PostgreSQL Integration for Log and Communication Data Storage

Tomasz Olejarczuk

Fontys

29-05-2024

Contents

PostgreSQL Integration for Log and Communication Data Storage

1. Summary

This document details the integration of PostgreSQL, a powerful open-source relational database, into the MATLAB Production Server API wrapper project. PostgreSQL serves as the central repository for storing logs generated by the MATLAB server, incoming requests from external applications, and corresponding responses. This document outlines the database schema, implementation details, and benefits of using PostgreSQL in this context.

2. Introduction

2.1 Background

Maintaining a detailed record of the interactions between external applications and the MATLAB server is crucial for several reasons:

- **Debugging and Troubleshooting:** Logs provide valuable insights into system behavior, errors, and performance issues, facilitating efficient debugging and troubleshooting.
- **Auditing and Compliance:** Storing request/response data allows for auditing of system usage, ensuring compliance with regulatory requirements or internal policies.
- **Data Analysis:** Analyzing historical data can reveal usage patterns, identify areas for optimization, and inform future development decisions.

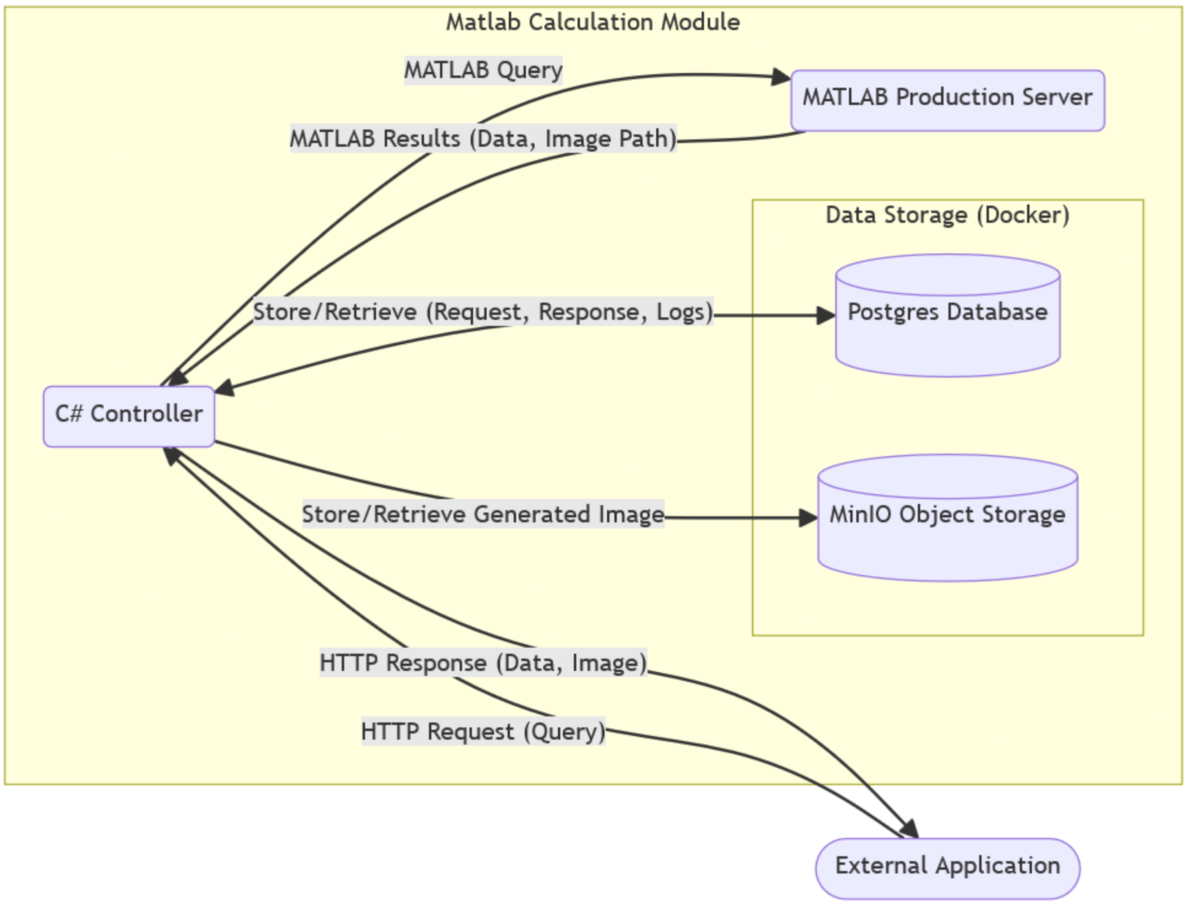
PostgreSQL was chosen for its reliability, scalability, and robust feature set, making it well-suited for managing log and communication data.

