LABELS AND SELECTORS

- Labels are the mechanism you use to organize K8s object
- A Label is a key value pair without any predefined meaning that can be attached to the object.
- Labels are similar to tags in AWS or git where you use a name to quick reference.

```
Kind: Pod
apiversion: v1
metadadat:
    name: delhipod
    labels:
        env: development
        class: pods
Specification
    containers
        - name: c00
          image: ubuntu.
```

```
kubectl apply -f Pod.yml
kubectl get pods --show-labels.
```

- Now, if you want to add a label to an existing pod.

```
kubectl label pods delhipod myname = bhupinder
```

- list pods matching a label

```
kubectl get pods -l env = development

kubectl delete pod -l env! = development

kubectl get pods
```

Labels do not provide uniqueness, many objects to carry the same label

The Api currently supports two type of selectors, 1) Equality based and 2) Set based.

A label selector can be made of multiple requirements which are comma separated.

```
Equality based: (=, !=)
        name: bhupinder
        class: nodes

Set based: (in, not in and exists)
        env in (prod, dev)
        env not in (team1, team2)
```

* Node Selector
One use case for selecting labels is to contrain 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.

```
spec:
  containers:
    - name: c00
      image: ubuntu
      command:
  nodeSelector:
    hardware: t2-medium
```

Scaling and Replication

- K8s was designed to orchestrate multiple containers and replication.

- Need for multiple containers / replication helps us with these

1) Reliability - By having multiple versions of an application you prevent problems if one or more fails.

2) Load balancing - having multiple versions of a container enables you to easily send traffic to different instances to prevent overloading of a single instances or node.

3) Scaling: when load does become too much for the no of existing instance. K8s enables you to easily scale up your application, adding additional instances as needed.

-> Rolling update: updates to a service by replacing pods one by one.

* REPLICATION CONTROLLER

A OBJECT that enables you to create multiple pod. then make sure the no of pods always exists.

To scale up

kubectl scale --replicas=8 rc -l
rs name = bhupinder

Replica Set
Next gen of Replication Controller

RC only support equality based
replica set also support equality based + set based

kind : Replica set
apiversion : apps/v1

kubectl get rs (To find all replica set)

DEPLOYMENT AND ROLLBACK

- Replication Controller and Replicaset is not able to do update and Rollback apps in cluster.

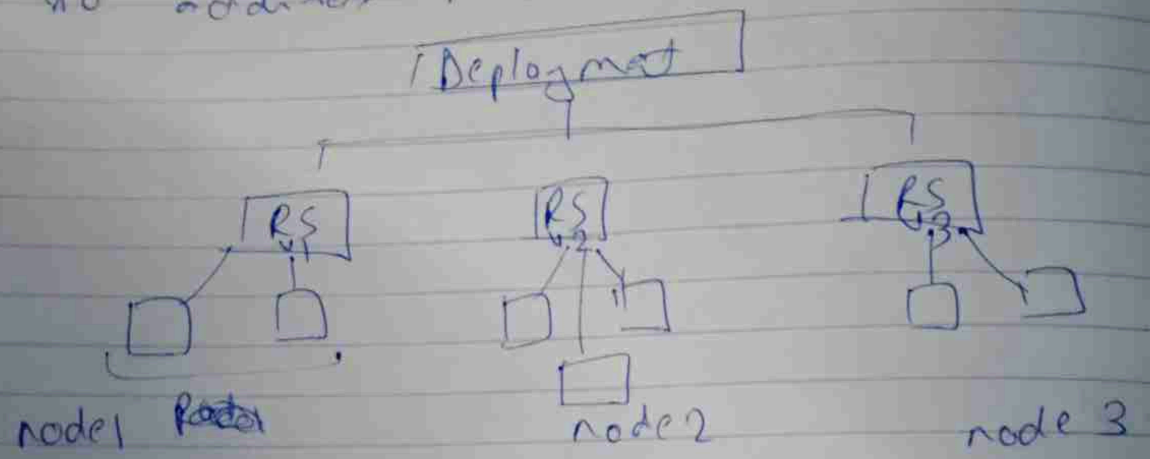
- A deployment object act as a supervisor for pods, giving you fine grained control over how and when a new pod is rolled out, updated or rolled back to previous state.

