DEPLOYMENT STRATEGY IN K8S

WHAT IS DEPLOYMENT STRATEGY

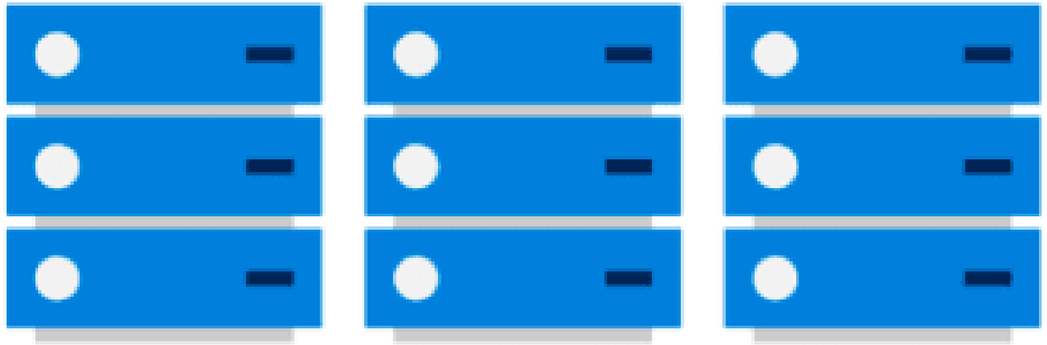
These are the techniques which are used to manage the rollout and scaling of applications within a Kubernetes cluster

1. Canary Deployment
2. Recreate Deployment.
3. Rolling update
4. Blue-Green Deployment

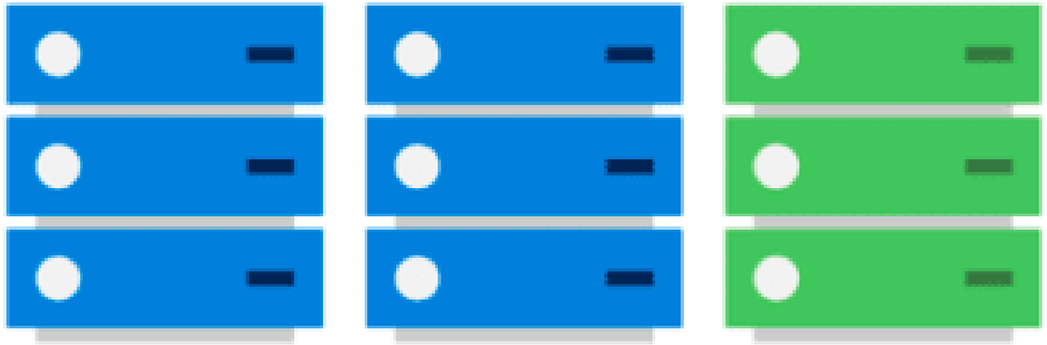
1. CANARY DEPLOYMENT

A Canary Deployment involves rolling out a new version of your application to a subset of your pods or a percentage of your traffic to test it before deploying it to the entire application in production.

