Getting Started with OpenShift for Developers Demo -- Script

This accompanies the video here.

Script

00:00

This demo will give you an introduction to OpenShift from the perspective of a developer. Throughout this demo we'll be deploying a parks map application. This consists of a frontend which is a map visualization tool, and two backend services which supply geo-spatial data to the application.

00:19

There are multiple ways that you can interact with an OpenShift cluster. There is a web console like you see here with both an Administrator and Developer Perspective. We'll spend a lot of time here in the demo.

switch to terminal window

You can also interact with OpenShift through the command line. Since OpenShift is a Kubernetes distribution, you can use the kubectl command line tool, or you can use oc, which is the OpenShift command line tool. The syntax between kubectl and oc are the same; they behave the same, except oc adds a few additional commands for your convenience. Let's check to see that we're authenticated to our cluster.

```
oc whoami
```

Okay, so we're signed in as user6.

01:05

Switch to web console, admin perspective

The first step in deploying our application is to deploy the parks map frontend, and we'll do that by deploying an existing container image. Let's toggle to the Developer Perspective.

Toggle to Developer

We land here in Topology view, which is empty because there are no workloads deployed yet in this project. You can see that there are 6 different ways that you can deploy applications, services, or components. Since we have an existing container image, we'll click this one.

Click Container Image

I'll paste in our image name (*docker.io/openshiftroadshow/parksmap:1.3.0*) and then I can either hit enter or click the magnifying glass. This will pull in the metadata for our image. Then, give our application a name (*workshop*) our component is named *parksmap.* You can choose either a standard Kubernetes Deployment or an OpenShift DeploymentConfig. We'll choose *DeploymentConfig*, since we're using some of the features it includes.

This box here (*Create a route to the application*) is checked by default. This will create a Route for our application, which will allow us to access it outside the cluster with a public URL.

You can add additional advanced features here. We'll add a few *Labels* because this application uses labels for discovering backend services, which we'll talk about later. So we have:

app=workshop
component=parksmap
role=frontend

Now click Create.

02:43

So you can see now in Topology view, we have this gray circle indicating our workshop application and then this circle here indicating our parksmap DeploymentConfig. You saw this circle turn from light to darker blue, indicating that the pod is up and running.

Click DeploymentConfig to open side panel.

When we click this you can see that the DeploymentConfig has several resources. It has a pod, a service, and a route. If we click the Overview tab here, you can see other information about that DeploymentConfig.

Now to show you what the application looks like, we can click this here. This indicates the route that we created. You could also get to that route from here (*Resources tab in the side panel for the DeploymentConfig)*.

So we'll open that up (*click the route icon*) and you can see that it's an empty map visualization.

03:30

Next, let's take a look at some of the self-healing capabilities of Kubernetes.

Click the Overview tab of the DeploymentConfig side panel

Right now we have one pod running. If we wanted to scale that up to 2 pods, there's a couple ways we could do that. One would be simply to click here on the up arrow. I actually accidentally clicked it twice, scaling it up to 3. That allows me to show you how to do this on the command line.

Switch to terminal window

So we type `oc scale --replicas=2 dc/parksmap`. So this will scale us to two recplicas, which is what we intended.

Switch to web console

You can see it's happening right here.

Switch to terminal window

But what I want to show you is on the command line. So to simulate something going wrong with one of the pods, what we'll do is kill one of them and watch what happens.

```
oc get pods
```

So I am going to grab this pod here (*copy a pod name*).

```
oc delete pod <that pod name>  && oc get pods
```

So that pod is marked for deletion, and once that completes you should be able to see a new pod running in its place. So now you can see that this pod we had before is still there but now we have this new one in place of this old one that we deleted. So it brought that back up for us without having to take any additional action.

Switch to web console

Now we're going to scale back down to 1 before we move on by clicking the down arrow.

