🚀 Mastering Kubernetes 🚀- 2024

Suresh Yadav

thesureshyadav76@gmail.com LinkedIn Profile

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
```

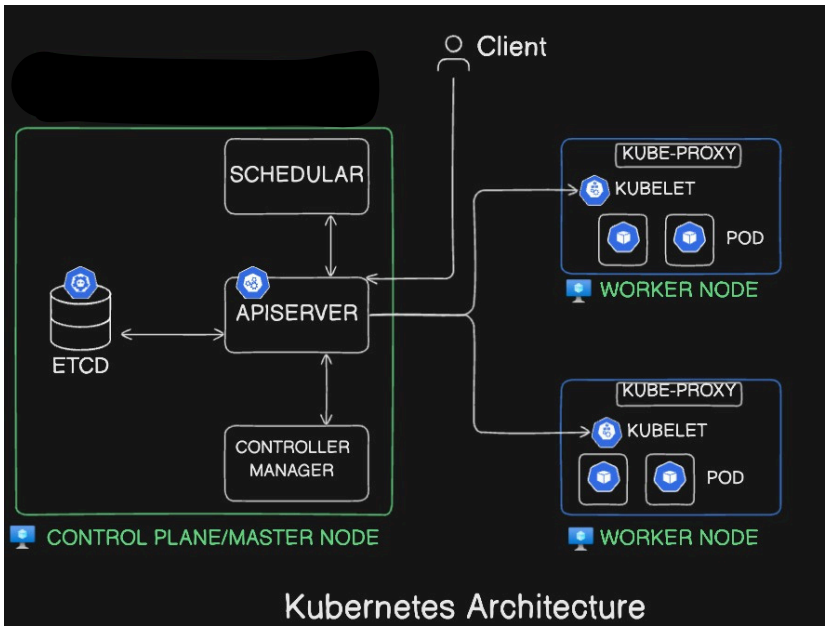
kind create cluster --image kindest/node:v1.29.4@sha256:3abb816a5b1061fb15c6e9e60856ec40d56 b7b52bcea5f5f1350bc6e2320b6f8 --name cka-multinodecluster1 --config mn1.yml

```
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

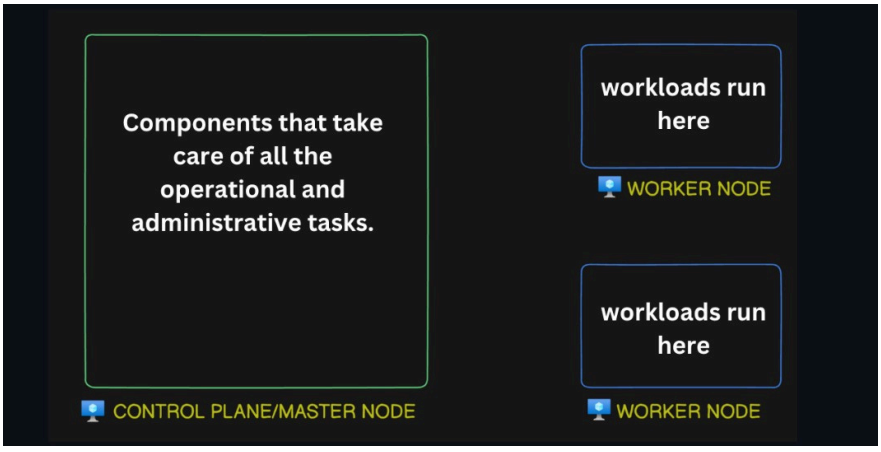


Master Node V/s Worker Node

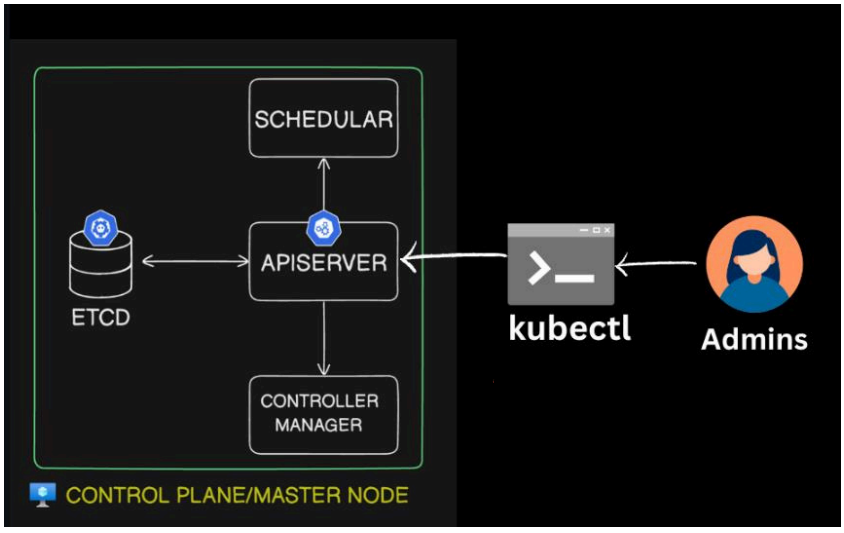
Role of Master Node:

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.

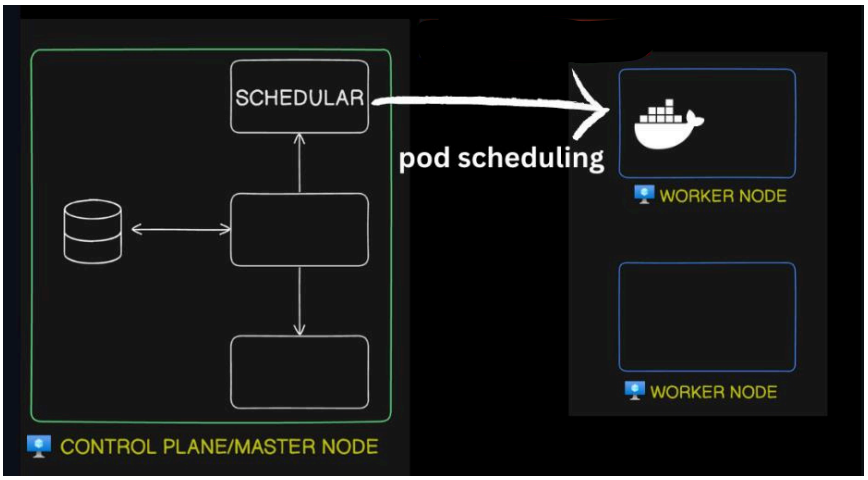


Components of Master Node 1. API Server The Kubernetes API server is a component of the Kubernetes control plane that exposes the
Kubernetes API, which is used by all other components of Kubernetes (such as kubectl, the CLI tool) to interact with the cluster. It acts as the front end for the Kubernetes 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.



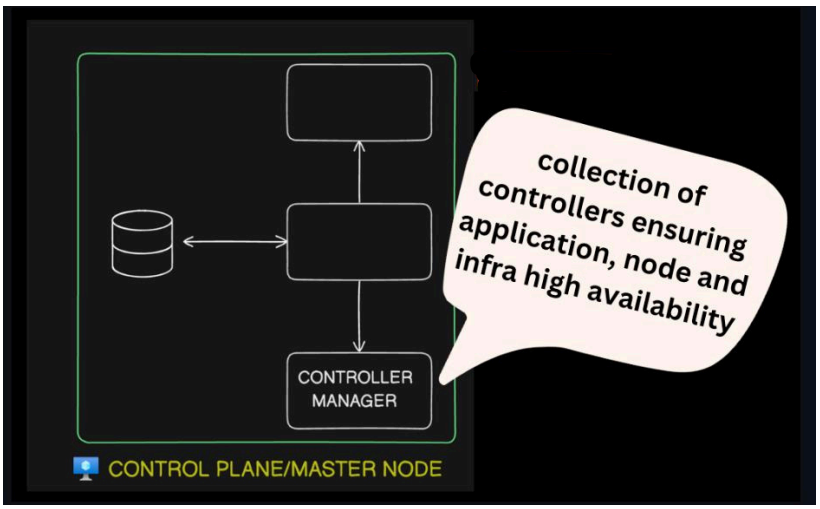
2. Scheduler

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.



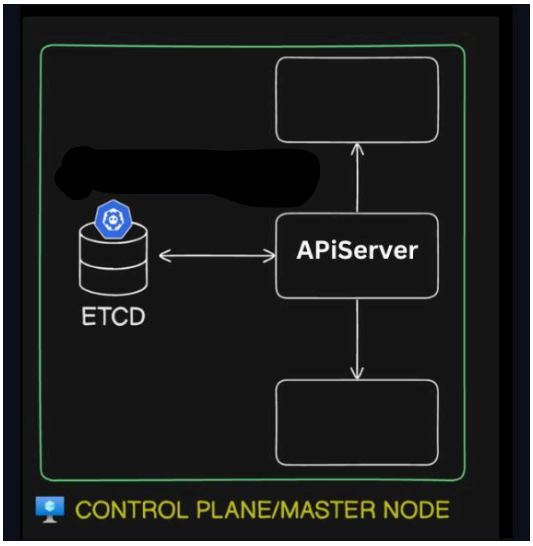
3. Controller Manager The Controller Manager in Kubernetes is a component of the control plane that manages different
types of controllers to regulate the state of the cluster and perform cluster-wide tasks. Each controller in the Controller Manager is responsible for managing a specific aspect of the 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

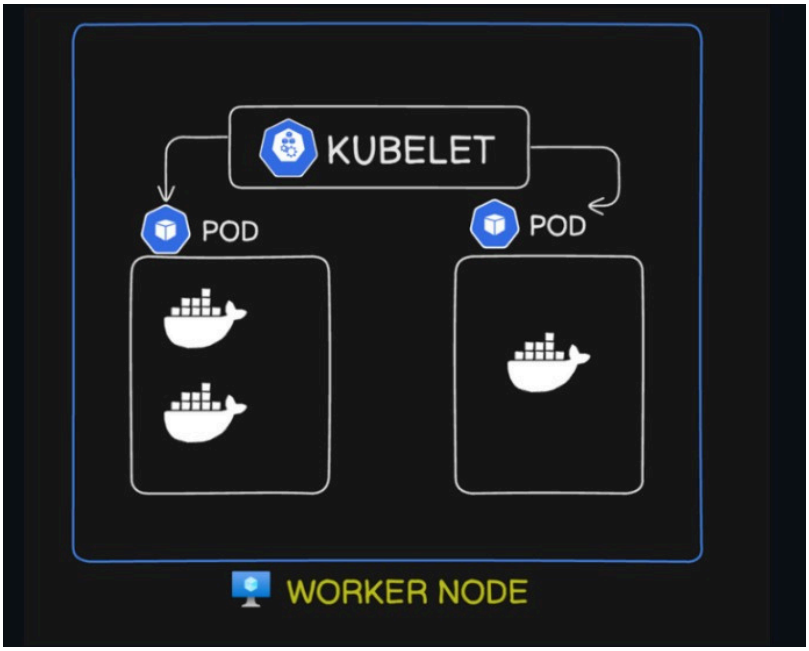
1. **Kubelet**

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.

