Non-Persisting State

In this lesson, we will deploy Jenkins and analyze its state.

WE'LL COVER THE FOLLOWING

Deploying Jenkins

This time we'll deploy Jenkins and see what challenges we will face.

Looking into the Definition

Let's take a look at the `volume/jenkins.yml` definition.

```
cat volume/jenkins.yml
```

The **output** is as follows.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: jenkins
  annotations:
    kubernetes.io/ingress.class: "nginx"
    ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
  - http:
      paths:
      - path: /jenkins
        backend:
          serviceName: jenkins
          servicePort: 8080
```

```
---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: jenkins
spec:
  selector:
    matchLabels:
      type: master
      service: jenkins
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        type: master
        service: jenkins
    spec:
      containers:
      - name: jenkins
        image: vfarcic/jenkins
        env:
        - name: JENKINS_OPTS
          value: --prefix=/jenkins

---

apiVersion: v1
kind: Service
metadata:
  name: jenkins
spec:
  ports:
  - port: 8080
  selector:
    type: master
    service: jenkins
```

There's nothing special in that YAML file. It defines an Ingress with `/jenkins` path, a Deployment, and a Service. We won't waste time with it. Instead, we'll move on and create the objects.

Creating the Objects

```
kubectl create \
    -f volume/jenkins.yml \
    --record --save-config

kubectl rollout status deploy jenkins
```

We created the objects and waited until the processes finished. Now we can open Jenkins in our browser of choice.

```
open "http://$(minikube ip)/jenkins"
```

Jenkins UI opened, thus confirming that the application is deployed correctly. Jenkins' primary function is to execute jobs, so it's only fair to create one.

```
open "http://$(minikube ip)/jenkins/newJob"
```

> Please type *test* in the *item name* field, select *Pipeline* as the type, and click the *OK* button.

There's no need to make the Pipeline do any specific set of tasks. For now, you should be fine if you just *Save* the job.

Killing the Pod and Analysing the State

Let's explore what happens if the main process inside the Jenkins container dies.

```
POD_NAME=$(kubectl get pods \
    -l service=jenkins,type=master \
    -o jsonpath="{.items[*].metadata.name}")

kubectl exec -it $POD_NAME kill 1
```

We retrieved the name of the Pod, and we used it to execute `kill 1` inside its only container. The result is a simulation of a failure. Soon afterward, Kubernetes detected the failure and recreated the container. Let's double-check all that.

```
kubectl get pods
```

The **output** is as follows.

```
NAME                       READY STATUS  RESTARTS AGE
jenkins-76d59945d8-zcz8m 1/1   Running 1        12m
```

We can see that a container is running. Since we killed the main process and, with it, the first container, the number of restarts was increased to one.

Let’s go back to Jenkins UI and check what happened to the job. I’m sure you already know the answer, but we’ll double check it anyways.

```
open "http://$(minikube ip)/jenkins"
```

As expected, the job we created is gone. When Kubernetes recreated the failed container, it created a new one from the same image. Everything we generated inside the running container is no more. We reset to the initial state.

In the next lesson, we will analyze the state of updated Jenkins Deployment and discuss the emptyDir Volume type.