Working with Kubernetes Objects

- Replication Controller
- ReplicaSets
- Deployments

Replication Controller

- Kubernetes encourages Desired State deployment
- Controllers are responsible for managing the pod lifecycle. It is responsible for making sure that the specified number of pod replicas are running at any point of time. They take care of Node failure.
- It is a best practice to use the replication controller to manage the pod life cycle rather than creating a pod again and again.

Replication.yaml

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx-rc
spec:
  replicas: 3
  selector:
    app: nginx-app
  template:
    metadata:
      ~~name: nginx-pod~~
      labels:
        app: nginx-app
    spec:
      containers:
      - name: mynginx-con
        image: nginx:1.18.0
        ports:
        - containerPort: 80
```

Run the following commands

- kubectl apply -f Replication.yaml
- kubectl get all
- kubectl delete pod/nginx-rc-<random-value>
- kubectl get all #New Pod created

Note that another POD is automatically created.

- kubectl scale replicationcontroller/nginx-rc --replicas=5
- kubectl get pods #2 new pods will be created
- kubectl delete -f Replication.yaml

ReplicaSet

- It is associated with a Pod and indicates how many instances of that Pod should be running within the cluster.
- It can be considered as a replacement of a replication controller. The key difference between the replica set and the replication controller is, the **replication controller** only supports **name equality-based selector** whereas the replica set supports **set-based selector**, a type of selection that enables a set of pods to be grouped so that the entire set is available for an operation defined in the ReplicaSet controller.
- The recommendation is to use replica sets along with higher-level controllers such as deployment.

Replica-set.yaml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: nginx-rs
spec:
 replicas: 5
 selector:
  matchLabels:
    app: nginx-app
  #matchExpressions:
  #- {key: app, operator: In, values: [nginx-app, demo-app]} #Provide space after ":"
 template:
  metadata:
   name: nginx-pod
   labels:
    app: nginx-app
  spec:
   containers:
   - name: mynginx-con
    image: nginx
    ports:
    - containerPort: 80
```

// supported values for **imagePullPolicy**: "Always", "IfNotPresent", "Never"

Note that the section below Template is what we have earlier used for Pod

Execute the following commands

- kubectl apply -f replica-set.yaml
- kubectl get replicaset nginx-rs
- kubectl **scale** --replicas=2 replicaset/nginx-rs
- kubectl get pods
- kubectl delete pods nginx-rs-wc28g
- kubectl get pods

  Note that the existing pod is deleted and immediately a new pod is created
- kubectl label pods <pod-name> app-
- kubectl get pods
- kubectl delete -f replica-set.yaml

  Note that deleting the replica set automatically deletes all the Pods

Replica Set is the next generation of Replication Controller. Replication controller is kinda imperative, but replica sets try to be as declarative as possible.

The main difference between a Replica Set and a Replication Controller right now is the selector support.

Replica Set	**Replication Controller**
Replica Set supports the new **set-based selector**. This gives more flexibility. for eg: **environment in (production, qa)** This selects all resources with key equal to environment and value equal to production or qa	Replication Controller only supports **equality-based selector.** For eg: **environment = production** This selects all resources with key equal to environment and value equal to production

Limitation of ReplicaSet:

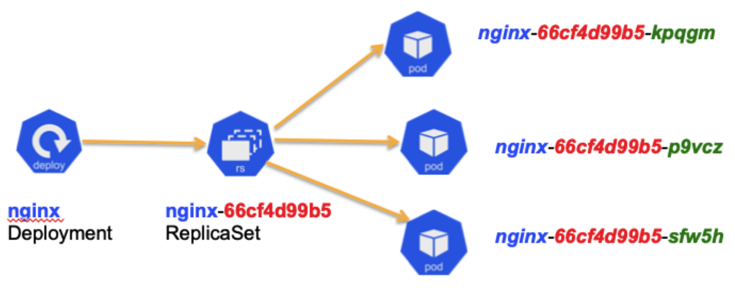
If we change the image in pod template, ReplicaSet will not automatically destroy the old pods and replace them with new pods.

About Deployments

- The deployment controller wraps around and extends the ReplicaSet controller.
- The Deployment instructs Kubernetes on how to **create and update instances** of your application. Once you've created a Deployment, the Kubernetes master **schedules** mentioned application instances onto individual Nodes in the cluster.
- Once the application instances are created, a Kubernetes Deployment Controller continuously monitors those instances. If the Node hosting an instance goes down or is deleted, the Deployment controller replaces it. **This provides a self-healing mechanism to address machine failure or maintenance.**
- Deployments are upgraded and higher version of replication controller. They manage the deployment of replicasets which is also an upgraded version of the replication controller. **They have the capability to update the replicaset and are also capable of rolling back to the previous version**.

