



Lab 8

Deploying to and Running Spring Boot Applications on Oracle Cloud

This lab will show you how you can deploy your Spring Boot application to and run it on the Oracle Cloud. There are several cloud services that we can leverage for running a Spring Boot application:

- Container Cloud Service – for running anything in a Docker Container, including a Spring Boot application (as discussed in lab 4)
- Application Container Cloud – for running an application in a prepared runtime (in our case Java; other runtimes available for ACC include Node, Ruby, Python, Go and PHP)
- Other options include IaaS Compute, Java Cloud Service (based on WebLogic) and the upcoming Container Engine Cloud (Kubernetes); the Fn Platform for serverless functions could provide another deployment target – which could be either cloud based or running on premises

In this lab we will cover the Application Container Cloud option.

Note: you can either use your own cloud environment – for example acquired through the \$300 trial program at cloud.oracle.com – or one that has been prepared for this workshop. In the latter case, ask the instructor for the credentials for the cloud environment.

You will need these details:

- Cloud Identity Domain
- Region
- Username
- Password

1. Starting point

The starting point for this lab is the service that was created in Lab 1 – the DroneBuzzers API. The code of this service is located in directory `lab 8/dronebuzzers`.



Spring Boot 101 - intro, demo & hands-on with Java, REST APIs, Containers & Cloud

The `lab 8/input` directory contains supporting files that are required for the deployment to ACCS.

Remember to run the `git pull` command so you have the latest version of the labs.

1. Create the ACC deployment package

The package that we deploy to the Application Container Cloud is an archive (zip or tar) that contains the application's jar file and one or two meta-data files that describe the application, how it should be deployed and started and how it should be scaled.

manifest.json

The file `manifest.json` in the `lab 8/input` directory is one such meta-data file. It specifies the runtime for the application – Java 8 – and the command used to run the application.

From the Spring documentation:

By default, SpringApplication converts any command line option arguments (that is, arguments starting with `--`, such as `--server.port=9000`) to a property and adds them to the Spring Environment. As mentioned previously, command line properties always take precedence over other property sources.

In order to have Spring listen at the port that is available (for example through the Application Container Cloud runtime environment) we have to make sure we define that port at the command line through the argument `--server.port`. This corresponds with the content of the command entry in the `manifest.json` file:

```
{ manifest.json x
1  {
2    "runtime": {
3      "majorVersion": "8"
4    },
5    "command": "java -jar app.jar --server.port=$PORT --server.address=$HOSTNAME",
6    "startupTime": "120",
7    "release": {
8      "build": "n/a",
9      "version": "1.0.0"
10   },
11   "notes": "DroneBuzzers REST API on Oracle Application Container Cloud - Java SE "
12 }
```

The reference `$PORT` is to an environment variable that ACCS sets at runtime.

The `server.address` command line parameter that takes its value from another runtime environment variable `$HOSTNAME` is passed to Spring as well at startup time.



Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud

Copy the manifest.json from directory lab 8/input to lab 8/dronebuzzers/src/main/resources – the directory that also contains the application.properties file.

deployment.json

The second meta-data file is called deployment.json. This file is optional – we will omit it here. The deployment.json file specifies how much memory to allocate to the application, how many application instances to create initially, additional environment variables, and service bindings to other Oracle Cloud services. You can specify these same options from the user interface, or you can upload this file using the REST API. If no values are specified or the file is omitted, memory and instance defaults are used.

2. Create a Deployable Distribution for the Spring Boot Application

The package we will deploy to the Application Container Cloud contains the manifest.json file as well as the JAR file for the Spring Boot application – that must contain all its dependencies!

To create that distribution package – and the JAR file with all dependencies for the Spring Boot application - we make use of Maven. The pom.xml file in directory lab 8/dronebuzzers describes the application and can be run to build a simple JAR file, using:

mvn package

```
2018-03-27 15:00:20.656 INFO 74916 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**/favicon.ico] onto handler of type [c
ework.web.servlet.resource.ResourceHttpRequestHandler]
2018-03-27 15:00:20.918 INFO 74916 --- [main] c.d.rest.DronebuzzersApplicationTests : Started DronebuzzersApplicationTests in 2.85 seconds (JVM
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.028 sec - in com.dronebuzzers.rest.DronebuzzersApplicationTests
2018-03-27 15:00:20.970 INFO 74916 --- [Thread-2] o.s.w.c.s.GenericWebApplicationContext : Closing org.springframework.web.context.support.GenericWe
@36b4fe2a: startup date [Tue Mar 27 15:00:18 CEST 2018]; root of context hierarchy

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:2.6:jar (default-jar) @ dronebuzzers-rest-service ---
[INFO] Building jar: lab 8\dronebuzzers\target\dronebuzzers-rest-service-1.0.0.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:1.5.10.RELEASE:repackage (default) @ dronebuzzers-rest-service ---
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 7.229 s
[INFO] Finished at: 2018-03-27T15:00:22+02:00
[INFO] Final Memory: 20M/375M
[INFO]
```

