Kubernetes Short Notes

Note

- Kubernetes is an open-source container management tool that automates container deployment, scaling & load balancing.
- It schedules, runs, and manages isolated containers that are running on virtual/physical/cloud machines.
- All top cloud providers support Kubernetes.
- One popular name for Kubernetes is K8s.

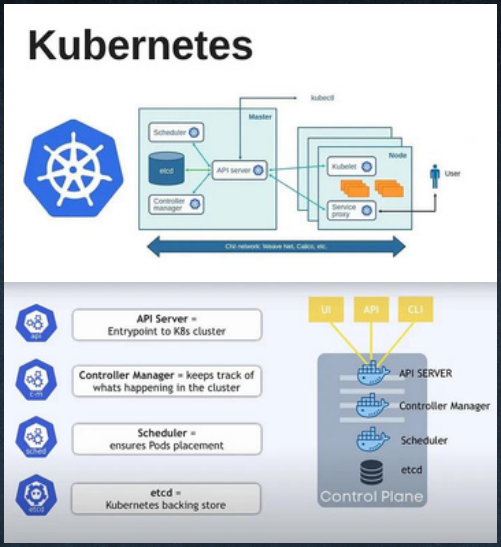
History

- Google developed an internal system called 'borg' (later named Omega) to deploy and manage thousands of google applications and services on their cluster.
- In 2014, google introduced Kubernetes an open-source platform written in 'Golang' and later donated to CNCF (Cloud Native Computing Foundation).

Features of Kubernetes

- Orchestration (clustering of any number of containers running on a different network).
- Autoscaling (Vertical & Horizontal)
- Auto Healing
- Load Balancing
- Platform Independent (Cloud/Virtual/Physical)
- Fault Tolerance (Node/POD/Failure)
- Rollback (Going back to the previous version)
- Health monitoring of containers
- Batch Execution (One time, Sequential, Parallel)

Architecture of Kubernetes



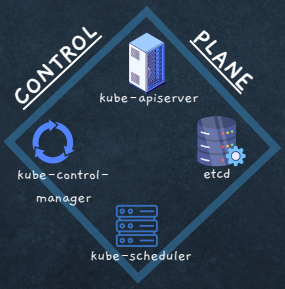
Working with kubernetes

- We create a Manifest (.yml) file
- Apply those to cluster (to master) to bring it into the desired state.
- POD runs on a node, which is controlled by the master.

Role of Master Node

- Kubernetes cluster contains containers running or Bare Metal / VM instances/cloud instances/ all mix.
- Kubernetes designates one or more of these as masters and all others as workers.
- The master is now going to run a set of K8s processes. These processes will ensure the smooth functioning of the cluster. These processes are called the 'Control Plane.
- Can be Multi-Master for high availability.
- Master runs control plane to run cluster smoothly.

Components of Control Plane



Kube-api-server → (For all communications)

- This api-server interacts directly with the user (i.e we apply .yml or .json manifest to kube-api-server)
- This kube-api-server is meant to scale automatically as per load.
- Kube-api-server is the front end of the control plane.

etcd

- Stores metadata and status of the cluster.
- etcd is a consistent and high-available store (key-value-store)
- Source of touch for cluster state (info about the state of the cluster)

→ etcd has the following features

1. Fully Replicated → The entire state is available on every node in the cluster.
2. Secure → Implements automatic TLS with optional client-certificate authentication.
3. Fast → Benchmarked at 10,000 writes per second.

Kube-schedular (action)

- When users request the creation & management of Pods, Kube-scheduler is going to take action on these requests.
- Handles POD creation and Management.
- Kube-scheduler match/assigns any node to create and run pods.
- A scheduler watches for newly created pods that have no node assigned. For every pod that the scheduler discovers, the scheduler becomes responsible for finding the best node for that pod to run.
- The scheduler gets the information for hardware configuration from configuration files and schedules the Pods on nodes accordingly.

Controller-Manager

- Make sure the actual state of the cluster matches the desired state.