05:11

When you have an application running, you may want to look at the logs for your pod. We can do that here, from the Resources tab on the DeploymentConfig side panel by clicking on this *View Logs* link to view the logs. And you can see in this window here, a streaming view into the logs for your application. If you scroll here, you'll notice that there are some errors in our logs. We'll take some actions to handle that in an upcoming step in the demo.

Switch to terminal window

You can also get to the logs from the command line. Type `oc get pods` again and then `oc logs -f <pod name>` and you'll get a view of your logs that way as well.

Switch to web console

Now this is fine if you have just a single pod or if you're not really concerned about your pod restarting, but if you want to look at logs across a larger scaled application, or if you want to be able to view historical information about the logs, then looking at them this way isn't going to be sufficient. So, aggregated logging is built into OpenShift. You can click this *Show in Kibana* link here and it will bring up our Kibana dashboard.

06:31

We log in here with the same credentials that we used for OpenShift. And we have to grant permissions here as well.

So here's our Kibana dashboard. We've already got a query in here, and we have to select our user6 project in the dropdown. The query in here says we want to see pods with this name (our current pod), and in the user6 namespace (which is our project), and then for the container named parksmap. We got this context when we came over here because we clicked the link from our pod logs. So it brings us right into the context that we want to look at.

If we were to search here for the error we saw as well append AND message:"Failure executing", you'll see these error messages we saw in our logs. These are related to a service account and we'll fix them in a later step.

You can also get to these logs from the Administrator view by going to Monitoring > Logging.

07:54

So let's take care of that error in reference to a service account. OpenShift creates a few special service accounts for every project, and the default service account is the one that takes the

responsibility for running pods. By changing the permissions for that service account we can fix this application and allow our application to do what we need it to do.

Go back to the Developer Perspective. Click the parksmap DeploymentConfig and then the Overview tab. Then click the user6 namespace. From there, click Role Bindings. Click Create Binding, give it a name of view, select a Role Name of view. Select Service Account, select user6 for the Subject Namespace, and then enter default for the Subject Name. Then click Create.

Now if you click back to Role Bindings, you see the new role binding with the name view for the Service account called default. You can also add those permissions via the command line if you prefer to do it that way.

Now we need to redeploy our application so it can take advantage of those changes. In Topology view, click the parksmap DeploymentConfig, the from the Actions dropdown select Start Rollout and that will trigger a new deployment. You can see that the latest version is 2 now.

You can also grant access to other users to access your project. One way to do that is to click Project Access in the left navigation. If there was another user, for example user1, I could give them access to our project as well: view, edit, or admin access. Grant view access to user one and then click Save.

10:18

Next we're going to deploy our national parks backend. We're going to deploy this one from a git repository using the source-to-image project, which takes source code and a builder image and creates your application image.

Navigate to the Gogs URL for this user (you can find it in the workshop content).

We have a Gogs git repository for this national parks app. Copy the URL. Then in Topology view, click the +Add entry in the left navigation. Click From Git, paste in the URL. Select Java, version 8.

The application will be named workshop. Name the component nationalparks. Select DeploymentConfig, and we do want a route to the application. As we did before, we're going to apply some labels: app=workshop component=nationalparks role=backend. Then click Create.

So what's different here, is that you can see this icon for a build. Our build is running. If you click on this you can see the build logs. Once the build is complete you'll see the build indicator turn to a green check mark to indicate the build was successful.

You can also view build logs from the command line.

Switch to terminal window

oc get builds
oc logs -f builds/nationalparks-1

This will get us the logs for that build. This is a Java based application that uses Maven, so the initial build can take a couple minutes.

Switch to web console

The build is complete now and we have a blue ring indicating that the pod is up and running. Click the route indicator. This is a backend application, so it's not meant to be accessed at this URL, but there is an endpoint /ws/info/ that gives us some information about the application.

13:07

We're going to have a MongoDB database that has location data. Let's create that database now.

Once again go to the +Add menu. This time use the Database option. In the developer catalog for databases, select MongoDB (Ephemeral). We don't need persistent storage for this demo.

