Lab: Configure an Application for Reliability

Deploy and troubleshoot a reliable application that defines health probes, compute resource requests, and compute resource limits so it can run N instances per node; and configure a horizontal pod autoscaler that scales to a maximum of N instances.

Outcomes

- Set the resource requests for a deployment.
- Configure a readiness probe.
- Create a horizontal pod autoscaler.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise. This command ensures that the cluster is accessible and that all resources are available for this exercise.

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

Instructions

In this exercise, you work in the `reliability-review` project that the `lab` command created. The command also deployed the `longload` application in the `reliability-review` project.

The API URL of your OpenShift cluster is https://api.ocp4.example.com:6443, and the `oc` command is already installed on your `workstation` machine.

1. Log in to the OpenShift cluster as the `developer` user with `developer` as the password, and change to the `reliability-review` project. You use this project for all your work in this exercise.

   Log in to the OpenShift cluster.

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

   Change to the `reliability-review` project.

   ```
   [student@workstation ~]$ oc project reliability-review
   ...output omitted...
   ```

2. The `longload` application in the `reliability-review` project fails to start. Diagnose and then fix the issue. The application needs 512 MiB of memory to work.

After you fix the issue, you can confirm that the application works by running the `~/DO180/labs/reliability-review/curl_loop.sh` script that the `lab` command prepared. The script sends requests to the application in a loop. For each request, the script displays the pod name and the application status. Press **Ctrl**+**C** to end the script.

List the pods in the project. The pod is in the `Pending` status. The name of the pod is diff

```
[student@workstation ~]$ oc get pods
NAME                        READY   STATUS    RESTARTS   AGE
longload-64bf8dd776-b6rkz   0/1     Pending   0          8m1s
```

Retrieve the events for the pod. No compute node has enough memory to accommodat

```
[student@workstation ~]$ oc describe pod longload-64bf8dd776-b6rkz
Name:             longload-64bf8dd776-b6rkz
Namespace:        reliability-review
...output omitted...
Events:
  Type     Reason            Age   From               Message
  ----     ------            ----  ----               -------
  Warning  FailedScheduling  8m    default-scheduler  0/1 nodes are available: 1 I
nodes are available: 1 No preemption victims found for incoming pod.
```

Verify that the `longload` deployment requests 8 GiB of memory by running the following

```
[student@workstation ~]$ oc get deployment longload -o \
  jsonpath='{.spec.template.spec.containers[0].resources.requests.memory}{"\n"}'
8Gi
```

Set the memory requests for the `longload` deployment to 512 MiB.

```
[student@workstation ~]$ oc set resources deployment/longload \
  --requests memory=512Mi
deployment.apps/longload resource requirements updated
```

Run the `watch` command and wait until the `longload` pod is running. The name of the po

```
[student@workstation ~]$ oc get pods
NAME                        READY   STATUS    RESTARTS   AGE
longload-5897c9558f-cx4gt   1/1     Running   0          86s
```

Press **Ctrl**+**C** to end the `watch` command after the pod displays `Running` in the `STATUS` co

Run the `~/DO180/labs/reliability-review/curl_loop.sh` script to confirm that the app `(7) Failed to connect …` message until the application is ready.

```
[student@workstation ~]$ ~/DO180/labs/reliability-review/curl_loop.sh
1 curl: (7) Failed to connect to master01.ocp4.example.com port 30372: Connection
2 longload-5897c9558f-cx4gt: app is still starting
3 longload-5897c9558f-cx4gt: app is still starting
4 longload-5897c9558f-cx4gt: app is still starting
5 longload-5897c9558f-cx4gt: Ok
6 longload-5897c9558f-cx4gt: Ok
7 longload-5897c9558f-cx4gt: Ok
8 longload-5897c9558f-cx4gt: Ok
...output omitted...
```

Press **Ctrl**+**C** to end the script after the application displays the `Ok` message.

3. When the `longload` application scales up, your customers complain that some requests fail. To replicate the issue, manually scale up the `longload` application to three replicas, and run the `~/DO180/labs/reliability-review/curl_loop.sh` script at the same time.

   The application takes seven seconds to initialize. The application exposes the `/health` API endpoint on HTTP port `3000`. Configure the `longload` deployment to use this endpoint, to ensure that the application is ready before serving client requests.

   Open a second terminal window and run the `~/DO180/labs/reliability-review/curl_l`

```
[student@workstation ~]$ ~/DO180/labs/reliability-review/curl_loop.sh
1 longload-5897c9558f-cx4gt: Ok
2 longload-5897c9558f-cx4gt: Ok
3 longload-5897c9558f-cx4gt: Ok
4 longload-5897c9558f-cx4gt: Ok
...output omitted...
```

   Leave the script running and do not interrupt it.

