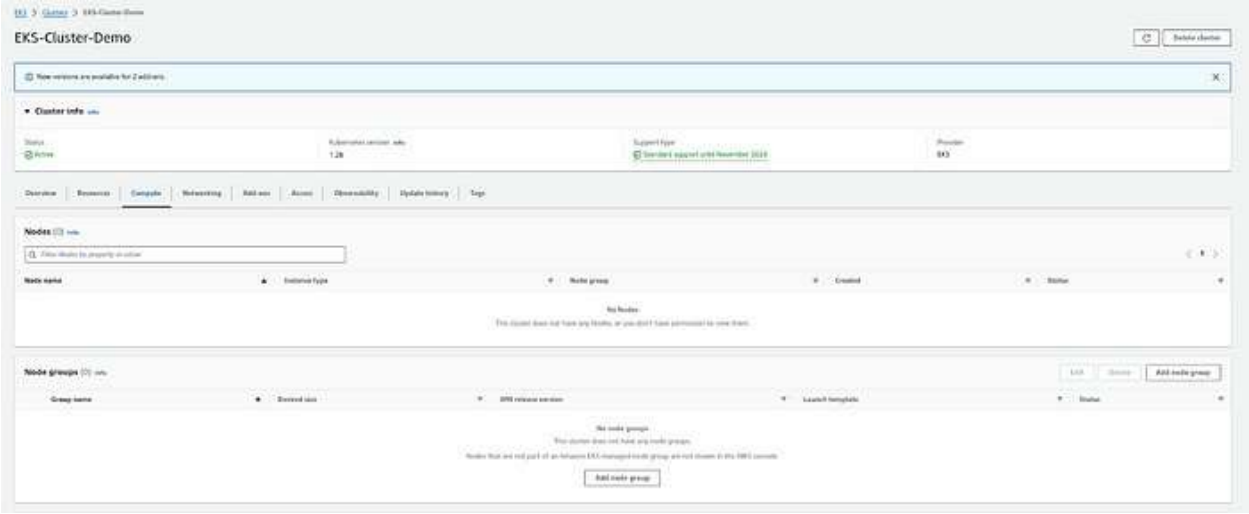
```
amanpathak@pop-os:~$ kubectl get nodes
No resources found
amanpathak@pop-os:~$
```

Trying to deploy the pod but it is in pending status because there is no worker node present where the pod can be created.

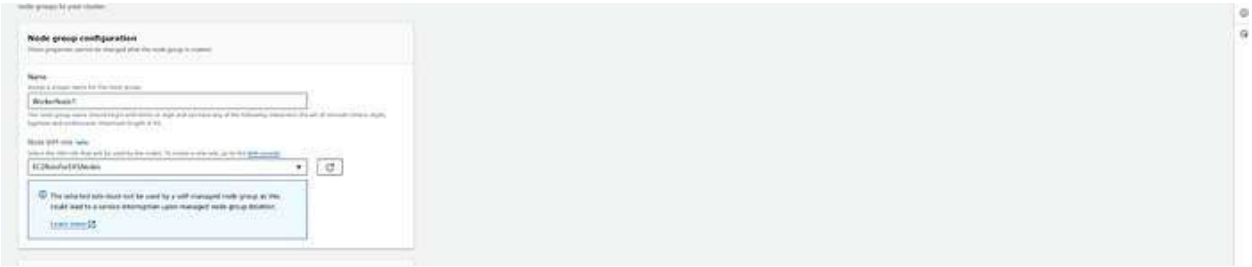
```
anaspathak@pop-os:~$ kubectl create deployment nginx --image=nginx
deployment.apps/nginx created
anaspathak@pop-os:~$
anaspathak@pop-os:~$ kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
nginx-7854ff8877-clhrd   0/1     Pending   0          10s
anaspathak@pop-os:~$ kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
nginx-7854ff8877-clhrd   0/1     Pending   0          52s
anaspathak@pop-os:~$
```

To create a worker node, Select the EKS Cluster and navigate to the Compute section.

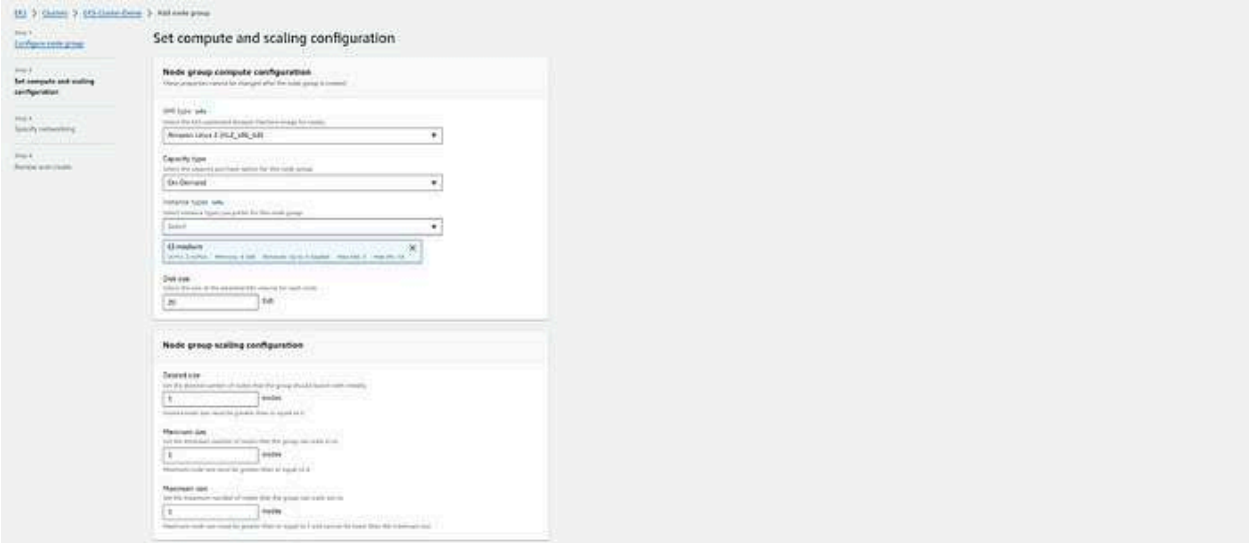
Click on Add node group.



Provide the name of your worker node and select the Worker Node role that we created earlier.

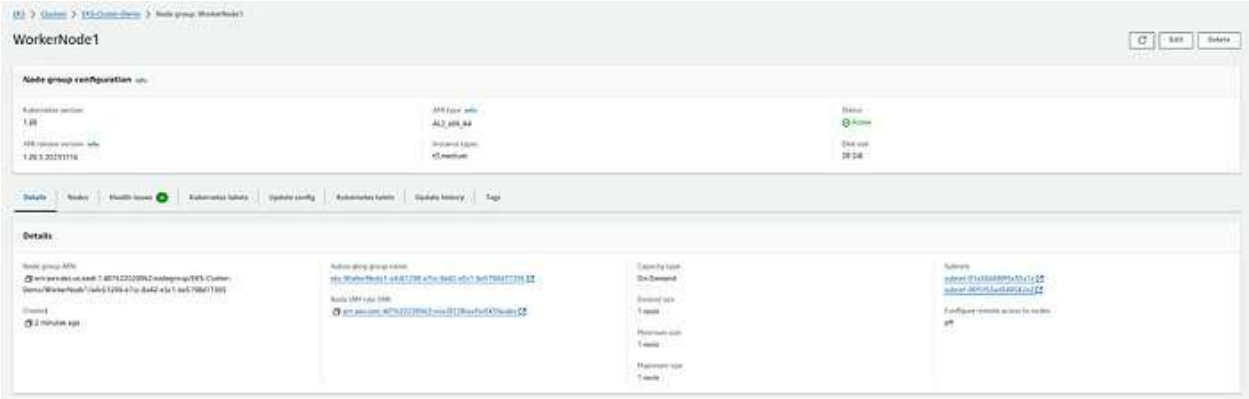


You can modify things according to your requirements. But the instance type t3.medium will be good because Kubernetes needs at least 2CPU.



Select the Subnets of the VPC that we have created above and click on Next.

Once the node is in Active status. Then, you can follow the next step.



Run the below command and you will see that our pending pod is now in running state.

kubectl get pods

```
amanpathak@pop-os:~$ kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
nginx-7854ff8877-clhrd   1/1     Running   0          8m53s
amanpathak@pop-os:~$
```

Now, delete the previous, and let's try to run the static application on the nginx server with the AWS load balancer

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-app-deployment
 labels:
   app: nginx-app
spec:
 replicas: 2
 selector:
   matchLabels:
     app: nginx-app
 template:
   metadata:
     labels:
       app: nginx-app
   spec:
     containers:
     -name: nginx-container
       image: avian19/nginx-ne:latest-1.0
       ports:
       -containerPort: 80
```

```
kubectl apply -f deployment.yml
```

```
amanpathak@pop-os:~/Github/Kubernetes-files/AWS-EKS$ kubectl apply -f deployment.yml
deployment.apps/nginx-app-deployment created
amanpathak@pop-os:~/Github/Kubernetes-files/AWS-EKS$
amanpathak@pop-os:~/Github/Kubernetes-files/AWS-EKS$ kubectl get pods
NAME                                   READY   STATUS    RESTARTS   AGE
nginx-app-deployment-d955457d5-cs4cq   1/1     Running   0          76s
nginx-app-deployment-d955457d5-jnfpd   1/1     Running   0          76s
amanpathak@pop-os:~/Github/Kubernetes-files/AWS-EKS$
```

Now, host the application outside of the Kubernetes Cluster by creating a service for the nginx application and observing the load balancer dns in the EXTERNAL-IP Column.

```
apiVersion: v1
kind: Service
metadata:
 name: nginx
spec:
 selector:
   app: nginx-app
 type: LoadBalancer
 ports:
```

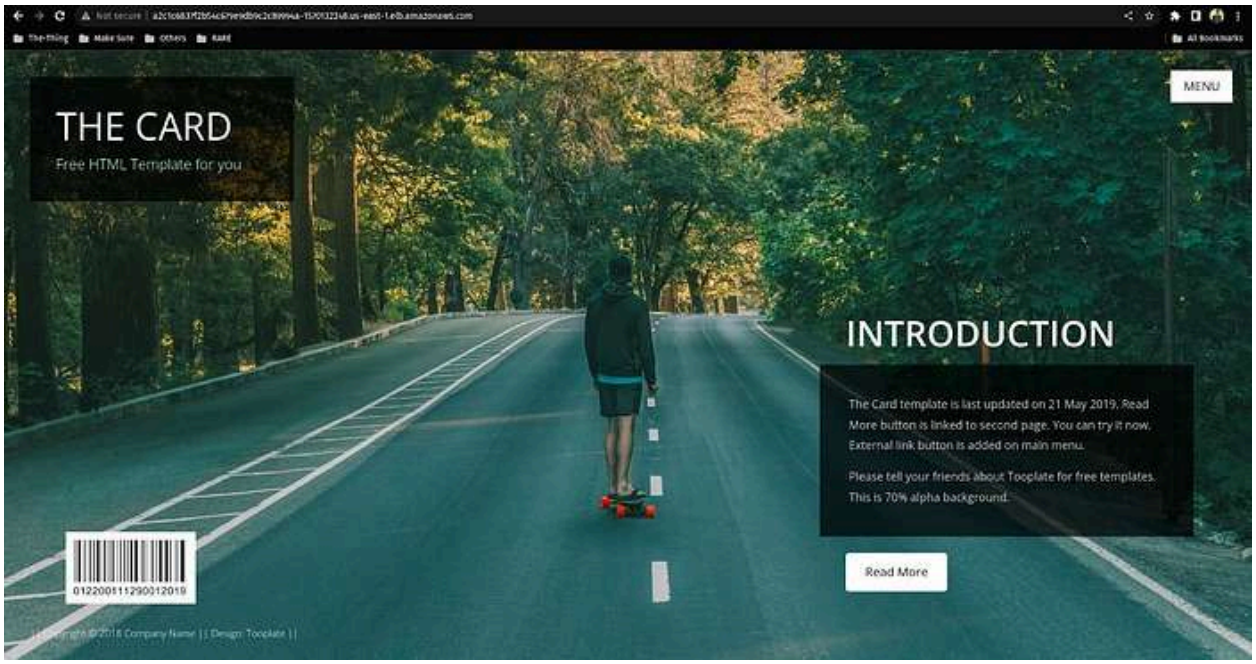
```
-protocol: TCP
  port: 80
```

```
kubectl apply -f svc.yaml
```

```
amanpathak@pop-os:~/Github/Kubernetes-files/AWS-EKS$ kubectl apply -f svc.yaml
service/nginx created
amanpathak@pop-os:~/Github/Kubernetes-files/AWS-EKS$
amanpathak@pop-os:~/Github/Kubernetes-files/AWS-EKS$ kubectl get svc
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP                                                                PORT(S)        AGE
kubernetes   ClusterIP      172.20.0.1      <none>                                                                     443/TCP        30m
nginx        LoadBalancer   172.20.22.247   a2c1c6837f2b54c679e9db9c2c89994a-1570132248.us-east-1.elb.amazonaws.com   80:31364/TCP   8s
amanpathak@pop-os:~/Github/Kubernetes-files/AWS-EKS$
```

Now, navigate to AWS Console and go to the Load Balancer section.

Copy the Load balancer DNS then, hit on the browser and see the magic.



Azure Kubernetes Service(AKS)

What is AKS?

AKS is a managed Kubernetes Service that helps us to deploy our applications without worrying about Control Plane and other things like regular updates support, Scaling, and high availability of your cluster.

Key Features:

- **Managed Control Plane** : AKS provides a fully managed control plane. So that, you don't need to configure things like the upgradation of Kubernetes cluster, patching, and monitoring. This will help us to focus on main things like deployment of the application.
- **Scalability** : AKS provides the scalability feature in which we can scale our worker nodes according to our application traffic which will help us to achieve high availability. AKS also enables the scale-up option for the AKS Clusters.
- **Windows container support:** You can run Linux containers on the AKS, But you can run Windows containers as well which will help developers to work on Windows base applications on the Kubernetes cluster.
- **Networking Configurations** : To create AKS, networking is required. So, you can configure the networking part according to your requirements which helps you to provide fine-grained control on the Cluster networking configurations.
- **Integration with other services** : You can integrate AKS with multiple Azure services like Azure AD, Azure Policy, Azure Monitor, etc to provide solutions within the cloud for container management.
- **Multi-worker nodes** : AKS can create multiple node pools according to the application's requirements.

If you want to read more features in a detailed way refer to the below link:

https://spot.io/resources/azure-kubernetes-service/

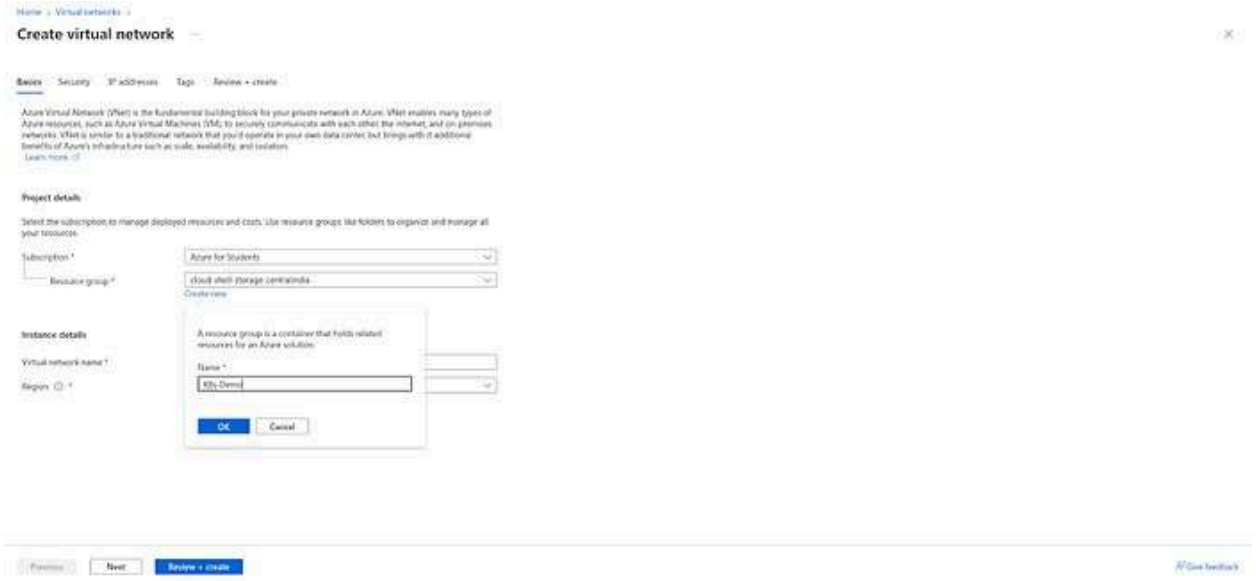
Azure AKS Costing

- If you are new to AKS and want to explore the AKS service then, AKS comes under afree tier account. But the resources like computing, networking, and storage will have a cost. So use wisely. Remember, you can't autoscale the clusters or nodes under the free tier.
- Now, If you want to use AKS in your Organization It will cost you 0.10$ per cluster per hour similar to AWS EKS Cost but the AKS cluster will be in the Standard tier. You can auto-scale your clusters and worker nodes in the Standard tier.
- There is one more tier which Premium and that will cost you 0.60$ per cluster per hour with advanced features. You can auto-scale your clusters and worker nodes in the Standard tier.

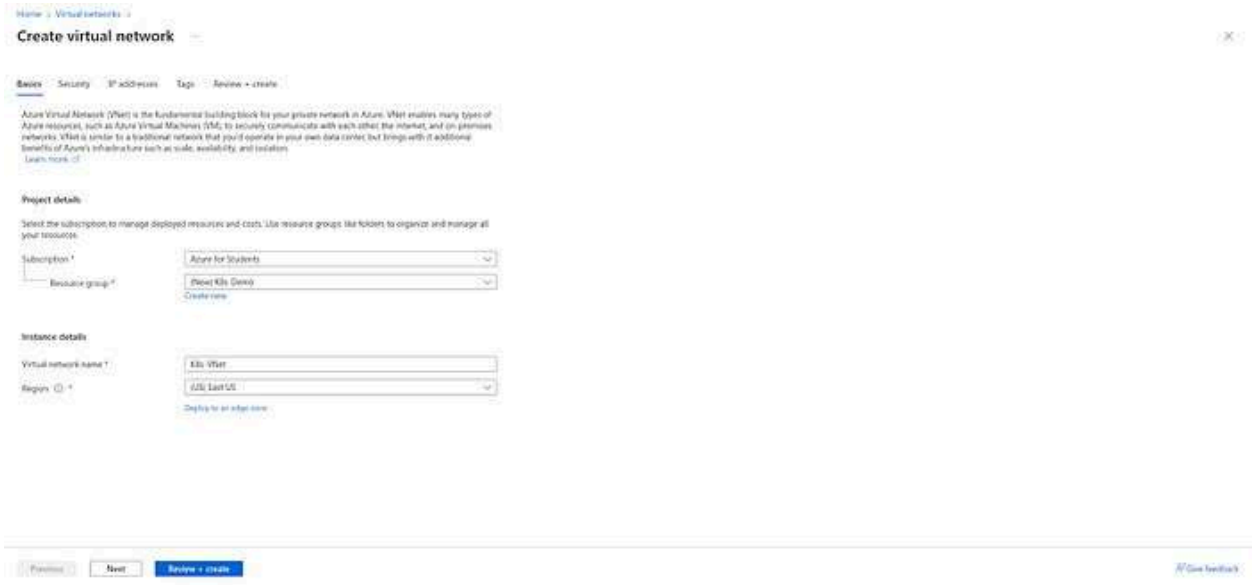
Let's Dive into the Demo! 🛠️

To create AKS, we need to configure VNet and other networking things. If you are not a beginner in the cloud feel free to skip the network configuration part. But if you are new to AKS or in the Azure Cloud, I would say to follow each step. So, it will help you to get a better understanding of each service that is related to Azure AKS.

First of all, create a separate Resource Group by clicking on Create new.

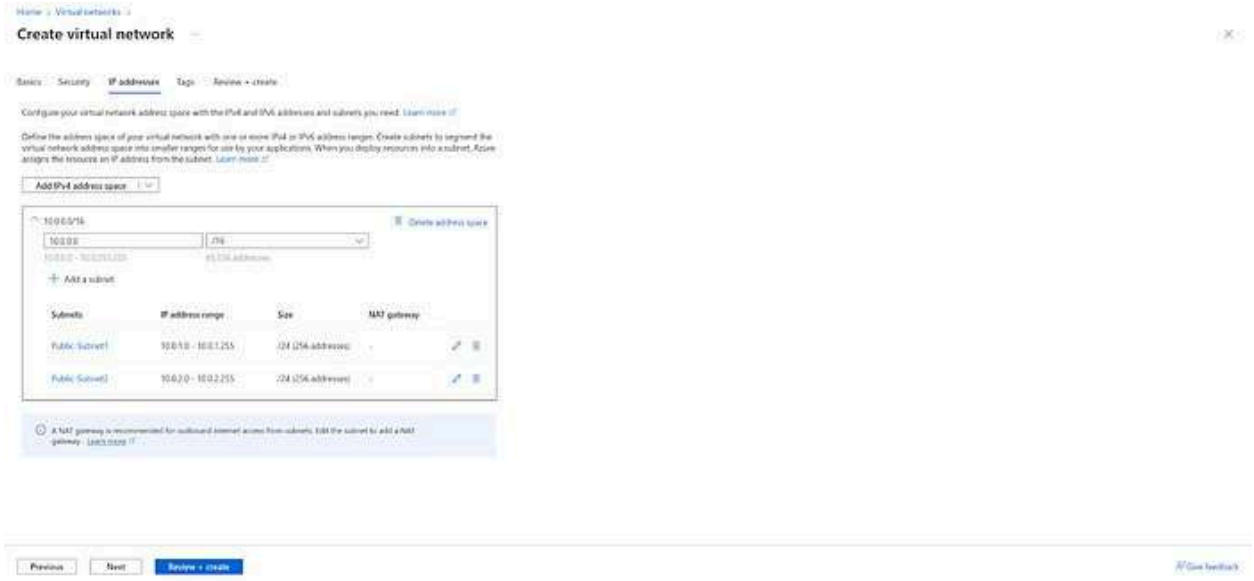


Provide the name of your Virtual Network and click on Next.

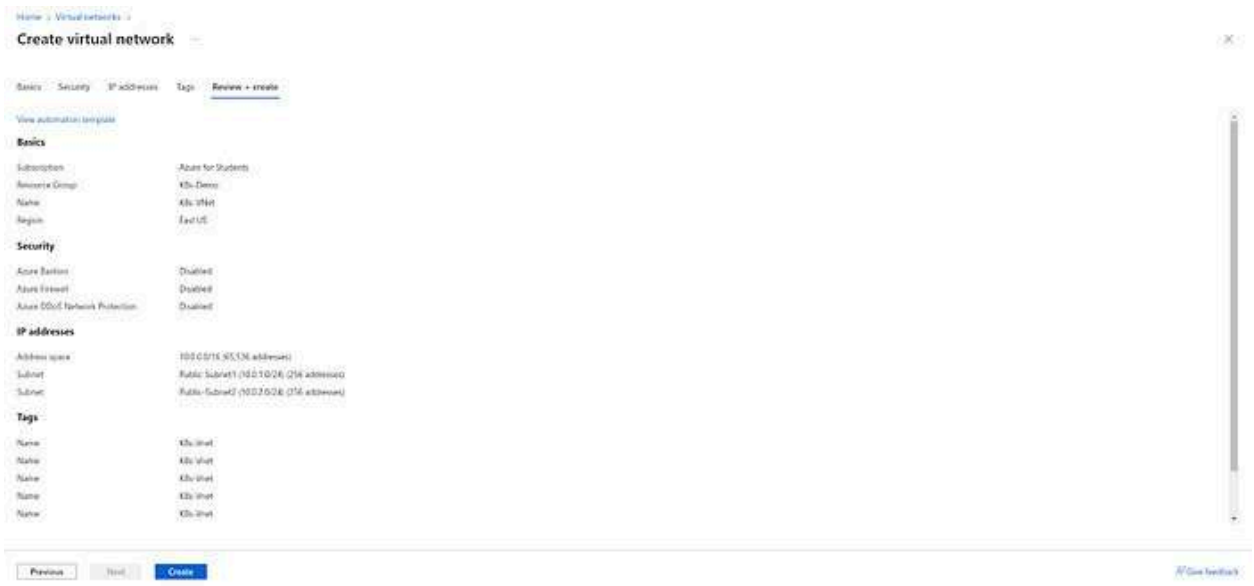


Now, we have to create two public subnets for the high availability of our Azure Kubernetes.

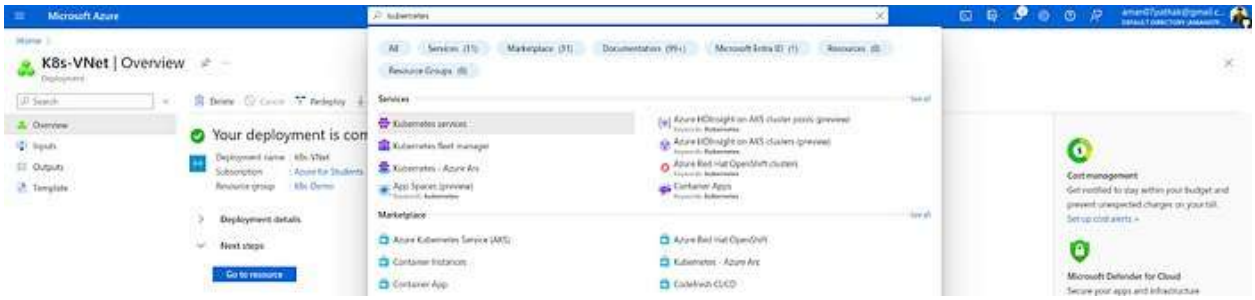
Delete the default subnet and click on Add a subnet then add two subnets with your desired IP address range and click on Next.



Click on Review + create to validate the error in the configurations. If there is no error feel free to click on Create.

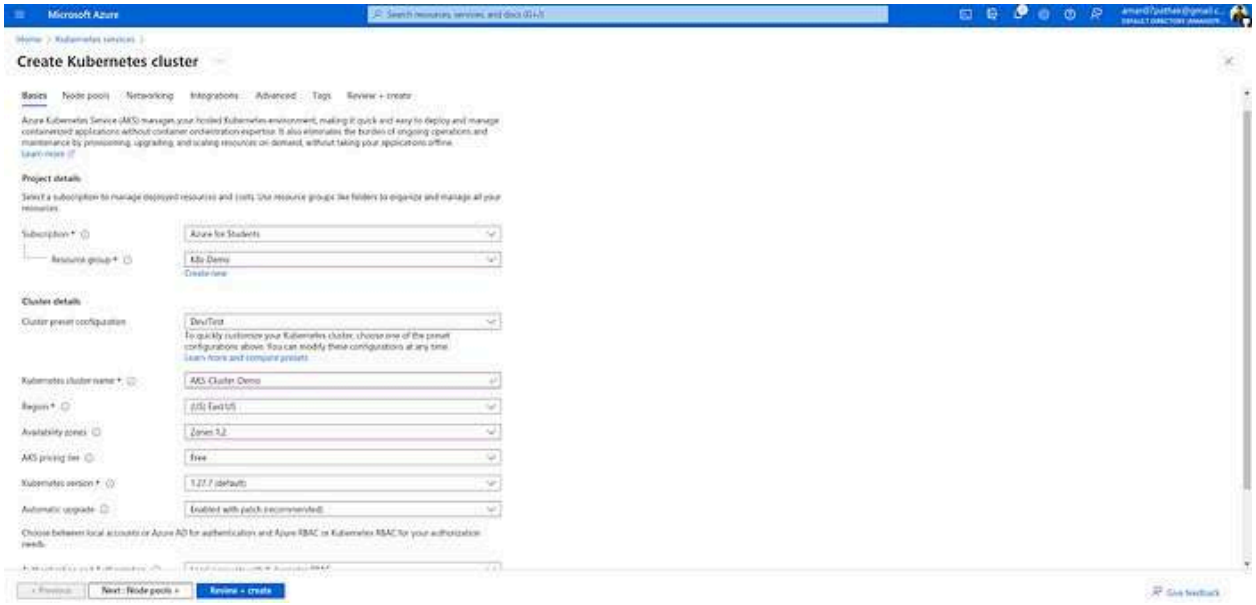


Once your deployment is done, in the search field enter Kubernetes services and click on the first one.



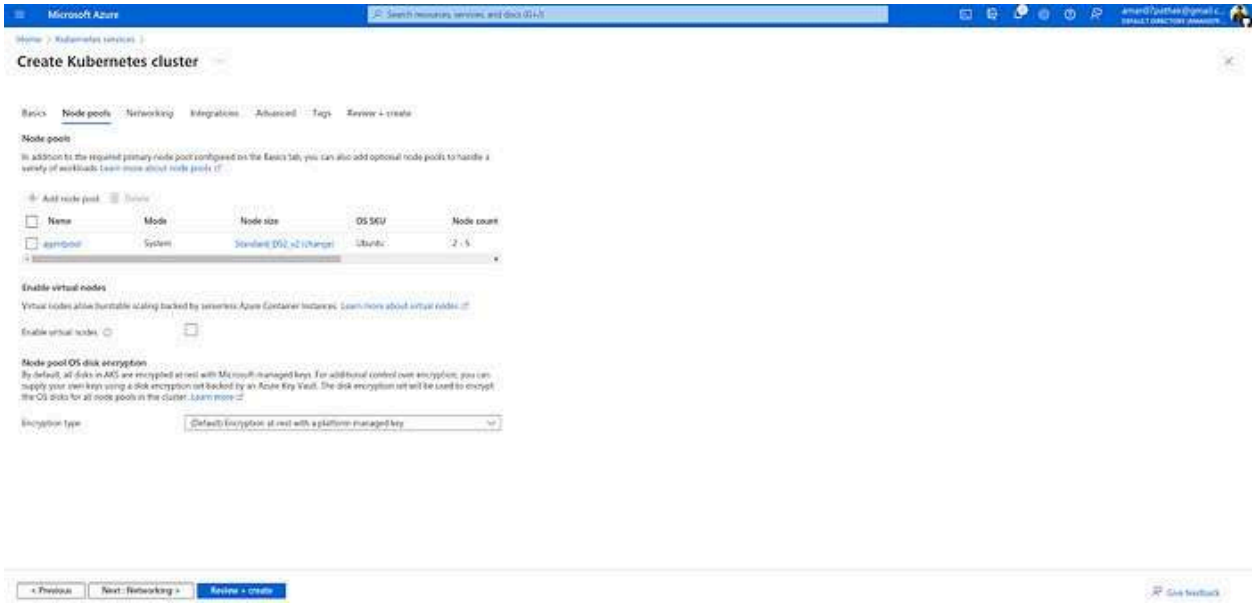
Click on Create and select Create a Kubernetes cluster.

Select the Same Resource Group that we have created while creating the Virtual Network. After that, provide the name to your AKS keep the things same as shown in the below snippet, and click on Node Pools.

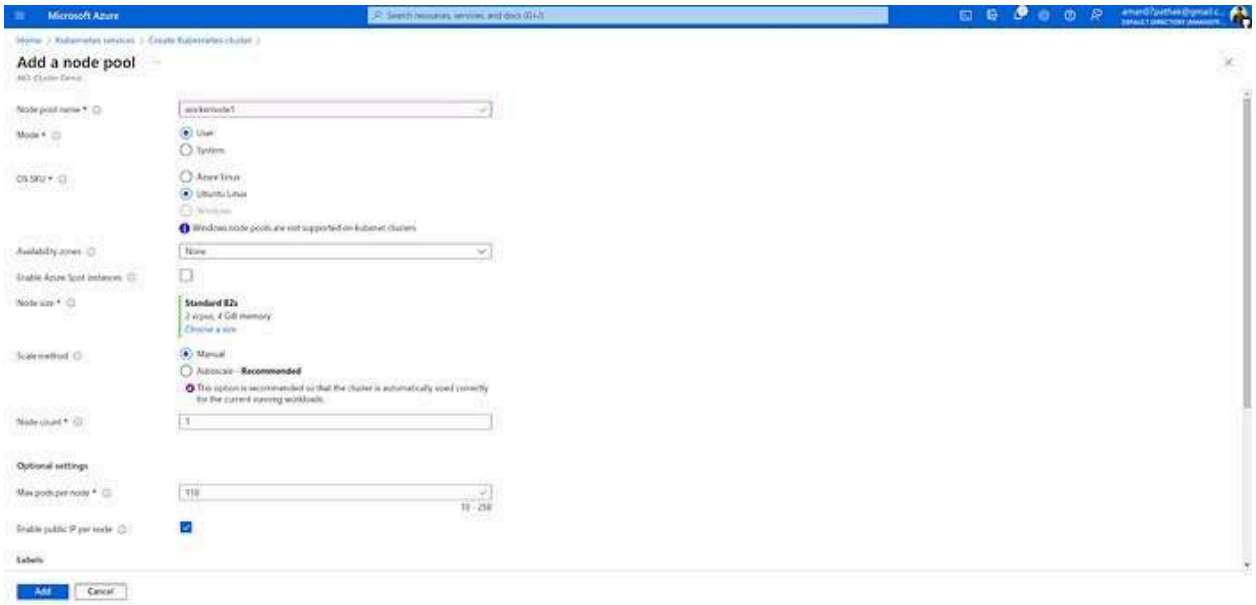


In the Node pools section, we have to add Worker Node. Remember one thing the default node(agentpool) is a system node. So you don't have to do anything with that node.

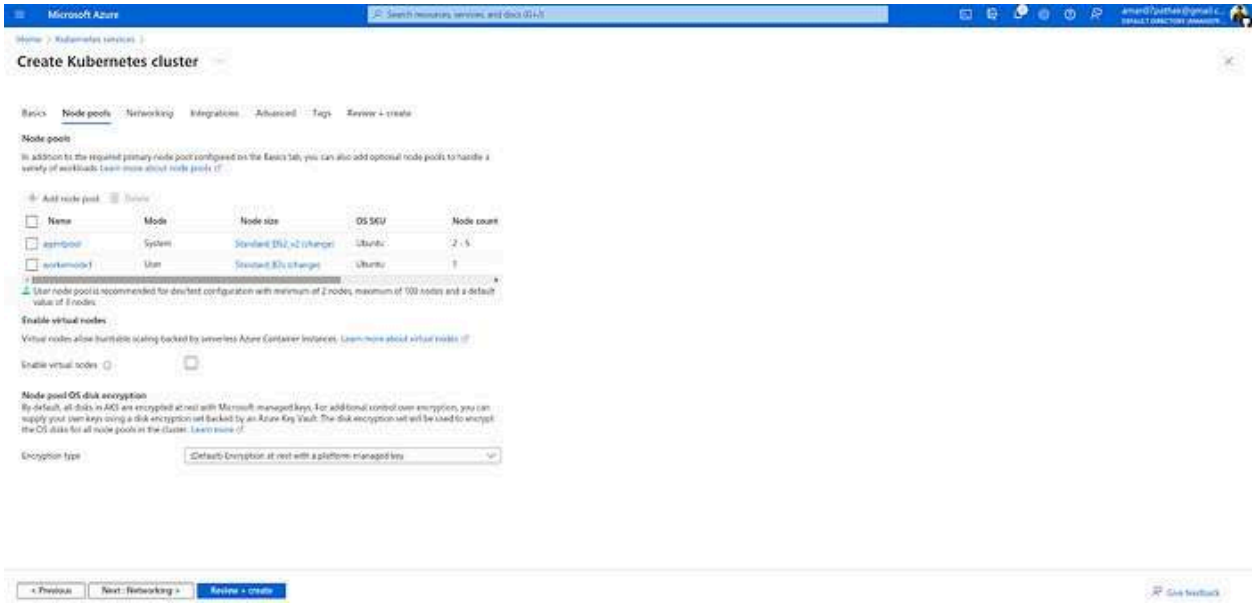
Click on Add node pool



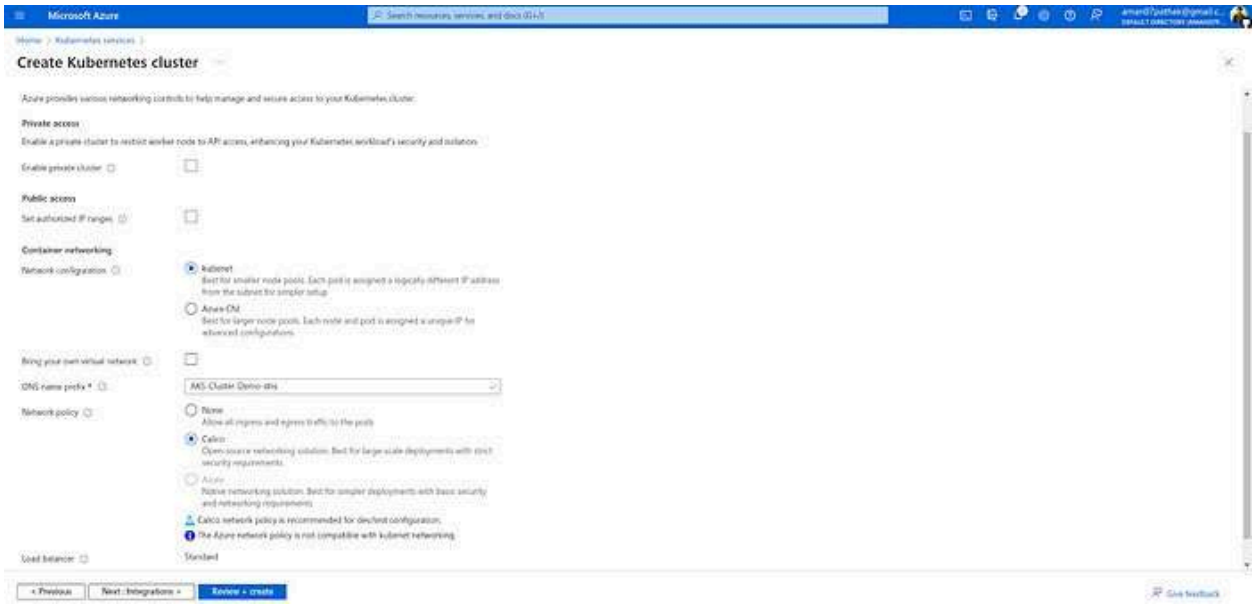
Provide the name to your worker node and keep the things same as shown in the below snippet such as Node size and Mode, etc.



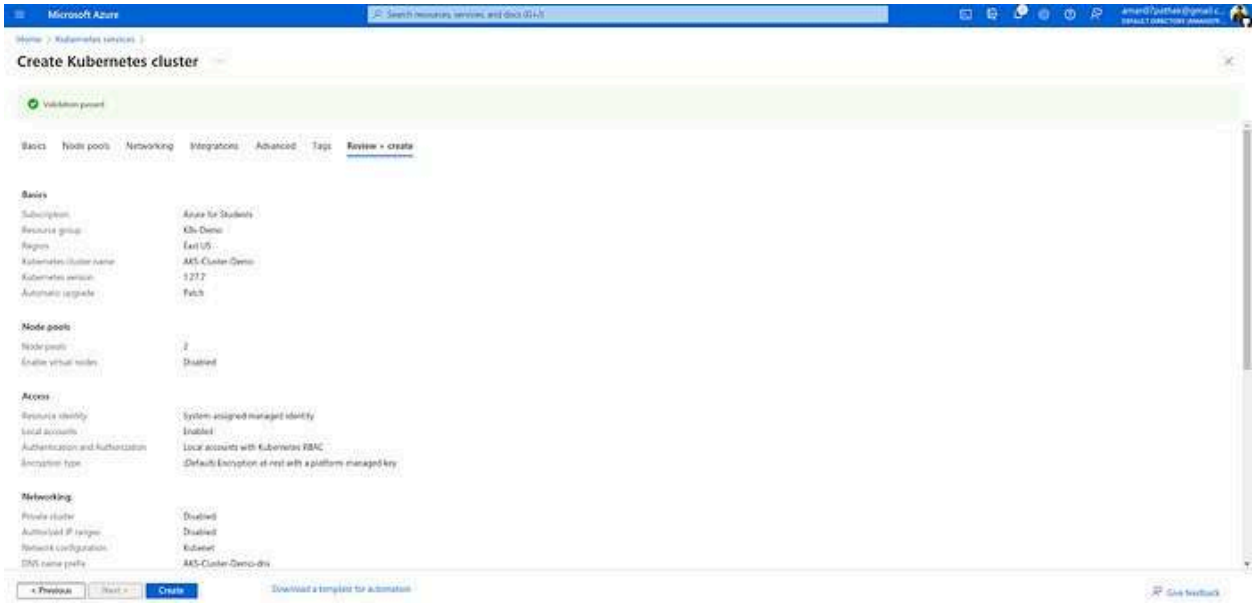
Now, click on Networking section.



Select the kubenet as Network configuration then, keep Calico as Network policy and click on Review + Create by skipping integrations and Advanced sections.

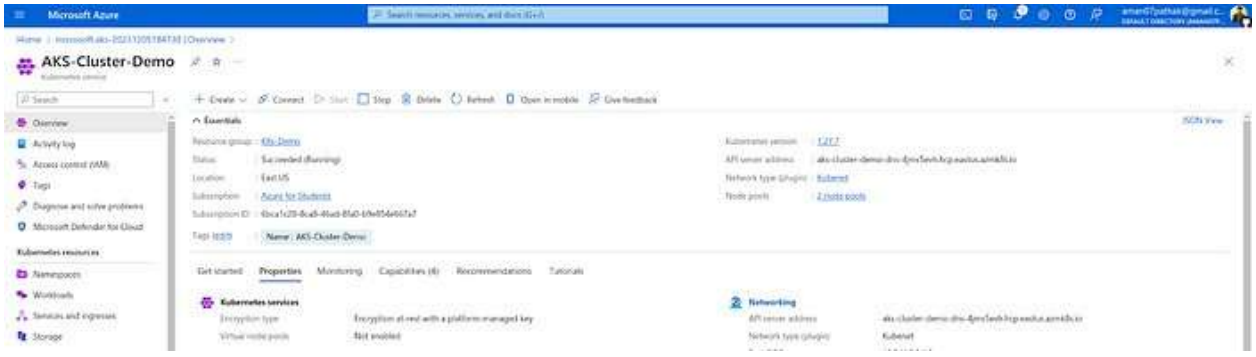


Once the validation is passed, click on Create.



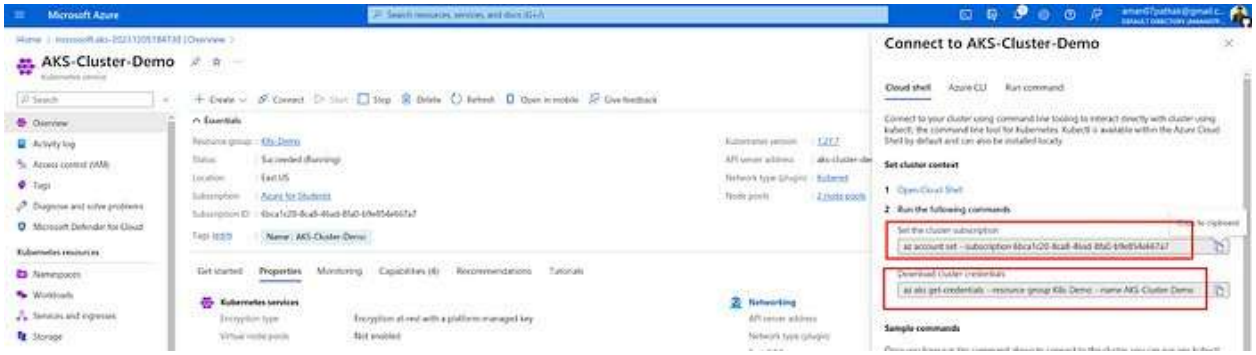
Once your deployment is complete, click on Go to resource.

Click on Connect.



You will have to run two commands that are showing on the right of the snippet to configure it on local or Cloud Shell.

But, we will configure it on our local. For that, we need to install Azure CLI and kubectl on our local machine.



To install kubectl on your local machine follow the below commands.

curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.3/2023-11-14/bin/linux/amd64/kubectl

```
amanpathak@pop-os:~$ curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.3/2023-11-14/bin/linux/amd64/kubectl
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 47.5M  100 47.5M    0     0  5843k      0  0:00:09  0:00:09 --:--:-- 7030k
amanpathak@pop-os:~$
```

curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.3/2023-11-14/bin/linux/amd64/kubectl.sha256

```
amanpathak@pop-os:~$ curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.3/2023-11-14/bin/linux/amd64/kubectl.sha256
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    73  100    73    0     0     57      0  0:00:01  0:00:01 --:--:--    57
amanpathak@pop-os:~$
```

sha256sum -c kubectl.sha256

```
amanpathak@pop-os:~$ sha256sum -c kubectl.sha256
kubectl: OK
amanpathak@pop-os:~$
```

openssl sha1 -sha256 kubectl

```
amanpathak@pop-os:~$ openssl sha1 -sha256 kubectl
SHA2-256(kubectl)= 3b9ffe2effbfc12a30b12739126f069fe8a7f13625e71ccb82c33ad1ea8f8092
amanpathak@pop-os:~$
```

chmod +x ./kubectl
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc

```
amanpathak@pop-os:~$ chmod +x ./kubectl
amanpathak@pop-os:~$ mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
amanpathak@pop-os:~$ echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
amanpathak@pop-os:~$
```

kubectl version --client

```
amanpathak@pop-os:~$ kubectl version --client
Client Version: v1.28.3-eks-e71965b
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
amanpathak@pop-os:~$
```

Once the kubectl is installed, you can install azurecli on your local by following the below commands.

Install Azure CLI on the local machine

```
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
```

To install AZ CLI on other OS, you can refer to the below link

https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-linux?pivots=apt

Now, login to your Azure Account by below command

```
az login
```

Once you run the below command, a new tab will open in your browser which will validate your account. If you have already logged into that account then, it will be automatically logged in on the terminal as well and you will see output like below snippet.

Once you log in, you can run the command that was given by Azure to connect with AKS

Now, you can run the command to list the nodes which will help you to validate the connection as well.