Enter *mongodb-nationalparks* for Database Service Name, for all the other username and password fields, enter *mongodb*. If we left those fields empty, OpenShift would autogenerate values for us. Click Create.

While that's coming up, click the secret in the Parameters section. It was created for us and we can use it for authentication with the database. Click Add Secret to Workload and then select *nationalparks* . Then click Save. That secret will be added as environment variables. Adding the secret to the workload triggers a new deployment.

If we want to add mongodb-nationalparks to our workshop application grouping, you can right-click on it, choose Edit Application Grouping and select workshop.

When we deployed this database we weren't presented with an option in the template to add the labels that we added to our other components. Let's do that via the command line now.

Switch to terminal window

oc label dc/mongodb-nationalparks svc/mongodb-nationalparks app=workshop component=nationalparks role=database --overwrite

This adds the labels to our mongodb-nationalparks DeploymentConfig and Service.

Switch to browser

Now, in the browser if you go to the backend service and go to the /ws/data/all endpoint you'll see an empty set of data. Then go to /ws/data/load and it will load data in the DB. Then go back to /ws/data/all you will see all the data.

The way this parksmap app is set up, it's using the OpenShift API and querying for routes and services in the project. If they have the label type=parksmap-backend, then the application will know to talk to those endpoints for map data. So now we need to label our route with type=parksmap-backend.

Switch to terminal window

oc label route nationalparks type=parksmap-backend

The way we're doing this isn't a requirement, but we're doing it this way to show you that you can use this kind of discovery for backend services.

Switch to browser

If you now refresh the parksmap frontend in your browser, we should see all the data showing up on the map.

16:55

Now we're going to take a look at application health checks via readiness probes and liveness probes. A liveness probe will check if the container for which it's configured is still running, and a readiness probe will check if the container is ready to service requests. Click the nationalparks DeploymentConfig and go to Actions > Edit Deployment Config. In a future version of OpenShift, you'll be able to do this via a form, rather than editing the YAML directly. Insert the following YAML after imagePullPolicy:

(can pull from the Application Health checks section of the guide):

```
readinessProbe:
 httpGet:
```

```
    path: /ws/healthz/
    port: 8080
    scheme: HTTP
  initialDelaySeconds: 20
  timeoutSeconds: 1
livenessProbe:
  httpGet:
    path: /ws/healthz/
    port: 8080
    scheme: HTTP
  initialDelaySeconds: 120
  timeoutSeconds: 1
```

Click Save. That should trigger a new deployment.

18:00

Now let's set up a pipeline in OpenShift to take care of our application lifecycle. There are many ways of doing this. We'll use a Jenkins pipeline and set it up so that the Jenkins pipeline will control when the builds and deployments happen rather than the triggers we have now for configuration changes and image changes.

We will remove some of those triggers. Click the nationalparks DeploymentConfig again and go to Actions > Edit Deployment Config. Remove the triggers section completely. Save.

Now go back to the Git repository in Gogs. We'll create a Jenkinsfile. Sign into Gogs. Create a new file called Jenkinsfile.workshop.

Paste in the code for Jenkinsfile.workshop in the workshop guide section for Automating Build and Deployment with Pipelines.

Commit.

Create the pipeline next. Go to the +Add menu. Select From Catalog and search for Jenkins. Click it and then click Instantiate Template. Keep the default settings, except set Disable memory intensive administrative monitors to *true* just to speed things up for the demo.

Back in Topology view, right click on the jenkins service and choose Edit Application Grouping and add it to the workshop application. Wait for the jenkins service to come up.

Next click +Add and then YAML.

Paste in the YAML from the workshop guide section for Automating Build and Deployment with Pipelines.

Click Create. This will create a pipeline that uses the Jenkinsfile.workshop from the repository.

You'll notice there's a deprecation message here referring to OpenShift Pipelines. OpenShift PIpelines is currently in tech preview and is based on Tekton to allow you to do more cloud-native CI/CD. But for this demo we're using the Jenkins template.