Imperative Command To Create Deployment

```
kubectl create deployment nginx-deployment --image nginx:1.16.0
```



```
kubectl scale deployment nginx-deployment --replicas 3
```

Note: If the Image is changed the old pod will be deleted and a new Pod will be created with new container.

Note: If replicaset is directly used – Container Image cannot be changed.

When we specified the scale command, we were technically updating the Deployment spec.
Since in Kubernetes, everything has a spec, this spec for the Deployment changed the ReplicaSet to a set of 3 replicas. Then that ReplicaSet controller decided to change it to 3 pods and there would be one pod for each replica. Then the control plane makes a decision about which nodes would get assigned those pods (in our case we have got one node).

Then if you had a multi-node setup, the **kubelet agent** would then receive the instruction from **API Server** and would then be responsible for create the pod in its local Docker Engine.
While all this seems like a lot of work going on the background to start another container, it actually happens really fast.

Declarative:

deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  annotations:
    kubernetes.io/change-cause: 'nginx:1.16.0'
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx-app
```

```
template:
  metadata:
    labels:
      app: nginx-app
  spec:
    containers:
    - name:  nginx-con
      image: nginx:1.16.0
      ports:
      - containerPort: 80
      imagePullPolicy: Always
```

Declarive Commands:

```
kubectl apply -f deployment.yaml
kubectl annotate deployment nginx-deployment kubernetes.io/change-cause="Version = nginx:1.16.0"
kubectl rollout history deployment.apps/nginx-deployment

//Edit the YAML file and change the image from nginx:1.16.0 to nginx:1.17.0
kubectl apply -f deployment.yaml
kubectl annotate deployment nginx-deployment kubernetes.io/change-cause="Version = nginx:1.17.0"
kubectl rollout history deployment.apps/nginx-deployment
//Note that the OLD Pods are now replaced with NEW Pods.
//For every change in the Pod spec (under template) – A new ReplicaSet is created.

//Edit the YAML file and change the image from nginx:1.17.0 to nginx:1.18.0
kubectl apply -f deployment.yaml
kubectl annotate deployment nginx-deployment kubernetes.io/change-cause="Version = nginx:1.18.0"
kubectl rollout history deployment.apps/nginx-deployment

kubectl rollout undo deployment nginx-deployment
//It will rollback to the immediate previous deployment (1.17.0)
kubectl rollout undo deployment nginx-deployment
//It will rollback to the immediate previous deployment (1.18.0)
kubectl rollout history deployment.apps/nginx-deployment
kubectl rollout undo deployment nginx-deployment --to-revision=1
//It will rollback to the revision=1 (1.16.0)
```

//Edit the image from nginx:1.18.0 to nginx:1.1888888.0

```
kubectl rollout status deployment.apps/nginx-deployment
kubectl rollout undo deployment nginx-deployment
kubectl rollout status deployment.apps/nginx-deployment
```

Viewing Logs of a Deployment:

```
kubectl logs deployment/nginx-deployment
```

Deployment Strategy:

When you update the pod definition in your Deployment, the DeploymentController will start a rolling update process. It does this by simply managing ReplicaSets for you. Assume you already have code running in your Deployment.

1. A new ReplicaSet is created with the new Pod configuration. The Replicas count is zero.
2. The Replicas count will be increased on the new ReplicaSet.
3. Once the pods are launched, the Replicas count on the original ReplicaSet are reduced.
4. This process will continue until the new ReplicaSet has the original Replicas count and the old ReplicaSet has a Replicas count of zero.
5. The old ReplicaSet will hang around empty. A rollback will reverse this process.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  . . .
  strategy:
    type: RollingUpdate #Recreate
    rollingUpdate:
      maxUnavailable: 25%
      maxSurge: 25%
```

Rolling Update:

- **maxUnavailable**: specifies how many pods can be **unavailable** at any time during rollout. You can specify absolute number or percentage. Default is 25%.
- **maxSurge**: How many pods can be created over the replicas count. You can specify absolute number or percentage. Default is 25%.

To Force Pull Image without changing Tag

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 10
  selector:
    matchLabels:
      app: nginx-app
  template:
    metadata:
      name: nginx-pod
      labels:
        app: nginx-app
    spec:
      containers:
      - name: mynginx-con
        image: nginx:latest
        ports:
        - containerPort: 80
        imagePullPolicy: Always  #IfNotPresent / Never
```

Execute the following command to let deployment use latest image even if YAML is not updated.

```
kubectl patch deployment nginx-deployment -p
"{\"spec\":{\"template\":{\"metadata\":{\"annotations\":{\"date\":\"`date +'%s'`\"}}}}}"
```