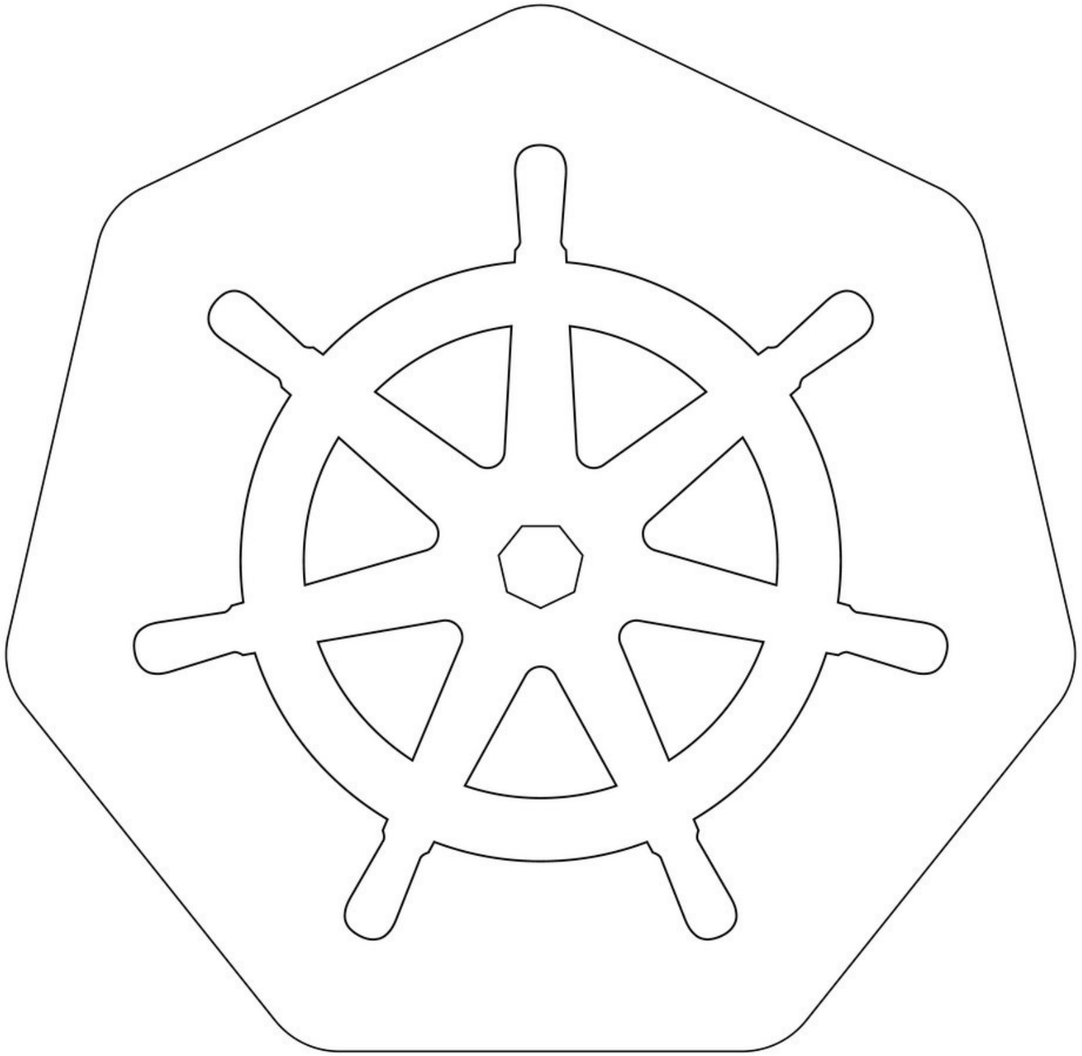
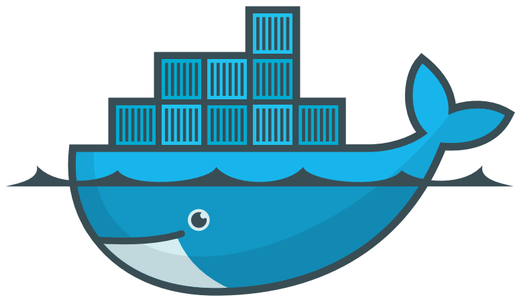


kubernetes

KUBERNETES:

KUBERNETES

K8S



WHY KUBERNETES

Earlier We used Docker Swarm as a container orchestration tool that we used to manage multiple containerized applications on our environments.

FEATURES	DOCKER SWARM	KUBERNETES
Setup	Easy	Complex
Auto Scaling	No Auto Scaling	Auto Scaling
Community	Good Community	Greater community for users like documentation, support and resources
GUI	No GUI	GUI

KUBERNETES:

