Guided Exercise: Scale and Expose Applications to External Access

Deploy one web server and access it through a Kubernetes ingress; and deploy another web server and access it through an OpenShift route.

Outcomes

In this exercise, you deploy two web applications to access them through an ingress object and a route, and scale them to verify the load-balance between the pods.

- Deploy two web applications.
- Create a route and an ingress object to access the web applications.
- Enable the sticky sessions for the web applications.
- Scale the web applications to load-balance the service.

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.

```
[student@workstation ~]$ lab start deploy-routes
```

Instructions

1. Create two web application deployments, named `satir-app` and `sakila-app`. Use the `registry.ocp4.example.com:8443/httpd-app:v1` container image for both deployments.

   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
   You have one project on this server: "web-applications"
   ...output omitted...
   ```

   Change to the `web-applications` project.

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

   Create the `satir-app` web application deployment by using the `registry.ocp4.example.com:8443/redhattraining/do180-httpd-app:v1` container image.

   ```
   [student@workstation ~]$ oc create deployment satir-app \
     --image registry.ocp4.example.com:8443/redhattraining/do180-httpd-app:v1
   deployment.apps/satir-app created
   ```

   After a few moments, verify that the deployment is successful.

   ```
   [student@workstation ~]$ oc get pods
   NAME                         READY   STATUS    RESTARTS   ...
   satir-app-787b7d7858-5dfsh   1/1     Running   0          ...
   ```

   ```
   [student@workstation ~]$ oc get deploy/satir-app
   NAME        READY   UP-TO-DATE   AVAILABLE   AGE
   satir-app   1/1     1            1           6m39s
   ```

   Create the `sakila-app` web application deployment by using the `registry.ocp4.example.com:8443/redhattraining/do180-httpd-app:v1` image.

   ```
   [student@workstation ~]$ oc create deployment sakila-app \
     --image registry.ocp4.example.com:8443/redhattraining/do180-httpd-app:v1
   deployment.apps/sakila-app created
   ```

   Wait a few moments and then verify that the deployment is successful.

```
[student@workstation ~]$ oc get pods
NAME                     READY   STATUS    RESTARTS   ...
sakila-app-6694...5kpd   1/1     Running   0          ...
satir-app-787b7...dfsh   1/1     Running   0          ...
```

```
[student@workstation ~]$ oc get deploy
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
sakila-app   1/1     1            1           41s
satir-app    1/1     1            1           62m
```

2. Create services for the web application deployments. Then, use the services to create a route for the `satir-app` application and an ingress object for the `sakila-app` application.

   Expose the `satir-app` deployment. Name the service `satir-svc`, and specify port `8080` as the port and target port.

```
[student@workstation ~]$ oc expose deployment satir-app --name satir-svc \
  --port 8080 --target-port 8080
service/satir-svc exposed
```

   Expose the `sakila-app` deployment to create the `sakila-svc` service.

```
[student@workstation ~]$ oc expose deployment sakila-app --name sakila-svc \
  --port 8080 --target-port 8080
service/sakila-svc exposed
```

   Verify the status of the services.

```
[student@workstation ~]$ oc get services
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    ...
sakila-svc   ClusterIP   172.30.230.41   <none>        8080/TCP   ...
satir-svc    ClusterIP   172.30.143.15   <none>        8080/TCP   ...
```

   Verify the status of the endpoints.

```
[student@workstation ~]$ oc get endpoints
NAME         ENDPOINTS       AGE
sakila-svc   10.8.0.66:8080   25s
satir-svc    10.8.0.65:8080   43s
```

   Verify the status of the pods.

```
[student@workstation ~]$ oc get pods -o wide
NAME                     READY  STATUS    RESTARTS  AGE   IP        ...
sakila-app-6694...5kpd   1/1    Running   0         92s   10.8.0.66 ...
satir-app-787b7...dfsh   1/1    Running   0         2m49s 10.8.0.65 ...
```

   Expose the `satir-svc` service to create a route named `satir` for the `satir-app` web application.