```
kubectl get nodes
```

Now, let's try to deploy the Apache application on AKS

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
 name: apache-app-deployment
 labels:
   app: apache-app
spec:
 replicas: 2
 selector:
   matchLabels:
     app: apache-app
 template:
   metadata:
     labels:
       app: apache-app
   spec:
     containers:
     -name: apache-container
       image: avian19/apache-ne:latest
       ports:
       -containerPort: 80
```

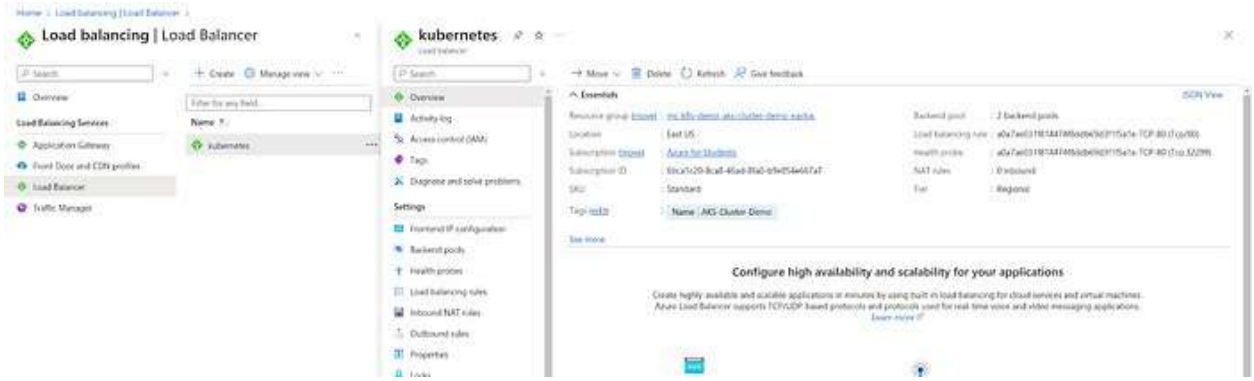
```
kubectl apply -f deployment.yml
```

Now, host the application outside of the Kubernetes Cluster by creating a service for the Apache application and observing the public in the EXTERNAL-IP Column.

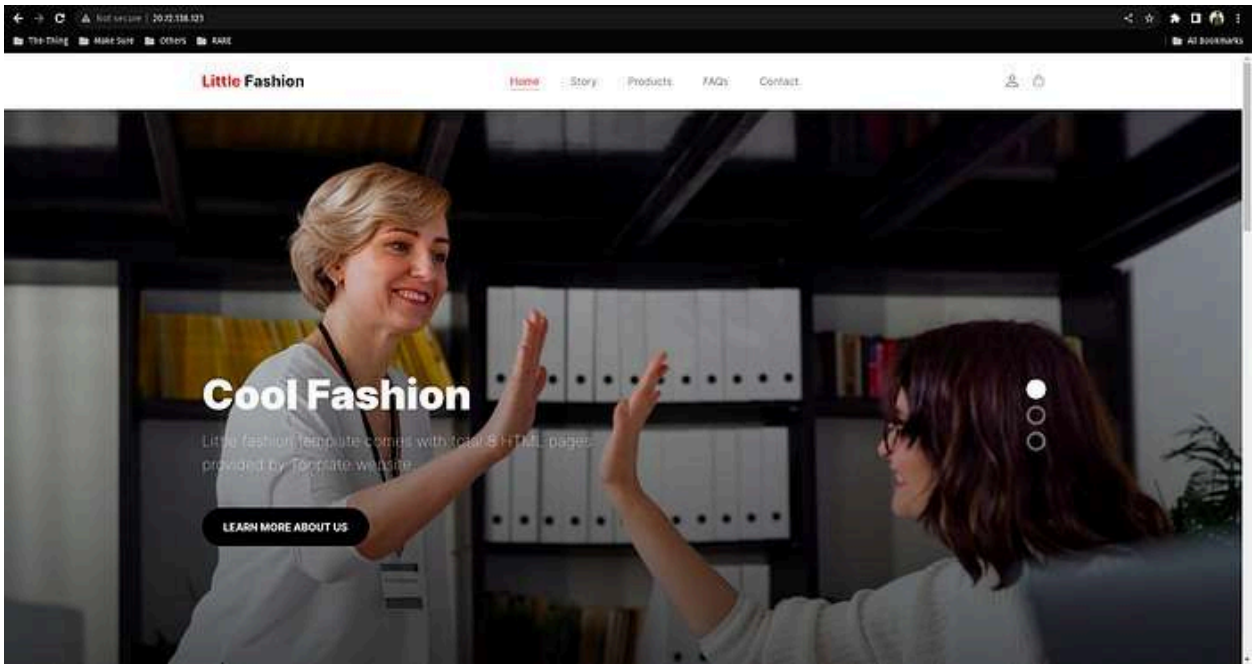
```
apiVersion: v1
kind: Service
metadata:
 name: apache
spec:
 selector:
 app: apache-app
 type: LoadBalancer
 ports:
 -protocol: TCP
   port: 80
```

```
kubectl apply -f svc.yaml
```

As our service type is LoadBalancer, AKS created one Load Balancer which will look like the below snippet.



Now, copy the EXTERNAL-IP that we get by running the list svc command in the previous step hit on the browser, and see the magic.



Google Kubernetes Engine (GKE)

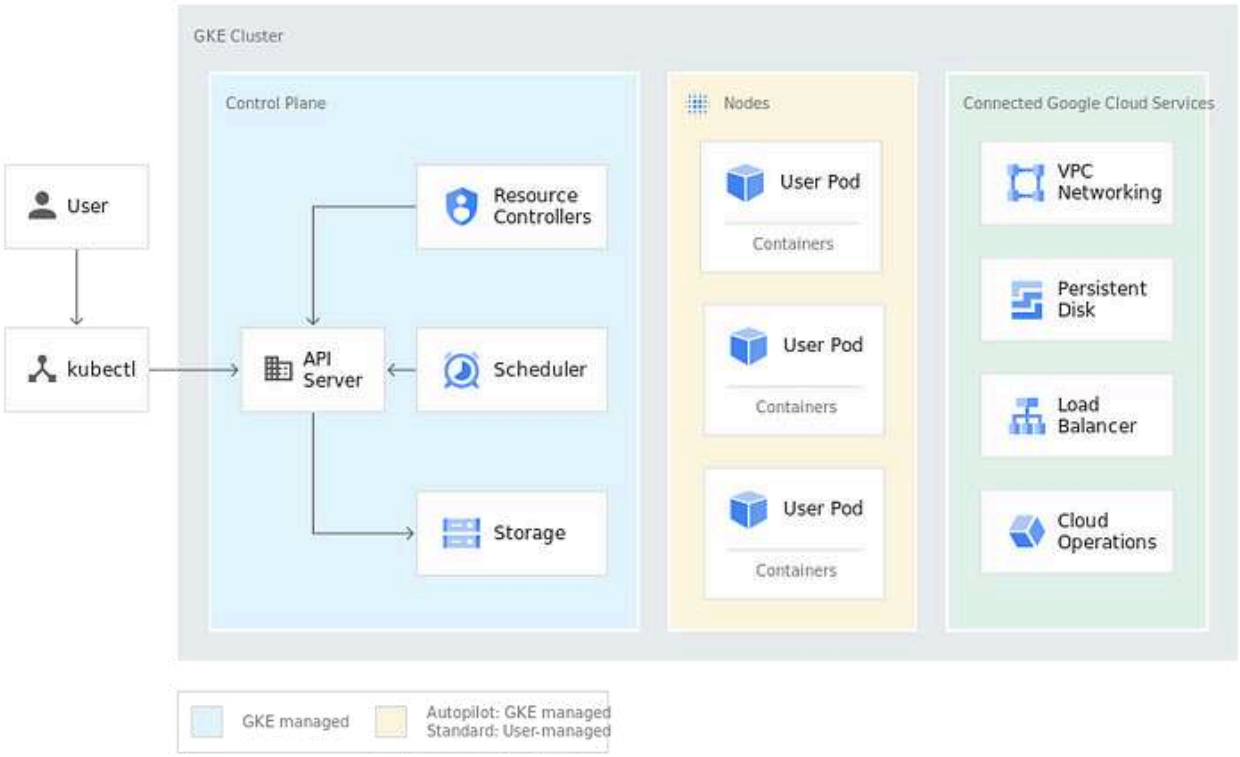
What is GKE?

GKE stands for Google Kubernetes Engine which is a managed Kubernetes service to deploy your application easily without worrying about Control Plane, Scaling, or updating to your Kubernetes Cluster.

One of the main advantages of GKE is that Kubernetes is developed by Google itself.

Key Features:

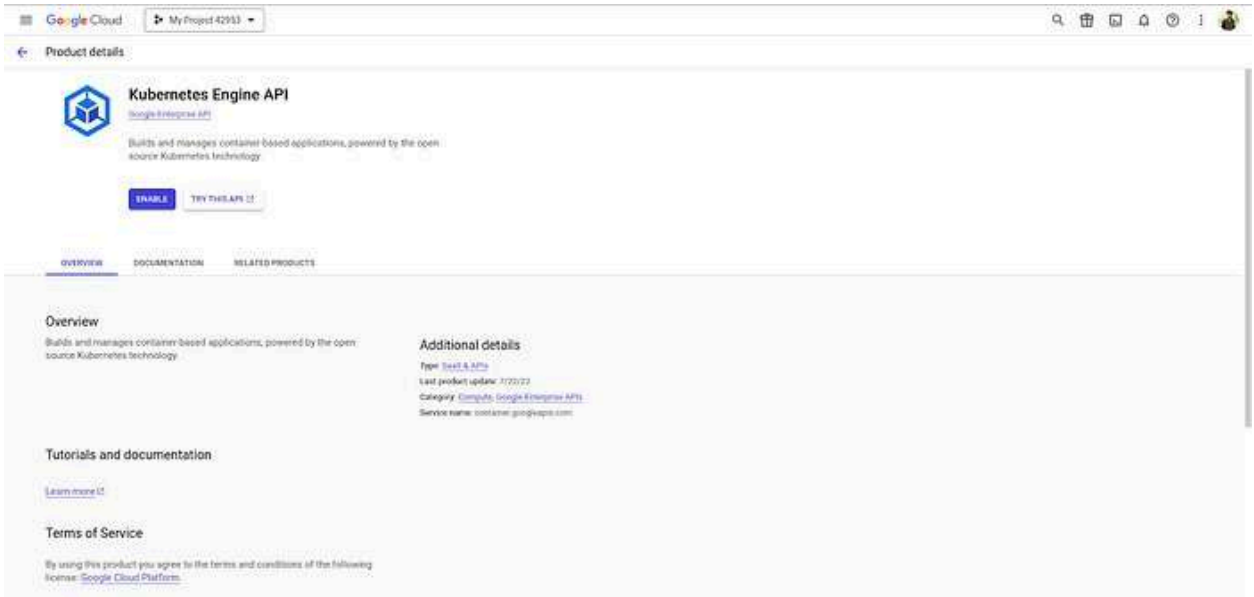
- **Multi-Versions** : GKE has the most available versions as Kubernetes developed Google
- itself.
- **Auto Upgrades** : The Control Plane and nodes will be automatically updated.
- **Auto Health Repair:** GKE provides automatic health repair for the nodes.
- **Security Enhancement:** GKE provides Containerized Optimised OS which will help you
- to enhance the security of your nodes.
- **Monitoring** : GKE provides monitoring support by integrating the monitoring services with GKE itself.
  Scalability : GKE provides scalability to your Kubernetes Clusters and Nodes according to the requirements.
  Multi-Cloud Support : You can run your applications anywhere without interruption using GKE on Google GKE, Azure AKS, or AWS EKS.



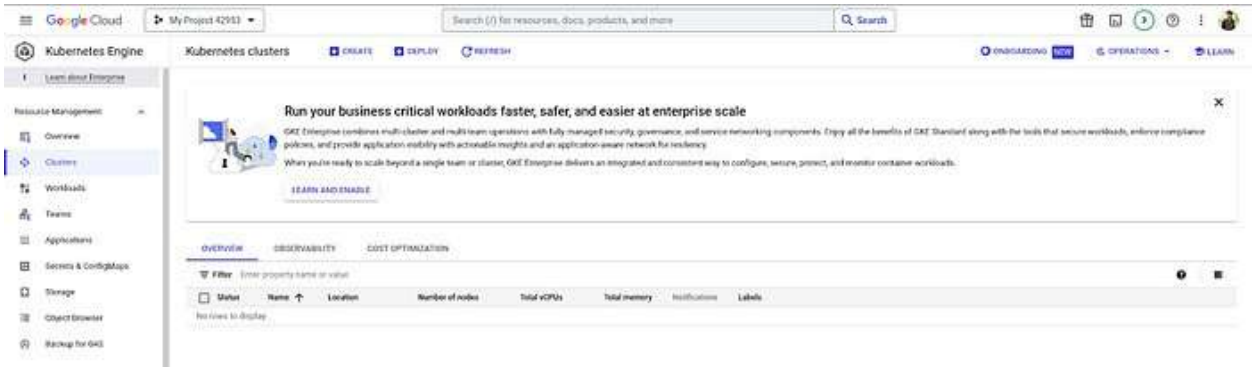
Let's Dive into the Demo!

If you have never created Kubernetes Cluster on Google Cloud before then, you have to enable it first to use it.

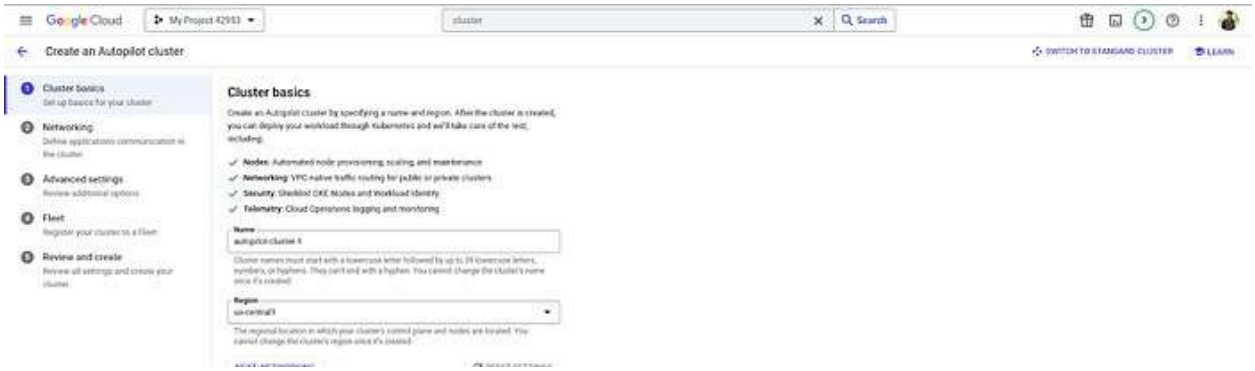
Click on Enable.



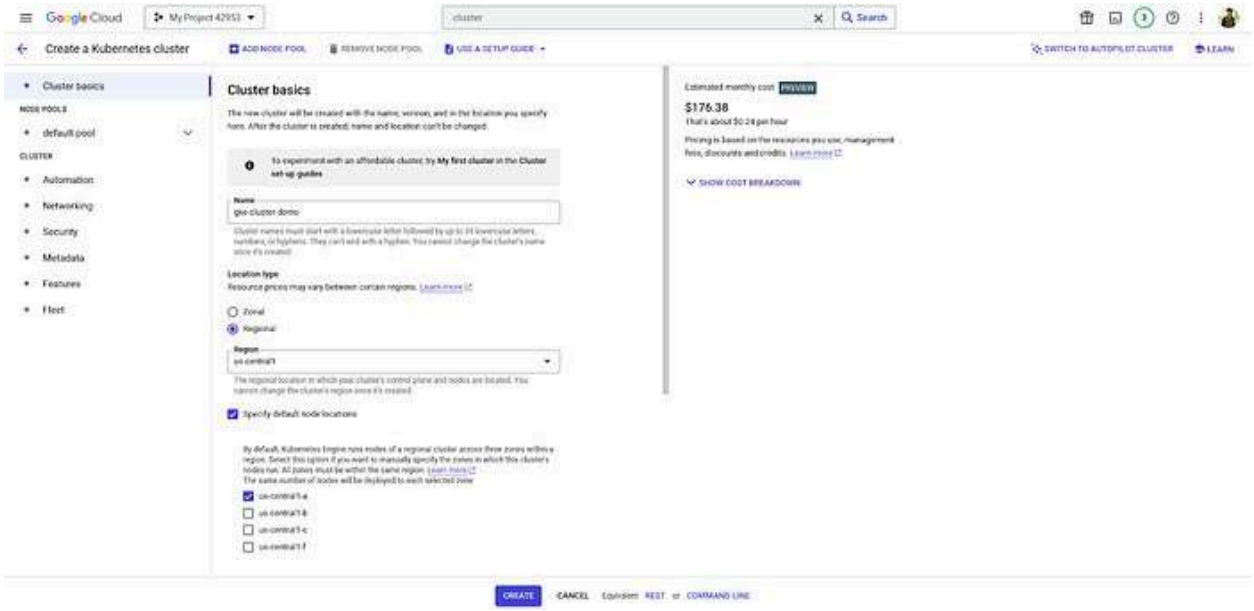
Click on CREATE



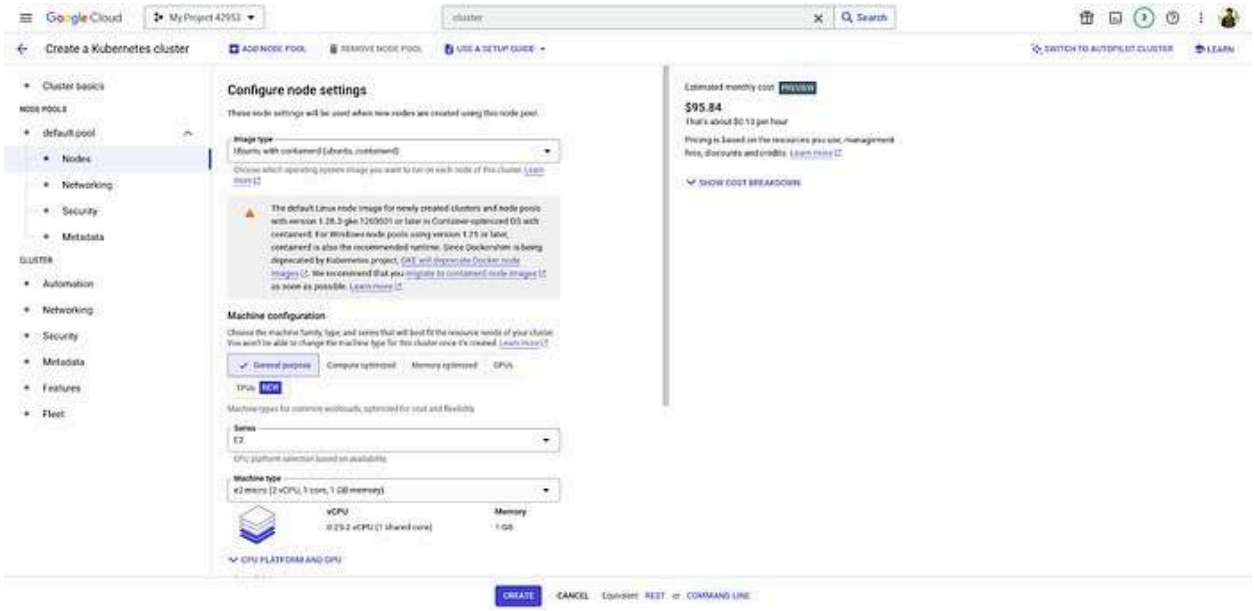
Now, we will create the Standard Kubernetes Cluster. To do that, click on SWITCH TO STANDARD CLUSTER.



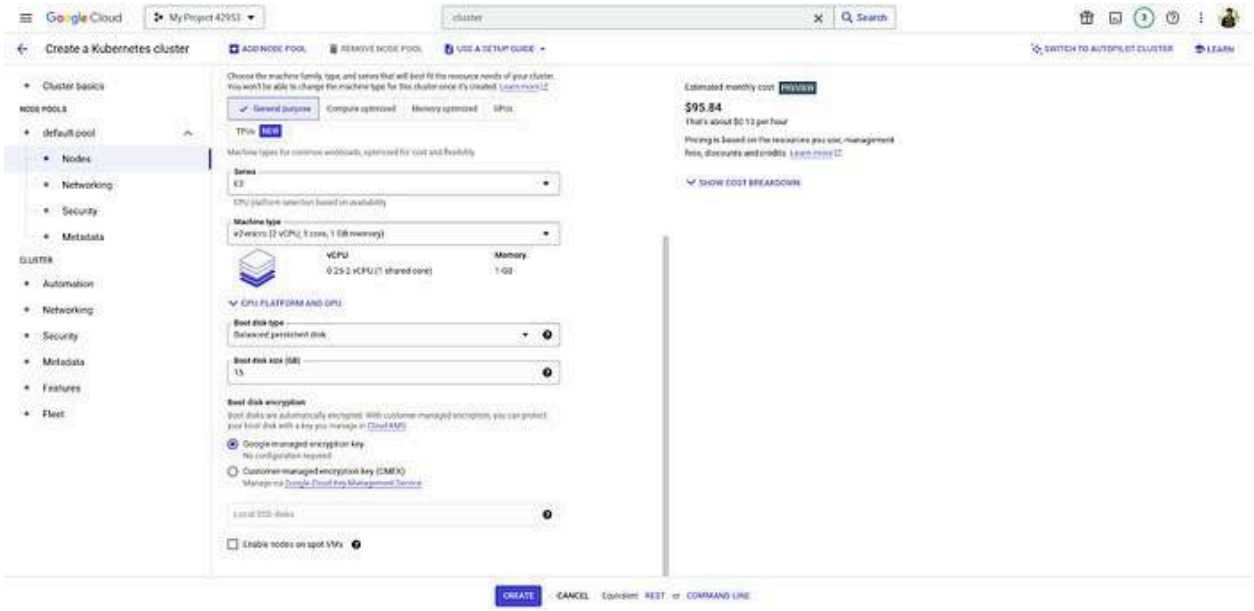
Provide the name to your cluster name and select the location type according to your credits remaining in the Google Cloud. Also, specify the node locations and click to expand default-pool which is showing on the left.



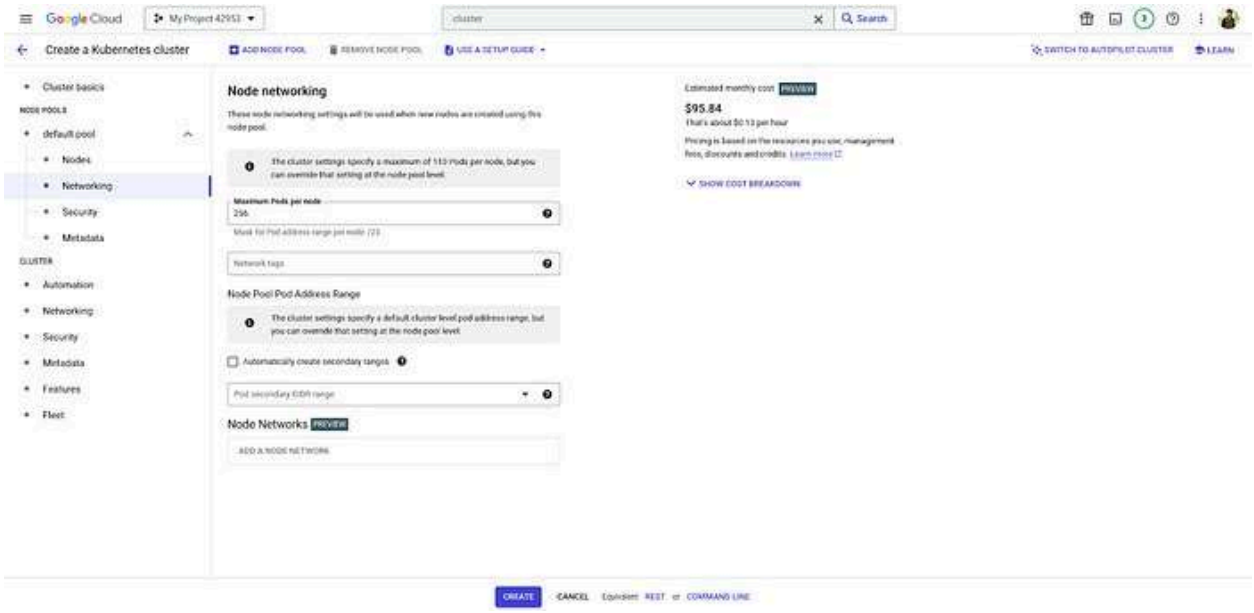
Select the configuration as given in the below screenshot to reduce some costs.



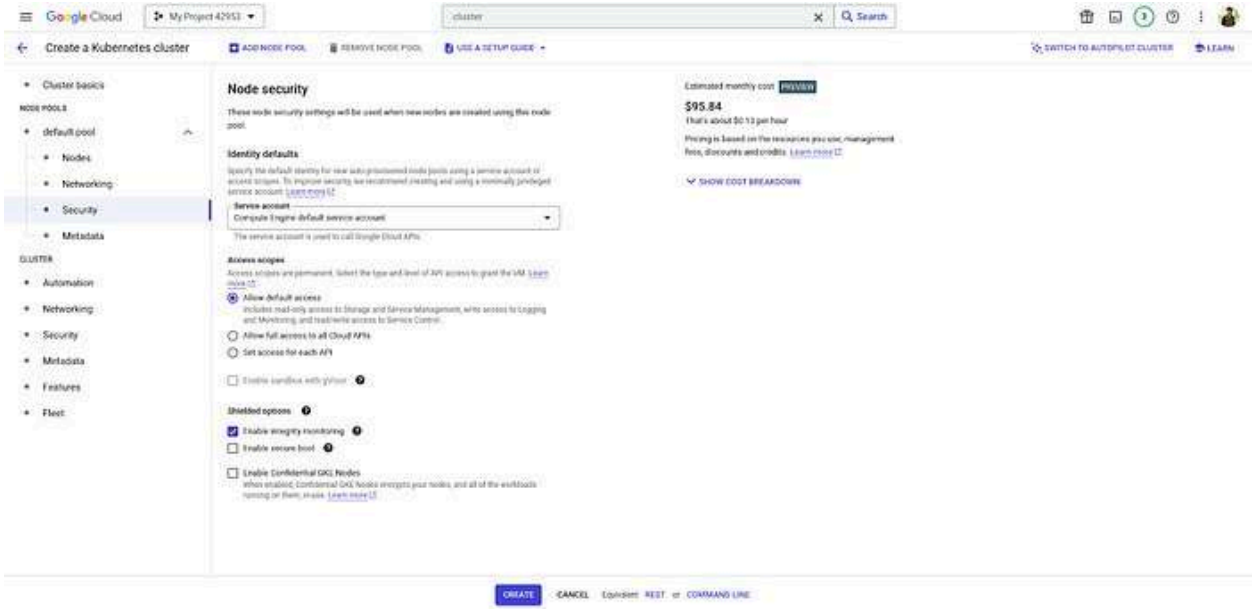
Provide the Root disk size of 15 GB because it will be sufficient according to our demonstration.



Click on the Networking and provide configurations as given below.



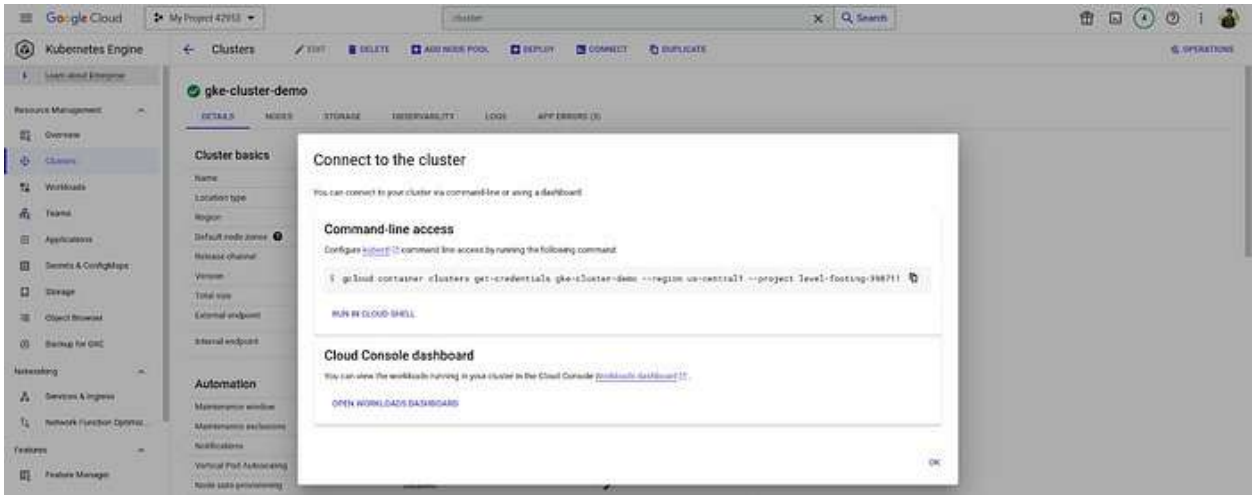
We don’t need to modify anything in Security and other things. So you can skip it and click on CREATE.



After clicking on CREATE. GCP will take some time to create the Kubernetes Cluster.

Once your Kubernetes Cluster is ready, then click on CONNECT.

You will get a command to configure it on your local machine. But there are two prerequisites that you have to follow which first is to configure gcloud-cli and install kubectl on your local machine. Let's do this.




```
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates gnupg curl sudo
curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg - dearmor -o
/usr/share/keyrings/cloud.google.gpg
echo "deb [signed-by=/usr/share/keyrings/cloud.google.gpg]
https://packages.cloud.google.com/apt cloud-sdk main" | sudo tee -a
/etc/apt/sources.list.d/google-cloud-sdk.list
```

Update your local machine and install Google Cloud cli

```
sudo apt-get update && sudo apt-get install google-cloud-cli
```

After installing gcloud cli, now we have to configure it with our account. To do that, run the below command and it will open your browser and select your main GCP account.

gcloud init

Now, you need one plugin to complete the configuration.

gcloud components install gke-gcloud-auth-plugin

After configuring Google Cloud CLI our first prerequisite is completed.
Now, we have to install kubectl which is mandatory to perform the commands on the Google Kubernetes Cluster.

To install kubectl on your local machine follow the below commands.

curl -O
https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.3/2023-11-14/bin/linux/amd64/kubectl

curl -O
https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.3/2023-11-14/bin/linux/amd64/kubectl.sha256

sha256sum -c kubectl.sha256

openssl sha1 -sha256 kubectl

chmod +x ./kubectl

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
```


```
kubectl version --client
```

Now, copy the command that is given by GCP to connect to the Kubernetes Cluster and paste it into your local machine.

To validate whether your cluster is working or not, list the nodes.

