CPSC 449 - Back-End Engineering - Fall 2019

Project 3, due December 11

In this project, you will work with your team to port Project 2 to a NoSQL database and add object caching to improve the performance of your XSPF playlist service.

Ops - ScyllaDB, Memcached, and measuring performance

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 3.6.8. It is the responsibility of the Operations role to ensure that your code runs on this platform.

By default, the Tuffix VM image is configured for 2 GB of RAM. If you are running the Tuffix VM, shut down the machine and increase the allocation to at least 3 GB (3072 MB) before attempting to start ScyllaDB.

Installing and configuring Docker

The recommended way to get up and running quickly with ScyllaDB is to run a Docker container. Use the following commands to install Docker on the Tuffix VM and allow the user `student` to run Docker commands:

```
$ sudo apt update
$ sudo apt install --yes docker.io
$ sudo usermod -aG docker $USER
```

Log out and back into the VM to pick up changes before running the docker command below.

Installing ScyllaDB

Ordinarily, Cassandra and ScyllaDB should run in a cluster of multiple servers. Since we are doing development on a single VM and RAM is at a premium, we will start only a single instance of ScyllaDB. Use the following command (entered all on one line):

```
$ docker run --name scylla -d scylladb/scylla --smp 1 --memory 1G
--overprovisioned 1 --developer-mode 1 --experimental 1
```

Once the image has been downloaded, wait a few moments, then check that ScyllaDB is up with

```
$ docker exec -it scylla nodetool status
```

If this command fails, you can check for errors in the ScyllaDB logs by running

```
$ docker logs scylla
```

Memory allocation

Note that the `--memory 1G` switch is a minimum value for starting ScyllaDB on a Tuffix VM with 3G of RAM. If you run ScyllaDB on another platform, you may need to adjust that figure. You may also wish to increase it if your host has more RAM available.

Leaving off the `--memory` switch entirely is not recommended unless you are on hardware dedicated exclusively to running ScyllaDB.

Managing ScyllaDB

Once ScyllaDB is up, you can execute CQL commands using

```
$ docker exec -it scylla cqlsh
```

If you need to stop ScyllaDB, use

```
$ docker stop scylla
```

and restart with

```
$ docker start scylla
```

If something goes very wrong, you can remove ScyllaDB completely and start over with

```
$ docker rm -f scylla && docker rmi scylladb/scylla
```

Installing Memcached

Use the following command to install and start the Memcached distributed object cache and administration tools:

```
$ sudo apt install --yes memcached libmemcached-tools
```

You can verify that the server is running by retrieving cache statistics:

```
$ memcstat --servers=localhost
```

Among the statistics you may need for later testing are `get_hits`, `get_misses`, and `curr_items`.

Starting the project

Once the ScyllaDB column families have been created and populated, start the microservices, and Kong as in Project 2. If resources are limited on your virtual machine, you may want to use foreman to start only a single instance of each service.

Check that the project is running correctly by verifying that you can load an XSPF playlist.

Cache testing

Once you have verified that the project is up, empty the Memcached cache with the following command:

```
$ memcflush --servers=localhost
```

Use `memcstat` to verify that the cache contains 0 bytes and make a note of the number of hits and misses.

Now open the Timings tab from the Network request details pane of the Network Monitor. Load an XSPF playlist and take a screenshot showing the time spent waiting for a response from the server.

Run `memcstat` again, and you should find that the cache now contains several cached objects. Reload the XSPF file, and note the difference in time when the XSPF server uses values from the cache rather than loading data from the other microservices a second time. If you do not see a difference, check the cache statistics and the documentation for the other libmemcached client applications to help debug the issue.

Dev 1 - Porting to ScyllaDB

Update the code for the tracks, playlists, users, and descriptions microservices to use ScyllaDB, a wide-column NoSQL database compatible with Cassandra.

Installing the Python Cassandra driver

Install the DataStax Python Cassandra Driver, either by running

```
$ sudo apt install --yes python3-cassandra
```

or following the instructions provided.

Verify that you can connect to ScyllaDB from Python by creating a `Cluster` object and retrieving a `Session` object. You will need to use the `nodetool` command shown above under *Installing ScyllaDB* to obtain the node's IP address.

Cassandra Flask extension

You may wish to install and use the Flask-Cassandra extension to configure your Cassandra connection, but this is not a requirement.

Cassandra data model

Create a single keyspace and at least one column family (the Cassandra equivalent of a table). You may wish to store user information in a separate column family, but in general all other microservices should connect to a single column family which will not be normalized. Keep in mind that Cassandra supports collections as a column type, and you should use them as appropriate.

Bear in mind the advice from Chapter 8 of the textbook: identify the access patterns for each API call, and create secondary indexes for any queries not based on primary key. See Cassandra Succinctly, Scylla Secondary Indexes, and Basic Rules of Cassandra Data Modeling for more information and examples.

Remember also that Cassandra's primary keys generally have two parts: a partition key and one or more clustering keys, and that only clustering keys are used for sorting. If you run into trouble writing queries, see these two articles for approaches:

- We Shall Have Order!
- Cassandra Time Series Data Modeling For Massive Scale

Working with the database

As with previous projects, you will need to drop and re-create database objects and test data. Use the `cqlsh` command shown above to run CQL commands interactively, or (preferably) encapsulate them in a Flask `init` custom command.

Dev 2 - Using Memcached to cache JSON responses

No matter how well your back-end is designed, you will often find that front-end concerns impact back-end code. In this case, you will probably have noticed that XPSF playlists are slow to load because they need to retrieve information from multiple backend microservices.

Installing pymemcache

Install the pymemcache library, either by running

```
$ sudo apt install --yes python3-pymemcache
```

or

```
$ pip3 install --user pymemcache
```

For a tutorial on programming with Memcached, see Python + Memcached: Efficient Caching in Distributed Applications.

Modifying the XSPF playlist service

Update the code for the XSPF service to check Memcached for a cached JSON object before making a request from any other microservice. If the cached value exists, use that value to construct the XSPF response.

If no cached value exists, proceed to make the HTTP request from the microservice as in Project 2, but update the code to store the JSON response in Memcached with an expiration of at least 60 seconds.

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 are 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, CQL schema, REST scripts, Kong configuration commands, `Procfile`, browser console screenshots, documentation, and any other relevant artifacts to the `project3/` 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.)

A printed submission sheet will be provided on 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.