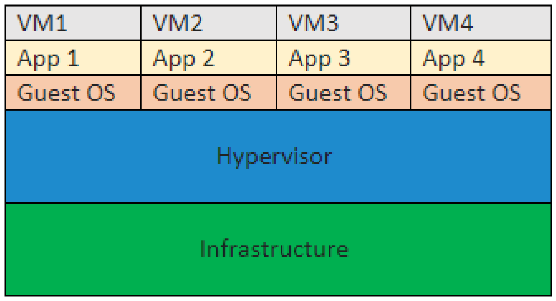
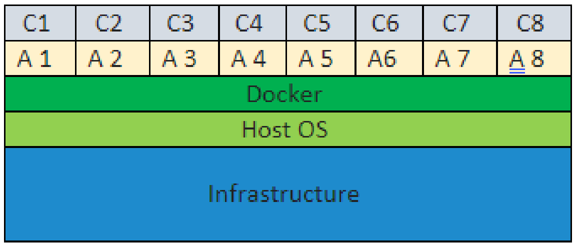
Cloud-Based system - Kubernetes, Persistence Architecture

We live in an era where a massive amount of data is generated every second, and several queries are handled each second. But how is this managed and stored??

Previously, we used to store all applications in a single system. Everything was working fine until our applications were small and easily managed. As the application size grew, the problem begins. The applications which need more resources hamper other applications' performance. To handle the resource requirements of each application, we need new systems. Maintaining several systems was costly for organizations.

To handle this problem, organizations started using Virtual Machines. Now on a single system, different VMs can be created, and each application can run on it as if it is running on a separate network. Each VM has its own set of OS, so they use more memory.





For more efficient memory management containers, they are similar to VMs, but they don't have their OS like VMs. Containers have their filesystem, CPU, memory, etc. same as VMs. It also provides numerous benefits like customized image, CI/CD (Continuous Integration and Continuous Deployment), separability of applications, efficient resource utilization, portability.

Now, manual monitoring, load balancing, version update, and auto-scaling are tedious tasks. All these problems are handled efficiently by Kubernetes. It runs smoothly on public, private, and hybrid clouds.

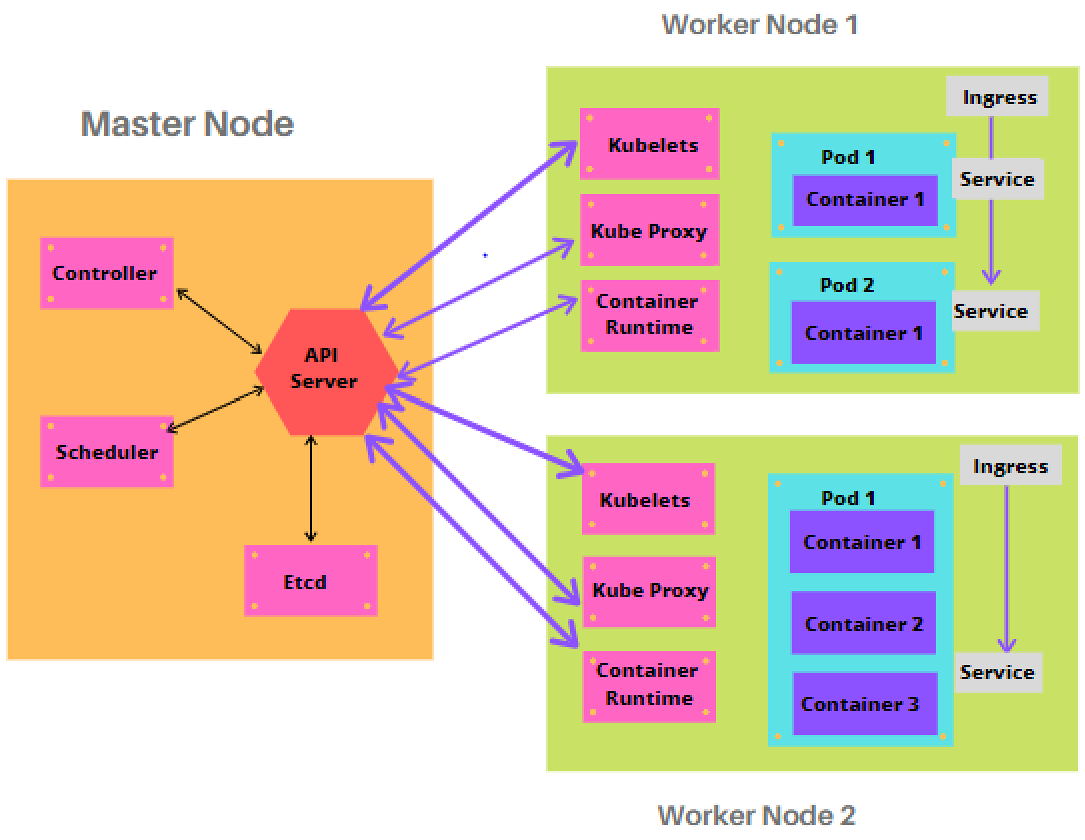
Kubernetes is an orchestration tool developed by Google to manage containerized applications in different environments like a physical machine, virtual machine, cloud, or hybrid environment. It is a highly available, scalable, and reliable system.

