



## **Kubernetes Deployment :**

A Kubernetes deployment is a resource object in Kubernetes that provides declarative updates to applications. A deployment allows you to describe an application's life cycle, such as which images to use for the app, the number of pods there should be, and the way in which they should be updated.

A Kubernetes object is a way to tell the Kubernetes system how you want your cluster's workload to look. After an object has been created, the cluster works to ensure that the object exists, maintaining the desired state of your Kubernetes cluster.

The process of manually updating containerized applications can be time consuming and tedious. Upgrading a service to the next version requires starting the new version of the pod, stopping the old version of a pod, waiting and verifying that the new version has launched successfully, and sometimes rolling it all back to a previous version in the case of failure.


Performing these steps manually can lead to human errors, and scripting properly can require a significant amount of effort, both of which can turn the release process into a bottleneck.

A Kubernetes deployment makes this process automated and repeatable. Deployments are entirely managed by the Kubernetes backend, and the whole update process is performed on the server side without client interaction.

A deployment ensures the desired number of pods are running and available at all times. The update process is also wholly recorded, and versioned with options to pause, continue, and roll back to previous versions.

### **Automate deployments with pre-made, repeatable Kubernetes patterns**

#### **The Kubernetes deployment object lets you:**

- Deploy a replica set or pod
  - Update pods and replica sets
  - Rollback to previous deployment versions
  - Scale a deployment
  - Pause or continue a deployment
- 



## **Application management strategies using Kubernetes deployment**

Managing your applications with a Kubernetes deployment includes the way in which an application should be updated. A major benefit of a deployment is the ability to start and stop a set of pods predictably.

### **Rolling update strategy**

A rolling update strategy provides a controlled, phased replacement of the application's pods, ensuring that there are always a minimum number available.

The deployment makes sure that, by default, a maximum of only 25% of pods are unavailable at any time, and it also won't over provision more than 25% of the number of pods specified in the desired state.

The deployment won't kill old pods until there are enough new pods available to maintain the availability threshold, and it won't create new pods until enough old pods are removed.

The deployment object allows you to control the range of available and excess pods through maxSurge and maxUnavailable fields.


With a rolling update strategy there is no downtime during the update process, however the application must be architected to ensure that it can tolerate the pod destroy and create operations.


During the update process 2 versions of the container are running at the same time, which may cause issues for the service consumers.

### **Recreate strategy**

A recreate strategy removes all existing pods before new ones are created. Kubernetes first terminates all containers from the current version and then starts all new containers simultaneously when the old containers are gone.

With a recreate deployment strategy there is some downtime while all containers with old versions are stopped and no new containers are ready to handle incoming requests.





However, there won't be 2 versions of the containers running at the same time, which may make it simpler for service consumers.

## **Declarative deployment pattern for Kubernetes**

Deployments are created by writing a manifest. The manifest is then applied to the Kubernetes cluster using `kubectl apply`, or you can use a declarative deployment pattern. Configuration files for Kubernetes can be written using YAML or JSON.

When creating a deployment, you'll describe the desired state and Kubernetes will implement it using either a rolling or recreate deployment strategy.


Using a declarative deployment pattern allows you to use a Kubernetes deployment to automate the execution of upgrade and rollback processes for a group of pods. Kubernetes patterns are reusable design patterns for container-based applications and services.

You can update a deployment by making changes to the pod template specification. When a change is made to the specification field, it triggers an update rollout automatically.

The rollout lifecycle consists of progressing, complete, and failed states. A deployment is progressing while it is performing update tasks, such as updating or scaling pods.

Complete indicates that all tasks were completed successfully and the system is in the desired state. A failed state is the result of some error that keeps the deployment from completing its tasks.

You can check or monitor the state of a deployment using the `kubectl rollout status` command.





## Manifest for creating Deployment :

```
#vim nginx-deploy.yml
```

```
apiVersion : apps/v1
```

```
kind : Deployment
```

```
metadata :
```

```
  name : nginx-deploy
```

```
  labels :
```

```
    app : nginx-app
```

```
spec :
```

```
  replicas : 3
```

```
  template :
```

```
    metadata :
```

```
      labels :
```

```
        app : nginx-app
```

```
    spec :
```

```
      containers :
```

```
        - name : nginx-container
```

```
          image : nginx:1.7.9
```

```
          ports :
```

```
            - containerPort : 80
```

```
  selector :
```

```
    matchLabels :
```

```
      app : nginx-app
```

```
:wq!
```

```
#kubectl create -f nginx-deploy.yml
```

```
#kubectl get deploy -l app=nginx-app
```

```
#kubectl get rs -l app=nginx-app
```

```
#kubectl get pods -l app=nginx-app
```

```
#kubectl describe deploy nginx-deploy
```

```
#kubectl set image deploy nginx-deploy nginx-container=nginx:1.9.1 --record
```


```
#kubectl rollout history deployment nginx-deploy
```

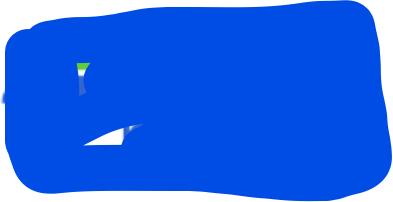
```
#kubectl rollout undo deployment nginx-deploy
```

```
#kubectl rollout status deployment nginx-deploy
```

```
#kubectl set image deploy nginx-deploy nginx-container=nginx:latest
```

```
#kubectl rollout status deployment nginx-deploy
```





#kubectl get deploy

### **Scaleup :**

```
#kubectl scale deployment nginx-deploy --replicas=5  
#kubectl get deploy  
#kubectl get pods -o wide
```

### **Scaledown :**

```
#kubectl scale deployment nginx-deploy --replicas=2  
#kubectl get deploy  
#kubectl get pods -o wide
```

### **Cleanup :**

```
#kubectl delete -f nginx-deploy.yml  
#kubectl get deploy  
#kubectl get rs  
#kubectl get pods
```

