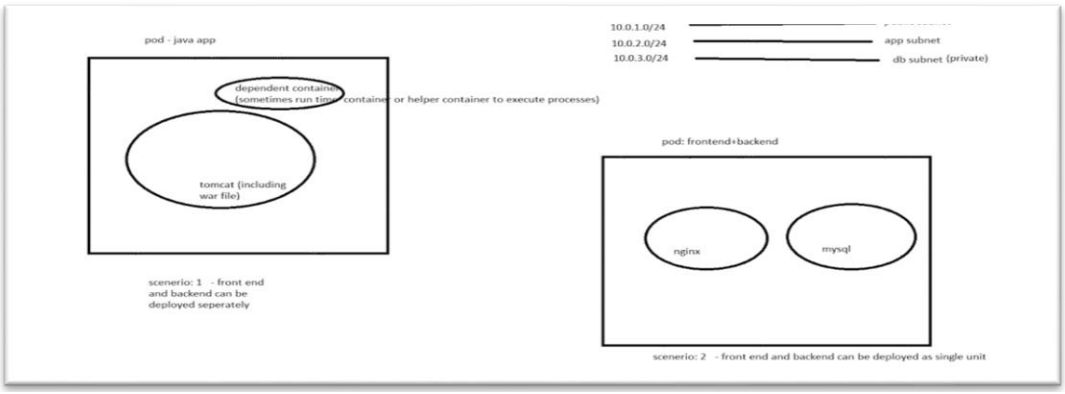
Kubernetes Cluster- A K8s Cluster consists of a

Cluster Master
Worker Nodes.

What is pod?

- Pod is the smallest unit which can be deployed in Kubernetes.
- Usually, we restrict any pod deployment on K8 Master.
- It contains one or more than one container (1st is Main container and 2nd is dependent container-sometimes run time container or helper container to execute process).
- It's a small group of tightly coupled containers.
- **Interview question**-(k8s assigns IP to Pods only)
- We have 2 methods for pod deployment-
    1) Front end and backend container can be deployed separately-
    2) Front end and backend can be deployed as a single unit.
- Generally, we prefer 1st method because in 2nd method there are chances of data corruption/loss.

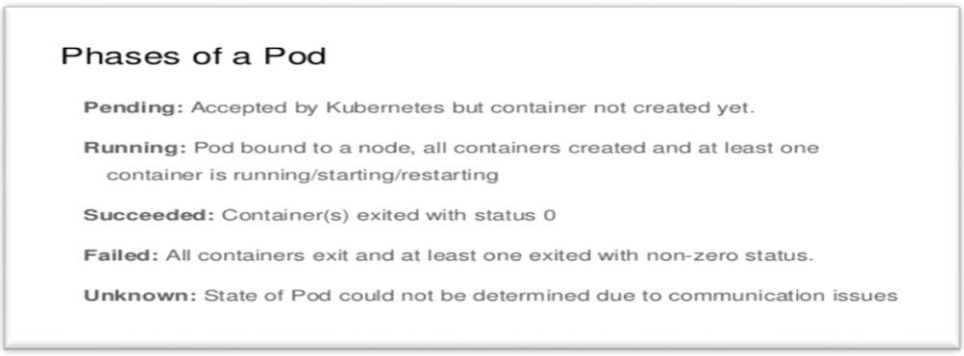


- **Pod Creation command**
  kubectl run -it –image=<image name>:<pod name>

  Example-
  kubectl run -it --image=pkw0301/nginx:latestmynginx
  kubectl=command line utility,run=create & run pod,-it=interactivemode & terminal

  pkw0301/nginx=docker image,Mynginx=pod name

- **Lifecycles of Pods**



1. **Master Component: -**

   1) **kubectl:**

   Command line utility (Means we can execute commands)

   When you use the kubectl command-line interface, for example, the CLI makes the necessary Kubernetes API calls for you.

   2) **Kube api server:**

   - This is a heart of k8s.
   - To work with Kubernetes objects–whether to create, modify, or delete them–you'll need to use the Kubernetes API.
   - It checks user authentication.
   - User authorization (create, get, put, update, delete etc)
   - It generates the key and send it to the user.
   - Exposes the K8s API. It's the frontend for Kubernetes control

   3) **etcd:**
   - it stores data (any type of data user related/pod related/node related/cluster related/network related/applications info/token/secret etc) in key value format.

   4) **scheduler:**

   - is responsible for scheduling pod creation on k8s node.
   - It checks which node is free accordingly it passes the info. to create the pod.
   - You can deploy pods on any specific node using affinity or anti-affinity rules.

