CLOUD FOUNDRY

Pivotal Cloud Foundry Developer

Lab Instructions

Application Deployment using Pivotal Cloud Foundry

Version 1.11.a

Pivotal

Copyright Notice

Pivotal

Table of Contents

Chapter 1. CF Tasks

1.1. Preface

`attendee-service` is configured to automatically create a database schema on startup. You can verify this by inspecting the source code of the `attendee-service` project.

Study the file `src/main/resources/application.yml`, and note how it contains the setting `generate-ddl`:

```
spring:
  application:
    name: attendee-service
  jpa:
    generate-ddl: true
```

This setting ensures that when the application starts, the schema will be auto generated from the JPA domain definition.

In this lab, we'll turn off this feature, and instead learn how to run a "one-off" task that will create the needed schema for us explicitly, as a separate step.

1.2. Setup

1. If you don't have it already, Maven is included in your lab files in `apache-maven-3.5.0-bin.zip`.

   - Unpack it and add …/apache-maven-3.5.0-bin/bin to your `PATH`
   - Close any terminal/command windows and open a new one to pick up the changed \PATH\

   Warning

   You will need access to Maven Central to download dependencies. If your company mandates its own local Maven repository, it may not contain the right dependencies and you won't be able to do this lab.

2. If necessary, get a copy of the source code for `attendee-service` (clone or download)

3. Edit the `application.yml` file and set `generate-ddl` to false. Rebuild the application jar file with:

```
mvn clean package
```

4. Do whatever's necessary to delete your existing attendee-service application.

5. Delete the mysql database you'd originally created for the attendee-service application using the command `cf delete-service`

6. Now, install the `mysql` cf cli plugin from the cloudfoundry plugins web site. This will add the command 'mysql' to your cf cli.

7. Recreate your attendees database, perhaps like this:

```
cf create-service cleardb spark attendee-db
```

   Validate that the service was created by invoking the `cf services` command.

8. Push attendee service once more. Assuming you have a manifest file for `attendee-service` that looks similar to this:

```
---
applications:
- name: attendee-service
  path: target/attendee-service-0.0.1-SNAPSHOT.jar
  memory: 768M
  random-route: true
  services:
  - attendee-db
```

   Invoke your `cf push` command from within the `attendee-service` base directory:

```
cf push
```

9. Now, invoke the `cf mysql` command, which should produce output similar to this:

```
MySQL databases bound to an app:

attendee-db
```

10. Let's check that the database has no tables at this time:

```
cf mysql attendee-db
```

   then:

```
mysql> show tables;
```

   And ensure that the output shows that no tables exist at this time.

In fact, let's go further:

1. Tail the logs for `attendee-service`

```
cf logs attendee-service
```

2. In a browser, visit the `attendees/` endpoint and verify that you get an error, and verify that the logs contain an error message complaining that the attendees table does not exit.

1.3. Run a "one-off" task to generate the schema

Familiarize yourself with Cloud Foundry tasks by reviewing the documentation.

It turns out that our spring boot application already has the knowledge for creating our schema. That knowledge is embedded in the spring data jpa libraries. The idea is simple:

- we'll run a spring-boot app not as a web service, but as a task by turning off the spring-boot property `spring.main.web-environment`
- next, for this run, we'll turn on the `spring.jpa.generate-ddl` property
- we'll specify a start command for our task that leverages everything that's already inside our droplet: the jdk and our spring boot application

Here's the command we want to run:

```
cf run-task attendee-service '.java-buildpack/open_jdk_jre/bin/java \
    -Dspring.jpa.generate-ddl=true \
    -Dspring.profiles.active=cloud \
    -Dspring.main.web-environment=false \
    -cp . org.springframework.boot.loader.JarLauncher'
```

You'll see output similar to this:

```
Creating task for app attendee-service in org eitan-org / space eitan-space as esuez@pivotal.io...
OK

Task has been submitted successfully for execution.
task name:   fb9e9533
task id:     1
```

Run the 'cf tasks' command to check on the status of our task:

```
cf tasks attendee-service
```

The output should resemble this:

```
id   name       state       start time                      command
1    fb9e9533   SUCCEEDED   Thu, 11 May 2017 02:45:15 UTC   .java-buildpack/open_jdk_jre/bin/java \
    -Dspring.jpa.generate-ddl=true \
    -Dspring.profiles.active=cloud \
    -Dspring.main.web-environment=false \
    -cp . org.springframework.boot.loader.JarLauncher
```

Note that the state is `SUCCEEDED`.

Let’s now verify that our table has been created:

```
cf mysql attendee-db
```

and:

```
mysql> show tables;
```

..should show:

```
+------------------------------+
| Tables_in_ad_12f26b7197bb693 |
+------------------------------+
| attendee                     |
+------------------------------+
1 row in set (0.03 sec)
```

Now, attempt to revisit the `attendees` endpoint once more, and you’ll see that the call succeeds.

Congratulations, you’ve just run a one-off task inside cloudfoundry!

Acknowledgements to Liu Dapeng and his published example.