```
kubectl get nodes
```

Now, let’s try to run the static application on nginx server with a GCP load balancer

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-app-deployment
 labels:
   app: nginx-app
spec:
 replicas: 2
 selector:
   matchLabels:
     app: nginx-app
 template:
   metadata:
     labels:
       app: nginx-app
   spec:
     containers:
     -name: nginx-container
       image: avian19/nginx-ne:latest-1.0
       ports:
       -containerPort: 80
```

```
kubectl apply -f deployment.yml
```

```
anaspathak@pop-os:~/Github/Kubernetes-Files/Google-GKE$ kubectl apply -f deployment.yml
deployment.apps/nginx-app-deployment created
anaspathak@pop-os:~/Github/Kubernetes-Files/Google-GKE$
anaspathak@pop-os:~/Github/Kubernetes-Files/Google-GKE$ kubectl get deploy
NAME                   READY   UP-TO-DATE   AVAILABLE   AGE
nginx-app-deployment   2/2     2            2           11s
anaspathak@pop-os:~/Github/Kubernetes-Files/Google-GKE$
anaspathak@pop-os:~/Github/Kubernetes-Files/Google-GKE$ kubectl get po
NAME                                    READY   STATUS    RESTARTS   AGE
nginx-app-deployment-d9554578d5-chmbk   1/1     Running   0          16s
nginx-app-deployment-d9554578d5-spww9   1/1     Running   0          16s
anaspathak@pop-os:~/Github/Kubernetes-Files/Google-GKE$
```

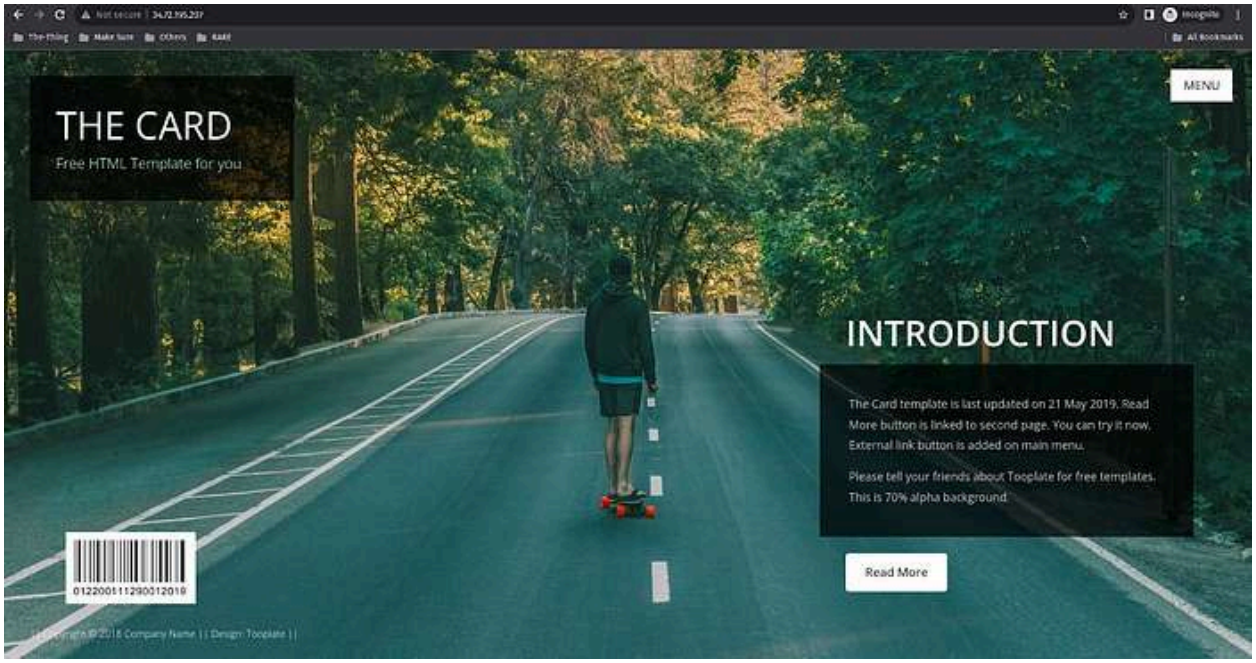
Now, host the application outside of the Kubernetes Cluster by creating a service for the nginx application and observe the Public IP in the EXTERNAL-IP Column.

```
apiVersion: v1
kind: Service
metadata:
 name: nginx
spec:
 selector:
   app: nginx-app
 type: LoadBalancer
 ports:
 -protocol: TCP
   port: 80
```

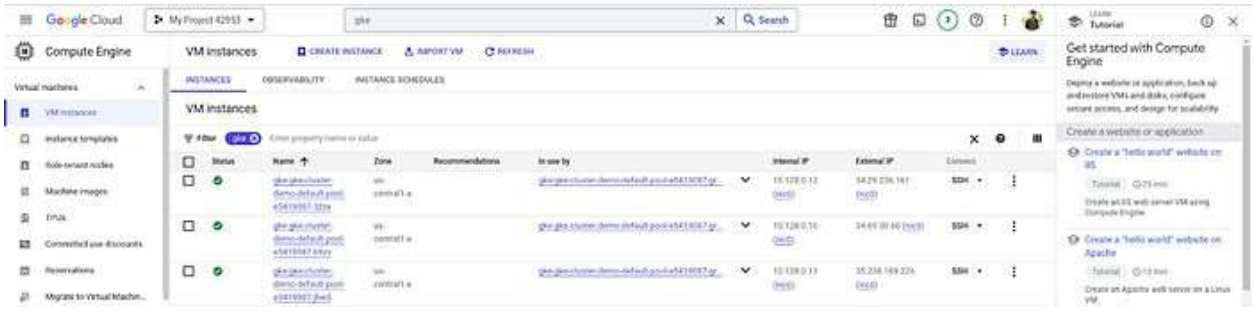
```
kubectl apply -f svc.yaml
```

```
anaspathak@pop-os:~/Github/Kubernetes-Files/Google-GKE$ kubectl apply -f svc.yaml
service/nginx created
anaspathak@pop-os:~/Github/Kubernetes-Files/Google-GKE$
anaspathak@pop-os:~/Github/Kubernetes-Files/Google-GKE$ kubectl get svc
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP      10.120.0.1     <none>        443/TCP        11m
nginx        LoadBalancer   10.120.4.246   <pending>     80:32595/TCP   9s
anaspathak@pop-os:~/Github/Kubernetes-Files/Google-GKE$
anaspathak@pop-os:~/Github/Kubernetes-Files/Google-GKE$ kubectl get svc
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
kubernetes   ClusterIP      10.120.0.1     <none>          443/TCP        12m
nginx        LoadBalancer   10.120.4.246   34.72.195.207   80:32595/TCP   72s
anaspathak@pop-os:~/Github/Kubernetes-Files/Google-GKE$
```

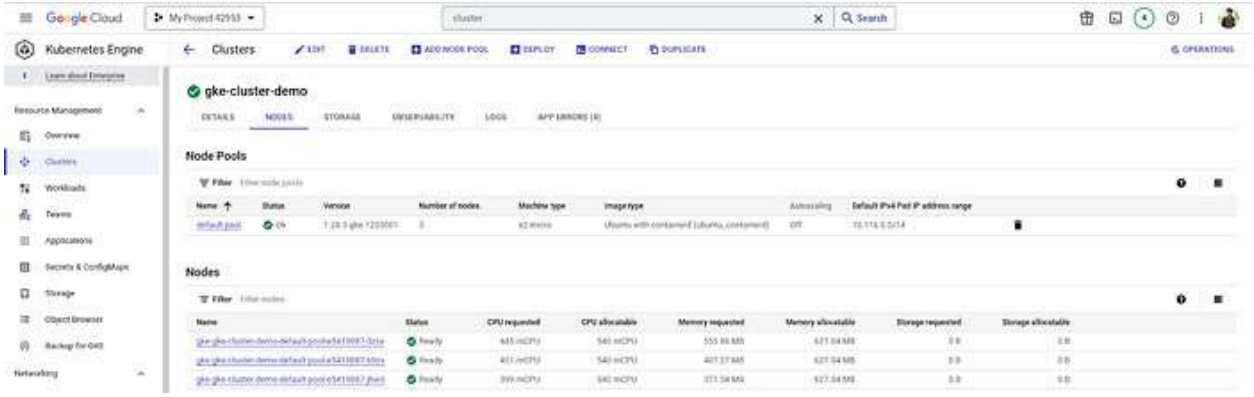
Copy the Public IP and paste it into your favorite browser and see the magic.



You can also decrease the number of nodes that are running in the below screenshot.

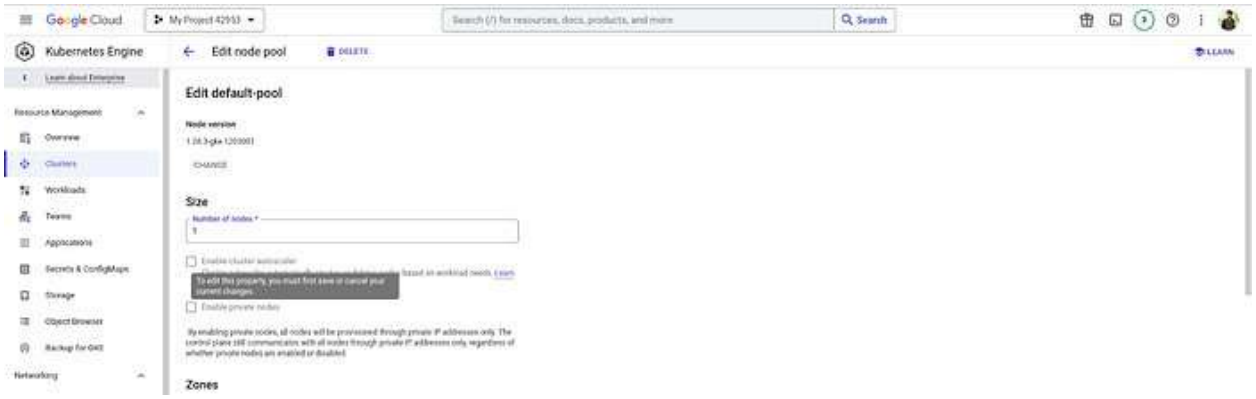


After going to the Cluster, click on default-pool which is showing under Node Pools.

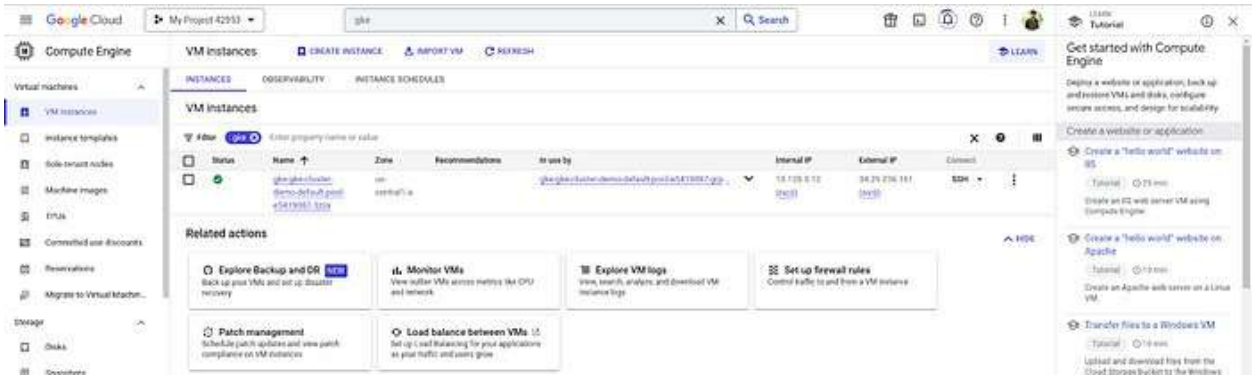


Click on edit

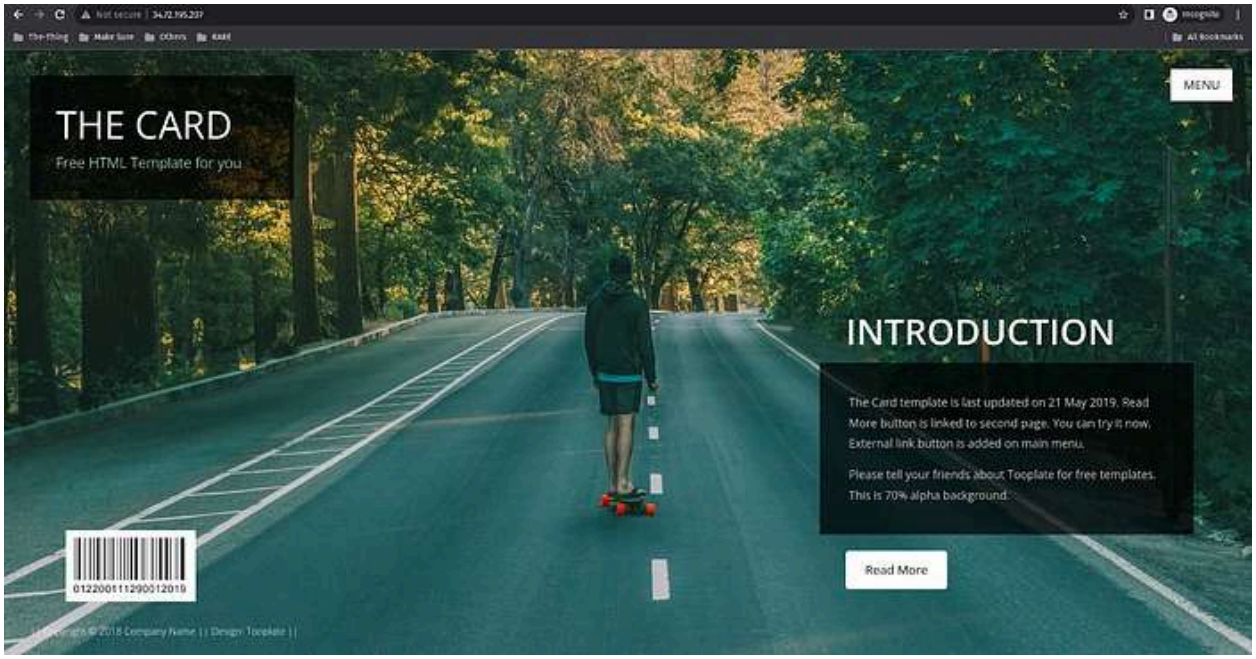
Replace 3 with the 1 to reduce the nodes then scroll down and click on Save.



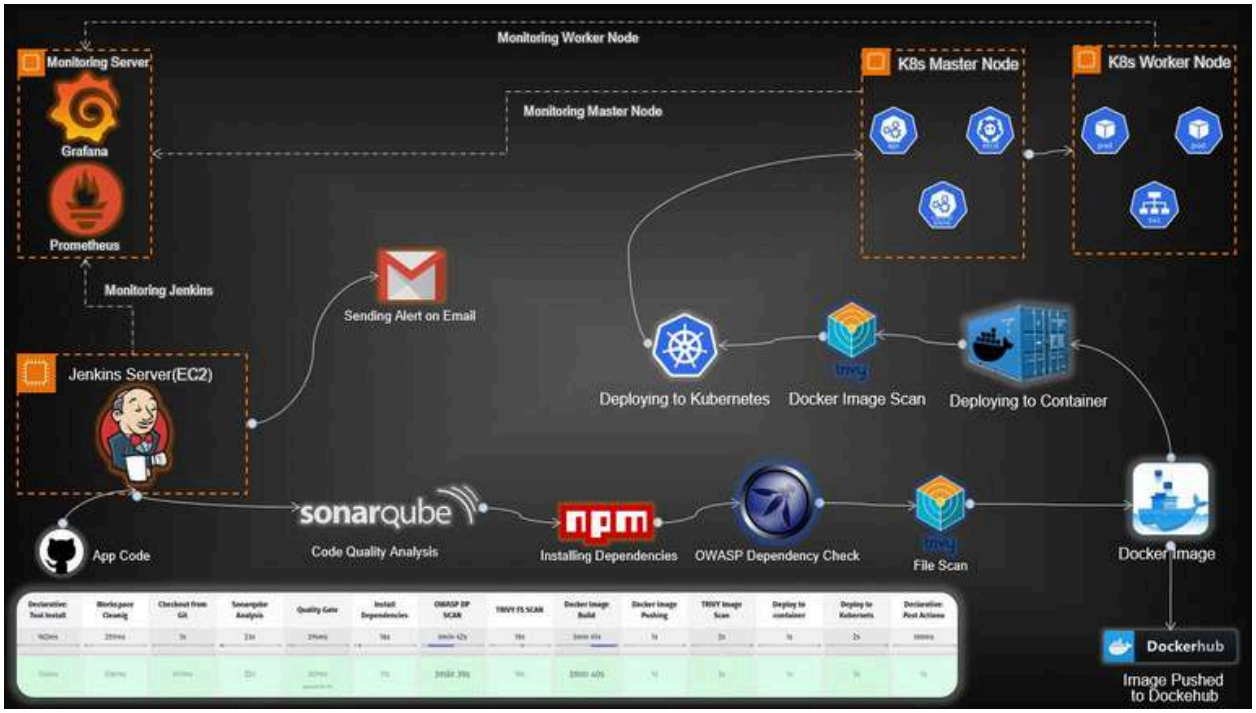
Once you Save the configurations, then you will see the VM Instances will be deleted.



Don’t worry your application will still running



End-to-End DevSecOps Kubernetes Project



Introduction:

In today's rapidly evolving tech landscape, deploying applications using Kubernetes has become a crucial aspect of modern software development. This guide provides a detailed walkthrough for setting up an end-to-end Kubernetes project, covering everything from infrastructure provisioning to application deployment and monitoring.

Prerequisites:

Before diving into the implementation, ensure you have the following in place:

- Basic understanding of Kubernetes concepts.
- Access to AWS or any other cloud provider for server instances.
- ATMDB API key for accessing movie databases in your Netflix Clone application.
- DockerHub account for pushing and pulling Docker images.
- Gmail account for email notifications.
- Jenkins, Kubernetes, Docker, and necessary plugins installed.

High-Level Overview:

1. **Infrastructure Setup** : Provisioned servers for Jenkins, Monitoring, and Kubernetes nodes.
2. **Toolchain Integration** : Integrated essential tools like Jenkins, SonarQube, Trivy, Prometheus, Grafana, and OWASP Dependency-Check.
3. **Continuous Integration/Continuous Deployment (CI/CD)** : Automated workflows with Jenkins pipelines for code analysis, building Docker images, and deploying applications on Kubernetes.
4. **Security Scanning:** Implemented Trivy and OWASP Dependency-Check to scan for vulnerabilities in code and Docker images.
5. **Monitoring and Visualization** : Set up Prometheus and Grafana for real-time monitoring and visualization of both hardware and application metrics.
6. **Email Notifications** : Configured Jenkins for email alerts based on pipeline results.

You will get the Jenkinsfile and Kubernetes Manifest files along with the Dockerfile. Feel free to modify it accordingly

Project GitHub Repo-

https://github.com/AmanPathak-DevOps/Netflix-Clone-K8S-End-to-End-Project

We need four servers for our today's Project

Jenkins Server- On this Server, Jenkins will be installed with some other tools such as sonarqube(docker container), trivy, and kubectl.

Monitoring Server-This Server will be used for Monitoring where we will use Prometheus, Node Exporter, and Grafana.

Kubernetes Master Server-This Server will be used as the Kubernetes Master Cluster Node which will deploy the applications on worker nodes.

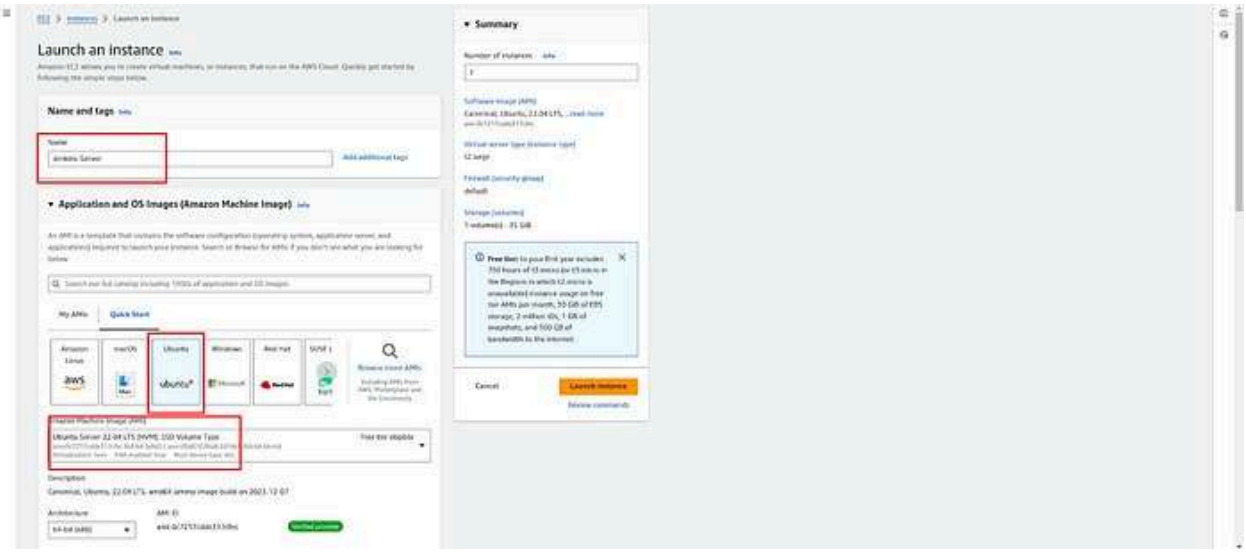
Kubernetes Worker Server-This Server will be used as the Kubernetes Worker Node on which the application will be deployed by the master node.

Let's create the following instances.

Jenkins Server

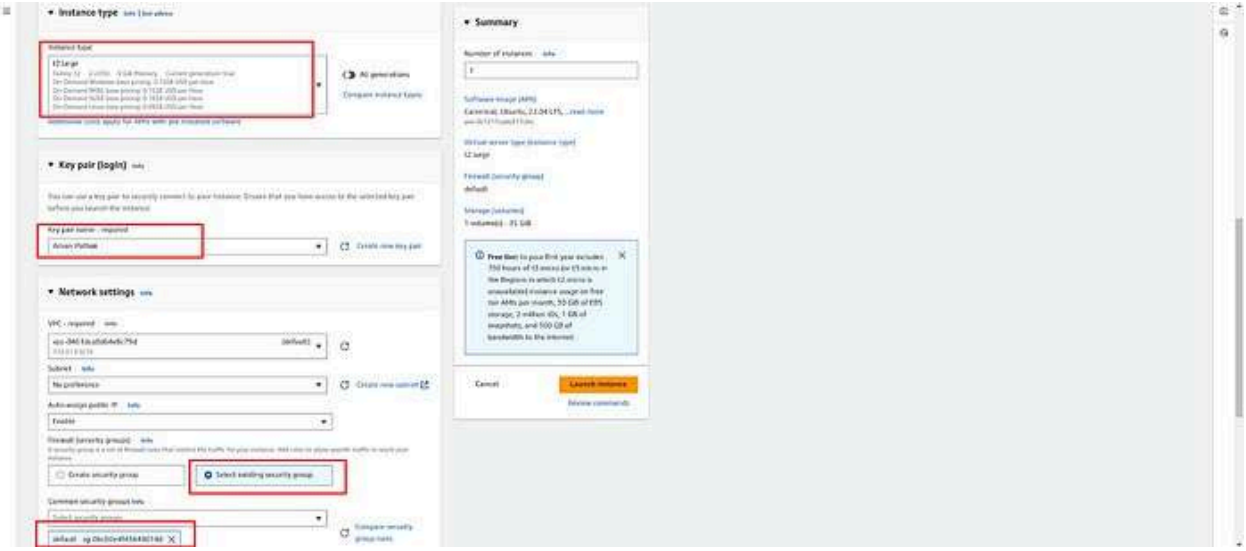
Click on Launch Instances.

Provide the name of your Jenkins instance, and select the Ubuntu OS 22.04 version.

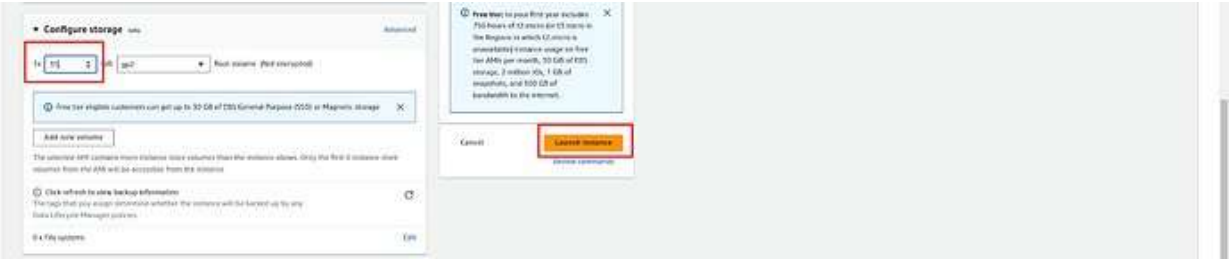


We need to configure multiple things on the Jenkins instance. So, select the t2.large instance type, provide the key or you can create if you want.

Keep the networking things as it is. But make sure to open all inbound and outbound traffic in the selected security groups.

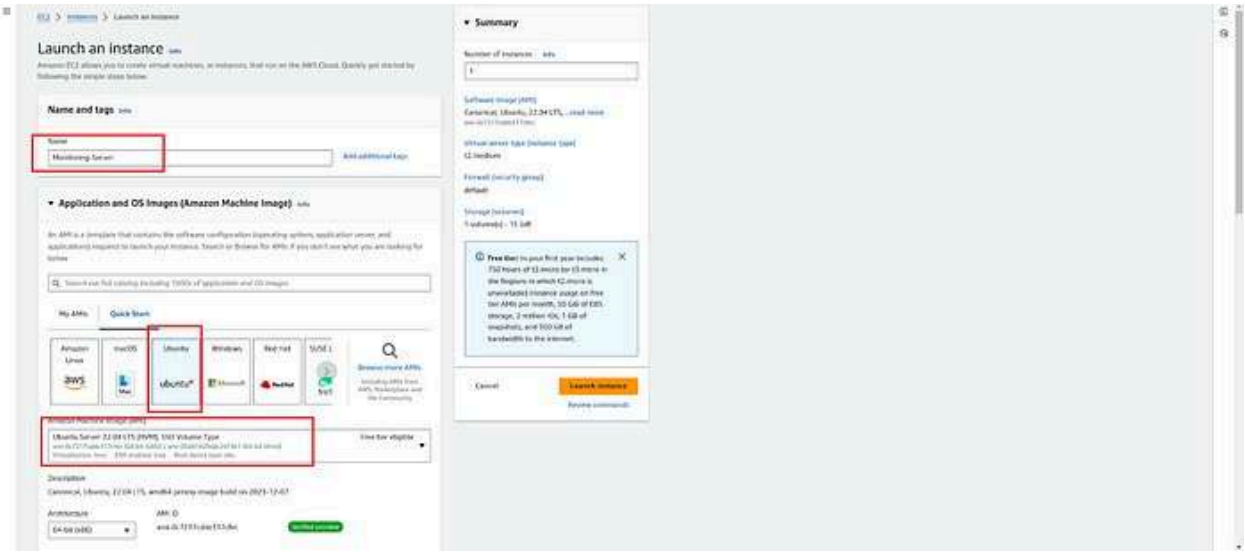


Increase the storage capacity for Jenkins Instance from 8GB to 35GB and click on Launch Instance.



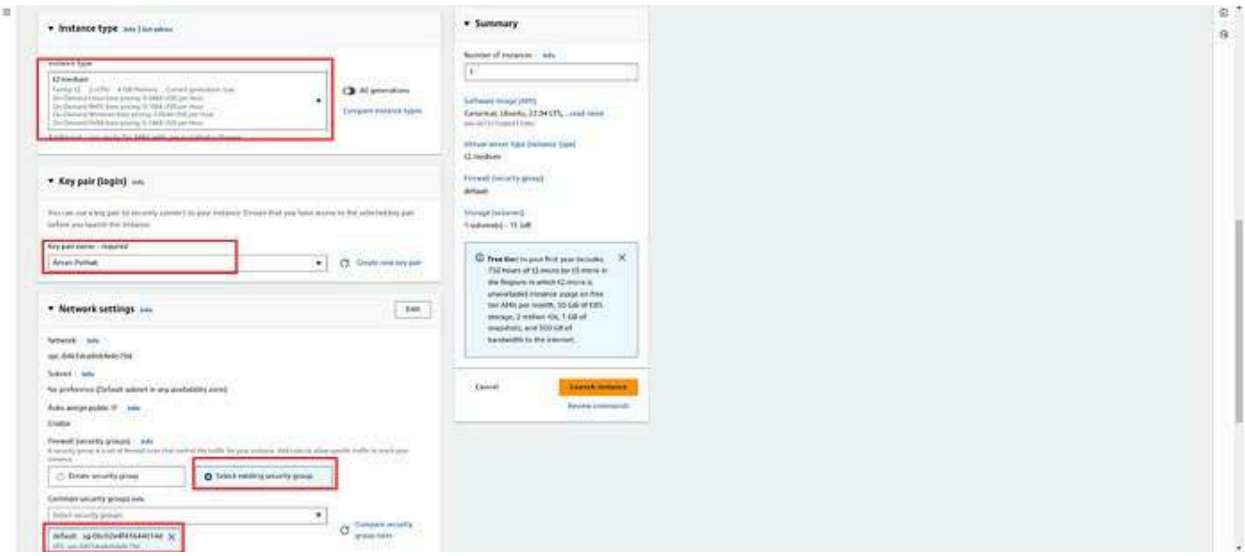
Monitoring Server

Provide the name of your Monitoring Instance, and select the Ubuntu 22.04 OS.

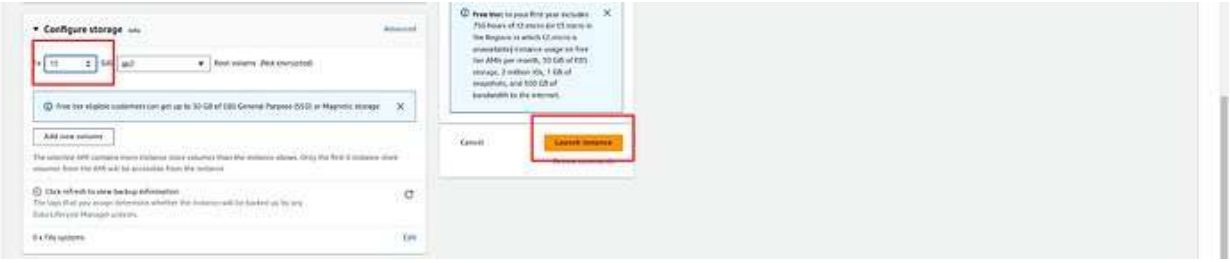


We need to configure the monitoring tools on this instance which needs a minimum of 4GB RAM. So, select the t2.medium instance type, provide the key or you can create if you want.

Keep the networking things as it is. But make sure to open all inbound and outbound traffic in the selected security groups.



Increase the storage capacity for Jenkins Instance from 8GB to 15GB and click on Launch Instance.

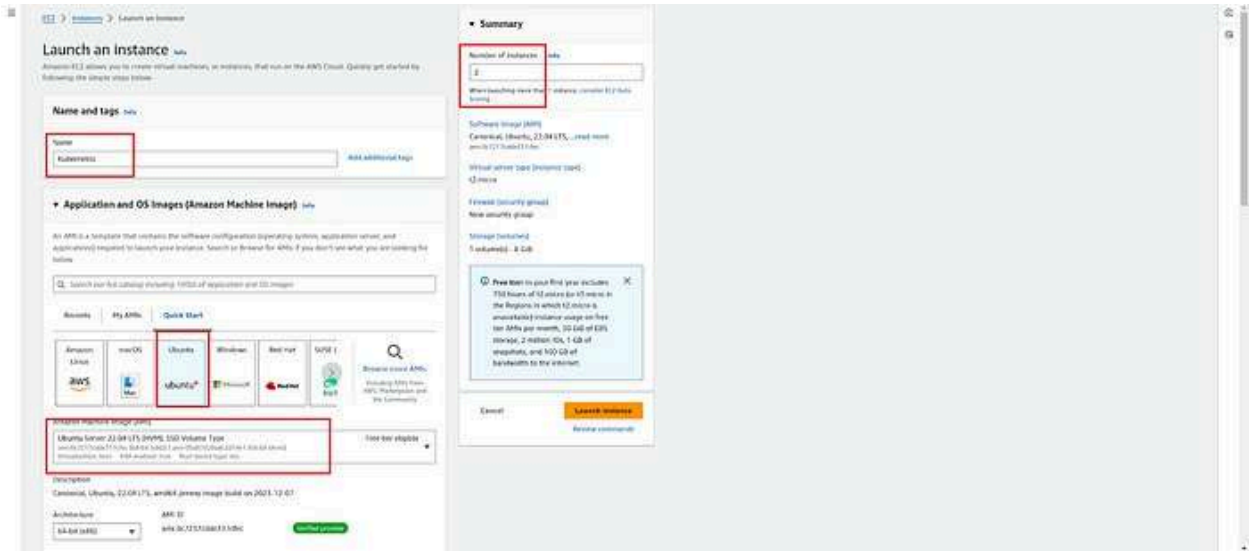


Kubernetes Master & Worker Node

We have to create two Kubernetes Nodes which need at least 2 CPUs.

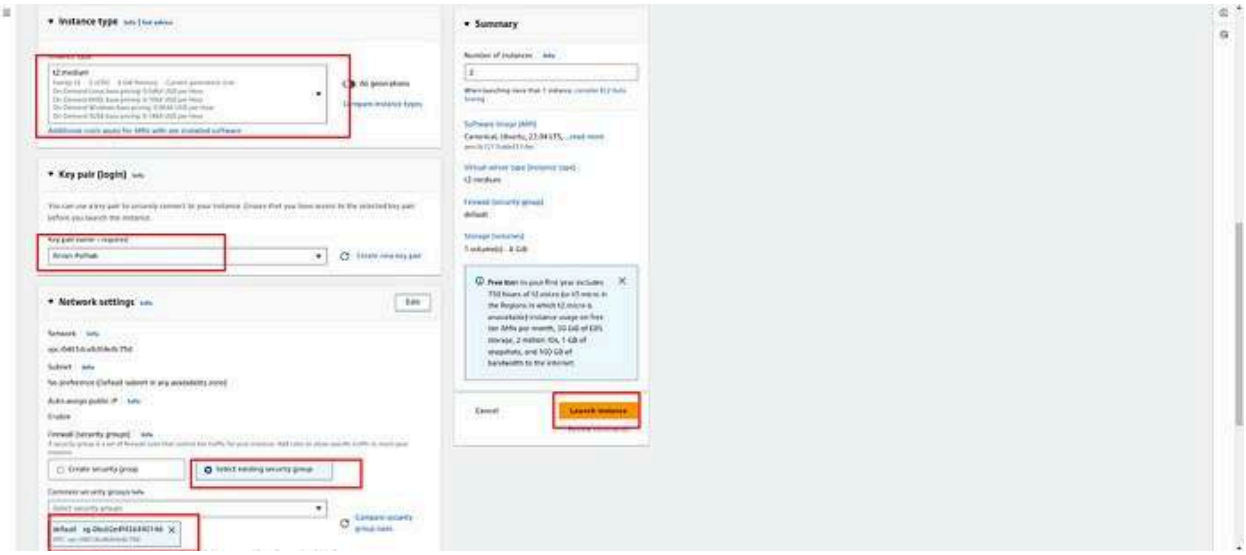
Provide the name of your Kubernetes Master Instance, and select the Ubuntu 22.04 OS.

In the Number of Instances, replace 1 with 2 because we need two Kubernetes Nodes.



Select the t2.medium instance type, provide the key or you can create if you want.

Keep the networking things as it is. But make sure to open all inbound and outbound traffic in the selected security groups then keep the rest of the things as it is and click on Launch Instance.



Rename the Kubernetes Servers and all four servers will look like the below snippet.

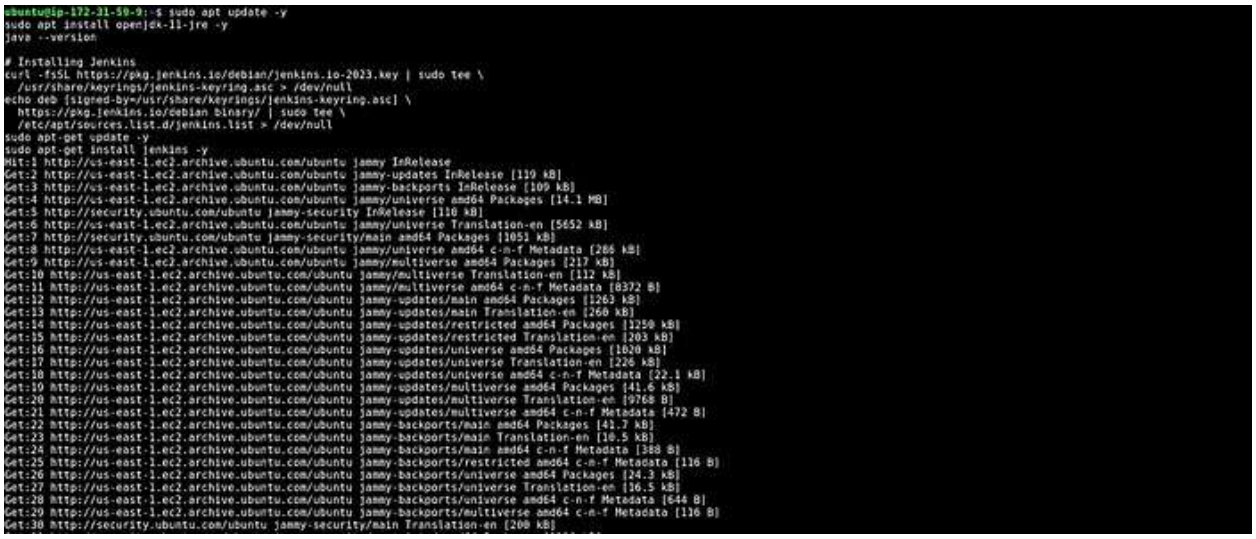
Log in to the Jenkins Server

```
amanpathak@pop-os:~/Downloads$ ssh -i "Aman-Pathak.pem" ubuntu@ec2-34-207-155-151.compute-1.amazonaws.com
The authenticity of host 'ec2-34-207-155-151.compute-1.amazonaws.com (34.207.155.151)' can't be established.
ED25519 key fingerprint is SHA256:VDNhbgo9h5obdeNH703RppSc/vdHT12F2j8B9LE43Ic.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-34-207-155-151.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 6.2.0-1017-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Wed Dec 27 13:57:18 UTC 2023

  System load:  0.16015625        Processes:             105
  Usage of /:   4.6% of 33.74GB   Users logged in:       0
  Memory usage: 2%                IPv4 address for eth0: 172.31.59.9
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-59-9:~$
```

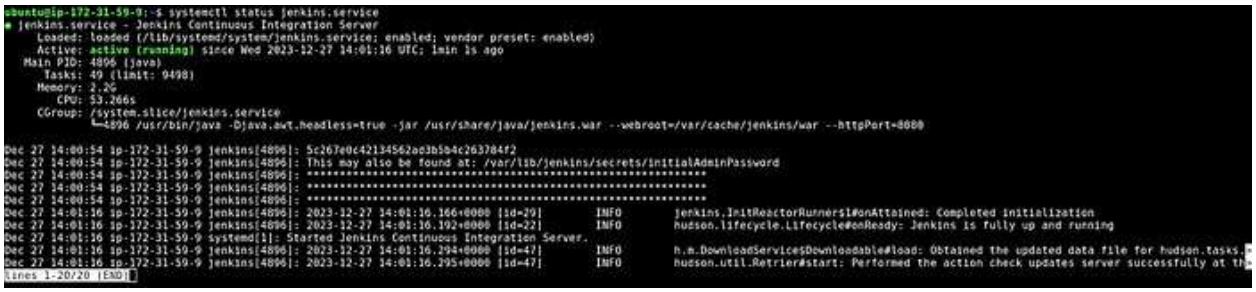
Download Open JDK and Jenkins

```
#Intsalling Java
sudo apt update -y
sudo apt install openjdk-11-jre -y
java --version
#Installing Jenkins
curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee \
```

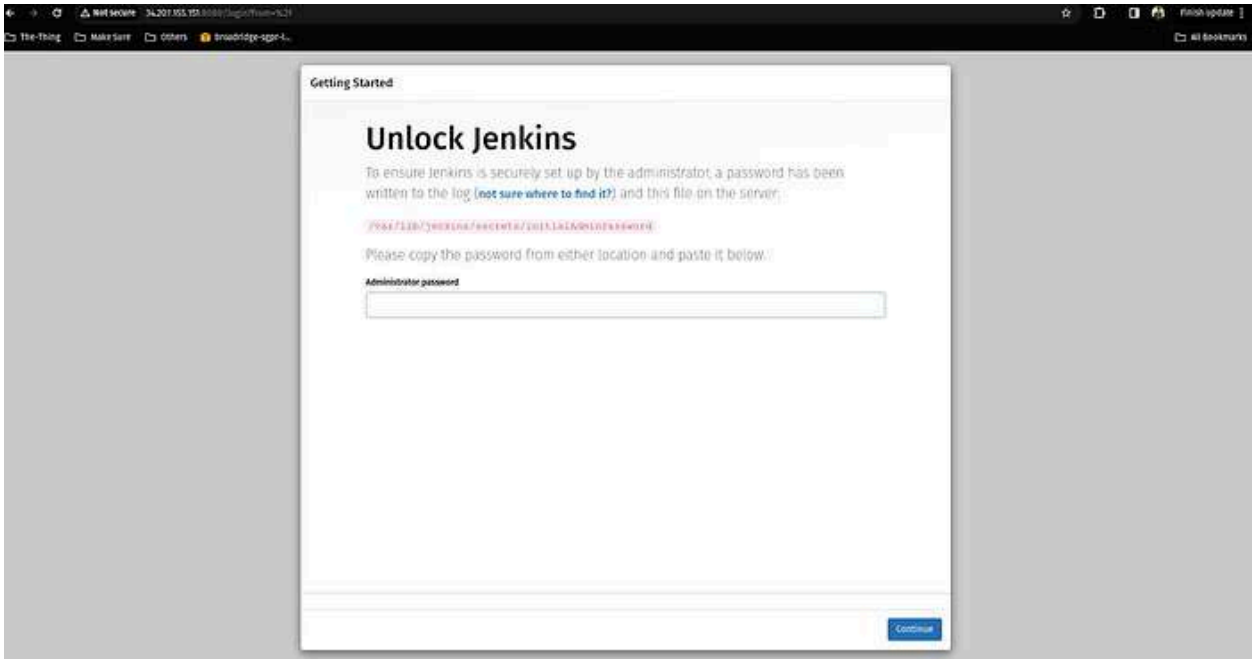
```
  /usr/share/keyrings/jenkins-keyring.asc > /dev/nul
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
 https://pkg.jenkins.io/debian binary/ | sudo tee \
 /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update -y
sudo apt-get install jenkins -y
```



Check the status of the Jenkins server



Copy your Jenkins Server Public IP and paste it into your favorite browser with port number 8080.