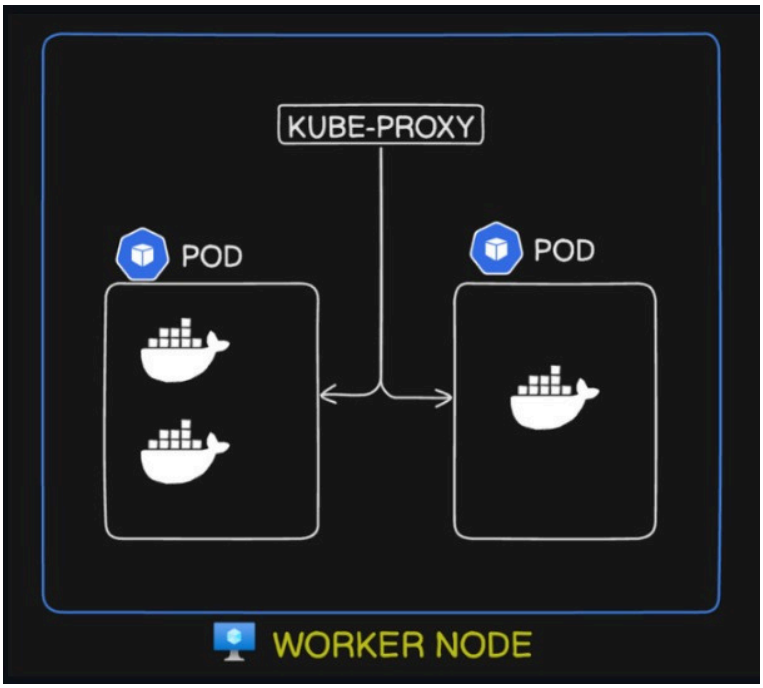
There are several modes in which kube-proxy can operate, including:

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

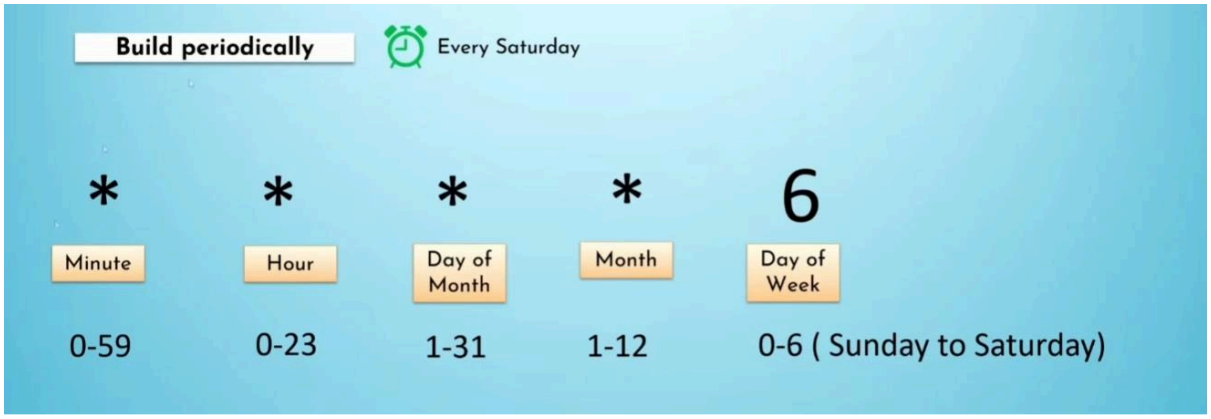
A Job creates one or more Pods and will continue to retry execution of the Pods until a specified number of them successfully terminate. As pods successfully complete, the Job tracks the successful completions. When a specified number of successful completions is reached, the task (ie, Job) is complete. 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.

