Introduction to Red Hat OpenShift

Objectives

In this lab, you will:

- Use the `oc` CLI (OpenShift command line interface)
- Use the OpenShift web console
- Build and deploy an application using s2i ('Source-to-image' build strategy)
- Inspect a BuildConfig and an ImageStream
- Autoscale the application

Verify the environment and command line tools

1. If a terminal is not already open, open a terminal window by using the menu in the editor: `Terminal > New Terminal`.

   Note: Please wait for some time for the terminal prompt to appear.

2. Verify that `oc` CLI is installed.

   ```
   oc version
   ```

You should see output similar to this, although the versions may be different.

3. Change to your project folder.

   NOTE: If you are already on `home/project` please skip this step

   ```
   cd /home/project
   ```

4. Clone the git repository that contains the artifacts needed for this lab, if it doesn't already exist.

```
[ ! -d 'CC201' ] && git clone https://github.com/ibm-developer-skills-network/CC201.git
```

Use the oc CLI

OpenShift projects are Kubernetes namespaces with additional administrative functions. Therefore, projects also provide isolation within an OpenShift cluster. You already have access to one project in an OpenShift cluster, and `oc` is already set to target that cluster and project.

Let's look at some basic `oc` commands. Recall that `oc` comes with a copy of `kubectl`, so all the `kubectl` commands can be run with `oc`.

1. List the Pods in this namespace.

```
oc get pods
```

You will likely see a few Pods that are part of the environment. You don't need to worry about these.

2. In addition to Kubernetes objects, you can get OpenShift specific objects.

```
oc get buildconfigs
```

Because you haven't created a BuildConfig yet, this will not return any resources.

3. View the OpenShift project that is currently in use.

```
oc project
```

This project is specific to you and provides isolation within the cluster so that you can deploy your own applications.

Use the OpenShift web console

In addition to the CLI, OpenShift provides an intuitive web console. This is a useful and powerful feature because it enables you to deploy applications, view resources, monitor applications and view logs, and much more right in the console.

Let's open up the console and have a look around.

1. To open openshift web console, click on the Skills Network button on the right, it will open the **Skills Network Toolbox** Then click the **Cloud** then **Open OpenShift console** as shown in the following image.

It can take a few minutes to become available after opening the lab environment, so if you get an error, wait a minute and try again.

2. The console should open to the project details for the project you have been assigned. Take a look at all the information OpenShift provides you in an intuitive, visual manner. Click through the Dashboard, Overview, and other tabs for this project to see additional information. You should see inventory on the resources that currently exist in this project, the YAML that defines this project, and much more.

3. Familiarize yourself with the items in the left navigation menu. You can see Operators, many different Kubernetes objects, and some OpenShift-specific objects, all of which we have talked about in this course. There won't yet be many instances of these objects, but they will fill up once we deploy our application.

4. Notice the word **“Administrator”** at the top left. This indicates that you are in the Administrator perspective. There is also a Developer perspective. Each perspective provides workflows specific to that persona. **Switch to the Developer perspective** to begin deploying an application. (If it says “Developer” already, don't change it.)

Deploy an application in the web console

The Developer perspective provides workflows specific to developer use cases, such as the ability to create and deploy applications. Let's start here! You are likely in the “Topology” view, which provides a visual

representation of applications. If not, switch to it to take a look.

1. Let us add a new application to this project. There are several ways to add a new application in Openshift.

2. Click the **+Add** button to add a new application.

3. Select **Git Repository (Import from Git)** among the options.

4. You will be redirected to **Import from Git** window. OpenShift will deploy an application using only one input from you: the application source.

5. In the **Git Repo URL** box, paste the sample one mentioned below.

   ```
   https://github.com/sclorg/nodejs-ex.git
   ```

   Note: Click on the Edit Import Strategy to see the various builder images. We shall be using the Node.js image for our application. Ensure that this image has been selected.

6. Keep the rest of the default options as they already are. Then scroll down and click **Create**.

In the Topology view, you should now see your newly created application.

NOTE: It will take several minutes for the application to appear. Refresh the browser if within 3 minutes, you don't see any application.

Note: Please wait for the first build to run successfully. You may temporarily see **ImagePullBackOff** and **ErrImagePull** errors while waiting.

View application in the web console

The Topology view provides quick links to a lot of important parts of an application:

- The outer circle gets the information on the application.
- The inner circle with the Node.js logo gives information about the Deployment.
- The GitHub icon is used to access the code repository.
- The check mark shows the most recent build (you will see circular arrows if the build is in progress).
- The arrow coming out of a box can be used to view the application in the browser if the application is externally available.