   In the first terminal window, scale up the `longload` application to three replicas.

```
[student@workstation ~]$ oc scale deployment/longload --replicas 3
deployment.apps/longload scaled
```

   Switch back to the second terminal, and watch the output of the `curl_loop.sh` script. So
   sends requests to the new pods before the application is ready.

```
...output omitted...
22 longload-5897c9558f-cx4gt: Ok
23 longload-5897c9558f-cx4gt: Ok
24 longload-5897c9558f-cx4gt: Ok
25 curl: (7) Failed to connect to master01.ocp4.example.com port 30372: Connection
26 curl: (7) Failed to connect to master01.ocp4.example.com port 30372: Connection
27 longload-5897c9558f-cx4gt: Ok
28 curl: (7) Failed to connect to master01.ocp4.example.com port 30372: Connection
29 longload-5897c9558f-cx4gt: Ok
30 curl: (7) Failed to connect to master01.ocp4.example.com port 30372: Connection
31 longload-5897c9558f-tpssf: app is still starting
32 longload-5897c9558f-kkvm5: app is still starting
33 longload-5897c9558f-cx4gt: Ok
34 longload-5897c9558f-tpssf: app is still starting
35 longload-5897c9558f-tpssf: app is still starting
36 longload-5897c9558f-tpssf: app is still starting
37 longload-5897c9558f-cx4gt: Ok
38 longload-5897c9558f-tpssf: app is still starting
39 longload-5897c9558f-cx4gt: Ok
40 longload-5897c9558f-cx4gt: Ok
...output omitted...
```

Leave the script running and do not interrupt it.

Add a readiness probe to the `longload` deployment.

```
[student@workstation ~]$ oc set probe deployment/longload --readiness \
  --initial-delay-seconds 7 \
  --get-url http://:3000/health
deployment.apps/longload probes updated
```

Scale down the `longload` application back to one pod.

```
[student@workstation ~]$ oc scale deployment/longload --replicas 1
deployment.apps/longload scaled
```

If scaling down breaks the `curl_loop.sh` script, then press **Ctrl**+**c** to stop the script in the
script.

To test your work, scale up the application to three replicas again.

```
[student@workstation ~]$ oc scale deployment/longload --replicas 3
deployment.apps/longload scaled
```

Watch the output of the `curl_loop.sh` script in the second terminal, and observe that no

```
...output omitted...
92 longload-7ddcc9b7fd-72dtm: Ok
93 longload-7ddcc9b7fd-72dtm: Ok
94 longload-7ddcc9b7fd-72dtm: Ok
95 longload-7ddcc9b7fd-qln95: Ok
96 longload-7ddcc9b7fd-wrxrb: Ok
97 longload-7ddcc9b7fd-qln95: Ok
98 longload-7ddcc9b7fd-wrxrb: Ok
99 longload-7ddcc9b7fd-72dtm: Ok
...output omitted...
```

Press **Ctrl**+**C** to end the script. Do not close the second terminal window.

4. Configure the `longload` application so that it automatically scales up when the average memory usage is above 60% of the memory requests value, and scales down when the usage is below this percentage. The minimum number of replicas must be one, and the maximum must be three. The resource that you create for scaling the application must be named `longload`.

   The `lab` command provides the `~/DO180/labs/reliability-review/hpa.yml` resource file as an example. Use the `oc explain` command to learn the valid parameters for the `hpa.spec.metrics.resource.target` attribute. Because the file is incomplete, you must update it first if you choose to use it.

   To test your work, use the `oc exec deploy/longload -- curl localhost:3000/leak` command to send an HTTP request to the application `/leak` API endpoint. Each request consumes an additional 480 MiB of memory. To free this memory, you can use the `~/DO180/labs/reliability-review/free.sh` script.

   Before you create the horizontal pod autoscaler resource, scale down the application to

   ```
   [student@workstation ~]$ oc scale deployment/longload --replicas 1
   deployment.apps/longload scaled
   ```

   Edit the `~/DO180/labs/reliability-review/hpa.yml` resource file. You can retrieve the p
   by using the `oc explain hpa.spec.metrics.resource` and `oc explain hpa.spec.metric`

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: longload
  labels:
    app: longload
spec:
  maxReplicas: 3
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: longload
  metrics:
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 60
```

Use the `oc apply` command to deploy the horizontal pod autoscaler.

```
[student@workstation ~]$ oc apply -f ~/DO180/labs/reliability-review/hpa.yml
horizontalpodautoscaler.autoscaling/longload created
```

In the second terminal, run the `watch` command to monitor the `oc get hpa longload` co
the `longload` horizontal pod autoscaler to report usage in the `TARGETS` column. The perc
cluster.

```
[student@workstation ~]$ watch oc get hpa longload
NAME       REFERENCE             TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
longload   Deployment/longload   13%/60%   1         3         1          75s
```

Leave the command running and do not interrupt it.

To test your work, run the `oc exec deploy/longload—curl localhost:3000/leak` comm
application to allocate 480 MiB of memory.

```
[student@workstation ~]$ oc exec deploy/longload -- curl -s localhost:3000/leak
longload-7ddcc9b7fd-72dtm: consuming memory!
```

In the second terminal, after a few seconds, observe that the `oc get hpa longload` com
horizontal pod autoscaler scales up the application to more than one replica. The percen
cluster.

```
NAME       REFERENCE             TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
longload   Deployment/longload   145%/60%   1         3         3          5m18s
```

To test your work, run the ~/DO180/labs/reliability-review/free.sh script in the first the memory. Ensure that the pod that was consuming memory now frees the memory. E if necessary.

```
[student@workstation ~]$ ~/DO180/labs/reliability-review/free.sh
longload-7ddcc9b7fd-72dtm: releasing memory!
```

In the second terminal, after ten minutes, observe that the `oc get hpa longload` comm horizontal pod autoscaler scales down the application to one replica. The percentages m

```
NAME       REFERENCE             TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
longload   Deployment/longload   12%/60%   1         3         1          15m28s
```

Press **Ctrl**+**C** to end the `watch` command. Close the second terminal window.

Evaluation

As the `student` user on the `workstation` machine, use the `lab` command to grade your work. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab grade reliability-review
```

Finish

As the `student` user 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-review
```