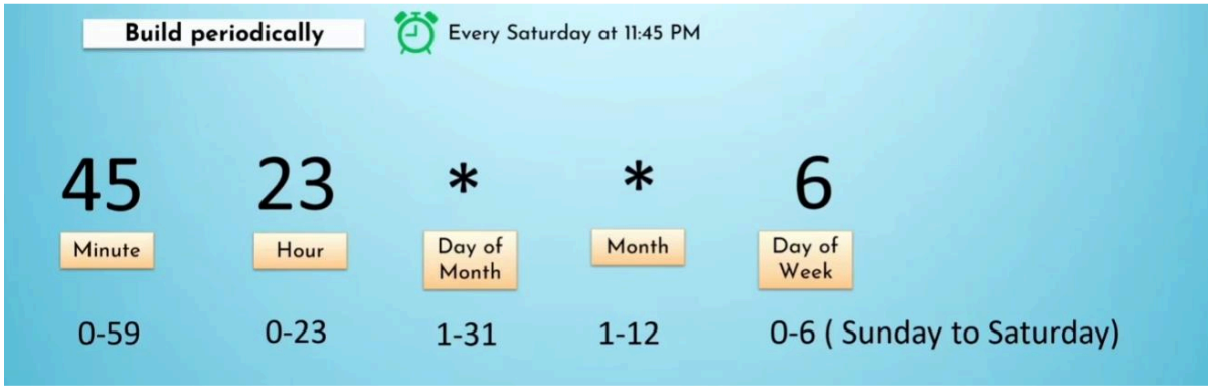
9. Cronjobs

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

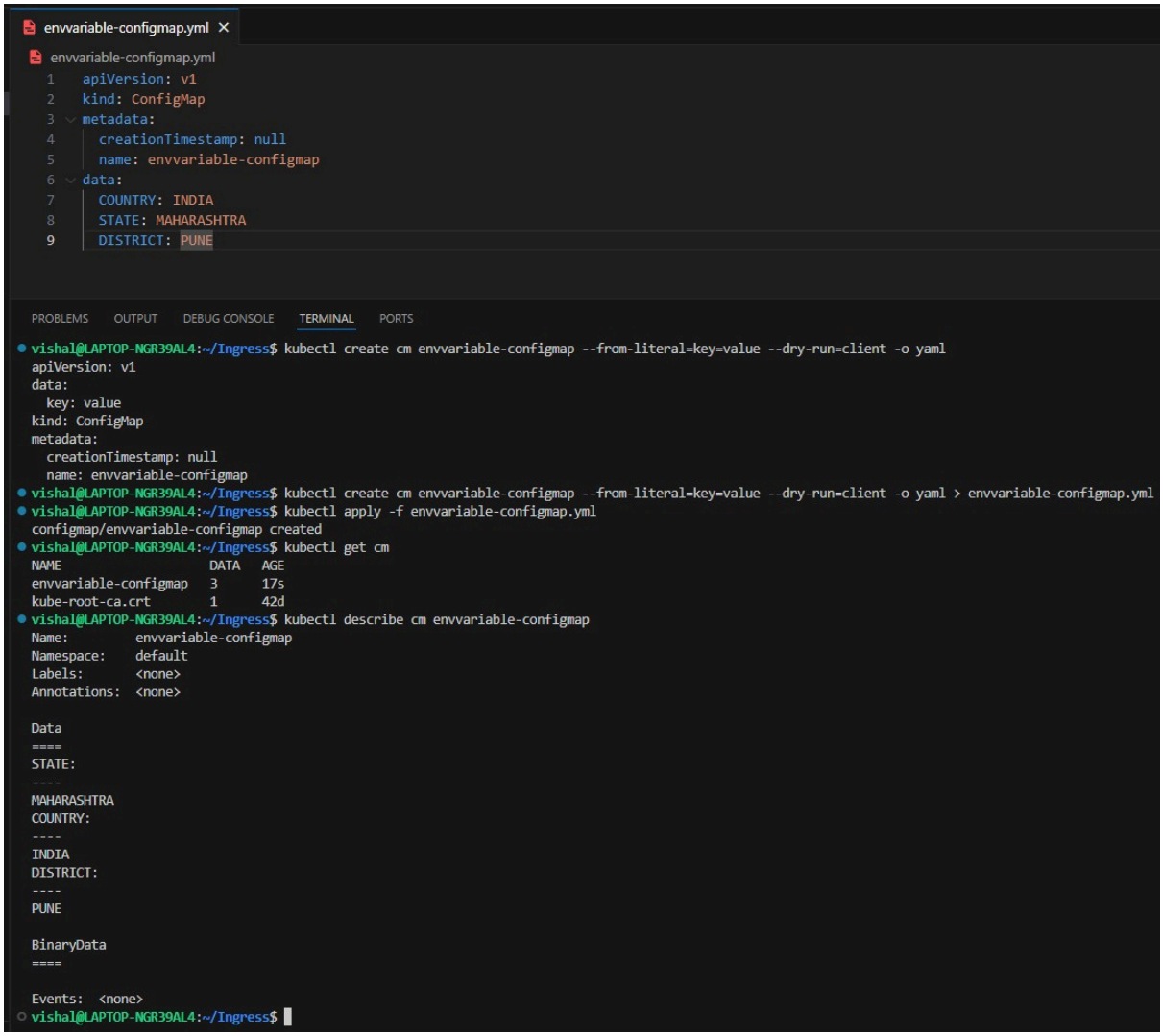
1. ConfigMap

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



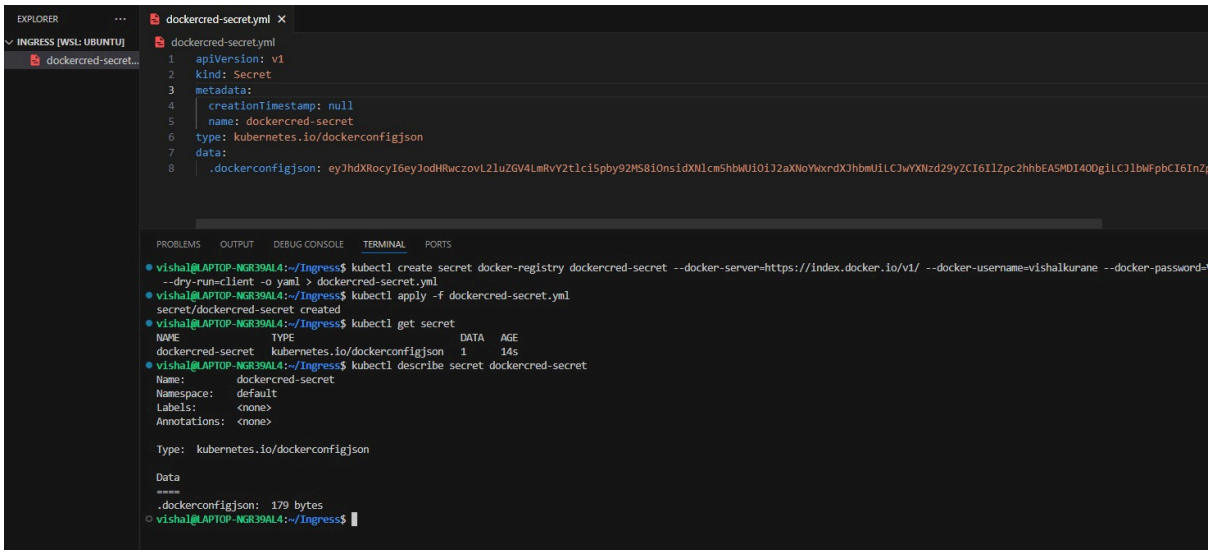
2. Secrets

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>

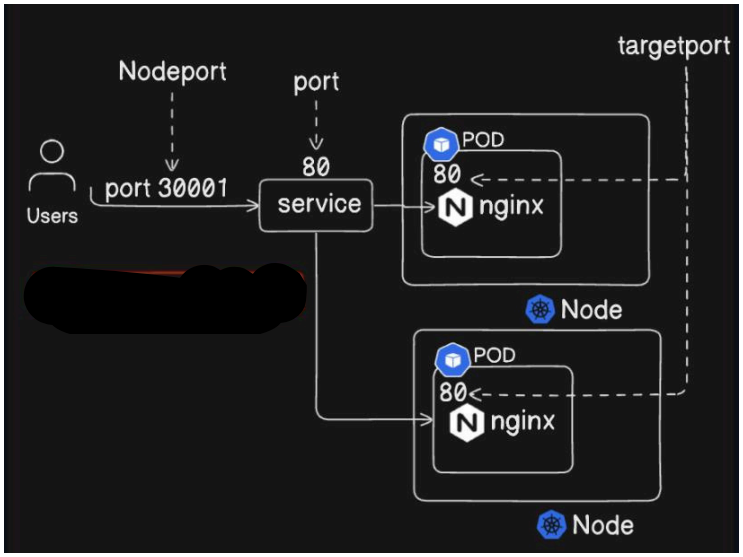


Services

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:

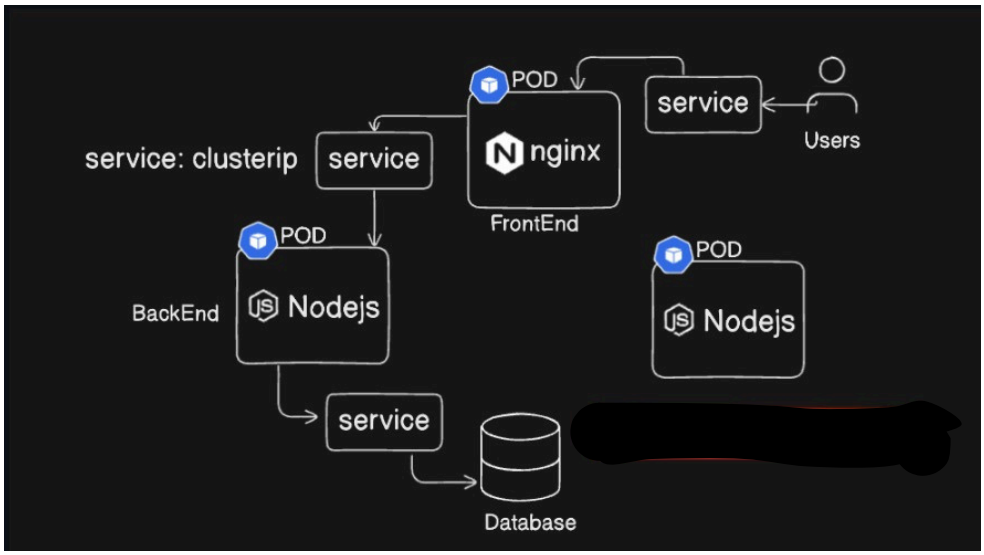
- **NodePort** exposes the service on a specific port on each node in the cluster.
- This allows external traffic to access the service by sending requests to the <NodeIP>:<NodePort>.
- The port range for NodePort services is typically between 30000-32767.

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:

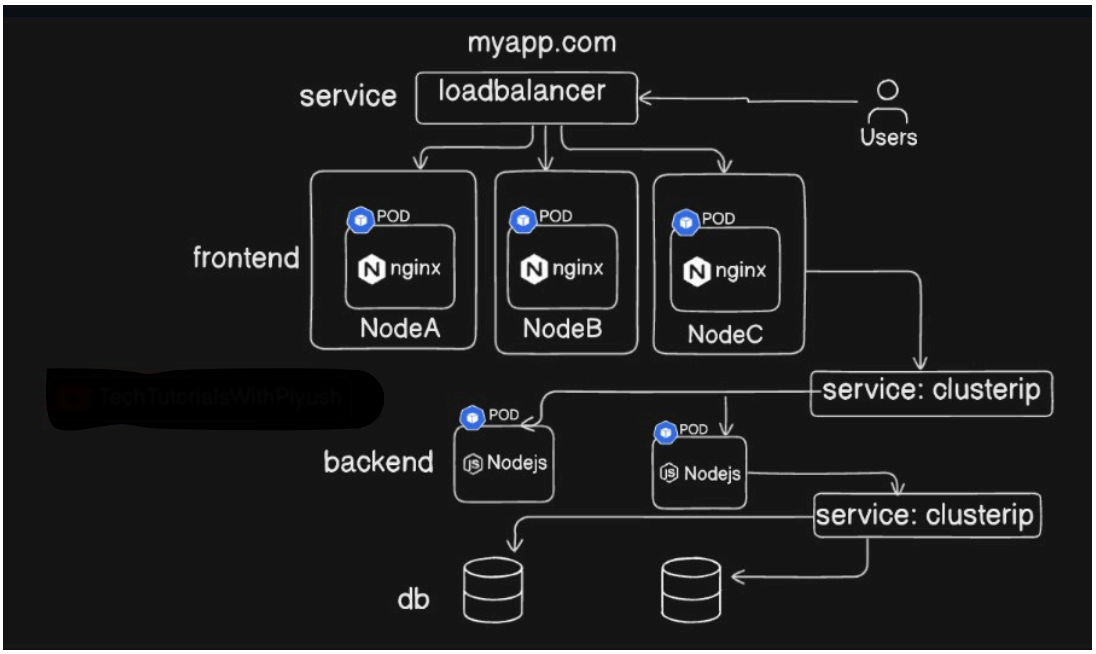
- **ClusterIP** is the default service type in Kubernetes.
- It makes the service accessible only within the Kubernetes cluster, using an internal IP address.
- Other services within the cluster can use this IP to communicate with the service.

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:

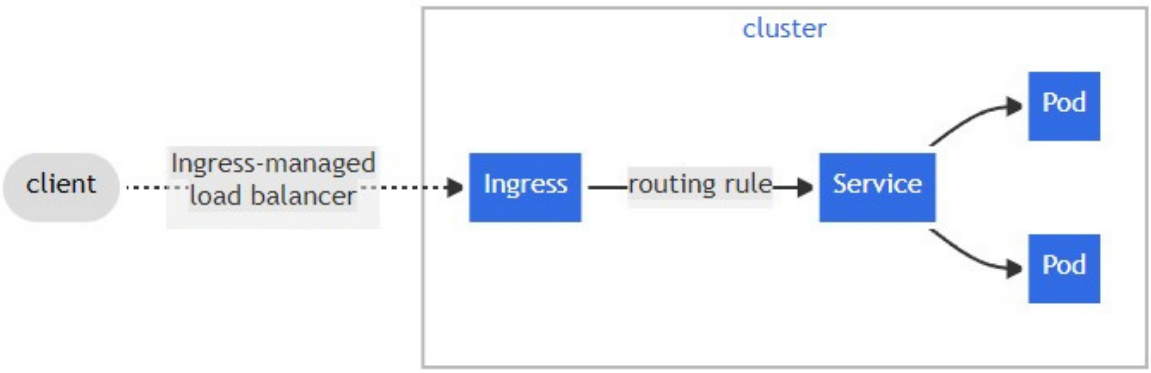
- **External Service Aliasing**: If you want to refer to an external service using a consistent name within your Kubernetes cluster, you can use ExternalName. For example, if your application needs to connect to a third-party API like api.example.com, you can create a Kubernetes service with an ExternalName 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.

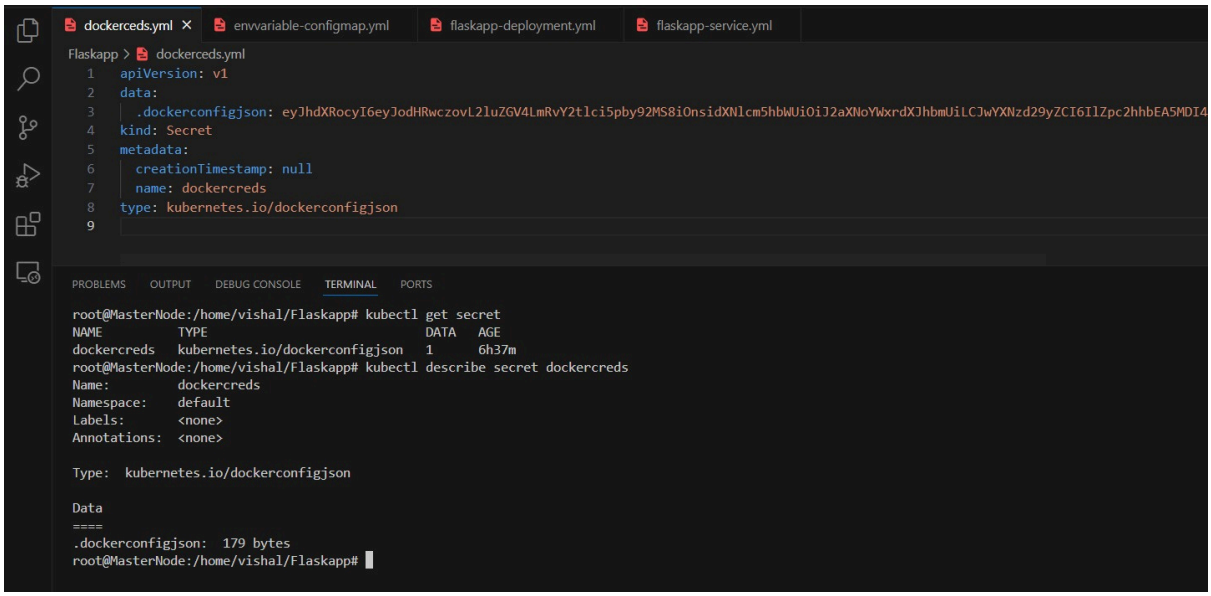
Benefits of Using Ingress:

- **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.