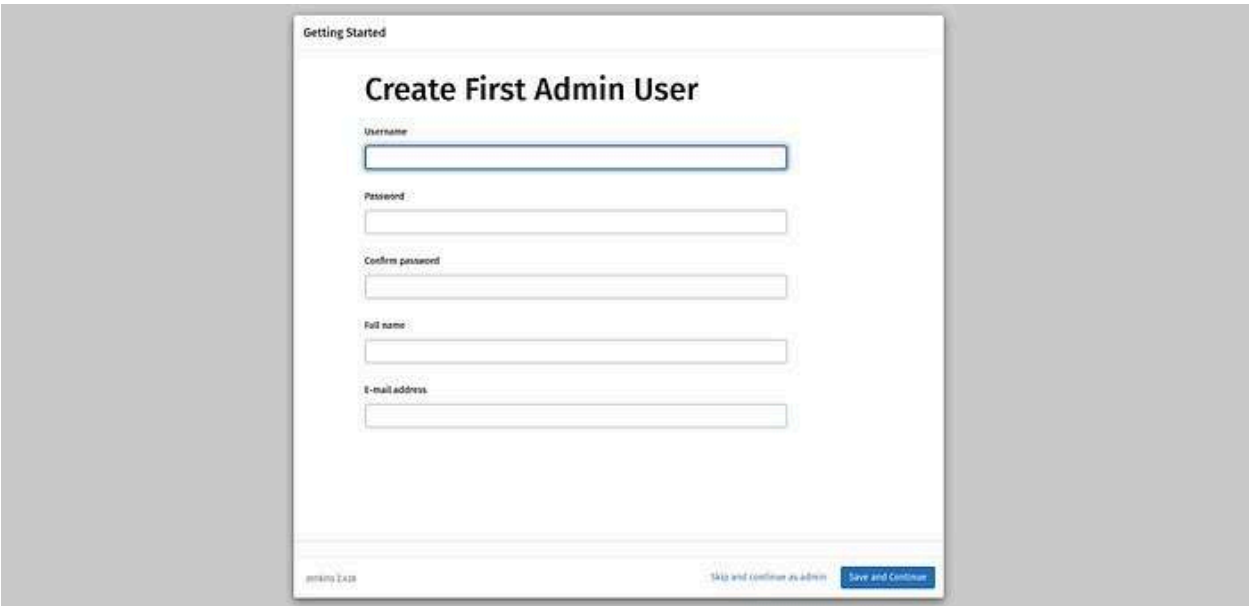
Run the command on your Jenkins server

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Copy the output and paste it into your above snippet text field and click on Continue.

Click on the Install suggested plugins

Click on the Skip and continue as admin



Click on Save and Finish

Install Docker and configure on the Jenkins Server

```
sudo apt update
usermod -aG docker jenk
docker ubuntu sudo systemctl re
chmod 777 /var/run/docker.sock
```

```
ubuntu@ip-172-31-59-9:~$ sudo apt update
sudo apt install docker.io
sudo su -
sudo usermod -aG docker jenkins
sudo usermod -aG docker ubuntu
systemctl restart docker
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Ign:4 https://pkg.jenkins.io/debian binary/ InRelease
Hit:5 https://pkg.jenkins.io/debian binary/ Release
Hit:6 http://security.ubuntu.com/ubuntu jammy-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
26 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io pigz runc ubuntu-fan
0 upgraded, 8 newly installed, 0 to remove and 26 not upgraded.
Need to get 69.7 MB of archives.
```

Install Sonarqube on your Jenkins Server

We will use a docker container for Sonarqube

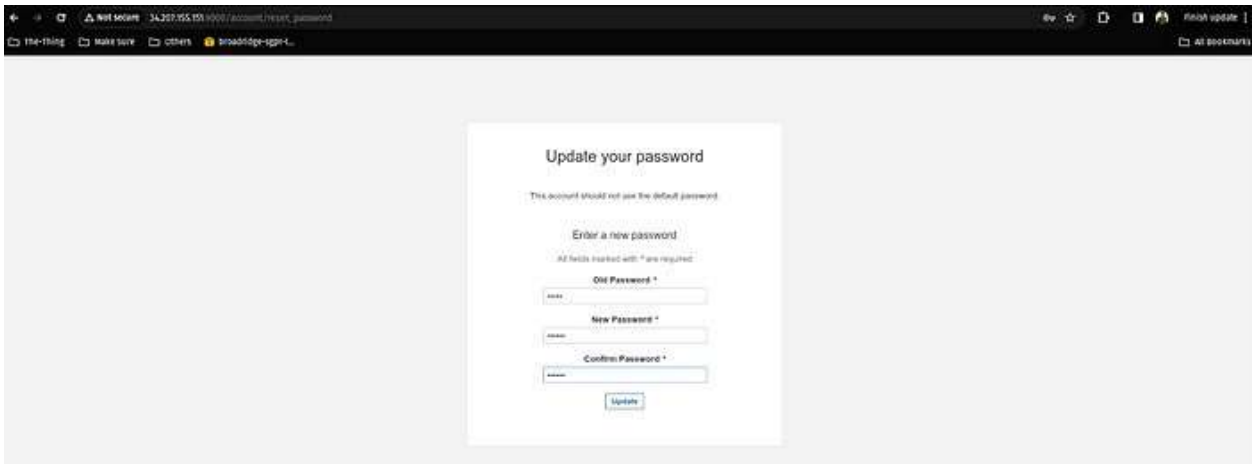
```
docker run -d — name sonar -p 9000:9000 sonarqube:lts-community
```

```
ubuntu@ip-172-31-59-9:~$ docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
Unable to find image 'sonarqube:lts-community' locally
lts-community: Pulling from library/sonarqube
3dd1811f9be59: Pull complete
0f838805bddf: Pull complete
e7eee5bcd0eb: Pull complete
515260e7965d8: Pull complete
ffcdc7c6c160: Pull complete
0d141c530e5b: Pull complete
bb9af13b2efe: Pull complete
Digest: sha256:49ac473fc9da07052cdd205e4581a5b369adeaf65832830d62be86e419ea2e1f
Status: Downloaded newer image for sonarqube:lts-community
9896b3945a5dbdb0714716d59336e053a149779b898d488a0ca9847e0de1fddc
ubuntu@ip-172-31-59-9:~$
```

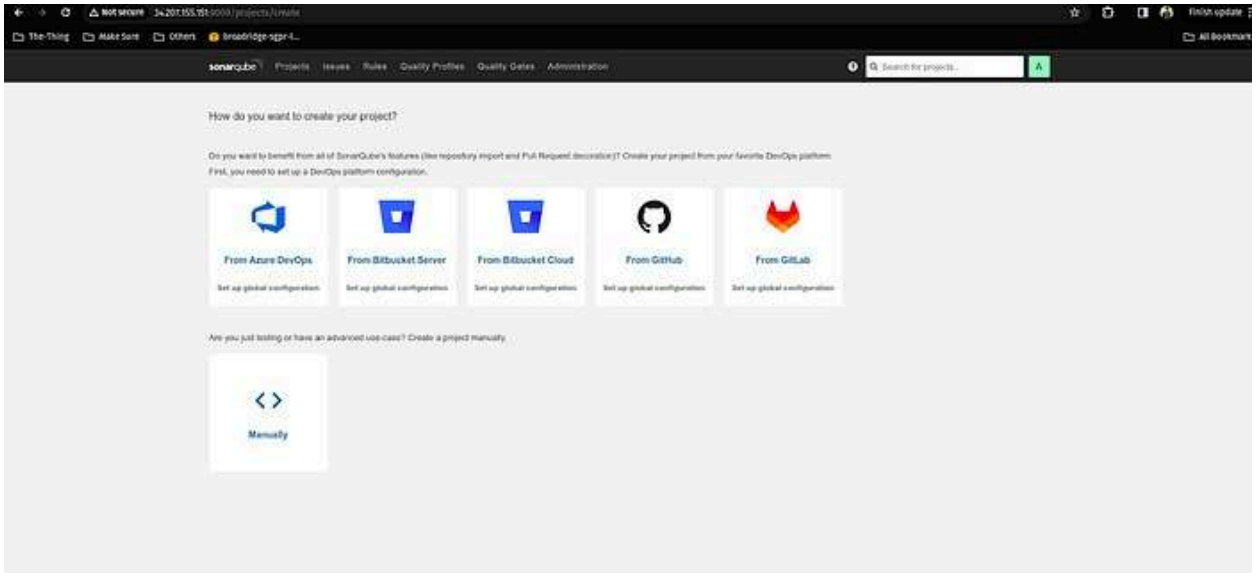
Now, copy your Public IP of Jenkins Server and add 9000 Port on your browser.

The username and password will be admin

Reset the password and click on Update



You will see your Sonarqube Server in the below snippet.



Install the Trivy tool on the Jenkins Server

```
sudo apt-get install wget apt-transport-https gnupg lsb-release
wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key add -
echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main | sudo tee -a /etc/apt/sources.list.d/trivy.list
sudo apt-get update
sudo apt-get install trivy
```

```
ubuntu@ip-172-31-59-9:~$ sudo apt-get install wget apt-transport-https gnupg lsb-release
wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key add -
echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main | sudo tee -a /etc/apt/sources.list.d/trivy.list
sudo apt-get update
sudo apt-get install trivy
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
lsb-release is already the newest version (11.1.0ubuntu4).
lsb-release set to manually installed.
wget is already the newest version (1.21.2-2ubuntu1).
wget set to manually installed.
gnupg is already the newest version (2.2.27-3ubuntu2.1).
gnupg set to manually installed.
The following NEW packages will be installed:
  apt-transport-https
0 upgraded, 1 newly installed, 0 to remove and 26 not upgraded.
Need to get 1510 B of archives.
After this operation, 170 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 apt-transport-https all 2.4.11 [1510 B]
Fetched 1510 B in 0s (90.5 kB/s)
Selecting previously unselected package apt-transport-https.
(Reading database ... 66052 files and directories currently installed.)
Preparing to unpack .../apt-transport-https_2.4.11_all.deb ...
Unpacking apt-transport-https (2.4.11) ...
Setting up apt-transport-https (2.4.11) ...
Scanning processes...
Scanning linux images...
```

Install and Configure the Prometheus, Node Exporter, and Grafana on the Monitoring Server

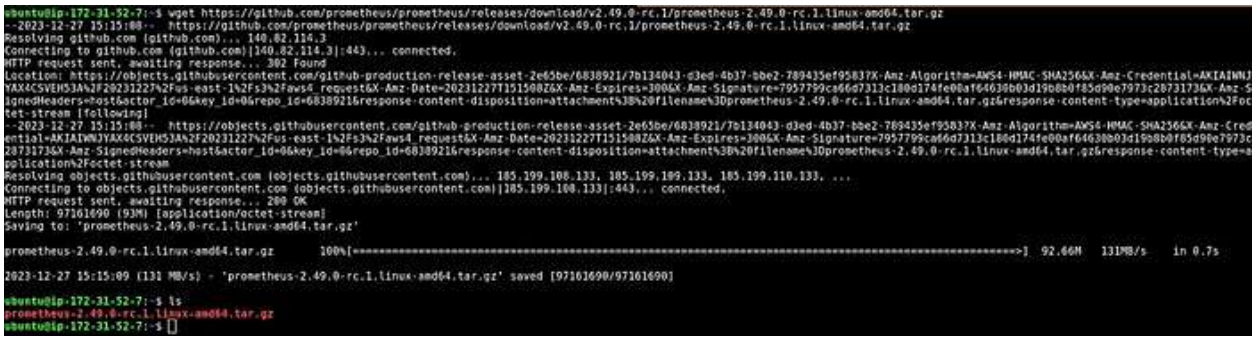
Login to the Monitoring Server

Create Prometheus user

```
sudo useradd \
-system \
-no-create-home \
-shell /bin/false prometheus
```

Download the Prometheus file on the Monitoring Server

```
wget
https://github.com/prometheus/prometheus/releases/download/v2.49.0-rc.1/prometheus-2.49.0-rc.1.linux-amd64.tar.gz
```



Untar the Prometheus downloaded package

```
tar -xvf prometheus-2.49.0-rc.1.linux-amd64.tar.gz
```

Create two directories /data and /etc/prometheus to configure the Prometheus

```
sudo mkdir -p /data /etc/prometheus
```

Now, enter into the prometheus package file that you have untar in the earlier step.

```
cd prometheus-2.49.0-rc.1.linux-amd64/
```

Move the prometheus and promtool files package in /usr/local/bin

```
sudo mv prometheus promtool /usr/local/bin/
```

Move the console and console_libraries and prometheus.yml in the /etc/prometheus

```
sudo mv consoles console_libraries/ prometheus.yml /etc/prometheus/
```

Provide the permissions to prometheus user

```
sudo chown -R prometheus:prometheus /etc/prometheus/ /data/
```

Check and validate the Prometheus

```
prometheus --version
```

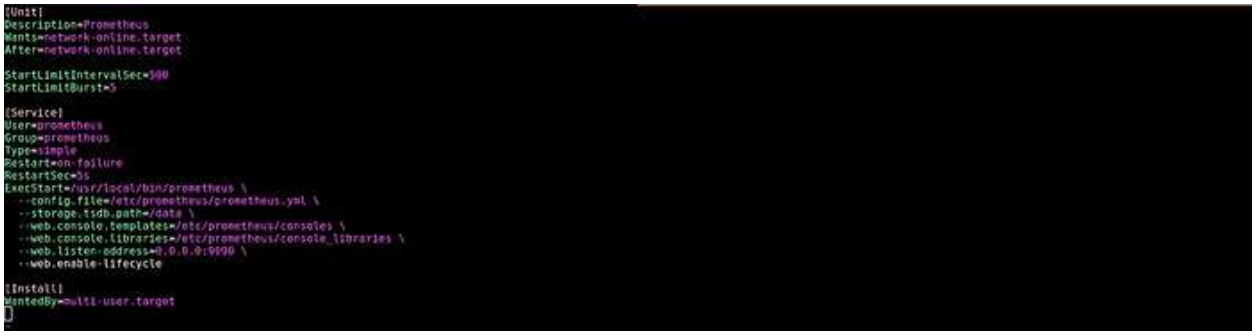
Create a systemd configuration file for prometheus

Edit the file /etc/systemd/system/prometheus.service

```
sudo vim /etc/systemd/system/prometheus.service
```

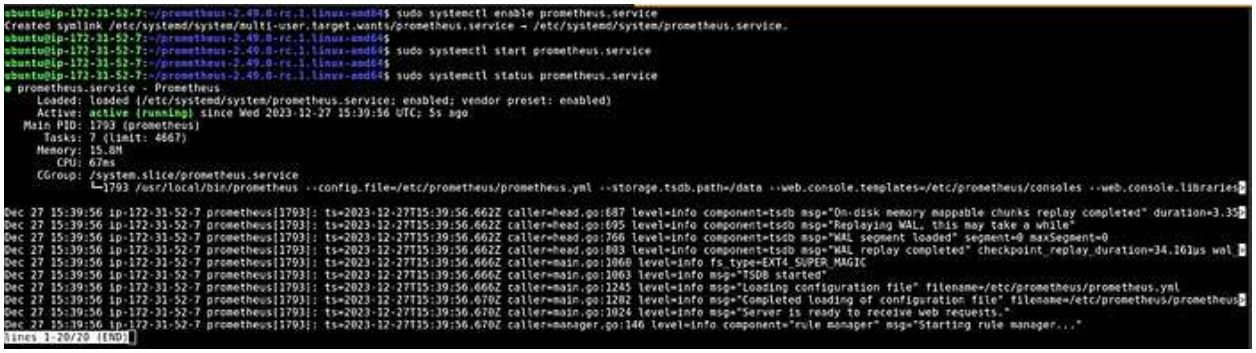
and paste the below configurations in your prometheus.service configuration file and save it

```
[Unit]
Description=Prometheus
Wants=network-online.target
After=network-online.target
StartLimitIntervalSec=500
StartLimitBurst=5
[Service]
User=prometheus
Group=prometheus
Type=simple
Restart=on-failure
RestartSec=5s
ExecStart=/usr/local/bin/prometheus \
-config.file=/etc/prometheus/prometheus.yml \
-storage.tsdb.path=/data \
-web.console.templates=/etc/prometheus/consoles \
-web.console.libraries=/etc/prometheus/console_libraries \
-web.listen-address=0.0.0.0:9090 \
-web.enable-lifecycle
[Install]
WantedBy=multi-user.target
```



Once you write the systemd configuration file for Prometheus, then enable it and start the Prometheus service.

```
sudo systemctl enable prometheus.service
sudo systemctl start prometheus.service
systemctl status prometheus.service
```



Once the Prometheus service is up and running then, copy the public IP of your Monitoring Server and paste it into your favorite browser with a 9090 port.

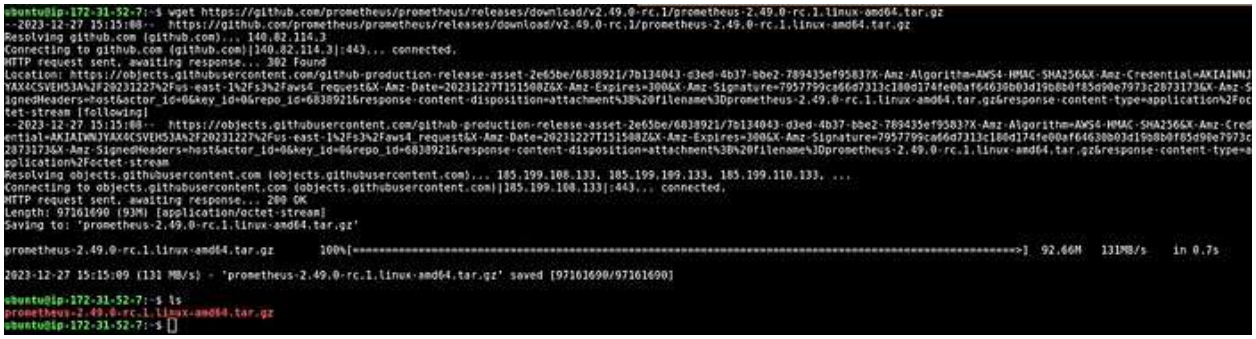
Now, we have to install a node exporter to visualize the machine or hardware level data such as CPU, RAM, etc on our Grafana dashboard.

To do that, we have to create a user for it.

```
sudo useradd \
-system \
-no-create-home \
-shell /bin/false node_exporter
```

Download the node exporter package

```
wget
https://github.com/prometheus/node_exporter/releases/download/v1.7.0/node_exporter-1.7.
0.linux-amd64.tar.gz
```



Untar the node exporter package file and move the node_exporter directory to the /usr/local/bin directory

```
tar -xvf node_exporter-1.7.0.linux-amd64.tar.gz
sudo mv node_exporter-1.7.0.linux-amd64/node_exporter /usr/local/bin/
```

Validate the version of the node exporter

```
node_exporter --version
```

Create the systemd configuration file for node exporter.

Edit the file

```
sudo vim /etc/systemd/system/node_exporter.service
```

Copy the below configurations and paste them into the /etc/systemd/system/node_exporter.service file.

```
[Unit]
Description=Node Exporter
Wants=network-online.target
After=network-online.target
StartLimitIntervalSec=500
StartLimitBurst=5
[Service]
User=node_exporter
Group=node_exporter
Type=simple
Restart=on-failure
RestartSec=5s
ExecStart=/usr/local/bin/node_exporter \
-collector.logind
[Install]
WantedBy=multi-user.target
```

```
[Unit]
Description=Node Exporter
Wants=network-online.target
After=network-online.target

StartLimitIntervalSec=500
StartLimitBurst=5

[Service]
User=node_exporter
Group=node_exporter
Type=simple
Restart=on-failure
RestartSec=5s
ExecStart=/usr/local/bin/node_exporter \
    --collector.logind

[Install]
WantedBy=multi-user.target
```

Enable the node exporter systemd configuration file and start it.

```
sudo systemctl enable node_export
node_exporter systemctl status node_
```

```
ubuntu@ip-172-31-52-7:~$ sudo systemctl enable node_exporter
Created symlink /etc/systemd/system/multi-user.target.wants/node_exporter.service → /etc/systemd/system/node_exporter.service.
ubuntu@ip-172-31-52-7:~$
ubuntu@ip-172-31-52-7:~$ sudo systemctl start node_exporter.service
ubuntu@ip-172-31-52-7:~$
ubuntu@ip-172-31-52-7:~$ systemctl status node_exporter.service
● node_exporter.service - Node Exporter
     Loaded: loaded (/etc/systemd/system/node_exporter.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2023-12-27 15:53:27 UTC; 10s ago
   Main PID: 1987 (node_exporter)
      Tasks: 5 (limit: 4667)
     Memory: 2.4M
        CPU: 9ms
     CGroup: /system.slice/node_exporter.service
             └─1987 /usr/local/bin/node_exporter --collector.logind

Dec 27 15:53:27 ip-172-31-52-7 node_exporter[1987]: ts=2023-12-27T15:53:27.699Z caller=node_exporter.go:117 level=info collector=thermal_zone
Dec 27 15:53:27 ip-172-31-52-7 node_exporter[1987]: ts=2023-12-27T15:53:27.699Z caller=node_exporter.go:117 level=info collector=time
Dec 27 15:53:27 ip-172-31-52-7 node_exporter[1987]: ts=2023-12-27T15:53:27.699Z caller=node_exporter.go:117 level=info collector=timex
Dec 27 15:53:27 ip-172-31-52-7 node_exporter[1987]: ts=2023-12-27T15:53:27.699Z caller=node_exporter.go:117 level=info collector=udp_queues
Dec 27 15:53:27 ip-172-31-52-7 node_exporter[1987]: ts=2023-12-27T15:53:27.699Z caller=node_exporter.go:117 level=info collector=uname
Dec 27 15:53:27 ip-172-31-52-7 node_exporter[1987]: ts=2023-12-27T15:53:27.699Z caller=node_exporter.go:117 level=info collector=vmstat
Dec 27 15:53:27 ip-172-31-52-7 node_exporter[1987]: ts=2023-12-27T15:53:27.699Z caller=node_exporter.go:117 level=info collector=xfs
Dec 27 15:53:27 ip-172-31-52-7 node_exporter[1987]: ts=2023-12-27T15:53:27.700Z caller=node_exporter.go:117 level=info collector=zfs
Dec 27 15:53:27 ip-172-31-52-7 node_exporter[1987]: ts=2023-12-27T15:53:27.700Z caller=tls_config.go:274 level=info msg="Listening on" address=[::]:9100
Dec 27 15:53:27 ip-172-31-52-7 node_exporter[1987]: ts=2023-12-27T15:53:27.700Z caller=tls_config.go:277 level=info msg="TLS is disabled." http2=false address=[::]:9100
ubuntu@ip-172-31-52-7:~$
```

Now, we have to add a node exporter to our Prometheus target section. So, we will be able to monitor our server.

edit the file

```
sudo vim /etc/prometheus/prometheus.yml
```

Copy the content in the file

```
-job_name: "node_exporter"
  static_configs:
    -targets: ["localhost:9100"]
```

```
# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
        - targets:
          # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: "prometheus"

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ["localhost:9090"]

  - job_name: "node_exporter"
    static_configs:
      - targets: ["localhost:9100"]
```

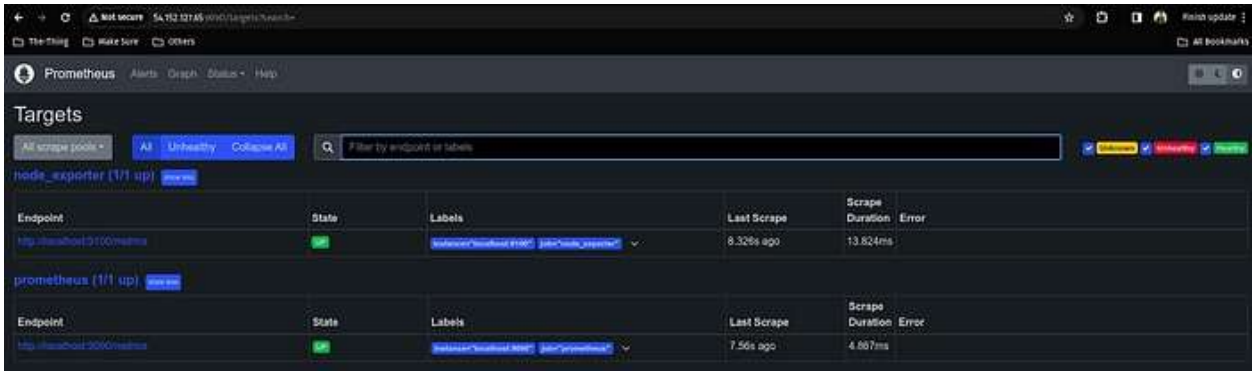
After saving the file, validate the changes that you have made using promtool.

```
promtool check config /etc/prometheus/prometheus.yml
```

If your changes have been validated then, push the changes to the Prometheus server.

```
curl -X POST http://localhost:9090/-/reload
```

Now, go to your Prometheus server and this time, you will see one more target section as node_exporter which should be up and running.



Now, install the Grafana tool to visualize all the data that is coming with the help of Prometheus.

```
sudo apt-get install -y apt-transport-https software-properties-common wget
sudo mkdir -p /etc/apt/keyrings/
wget -q -O - https://apt.grafana.com/gpg.key | gpg - dearmor | sudo tee
/etc/apt/keyrings/grafana.gpg > /dev/null
echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com stable main" |
sudo tee -a /etc/apt/sources.list.d/grafana.list
echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com beta main" |
sudo tee -a /etc/apt/sources.list.d/grafana.list
sudo apt-get update
```

```
ubuntu@ip-172-31-52-7:~$ sudo apt-get install -y apt-transport-https software-properties-common wget
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'apt' instead of 'apt-transport-https'
wget is already the newest version (1.21.2-2ubuntu1).
wget set to manually installed.
apt is already the newest version (2.4.11).
apt set to manually installed.
software-properties-common is already the newest version (0.99.22.8).
software-properties-common set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
ubuntu@ip-172-31-52-7:~$
ubuntu@ip-172-31-52-7:~$ sudo mkdir -p /etc/apt/keyrings/
wget -q -O - https://apt.grafana.com/gpg.key | gpg --dearmor | sudo tee /etc/apt/keyrings/grafana.gpg > /dev/null
ubuntu@ip-172-31-52-7:~$
ubuntu@ip-172-31-52-7:~$
ubuntu@ip-172-31-52-7:~$ echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com stable main" | sudo tee -a /etc/apt/sources.list.d/grafana.list
deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com stable main
ubuntu@ip-172-31-52-7:~$
ubuntu@ip-172-31-52-7:~$
ubuntu@ip-172-31-52-7:~$ echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com beta main" | sudo tee -a /etc/apt/sources.list.d/grafana.list
deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com beta main
ubuntu@ip-172-31-52-7:~$
ubuntu@ip-172-31-52-7:~$
ubuntu@ip-172-31-52-7:~$ # Updates the list of available packages
sudo apt-get update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:5 https://apt.grafana.com stable InRelease [5984 B]
Get:6 https://apt.grafana.com beta InRelease [5976 B]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [14.1 MB]
Get:8 https://apt.grafana.com stable/main amd64 Packages [104 kB]
Err:6 https://apt.grafana.com beta InRelease
```

Install the Grafana
sudo apt-get install grafana

```
ubuntu@ip-172-31-52-7:~$ # Installs the latest OSS release:
sudo apt-get install grafana
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  fontconfig-config fonts-dejavu-core libfontconfig1 musl
The following NEW packages will be installed:
  fontconfig-config fonts-dejavu-core grafana libfontconfig1 musl
0 upgraded, 5 newly installed, 0 to remove and 26 not upgraded.
Need to get 105 MB of archives.
After this operation, 388 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 fonts-dejavu-core all 2.37-2build1 [1041 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 fontconfig-config all 2.13.1-4.2ubuntu5 [29.1 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 libfontconfig1 amd64 2.13.1-4.2ubuntu5 [131 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 musl amd64 1.2.2-4 [407 kB]
Get:5 https://apt.grafana.com stable/main amd64 grafana amd64 10.2.3 [104 MB]
Fetched 105 MB in 3s (36.6 MB/s)
Selecting previously unselected package fonts-dejavu-core.
(Reading database ... 64799 files and directories currently installed.)
```

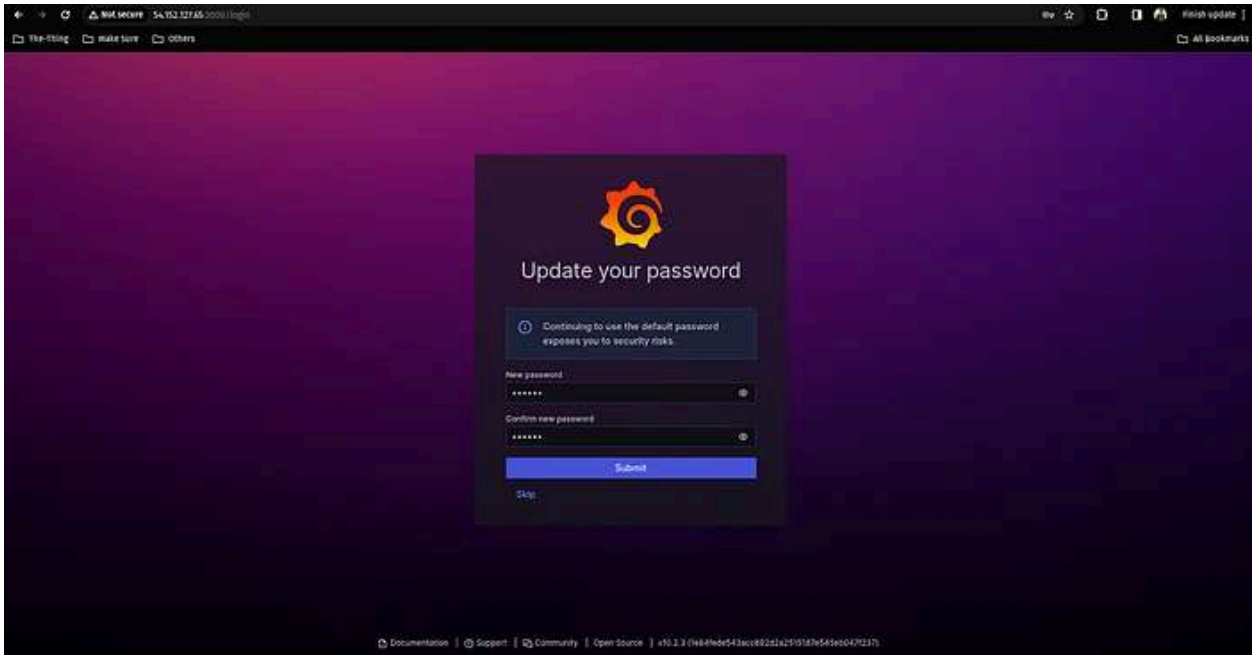
Enable and start the Grafana Service
sudo systemctl enable grafana-server.service
sudo systemctl start grafana-server.service
sudo systemctl status grafana-server.service

```
ubuntu@ip-172-31-52-7:~$ sudo systemctl enable grafana-server.service
Synchronizing state of grafana-server.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable grafana-server
Created symlink /etc/systemd/system/multi-user.target.wants/grafana-server.service → /lib/systemd/system/grafana-server.service.
ubuntu@ip-172-31-52-7:~$
ubuntu@ip-172-31-52-7:~$ sudo systemctl start grafana-server.service
ubuntu@ip-172-31-52-7:~$
ubuntu@ip-172-31-52-7:~$ sudo systemctl status grafana-server.service
● grafana-server.service - Grafana instance
     Loaded: loaded (/lib/systemd/system/grafana-server.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2023-12-27 16:07:44 UTC; 5s ago
       Docs: http://docs.grafana.org
   Main PID: 2834 (grafana)
      Tasks: 6 (limit: 4667)
     Memory: 37.6M
        CPU: 1.772s
     CGroup: /system.slice/grafana-server.service
             └─2834 /usr/share/grafana/bin/grafana server --config=/etc/grafana/grafana.ini --pidfile=/run/grafana/grafana-server.pid --packaging=deb cfg:default.paths.logs=/var/log/grafana

Dec 27 16:07:50 ip-172-31-52-7 grafana[2834]: logger=migrator t=2023-12-27T16:07:50.0509517952 level=info msg="Migration successfully executed" id="Update uid column values in playlist" dur
Dec 27 16:07:50 ip-172-31-52-7 grafana[2834]: logger=migrator t=2023-12-27T16:07:50.0572505192 level=info msg="Executing migration" id="Add index for uid in playlist"
Dec 27 16:07:50 ip-172-31-52-7 grafana[2834]: logger=migrator t=2023-12-27T16:07:50.0584215382 level=info msg="Migration successfully executed" id="Add index for uid in playlist" duration=1
Dec 27 16:07:50 ip-172-31-52-7 grafana[2834]: logger=migrator t=2023-12-27T16:07:50.0647844822 level=info msg="Executing migration" id="update group index for alert rules"
Dec 27 16:07:50 ip-172-31-52-7 grafana[2834]: logger=migrator t=2023-12-27T16:07:50.0653048352 level=info msg="Migration successfully executed" id="update group index for alert rules" durat
Dec 27 16:07:50 ip-172-31-52-7 grafana[2834]: logger=migrator t=2023-12-27T16:07:50.0714219892 level=info msg="Executing migration" id="managed folder permissions alert actions repeated mig
Dec 27 16:07:50 ip-172-31-52-7 grafana[2834]: logger=migrator t=2023-12-27T16:07:50.0717521222 level=info msg="Migration successfully executed" id="managed folder permissions alert actions
Dec 27 16:07:50 ip-172-31-52-7 grafana[2834]: logger=migrator t=2023-12-27T16:07:50.0774581092 level=info msg="Executing migration" id="admin only folder/dashboard permission"
Dec 27 16:07:50 ip-172-31-52-7 grafana[2834]: logger=migrator t=2023-12-27T16:07:50.0779915732 level=info msg="Migration successfully executed" id="admin only folder/dashboard permission" d
Dec 27 16:07:50 ip-172-31-52-7 grafana[2834]: logger=migrator t=2023-12-27T16:07:50.0835639432 level=info msg="Executing migration" id="add action column to seed_assignment"
lines 1-21/21 (END)
```

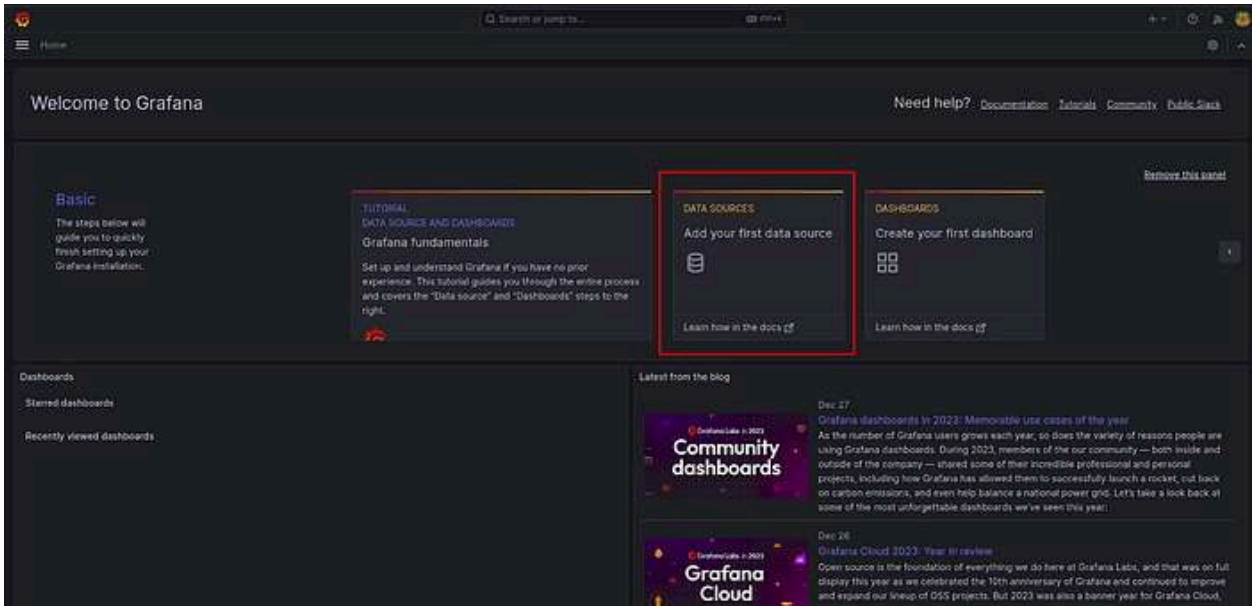
To access the Grafana dashboard, copy the public IP address of the Monitoring Server and paste it into your favorite browser with port 3000

username and password will be admin

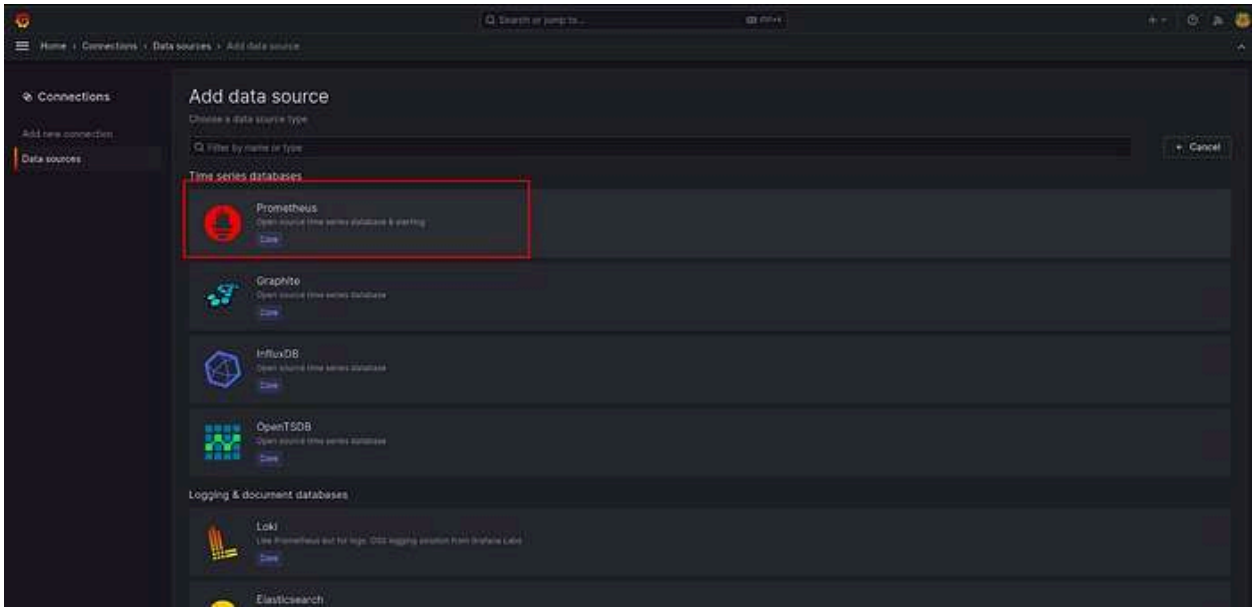
Reset the password



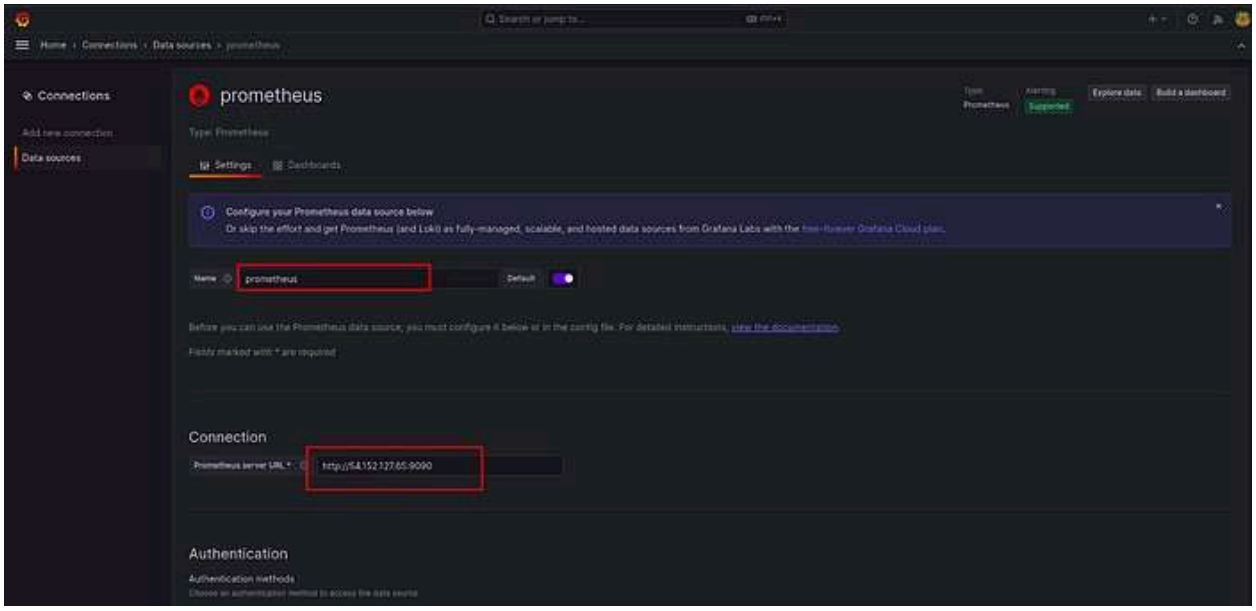
Click on Data sources



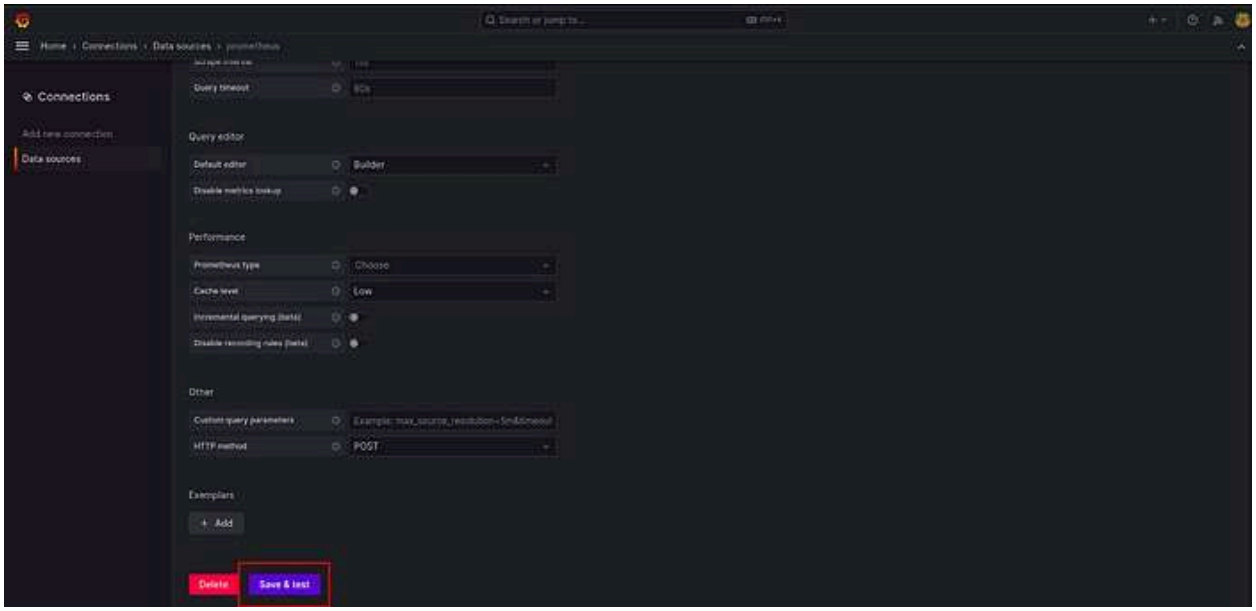
Select the Prometheus



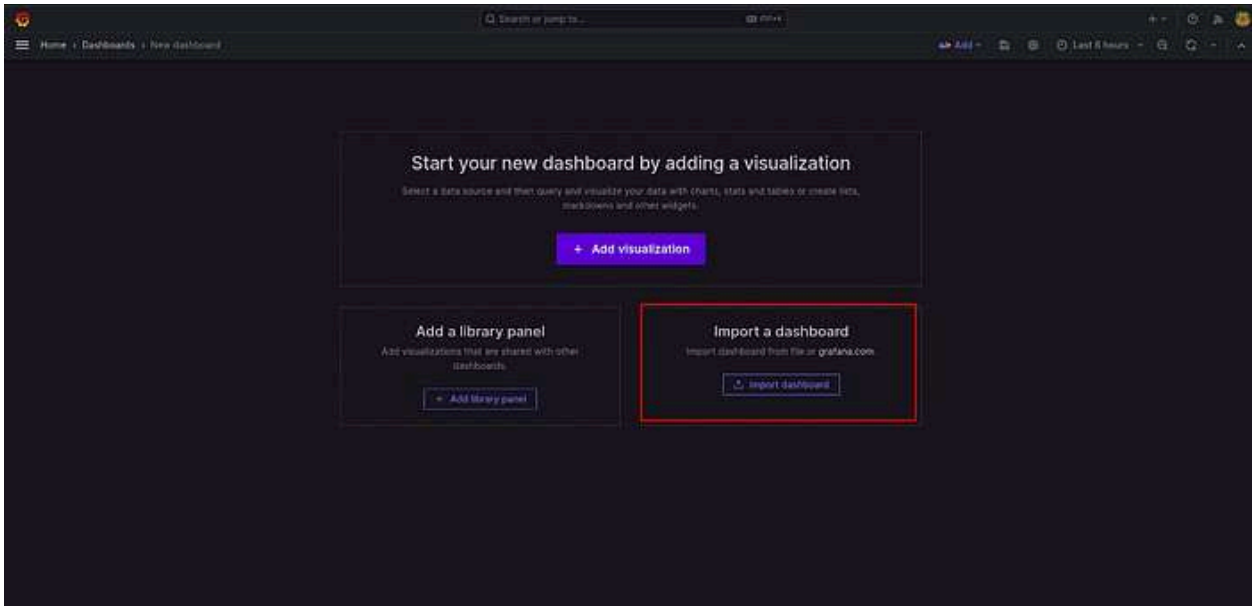
Provide the Monitoring Server Public IP with port 9090 to monitor the Monitoring Server.



