CPSC 449 - Web Back-End Engineering

Project 2 - Fall 2023

In this project, you will work with a team to create a new user authentication service with read replication, then use it to implement authentication and load balancing through an API gateway.

Project Teams

This project must be completed in a team of five or six. 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?

Tasks

Implement an authentication service

In Project 1, you implemented a RESTful enrollment service with functionality for three different kinds of users: students, instructors, and the registrar. While you may have implemented authorization checks for the various endpoints (e.g., that students can only drop themselves, or that only the registrar can freeze automatic enrollment), those API calls are not authenticated — anyone can make an API call on behalf of another user. A student, for example, might call the endpoint to administratively drop another student, as long as they provide the username of the instructor for the course.

Create a new RESTful service for account creation and authentication. Expose the following resources and operations:

Users resources

Each user should be identified by a *username* and *password* and should have one or more *roles*. Passwords should be stored using the method described in Simon Willison's Password hashing in Python with pbkdf2.

The API should allow users to:

- Register a new user
- Check a user's password

If a user is successfully authenticated, the service should return a set of JWT claims. For an example of the format of the claims, see the output of the `mkclaims.py` command-line tool.

Avoiding coupling between services

The users service should have its own, separate SQLite database, with its own schema, and should run independently of the enrollment service. In particular, neither service:

- Should rely on the other service to be available
- Access the database belonging to the other service
- Rely on internal details (e.g. database keys) of the other service.

The last requirement will likely mean duplicating information (e.g. usernames) across databases. While the schema in a single relational database should usually be normalized, denormalization is a common technique for performance and scaling.

It may also mean that you will need to change the SQL schema or Python code for the enrollment service.

Configure authentication through the API gateway

In order to authenticate access to the enrollment service, you will need each enrollment endpoint to be proxied through KrakenD. Configure this in a manner similar to Exercise 2.

Configure JWT token signing

Now configure KrakenD to Generate signed JWT tokens by delegating to your new service's authentication endpoint. You will need to generate a new JSON Web Key with the `mkjwks.py` command-line tool and supply it to KrakenD for use when cryptographically signing the JWT token.

Configure JWT validation

Update the configuration for each enrollment endpoint to use Basic JWT validation.

Testing

Use the login endpoint to retrieve a signed JWT, then use it to test that you can only access enrollment endpoints by including a `Bearer` in the `Authorization` header.

Tip: an easy way to send authorization headers is with HTTPie for Terminal and the httpie-jwt-auth plugin.

Configure load balancing for the enrollment service

Restart Foreman with the `--formation` switch, starting a single instance each of KrakenD and the users service and three instances of the enrollment service.Test that each service instance can be accessed separately.

Note: the original article introducing Foreman shows more than one process being run using the command-line switch `-c` (for "concurrency"), but current versions use `-m` or `--formation` instead.

Configure KrakenD load balancing to round-robin between the three instances of the enrollment service.

Testing

Load balancing is correctly configured when consecutive calls to the enrollment service through KrakenD are routed to separate instances.

You can check that load-balancing is working correctly by examining the Foreman logs. Make several requests to KrakenD for an enrollment service endpoint, and verify that they are routed to different service instances (e.g. `enrollment.1`, `enrollment.2`, and `enrollment.3`).

Add read replication to the users service

Finally, add read replication to the users service to improve the availability and performance of the users database.

Configure database replicas

Use LiteFS to create three replicas of the database for the users service: a primary and two secondaries. Configure this in a manner similar to Exercise 2.

Note: because each replica requires its own YAML configuration file, you will need to configure each process for the users service separately in your `Procfile` rather than running with `--formation`.

Update the users service

Modify the users service so that database reads caused by calls to the authentication endpoint are distributed across the two secondary replicas, while calls to register new users continue to use the primary database.

You will need to modify:

- The configuration file for the users service to include all three database URLs
- The database connection code (e.g. `get_db()`) to manage multiple connections.
- The code for each endpoint.

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

Platform

Your project must run successfully on Tuffix 2022.Provide the source code and any other necessary files (for example, data or shell scripts). You will also need to provide either a plain-text README file or documentation in PDF format describing how to use your project.

Note: this is a back-end project only. While your browser may be able to display JSON objects returned in response to an HTTP GET request, you do not need to implement a front-end or other user interface for testing API methods. You may assume that endpoints will be tested with an HTTP client program such as HTTPie or curl, or with the interactive API docs generated by FastAPI.

Libraries, tools and code

The requirements for this project are the same as for Project 1, with the addition of the KrakenD API gateway and the LiteFS distributed filesystem for replicating SQLite databases. In order to run the `mkjwk.py` command-line tool, you will need the jwcrypto library, and you may wish to use the httpie-jwt-auth plugin.

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 new functionality specified for this project** must be your own original work or the original work of other members of your current team, with the usual exceptions of sources specified in previous projects, code provided by the instructor, and documentation for the various libraries and tools. For more information, consult the Syllabus.

Documenting your results

Document your running system, including APIs, input and output formats, configuration files, and any other ancillary material such as programs or scripts in a report in PDF format.

At the beginning of your report, include the following:

- The names of each member of the team who participated in the project
- The course number, section, and semester
- The project number

For each task:

- Identify what is to be done.

- Document the tools to be used.
- Describe your results, including definitions, diagrams, screenshots, or code as appropriate.
- Comment on the results, including references you consulted, variations you considered, or issues you encountered.

Submission

Submit your work as a tarball (`.tar.gz`, `.tgz`, `.tar.Z`, `.tar.bz2`, or `.tar.xz`) file through Canvas by the due date. Only one submission is required for a 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)

The project is a success. All requirements met. The quality of the work is high.

Basically Correct (4 points)

The 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 been completed enough to assess its quality fairly or is on the wrong track.

Did Nothing (0 points)

The project was not submitted, contained work belonging to someone else, or was of such low quality that there is nothing to assess.