```
[student@workstation ~]$ oc expose service satir-svc --name satir
route.route.openshift.io/satir exposed
```

```
[student@workstation ~]$ oc get routes
NAME    HOST/PORT                                    ... SERVICES    PORT   ...
satir   satir-web-applications.apps.ocp4.example.com ... satir-svc   8080   ...
```

   Create an ingress object named `ingr-sakila` for the `sakila-svc` service. Configure the `--rule` option with the following values:

Field	Value
Host	`ingr-sakila.apps.ocp4.example.com`
Service name	`sakila-svc`
Port number	`8080`

```
[student@workstation ~]$ oc create ingress ingr-sakila \
  --rule "ingr-sakila.apps.ocp4.example.com/*=sakila-svc:8080"
ingress.networking.k8s.io/ingr-sakila created
```

```
[student@workstation ~]$ oc get ingress
NAME        ... HOSTS                              ADDRESS       PORTS ...
ingr-sakila ... ingr-sakila.apps.ocp4.example.com  router...com  80    ...
```

Confirm that a route exists for the `ingr-sakila` ingress object.

```
[student@workstation ~]$ oc get routes
NAME           HOST/PORT                                    ... SERVICES   PORT
ingr-sakila... ingr-sakila.apps.ocp4.example.com            ... sakila-svc <all>
satir          satir-web-applications.apps.ocp4.example.com ... satir-svc  8080
```

A specific port is not assigned to routes that ingress objects created. By contrast, a route that an exposed service created is assigned the same ports as the service.

Use the `curl` command to access the `ingr-sakila` ingress object and the `satir` route. The output states the name of the pod that is servicing the request.

```
[student@workstation ~]$ curl ingr-sakila.apps.ocp4.example.com
Welcome to Red Hat Training, from sakila-app-66947cdd78-x5kpd
```

```
[student@workstation ~]$ curl satir-web-applications.apps.ocp4.example.com
Welcome to Red Hat Training, from satir-app-787b7d7858-bdfsh
```

3. Scale the web application deployments to load-balance their services. Scale the `sakila-app` deployment to two replicas, and scale the `satir-app` deployment to three replicas.

   Scale the `sakila-app` deployment with two replicas.

```
[student@workstation ~]$ oc scale deployment sakila-app --replicas 2
deployment.apps/sakila-app scaled
```

   Wait a few moments and then verify the status of the replica pods.

```
[student@workstation ~]$ oc get pods
NAME                    READY   STATUS    RESTARTS   ...
sakila-app-6694...5kpd   1/1     Running   0          ...
sakila-app-6694...rfzg   1/1     Running   0          ...
satir-app-787b...dfsh    1/1     Running   0          ...
```

   Scale the `satir-app` deployment with three replicas.

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

   Wait a few moments and then verify the status of the replica pods.

```
[student@workstation ~]$ oc get pods -o wide
NAME                    READY  STATUS   RESTARTS  ... IP        ...
sakila-app-6694...5kpd  1/1    Running  0         ... 10.8.0.66 ...
sakila-app-6694...rfzg  1/1    Running  0         ... 10.8.0.67 ...
satir-app-787b...dfsh   1/1    Running  0         ... 10.8.0.65 ...
satir-app-787b...z8xm   1/1    Running  0         ... 10.8.0.69 ...
satir-app-787b...7bhj   1/1    Running  0         ... 10.8.0.70 ...
```

   Retrieve the service endpoints to confirm that the services are load-balanced between the additional replica pods.

```
[student@workstation ~]$ oc get endpoints
NAME         ENDPOINTS                                     ...
sakila-svc   10.8.0.66:8080,10.8.0.67:8080                 ...
satir-svc    10.8.0.65:8080,10.8.0.69:8080,10.8.0.70:8080  ...
```

4. Enable the sticky sessions for the `sakila-app` web application. Then, use the `curl` command to confirm that the sticky sessions are working for the `ingr-sakila` object.

   Configure a cookie for the `ingr-sakila` ingress object.

```
[student@workstation ~]$ oc annotate ingress ingr-sakila \
  ingress.kubernetes.io/affinity=cookie
ingress.networking.k8s.io/ingr-sakila annotated
```

Use the `curl` command to access the `ingr-sakila` ingress object. The output states the name of the pod that is servicing the request. Notice that the connection is load-balanced between the replicas.

```
[student@workstation ~]$ for i in {1..10}; do \
  curl ingr-sakila.apps.ocp4.example.com ; done
Welcome to Red Hat Training, from sakila-app-66947cdd78-x5kpd
Welcome to Red Hat Training, from sakila-app-66947cdd78-xrfzg
Welcome to Red Hat Training, from sakila-app-66947cdd78-x5kpd
...output omitted...
```

Use the `curl` command to save the `ingr-sakila` ingress object cookie to the `/tmp/cookie_jar` file.