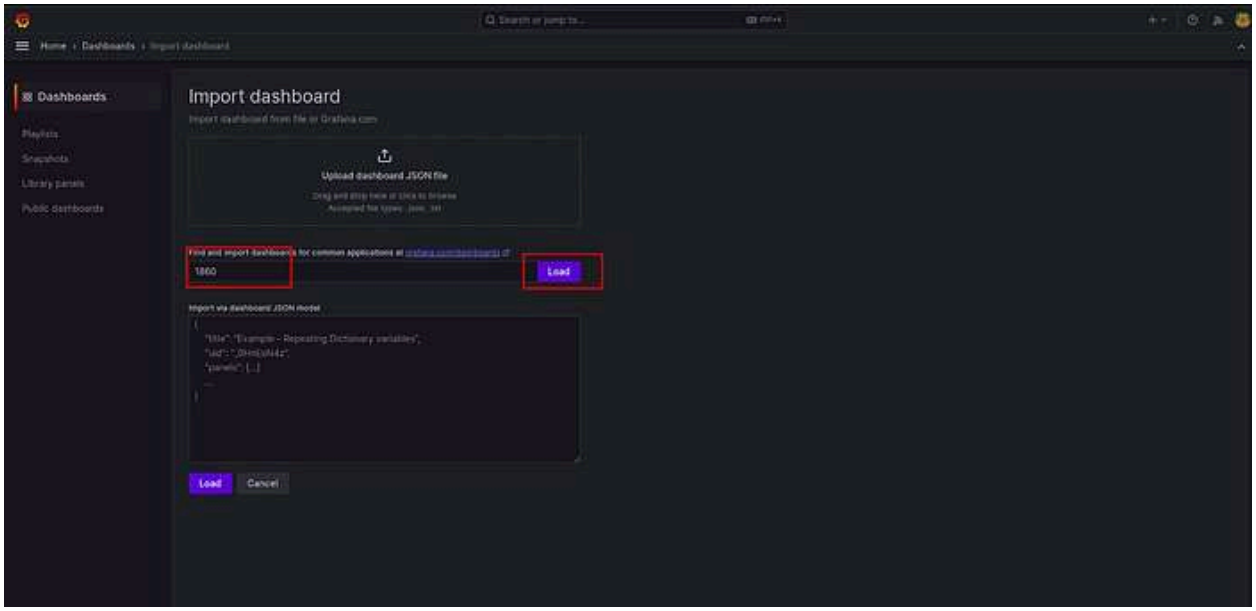
Click on Save and test.



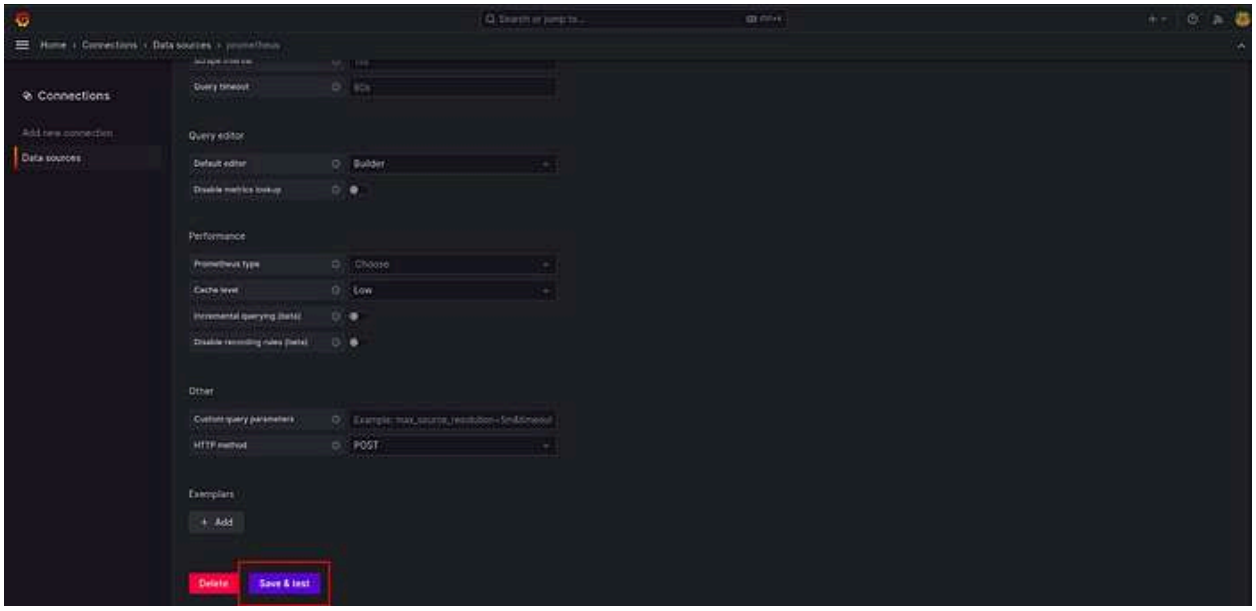
Go to the dashboard section of Grafana and click on the Import dashboard.



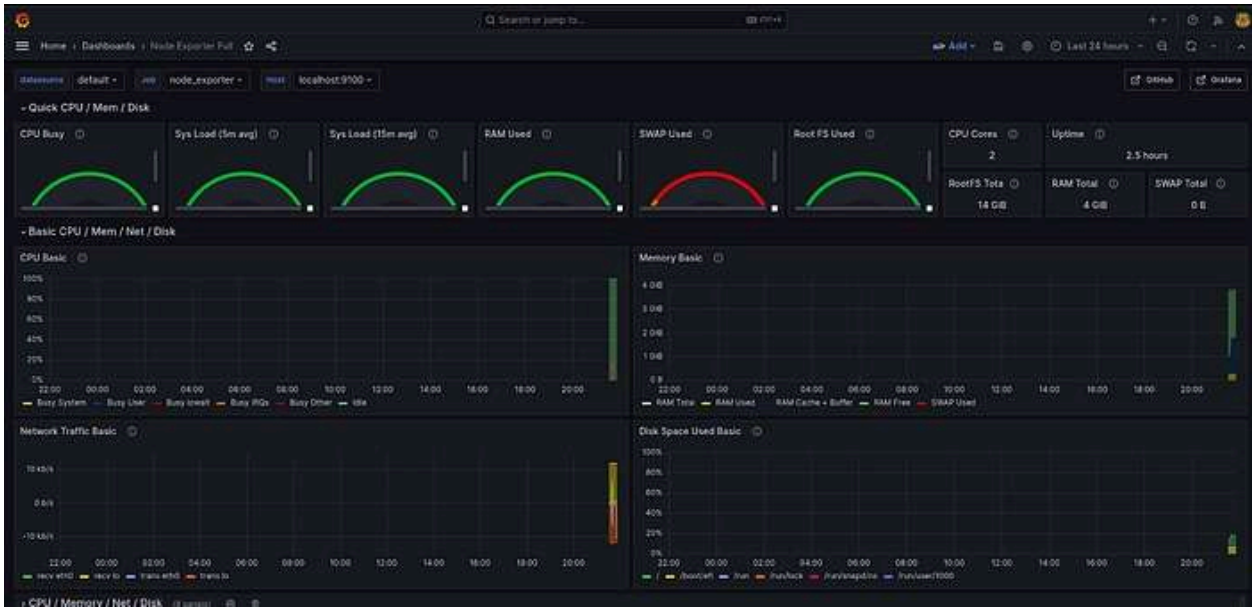
Add 1860 for the node exporter dashboard and click on Load.



Then, select the Prometheus from the drop down menu and click on Import



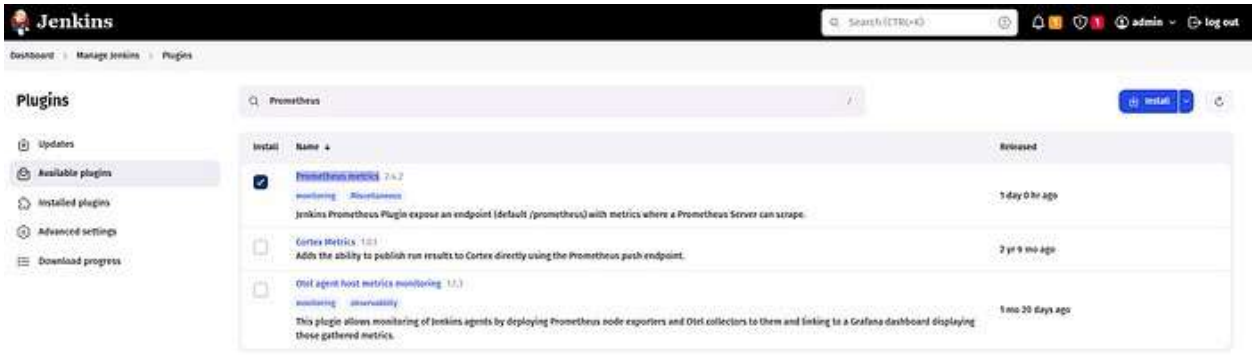
The dashboard will look like this



Now, we have to monitor our Jenkins Server as well.

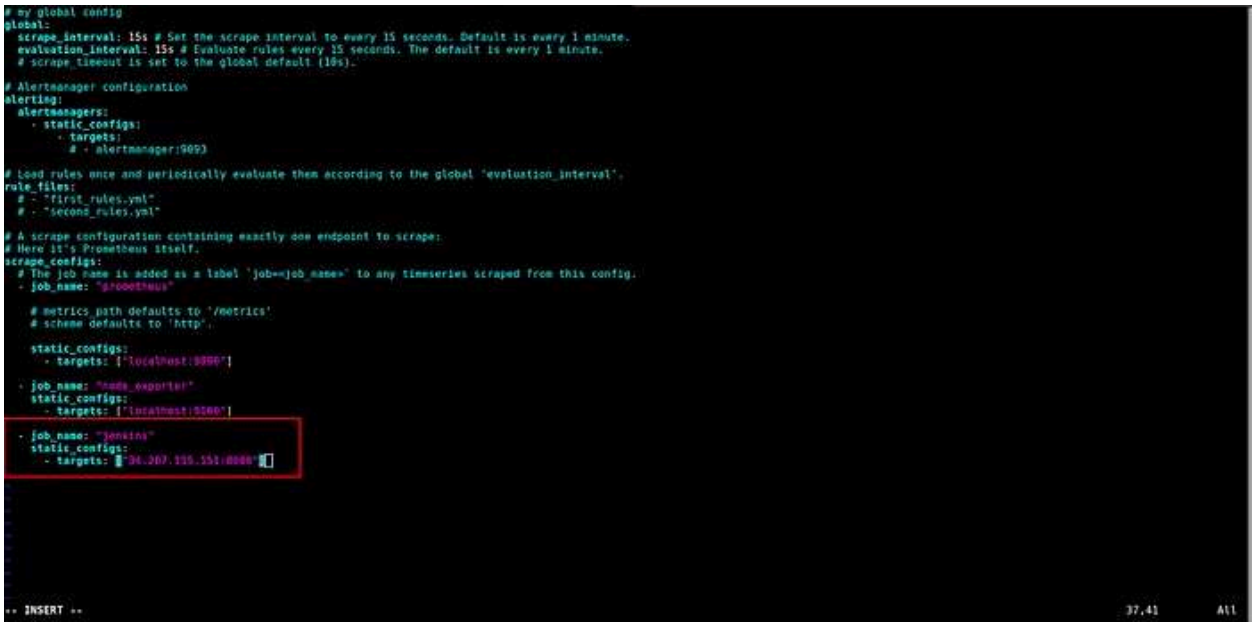
For that, we need to install the Prometheus metric plugin on our Jenkins.

Go to Manage Jenkins -> Plugin search for Prometheus metrics install it and restart your Jenkins.



Edit the /etc/prometheus/prometheus.yml file

```
sudo vim /etc/prometheus/prometheus.yml
-job_name: "jenkins"
   static_configs:
     -targets: ["<jenkins-server-public-ip>:8080"]
```



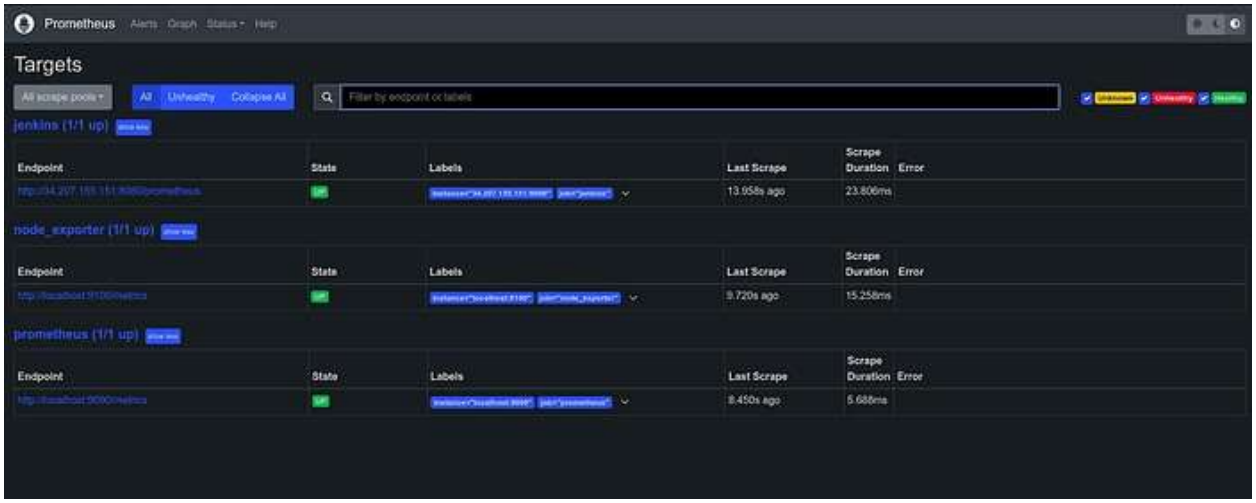
Once you add the Jenkins job, validate the Prometheus config file whether it is correct or not by running the below command.

```
promtool check config /etc/prometheus/prometheus.yml
```

Now, push the new changes on the Prometheus server

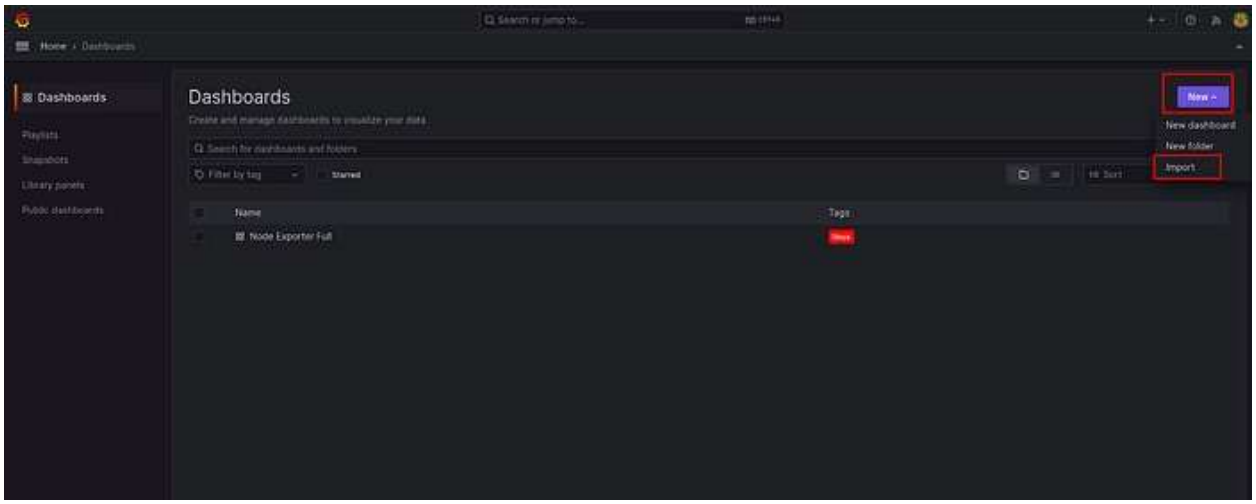
```
curl -X POST http://localhost:9090/-/reload
```

Copy the public IP of your Monitoring Server and paste on your favorite browser with a 9090 port with /target. You will see the targets that you have added in the /etc/prometheus/prometheus.yml file.

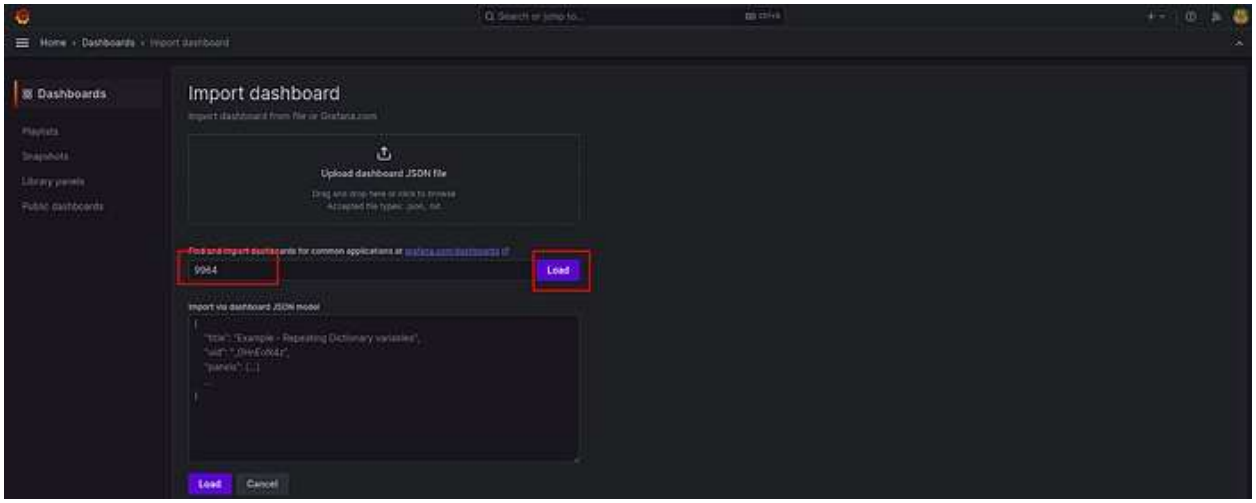


To add the Jenkins Dashboard on your Grafana server.

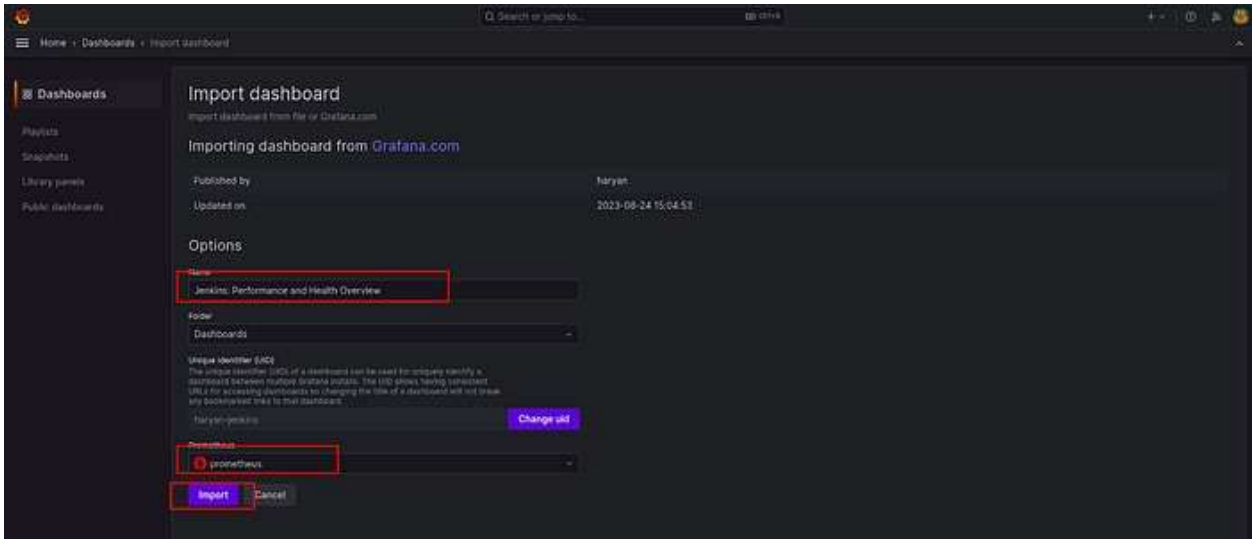
Click on New -> Import.



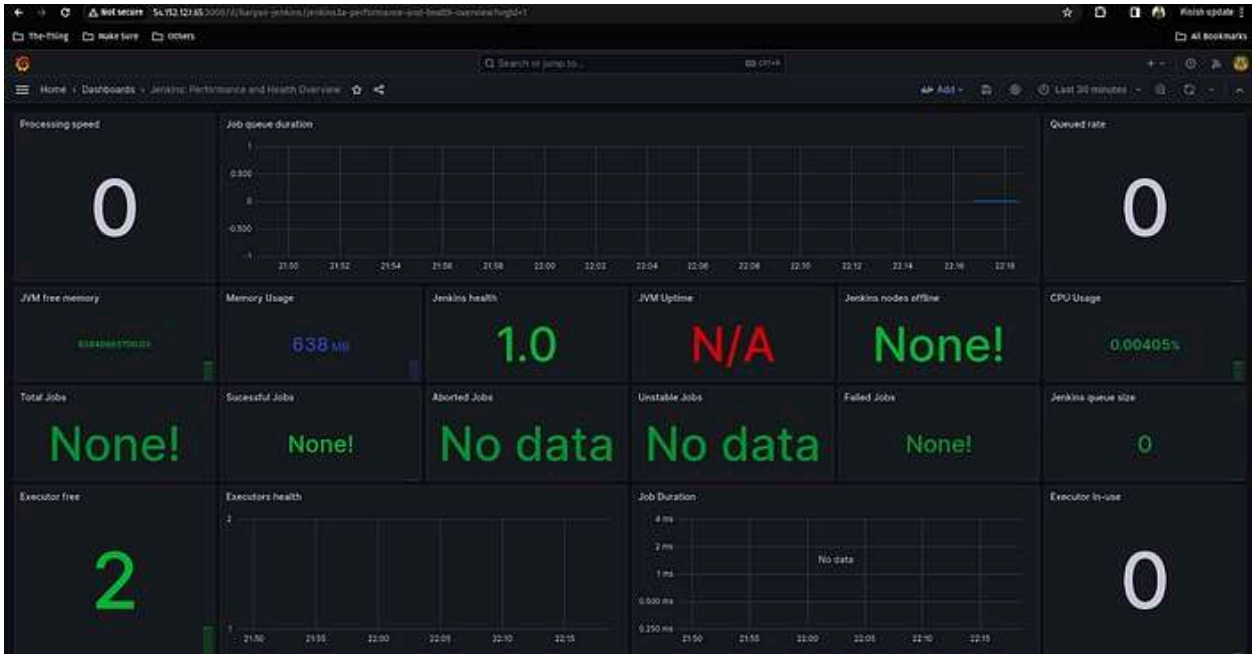
Provide the 9964 to Load the dashboard.



Select the default Prometheus from the drop-down menu and click on Import.



You will see your Jenkins Monitoring dashboard in the below snippet.

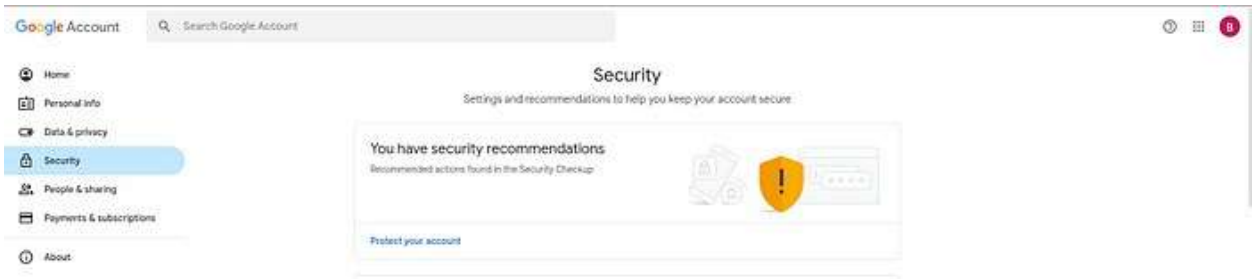


Now, we have to integrate Email Alert. So, if our Jenkins pipeline will succeed or fail we will get a notification alert on our email.

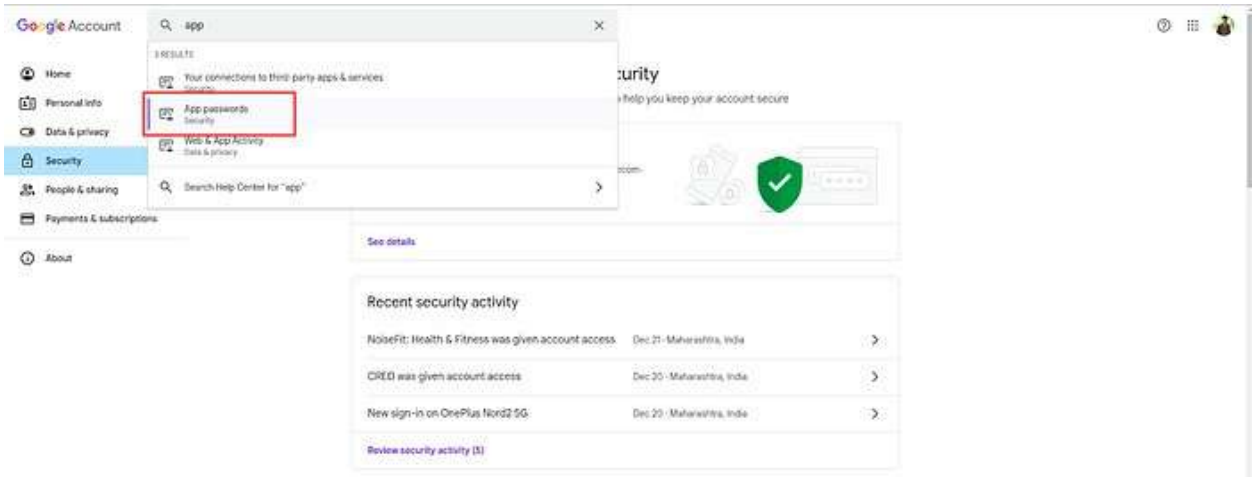
To do that, we need to install the Jenkins Plugin, whose name is Email Extension Template.

Manage Jenkins -> Plugins and install the Email Extension Template plugin.

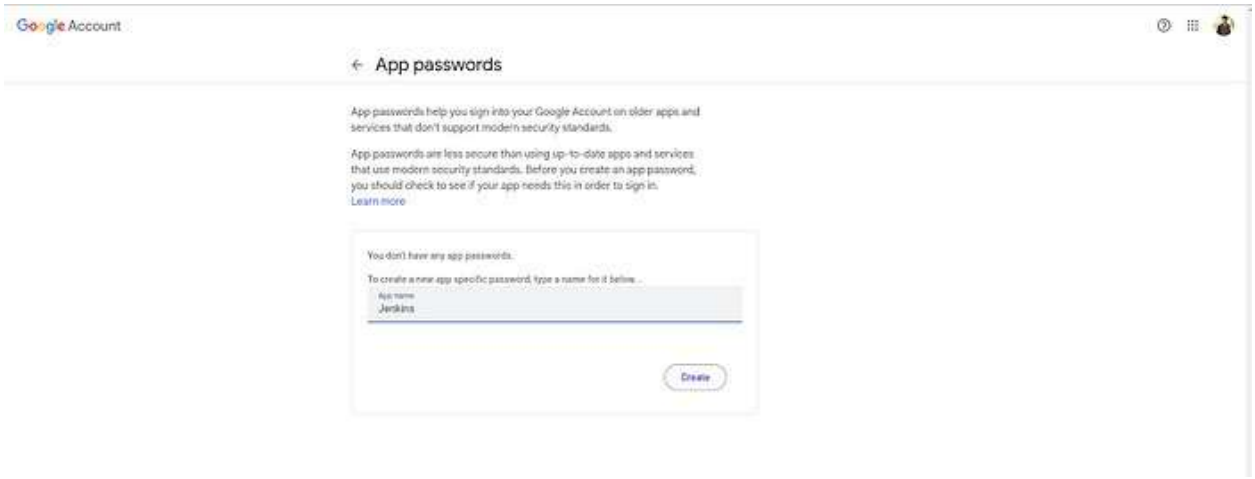
After installing the plugin, go to your email ID and click on Manage account and you will see what looks like the below snippet.



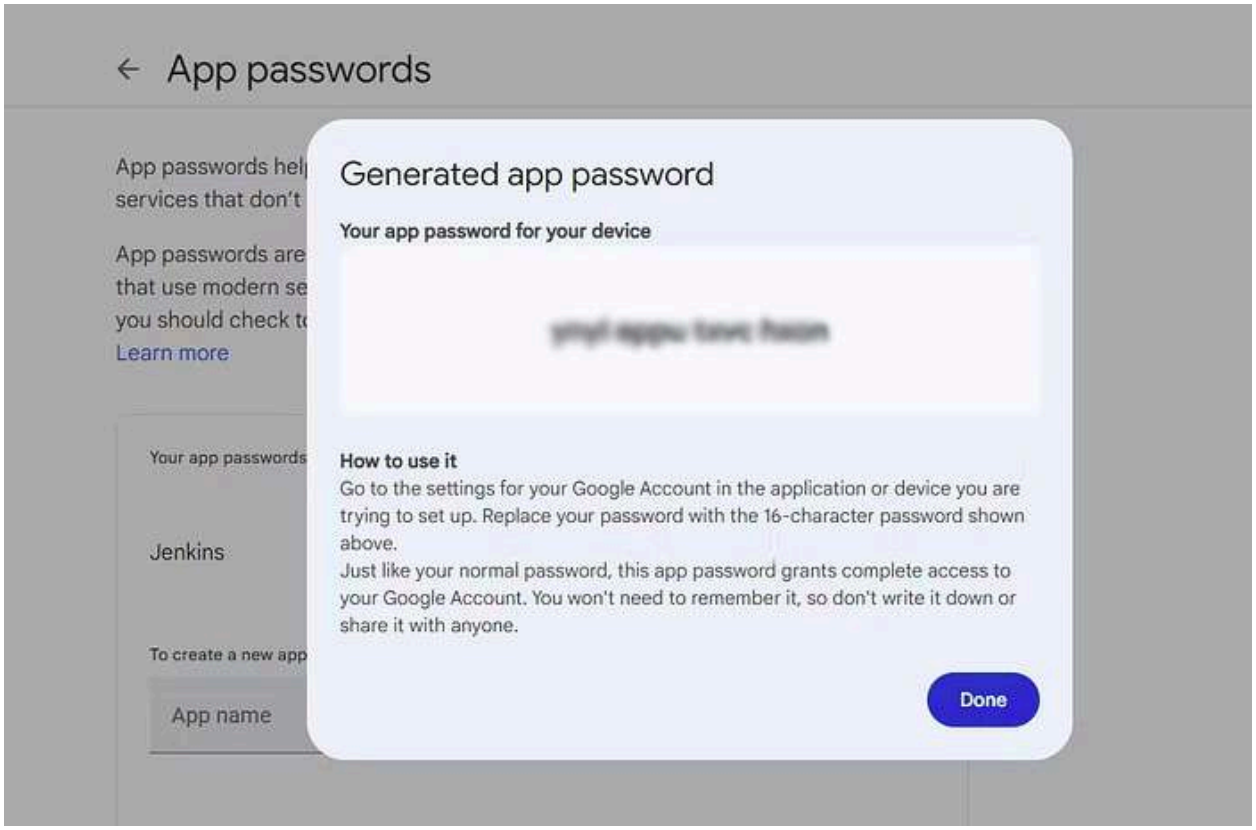
In the Security section, search for App passwords and click on it.



Gmail will prompt you for the password. Provide the password then you have to provide the name of your app where you are integrating email service.



You will get your password below. Copy the password and keep it secure somewhere.



Add your email ID and the password that you have generated in the previous step.

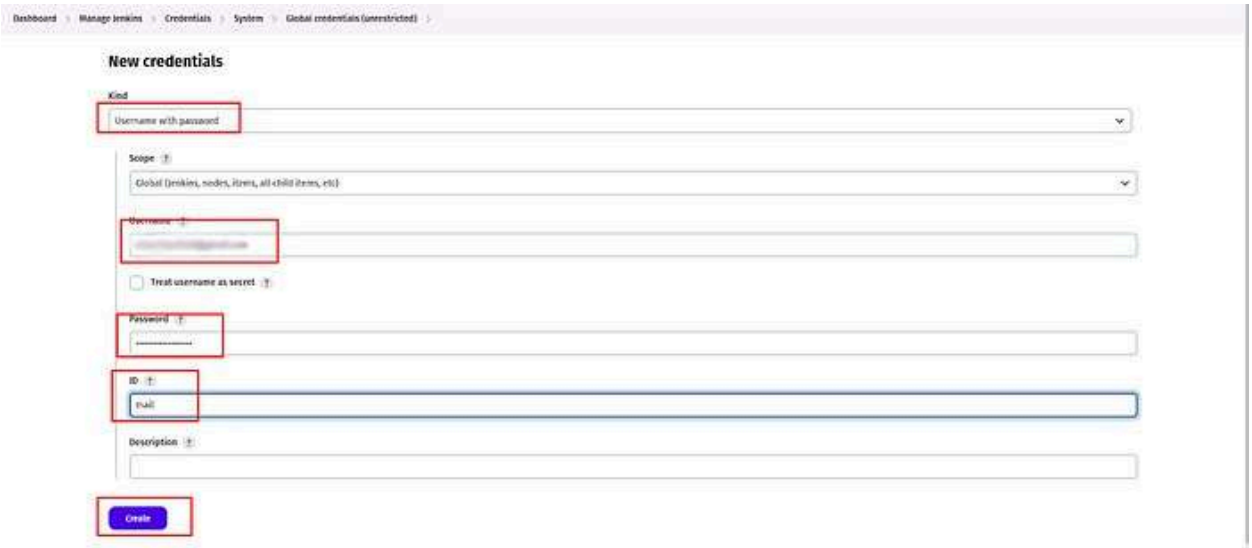
Go to Manage Jenkins -> Credentials.

Click on (global).

Click on Add credentials

Select the Username with password in Kind.

Provide your mail ID and generated password then provide the ID as mail to call both credentials.