- When using deployment object, we first define the state of the app, then k8s cluster schedules mentioned app instance ~~ooo~~ onto specific individual nodes.

- A deployment provides declarative updates for pods and replicaset.

- K8s then monitor, if the node hosting an instance goes down or pod is deleted the deployment controller replace it

- This provides a self healing mechanism to address machine failure or maintenance



★ Use Case of deployment

1) Create a deployment to rollout a replicaset

2) Declare the new state of pods by updating the pod template spec of the deployment

3) Rollback to an earlier deployment revisions

4) Scale up the deployment to facilitates more load

5) Clean up older replicasets that you don't need anymore

6) Pause the deployment

TO rollback to specific version

-- to-revision

Kubectl rollout undo
deploy/mydeployments --to-revision=2

Note: That the name of RS is always formatted as
[Deployment-name] - [Randomstring]

Kubectl get deploy (To check deployment)

when you inspect you can find these
NAME
READY
UP TO DATE
AVAILABLE
AGE

type

- kubectl describedeploy my deployment (To check how deployment creates)
- kubectl get rs
- kubectl scale --replicas=1 deploy mydeployment
  (To scale up or down)
- kubectl logs -f <podname> (To check what is running inside container)

- kubectl rollout status deployment mydeployments
- " history "
- " undo "

Reason for failed Deployment

Insufficient Quota, Readiness probe failures
Image pull errors Insufficient permissions
App runtime misconfiguration
Limit Ranges

SERVICES, NODEPORT AND VOLUMES

K8s networking addes four concerns:-

1) containers within a pod use networking to communicate via loopback

2) cluster networking provides communication between different pods.

3) The service resources lets you expose an application running in pods to be reachable from outside your cluster

4) You can also use service to publish services only for consumption inside your cluster.

Note. Container to container communication on same pod happens through localhost within containers

To go into a particular container,

```
kubectl exec testpod -it -c c00 -- /bin/bash
```

To install curl into container,

```
apt update && apt install curl

curl localhost:80
kubectl delete -f pod1.yml
```

Note: Pod to Pod communication on same worker node happens through Pod IP.

By default Pod's IP will not be accessible outside the node

Objects - Services

Service provide Virtual IP.

[illegible]

Virtual IP will map to pod of R.S.

1) When using RC, Pods are terminated and creating during scaling or replication operations

2) Using deployment, while updating the image version the pod are terminated and new pods take the place of other pods

3) service is an logical bridge btw pods and end user

4) Kube proxy is the one which keeps the mapping between the virtual IP and the pods upto date.

5) Labels are used to select which are the pods to be put under a service.

6) Creating a service will create an endpoint to access the pods / application in it

* Services can be exposed in different ways by specifying a type in service specification

1) Cluster IP
2) Node port
3) Load balancer
4) Headless

Service port; 30000 - 32767

Cluster IP
- exposes service only internally within the cluster

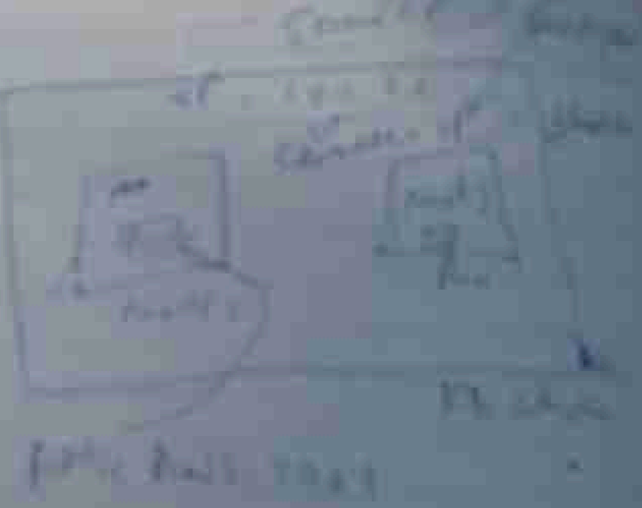
* Mainly used to communicate btw components of microservices

Node port:

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

Nodeport service:
Selected port & node

The default
Nodeport range [illegible]



[illegible] 30007

★ VOLUMES
A volume is essentially a directory backed by a storage medium. The storage medium and its content are determined by the volume type.

A vol is attached to Pod