```
[student@workstation ~]$ curl ingr-sakila.apps.ocp4.example.com \
  -c /tmp/cookie_jar
Welcome to Red Hat Training, from sakila-app-66947cdd78-xrfzg
```

Confirm that the cookie exists in the `/tmp/cookie_jar` file.

```
[student@workstation ~]$ cat /tmp/cookie_jar
...output omitted...
#HttpOnly_ingr-sakila.apps.ocp4.example.com     FALSE   /       FALSE   0       b9b484110526b4b1b3159860d3aebe04        92
1e139c5145950d00424bf3b0a46d22
```

The cookie provides session stickiness for connections to the `ingr-sakila` route. Use the `curl` command and the cookie in the `/tmp/cookie_jar` file to connect to the `ingr-sakila` route again. Confirm that you are connected to the same pod that handled the request in the previous step.

```
[student@workstation ~]$ for i in {1..10}; do \
  curl ingr-sakila.apps.ocp4.example.com -b /tmp/cookie_jar; done
Welcome to Red Hat Training, from sakila-app-66947cdd78-xrfzg
Welcome to Red Hat Training, from sakila-app-66947cdd78-xrfzg
Welcome to Red Hat Training, from sakila-app-66947cdd78-xrfzg
...output omitted...
```

Use the `curl` command to connect to the `ingr-sakila` route without the cookie. Observe that session stickiness occurs only with the cookie.

```
[student@workstation ~]$ for i in {1..10}; do \
  curl ingr-sakila.apps.ocp4.example.com ; done
Welcome to Red Hat Training, from sakila-app-66947cdd78-x5kpd
Welcome to Red Hat Training, from sakila-app-66947cdd78-xrfzg
Welcome to Red Hat Training, from sakila-app-66947cdd78-x5kpd
...output omitted...
```

5. Enable the sticky sessions for the `satir-app` web application. Then, use the `curl` command to confirm that sticky sessions are active for the `satir` route.

   Configure a cookie with the `hello` value for the `satir` route.

```
[student@workstation ~]$ oc annotate route satir \
  router.openshift.io/cookie_name="hello"
route.route.openshift.io/satir annotated
```

   Use the `curl` command to access the `satir` route. The output states the name of the pod that is servicing the request. Notice that the connection is load-balanced between the three replica pods.

```
[student@workstation ~]$ for i in {1..10}; do \
  curl satir-web-applications.apps.ocp4.example.com; done
Welcome to Red Hat Training, from satir-app-787b7d7858-bdfsh
Welcome to Red Hat Training, from satir-app-787b7d7858-gz8xm
Welcome to Red Hat Training, from satir-app-787b7d7858-q7bhj
...output omitted...
```

   Use the `curl` command to save the `hello` cookie to the `/tmp/cookie_jar` file.

```
[student@workstation ~]$ curl satir-web-applications.apps.ocp4.example.com \
  -c /tmp/cookie_jar
Welcome to Red Hat Training, from satir-app-787b7d7858-q7bhj
```

   Confirm that the `hello` cookie exists in the `/tmp/cookie_jar` file.

```
[student@workstation ~]$ cat /tmp/cookie_jar
...output omitted...
#HttpOnly_satir-web-applications.apps.ocp4.example.com  FALSE   /       FALSE   0       hello   b7dd73d32003e513a072e25a32
b6c881
```

The `hello` cookie provides session stickiness for connections to the `satir` route. Use the `curl` command and the `hello` cookie in the `/tmp/cookie_jar` file to connect to the `satir` route again. Confirm that you are connected to the same pod that handled the request in the previous step.

```
[student@workstation ~]$ for i in {1..10}; do \
  curl satir-web-applications.apps.ocp4.example.com -b /tmp/cookie_jar; done
Welcome to Red Hat Training, from satir-app-787b7d7858-q7bhj
Welcome to Red Hat Training, from satir-app-787b7d7858-q7bhj
Welcome to Red Hat Training, from satir-app-787b7d7858-q7bhj
...output omitted...
```

Use the `curl` command to connect to the `satir` route without the `hello` cookie. Observe that session stickiness occurs only with the cookie.

```
[student@workstation ~]$ for i in {1..10}; do \
  curl satir-web-applications.apps.ocp4.example.com; done
Welcome to Red Hat Training, from satir-app-787b7d7858-gz8xm
Welcome to Red Hat Training, from satir-app-787b7d7858-q7bhj
Welcome to Red Hat Training, from satir-app-787b7d7858-bdfsh
...output omitted...
```

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 deploy-routes
```