CKA- 2024

Certified Kubernetes Administrator

CKA Domains & Competencies

1. **Storage 10%**
   - Understand storage classes, persistent volumes
   - Understand volume mode, access modes and reclaim policies for volumes
   - Understand persistent volume claims primitive
   - Know how to configure applications with persistent storage

2. **Troubleshooting 30%**
   - Evaluate cluster and node logging
   - Understand how to monitor applications
   - Manage container stdout & stderr logs
   - Troubleshoot application failure
   - Troubleshoot cluster component failure
   - Troubleshoot networking

3. **Workloads & Scheduling 15%**
   - Understand deployments and how to perform rolling update and rollbacks
   - Use ConfigMaps and Secrets to configure applications
   - Know how to scale applications
   - Understand the primitives used to create robust, self-healing, application deployments
   - Understand how resource limits can affect Pod scheduling
   - Awareness of manifest management and common templating tools

4. **Cluster Architecture, Installation & Configuration 25%**
   - Manage role-based access control (RBAC)
   - Use Kubeadm to install a basic cluster
   - Manage a highly-available Kubernetes cluster
   - Provision underlying infrastructure to deploy a Kubernetes cluster
   - Perform a version upgrade on a Kubernetes cluster using Kubeadm
   - Implement etcd backup and restore

5. **Services & Networking 20%**
   - Understand host networking configuration on the cluster nodes
   - Understand connectivity between Pods
   - Understand ClusterIP, NodePort, LoadBalancer service types and endpoints
   - Know how to use Ingress controllers and Ingress resources
   - Know how to configure and use CoreDNS
   - Choose an appropriate container network interface plugin

Contents

KIND Cluster Installation

Kind Installation in Windows WSL

```
vishal@LAPTOP-NGR39AL4:/$ curl -Lo ~/kind https://kind.sigs.k8s.io/dl/v0.20.0/kind-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    97  100    97    0     0    222      0 --:--:-- --:--:-- --:--:--   222
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 6304k  100 6304k    0     0  1712k      0  0:00:03  0:00:03 --:--:-- 2089k
vishal@LAPTOP-NGR39AL4:/$ chmod +x ~/kind
vishal@LAPTOP-NGR39AL4:/$ sudo mv ~/kind /usr/local/bin/kind
vishal@LAPTOP-NGR39AL4:/$ kind --version
kind version 0.20.0
```

Commands

curl -Lo ~/kind https://kind.sigs.k8s.io/dl/v0.20.0/kind-linux-amd64
chmod +x ~/kind
sudo mv ~/kind /usr/local/bin/kind

Creating a Single Node Cluster

kind create cluster --image kindest/node:v1.29.4@sha256:3abb816a5b1061fb15c6e9e60856ec40d56b7b52bcea5f5f1350bc6e2320b6f8 --name cka-cluster1

```
vishal@LAPTOP-NGR39AL4:/$ kind create cluster --image kindest/node:v1.29.4@sha256:3abb816a5b1061fb15c6e9e60856ec40d56b7b52bcea5f5f1350bc6e2320b6f8 --name cka-cluster1
Creating cluster "cka-cluster1" ...
 ✓ Ensuring node image (kindest/node:v1.29.4) 🖼 ^[S
 ✓ Preparing nodes 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
Set kubectl context to "kind-cka-cluster1"
You can now use your cluster with:

kubectl cluster-info --context kind-cka-cluster1

Not sure what to do next? 😅  Check out https://kind.sigs.k8s.io/docs/user/quick-start/
vishal@LAPTOP-NGR39AL4:/$ kubectl cluster-info --context kind-cka-cluster1
Kubernetes control plane is running at https://127.0.0.1:39403
CoreDNS is running at https://127.0.0.1:39403/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
vishal@LAPTOP-NGR39AL4:/$
```

Install Kubectl

```
vishal@LAPTOP-NGR39AL4:/$ kubectl version --client
Client Version: v1.29.2
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
vishal@LAPTOP-NGR39AL4:/$
```

Deleting a Cluster in KIND

kind delete cluster --name kind-cka-multinodecluster

```
root@LAPTOP-NGR39AL4:~# kubectl config get-contexts
CURRENT   NAME                          CLUSTER                       AUTHINFO                      NAMESPACE
*         kind-cka-multinodecluster     kind-cka-multinodecluster     kind-cka-multinodecluster
root@LAPTOP-NGR39AL4:~# kind delete cluster --name kind-cka-multinodecluster
Deleting cluster "kind-cka-multinodecluster" ...
root@LAPTOP-NGR39AL4:~#
```

Creating a Multinode Cluster

```
vishal@LAPTOP-NGR39AL4:~$ cat mn1.yml
# three node (two workers) cluster config
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  extraPortMappings:
  - containerPort: 30001
    hostPort: 30001
- role: worker
- role: worker
```

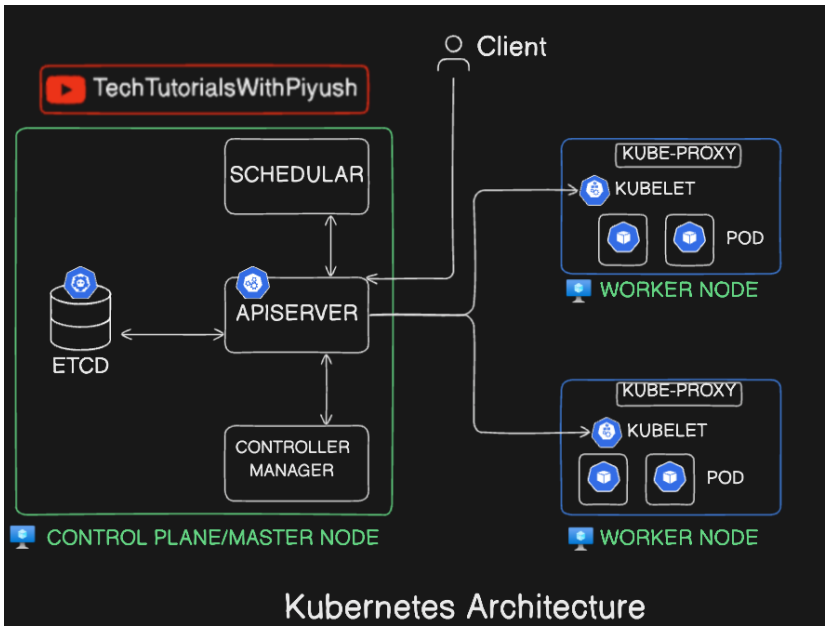
kind create cluster --image kindest/node:v1.29.4@sha256:3abb816a5b1061fb15c6e9e60856ec40d56b7b52bcea5f5f1350bc6e2320b6f8 --name cka-multinodecluster1 --config mn1.yml

```
vishal@LAPTOP-NGR39AL4:~$ ls
mn1.yml
vishal@LAPTOP-NGR39AL4:~$ kind create cluster --image kindest/node:v1.29.4@sha256:3abb816a5b1061fb15c6e9e60856ec40d56b7b52bcea5f5f1350bc6e2320b6f8 --name cka-multinodecluster1 --config mn1.yml
Creating cluster "cka-multinodecluster1" ...
 ✓ Ensuring node image (kindest/node:v1.29.4) 🖼
 ✓ Preparing nodes 📦 📦 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
 ✓ Joining worker nodes 🚜
Set kubectl context to "kind-cka-multinodecluster1"
You can now use your cluster with:

kubectl cluster-info --context kind-cka-multinodecluster1

Have a nice day! 👋
vishal@LAPTOP-NGR39AL4:~$ kubectl config get-contexts
CURRENT   NAME                          CLUSTER                       AUTHINFO                      NAMESPACE
          kind-cka-cluster1             kind-cka-cluster1             kind-cka-cluster1
*         kind-cka-multinodecluster1    kind-cka-multinodecluster1    kind-cka-multinodecluster1
vishal@LAPTOP-NGR39AL4:~$ kubectl get nodes
NAME                                  STATUS   ROLES           AGE     VERSION
cka-multinodecluster1-control-plane   Ready    control-plane   5m31s   v1.29.4
cka-multinodecluster1-worker          Ready    <none>          5m6s    v1.29.4
cka-multinodecluster1-worker2         Ready    <none>          5m6s    v1.29.4
vishal@LAPTOP-NGR39AL4:~$ kubectl config use-context kind-cka-cluster1
Switched to context "kind-cka-cluster1".
vishal@LAPTOP-NGR39AL4:~$ kubectl get nodes
NAME                         STATUS   ROLES           AGE   VERSION
cka-cluster1-control-plane   Ready    control-plane   72m   v1.29.4
vishal@LAPTOP-NGR39AL4:~$
```

```
vishal@LAPTOP-NGR39AL4: ~
vishal@LAPTOP-NGR39AL4:~/p1$ kubectl run nginx --image=nginx --dry-run=client -o yml
error: unable to match a printer suitable for the output format "yml", allowed formats are: go-template,go-template-file,json,jsonpath,jsonpath-as-json,jsonpath-file,name,te
vishal@LAPTOP-NGR39AL4:~/p1$ kubectl run nginx --image=nginx --dry-run=client -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
vishal@LAPTOP-NGR39AL4:~/p1$ ls
vishal@LAPTOP-NGR39AL4:~/p1$ kubectl run nginx --image=nginx --dry-run=client -o yaml > testpod.yaml
vishal@LAPTOP-NGR39AL4:~/p1$ ls
testpod.yaml
vishal@LAPTOP-NGR39AL4:~/p1$ cat testpod.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
vishal@LAPTOP-NGR39AL4:~/p1$
```

Kubernetes Architecture

Kubernetes Architecture



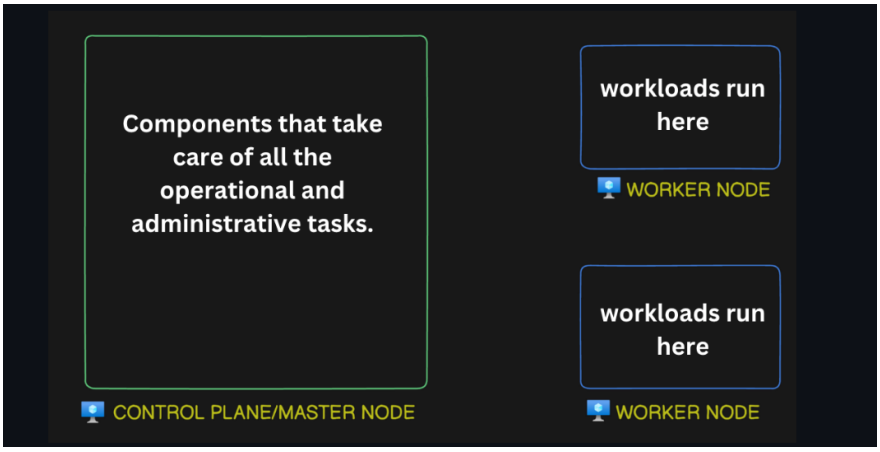
Master Node V/s Worker Node

Role of Master Node:

The Master Node is the brain of the Kubernetes cluster. It manages the cluster's overall state, ensuring that the desired state (as defined by the user) matches the actual state of the cluster. The master node components take care of all the operational and administrative tasks.

Role of Worker Node:

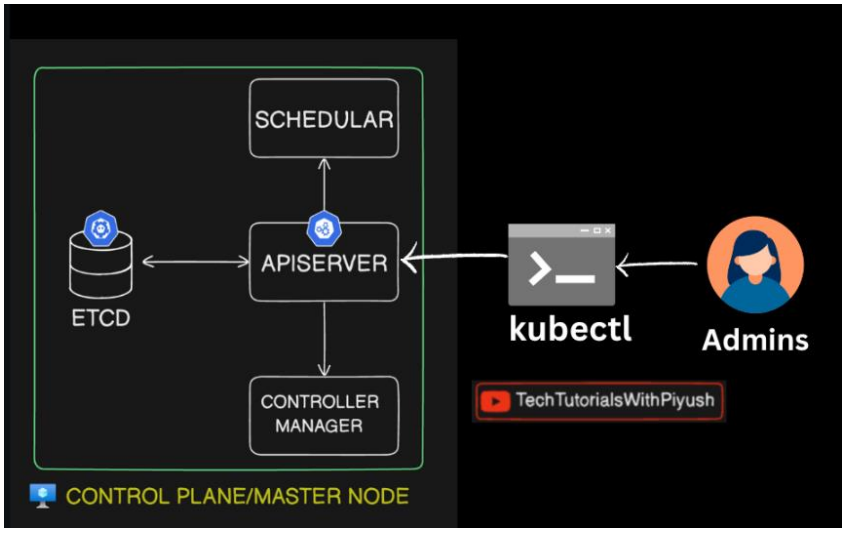
Worker Nodes are the machines where the actual application workloads (pods) run. They are responsible for running the containers and ensuring that the application remains functional.



Components of Master Node

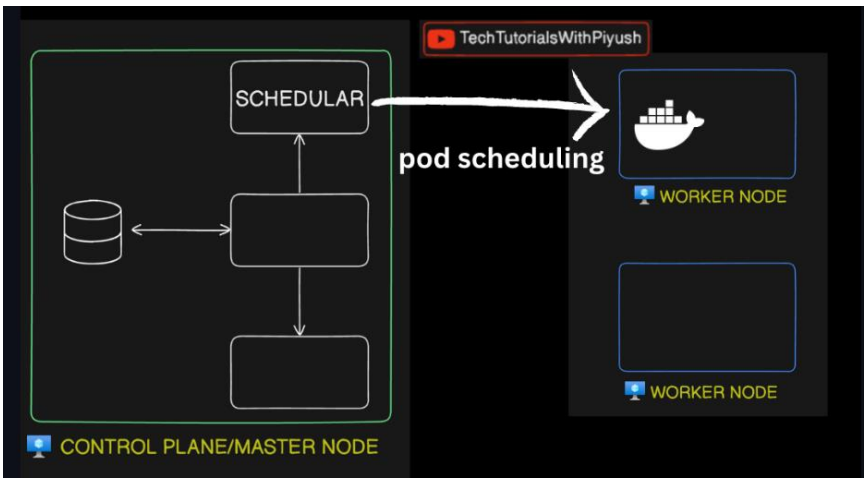
1. API Server

The Kubernetes API server is a component of the Kubernetes control plane that exposes the Kubernetes API, which is used by all other components of Kubernetes (such as kubectl, the CLI tool) to interact with the cluster. It acts as the front end for the Kubernetes control plane and client interacts with the cluster using ApiServer. It responsible for validating and processing API requests, maintaining the desired state of the cluster, and handling API resources such as pods, services, replication controllers, and others.



2. Scheduler

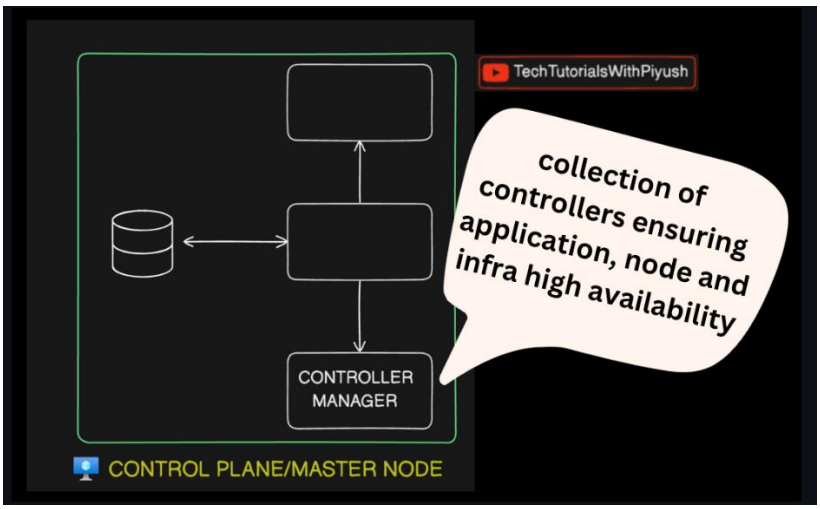
The Scheduler in Kubernetes is a component responsible for scheduling workloads (such as pods) onto nodes in the cluster. It watches for newly created pods with no assigned node, selects an appropriate node for each pod, and then binds the pod to that node. The scheduler considers factors such as resource requirements, hardware/software constraints, affinity and anti-affinity specifications, data locality, and other policies defined by the user or cluster administrator when making scheduling decisions.



3. Controller Manager

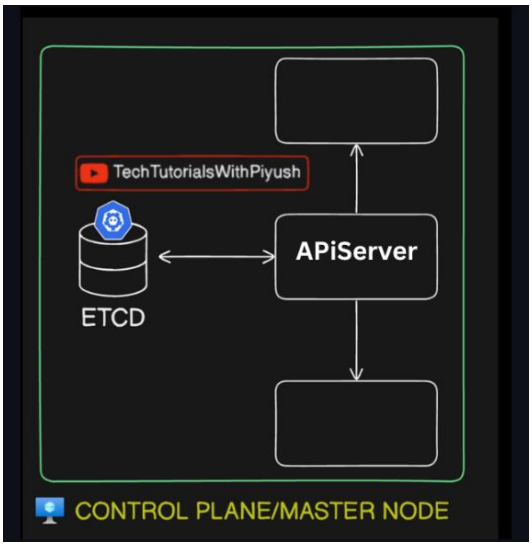
The Controller Manager in Kubernetes is a component of the control plane that manages different types of controllers to regulate the state of the cluster and perform cluster-wide tasks. Each controller in the Controller Manager is responsible for managing a specific aspect of the cluster's desired state, such as ReplicaSetController, DeploymentController, NamespaceController, and others.

These controllers continuously work to ensure that the current state of the cluster matches the desired state specified by users or applications. They monitor the cluster state through the Kubernetes API server, detect any differences between the current and desired states, and take corrective actions to reconcile them, such as creating or deleting resources as needed.



4. ETCD Server

Etcd is a distributed key-value storage that is used as the primary datastore in Kubernetes for storing cluster state and configuration information. It is a critical component of the Kubernetes control plane and is responsible for storing information such as cluster configuration, the state of all Kubernetes objects (such as pods, services, and replication controllers), and information about nodes in the cluster. Etcd ensures consistency and reliability by using a distributed consensus algorithm to replicate data across multiple nodes in the etcd cluster.



5. Cloud Control Manager (Optional)

The "Cloud Control Manager" is a component in the Kubernetes ecosystem that is part of the Cloud Provider Interface (CPI). It is responsible for managing interactions between Kubernetes and the underlying cloud provider's services and resources.

The Cloud Control Manager facilitates functionalities such as:

1. **Node management:** It interacts with the cloud provider's APIs to manage the lifecycle of nodes in the cluster, including creating, deleting, and updating nodes
2. **Load balancer management:** It manages the creation and configuration of load balancers provided by the cloud provider for Kubernetes services.
3. **Volume management:** It handles the provisioning and management of storage volumes (e.g., EBS volumes on AWS, persistent disks on GCP) used by Kubernetes pods.
4. **Networking:** It manages the networking configuration, including setting up routes, load balancers, and firewall rules, to ensure that pods can communicate with each other and with external services.

The Cloud Control Manager abstracts the cloud-specific details from the core Kubernetes components, allowing Kubernetes to be used across different cloud providers without requiring changes to the core Kubernetes codebase.

Components of Worker Node

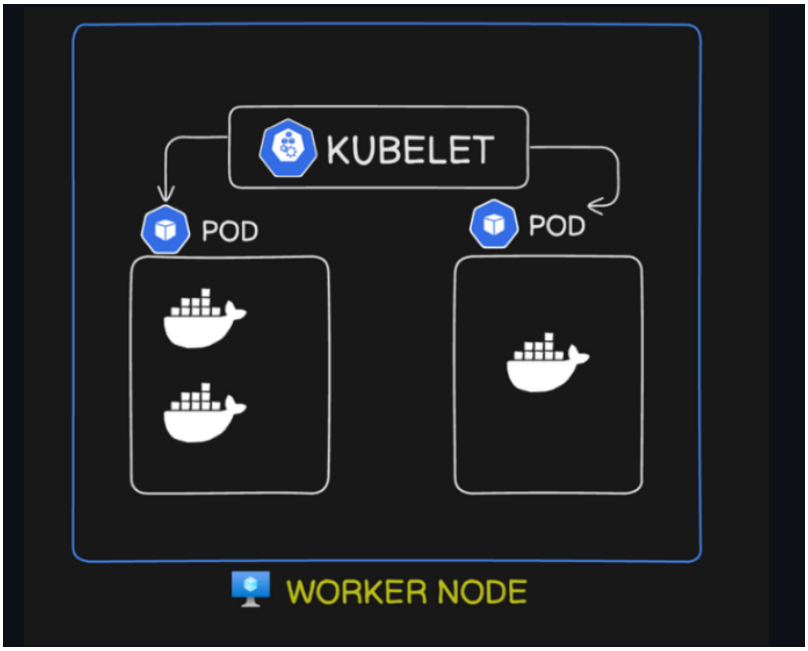
1. Kubelet

In Kubernetes, Kubelet is the primary node agent that runs on each node in the cluster. It is responsible for managing the containers running on the node and ensuring that they are healthy and running as expected.

Some of the key responsibilities of Kubelet include:

1. **Pod Lifecycle Management:** Kubelet is responsible for starting, stopping, and maintaining containers within a pod as directed by the Kubernetes API server.
2. **Node Monitoring:** Kubelet monitors the health of the node and reports back to the Kubernetes control plane. If the node becomes unhealthy, the control plane can take corrective actions, such as rescheduling pods to other healthy nodes.
3. **Resource Management:** Kubelet manages the node's resources (CPU, memory, disk, etc.) and enforces resource limits and requests specified in pod configurations.
4. **Networking:** Kubelet manages the network setup for pods on the node, including setting up networking rules, IP addresses, and ensuring that pods can communicate with each other and the outside world.
5. **Volume Management:** Kubelet manages pod volumes, including mounting and unmounting volumes as specified in the pod configuration.

Overall, Kubelet plays a crucial role in ensuring that pods are running correctly on each node in the Kubernetes cluster and that the cluster remains healthy and operational.



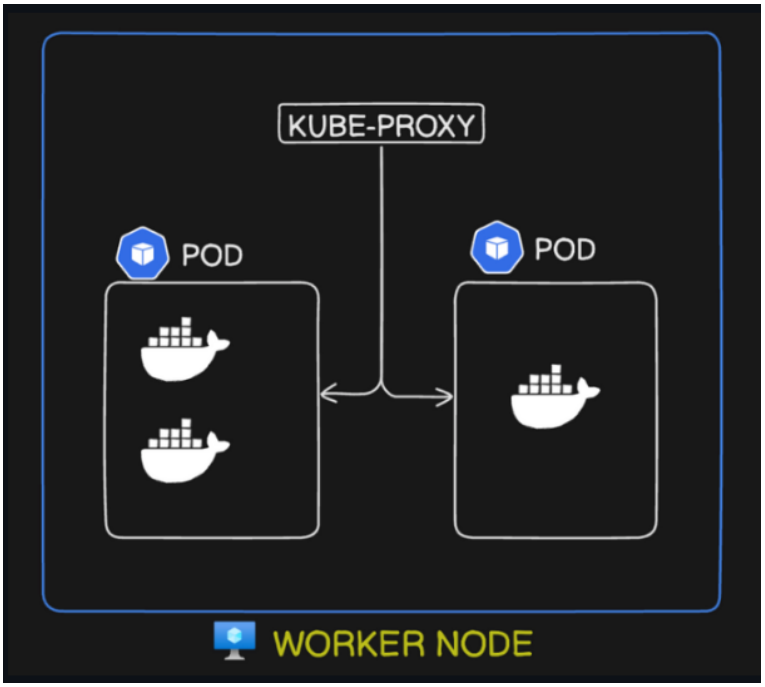
2. Kube-Proxy

In Kubernetes, kube-proxy is a network proxy that runs on each node in the cluster. It is responsible for implementing part of the Kubernetes Service concept, which enables network communication to your Pods from network clients inside or outside of your cluster. kube-proxy maintains network rules on each node. These network rules allow network communication to be forwarded to the appropriate Pod based on IP address and port number. kube-proxy operates at both the TCP and UDP levels.

There are several modes in which kube-proxy can operate, including:

1. **User space mode:** In this mode, kube-proxy opens a port on the node's IP address. When traffic is received on this port, kube-proxy reads the destination IP address and port from the packet and forwards it to the appropriate Pod. This mode is relatively simple but can be inefficient for high traffic loads.
2. **iptables mode:** In this mode, kube-proxy installs iptables rules on the node to forward traffic to the appropriate Pod. This mode is more efficient than user space mode and is the default mode for kube-proxy.
3. **IPVS mode:** IPVS (IP Virtual Server) is an advanced method of load balancing in the Linux kernel. In this mode, kube-proxy uses IPVS to perform load balancing for Services with type=LoadBalancer or type=NodePort. This mode can provide better performance and scalability compared to iptables mode.

Overall, kube-proxy plays a critical role in enabling network communication to your Pods in a Kubernetes cluster and is essential for the functioning of Kubernetes Services.



3. Container Runtime

In Kubernetes, a container runtime is the software responsible for running containers. It is an essential component of the Kubernetes architecture because Kubernetes itself does not run containers directly; instead, it relies on a container runtime to do so.

The container runtime is responsible for:

1. Pulling container images from a container registry (e.g., Docker Hub, Azure Container Registry).
2. Creating and managing container lifecycle (start, stop, pause, delete).
3. Managing container networking and storage.
4. Providing container isolation and resource constraints.

Some popular container runtimes used with Kubernetes include:

1. **Docker:** Docker was the original default container runtime for Kubernetes. It provides a comprehensive set of tools for building, managing, and running containers.
2. **Containerd:** Containerd is an industry-standard core container runtime that provides a lightweight and reliable platform for managing containers. Post version v1.24 containerd is a default container runtime for Kubernetes.
3. **CRI-O:** CRI-O is an implementation of the Kubernetes Container Runtime Interface (CRI) that is optimized for Kubernetes. It provides a minimalistic runtime focused on running containers according to the Kubernetes specifications.
4. **rkt (pronounced "rocket"):** rkt is a container runtime developed by CoreOS that focuses on security, simplicity, and composability. It is designed to be compatible with the Kubernetes CRI.

The choice of container runtime can impact factors such as performance, security, and manageability of your Kubernetes clusters.

Workloads in Kubernetes

1. Pods

Pods are the smallest deployable units of computing that you can create and manage in Kubernetes. A Pod (as in a pod of whales or pea pod) is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers. A Pod's contents are always co-located and co-scheduled, and run in a shared context.

Pods that run a **single container**: The "one-container-per-Pod" model is the most common Kubernetes use case; in this case, you can think of a Pod as a wrapper around a single container; Kubernetes manages Pods rather than managing the containers directly.

Pods that run **multiple containers** that need to work together: A Pod can encapsulate an application composed of multiple co-located containers that are tightly coupled and need to share resources. These co-located containers form a single cohesive unit. **Multi Container** includes initcontainer and sidecar/helper container.

2. Static Pods

Static Pods are special types of pods managed directly by the kubelet on each node rather than through the Kubernetes API server.

Key Characteristics of Static Pods:

- Not Managed by the Scheduler: Unlike deployments or replicasets, the Kubernetes scheduler does not manage static pods.
- Defined on the Node: Configuration files for static pods are placed directly on the node's file system, and the kubelet watches these files.
- Some examples of static pods are: ApiServer, Kube-scheduler, controller-manager, ETCD etc

3. ReplicationController

ReplicationController was one of the original Kubernetes controllers and is now considered deprecated in favour of ReplicaSet.

4. ReplicaSet

A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods. ReplicaSet is the newer version of ReplicationController and was introduced in Kubernetes version 1.2 as part of the move to the apps/v1 API group. Primarily used by Deployments to manage the underlying pods. Direct use of ReplicaSets is uncommon, as Deployments provide additional features like rolling updates.

5. Deployments

A Deployment provides declarative updates for Pods and ReplicaSets. You describe a desired state in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

6. StatefulSets

StatefulSet is the workload API object used to manage stateful applications. Manages the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods. Like a Deployment, a StatefulSet manages Pods that are based on an identical container spec. Unlike a Deployment, a StatefulSet maintains a sticky identity for each of its Pods. These pods are created from the same spec, but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling.

If you want to use storage volumes to provide persistence for your workload, you can use a StatefulSet as part of the solution. Although individual Pods in a StatefulSet are susceptible to failure, the persistent Pod identifiers make it easier to match existing volumes to the new Pods that replace any that have failed

7. DeamonSet

DeamonSet ensures that the number of replicas is equal to number of nodes and each node has one running replica at a time. If you create a deamonset in a cluster of 5 nodes, then 5 pods will be created. It is used for

1. Monitoring Agents
2. Logging events
3. Networking CNIs

8. Jobs

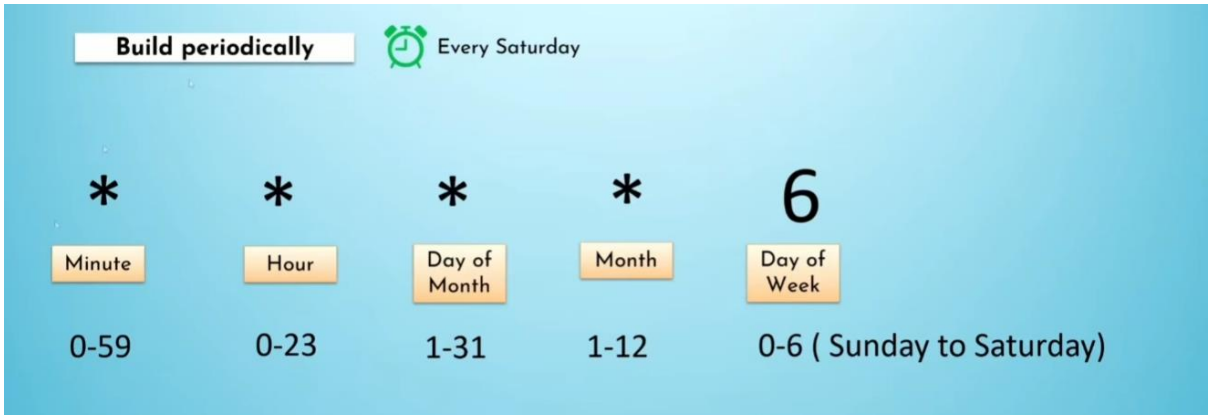
A Job creates one or more Pods and will continue to retry execution of the Pods until a specified number of them successfully terminate. As pods successfully complete, the Job tracks the successful completions. When a specified number of successful completions is reached, the task (ie, Job) is complete. Deleting a Job will clean up the Pods it created. Suspending a Job will delete its active Pods until the Job is resumed again.

A simple case is to create one Job object in order to reliably run one Pod to completion. The Job object will start a new Pod if the first Pod fails or is deleted (for example due to a node hardware failure or a node reboot). You can also use a Job to run multiple Pods in parallel.

