CPSC 449 - Web Back-End Engineering - Fall 2019

Project 2, due November 27

This project expands on Project 1, introducing an API gateway, sharding the tracks database, and generating playlists in a format that can be opened in a music player.

Dev 1 - Sharding the tracks database

Replace the single SQLite database configuration from the previous project with four SQLite databases: three databases holding shards of the table (or tables) containing track information, and one database containing the rest of the tables.

If you are using the `sqlite3` module directly, you will need to modify `get_db()` to store connections to all databases in the application context. If you are using PugSQL, you will need to create four different modules.

Split your tracks into shards based on the track ID so that the tracks for a given playlist will be spread across all three databases. Map the shard key modulo 3 to the database partition.

Using Track IDs as shard keys

Since the SQLite databases are independent, using `INTEGER AUTOINCREMENT`, `ROWID` or similar values as primary keys in sharded tables may lead to duplicate values. One way to solve this problem is to use globally unique identifiers as keys.

While SQLite does not directly support GUIDs, see the StackOverflow answer Proper way to store GUID in sqlite for an example of configuring the Python `sqlite3` module to use `uuid.UUID` objects as primary keys (though note that the call to `buffer()` is no longer required in Python 3).

Database initialization

You will need to update your database initialization to create the same tables in each shard. If you `INSERT` test data into the database directly (rather than populating only through the Web Service API), make sure that you insert items into the correct shard.

Depending on your API and how it was populated in the previous project, you may also need to update the population script (e.g. if URLs include Track IDs).

Dev 2 - Creating XSPF playlists

Create a new microservice that will, given a Playlist ID, will generate an XML playlist in XSPF format. This service will be placed in front of the API gateway and will contact the other microservices through the API gateway as needed.

Your XSPF response should include all data associated with the target Playlist id stored in the Tracks, Playlists, Users, and Descriptions microservices. Note that there are XSPF fields for every piece of data maintained by the microservices, though you may need to consult the specification to find them.

Requesting data from microservices

In order to retrieve the data for a feed, your XSPF microservice will need to make HTTP requests to the other microservices. While the Python standard library includes a module for making HTTP requests, the API is somewhat cumbersome. A better library for making HTTP calls is Requests, which can be installed on Tuffix with the following command:

```
$ pip3 install --user requests
```

When making HTTP requests, use the base URL for the API gateway (e.g. `http://localhost:8000/tracks`) rather than the base URL for an individual microservice (e.g. `http://localhost:5000/tracks`).

Generating XSPF

While you are welcome to generate XSPF by pasting strings together, you may find it easier to use Jinja2 templates or a module such as xspf.

Testing playlists

In order to actually stream music, you will need valid media URLs for each track. Work with your teammate in the Ops role to upload media files into MinIO.

Before trying to play your music, validate your playlist XML and correct any errors.

Now open your playlist as a network stream using a media player that supports XSPF. VLC, included in Tuffix, should allow this. (Parole supports XSPF in local files, but not from network locations; you can test this by saving the XSPF file from your browser then opening the file in Parole.)

Note: Depending on the version of VLC, you may not be able to play network streams unless their URLs end with `.xspf` (e.g. `http://localhost:8000/playlist/1.xspf`). If your playlist URL includes query string parameters (e.g. `http://localhost:8000/playlist?id=1`), it may not play in VLC, even if the MIME type is set correctly to `application/xspf+xml`. You may be able to fix this by:

1. Upgrading to a newer version of VLC.
2. Forcing the URL to end with `.xspf` by tacking on an additional dummy query string parameter (e.g., `http://localhost:8000/playlist?id=1&unused=.xspf`).

Ops - Kong and MinIO

Project 1 introduced four different microservices, each running on a different port. In this project you will consolidate those services behind the Kong API Gateway. The XSPF microservice will then be able to access the other microservices through Kong's port 8000 rather than needing to track the individual Flask ports.

Installing and setting up Kong

Follow Step 1 of the Ubuntu Installation instructions to install Kong on Tuffix using the `.deb` package for Ubuntu 18.04 Bionic. Then use the following commands to install PostgreSQL and create a database and configuration file for Kong:

```
$ sudo apt install --yes postgresql
$ sudo -u postgres psql -c "CREATE USER kong WITH ENCRYPTED PASSWORD 'kong'"
$ sudo -u postgres psql -c 'CREATE DATABASE kong OWNER kong'
$ sudo cp /etc/kong/kong.conf.default /etc/kong/kong.conf
```

Now edit `/etc/kong/kong.conf` to ucomment `pg_password` and change it to `kong`, then use the following commands to start Kong:

```
$ sudo kong migrations bootstrap
$ ulimit -n 4096 && sudo kong start
```

You can then proceed with the `curl` command in Step 4 of the Ubuntu Installation instructions to verify that Kong is running.