It means Kubernetes helps us manage hundreds and thousands of containers in different environments. It is highly available means have zero downtime. Scalable means we can increase or decrease resources as per need. Portable means container, once created, can run on different systems. Also, several replicas can be made as per requirement.

Suppose an organization has 100 microservices running on the separate containers on the Docker engine. In that case, if anyone fails, the Kubernetes will create another Pod containing the same image and destroy an unhealthy one. It will keep all replicas of the Container up and running.

Kubernetes handles these functionalities by the master and worker nodes. There can be at least one master node and several worker nodes. Each worker node has a Kubelet process running on it. Kubelet is a Kubernetes process that makes it possible for the cluster to communicate with each other and executes some tasks on nodes like the running application process. Each worker node has a container of several applications deployed on it. Depending on the workload requirement, we have several Pod running in the node. The actual application runs on the worker node.

The master node has processes like an API server, Controller, Scheduler, Etcd. Worker nodes have Kubelets, Container Runtime, Kube Proxy, Pods, and other additional configurations. In Kubernetes, we can use the YAML file or JSON file to give instructions or commands. The Master node handles essential processes, and the worker nodes follow the instruction given by it.



API Server

It is the entry point to the Kubernetes cluster. Different K8s clients interact with the API server like UI, API, CLI, etc. It is a cluster gateway and acts as a gatekeeper for authentication. All the requests coming to the application handled by it then gets redirected to nodes. It is the most important component that acts as immediate between controller, scheduler and components of worker node. It exposes API of Pod to end users and handles request for specification of labels and images. Labels such as one pod for production other for development.

Controller

It tacks whatever is happening in the cluster, whether something needs to repair or if the Container dies, it makes another replica.

Scheduler

It ensures each Container is placed correctly according to workload and available resources. It decides the next Pod should be scheduled on which worker node. When user want to create a container, the request will go to API server. Then it will tell Kubelet that one container needs to be added. If there is enough space in previously available Pods then scheduler will inform API server to create pod in that node otherwise new pod is created and container is placed in it.

Etcd

Etcd is a database or cluster brain. API server stores data of Pod in Etcd in the form of key-value pairs. All data related to config, secret, replica, controller, roll bindings, etc. are also stored in Etcd. It holds the current state of the K8s cluster, except for application data, Etcd stores all data.