→ Two possible choices for controller manager

1. If K8s is on the cloud, then it will be a cloud controller manager.
2. If K8s is on non-cloud, then it will be kube-controller-manager.

Components on the master that runs the controller

1. **Node Controller** → For checking the cloud provider to determine if a node has been detected in the cloud after it stops responding.
2. **Route-Controller** → Responsible for setting up a network, and routes on your cloud.
3. **Service-Controller** → Responsible for load Balancers on your cloud against services of type Load Balancer.
4. **Volume-Controller** → For creating, attaching, and mounting volumes and interacting with the cloud provider to orchestrate volume.

Nodes (Kubelet and Container Engine)

- Node is going to run 3 important pieces of software/process.

Kubelet

- The agent running on the node.
- Listens to Kubernetes master (eg- Pod creation request).
- Use port 10255.
- Send success/Fail reports to master.

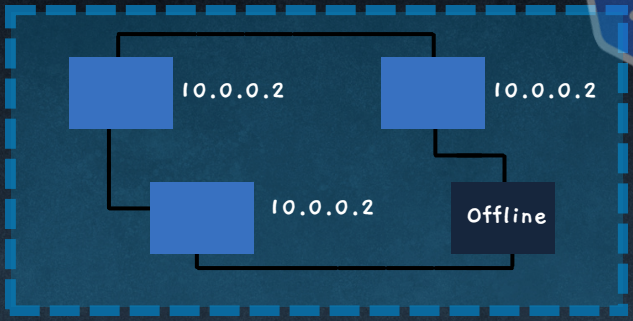
Container Engine

- Works with kubelet
- Pulling images
- Start/Stop Containers
- Exposing containers on ports specified in the manifest.

Kube-Proxy

- Assign IP to each pod.
- It is required to assign IP addresses to Pods (dynamic)
- Kube-proxy runs on each node & this makes sure that each pod will get its unique IP Address.
- These 3 components collectively consist of 'node'.

POD



POD

All containers in Kubernetes are contained within Pods. The simplest component of the Kubernetes architecture is a pod. You may think of them as your container's equivalent of a wrapper. Each pod receives a unique IP address that it can use to communicate with other pods in the cluster.

Pods typically only contain one container, however, if several containers are required to share resources, they may do so. A Pod's multiple containers can connect with one another over localhost if there are multiple containers in it.

Since Kubernetes pods are "mortal," they can be generated and deleted as needed by the application. For instance, the backend of a web application may experience a spike in CPU utilization. Eight new Pods would be formed if the cluster scaled up the number of backend Pods from two to ten as a result of this circumstance. The Pods may scale back to two after the traffic dies down, in which case eight pods would be destroyed.

POD

Here is an example of Pod manifest:

my-apache-pod.yaml

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4  name: apache-pod
5  labels:
6    app: web
7 spec:
8  containers:
9  - name: apache-container
10   image: httpd
```

Each manifest has four necessary parts:

1. The version of the API in use
2. The kind of resource you' d like to define
3. Metadata about the resource
4. Though not required by all objects, a spec, which describes the desired behavior of the resource, is necessary for most objects and controllers.

In this example, the kind is a Pod and the API version is v1. The metadata field is where names, labels, and annotations are applied. While labels, which will be more important for defining Services and Deployments, group like resources, names distinguish resources. Annotations allow you to add any type of data to the resource.

POD

- The smallest unit in Kubernetes.
- POD is a group of one or more containers that are deployed together on the same host.
- A cluster is a group of nodes.
- A cluster has at least one worker node and a master node.
- In Kubernetes, the control unit is the POD, not the containers.
- Consist of one or more tightly coupled containers.
- POD runs on a node, which is controlled by the master.
- Kubernetes only knows about PODs (Does not know about individual containers).
- Cannot start containers without a POD.
- One POD usually contains One Container.

Multi Container PODs →

- Share access to memory space.
- Connect using Localhost (Container-Port)
- Share access to the Same Volume.
- Containers within POD are deployed in an all-or-nothing manner.
- The entire POD is hosted on the same node (Scheduler will decide which node).
- There is no auto-healing or scaling by default.

Higher-level Kubernetes Objects

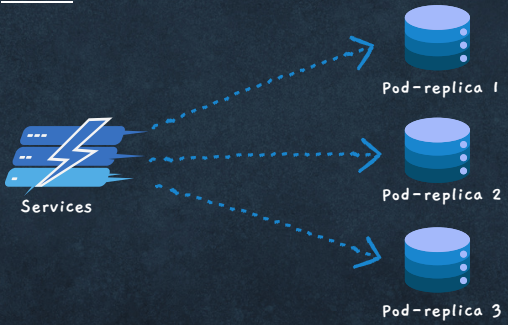
- **Replication Set** → Auto scaling and auto-healing.
- **Deployment** → Versioning and Rollback.
- **Service** → Static (Non-ephemeral) IP and Networking.
- **Volume** → Non-ephemeral storage [Ephemeral → Storage outside the node]

Working of Kubernetes

Pods

Pods can have one or more containers coupled together. They are the basic unit of Kubernetes. To increase High Availability, we always prefer pods to be in replicas

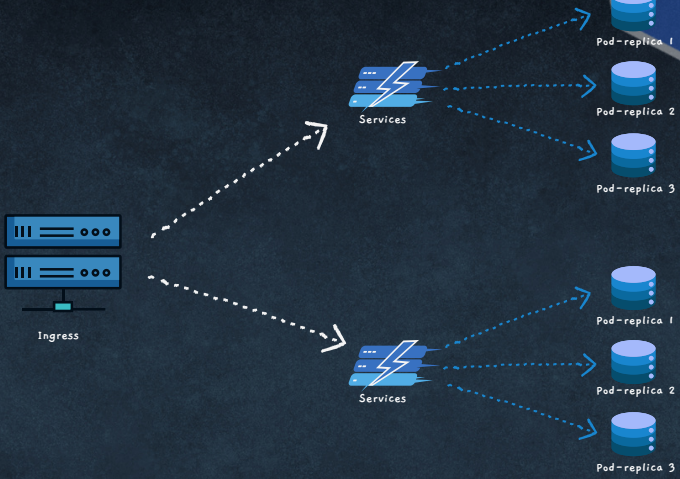
Service



Services are used to load balance the traffic among the pods. It follows round robin distribution among the healthy pods

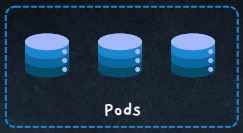
Working of Kubernetes

Ingress



An Ingress is an object that allows access to your Kubernetes services from outside the Kubernetes cluster. You configure access by creating a collection of rules that define which inbound connections reach which services.

Deployments in Kubernetes



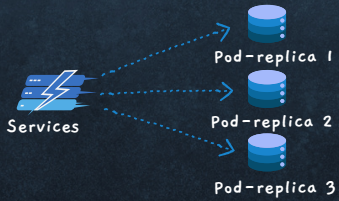
Deployment in Kubernetes is a controller which helps your applications reach the desired state, the desired state is defined inside the deployment file

Creating a Deployment

Syntax