9. Cronjobs

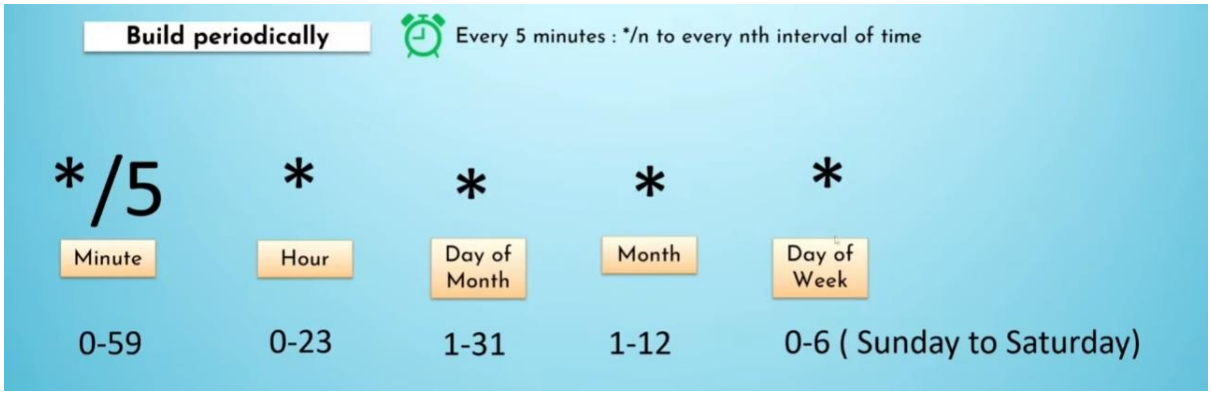
A CronJob creates Jobs on a repeating schedule. CronJob is meant for performing regular scheduled actions such as backups, report generation, and so on. One CronJob object is like one line of a crontab (cron table) file on a Unix system. It runs a Job periodically on a given schedule, written in Cron format. CronJobs have limitations and idiosyncrasies. For example, in certain circumstances, a single CronJob can create multiple concurrent Jobs.

Cron Format:


Build periodically
Every Saturday
*
*
*
*
6
Minute
Hour
Day of Month
Month
Day of Week
0-59
0-23
1-31
1-12
0-6 (Sunday to Saturday)


Build periodically
Every Saturday at 11 PM
*
23
*
*
6
Minute
Hour
Day of Month
Month
Day of Week
0-59
0-23
1-31
1-12
0-6 (Sunday to Saturday)


Build periodically
Every Saturday at 11:45 PM
45
23
*
*
6
Minute
Hour
Day of Month
Month
Day of Week
0-59
0-23
1-31
1-12
0-6 (Sunday to Saturday)


Build periodically
Every 5 minutes : */n to every nth interval of time
*/5
*
*
*
*
Minute
Hour
Day of Month
Month
Day of Week
0-59
0-23
1-31
1-12
0-6 (Sunday to Saturday)

ConfigMap and Secrets

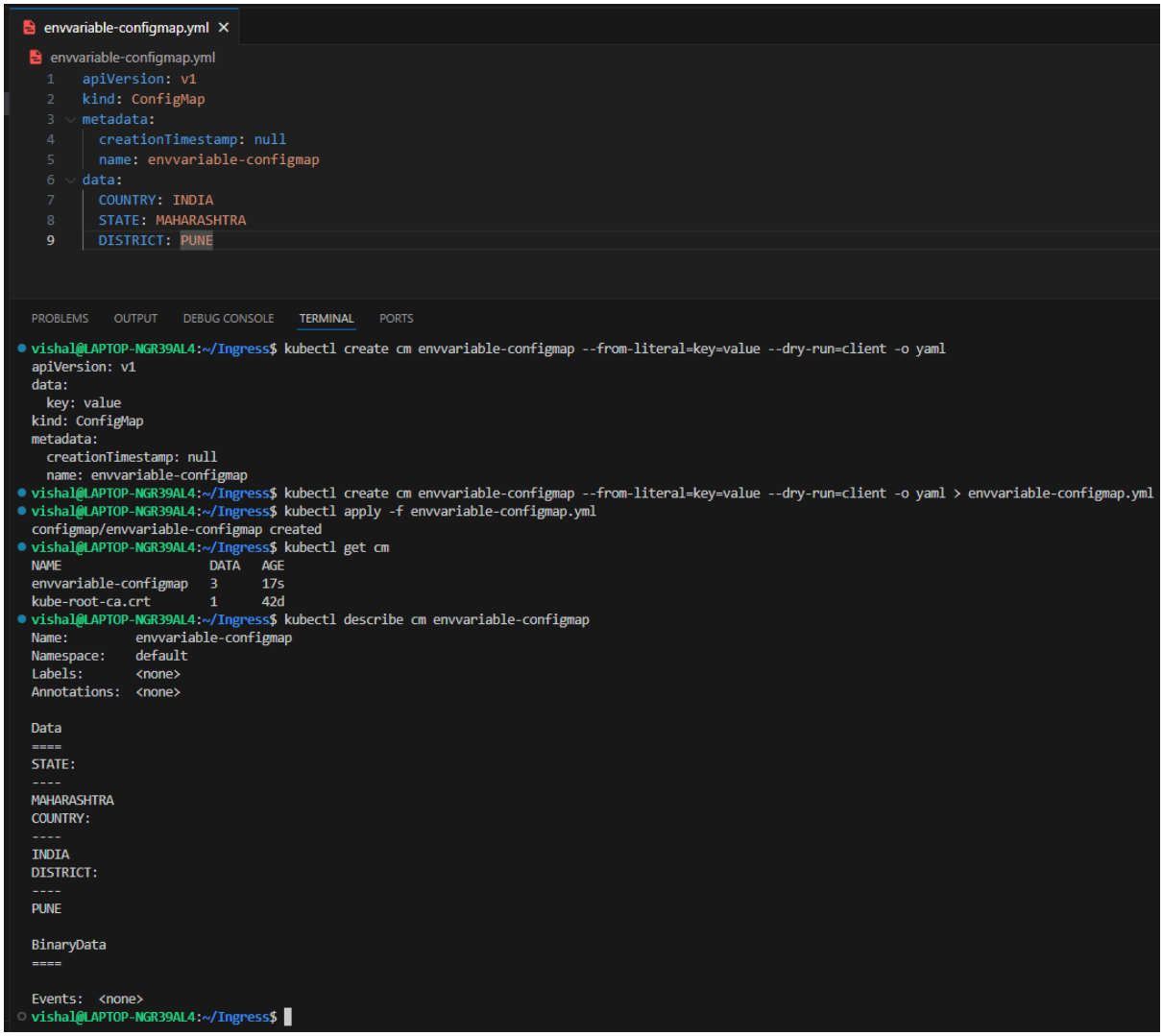
1. ConfigMap

A ConfigMap is an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume. A ConfigMap allows you to decouple environment-specific configuration from your container images, so that your applications are easily portable.

- When your manifest grows it becomes difficult to manage multiple env vars
- You can take this out of the manifest and store as a config map object in the key-value pair
- Then you can inject that config map into the pod
- You can reuse the same config map into multiple pods

Command to create a configmap

➔ kubectl create cm <configmapname> --from-literal=color=blue



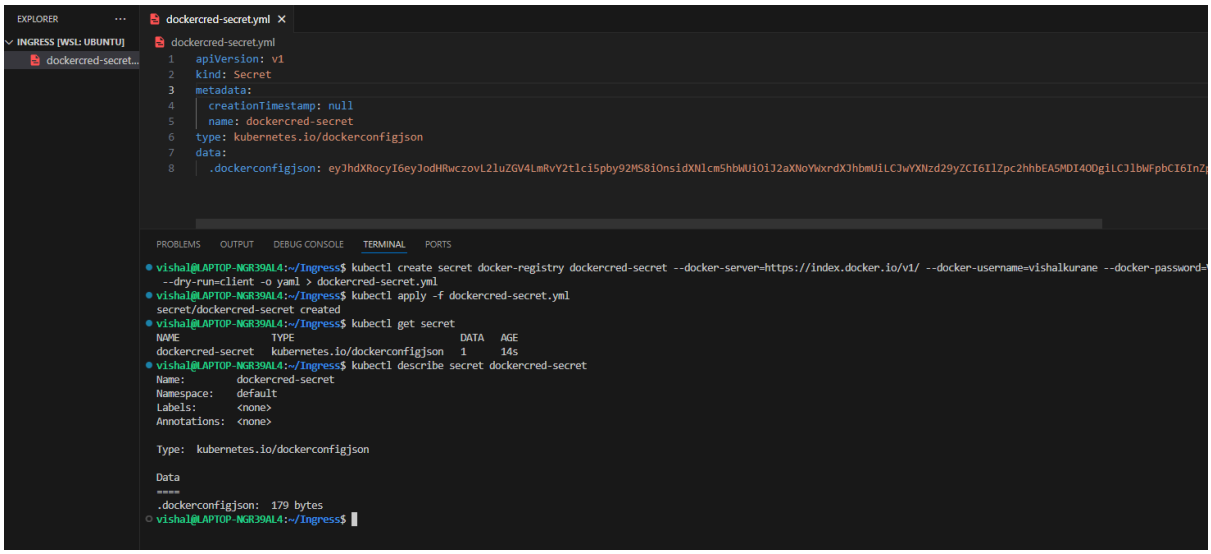
2. Secrets

A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a Pod specification or in a container image. Using a Secret means that you don't need to include confidential data in your application code.

Because Secrets can be created independently of the Pods that use them, there is less risk of the Secret (and its data) being exposed during the workflow of creating, viewing, and editing Pods. Kubernetes, and applications that run in your cluster, can also take additional precautions with Secrets, such as avoiding writing sensitive data to non-volatile storage. Secrets are similar to ConfigMaps but are specifically intended to hold confidential data.

Create Secret for Docker Credentials using imperative command

➔ kubectl create secret docker-registry dockercred-secret --docker-server=https://index.docker.io/v1/ --docker-username=vishalkurane --docker-password=<Password> --docker-email=<email>

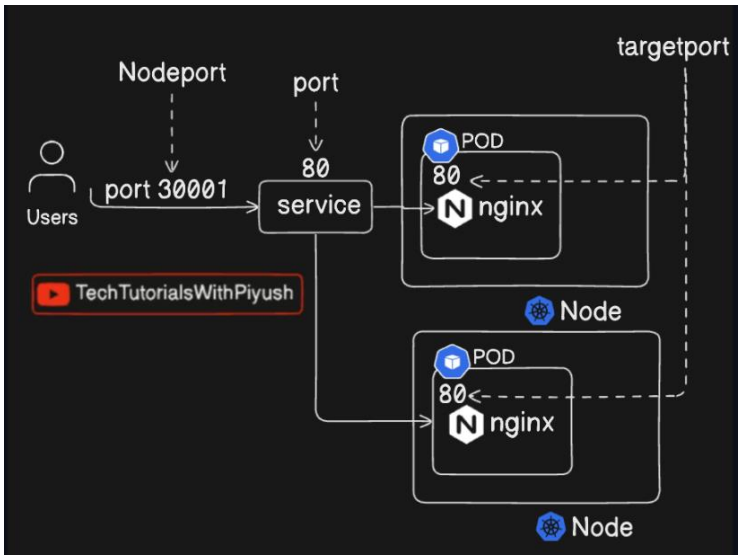


Services

1. **NodePort**

To access the application externally on a particular Node Port

- **Node Port:** Node Port is a port of service which is exposed to the external world. Range of Nodeport- 30000 – 32767
- **Internal Service Port:** Service will be expose internally within the cluster through internal service port. (e.g.: 80)
- **Target Port:** Target port is a port on which the application is listening. (e.g.: 80)



Explanation:

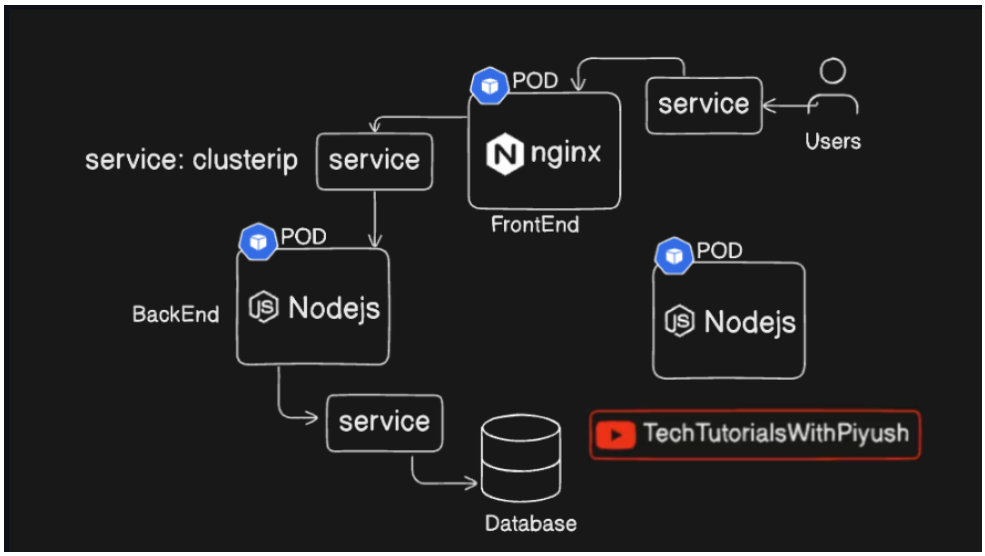
- **NodePort** exposes the service on a specific port on each node in the cluster.
- This allows external traffic to access the service by sending requests to the <NodeIP>:<NodePort>.
- The port range for NodePort services is typically between 30000-32767.

Use Case Scenario:

- **Direct External Access**: If you want to expose a service to be accessible from outside the cluster but don't have a load balancer, you can use NodePort. For example, if you have a web application running in a Kubernetes cluster and you want to test it externally, you might use NodePort to expose it.

2. **ClusterIP**

Used to expose and access the service internally within the cluster.



Explanation:

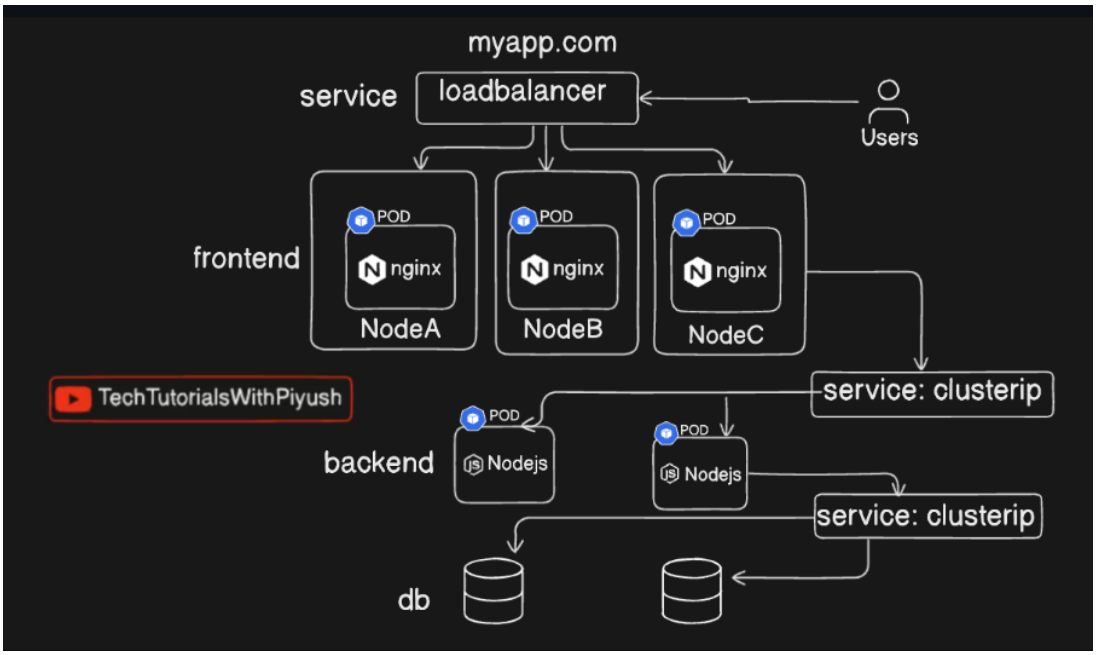
- **ClusterIP** is the default service type in Kubernetes.
- It makes the service accessible only within the Kubernetes cluster, using an internal IP address.
- Other services within the cluster can use this IP to communicate with the service.

Use Case Scenario:

- **Internal Microservices Communication**: If you have a set of microservices within a Kubernetes cluster that need to communicate with each other (e.g., a backend service accessing a database service), you would use a ClusterIP service. This keeps the communication internal and secure without exposing the services externally.

3. LoadBalancer

Your loadbalancer service will act as nodeport if you are not using any managed cloud Kubernetes such as GKE, AKS, EKS etc. In a managed cloud environment, Kubernetes creates a load balancer within the cloud project, which redirects the traffic to the Kubernetes Loadbalancer service.



Explanation:

- **LoadBalancer** creates an external load balancer (if supported by the cloud provider) that forwards traffic to your service.
- This is the easiest way to expose a service to the internet when running Kubernetes on a cloud provider like AWS, Azure, or GCP.
- It automatically provisions a load balancer and assigns a public IP to your service.

Use Case Scenario:

- **Publicly Accessible Applications**: If you're deploying a production application that needs to be accessible to the public, like a web application or API, you would use a LoadBalancer service. For example, a public-facing e-commerce website running in Kubernetes could be exposed using a LoadBalancer service.

4. **ExternalName**

Explanation:

- **ExternalName** maps a Kubernetes service to a DNS name outside the cluster.
- Instead of proxying traffic, it simply returns a CNAME record with the value of the external name specified.
- No IP is assigned to the service, and no proxying of traffic is involved.

Use Case Scenario:

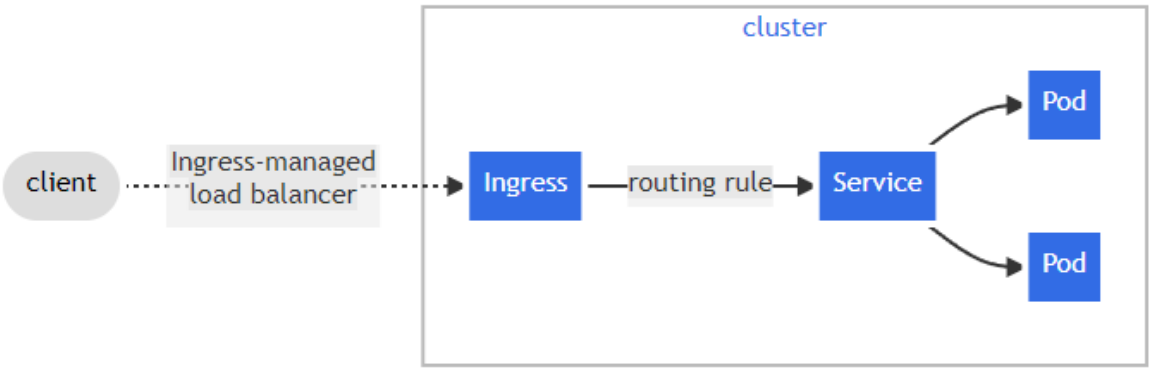
- **External Service Aliasing**: If you want to refer to an external service using a consistent name within your Kubernetes cluster, you can use ExternalName. For example, if your application needs to connect to a third-party API like api.example.com, you can create a Kubernetes service with an ExternalName that maps to api.example.com. This way, internal applications can use the Kubernetes service name, and any changes to the external service's URL can be managed centrally.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
  namespace: prod
spec:
  type: ExternalName
  externalName: my.database.example.com
```

Ingress and Ingress Controller

1. Ingress

Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.



- In Kubernetes, **Ingress** is an API object that manages external access to services within a cluster, typically HTTP and HTTPS routes.
- Ingress provides a way to define rules for routing external HTTP(S) traffic to different services within the cluster based on the request's path or host.
- Ingress can also provide other features like SSL termination, name-based virtual hosting, and load balancing.

Components of Ingress:

- **Ingress Resource**: The YAML/JSON configuration file where you define the routing rules for directing external traffic to the services.
- **Ingress Controller**: The component that implements the Ingress resource. It watches the Ingress resources and updates its configuration to manage routing.

Routing Rules:

In Kubernetes Ingress, rules for routing external traffic to internal services can be defined in several ways. Below is a list of the most common methods:

1. **Host-Based Routing**
   - Routes traffic based on the Host header in the incoming request.
   - Useful for hosting multiple applications on the same IP address but under different domain names or subdomains.

1. **Path-Based Routing**
   - Routes traffic based on the URL path in the incoming request.
   - Useful for directing different paths under the same domain to different services.

2. **Path Type**
   - Specifies how the Ingress controller should match the URL path with the request.
   - Kubernetes supports the following path Type values:
     - **Prefix**: Matches based on a URL path prefix. Requests are routed if the beginning of the path matches.
     - **Exact**: Matches the URL path exactly as specified.
     - **ImplementationSpecific**: Allows the Ingress controller to determine how the path should be matched.

3. **Regex-Based Routing (Depending on Ingress Controller)**
   - Some Ingress controllers support custom regex-based routing for more complex routing scenarios.
   - Useful for matching more complex URL patterns.

4. **TLS/SSL Termination**
   - You can define rules to handle HTTPS traffic, terminating SSL at the Ingress controller.
   - You provide a TLS certificate that the Ingress controller uses to serve HTTPS requests.

5. **Default Backend**
   - A default backend is a catch-all service for requests that do not match any specified Ingress rules.
   - Useful for handling invalid requests, providing custom 404 pages, or default routing behaviour.

6. **Annotations for Custom Behaviour**
   - Ingress resources can use annotations to define custom behaviours like URL rewriting, redirects, and more.
   - These are controller-specific and add flexibility for advanced use cases.

2. Ingress Controller

- The **Ingress Controller** is responsible for fulfilling the Ingress resource by configuring a load balancer or proxy to handle traffic according to the Ingress rules.
- It runs as a pod in the Kubernetes cluster and watches for changes in Ingress resources, updating its configuration dynamically.
- There are various Ingress Controllers available, such as **NGINX Ingress Controller**, **Traefik**, **HAProxy**, **AWS ALB Ingress Controller**, and others.

How It Works:

- When you create an Ingress resource, the Ingress Controller automatically configures itself to route traffic according to the rules specified.
- It can manage tasks like:
    - Routing traffic based on URL paths or hostnames.
    - Terminating SSL/TLS traffic.
    - Performing load balancing.
    - Redirecting traffic (e.g., HTTP to HTTPS).

Use Case Scenario

Scenario: Imagine you're hosting multiple microservices in a Kubernetes cluster, each providing different functionalities of an e-commerce application—such as frontend, inventory, and payment. You want to expose these services to the public under a single domain name with different paths.

- **Without Ingress**:
    - You might expose each service individually using a LoadBalancer service, leading to multiple public IPs, one for each service.
    - Users would need to remember different IP addresses or subdomains to access different parts of the application.
- **With Ingress**:
    - You can define an Ingress resource that routes traffic based on paths:
        - example.com/ -> frontend-service
        - example.com/inventory -> inventory-service
        - example.com/payment -> payment-service
    - This way, users access different services via the same domain but different paths, making the application more user-friendly.
    - You can also terminate SSL traffic at the Ingress level, providing HTTPS for all services with a single SSL certificate.

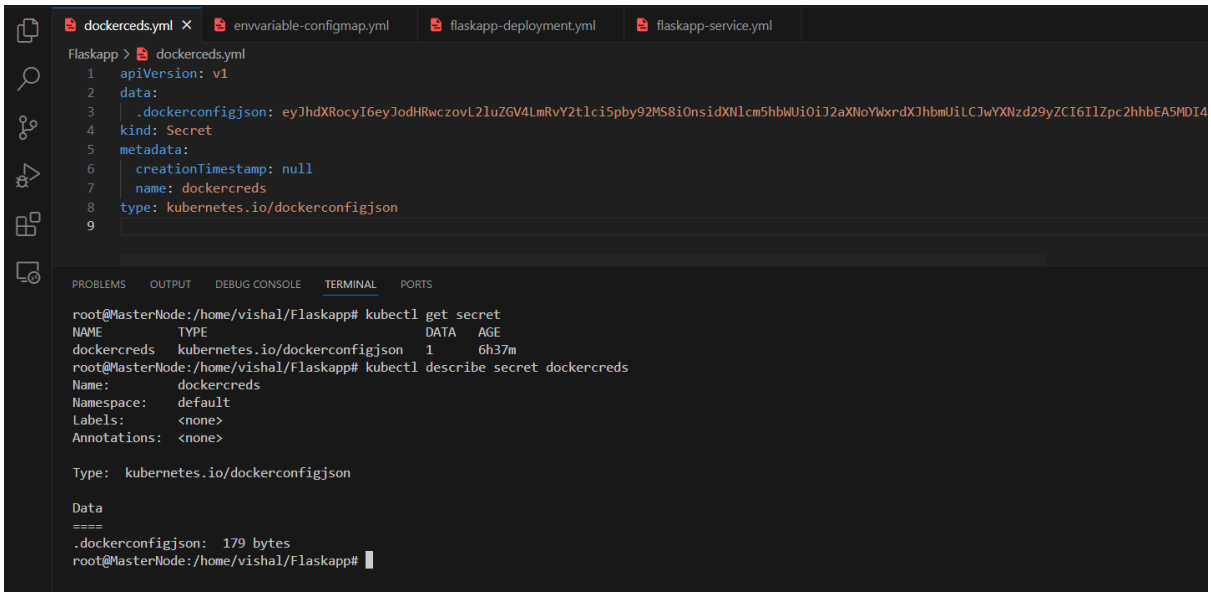
Benefits of Using Ingress:

- **Consolidation**: Centralized routing and SSL termination, reducing the need for multiple LoadBalancers.
- **Ease of Management**: Simplifies managing access to multiple services with one or a few Ingress resources.
- **Cost Efficiency**: Reduces cloud costs by minimizing the number of public IP addresses needed.
- **Enhanced Security**: SSL termination and redirection rules help in enforcing secure communication protocols.

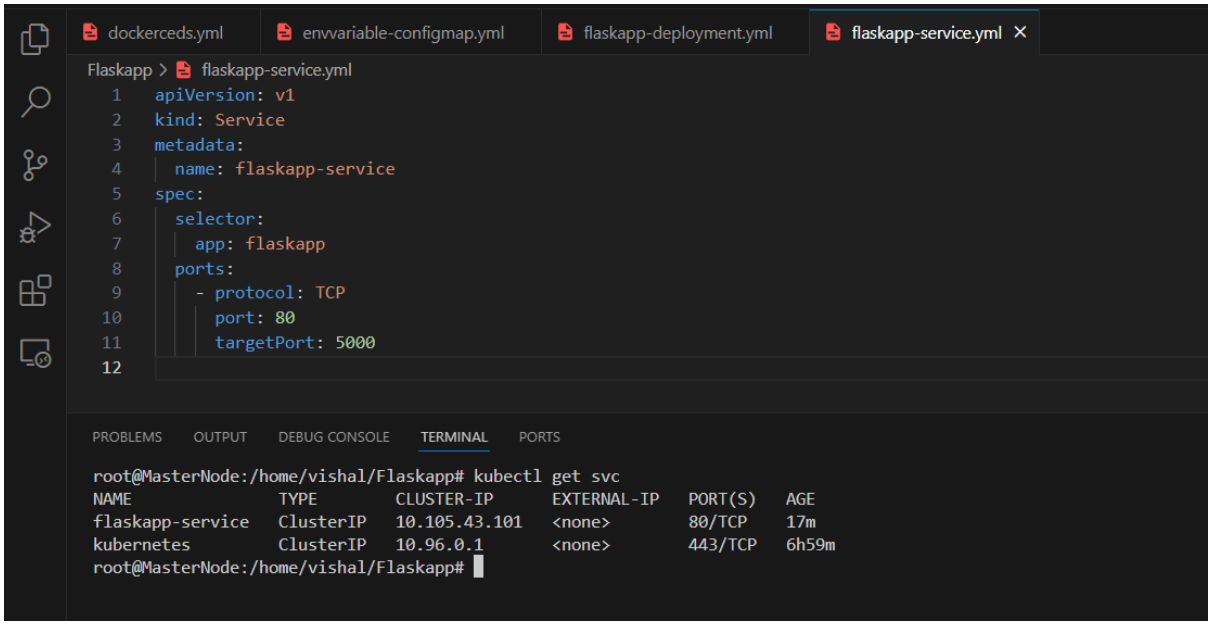
Project: Deploy a Flask application with and without Ingress

1. Create Image pull secret from private Docker registry

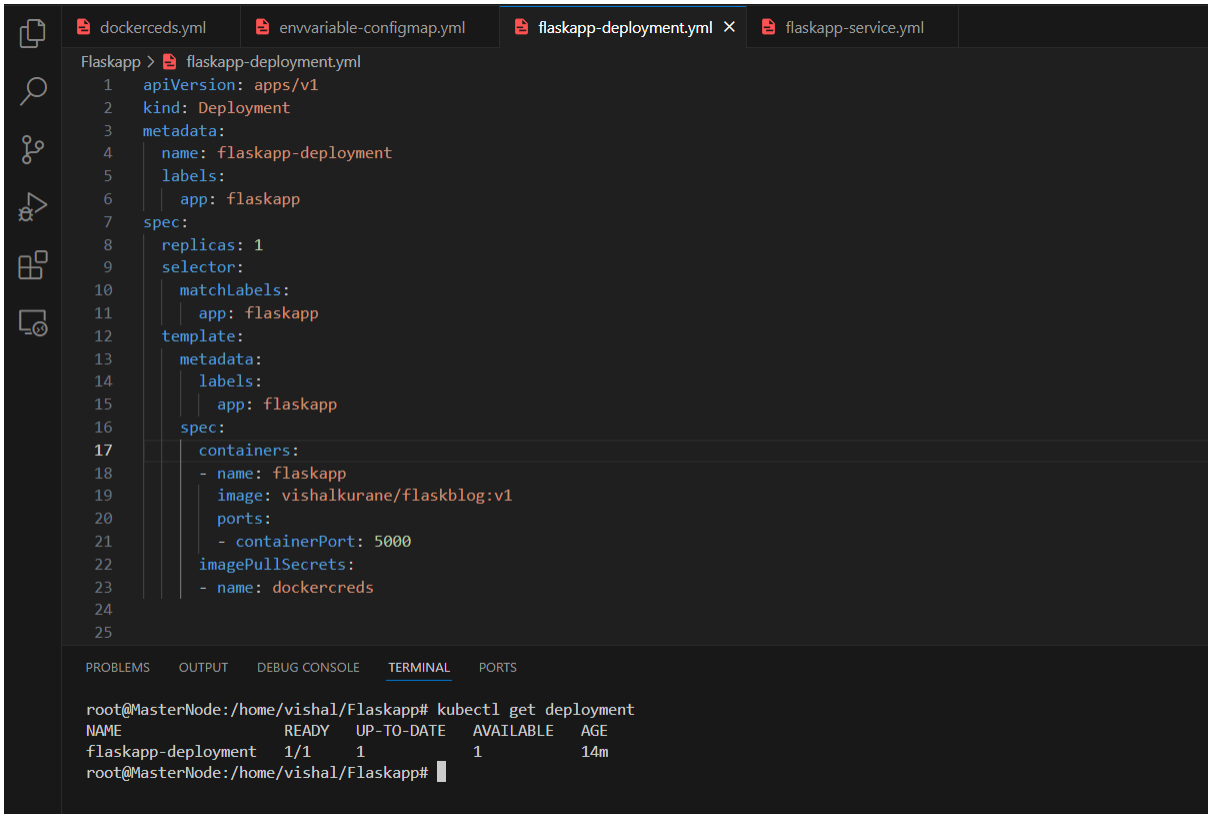
➔ kubectl create secret docker-registry dockercred-secret --docker-server=https://index.docker.io/v1/ --docker-username=vishalkurane --docker-password=<Password> --docker-email=<email>



2. Create a Service of type ClusterIP



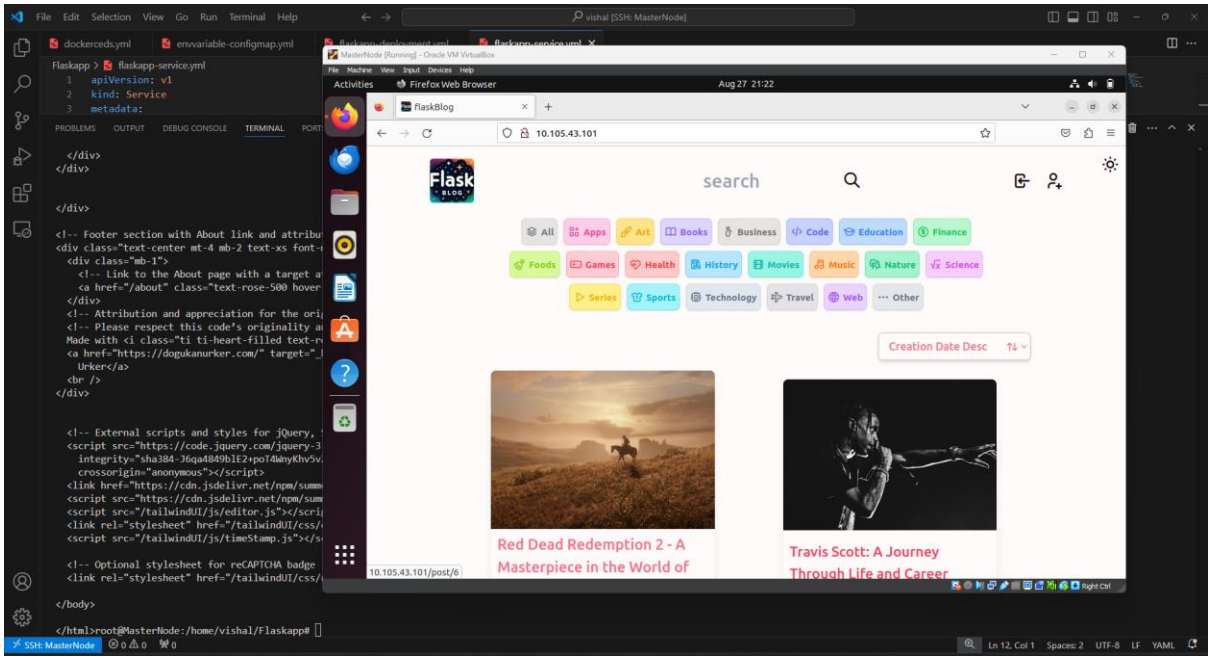
3. Create Deployment



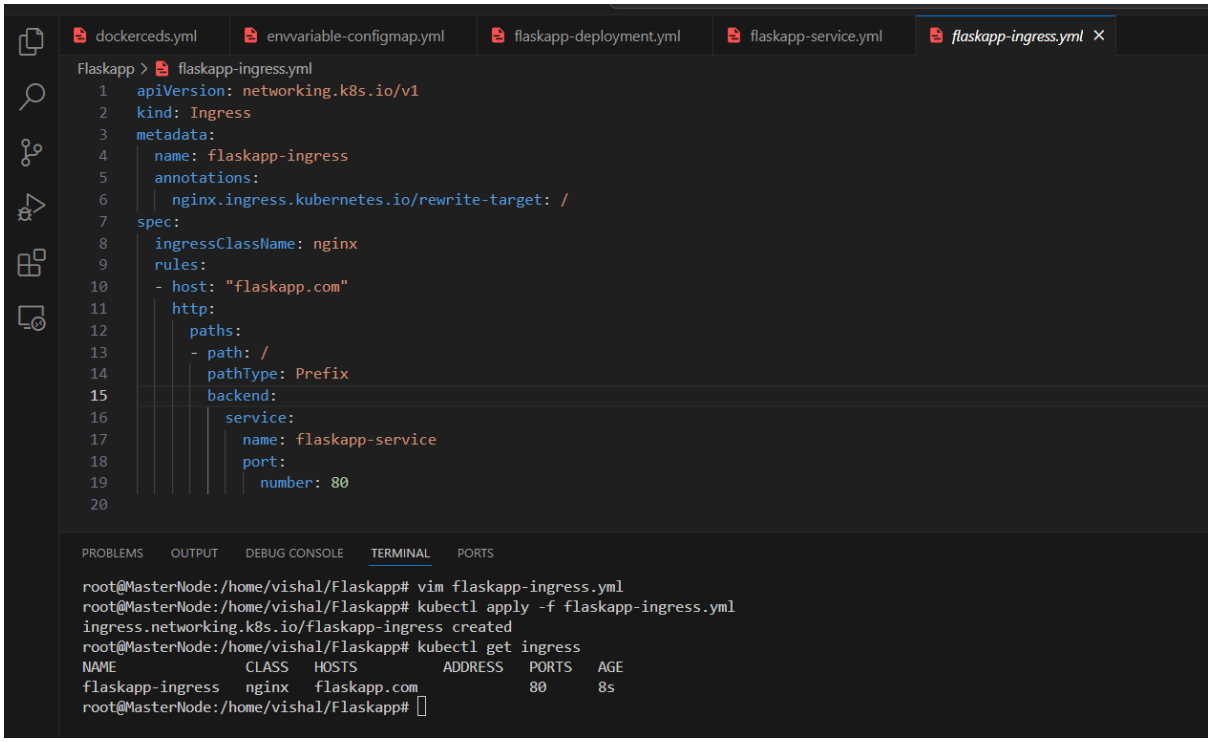
4. Check weather service is able to communicate with pod

➔ Curl <ClusterIP>