Pod

It is the smallest unit of the Kubernetes cluster. The Pod is a wrapper of a container. Inside of a Pod, we can have multiple containers running. Usually, we have one Pod for each Container having volume and port space. The Port number of each Container in Pod is unique. Each Pod is its self-contained server with its IP address. We only work with Pod inside the Kubernetes cluster, which is an abstraction of a container. Pod communicates internally with the help of internal IP addresses. Why Container in Pod only? Why not Container only? It is because of K8s support multiple container runtimes like docker, rocket, etc. If we install the Container, then it will be specific to either of docker or rocket. Also, Pod handles configuration regarding their containers. So, if any Container is unhealthy, then Pod configuration helps to create another copy of the Container.

Service

It is a substitute for the IP addresses whenever a Pod dies; a new IP address is assigned to the Pod that needs to be updated everywhere. Instead of having dynamic IP addresses, we use Services for each Pod that communicates with each other by its fixed IP and performs load balancing. The life cycle of service and Pod is not the same, so even if Pod dies, the service remains.

Ingress

It routes traffic into the cluster. We need an external service if we want our application to be accessible from a browser. External service opens communication from external sources. So external requests go to Ingress, and it forwards to service.

Config Map

Stores data of external configuration of applications like URLs of database or services.

Secret

It is the same as the config map that but used to store credentials in encoded format.

Volume

In Kubernetes, each container can read and write to its own, isolated filesystem inside pod. But, data on that filesystem will be destroyed when the container is restarted. To solve this, Kubernetes has Persistent Volume. It is configured explicitly.

Kubelet

It interacts with both the Container and node and is responsible for monitoring Pod. It starts the Pod with the Container inside it. Kubelet manages only those containers that created by Kubernetes; manually created containers are not handled by it. Kubectl commands from terminal window

```
kubectl [command] [TYPE] [NAME] [flags]
```

i.e.

```
kubectl get pod pod1
```

```
kubectl create -f firstpod.yaml
```

Kubeproxy

It handles traffic requests of the application it takes all the network related rules. It is an intelligent process that transfer request to the nearest node available.

Container runtime

The container runtime is the software that is responsible for running containers. K8s supports several container runtimes: Docker, Container, CRI-O, and any implementation of the Kubernetes CRI (Container Runtime Interface).

Addons

K8s provide various addons like DNS, Web UI, Container Resource Monitoring, etc. to offer other functionalities.

If we want to add a container, the request will go to the API server, and the API server will interact with the scheduler. The scheduler will then check if there is enough space available in other nodes to maintain load balancing. If there is enough space, the scheduler will inform the API server to send the Container to the node; otherwise, a new node with the Pod is created.

The master node manages interaction with the cluster like scheduling, monitoring, restart, joining new nodes, etc. and worker nodes perform the task assigned by the master node.

In Kubernetes, we work with deployment and StatefulSet. Deployment is an abstraction of Pod. It provides us the convenience of using stateless applications and adding other configurations for replicas, etc. StatefulSet is useful for application or database where we need to maintain states. In practice, there are multiple master node and worker nodes.

So, steps in Kubernetes are:

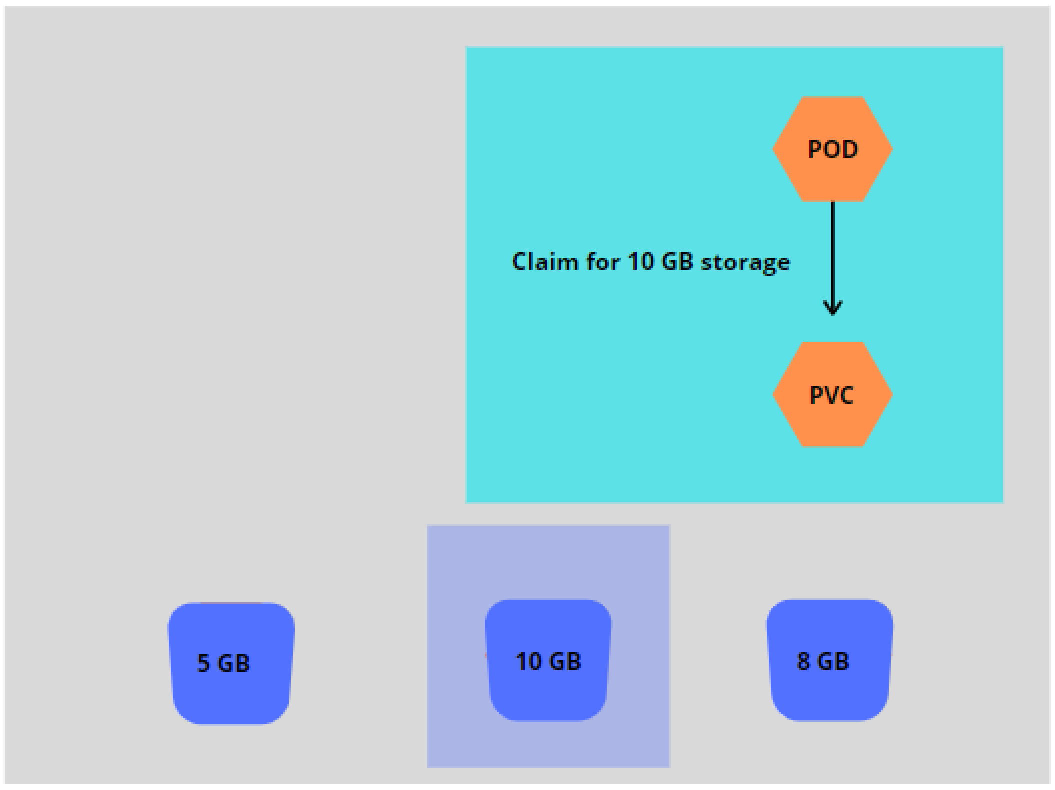
- Create a Cluster
- Deploy an App

- Explore Your App
- Expose Your App Publicly
- Scale Your App
- Update Your App

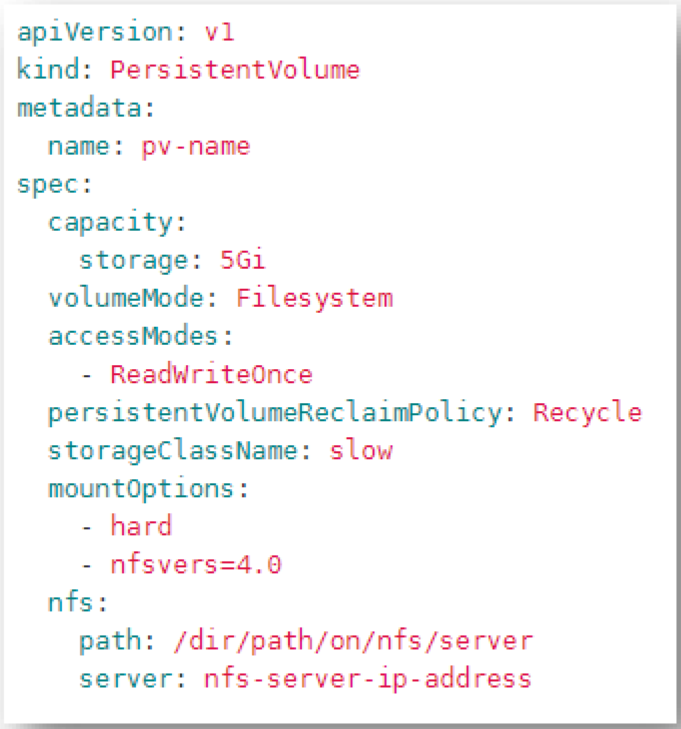
Kubernetes Persistent

Consider a case where we have a MySQL pod that our application uses. Data gets created and updated on the database in the Pod. But when we restart the Pod, the changes are lost. It is because data is not stored permanently. We need to configure explicitly as the data storage should not depend on the storage life cycle. It should be available even if the Pod dies. The new Pod will pick up from where it left off by reading data from existing storage.

