Deploying with Blue/Green and Canary Strategies

Relevant Documentation

- Deployment
- Service

Exam Tips

- You can use multiple Deployments to set up blue/green environments in Kubernetes.
- Use labels and selectors on Services to direct user traffic to different Pods.
- A simple way to set up a canary environment in Kubernetes is to use a Service that selects Pods from 2 different Deployments. Vary the number of replicas to direct fewer users to the canary environment.

Lesson Reference

Log in to the **control plane node**.

Create a Deployment.

```
vi blue-deployment.yml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: blue-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bluegreen-test
      color: blue
  template:
    metadata:
      labels:
        app: bluegreen-test
        color: blue
    spec:
      containers:
      - name: nginx
        image: linuxacademycontent/ckad-nginx:blue
        ports:
        - containerPort: 80
```

```
kubectl apply -f blue-deployment.yml
```

Create a Service to route traffic to the Deployment's Pods.

```
vi bluegreen-test-svc.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: bluegreen-test-svc
spec:
  selector:
    app: bluegreen-test
    color: blue
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

```
kubectl apply -f bluegreen-test-svc.yml
```

Get the Service's cluster IP address.

```
kubectl get service bluegreen-test-svc
```

Test the Service.

```
curl <bluegreen-test-svc cluster IP address>
```

Create a green Deployment.

```
vi green-deployment.yml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: green-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bluegreen-test
      color: green
  template:
    metadata:
      labels:
        app: bluegreen-test
        color: green
    spec:
      containers:
      - name: nginx
        image: linuxacademycontent/ckad-nginx:green
        ports:
        - containerPort: 80
```

```
kubectl apply -f green-deployment.yml
```

Verify the status of the green deployment.

```
kubectl get deployment green-deployment
```

Test the green Deployment's Pod directly to verify it is working before sending user traffic to it.

First, get the IP address of the green Deployment's Pod.

```
kubectl get pods -o wide
```

Copy the `green-deployment` Pod's IP address and use it to test the Pod.

```
curl <green-deployment Pod IP address>
```

Edit the Service selector so that it points to the green Deployment's Pods.

```
kubectl edit service bluegreen-test-svc
```

```
spec:
  selector:
    app: bluegreen-test
    color: green
```

Test the Service again. The response should indicate that you are communicating with the green version Pod.

```
kubectl get service bluegreen-test-svc

curl <bluegreen-test-svc cluster IP address>
```

Now let's use a canary Deployment strategy.

Create a main Deployment.

```
vi main-deployment.yml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: main-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: canary-test
      environment: main
  template:
    metadata:
      labels:
        app: canary-test
        environment: main
    spec:
      containers:
      - name: nginx
        image: linuxacademycontent/ckad-nginx:1.0.0
        ports:
        - containerPort: 80
```

```
kubectl apply -f main-deployment.yml
```

Create a Service.

```
vi canary-test-svc.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: canary-test-svc
spec:
  selector:
    app: canary-test
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

```
kubectl apply -f canary-test-svc.yml
```

Get the Service's cluster IP address.

```
kubectl get service canary-test-svc
```

Test the Service. At this point, you should only get responses from the `main` environment.

```
curl <canary-test-svc cluster IP address>
```

Create a canary Deployment.

```
vi canary-deployment.yml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: canary-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: canary-test
      environment: canary
  template:
    metadata:
      labels:
        app: canary-test
        environment: canary
    spec:
      containers:
      - name: nginx
        image: linuxacademycontent/ckad-nginx:canary
        ports:
        - containerPort: 80
```

```
kubectl apply -f canary-deployment.yml
```

Test the Service again. Re-run this command multiple times. You should get responses from both the main environment and the `canary` environment.

```
curl <canary-test-svc cluster IP address>
```