**Purpose**

You will create [MySQL](https://www.mysql.com/) databases for the PAL Tracker project. MySQL is a good choice for a data store because it is well tested and understood. A relational database like MySQL is a good fit for almost any data store need.

This lab introduces a tool called [Flyway](https://flywaydb.org/documentation/) to perform database [migrations](https://en.wikipedia.org/wiki/Schema_migration). Migration tools such as Flyway work by having the team writing the application also write the SQL that evolves the database schema. This has many benefits. Migrations can be applied to different environments reproducibly. They also version the schema in lock-step with the code. You know the last migration that was run with looking at a version of the code. Lastly, migrations reduce organizational overhead. It gives the team of engineers responsible for the application the ability to easily migrate the database schema.

**Discussion points:**

* Database migration overview
* SSH tunnel discussion (your network might block ssh).
* Flyway CLI
* Alternative migration tools ([Evolve](https://github.com/lecaillon/Evolve), [EF Core Migrations](https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/migrations), [DbUp](http://dbup.github.io/))

**Get started**

Before starting the lab, pull in a script that we will explore/use later to migrate our databases on PCF:

cd ~/workspace/pal-tracker

git cherry-pick migration-start

**Create the Database DDL**

1. Create a folder in the root of your project called **databases**.
2. Create a folder inside the **databases** folder called **tracker**.
3. Create a Database Definition Language (DDL) file called **create\_databases.sql** with the following content:

[Hide create\_databases.sql](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/database-migrations/index.html" \l "pal-trackerab0fc0b6-3fec-4cc9-8d6a-2047ff9ca58a)

pal-tracker/databases/tracker/create\_databases.sql

DROP DATABASE IF EXISTS tracker\_dotnet\_dev;

DROP DATABASE IF EXISTS tracker\_dotnet\_test;

CREATE DATABASE tracker\_dotnet\_dev;

CREATE DATABASE tracker\_dotnet\_test;

CREATE USER IF NOT EXISTS 'tracker\_dotnet'@'localhost'

IDENTIFIED BY 'password';

GRANT ALL PRIVILEGES ON tracker\_dotnet\_dev.\* TO 'tracker\_dotnet'@'localhost';

GRANT ALL PRIVILEGES ON tracker\_dotnet\_test.\* TO 'tracker\_dotnet'@'localhost';

This file will be used to perform the following actions during local development.

* + Drop old databases
  + Create a development version of the database
  + Create a test version of the database
  + Create a **tracker\_dotnet** user in the database that has access to the databases

**Create Migrations**

Create a file called **V1\_\_initial\_schema.sql** in **databases/tracker/migrations/** with the following content:

[Hide V1\_\_initial\_schema.sql](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/database-migrations/index.html" \l "pal-tracker89964623-ab45-4886-94aa-5ac2063a6275)

pal-tracker/databases/tracker/migrations/V1\_\_initial\_schema.sql

CREATE TABLE time\_entries (

id BIGINT(20) NOT NULL AUTO\_INCREMENT,

project\_id BIGINT(20),

user\_id BIGINT(20),

date DATE,

hours INT,

PRIMARY KEY (id)

)

ENGINE = innodb

DEFAULT CHARSET = utf8;

This file holds the instructions to create the time entries table with the following columns:

* id (integer, primary key)
* project\_id (integer)
* user\_id (integer)
* date (date)1
* hours (integer)

Flyway helps keep track of database versions as a project progresses. Flyway will automatically run our migrations to the latest version assuming that the migration file names follow a certain convention. In this case, the convention is **V##**\_\_ where the hashes are digits corresponding to the migration's order in the list of migrations. Beware that those are two underscores. Read more about flyway naming conventions [here](https://flywaydb.org/documentation/migrations#sql-based-migrations).

1 We will use a date type here because we do not need to track time information. This decision depends on the requirements of the software.

**Migrate local database**

1. Run the newly created **create\_databases.sql** script against your local MySQL instance. This creates the development and test versions of the tracker database, but they do not have any tables or columns yet.
2. If you have completed the setup lab or are working on a PAL provided machine, then Flyway should already be installed. If for some reason it is not, you can download and install the Flyway CLI [here](https://flywaydb.org/documentation/commandline/#download-and-installation).
3. Run the migrations on the development database.
4. flyway -url="jdbc:mysql://localhost:3306/tracker\_dotnet\_dev" -user=tracker\_dotnet -password=password -locations=filesystem:databases/tracker migrate

User will be *root* with no password, if you followed the standard process of installing MySQL. If you are unable to connect with these credentials, follow the [password reset instructions](https://dev.mysql.com/doc/refman/5.7/en/resetting-permissions.html).

