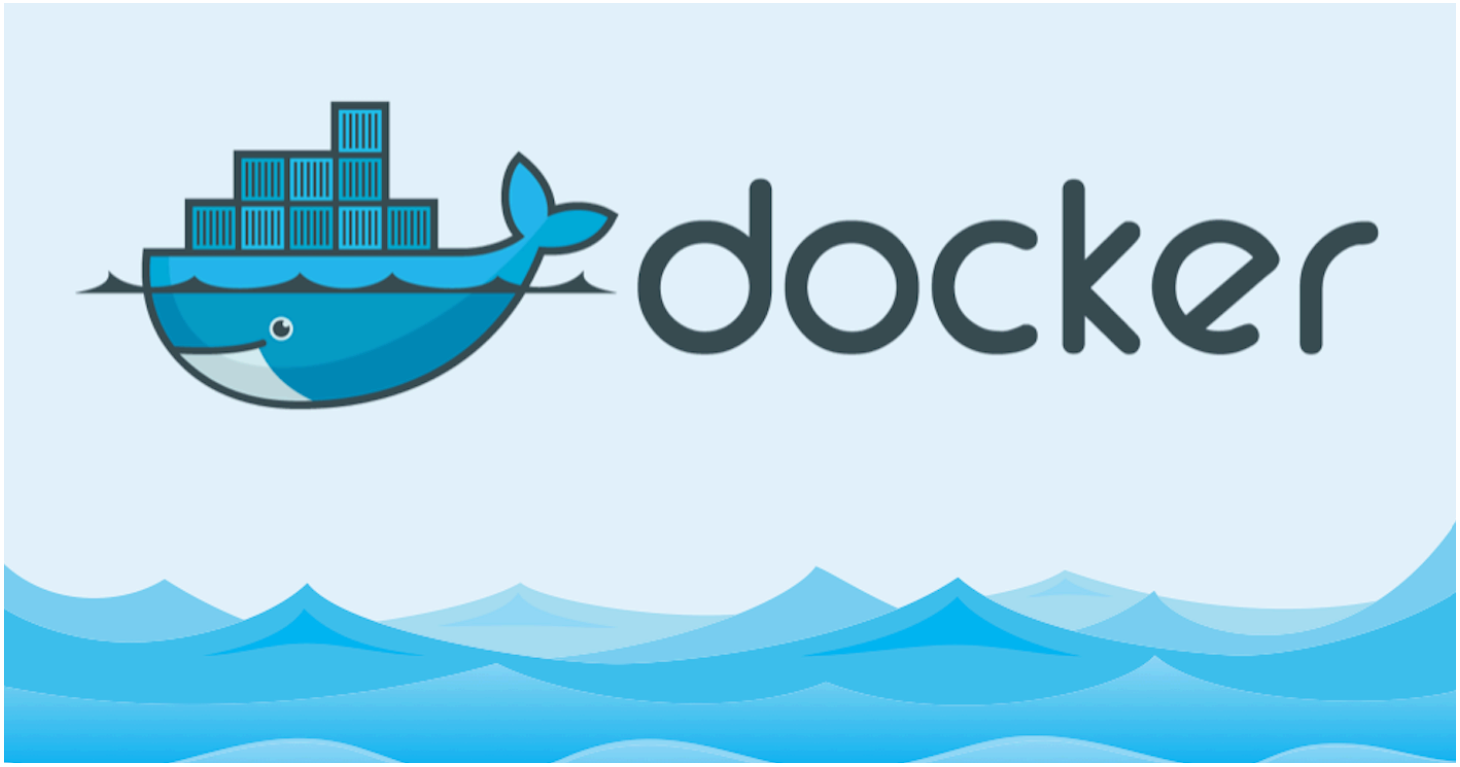
Containerized Data Collection and Analytics System