```
root@MasterNode:/home/vishal/Flaskapp# kubectl get svc
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
flaskapp-service   ClusterIP   10.105.43.101   <none>        80/TCP    19m
kubernetes         ClusterIP   10.96.0.1       <none>        443/TCP   7h1m
root@MasterNode:/home/vishal/Flaskapp# curl 10.105.43.101
```

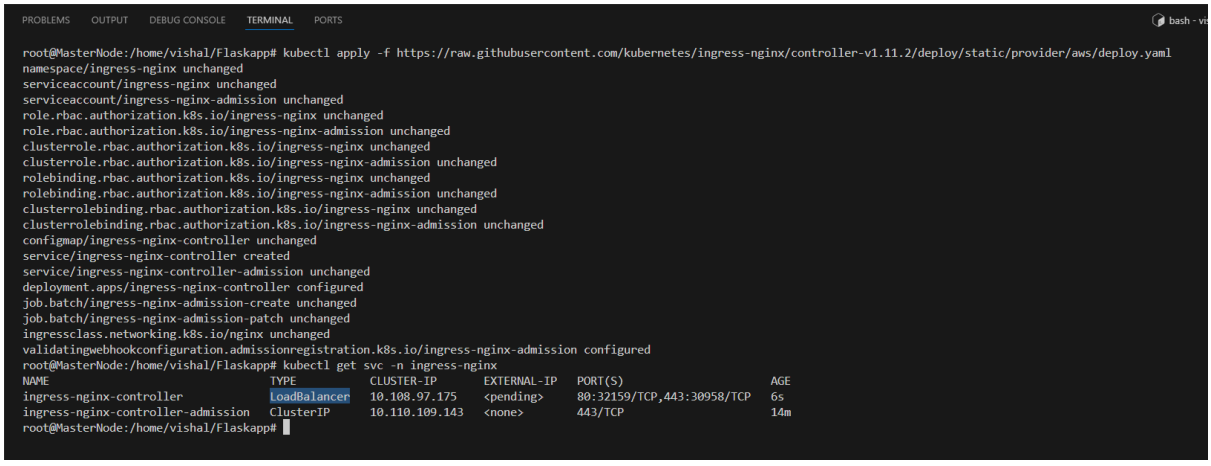


5. Create an Ingress resource



6. Create an Ingress Controller

https://github.com/kubernetes/ingress-nginx?tab=readme-ov-file
https://kubernetes.github.io/ingress-nginx/deploy/
Ingress Controller for AWS:
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.11.2/deploy/static/provider/aws/deploy.yaml



Now we have the service ‘ingress-nginx-controller’ of type LoadBalancer. Edit it to NodePort

```
  selector:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer: {}
```

```
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
-- INSERT --
```

```
root@MasterNode:/home/vishal/Flaskapp# kubectl get svc -n ingress-nginx
NAME                                 TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
ingress-nginx-controller             LoadBalancer   10.108.97.175    <pending>     80:32159/TCP,443:30958/TCP   6s
ingress-nginx-controller-admission   ClusterIP      10.110.109.143   <none>        443/TCP                      14m
root@MasterNode:/home/vishal/Flaskapp# kubectl edit svc ingress-nginx-controller -n ingress-nginx
service/ingress-nginx-controller edited
root@MasterNode:/home/vishal/Flaskapp# kubectl get svc -n ingress-nginx
NAME                                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
ingress-nginx-controller             NodePort    10.108.97.175    <none>        80:32159/TCP,443:30958/TCP   4m15s
ingress-nginx-controller-admission   ClusterIP   10.110.109.143   <none>        443/TCP                      19m
root@MasterNode:/home/vishal/Flaskapp#
```

Delete and recreate Ingress resource. IP Address is now generated

```
root@MasterNode:/home/vishal/Flaskapp# kubectl delete ingress flaskapp-ingress
ingress.networking.k8s.io "flaskapp-ingress" deleted
root@MasterNode:/home/vishal/Flaskapp# kubectl apply -f flaskapp-ingress.yml
ingress.networking.k8s.io/flaskapp-ingress created
root@MasterNode:/home/vishal/Flaskapp# kubectl get ingress
NAME               CLASS   HOSTS           ADDRESS   PORTS   AGE
flaskapp-ingress   nginx   flaskapp.com              80      3s
root@MasterNode:/home/vishal/Flaskapp# kubectl get ingress
NAME               CLASS   HOSTS           ADDRESS         PORTS   AGE
flaskapp-ingress   nginx   flaskapp.com    10.108.97.175   80      72s
root@MasterNode:/home/vishal/Flaskapp#
```

7. Using the curl command, map an IP address to the host which is specified in Ingress

curl flaskapp.com --resolve flaskapp.com:80:10.108.97.175

We can also update the /etc/host file to resolve the DNS internally

```
root@MasterNode:/home/vishal/Flaskapp# kubectl get ingress
NAME              CLASS   HOSTS          ADDRESS   PORTS   AGE
flaskapp-ingress  nginx   flaskapp.com             80      3s
root@MasterNode:/home/vishal/Flaskapp# kubectl get ingress
NAME              CLASS   HOSTS          ADDRESS         PORTS   AGE
flaskapp-ingress  nginx   flaskapp.com   10.108.97.175   80      72s
root@MasterNode:/home/vishal/Flaskapp# curl flaskapp.com --resolve flaskapp.com:80:10.108.97.175
```

dockerceds.yml | envvariable-configmap.yml | flaskapp-deployment.yml | flaskapp-service.yml | flaskapp-ingress.yml

Flaskapp > flaskapp-service.yml

```
1  apiVersion: v1
2  kind: Service
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
    </div>
  </div>
</div>


</div>

<!-- Footer section with About link and attribution -->
<div class="text-center mt-4 mb-2 text-xs font-medium">
  <div class="mb-1">
    <!-- Link to the About page with a target attribute to open in a new tab -->
    <a href="/about" class="text-rose-500 hover:text-rose-600 duration-150" target="_blank">About</a>
  </div>
  <!-- Attribution and appreciation for the original code -->
  <!-- Please respect this code's originality and credit the original repo. Thanks for your cooperation. 💜 -->
  Made with <i class="ti ti-heart-filled text-rose-500"></i> by
  <a href="https://dogukanurker.com/" target="_blank" class="text-rose-500 hover:text-rose-600 font-medium">Dogukan
    Urker</a>
  <br />
</div>


  <!-- External scripts and styles for jQuery, Summernote editor, timestamp, and optional reCAPTCHA badge -->
  <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"
    integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n"
    crossorigin="anonymous"></script>
  <link href="https://cdn.jsdelivr.net/npm/summernote@0.8.18/dist/summernote-lite.min.css" rel="stylesheet" />
  <script src="https://cdn.jsdelivr.net/npm/summernote@0.8.18/dist/summernote-lite.min.js"></script>
  <script src="/tailwindUI/js/editor.js"></script>
  <link rel="stylesheet" href="/tailwindUI/css/editor.css" />
  <script src="/tailwindUI/js/timeStamp.js"></script>

  <!-- Optional stylesheet for reCAPTCHA badge -->
  <link rel="stylesheet" href="/tailwindUI/css/recaptcha.css" />

</body>

</html>root@MasterNode:/home/vishal/Flaskapp#
```

Taints, Toleration and Node Affinity

1. Taints:

We provide taint on node. A taint marks a node with a specific characteristic, such as "gpu=true". By default, pods cannot be scheduled on tainted nodes unless they have a special permission called toleration. When a toleration on a pod matches with the taint on the node then only that pod will be scheduled on that node.

2. Toleration:

We provide toleration on pod. Toleration allows a pod to say, "Hey, I can handle that taint. Schedule me anyway!" You define tolerations in the pod specification to let them bypass the taints.

Effects of Taints and Tolerance

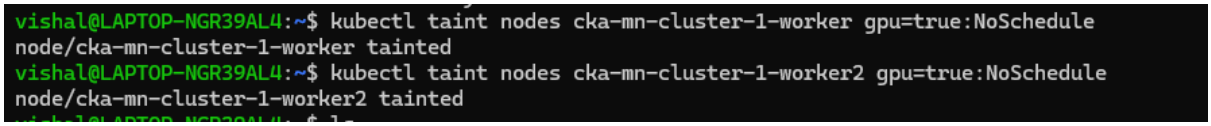
1. NoSchedule (Newer Pods)
2. PreferNoSchedue (No Guaranty)
3. NoExecution (Existing/Newer Pods)

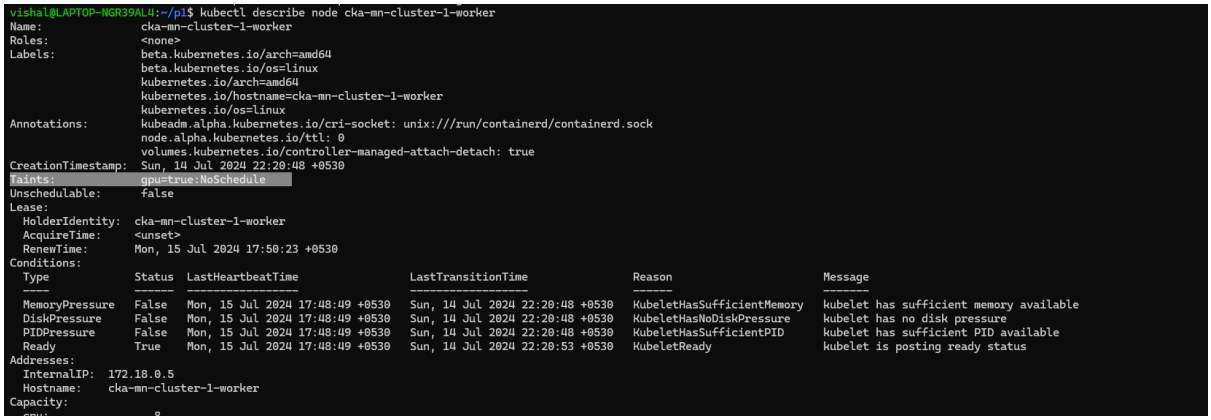
Taint a node using below command:

➔ kubectl taint nodes cka-mn-cluster-1-worker gpu=true:NoSchedule

Remove taint using – at the end of the command:

➔ kubectl taint nodes cka-mn-cluster-1-worker gpu=true:NoSchedule-





Taints applied at node level, Tolerations at pod level - This gives node ability to allow which pods to be scheduled on them. (Node centric approach)

```
vishal@LAPTOP-NGR39AL4:~/p1$ kubectl apply -f testpod.yaml
pod/nginx created
vishal@LAPTOP-NGR39AL4:~/p1$ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   0/1     Pending   0          9s
vishal@LAPTOP-NGR39AL4:~/p1$ kubectl describe pod nginx
Name:             nginx
Namespace:        default
Priority:         0
Service Account:  default
Node:             <none>
Labels:           run=nginx
Annotations:      <none>
Status:           Pending
IP:
IPs:              <none>
Containers:
  nginx:
    Image:        nginx
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-m469l (ro)
Conditions:
  Type           Status
  PodScheduled   False
Volumes:
  kube-api-access-m469l:
    Type:                     Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:   3607
    ConfigMapName:            kube-root-ca.crt
    ConfigMapOptional:        <nil>
    DownwardAPI:              true
QoS Class:                    BestEffort
Node-Selectors:               <none>
Tolerations:                  node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                              node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type     Reason            Age   From               Message
  ----     ------            ----  ----               -------
  Warning  FailedScheduling  23s   default-scheduler  0/3 nodes are available: 1 node(s) had untolerated taint {node-role.kubernetes.io/control-plane: }, 2 node(s) had untolerated taint {gpu: true}. preemption
: 0/3 nodes are available: 3 Preemption is not helpful for scheduling.
```

Provide the tolerance as show in below to run a pod on node:

vishal@LAPTOP-NGR39AL4: ~

```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  tolerations:
  - key: "gpu"
    operator: "Equal"
    value: "true"
    effect: "NoSchedule"
status: {}
~
~
```

```
vishal@LAPTOP-NGR39AL4:~/p1$ kubectl apply -f testpod.yaml
pod/nginx created
vishal@LAPTOP-NGR39AL4:~/p1$ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          6s
vishal@LAPTOP-NGR39AL4:~/p1$ kubectl describe pod  nginx
Name:             nginx
Namespace:        default
Priority:         0
Service Account:  default
Node:             cka-mn-cluster-1-worker/172.18.0.5
Start Time:       Mon, 15 Jul 2024 17:58:34 +0530
Labels:           run=nginx
Annotations:      <none>
Status:           Running
IP:               10.244.2.7
IPs:
  IP:  10.244.2.7
Containers:
  nginx:
    Container ID:   containerd://1d49829084da8861991301b35d4ac7a525b690d530a5c1b0452f724c7dd4ddec
    Image:          nginx
    Image ID:       docker.io/library/nginx@sha256:67682bda769fae1ccf5183192b8daf37b64cae99c6c3302650f6f8bf5f0f95df
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Mon, 15 Jul 2024 17:58:38 +0530
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-fwvc9 (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-fwvc9:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 gpu=true:NoSchedule
                             node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
```

3. **NodeSelector**: Using NodeSelector will not guaranty that the pod will be scheduled on a particular desired node. This give pod ability to decide on which node it has to go. (Pod centric approach)

4. **NodeAffinity**:

Node Affinity lets you define complex rules for where your pods can be scheduled based on node labels. Think of it as creating a wishlist for your pod's ideal home!

Key Features:

1. **Flexibility**: Define precise conditions for pod placement.
2. **Control**: Decide where your pods can and cannot go with greater granularity.
3. **Adaptability**: Allow pods to stay on their nodes even if the labels change after scheduling.

Properties of NodeAffinity

1. requiredDuringSchedulingIgnoredDuringExecution

If the required condition is not satisfied with the node lables, the pods will not get scheduled.

2. preferredDuringSchedulingIgnoredDuringExecution

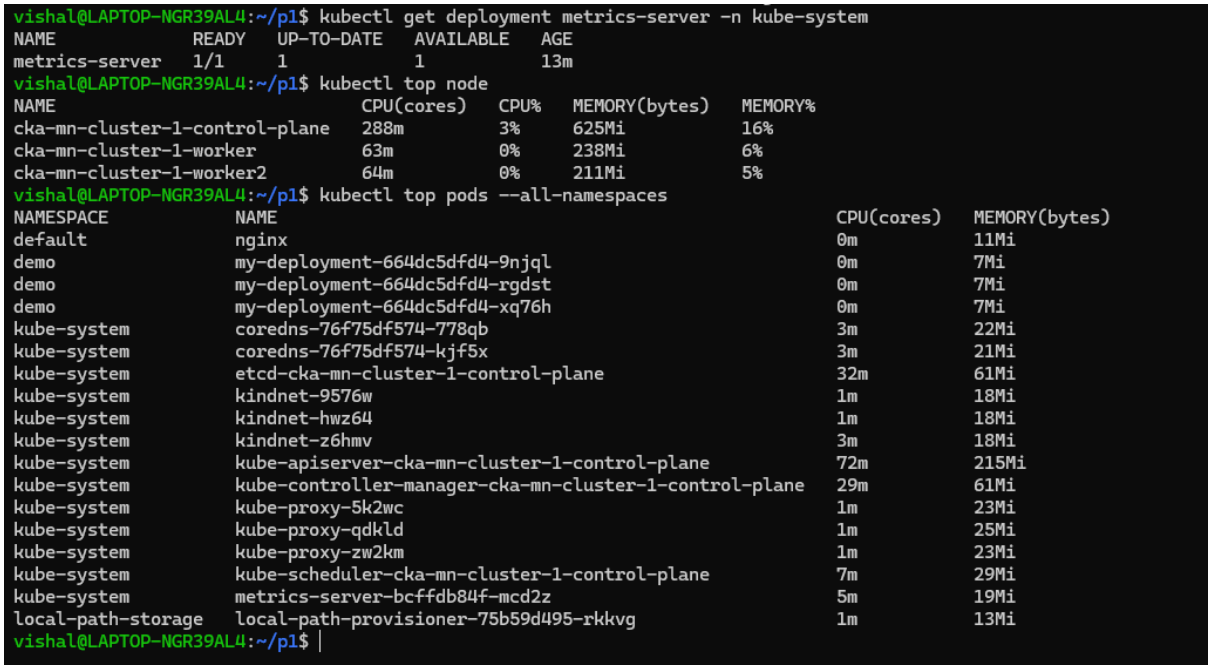
If the required condition is not satisfied with the node lables, still the pods will get scheduled.

(Note: It is always preferred to use Node Affinity (Labels) with Node Selector (Taints))

Install Metrics Server in a Kubernetes Cluster

Steps to install the Metrics Server in a Kubernetes cluster

1. **Download the Metrics Server YAML:**
   kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
2. **Verify the Deployment:**
   kubectl get deployment metrics-server -n kube-system
3. **Edit the Metrics Server Deployment:**
   kubectl edit deployment metrics-server -n kube-system
4. **Add the --kubelet-insecure-tls Flag:**
   - --kubelet-insecure-tls
5. **Verify the Deployment:**
   kubectl get deployment metrics-server -n kube-system
6. **Test the Metrics Server:**
   kubectl top nodes
   kubectl top pods --all-namespaces



```
vishal@LAPTOP-NGR39AL4:~/p1$ kubectl get deployment metrics-server -n kube-system
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
metrics-server   1/1     1            1           13m
vishal@LAPTOP-NGR39AL4:~/p1$ kubectl top node
NAME                           CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
cka-mn-cluster-1-control-plane   288m         3%     625Mi           16%
cka-mn-cluster-1-worker          63m          0%     238Mi           6%
cka-mn-cluster-1-worker2         64m          0%     211Mi           5%
vishal@LAPTOP-NGR39AL4:~/p1$ kubectl top pods --all-namespaces
NAMESPACE            NAME                                                     CPU(cores)   MEMORY(bytes)
default              nginx                                                    0m           11Mi
demo                 my-deployment-664dc5dfd4-9njql                           0m           7Mi
demo                 my-deployment-664dc5dfd4-rgdst                           0m           7Mi
demo                 my-deployment-664dc5dfd4-xq76h                           0m           7Mi
kube-system          coredns-76f75df574-778qb                                 3m           22Mi
kube-system          coredns-76f75df574-kjf5x                                 3m           21Mi
kube-system          etcd-cka-mn-cluster-1-control-plane                      32m          61Mi
kube-system          kindnet-9576w                                            1m           18Mi
kube-system          kindnet-hwz64                                            1m           18Mi
kube-system          kindnet-z6hmv                                            3m           18Mi
kube-system          kube-apiserver-cka-mn-cluster-1-control-plane            72m          215Mi
kube-system          kube-controller-manager-cka-mn-cluster-1-control-plane   29m          61Mi
kube-system          kube-proxy-5k2wc                                         1m           23Mi
kube-system          kube-proxy-qdkld                                         1m           25Mi
kube-system          kube-proxy-zw2km                                         1m           23Mi
kube-system          kube-scheduler-cka-mn-cluster-1-control-plane            7m           29Mi
kube-system          metrics-server-bcffdb84f-mcd2z                           5m           19Mi
local-path-storage   local-path-provisioner-75b59d495-rkkvg                   1m           13Mi
vishal@LAPTOP-NGR39AL4:~/p1$
```

Resources, Requests and Limits

1. Resource

- Resources in Kubernetes refer to the compute resources (primarily CPU and memory) that a container consumes.
- Kubernetes allows you to define how much CPU and memory a container requires (requests) and the maximum it can use (limits).
- These definitions help the Kubernetes scheduler make decisions on where to place pods based on the available resources in the cluster.