2.2 Objectives

- Design a database schema to effectively store logs, requests, and responses.
- Implement mechanisms for efficient data insertion and retrieval.
- Ensure data integrity and security.

3. Methodology

3.1 Architecture


Matlab Calculation Module
MATLAB Query
MATLAB Production Server
MATLAB Results (Data, Image Path)
Data Storage (Docker)
Store/Retrieve (Request, Response, Logs)
Postgres Database
C# Controller
Store/Retrieve Generated Image
MinIO Object Storage
HTTP Response (Data, Image)
HTTP Request (Query)
External Application

3.2 Technologies

- **PostgreSQL:** Relational database management system.
- **C#:** Programming language used for the API wrapper's interaction with PostgreSQL.
- **Entity Framework (EF):** Object-relational mapper (ORM) that simplifies database access and management in .NET applications.
- **Docker:** Containerization platform for deploying PostgreSQL.

3.3 Configuration

3.3.1 Deployment and Orchestration

PostgreSQL is deployed as a container within the same Docker Compose environment that houses the MATLAB server, API wrapper, MinIO, and RabbitMQ. This setup streamlines the management and scaling of the entire system. The `docker-compose.yml` file defines the necessary configuration for the PostgreSQL container:

```
postgres:
image: postgres:15
restart: always
environment:
    - POSTGRES_PASSWORD=$POSTGRES_PASSWORD
    - POSTGRES_DB=$POSTGRES_DB
ports:
    - 5432:5432  # Expose standard PostgreSQL port
networks:
    - matlab-network
```

3.3.2 Environment Variables (`.env`)

The configuration of PostgreSQL relies on environment variables defined in the `.env` file:

- `POSTGRES_PASSWORD`: Sets the password for the default PostgreSQL user (`postgres`).
- `POSTGRES_DB`: Specifies the name of the database to be created (`teamium` in this case).

3.3.3 Key Configuration Details

- **Image:** The `postgres:15` image is used, providing a PostgreSQL 15 server.
- **Restart Policy:** The `restart: always` policy ensures that the PostgreSQL container automatically restarts if it fails.
- **Ports:** Port 5432, the default PostgreSQL port, is exposed and mapped to the host machine.
- **Environment Variables:** The environment variables are used to set the password for the default PostgreSQL user and the name of the database.
- **Networking:** The `matlab-network` bridge network enables communication between all containers in the Docker Compose setup.

3.3.4 Database Initialization

- **User Creation (Optional):** After the PostgreSQL container starts, you can optionally create a new user (e.g., `teamium`) and grant it appropriate permissions to the `teamium` database. This can be done using the `psql` command-line tool or any PostgreSQL client.
- **Schema Creation:** The C# API wrapper, using Entity Framework, automatically creates the database schema (tables `logs`, `requests`, and `responses`) if they don't already exist when the application starts.

3.3.5 Security Considerations

- **Credentials:** It's crucial to store the PostgreSQL credentials (e.g., `POSTGRES_PASSWORD`) securely, as they provide access to the database.
- **Network Isolation:** The `matlab-network` bridge network helps isolate communication between the containers, but additional security measures (e.g., firewall rules) may be necessary depending on the deployment environment.
- **Data Encryption:** Consider enabling PostgreSQL's built-in encryption features to protect sensitive data both in transit and at rest.
- **Input Validation:** The API wrapper should implement strict input validation to prevent SQL injection attacks.

3.4 Implementation Details

3.4.1 Database Interactions with Entity Framework Core

As previously mentioned, the C# API wrapper is designed to leverage Entity Framework (EF) Core for database interactions. Here's how the planned entity model will be used:

Entity Model:

- **`MatlabQuery`:** Represents a query sent to the MATLAB server.
- **`Iteration`:** Stores details about individual iterations within a MATLAB query.
- **`CurrentIteration`:** Tracks the most recent iteration for a given MATLAB query.
- **`Member`:** Represents a member participating in the calculation.
- **`Position`:** Captures the 3D coordinates of a member's position.
- **`MatlabOutput`:** Represents the output from the MATLAB server.
- **`MatlabOutputWithImage`:** Extends `MatlabOutput` to include the generated heatmap image.

Log Storage: Initially, logs will be stored directly in the PostgreSQL database within a dedicated `logs` table. This will leverage EF Core for seamless insertion and retrieval. However, as the project evolves and WebSockets are introduced for real-time log streaming, this approach might need to be reevaluated.