This will run for a little while before completing with the BUILD SUCCESS message. A few lines earlier, Maven will have reported the creation of a jar file. This file is located in directory lab 8/dronebuzzers/target and is called (after the artifactId and version specified in the pom.xml file) dronebuzzers-rest-service-1.0.0.jar.

However: **this file does not contain the dependencies that are required for ACCS.**

So, we will add a special Maven plugin to create the JAR that can be deployed to ACCS.



Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud

Add the contents of the file lab 8/input/add-to-pom.xml to the plugins sections of the pom.xml file in lab 8/dronebuzzers.

Look at the plugin you have added: it refers to a file called deploy.xml. This file in turn determines what kind of end result Maven should bake. In this case, we want a final zip-file that contains the self contained JAR file as well as the manifest.json file. Copy file deploy.xml from lab 8/input to lab 8/dronebuzzers. Check the contents of this file:

```
deploy.xml ✕
1  <assembly xmlns="http://maven.apache.org/plugins/maven-assembly-plugin/assembly/1.1.3"
2      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xsi:schemaLocation="http://maven.apache.org/plugins/maven-assembly-plugin/assembly/1.1.3
4      <id>distribution</id>
5      <formats>
6          <format>zip</format>
7          <format>tar.gz</format>
8      </formats>
9      <includeBaseDirectory>false</includeBaseDirectory>
10     <files>
11         <file>
12             <source>${project.build.directory}/${artifactId}-${version}.${packaging}</source>
13             <outputDirectory></outputDirectory>
14             <destName>app.jar</destName>
15         </file>
16         <file>
17             <source>src/main/resources/manifest.json</source>
18             <outputDirectory></outputDirectory>
19             <destName>manifest.json</destName>
20         </file>
21         <file>
22             <source>src/main/resources/application.properties</source>
23             <outputDirectory></outputDirectory>
24             <destName>application.properties</destName>
25         </file>
26     </files>
27 </assembly>
```

Deploy.xml instructs the Maven Assembly plugin to produce two files in the target directory (lab 8/dronebuzzers/target): a tar.gz and a zip file (dronebuzzersAPI.zip and dronebuzzersAPI.tar.gz). Both files contain:

- the self-contained jar-file (dronebuzzers-rest-service-1.0.0.jar renamed to app.jar)
- manifest.json
- application.properties file.

After making these changes, run again

```
mvn package
```



Spring Boot 101 - intro, demo & hands on with Java, REST APIs, Containers & Cloud

It will now run a for a little bit longer and create the two distribution packages in zip and tar.gz format:

```
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.681 sec - in com.dronebuzzers.rest.DronebuzzersApplicationTests
2018-03-27 17:29:54.470 INFO 30400 --- [Thread-2] o.s.w.c.s.GenericWebApplicationContext : Closing org.springframework.web
@574b560f: startup date [Tue Mar 27 17:29:52 CEST 2018]; root of context hierarchy

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] --- maven-jar-plugin:2.6:jar (default-jar) @ dronebuzzers-rest-service ---
[INFO] Building jar: lab 8\dronebuzzers\target\dronebuzzers-rest-service-1.0.0.jar
[INFO] --- spring-boot-maven-plugin:1.5.10.RELEASE:repackage (default) @ dronebuzzers-rest-service ---
[INFO] --- maven-assembly-plugin:2.6:single (default) @ dronebuzzers-rest-service ---
[INFO] Building zip: lab 8\dronebuzzers\target\dronebuzzersAPI.zip
[INFO] Building tar: lab 8\dronebuzzers\target\dronebuzzersAPI.tar.gz
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 10.064 s
[INFO] Finished at: 2018-03-27T17:29:57+02:00
[INFO] Final Memory: 24M/379M
[INFO] -----
```

Note: these 2 distribution packages can both be used with ACCS.

