CPSC 449 - Web Back-End Engineering

Project 3, Fall 2022

Last updated Tuesday November 8, 15:25 PST

The service you implemented in Project 1 is a good start, but there are several limitations:

- There is only a single instance of the service
- The service is a monolith that handles two different sets of functionality
- Most endpoints are not authenticated

In this project you will make some design changes to the code to allow for future scaling, then implement authentication and load balancing through an API gateway.

Learning Goals

The following are the learning goals for this project:

1. Gaining further experience with implementing back-end APIs in Python and Quart.
2. Working with existing code produced by another team.
3. Running and debugging multiple instances of a web service process.
4. Implementing read-replication for a relational database.
5. Learning to design a data model for a key-value store.
6. Exploring the features of the Redis NoSQL database.
7. Introducing polyglot persistence to a service by splitting data between data stores.

Project Teams

This project must be completed in a team of three or four students. The instructor will assign teams for this project in Canvas.

See the following sections of the Canvas documentation for instructions on group submission:

- How do I submit an assignment on behalf of a group?

Platform

Per the Syllabus, this project is written for Tuffix 2020. Instructions are provided only for that platform. While it may be possible for you to run portions of this project on other platforms, debugging or troubleshooting any issues you may encounter while doing so are entirely your responsibility.

Libraries, Tools, and Code

The requirements for this project are the same as for Project 2, with the addition of LiteFS and the Redis NoSQL database and client libraries.

In order to complete this project, you will need to have access to a working version of the service from the previous project. Members of the team are welcome to re-use the service they developed with a different team in the last project. If no member of your team was on a team that produced a complete, working version of a service from a previous project, you are encouraged to collaborate and share code across teams.

Note, however, that **all functionality specified below** must be your own original work or the original work of other members of your current team, with the usual exception of sources specified in previous projects such as code provided by the instructor and documentation for the various libraries.

Read Replication

At this point, you should have four service instances behind Nginx acting as an API gateway:

- A single instance of the Users services, and
- Three instances of the Games service

Currently all three instances of the Games service share a single database, but in Exercise 2 you saw how to configure replication for SQLite with LiteFS.

Configuring replicas

Configure three replicas of the database for the Games service: a primary and two secondaries.

Note: because each replica requires its own YAML configuration file, you will need to configure each process separately in your `Procfile` rather than running with `--formation`. Pay close attention, because this will mean that the port assignments for the Games service will change (e.g. from 5100, 5101, and 5102 to 5100, 5200, and 5300). This means that you will need to adjust your upstream load-balancing configuration.

Distributing reads

With read replication, all writes continue to go to the primary database, but reads can be done from any replica. (See the Lecture from Weeks 8 and 9 or Master-Replica Replication by Arpit Bhayani for additional details.)

In Exercise 3 you identified the reads (i.e. `SELECT` statements) in the Games service. Now modify the code for the Games service so that reads are distributed across the three replicas, while writes continue to go to the primary database. You will need to modify:

- The configuration file for the Games service to include all three database URLs
- The database connection code (e.g. `_get_db()` and `close_connection()`) to manage multiple connections.
- The code for each route that reads from the database to choose from a replica.

  Hint: Consider `itertools.cycle()` from the Python Standard Library, but note that this would be a problem if one of the replicas become unavailable.

  Note: You may need to be careful about the order of `SELECT` statements vs. `INSERT` / `UPDATE` / `DELETE` statements, lest you lose Read-your-write Consistency.

Adding a New Service

Create a RESTful microservice to maintain a leaderboard for Wordle games. Your service should expose the following operations:

- Posting the results of a game (win or loss), along with the number of guesses the user made.

  Note: A loss always requires six guesses, but some games that take six guesses result in a win.

- Retrieving the top 10 users by average score.

Note: As with Project 2, this new microservice should be decoupled from the rest of the services. In particular, the Games service does not (yet) need to report results to the Leaderboard service.

Scoring

Each game should be scored on the following scale:

Game Status	Score
Won in 1 guess	6
Won in 2 guesses	5
Won in 3 guesses	4
Won in 4 guesses	3

Won in 5 guesses	2
Won in 6 guesses	1
Lost	0

A user's score is their average score across all games that they have played.

Storage

Use Redis to persist the data for your Leaderboard service. In addition to functioning as a key-value store, Redis provides other data types such as counters, lists, sets, and maps.

To get started calling Redis from your code, see "How to Use Redis with Python" by Brad Solomon, starting with Redis as a Python Dictionary.

Installing Python libraries for Redis

You will need a Redis client library to access Redis from Python. To install both redis-py and the high-speed Hiredis parser on Tuffix 2020, use the following command:

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

Note: While the redis-py library supports asynchronous I/O and you are welcome to take advantage of it, you are **not** required to do so.

Data Model

Use appropriate Redis data types, including strings and lists to track the state of games.

Directing traffic through the API gateway

The top-10 endpoint should be accessible to the public (i.e. without authentication) through the API gateway. The endpoint for reporting results should be accessible only to internal services and not routed through the API gateway at all.

Submission

Part 1 - Submit the code

Your submission should consist of a tarball (`.tar.gz`, `.tgz`, `.tar.Z`, `.tar.bz2`, or `.tar.xz`) file containing the following items:

1. A README file identifying the members of the team and describing how to initialize SQLite and Redis, configure Nginx, and start the services

2. The modified Python source code for the Games service
3. The Python source code for the new Leaderboard service
4. `Procfile` definitions for the services
5. Initialization and population scripts for the databases
6. Nginx configuration files
7. Any other necessary configuration files

Note: Do **not** include compiled `.pyc` files, the contents of `__pycache__` directories, or other binary files, including SQLite database files. If you use Git, this includes the contents of the `.git/` directory. See Git Archive: How to export a git project for details.

If you use Quart-Schema to create a `/docs` endpoint, you do not need to update the documentation for the APIs, but be sure to update the names of functions and parameters so that they are represented appropriately in that interface.

Submit your tarball through Canvas by the due date. Only one submission is required for a team.

The Canvas submission deadline includes a grace period. Canvas will mark submissions after the first submission deadline as late, but your grade will not be penalized. If you miss the second deadline, you will not be able to submit and will not receive credit for the project.

Reminder: do not attempt to submit projects via email. Projects must be submitted via Canvas, and instructors cannot submit projects on students' behalf. If you miss a submission deadline, contact the instructor as soon as possible. If the late work is accepted, the assignment will be re-opened for submission via Canvas.

Part 2 - Team member evaluation

A Canvas Quiz due the week after the project will ask you to provide private, candid feedback on the relative contributions of each member of the team.

Grading

The project will be evaluated on the following five-point scale, inspired by the general rubric used by Professor Michael Ekstrand at Boise State University:

Exemplary (5 points)

Project is a success. All requirements met. Quality of the work is high.

Basically Correct (4 points)

Project is an overall success, but some requirements are not met completely, or the quality of the work is inconsistent.

Solid Start (3 points)

The project is mostly finished, but some requirements are missing or the quality of the work does not yet meet professional standards.

Serious Issues (2 points)

The project has fundamental issues in its implementation or quality.

Did Something (1 point)

The project was started but has not completed enough to fairly assess its quality, or is on entirely the wrong track.

Did Nothing (0 points)

Project was not submitted, contained work belonging to someone other than the members of the team, or was of such low quality that there is nothing to assess.