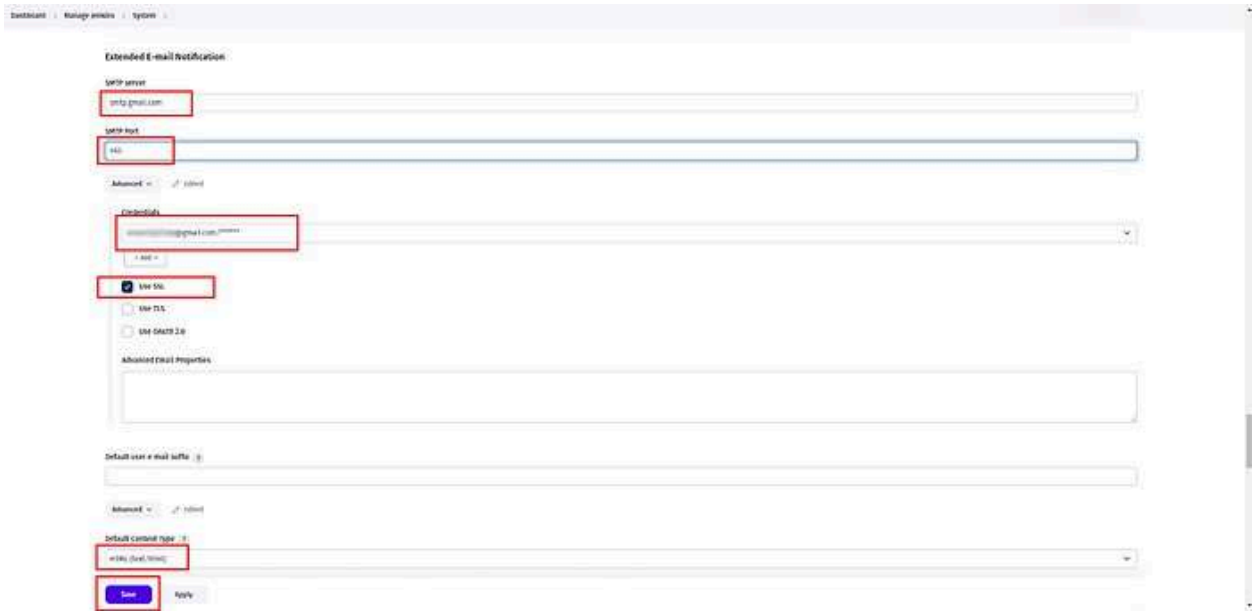
You can see we have added the credentials for the mail.

Now, we have to configure our mail for the alerts.

Go to Jenkins -> Manage Jenkins -> System

Search for Extend E-mail Notification.

Provide the smtp.gmail.com in the SMTP server and 465 in the SMTP port.



Then, On the same page Search for Extend E-mail Notification.

Provide the smtp.gmail.com in the SMTP server and 465 in the SMTP port.

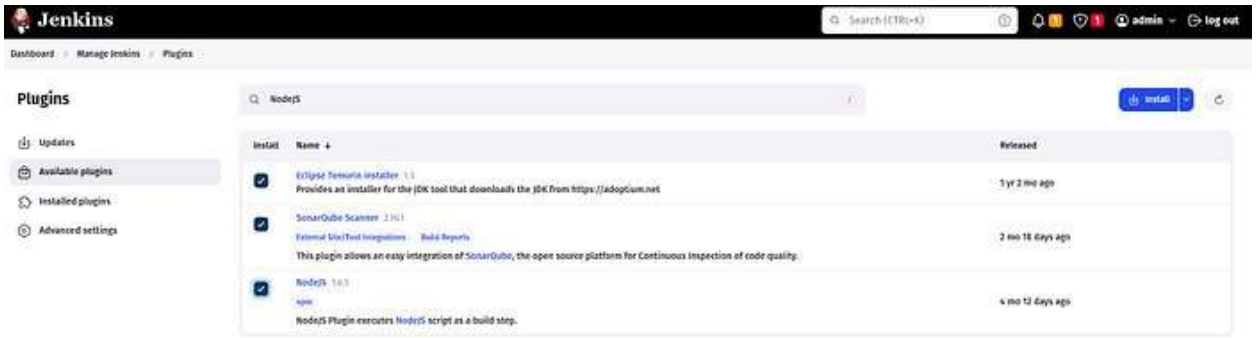
Select Use SMTP Authentication and provide the Gmail ID and its password in the Username and password.
To validate whether Jenkins can send the emails to you or not, check the Test configuration by sending a test e-mail.

You can see below for the reference.

Now, we will set up our Jenkins Pipeline. But there are some plugins required to work with them.

Download the following plugins

Eclipse Temurin installer
SonarQube Scanner
NodeJS



Now, configure the plugins

Go to Manage Jenkins -> Tools

Click on Add JDK and provide the following things below

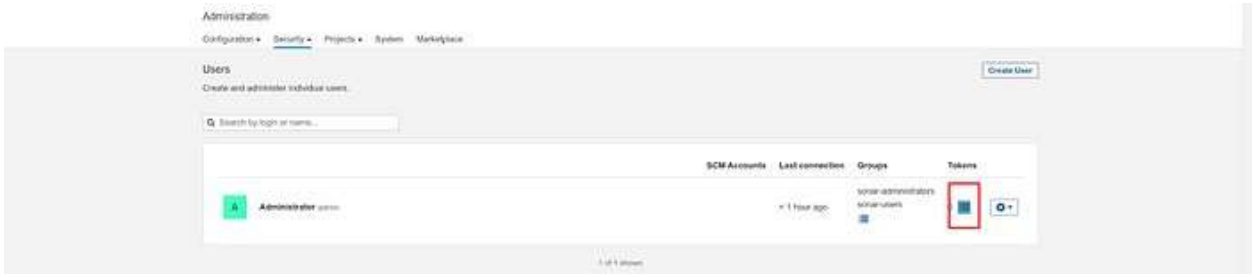
Click on Add NodeJS and provide the following things below

Now, we will configure Sonarqube

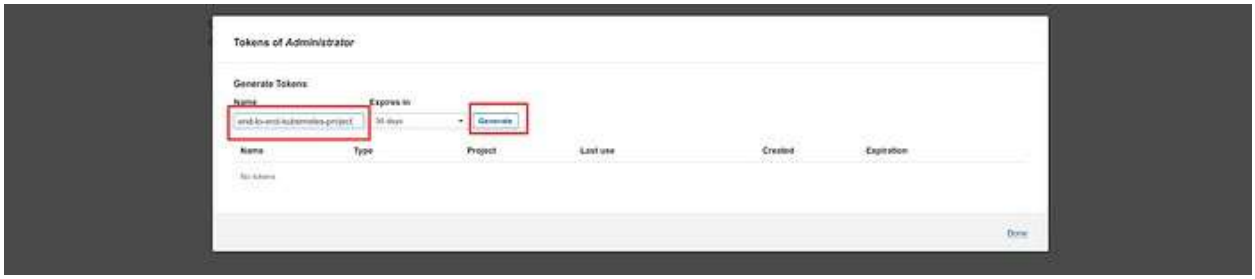
To access the sonarqube, copy the Jenkins Server public IP with port number 9000

Then, click Security and click on Users.

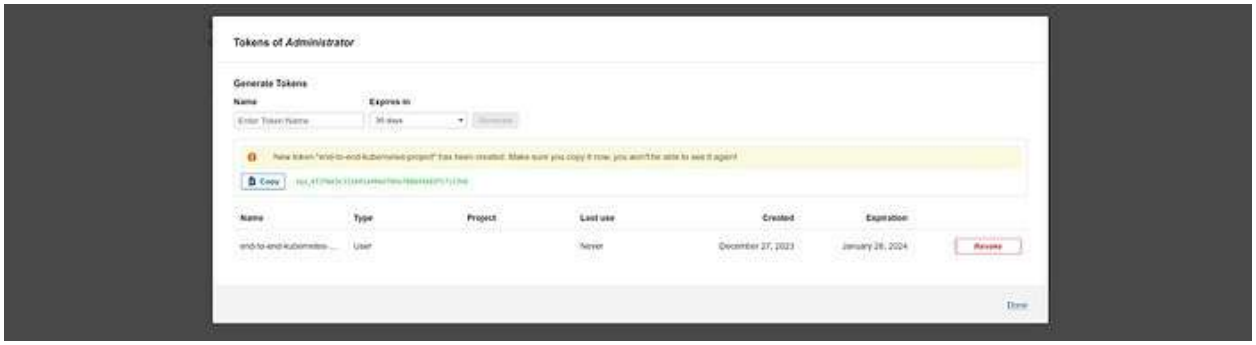
Click on the highlighted blue box on the right to generate the token.



Now provide the name of your token and click on Generate.



Copy the generated token and keep it somewhere.

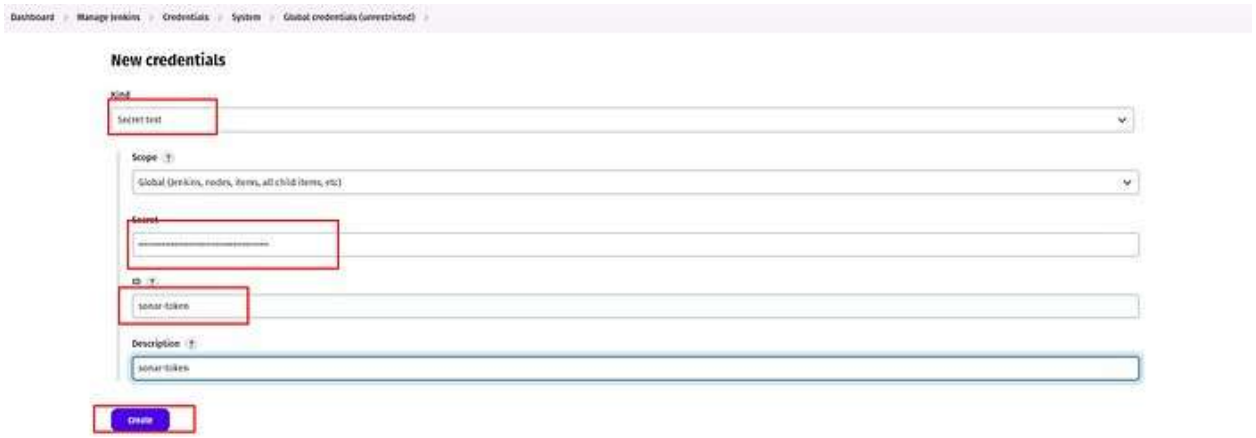


Now, add the token to your Jenkins credentials

Go to Manage Jenkins -> Credentials.

Select the Secret text in Kind.

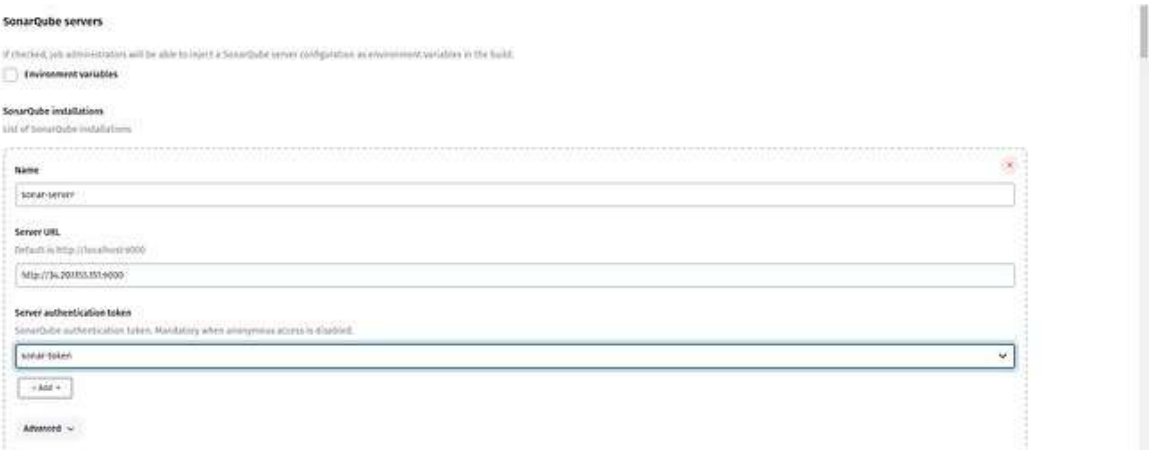
Provide your token then provide the ID as sonar-token to call the credentials.



Go to Manage Jenkins -> System

Click on Add Sonarqube

Provide the name sonar-server with the Server URL and select the credentials that we have added.



Go to Manage Jenkins -> Tools

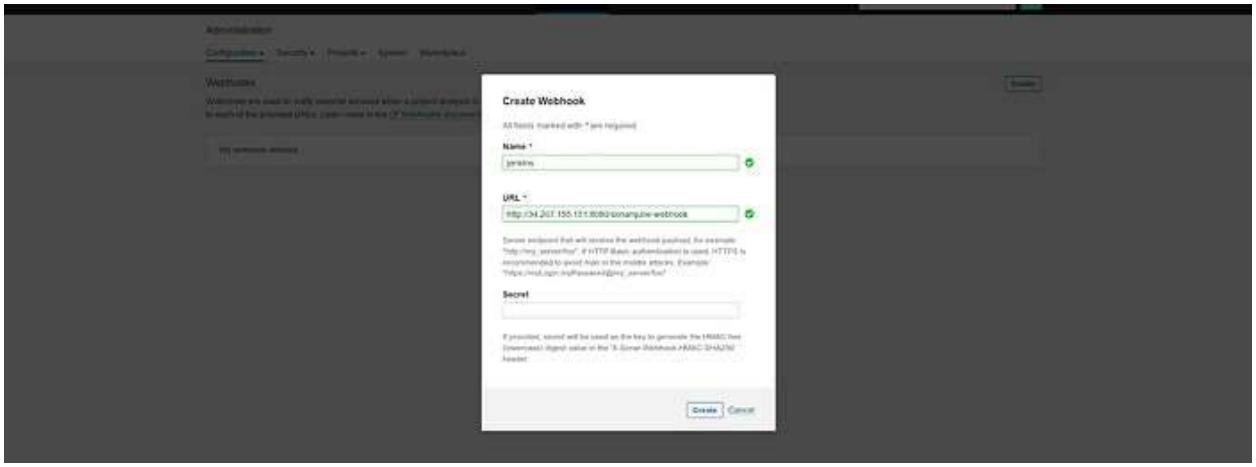
Find Sonarqube Scanner and click on Add

Provide the name sonar-server and select the latest version of Sonarqube.

To create a webhook, click on Configuration and select Webhooks.

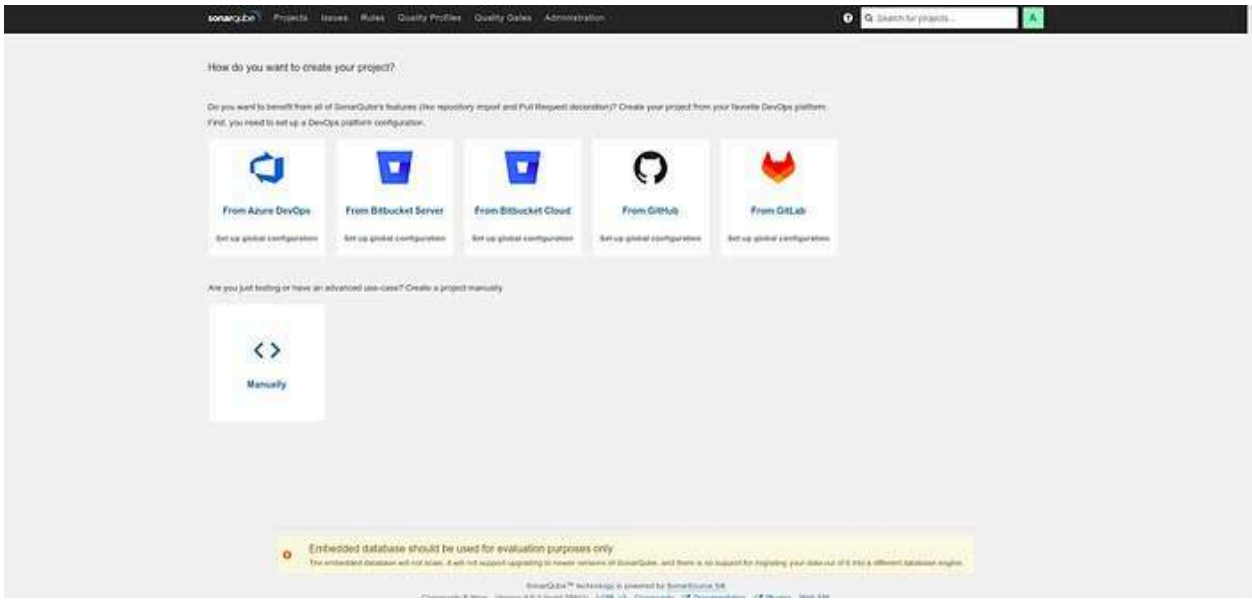
Click on Create.

Provide the name and Jenkins URL like below and click on Create.



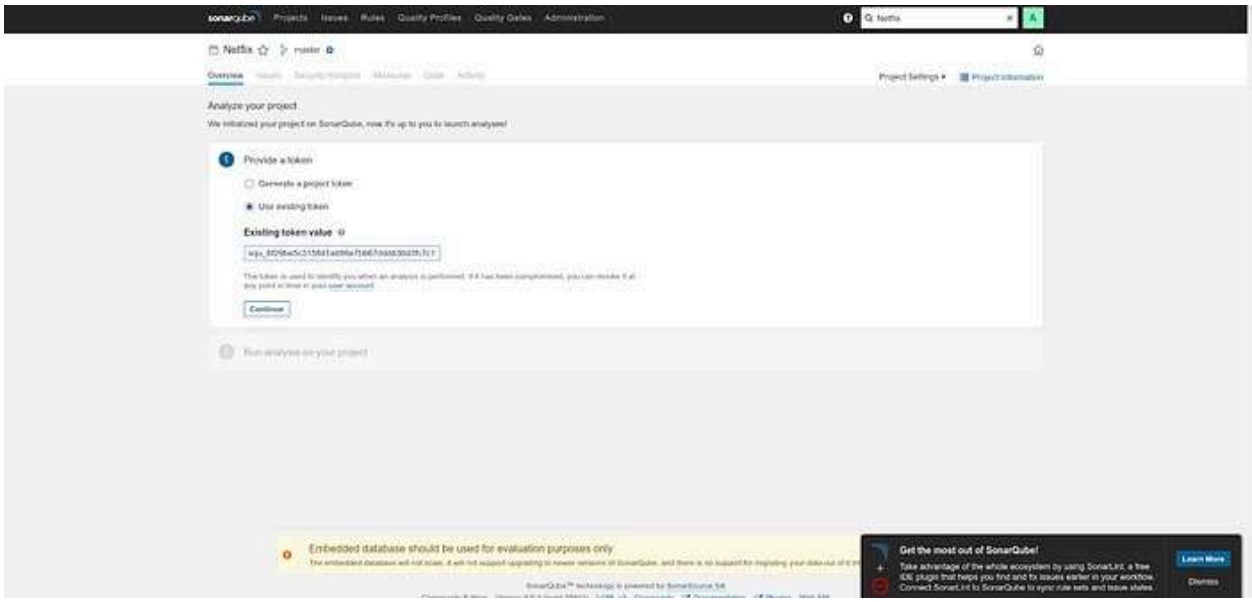
The Webhook will be showing the below snippet.

To create a project, click on Manually.

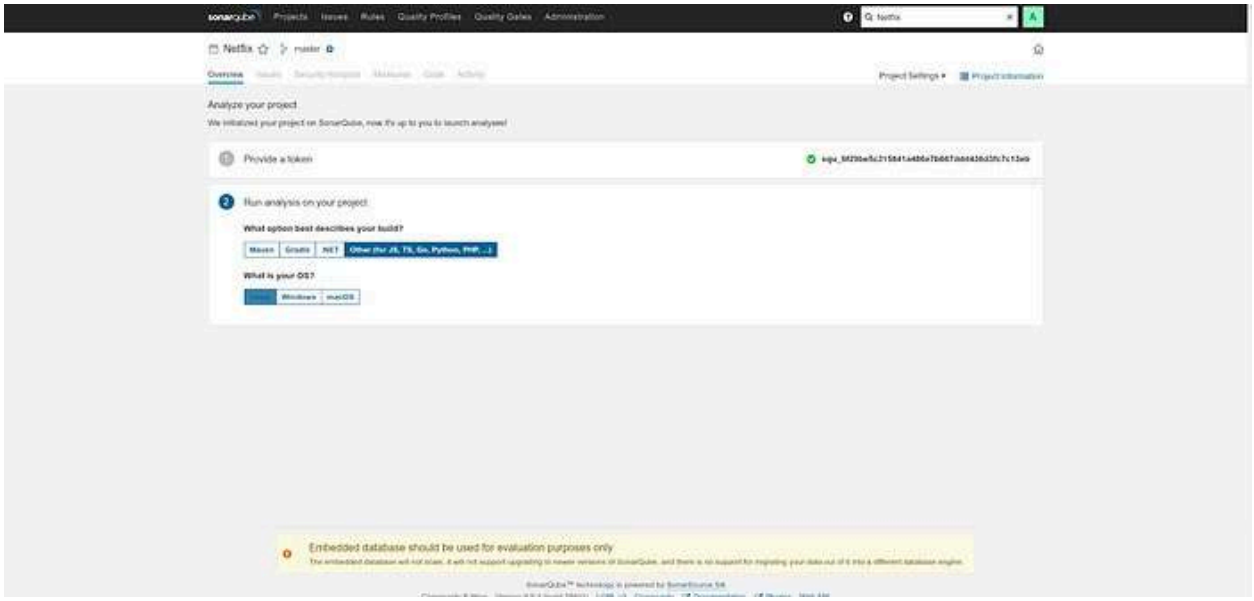


Provide the name of your project and click on Set up.

Select the existing token and click on continue.



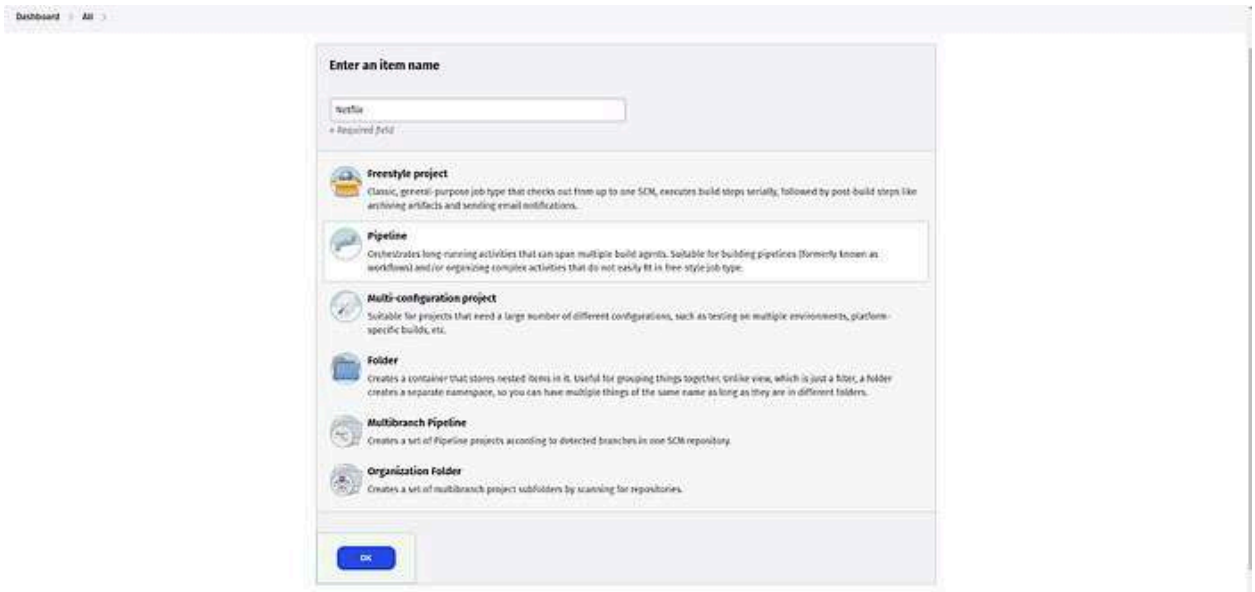
Select the Other as your build and Linux as OS.



Now, we will create the Jenkins Pipeline

Click on Create item.

Provide the name of your Jenkins Pipeline and select Pipeline.

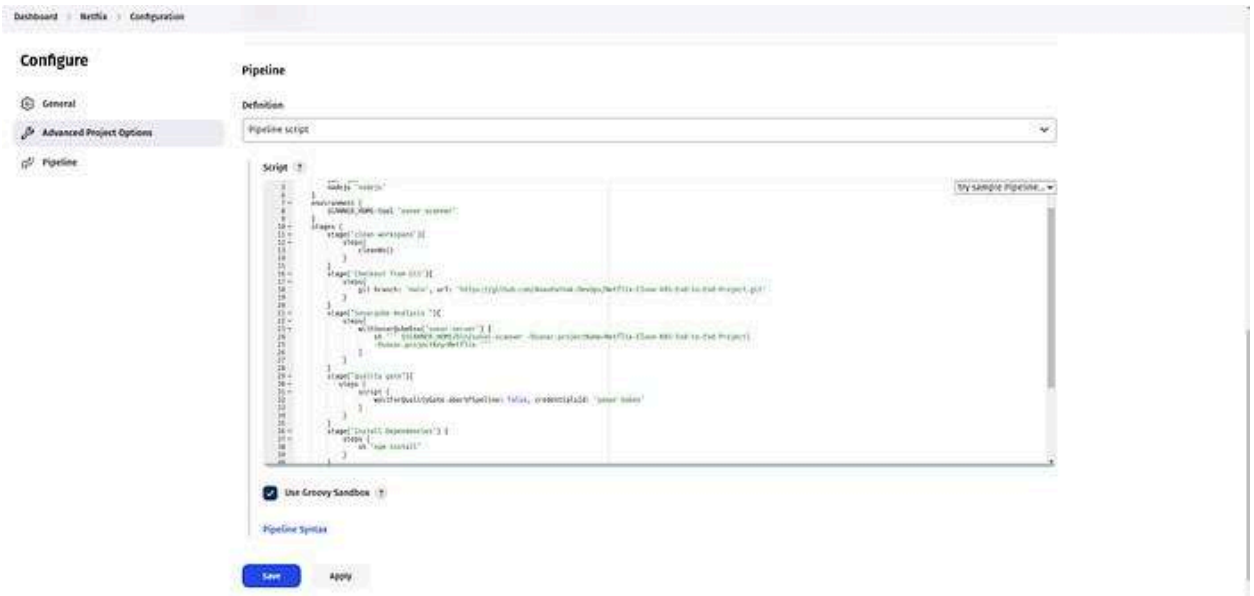


Currently, we are just creating a pipeline for Sonarqube analysis of the code, quality gate for Sonarqube, and installing the dependencies.

In the post-build, we have added email alerts for the success or failure of the pipeline.

```
pipeline{
    agent any
    tools{
        jdk 'jdk'
        nodejs 'nodejs'
    }
    environment {
        SCANNER_HOME=tool 'sonar-server'
    }
    stages {
        stage('Workspace Cleaning'){
            steps{
                cleanWs()
            }
        }
        stage('Checkout from Git'){
            steps{
                git branch: 'master', url:
'https://github.com/AmanPathak-DevOps/Netflix-Clone-K8S-End-to-End-Project.git'
            }
        }
        stage("Sonarqube Analysis"){
            steps{
                withSonarQubeEnv('sonar-server') {
                    sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=Netflix \
                    -Dsonar.projectKey=Netflix \
                    '''
                }
            }
        }
        stage("Quality Gate"){
            steps {
                script {
                    waitForQualityGate abortPipeline: false, credentialsId: 'sonar-token'
                }
            }
        }
        stage('Install Dependencies') {
            steps {
                sh "npm install"
            }
        }
    }
    post {
     always {
        emailext attachLog: true,
            subject: "'${currentBuild.result}'",
```

```
            body: "Project: ${env.JOB_NAME}<br/>" +
                "Build Number: ${env.BUILD_NUMBER}<br/>" +
                "URL: ${env.BUILD_URL}<br/>",
            to: 'aman07pathak@gmail.com',
            attachmentsPattern: 'trivyfs.txt,trivyimage.txt'
        }
    }
}
```



Click on build pipeline and after getting the success of the pipeline.

You will see the Sonarqube code quality analysis which will look like the below snippet.

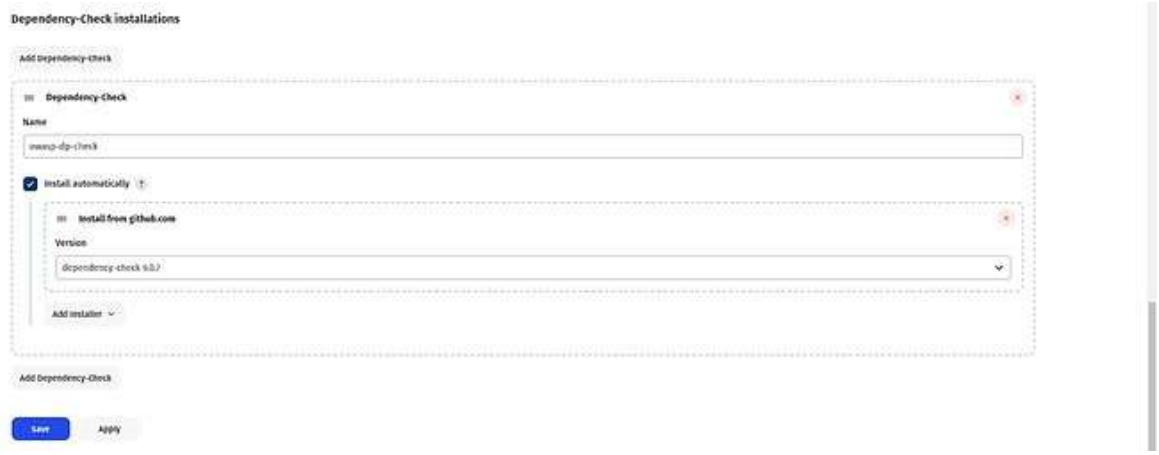
Now, we have to add one more tool for our application named OWASP Dependency-check.

Go to Manage Jenkins -> Plugins

Search for OWASP Dependency-Check and install it.

After installing, make sure to configure the OWASP.

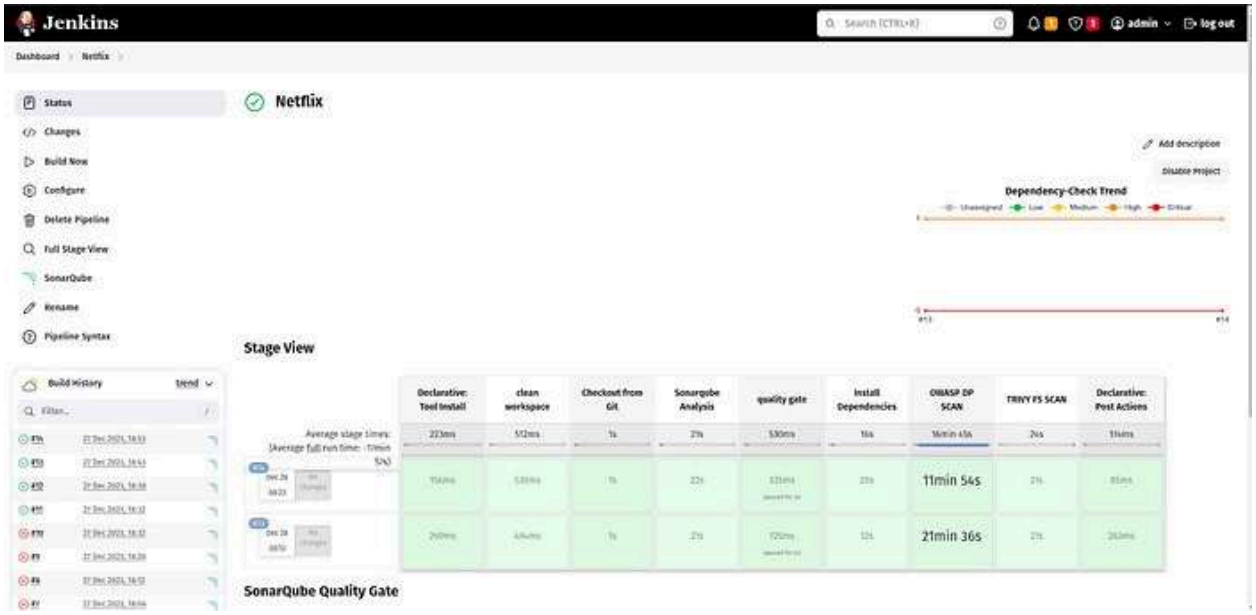
Provide the name select the latest version of OWASP and click on Save.



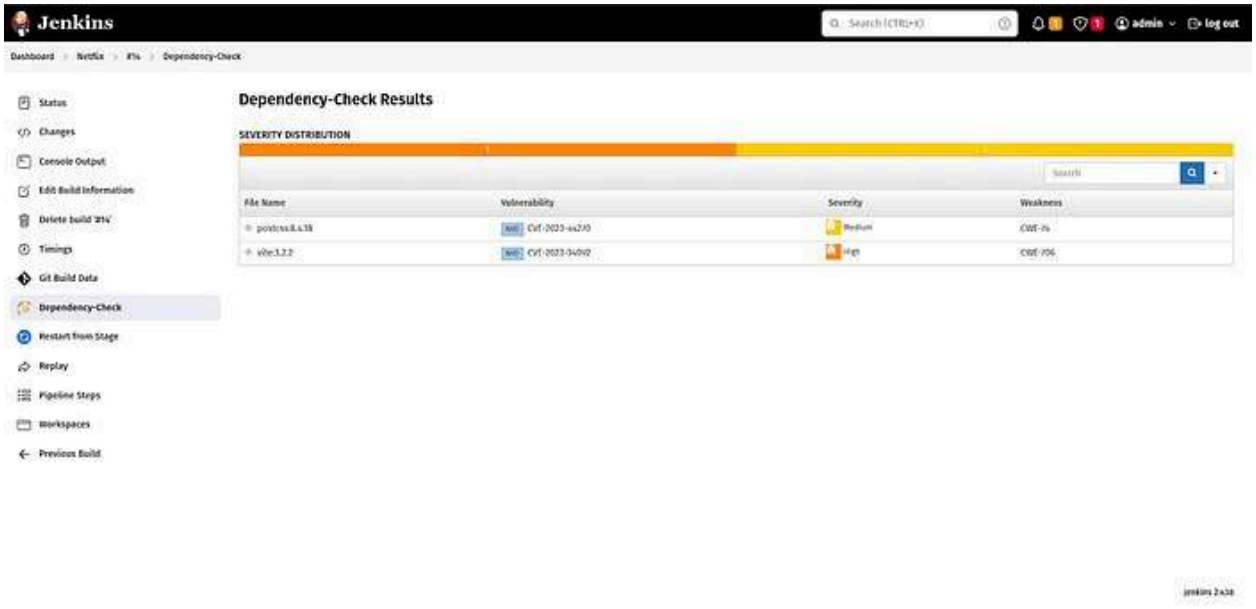
Now, add the OWASP dependency check stage in the Jenkins pipeline and click on Save.

```
stage('OWASP DP SCAN') {
            steps {
                dependencyCheck additionalArguments: '--scan ./ --disableYarnAudit
--disableNodeAudit', odcInstallation: 'owasp-dp-check'
                dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
            }
        }
stage('TRIVY FS SCAN') {
            steps {
                sh "trivy fs . > trivyfs.txt"
            }
        }
```

Now, click on Build Now.



Once your pipeline is successful. Then, scroll down and you will see a dependency check. Click on it and you will see the output like the below snippet.

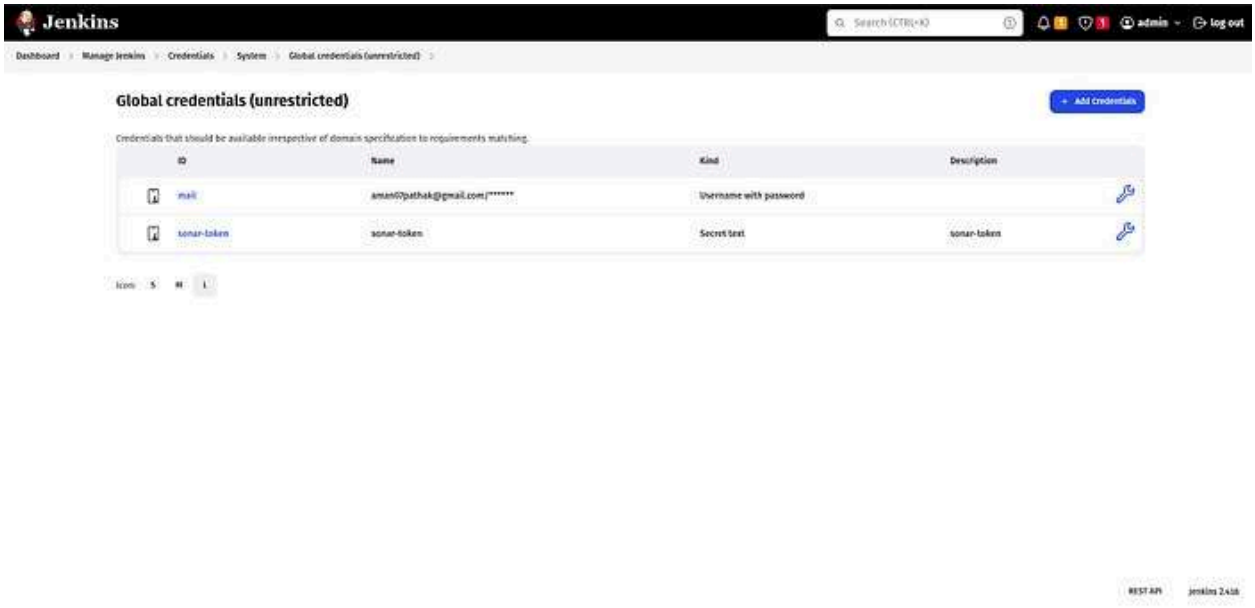


Now, we have to build our Docker Image and push it to DockerHub

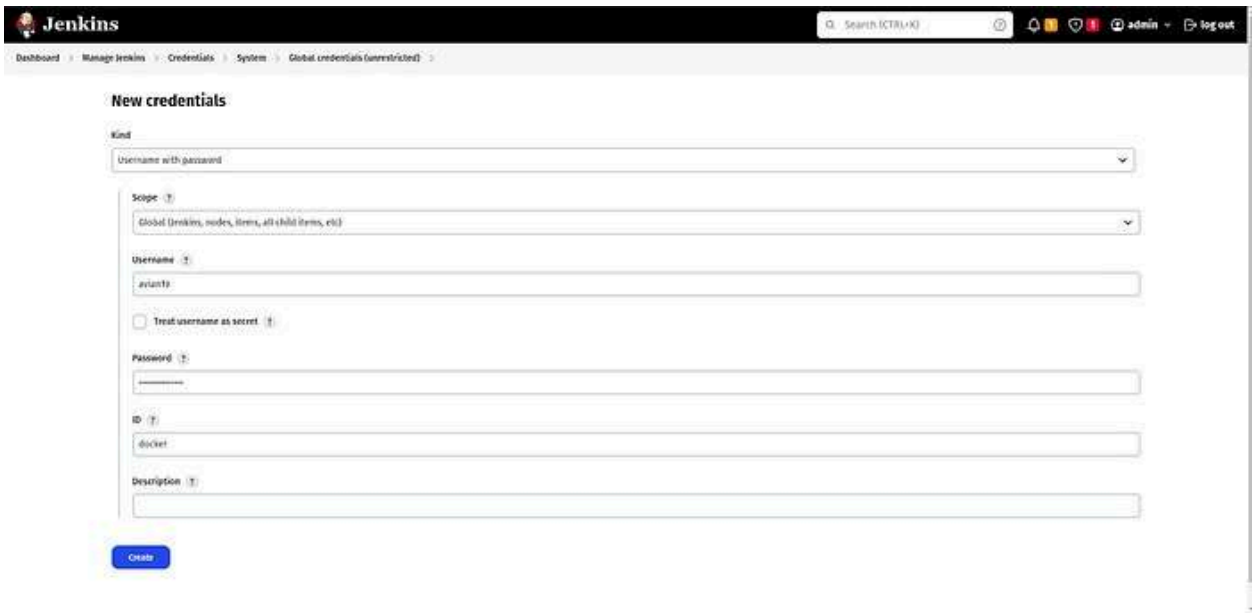
To do that, we need to configure the following things.