```
vishal@LAPTOP-NGR39AL4:~/p1$ cat php-hpa.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
spec:
  selector:
    matchLabels:
      run: php-apache
  template:
    metadata:
      labels:
        run: php-apache
    spec:
      containers:
      - name: php-apache
        image: registry.k8s.io/hpa-example
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
```

2. Requests

- Minimum resource allocated to the respective work load. 200m -> 0.2 of 1 CPU
- Requests specify the minimum amount of CPU and memory that a container needs to run.
- Kubernetes uses requests to decide where to place a pod on a node. It guarantees that the container will always get at least the requested resources.
- If a node doesn't have enough resources to fulfill the request of a container, the pod won't be scheduled on that node.

3. Limits

- Maximum resource allocated to the respective work load. 500m -> 0.5 of 1 CPU
- Limits define the maximum amount of CPU and memory a container is allowed to use.
- If a container tries to use more resources than the limit, Kubernetes will throttle its CPU usage or, in the case of memory, kill the container (with an "OOMKilled" error) and restart it.
- Limits protect your cluster from resource exhaustion by preventing a single container from monopolizing resources.

4. **CPU Units**

- CPU resources are measured in CPU cores. Kubernetes allows you to specify fractional or whole numbers of CPU cores.
- 1 CPU in Kubernetes is equivalent to 1 vCPU/Core for cloud providers or 1 Hyperthread on a bare-metal Intel processor.
- **Unit:**
    - m (millicores):
        - The most common unit is millicores.
        - 1000 millicores = 1 CPU core.
        - This unit allows for fine-grained allocation of CPU resources.
        - Example: 100m represents 100 millicores, which is 10% of a CPU core.
    - Whole Numbers:
        - You can also specify CPU in whole numbers.
        - Example: 1 represents 1 full CPU core.

5. **Memory Units**

- Memory resources are measured in bytes. Kubernetes allows you to specify memory in multiples of bytes using standard suffixes.
- **Units:**
    - Ki (kibibyte): 1 KiB = 1024 bytes
    - Mi (mebibyte): 1 MiB = 1024 KiB = 1,048,576 bytes
    - Gi (gibibyte): 1 GiB = 1024 MiB = 1,073,741,824 bytes
    - Ti (tebibyte): 1 TiB = 1024 GiB = 1,099,511,627,776 bytes
    - Pi (pebibyte): 1 PiB = 1024 TiB = 1,125,899,906,842,624 bytes
    - Ei (exbibyte): 1 EiB = 1024 PiB = 1,152,921,504,606,846,976 bytes

6. **Summary of Common Units**

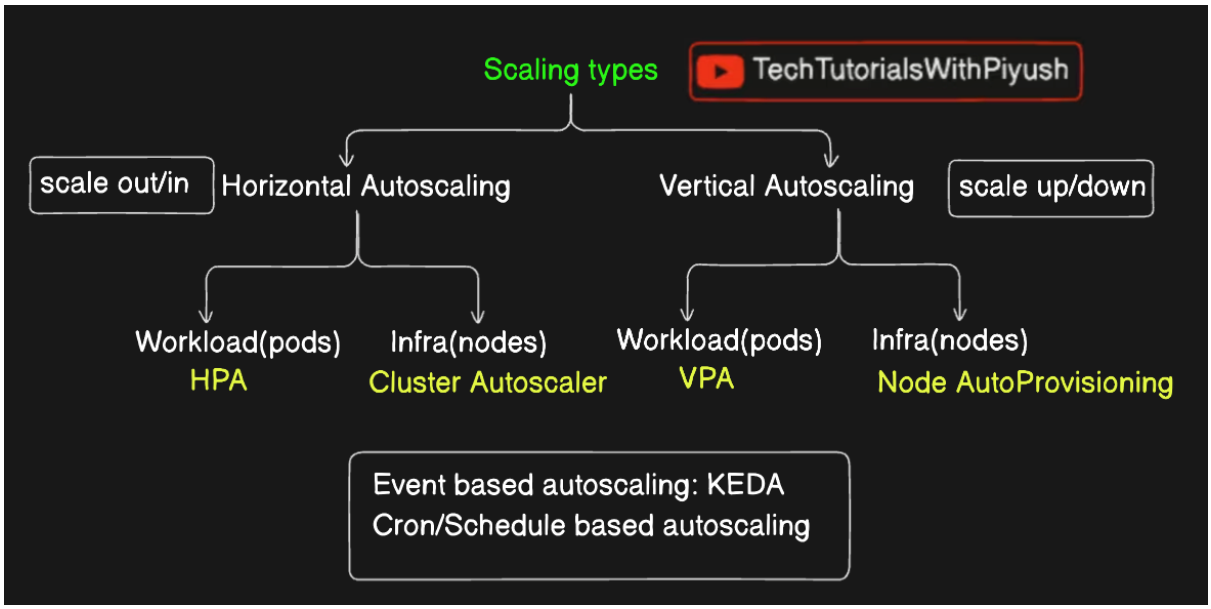
Resource	Unit	Description	Example Value
CPU	m	Millicores (1/1000 of a core)	500m (0.5 CPU core)
CPU	-	Whole CPU cores	2 (2 CPU cores)
Memory	Ki	Kibibytes (1024 bytes)	1024Ki (1 MiB)
Memory	Mi	Mebibytes (1024 KiB)	256Mi (256 MiB)
Memory	Gi	Gibibytes (1024 MiB)	1Gi (1 GiB)

Considerations

- CPU Requests and Limits:
    - Kubernetes schedules pods based on the requested CPU.
    - If the container uses more CPU than the limit, it will be throttled (limited in its CPU usage).
- Memory Requests and Limits:
    - Kubernetes guarantees that the container will get the requested memory.
    - If a container tries to use more memory than the limit, it will be killed and potentially restarted with an "OOMKilled" (Out of Memory) status.
    - Defining CPU and memory resources with appropriate units ensures that your applications get the right amount of resources and helps in preventing resource contention in a Kubernetes cluster.

Kubernetes Autoscaling

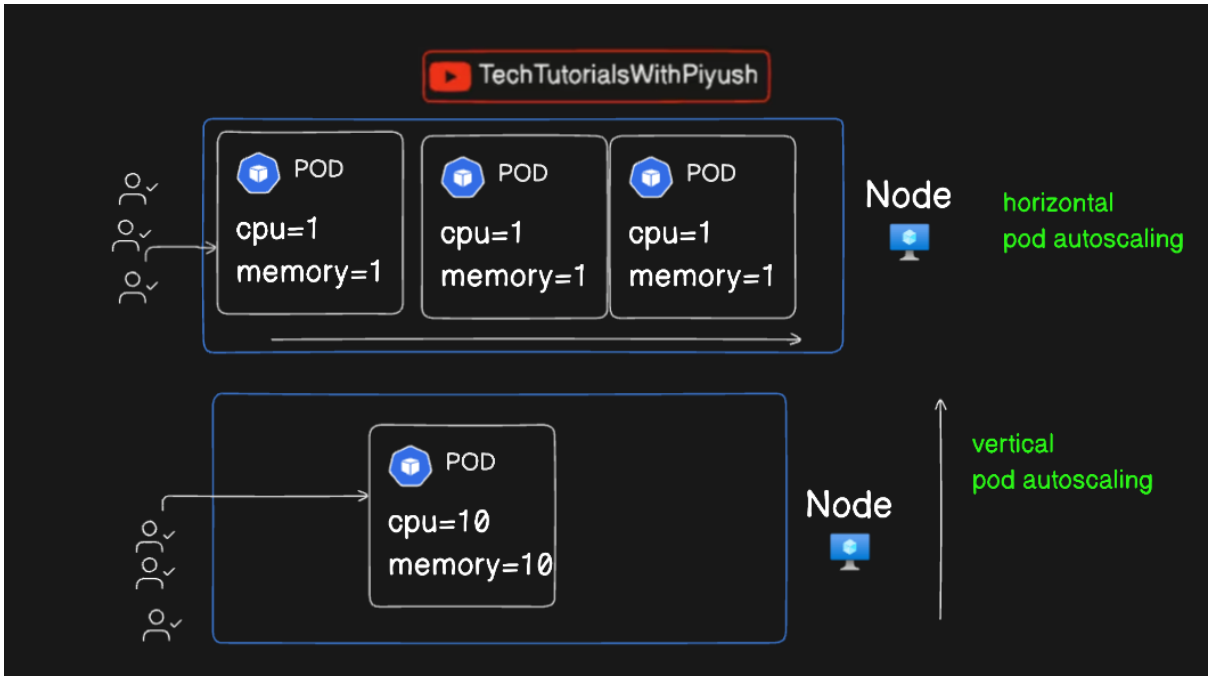
Autoscaling types



Horizontal Vs Vertical Autoscaling

Horizontal Pod Autoscaling (HPA) in Kubernetes automatically adjusts the number of pods in a deployment, replication controller, or replica set based on observed CPU utilization or other custom metrics. HPA comes default with Kubernetes installation.

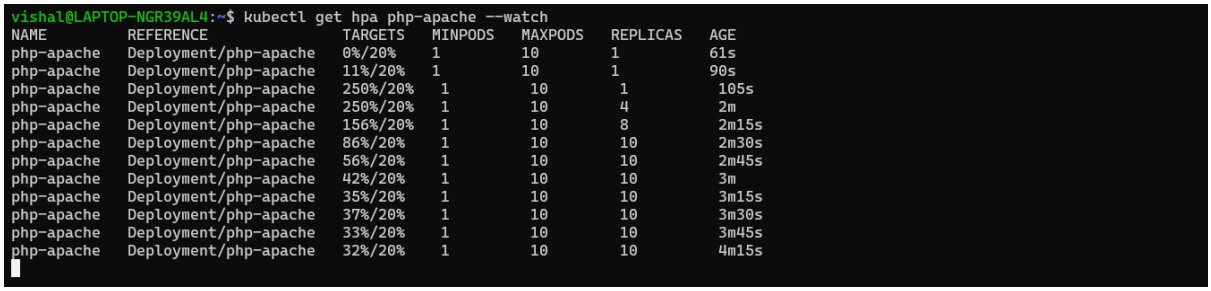
Vertical Pod Autoscaling (VPA) in Kubernetes adjusts the resource requests and limits (CPU and memory) for containers within a pod based on their usage.



Horizontal Pod Autoscaling (HPA) Practical

- ➔ kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
- ➔ kubectl run -i --tty load-generator --rm --image=busybox:1.28 --restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
- ➔ kubectl get hpa php-apache --watch
- ➔ kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10

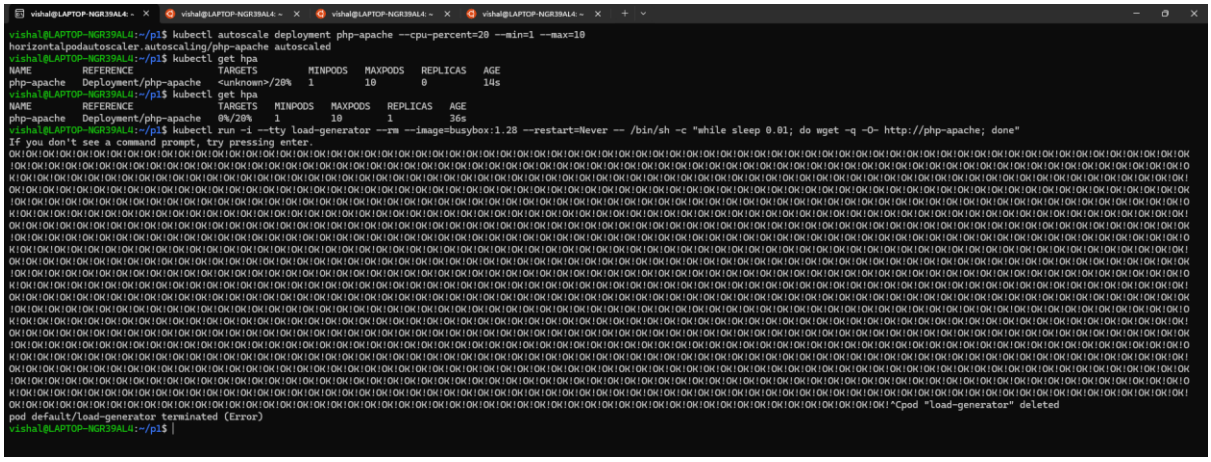
When Load is increasing



```
vishal@LAPTOP-NGR39AL4:~$ kubectl get pods -w
NAME                          READY   STATUS              RESTARTS   AGE
php-apache-598b474864-2dg2s   1/1     Running             0          3m2s
load-generator                0/1     Pending             0          0s
load-generator                0/1     Pending             0          0s
load-generator                0/1     ContainerCreating   0          0s
load-generator                1/1     Running             0          1s
php-apache-598b474864-6kmwk   0/1     Pending             0          0s
php-apache-598b474864-6kmwk   0/1     Pending             0          0s
php-apache-598b474864-8bd7x   0/1     Pending             0          0s
php-apache-598b474864-77fhl   0/1     Pending             0          0s
php-apache-598b474864-8bd7x   0/1     Pending             0          0s
php-apache-598b474864-6kmwk   0/1     ContainerCreating   0          0s
php-apache-598b474864-77fhl   0/1     Pending             0          0s
php-apache-598b474864-8bd7x   0/1     ContainerCreating   0          0s
php-apache-598b474864-77fhl   0/1     ContainerCreating   0          0s
php-apache-598b474864-8bd7x   1/1     Running             0          2s
php-apache-598b474864-bs56x   0/1     Pending             0          0s
php-apache-598b474864-vndjs   0/1     Pending             0          0s
php-apache-598b474864-bs56x   0/1     Pending             0          0s
php-apache-598b474864-2lvb4   0/1     Pending             0          0s
php-apache-598b474864-vndjs   0/1     Pending             0          0s
php-apache-598b474864-44wk8   0/1     Pending             0          0s
php-apache-598b474864-2lvb4   0/1     Pending             0          0s
php-apache-598b474864-44wk8   0/1     Pending             0          0s
php-apache-598b474864-bs56x   0/1     ContainerCreating   0          0s
php-apache-598b474864-vndjs   0/1     ContainerCreating   0          0s
php-apache-598b474864-2lvb4   0/1     ContainerCreating   0          0s
php-apache-598b474864-44wk8   0/1     ContainerCreating   0          0s
php-apache-598b474864-bs56x   1/1     Running             0          2s
php-apache-598b474864-2lvb4   1/1     Running             0          3s
php-apache-598b474864-qh7f6   0/1     Pending             0          0s
php-apache-598b474864-cgt2j   0/1     Pending             0          0s
php-apache-598b474864-qh7f6   0/1     Pending             0          0s
php-apache-598b474864-cgt2j   0/1     Pending             0          0s
php-apache-598b474864-qh7f6   0/1     ContainerCreating   0          0s
php-apache-598b474864-cgt2j   0/1     ContainerCreating   0          0s
php-apache-598b474864-qh7f6   1/1     Running             0          3s
php-apache-598b474864-77fhl   1/1     Running             0          42s
php-apache-598b474864-6kmwk   1/1     Running             0          42s
php-apache-598b474864-vndjs   1/1     Running             0          28s
php-apache-598b474864-44wk8   1/1     Running             0          29s
php-apache-598b474864-cgt2j   1/1     Running             0          14s
```

```
vishal@LAPTOP-NGR39AL4:~$ kubectl get deploy
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
php-apache   10/10   10           10          4m52s
vishal@LAPTOP-NGR39AL4:~$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
load-generator                1/1     Running   0          103s
php-apache-598b474864-2dg2s   1/1     Running   0          5m8s
php-apache-598b474864-2lvb4   1/1     Running   0          60s
php-apache-598b474864-44wk8   1/1     Running   0          60s
php-apache-598b474864-6kmwk   1/1     Running   0          75s
php-apache-598b474864-77fhl   1/1     Running   0          75s
php-apache-598b474864-8bd7x   1/1     Running   0          75s
php-apache-598b474864-bs56x   1/1     Running   0          60s
php-apache-598b474864-cgt2j   1/1     Running   0          45s
php-apache-598b474864-qh7f6   1/1     Running   0          45s
php-apache-598b474864-vndjs   1/1     Running   0          60s
vishal@LAPTOP-NGR39AL4:~$
```

When Load in decreasing




```
vishal@LAPTOP-NGR39AL4:~$ kubectl get pods -w
NAME                          READY   STATUS    RESTARTS   AGE
php-apache-598b474864-2dg2s   1/1     Running   0          9m57s
php-apache-598b474864-2lvb4   1/1     Running   0          5m49s
php-apache-598b474864-44wk8   1/1     Running   0          5m49s
php-apache-598b474864-6kmwk   1/1     Running   0          6m4s
php-apache-598b474864-77fhl   1/1     Running   0          6m4s
php-apache-598b474864-8bd7x   1/1     Running   0          6m4s
php-apache-598b474864-bs56x   1/1     Running   0          5m49s
php-apache-598b474864-cgt2j   1/1     Running   0          5m34s
php-apache-598b474864-qh7f6   1/1     Running   0          5m34s
php-apache-598b474864-vndjs   1/1     Running   0          5m49s
php-apache-598b474864-77fhl   1/1     Terminating   0          9m1s
php-apache-598b474864-qh7f6   1/1     Terminating   0          8m31s
php-apache-598b474864-bs56x   1/1     Terminating   0          8m46s
php-apache-598b474864-8bd7x   1/1     Terminating   0          9m1s
php-apache-598b474864-6kmwk   1/1     Terminating   0          9m1s
php-apache-598b474864-vndjs   1/1     Terminating   0          8m46s
php-apache-598b474864-44wk8   1/1     Terminating   0          8m46s
php-apache-598b474864-qh7f6   0/1     Terminating   0          8m31s
php-apache-598b474864-6kmwk   0/1     Terminating   0          9m1s
php-apache-598b474864-8bd7x   0/1     Terminating   0          9m1s
php-apache-598b474864-77fhl   0/1     Terminating   0          9m1s
php-apache-598b474864-bs56x   0/1     Terminating   0          8m46s
php-apache-598b474864-vndjs   0/1     Terminating   0          8m46s
php-apache-598b474864-44wk8   0/1     Terminating   0          8m46s
php-apache-598b474864-44wk8   0/1     Terminating   0          8m46s
php-apache-598b474864-vndjs   0/1     Terminating   0          8m46s
php-apache-598b474864-6kmwk   0/1     Terminating   0          9m1s
php-apache-598b474864-77fhl   0/1     Terminating   0          9m1s
php-apache-598b474864-6kmwk   0/1     Terminating   0          9m1s
php-apache-598b474864-6kmwk   0/1     Terminating   0          9m1s
php-apache-598b474864-77fhl   0/1     Terminating   0          9m1s
php-apache-598b474864-77fhl   0/1     Terminating   0          9m2s
php-apache-598b474864-vndjs   0/1     Terminating   0          8m47s
php-apache-598b474864-vndjs   0/1     Terminating   0          8m47s
php-apache-598b474864-44wk8   0/1     Terminating   0          8m47s
php-apache-598b474864-44wk8   0/1     Terminating   0          8m47s
php-apache-598b474864-qh7f6   0/1     Terminating   0          8m32s
php-apache-598b474864-8bd7x   0/1     Terminating   0          9m2s
php-apache-598b474864-qh7f6   0/1     Terminating   0          8m32s
php-apache-598b474864-qh7f6   0/1     Terminating   0          8m32s
php-apache-598b474864-8bd7x   0/1     Terminating   0          9m2s
php-apache-598b474864-8bd7x   0/1     Terminating   0          9m2s
php-apache-598b474864-bs56x   0/1     Terminating   0          8m47s
php-apache-598b474864-bs56x   0/1     Terminating   0          8m47s
php-apache-598b474864-bs56x   0/1     Terminating   0          8m47s
php-apache-598b474864-2dg2s   1/1     Terminating   0          13m
php-apache-598b474864-2lvb4   1/1     Terminating   0          9m1s
```

```
vishal@LAPTOP-NGR39AL4:~$ kubectl get hpa php-apache --watch
NAME         REFERENCE                  TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   0%/20%    1         10        1          61s
php-apache   Deployment/php-apache   11%/20%   1         10        1          90s
php-apache   Deployment/php-apache   250%/20%   1         10        1          105s
php-apache   Deployment/php-apache   250%/20%   1         10        4          2m
php-apache   Deployment/php-apache   156%/20%   1         10        8          2m15s
php-apache   Deployment/php-apache   86%/20%    1         10        10         2m30s
php-apache   Deployment/php-apache   56%/20%    1         10        10         2m45s
php-apache   Deployment/php-apache   42%/20%    1         10        10         3m
php-apache   Deployment/php-apache   35%/20%    1         10        10         3m15s
php-apache   Deployment/php-apache   37%/20%    1         10        10         3m30s
php-apache   Deployment/php-apache   33%/20%    1         10        10         3m45s
php-apache   Deployment/php-apache   32%/20%    1         10        10         4m15s
php-apache   Deployment/php-apache   32%/20%    1         10        10         4m30s
php-apache   Deployment/php-apache   34%/20%    1         10        10         4m45s
php-apache   Deployment/php-apache   35%/20%    1         10        10         5m
php-apache   Deployment/php-apache   30%/20%    1         10        10         5m15s
php-apache   Deployment/php-apache   31%/20%    1         10        10         5m30s
php-apache   Deployment/php-apache   32%/20%    1         10        10         5m45s
php-apache   Deployment/php-apache   6%/20%     1         10        10         6m
php-apache   Deployment/php-apache   2%/20%     1         10        10         6m15s
php-apache   Deployment/php-apache   0%/20%     1         10        10         6m30s
php-apache   Deployment/php-apache   0%/20%     1         10        10         10m
php-apache   Deployment/php-apache   0%/20%     1         10        3          11m
php-apache   Deployment/php-apache   0%/20%     1         10        1          11m
```

```
vishal@LAPTOP-NGR39AL4:~$ kubectl get deploy
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
php-apache   1/1     1            1           13m
vishal@LAPTOP-NGR39AL4:~$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
php-apache-598b474864-cgt2j   1/1     Running   0          9m43s
vishal@LAPTOP-NGR39AL4:~$
```

Health Probes in Kubernetes

Health Probes in Kubernetes:

1. **Startup Probe** – For slow/legacy applications
   The kubelet uses startup probes to know when a container application has started. If such a probe is configured, liveness and readiness probes do not start until it succeeds, making sure those probes don't interfere with the application startup. This can be used to adopt liveness checks on slow starting containers, avoiding them getting killed by the kubelet before they are up and running.

Purpose:

- The Startup Probe is designed to detect and handle applications that are slow to start, ensuring that they are not prematurely killed by the liveness probe or removed from the service's endpoints by the readiness probe.

How It Works:

- The Startup Probe is configured similarly to the liveness and readiness probes, using an HTTP GET request, a TCP socket, or an exec command to check the application's health.
- During the startup phase, Kubernetes will rely on the startup probe's success or failure to decide whether the container should be restarted.
- Once the startup probe succeeds, it is disabled, and the liveness and readiness probes take over.

2. **Readiness Probe** – Ensure the application is ready

Purpose:

- The **Readiness Probe** is used to determine if a container is ready to start accepting traffic. If the readiness probe fails, Kubernetes will temporarily remove the pod from the service's load balancers, meaning it won't receive any traffic until it passes the readiness check again.

When to Use:

- Use readiness probes to prevent traffic from being routed to containers that are not yet ready to serve requests (e.g., during startup or initialization).
- They are critical for applications that take time to initialize, or that might need to temporarily reject traffic (e.g., during maintenance).

Types of Readiness Probes:

2.1. Readiness command
2.2. Readiness HTTP request
2.3. TCP Readiness probe

3. **Liveness Probe** – Restarts the application if it fails

Purpose:

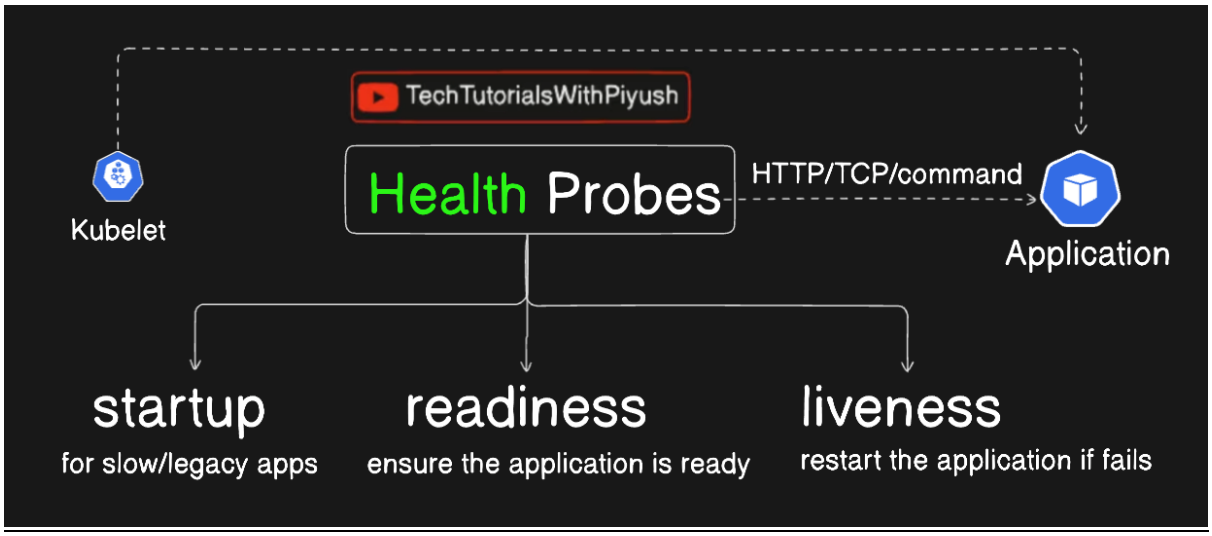
- The **Liveness Probe** is used to determine if a container is running. If the liveness probe fails, Kubernetes will kill the container, and it will be restarted according to the pod's restart policy.

When to Use:

- Use liveness probes to detect and recover from situations where an application is stuck or dead (e.g., an application is in an infinite loop or deadlocked).
- Liveness probes ensure that unhealthy containers are terminated and restarted, which is essential for maintaining availability.

Types of Liveness Probes:

3.1. liveness command
3.2. liveness HTTP request
3.3. TCP liveness probe
3.4. gRPC liveness probe



Role Based Access Control (RBAC) in Kubernetes:

Role-Based Access Control (RBAC) in Kubernetes is a mechanism used to regulate access to resources in a Kubernetes cluster. It provides fine-grained control over who can do what within the cluster, ensuring that users and applications have the appropriate permissions without compromising security.

Role

- A **Role** defines a set of permissions or rules that determine what actions can be performed on what resources.
- Roles are namespace-specific, meaning they apply to resources within a specific namespace.

ClusterRole

- A **ClusterRole** is similar to a Role but applies cluster-wide, rather than to a specific namespace. It can also be used to grant access to non-namespaced resources like nodes and persistent volumes.

RoleBinding

- A **RoleBinding** grants the permissions defined in a Role to a user, group, or service account within a specific namespace.

ClusterRoleBinding

- A **ClusterRoleBinding** grants the permissions defined in a ClusterRole to a user, group, or service account across the entire cluster.

How RBAC Works?

RBAC works by associating users or service accounts with specific roles that define what actions they are permitted to take on various resources. These roles are then bound to the users or groups via RoleBindings or ClusterRoleBindings. **Roles** and **RoleBindings** are scoped to namespaces, providing control over resources within a specific namespace. **ClusterRoles** and **ClusterRoleBindings** provide cluster-wide permissions, including non-namespaced resources.

Use Case Scenarios

1. **Restricting Access to a Specific Namespace**: You have multiple development teams working in a Kubernetes cluster, each with their own namespace. You can create Roles and RoleBindings to ensure that each team only has access to their own namespace.
2. **Granular Control for Service Accounts**: A microservice running in your cluster needs to read secrets and config maps but should not have permissions to delete them. You can create a Role with get permissions on secrets and config maps, and bind it to the service account used by the microservice.
3. **Admin Access for Cluster Operators**: You want to grant full cluster-wide administrative access to a cluster operator. You can create a ClusterRoleBinding that binds the cluster-admin ClusterRole to the operator's user account.
4. **Read-Only Access for Auditors**: An auditor needs to inspect the configuration and state of the cluster without making any changes. You can create a ClusterRole with read-only permissions and bind it to the auditor’s account using a ClusterRoleBinding.

Steps to Creating Users in Kubernetes with RBAC

1. Create a Client Certificate for the User

Generate a private key for vishal
openssl genrsa -out vishal.key 2048

Create a certificate signing request (CSR) for vishal
openssl req -new -key vishal.key -out vishal.csr -subj "/CN=vishal/O=group1"

# Sign the CSR with the Kubernetes CA to get the client certificate
sudo openssl x509 -req -in vishal.csr -CA /etc/kubernetes/pki/ca.crt -CAkey /etc/kubernetes/pki/ca.key -CAcreateserial -out vishal.crt -days 365

2. Create a Kubeconfig File for the User

kubectl config set-cluster kubernetes --certificate-authority=/etc/kubernetes/pki/ca.crt --embed-certs=true --server=https://<your-kubernetes-api-server> --kubeconfig=vishal.kubeconfig

kubectl config set-credentials vishal --client-certificate=vishal.crt --client-key=vishal.key --embed-certs=true --kubeconfig=vishal.kubeconfig

kubectl config set-context vishal-context --cluster=kubernetes --user=vishal --kubeconfig=vishal.kubeconfig

kubectl config use-context vishal-context --kubeconfig=vishal.kubeconfig

3. Create a Role/clusterRole

4. Crate a RoleBinding/ClusterRoleBinding

5. Switch to the Vishal User

To use the new configuration, copy vishal.kubeconfig to ~/.kube/config (or set the KUBECONFIG environment variable to point to vishal.kubeconfig).

export KUBECONFIG=/home/vishal/vishal.kubeconfig

Now, User vishal can use kubectl to interact with the cluster within the permissions granted by the RBAC settings.

Issue a certificate for a user in Kubernetes

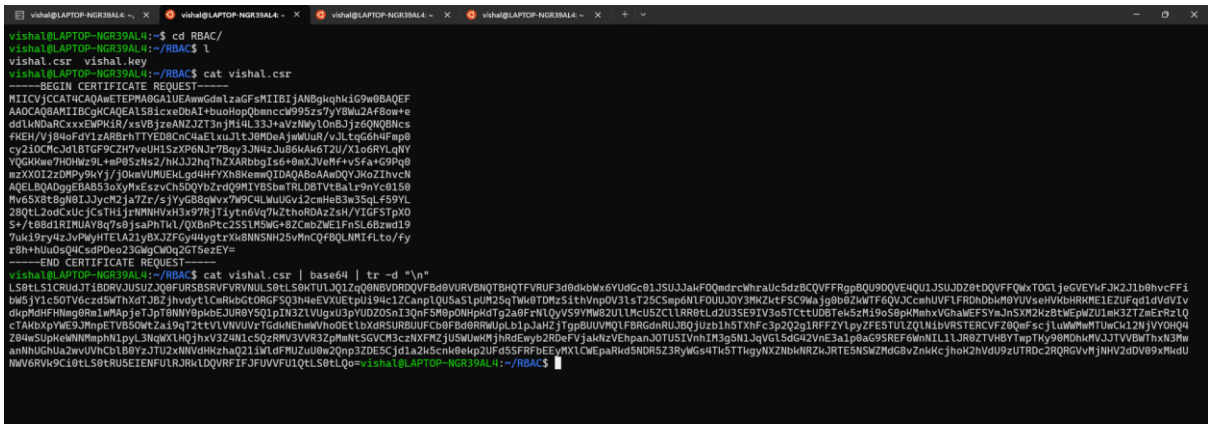
1. Generate a key for client:
   ➔ openssl genrsa -out vishal.key 2048

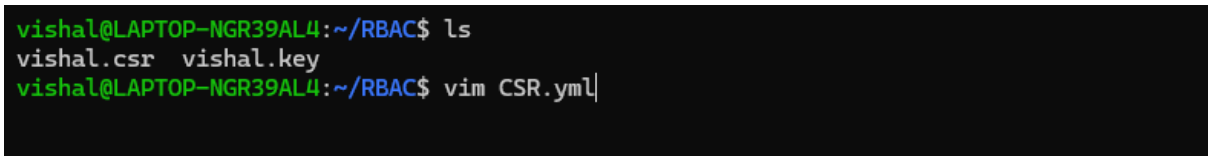
```
vishal@LAPTOP-NGR39AL4:~/RBAC$ openssl genrsa -out vishal.key 2048
vishal@LAPTOP-NGR39AL4:~/RBAC$ ls
vishal.key
vishal@LAPTOP-NGR39AL4:~/RBAC$ cat vishal.key
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCVLyJzF4NsAj5u
6geilBuadxxb33nOzvJjxa7YB/yjD5512WQ0NpELHHERY8qJH/GxUGPN4A1kllPe
eMyLgvfcn5pXM1bKU6cEmPPpA1AE1yx8oQf9WPzigV1jXMBEGuFNNgQPwKcLhoSX
G4mW0nQwN4CPBZS5H+8ku2obqHgWanRzLaI4Ixwl2UFMYX0Jkfu95QfVLNc/o0mv
sGrLck3jMm7zqQCTpPZT9fWjpFguo1hhAYorB7sc4dbP0v6Y/RLM2zb+EoknaGpO
FlcBFtuAizr7SZclV4x/69J9r4b0+rSbNdc4jbMMw/L2RiP+M6SZVQxQSQuB3gd9
heHwp6bBAgMBAAECggEASIHGqBKDDQFQL98uA84hVmNS1LJahPHfLC/dyWCkYOpL
9XJJufjVaQWTTMToINp2QVxVja+6Qquvqp6ip6HpbwDhZkVtBtw+fkUKszMpdpjQ
JeisNe3Exx/h8FOZnOBFpZ29vpw/dCIWmkzHhBbYYsDvClRHkuxa8krmiPfKGfY4
UgC2dM8VVq0PZhr0d62zpC3+2BDsEQTXMHNenNPC1al8Vga0OaptbxqCW4/QIPPw
9LA6kh7omwF0DxK+z4Lc8IFmyJkQsP8lJDQSGBLbmB4NM77vW86ChkxeuwME/jK3
/ZbKggpD6/+8YB5MUERKPK9cz0K5RZ6E9W6A/3sfkwKBgQDOQRYOmFMu/JW45XcH
4acJgYmvjNoYG8vIxaRMRaj8I5r20ff7RKFjdxe0mLP0wS22yx4PD10PKqJZ0+Lf
0RSuAEMpu+5bSjZKDyt1czM8ZAuxq6I1ANywMTrm6DBi8VoU4rqwdLbE+zK7rmYn
ACgsPq3p5Jj3D8+3pVkrXkPWNwKBgQC5KlLzXpI4hsifJkh5c15BU0TzDjBrWrIF
aR/m43qyFL/oXWX8H5P9lRoR8fA78YYPRSGBNJPig7XDz3VhIO/P9Ut5/M+bZXw+
E5R+FB3BkwoXrEeZ3YvHqGGgY+wRFMbbz6dThlBUcDjn6Hc98n6nLJNgG6aPSf0n
/C9YiuPuxwKBgFFIzUyLM5085jSsGjD8+TuB1aVduUXvDmzdl1aHzYUw8w16ymQ4
M/JjKPB/jcxTpZR4kCwXPfGQN9N28d6LI78/UzjLW7CWmratK89pyzNtqj4nChIh
eNfnTvNZZlhYDcxVXoXUOYOADApzPMb6BNsLCLc45B/aF16kx3OT/uAZAoGBAJmQ
UFsmA359ycSzq0eONo1W0cgB7IYcIaf7F0oVPpySONYtH3eJeOLF1WksjWApXlUr
sC3HvR4OvaA3XdG/zyNY+8+fq03lTLTu6Cfn5wWzPJuFdZxV8KcVLmImfJFt+b4y
6UKMEfL1Xzx1vu7RPdjHTzJVvDz5z8ppi5qTFZXhAoGAWiqpBsWhC5pbtVtbeSIT
W5r6Yfh2P2D54bikRQwoQcy1cYnFLqwcWwtmJRC19BX27AVnDXW24QNQ9g0W+WFe
PaQE6WKUDbIXB86a4LvKIW9ELrH/+ZlLrBrhc0qqb756p3juHFH2QrS2AbYMSUNr
eQkl3A701uZhYXYHXpat9ZE=
-----END PRIVATE KEY-----
vishal@LAPTOP-NGR39AL4:~/RBAC$
```

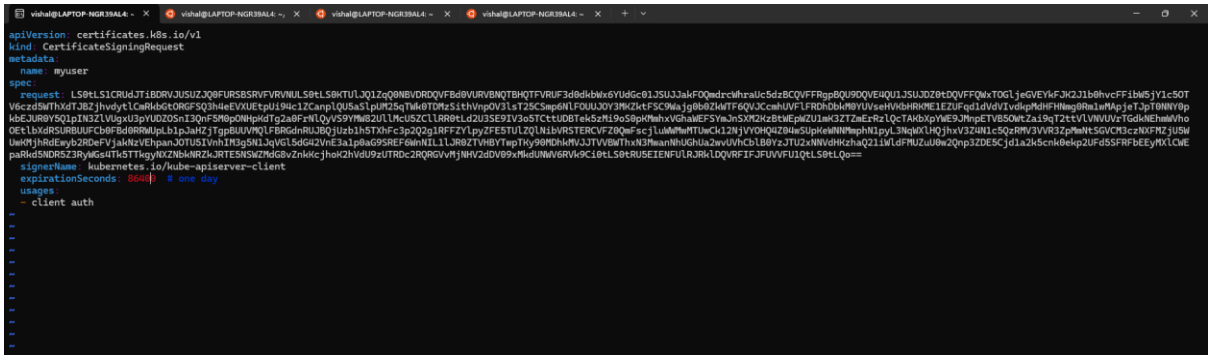
2. Generate a CSR
   ➔ openssl req -new -key vishal.key -out vishal.csr -subj "/CN=vishal"

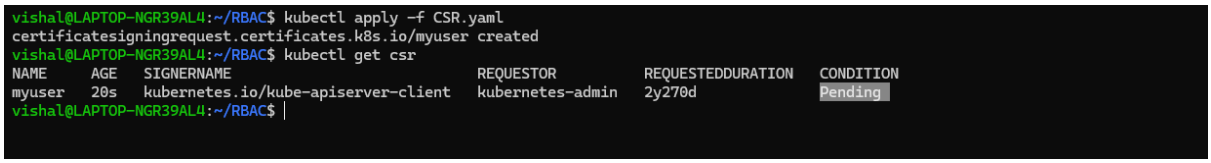
```
vishal@LAPTOP-NGR39AL4:~/RBAC$ openssl req -new -key vishal.key -out vishal.csr -subj "/CN=vishal"
vishal@LAPTOP-NGR39AL4:~/RBAC$ ls
vishal.csr  vishal.key
vishal@LAPTOP-NGR39AL4:~/RBAC$ cat vishal.csr
-----BEGIN CERTIFICATE REQUEST-----
MIICVjCCAT4CAQAwETEPMA0GA1UEAwwGdmlzaGFsMIIBIjANBgkqhkiG9w0BAQEF
AAOCAQ8AMIIBCgKCAQEAlS8icxeDbAI+buoHopQbmnccW995zs7yY8Wu2Af8ow+e
ddlkNDaRCxxxEWPKiR/xsVBjzeANZJZT3njMi4L33J+aVzNWylOnBJjz6QNQBNcs
fKEH/Vj84oFdY1zARBrhTTYED8CnC4aElxuJltJ0MDeAjwWUuR/vJLtqG6h4Fmp0
cy2iOCMcJdlBTGF9CZH7veUH1SzXP6NJr7Bqy3JN4zJu86kAk6T2U/X1o6RYLqNY
YQGKKwe7HOHWz9L+mP0SzNs2/hKJJ2hqThZXARbbgIs6+0mXJVeMf+vSfa+G9Pq0
mzXXOI2zDMPy9kYj/jOkmVUMUEkLgd4HfYXh8KemwQIDAQABoAAwDQYJKoZIhvcN
AQELBQADggEBAB53oXyMxEszvCh5DQYbZrdQ9MIYBSbmTRLDBTVtBalr9nYc0150
Mv65X8t8gN0IJJycM2ja7Zr/sjYyGB8qWvx7W9C4LWuUGvi2cmHeB3w35qLf59YL
28QtL2odCxUcjCsTHijrNMNHVxH3x97RjTiytn6Vq7kZthoRDAzZsH/YIGFSTpXO
S+/t08d1RIMUAY8q7s0jsaPhTkl/QXBnPtc2SSlM5WG+8ZCmbZWE1FnSL6Bzwd19
7uki9ry4zJvPWyHTElA21yBXJZFGy44ygtrXk8NNSNH25vMnCQfBQLNMIfLto/fy
r8h+hUuOsQ4CsdPDeo23GWgCWOq2GT5ezEY=
-----END CERTIFICATE REQUEST-----
vishal@LAPTOP-NGR39AL4:~/RBAC$
```

3. As a Kubernetes admin create a “CertificateSingningRequest” yaml manifest using the CSR created previously.

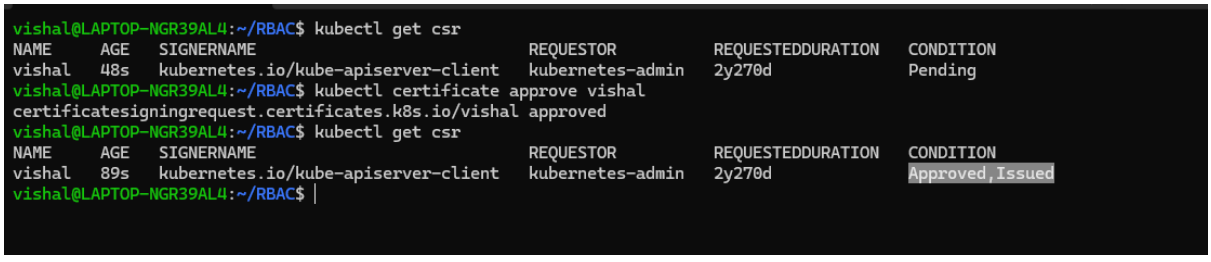








4. Approve the CSR



5. Get the certificate from Approved CSR

Now you have an approved certificate for user “vishal”

Role and Role Binding

An RBAC Role or ClusterRole contains rules that represent a set of permissions. Permissions are purely additive (there are no "deny" rules).