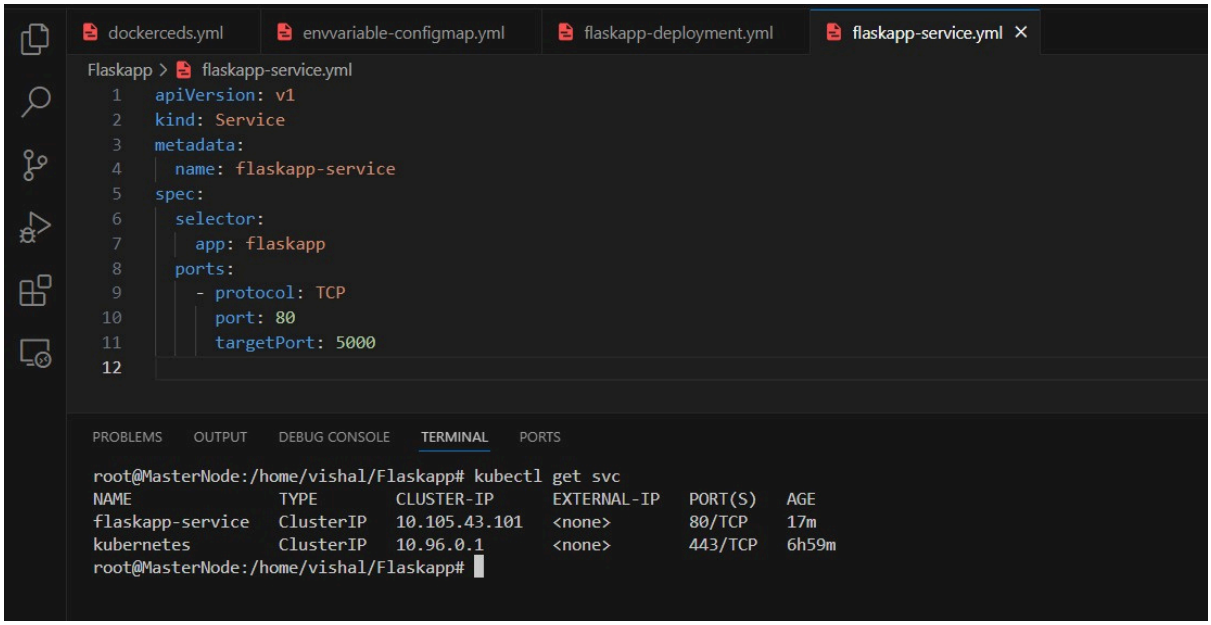
Project: Deploy a Flask application with and without Ingress

1. Create Image pull secret from private Docker registry

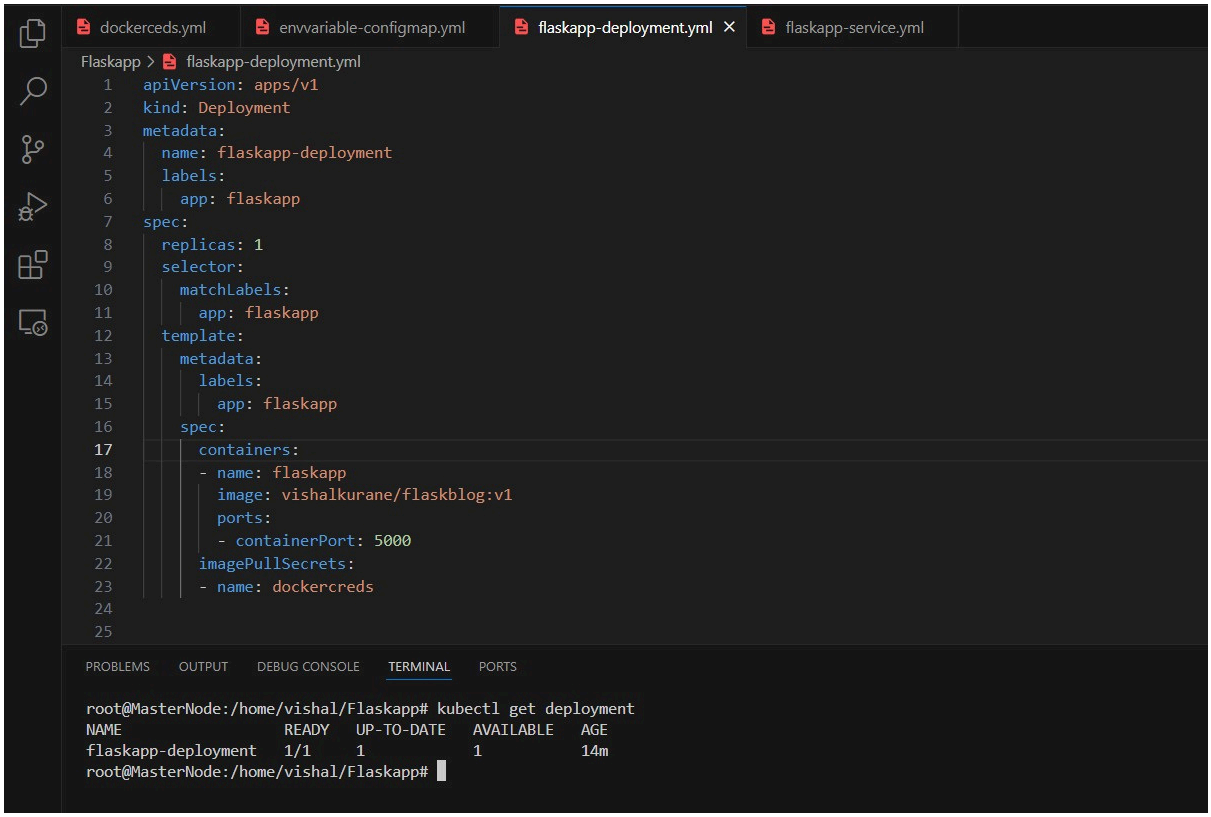
➔ kubectl create secret docker-registry dockercred-secret --docker-server=https://index.docker.io/v1/ --docker-username=vishalkurane --docker-password=<Password> --docker-email=<email>



