(../..)

Requirements

Lab Requirements (../requirements)

Purpose of this lab

- How to implement the Service Broker API
- How to deploy a service broker as an app to Pivotal Cloud Foundry
- How to register the service broker with the Cloud Controller
- How to make the single plan in the catalog "public" in your organization
- Estimated Time: 60 minutes

Setup

1. Download the zip file (cloudfoundry-mongodb-service-broker.zip). The zip file contains source code and jar ready for you to deploy (no building necessary). Copy the file to folder: `~/pivotal-cloud-foundry-developer-workshop/`. You will need to create this directory in your home (https://en.wikipedia.org/wiki/Home_directory) directory.

2. Extract the the zip file to `~/pivotal-cloud-foundry-developer-workshop/cloudfoundry-mongodb-service-broker`.

3. Import applications into your IDE (IntelliJ).

Service Broker API Overview

1. Review the documentation (http://docs.pivotal.io/pivotalcf/services/api.html#api-overview), specifically the sequence diagram. This is what we will implement.

Create a MongoDB Service Broker

About this Broker

This broker is implemented with Spring Boot and leverages the Spring Cloud Service Broker (https://github.com/spring-cloud/spring-cloud-cloudfoundry-service-broker) project. As a result, a lot of the work has been done for us. In a nutshell, this means that Service Broker endpoints have been mapped (provided) but there a few beans and interfaces we need to implement to complete a broker implementation.

Implement Catalog Management

1. Review the documentation on implementing catalog management (http://docs.pivotal.io/pivotalcf/services/api.html#catalog-mgmt).

2. We need to implement catalog management in our `mongodb-service-broker` application. Fortunately, all the Service Broker API endpoints have been mapped by the Spring Cloud Service Broker project. For instance, the `v2/catalog` endpoint (https://github.com/spring-cloud/spring-cloud-cloudfoundry-service-broker/blob/master/src/main/java/org/springframework/cloud/servicebroker/controller/CatalogController.java).

3. We have an endpoint, but the Spring Cloud Service Broker can't provide all the implementation. We need to describe our catalog. To to that, all we need to do is provide a `Catalog` bean.

   Review the following file: `~/pivotal-cloud-foundry-developer-workshop/cloudfoundry-mongodb-service-broker/src/main/java/org/springframework/cloud/servicebroker/mongodb/config/CatalogConfig.java`.

```
@Configuration
public class CatalogConfig {

  @Bean
  public Catalog catalog() {
    return new Catalog(Collections.singletonList(
        new ServiceDefinition(
            getEnvOrDefault("SERVICE_ID","mongodb-service-broker"), //env variable
            getEnvOrDefault("SERVICE_NAME","MongoDB"), //env variable
            "A simple MongoDB service broker implementation",
            true,
            false,
            Collections.singletonList(
                new Plan(getEnvOrDefault("PLAN_ID","mongo-plan"), //env variable
                    "standard",
                    "This is a default mongo plan.  All services are created equally.",
                    getPlanMetadata(),
                    true)),
            Arrays.asList("mongodb", "document"),
            getServiceDefinitionMetadata(),
            null,
            null)));
  }
//...
}
```

4. Push the `mongodb-service-broker` application.

```
cd ~/pivotal-cloud-foundry-developer-workshop/cloudfoundry-mongodb-service-broker/
cf push mongodb-service-broker -p build/libs/cloudfoundry-mongodb-service-broker.jar -m 512M --random-r
oute --no-start
```

5. Set environment variables.

   These environment variables get used by the broker to generate the catalog. These values should be unique across the entire Pivotal Cloud Foundry instance to meet the broker API specifications.

   As a convention, append your initials to where specified.

```
cf set-env mongodb-service-broker SERVICE_ID mongodb-service-broker-<initials>
cf set-env mongodb-service-broker SERVICE_NAME MongoDB-<initials>
cf set-env mongodb-service-broker PLAN_ID mongo-plan-<initials>
```

   For example:

```
cf set-env mongodb-service-broker SERVICE_ID mongodb-service-broker-dnr
cf set-env mongodb-service-broker SERVICE_NAME MongoDB-dnr
cf set-env mongodb-service-broker PLAN_ID mongo-plan-dnr
```

   You can safely ignore the "TIP: Use 'cf restage' to ensure your env variable changes take effect" message.

