(../..)

Getting Started

Checkout the `background-jobs-start` tag.

```
git checkout background-jobs-start
```

Explore the application

Check out the `AlbumsUpdater` class and the `AlbumsUpdateScheduler` class. These classes are responsible for updating the albums database. They will sync the database with the contents of an `albums.csv` file in an *S3 blobstore*.

What would happen if you deployed this application and scaled it to 3 instances?

Let's fix it!

Ensuring only one instance runs the background job

You will use your database to track whether an app has run the job in the past 3 minutes. How should you do this?

Advanced path

Discuss the above questions with a partner. What is a logical and minimal way of approaching this problem? When you have decided on a solution, get feedback from an instructor.

Implement your solution.

If you run into problems, check out the base path (below). Otherwise, skip the following section.

Base path

1. Set up your database with the following structure:

   ```
   DROP TABLE IF EXISTS album_scheduler_task;

   CREATE TABLE album_scheduler_task (started_at TIMESTAMP NULL DEFAU
   LT NULL);

   INSERT INTO album_scheduler_task (started_at) VALUES (NULL);
   ```

   You can create a `.ddl` file to save this configuration.

2. Inject the `DataSource` into the `AlbumsUpdateScheduler` then build a `JdbcTemplate` field using it.

3. Wrap the call to `albumsUpdater.update()` in an `if` statement that checks whether the database row has been updated within the last 2 minutes.

   For instance:

   ```
   if (startAlbumSchedulerTask()) {
       logger.debug("Starting albums update");
       albumsUpdater.update();
       logger.debug("Finished albums update");

   } else {
      logger.debug("Nothing to start");
   }
   ```

What query must the `startAlbumSchedulerTask` run against your database? Attempt to write the `startAlbumSchedulerTask` method. Scroll down to the bottom of the page to see an implementation of `startAlbumSchedulerTask` if you get stuck.

When you have completed your album scheduler, run the app locally. Does it work?

Updating the Cloud Foundry database

In order to change the database structure on PCF, we need to connect to it. On PCF, the instances of mySQL are isolated from the outside world. Fortunately, Cloud Foundry provides us the tools to access it via an SSH tunnel.

Follow the official PCF instructions on setting up an SSH tunnel (https://docs.pivotal.io/pivotalcf/1-9/devguide/deploy-apps/ssh-services.html#ssh-tunnel) to your mySQL instance.

Once you can connect using the `mysql` command line client, use it to run our `ddl` file.

```
mysql -u[the-user] -h 0 -p -D [the-database] -P 63306 < schema.ddl
```

Deploying to Pivotal Cloud Foundry

Now open a terminal that will be tailing the application logs.

```
cf logs moviefun
```

Then deploy the app to Pivotal Cloud Foundry and scale it to 3 instances, while still watching the logs.

```
cf push moviefun -p target/moviefun.war
cf scale moviefun -i 3
```

Is there more than one instance picking up the job at any given time? If yes, you may need to compare your implementation with our solution below.

Upload a `CSV` file to your S3 bucket and check that the albums are correctly being updated.

An implementation of `startAlbumSchedulerTask`

```
private boolean startAlbumSchedulerTask() {
   int updatedRows = jdbcTemplate.update(
       "UPDATE album_scheduler_task" +
           " SET started_at = now()" +
           " WHERE started_at IS NULL" +
           " OR started_at < date_sub(now(), INTERVAL 2 MINUTE)"
   );

   return updatedRows > 0;
}
```

Assignment

Once you are done with the lab and the application is deployed and working on PCF, you can submit the assignment using the `submitReplatformingBackgroundJobsWithDb` gradle task. It requires you to provide the `movieFunUrl` project property. For example:

```
cd ~/workspace/assignment-submission
./gradlew submitReplatformingBackgroundJobsWithDb -PmovieFunUrl=http://my-movie-fun.cfapps.io
```

(https://pivotal.io)