CPSC 449 - Web Back-End Engineering

Project 2, Spring 2020, due May 5

Last updated Thursday, May 1, 6:35 pm PDT

This project expands on Project 1, adding polyglot persistence to your existing microservices and using the information from those services to expose a new microservice representing the data needed for the home page of our Reddit-like site.

Note: Update the Tuffix repositories before installing software

The current version of Tuffix (Tuffix 2019 Edition r2) is based on Ubuntu 19.04 Disco, which reached end-of-life on January 23. The repositories for this version of Ubuntu were relocated on April 15 to a new server, `old-releases.ubuntu.com`.

In order to use the `apt` commands listed below, you will need to update the list of software sources to use the new server name. Run the following command to update the list:

```
$ sudo sed -i -re \
's/([a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' \
/etc/apt/sources.list
```

Dev 1 - Porting the posting microservice to Amazon DynamoDB Local

Update the code for the posting microservice to use Amazon DynamoDB Local, a version of Amazon's DynamoDB service that runs locally on your own computer.

Installing DynamoDB Local

Follow the instructions in Deploying DynamoDB Locally on Your Computer to download and install DynamoDB Local. When you reach Step 4, you will need to use the following command to install the AWS CLI on Tuffix:

```
$ sudo apt update
$ sudo apt install --yes awscli
```

The instructions shown for configuring the AWS CLI in Step 4 are incomplete. You can see the details in Quickly Configuring the AWS CLI, but the following command should suffice:

```
$ aws configure

AWS Access Key ID [None]: fakeMyKeyId
```

```
AWS Secret Access Key [None]: fakeSecretAccessKey
Default region name [None]: us-west-2
Default output format [None]: table
```

When you have completed Step 5, you may wish to read Using the AWS CLI with DynamoDB to see what else can be done with the AWS CLI.

Installing the Boto 3 library

You will also need the Boto 3 library in order to access DynamoDB from Python. Install Boto 3 with the following command:

```
$ sudo apt install --yes python3-boto3
```

Once Boto 3 is installed, you can work through the Getting Started Developing with Python and DynamoDB tutorial to verify that you can connect to DynamoDB Local from Python.

DynamoDB Flask extension

You may wish to install the Flask-Dynamo extension to configure and access your DynamoDB connection from Flask:

```
$ pip3 install --user flask-dynamo
```

Note: This is not a requirement. You may use the Boto 3 library directly to port your Flask service to DynamoDB.

Configuring Flask-Dynamo for DynamoDB Local

In order to use DynamoDB Local, Flask-Dynamo requires the following environment variables to be set:

```
DYNAMO_ENABLE_LOCAL=True
DYNAMO_LOCAL_HOST=localhost
DYNAMO_LOCAL_PORT=8000
```

Remember that these can be set in a `.env` file.

Note: There is a bug in the example code shown in the Specify your Tables section of the Flask-Dynamo Quickstart.

DynamoDB data model

Create at least one denormalized table for posts; you may not need any others. See The Ten Rules for Data Modeling with DynamoDB to get started.

As with Cassandra, DynamoDB's primary keys generally have two parts: a partition key and a sort key. If you run into trouble writing queries, see Using Sort Keys to Organize Data in Amazon DynamoDB.

Dev 2 - Porting to the voting microservice to Redis

Update the code for the voting microservice to use Redis, a key-value database that supports a variety of data structures.

Installing Redis

To install and start Redis on Tuffix, use the following command:

```
$ sudo apt update
$ sudo apt install --yes redis python3-hiredis
```

You can verify that Redis is up and running with:

```
$ redis-cli ping
```

Installing the high-speed Python library for Redis

You will need the redis-py library to access Redis from Python. To install both redis-py and the high-speed Hiredis parser, use the following command:

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

Once redis-py is installed, read How to Use Redis With Python to get started.

Note: Since Redis is already up and running, you can skip directly to Ten or So Minutes to Redis without launching `redis-server`.

Configuring the Flask-and-Redis Flask extension

You may wish to install the Flask-And-Redis extension to configure your Redis connection from Flask:

```
$ pip3 install --user Flask-and-Redis
```

Note: This is not a requirement. You may use the redis-py library directly to port your Flask service to redis.

Redis data model

Redis offers a variety of data types that may be useful in tracking votes. For some tips on how they might be used, see Top Redis Use Cases by Core Data Structure Types and A Collection of Redis Use Cases.

Dev 3 - Aggregating posts and votes with a BFF

While our primary concern in this course is the back-end, it is important to bear in mind that a microservice architecture does complicate the task of the front-end, requiring front-end servers (or, equivalently, native code for desktop or mobile devices) to manage requests to multiple endpoints.

While reducing this complexity is, to some extent, the intent of back-end technologies such as GraphQL, an alternative is the Backends For Frontends (BFF) pattern, where a new back-end service is introduced to pull data from the other services and expose it in a format that can be used directly by the front-end.

Create a new microservice that will provide RSS feeds for the home page of our Reddit-like site, combining data from the posts and votes microservices.

RSS feeds provided by the BFF

Your new front-page microservice should expose endpoints for the following syndication feeds with `Content-Type` header `application/rss+xml`:

- The 25 most recent posts to a particular community
- The 25 most recent posts to any community
- The top 25 posts to a particular community, sorted by score
- The top 25 posts to any community, sorted by score
- The hot 25 posts to any community, ranked using Reddit's "hot ranking" algorithm.

Note: The first two feeds can be constructed with data directly from the posting microservice, but the rest of the feeds will need to combine data from both the posting and voting microservices.

Making HTTP requests

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

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

Generating RSS feeds

While you are welcome to generate RSS XML documents by pasting strings together, you may find it easier to use Flask's Jinja2 Templates or a specialized module such as rfeed or feedgen.

To install rfeed on Tuffix, use the following command:

```
$ pip3 install --user rfeed
```

To install feedgen on Tuffix, use the following commands:

```
$ sudo apt update
$ sudo apt install --yes python3-lxml
$ pip3 install --user feedgen
```

Testing RSS feeds

The Mozilla Thunderbird email client (listed in Tuffix's Xubuntu Menu as "Mail Reader") includes support for RSS feeds.

Before testing your own feeds, make sure that you can read an existing feed such as `https://www.reddit.com/r/webdev.rss`.

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 each team member role to ensure that their code runs on this platform.

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 a different team on this project.

Roles

Since there are only two projects this semester, we will suspend the requirement described in Project 1 that each student act in an Operations role for at least one project.

Instead, the student who worked in **Ops** on Project 1 should take on the **Dev 3** role as described above. The other two Dev roles can be assigned as appropriate.

Responsibilities

Each team member is responsible for documenting their work and assisting other team members with integrating their work together.

Each team member is also responsible for ensuring that their work can be deployed and runs correctly on Tuffix.

Students in two-member teams will take one of the Dev 1 or Dev 2 roles and share the Dev 3 role. Students in four-member teams should assign Dev roles as appropriate and assign testing to the fourth team member.

Submission

Submit your project by uploading your Python code, schema, scripts, configuration files, documentation, and any other relevant artifacts to the `project2/` subdirectory of the folder that was shared with you on Dropbox. A submission sheet will be provided in the same Dropbox folder.

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.)

To finalize your submission, upload your files and fill out the sheet with the requested information by the end of class on the due date. Do not respond to the questions in the submission sheet using the Dropbox commenting feature; modify the document itself. Failure to follow any of these submission instructions exactly will incur a **10%** penalty on the project grade for all team members.