**1. Overview**

In this tutorial, we'll show how to deploy an application from our [Bootstrap a Simple Application using Spring Boot](https://www.baeldung.com/spring-boot-start) tutorial to [Openshift](https://www.openshift.com/).

As part of this we'll:

* Install and configure Openshift development tools.
* Create an Openshift project and MySQL deployment.
* Configure the application for [**Spring Cloud Kubernetes**](https://github.com/spring-cloud/spring-cloud-kubernetes).
* Create and deploy the application in a container using the [**Fabric8 Maven plugin**](https://github.com/fabric8io/fabric8-maven-plugin) and test and scale the application.

**2. Openshift Configuration**

First, **we need to**[**install Minishift**](https://docs.okd.io/latest/minishift/getting-started/installing.html)**, the local single-node Openshift cluster, and the**[**Openshift client**](https://docs.okd.io/latest/cli_reference/get_started_cli.html#installing-the-cli).

Before using Minishift we need to configure permissions for the developer user:

|  |  |
| --- | --- |
| 1  2  3  4 | minishift addons install --defaults  minishift addons enable admin-user  minishift start  oc adm policy --as system:admin add-cluster-role-to-user cluster-admin developer |

Now we want to use the Openshift Console to create a MySQL service. We can launch the browser URL using:

|  |  |
| --- | --- |
| 1 | minishift console |

If you are not logged in automatically, then use *developer/developer.*

Create a project named *baeldung-demo* and then create a MySQL database service from the catalog. Provide *baeldung-db* for the Database Service, *baeldung\_db*for the MySQL Database Name, and leave the other values at their defaults.

We now have a service and secrets for access to the database. Take note of the database connection url: *mysql://baeldung-db:3306/baeldung\_db*

We also need to allow applications to read configuration like Kubernetes Secrets and ConfigMaps:

|  |  |
| --- | --- |
| 1  2 | oc create rolebinding default-view --clusterrole=view \    --serviceaccount=baeldung-demo:default --namespace=baeldung-demo |

**3. Spring Cloud Kubernetes Dependencies**

**We'll use the**[**Spring Cloud Kubernetes**](https://github.com/spring-cloud/spring-cloud-kubernetes)**project to enable the cloud-native APIs for Kubernetes that underpin Openshift:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | <profile>    <id>openshift</id>    <dependencyManagement>      <dependencies>        <dependency>          <groupId>org.springframework.cloud</groupId>          <artifactId>spring-cloud-kubernetes-dependencies</artifactId>          <version>0.3.0.RELEASE</version>          <type>pom</type>          <scope>import</scope>        </dependency>        <dependency>          <groupId>org.springframework.cloud</groupId>          <artifactId>spring-cloud-dependencies</artifactId>          <version>Finchley.SR2</version>          <type>pom</type>          <scope>import</scope>        </dependency>      </dependencies>    </dependencyManagement>      <dependencies>      <dependency>        <groupId>org.springframework.cloud</groupId>        <artifactId>spring-cloud-starter-kubernetes-config</artifactId>      </dependency>      <dependency>        <groupId>org.springframework.boot</groupId>        <artifactId>spring-boot-starter-actuator</artifactId>      </dependency>    </dependencies>  </profile> |

**We'll also use the**[**Fabric8 Maven plugin**](https://github.com/fabric8io/fabric8-maven-plugin)**to build and deploy the container:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <plugin>      <groupId>io.fabric8</groupId>      <artifactId>fabric8-maven-plugin</artifactId>      <version>3.5.37</version>      <executions>        <execution>          <id>fmp</id>          <goals>            <goal>resource</goal>            <goal>build</goal>          </goals>        </execution>      </executions>  </plugin> |

**4. Application Configuration**

Now **we need to provide configuration to ensure the correct Spring Profiles and Kubernetes Secrets are injected as environmental variables**.

**Let's create a YAML fragment in *src/main/fabric8*** so that the Fabric8 Maven plugin will use it when creating the deployment configuration.