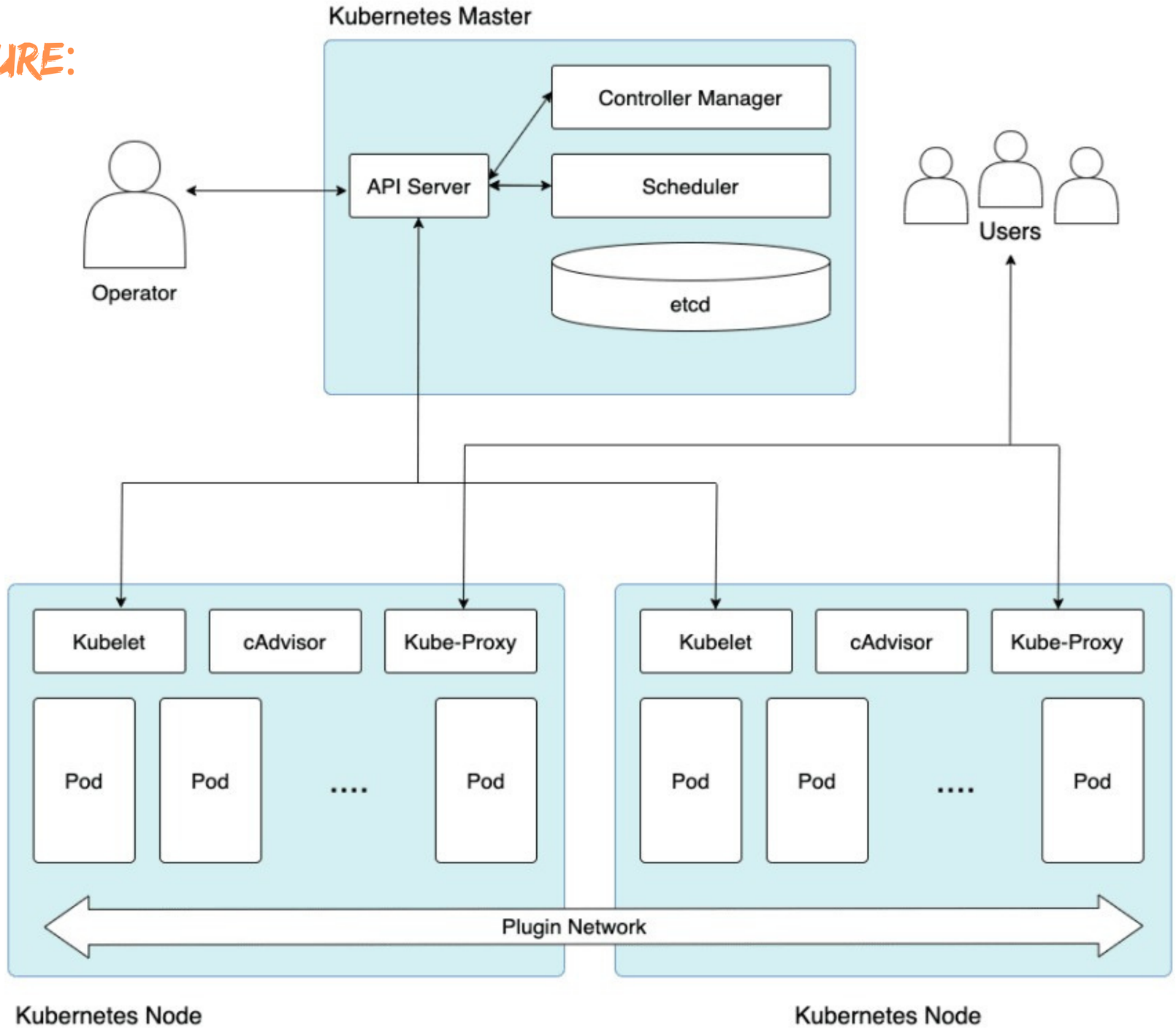
IT is an open-source container orchestration platform.
It is used to automates many of the manual processes like deploying, managing, and scaling containerized applications.
Kubernetes was developed by GOOGLE using GO Language.
Google donated K8's to CNCF in 2014.
1st version was released in 2015.

WHY KUBERNETES:

Containers are a good and easy way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. In docker we used docker swarm for this. but any how docker has drawbacks!

so we moved to KUBERNETES.

AECHITECTURE:

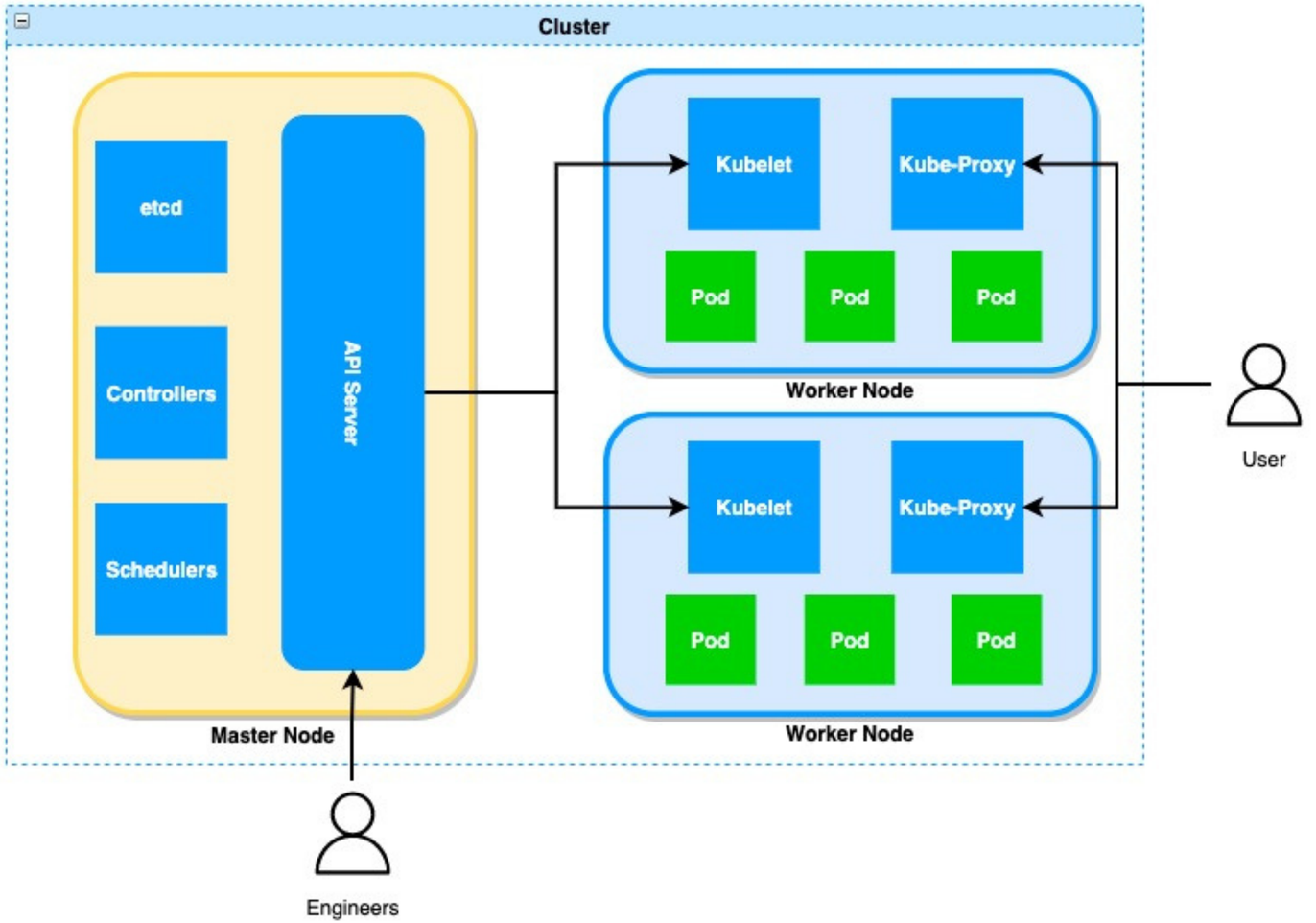


CLUSTER:

- It is a group of serversIt will have both
- manager and worker nodes.
- Master Node is used to assign tasks to Worker Nodes.
- Worker node will perform the task.
- we have 4 components in Master Node

  1.API Server
  2.ETCD
  3. Controllers-manager
  4. Schedulers
- we have 4 components in Worker Node.

  1. Kubelet
  2. Kube-Proxy
  3. Pod
  4. Container



API SERVER:

- It is used to accept the request from the user and store the request in ETCD.

ETCD:

- It is like a database to our k8's
- it is used to store the requests from API Server in the KEY-VALUE format.

SCHEDULER:

- It is used to search pending tasks which are present in ETCD.
- If any pending task found in ETCD, it will schedule in worker node.
- It will decide in which worker node our task should gets executed. It will decide by communication with the kubelet in worker node

CONTROLLERS:

- It is used to perform the operations which is scheduled by the scheduler.
- it wil control the containers creation in worker nodes.

KUBELET:

- Agent ensures that pod is running or not.

KUBE-PROXY

- It is used to maintain a network connection between worker and manager nodes.

POD:

- A group of one or more containers.

CONTAINER:

- It is a virtual machine which does not have any OS.
- it is used to run the applications in worker nodes.

KUBERNETES CLUSTER SETUP:

There are multiple ways to setup kubernetes cluster.

1.SELF MANAGER K8'S CLUSTER
a.mini kube (single node cluster)
b.kubeadm(multi node cluster)
c. KOPS

2. CLOUD MANAGED K8'S CLUSTER
a. AWS EKS
b.AZURE AKS
c.GCP GKS
d.IBM IKE

MINIKUBE:

It is a tool used to setup single node cluster on K8's.
It contains API Servers, ETDC database and container runtime
It helps you to containerized applications.
It is used for development, testing, and experimentation purposes on local.
Here Master and worker runs on same machine
It is a platform Independent.
By default it will create one node only.
Installing Minikube is simple compared to other tools.

NOTE: But we dont implement this in real-time

MINIKUBE SETUP:

REQUIREMENTS:

- 2 CPUs or more
- 2GB of free memory
- 20GB of free disk space
- Internet connection
- Container or virtual machine manager, such as: Docker.

UPDATE SERVER:

1 apt update -y

2 apt upgrade -y

INSTALL DOCKER:

3 sudo apt install curl wget apt-transport-https -y

4 sudo curl -fsSL https://get.docker.com -o get-docker.sh

sudo sh get-docker.sh

INSTALL MINIKUBE:

5 sudo curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

6 sudo mv minikube-linux-amd64 /usr/local/bin/minikube

7 sudo chmod +x /usr/local/bin/minikube

8 sudo minikube version

INSTALL KUBECTL:

9 sudo curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

10 sudo curl -LO "https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"

11 sudo echo "$(cat kubectl.sha256) kubectl" | sha256sum --check

12 sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

13 sudo kubectl version --client

14 sudo kubectl version --client --output=yaml

15 sudo minikube start --driver=docker --force

KUBECTL:

- kubectl is the CLI which is used to interact with a Kubernetes cluster.
- We can create, manage pods, services, deployments, and other resources
- We can also monitoring, troubleshooting, scaling and updating the pods.
- To perform these tasks it communicates with the Kubernetes API server.
- It has many options and commands, to work on.
- The configuration of kubectl is in the $HOME/.kube directory.
- The latest version is 1.27

SYNTAX:

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

kubectl api-resources : to list all api resources

POD:

- It is a smallest unit of deployment in K8's.
- It is a group of containers.
- Pods are ephemeral (short living objects)
- Mostly we can use single container inside a pod but if we required, we can create multiple containers inside a same pod.
- when we create a pod, containers inside pods can share the same network namespace, and can share the same storage volumes .
- While creating pod, we must specify the image, along with any necessary configuration and resource limits.
- K8's cannot communicate with containers, they can communicate with only pods.
- We can create this pod in two ways,
    - 1. Imperative(command)
    - 2. Declarative (Manifest file)

POD CREATION:

IMPERATIVE:

The imperative way uses kubectl command to create pod.
This method is useful for quickly creating and modifying the pods.
SYNTAX: kubectl run pod_name --image=image_name
COMMAND: *kubectl run pod-1 --image=nginx*

kubectl : command line tool run : action
pod-1 : name of pod
nginx : name of image

KUBECTL:

DECLARATIVE:

The Declarativ tive way we need to create a Manifest file in YAML Extension.
This file contains the desired state of a Pod.
It takes care of creating, updating, or deleting the resources.
This manifest file need to follow the yaml indentation.
YAML file consist of KEY-VALUE Pair.
Here we use create or apply command to execute the Manifest file.