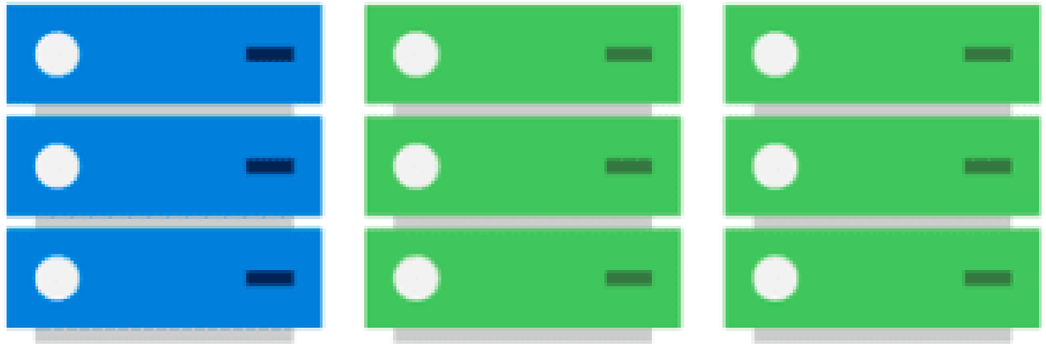
State 0



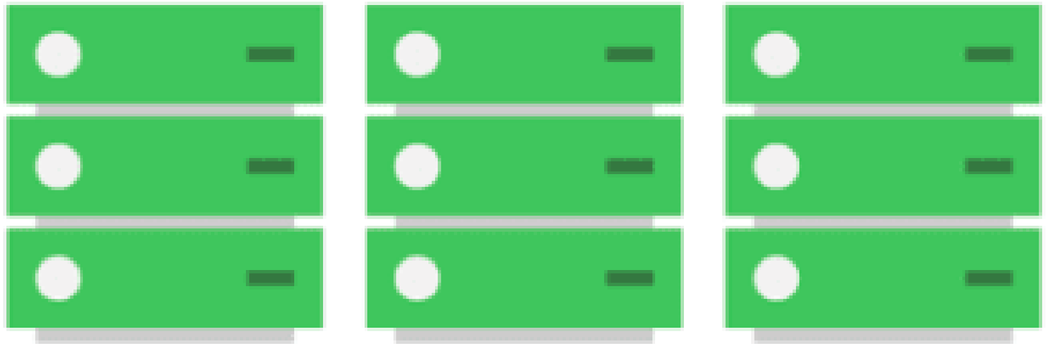
State 1



State 2



Final State



2. RECREATE DEPLOYMENT:

In this strategy, the existing version of the application is terminated completely, and a new version is deployed in its place. This approach is simple but may cause downtime during the update.

3. Rolling Update

pod

create new pods

It starts by creating a few pods of the new version

Monitor Pods

It monitors the new pods to make sure they are healthy and working well.

Route traffic

If the new pods are working fine, Kubernetes gradually increases their number while reducing the old pods.

pod

Delete old pods

This process continues until all pods are running the new version, and the old version has been phased out.

create a deployment file

```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:1.21.1
        ports:
        - containerPort: 80
```

execute this file : kubectl create -f deployment.yml

Check the deployments now : kubectl get deployment

Check the RS : kubectl get rs

```
root@ip-172-31-7-191:~/myfiles# kubectl get deployment
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment    3/3     3            3           12s
```

```
root@ip-172-31-7-191:~/myfiles# kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-7d5bb7fdcc   3         3         3       33s
```

Now update the image: kubectl set image deployment/nginx-deployment nginx=shaikmustafa/cycle
Check the RS : kubectl get rs

```
root@ip-172-31-7-191:~/myfiles# kubectl set image deployment/nginx-deployment nginx=shaikmustafa/cycle
deployment.apps/nginx-deployment image updated
```

```
root@ip-172-31-7-191:~/myfiles# kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-7d5bb7fdcc   0         0         0       2m58s
nginx-deployment-855cf4d748   3         3         3       12s
```

Now we can observe, old pods from RS-1 was completely deleted and RS-2 created new pods for version-2 it contains some downtime, To avoid this we will use Rolling Updates

create a deployment file with strategy:

```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:1.21.1
        ports:
        - containerPort: 80
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
```

execute this file : kubectl create -f deployment.yml

Check the deployments now : kubectl get deployment

Check the RS : kubectl get rs

```
root@ip-172-31-7-191:~/myfiles# kubectl get deployment
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment    3/3     3            3           12s
```

```
root@ip-172-31-7-191:~/myfiles# kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-7d5bb7fdcc   3         3         3       33s
```

- **maxSurge** specifies how many new Pods can be created by the Deployment controller in a “roll” in addition to the number of DESIRED
- **maxUnavailable** refers to how many old Pods can be deleted by the Deployment controller in a “rolling”