   5) **Controller- Manager:**

   - kube-controller- manager —process that run controller to handle cluster background task such as.

- Example-If some pot is not working properly, to create new pot or not that decision power is with controller manager.
- cloud-controller-manager — Runs controllers that interact with cloud

Node Components:

6) **kubelet:** -

- is responsible for sharing node resource info such as CPU/memory/disk usage to k8s master or to kube api server.
- is also responsible for sharing pod info such as running state/stopped state etc. to kubeapi server.
- it also helps k8s to create new pods on k8s node instances
- Responsible for everything on the Worker Node. It communicates with the Master's API server. Think *brain* for Worker Node.

7) **kubeproxy:**

- is responsible for route network connection between pods (consider front-end and backend pods).
- is also responsible for load balancing between pods.

8) ***Container Runtime*** — Downloads images and runs containers. For example, Docker is a Container Runtime. Think *Docker*.

KubernetesArchitecture –

1. **Kubernetes Cluster**:

   Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit.
2. A Kubernetes cluster consists of two types of resources:
   - ❖ The Master coordinates the cluster
   - ❖ Nodes are the worker that run applications

3. In Kubernetes we can interact with kubeapiserver (or Input to the kubeapiserver) through-
   - ❖ Kubectl
   - ❖ Dashboard

4. Before processing, kube-apiserver 1st checks whether user is authorized or not.

5. If user is authorized, kubeapiserver checks **the commands** and send it to controller manager.

   Here both are working together and checks what type of activity user wants to perform.

After this, kube- api-server passes the **instruction** to the etcd (all data is stored in key value format). etcd accordingly checks the instruction and passes the key to the kubeapiserver.

6. Then kube apiserver passes the instruction to the scheduler. In the backend scheduler connects with the etcd.

7. Scheduler passes the instruction (Pod creation) to the kubelet for pod creation by checking availability of node or state of node.

8. Purpose of kubeproxy is networking.
9. In k8s all components are imp and they work together. If any component get failed, k8s will not provide required output.

Advantage:-

Due to this type of architecture, we can handle millions of Pots.
For Kubernetes master we need t2 medium instance (This is chargeable service.).Bz 1 GB free space is required. For worker nodes we need t2 micro.

Resources in kubernetes/The Six Layers of K8s

1. Deployments,
2. replica sets,
3. pod,
4. Node cluster,
5. Node processes,
6. Docker container,
7. replication controller,
8. Persistent volume,
9. Persistence volume claim

Pod:

Pod is the smallest unit which can be deployed. It contains one or more than one container.
Usually, we restrict any pod deployment on K8 Master.

Problems:-

- There is no replica connect (you can't scale environment automatically. it doesn't support horizontal scaling)
- Desired state cannot be managed.
- It doesn't support selector feature (supports only labels)

Replication Controller (RC):

- It supports replicas (you can scale environment automatically; IT support horizontal scaling HPA (horizontal pod scaling).
- It supports desired number (if you want to run 100 copies of your apps, then k8s will ensure to run 100 pod at any moment).
- It supports set based selector (full fledge)
- Replication Controllers perform the same function as ReplicaSets , but Replication Controllers are old school.
- ReplicaSets are the smart way to manage replicated Pods in K8.

problem:
1. Rolling update feature is missing , it's very diff to roll out new feature.

Replica Sets (RS):

1. It supports replicas (you can scale environment automatically; It support horizontal scaling HPA (**horizontal pod scaling)**. It runs those copies replicas of pod which are useful. It starts to kill pods if necessary. It handles pods failures and it does help check as needed.
2. **Replica Set creates and manages Pods**. If a Pod shuts down because a Node fails, a Replica Set can automatically replace the Pod on another Node.
3. It supports desired number (if you want to run 100 copies of your apps, then k8s will ensure to run 100 pod at any moment)
4. It supports set based selector (it gives us more flexibility to deploy resources in dev/qa/prod.)
5. It supports Roll out (from deployed version 1 to version2), we can roll update on the fly.
6. A Volume can be attached to a Replica Set, but it's required for a StatefulSet.

==========

What do you do when your app has state you need to keep track of?

Use a **StatefulSet**-

Like a ReplicaSet , **a StatefulSet manages deployment and scaling of a group of Pods based on a container spec.**

Unlike a Deployment, a **StatefulSet's Pods are not interchangeable. Each Pod has a unique, persistent identifier that the controller maintains over any rescheduling.**

StatefulSets for good for persistent, stateful backends like databases. The state information for the Pod is held in a Volume associated with the StatefulSet.
Like a ReplicaSet for stateful processes. Think *state*.