SYNTAX: kubectl create/apply -f file_name

CREATE: if you are creating the object for first time we use create only.
APPLY: if we change any thing on files and changes need to apply the resources.

KUBECTL:

MANIFEST FILE:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

apiVersion: For communicating with master node

Kind: it is a type of resource

Metadata: data about pod

Spec: it is a specifications of a container

Name : Name of the Container

Image : Conatiner image

Ports : To Expose the Application

- : It is called as Array

COMMANDS:

- To get all the pods: **kubetctl get pods (or) kubectl get pod (or) kubectl get po**
- To delete a pod: **kubectl delete pod pod_name**
- To get IP of a pod: **kubectl get po pod_name -o wide**
- To get IP of all pods: **kubectl get po -o wide**
- To get all details of a pod: **kubectl describe pod podname**
- To get all details of all pods: **kubectl describe po**
- To get the pod details in YAML format: **kubectl get pod pod-1 -o yaml**
- To get the pod details in JSON format: **kubectl get pod pod-1 -o json**
- To enter into a pod: **kubectl exec -it pod_name -c cont_name bash**
- To get the logging info of our pod: **kubectl logs pod_name**
- To get the logs of containers inside the pod: **kubectl logs pod_name -c cont-name**

WE DEPLOYED THE POD

LETS ACCESS OUR APPLICATION

I CANT ABLE TO ACCESS MY APPLICATION

The reason why we are not able to access the application: In Kubernetes, if you want to access the application we have to expose our pod. To Expose these pods we use Kubernetes services.

KUBERNETES SERVICES

- Service is a method for exposing Pods in your cluster.
- Each Pod gets its own IP address But we need to access from IP of the Node..
- If you want to access pod from inside we use Cluster-IP.
- If the service is of type NodePort or LoadBalancer, it can also be accessed. from outside the cluster.
- It enables the pods to be decoupled from the network topology, which makes it easier to manage and scale applications

TYPES:

- CLUSTER-IP
- NODE PORT
- LOAD BALANCER

TYPES OF SERVICES

- ClusterIP: A ClusterIP service provides a stable IP address and DNS name for pods within a cluster. This type of service is only accessible within the cluster and is not exposed externally.
- NodePort: A NodePort service provides a way to expose a service on a static port on each node in the cluster. This type of service is accessible both within the cluster and externally, using the node's IP address and the NodePort.
- LoadBalancer: A LoadBalancer service provides a way to expose a service externally, using a cloud provider's load balancer. This type of service is typically used when an application needs to handle high traffic loads and requires automatic scaling and load balancing capabilities.
- ExternalName: An ExternalName service provides a way to give a service a DNS name that maps to an external service or endpoint. This type of service is typically used when an application needs to access an external service, such as a database or API, using a stable DNS name.

COMPONENTS OF SERVICES

A service is defined using a Kubernetes manifest file that describes its properties and specifications. Some of the key properties of a service include:

- Selector: A label selector that defines the set of pods that the service will route traffic to.
- Port: The port number on which the service will listen for incoming traffic.
- TargetPort: The port number on which the pods are listening for traffic.
- Type: The type of the service, such as ClusterIP, NodePort, LoadBalancer, or ExternalName.

CLUSTER-IP:

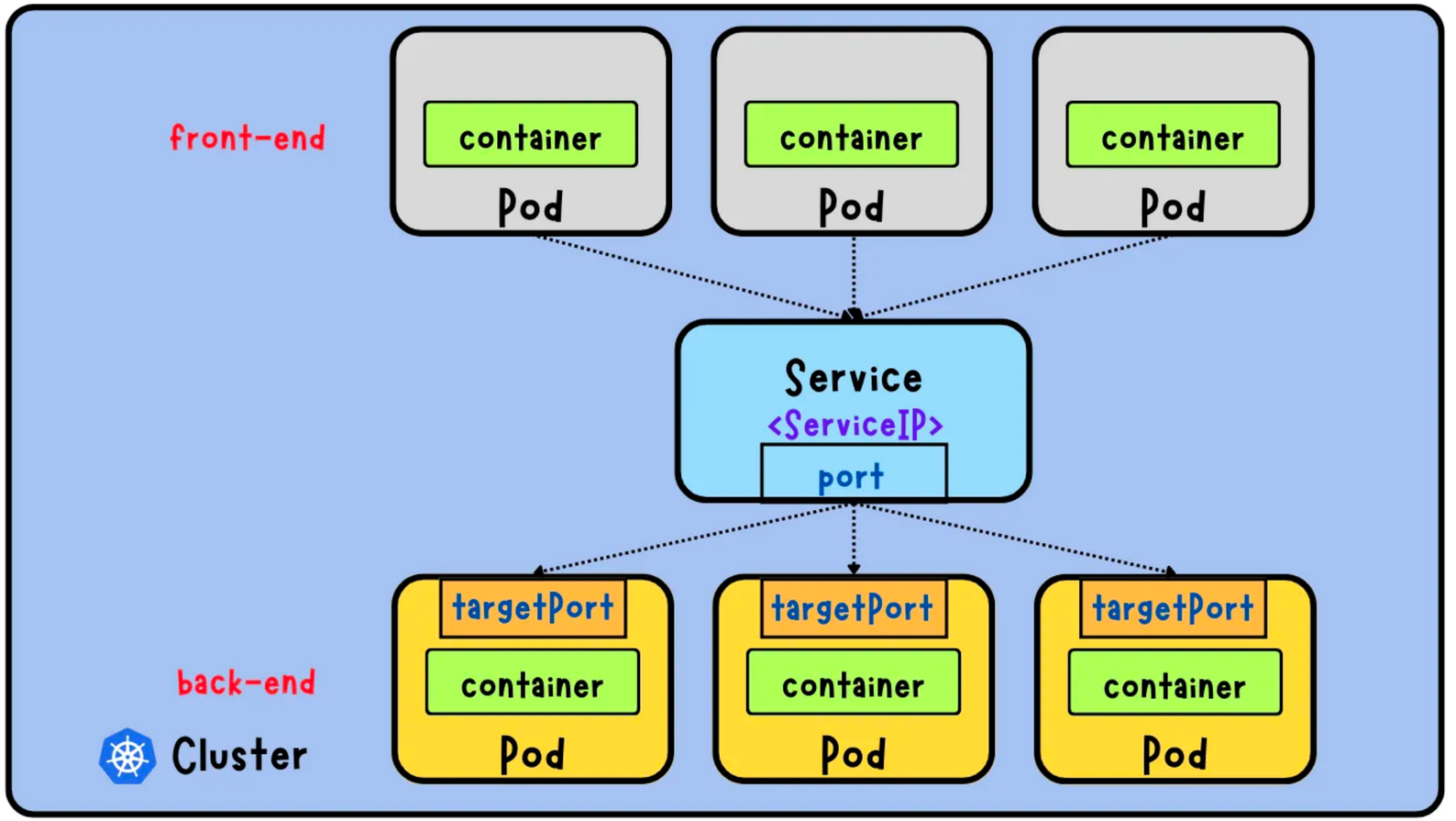
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
  labels:
    app: swiggy
spec:
  containers:
    - name: cont1
      image: rahamshaik/tic-tac-toe-1:latest
      ports:
      - containerPort: 80
```

- To deploy the application we create a container, which stores inside the pod.
- After container is created we will be not able to access the application.
- Because we cannot access the pods and ports from the cluster.
- To Avoid this we are creating the Services.

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: frontend
spec:
  type: ClusterIP
  selector:
    app: swiggy
  ports:
    - port: 80
      targetPort: 80
```