By Hamza Alhalabi

Table of Contents

Introduction

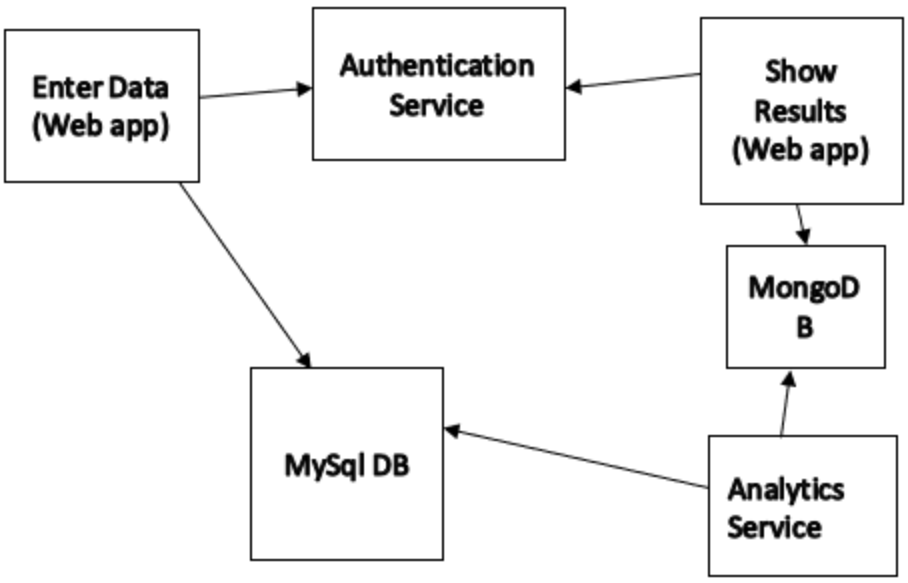
This project is an implementation of the use of **Docker** containerization and **Docker-Compose** technologies with microservices like apps, databases and other services.

It is supposed to help in understanding why Docker is a great tool for developers and how the process of building and shipping software became easier because of it.

We also created a basic **CI/CD** pipeline using GitHub Action to familiarize ourselves with the professional techniques in software engineereing.

System Architecture

The system follows a microservices architecture pattern where each component is containerized and independently deployable.



The architecture consists of five main services:

1. **Data Entry Service**: A web application that allows users to input developer data which is stored in a MySQL database.
2. **Authentication Service**: A standalone service that manages user authentication and session handling.
3. **Analytics Service**: A service that processes data from MySQL, performs statistical calculations, and stores results in MongoDB.
4. **Results Display Service**: A web application that retrieves and displays analytics data from MongoDB.
5. **Database Services**: MySQL for storing raw data and MongoDB for storing processed analytics results.

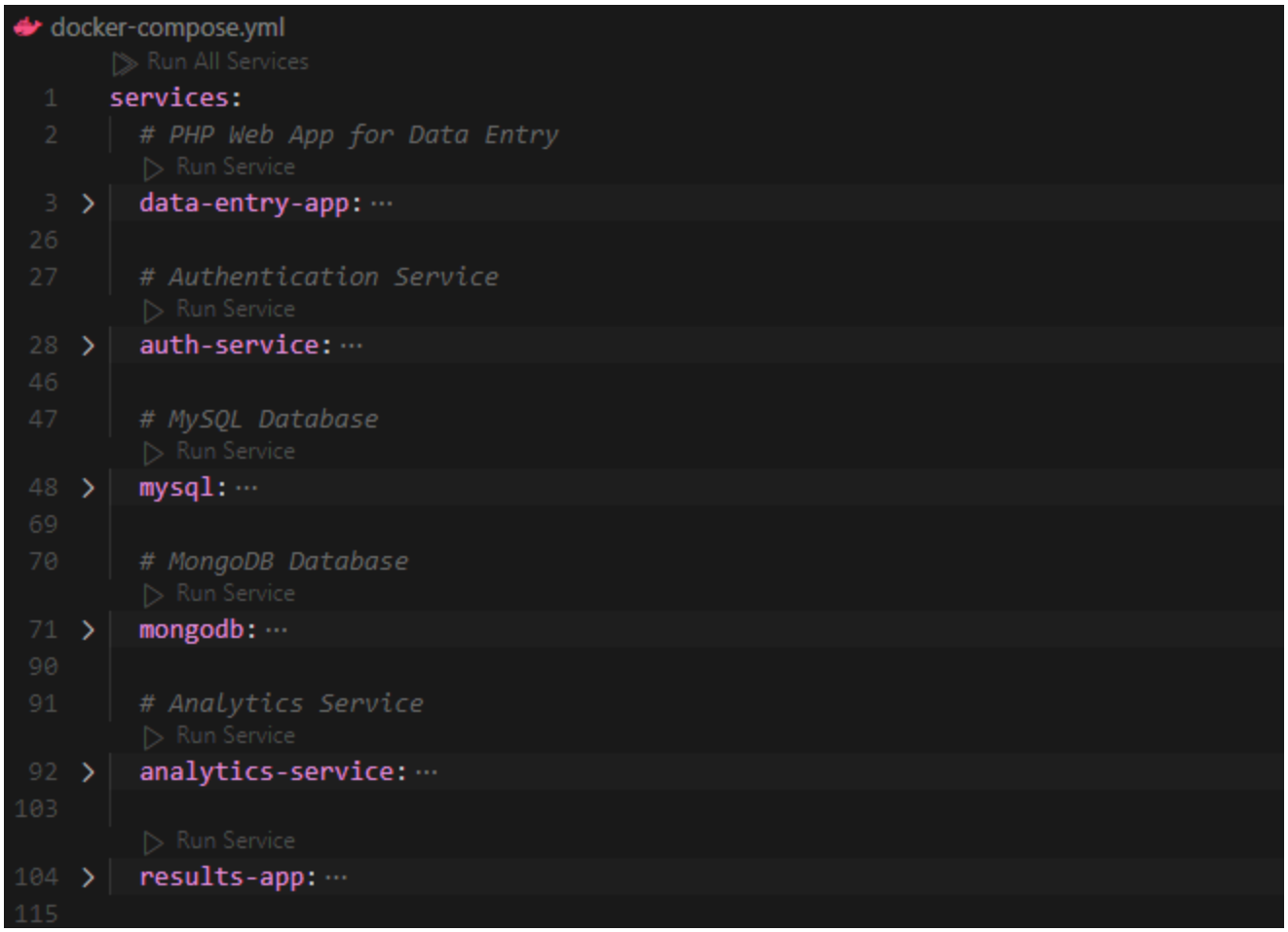
All services communicate via a Docker network bridge, enabling seamless interaction while maintaining isolation. The system is designed to be scalable, with each component running in its own container and capable of independent scaling.