DaemonSet-

DaemonSets are for continuous process. They run one Pod per Node. Each new Node added to the cluster automatically gets a Pod started by the Daemon Set. Daemon Sets are useful for ongoing background tasks such as **monitoring and log collection.**

StatefulSets and DaemonSets are not controlled by a Deployment. Although they are at the same level of abstraction as a ReplicaSet, there is not a higher level of abstraction for them in the current API. One automatic Pod per Node. Think *monitor*.

Problems:
1. It support set-based selector but there are so many limitation (feature is there but not full fledge)
(you can use it for small env, people are still using RS)

2. The major difference is that the **rolling-update command** works with Replication Controllers, but won't work with a Replica Set. **This is because Replica Sets are meant to be used as the backend for Deployments.**

Deployment:

- It has all the features of pod, RS and RC.
- It supports replicas (you can scale environment automatically; IT support horizontal scaling HPA (horizontal pod scaling).
- It supports desired number (if you want to run 100 copies of your apps, then k8s will ensure to run 100 pod at any moment).
- It supports set based selector (full fledge).
- It supports Roll out (from deployed version 1 to version2), we can roll update on the fly. Also, we can Roll back to the previous versions.
- you can roll out new feature without any downtime (it has max unavailability [25% , it means k8s can delete only 25% of total deployed resource] and max surge [25% , k8s can create 25% new resources] feature)
- **It can be used for declaration of your application which contain the image tag etc, and environmental variables data volumes.**
- **Declares the state of the pods** by updating the pod template spec of their deployment, a new replica that was created and the deployment managers moving the pods from the old replica sets to the new one at a controlled rate (all are pods are not getting updated in a single shot). Each new replica sets updates the revision of the deployment.
- Pause the deployment, this is to apply multiple fixes to its pod templates, and then resume it to start a new rollout.
- Use the status of that deployment as an indicator that the rollout has stuck.
- Clean up old replica sets. If they don't need any more
- If you want to make a stateless app that will run continuously, such as an HTTP server, you want a Deployment.
- Deployments allow you to update a running app without downtime. Deployments also specify a strategy to restart Pods when they die.
- You can create a Deployment from the command line or a configuration file.
- ----------**Job**: Run a container to completion. Think *batch*.

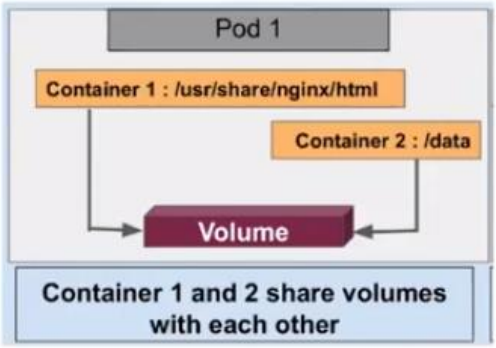
CronJob: Repeated Job. Think *time*.

Volume: Holds data. Think disk.

Persistence volume (PV)/Persistence volume claim (PVC)-

System for allocating storage . Think *storage claim*.

A single volume can assign to two or more containers in the same pod to share data. Volume lifecycle is depended on the pod in which it is put. It's a place where the developer keeps the data for his application support it can be some part of coding.



A ***PersistentVolume*** **(PV)** is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes but have a lifecycle independent of any individual Pod that uses the PV.

A ***PersistentVolumeClaim*** **(PVC)** is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (e.g., they can be mounted ReadWriteOnce, ReadOnlyMany or ReadWriteMany, see AccessModes).

https://kubernetes.io/docs/concepts/storage/persistent-volumes/

Access Modes	A PV can have the following access modes: • ReadWriteOnce—enables read and write and can be mounted by only one node • ReadOnlyMany—enables read only and can be mounted by multiple nodes • ReadWriteMany—both read and write, can be mounted by several nodes Note: Different storage plugins support some of these access modes.
Reclaim Policy	The reclaim policy specifies what happens when the node no longer needs the persistent storage. It can be set to **Retain**, meaning the PV is kept alive until it is explicitly deleted; **Recycle**, meaning the data is scrubbed but can be restored later; and **Delete**, meaning it is irreversibly deleted. Note: Different storage plugins support some of these reclamation policies.

Install awscli on k8s master node:

https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-linux.html

==========