```
kubectl create -f nginx.yaml
```

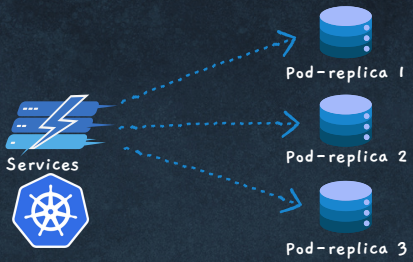
Creating a Service



A Service is basically a round-robin load balancer for all the pods, which match with it' s name or selector. It constantly monitors the pods, in case a pod gets unhealthy, the service will start deploying the traffic to the other healthy pods.

Service Types

- ClusterIP: Exposes the service on cluster-internal IP
- NodePort: Exposes the service on each Node's IP at a static port LoadBalancer: Exposes the service externally using a cloud provider's load balancer.
- ExternalName: Maps the service to the contents of the ExternalName



Creating a NodePort Service

Syntax

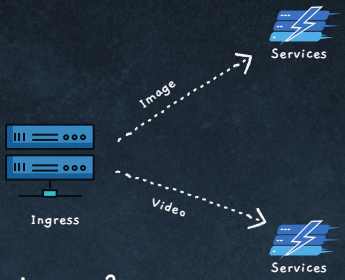
```
kubectl create service nodeport <name-of-service> --
      tcp=<port-of-service>:<port-of-container>
```

To know the port, on which the service is being exposed

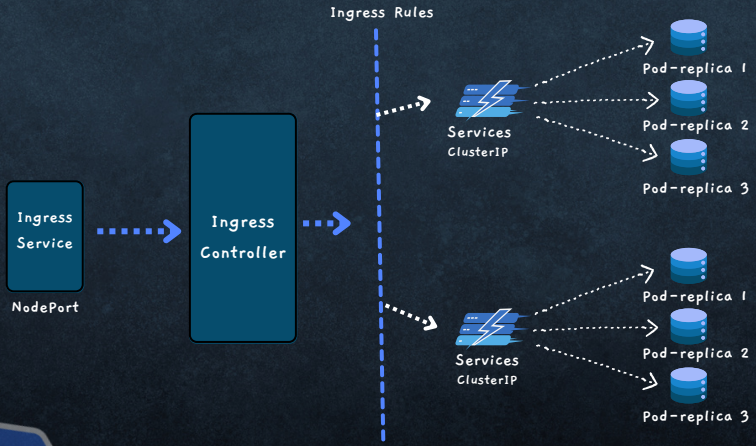
```
kubectl get svc nginx
```

Creating an Ingress

Kubernetes ingress is a collection of routing rules that govern how external users access services running in a Kubernetes cluster.



What is an ingress?



Kubernetes Objects

- Kubernetes uses objects to represent the state of your cluster.
- What Containerized applications are running (and Which node)?
- The policies around how those applications behave such as restart policies, upgrades, and fault tolerance.
- Once You create the object, the Kubernetes system will constantly work to ensure that the object exists and maintains cluster's desired state.
- Every Kubernetes object includes two nested fields that govern the object Config. The object spec and object status.
- The spec, Which We provide, describes your desired state for the object and the characteristics that you want the object to have.
- The status describes the actual state of the object and is supplied and updated by the Kubernetes system.
- All objects are identified by a unique name and a UID.

Relationship b/w these Objects

- Pod manages Containers.
- Replica set manage Pods.
- services expose - Pod Processes to the outside world.
- Configmaps and secrets help you configure Pods.

Kubernetes Object

- It represents as JSON or YAML files.
- You create these and then push them to the Kubernetes API with Kubectl.

Labels and Selectors

- Labels are the mechanism you use to organize Kubernetes Objects.
- A Label is a Key-value Pair without any predefined meaning that Can be attached to the objects
- Labels are Similar to tags in AWs or git where you use a name for a quick reference.
- So you are to Chase labels it as You need it to refer to an environment that is used for dev or testing or Production, refer a Product group like Department A, Department B.
- Multiple labels can be added to a single object.

Labels-Selectors

- Unlike name/UIDs, labels do not provide Uniqueness, as in general, We can expect many objects to carry the same label.
- Once labels are attached to an object, we would need filters to narrow down and these are called label selectors.

Node Selector

- One use case for selecting labels is to Constrain the set of nodes onto which a pod can schedule i.e. you can tell a pod to only be able to run on particular nodes.
- Generally, such Constraints are Unnecessary, as the Scheduler will automatically do a reasonable placement, but in certain circumstances, we might need it.
