**Deployments**:

* A deployment is kubernetes object that manages a set of identical pods ,ensuring specific number of pods , ensuring that a specified number of pod are running at a any given number of time
* It provides a declarative approach to managing Kubernetes objects, allowing for automated rollouts and rollbacks of containerized applications.
* Deployment manages the deployment of a new version of an application.
* It also helps us roll back to a previous version by creating a replica set and updating it with the new configuration.

**Use Case :** The following are typical use cases for Deployments:

1. Create a Deployment to rollout a ReplicaSet. The ReplicaSet creates Pods in the background. Check the status of the rollout to see if it succeeds or not.
2. Declare the new state of the Pods by updating the PodTemplateSpec of the Deployment. A new ReplicaSet is created and the Deployment manages moving the Pods from the old ReplicaSet to the new one at a controlled rate. Each new ReplicaSet updates the revision of the Deployment.
3. Rollback to an earlier Deployment revision if the current state of the Deployment is not stable. Each rollback updates the revision of the Deployment.
4. Scale up the Deployment to facilitate more load.
5. Pause the rollout of a Deployment to apply multiple fixes to its PodTemplateSpec and then resume it to start a new rollout.
6. Use the status of the Deployment as an indicator that a rollout has stuck.
7. Clean up older ReplicaSets that you don't need anymore

**Notes:**

* If there are problem in deployment, k8s will automatically rollback to the previous version, however you can explicitly rollback to a specific version, as in our case to revision 1 (the original pod version)
* You can rollback to a specific version by "--to-revision" option i.e. Kubectl rollout undo deploy/deployment\_name --to-revision=2 2.
* The name of replicasets is always formatted as [deployment-name]-[random string]
* Command: kubectl get deploy

When you inspect the deployment in your cluster the flowing field display

NAME: list the name of the deployment in the namespace

READY: ready/desired i.e. 2/2

UP-TO-DATE: display the no of replica that have been updated to achieve the desired state

AVAILABLE: Display how many replicas of the application are available to your users

AGE: Display the amount of time that app has been Running

**Commands:**

* Create the Deployment by running the following command: ***kubectl apply -f*** <https://k8s.io/examples/controllers/nginx-deployment.yaml>
* Run ***kubectl get deployments*** to check if the Deployment was created
* To see the Deployment rollout status, ***run kubectl rollout status deployment/ubuntu-deployment***
* To see the ReplicaSet (rs) created by the Deployment, ***run kubectl get r***

**Failed Deployment:** Your Deployment may get stuck trying to deploy its newest ReplicaSet without ever completing. This can occur due to some of the following factors:

• Insufficient quota

• Readiness probe failures

• Image pull errors

• Insufficient permissions

• Limit ranges

• Application runtime misconfiguration

**Updating a Deployment Follow the steps given below to update your Deployment:**

1. Let's update the nginx Pods to use the nginx:1.16.1 image instead of the nginx:1.14.2 image

***kubectl set image deployment.v1.apps/ubuntu-deployment ubuntu=ubuntu:16.04***

Also change the echo statement in command

1. See the rollout status, run:

***kubectl rollout status deployment/ubuntu-deployment***

1. Run ***kubectl get rs*** to see that the Deployment updated the Pods by creating a new ReplicaSet and scaling it up to 3 replicas, as well as scaling down the old ReplicaSet to 0 replicas

**Checking Rollout History of a Deployment Follow the steps given below to check the rollout history:**

1. First, check the revisions of this Deployment:

***kubectl rollout history deployment/ubuntu-deployment***

1. To see the details of each revision, run:

***kubectl rollout history deployment/ubuntu-deployment --revision=2***

**Rolling Back to a Previous Revision**

Follow the steps given below to rollback the Deployment from the current version to the previous version, which is version 2.

1. Now you've decided to undo the current rollout and rollback to the previous revision:

***kubectl rollout undo deployment/ubuntu-deployment***

Alternatively, you can rollback to a specific revision by specifying it with --to-revision:

***kubectl rollout undo deployment/ubuntu-deployment --to-revision=2***

1. Check if the rollback was successful and the Deployment is running as expected, run:

***kubectl get deployment ubuntu-deployment***

1. Get the description of the Deployment:

***kubectl describe deployment ubuntu-deployment***

**Scaling a Deployment**

1. You can scale a Deployment by using the following command:

***kubectl scale deployment/ubuntu-deployment --replicas=10***

1. Assuming horizontal Pod autoscaling is enabled in your cluster, you can setup an autoscaler for your Deployment and choose the minimum and maximum number of Pods you want to run based on the CPU utilization of your existing Pods.

***kubectl autoscale deployment/ubuntu-deployment --min=10 --max=15 --cpu-percent=80***

1. **Proportional scaling**

RollingUpdate Deployments support running multiple versions of an application at the same time. When you or an autoscaler scales a RollingUpdate Deployment that is in the middle of a rollout (either in progress or paused), the Deployment controller balances the additional replicas in the existing active ReplicaSets (ReplicaSets with Pods) in order to mitigate risk. This is called proportional scaling.

**Pausing and Resuming a rollout of a Deployment**

When you update a Deployment, or plan to, you can pause rollouts for that Deployment before you trigger one or more updates. When you're ready to apply those changes, you resume rollouts for the Deployment. This approach allows you to apply multiple fixes in between pausing and resuming without triggering unnecessary rollouts .

1. Get the Deployment details:

***kubectl get deploy***

1. Pause by running the following command:

***kubectl rollout pause deployment/ubuntu-deployment***

1. Then update the image of the Deployment:

***kubectl set image deployment.v1.apps/ubuntu-deployment ubuntu=ubuntu:16.04***

1. Notice that no new rollout started:

***kubectl rollout history deployment/ubuntu-deployment***

1. Get the rollout status to verify that the existing ReplicaSet has not changed:

***kubectl get rs***

1. You can make as many updates as you wish, for example, update the resources that will be used:

***kubectl set resources deployment/ubuntu-deployment -c=ubuntu--limits=cpu=200m,memory=512Mi***

1. Eventually, resume the Deployment rollout and observe a new ReplicaSet coming up with all the new updates:

***kubectl rollout resume deployment/ubuntu-deployment***

1. Watch the status of the rollout until it's done.

***kubectl get rs -w***

**Example:**

apiVersion: apps/v1

kind: Deployment

metadata:

name: ubuntu-deployment

labels:

app: ubuntu

spec:

replicas: 3

selector:

matchLabels:

app: ubuntu

template:

metadata:

labels:

app: ubuntu

spec:

containers:

- name: ubuntu

image: ubuntu:20.04

command: ["/bin/bash", "-c", "while true; do echo 'hello sagar'; sleep 5; done"]