Go to the Builds link in the left navigation, and click on nationalparks-build, the builds, then nationalparks-build-1. Click View logs and log into Jenkins with your OpenShift credentials.

You can see the logs there as it progresses.

Go back to the web console

In Topology view, click on the nationalparks DeploymentConfig and you can see that the version has incremented.

21:50

Next we're going to configure some webhooks to trigger the pipeline execution every time there's a change to our Git repository. So go back to the Builds menu. Click nationalparks-build and scroll to the webhooks section. And click "Copy URL with Secret" next to the Generic webhook.

Then go to Gogs. Click Settings > Webhooks. Under Add Webhook, select Gogs. Paste in the webhook URL, switch the Content Type to application/x-www-form-urlencoded and click Add Webhook.

Back in the code, go to src/main/java/com/openshift/evg/roadshow/parks/rest and click into BackendController.java. Click Edit and change "National Parks" to "Amazing National Parks" and commit that change.

Switch to web console

Click into Builds from nationalparks-build, and you'll see a new build, nationalparks-build-2.

Switch to terminal window

Now let's type oc get dc so you can get all the DeploymentConfigs. Our nationalparks deployment is at revision 5. Let's roll back to revision 4.

oc rollback nationalparks

You'll get a message that it was rolled back to nationalparks-4.

You can also roll forward. If we want to roll forward to nationalparks-5:

oc rollback nationalparks-5

24:35

In the next section of the demo, we're going to deploy a backend service that consists of a REST API and MongoDB but this time the application will already be wired together. We're going to use a template for this.

If you want to take a look at the template (https://raw.githubusercontent.com/openshift-roadshow/mlbparks/master/ose3/application-template-eap.json) you can see all the information that's needed to generate and deploy this application.

Switch to terminal window

From the command line you can run
oc create -f
https://raw.githubusercontent.com/openshift-roadshow/mlbparks/master/ose3/application-template-eap.json

You get a message back that the mlbparks template is created. From here there are 2 ways you can instantiate the template. You can do it from the command line with oc new-app or you can do it in the web console.

Switch to web console

Click +Add and then From Catalog and search for mlb. Click it. Instantiate Template and in the form you'll see all the parameters that can be configured. For application name, enter mlbparks. Then click Create.

In Topology view you'll see both mlbparks and mongodb-mlbparks. Right click and choose Edit Application grouping to move them to the workshop application grouping if needed.

When we deployed the nationalparks backend, we saw how s2i helps you get from source code to a running container. However for fast iterative development that may not be the best option.

For example if you wanted to change some CSS or a small change before going through the commit cycle, you may want to do it a different way. There are a few ways to address iterative development. One way is to use a command line tool called odo. But for this example we'll show you how to do this another way.

From the oc CLI tool, you can trigger a deployment from your local machine. In this case we'll use s2i again but tell OpenShift to just use the .war file from our local machine and put that in the image. This lets us do quick builds on our local machine and then quickly send up the .war file.

Switch to terminal window

git clone https://github.com/openshift-roadshow/mlbparks.git
cd mlbparks
mvn package

Look at the location of the ROOT.war file.

Go in and make a code change in src/main/java/com/openshift/evg/roadshow/rest/
Edit BackendController.java. Change "MLB Parks" to "Amazing MLB Parks" and save.

Go back up to the mlbparks directory, and run mvn package again. Now we want to kick off a build using our new war file.

oc start-build bc/mlbparks --from-file=target/ROOT.war --follow

Let's watch from the web console

Switch to web console.

Click the mlbparks Deployment Config and see that the build is running. Once that's done, click the Route icon. Append /ws/info/ if needed and you'll see the "Amazing MLB Parks" message.

That's one way to speed up your build and deploy process. Remember that the odo command line tool is also very helpful for fast iterative development.

That completes this demo walking you through OpenShift from the perspective of a developer.