6. Start `mongodb-service-broker`

```
cf start mongodb-service-broker
```

7. Verify your work. Call the application `v2/catalog` endpoint through a browser. Because the application is secured with Basic Auth you will need to provide credentials.

   Username: pivotal
   Password: keepitsimple

You should see response similar to the following (pic is using the JSON Formatter for Chrome (https://chrome.google.com/webstore/detail/json-formatter/bcjindcccaagfpapjjmafapmmgkkhgoa?hl=en):

mongodb-service-broker-unpersonalizing-pitsaw.cfapps.io/v2/catalog


```
{
  "services": [
    {
      "id": "mongodb-service-broker-dnr",
      "name": "MongoDB-dnr",
      "description": "A simple MongoDB service broker implementation",
      "bindable": true,
      "plan_updateable": false,
      "plans": [
        {
          "id": "mongo-plan-dnr",
          "name": "standard",
          "description": "This is a default mongo plan.  All services are created equally.",
          "metadata": {
            "bullets": [
              "Shared MongoDB server",
              "100 MB Storage (not enforced)",
              "40 concurrent connections (not enforced)"
            ]
          },
          "free": true
        }
      ],
      "tags": [
        "mongodb",
        "document"
      ],
      "metadata": {
        "longDescription": "MongoDB Service",
        "documentationUrl": "https://github.com/spring-cloud-samples/cloudfoundry-mongodb-service-broker",
        "providerDisplayName": "Pivotal",
        "displayName": "MongoDB",
        "imageUrl": "http://info.mongodb.com/rs/mongodb/images/MongoDB_Logo_Full.png",
        "supportUrl": "https://github.com/spring-cloud-samples/cloudfoundry-mongodb-service-broker"
      },
      "requires": [],
      "dashboard_client": null
    }
  ]
}
```


8. Register your Service Broker.

   We will be creating a Space-Scoped (http://docs.pivotal.io/pivotalcf/services/managing-service-brokers.html) broker. Space-Scoped brokers help you during the development/testing of your service broker, because they are private to a space and don't require an `admin` to enable access (list it in the marketplace, provision service instances, etc).

   A unique broker name is required. Use your initials.

   For Example:

   ```
   cf create-service-broker mongodb-service-broker-dnr pivotal keepitsimple https://mongodb-service-broker
   -pert-dagger.pcfi1.fe.gopivotal.com --space-scoped
   ```

9. View the Service Brokers in your installation. You should see your new Service Broker.

   ```
   cf service-brokers
   ```

10. View service access.

    ```
    cf service-access
    ```

    Notice that your service access is set to `none`, because this is space-scoped broker.

11. Verify that your service is listed in the marketplace.

    ```
    cf marketplace
    ```

Congratulations, you have implemented and tested the catalog endpoint in your service broker!

Implement Provisioning and Deprovisioning

1. Review the documentation on implementing provisioning (http://docs.pivotal.io/pivotalcf/services/api.html#provisioning) and deprovisioning (http://docs.pivotal.io/pivotalcf/services/api.html#deprovisioning).

2. We need to implement provisioning/deprovisioning in our `mongodb-service-broker` application. To do so we just need to implement the ServiceInstanceService (https://github.com/spring-cloud/spring-cloud-cloudfoundry-service-broker/blob/master/src/main/java/org/springframework/cloud/servicebroker/service/ServiceInstanceService.java) interface, because the Spring Cloud Service Broker project has already done the mapping (https://github.com/spring-cloud/spring-cloud-cloudfoundry-service-broker/blob/master/src/main/java/org/springframework/cloud/servicebroker/controller/ServiceInstanceController.java).