Technologies and Tools

Docker and Containerization

The project utilizes the following Docker technologies:

- **Docker Engine**: Provides the core container runtime environment for all services.
- **Docker Compose**: Orchestrates the multi-container application, defining service dependencies, networks, and volumes.



- **Docker Volumes**: Persistent storage for database data and PHP sessions, ensuring data durability across container restarts.

```
volumes:
  mysql-data:
  php_sessions:
  mongodb-data:
```

- **Docker Networks**: Custom bridge network enabling secure communication between containers.

```
networks:
  app-network:
    driver: bridge
```

- **Docker Health Checks**: Implemented for database services to ensure they are fully operational before dependent services start.

```
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost", "-u", "root", "-prootpassword"]
      interval: 5s
      timeout: 5s
      retries: 5
      start_period: 30s
```

```
    healthcheck:
      test: ["CMD", "mongosh", "--eval", "db.adminCommand('ping')"]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 30s
```

Programming Languages and Frameworks

The system leverages multiple programming languages and frameworks, each one was selected based on its strengths for the specific task of each microservice:

- **PHP**: Used for the data entry application, authentication service, and results display application. PHP was chosen for its simplicity and widespread use in web development.
- **Python 3**: Powers the analytics service, utilizing libraries for database connectivity and statistical processing.
- **Composer**: PHP dependency management for the results-app service.
- **HTML & CSS**: Frontend fundamental technologies used in web applications for stucturing and styling.

Databases

The system incorporates two database technologies, this dual-database approach demonstrates a polyglot persistence pattern, where each database technology is used for its specific strengths:

- **MySQL**: Relational database used for storing structured developer data including names, programming languages, experience levels, and working hours. MySQL was chosen mainly because it was specified in the assignment instructions, and for its ACID compliance, transaction support, and schema enforcement which is ideal for the initial data collection phase.
- **MongoDB**: NoSQL document database used for storing processed analytics results. MongoDB was selected for its flexibility in storing semi-structured data and efficient retrieval of analytics documents. It provides optimal performance for the reporting and visualization requirements of the results display service.

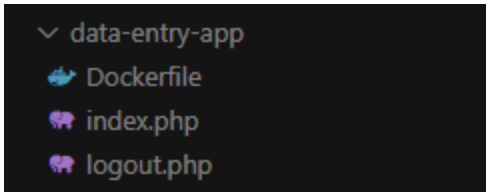
Microservices Implementation

Data Entry Service

The data entry service is implemented as a PHP web application that provides a user interface for submitting developer data. Key features include:

- **User Interface**: A responsive form with input validation for collecting developer information.
- **Session Management**: Integration with the authentication service for secure access.
- **Database Interaction**: Secure storage of collected data in the MySQL database.
- **Data Validation**: Client and server-side validation to ensure data integrity.