2. Create a Service of type ClusterIP

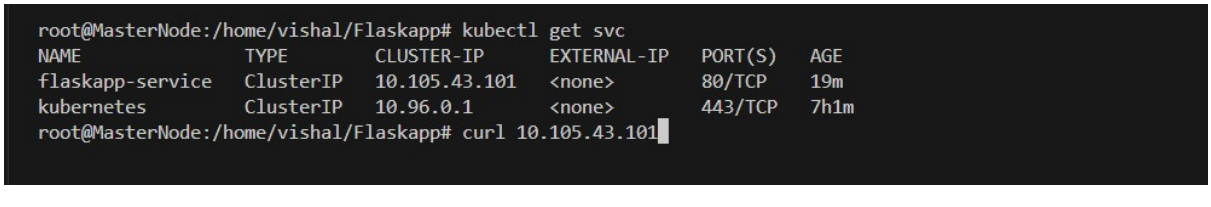


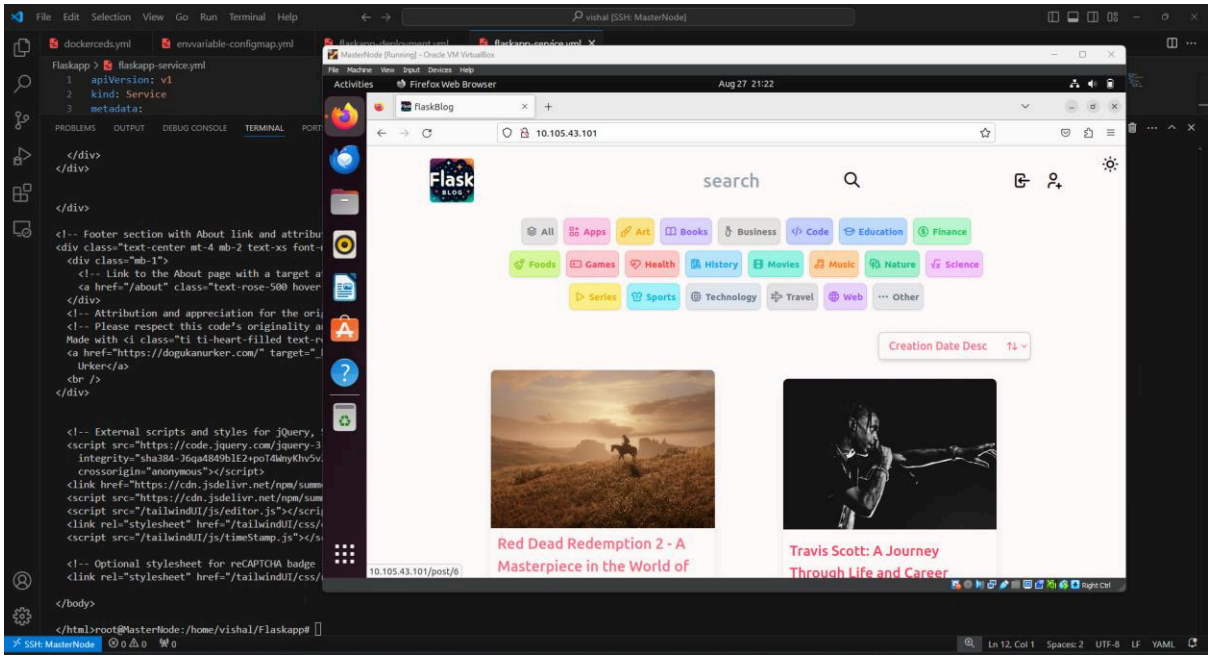
3. Create Deployment



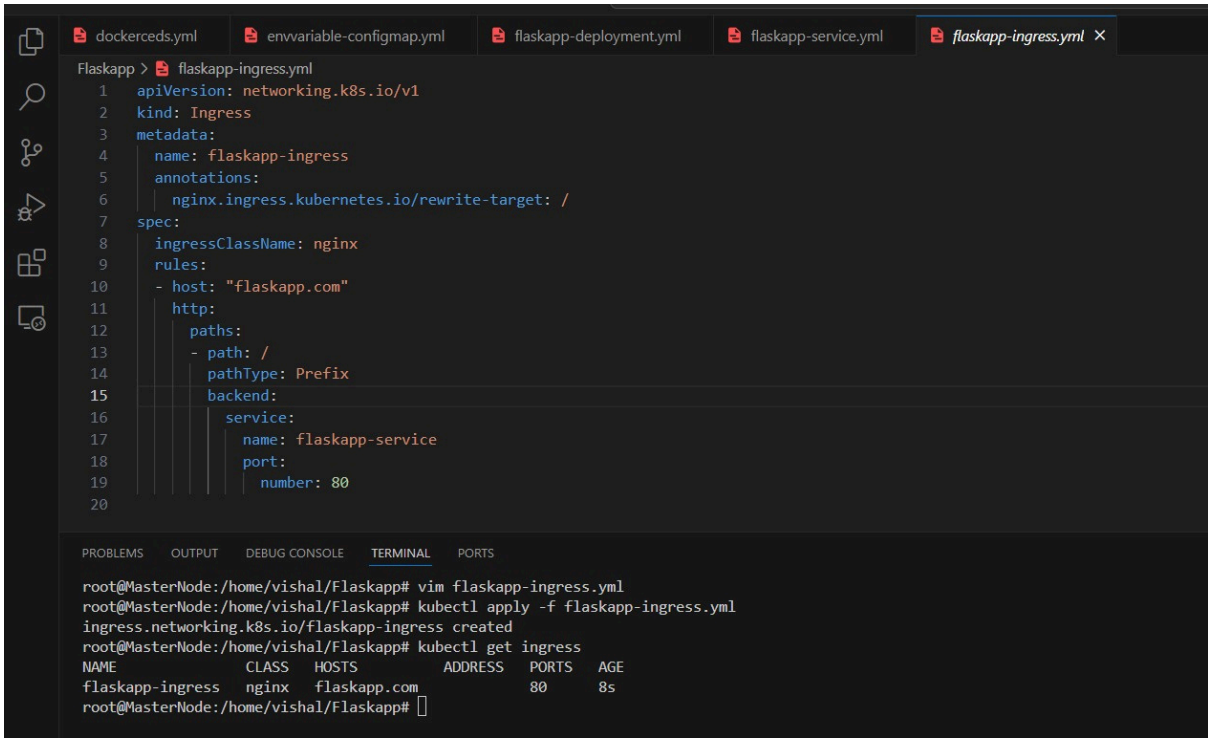
4. Check weather service is able to communicate with pod

➔ Curl <ClusterIP>



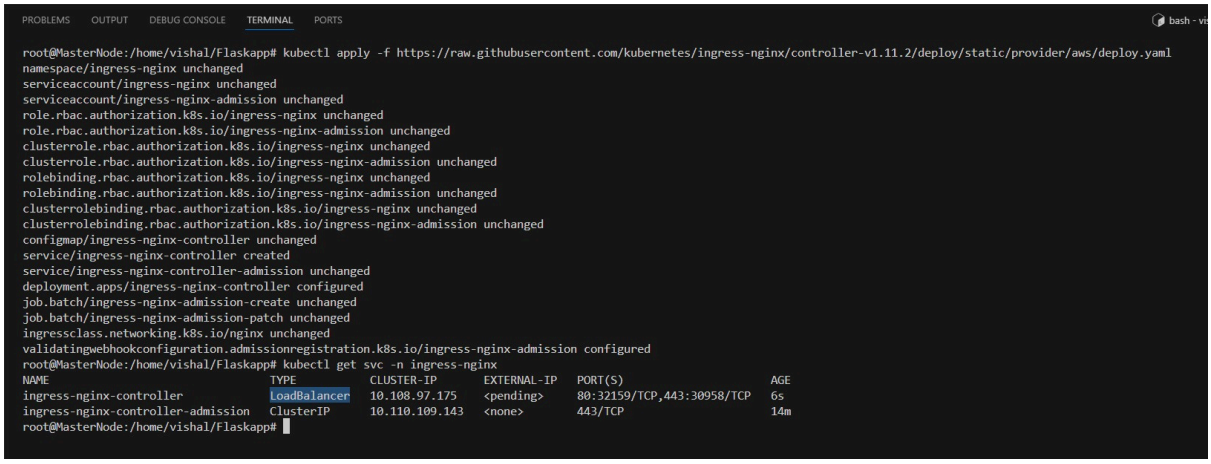


5. Create an Ingress resource



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

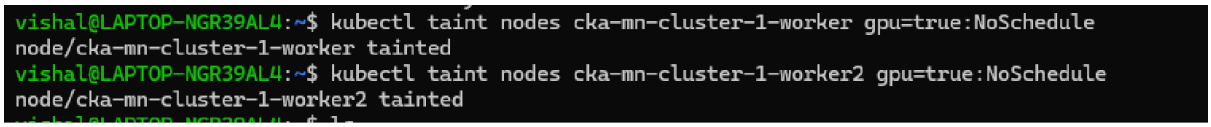
1. NoSchedule (Newer Pods)
2. PreferNoSchedue (No Guaranty)
3. NoExecution (Existing/Newer Pods)

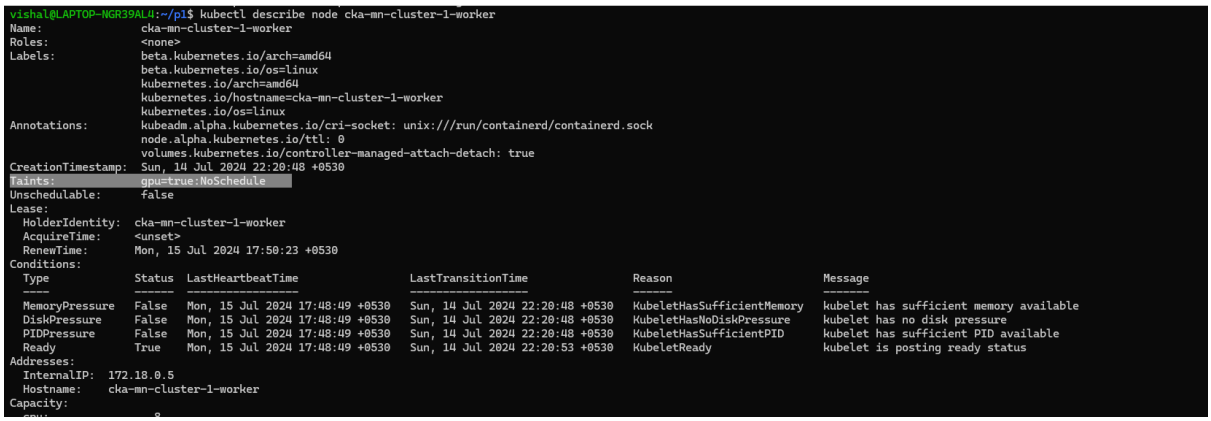
Taint a node using below command:

➔ 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:**

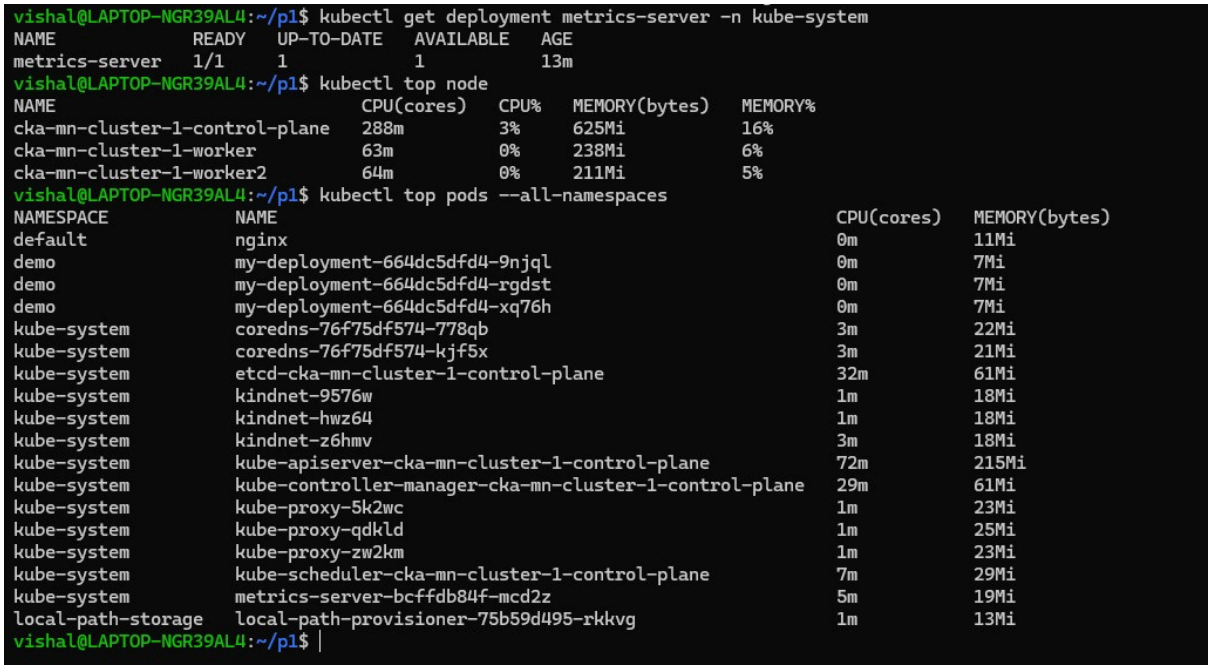
   kubectl edit deployment metrics-server -n kube-system
4. **Add the --kubelet-insecure-tls Flag:**