We also need to add a section for the Spring Boot actuator since the default in Fabric8 still tries to access */health* instead of */actuator/health:*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | spec:    template:      spec:        containers:        - env:          - name: SPRING\_PROFILES\_ACTIVE            value: mysql          - name: SPRING\_DATASOURCE\_USER            valueFrom:              secretKeyRef:                name: baeldung-db                key: database-user          - name: SPRING\_DATASOURCE\_PASSWORD            valueFrom:              secretKeyRef:                name: baeldung-db                key: database-password          livenessProbe:            httpGet:              path: /actuator/health              port: 8080              scheme: HTTP            initialDelaySeconds: 180          readinessProbe:            httpGet:              path: /actuator/health              port: 8080              scheme: HTTP            initialDelaySeconds: 30 |

Next, **we'll save a *ConfigMap* in *openshift/configmap.yml****,* this contains the data for an *application.properties* with the MySQL URL:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | apiVersion: v1  kind: ConfigMap  metadata:    name: spring-boot-bootstrap  data:    application.properties: |-      spring.datasource.url=jdbc:<mysql://baeldung-db:3306/baeldung_db> |

**Before interacting with Openshift using the command line client we need to log in**. In the top right of the web console is a user icon from which we can select the drop-down menu labeled “Copy Login Command”. Then use in a shell:

|  |  |
| --- | --- |
| 1 | oc login https://192.168.42.122:8443 --token=<some-token> |

Let's make sure we're using the correct project:

|  |  |
| --- | --- |
| 1 | oc project baeldung-demo |

And then**we upload the *ConfigMap****:*

|  |  |
| --- | --- |
| 1 | oc create -f openshift/configmap.yml |

**5. Deployment**

During deployment, the Fabric8 Maven plugin tries to determine the configured port. The existing *application.properties* file in our sample application uses an expression to define the port, which the plugin is unable to parse. Therefore, we have to comment the line:

|  |  |
| --- | --- |
| 1 | #server.port=${port:8080} |

from the current *application.properties*.

We are now ready for deployment:

|  |  |
| --- | --- |
| 1 | mvn clean fabric8:deploy -P openshift |

We can watch the deployment progress until we see our application running:

|  |  |
| --- | --- |
| 1 | oc get pods -w |

Should provide a listing:

|  |  |
| --- | --- |
| 1  2  3 | NAME                            READY     STATUS    RESTARTS   AGE  baeldung-db-1-9m2cr             1/1       Running   1           1h  spring-boot-bootstrap-1-x6wj5   1/1       Running   0          46s |

Before we test the application, we need to determine the route:

|  |  |
| --- | --- |
| 1 | oc get routes |

Will print the routes in the current project:

|  |  |
| --- | --- |
| 1  2 | NAME                    HOST/PORT                                                   PATH      SERVICES                PORT      TERMINATION   WILDCARD  spring-boot-bootstrap   spring-boot-bootstrap-baeldung-demo.192.168.42.122.nip.io             spring-boot-bootstrap   8080                    None |

Now, let's verify that our application is working by adding a book:

|  |  |
| --- | --- |
| 1  2 | http POST http://spring-boot-bootstrap-baeldung-demo.192.168.42.122.nip.io/api/books \    title="The Player of Games" author="Iain M. Banks" |

Expecting the following output:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | HTTP/1.1 201  {      "author": "Iain M. Banks",      "id": 1,      "title": "The Player of Games"  } |

**6. Scaling the Application**

Let's scale the deployment to run 2 instances:

|  |  |
| --- | --- |
| 1 | oc scale --replicas=2 dc spring-boot-bootstrap |

And we can use the same steps as earlier, then, to watch it deploy, get the routes, and test the endpoint.

Openshift provides for a wide range of options for [managing performance and scaling](https://docs.openshift.com/container-platform/3.11/scaling_performance/index.html) beyond the scope of this article.

**7. Conclusion**

In this tutorial, we:

* Installed and configured the Openshift development tools and local environment
* Deployed a MySQL service
* Created a ConfigMap and Deployment configuration to provide database connection properties
* Built and deployed a container for our configured Spring Boot application, and
* Tested and scaled the application.

For more details, check out [the detailed Openshift documentation](http://docs.openshift.com/container-platform/3.11/welcome/index.html).

The complete source code of our examples here is, as always, [over on GitHub](https://github.com/eugenp/tutorials/tree/master/spring-boot-bootstrap).