1. Do the same on the test database.
2. flyway -url="jdbc:mysql://localhost:3306/tracker\_dotnet\_test" -user=tracker\_dotnet -password=password -locations=filesystem:databases/tracker migrate
3. Inspect the databases with your favorite MySQL tool and verify that the new **time\_entries** table looks correct. For example:
4. mysql -u root

NOTE: this command does not work in git bash. Please run in Powershell.

use tracker\_dotnet\_dev;

describe time\_entries;

+------------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+------------+--------------+------+-----+---------+----------------+

| id | bigint(20) | NO | PRI | NULL | auto\_increment |

| project\_id | bigint(20) | YES | | NULL | |

| user\_id | bigint(20) | YES | | NULL | |

| date | date | YES | | NULL | |

| hours | int(11) | YES | | NULL | |

+------------+--------------+------+-----+---------+----------------+

**Migrate on PCF**

You are building a cloud native application which assumes that backing services (like a database) are provided by the platform. Since the Tracker application now needs a database, you must create the database service and bind your application to it. This instructs the platform to reserve the instance of the service and then provide the connection information to your application.

1. Use the cf marketplace command to find a MySQL database service on your Pivotal Cloud Foundry installation. Use the cf create-service command to create an instance of this service named tracker-database in the *review* and *production* spaces.
2. In both *review* and *production* spaces, bind the service instance to your application using the cf bind-servicecommand.

This instructs Cloud Foundry to provide the connection information for the mysql database you created earlier to your application. This is given to your application as the **VCAP\_SERVICES** environment variable.

**Connecting to PCF Databases**

The database instance that your application has access to is protected by a firewall so you will not be able to connect to it directly. Instead, you can open a SSH tunnel which will use your application as a proxy so that the Flyway CLI running on your CI server can migrate the database managed by Cloud Foundry.

This functionality does assume you have SSH access on the Cloud Foundry instance you are working with, this is the case for the foundation we use for this class.

1. Modify build deployToReview pipeline job to install the tools necessary for migrations and run the migrations:
2. deployToReview:
3. # ...
4. echo "deb https://packages.cloudfoundry.org/debian stable main" | tee /etc/apt/sources.list.d/cloudfoundry-cli.list
5. apt-get update
6. apt-get install cf-cli
7. + - run:
8. + name: Install jq
9. + command: |
10. + apt-get -y install jq
11. +
12. + - run:
13. + name: Install flyway
14. + command: |
15. + curl https://repo1.maven.org/maven2/org/flywaydb/flyway-commandline/5.1.1/flyway-commandline-5.1.1-linux-x64.tar.gz | tar xvz
16. +
17. - run:
18. name: Unzip artifacts
19. command: |
20. mkdir -p $EXTRACT\_DEST
21. tar -xvzf artifacts/pal-tracker-$CIRCLE\_SHA1.tgz -C $EXTRACT\_DEST
22. - run:
23. name: Deploy
24. command: |
25. cd $EXTRACT\_DEST
26. cf login -a $CF\_API -u $CF\_USERNAME -p $CF\_PASSWORD -o $CF\_ORG -s $ENVIRONMENT
27. cf push -f manifest-$ENVIRONMENT.yml
28. + bash ./migrate-databases.sh pal-tracker . /root/project
29. Modify build deployToProd pipeline job to install the tools necessary for migrations and run the migrations:
30. deployToProd:
31. # ...
32. echo "deb https://packages.cloudfoundry.org/debian stable main" | tee /etc/apt/sources.list.d/cloudfoundry-cli.list
33. apt-get update
34. apt-get install cf-cli
35. + - run:
36. + name: Install jq
37. + command: |
38. + apt-get -y install jq
39. +
40. + - run:
41. + name: Install flyway
42. + command: |
43. + curl https://repo1.maven.org/maven2/org/flywaydb/flyway-commandline/5.1.1/flyway-commandline-5.1.1-linux-x64.tar.gz | tar xvz
44. +
45. - run:
46. name: Unzip artifacts
47. command: |
48. mkdir -p $EXTRACT\_DEST
49. tar -xvzf artifacts/pal-tracker-$CIRCLE\_SHA1.tgz -C $EXTRACT\_DEST
50. - run:
51. name: Deploy
52. command: |
53. cd $EXTRACT\_DEST
54. cf login -a $CF\_API -u $CF\_USERNAME -p $CF\_PASSWORD -o $CF\_ORG -s $ENVIRONMENT
55. cf push -f manifest-$ENVIRONMENT.yml
56. + bash ./migrate-databases.sh pal-tracker . /root/project
57. Include the migrate-databases.sh script and the migrations in the build artifact output from build.sh:
58. cp manifest-\*.yml $build\_output
59. +cp scripts/migrate-databases.sh $build\_output
60. +cp -r databases $build\_output
61. dotnet publish src/PalTracker --configuration Release \
62. --output $build\_output/src/PalTracker/bin/Release/netcoreapp2.1/publish
63. Commit and push your code to GitHub, then let your pipeline deploy the new application. Trigger a production deploy once you are confident that everything is working.

**Assignment submission**

Submit the assignment using the **cloudNativeDeveloperDatabaseMigrations** gradle task.

cd ~/workspace/assignment-submission

./gradlew cloudNativeDeveloperDatabaseMigrations -PserverUrl=https://[app-url]

**Extra**

Write a few other migrations if you are finished with this assignment before the rest of the class is done. Explore some of the other [Flyway commands](https://flywaydb.org/documentation/) and investigate how you can use them to manipulate the database. Next, try implementing your migrations using a different [versioning scheme](https://flywaydb.org/documentation/migrations#versioned-migrations).