- We can use labels to tag nodes.
- You the nodes are tagged, so you can use the label selectors to specify the pods run only on specific moves.
- First, we give a label to the node.
- Then use the node selector to the Pod Configuration.

Scaling and Replication

- Kubernetes was designed to Orchestrate multiple constraints and replication.
- Need multiple containers/ replication helps us with these.

Reliability

- By having multiple versions of an application, you prevent problems if one or more falls.

Load Balancing

- Having multiple versions of a container enables you to easily send traffic to different instances to prevent overloading of a single instance or node.

Scaling

- When the load does become too much for the number of existing instances, Kubernetes enables you to easily scale up your application, adding additional Instances as needed.

Rolling Updates

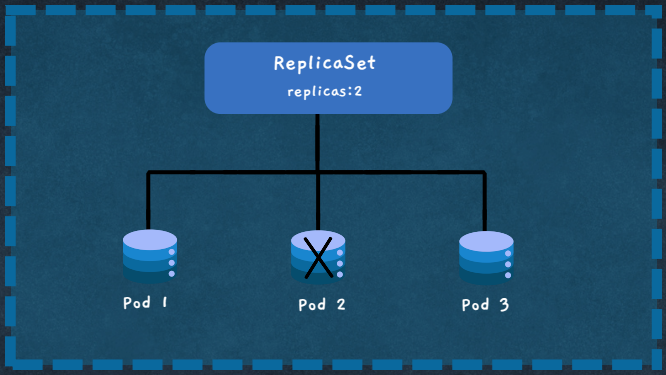
- Updates to a service by replacing pods one by one.

Replication Controller

- A replication controller is an object that enables you to easily create multiple Pods, then make sure that number of Pods always exists.
- If a pad is created using an RC-replication controller will be automatically replaced if they do crash, failed, or is terminated.

- RC is recommended if you just want to make sure 1 Pod is always running, even after the system reboots.
- You can run the RC with 1 replica & the RC Will make sure the Pod is always running.

Replica Set



Replica Set

- A replica set is a next-generation Replication controller.
- The replication controller only supports equality-based selectors whereas the replica set supports set-based selectors i.e. filtering according to set of values.
- The replica set rather than the replication controller is used by other objects like deployment.

Deployment & Rollback

- The replication controller & replica set is not able to do updates & rollback apps in the cluster.
- A deployment object act as a supervisor for pods, giving you fine-grained control over how and when a new pod is rolled out, updated, or rolled back to a previous state.
- When using a deployment object, we first define the state of the app, then the K8s cluster schedules mentioned app instance onto specific individual nodes.
- K8s then monitors, if the node hosting an instance goes down or the pod is deleted the deployment controller replaces it.

The following are typical use cases of Deployments

1. create a deployment to roll out a Replicaset – The replica set creates pods in the background. check the status of the rollout to see if it succeeds or not.
2. Declare the new state of the Pods – By updating the PodTemplateSpec of the deployment. A new replica set is created and the deployment manages to move the pods from the old replica set to the new one at a controlled rate. Each new replica set updates the revision of the Deployment.
3. Rollback to an earlier deployment revision – If the current state of the deployment is not stable. Each rollback updates the revision of the Deployment.
4. Scale up the deployment to facilitate more load.
5. Pause the deployment to apply multiple fixes to its PodTemplateSpec and then resume it to start a new rollout.
6. Clean up older Replicaset that You don't need anymore.

Deployment & Rollback

- You can rollback to a specific version by specifying it with:

```
--to-revision
```

- For eg:

```
kubectl rollout undo
deploy/mydeployments --to-revision=2
```

Note:

- That the name of the ReplicaSet is always formatted as

```
[Deployment-name]-[Random-String]
```

Cmnd

```
kubectl get deploy
```

- When you inspect the deployments in your cluster, the following fields are display.

NAME -List the names of the deployments in the namespace.

READY -Display how many replicas of the application are available to your users. If follows the pattern ready/desired.

- UP-TO-DATE - Display the number of replicas that have been updated to achieve the desired state.
- Available - Displays how many replicas of the application are available to your users.
- AGE - Display the amount of time that the application has been running.

Imp. Cmnds to Know:

- To check deployment was created or not

```
kubectl get deploy
```

- To check, how deployment creates ReplicaSet & Pods

```
-> kubectl describe deploy mydeployments
-> kubectl get rs
```

- mydeployments is an example here

- To scale up/down

```
kubectl scale --replicas=1 deploy mydeployments
```

- To rollout/rollback status

```
kubectl rollout status deployment mydeployments
```

To see history of your versions or about deployments

```
kubectl rollout history deployment mydeployments
```

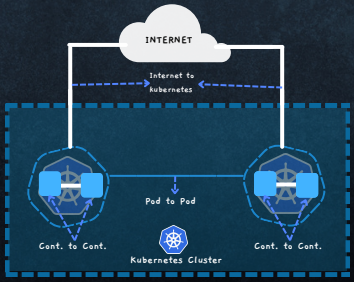
To go previous step

```
kubectl rollout undo deploy/mydeployments
```

Kubernetes Networking

Kubernetes Networking addresses four concerns

- Containers within a pod use networking to communicate via loopback.
- Cluster networking provides communication between different pods.
- The service resources let you expose an application running in pods to be reachable from outside your cluster.
- You can also use the service to publish services only for consumption inside your cluster.
- Container-to-container communication on the same pod happens through localhost within the containers.



Object Services

- Each Pod gets its own IP address, however, in a deployment, the set of pods running in one moment in time could be different from the set of Pods running that application moment later.
- This leads to a problem: If some set of Pods (cal them 'backends') Provide functionality to other Pods ('Call them frontends) inside your clusters how do the frontends find out and keep track of which IP address to connect to, so that the frontend can use the backend part of the workload.
- When using RC, Pods are terminated and created during scaling or replication Operations.
- When using deployments, while updating the image version the pods are terminated and new pods take the Place of other pods.
- Pods are very dynamic i.e. they come and go on the k8s cluster and on any of the available nodes & it would be difficult to access the pods as the Pods IP changes once it's recreated.
- Service allows clients to reliably connect to the containers running in the Pod using the VIP.
- Although each Pod has a unique IP address, these IPs are not exposed outside the cluster.
- Services help to expose the VIP mapped to the pods & allow applications to receive traffic.
- Labels are used to select which the pods to be put under a service.

Service can be exposed in different ways by specifying a type in the service spec.

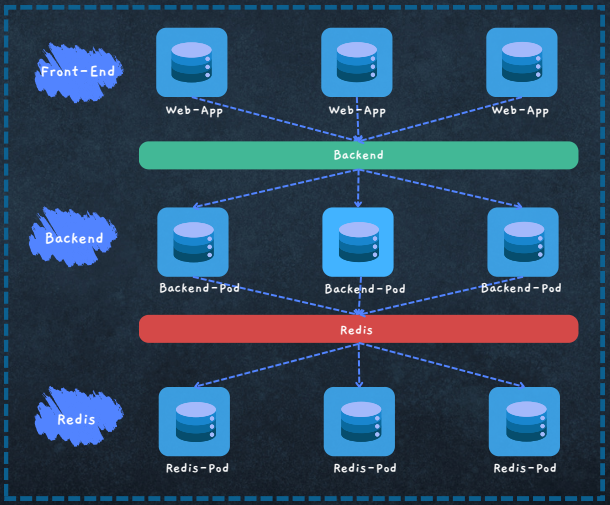
1. Cluster IP
2. NodePort
3. Load Balancer created by cloud providers that will route external traffic to every node on the NodePort (eg-ELB on AWS)

4. Headless – Creates several endpoints that are used to produce DNS records. Each DNS record is bound to a Pod.

- By default service can run only between ports 30,000–32,767
- The set of pods targeted by a service is usually determined by a selector.

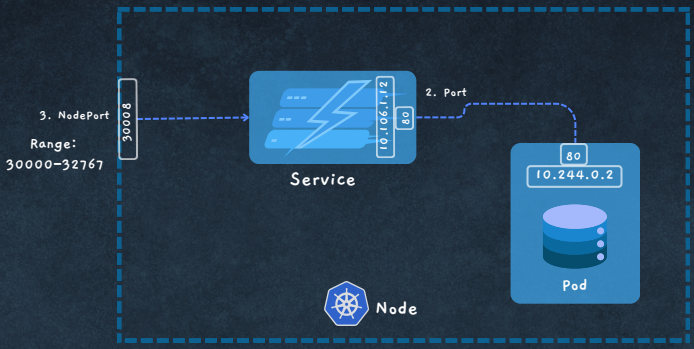
Cluster IP

- Expose VIP only reachable from within the cluster.
- Mainly used to communicate between components of MicroServices.



NodePort

- Makes a service accessible from outside the cluster.
- Exposes the service on the same port of each selected node in the cluster using NAT.

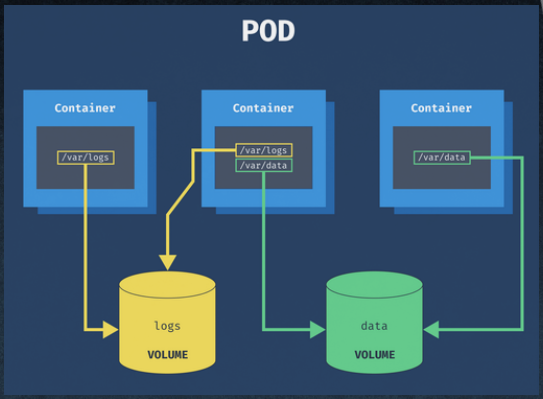


service-definition.yml