   Review the following file: `~/pivotal-cloud-foundry-developer-workshop/cloudfoundry-mongodb-service-broker/src/main/java/org/springframework/cloud/servicebroker/mongodb/service/MongoServiceInstanceService.java`

   Provisioning Code:

```
@Service
public class MongoServiceInstanceService implements ServiceInstanceService {
//...

  @Override
  public CreateServiceInstanceResponse createServiceInstance(CreateServiceInstanceRequest request) {
    // make sure we haven't provisioned this before (check broker database)
    ServiceInstance instance = repository.findOne(request.getServiceInstanceId());
    if (instance != null) {
      throw new ServiceInstanceExistsException(request.getServiceInstanceId(), request.getServiceDefini
tionId());
    }

    instance = new ServiceInstance(request);

    if (mongo.databaseExists(instance.getServiceInstanceId())) {
      // ensure the instance is empty
      mongo.deleteDatabase(instance.getServiceInstanceId());
    }

    DB db = mongo.createDatabase(instance.getServiceInstanceId());
    if (db == null) {
      throw new ServiceBrokerException("Failed to create new DB instance: " + instance.getServiceInstan
ceId());
      }
    //save to broker database for record keeping
    repository.save(instance);

    return new CreateServiceInstanceResponse();
  }
//...
}
```

The `createServiceInstance` method is where our broker provisions the database. But to do so two things must happen:

1. Record details in the broker database that we are provisioning a service instance (a MongoDB database)

2. Create the database

   Deprovisioning Code:

```
@Service
public class MongoServiceInstanceService implements ServiceInstanceService {

    //...

  @Override
  public DeleteServiceInstanceResponse deleteServiceInstance(DeleteServiceInstanceRequest request) thro
ws   MongoServiceException {
    String instanceId = request.getServiceInstanceId();
    //locate record in broker database
    ServiceInstance instance = repository.findOne(instanceId);

    if (instance == null) {
      throw new ServiceInstanceDoesNotExistException(instanceId);
    }

    // delete mongo database
    mongo.deleteDatabase(instanceId);

    // delete record from broker database
    repository.delete(instanceId);
    return new DeleteServiceInstanceResponse();
  }
}
```

The `deleteServiceInstance` method is where our broker deprovisions the database. To do so two things must happen:

1. Delete the database
2. Delete the record of the service instance in the broker database

Implement Binding and Unbinding

1. Review the documentation on implementing binding (http://docs.pivotal.io/pivotalcf/services/api.html#binding) and unbinding (http://docs.pivotal.io/pivotalcf/services/api.html#unbinding).

2. We need to implement binding/unbinding in our `mongodb-service-broker` application. To do so we just need to implement the ServiceInstanceBindingService (https://github.com/spring-cloud/spring-cloud-cloudfoundry-service-broker/blob/master/src/main/java/org/springframework/cloud/servicebroker/service/ServiceInstanceBindingService.java) interface, because the Spring Cloud Service Broker project has already done the mapping (https://github.com/spring-cloud/spring-cloud-cloudfoundry-service-broker/blob/master/src/main/java/org/springframework/cloud/servicebroker/controller/ServiceInstanceBindingController.java).

   Review the following file: `~/pivotal-cloud-foundry-developer-workshop/cloudfoundry-mongodb-service-broker/src/main/java/org/springframework/cloud/servicebroker/mongodb/service/MongoServiceInstanceBindingService.java`

   Binding Code:

```java
@Service
public class MongoServiceInstanceBindingService implements ServiceInstanceBindingService {

  //...

  @Override
  public CreateServiceInstanceBindingResponse createServiceInstanceBinding(CreateServiceInstanceBinding
Request request) {

    String bindingId = request.getBindingId();
    String serviceInstanceId = request.getServiceInstanceId();

    ServiceInstanceBinding binding = bindingRepository.findOne(bindingId);
    if (binding != null) {
      throw new ServiceInstanceBindingExistsException(serviceInstanceId, bindingId);
    }

    String database = serviceInstanceId;
    String username = bindingId;
    String password = "password";


    mongo.createUser(database, username, password);

    Map<String, Object> credentials =
        Collections.singletonMap("uri", (Object) mongo.getConnectionString(database, username, password
));

    binding = new ServiceInstanceBinding(bindingId, serviceInstanceId, credentials, null, request.getBo
undAppGuid());
    bindingRepository.save(binding);

    return new CreateServiceInstanceAppBindingResponse().withCredentials(credentials);
  }

  //...
}
```