```
vishal@LAPTOP-NGR39AL4:~/RBAC$ ls -ltr
total 24
-rw------- 1 vishal vishal 1704 Jul 28 19:27 vishal.key
-rw-r--r-- 1 vishal vishal  887 Jul 28 19:34 vishal.csr
-rw-r--r-- 1 vishal vishal 1411 Jul 28 20:21 CSR.yaml
-rw-r--r-- 1 vishal vishal 4832 Jul 28 20:24 IssuedCerVishal.yaml
-rw-r--r-- 1 vishal vishal 1090 Jul 28 20:32 IssuedCertificate.crt
vishal@LAPTOP-NGR39AL4:~/RBAC$ vim role.yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
~
~
~
~
~
```

```
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl apply -f role.yaml
role.rbac.authorization.k8s.io/pod-reader created
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl get role
NAME         CREATED AT
pod-reader   2024-07-28T15:09:37Z
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl describe role pod-reader
Name:         pod-reader
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
  ---------  -----------------  --------------  -----
  pods       []                 []              [get watch list]
vishal@LAPTOP-NGR39AL4:~/RBAC$
```

Create a Role Binding object to add particular user in a pod-reader role

```yaml
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "jane" to read pods in the "default" namespace.
# You need to already have a Role named "pod-reader" in that namespace.
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
# You can specify more than one "subject"
- kind: User
  name: vishal # "name" is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  # "roleRef" specifies the binding to a Role / ClusterRole
  kind: Role #this must be Role or ClusterRole
  name: pod-reader # this must match the name of the Role or ClusterRole you wish to bind to
  apiGroup: rbac.authorization.k8s.io
~
~
~
```

```
vishal@LAPTOP-NGR39AL4:~/RBAC$ vim rolebinding.yaml
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl apply -f rolebinding.yaml
rolebinding.rbac.authorization.k8s.io/read-pods created
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl get rolebinding
NAME        ROLE              AGE
read-pods   Role/pod-reader   16s
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl describe read-pod
error: the server doesn't have a resource type "read-pod"
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl describe read-pods
error: the server doesn't have a resource type "read-pods"
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl describe rolebinding read-pods
Name:         read-pods
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  pod-reader
Subjects:
  Kind  Name    Namespace
  ----  ----    ---------
  User  vishal
vishal@LAPTOP-NGR39AL4:~/RBAC$
```

Before applying role binding:

```
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl auth can-i get pods
yes
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl auth can-i get pods --as vishal
no
```

After applying role binding:

```
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl auth can-i get pods --as vishal
yes
vishal@LAPTOP-NGR39AL4:~/RBAC$
```

Login as a user “vishal”

1. Set User Credentials and contexts

```
vishal@LAPTOP-NGR39AL4:~/RBAC$ ls -ltr
total 32
-rw------- 1 vishal vishal 1704 Jul 28 19:27 vishal.key
-rw-r--r-- 1 vishal vishal  887 Jul 28 19:34 vishal.csr
-rw-r--r-- 1 vishal vishal 1411 Jul 28 20:21 CSR.yaml
-rw-r--r-- 1 vishal vishal 4832 Jul 28 20:24 IssuedCerVishal.yaml
-rw-r--r-- 1 vishal vishal 1090 Jul 28 20:32 IssuedCertificate.crt
-rw-r--r-- 1 vishal vishal  217 Jul 28 20:39 role.yaml
-rw-r--r-- 1 vishal vishal  647 Jul 28 20:42 rolebinding.yaml
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl config set-credentials vishal --client-key=vishal.key --client-certificate=IssuedCertificate.crt --embed-certs=true
User "vishal" set.
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl config get-contexts
CURRENT   NAME                          CLUSTER                       AUTHINFO                      NAMESPACE
          kind-cka-cluster1             kind-cka-cluster1             kind-cka-cluster1
*         kind-cka-mn-cluster-1         kind-cka-mn-cluster-1         kind-cka-mn-cluster-1
          kind-cka-multinodecluster1    kind-cka-multinodecluster1    kind-cka-multinodecluster1
```

```
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl config set-context vishal --cluster=kind-cka-mn-cluster-1 --user=vishal
Context "vishal" created.
vishal@LAPTOP-NGR39AL4:~/RBAC$
```

2. Use the context

```
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl config get-contexts
CURRENT   NAME                          CLUSTER                       AUTHINFO                      NAMESPACE
          kind-cka-cluster1             kind-cka-cluster1             kind-cka-cluster1
*         kind-cka-mn-cluster-1         kind-cka-mn-cluster-1         kind-cka-mn-cluster-1
          kind-cka-multinodecluster1    kind-cka-multinodecluster1    kind-cka-multinodecluster1
          vishal                        kind-cka-mn-cluster-1         vishal
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl config use-context vishal
Switched to context "vishal".
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl whoami
error: unknown command "whoami" for "kubectl"
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl auth whoami
ATTRIBUTE   VALUE
Username    vishal
Groups      [system:authenticated]
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
php-apache-598b474864-cgt2j   1/1     Running   0          6h47m
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl get svc
Error from server (Forbidden): services is forbidden: User "vishal" cannot list resource "services" in API group "" in the namespace "default"
vishal@LAPTOP-NGR39AL4:~/RBAC$
```

All resources at namespace level:

```
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl api-resources --namespaced=true
\NAME                            SHORTNAMES   APIVERSION                     NAMESPACED   KIND
bindings                                      v1                             true         Binding
configmaps                       cm           v1                             true         ConfigMap
endpoints                        ep           v1                             true         Endpoints
events                           ev           v1                             true         Event
limitranges                      limits       v1                             true         LimitRange
persistentvolumeclaims           pvc          v1                             true         PersistentVolumeClaim
pods                             po           v1                             true         Pod
podtemplates                                  v1                             true         PodTemplate
replicationcontrollers           rc           v1                             true         ReplicationController
resourcequotas                   quota        v1                             true         ResourceQuota
secrets                                       v1                             true         Secret
serviceaccounts                  sa           v1                             true         ServiceAccount
services                         svc          v1                             true         Service
controllerrevisions                           apps/v1                        true         ControllerRevision
daemonsets                       ds           apps/v1                        true         DaemonSet
deployments                      deploy       apps/v1                        true         Deployment
replicasets                      rs           apps/v1                        true         ReplicaSet
statefulsets                     sts          apps/v1                        true         StatefulSet
localsubjectaccessreviews                     authorization.k8s.io/v1        true         LocalSubjectAccessReview
horizontalpodautoscalers         hpa          autoscaling/v2                 true         HorizontalPodAutoscaler
cronjobs                         cj           batch/v1                       true         CronJob
jobs                                          batch/v1                       true         Job
leases                                        coordination.k8s.io/v1         true         Lease
endpointslices                                discovery.k8s.io/v1            true         EndpointSlice
events                           ev           events.k8s.io/v1               true         Event
pods                                          metrics.k8s.io/v1beta1         true         PodMetrics
ingresses                        ing          networking.k8s.io/v1           true         Ingress
networkpolicies                  netpol       networking.k8s.io/v1           true         NetworkPolicy
poddisruptionbudgets             pdb          policy/v1                      true         PodDisruptionBudget
rolebindings                                  rbac.authorization.k8s.io/v1   true         RoleBinding
roles                                         rbac.authorization.k8s.io/v1   true         Role
csistoragecapacities                          storage.k8s.io/v1              true         CSIStorageCapacity
vishal@LAPTOP-NGR39AL4:~/RBAC$ \
```

All resources at cluster level:

```
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl api-resources --namespaced=false
NAME                              SHORTNAMES   APIVERSION                             NAMESPACED   KIND
componentstatuses                 cs           v1                                     false        ComponentStatus
namespaces                        ns           v1                                     false        Namespace
nodes                             no           v1                                     false        Node
persistentvolumes                 pv           v1                                     false        PersistentVolume
mutatingwebhookconfigurations                  admissionregistration.k8s.io/v1        false        MutatingWebhookConfiguration
validatingwebhookconfigurations                admissionregistration.k8s.io/v1        false        ValidatingWebhookConfiguration
customresourcedefinitions         crd,crds     apiextensions.k8s.io/v1                false        CustomResourceDefinition
apiservices                                    apiregistration.k8s.io/v1              false        APIService
selfsubjectreviews                             authentication.k8s.io/v1               false        SelfSubjectReview
tokenreviews                                   authentication.k8s.io/v1               false        TokenReview
selfsubjectaccessreviews                       authorization.k8s.io/v1                false        SelfSubjectAccessReview
selfsubjectrulesreviews                        authorization.k8s.io/v1                false        SelfSubjectRulesReview
subjectaccessreviews                           authorization.k8s.io/v1                false        SubjectAccessReview
certificatesigningrequests        csr          certificates.k8s.io/v1                 false        CertificateSigningRequest
flowschemas                                    flowcontrol.apiserver.k8s.io/v1        false        FlowSchema
prioritylevelconfigurations                    flowcontrol.apiserver.k8s.io/v1        false        PriorityLevelConfiguration
nodes                                          metrics.k8s.io/v1beta1                 false        NodeMetrics
ingressclasses                                 networking.k8s.io/v1                   false        IngressClass
runtimeclasses                                 node.k8s.io/v1                         false        RuntimeClass
clusterrolebindings                            rbac.authorization.k8s.io/v1           false        ClusterRoleBinding
clusterroles                                   rbac.authorization.k8s.io/v1           false        ClusterRole
priorityclasses                   pc           scheduling.k8s.io/v1                   false        PriorityClass
csidrivers                                     storage.k8s.io/v1                      false        CSIDriver
csinodes                                       storage.k8s.io/v1                      false        CSINode
storageclasses                    sc           storage.k8s.io/v1                      false        StorageClass
volumeattachments                              storage.k8s.io/v1                      false        VolumeAttachment
vishal@LAPTOP-NGR39AL4:~/RBAC$
```

Cluster Role and Cluster Role Binding

*(**Note:** Roles are at the Namespace level and Cluster Role are at the Cluster and Node scope level)*

Create a Cluster Role

```
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl create clusterrole --help
Create a cluster role.

Examples:
  # Create a cluster role named "pod-reader" that allows user to perform "get", "watch" and "list" on pods
  kubectl create clusterrole pod-reader --verb=get,list,watch --resource=pods
```

```
Usage:
  kubectl create clusterrole NAME --verb=verb --resource=resource.group [--resource-name=resourcename]
[--dry-run=server|client|none] [options]

Use "kubectl options" for a list of global command-line options (applies to all commands).
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl create clusterrole node-reader --verb=get,list,watch --resource=node
clusterrole.rbac.authorization.k8s.io/node-reader created
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl get clusterrole | grep node-reader
node-reader                                                            2024-07-28T16:38:46Z
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl describe node-reader
error: the server doesn't have a resource type "node-reader"
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl describe clusterrole node-reader
Name:         node-reader
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
  ---------  -----------------  --------------  -----
  nodes      []                 []              [get list watch]
vishal@LAPTOP-NGR39AL4:~/RBAC$
```

Create Cluster Role Binding:

```
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl create clusterrolebinding --help
Create a cluster role binding for a particular cluster role.

Examples:
  # Create a cluster role binding for user1, user2, and group1 using the cluster-admin cluster role
  kubectl create clusterrolebinding cluster-admin --clusterrole=cluster-admin --user=user1 --user=user2 --group=group1
```

```
Usage:
  kubectl create clusterrolebinding NAME --clusterrole=NAME [--user=username] [--group=groupname]
[--serviceaccount=namespace:serviceaccountname] [--dry-run=server|client|none] [options]

Use "kubectl options" for a list of global command-line options (applies to all commands).
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl create clusterrolebinding node-reader-binding --clusterrole=node-reader-binding --user=vishal
clusterrolebinding.rbac.authorization.k8s.io/node-reader-binding created
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl get clusterrolebinding | grep node-reader
node-reader-binding                                    ClusterRole/node-reader-binding                                    22s
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl describe clusterrolebinding node-role-binding
Error from server (NotFound): clusterrolebindings.rbac.authorization.k8s.io "node-role-binding" not found
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl describe clusterrolebinding node-reader-binding
Name:         node-reader-binding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  node-reader-binding
Subjects:
  Kind  Name    Namespace
  ----  ----    ---------
  User  vishal
vishal@LAPTOP-NGR39AL4:~/RBAC$
```

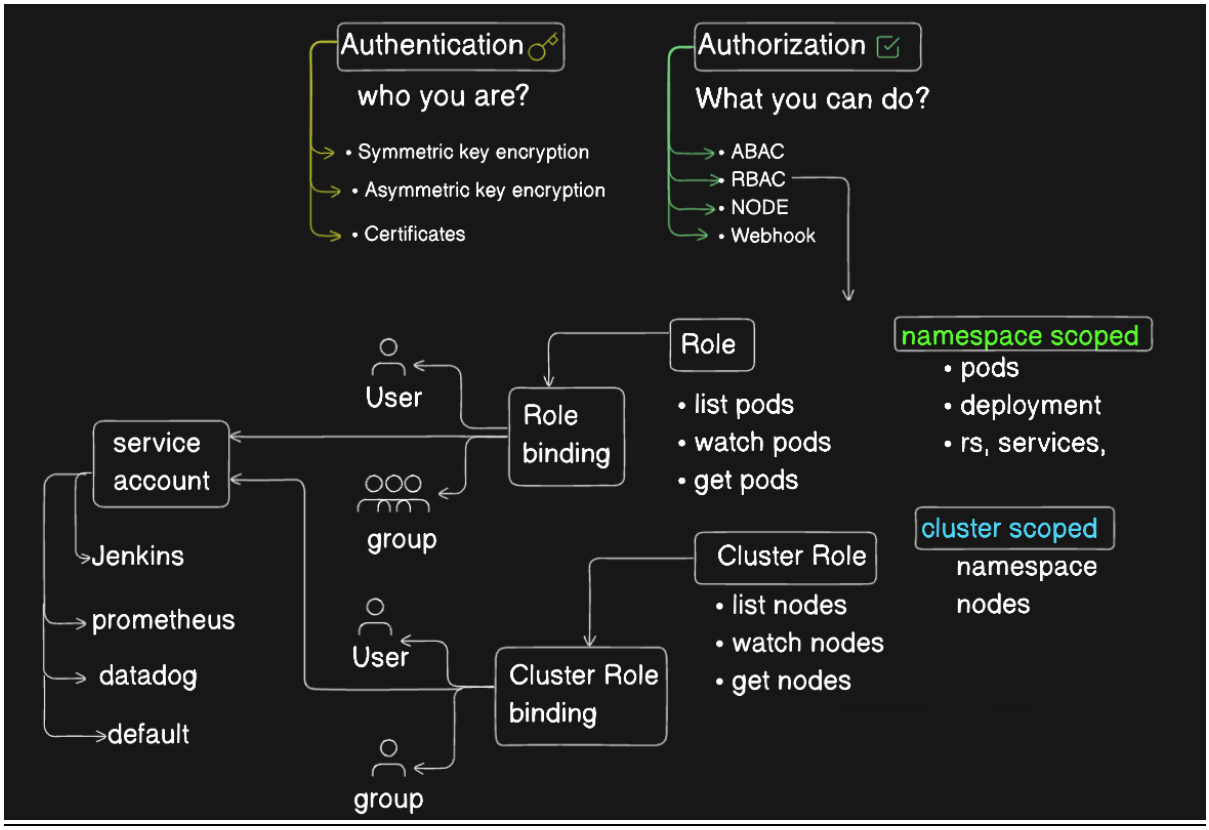
As a user “vishal” we are getting the access for node using ClusterRoleBinding

```
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl delete clusterrolebinding node-reader-binding
clusterrolebinding.rbac.authorization.k8s.io "node-reader-binding" deleted
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl create clusterrolebinding node-reader-binding --clusterrole=node-reader --user=vishal
clusterrolebinding.rbac.authorization.k8s.io/node-reader-binding created
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl auth can-i get nodes
Warning: resource 'nodes' is not namespace scoped

yes
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl auth can-i get nodes --as vishal
Warning: resource 'nodes' is not namespace scoped

yes
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl config get-contexts
CURRENT   NAME                          CLUSTER                       AUTHINFO                      NAMESPACE
          kind-cka-cluster1             kind-cka-cluster1             kind-cka-cluster1
*         kind-cka-mn-cluster-1         kind-cka-mn-cluster-1         kind-cka-mn-cluster-1
          kind-cka-multinodecluster1    kind-cka-multinodecluster1    kind-cka-multinodecluster1
          vishal                        kind-cka-mn-cluster-1         vishal
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl config use-context vishal
Switched to context "vishal".
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl auth whoami
ATTRIBUTE   VALUE
Username    vishal
Groups      [system:authenticated]
vishal@LAPTOP-NGR39AL4:~/RBAC$ kubectl get nodes
NAME                              STATUS   ROLES           AGE   VERSION
cka-mn-cluster-1-control-plane    Ready    control-plane   14d   v1.29.4
cka-mn-cluster-1-worker           Ready    <none>          14d   v1.29.4
cka-mn-cluster-1-worker2          Ready    <none>          14d   v1.29.4
vishal@LAPTOP-NGR39AL4:~/RBAC$
```

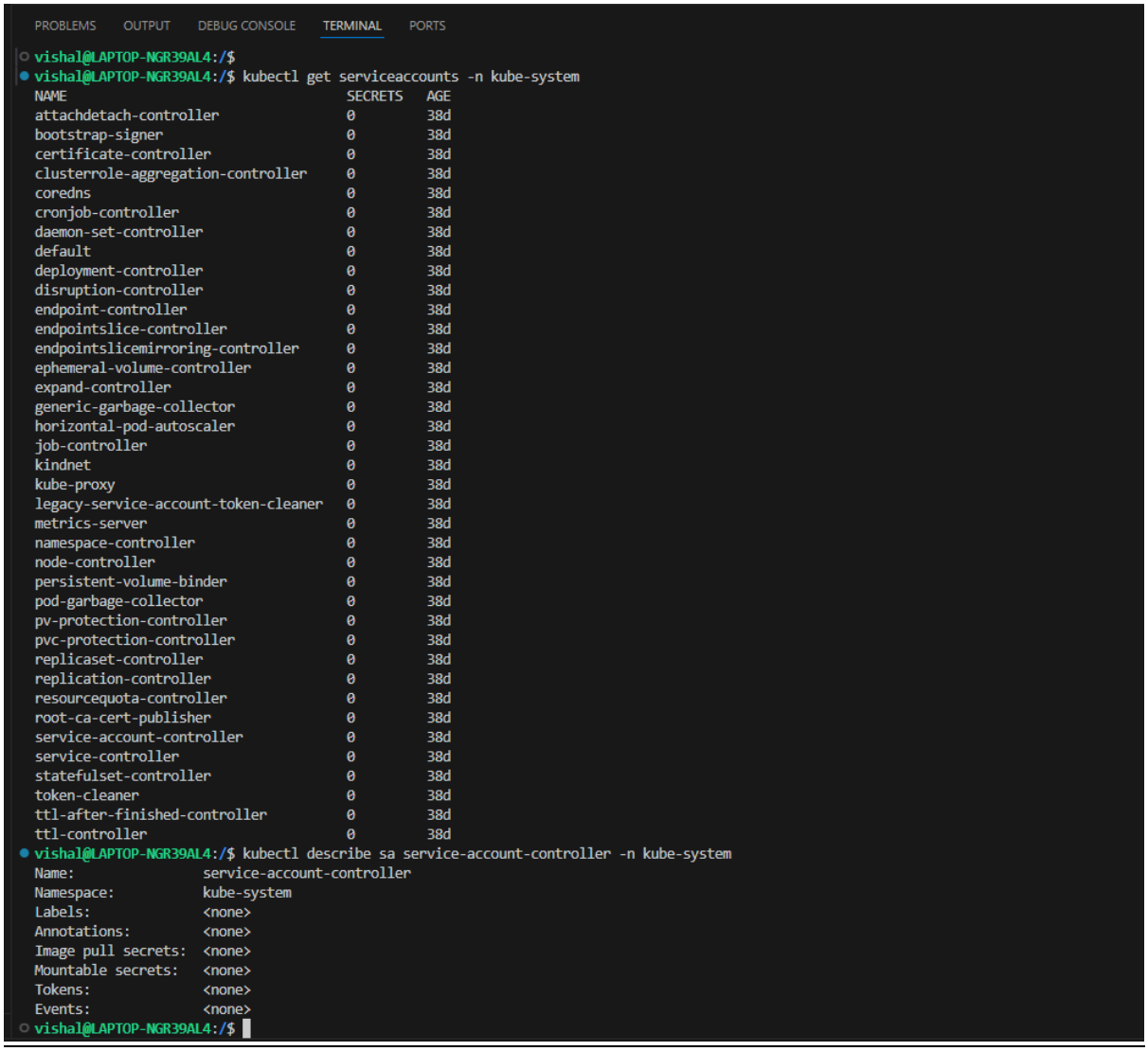
Service Account

There are multiple types of accounts in Kubernetes that interact with the cluster. These could be user accounts used by Kubernetes Admins, developers, operators, etc., and service accounts primarily used by other applications/bots or Kubernetes components to interact with other services. Kubernetes also create 1 default service account in each of the default namespace such as kube-sytem, kube-node-lease and so on

- ➔ kubectl create sa <name>
- ➔ kubectl get sa