Installing MinIO and uploading media

Visit `https://docs.min.io/` to download the MinIO server binary for GNU/Linux. Create a new directory to store the data for the server, and follow the directions to start the MinIO server.

When the MinIO server starts, it will print an endpoint, access key, and secret key. Make a note of these values so that you can access the server later.

Use the information provided to log into the MinIO Browser and create a bucket named `tracks`. Edit the policy for this bucket to add the prefix `*` as Read Only. Upload some music files and verify that you can access them via URLs such as `http://localhost:9000/tracks/music-file.mp3`.

Configuring Kong

Kong is configured via a REST API, and its documentation uses `curl` commands as examples. As you configure Kong, be sure to save the `curl` commands that you use so that the configuration can be reproduced.

Create a Service for MinIO and a Route with the path `/media`. Check that you can access media in the `tracks` bucket via URLs such as `http://localhost:8000/media/music-file.mp3`.

Use the `foreman start -c` switch to start 3 instances of each of the original four microservices. (See *Advanced Options* in Introducing Foreman for details, but note that the option has changed to -m or --formation if you are running a more recent version.)

Configure the Kong Ring-balancer with a Service, Route, and Upstream for each microservice and a Target for each instance. (See the "Blue" environment for examples, but do not weight the targets differently.)

Check that when you make repeated requests to URLs such as `http://localhost:8000/tracks` or `http://localhost:8000/playlists`, foreman logs activity sent to each of the upstream service instances.

Test platform

You may use any platform to develop services and tests, but note that per the Syllabus the test environment for projects in this course is a Tuffix VM with Python 3.6.7. It is the responsibility of the Operations role to ensure that your code runs on this platform.

Tips

- Not all primary keys in the application need to be GUIDs: only those that may conflict across sharded tables.

- If you insert test data through SQLite, you may find it easier to insert sharded test data with UUIDs by writing Python code rather than directly executing a `.sql` file.

- If you find that the SQLite3 command-line utility has trouble displaying UUIDs, try this utility.

- The equivalent of

  ```
  sqlite3.connect('test.db', detect_types=sqlite3.PARSE_DECLTYPES)
  ```

for PugSQL is to import the `sqlite3` module, then use the SQLAlchemy-compatible connection string

```
f'sqlite:///test.db?detect_types={sqlite3.PARSE_DECLTYPES}'
```

These can both be done inside a config file.

- If it looks like you need to query tracks across shards, take another look at the required operations from Project 1: it may be that the developer from the last project implemented more API operations than were required.

Teams

This project is written for teams of 3 students. Teams of 2 or 4 students must be pre-approved by the instructor, and only two teams of 2 (or one team of 4) will be approved. This project may not be submitted individually.

Team choices are not permanent; you may choose to work with different teams in later projects.

Roles

Teams must agree on a role for each member, and each project will specify a set of responsibilities for each role. There two Development roles and one Operations role for each project.

In order to receive full credit for the projects, each student must act in a Development role and an Operations role for at least one project.

Responsibilities

Responsibilities for each role are detailed above. Each team member is responsible for documenting their work and assisting other team members with integrating their work together.

Students in two-member teams will each take a Dev role and split the Ops responsibilities. Students in four-member teams will take one microservice each and split the Ops responsibilities.

Submission

Submit your project by uploading your Python code, SQL schema, REST scripts, Kong configuration commands, `Procfile`, documentation, and any other relevant artifacts to the `project2/` subdirectory of the folder that was shared with you on Dropbox. Only one submission is required, but for safety consider uploading copies to each team member's submission folder. (Make certain, however, that the copies are the same in each case; the instructor will not attempt to ascertain which is the “real” submission.) Files should be submitted by 7:00pm on the due date.

A printed submission sheet will be provided the week after the due date. To finalize your submission, fill out the sheet with the requested information and hand it in to the professor by the end of class on that day. Failure to follow any submission instructions exactly will incur a 10% penalty on the project grade for all team members.