The `createServiceInstanceBinding` method is where our broker binds an application to the provisioned service instance (database). But to do so two things must happen:

1. Create a unique set of credentials for this binding request in MongoDB
2. Create a record of the binding in the broker database

   Unbinding Code:

```java
@Service
public class MongoServiceInstanceBindingService implements ServiceInstanceBindingService {

  @Override
  public void deleteServiceInstanceBinding(DeleteServiceInstanceBindingRequest request) {
    String bindingId = request.getBindingId();
    ServiceInstanceBinding binding = getServiceInstanceBinding(bindingId);

    if (binding == null) {
      throw new ServiceInstanceBindingDoesNotExistException(bindingId);
    }

    mongo.deleteUser(binding.getServiceInstanceId(), bindingId);
    bindingRepository.delete(bindingId);
  }
}
```

The `deleteServiceInstanceBinding` method is where our broker unbinds an application to the provisioned service instance (database). But to do so two things must happen:

1. Delete the credentials (user) for this binding request in MongoDB
2. Delete the record of the binding in the broker database

Congratulations! You have created a simple service broker.

Use the MongoDB Service Broker

1. Configure the `mongodb-service-broker` application to use a MongoDB instance.

A MongoDB instance can be obtained in the following ways:

1. Your instructor will provision MongoDB and provide connectivity details to you

2. Use a MongoDB instance in your environment

3. See your instructor to provision a MongoDB Instance

   Make sure to obtain the IP address of your MongoDB instance before moving forward. The broker will attempt to connect to MongoDB on port 27017.

   Note: MongoDB security configuration should not be enabled (`security.authorization = false`).

   ```
   cf set-env mongodb-service-broker MONGODB_HOST <IP-ADDRESS>
   ```

   You can safely ignore the "TIP: Use 'cf restage' to ensure your env variable changes take effect" message.