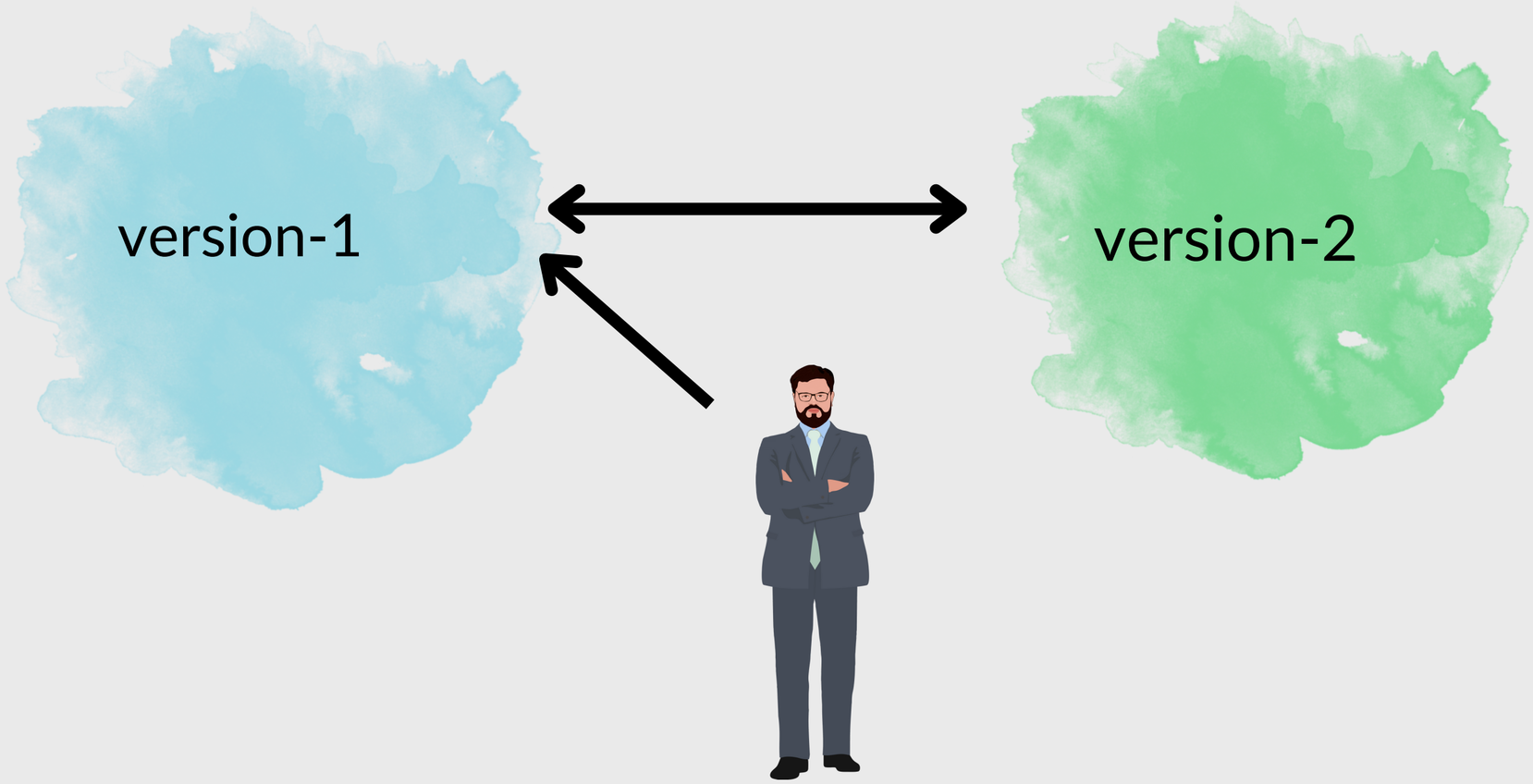
Now update the image: kubectl set image deployment/nginx-deployment nginx=shaikmustafa/cycle
Check the RS : kubectl get rs

```
root@ip-172-31-7-191:~/myfiles# kubectl set image deployment/nginx-deployment nginx=shaikmustafa/cycle
deployment.apps/nginx-deployment image updated
```

```
nginx-deployment-8449945bc4   2         2         0         1s
nginx-deployment-855cf4d748   2         2         2         51s
```

So we have 3 Pod copies, then the controller will always ensure that at least 2 Pods are available during the “rolling update” process, and at most only 4 Pods exist in the cluster at the same time. This strategy is a field of the Deployment object, named RollingUpdateStrategy

4. BLUE-GREEN DEPLOYMENT



A Blue/Green deployment is a way of accomplishing a zero-downtime upgrade to an existing application. The “Blue” version is the currently running copy of the application and the “Green” version is the new version. Once the green version is ready, traffic is rerouted to the new version

Step 1: Create Blue Deployment

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

create same deployment with same image but change names and env of the deployment.

Now execute both the deployments

kubectl create -f blue.yml
kubectl create -f green.yml

Step 2: Create Green Deployment

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

Step 3: Create a Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: demoappp-service
spec:
  selector:
    app: demoapp
  ports:
  - name: http
    port: 80
    targetPort: 80
  type: LoadBalancer
~
```

Now execute the service file

kubectl create -f svc.yml

Now verify that nginx image is running in the browser or not. That means we can see the application running in the blue environment.

Step 5: Perform Blue-Green Deployment

Now that we have both blue and green deployments running, we can perform the Blue-Green Deployment by routing traffic from the blue deployment to the green deployment.

Now update the image in green-deployment file from nginx to httpd

```
spec:
  containers:
  - name: demo
    image: httpd
    ports:
    - containerPort: 80
```

kubectl apply -f green.yml

Now update the service - to route traffic to the green deployment. To do this, update the label selector in the service manifest to select the green deployment.

```
  selector:
    app: demoapp
    env: green
```

kubectl apply -f svc.yml

Finally, we need to verify that the deployment was successful