The new Pod can restart on any node, so the storage must be available on all nodes in the Kubernetes cluster. Also, we need high availability so that the service should continue even if the Kubernetes cluster crashes.

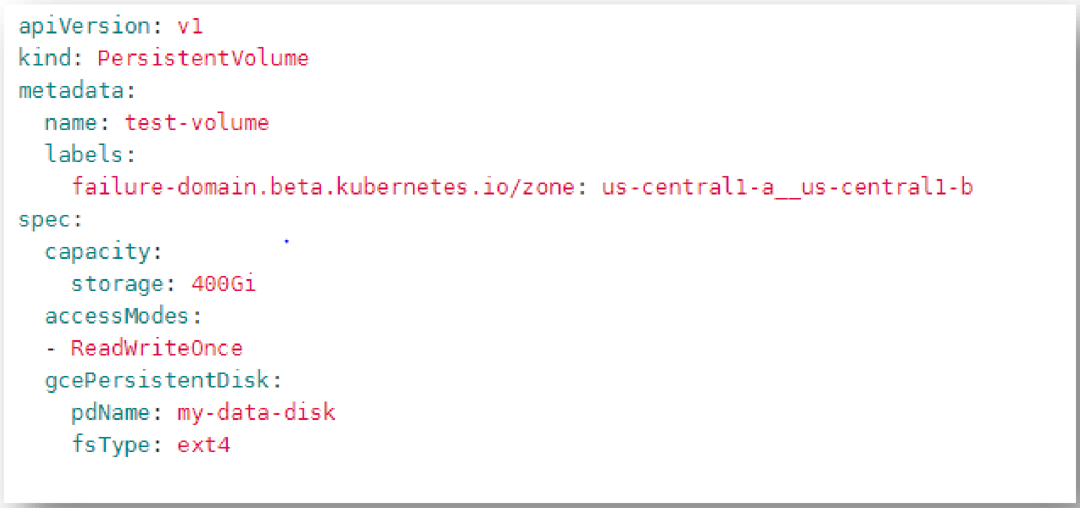


Kubernetes' Persistent Volume (PV) is a cluster resource such as RAM, CPU that stores data. PV, just like other components that get created using the YAML file. We only need to define storage type and size as per requirements in the YAML file's spec section. PV is not namespace; they are available to the whole cluster.



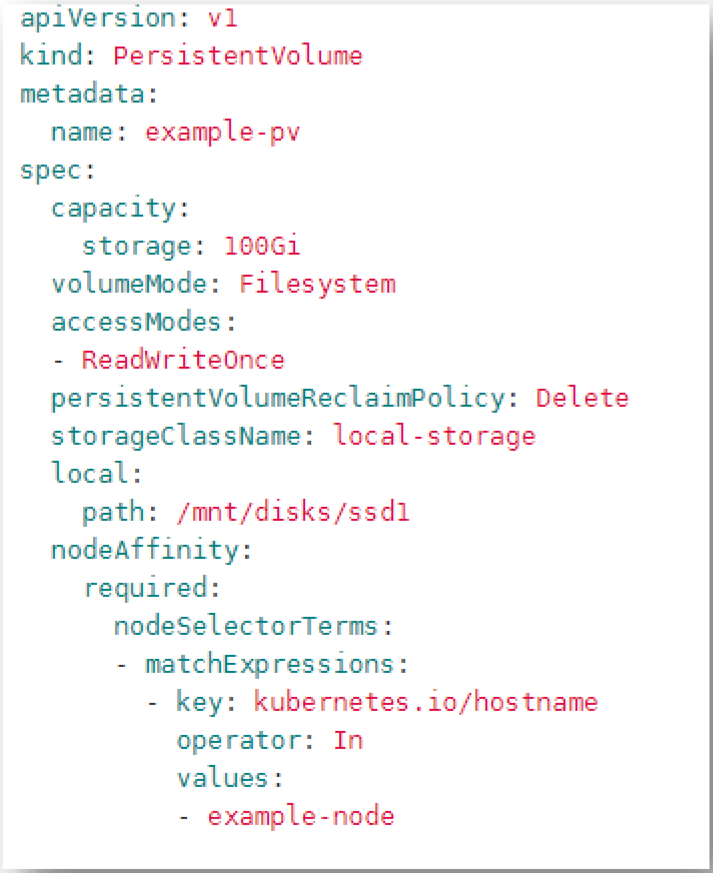
```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-name
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.0
  nfs:
    path: /dir/path/on/nfs/server
    server: nfs-server-ip-address
```