3. Deploy the Application Archive to ACC through the Service Console

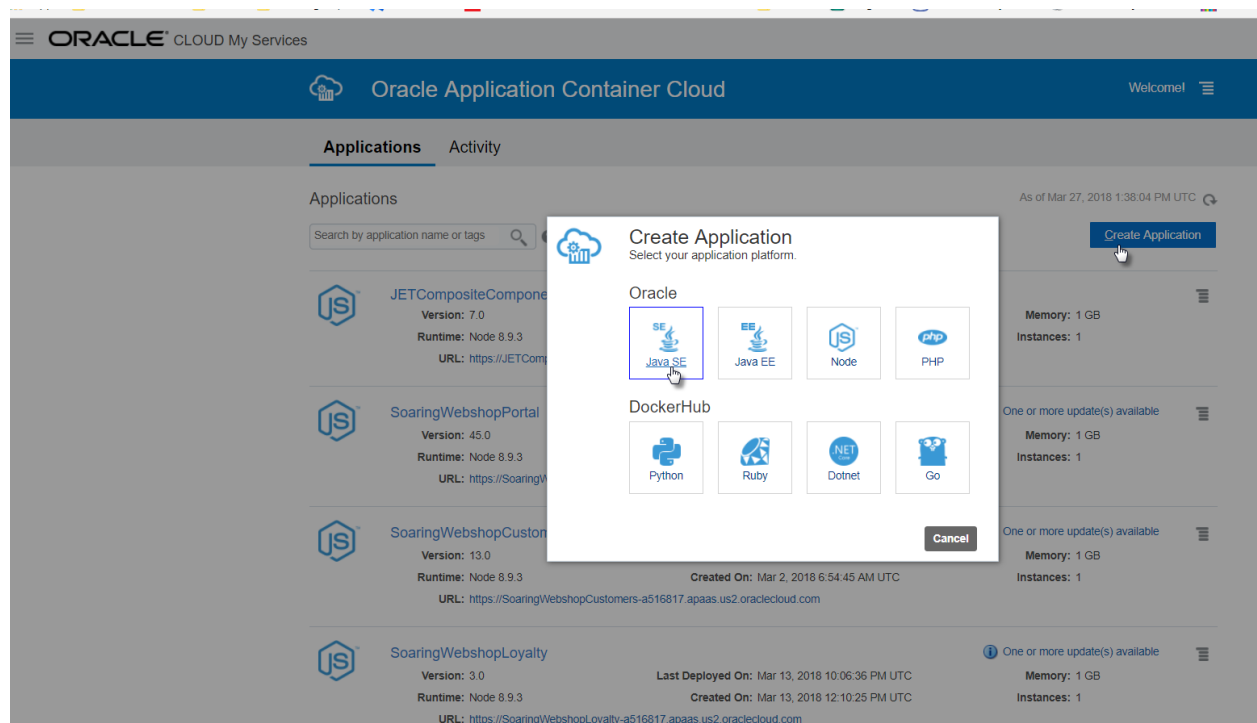
The application can be deployed to ACCS through the PSM command line tool or through the browser based console. You would use PSM in any automated scenario. For now we will just use the user interface to upload and deploy the application to its runtime container on ACCS.

Login to the Oracle Cloud environment and open the Service Console for Application Container Cloud (<https://apaas.us.oraclecloud.com/apaas/faces/aPaaSRunner.jspx> for Region US).

Click on Create Application. Then click on Java SE as the runtime of choice:



Spring Boot 101 - intro, demo & hands on with Java, REST APIs, Containers & Cloud



A popup window is presented. Here you need to specify the name of the application and upload the zip (or tar) file that contains the application (and the manifest.json file). You can change the settings for number of instances and the allocated memory. Let's start small.

A screenshot of the 'Create Application' dialog box in a cloud management console. The dialog has a title bar 'Create Application' and a sub-header 'Application' with a Spring logo. It contains several configuration fields: 'Region' is a dropdown set to 'No Preference'; '* Name' is a text input containing 'DroneBuzzersAP'; '* Application' has two radio buttons, 'Upload Archive' (selected) and 'Deploy Sample'; next to 'Upload Archive' is a 'Choose File' button and a file name 'droneb...st.zip'; 'Instances' is a spinner set to '1'; 'Memory (GB)' is a spinner set to '1'; and 'Application Cache' is a dropdown set to 'Select'. At the bottom left is a 'More Options' link with a right-pointing triangle. At the bottom right are two buttons: a blue 'Create' button and a grey 'Cancel' button. A mouse cursor is clicking on the 'Create' button.


Click on Create to start the deployment process.

Let's wait a little until the uploaded archive is processed and validated:



Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud

Create Application

 Application

Processing Archive

51%

Region: No Preference ?

* Name: DroneBuzzersAPI ?

* Application: ☒ Upload Archive Choose File droneb...st.zip x ?
☐ Deploy Sample ?

Instances: 1 ?

