**Kubernetes Deployment**

Deployment controller provides declarative updates for pods.

* Create our deployment to rollout replica sets, the replication creates pods in the background, check the status of rollout if it succeeds or not.
* Declares the state of the pods by updating the pod template spec of their deployment, a new replica that was created and the deployment managers moving the pods from the old replica sets to the new one at a controlled rate (all are pods are not getting updated in a single shot). Each new replica sets updates the revision of the deployment.
* Roll back to an earlier deployment version if the current state of the deployment is not stable. Each rollback updates the revision of that deployment.
* Scale up the deployment to facilitate to more load.
* Pause the deployment, this is to apply multiple fixes to its pod templates, and then resume it to start a new rollout.
* Use the status of that deployment as an indicator that the rollout has stuck.
* Clean up old replica sets. If they don't need any more

**Perform Rolling Update**

A rolling update is the process of updating an application — whether it is a new version or just updated configuration

To update a service without an outage, kubectl supports what is called [rolling update](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands/#rolling-update), which updates one pod at a time, rather than taking down the entire service at the same time.

Once the rollout is complete, the old controller is deleted

Assume that we have a current replication controller named foo and it is running image image:v1

**Syntax: kubectl rolling-update NAME NEW\_NAME --image=IMAGE:TAG**

**kubectl rolling-update foo [foo-v2] --image=myimage:v2**

**Recovery**

If a rollout fails or is terminated in the middle, it is important that the user be able to resume the roll out

Recovery is achieved by issuing the same command again:

**kubectl rolling-update foo [foo-v2] --image=myimage:v2**

**Aborting a rollout**

Abort is assumed to want to reverse a rollout in progress.

**kubectl rolling-update foo [foo-v2] --rollback**

**mkdir && cd /root/ngninx1.9-lab**

**vi index.htm >>> hi, this is nginx 1.9.1**

**Lab 1: vi Dockerfile**

FROM nginx:1.9.1

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

**docker build -t pkw0301/nginx:1.9.1 .**

s

Please put different content for both the index.html

**Mkdir && cd /root/ngninx1.7-lab**

**vi index.htm >>> hi, this is nginx 1.7**

**vi Dockerfile**

FROM nginx:1.7.9

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

**docker build -t pkw0301/nginx:1.7.9 .**

**docker images**

**vi replication-nginx-1.7.9.yaml**

**We can create k8s manifest file :-**

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-nginx

labels:

app: ngnix

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: pkw0301/nginx:1.7.9

ports:

- containerPort: 80

kubectl apply -f replication-nginx-1.7.9.yaml

kubectl get ReplicationController

or

kubectl get rc

kubectl describe ReplicationController <name of your ReplicationController >

kubectl expose rc <rc name> --port=80 –target-port=80 –type=NodePort

kubectl get svc

<VM IP>:<NodePort>

To update version from1.7.9 to 1.9.1,  (imperative method)

kubectl set image deployment.v1.apps/my-nginx nginx=pkw0301/nginx:1.9.1

kubectl get svc

<VM IP>:<NodePort>

A rolling update works by:

1. Creating a new replication controller with the updated configuration.
2. Increasing/decreasing the replica count on the new and old controllers until the correct number of replicas is reached.
3. Deleting the original replication controller.

If you encounter a problem, you can stop the rolling update **midway and revert** to the previous version using **--rollback**:

kubectl rolling-update my-nginx --rollback

expose your rc

kubectl expose rc my-nginx --port=80 --target-port=80 --type=NodePort

Now if you want to rollback once rollout completed

kubectl rolling-update my-nginx --image=nginx:1.7.9

## **Rolling Updates with a Deployment**

vi deployment-rollingupdate.yml

apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2

kind: Deployment

metadata:

name: nginx-deployment

spec:

selector:

matchLabels:

app: nginx

replicas: 2 # tells deployment to run 2 pods matching the template

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.7.9

ports:

- containerPort: 80

Kubectl create -f deployment-rollingupdate.yml

Kubectl get deployment

Kubectl describe deployment <deployment name>

To make change in our application lets update nginx image version from 1.7.9 to 1.9.8

Once you change in your yaml

kubectl apply –f <ymal file name>

kubectl get deployment

kubectl describe deployment <deployment name>

kubectl rollout status deployment/nginx-deployment

**Rollback**

kubectl rollout undo deployment/nginx-deployment

Once you rollback verify your configuration

kubeclt describe deployment <deployment name>

**Note**: During this new feature rollout

##### **Max Unavailable (means how many would be available)**

The maximum number of Pods that can be unavailable during the update process (The default value is 25 % for unavailable, then 75% would be available)

Ex: lets imagine we have 4 pods, here once we rollout new feature then 25% would be unavailable (means 3 pods will not get updated, one 1 gets updates.

Once 1st pod get updated then 2nd pod would get new updates and so on)

##### **Max Surge**

The maximum number of Pods that can be created over the desired number of Pods (The default value is 25 %.)

(let’s imagine we have 4 pods, then k8s will creates 1 pods in 1st rollout attempt and so on)

(Ideally, pods unavailability should be equal to max surge, so that k8s can handle load request pleasantly )