```
vishal@LAPTOP-NGR39AL4:/$ kubectl get serviceaccounts
NAME      SECRETS   AGE
default   0         38d
vishal@LAPTOP-NGR39AL4:/$ kubectl describe sa default
Name:                default
Namespace:           default
Labels:              <none>
Annotations:         <none>
Image pull secrets:  <none>
Mountable secrets:   <none>
Tokens:              <none>
Events:              <none>
vishal@LAPTOP-NGR39AL4:/$
```


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
vishal@LAPTOP-NGR39AL4:/$
vishal@LAPTOP-NGR39AL4:/$ kubectl get serviceaccounts -n kube-system
NAME SECRETS AGE
attachdetach-controller 0 38d
bootstrap-signer 0 38d
certificate-controller 0 38d
clusterrole-aggregation-controller 0 38d
coredns 0 38d
cronjob-controller 0 38d
daemon-set-controller 0 38d
default 0 38d
deployment-controller 0 38d
disruption-controller 0 38d
endpoint-controller 0 38d
endpointslice-controller 0 38d
endpointslicemirroring-controller 0 38d
ephemeral-volume-controller 0 38d
expand-controller 0 38d
generic-garbage-collector 0 38d
horizontal-pod-autoscaler 0 38d
job-controller 0 38d
kindnet 0 38d
kube-proxy 0 38d
legacy-service-account-token-cleaner 0 38d
metrics-server 0 38d
namespace-controller 0 38d
node-controller 0 38d
persistent-volume-binder 0 38d
pod-garbage-collector 0 38d
pv-protection-controller 0 38d
pvc-protection-controller 0 38d
replicaset-controller 0 38d
replication-controller 0 38d
resourcequota-controller 0 38d
root-ca-cert-publisher 0 38d
service-account-controller 0 38d
service-controller 0 38d
statefulset-controller 0 38d
token-cleaner 0 38d
ttl-after-finished-controller 0 38d
ttl-controller 0 38d
vishal@LAPTOP-NGR39AL4:/$ kubectl describe sa service-account-controller -n kube-system
Name: service-account-controller
Namespace: kube-system
Labels: <none>
Annotations: <none>
Image pull secrets: <none>
Mountable secrets: <none>
Tokens: <none>
Events: <none>
vishal@LAPTOP-NGR39AL4:/$

Network Policies in Kubernetes

1. CNI (Container Network Interface)

CNI stands for Container Network Interface. It's a standard for configuring network interfaces in Linux containers, used by container orchestrators like Kubernetes. CNI provides a framework for plugins to manage container networking, allowing different networking solutions to be easily integrated.

CNI Plugins does not comes with Kubernetes installation so we need to install the CNI plugins externally. Below are some popular CNI plugins.

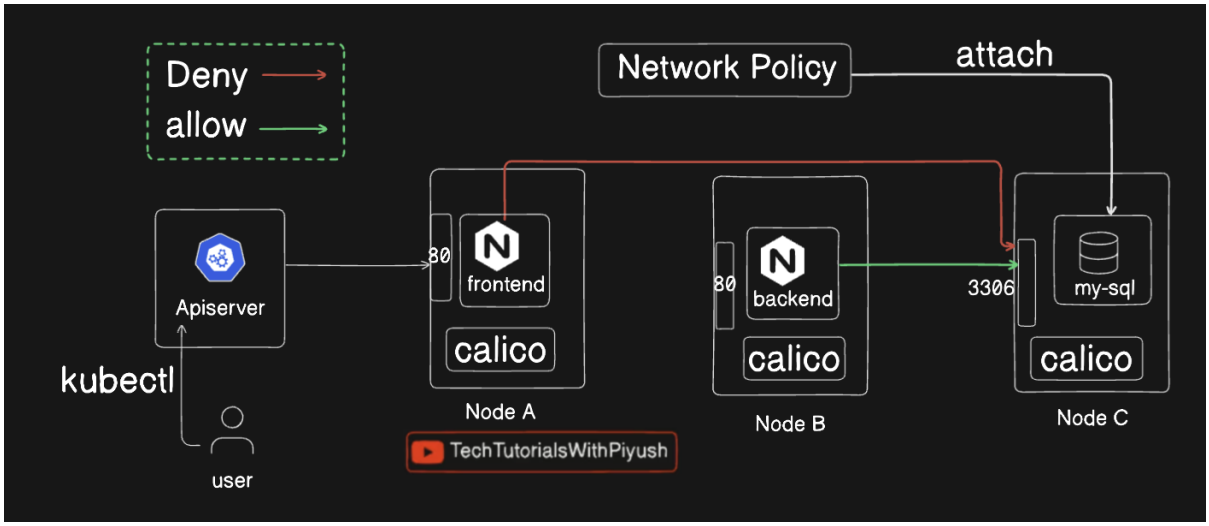
1. Antrea
2. Weave-net
3. Flannel and Kindnet (Doesn't support Network Policies)
4. Calico
5. Cilium
6. Kube-router
7. Romana

CNI is deployed as a Demon set. So CNI pod is running on each Node in Cluster.

Refer: https://kubernetes.io/docs/tasks/administer-cluster/declare-network-policy/

2. Network Policy

Network policy allows you to control the Inbound and Outbound traffic to and from the cluster. For example, you can specify a deny-all network policy that restricts all incoming traffic to the cluster, or you can create an allow-network policy that will only allow certain services to be accessed by certain pods on a specific port.



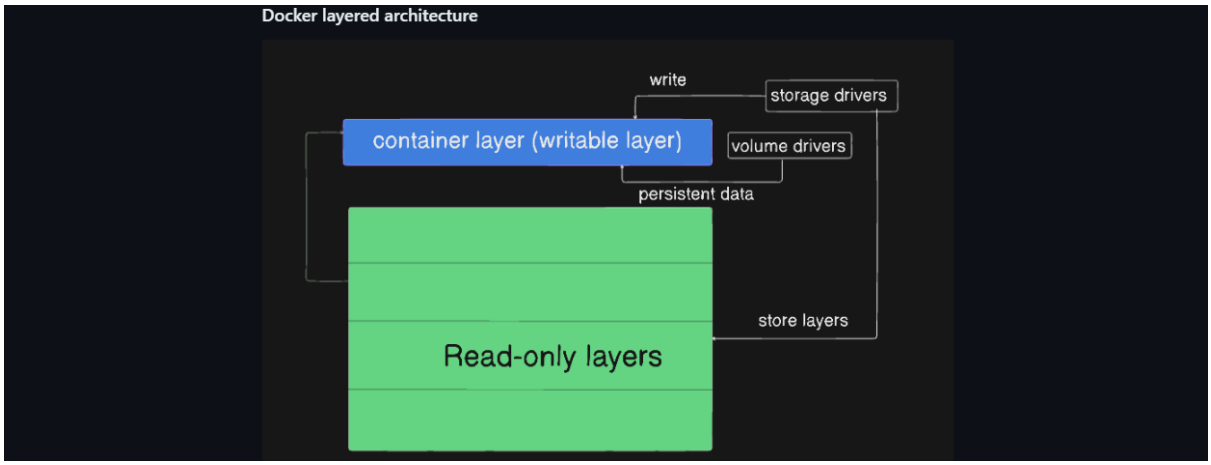
Document to install calico on your cluster

https://docs.tigera.io/calico/latest/getting-started/kubernetes/kind

Docker Storage

Note:

1. *Default Directory for Docker is "/var/lib/docker" at this directory all docker data is stored.*



1. **Storage Driver:** It is responsible for writing data in the container and also stores layers with in container. The Docker storage driver is a component that manages how Docker handles the storage of container data, including images and volumes. Each storage driver uses different techniques to store and manage container data on the host filesystem, and the choice of driver can impact performance, efficiency, and compatibility with different file systems.
    a. Overlay2
    b. Zfs
    c. Vfs

2. **Volume**: Docker volumes are a fundamental feature for managing persistent data in Docker containers. They provide a mechanism for containers to store data that persists beyond the lifecycle of the container itself. This is crucial for maintaining data integrity and consistency, especially for applications that require data to be stored independently of the container's lifecycle.

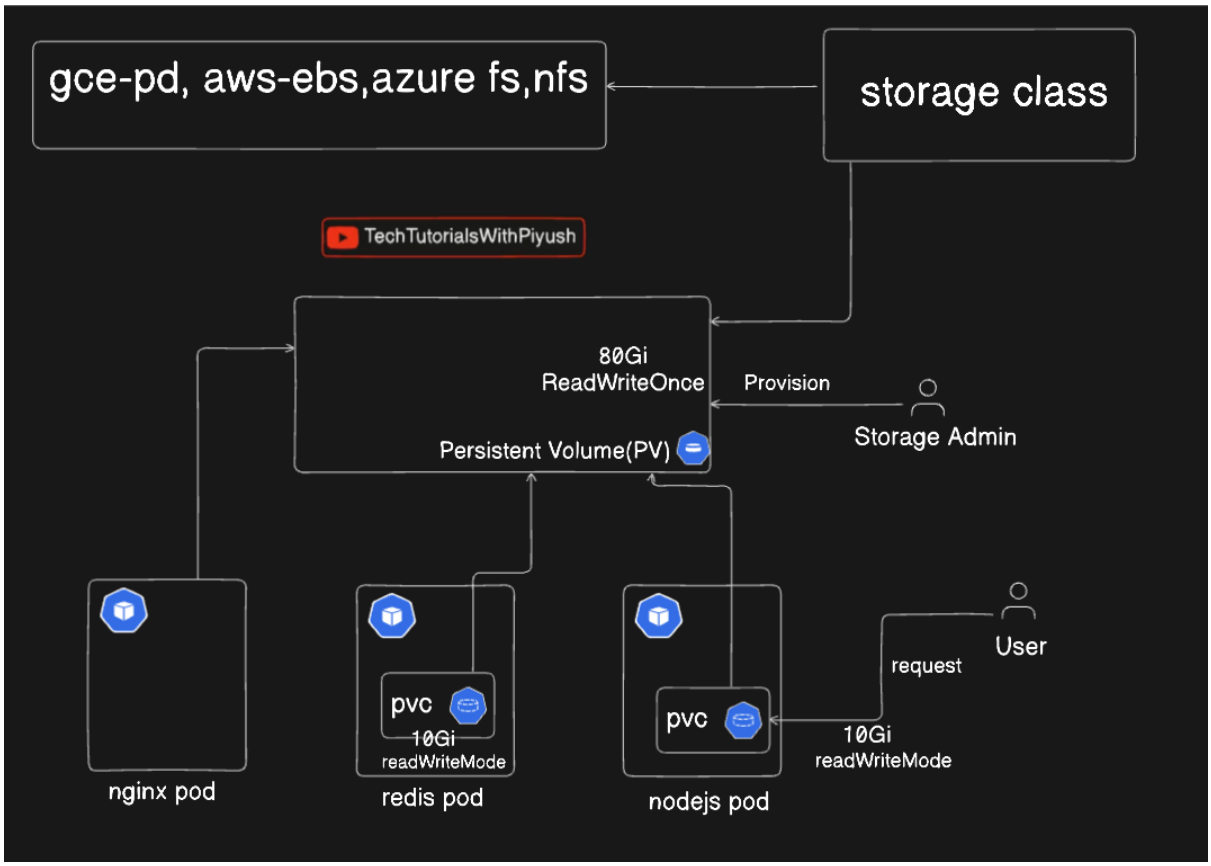
   A Docker bind mount is a type of volume that allows you to map a specific directory or file from the host filesystem directly into a container. This is useful when you need to provide the container with access to files or directories on the host, or when you want to share data between the host and the container.

3. **Volume Drivers:** It is responsible for making the data persistent. Docker Volume Driver is a plugin interface that allows Docker to interact with external storage systems or services. While Docker volumes are used to persist data generated by and used by Docker containers, the Volume Driver enables more advanced or specialized volume management by integrating with various storage backends. There are various volume drivers available, each designed to interact with specific types of storage systems. Some common examples include:
    - Local Driver: The default driver, which stores volumes on the local filesystem.
    - NFS Driver: Allows you to use NFS (Network File System) for storing volumes.
    - Cloud Providers: Drivers that integrate with cloud storage services like Amazon EBS, Google Cloud Storage, or Azure Blob Storage.
    - rexray: Integrates with storage services like Amazon EBS, EMC, or other SAN/NAS solutions.

Kubernetes Storage

Note:

1. *Default Directory for Kubernetes is "etc/kubernetes/" in Control plain Node. At this directory where all Kubernetes control plain components manifest, certificate and all other data is stored.*



1. **Storage class**:

StorageClass defines the type of storage available in a Kubernetes cluster. It specifies the storage characteristics and provides a way to dynamically provision persistent storage. A StorageClass allows you to abstract away the details of the underlying storage provider and define parameters like performance, replication, and more.

Key Features:

- **Provisioning**: Defines how storage is dynamically provisioned using a specific provisioner. For example, it can be used with cloud providers like AWS EBS, GCP Persistent Disks, or Azure Disks, or with on-premises solutions.
- **Parameters**: Allows specifying parameters related to the storage backend, such as disk type, IOPS, replication factor, etc.
- **Reclaim Policy**: Specifies what happens to the volume when the PersistentVolumeClaim (PVC) is deleted (e.g., delete the volume or retain it for manual cleanup).

2. **Persistent Volume**:

PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using a StorageClass. PVs are a representation of the physical storage resource in the cluster.

Key Features:

- **Lifecycle**: PVs exist independently of the pods that use them and can be reused or reclaimed as per the defined policies.
- **Attributes**: Includes information about the storage capacity, access modes, and the storage class it belongs to.
- **Reclaim Policy**: Defines what happens to the volume when it is released by a PVC (e.g., retain, delete, or recycle).

3. **Persistent Volume claim**:

PersistentVolumeClaim (PVC) is a request for storage by a user or application. It is used to request a specific amount of storage with certain attributes (like access modes) from available PersistentVolumes.

Key Features:

- **Request**: Allows users to specify the amount of storage required and the access mode.
- **Binding**: PVCs are bound to available PVs that match the requested criteria. If a suitable PV is found, it will be bound to the PVC.
- **Dynamic Provisioning**: If no suitable PV is available, the PVC may trigger the creation of a new PV based on the StorageClass.

In Kubernetes, **Access Modes** and **Reclaim Policies** are key attributes for managing PersistentVolumes (PVs) and PersistentVolumeClaims (PVCs). They define how storage resources are accessed and managed within the cluster.

Access Modes

Access Modes define how a PersistentVolume (PV) can be mounted and accessed by containers. They specify the level of access a pod has to the volume. Kubernetes supports the following access modes:

1. **ReadWriteOnce (RWO)**:
    - **Description**: The volume can be mounted as read-write by a single node.
    - **Usage**: This is useful for applications that need to write data and can operate from a single node, such as a single-instance database or an application server.
2. **ReadOnlyMany (ROX)**:
    - **Description**: The volume can be mounted as read-only by many nodes.
    - **Usage**: Suitable for scenarios where multiple nodes need to read data from the volume but not write to it, such as serving static content or configuration files.
3. **ReadWriteMany (RWX)**:
    - **Description**: The volume can be mounted as read-write by many nodes simultaneously.

- **Usage**: This mode is used for applications that require concurrent read and write access from multiple nodes, such as distributed file systems or shared data storage solutions.

4. **ReadWriteOncePod (RWOP)** (introduced in Kubernetes 1.22):
    - **Description**: The volume can be mounted as read-write by a single pod only, but the pod can be on multiple nodes.
    - **Usage**: Useful for workloads that require exclusive access to the volume within a single pod, but that pod can be rescheduled across nodes.

Reclaim Policies

Reclaim Policies determine what happens to a PersistentVolume (PV) when its associated PersistentVolumeClaim (PVC) is deleted. They define the lifecycle of the PV and how it should be handled after it is no longer needed.

1. **Retain**:
    - **Description**: The PV is retained and not deleted after the PVC is deleted. The volume will remain in the cluster and must be manually reclaimed by an administrator.
    - **Usage**: This policy is used when you want to preserve the data even after the PVC is deleted, allowing for manual cleanup or data recovery.
2. **Delete**:
    - **Description**: The PV and its associated storage are deleted when the PVC is deleted. This is often used with dynamically provisioned volumes where the underlying storage is managed by the cloud provider or storage system.
    - **Usage**: Suitable for ephemeral data where the volume should be automatically cleaned up and not retained after use, such as temporary or cache storage.
3. **Recycle** (deprecated):
    - **Description**: The PV is scrubbed and made available for reuse when the PVC is deleted. Scrubbing typically involves deleting the data on the volume before it is made available for new claims.
    - **Usage**: This policy was used to make the volume available for new claims after cleaning. It has been deprecated in favor of using Retain and manual management.

DNS in Kubernetes

In Kubernetes, DNS (Domain Name System) plays a crucial role in enabling service discovery and name resolution within the cluster. DNS in Kubernetes is used to automatically assign DNS names to services, allowing them to be accessed by other services and pods using human-readable names rather than IP addresses. Kubernetes ensures that DNS records are created and maintained dynamically as services are added, removed, or updated.

Key DNS Concepts:

1. **Service Discovery**:
    - Kubernetes uses DNS for service discovery, enabling pods to find and communicate with services using DNS names.
    - Each service in Kubernetes is automatically assigned a DNS name, which is based on the service name and the namespace. For example, a service named my-service in the default namespace would be accessible via my-service.default.svc.cluster.local.
2. **DNS Namespaces**:
    - Kubernetes uses a fully qualified domain name (FQDN) convention that includes the service name, namespace, and a domain suffix like svc.cluster.local.
    - Example: my-service.my-namespace.svc.cluster.local.
3. **Pod DNS**:
    - Pods are automatically assigned DNS names within the cluster. For example, each pod can be resolved by its name when communicating within the same namespace.
4. **Headless Services**:
    - Headless services (spec.clusterIP: None) are used when you don't need or want a cluster IP address, but still want to take advantage of the service's DNS features. In this case, Kubernetes will create DNS records for each pod backing the service, allowing clients to connect to individual pods directly.
5. **Service Records**:
    - **A Record**: For services, Kubernetes creates an "A" record pointing to the service's cluster IP.
    - **SRV Record**: For headless services, Kubernetes creates "SRV" records, which map the service name to the pod IPs and ports.

1. **CodeDNS**

CoreDNS is the default DNS server in Kubernetes as of version 1.13. It is a flexible, extensible, and scalable DNS server that is used to resolve service names to IP addresses within a Kubernetes cluster.

Features of CoreDNS:

1. **Modular Architecture**: CoreDNS uses a plugin-based architecture, allowing users to add or remove functionalities as needed. Plugins can be used for service discovery, DNS forwarding, caching, load balancing, etc.
2. **Customization**: CoreDNS is highly customizable through its configuration file, Corefile, where you can define various DNS zones, forwarding rules, caching, and more.
3. **Performance**: CoreDNS is optimized for performance and can handle high query loads with low latency.
4. **Extensibility**: CoreDNS can be extended with custom plugins, making it suitable for various use cases beyond just Kubernetes service discovery.

2. **KubeDNS**

KubeDNS is a DNS server and service discovery tool used within Kubernetes clusters to manage DNS resolution for services and pods. It provides a way for services and pods within a Kubernetes cluster to find and communicate with each other using DNS names rather than IP addresses.

Kube-DNS was the DNS server used in Kubernetes before CoreDNS became the default. While it is still supported, it has been largely deprecated in favor of CoreDNS due to CoreDNS's better performance, modularity, and flexibility.

Features of Kube-DNS:

1. **Basic Service Discovery**: Kube-DNS provides basic DNS service discovery within a Kubernetes cluster, allowing services and pods to be resolved by their DNS names.
2. **Simple Configuration**: Kube-DNS has a simpler, less flexible configuration compared to CoreDNS. It does not support plugins or the same level of customization.
3. **Limitations**: Kube-DNS is less performant and scalable than CoreDNS, particularly in large clusters or clusters with complex DNS requirements.

Note:

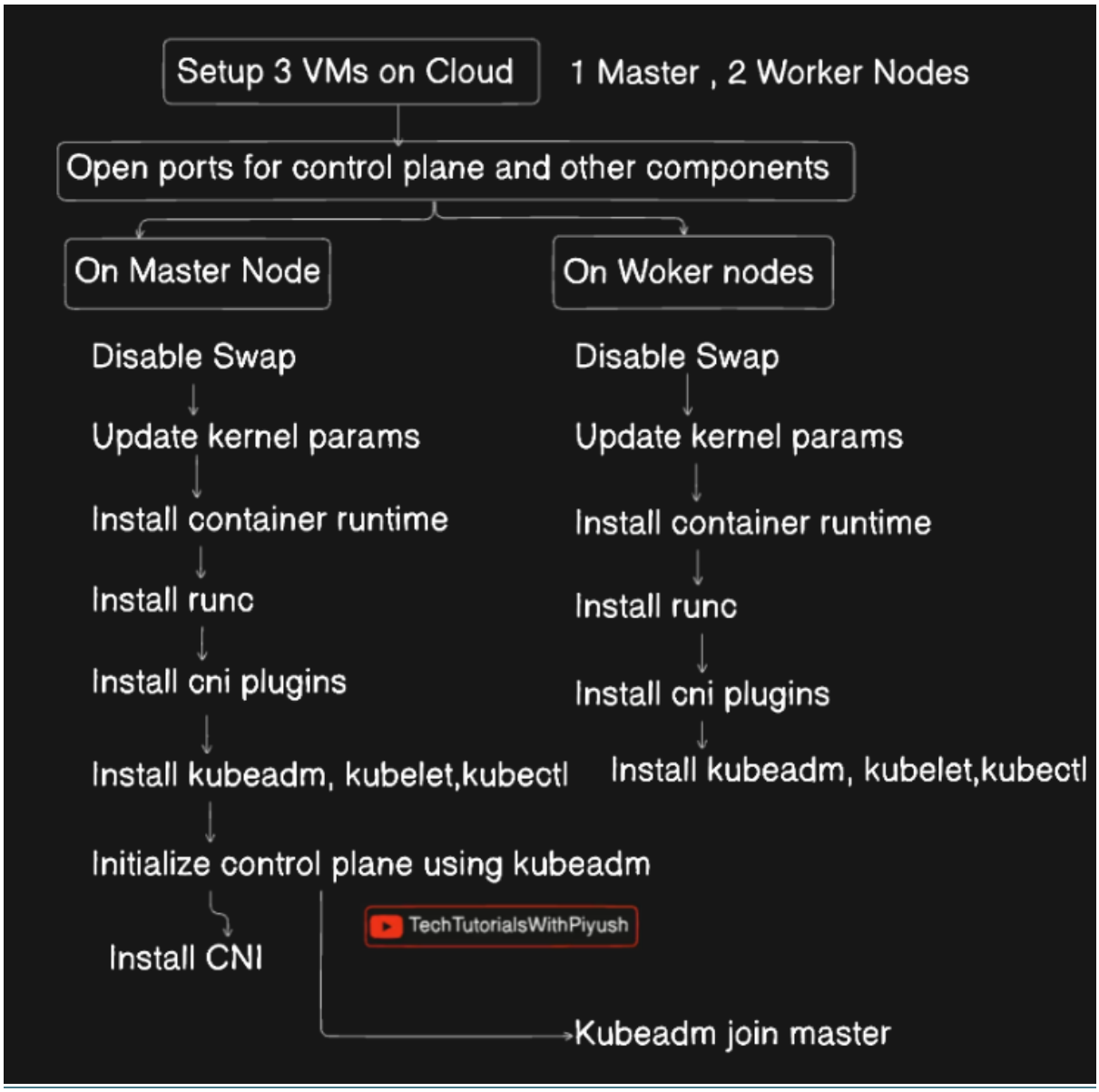
3. Core-DNS is a DNS service/DNS server we are using and Kube-DNS is a Kubernetes service with respect to Core DNS
4. KubeDNS and CoreDNS are related in that CoreDNS has replaced KubeDNS as the default DNS service in Kubernetes clusters, but they are distinct and independent DNS solutions.
5. KubeDNS was the initial DNS service, while CoreDNS is a more advanced and flexible replacement that now serves as the default DNS solution in Kubernetes.

Configure self-managed Multinode cluster using Kubeadm utility (On-premises/Cloud)

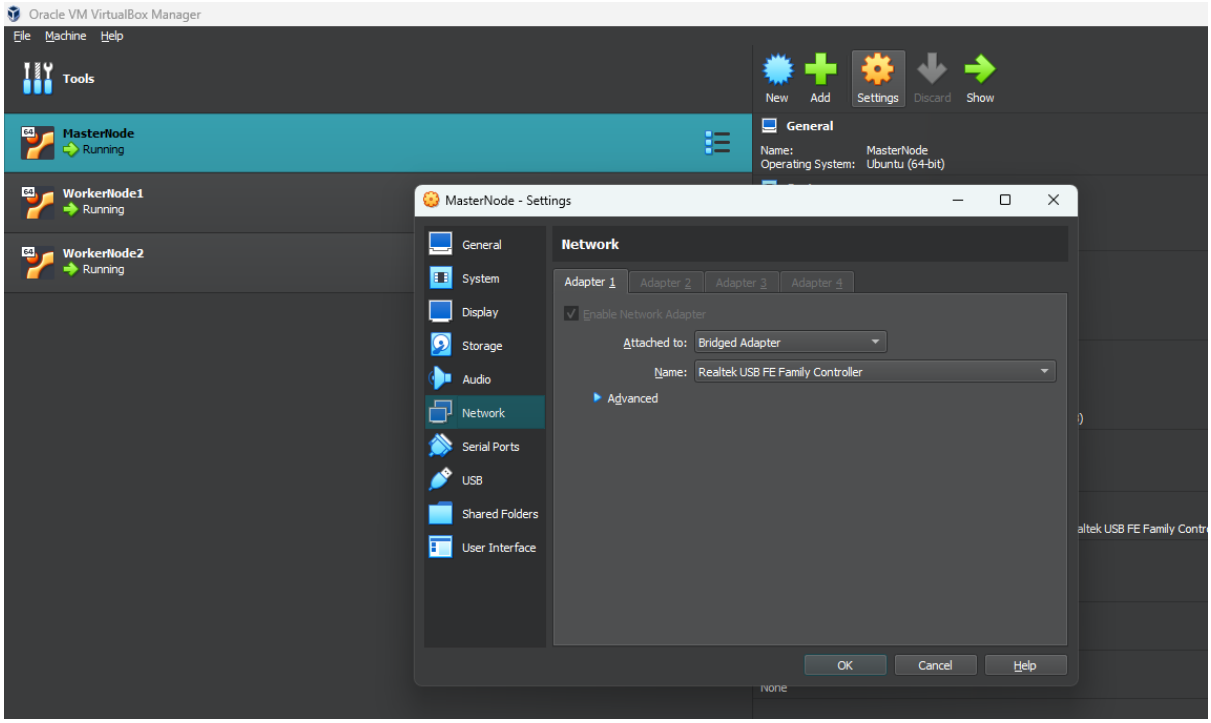
Note: After Kubernetes 1.24 default container runtime is Containerd and not Docker

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

Steps to set up the Kubernetes cluster



1. **Create 3 Virtual Machines using Virtual Box (Ubuntu-22.04.3)**
   MasterNode
   WorkerNode1
   WorkerNode2



Set the Network Adapter as “Bridge Adapter” so that local Machine and all node can communicate with each other. Set up the SSH Server on each Node so that we can work remotely. Execute below command in each node.

Note: Assign a static IP to Master Node.

- ➔ sudo apt-get update
- ➔ sudo apt-get upgrade -y
- ➔ sudo apt-get install -y openssh-server
- ➔ sudo systemctl start ssh
- ➔ sudo systemctl enable ssh
- ➔ sudo apt-get install net-tools
- ➔ ifconfig
- ➔ ping <IP of Node> / ping <hostname of node>

We are successfully able to ping all the nodes from local system.

```
vishal->ping MasterNode

Pinging MasterNode.local [192.168.56.101] with 32 bytes of data:
Reply from 192.168.56.101: bytes=32 time=1ms TTL=64
Reply from 192.168.56.101: bytes=32 time=2ms TTL=64
Reply from 192.168.56.101: bytes=32 time=3ms TTL=64
Reply from 192.168.56.101: bytes=32 time=1ms TTL=64

Ping statistics for 192.168.56.101:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 3ms, Average = 1ms

vishal->ping WorkerNode1

Pinging WorkerNode1.local [fe80::7b49:e3ca:665e:82aa%15] with 32 bytes of data:
Reply from fe80::7b49:e3ca:665e:82aa%15: time<1ms
Reply from fe80::7b49:e3ca:665e:82aa%15: time=2ms
Reply from fe80::7b49:e3ca:665e:82aa%15: time=4ms
Reply from fe80::7b49:e3ca:665e:82aa%15: time<1ms

Ping statistics for fe80::7b49:e3ca:665e:82aa%15:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 4ms, Average = 1ms

vishal->ping WorkerNode2

Pinging WorkerNode2.local [fe80::5576:db9:cb88:ceff%15] with 32 bytes of data:
Reply from fe80::5576:db9:cb88:ceff%15: time=1ms
Reply from fe80::5576:db9:cb88:ceff%15: time=3ms
Reply from fe80::5576:db9:cb88:ceff%15: time<1ms
Reply from fe80::5576:db9:cb88:ceff%15: time=3ms

Ping statistics for fe80::5576:db9:cb88:ceff%15:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 3ms, Average = 1ms

vishal->
```

Also, we are successfully able to connect to All nodes through SSH

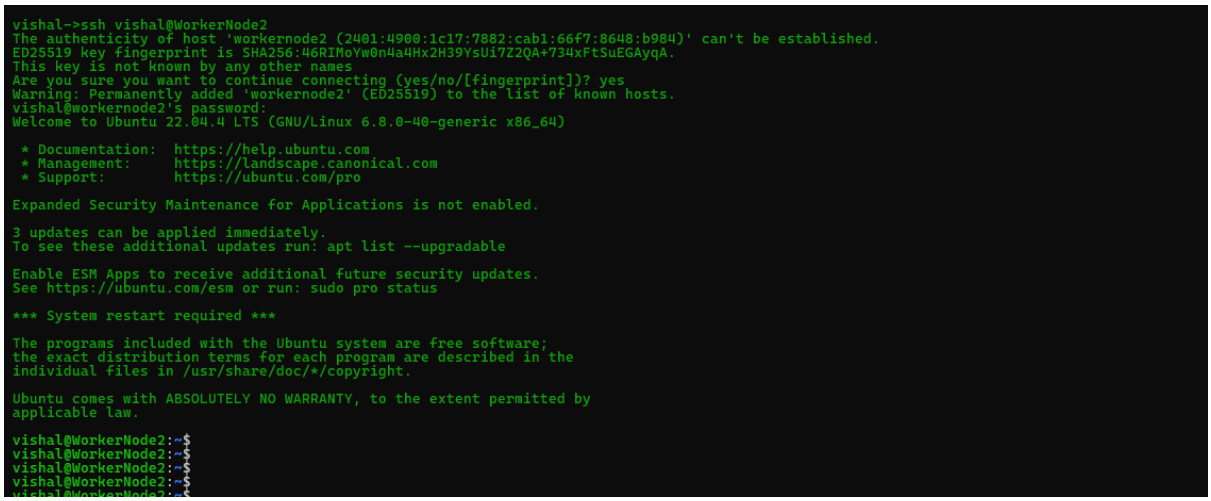
MasterNode

```
vishal->ssh vishal@MasterNode
vishal@masternode's password:
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.8.0-40-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

3 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings

Last login: Sun Aug 25 23:09:40 2024 from 2401:4900:1c17:7882:fcdc:4bab:77f8:d5ae
vishal@MasterNode:~$
vishal@MasterNode:~$
vishal@MasterNode:~$ exit
logout
Connection to masternode closed.
```

WorkerNode1

```
vishal->ssh vishal@WorkerNode1
vishal@workernode1's password:
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.8.0-40-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

3 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

*** System restart required ***
Last login: Sun Aug 25 23:12:16 2024 from 2401:4900:1c17:7882:fcdc:4bab:77f8:d5ae
vishal@WorkerNode1:~$ exit
logout
Connection to workernode1 closed.