- In this code we are exposing the applcation. here we use clusterip service
- By using this we can access application inside the cluster only.
- But if we want to access the application from outside we need to use nodeport.
- ClusterIP will assign one ip for service to access the pod.
- Just Replace *ClusterIP=NodePort*
- kubectl apply -f filename.yml

Three Types of Services: ClusterIP



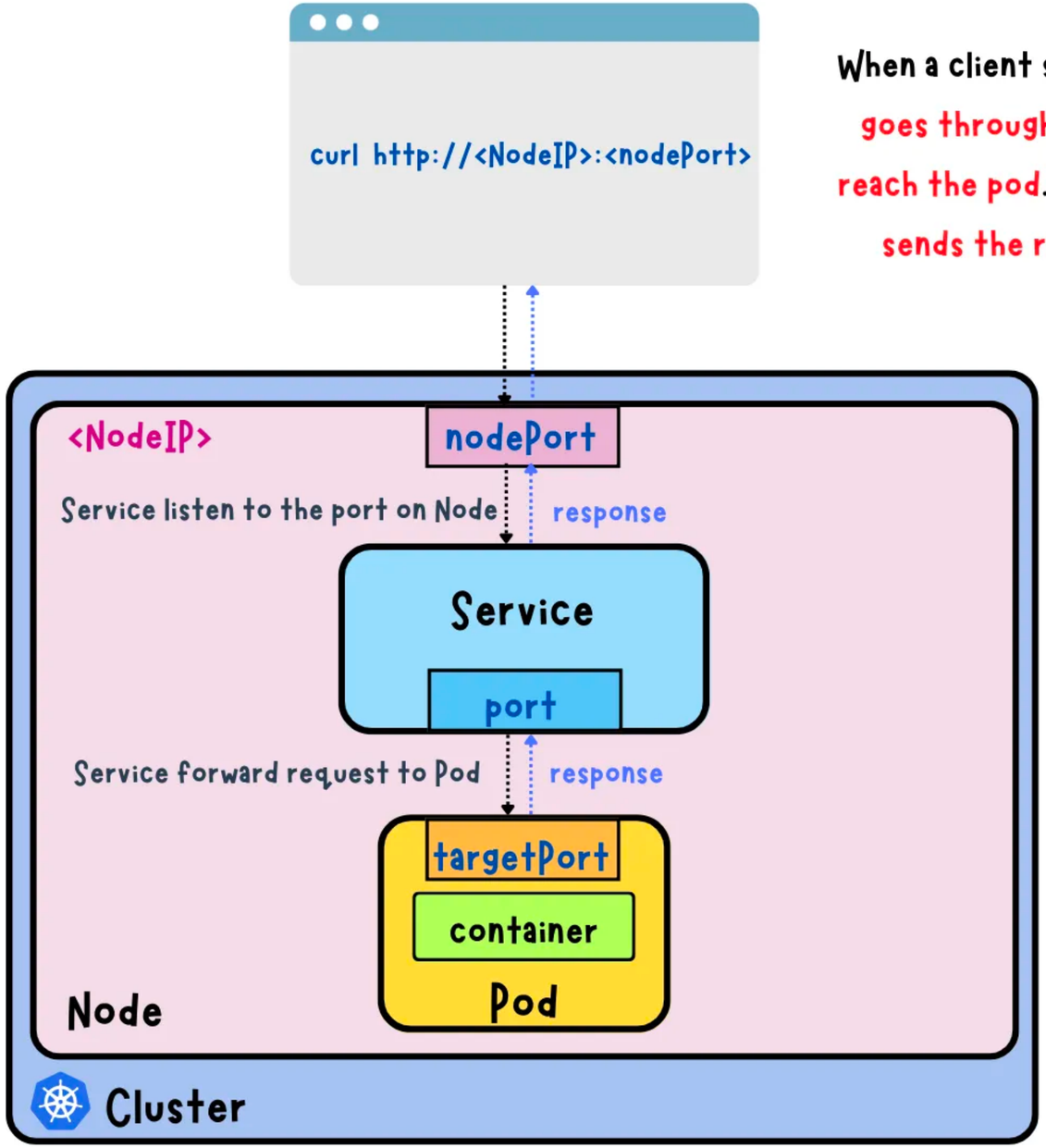
port: port on Service

targetPort: port on Pod (destination)