   - --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



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

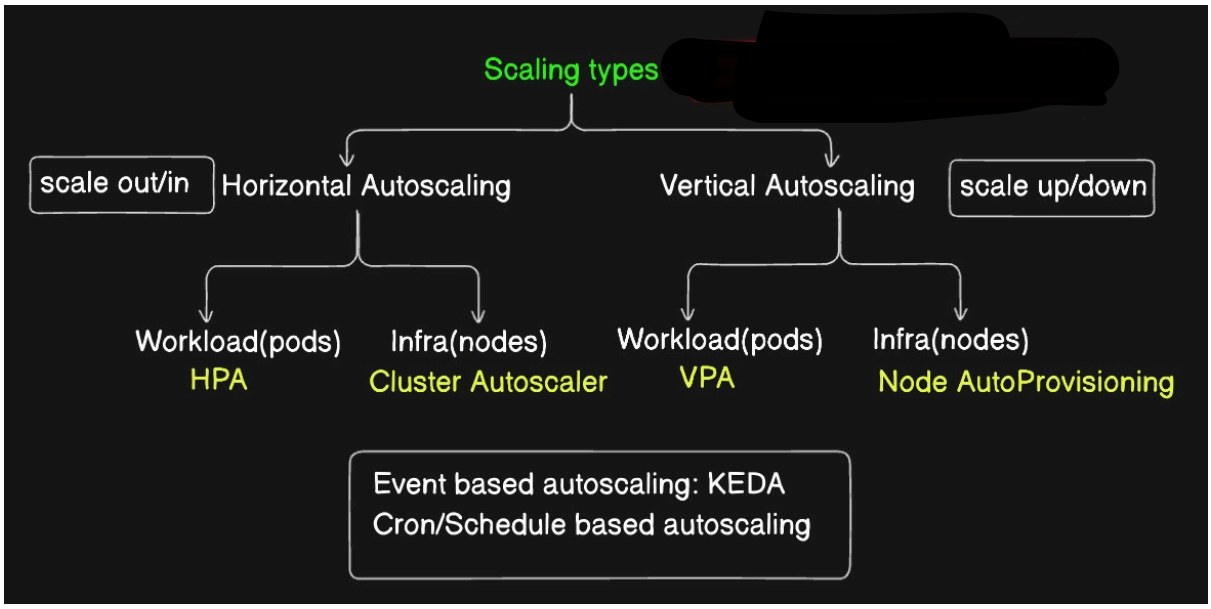
Resource	Unit	Description	Example Value
CPU	m	Millicores (1/1000 of a core)	500m (0.5 CPU core)
CPU	-	Whole CPU cores	2 (2 CPU cores)
Memory	Ki	Kibibytes (1024 bytes)	1024Ki (1 MiB)
Memory	Mi	Mebibytes (1024 KiB)	256Mi (256 MiB)
Memory	Gi	Gibibytes (1024 MiB)	1Gi (1 GiB)

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

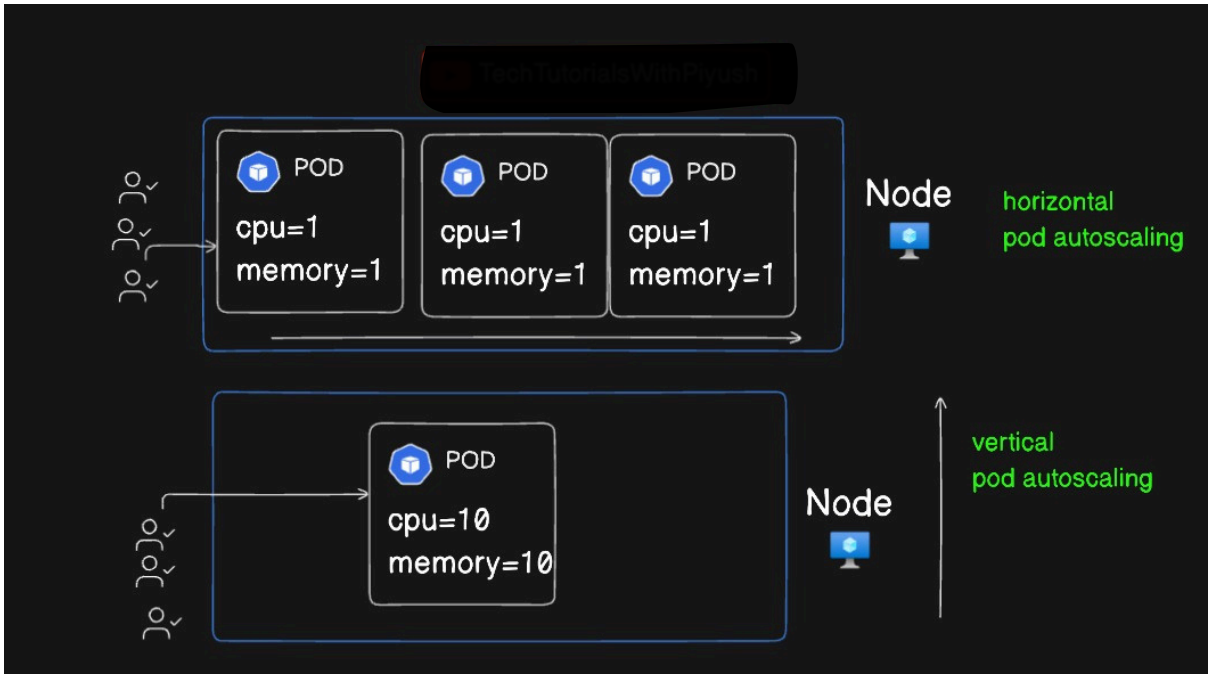
Autoscaling types



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.

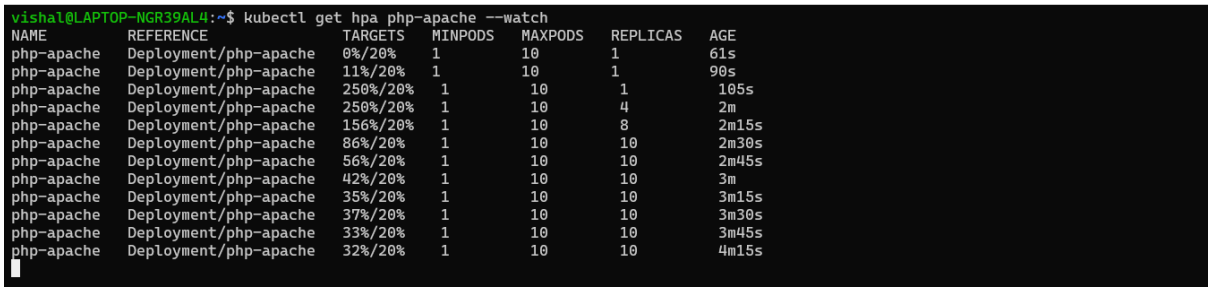
Vertical Pod Autoscaling (VPA) in Kubernetes adjusts the resource requests and limits (CPU and memory) for containers within a pod based on their usage.