```
apiVersion: v1
kind: Service
metadata:
    name: myappp-service

spec:
    type: Nodeport
    ports:
     - targetPort: 80
      *port: 80
       nodePort: 30008
```

Volumes



- Containers are short-lived in Nature.
- All data stored inside a container is deleted if the container crashes. However, the kubelet will restart in a clean state, Which means that it will not have any of the old data.
- To overcome this problem, Kubernetes uses volumes. A Volume is essentially a directory backed by a storage medium. The storage medium and its content are determined by the volume type.
- In Kubernetes, a volume is attached to a Pod and shared among the Containers of that Pad.
- The Volume has the same life span as the Pod, and it outlives the containers of the Pod this allows data to be preserved across container restarts.

Volumes Types

- node-local types such as emptydir and hostpath.
- File sharing types such as nfs.
- Cloud provider-specific types like AWS ElasticBlockStore, AzureDisk.
- Distributed file system types, for example glusterfs or cephfs.
- Special purpose types like secret, gitrepo.

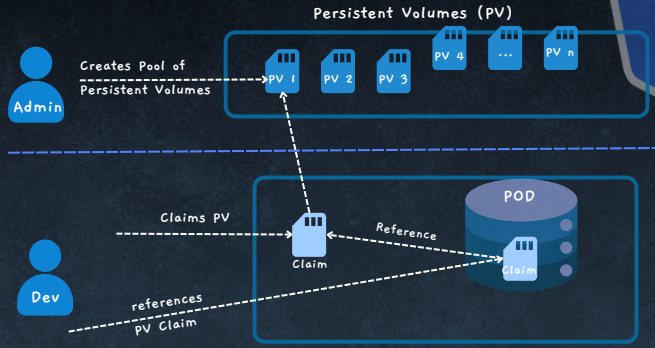
EmptyDir

- Use this when we want to share contents between multiple containers on the same pod & not to the host machine.
- An emptydir volume is first created when a pod is assigned to a node, and exist as long as that pod is running on that node.
- As the name says, it is initially empty.
- Containers in the pod can all read and write the same files in the emptydir volume, though that volume can be mounted at the same or different paths in each containers.
- When a pod is removed from a node for any reason, the data in the emptydir is deleted forever.

HostPath

- Use this when we want to access the content of a pod/container from the host machine.
- A hostpath volume mounts a file or directory from the host node's filesystem into your pod.

Persistent Volume



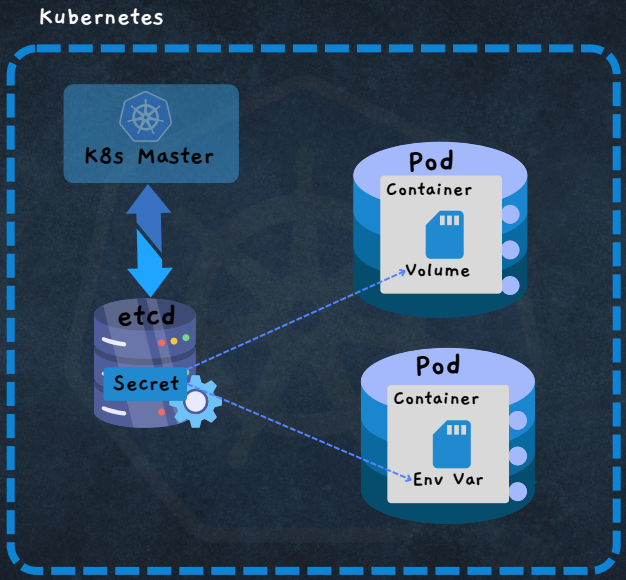
- A persistent volume (PV) is a cluster-wide resource that you can use to store data in a way that persists beyond the lifetime of a pod.
- The PV is not backed by locally attached storage on a work node but by a networked storage system such as EBS or NFS or a distributed filesystem like ceph.
