K8's Common Commands Used:

- To Create Resources From a File

  kubectl create -f <filename>.yml

- To Create Resources From Multiple Files

  kubectl create -f <file1>.yml -f <file2>.yml

- To Start a Single Instance of Nginx

  kubectl run nginx --image=nginx

- To List All the Services

  kubectl get services

- To List Pods in All NameSpaces

  kubectl get pods -A

- To List Pods in the namespace with more info

  kubectl get pods -o wide

- To List a Particular Deployment

  kubectl get deployment <deployment-name>

- To Describe Nodes

  kubectl describe nodes <node-name>

- To describe Pods

  kubectl describe pods

- To Delete a pod using filename.

  kubectl delete -f <filename>.yml

- To Delete pods and services with same names "foo" and "foo1"

  kubectl delete pod,service foo foo1

- To Show the logs for pods

  kubectl logs <Pod_name>

- Run command directly on a Pod

  kubectl exec <Pod_name> -- ls /

- Show master and services addresses.

  kubectl cluster-info

We can deploy our application in two different ways into Kubernetes using different resources :

1. Deployment
2. StatefulSets

Understanding Deployment :

A Kubernetes Deployment provides means for managing a set of pods. These could be one or more running containers or a group of duplicate pods, known as *ReplicaSets*. *Deployment* allows us to easily keep a group of identical pods running with a common configuration.

First, we define our Kubernetes *Deployment* and then deploy it. Kubernetes will then work to make sure all pods managed by the deployment meet whatever requirements we have set. ***Deployment* is a supervisor for pods. It gives us fine-grained control over how and when a new pod version is rolled out. It also provides control when we have to rollback to a previous version.**

In Kubernetes *Deployment* with a replica of 1, the controller will verify whether the current state is equal to the desired state of *ReplicaSet,* i.e., 1. If the current state is 0, it will create a *ReplicaSet.* The ReplicaSet will further create the pods. When we create a Kubernetes *Deployment* with the name *web-app*, it will create a *ReplicaSet* with the name *web-app-<replica-set-id>.* This replica will further create a pod with name *web-app-<replica-set->-<pod-id>*.

Kubernetes *Deployment* is usually used for stateless applications. **However, we can save the state of *Deployment* by attaching a Persistent Volume to it and make it stateful.** The deployed pods will share the same Volume, and the data will be the same across all of them.

How to do Deployment in Kubernetes ?

Whenever we develop some application we talk about putting it in containers and deploying to the Kubernetes cluster.
Let's start with simple application, put it in container and deploy it into Kubernetes cluster :

- Develop a simple static web application page.
- Deploy a simple Docker container for webserver
- Put a custom file for a static webpage
- Create custom container image
- Push it to Docker Hub

- Using the same image, deploy it on your Kubernetes cluster.

Implementation Steps :

We can choose either nginx or apache as webserver. Let's create a simple docker container for the same.

docker run -d -p 80:80 - - name webserver nginx

Let's check if our nginx is loading or not using http://0.0.0.0:80

You can also use default IP of container.

Now, create a simple static html page inside the container.

Login to the container, using docker exec command.

Copy the html file into /usr/share/nginx/html/index.html

We can also create a dockerfile and build the image out of it.

```
FROM nginx
COPY index.html /usr/share/nginx/html/index.html
```

Build an image:

docker build -t my-nginx .

Run the container:

docker run -d -p 80:80 - -name my-nginx-run

Verify if your html page is loading or not.

http:<container-ip>:80

To push the image to Docker Hub, login to docker using:

docker login

Push the image using command:

docker push <dockerhub-username>/<image-name>

We are good with creating the container and pushing the image to DockerHub.

Now, let’s create a Kubernetes Deployment file for deploying the image to Kubernetes cluster.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: new-nginx-deployment
spec:
  selector:
    matchLabels:
      app: new-nginx1
    replicas: 2
    template:
      metadata:
       labels:
        app: new-nginx1
```

```
spec:
  containers:
    - name: <image-name>
      image: <dockerhub-username>/<image-name>
      ports:
        - containerPort: 80
```

Create the deployment:

```
kubectl create -f new-nginx1.yml
kubectl get  deployments
```

This will deploy the image to Kubernetes container and run the pod with maximum of 2 replicas.

Hence, this is how we can deploy our web application into Kubernetes cluster.

Understanding StatefulSets:

In Kubernetes *Deployment*, we treat our pods like cattle, not like pets. If one of the cattle members gets sick or dies, we can easily replace it by purchasing a new head. Such an action is not noticeable. Similarly, if one pod goes down in deployment, it brings up another one. **In *StatefulSet*s, pods are given names and are treated like pets.** If one of your pets got sick, it's immediately noticeable. The same is in the case of *StatefulSet*s*,* as it interacts with pods by their name.

*StatefulSet*s provides to each pod in it two stable unique identities. First, the **Network Identity enables us to assign the same DNS name to the pod regardless of the number of restarts**. The IP addresses might still be

different, so consumers should depend on the DNS name (or watch for changes and update the internal cache).

Secondly, **the Storage Identity remains the same**. The Network Identity always receives the same instance of Storage, regardless of which node it's rescheduled on.

StatefulSet is also a Controller, but unlike Kubernetes *Deployment*, it doesn't create *ReplicaSet* rather, it creates the *pod* with a unique naming convention. Each *pod* receives DNS name according to the pattern: <*statefulset name*>-<*ordinal index*>*.* For example, for *StatefulSet* with the name *mysql,* it will be *mysql-0*.

Every replica of a stateful set will have its own state, and each of the pods will be creating its own PVC(Persistent Volume Claim). So a *StatefulSet* with 3 replicas will create 3 pods, each having its own Volume, so total 3 PVCs. As *StatefulSet*s works with data, we should be careful while stopping pod instances by allowing the required time to persist data from memory to disk. There still might be valid reasons to perform force deletion, for example, when Kubernetes Node fails.

Usage:

*StatefulSet*s enable us to deploy stateful applications and clustered applications. They save data to persistent storage, such as Compute Engine persistent disks. They are suitable for deploying Kafka, MySQL, Redis, ZooKeeper, and other applications (needing unique, persistent identities and stable hostnames).

For instance, a Solr database cluster is managed by several Zookeeper instances. For such an application to function correctly, each Solr instance must be aware of the Zookeeper instances that are controlling it. Similarly, the Zookeeper instances themselves establish connections between each other to elect a master node. Due to such a design, Solr clusters are an example of

stateful applications. Other examples of stateful applications include MySQL clusters, Redis, Kafka, MongoDB, and others. In such scenarios, *StatefulSet*s are used.

Deployment vs StatefulSets

Deployment	StatefulSet
Deployment is used to deploy stateless applications	*StatefulSet*s is used to deploy stateful applications
Pods are interchangeable	Pods are not interchangeable. Each pod has a persistent identifier that it maintains across any rescheduling
Pod names are unique	Pod names are in sequential order
Service is required to interact with pods in a deployment	A Headless Service is responsible for the network identity of the pods
The specified PersistentVolumeClaim is shared by all pod replicas. In other words, shared volume	The specified volumeClaimTemplates so that each replica pod gets a unique PersistentVolumeClaim associated with it. In other words, no shared volume

A Kubernetes *Deployment* is a resource object in Kubernetes that provides declarative updates to applications. A deployment allows us to describe an

application's life cycle. Such as, which images to use for the app, the number of pods there should be, and how they should be updated.

A *StatefulSet*s are more suited for stateful apps. A stateful application requires pods with a unique identity (for instance, hostname). A pod will be able to reach other pods with well-defined names. It needs a Headless Service to connect with the pods. A Headless Service does not have an IP address. Internally, it creates the necessary endpoints to expose pods with DNS names.

The *StatefulSet* definition includes a reference to the Headless Service, but we have to create it separately. *StatefulSet* needs persistent storage so that the hosted application saves its state and data across restarts. Once the *StatefulSet* and the Headless Service are created, a pod can access another one by name prefixed with the service name.

Practical Scenario :

This practical scenario demonstrates how a StatefulSet differs from a Deployment:

Consider a web app that uses a relational database to store data. When traffic to the application increases, administrators intend to scale up the number of pods to support the workload. A straightforward approach is simply to change the replica count within the Deployment's configuration manifest; then the Deployment controller will take care of scaling. Since new pod replicas are assigned the same set of ConfigMaps and environment variables when starting, they communicate with the backend the same way as the original pod, retaining the user experience for incoming traffic.

Similar to the web servers, the relational database may also need to be scaled up to meet the increased workload. Since the master and replica pods need to implement a leader-follower pattern, the pods of the database cannot be

created or deleted randomly. In addition, while each pod needs to sync its data with the previous pod, it retains its own copy of the data stored. In such an instance, a StatefulSet helps create the database pods in an ordered sequence where every new pod acquires its copy of data from the last pod generated. If a pod fails, the StatefulSet controller automatically deploys new pod replicas incrementally with the same identity and attaches them to the same PVC.