4. Restart the application.

   ```
   cf restart mongodb-service-broker
   ```

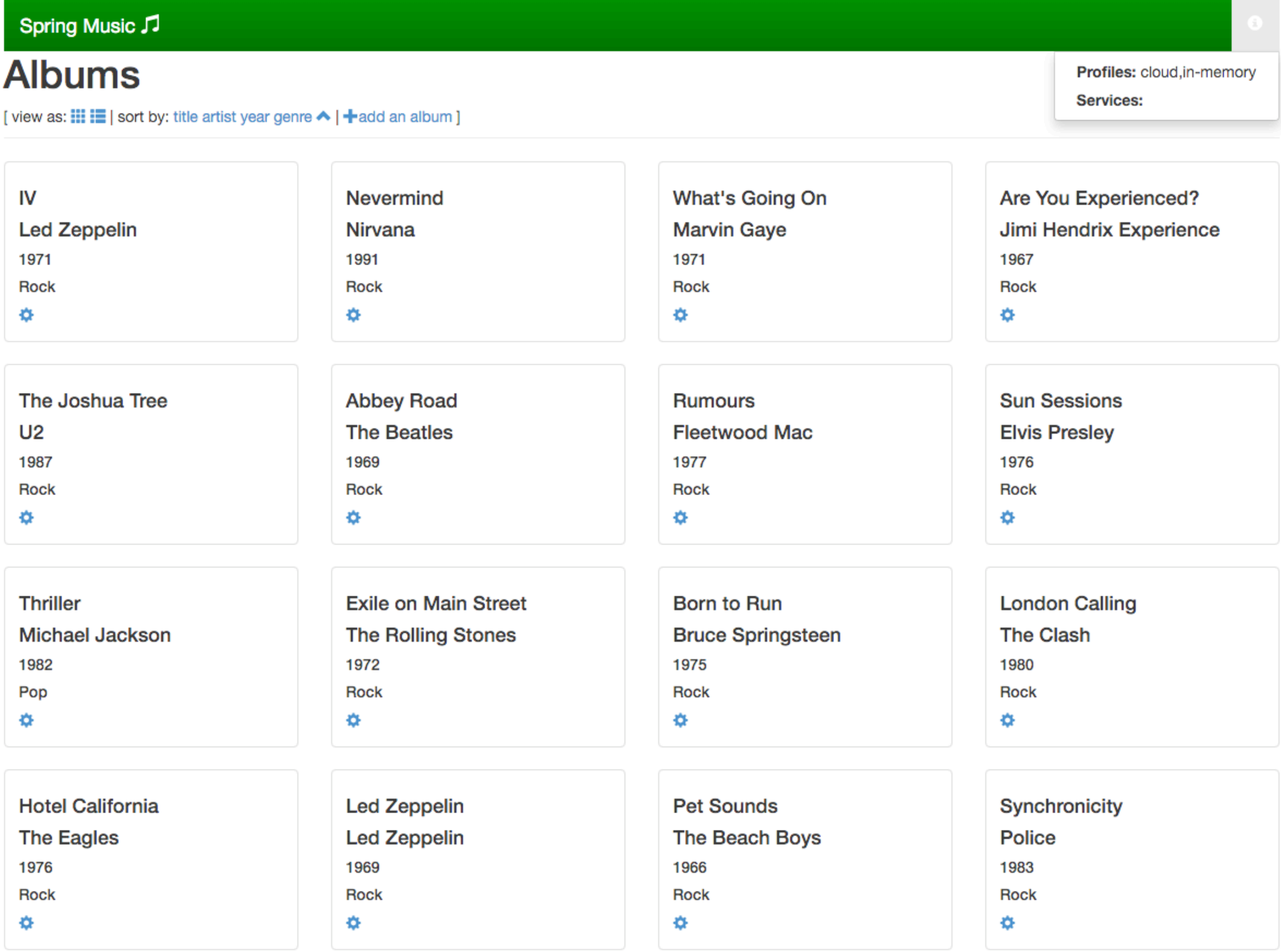
5. Download Spring-Music (spring-music.war). Copy the file to folder: `~/pivotal-cloud-foundry-developer-workshop/spring-music/` . You will need to create this directory in your home directory.

   Source (https://github.com/pivotal-education/spring-music) is not required, but you may be curious how it works as you move through the labs.

6. Push `spring-music` .

   ```
   cd ~/pivotal-cloud-foundry-developer-workshop/spring-music/
   cf push spring-music -p ./spring-music.war -m 512M --random-route
   ```

7. View `spring-music` in a browser. Click on the `i` button on the top right of the screen. Notice that there are no services attached and `spring-music` is using an embedded database.



8. Create a MongoDB service instance.

   For Example:

   ```
   cf create-service MongoDB-dnr standard mongo-service
   ```

9. Bind the `spring-music` to `mongo-service`.

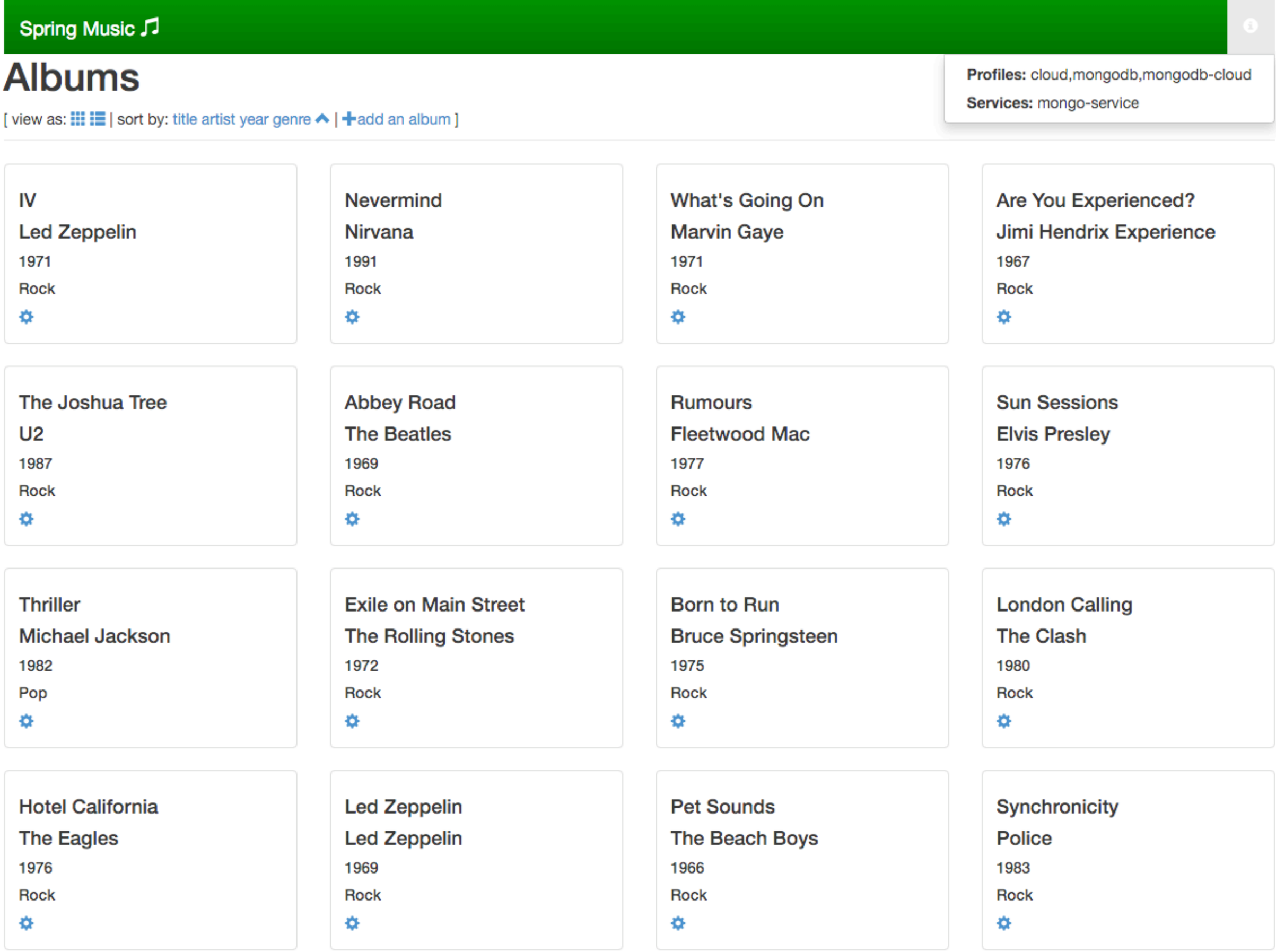
   ```
   cf bind-service spring-music mongo-service
   ```

   You can safely ignore the "TIP: Use 'cf restage spring-music' to ensure your env variable changes take effect" message.

10. Restart `spring-music`

    ```
    cf restart spring-music
    ```

11. Refresh `spring-music` in the browser. Click on the `i` button in the top right of the screen. You are now using MongoDB!



Clean up

1. Delete `spring-music`.

```
cf delete spring-music
```

2. Delete the `mongo-service` service instance.

```
cf delete-service mongo-service
```

3. Delete the service broker.

   For example:

```
cf delete-service-broker mongodb-service-broker-dnr
```

4. Delete `mongodb-service-broker` application.

```
cf delete mongodb-service-broker
```

5. If provisioned, terminate your AWS MongoDB instance by going to your AWS EC2 dashboard, selecting the MongoDB instance, and clicking `Actions → Instance State → Terminate`.

Beyond the class

Review other sample brokers (http://docs.pivotal.io/pivotalcf/services/examples.html).

(https://pivotal.io)

course version: 1.5.3