The service is built using PHP with minimal dependencies, making it lightweight and fast. It communicates with both the authentication service and MySQL database through the Docker network.



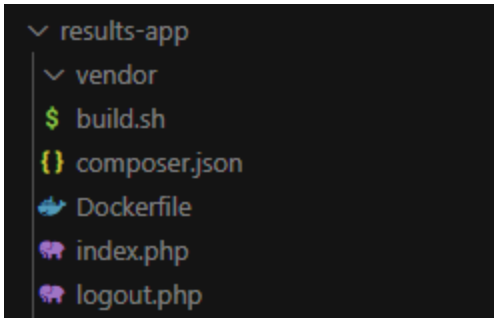
It can be accessed from `http://localhost:8080`

Results Display Service

The results display service is a PHP web application that provides visualization of the processed analytics data. Features include:

- **Dashboard Interface**: Presents statistics about developer data in an organized dashboard.
- **Real-time Updates**: Regular polling of MongoDB to display the latest analytics results.
- **Interface Design**: The page is styled to be consistent with other pages in the system.

The service uses PHP with MongoDB driver dependencies managed through Composer. It focuses on presenting insights derived from the raw data in a user-friendly manner.



It can be accessed from `http://localhost:8082`

Authentication Service

The authentication service provides centralized user management and access control for the system. Its implementation includes:

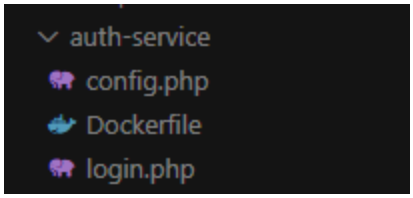
- **Login Mechanism**: Secure authentication with username/password validation.
- **Session Handling**: Creation and management of secure sessions shared across services.
- **PHP Sessions**: Uses Docker volumes to enable session persistence across containers.

```
volumes:
  - ./auth-service:/var/www/html
  - php_sessions:/var/lib/php/sessions
```

- **Configuration Management**: Centralized configuration for database connections and security settings, instead of adding another database, I just created a static users information for simplicity.

```php
$users = [
    'hamza' => [
        'password' => 'hamza123',
        'name' => 'Hamza'
    ],
    'michael' => [
        'password' => 'michael123',
        'name' => 'Michael'
    ],
    'shadi' => [ ···
    ],
    'user4' => [ ···
    ],
    'user5' => [ ···
    ],
];
```

The service is designed to be minimal yet secure, handling only the essential authentication needs of the system.



It can be accessed from `http://localhost:8081`

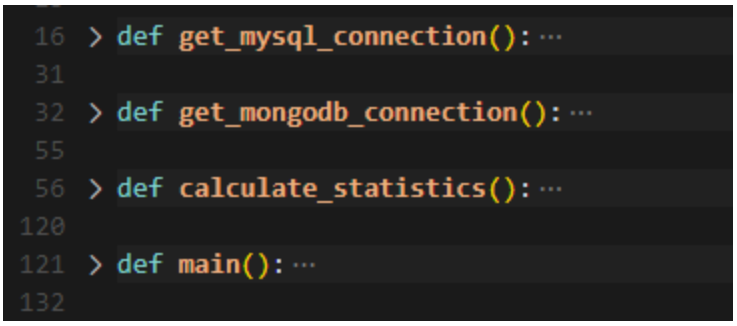
Analytics Service

The analytics service is a **Python**-based background process that performs statistical analysis on the collected data. Key aspects include:

- **Scheduled Processing**: Runs calculations on a regular interval (every minute in the current implementation).

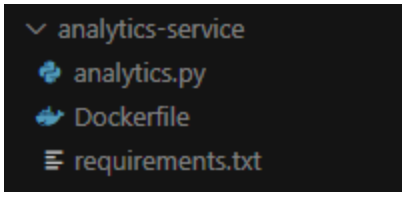
```python
while True:
    try:
        calculate_statistics()
    except Exception as e:
        print(f"Error in main loop: {str(e)}", flush=True)
        print(f"Error type: {type(e)}", flush=True)
    # Wait for 10 minutes
    print("Waiting one minute before next calculation...", flush=True)
    time.sleep(60)
```

- **Statistical Analysis**: Computes metrics such as total records, unique developers, average working hours, and experience levels.



