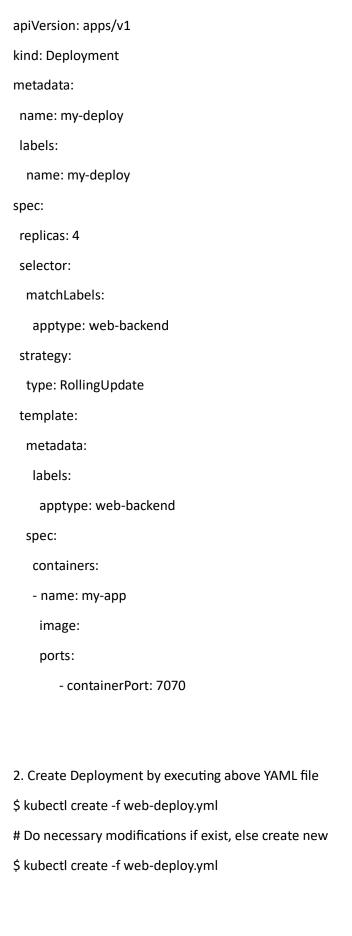
# **Deployment.yml**



- # Completely Modify Pod Template
- \$ kubectl replace -f web-deploy.yml
- 3. View Deployments
- \$ kubectl get deployments
- \$ kubectl get deploy
- \$ kubectl get deploy -o wide
- \$ kubectl get deploy <deployment-name> -o json
- \$ kubectl get deploy <deployment-name> -o yaml
- 4. View Deployment Description
- \$ kubectl describe deploy <deployment-name>
- 5. We can modify generated/updated YAML file
- \$ kubectl edit deploy <deployment-name>
- ## change replicas: count to any other value then (ESC):wq
- # We can modify our YAML file and then execute apply command
- \$ kubectl apply -f web-deploy.yml
- ## We can Even scale using command also
- \$ kubectl scale deploy <deployment-name> --replicas=<desired-replica-count>
- 6. Delete Deployment
- \$ kubectl delete deploy <deployment-name>
- \$ kubectl delete -f web-deploy.yml

## **Kubernetes (K8s)**

Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications. The open source project is hosted by the Cloud Native Computing Foundation (CNCF).

It provides a scalable and resilient framework for automating the deployment, scaling, and management of applications across clusters of servers.

#### A SMALL HISTORY OF K8S:

- In the early 2000s, Google started developing a system called Borg to manage their internal containerized applications.
- Borg enabled Google to run applications at scale, providing features such as automatic scaling, service discovery, and fault tolerance.
- In 2014, Google open-sourced a version of Borg called Kubernetes.
- Kubernetes was donated to the Cloud Native Computing Foundation (CNCF), a neutral home for open-source cloud-native projects, in July 2015.
- Kubernetes 1.8 added significant enhancements for storage, security, and networking. Key features included the stable release of the stateful sets API, expanded support for volume plugins, and improvements in security policies.
- Check URL: https://kubernetes.io/releases/ for more release details.

## Control Plane / Master Node

The control plane's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for example, starting up a new pod when a deployment's replicas field is unsatisfied).

Control plane components can be run on any machine in the cluster. Do not run user containers on this machine.

#### Node Components / Worker Nodes

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

- 1. Master Node: The master node is responsible for managing the cluster and coordinating the overall state of the system. It includes the following components:
- a. API Server: The API server is the central control point for all interactions with the cluster. It exposes the Kubernetes API and handles requests from users and other components.
- b. Scheduler: The scheduler is responsible for assigning workloads (pods) to individual worker nodes based on resource requirements, constraints, and other policies.

- c. Controller Manager: The controller manager runs various controllers that monitor the cluster state and drive it towards the desired state. Examples include the replication controller, node controller, and service controller.
- d. etcd: etcd is a distributed key-value store used by Kubernetes to store cluster state and configuration data.

Pod: The basic building block of Kubernetes. A pod represents a single instance of a running process within the cluster. It can encapsulate one or more containers that share the same network and storage resources.

### Namespace:

Namespace (short name = ns):

namespace is a virtual cluster or logical partition within a cluster that provides a way to organize and isolate resources. It allows multiple teams or projects to share the same physical cluster while maintaining resource separation and access control.