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: frontend
spec:
  type: NodePort
  selector:
    app: swiggy
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30001
```

- In this code we are exposed the application from anywhere (inside & outside)
- We need to Give public ip of node where pod is running.
- Node Port Range= 30000 - 32767
- here i hae defined port number as 30001
- if we dont specify the port it will assign automatically.
- kubectl apply -f filename.yml.
- NodePort expose service on a static port on each node.
- NodePort services are typically used for smaller applications with a lower traffic volume.
- To avoid this we are using the LoadBalancer service.
- Just Replace *NodePort=LoadBalancer*

Three Types of Services: NodePort



When a client sends a request to a NodePort service, it goes through the NodePort, Port, and TargetPort to reach the pod. The pod then processes the request and sends the response back through the same ports.

nodePort: port on Node

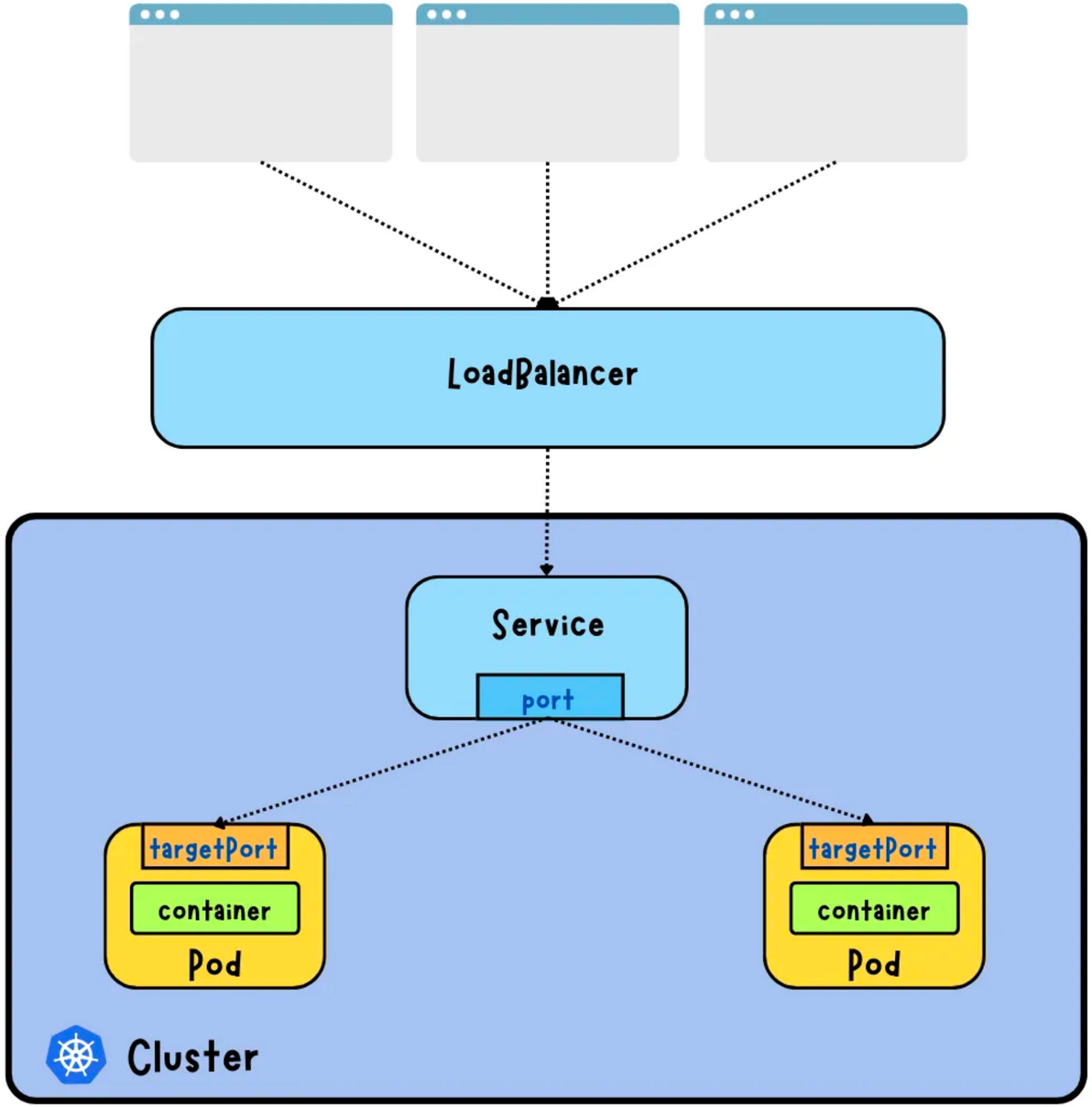
port: port on Service

targetPort: port on Pod (destination)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
spec:
  type: LoadBalancer
  selector:
    app: swiggy
  ports:
    - port: 80
      targetPort: 80
```

- In LoadBalaner we can expose application externally with the help of Cloud Provider LoadBalancer.
- it is used when an application needs to handle high traffic loads and requires automatic scaling and load balancing capabilities.
- After the LoadBalancer service is created, the cloud provider will created the Load Balancer.
- This IP address can be used by clients outside the cluster to access the service.
- The LoadBalancer service also automatically distributes incoming traffic across the pods that match the selector defined in the YAML manifest.
- access : publicip:port and LB url
- http://a0dce056441c04035918de4bfb5bff97-40528368.us- east-1.elb.amazonaws.com/

Three Types of Services: LoadBalancer



THE DRAWBACK

In the above both methods we can be able to create a pod
but what if we delete the pod ?

once you delete the pod we cannot able to access the pod,
so it will create a lot of difficulty in Real time

NOTE: Self healing is not Avaliable here.

