kubernetes

DEPLOYMENTS & REPLICASETS

❖ WORKLOAD RESOURCES:

➢ DEPLOYMENTS:

- A Deployment is a management tool for controlling the behavior of pods.
- Deployment manages a set of Pods to run an application workload, usually one that doesn't maintain state.
- Deployments are designed for stateless applications where any pod can replace another without issues. They handle scaling and rolling updates.
- A Deployment provides declarative updates for Pods and Replicasets.

USE CASES:

- The following are typical use cases for Deployments:

  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.

  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.

  ROLLBACK TO AN EARLIER DEPLOYMENT REVISION: if the current state of the Deployment is not stable. Each rollback updates the revision of the Deployment.

  SCALE UP THE DEPLOYMENT TO FACILITATE MORE LOAD: You can set up 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.

  PAUSE THE ROLLOUT OF A DEPLOYMENT: to apply multiple fixes to its PodTemplateSpec and then resume it to start a new rollout.

  USE THE STATUS OF THE DEPLOYMENT: as an indicator that a rollout has stuck.

  CLEAN UP OLDER REPLICASETS: that you don't need anymore.

BENEFITS:

DECLARATIVE UPDATES: You define the desired state of your application, and Kubernetes will handle the rest. This makes it easy to manage updates and maintain consistency.

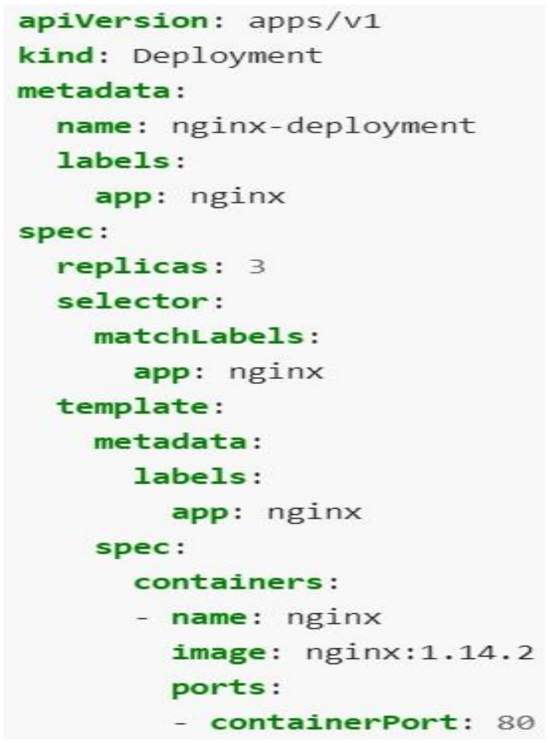
ROLLING UPDATES: Deployments support rolling updates, allowing you to update applications without downtime. If something goes wrong, you can roll back to a previous state.

SELF-HEALING: Deployments ensure that the desired number of pods are always running. If a pod fails, the deployment controller will automatically replace it.

SCALABILITY: You can scale the application up or down by changing the number of replicas. Kubernetes will handle the distribution of pods across the cluster.

LOAD BALANCING: Kubernetes automatically distributes traffic across all running pods, ensuring efficient resource utilization and improved application performance.

CREATING A DEPLOYMENT:



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

→ **Create the Deployment:**

```
$kubectl create -f <file-name>
```

→ **To see the Deployment rollout status**

```
$kubectl rollout status deployment/nginx-deployment
$kubectl get deployments
$kubectl describe deployment file-name
```

→ **To Check the replicaset (rs):**

```
$kubectl get rs
```

→ **To see the labels automatically generated for each Pod**

```
$kubectl get pods --show-labels
```

UPDATING A DEPLOYMENT:

→ **update the nginx Pods to use the nginx:1.16.1 image instead of the nginx:1.14.2 image:**

```
$kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1
```

→ **Alternatively, you can edit the Deployment and change:**

```
$kubectl edit deployment/nginx-deployment
```

→ **To see the rollout status:**

```
$kubectl rollout status deployment/nginx-deployment
$kubectl get pods
$kubectl describe deployments
```

→ **To see that the Deployment updated the Pods by creating a new ReplicaSet and scaling it up to 3 replicas:**

```
$kubectl get rs
```

→ **Get details of your Deployment:**

```
$kubectl describe deployments
```

ROLLING BACK A DEPLOYMENT:

- Sometimes, you may want to rollback a Deployment; for example, when the Deployment is not stable, such as crash looping. By default, all of the Deployment's rollout history is kept in the system so that you can rollback anytime you want.

  ```
  $kubectl set image deployment/nginx-deployment nginx=nginx:1.161
  ```

  ```
  $kubectl rollout status deployment/nginx-deployment
  ```

→ **Checking Rollout History of a Deployment:**

```
$kubectl rollout history deployment/nginx-deployment
```

→ **To see the details of each revision:**

```
$kubectl rollout history deployment/nginx-deployment --revision=2
```

→ **Rolling back to a previous revision:**

```
$kubectl rollout undo deployment/nginx-deployment
```

(or)

```
$kubectl rollout undo deployment/nginx-deployment --to-revision=2
```

→ **To check the deployments:**

```
$kubectl get deployment nginx-deployment
```

```
$kubectl describe deployment nginx-deployment
```

SCALING A DEPLOYMENT:

- Assuming **horizontal Pod autoscaling** is enabled in your cluster, you can set up 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 scale deployment/nginx-deployment --replicas=5
  ```

  ```
  $kubectl autoscale deployment/nginx-deployment --min=5 --max=10 --cpu-percent=60
  ```

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.

→ **To Pause the deployment:**

```
$kubectl rollout pause deployment/nginx-deployment
```

→ **Then update the image of the Deployment:**

```
$kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1
```

→ **Notice that no new rollout started:**

```
$kubectl rollout history deployment/nginx-deployment

$kubectl get rs
```

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

```
$kubectl set resources deployment/nginx-deployment -c=nginx --
limits=cpu=200m,memory=512Mi
```

→ **Resume the Deployment rollout and observe a new ReplicaSet coming up with all the new updates:**

```
$kubectl rollout resume deployment/nginx-deployment
```

→ **Watch the status of the rollout until it's done.**

```
$kubectl get rs -w

$kubectl ger rs --watch
```

DELETE A DEPLOYMENTS

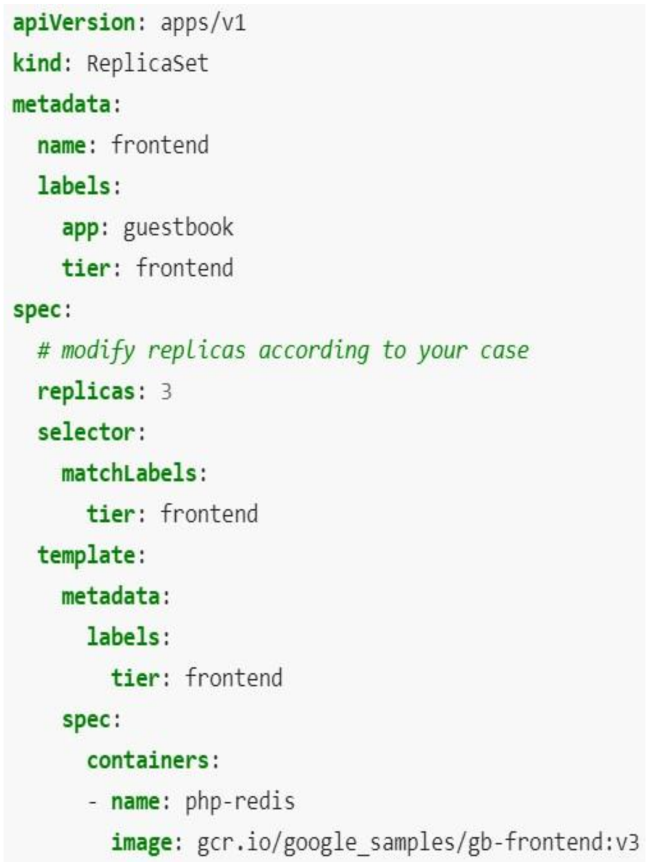
```
$kubectl get deployments

$kubectl delete deployment <deployment-name>

$kubectl get deployments
```

➢ REPLICASET:

- It is used to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.
- It has two main features: a pod template for creating new pods whenever existing ones fail, and a replica count for maintaining the desired number of replicas that the controller is supposed to keep running.
- A ReplicaSet also works to ensure additional pods are scaled down or deleted whenever an instance with the same label is created.



```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

→ Create a Replicaset:

$kubectl create -f file-name

$kubectl get rs

$kubectl describe rs rs-name

→ Delete a Replicaset:

$kubectl delete rs <rs-name>