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

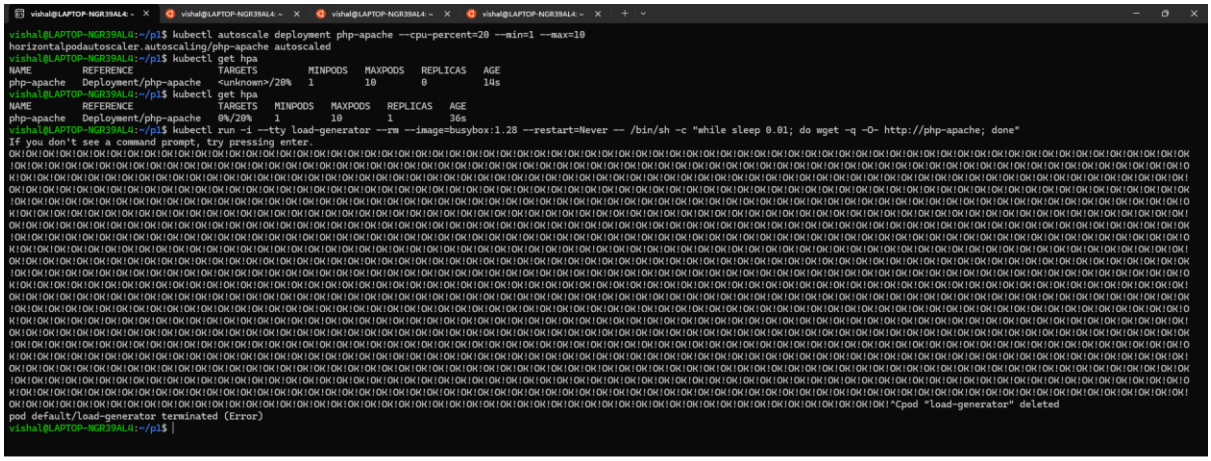
When Load is increasing



```
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:

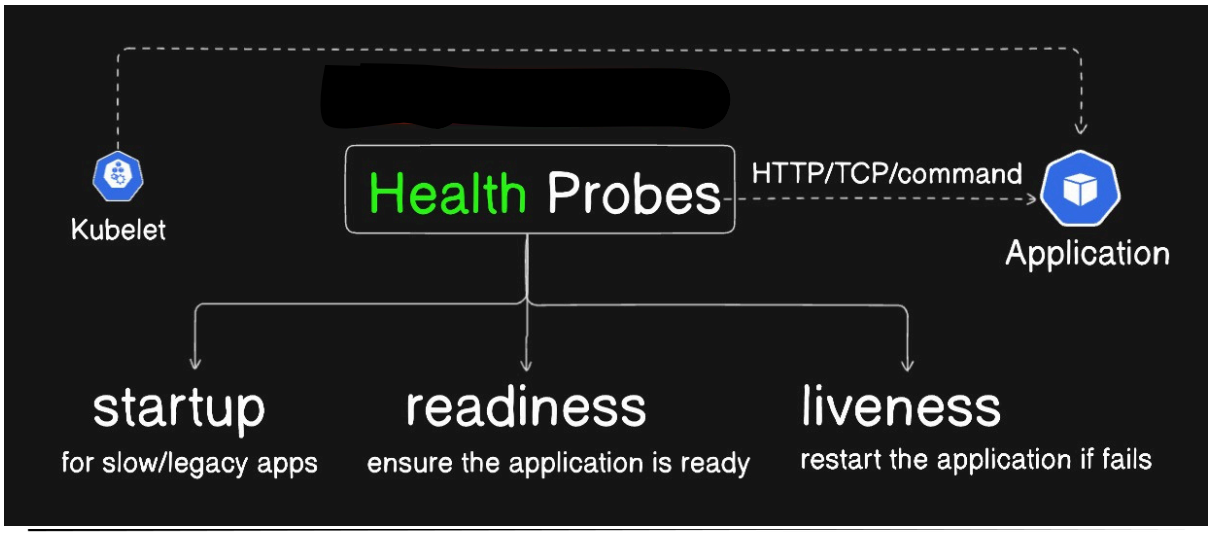
- The **Liveness Probe** is used to determine if a container is running. If the liveness probe fails, Kubernetes will kill the container, and it will be restarted according to the pod's restart policy.

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

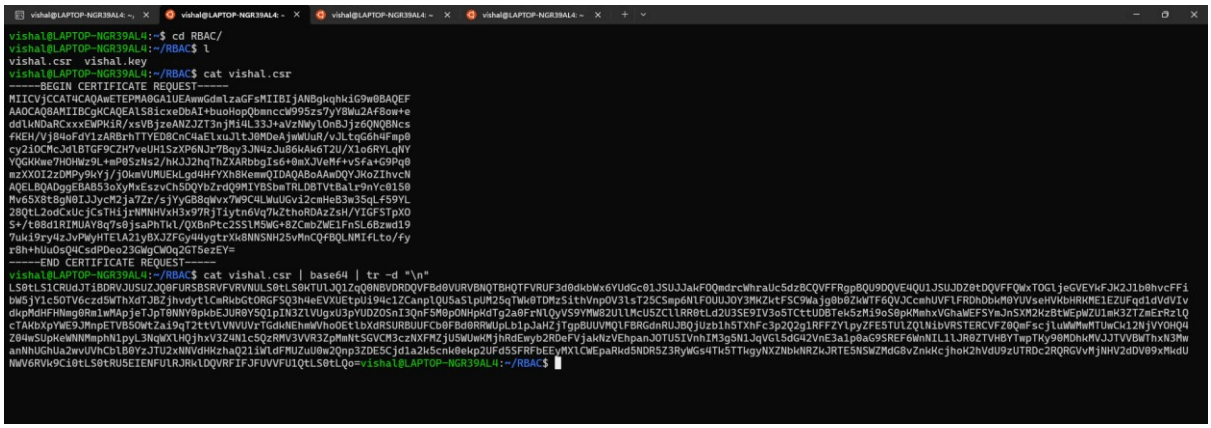
1. Generate a key for client:
   ➔ openssl genrsa -out vishal.key 2048

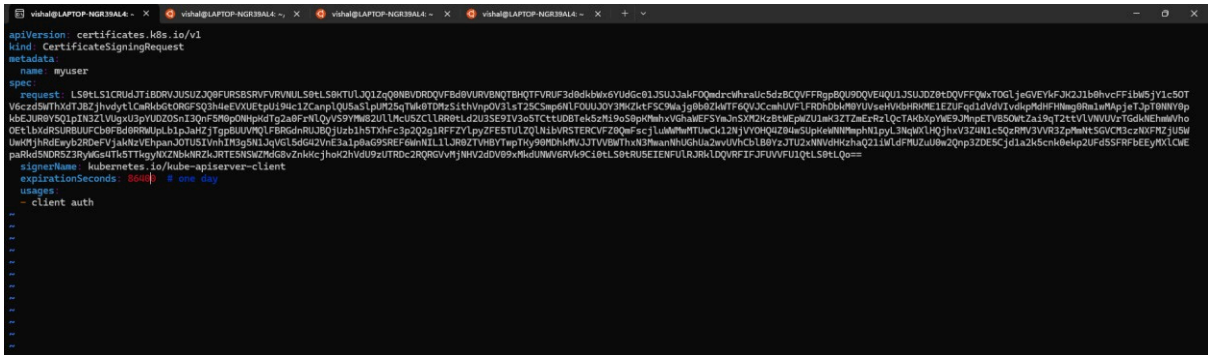
```
vishal@LAPTOP-NGR39AL4:~/RBAC$ openssl genrsa -out vishal.key 2048
vishal@LAPTOP-NGR39AL4:~/RBAC$ ls
vishal.key
vishal@LAPTOP-NGR39AL4:~/RBAC$ cat vishal.key
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCVLyJzF4NsAj5u
6geilBuadxxb33nOzvJjxa7YB/yjD5512WQ0NpELHHERY8qJH/GxUGPN4A1kllPe
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$
```

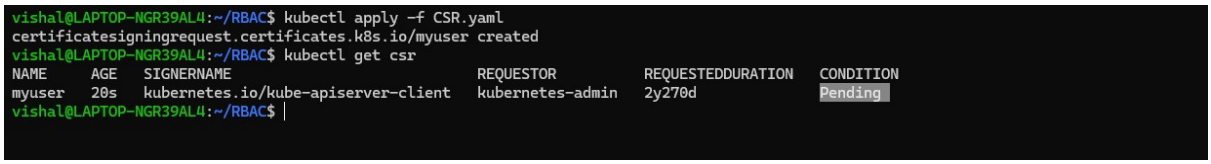
2. Generate a CSR
   ➔ openssl req -new -key vishal.key -out vishal.csr -subj "/CN=vishal"

```
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$
```

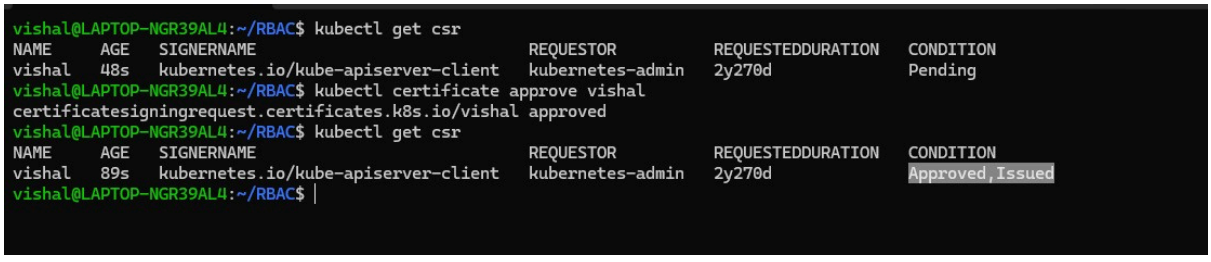
3. As a Kubernetes admin create a “CertificateSingningRequest” yaml manifest using the CSR created previously.







4. Approve the CSR



5. Get the certificate from Approved CSR

Now you have an approved certificate for user “Sures”

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-598b4774864-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$
```

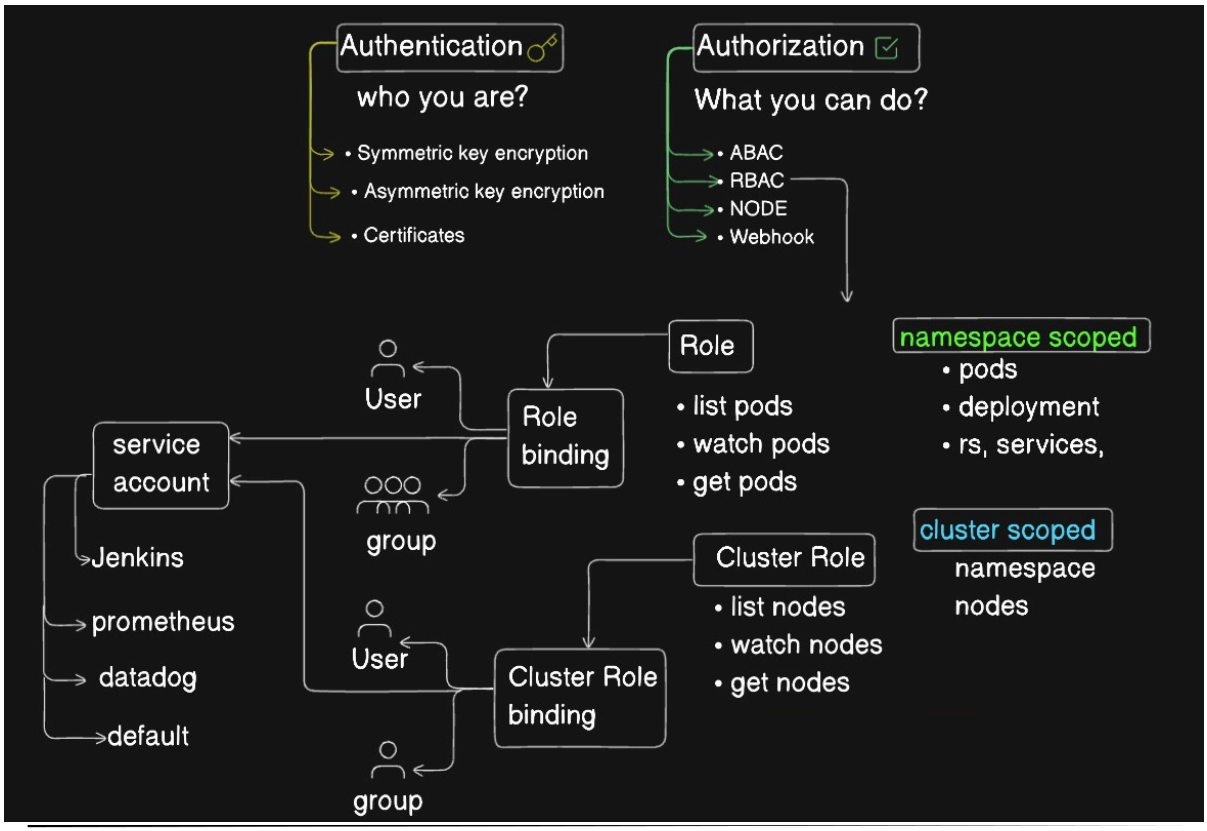
As a user “vishal” we are getting the access for node using ClusterRoleBinding

```
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$
```