vishal->
```

WorkerNode2



2. Open the require ports in each node

https://kubernetes.io/docs/reference/networking/ports-and-protocols/

Command need to execute on Master Node:

 # IP Table for Control plane
sudo iptables -A INPUT -p tcp --dport 6443 -j ACCEPT # Kube-apiserver
sudo iptables -A INPUT -p tcp --dport 2379:2380 -j ACCEPT # Etcd
sudo iptables -A INPUT -p tcp --dport 10250 -j ACCEPT # Kubelet API
sudo iptables -A INPUT -p tcp --dport 10259 -j ACCEPT # Kube-scheduler
sudo iptables -A INPUT -p tcp --dport 10257 -j ACCEPT # Kube-controller-manager

#Save the iptables Rules
sudo iptables-save | sudo tee /etc/iptables/rules.v4

#To load the saved rules on boot, ensure that the iptables-persistent package is installed
sudo apt-get install iptables-persistent

# Immediately apply and enforce changes without needing to reboot
sudo iptables-restore < /etc/iptables/rules.v4

sudo iptables -L

# Check if the Service is Running
sudo netstat -tuln | grep 6443

Command need to execute on Worker Node:

 # IP Table for Worker nodes

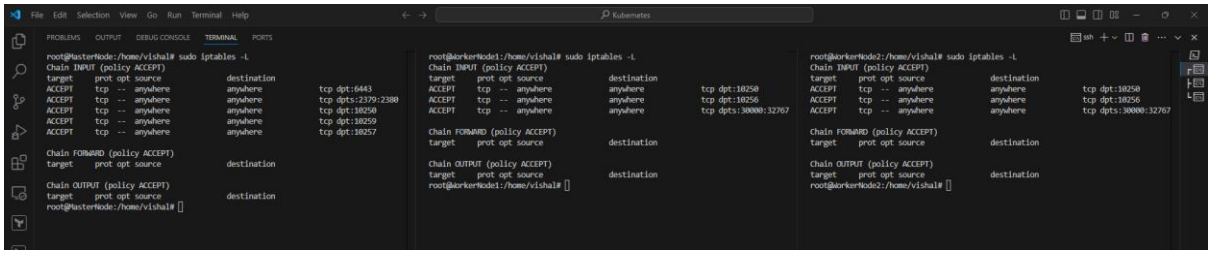
sudo iptables -A INPUT -p tcp --dport 10250 -j ACCEPT # Kubelet API
sudo iptables -A INPUT -p tcp --dport 10256 -j ACCEPT # Kube-proxy
sudo iptables -A INPUT -p tcp --dport 30000:32767 -j ACCEPT # NodePort range

#Save the iptables Rules
sudo iptables-save | sudo tee /etc/iptables/rules.v4

#To load the saved rules on boot, ensure that the iptables-persistent package is installed
sudo apt-get install iptables-persistent

# Immediately apply and enforce changes without needing to reboot
sudo iptables-restore < /etc/iptables/rules.v4

sudo iptables -L



3. Disable the SWAP on each node using the below commands

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

4. Update Kernel Parameters

Forwarding IPv4 and letting iptables see bridged traffic using 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

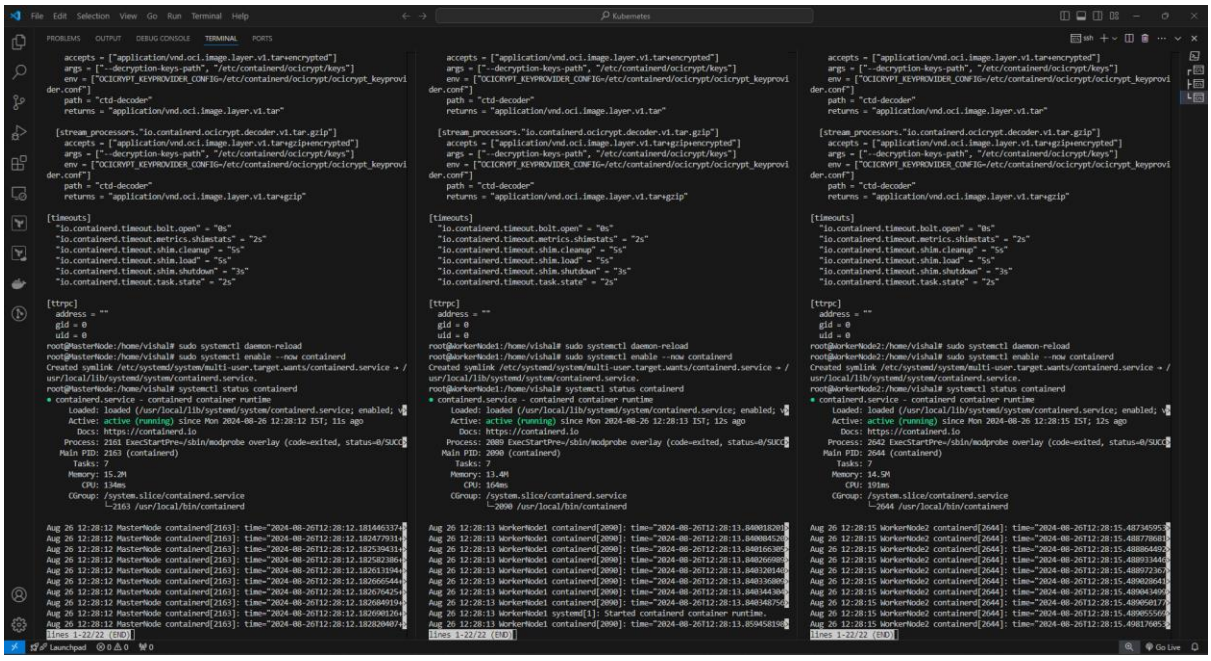
➔ # Verify that the br_netfilter, overlay modules are loaded by running the following cmd:
➔ lsmod | grep br_netfilter
➔ lsmod | grep overlay

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

5. Install container runtime

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

➔ # Check that containerd service is up and running
➔ systemctl status containerd

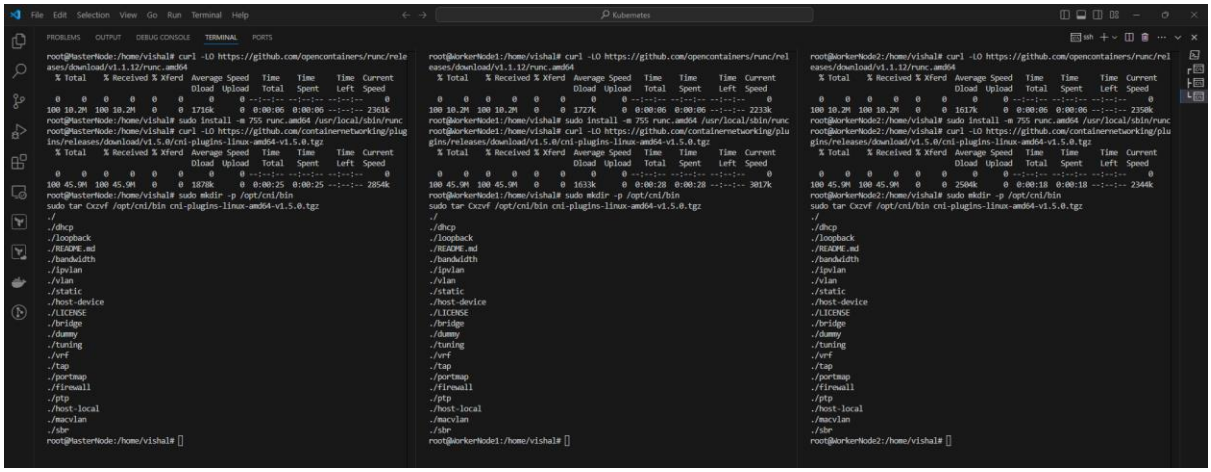


6. **Install runc**

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

7. **Install CNI plugin**

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



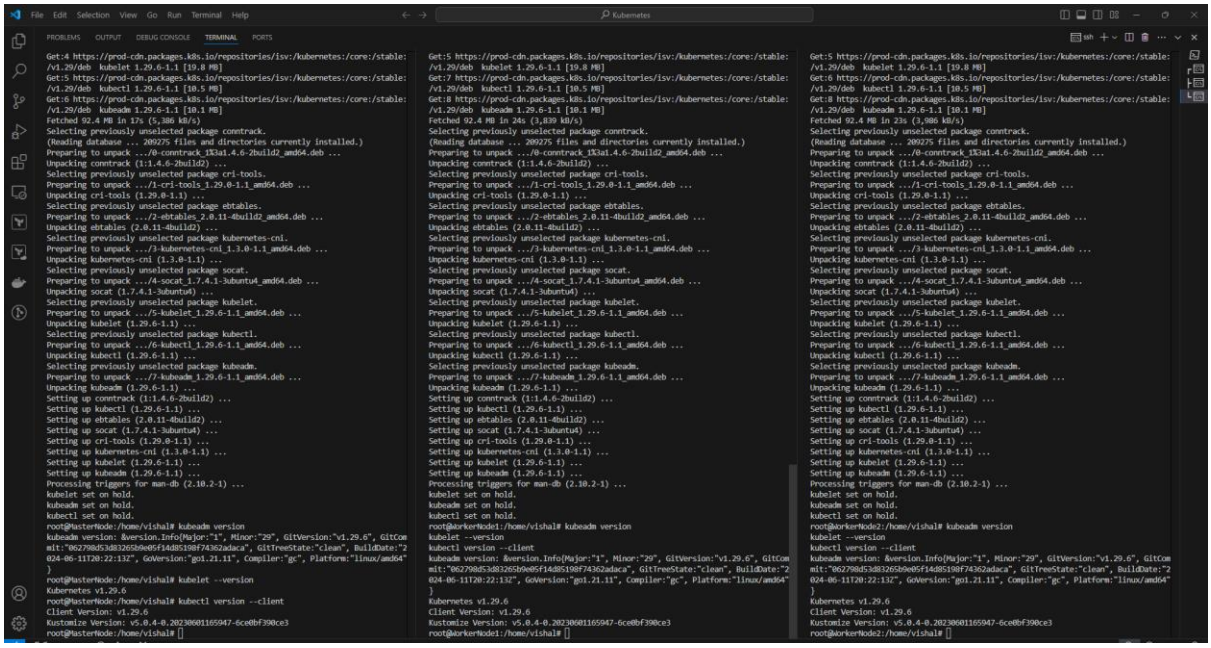
8. **Install kubeadm, kubelet and kubectl**

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

➔ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
➔ echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

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

➔ kubeadm version
➔ kubelet --version
➔ kubectl version --client



9. Configure ‘crictl’ client to work with containerd

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

10. Initialize control plane

➔ sudo kubeadm init --pod-network-cidr=192.168.0.0/16 --apiserver-advertise-address=192.168.1.11 --node-name master

11. Prepare kubeconfig

This commands are provide in the output of previous commands

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

```
kubeadm join 192.168.1.11:6443 --token kyelx8.cjaab7dp8aps6vsd \
        --discovery-token-ca-cert-hash sha256:4b181d2b869b69f5b62c27afd5b22f36eca2
a8635086c0d6083000e01a8bf2a7
root@MasterNode:/home/vishal# mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
root@MasterNode:/home/vishal# kubectl get nodes
NAME     STATUS     ROLES           AGE     VERSION
master   NotReady   control-plane   2m24s   v1.29.6
root@MasterNode:/home/vishal#
```

12. Install calico CNI

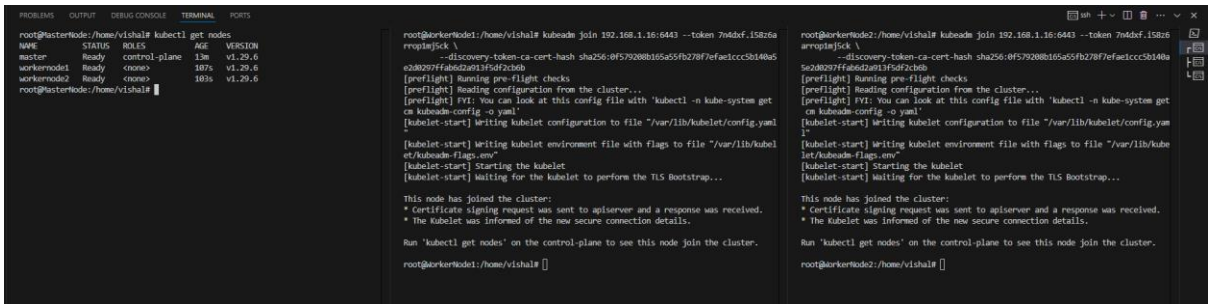
- ➔ kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.28.0/manifests/tigera-operator.yaml
- ➔ curl https://raw.githubusercontent.com/projectcalico/calico/v3.28.0/manifests/custom-resources.yaml -O
- ➔ kubectl apply -f custom-resources.yaml

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

root@MasterNode:/home/vishal# kubectl get nodes
NAME     STATUS   ROLES           AGE     VERSION
master   Ready    control-plane   4m43s   v1.29.6
root@MasterNode:/home/vishal# kubectl get pods -A
NAMESPACE          NAME                                       READY   STATUS    RESTARTS   AGE
calico-apiserver   calico-apiserver-77cdd9957f-72f9t          0/1     Running   0          56s
calico-apiserver   calico-apiserver-77cdd9957f-gkj69          1/1     Running   0          56s
calico-system      calico-kube-controllers-c5cc476b8-gr6rr    1/1     Running   0          3m26s
calico-system      calico-node-77x92                          1/1     Running   0          3m26s
calico-system      calico-typha-645c5c5fcf-98rck              1/1     Running   0          3m26s
calico-system      csi-node-driver-gb6fn                      2/2     Running   0          3m26s
kube-system        coredns-76f75df574-dmmpq                   1/1     Running   0          4m32s
kube-system        coredns-76f75df574-dskkw                   1/1     Running   0          4m32s
kube-system        etcd-master                                1/1     Running   0          4m45s
kube-system        kube-apiserver-master                      1/1     Running   0          4m47s
kube-system        kube-controller-manager-master             1/1     Running   0          4m45s
kube-system        kube-proxy-ggz6x                           1/1     Running   0          4m32s
kube-system        kube-scheduler-master                      1/1     Running   0          4m44s
tigera-operator    tigera-operator-76c4974c85-9b6zx           1/1     Running   0          3m42s
root@MasterNode:/home/vishal#
```

13. Join the worker nodes to Master

Run the command generated in step 10 on the Master node

- ➔ kubeadm join 192.168.1.16:6443 --token 7n4dxf.i58z6arrop1mj5ck --discovery-token-ca-cert-hash sha256:0f579208b165a55fb278f7efae1ccc5b140a5e2d0297ffab6d2a913f5df2cb6b



```
root@MasterNode:/home/vishal# kubectl get nodes
NAME          STATUS   ROLES           AGE     VERSION
master        Ready    control-plane   14m     v1.29.6
workernode1   Ready    <none>          2m48s   v1.29.6
workernode2   Ready    <none>          2m44s   v1.29.6
root@MasterNode:/home/vishal#
```

14. Notes

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

➔ kubeadm token create --print-join-command

If all the above steps were completed, you should be able to run kubectl get nodes on the master node, and it should return all the 3 nodes in ready status.

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

➔ kubectl get pods -A

This is how we have successfully configured Multinode Cluster with Kubernetes Version v1.29.6 using Kubeadm Utility.

Upgrade Kubernetes Multi Node Cluster using Kubeadm

Drain – Evicting all the pods from a particular node

Cordon – Making the node unschedulable

Uncordon - Making the node schedulable

- ➔ kubectl node worker1 drain …… (this will perform drain as well as cordon)
- ➔ kubectl node worker1 uncordon

Kubernetes Version 1.30.2

1 -> Major Release

30 -> Minor Release (2-3 Months)

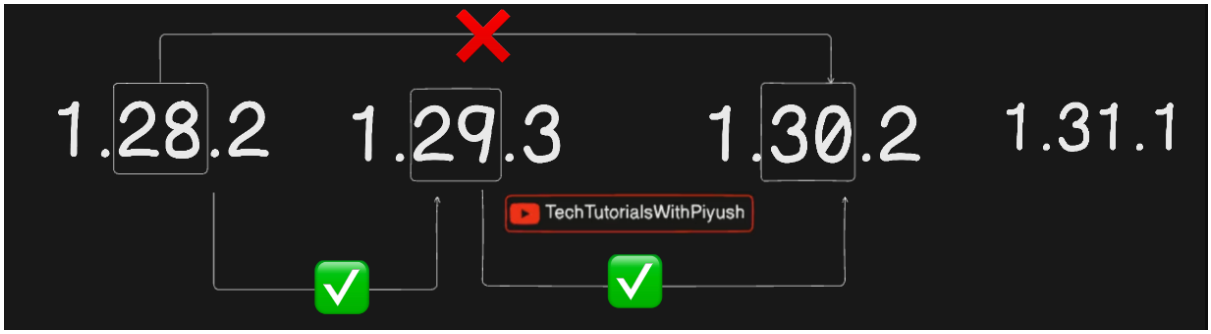
2 -> Patch (Frequent Bug Fixes)

Notes:

1. Kubernetes only support only 3 latest Minor Version that means only 2 previous versions
2. We cannot upgrade the version excluding any intermediate minor version. We need to upgrade it in a sequence.

   Example:
   We have current version 1.28.1 and we need to upgrade the version to 1.30.2 then first we need to upgrade the version to 1.29.1 and then we need to upgrade the version to 1.30.2

3. Suppose kube API server is on version 1.30.X then the other components can be at version as mentioned below else it will create a compatibility issue.
   Scheduler and controller -> X-1
   Kublet and kubectl -> X-2



Node Upgrading Strategies:

1. All at once – This strategy is not recommended
2. Rolling Update – We upgrade one node at a time
3. Blue Green – Adding new nodes

Reference: https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/

Upgrade Control Plane Nodes

1. Prerequisites

If you're using the community-owned package repositories (pkgs.k8s.io), you need to enable the package repository for the desired Kubernetes minor release.

➔ pager /etc/apt/sources.list.d/kubernetes.list

```
root@MasterNode:/home/vishal# kubectl get nodes
^[[ANAME          STATUS   ROLES           AGE   VERSION
master        Ready    control-plane   54m   v1.29.6
workernode1   Ready    <none>          42m   v1.29.6
workernode2   Ready    <none>          42m   v1.29.6
^[[Aroot@MasterNode:/home/vishal# vim /etc/apt/sources.list.d/kubernetes.list
root@MasterNode:/home/vishal#
```

➔ vim /etc/apt/sources.list.d/kubernetes.list

Change the version from v1.29 to v1.30

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /
~
~
~
```

2. Determine which version to upgrade to

Below command will show the available kubeadm packages

➔ sudo apt update
➔ sudo apt-cache madison kubeadm

```
root@MasterNode:/home/vishal# sudo apt update
Hit:1 http://in.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://in.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:3 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.30/deb  InRelease [1,18
6 B]
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:5 http://in.archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:6 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.30/deb  Packages [8,064
 B]
Fetched 266 kB in 2s (121 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
327 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@MasterNode:/home/vishal# sudo apt-cache madison kubeadm
   kubeadm | 1.30.4-1.1 | https://pkgs.k8s.io/core:/stable:/v1.30/deb  Packages
   kubeadm | 1.30.3-1.1 | https://pkgs.k8s.io/core:/stable:/v1.30/deb  Packages
   kubeadm | 1.30.2-1.1 | https://pkgs.k8s.io/core:/stable:/v1.30/deb  Packages
   kubeadm | 1.30.1-1.1 | https://pkgs.k8s.io/core:/stable:/v1.30/deb  Packages
   kubeadm | 1.30.0-1.1 | https://pkgs.k8s.io/core:/stable:/v1.30/deb  Packages
root@MasterNode:/home/vishal#
```

3. Upgrade kubeadm

Kubeadm version Before upgrading is v1.29.6

```
root@MasterNode:/home/vishal# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"29", GitVersion:"v1.29.6", GitCommit:"062798d53d83265b9e05f1
4d85198f74362adaca", GitTreeState:"clean", BuildDate:"2024-06-11T20:22:13Z", GoVersion:"go1.21.11", Compiler:
"gc", Platform:"linux/amd64"}
root@MasterNode:/home/vishal#
```

```
# replace x in 1.31.x-* with the latest patch version
sudo apt-mark unhold kubeadm && \
sudo apt-get update && sudo apt-get install -y kubeadm='1.31.x-*' && \
sudo apt-mark hold kubeadm
```

- ➔ sudo apt-mark unhold kubeadm && \
- ➔ sudo apt-get update && sudo apt-get install -y kubeadm='1.30.4-1.1' && \
- ➔ sudo apt-mark hold kubeadm

```
root@MasterNode:/home/vishal# sudo apt-mark unhold kubeadm && \
sudo apt-get update && sudo apt-get install -y kubeadm='1.30.4-1.1' && \
sudo apt-mark hold kubeadm
Canceled hold on kubeadm.
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:3 http://in.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.30/deb  InRelease
Get:4 http://in.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Hit:5 http://in.archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:6 http://in.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1,987 kB]
Fetched 2,244 kB in 3s (643 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  cri-tools
The following packages will be upgraded:
  cri-tools kubeadm
2 upgraded, 0 newly installed, 0 to remove and 326 not upgraded.
Need to get 31.7 MB of archives.
After this operation, 4,937 kB of additional disk space will be used.
Get:1 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.30/deb  cri-tools 1.30.1-1.1 [21.3 MB]
Get:2 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.30/deb  kubeadm 1.30.4-1.1 [10.4 MB]
Fetched 31.7 MB in 6s (5,024 kB/s)
(Reading database ... 210905 files and directories currently installed.)
Preparing to unpack .../cri-tools_1.30.1-1.1_amd64.deb ...
Unpacking cri-tools (1.30.1-1.1) over (1.29.0-1.1) ...
Preparing to unpack .../kubeadm_1.30.4-1.1_amd64.deb ...
Unpacking kubeadm (1.30.4-1.1) over (1.29.6-1.1) ...
Setting up cri-tools (1.30.1-1.1) ...
Setting up kubeadm (1.30.4-1.1) ...
kubeadm set on hold.
root@MasterNode:/home/vishal# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"30", GitVersion:"v1.30.4", GitCommit:"a51b3b711150f57ffc1f526a640ec058514ed596", GitTreeState:"clean", BuildDate:"2024-08-14T
19:02:46Z", GoVersion:"go1.22.5", Compiler:"gc", Platform:"linux/amd64"}
root@MasterNode:/home/vishal#
```

Post upgrading the kubeadm version is v1.30.4

4. Verify the upgrade plan

➔ sudo kubeadm upgrade plan

```
root@MasterNode:/home/vishal# sudo kubeadm upgrade plan
[preflight] Running pre-flight checks.
[upgrade/config] Reading configuration from the cluster...
[upgrade/config] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[upgrade] Running cluster health checks
[upgrade] Fetching available versions to upgrade to
[upgrade/versions] Cluster version: 1.29.8
[upgrade/versions] kubeadm version: v1.30.4
W0826 16:08:23.720491   20223 version.go:104] could not fetch a Kubernetes version from the internet: unable to get URL "https://dl.k8s.io/release/stable.txt": Get "https://c
dn.dl.k8s.io/release/stable.txt": context deadline exceeded (Client.Timeout exceeded while awaiting headers)
W0826 16:08:23.736733   20223 version.go:105] falling back to the local client version: v1.30.4
[upgrade/versions] Target version: v1.30.4
W0826 16:08:33.925849   20223 version.go:104] could not fetch a Kubernetes version from the internet: unable to get URL "https://dl.k8s.io/release/stable-1.29.txt": Get "http
s://cdn.dl.k8s.io/release/stable-1.29.txt": context deadline exceeded (Client.Timeout exceeded while awaiting headers)
W0826 16:08:33.925905   20223 version.go:105] falling back to the local client version: v1.30.4
[upgrade/versions] Latest version in the v1.29 series: v1.30.4

Components that must be upgraded manually after you have upgraded the control plane with 'kubeadm upgrade apply':
COMPONENT   NODE          CURRENT   TARGET
kubelet     master        v1.29.6   v1.30.4
kubelet     workernode1   v1.29.6   v1.30.4
kubelet     workernode2   v1.29.6   v1.30.4

Upgrade to the latest version in the v1.29 series:

COMPONENT                 NODE      CURRENT    TARGET
kube-apiserver            master    v1.29.8    v1.30.4
kube-controller-manager   master    v1.29.8    v1.30.4
kube-scheduler            master    v1.29.8    v1.30.4
kube-proxy                          1.29.8     v1.30.4
CoreDNS                             v1.11.1    v1.11.1
etcd                      master    3.5.12-0   3.5.12-0

You can now apply the upgrade by executing the following command:

        kubeadm upgrade apply v1.30.4

_____________________________________________________________________


The table below shows the current state of component configs as understood by this version of kubeadm.
Configs that have a "yes" mark in the "MANUAL UPGRADE REQUIRED" column require manual config upgrade or
resetting to kubeadm defaults before a successful upgrade can be performed. The version to manually
upgrade to is denoted in the "PREFERRED VERSION" column.

API GROUP                 CURRENT VERSION   PREFERRED VERSION   MANUAL UPGRADE REQUIRED
kubeproxy.config.k8s.io   v1alpha1          v1alpha1            no
kubelet.config.k8s.io     v1beta1           v1beta1             no
_____________________________________________________________________

root@MasterNode:/home/vishal#
```

Kubelet needs to be upgrade manually after the update is done

5. Execute the upgrade (This will Upgrade the control plane components)

➔ kubeadm upgrade apply v1.30.4

```
[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
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.30.4". Enjoy!

[upgrade/kubelet] Now that your control plane is upgraded, please proceed with upgrading your kubelets if you haven't already done so.
root@MasterNode:/home/vishal#
```

We have successfully upgraded the cluster to version v1.30.4

6. Drain the Nodes before upgrading kubelet and kubectl

kubectl drain master --ignore-daemonsets

```
root@MasterNode:/home/vishal# kubectl get nodes
NAME          STATUS   ROLES           AGE   VERSION
master        Ready    control-plane   97m   v1.29.6
workernode1   Ready    <none>          85m   v1.29.6
workernode2   Ready    <none>          85m   v1.29.6
root@MasterNode:/home/vishal# kubectl drain master --ignore-daemonsets
node/master cordoned
Warning: ignoring DaemonSet-managed Pods: calico-system/calico-node-77x92, calico-system/csi-node-driver-gb6fn, kube-system/kube-proxy-ggz6x
evicting pod tigera-operator/tigera-operator-76c4974c85-9b6zx
evicting pod calico-system/calico-kube-controllers-c5cc476b8-gr6rr
evicting pod calico-apiserver/calico-apiserver-77cdd9957f-72f9t
evicting pod calico-apiserver/calico-apiserver-77cdd9957f-gkj69
evicting pod kube-system/coredns-76f75df574-dmmpq
evicting pod calico-system/calico-typha-645c5c5fcf-98rck
evicting pod kube-system/coredns-76f75df574-dskkw
I0826 16:31:13.198940   25036 request.go:697] Waited for 1.123804547s due to client-side throttling, not priority and fairness, request: GET:https://192.168.1.16:6443/api/v1/
namespaces/tigera-operator/pods/tigera-operator-76c4974c85-9b6zx
pod/tigera-operator-76c4974c85-9b6zx evicted
pod/calico-kube-controllers-c5cc476b8-gr6rr evicted
pod/calico-apiserver-77cdd9957f-gkj69 evicted
pod/calico-typha-645c5c5fcf-98rck evicted
pod/calico-apiserver-77cdd9957f-72f9t evicted
pod/coredns-76f75df574-dskkw evicted
pod/coredns-76f75df574-dmmpq evicted
node/master drained
root@MasterNode:/home/vishal# kubectl get nodes
NAME          STATUS                     ROLES           AGE   VERSION
master        Ready,SchedulingDisabled   control-plane   98m   v1.29.6
workernode1   Ready                      <none>          86m   v1.29.6
workernode2   Ready                      <none>          86m   v1.29.6
root@MasterNode:/home/vishal#
```

7. Upgrade kubelet and kubectl

```
# replace x in 1.31.x-* with the latest patch version
sudo apt-mark unhold kubelet kubectl && \
sudo apt-get update && sudo apt-get install -y kubelet='1.31.x-*'
kubectl='1.31.x-*' && \
sudo apt-mark hold kubelet kubectl
```

- ➔ sudo apt-mark unhold kubelet kubectl && \
- ➔ sudo apt-get update && sudo apt-get install -y kubelet='1.30.4-1.1' kubectl='1.30.4-1.1' && \
- ➔ sudo apt-mark hold kubelet kubectl

8. Restart the Kublet

sudo systemctl daemon-reload
sudo systemctl restart kubelet

```
root@MasterNode:/home/vishal# sudo apt-mark unhold kubelet kubectl && \
sudo apt-get update && sudo apt-get install -y kubelet='1.30.4-1.1' kubectl='1.30.4-1.1' && \
sudo apt-mark hold kubelet kubectl
Canceled hold on kubelet.
Canceled hold on kubectl.
Hit:1 http://in.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:4 http://in.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:5 http://in.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:3 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.30/deb  InRelease
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages will be upgraded:
  kubectl kubelet
2 upgraded, 0 newly installed, 0 to remove and 324 not upgraded.
Need to get 28.9 MB of archives.
After this operation, 10.2 MB disk space will be freed.
Get:1 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.30/deb  kubectl 1.30.4-1.1 [10.8 MB]
Get:2 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.30/deb  kubelet 1.30.4-1.1 [18.1 MB]
Fetched 28.9 MB in 6s (4,983 kB/s)
(Reading database ... 210905 files and directories currently installed.)
Preparing to unpack .../kubectl_1.30.4-1.1_amd64.deb ...
Unpacking kubectl (1.30.4-1.1) over (1.29.6-1.1) ...
Preparing to unpack .../kubelet_1.30.4-1.1_amd64.deb ...
Unpacking kubelet (1.30.4-1.1) over (1.29.6-1.1) ...
Setting up kubectl (1.30.4-1.1) ...
Setting up kubelet (1.30.4-1.1) ...
kubelet set on hold.
kubectl set on hold.
root@MasterNode:/home/vishal# sudo systemctl daemon-reload
root@MasterNode:/home/vishal# sudo systemctl restart kubelet
root@MasterNode:/home/vishal#
```

9. Uncordon the MasterNode

➔ Kubectl uncordon master

```
root@MasterNode:/home/vishal# kubectl get nodes
NAME          STATUS                     ROLES           AGE    VERSION
master        Ready,SchedulingDisabled   control-plane   106m   v1.30.4
workernode1   Ready                      <none>          94m    v1.29.6
workernode2   Ready                      <none>          94m    v1.29.6
root@MasterNode:/home/vishal# kubectl version
Client Version: v1.30.4
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
Server Version: v1.30.4
root@MasterNode:/home/vishal# kubectl uncordon master
node/master uncordoned
root@MasterNode:/home/vishal# kubectl get nodes
NAME          STATUS   ROLES           AGE    VERSION
master        Ready    control-plane   107m   v1.30.4
workernode1   Ready    <none>          95m    v1.29.6
workernode2   Ready    <none>          95m    v1.29.6
root@MasterNode:/home/vishal#
```

We have successfully upgraded the cluster and all the control plain components to version v1.30.4

Upgrade worker nodes

Below steps are same as we perform on Master Node

1. **Prerequisites**

If you're using the community-owned package repositories (pkgs.k8s.io), you need to enable the package repository for the desired Kubernetes minor release.

- ➔ pager /etc/apt/sources.list.d/kubernetes.list
- ➔ vim /etc/apt/sources.list.d/kubernetes.list

Change the version from v1.29 to v1.30

2. **Upgrade kubeadm**

- ➔ sudo apt-mark unhold kubeadm && \
- ➔ sudo apt-get update && sudo apt-get install -y kubeadm='1.30.4-1.1' && \
- ➔ sudo apt-mark hold kubeadm

3. **Upgrades the local kubelet configuration**

- ➔ sudo kubeadm upgrade node

4. **Drain the WorkerNode 1 from MasterNode**

- ➔ kubectl drain WorkerNode1 --ignore-daemonsets

5. **Upgrade kubelet and kubectl**

- ➔ sudo kubeadm upgrade node

6. **Restart the Kublet**

- ➔ sudo systemctl daemon-reload
- ➔ sudo systemctl restart kubelet

7. **Uncordon the MasterNode**

- ➔ Kubectl uncordon master

We have successfully upgraded the WorkerNode1 to version v1.30.4

ETCD Backup and Restore

ETCD – It's a key value data storage, which store all the cluster state, configuration data and all the manifest object in the Kubernetes cluster in a key value database. ETCD is a source of everything running in a cluster and backing up ETCD is sufficient. And we can restore this backup whenever we need. **Data Directory** /var/lib/etcd

We only get access to ETCD on a self-managed cluster. No any cloud we cannot have access on ETCD. In this case we user third part tools.

```
root@MasterNode:/home/vishal# cd /etc/k
kernel/          kernel-img.conf  kerneloops.conf  kubernetes/
root@MasterNode:/home/vishal# cd /etc/kubernetes/manifests/
root@MasterNode:/etc/kubernetes/manifests# ls -lrt
total 16
-rw------- 1 root root 2394 Aug 27 14:18 etcd.yaml
-rw------- 1 root root 4032 Aug 27 14:18 kube-apiserver.yaml
-rw------- 1 root root 3544 Aug 27 14:18 kube-controller-manager.yaml
-rw------- 1 root root 1463 Aug 27 14:18 kube-scheduler.yaml
root@MasterNode:/etc/kubernetes/manifests# less etcd.yaml
```

ETCD has 2 volume mounts

Data Directory: /var/lib/etcd -> In this directory all the configuration data is stored

Cert Directory: /etc/kubernetes/pki/etcd -> In this directory all the cert related to ETCD is stored

```
    volumeMounts:
    - mountPath: /var/lib/etcd
      name: etcd-data
    - mountPath: /etc/kubernetes/pki/etcd
      name: etcd-certs
```

We need to take backup of data directory with the help of utility called **etcdctl**. This utility interacts with ETCD and perform task like backup and restore.

```
root@MasterNode:/etc/kubernetes/manifests# etcdctl
Command 'etcdctl' not found, but can be installed with:
snap install etcd         # version 3.4.22, or
apt  install etcd-client  # version 3.3.25+dfsg-7ubuntu0.22.04.1
See 'snap info etcd' for additional versions.
root@MasterNode:/etc/kubernetes/manifests#
SSH: MasterNode  ⊗ 0 ⚠ 0  1
```

1. Install this ‘etcdctl’ client utility

sudo apt install etcd-client

```
root@MasterNode:/etc/kubernetes/manifests# sudo apt install etcd-client
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  etcd-client
0 upgraded, 1 newly installed, 0 to remove and 325 not upgraded.
Need to get 4,575 kB of archives.
After this operation, 15.3 MB of additional disk space will be used.
Get:1 http://in.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 etcd-client amd64 3.3.25+dfsg-7ubuntu0.22.04.1 [4,575 kB]
Fetched 4,575 kB in 2s (2,692 kB/s)
Selecting previously unselected package etcd-client.
(Reading database ... 210905 files and directories currently installed.)
Preparing to unpack .../etcd-client_3.3.25+dfsg-7ubuntu0.22.04.1_amd64.deb ...
Unpacking etcd-client (3.3.25+dfsg-7ubuntu0.22.04.1) ...
Setting up etcd-client (3.3.25+dfsg-7ubuntu0.22.04.1) ...
Processing triggers for man-db (2.10.2-1) ...
root@MasterNode:/etc/kubernetes/manifests# etcdctl
NAME:
   etcdctl - A simple command line client for etcd.

