Assignment 3.1: Container Virtualization

Lab Group 32

Ivy Rui Wang (iwa211), Delong Yuan (dyu208)

I. Project Overview

In this assignment, we've enhanced our Flask-based URL shortening and user authentication services by incorporating a MySQL database for persistent data storage and employing an Nginx reverse proxy for unified port access. This assignment also involved containerizing the services, optimizing image sizes through multi-stage builds, and orchestrating the deployment using Docker-Compose.

II. Implementation details

A. Integration of MySQL Database

To manage and persist data, we incorporated a MySQL database to store data for both the authentication and URL shortening services.

1) **Database Schema Design:** For the authentication service, username, password and salt for password hashing are stored. For the URL shortening service, the long url, shortened url, id(a unique identifier for each record) and username are stored.
2) **Data Model Implementation:** We defined a model for the services to interact with the database. The model uses SQLAlchemy to create a database engine with a thread-safe connection pool. Then it uses a session factory for creating session objects to interact with the database in a multi-threaded environment. These sessions are used in a set of methods to perform basic operations on data in the database. For the put and update method, if an exception occurs during the process, a rollback operation will be performed to guarantee the consistency of the database.

B. Nginx Reverse Proxy Implementation

In the assignment 2, we used Flask's blueprint for creating a unified entry point for two services. In this assignment, we used Nginx to create a unified entry point for both services as required by the assignment description.

We defined routing rules that forwards requests to different back-end services. Specifically, when the request path is root (/), the request will be forwarded to the url-shortener service on port 5001. When the request path is users (/users/), the request will be forwarded to the authentication service on port 5002. The Nginx server is configured to listen on port 80, which is the default port for HTTP traffic.

C. Containerization and Docker Hub

Services are containerized with images stored on Docker Hub for easy distribution.

1) **Multi-Stage Docker Build Process:** For MySQL database containerization, the SQL script for initializing is put on the entry point for the MySQL container. For Nginx, we copy its configure file into the container to overwrite the default configure file. For the url-shorten service and authentication service, first install a series of compilation tools and dependency packages, including gcc, libressl-dev, musl-dev, libffi-dev, cryptography, etc. Then other Python dependencies listed in requirements.txt are installed. We added the "–no-cache-dir" option to reduce build-time cache usage. Copy the Python dependencies from the image builder of the previous build phase to the current image to avoid bringing build tools and dependency packages into the final application image and to reduce the size of the image. Then expose a port for the service and wait for the database to be initialized. Once the database is ready, start the service and listen to a certain port.
2) **Docker Hub:** Images for each service were pushed to Docker Hub, allowing for version control and easy deployment across different environments.

D. Docker-Compose Orchestration

To orchestrate the containers and ensure the services work together seamlessly, we wrote a docker-compose file. In the docker-compose file, we defined four services corresponding to the images hosted on Docker Hub.

1) **MySQL Service:** The MySQL service needs the root password and database name as environment, and we defined a volume for persistent data storage, located at /var/lib/mysql. This ensures that the database data remains intact across container restarts or updates.
2) **URL Shorten Service & Authentication Service:** For the url-shortener service and authentication service, we set the environment variables for connecting to the database and used 'depends_on: ' to make sure they're activated after the database service.
3) **Nginx Service:** For the Nginx service, we make sure they're activated after the other services in the same way. External access to the Nginx service is set on port 5003 of our host machine. Then Nginx can distribute traffic to different services based on different request paths.

4) **Networking:** We established a custom network named *wscb_network* to facilitate communication between the services. This network isolates our application's components, enhancing security and simplifying inter-service communication.

III. Responsibilities

Task	Ivy Wang	Delong Yuan
Specification Design	Collaborator	Collaborator
Code Implementation	Collaborator	Collaborator
Writing Report	Collaborator	Collaborator

TABLE I: Division of responsibilities between team members