- **Cross-Database Operations**: Retrieves data from MySQL and stores processed results in MongoDB.
- **Error Handling**: Robust error recovery with retry mechanisms for transient database connection issues.

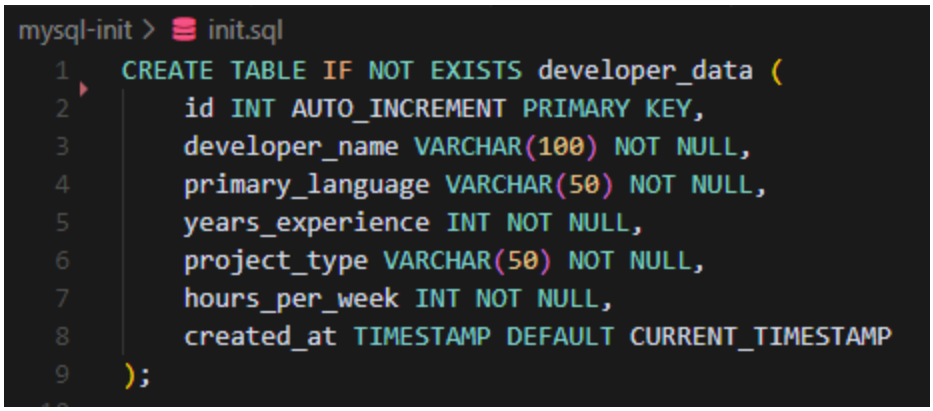
The service demonstrates a data pipeline pattern, transforming raw collected data into meaningful analytics that can be visualized in the results display service.



Database Services

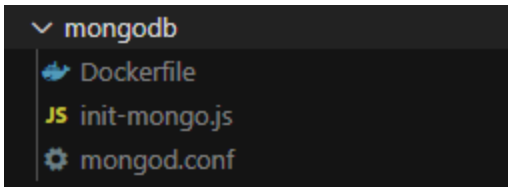
The database services are configured to provide reliable data storage and retrieval:

- **MySQL Configuration**:
  - Initialized with a schema for developer data including tables with appropriate columns and constraints.
  - Configured with specific user credentials and permissions.
  - Persistence through Docker volumes.



- **MongoDB Configuration**:
  - Custom configuration for performance optimization.

- Initialization script to create necessary collections and indexes.
- User authentication and authorization settings.
- Data persistence through Docker volumes.




```
mongodb > JS init-mongo.js > ...
1   db = db.getSiblingDB('analytics_data');
2
3   db.createUser({
4     user: 'analytics_user',
5     pwd: 'analytics_password',
6     roles: [
7       { role: 'readWrite', db: 'analytics_data' }
8     ]
9   });
10
11  // Create collections with validation if needed
12  db.createCollection('analytics_results');
```

Both databases are configured with health checks to ensure availability before dependent services start.

```
analytics-service:
  build:
    context: ./analytics-service
  depends_on:
    mysql:
      condition: service_healthy
    mongodb:
      condition: service_healthy
```

CI/CD Pipeline

The CI/CD pipeline for the project is implemented using GitHub Actions, providing basic automated building to ensure that there are no errors during the process.

Containerized-Analytics-System / .github / workflows / **docker-build.yml**

View Runs

Hamza-Alhalabi-03 Update docker-build.yml ✓ 40418a5 · 5 hours ago History

Code Blame 28 lines (22 loc) · 475 Bytes Raw

```yaml
name: Docker Build

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main
```

