This example replaces the in-memory data structure of the corporatedirectory example with a database using JPA.

**Setup**

You will need to create an employee user in your DBCS PDB. This user will need rights to create and drop schema. Once the user is created the application will take care of table creation and population.

**Deploying via Service Console**

**Create Application**

Because the application depends on a service binding being created in order to run you will need to create a Java application using the sample option.

**Define Database Service Binding**

Once the application is created you can go to the Deployments tab and define the service binding to your DBCS instance. Once created your list of environment variables should include something like:

DBAAS\_DEFAULT\_CONNECT\_DESCRIPTOR 129.191.0.93:1521/PDB1.jcsdemo200.oraclecloud.internal

DBAAS\_USER\_NAME employee

DBAAS\_USER\_PASSWORD \*\*\*\*\*\*\*\*

DBAAS\_LISTENER\_HOST\_NAME 129.191.0.93

DBAAS\_LISTENER\_PORT 1521

DBAAS\_DEFAULT\_SID ORCL

DBAAS\_DEFAULT\_SERVICE\_NAME PDB1.jcsdemo200.oraclecloud.internal

The application will read the following environment variables (see EmployeeService.java) to log in:

DBAAS\_DEFAULT\_CONNECT\_DESCRIPTOR

DBAAS\_USER\_NAME

DBAAS\_USER\_PASSWORD

**Build and Deploy the Application**

Build the application locally with build.sh--it just runs Maven. The output is a zip file in the target folder that is ready to deploy.

Note: You can test the application locally by setting up an SSH tunnel to the database cloud service DB and defining the required environment variables. The launch command is in the manifest.json file.

**Exploring the Application**

Opening the application will result in a page with a list of employees. Behind the scenes, the application will create tables and populate them if necessary.

If you create/edit employees you can see the generated SQL by obtaining and downloading application logs.