WARNING:
   Environment variable ETCDCTL_API is not set; defaults to etcdctl v2.
   Set environment variable ETCDCTL_API=3 to use v3 API or ETCDCTL_API=2 to use v2 API.

USAGE:
MasterNode  0 0  1
```

We always need to pass below environment variable along with the command. This will use the latest version of client utility. etcd supports built-in snapshot. A snapshot may either be created from a live member with the etcdctl snapshot save command or by copying the member/snap/db file from an etcd data directory that is not currently used by an etcd process. Creating the snapshot will not affect the performance of the member.

Env Variable: ETCDCTL_API=3

ETCDCTL_API=3 etcdctl snapshot

```
root@MasterNode:/etc/kubernetes/manifests# ETCDCTL_API=3 etcdctl snapshot
NAME:
        snapshot - Manages etcd node snapshots

USAGE:
        etcdctl snapshot <subcommand> [flags]

API VERSION:
        3.3


COMMANDS:
        restore Restores an etcd member snapshot to an etcd directory
        save    Stores an etcd node backend snapshot to a given file
        status  Gets backend snapshot status of a given file

OPTIONS:
  -h, --help[=false]    help for snapshot

GLOBAL OPTIONS:
      --cacert=""                               verify certificates of TLS-enabled secure servers using this CA bundle
      --cert=""                                 identify secure client using this TLS certificate file
      --command-timeout=5s                      timeout for short running command (excluding dial timeout)
      --debug[=false]                           enable client-side debug logging
      --dial-timeout=2s                         dial timeout for client connections
  -d, --discovery-srv=""                        domain name to query for SRV records describing cluster endpoints
      --endpoints=[127.0.0.1:2379]              gRPC endpoints
      --hex[=false]                             print byte strings as hex encoded strings
      --insecure-discovery[=true]               accept insecure SRV records describing cluster endpoints
      --insecure-skip-tls-verify[=false]        skip server certificate verification (CAUTION: this option should be enabled only for testing purposes)
      --insecure-transport[=true]               disable transport security for client connections
      --keepalive-time=2s                       keepalive time for client connections
      --keepalive-timeout=6s                    keepalive timeout for client connections
      --key=""                                  identify secure client using this TLS key file
      --user=""                                 username[:password] for authentication (prompt if password is not supplied)
  -w, --write-out="simple"                      set the output format (fields, json, protobuf, simple, table)
```

2. Take Snapshot using etcdctl

ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 ****
--cacert=<trusted-ca-file> \
--cert=<cert-file> \
--key=<key-file> \
snapshot save <backup-file-location>

ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 ****
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key \
snapshot save /opt/etcd-backup.db

```
root@MasterNode:/etc/kubernetes/manifests# cat etcd.yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubeadm.kubernetes.io/etcd.advertise-client-urls: https://192.168.1.12:2379
  creationTimestamp: null
  labels:
    component: etcd
    tier: control-plane
  name: etcd
  namespace: kube-system
spec:
  containers:
  - command:
    - etcd
    - --advertise-client-urls=https://192.168.1.12:2379
    - --cert-file=/etc/kubernetes/pki/etcd/server.crt
    - --client-cert-auth=true
    - --data-dir=/var/lib/etcd
    - --experimental-initial-corrupt-check=true
    - --experimental-watch-progress-notify-interval=5s
    - --initial-advertise-peer-urls=https://192.168.1.12:2380
    - --initial-cluster=master=https://192.168.1.12:2380
    - --key-file=/etc/kubernetes/pki/etcd/server.key
    - --listen-client-urls=https://127.0.0.1:2379,https://192.168.1.12:2379
    - --listen-metrics-urls=http://127.0.0.1:2381
    - --listen-peer-urls=https://192.168.1.12:2380
    - --name=master
    - --peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt
    - --peer-client-cert-auth=true
    - --peer-key-file=/etc/kubernetes/pki/etcd/peer.key
    - --peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    - --snapshot-count=10000
    - --trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    image: registry.k8s.io/etcd:3.5.12-0
    imagePullPolicy: IfNotPresent
    livenessProbe:
      failureThreshold: 8
      httpGet:
        host: 127.0.0.1
        path: /health?exclude=NOSPACE&serializable=true
        port: 2381
```

```
root@MasterNode:/etc/kubernetes/manifests# ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key \
  snapshot save /opt/etcd-backup.db
2024-08-28 18:54:15.840673 I | clientv3: opened snapshot stream; downloading
2024-08-28 18:54:16.115067 I | clientv3: completed snapshot read; closing
Snapshot saved at /opt/etcd-backup.db
root@MasterNode:/etc/kubernetes/manifests#
```

Backup has been taken successfully

```
root@MasterNode:/etc/kubernetes/manifests# ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key \
  snapshot save /opt/etcd-backup.db
2024-08-28 18:54:15.840673 I | clientv3: opened snapshot stream; downloading
2024-08-28 18:54:16.115067 I | clientv3: completed snapshot read; closing
Snapshot saved at /opt/etcd-backup.db
root@MasterNode:/etc/kubernetes/manifests# cd /opt/
root@MasterNode:/opt# ls
cni  containerd  etcd-backup.db
root@MasterNode:/opt# du -sh /opt/etcd-backup.db
7.8M    /opt/etcd-backup.db
root@MasterNode:/opt#
```

3. Restore the Snapshot using etcdctl

ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 ****
--cacert=<trusted-ca-file> \
--cert=<cert-file> \
--key=<key-file> \
snapshot restore <backup-file-location> \
--data-dir=<data-dir-location>

(--data-dir= …… New Data Directory you want to create)

ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 ****
--cacert=/etc/kubernetes/pki/etcd/ca.crt \
--cert=/etc/kubernetes/pki/etcd/server.crt \
--key=/etc/kubernetes/pki/etcd/server.key \
 snapshot restore /opt/etcd-backup.db \
--data-dir=/var/lib/etcd-restored-from-backup

Post restore make sure to restart the API server and Kubelet

Kubernetes logging and Monitoring

Logging in Kubernetes

Kubernetes itself does not come with a built-in centralized logging solution, but logs can be accessed and managed in several ways:

1. Accessing Logs from Pods

Command: kubectl logs <pod-name> [-c <container-name>] [--namespace <namespace>]

Description: This command fetches the logs from a specific pod. If a pod has multiple containers, use the -c flag to specify the container name. You can also use the --namespace flag to specify the namespace.

2. Streaming Logs in Real-Time

Command: kubectl logs -f <pod-name> [-c <container-name>] [--namespace <namespace>]

Description: Use the -f flag to stream logs in real-time from a specific pod or container. This is useful for monitoring logs as they are being generated.

3. Getting Logs from All Pods in a Namespace

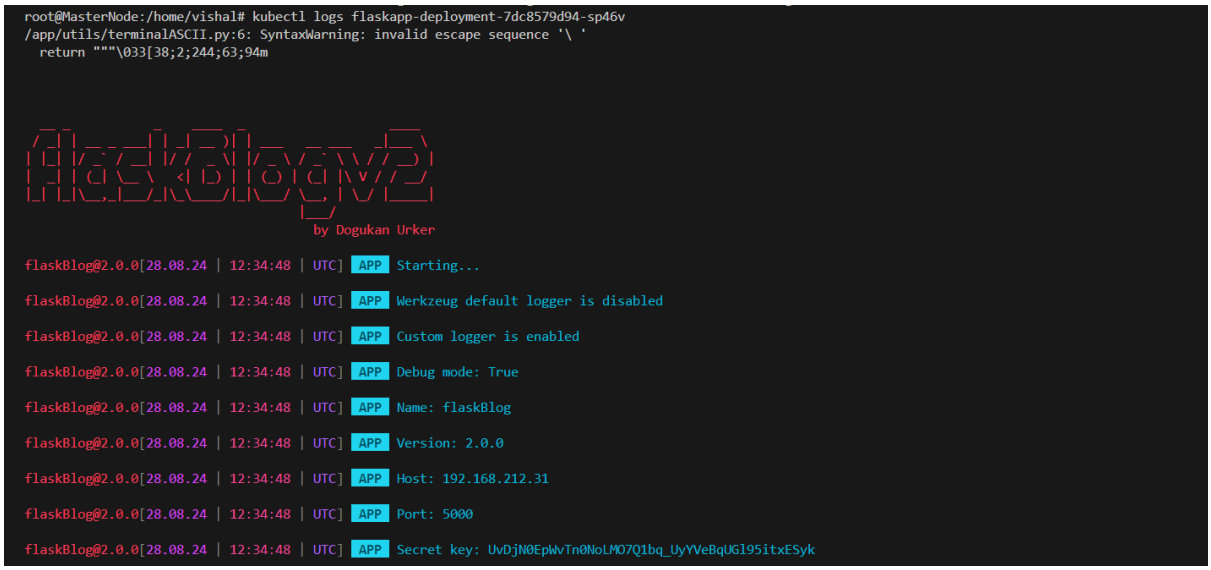
Command: kubectl logs -l <label-selector> --all-containers=true --namespace=<namespace>

Description: Use label selectors to fetch logs from all pods in a namespace that match a specific label.

4. Cluster Component Logs

Command: journalctl -u kubelet

Description: Use this command on a node to access logs from the kubelet service. journalctl is a systemd utility to view logs for system services.



Centralized Logging Solutions

While the above methods are useful for basic logging tasks, a more sophisticated and scalable approach involves using centralized logging solutions. These tools collect, aggregate, store, and analyze logs from across the Kubernetes cluster.

1. **ELK Stack (Elasticsearch, Logstash, Kibana)**
    - **Description**: The ELK stack is a popular open-source solution for log management. Logs from Kubernetes pods and nodes are collected using **Filebeat** or **Fluentd**, processed by **Logstash**, stored in **Elasticsearch**, and visualized in **Kibana**.
    - **Components**:
        - **Filebeat/Fluentd**: Collects logs from pods and nodes.
        - **Logstash**: Processes and transforms logs.
        - **Elasticsearch**: Stores logs for querying and analysis.
        - **Kibana**: Provides a web interface for searching and visualizing logs.
2. **Fluentd and Fluent Bit**
    - **Description**: **Fluentd** is an open-source data collector, and **Fluent Bit** is its lightweight counterpart. They are commonly used for log collection and forwarding in Kubernetes environments. Logs can be forwarded to various destinations like Elasticsearch, CloudWatch, or other storage backends.
    - **Usage**: Deploy Fluentd/Fluent Bit as DaemonSets in your cluster to collect logs from all nodes and forward them to a central location.
3. **Promtail and Loki (Grafana Stack)**
    - **Description**: **Promtail** is an agent that collects logs from Kubernetes nodes and forwards them to **Loki**, a log aggregation system developed by Grafana. Logs can then be visualized and queried in **Grafana**.
    - **Usage**: Loki is designed to work well with Prometheus and Grafana, providing an integrated solution for monitoring metrics and logs together.

Monitoring in Kubernetes

Metrics Server in Kubernetes is primarily focused on gathering real-time resource usage metrics related to **CPU** and **memory** for nodes and pods in the cluster. These metrics are crucial for tasks like horizontal pod autoscaling, monitoring resource utilization, and understanding the current load on your Kubernetes environment.

1. **Metrics Provided by Metrics Server**

A. **CPU Usage**:

Nodes: The CPU usage of each node is reported in millicores (m), where 1000m equals one CPU core.

Pods: The CPU usage of each pod, aggregated across all containers within that pod, is reported in millicores (m).

B. **Memory Usage**:

Nodes: The memory usage of each node is reported in bytes (e.g., MiB, GiB).

Pods: The memory usage of each pod, aggregated across all containers within that pod, is reported in bytes.

2. **Limitations of Metrics Server**

- **No Disk I/O, Network, or Custom Metrics**: The Metrics Server does not gather metrics related to disk I/O, network usage, or other custom metrics. It's limited to CPU and memory usage only.
- **No Historical Data**: Metrics Server only provides real-time metrics and does not store historical data. If you need to analyze trends or historical performance, you would need to use a more comprehensive monitoring solution like Prometheus, which can store and query historical data.

3. **Commands to Use with Metrics Server**

1. **Check Nodes Metrics:**

kubectl top nodes

Description: Displays the current CPU and memory usage for each node in the cluster.

2. **Check Pods Metrics:**

kubectl top pods

Description: Shows the current CPU and memory usage for each pod in the default namespace.

3. **Check Pods' Metrics in a Specific Namespace:**

kubectl top pods -n <namespace>

Description: Displays the current CPU and memory usage for each pod in a specified namespace.

4. **Check Metrics for a Specific Pod:**

kubectl top pod <pod-name> --namespace=<namespace>

Description: Retrieves the current CPU and memory usage for a specific pod.

5. **Check Metrics for a Specific Node:**

kubectl top node <node-name>

Description: Displays the current CPU and memory usage for a specific node.

6. **List All Nodes and Their Metrics:**

kubectl top nodes --sort-by=cpu

Description: Lists all nodes and sorts them by CPU usage.

7. **List All Pods and Their Metrics Sorted by CPU/Memory Usage:**

kubectl top pods --sort-by=cpu

kubectl top pods --sort-by=memory

Description: Sorts and lists all pods by either CPU or memory usage.

Install the Metrics Server

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

vishal@LAPTOP-NGR39AL4:/$ kubectl get pods -n kube-system --no-headers=false | grep metric
metrics-server-bcffdb84f-mcd2z                          1/1     Running   10 (3h55m ago)   37d
vishal@LAPTOP-NGR39AL4:/$
vishal@LAPTOP-NGR39AL4:/$
vishal@LAPTOP-NGR39AL4:/$ kubectl get pods -n kube-system | (head -n 1 && grep metric)
NAME                                                    READY   STATUS    RESTARTS         AGE
metrics-server-bcffdb84f-mcd2z                          1/1     Running   10 (3h55m ago)   37d
vishal@LAPTOP-NGR39AL4:/$
```

```
vishal@LAPTOP-NGR39AL4:~$ kubectl top nodes
NAME                           CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
cka-mn-cluster-1-control-plane   141m         1%     611Mi           15%
cka-mn-cluster-1-worker          43m          0%     285Mi           7%
cka-mn-cluster-1-worker2         30m          0%     182Mi           4%
vishal@LAPTOP-NGR39AL4:~$ kubectl top pods
NAME                                 CPU(cores)   MEMORY(bytes)
flaskapp-deployment-c65d985d5-htzdr   3m           63Mi
vishal@LAPTOP-NGR39AL4:~$ kubectl top pods -n kube-system
NAME                                                    CPU(cores)   MEMORY(bytes)
coredns-76f75df574-778qb                                2m           19Mi
coredns-76f75df574-kjf5x                                3m           20Mi
etcd-cka-mn-cluster-1-control-plane                     24m          52Mi
kindnet-9576w                                           1m           16Mi
kindnet-hwz64                                           1m           20Mi
kindnet-z6hmv                                           1m           20Mi
kube-apiserver-cka-mn-cluster-1-control-plane           60m          221Mi
kube-controller-manager-cka-mn-cluster-1-control-plane  20m          57Mi
kube-proxy-5k2wc                                        3m           24Mi
kube-proxy-qdk1d                                        1m           27Mi
kube-proxy-zw2km                                        1m           26Mi
kube-scheduler-cka-mn-cluster-1-control-plane           4m           28Mi
metrics-server-bcffdb84f-mcd2z                          4m           37Mi
vishal@LAPTOP-NGR39AL4:~$
```

4. Use Cases for Metrics Server

- **Horizontal Pod Autoscaler (HPA)**: Metrics Server is commonly used by the HPA to scale pods based on CPU and memory usage.
- **kubectl top Command**: It powers the kubectl top command, which is used to display resource usage metrics for nodes and pods.

Other Monitoring Solutions

If you need more detailed metrics beyond CPU and memory (e.g., disk I/O, network traffic, custom application-level metrics), or if you need historical data for analysis and alerting, you would typically integrate other monitoring tools like **Prometheus**, **Grafana**, or **ELK Stack** into your Kubernetes environment.

- **Prometheus**: For more extensive monitoring and alerting, including custom metrics, historical data, and advanced querying.
- **Grafana**: For visualization of metrics collected by Prometheus or other data sources.
- **ELK Stack**: For centralized log management, which can complement metrics gathered by Metrics Server.

5. Prometheus

Overview: Prometheus is an open-source monitoring and alerting toolkit, widely adopted for its efficiency and powerful features in collecting and storing time-series data. Originally developed by SoundCloud, Prometheus is now a part of the Cloud Native Computing Foundation (CNCF) and has become a standard for monitoring containerized environments like Kubernetes.

Key Features:

- **Time-Series Database**: Prometheus stores metrics as time-series data, meaning it records a series of data points over time. Each time-series is identified by a metric name and a set of key-value pairs called labels.
- **Multi-Dimensional Data Model**: Prometheus supports a flexible data model with labels, allowing you to group and filter metrics effectively. This makes it easy to perform complex queries.
- **Pull-Based Model**: Prometheus uses a pull model to scrape metrics from instrumented applications and services at regular intervals. This approach contrasts with a push model, where metrics are sent to the monitoring system by the application.
- **PromQL (Prometheus Query Language)**: Prometheus includes a powerful query language, PromQL, that allows you to slice and dice time-series data to generate custom metrics, dashboards, and alerts.
- **Alerting**: Prometheus can evaluate rules over your metrics data and trigger alerts based on those rules. Alerts are sent to an Alertmanager, which handles deduplication, grouping, and routing to different notification channels like email, Slack, or PagerDuty.
- **Service Discovery**: Prometheus can automatically discover targets to scrape metrics from, including Kubernetes services, Consul, and other dynamic environments.

Use Cases:

- **Monitoring System Metrics**: Collect metrics from servers, databases, and other infrastructure components.
- **Application Performance Monitoring (APM)**: Monitor application performance metrics like request rates, error rates, and latency.
- **Kubernetes Monitoring**: Monitor the health and performance of Kubernetes clusters, including nodes, pods, and services.

Integration with Other Tools:

- **Grafana**: Prometheus metrics can be visualized in Grafana, which provides powerful dashboard capabilities.
- **Thanos/Cortex**: For long-term storage of Prometheus metrics and to make Prometheus highly available and scalable across multiple data centers.

6. Grafana

Overview: Grafana is an open-source analytics and monitoring platform designed for creating and sharing dashboards. It supports a wide range of data sources, including Prometheus, Elasticsearch, InfluxDB, and many others. Grafana is known for its flexibility in visualizing data and creating interactive dashboards.

Key Features:

- **Multi-Source Support**: Grafana can connect to multiple data sources simultaneously, allowing you to visualize metrics, logs, and traces from different systems in a single dashboard.
- **Custom Dashboards**: Grafana allows users to create highly customizable and interactive dashboards with various visualizations like graphs, heatmaps, tables, and more. Dashboards can be shared across teams and organizations.
- **Alerting**: Grafana supports alerting based on thresholds set on visualized data. Alerts can be sent via various notification channels, including email, Slack, or webhook integrations.
- **Templating**: Grafana supports dynamic dashboards using template variables, making it easier to create reusable and adaptable dashboards.
- **Annotations**: Grafana allows you to add annotations directly to your graphs, marking specific events or incidents, which helps in correlating metrics with occurrences.
- **Plugins**: Grafana has an extensive plugin system, enabling integration with different data sources and the addition of new panels, data transforms, and other features.

Use Cases:

- **Unified Monitoring**: Create dashboards that unify data from multiple sources like Prometheus for metrics, Loki for logs, and Jaeger for traces.
- **Custom Dashboards for Operations Teams**: Provide operations teams with real-time insights into system health and performance.
- **Alerting and Incident Response**: Set up alerting based on specific thresholds and conditions visualized on dashboards.

Integration with Other Tools:

- **Prometheus**: Grafana is often used with Prometheus to visualize metrics.
- **Loki**: For visualizing logs alongside metrics in Grafana dashboards.
- **ElasticSearch**: For visualizing search-based data or logs stored in Elasticsearch.

Application Failure Troubleshooting

ImagePullBackoff

When a kubelet starts creating containers for a Pod using a container runtime, it might be possible the container is in Waiting state because of ImagePullBackOff.

The status ImagePullBackOff means that a container could not start because Kubernetes could not pull a container image for reasons such as

- Invalid image name or
- Pulling from a private registry without imagePullSecret.

The BackOff part indicates that Kubernetes will keep trying to pull the image, with an increasing back-off delay.

Kubernetes raises the delay between each attempt until it reaches a compiled-in limit, which is 300 seconds (5 minutes).

CrashLoopBackOff

When you see "CrashLoopBackOff," it means that kubelet is trying to run the container, but it keeps failing and crashing. After crashing, Kubernetes tries to restart the container automatically, but if the container keeps failing repeatedly, you end up in a loop of crashes and restarts, thus the term "CrashLoopBackOff."

This situation indicates that something is wrong with the application or the configuration that needs to be fixed.

Common Situations of CrashLoopBackOff

The CrashLoopBackOff error in Kubernetes indicates that a container is repeatedly crashing and restarting. Here are explanations of how the CrashLoopBackOff error can occur due to the specific reasons you listed:

Misconfigurations

Misconfigurations can encompass a wide range of issues, from incorrect environment variables to improper setup of service ports or volumes. These misconfigurations can prevent the application from starting correctly, leading to crashes. For example, if an application expects a certain environment variable to connect to a database and that variable is not set or is incorrect, the application might crash as it cannot establish a database connection.

Errors in the Liveness Probes

Liveness probes in Kubernetes are used to check the health of a container. If a liveness probe is incorrectly configured, it might falsely report that the container is unhealthy, causing Kubernetes to kill and restart the container repeatedly. For example, if the liveness probe checks a URL or port that the application does not expose or checks too soon before the application is ready, the container will be repeatedly terminated and restarted.

The Memory Limits Are Too Low

If the memory limits set for a container are too low, the application might exceed this limit, especially under load, leading to the container being killed by Kubernetes. This can happen repeatedly if the

workload does not decrease, causing a cycle of crashing and restarting. Kubernetes uses these limits to ensure that containers do not consume all available resources on a node, which can affect other containers.

Wrong Command Line Arguments

Containers might be configured to start with specific command-line arguments. If these arguments are wrong or lead to the application exiting (for example, passing an invalid option to a command), the container will exit immediately. Kubernetes will then attempt to restart it, leading to the CrashLoopBackOff status. An example would be passing a configuration file path that does not exist or is inaccessible.

Bugs & Exceptions

Bugs in the application code, such as unhandled exceptions or segmentation faults, can cause the application to crash. For instance, if the application tries to access a null pointer or fails to catch and handle an exception correctly, it might terminate unexpectedly. Kubernetes, detecting the crash, will restart the container, but if the bug is triggered each time the application runs, this leads to a repetitive crash loop.

Pods not schedulable

In Kubernetes, the scheduler is responsible for assigning pods to nodes in the cluster based on various criteria. Sometimes, you might encounter situations where pods are not being scheduled as expected. This can happen due to factors such as node constraints, pod requirements, or cluster configurations.

1. Node Selector
2. Node Affinity
3. Taints
4. Tolerations

Imperative Methods and Advance kubectl commands

1. Commands for dry run and to create a yaml file

kubectl run nginx --image=nginx --dry-run=client -o yaml > pod.yaml

kubectl create deployment my- deployment --image=nginx --dry-run=client -o yaml > deployment.yaml

2. Commands for Creating Resources

Pod:

kubectl run nginx --image=nginx --dry-run=client -o yaml > testpod.yaml

Deployment

kubectl create deployment my- deployment --image=nginx --dry-run=client -o yaml > deployment.yaml

ReplicaSet:

(Note: In imperative methods the –image parameter is not supported for replicaset. we are creating a Deployment and then we are updating the deployment file to convert it to replicaset file)

Service:

kubectl expose deployment nginx-deployment --port=80 --target-port=80 --type=ClusterIP

ConfigMap:

kubectl create configmap my-config --from-literal=app.name=MyApp

Secret:

kubectl create secret generic my-secret --from-literal=password=mypassword

3. Commands for Updating Resource

Scale a Deployment:

kubectl scale deployment nginx-deployment --replicas=3

Rolling Update a Deployment:

kubectl rollout restart deployment/nginx-deployment

Label a Resource:

kubectl label <resource-type> <resource-name> <key>=<value>

4. Commands for Managing Configurations

Edit a resource

kubectl edit <resource-type> <resource-name>

5. Commands for viewing History

View Deployment History:

kubectl rollout history deployment/nginx-deployment

6. Commands for Taints and Tolerations

Taint a Node:

kubectl taint nodes <node-name> <key>=<value>:<effect>

kubectl taint nodes node1 key=value:NoSchedule

Remove a Taint from a Node:

kubectl taint nodes <node-name> <key>:<effect>-

kubectl taint nodes node1 key=value:NoSchedule-

7. Commands for Node and Cluster Operations

Drain a Node (Prepare for Maintenance):

kubectl drain <node-name>

kubectl drain node1 --ignore-daemonsets –force

Cordon a Node (Mark as Unschedulable):

kubectl cordon <node-name>

kubectl cordon <node-name>

Uncordon a Node (Mark as Schedulable):

kubectl uncordon <node-name>

kubectl uncordon node1

8. Commands for Port Forwarding and Proxy

Port Forwarding

kubectl port-forward <pod-name> <local-port>:<pod-port>

kubectl port-forward nginx-pod 8080:80

Proxy to the Kubernetes API Server:

kubectl proxy --port=<port>

kubectl proxy --port=8001

9. Executing Commands in a Pod

Run a Command in a Pod:

kubectl exec <pod-name> -- <command>

kubectl exec nginx-pod -- ls /usr/share/nginx/html

10. Commands for Managing Cluster Resources

View API Resources:

kubectl api-resources

View API Version:

kubectl api-versions

11. Miscellaneous Commands

Apply a Label to a Node:

kubectl label nodes <node-name> <key>=<value>

kubectl label nodes node1 disktype=ssd

Rollout Status:

kubectl rollout status deployment/<deployment-name>

kubectl rollout status deployment/nginx-deployment

12. Backup and Restore

Export Resources to YAML

kubectl get all --export -o yaml > backup.yaml

Restore from a YAML Backup

kubectl apply -f backup.yaml

13. Command to check the username

Displays the user name associated with the current context in the kubeconfig file.

kubectl auth whoami

14. Command to check the user access control

Checks if an action is allowed based on the current user's access control.

kubectl auth can-i <verb> <resource>

kubectl auth can-i get pods

15. Command to count the Number of running pods or any workload within the cluster

kubectl get all --no-headers | wc -l

These imperative commands provide a quick and direct way to manage your Kubernetes cluster. For more complex setups, you can always generate YAML manifests using these commands and then further customize them.

JSONPath Expressions

Kubectl supports JSONPath template

JSONPath template is composed of JSONPath expressions enclosed by curly braces {}. Kubectl uses JSONPath expressions to filter on specific fields in the JSON object and format the output. In addition to the original JSONPath template syntax, the following functions and syntax are valid:

1. Use double quotes to quote text inside JSONPath expressions.
2. Use the range, end operators to iterate lists.
3. Use negative slice indices to step backwards through a list. Negative indices do not "wrap around" a list and are valid as long as -index + listLength >= 0.

JSONPath is a query language used to navigate and extract specific data from JSON-formatted data structures. In Kubernetes, JSONPath expressions are particularly useful when working with Kubernetes resources via the command-line interface (kubectl). By using JSONPath, you can filter and format the output of kubectl commands to extract only the information you need.

Basic Syntax of JSONPath

A JSONPath expression consists of the following elements:

1. **Root ($):** Represents the root of the JSON document. It's similar to the . in XPath.
2. **Dot Notation (.):** Used to access child elements. For example, .metadata accesses the metadata field.
3. **Bracket Notation ([ ]):** Used to access elements by index or key. For example, ['items'][0] accesses the first item in an array.
4. **Wildcard (*):** Matches all elements at the current level. For example, .items[*] matches all elements in the items array.
5. **Filters ([?()]):** Allows for conditional expressions. For example, ['items'][?(@.status.phase=="Running")] filters items where the status.phase is "Running".
6. **Current Context (@):** Refers to the current element being processed. It is similar to this in programming languages.

Examples using kubectl and JSONPath expressions

kubectl get pods -o json

kubectl get pods -o=jsonpath='{@}'

kubectl get pods -o=jsonpath='{.items[0]}'

kubectl get pods -o=jsonpath='{.items[0].metadata.name}'

kubectl get pods -o=jsonpath="{.items[*]['metadata.name', 'status.capacity']}"

kubectl get pods -o=jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.status.startTime}{"\n"}{end}'

kubectl get pods -o=jsonpath='{.items[0].metadata.labels.kubernetes\.io/hostname}'

Using JSONPath in Kubernetes

When working with kubectl, you can use the -o jsonpath=<expression> option to format the output. Below are some practical examples:

1. Extracting the Names of All Pods in a Namespace

kubectl get pods -o jsonpath="{.items[*].metadata.name}"

- **Explanation:**
    - {}: Encloses the entire JSONPath expression.
    - .items[*].metadata.name: Accesses the name field of the metadata of each item in the items array (which represents all the pods).

2. Extracting the Status of a Specific Pod

kubectl get pod <pod-name> -o jsonpath="{.status.phase}"

- **Explanation:**
    - .status.phase: Accesses the phase field in the status object, which represents the current phase (e.g., Running, Pending) of the pod.

3. Filtering Pods by Label

kubectl get pods -l app=myapp -o jsonpath="{.items[*].metadata.name}"

- **Explanation:**
    - -l app=myapp: Filters the pods by the label app=myapp.
    - .items[*].metadata.name: Extracts the names of the filtered pods.

4. Extracting Container Images Used in a Pod

kubectl get pod <pod-name> -o jsonpath="{.spec.containers[*].image}"

- **Explanation:**
    - .spec.containers[*].image: Accesses the image field for each container in the containers array.

5. Filtering Pods by Status

kubectl get pods -o jsonpath="{.items[?(@.status.phase=='Running')].metadata.name}"

- **Explanation:**
    - [?(@.status.phase=='Running')]: Filters the pods where the status.phase is "Running".
    - .metadata.name: Extracts the names of the filtered pods.

Advanced JSONPath Features

1. **Accessing Nested Arrays:**
    - To access nested arrays or objects, you can chain multiple dot or bracket notations. For example:

kubectl get pod <pod-name> -o jsonpath="{.spec.containers[0].resources.limits.cpu}"

    - This expression accesses the CPU limit of the first container in the pod.

2. **Using the Length Operator:**
    - You can use the length() function to get the number of elements in an array:

kubectl get pods -o jsonpath="{.items.length()}"

    - This returns the number of pods in the namespace.

3. **Combining Multiple Fields:**
    - You can extract and format multiple fields at once:

kubectl get pod <pod-name> -o jsonpath="{.metadata.name}: {.status.phase}"

    - This returns the pod name followed by its status phase.

JSONPath Operators and Functions

- **@:** Current node.
- **.:** Child operator.
- **[]:** Subscript operator (can contain index, slice, or filter expressions).
- ***:** Wildcard operator.
- **..:** Recursive descent (searches for children at any depth).
- **length():** Returns the length of an array or string.