To overcome this on we use some Kubernetes components called RC, RS, DEPLOYMENTS, DAEMON SETS, etc ...

COMMANDS FOR SERVICES:

1. Creates a new service based on the YAML file

kubectl create -f filename

2. Create a ClusterIP service

kubectl create service clusterip <service-name> --tcp=<port>:<targetPort>

3. Create a NodePort service

kubectl create service nodeport <service-name> --tcp=<port>:<targetPort>
--node-port=<nodePort>

4. Create a LoadBalancer service

kubectl create service **loadbalancer** <service-name> --tcp=<port>:<targetPort>

5. Create a service for a pod

```
kubectl expose pod <pod-name> --name=<service-name> \
 --port=<port> --target-port=<targetPort> --type=<service-type>
```

6. Create a service for a deployment

```
kubectl expose deployment <deployment-name> --name=<service-name> \
 --port=<port> --target-port=<targetPort> --type=<service-type>
```

7. Retrieve a list of services that have been created

```
kubectl get services
```

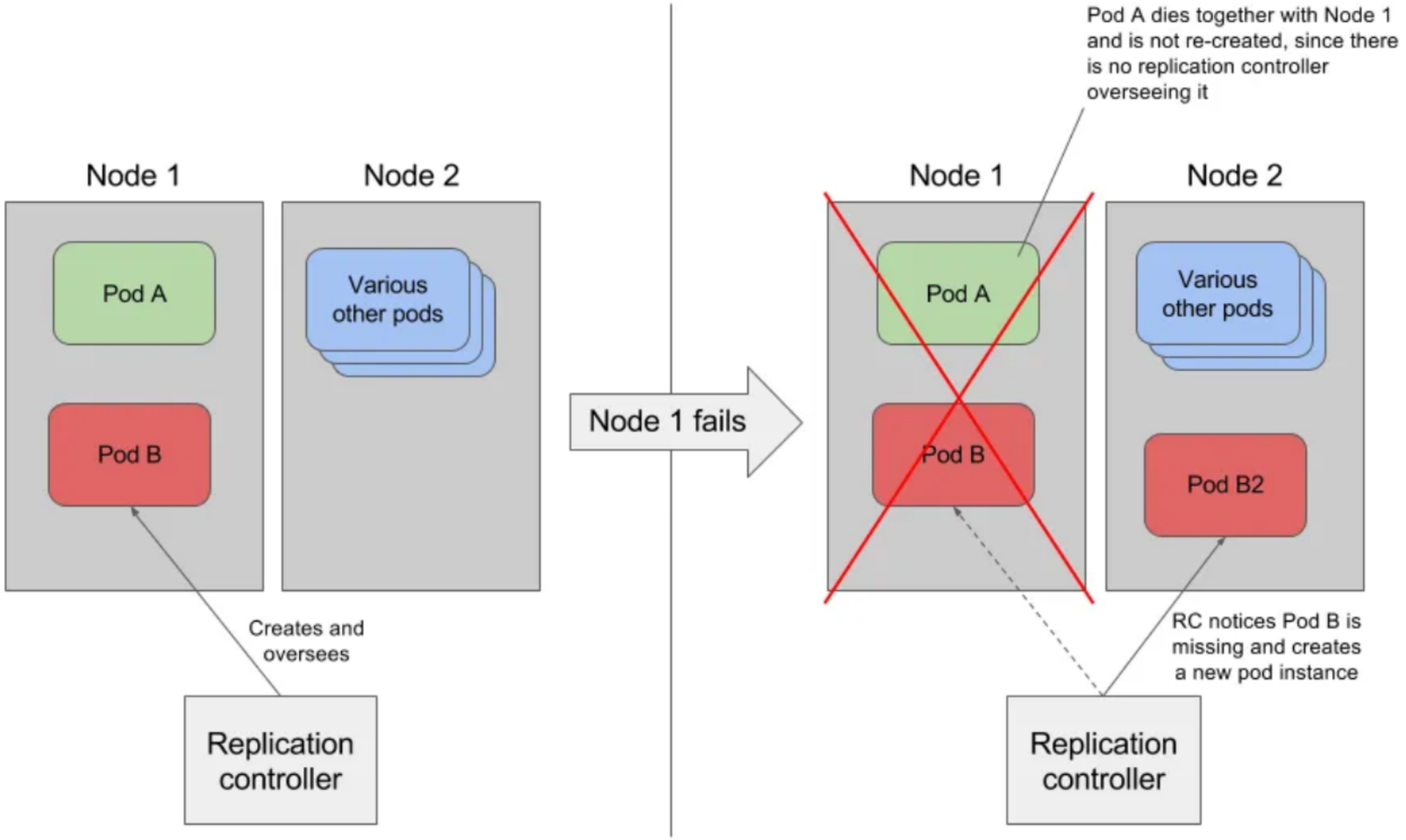
8. Retrieve detailed information about a specific service

```
kubectl describe services <service-name>
```

REPLICATION CONTROLLER:

- Replication controller can run specific number of pods as per our requirement.
- It is the responsible for managing the pod lifecycle
- It will make sure that always pods are up and running.
- If there are too many pods, RC will terminates the extra pods.
- If there are two less RC will create new pods.
- This Replication controller will have slef-healing capability, that means automatically it will creates.
- If a pod is failed, terminated or deleted then new pod will get crated automatically. Replication Controllers use labels to identify the pods that they are managing.
- We need to specifies the desired number of replicas in YAML file.

REPLICATION CONTROLLER:



```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: my-replicationcontroller
spec:
  replicas: 3
  selector:
    app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-container
          image: my-image:latest
          ports:
          - containerPort: 80
```

SELECTOR: It select the resources based on labels. Labels are key value pairs.