Data stored on local NFS storage



```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: test-volume
  labels:
    failure-domain.beta.kubernetes.io/zone: us-central1-a__us-central1-b
spec:
  capacity:
    storage: 400Gi
  accessModes:
  - ReadWriteOnce
  gcePersistentDisk:
    pdName: my-data-disk
    fsType: ext4
```

Data stored on cloud storage



```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 100Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: /mnt/disks/ssd1
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - example-node
```

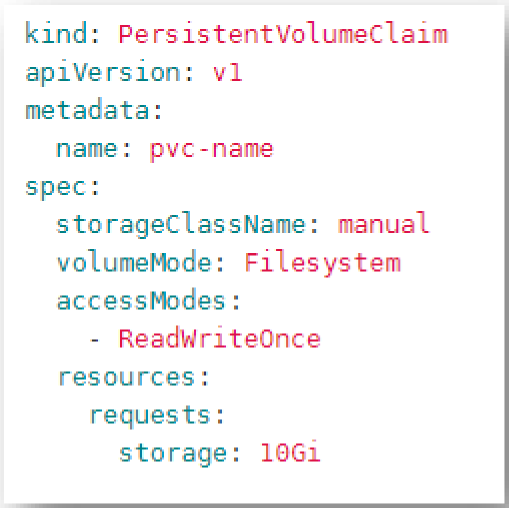
Data stored on node itself which has additional nodeAffinity attribute.

PV is just an abstract component. It must have actual storage like a local hard drive, external NFS servers outside of the K8 cluster, or cloud storage like AWS, block storage, or Google cloud storage.

K8 doesn't care about the actual data storage, as it is an external plugin of the K8 cluster. It gives a persistent volume component as an interface to the existing repository. The administrator decides what type of storage is needed by applications and services and how to take it's back up and prevent it from being corrupted by maintaining replicas and adding authentication and authorization rules.

Persistent Volume like RAM and CPU should be there in the Kubernetes when the Pod gets created. Two roles in K8, one is of administrator and the other of the user. The system administrator or DevOps engineer or company set up the cluster and maintains it, and make sure there is enough space available. K8 user deploys the application either directly or through CI pipelines. Developers create their application and deploy in Kubernetes, and administrator config the actual storage like NFS or cloud storage. There are two volume type ConfigMap and Secret.

Developers need to explicitly configure the YAML file to use the persistent volume, i.e., the application has to claim the storage. The claim is made by a Persistent Volume Claim (PVC). PVC claims the storage with some capacity and storage type with access type like read-only, write-only, etc. And whichever matches the criteria will be used for the application. As Pods consume node resources similarly, PVCs consume PV resources. As, Pods can request specific levels of resources (CPU and Memory). Claims can request exact size and access modes (e.g., ReadWriteOnce, ReadOnlyMany or ReadWriteMany,etc.).



```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-name
spec:
  storageClassName: manual
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Example of PVC YAML file.

Here, the administrator provisions storage resources and Pod claims for request through PVC and allocate the storage to the pod. Now if pod dies or restarts, it takes values from the persistent storage. As long as data is safely stored no need to worry about the storage. PersistentVolume types are implemented as plugins. Kubernetes supports plugins like GCEPersistentDisk, AWSElasticBlockStore, AzureFile, AzureDisk, CSI, FC (Fibre Channel), FlexVolume, NFS, etc.