- K8s provides APIs for users and administrators to manage and consume storage. To manage and consume storage. To manage the volume, it uses the persistent volume API resource type and to consume it, uses the persistent volume-claim API resource type.

Persistent Volume Claim

- In order to use a PV you need to claim it first, using a persistent volume claim (PVC).
- The PVC requests a PV with your desired specification (size, accessnodes, speed etc) from Kubernetes, and once a suitable Persistent volume is found, it is bound to a PersistentVolumeClame.

- After a successful bond to a pod, you can mount it as a volume.
- Once a user finishes their work, the attached PersistentVolume can be released. The underlying PV can there be reclaimed and recycled for future usage.

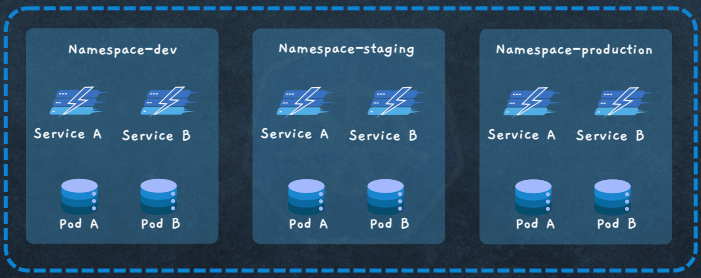
Secrets in Kubernetes



- You don't want sensitive information such as a database password or an API key kept around in clear text.
- Secrets are name spaced objects, that is exist in the context of a namespace.

- You can access them via a volume on an environment variable from a container running in a pod.
- The secret data on nodes is stored in tmpfs volume (tmps is a filesystem that keeps all files in virtual memory.) Everything in tmpfs is temporary in the sense that no files will be created on your hard drive.
- A per-secret size limit of 1 MB exists.
- The API server stores secrets as plaintext in etcd. Secrets can be created –
    1. From a text file.
    2. From a YAML file.

Namespace in Kubernetes



- A namespace is a group of related elements that each have a unique name or identifier. A namespace is used to uniquely identify one or more names from other similar names of different objects, groups, or the namespace in general.
- A mechanism to attach authorization and policy to a subsection of the cluster.

Create namespace:

demo-namespace.yml

```
apiVersion: v1
kind: Namespace
 metadata:
 name: demo
 labels:
   env: dev
```

```
$ kubectl create -f demo-namespace.yml
$ kubectl get namespaces -o=json
$ kubectl get namespaces demo -o=json/yaml/name
```

Using Namespace:

Get an object from namespace

```
kubectl namespace=<namespace> get pods
```

Setting default namespace

```
  $ kubectl config set-context
  $(kubectl config current-context) — namespace=<name>
  $ kubectl config view | grep namespace:
```