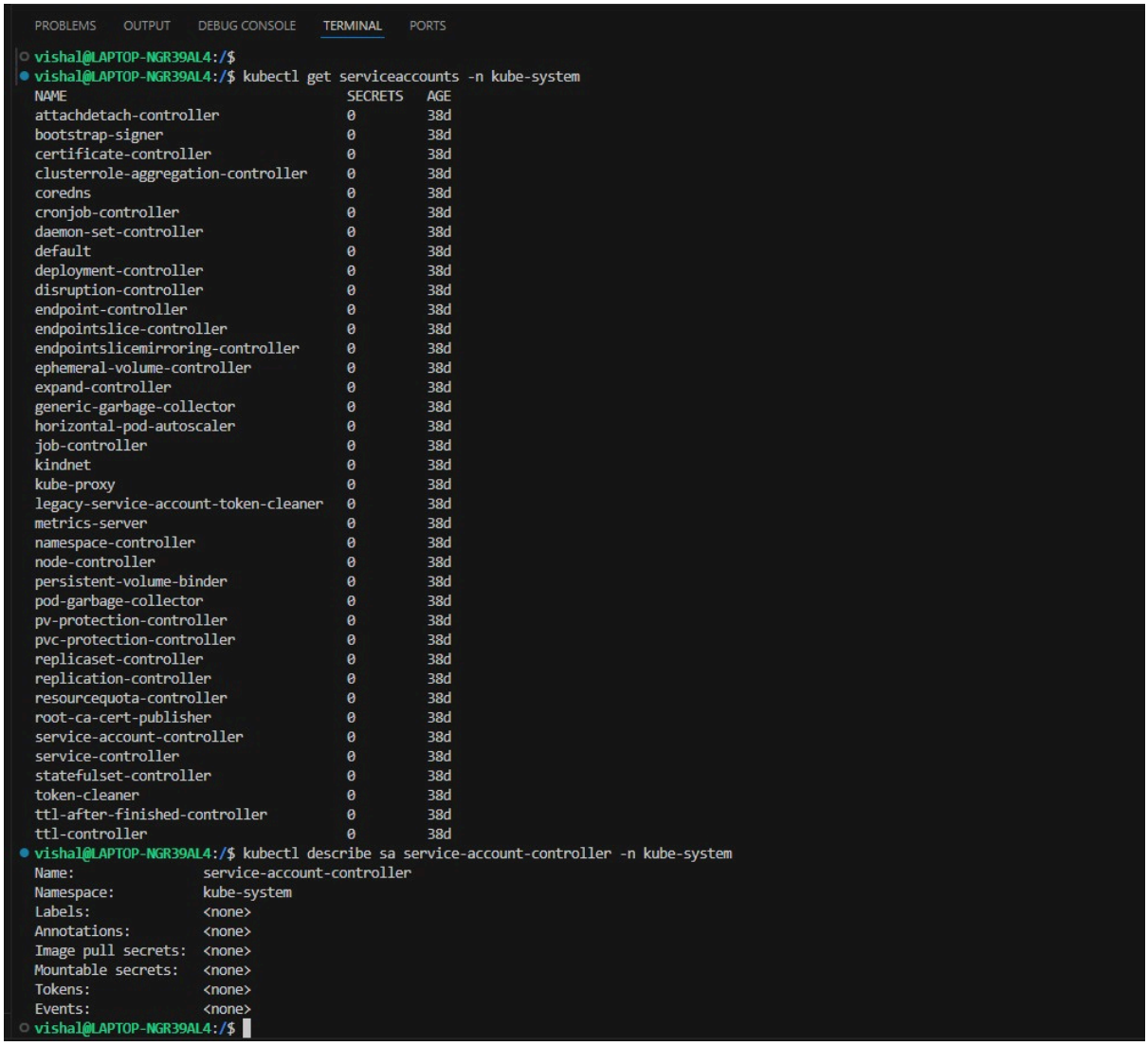
Service Account

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)

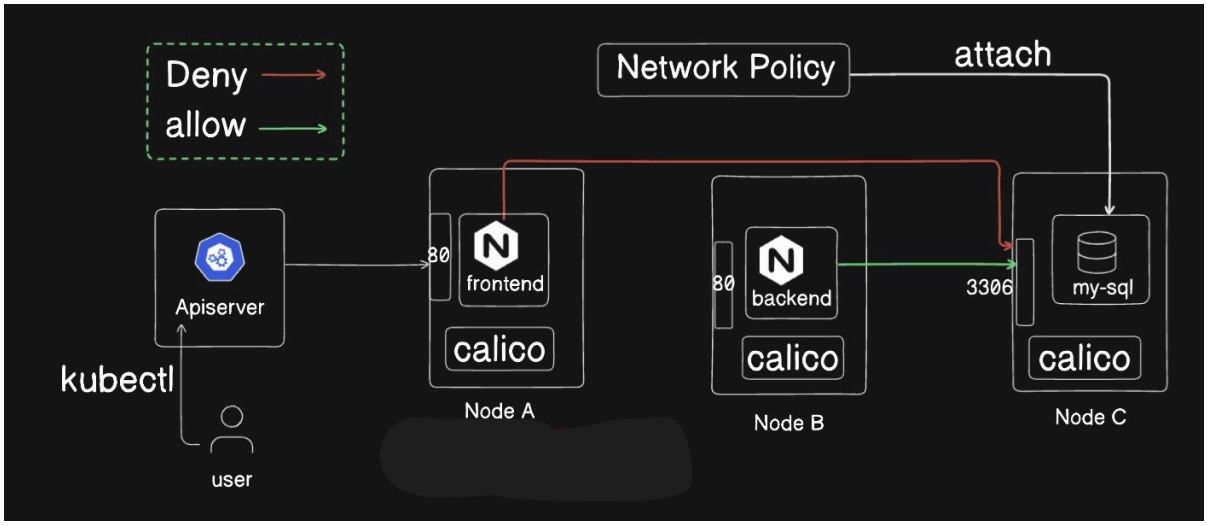
CNI stands for Container Network Interface. It's a standard for configuring network interfaces in Linux containers, used by container orchestrators like Kubernetes. CNI provides a framework for plugins to manage container networking, allowing different networking solutions to be easily integrated.

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.



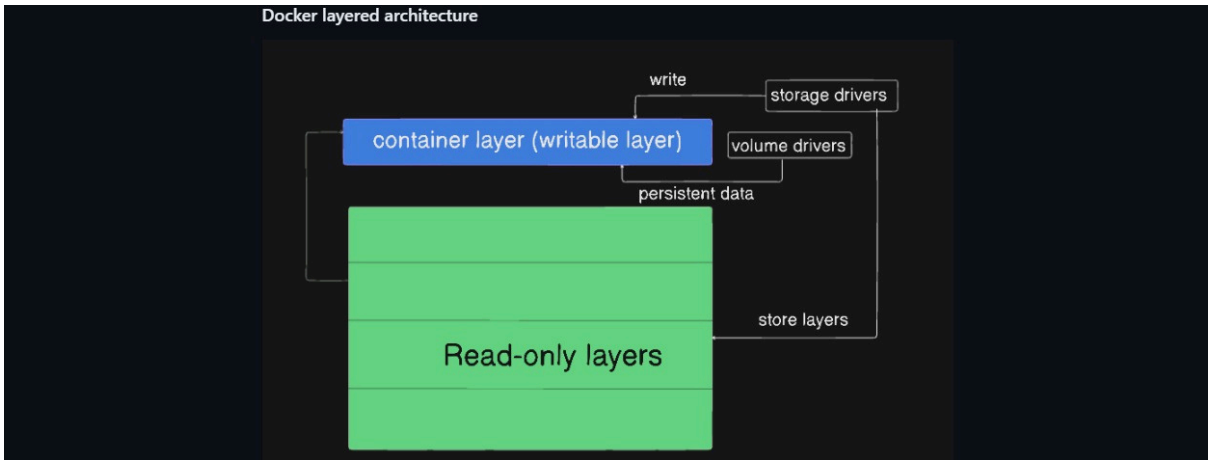
Document to install calico on your cluster

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.

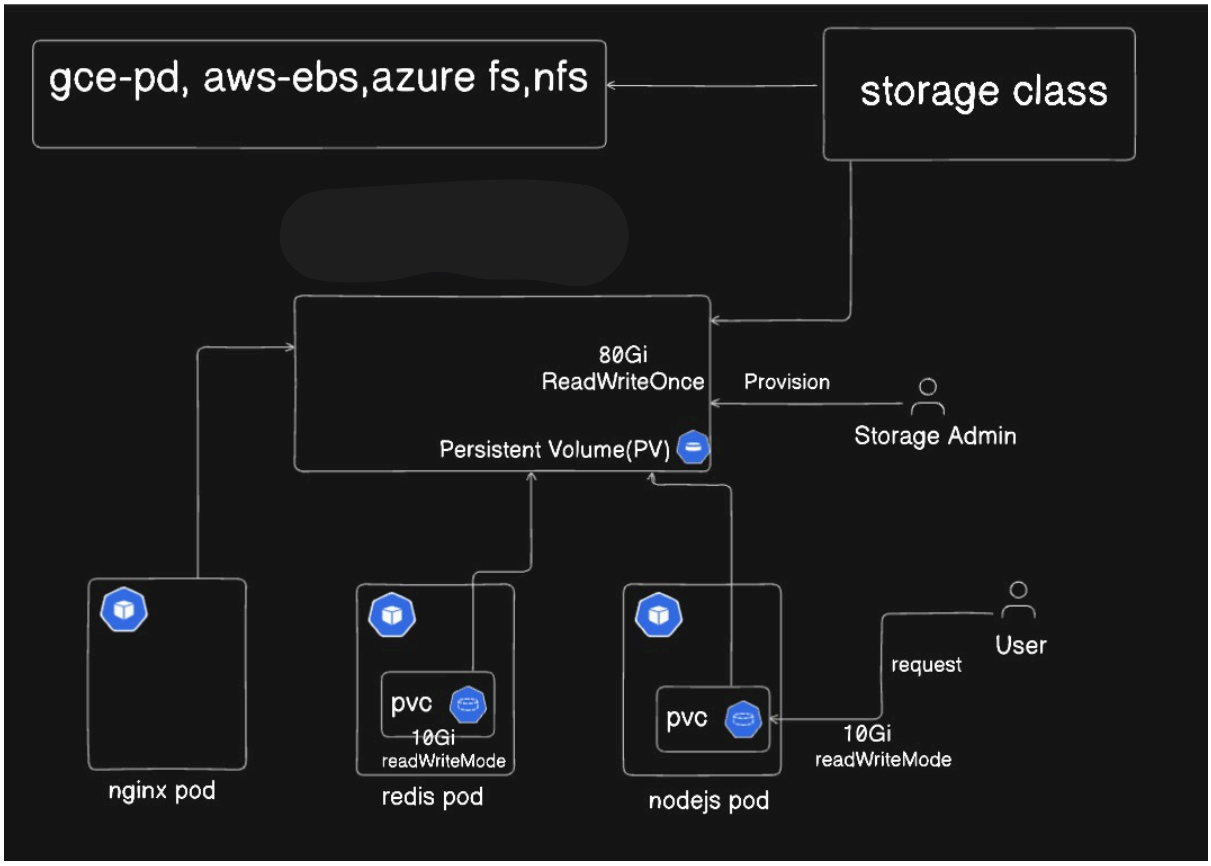
   A Docker bind mount is a type of volume that allows you to map a specific directory or file from the host filesystem directly into a container. This is useful when you need to provide the container with access to files or directories on the host, or when you want to share data between the host and the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