Go to Manage Jenkins -> Credentials

Add Docker Credentials to your Jenkins

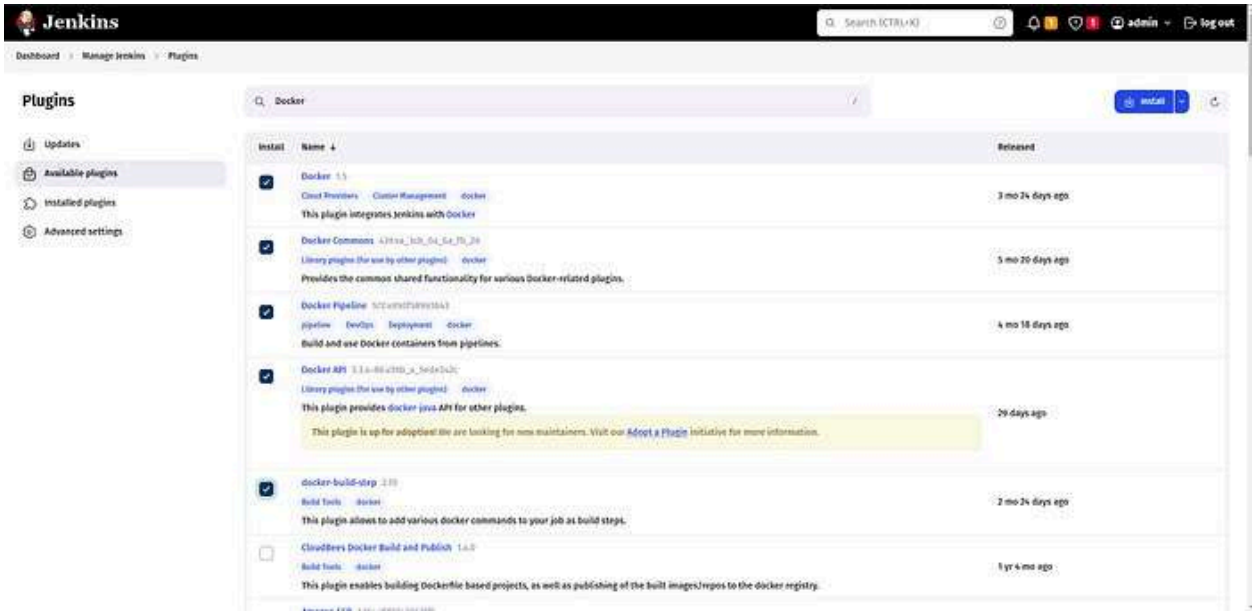


Add your credentials and click on Create.

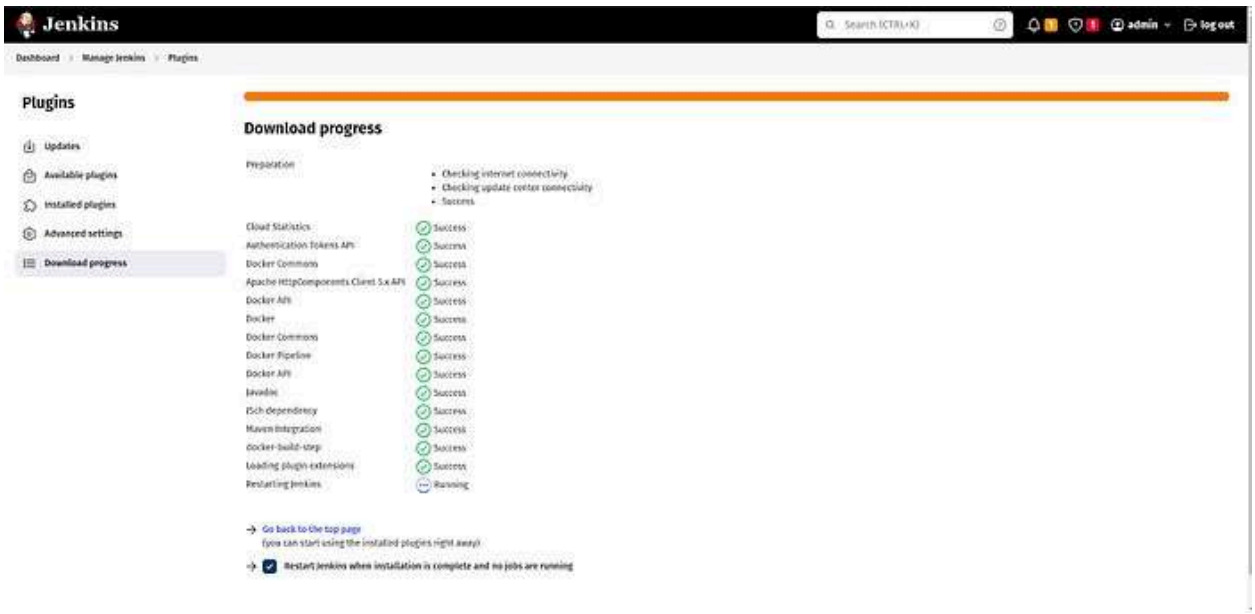


Install the following Docker plugins on your Jenkins

Docker
Docker Commons
Docker Pipeline
Docker API
docker-build-step



Restart your Jenkins



Configure the tool in Jenkins

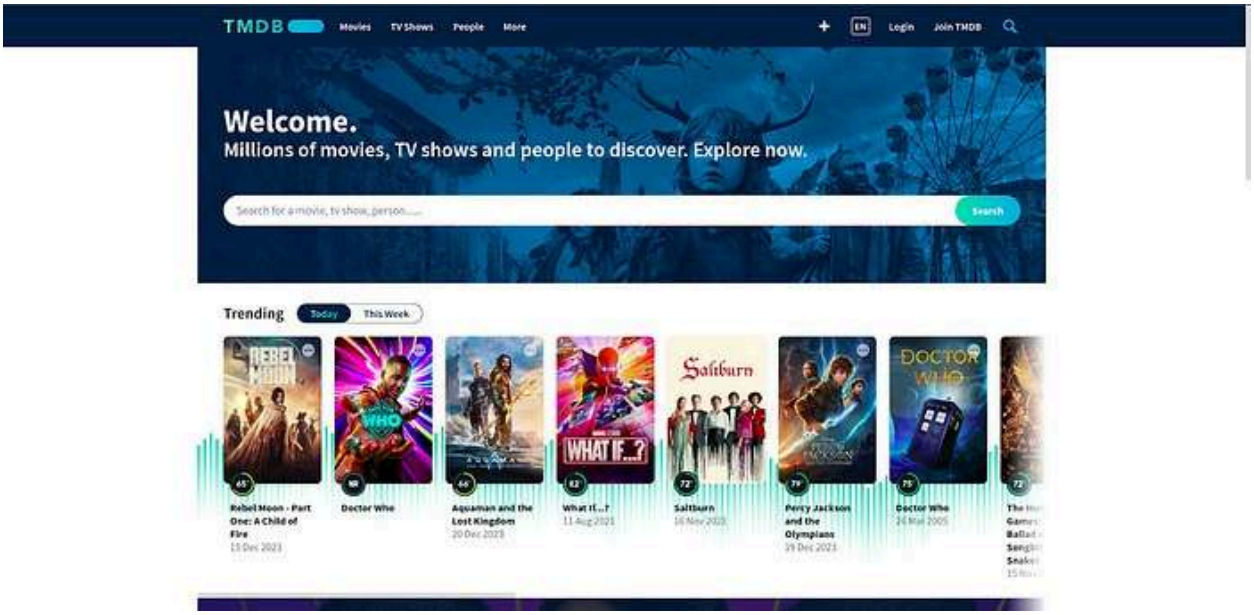
Go to Manage Jenkins -> Tools and provide the below details.

Our application is Netflix Clone. So we need some movie databases on our application.

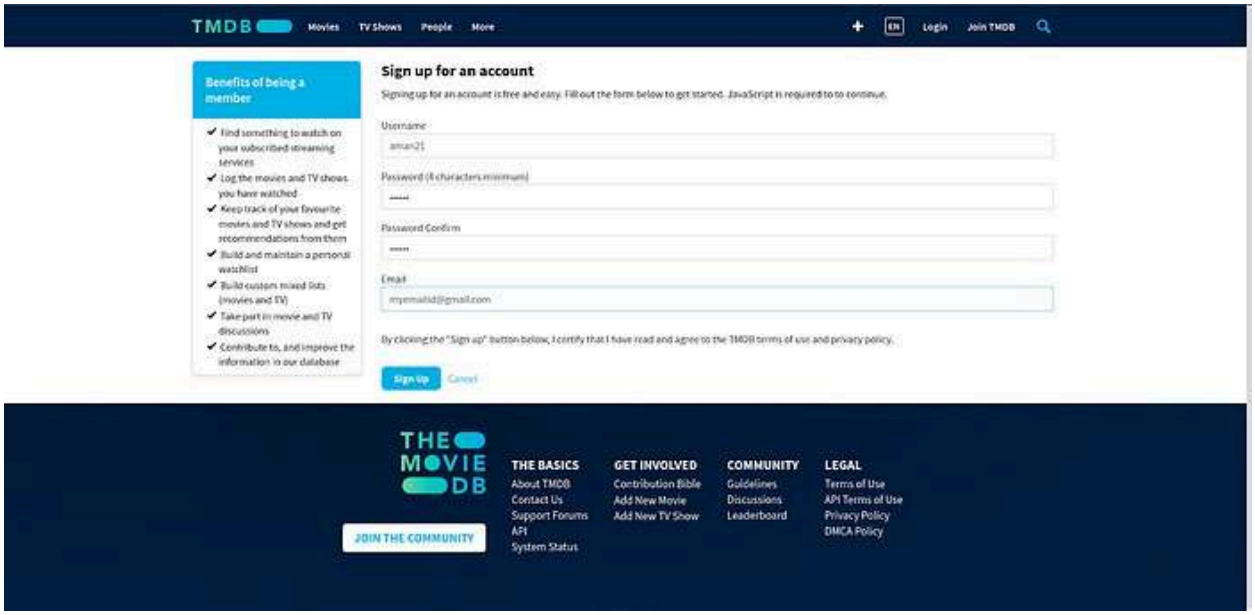
For that, we have one application that will provide the API. So, we can use the API to get the movies on our application.
TMDB is one of them

Go to this link https://www.themoviedb.org/

Click on Join TMDB

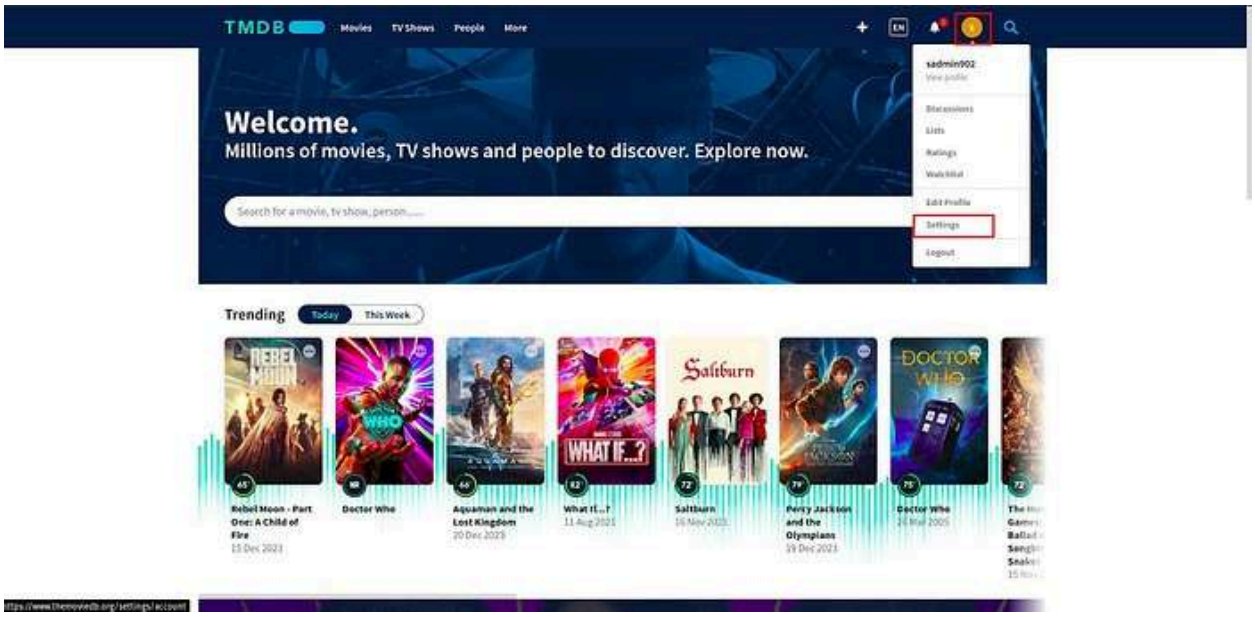


Enter the details and click on SignUp

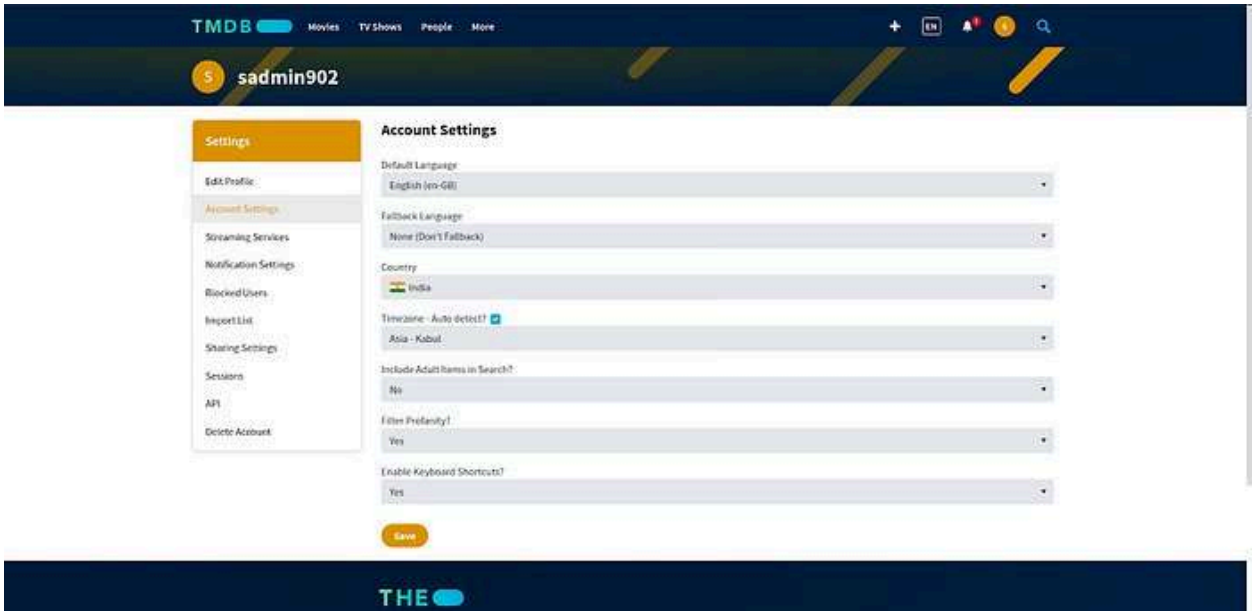


Once you sign up, you will get a confirmation email on your account. Confirm it.

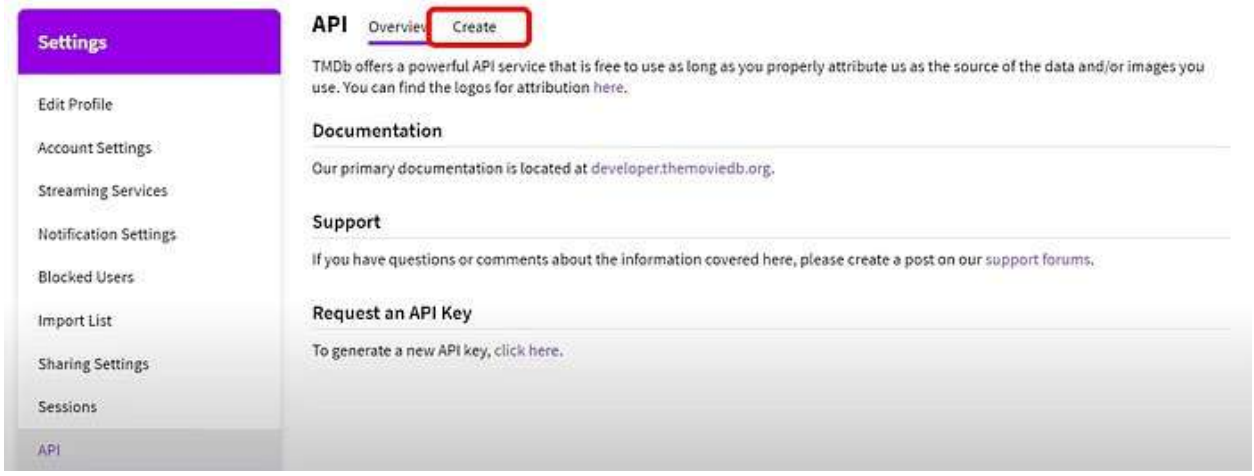
Log in to your TMDB account and go to the settings.



Go to the API section.

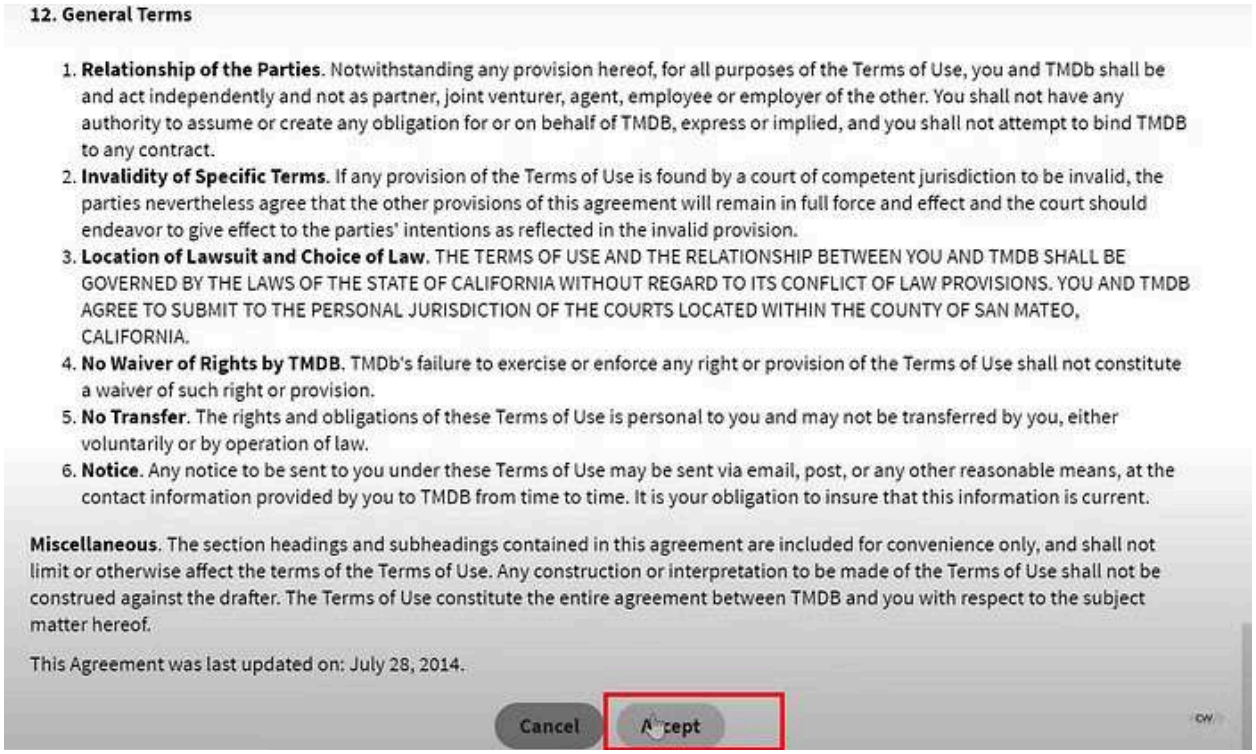


Click on Create to generate API

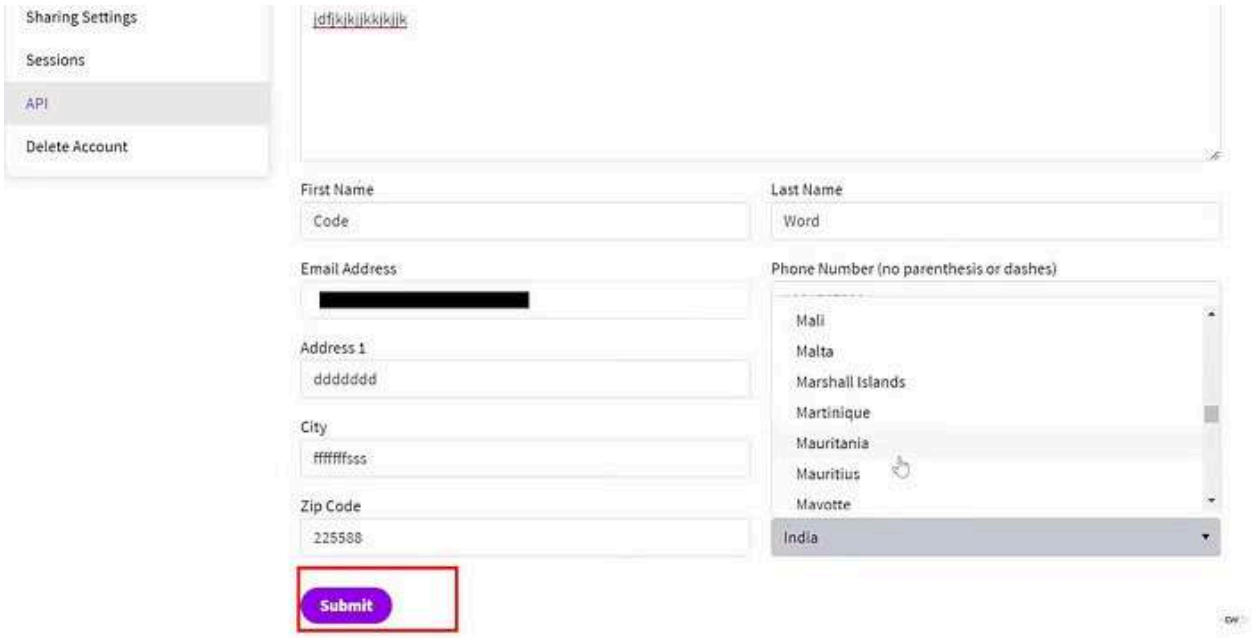


Select Developer.

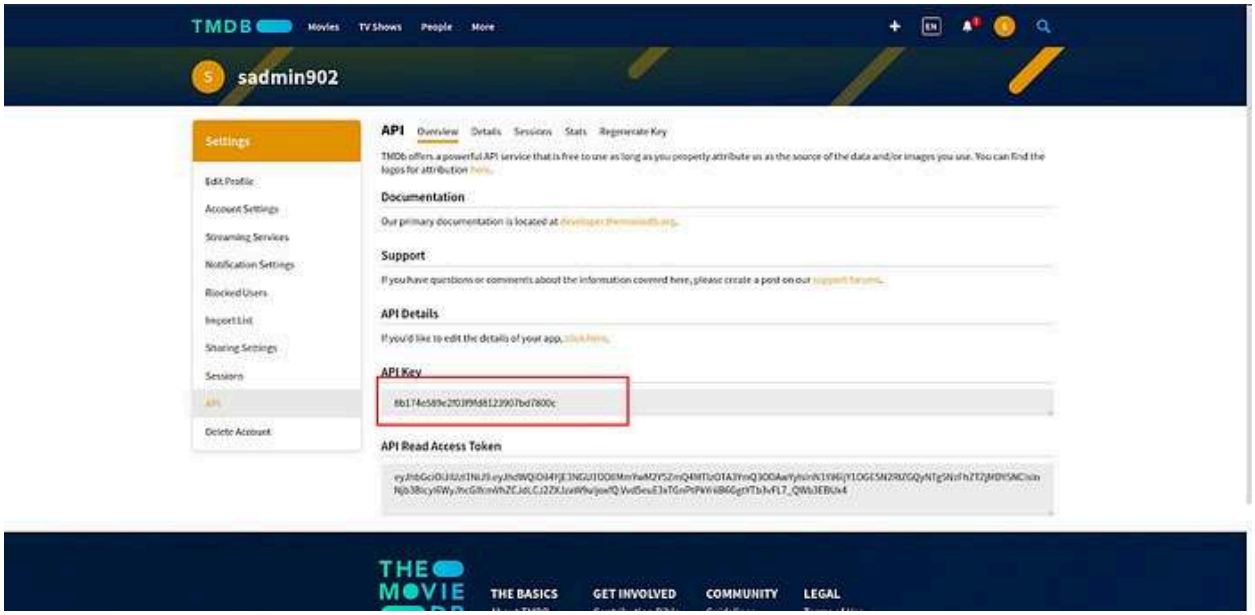
Accept the Terms & Conditions.



Provide the basic details and click on Submit.



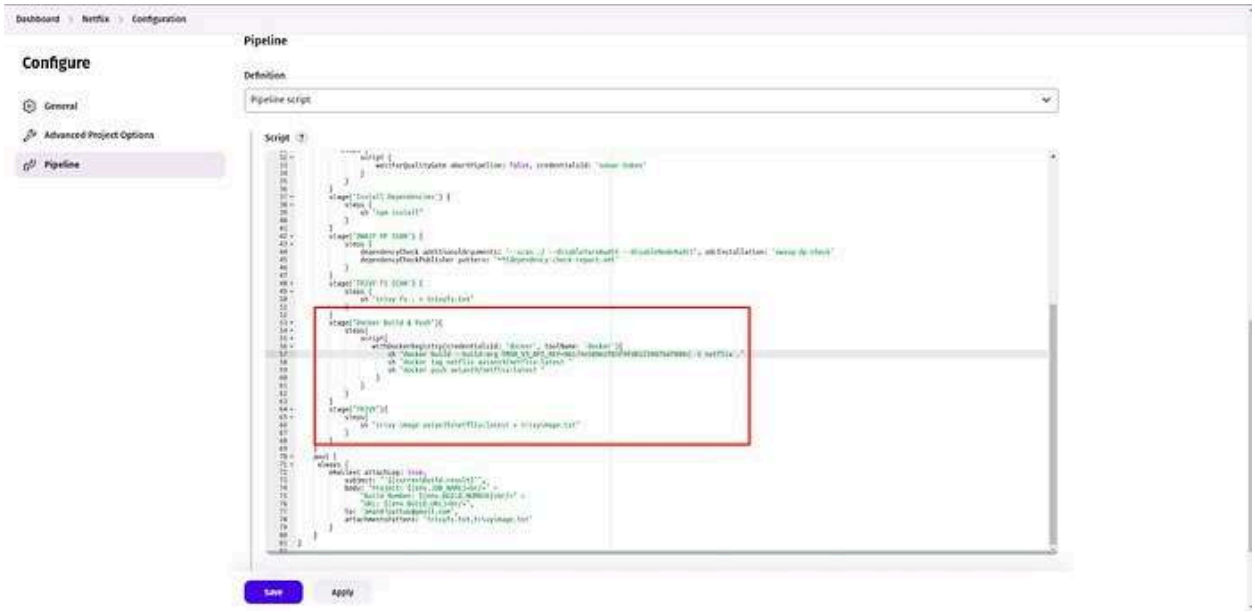
After clicking on Submit. You will get your API. Copy the API and keep it somewhere.



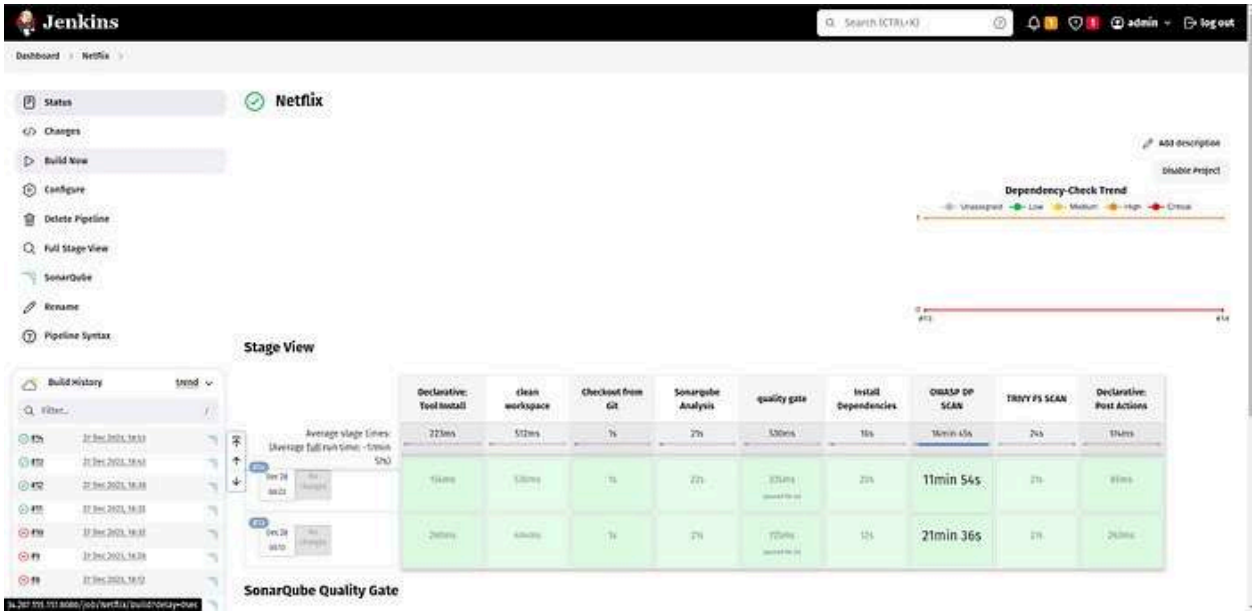
Now, we have to configure our Docker images where we will build our image with the help of new code and then, push it to DockerHub.

After pushing the image, we will scan our DockerHub Image to find the vulnerabilities in the image.

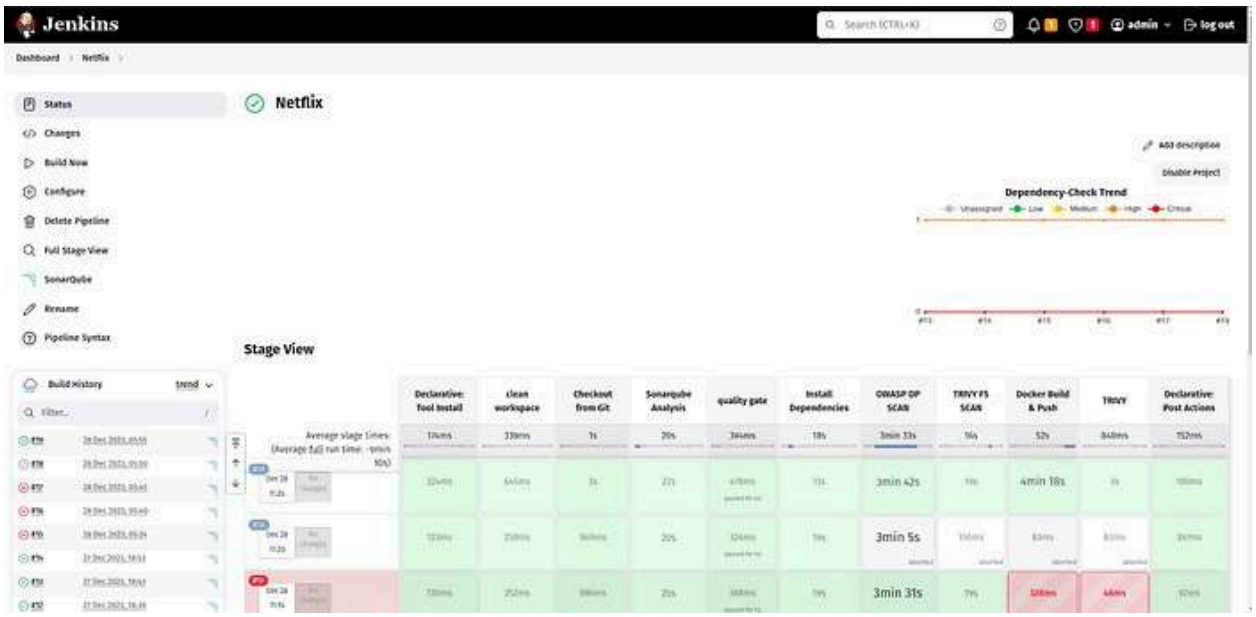
Make sure to replace the API with your API and if you are pushing Dockerfile on your Dockerhub account then, replace my username of the Dockerhub with yours.



Click on Build



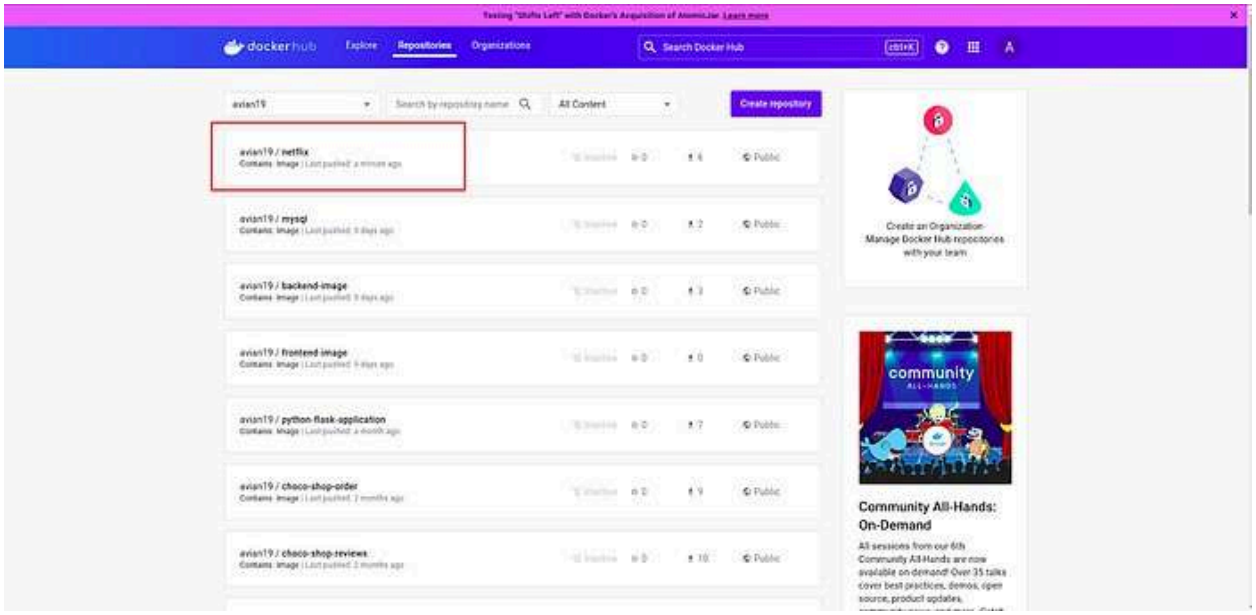
As you can see Our Pipeline is successful.



Now, validate whether the docker image has been pushed to DockerHub or not.

Log in to your Dockerhub account.

As you can see in the below screenshot, Our Docker image is present on Docker Hub.



Now, we have to deploy our application using Kubernetes.

To do that, we need to install kubectl on the Jenkins server.

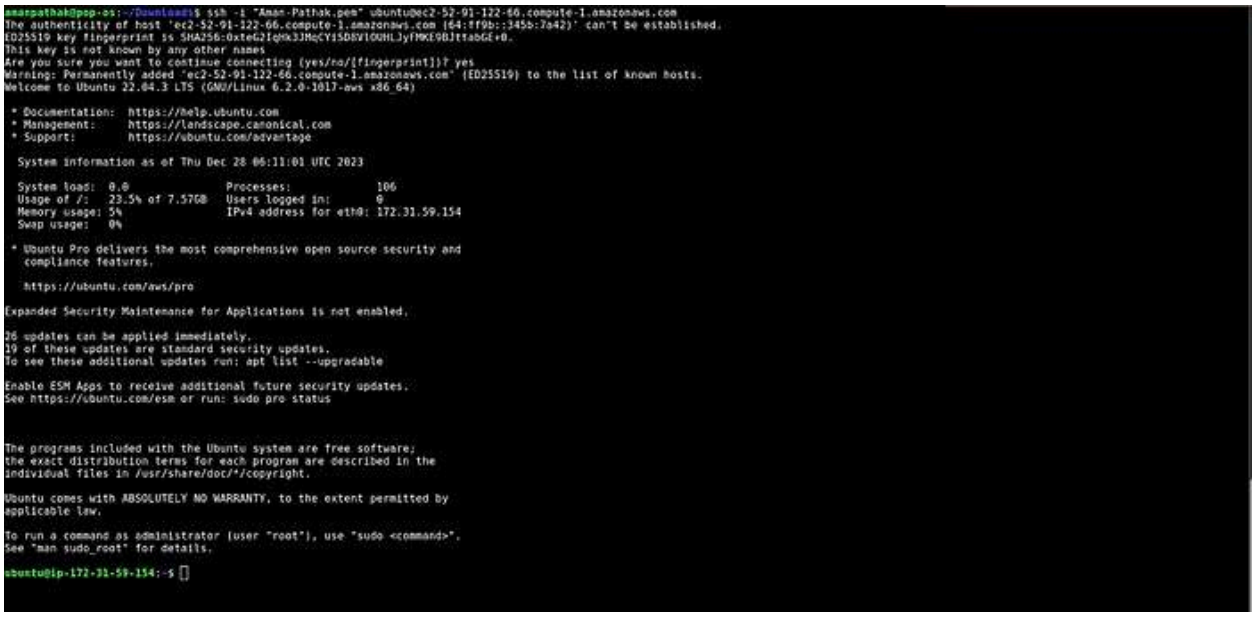
```
sudo apt update
sudo apt install curl
curl -LO https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
kubectl version --client
```

```
ubuntu@ip-172-31-59-9:~$ sudo apt update
sudo apt install curl
curl -LO https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
kubectl version --client
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Hit:5 https://aquasecurity.github.io/trivy-repo/deb jammy InRelease
Ign:6 https://pkg.jenkins.io/debian binary/ InRelease
Hit:7 https://pkg.jenkins.io/debian binary/ Release
Fetched 229 kB in 1s (304 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
7 packages can be upgraded. Run 'apt list --upgradable' to see them.
W: https://aquasecurity.github.io/trivy-repo/deb/dists/jammy/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATION section in apt-key(8) for deta
ils.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
curl is already the newest version (7.81.0-1ubuntu1.15).
curl set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 7 not upgraded.
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   138  100   138    0     0   1132      0 --:--:-- --:--:-- --:--:--  1140
100 47.4M  100 47.4M    0     0  76.6M      0 --:--:-- --:--:-- --:--:-- 76.6M
Client Version: v1.29.0
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
ubuntu@ip-172-31-59-9:~$
ubuntu@ip-172-31-59-9:~$ kubectl --version
error: unknown flag: --version
See 'kubectl --help' for usage.
ubuntu@ip-172-31-59-9:~$ kubectl version
Client Version: v1.29.0
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
Error from server (Forbidden): <html><head><meta http-equiv='refresh' content='1;url=/login?from=%2Fversion%3Ftimeout%3D32s'/><script id='redirect' data-redirect-url='/login?from=%2Fversion%
3Ftimeout%3D32s' src='/static/f25a94ce/scripts/redirect.js'></script></head><body style='background-color:white; color:white;'>
Authentication required
<!--
-->

</body></html>
ubuntu@ip-172-31-59-9:~$
```

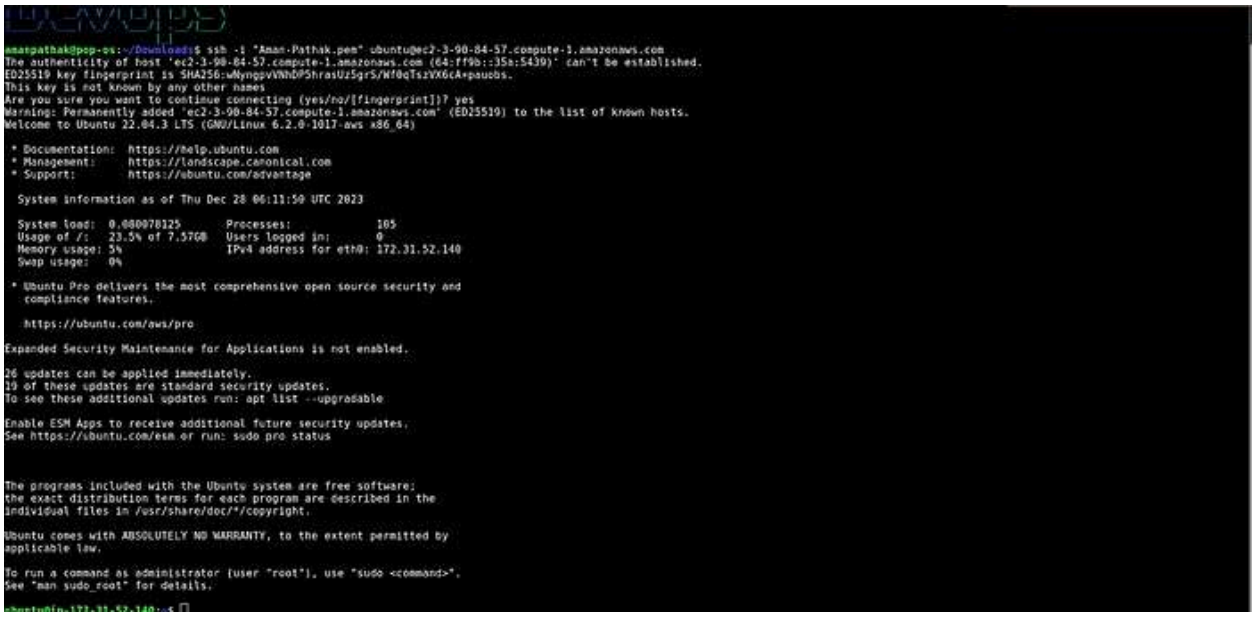
As you know, we have two Kubernetes Nodes of which one is the Master and the other one is the Worker Node.