Possible Modifications for WebSocket Log Handling:

- **Reduced Database Load:** To avoid overwhelming the database with constant log writes, you might consider alternative storage mechanisms for WebSocket-delivered logs. Options include:
  - **File-Based Logging:** Appending logs to a file on disk for later analysis or archiving.
  - **Dedicated Log Aggregator:** Integrating with a specialized log aggregation service (e.g., ELK stack, Graylog) to collect, process, and analyze logs from various sources, including WebSockets.
  - **In-Memory Storage:** Temporarily storing logs in memory and periodically flushing them to the database or another storage.

3.4.2 Real-Time Log Streaming with WebSockets

The API wrapper will incorporate a WebSocket endpoint to provide clients with a real-time stream of MATLAB server logs.

Key considerations for this implementation include:

- **Establishing WebSocket Connections:** The wrapper will need to establish and manage WebSocket connections with clients, ensuring proper authentication and authorization.
- **Log Formatting:** Logs will be formatted in a suitable way (e.g., JSON) for transmission over WebSockets.
- **Client-Side Handling:** Client applications will need to implement WebSocket clients to receive and display the log stream.
- **Error Handling:** The wrapper should gracefully handle WebSocket disconnections and errors.

3.4.3 Database Interaction for other entities

The other entities will stay the same and will be created, updated and retrieved from the db using entity framework methods such as `.Add()`, `.Update()` and `.Find()` or LINQ queries.

4. Results & Insights

The design and planned implementation of the PostgreSQL database integration for the MATLAB Production Server API wrapper project have yielded valuable insights:

4.1 Key Design Decisions

1. **Entity Framework Core (EF Core):** Choosing EF Core as the ORM framework simplifies database interactions by providing a layer of abstraction between the C# code and the PostgreSQL database. This allows for easier data modeling, querying, and management.
2. **Relational Database Model:** Opting for a relational database like PostgreSQL ensures data integrity, consistency, and the ability to establish relationships between different data entities (e.g., `MatlabQuery`, `Iteration`, `Member`).
3. **Table Structure:** The planned database schema, with tables for `logs`, `requests`, and `responses`, provides a clear and organized structure for storing log entries, incoming requests, and corresponding responses.
4. **WebSocket Integration (Future):** The decision to incorporate WebSockets for real-time log streaming demonstrates a forward-thinking approach to providing users with immediate feedback and insights into the MATLAB server's operations.

4.2 Anticipated Outcomes

1. **Efficient Data Storage and Retrieval:** EF Core's capabilities will enable the API wrapper to efficiently store and retrieve data from the PostgreSQL database, ensuring smooth operation and responsiveness.
2. **Scalability:** PostgreSQL's ability to handle large datasets and high traffic volumes will allow the system to scale as the number of requests and log entries increases.
3. **Data Integrity:** The relational model and the use of foreign keys will enforce data integrity, preventing inconsistencies and ensuring that relationships between entities are maintained.
4. **Improved User Experience:** The planned WebSocket integration will enhance the user experience by providing real-time log updates, enabling users to monitor the progress of their requests and quickly identify any issues.

4.3 Potential Challenges

1. **Database Performance:** As the volume of data grows, it will be important to monitor database performance and optimize queries to maintain responsiveness.
2. **WebSocket Implementation:** Implementing WebSockets for real-time log streaming will require careful consideration of connection management, error handling, and potential security implications.
3. **Data Modeling Refinement:** As the project evolves, the database schema may need to be refined to accommodate new features or changes in requirements.

Overall, the planned PostgreSQL integration is expected to provide a robust and scalable solution for storing and managing log and communication data within the MATLAB Production Server API wrapper project. The use of EF Core and the well-structured database schema will contribute to efficient data handling, while the future implementation of WebSockets will enhance the user experience by providing real-time insights into system operations.

5. Conclusion

In conclusion, the planned integration of PostgreSQL into the MATLAB Production Server API wrapper project demonstrates a thoughtful approach to data management and system observability. By leveraging the strengths of a relational database like PostgreSQL and the flexibility of Entity Framework Core, the project is well-positioned to handle the storage and retrieval of log data, requests, and responses in a structured and efficient manner.

While the database implementation is still in progress, the careful consideration given to the database schema design and the planned integration of WebSockets for real-time log streaming indicate a commitment to both functionality and user experience. The anticipation of potential challenges, such as database performance optimization and the need for alternative log storage mechanisms with the introduction of WebSockets, demonstrates a proactive approach to ensuring the system's long-term scalability and reliability.

Overall, the PostgreSQL integration is bound to be a valuable asset to the MATLAB Production Server API wrapper project, providing essential data storage capabilities and enabling future enhancements that will further optimize system performance and deliver a seamless user experience.