Open **ready to submitting** #7

Hamza-Alhalabi-03 wants to merge 2 commits into `main` from `develop`

Merge pull request #6 from Hamza-Alhalabi-03/main ... Verified bf5d505

changed login to index (names) a3c7f62

Some checks haven't completed yet

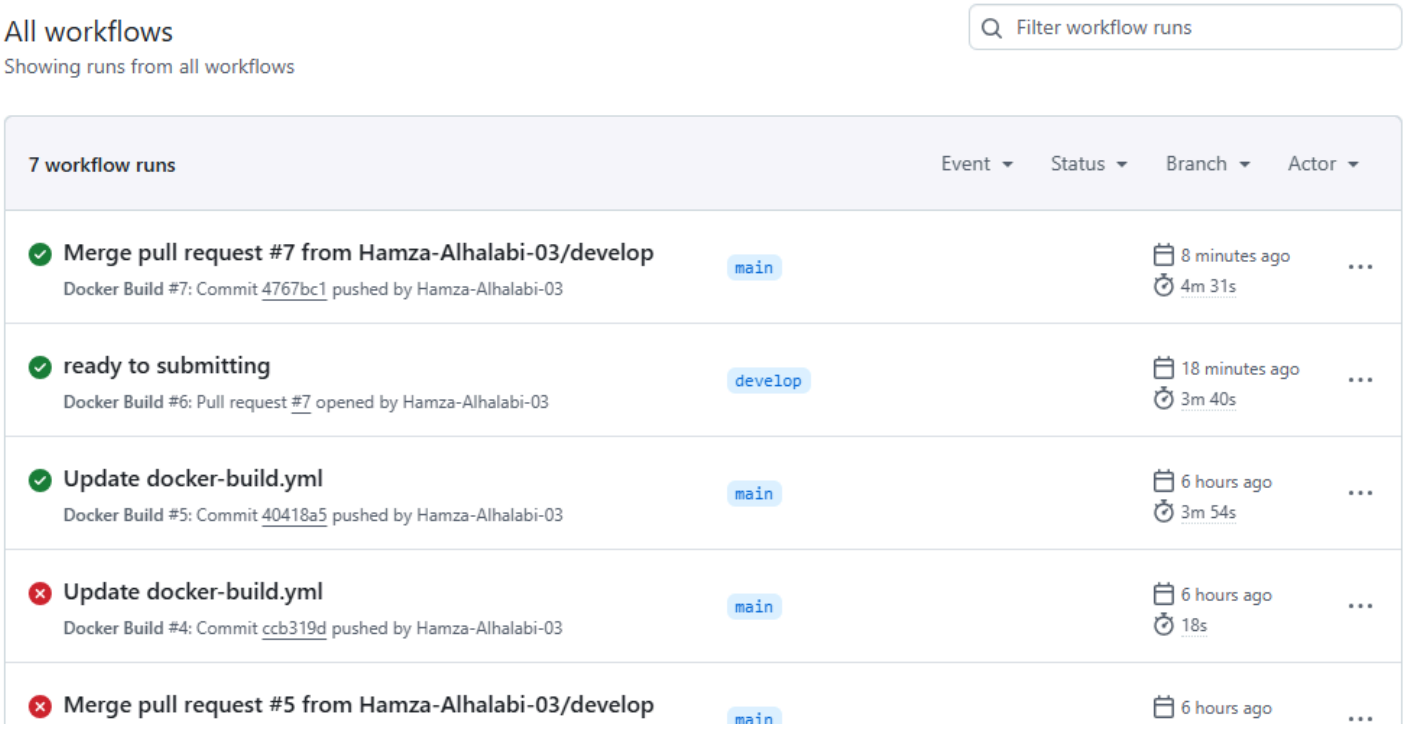
1 in progress check

Docker Build / build (pull_request) *Started now — This check has started...*

Checking for the ability to merge automatically...

Hang in there while we check the branch's status.

Merge pull request You can also merge this with the command line. View command line instructions.



Challenges and Solutions

1. **Service Dependency Ordering**: Ensuring services start in the correct order, particularly database dependencies.
    - **Solution**: Implemented Docker Compose health checks and condition-based dependency rules.
2. **Cross-Container Session Management**: Sharing PHP sessions between the data entry and authentication services.
    - **Solution**: Used Docker volumes to create a shared session storage location, and using cookies instead of normal sessions. I tried to use JWT but it did not work.
3. **Database Connectivity from Python**: Connection reliability between the analytics service and databases.
    - **Solution**: Implemented retry logic with exponential backoff for handling transient connection issues.
4. **Data Type Consistency**: Ensuring consistent data types between MySQL and MongoDB.
    - **Solution**: Added explicit type casting in the analytics service and conversion functions for decimal/float handling.
5. **Avoiding Frameworks**: I faced a problem when trying to choose the suitable technology to implement the web apps, I wanted to avoid using frameworks that I am not familiar with even they was better.

- **Solution**: I sticked to the old PHP pages, I know that it is not the best choice, but I just wanted to get the work done!

Conclusion

This containerized data collection and analytics system demonstrates the power and flexibility of Docker in creating complex microservice architectures. Key achievements include:

- Successful implementation of a multi-service containerized application
- Integration of different programming languages and database technologies
- Automated data processing pipeline from collection to visualization
- Scalable architecture that can be extended with additional services

The project showcases modern development practices including containerization, microservices architecture, and CI/CD automation.