Guided Exercise: Application High Availability with Kubernetes

Simulate different types of application failures and observe how Kubernetes handles them.

Outcomes

- Explore how the `restartPolicy` attribute affects crashing pods.
- Observe the behavior of a slow-starting application that has no configured probes.
- Use a deployment to scale the application, and observe the behavior of a broken pod.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command ensures that the following conditions are true:

- The `reliability-ha` project exists.
- The resource files are available in the course directory.
- The classroom registry has the `long-load` container image.

The `long-load` container image contains an application with utility endpoints. These endpoints perform such tasks as crashing the process and toggling the server's health status.

```
[student@workstation ~]$ lab start reliability-ha
```

Instructions

1. As the `developer` user, create a pod from a YAML manifest in the `reliability-ha` project.

   Log in to the OpenShift cluster as the `developer` user with `developer` as the password.

   ```
   [student@workstation ~]$ oc login -u developer -p developer \
     https://api.ocp4.example.com:6443
   Login successful.

   ...output omitted...
   ```

   Select the `reliability-ha` project.

   ```
   [student@workstation ~]$ oc project reliability-ha
   Now using project "reliability-ha" on server "https://api.ocp4.example.com:6443".
   ```

   Go to the lab materials directory and view the contents of the pod definition. In particular, the manifest sets `restartPolicy` to `Always`.

   ```
   [student@workstation ~]$ cd DO180/labs/reliability-ha
   ```

   ```
   [student@workstation reliability-ha]$ cat long-load.yaml
   apiVersion: v1
   kind: Pod
   metadata:
     name: long-load
   spec:
     containers:
     - image: registry.ocp4.example.com:8443/redhattraining/long-load:v1
       name: long-load
       securityContext:
         allowPrivilegeEscalation: false
     restartPolicy: Always
   ```

   Create a pod by using the `oc apply` command.

   ```
   [student@workstation reliability-ha]$ oc apply -f long-load.yaml
   pod/long-load created
   ```

   Send a request to the pod to confirm that it is running and responding.

   ```
   [student@workstation reliability-ha]$ oc exec long-load -- \
     curl -s localhost:3000/health
   Ok
   ```

2. Trigger the pod to crash, and observe that the `restartPolicy` instructs the cluster to re-create the pod.

   Observe that the pod is running and did not restart.

```
[student@workstation reliability-ha]$ oc get pods

NAME        READY   STATUS    RESTARTS   AGE
long-load   1/1     Running   0          1m
```

   Send a request to the `/destruct` endpoint in the application. This request triggers the process to crash.

```
[student@workstation reliability-ha]$ oc exec long-load -- \
  curl -s localhost:3000/destruct
command terminated with exit code 52
```

   Observe that the pod is running and restarted one time.

```
[student@workstation reliability-ha]$ oc get pods
NAME        READY   STATUS    RESTARTS      AGE
long-load   1/1     Running   1 (34s ago)   4m16s
```

   Delete the `long-load` pod.

```
[student@workstation reliability-ha]$ oc delete pod long-load
pod "long-load" deleted
```

   The cluster does not re-create the pod, because you manually created the pod outside of a workload resource, such as a deployment.

3. Use a restart policy of `Never` to create the pod, and observe that the cluster does not re-create the pod on crashing.

   Modify the `long-load.yaml` file so that the `restartPolicy` is set to `Never`.

```
...output omitted...
  restartPolicy: Never
```

   Create the pod with the updated YAML file.

```
[student@workstation reliability-ha]$ oc apply -f long-load.yaml
pod/long-load created
```

   Send a request to the pod to confirm that the pod is running and that the application is responding.

```
[student@workstation reliability-ha]$ oc exec long-load -- \
  curl -s localhost:3000/health
Ok
```

   Send a request to the `/destruct` endpoint in the application to crash it.

```
[student@workstation reliability-ha]$ oc exec long-load -- \
  curl -s localhost:3000/destruct
command terminated with exit code 52
```

   Observe that the cluster does not restart the pod and is in an error state.

```
[student@workstation reliability-ha]$ oc get pods
NAME        READY   STATUS    RESTARTS   AGE
long-load   0/1     Error     0          2m36s
```

   Delete the `long-load` pod.

```
[student@workstation reliability-ha]$ oc delete pod long-load
pod "long-load" deleted
```

4. Because the cluster does not know when the application inside the pod is ready to receive requests, you must add a startup delay to the application. A later exercise covers adding this capability by using probes.