Memory (GB): 1 ?


Application Cache: Select ?


▶ More Options

Create Cancel

And wait a little longer for the actual deployment:

Create Application

 Application

 Creating Application...

Region: No Preference ?

* Name: DroneBuzzersAPI ?

* Application: ☒ Upload Archive Choose File droneb...st.zip x ?

After a little while, the application creation request is being processed for real:



Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud

The screenshot shows the Oracle Application Container Cloud (OACC) dashboard. The top navigation bar is blue with the OACC logo and the text "Oracle Application Container Cloud". On the right, it says "Welcome!" with a menu icon. Below the navigation bar, there are two tabs: "Applications" (selected) and "Activity". The main content area is titled "Applications" and includes a search bar with the placeholder "Search by application name or tags" and a "Create Application" button. A green notification banner states: "Application creation request has been accepted." Below this, there is a table of applications. The first application is "DroneBuzzersAPI", which is in the "Creating Application..." state. It shows details: Version: 1.0.0, Runtime: Java SE 8u151, Last Deployed On: Mar 27, 2018 2:06:08 PM UTC, Submitted On: Mar 27, 2018 2:06:08 PM UTC, Memory: 0GB, and Instances: 0. The second application is "JETCompositeComponentsShowroom".

After 1-3 minutes, the application will be up and running. Unless it is not, and we have to inspect log files to find out what failed and why.

When the application container is fully prepared, the public URL is provided:

This screenshot shows the OACC dashboard after the application is fully deployed. The "DroneBuzzersAPI" application is now in a running state. The details for this application are: Version: 1.0.0, Runtime: Java SE 8u151, Last Deployed On: Mar 27, 2018 2:06:08 PM UTC, Submitted On: Mar 27, 2018 2:06:08 PM UTC, Memory: 1 GB, and Instances: 1. A red box highlights the "URL" field, which contains the value: "https://DroneBuzzersAPI-a516817.apaas.us2.oraclecloud.com". The "JETCompositeComponentsShowroom" application is also visible below it.

After creation is complete, you can check details of the application by clicking on the name. You will information such as this:

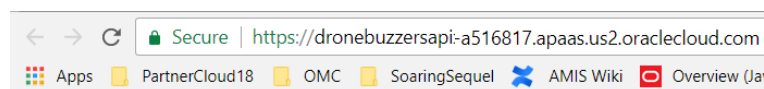


Spring Boot 101 - intro, demo & hands on with Java, REST APIs, Containers & Cloud

4. Testing the DroneBuzzersAPI on the Cloud

Now we can try to access the DroneBuzzersAPI as it is running on the Public Cloud.

If you just click on the URL – or copy and paste it into your browser – you will get a response like this:



Whitelabel Error Page

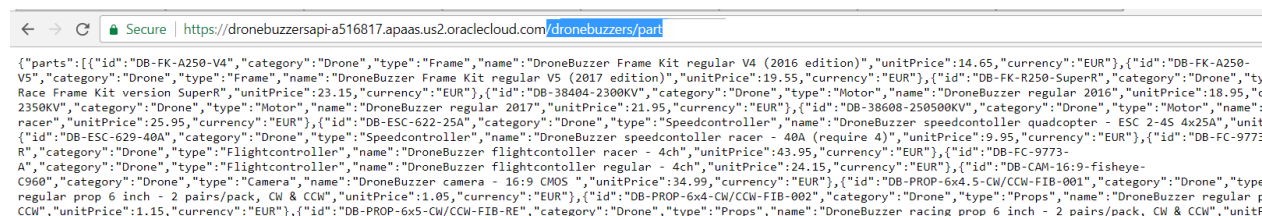
This application has no explicit mapping for /error, so you are seeing this as a fallback

Tue Mar 27 15:39:23 UTC 2018

There was an unexpected error (type=Not Found, status=404).

No message available

However, just by adding /dronebuzzers/part to the URL, you will get a list of all parts:



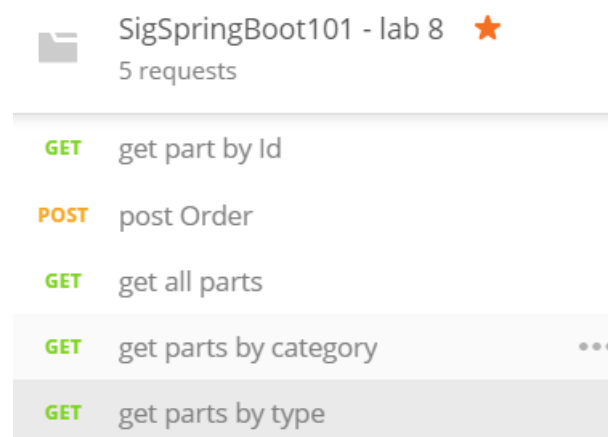


Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud

Now would be a good time to return to the Postman collection we have used for testing the DroneBuzzers API in lab 1. Change the endpoint of the API to the URL defined in the Oracle Container Cloud service, and you should be able to run through all requests.