Login to your both Kubernetes Master and Worker Nodes

Master Node



Worker Node



Add the hostname to your Kubernetes master node
`sudo hostnamectl set-hostname K8s-Master`

Add the hostname to your Kubernetes worker node
`sudo hostnamectl set-hostname K8s-Worker`

Run the below commands on the both Master and worker Nodes.

```
sudo su
swapoff -a; sed -i '/swap/d' /etc/fstab
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
sudo modprobe overlay
sudo modprobe br_netfilter
#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
#Apply sysctl params without reboot
sudo sysctl - system
apt update
sudo apt-g
curl -fsSL https://packages.cloud.google.com/apt/doc/apt-ke
/etc/apt/keyrings/kubernetes-archive-keyring.gpg
echo "deb [signed-by=/etc/apt/keyrings/kubernete
https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list
apt update
apt-get inst
apt install docker.io -y
sudo mkdir /etc/containerd
sudo sh -c "containerd con
sudo sed -i 's/ SystemdCgroup = false/ SystemdCgroup = true/' /et
systemctl restart containerd.service
systemctl restart kubelet.service
systemctl enable kubelet.service
Now, run the following command
command that is highlighted in th
kubeadm config images pull
kubeadm init
```

```
[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 11.002065 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 k8s-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 k8s-master as control-plane by adding the taints [node-role.kubernetes.io/control-plane:NoSchedule]
[bootstrap-token] Using token: deq9nl.y34go2zii10fu8c1
[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/

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

kubeadm join 172.31.59.154:6443 --token deq9nl.y34go2zii10fu8c1 \
        --discovery-token-ca-cert-hash sha256:e93c56bd59b175b81845a671a82ffd1839e42272d922f9c43ca8d8f6d145ce02
root@K8s-Master:/home/ubuntu#
```

Exit from the root user and run the below commands

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

```
ubuntu@ip-172-31-59-154:~$ mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
ubuntu@ip-172-31-59-154:~$
ubuntu@ip-172-31-59-154:~$ ls .kube/
config
ubuntu@ip-172-31-59-154:~$
```

Run on the Worker Node

Run the below command as a root user

```
kubeadm join 172.31.59.154:6443 - token deq9nl.y34go2zii10fu8c1 \
-discovery-token-ca-cert-hash
sha256:e93c56bd59b175b81845a671a82ffd1839e42272d922f9c43ca8d8f6d145ce02
```

```
root@K8s-Worker:/home/ubuntu# kubeadm join 172.31.59.154:6443 --token deq9nl.y34go2zii10fu8c1 \
        --discovery-token-ca-cert-hash sha256:e93c56bd59b175b81845a671a82ffd1839e42272d922f9c43ca8d8f6d145ce02
[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@K8s-Worker:/home/ubuntu#
```

Both nodes are not ready because the network plugin is not installed on the master node

```
ubuntu@K8s-Master:~$ kubectl get nodes
NAME         STATUS     ROLES           AGE     VERSION
k8s-master   NotReady   control-plane   9m30s   v1.28.2
k8s-worker   NotReady   <none>          56s     v1.28.2
ubuntu@K8s-Master:~$
```

Only on the Master Node

Run the below command to install the network plugin on the Master node

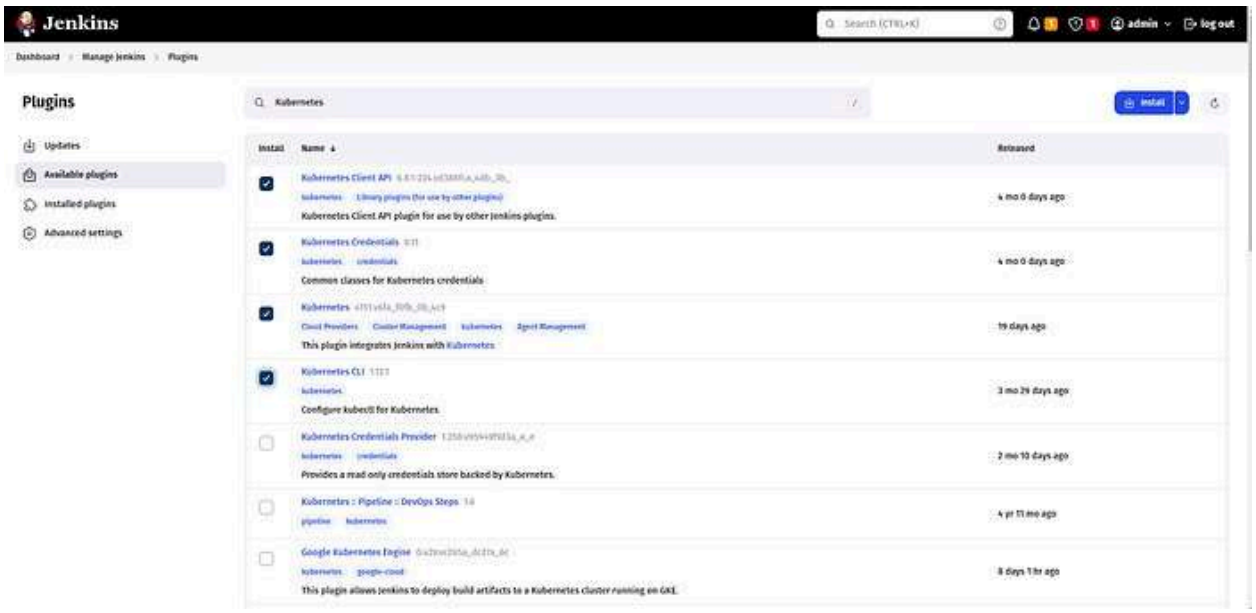
kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml

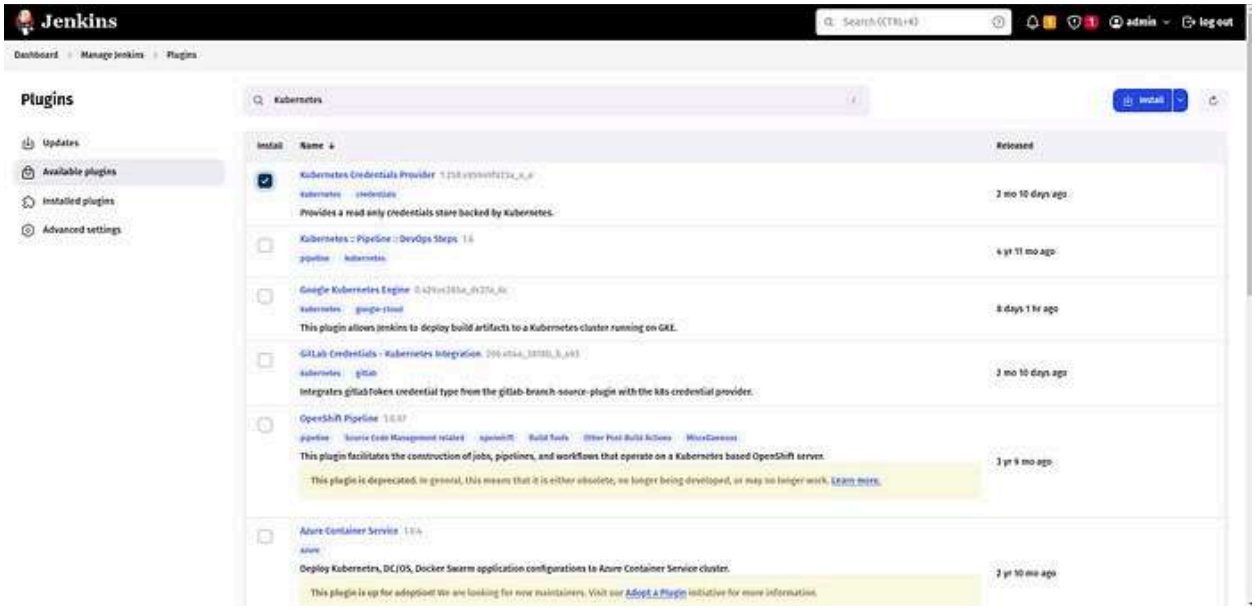
```
ubuntu@K8s-Master:~$ kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml
poddisruptionbudget.policy/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
serviceaccount/calico-node created
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipreservations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
ubuntu@K8s-Master:~$
```

Both nodes are ready.

```
ubuntu@K8s-Master:~$ kubectl get nodes
NAME         STATUS   ROLES           AGE     VERSION
k8s-master   Ready    control-plane   18m     v1.28.2
k8s-worker   Ready    <none>          2m22s   v1.28.2
ubuntu@K8s-Master:~$
```

Install the following Kubernetes Plugins on your Jenkins
Kubernetes
Kubernetes Credentials
Kubernetes Client API
Kubernetes CLI
Kubernetes Credential Provider





Now, we will set Kubernetes Monitoring for both Master and worker Nodes

Run the below command on both Kubernetes Nodes

```
sudo useradd \
--system \
--no-create-home \
--shell /bin/false prometheus
```

Download the node exporter package on both Kubernetes Nodes and Untar the node exporter package file and move the node_exporter directory to the /usr/local/bin directory

```
wget https://github.com/prometheus/node_exporter/releases/download/v1.7.0/node_exporter-1.7.0.linux-amd64.tar.gz
tar -xvf node_exporter-1.7.0.linux-amd64.tar.gz
sudo mv node_exporter-1.7.0.linux-amd64/node_exporter /usr/local/bin/
```

Create the systemd configuration file for node exporter.

Edit the file

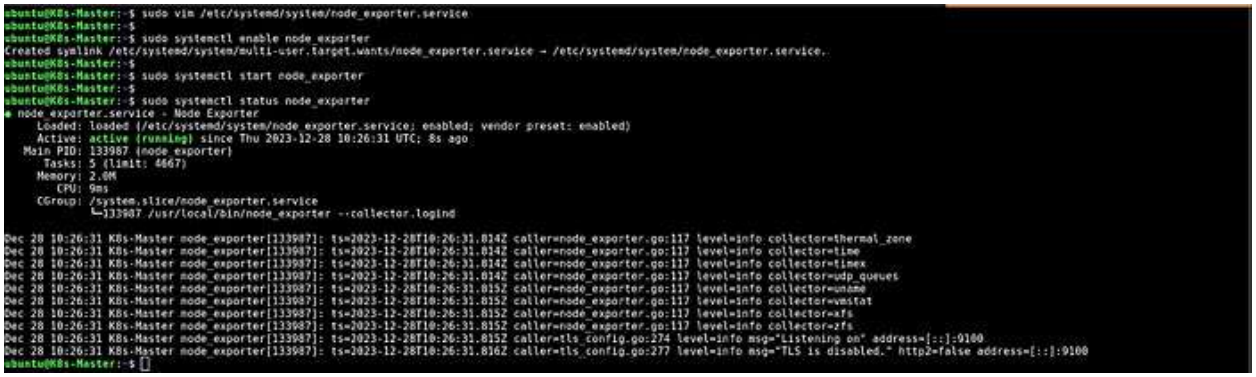
```
sudo vim /etc/systemd/system/node_exporter.service
```

Copy the below configurations and paste them into the /etc/systemd/system/node_exporter.service file.

```
[Unit]
Description=Node Exporter
Wants=network-online.target
After=network-online.target
StartLimitIntervalSec=500
StartLimitBurst=5
[Service]
User=node_exporter
Group=node_exporter
Type=simple
Restart=on-failure
RestartSec=5s
ExecStart=/usr/local/bin/node_exporter \
--collector.logind
[Install]
WantedBy=multi-user.target
```

Enable the node exporter systemd configuration file and start it.

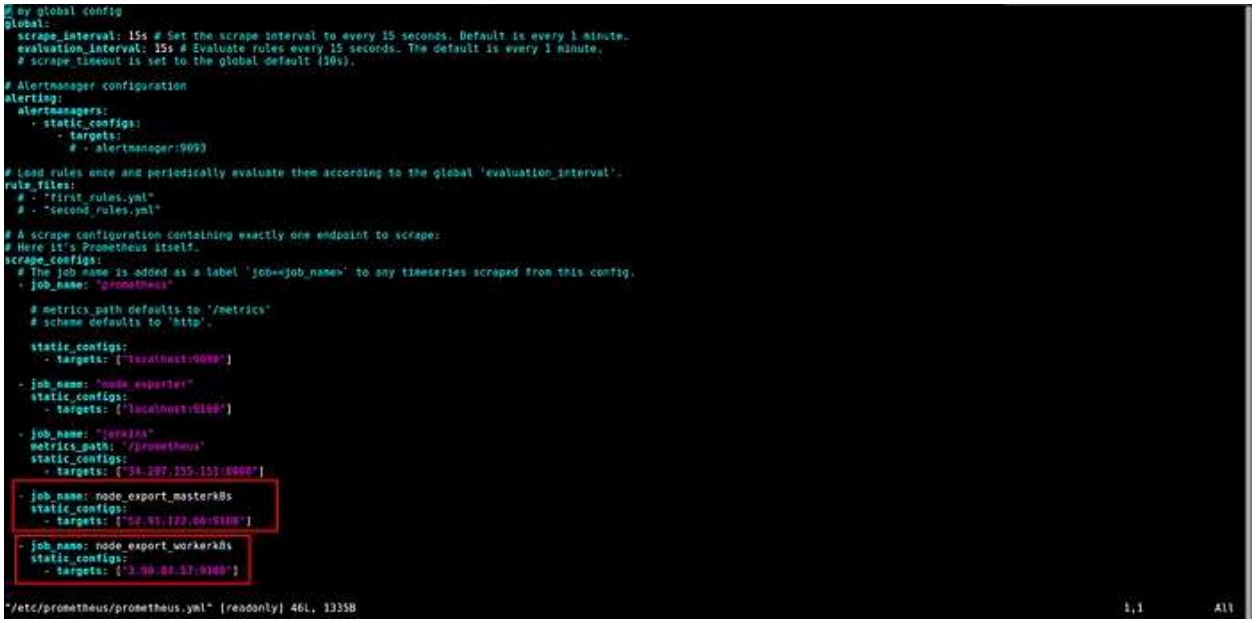
```
sudo systemctl enable node_exporter
node_exporter systemctl status node_exporter
```



Now, we have to add a node exporter to our Prometheus target section. So, we will be able to monitor both Kubernetes Servers.

edit the file
`sudo vim /etc/prometheus/prometheus.yml`

Add both job names(Master & Worker nodes) with their respective public.



After saving the file, validate the changes that you have made using promtool.
`promtool check config /etc/prometheus/prometheus.yml`

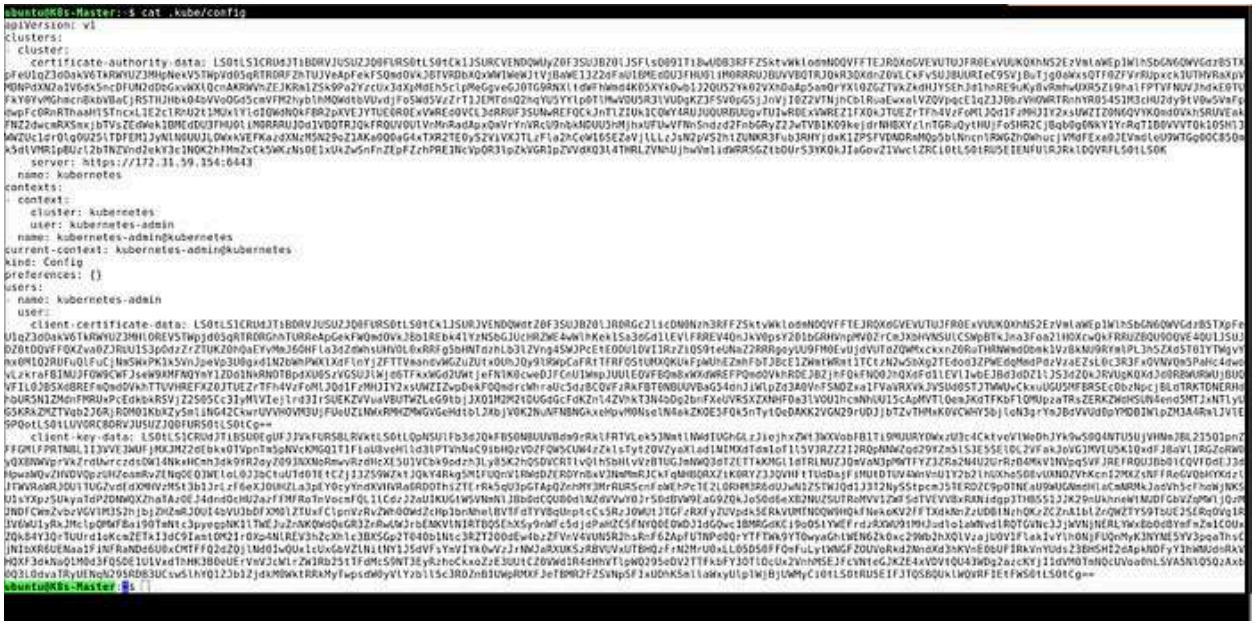
If your changes have been validated then, push the changes to the Prometheus server.

`curl -X POST http://localhost:9090/-/reload`

As you know, Jenkins will deploy our application on the Kubernetes Cluster. To do that, Jenkins must have the access keys or something to connect with the master node.

To do that copy the content inside .kube/config on Kubernetes Master node.

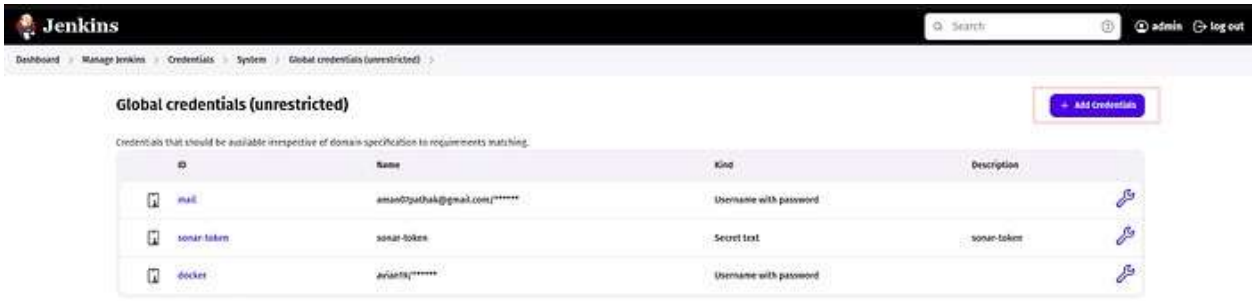
`cat .kube/config`



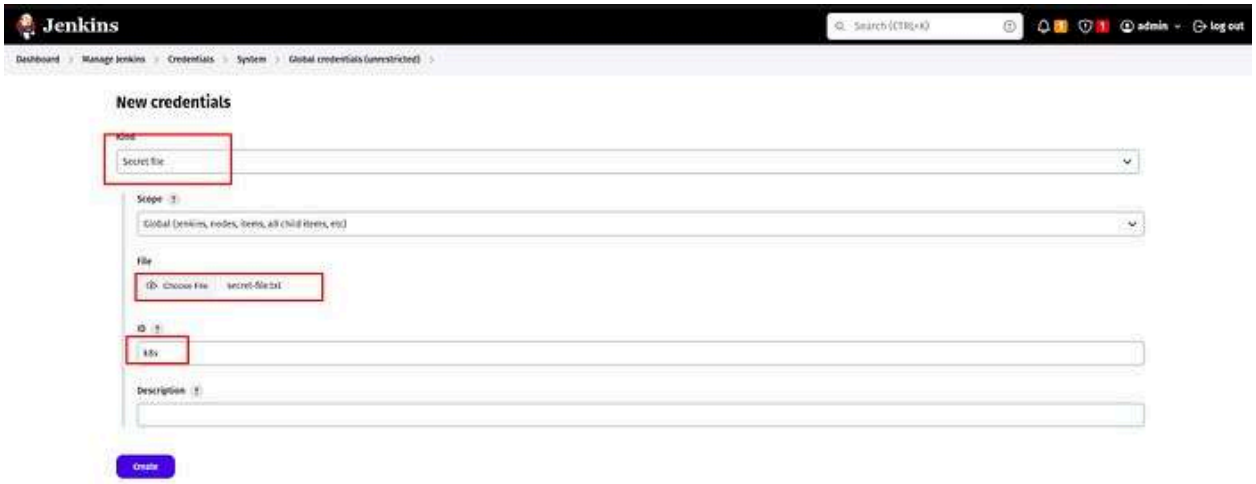
Save the file with the .txt extension.

Now, add the Secret file in Jenkins Credentials.

Click on Add credentials.

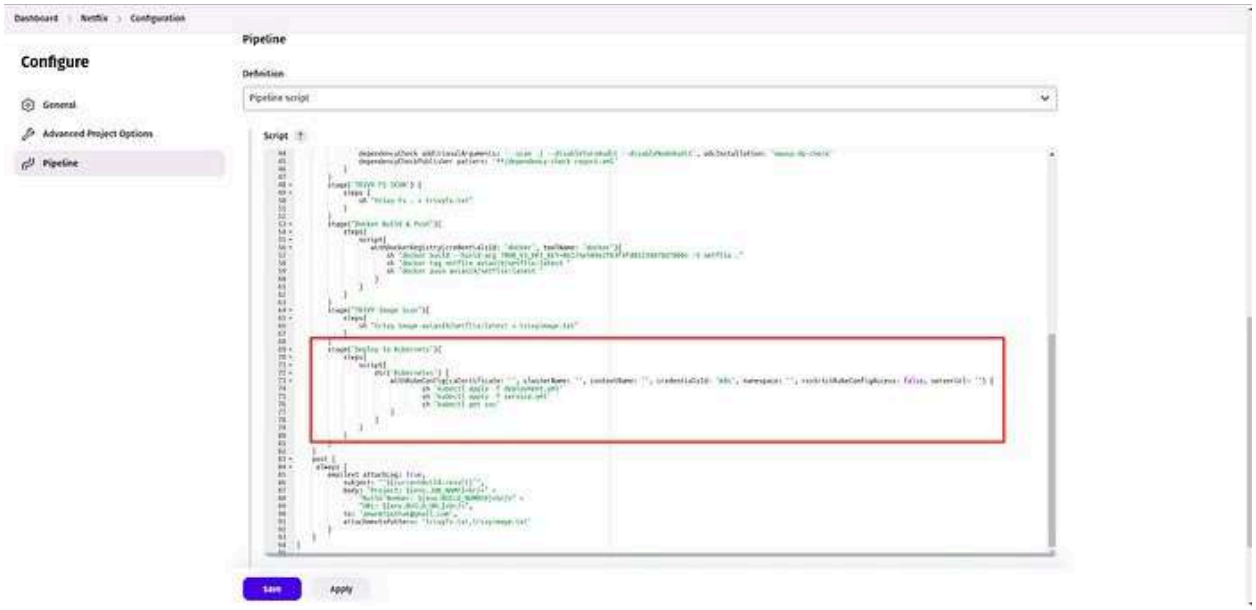


Select the Secret file and provide the Secret file that you have saved earlier enter the ID k8s then click on Create.



Now, Add the deploy to the Kubernetes stage in your Jenkins pipeline.

```
stage('Deploy to Kubernetes'){
        steps{
            script{
                dir('Kubernetes') {
                    withKubeConfig(caCertificate: '', clusterName: '', contextName: '',
credentialsId: 'k8s', namespace: '', restrictKubeConfigAccess: false, serverUrl: '') {
                        sh 'kubectl apply -f deployment.yml'
                        sh 'kubectl apply -f service.yml'
                        sh 'kubectl get svc'
                        sh 'kubectl get all'
                    }
                }
            }
        }
    }
```

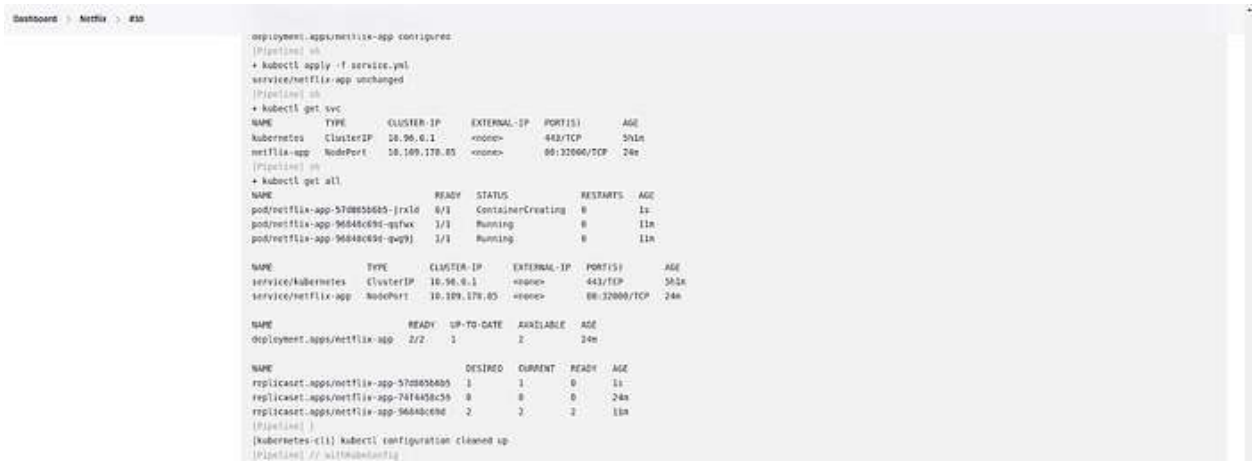


Click on Build Now

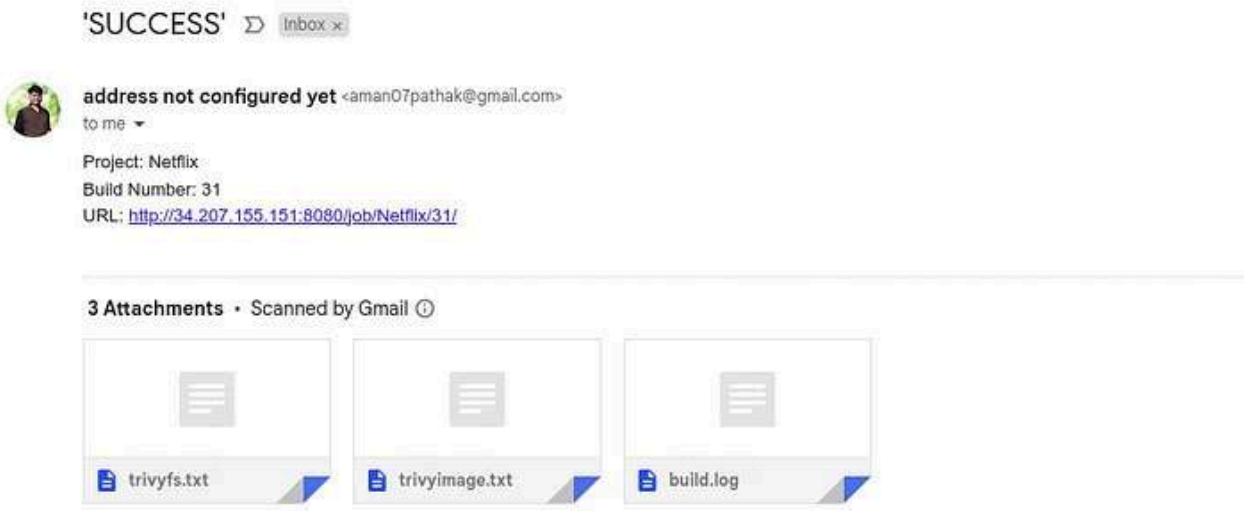
You will see that our Application has been deployed successfully on Kubernetes.

You can validate whether your pods are running or not from your Kubernetes master node.

Also, you can check the Console logs for the earlier results.



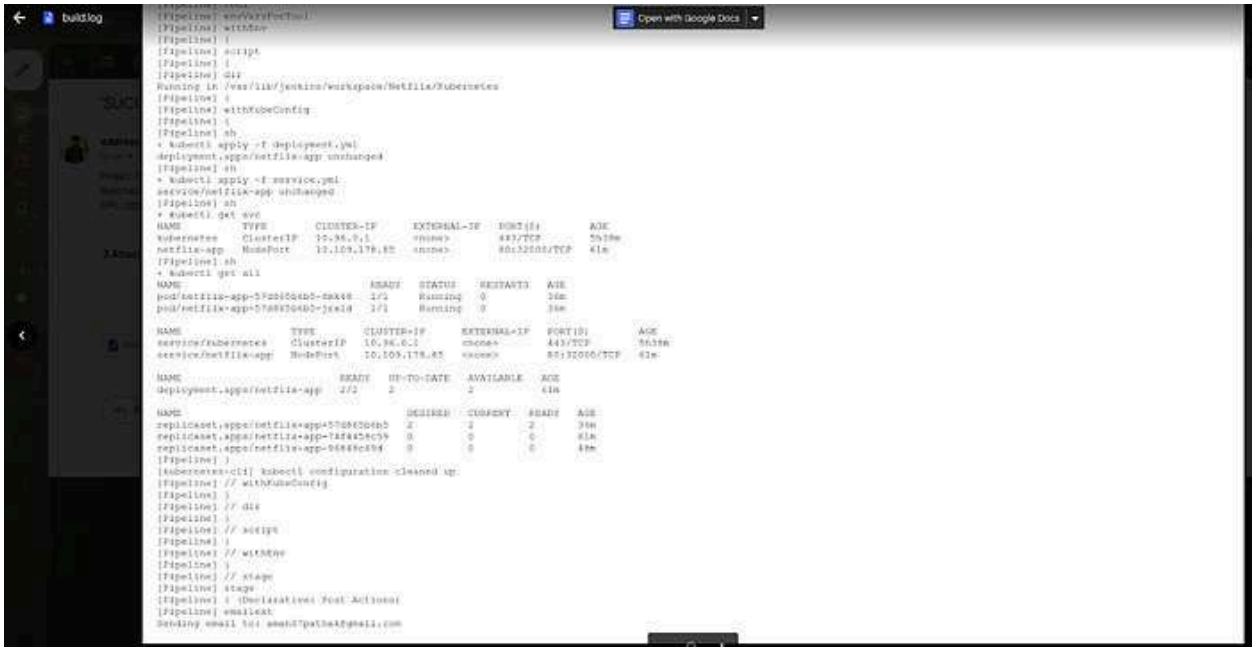
We got the email that our pipeline was successful.



We get the trivyfs.txt file which contains the vulnerabilities.

Also, we got the vulnerabilities for our Docker Image.

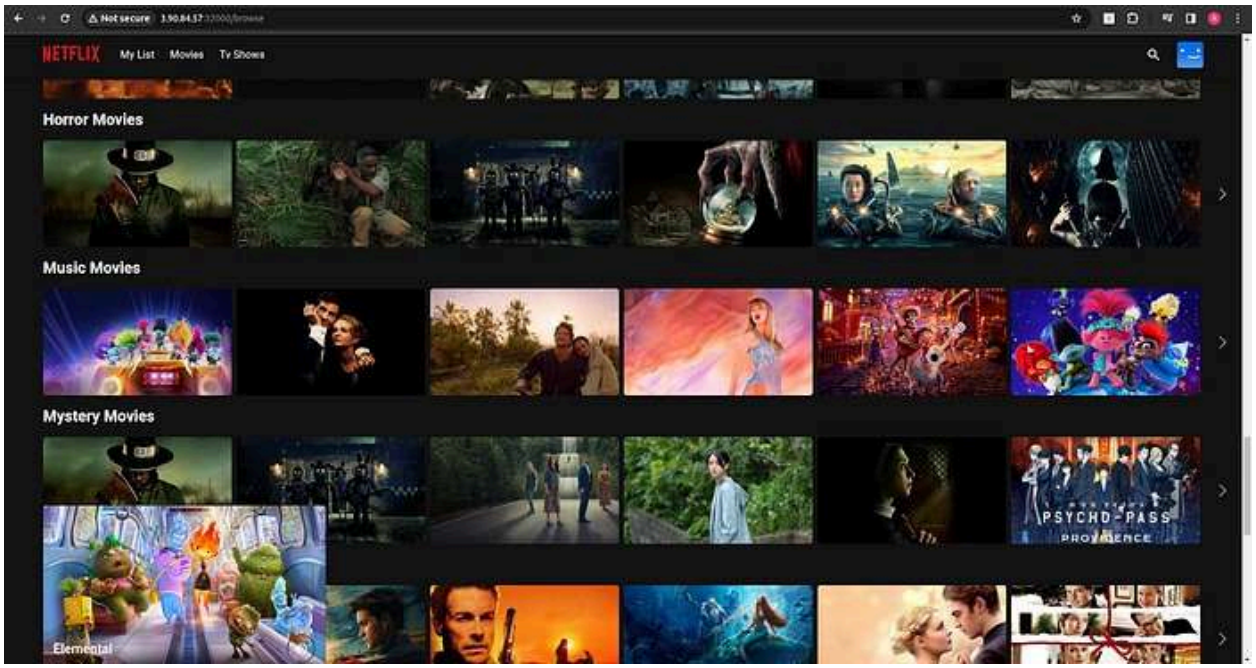
Jenkins sent the console logs by email.



If you want to access your Netflix Clone Application.

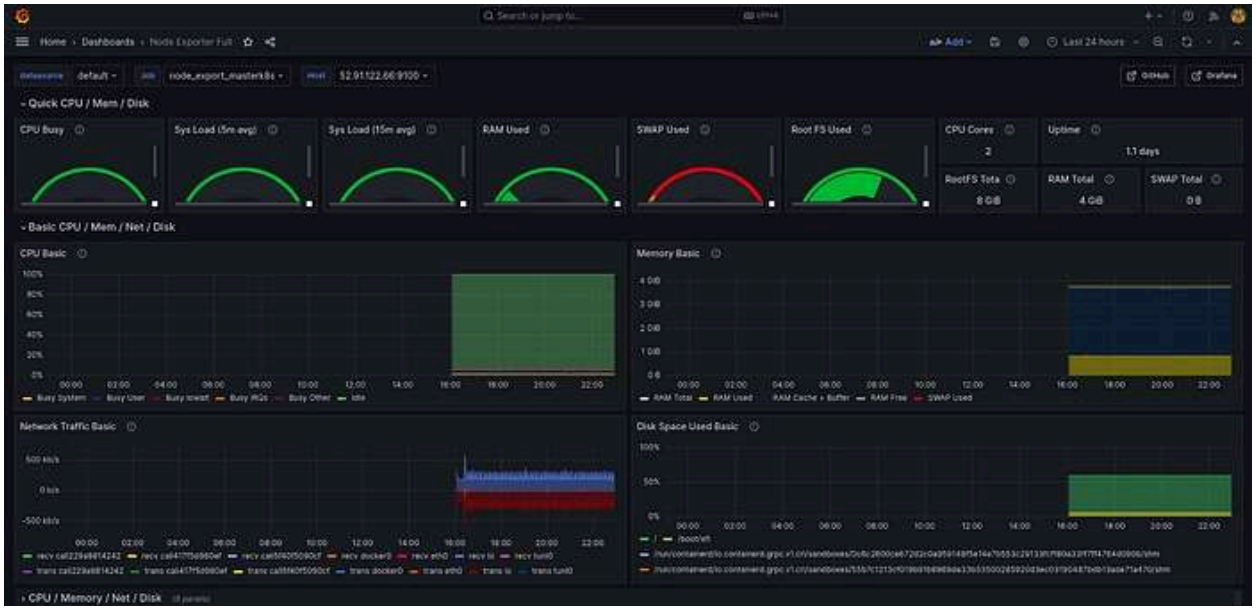
Copy the Public IP of Worker Node and paste it on your favorite browser with port 32000 and see the magic.

Another Snippet of our Netflix Clone application.

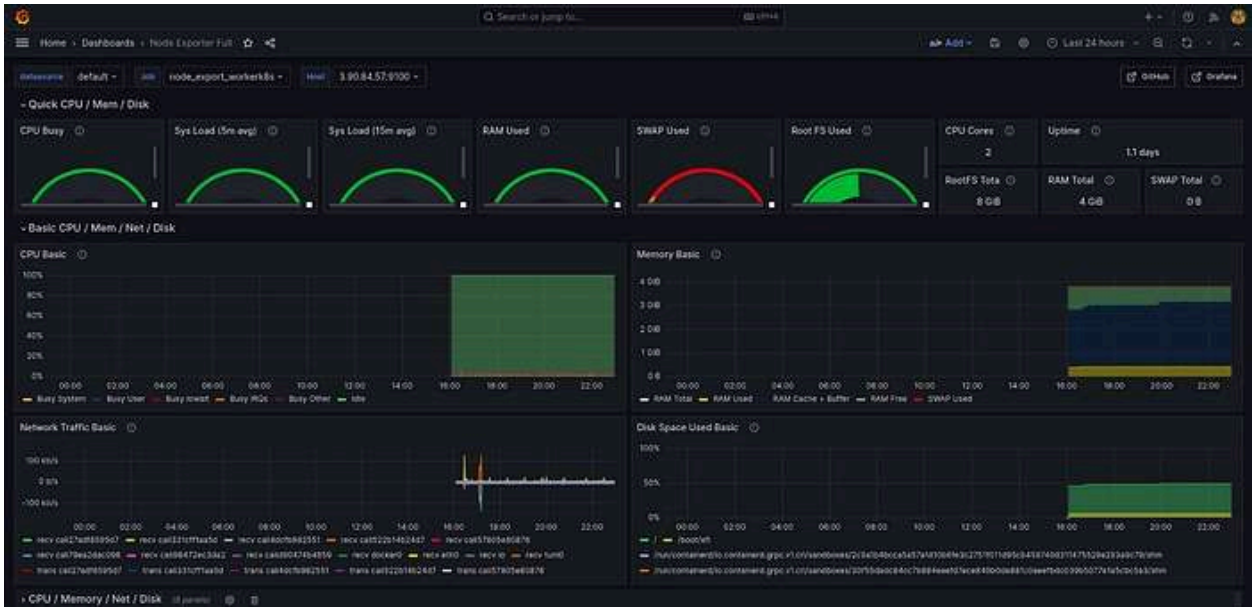


Go to the Grafana Dashboard and select Node Exporter.

You will see the real-time hardware specs of your Kubernetes master node.



You will see the real-time hardware specs of your Kubernetes worker node.




code
plan
release
deploy
operate
monitor
test
build
Dev
Ops
to the learners
thank you!
from aman pathak
HELM