- # To create a namespace:
- \$ kubectl create namespace < namespace name >
- \$ kubectl create ns my-bank
- # To switch to a specific namespace: (make this as default type)
- \$ kubectl config set-context --current --namespace=<namespace-name>
- # To list all namespaces:
- \$ kubectl get namespaces
- # To get resources within a specific namespace:
- \$ kubectl get <resource-type> -n <namespace-name>
- \$ kubectl get deploy -n my-bank
- \$ kubectl get deploy --namespace my-bank
- \$ kubectl get all --namespace my-bank
- # To delete a namespace and all associated resources:

```
$ kubectl delete namespace < namespace - name>
$ kubectl delete ns my-bank
kubectl create ns mydeploy
kubectl apply -f deploy.yml -n mydeploy
Pod.yml
apiVersion: v1
kind: Pod
metadata:
name: my-pod
labels:
   app: web-app
       type: backend
spec:
containers:
  - name: web-app
   image: nithesh04/web-app:latest
   ports:
    - containerPort: 80
Replicates.yml
Yaml_File
kind: ReplicaSet
metadata:
name: my-rs
labels:
  name: my-rs
```

```
spec:
 replicas: 4
 selector:
  matchLabels:
   apptype: web-backend
 template:
  metadata:
   labels:
    apptype: web-backend
  spec:
   containers:
   - name: my-app
    image:
    ports:
     - containerPort: 8080
2. Create ReplicaSet by executing above YAML file
$ kubectl create -f rs-test.yml
# Do necessary modifications if exist, else create new
$ kubectl apply -f rs-test.yml
# Completely Modify Pod Template
$ kubectl replace –f rs-test.yml
3. View ReplicaSets
$ kubectl get replicasets
$ kubectl get rs
$ kubectl get rs -o wide
$ kubectl get rs <replica-set-name> -o json
$ kubectl get rs <replica-set-name> -o yaml
```

- 4. View ReplicaSet Description
- \$ kubectl describe rs <replica-set-name>
- 5. We can modify generated/updated YAML file
- \$ kubectl edit rs <replica-set-name>
- ## change replicas: count to any other value then (ESC):wq
- # We can modify our YAML file and then execute apply command
- \$ kubectl apply -f rs-test.yml
- ## We can Even scale using command also
- \$ kubectl scale replicaset <replicaset-name> --replicas=<desired-replica-count>
- 6. Delete ReplicaSet
- \$ kubectl delete rs <replica-set-name>
- \$ kubectl delete -f rs-test.yml

services.yml

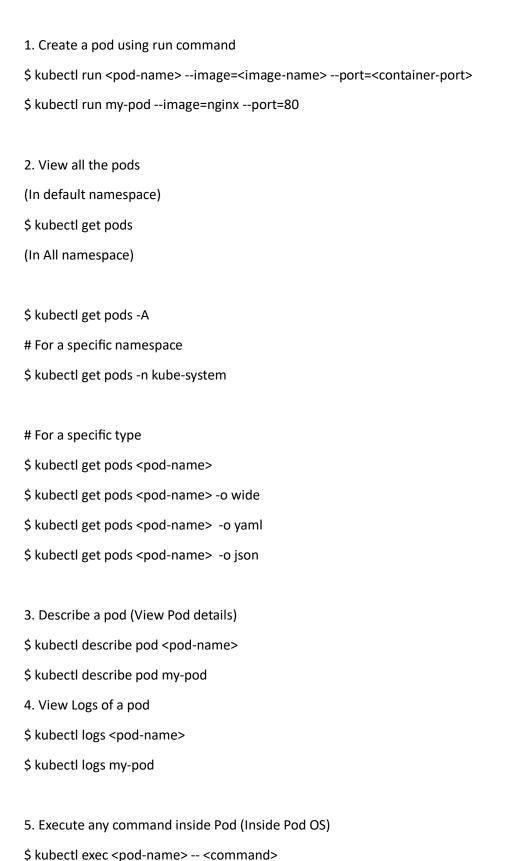
Services (short name = svc):

Service is an abstraction that defines a logical set of pods and a policy to access them. Services enable network connectivity and load balancing to the pods that are part of the service, allowing other components within or outside the cluster to interact with the application.

Service Types: Kubernetes supports different types of services:

- 1. NodePort: Exposes the service on a static port on each selected node's IP. This type makes the service accessible from outside the cluster by the <NodeIP>:<NodePort> combination.
- 2. ClusterIP: Exposes the service on a cluster-internal IP. This type makes the service only reachable within the cluster.
- 3. LoadBalancer: Creates an external load balancer in cloud environments, which routes traffic to the service.

## **Kubernets commands:**



# 6. Pod creation

sudo nano <podname>

kubectl apply -f <podname>

# 7. EXECUTE

\$ kubectl exec -it my-pod -- /bin/sh