This will helps to pass a command to the pods with lable app=nginx

TO EXECUTE : kubectl create -f file_name.yml

TO GET : kubectl get rc

TO DESCRIBE : kubectl describe rc/nginx

TO SCALE UP : kubectl scale rc rc_name --replicas 5

TO SCALE DOWN : kubectl scale rc rc_name --replicas 2 (drops from 5 to 2)

TO DELETE REPLICA CONTROLLER : kubectl delete rc rc_name

IF WE DELETE RC, PODS ARE ALSO GETS DELETED, BUT IF WE DON'T WANT TO DELETE PODS, WE WANT TO DELETE ONLY REPLICA SETS THEN

kubectl delete rc rc_name --cascade=orphan

kubectl get rc rc_name

kubectl get pod

Now we deleted the RC but still pods are present, if we want to assign this pods to another RC use the same selector which we used on the RC last file.

THE DRAWBACK

RC used only equality based selector
ex: env=prod

here we can assign only one value (equality based selector)

We are not using these RC in recent times, because RC is replaced by RS(Replica Set)

REPLICASET:

- it is nothing but group of identical pods. If one pod crashes automatically replica sets gives one more pod immediately.
- it uses labels to identify the pods
- Difference between RC and RS is selector and it offers more advanced functionality.
- The key difference is that the replication controller only supports ***equality-based selectors*** whereas the replica set supports ***set-based selectors***.
- it monitoring the number of replicas of a pod running in a cluster and creating or deleting new pods.
- It also provides better control over the scaling of the pod.
- A ReplicaSet identifies new Pods to acquire by using its selector.
- we can provide multiple values to same key.
- ex: env in (prod,test)

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-container
          image: my-image:latest
          ports:
          - containerPort: 80
```

- Replicas: Number of pod copies we need to create
- Matchelabel: label we want to match with pods
- Template: This is the template of pod.
- Note: give apiVersion: apps/v1 on top

- Labels : mandatory to create RS (if u have 100 pods in a cluster we can inform which pods we need to take care by using labels) if u labeled some pods a raham then all the pods with label raham will be managed.

COMMANDS:

TO EXECUTE	:	kubectl create -f replicaset-nginx.yaml
TO LIST	:	kubectl get replicaset/rs
TO GET INFO	:	kubectl get rs -o wide
TO GET IN YAML	:	kubectl get rs -o yaml
TO GET ALL RESOURCES	:	kubectl get all

Now delete a pod and do list now it will be created automatically

TO DELETE A POD	:	Kubectl delete po pod_name
TO DELETE RS	:	kubectl delete rs
TO SHOW LABLES OF RS	:	Kubectl get po –show-labels
TO SHOW POD IN YAML	:	Kubectl get pod -o yaml

THE DRAWBACK

Here Replica set is an lower level object which focus on maintaining desired number of replica pods.

To manage this replica set we need a higher-level object called deployment which provide additional functionality like rolling updates and roll backs.

Deployments use ReplicaSets under the hood to manage the actual pods that run the application.

DEPLOYMENT:

- It has features of Replicaset and some other extra features like updating and rollbacking to a particular version.
- The best part of Deployment is we can do it without downtime.
- you can update the container image or configuration of the application.
- Deployments also provide features such as versioning, which allows you to track the history of changes to the application.
- It has a pause feature, which allows you to temporarily suspend updates to the application
- Scaling can be done manually or automatically based on metrics such as CPU utilization or requests per second.
- Deployment will create ReplicaSet, ReplicaSet will created Pods.
- If you delete Deployment, it will delete ReplicaSet and then ReplicaSet will delete Pods.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: nginx
  name: nginx-deploy
spec:
  replicas: 2
  selector:
    matchLabels:
      run: nginx
  template:
    metadata:
      labels:
        run: nginx
    spec:
      containers:
      - image: nginx
        name: nginx
```

- Replicas: Number of pod copies we need to create
- Matchelabel: label we want to match with pods
- Template: This is the template of pod.

- Labels : mandatory to create RS (if u have 100 pods in a cluster we can inform which pods we need to take care by using labels) if u labeled some pods a raham then all the pods with label raham will be managed.

MANIFEST FILE:

To create a replica set : kubectl create -f replicaset-nginx.yaml

To see a deployment : kubectl get deployment/deploy

To see full details of deployment : kubectl get deploy -o wide

To view in YAML format : kubectl get deploy -o yaml

To get all info : kubectl describe deploy

COMMAND TO CREATE DEPLOYMENT:

kubectl create deployment deployment-name --image=image-name --replicas=4

Now delete a pod and do list now it will be created automatically

To delete a pod: Kubectl delete po pod_name

To check logs : kubectl logs pod_name

To delete a deployment : kubectl delete deploy deploy_name

When you inspect the Deployments in your cluster, the following fields are displayed:

- ***NAME*** lists the names of the Deployments in the namespace.
- ***READY*** displays how many replicas of the application are available to your users. It follows the pattern ready/desired.
- ***UP***-TO-DATE displays 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*** displays the amount of time that the application has been running.

UPDATING A DEPLOYMENT:

update the container image:

kubectl set image deployment/deployment-name my-container=image-name

or

kubectl edit deployment/deployment-name

To roll back:

kubectl rollout status deployment/deployment-name

To check the status:

kubectl rollout status deployment/deployment-name

To get the history:

kubectl rollout history deployment/deployment-name

To rollback specific version:

kubectl rollout undo deployment/deployment-name --to-revision=number

FINAL PROJECT:

- CREATE A CLUSTER IN KOPS WITH 1 MASTER & 2 WORKERS
- CREATE A NAME SPACE
- CREATE A DEPLOYMENT WITH IN THE NAMESPACE
- INCREASE THE REPLICAS
- EXPOSE IT USING NODE PORT
- ACCESS THE APPLICATION