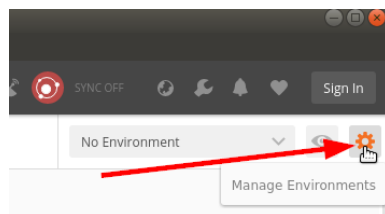
Open Postman and import the file SigSpringBoot101 - lab 8.postman_collection.json from directory lab 8/postman.

You will see five requests – that you cannot actually execute, because they all depend on an environment variable called DRONEBUZZER_API_ENDPOINT.



Create a new environment in Postman and define a key DRONEBUZZER_API_ENDPOINT that should be set to the value of your Application's URL on Application Container Cloud.

Click the Manage Environments button:



Add an environment named SpringBoot101:



Click the environment to edit it

Add a Key named `DRONEBUZZER_API_ENDPOINT`, and give it the Value of the ACCS service endpoint:

Now you can execute the requests and in doing so test the DroneBuzzerAPI, running on ACC:



Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud

The screenshot shows a REST client interface with a GET request to `{{DRONEBUZZER_API_ENDPOINT}}/dronebuzzers/part/type/Motor`. The response status is 200 OK, with a time of 910 ms and a size of 632 B. The response body is displayed in JSON format, showing a list of drone parts.

```
1 {
2   "parts": [
3     {
4       "id": "DB-38404-2300KV",
5       "category": "Drone",
6       "type": "Motor",
7       "name": "DroneBuzzer regular 2016",
8       "unitPrice": 18.95,
9       "currency": "EUR"
10    },
11    {
12      "id": "DB-38406-2350KV",
13      "category": "Drone",
14      "type": "Motor",
15      "name": "DroneBuzzer regular 2017",
16      "unitPrice": 21.95,
17      "currency": "EUR"
18    },
19    {
20      "id": "DB-38608-250500KV",
21      "category": "Drone",
22      "type": "Motor",
23      "name": "DroneBuzzer regular 2018",
24      "unitPrice": 24.95,
25      "currency": "EUR"
26    }
27  ]
28 }
```

You have managed to take the vanilla Spring Boot application created in Lab 1 and deployed it successfully on Application Container Cloud. Crucial are the correct packaging of the Spring Boot application in a self-contained JAR file and the appropriate configuration of the manifest.json file that is the instruction for ACC.

5. Additional pointers

Some resources that may be helpful when looking into combination of SpringBoot and Oracle Application Container Cloud:

- Blog: Carsten Wiesbaum: Spring-Boot and Oracle Application Container Cloud - <http://www.esentri.com/blog/2017/04/12/spring-boot-and-oracle-application-container-cloud/>
- Blog: Carsten Wiesbaum: Deploy to ACC with Maven - <https://www.esentri.com/blog/2017/06/28/deploy-to-oracle-application-container-cloud-using-maven/>
- Spring Boot Documentation - How to Externalize SpringBoot properties (such as port): <https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html>
- Medium Article: Vinay Kumar -Deploying a Spring Boot Microservice to Application Container — Oracle Cloud (<https://medium.com/oracledevs/deploying-a-spring-boot-micro-services-application-to-oracle-application-container-cloud-95238f1b7ab7>)



Spring Boot 101 - intro, demo & handson with Java, REST APIs, Containers & Cloud

- Blog: Maarten Smeets: Application Container Cloud Service (ACCS): Using the Application Cache from a Spring Boot application (<http://javaoraclesoa.blogspot.nl/2018/03/application-container-cloud-service.html>)
- Medium Article: Abhishek Gupta - Using Prometheus to monitor [Spring Boot] apps on Oracle Application Container Cloud (<https://medium.com/oracledevs/using-prometheus-to-monitor-apps-on-oracle-cloud-7b75dea9f2f3>)
- Documentation Oracle Application Container Cloud Service and Java SE applications: <https://docs.oracle.com/en/cloud/paas/app-container-cloud/create-sample-java-se-applications.html>
- Documentation on Meta Data Files for Applications deployed to Application Container Cloud - <https://docs.oracle.com/en/cloud/paas/app-container-cloud/dvcjv/creating-meta-data-files.html>