Let's try some specific steps:

1. Click the inner circle with the Node.js logo to bring up information on the Deployment and observe the four resources associated with this Deployment: a Pod that runs the containerized application; a Build

that uses the s2i strategy to build the application into a container image; a Service that exposes the application as a network service; and a Route that provides an externally reachable hostname.

Note: Please wait for status of the pod to change to 'Running' and for the Build to complete.

2. Click **View logs** on the line that says **Build #1**.

3. Read the logs to see a few key completed steps. The repository is cloned, a Dockerfile is generated, an image is built, and the image is pushed to the internal registry.

4. Click the **Details** tab for this Build.

5. And then click the link under **Owner** (at the very bottom) that says BC (Build Config).

6. If you look at the **Details** and **YAML** tabs, you'll see many concepts that we talked about in this module: triggers, build strategy, webhooks, and more.

7. On the **Details** tab, click the link under **Output To** that says IST (ImageStreamTag).

8. You can now see the ImageStreamTag that was created as an output of the build. Click the **History** tab to see the image in the internal registry to which this ImageStreamTag points.

9. Return to the Topology view and click on your Deployment info. Click the Route that OpenShift automatically created for you. This will open the application in the browser.

   Note: Please note down this URL as it will be used in the next section

Autoscaling the `nodejs-ex-git` application

Now that the `nodejs-ex-git` app is successfully up and running, let's set up a horizontal pod autoscaler (HPA) so that it can handle any load that comes its way. Make sure to keep the `nodejs-ex-git` app open in a browser tab so that it continues to make requests and consume resources so that it can be successfully autoscaled.

First, we need to set resource requests and limits for the containers that will run. If a container requests a resource like CPU or memory, Kubernetes will only schedule it on a node that can give it that resource. On the other hand, limits prevent a container from consuming more than a certain amount of a resource.

In this case, we're going to request 3 millicores of CPU and 40 MB of RAM. We'll limit the containers to 30 millicores and 100 MB. These numbers are contrived in order to ensure that the app scales.

1. From the Topology view, click the `nodejs-ex-git` Deployment. Then click Actions > Edit Deployment.

2. In the template.spec.containers section, find resources: {}. Replace that with the following text. Make sure the spacing is correct as YAML uses strict indentation.

```
resources:
  limits:
    cpu: 30m
    memory: 100Mi
  requests:
    cpu: 3m
```

```
            memory: 40Mi
```

3. Click Save.

4. Switch to the Administrator perspective.

5. Select Workloads > Horizontal Pod Autoscalers

6. Click Create Horizontal Pod Autoscaler

7. Paste the following YAML into the editor

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: nodejs-ex-git-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nodejs-ex-git
  minReplicas: 1
  maxReplicas: 3
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 10
```

This HPA indicates that we're going to scale based on CPU usage. Generally you want to scale when your CPU utilization is in the 50-90% range. For this example, we're going to use 10% so that the app is more likely to need scaling. The minReplicas and maxReplicas fields indicate that the Deployment should have between one and three replicas at any given time depending on load.

8. Click Create

9. Run the below command on the terminal in Theia to increase the load on the `nodejs-ex-git` and view the Autoscaling:

   ```
   for i in $(seq 1000); do curl -s -L <your app URL> & done
   ```

   Note: Replace `<your app URL>` with the URL that you obtained in Step 9 of the previous section.

The command will keep giving an output as below indicating successful load generation:

> Note: You can also verify autoscaling by directly executing your app URL in your browser. Add some fruit names along with their quantities on the application UI, and click 'Save'. Although the added fruits may not appear on the UI, this action will trigger a load change, causing the pods to autoscale to 3 after some time.

10. Click on `nodejs-ex-git` under `Scale Target`.

11. If you wait, you'll see both Current Replicas and Desired Replicas become three. This is because the HPA detected sufficient load to trigger a scale up to the maximum number of Pods, which is three. You can also view the Last Scale Time as well as the current and target CPU utilization. The target is obviously 1% since that's what we set it to. Note that it can take a few minutes to trigger the scale up.

Wow! OpenShift did some pretty incredible work on your behalf. All it needed was a code repository and it was able to build the code into a container image, push that image to a registry, create a Deployment that references that image, and also expose the application to the internet with a hostname.

Congratulations! You have completed the lab for the fourth module of this course.