   Update the `long-load.yaml` file by adding a startup delay and use a restart policy of `Always`. Set the `START_DELAY` variable to 60,000 milliseconds (one minute) so that the file exactly matches the following excerpt:

```
...output omitted...
spec:
  containers:
  - image: registry.ocp4.example.com:8443/redhattraining/long-load:v1
    imagePullPolicy: Always
    securityContext:
      allowPrivilegeEscalation: false
    name: long-load
    env:
      - name: START_DELAY
        value: "60000"
  restartPolicy: Always
```

> **NOTE**
>
> Although numbers are a valid YAML type, you must pass environment variables as strings. YAML syntax is also indentation-sensitive.
>
> For these reasons, ensure that your file appears *exactly* as in the preceding example.

Apply the YAML file to create the pod and proceed within one minute to the next step.

```
[student@workstation reliability-ha]$ oc apply -f long-load.yaml
pod/long-load created
```

Within one minute of pod creation, verify the status of the pod. The status shows as `Running` even though it is not. Try to send a request to the application, and observe that it fails.

```
[student@workstation reliability-ha]$ oc get pods
NAME        READY   STATUS    RESTARTS   AGE
long-load   1/1     Running   0          16s
```

```
[student@workstation reliability-ha]$ oc exec long-load -- \
  curl -s localhost:3000/health
app is still starting
```

After waiting one minute for the application to start, send another a request to the pod to confirm that it is running and responding.

```
[student@workstation reliability-ha]$ oc exec long-load -- \
  curl -s localhost:3000/health
Ok
```

5. Use a deployment to scale up the number of deployed pods. Observe that deleting the pods causes service outages, even though the deployment handles re-creating the pods.

   Review the `long-load-deploy.yaml` file, which defines a deployment, service, and route. The deployment creates three replicas of the application pod.

   In each pod, the deployment file sets a `START_DELAY` environment variable to 15,000 milliseconds (15 seconds). In each pod, the application responds that it is not ready until after the delay.

```
[student@workstation reliability-ha]$ cat long-load-deploy.yaml
...output omitted...
    spec:
      containers:
      - image: registry.ocp4.example.com:8443/redhattraining/long-load:v1
        imagePullPolicy: Always
        name: long-load
        env:
        - name: START_DELAY
          value: "15000"
...output omitted...
```

   Start the load test script, which sends a request to the `/health` API endpoint of the application every two seconds. Leave the script running in a visible terminal window.

```
[student@workstation reliability-ha]$ ./load-test.sh
...output omitted...
```

In a new terminal window, apply the `~/DO180/labs/reliability-ha/long-load-deploy.yaml` file.

```
[student@workstation ~]$ oc apply -f \
  ~/DO180/labs/reliability-ha/long-load-deploy.yaml
deployment.apps/long-load created
service/long-load created
route.route.openshift.io/long-load created
```

Watch the output of the load test script as the pods and the application instances start. After a delay, the requests succeed.

```
...output omitted...
Ok
Ok
Ok
...output omitted...
```

By using the `/togglesick` API endpoint of the application, put one of the three pods into a broken state.

```
[student@workstation ~]$ curl \
  long-load-reliability-ha.apps.ocp4.example.com/togglesick
no output expected
```

Watch the output of the load test script as some requests start failing. Because of the load balancer, the exact order of the output is random.

```
...output omitted...
Ok
app is unhealthy
app is unhealthy
Ok
Ok
...output omitted...
```

Press **Ctrl**+**C** to end the load test script.

Return to the `/home/student/` directory.

```
[student@workstation reliability-ha]$ cd /home/student/
[student@workstation ~]$
```

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish reliability-ha
```