Atypon training
Java and DevOps (May 2023)
Section 2

Docker

Microservices System

Assignment 7

Name: Mohammad Al-Qaisy

Date: 14th Aug 2023

Table of Contents

Introduction

This report centers on a meticulously designed microservices system aimed at streamlining data collection, analysis, and presentation. The system's core objectives are to enable users to seamlessly input data, obtain relevant analytics, and make informed decisions based on the extracted insights. By employing a modular approach, this microservices architecture delivers enhanced scalability, security, and efficiency, aligning perfectly with the goals of the project.

Our exploration of this microservices system will encompass the distinct roles and functionalities of each service, the mechanisms of data flow, and the interactions that enable the seamless execution of tasks. The report will provide comprehensive insights into the following key components:

Enter Data (Web App): This service serves as a user-friendly interface for data input, allowing authorized users to contribute a wide range of integer number types, from grades to temperatures. Before accessing this service, users must validate their credentials through the Authentication Service. Subsequently, entered data is seamlessly channeled to the MySQL DB service for secure storage.

Authentication Service: This vital service ensures the security and authenticity of user access. By verifying user credentials, the Authentication Service plays a pivotal role in safeguarding data and system integrity. Users are granted access to the Enter Data and Show Results services only after their credentials are validated.

Analytics Service: Focused on generating valuable insights, the Analytics Service extracts data from the MySQL DB service. It computes essential statistics such as maximum, minimum, average values and more. These calculated statistics are subsequently written to the Mongo DB Service for easy retrieval and presentation.

Show Results (web app): The Show Results service takes on the role of presenting simplified analytics to users. Just as with data entry, users undergo authentication via the Authentication Service. After successful validation, users gain access to analytics that are retrieved from the Mongo DB service.

Through this report, we will delve into the specific tasks required for the successful implementation of this microservices system. We will explore how data is collected, stored, analyzed, and presented, while emphasizing the importance of security, efficient communication between services, and the utilization of modern technologies to achieve these goals.

By the end of this report, you will have gained a clear understanding of the steps involved in developing and deploying this microservices system. Our exploration will encompass both the technical aspects of configuring the system and the underlying principles that govern the interactions between its components.

Enter Data (web app)

The "Enter Data" microservice constitutes an essential component within the microservices architecture, offering users a seamless and secure platform to contribute diverse data types and insights. This section sheds light on the functionalities, interaction flow, and code structure of the "Enter Data" microservice.

Functionality Overview

The primary objective of the "Enter Data" microservice is to facilitate user data input after validating user credentials through the Authentication Service. Upon successful authentication, users can input data, which is then processed, validated, and sent to the MySQL DB service for secure storage.

User Authentication

The authentication process is orchestrated by the Login Controller, which utilizes the credentials provided by users to authenticate against the Authentication Service. Upon successful authentication, users are granted access to input data.

Data Entry and Processing

Once authenticated, users interact with the EnterDataController to input data. The entered data is then validated and processed to ensure its integrity. Any invalid data is flagged, and users are informed of the error, preventing erroneous or inconsistent data from entering the system.

Data Storage

Validated and processed data is dispatched to the MySQL DB service via the REST API. This ensures the data's secure storage and easy retrieval when needed for further analysis and presentation.

Interaction Flow

1- User Authentication:

- Users access the login page provided by the Login Controller.
- Upon submitting valid credentials, the controller sends a POST request to the Authentication Service for verification.
- Successful authentication results in a session attribute ("loggedInUser") being set to true.

2- Data Entry:

- Authenticated users can access the EnterDataController to input data.
- The controller validates and processes the entered data, ensuring its conformity and integrity.
- In case of any invalid data, users are presented with an error message.

3- Data Storage:

- Processed and validated data is sent to the MySQL DB service using the REST API.
- The data is securely stored in the database, ready for further analysis and presentation.

Code Structure

The Login Controller handles user authentication. It utilizes a REST template to send POST requests to the Authentication Service, allowing users to validate their credentials.

The EnterDataController manages data entry. It validates and processes input data, communicating with both the Authentication Service and the MySQL DB service to ensure a seamless flow.

The Logout Controller facilitates user logout by invalidating the session.

Conclusion

The "Enter Data (Web App)" microservice exemplifies the core principles of the microservices architecture by offering a specialized functionality that seamlessly integrates into the larger ecosystem. This service plays a pivotal role in data collection while adhering to secure authentication practices and data integrity checks.

Authentication

The "Authentication" microservice plays a pivotal role in ensuring the security and integrity of the microservices system. This section delves into the functionalities, interactions, and code structure of the "Authentication" microservice.

Functionality Overview

The core purpose of the "Authentication" microservice is to validate user credentials and grant access to authorized users. This service is responsible for safeguarding the system against unauthorized access while providing a seamless user experience.

User Authentication

The AuthenticationController takes center stage in the authentication process. Upon receiving user credentials in the form of a User object, the controller leverages the UserDao to verify the provided credentials against the predefined list of users. Successful authentication results in the return of a boolean value indicating the authentication status.

Interaction Flow

User Authentication:

- Clients send a POST request to the "Authentication" microservice's login endpoint, providing user credentials encapsulated in a User object.
- The AuthenticationController receives the credentials and utilizes the UserDao to verify their authenticity.
- Based on the outcome, a boolean value (true or false) is returned in the response to indicate successful or unsuccessful authentication.

Code Structure

- AuthenticationController: This controller handles the authentication process. It receives a User object containing the username and password from the client's POST request. The authenticate method in this controller leverages the UserDao to check whether the provided credentials match those of any user in the predefined list. The outcome is communicated back to the client through a ResponseEntity containing a boolean value.
- User and UserDao Classes: The User class defines the structure of user credentials, while the UserDao class manages the list of users and provides methods to check user credentials and add new users. The UserDao class ensures that duplicate usernames are not allowed when adding new users.

Conclusion

The "Authentication" microservice serves as the gatekeeper of the microservices system, ensuring that only authorized users can access the functionalities provided by the Enter Data and Show Results services. By validating user credentials against a predefined list of users, this microservice plays a critical role in safeguarding data and maintaining system security.

MySQL

The "MySQL" microservice serves as the foundation for data persistence within the microservices architecture. This section provides a comprehensive examination of the functionalities, interaction flow, and code structure of the "MySQL" microservice, including the important role of the NumberDatabase interface.

Functionality Overview

At its core, the "MySQL" microservice is designed to facilitate secure and efficient data storage and retrieval. By leveraging the NumberDatabase interface, this microservice ensures that numeric data is seamlessly managed within the MySQL database.

Data Storage and Retrieval

The NumbersDbController orchestrates the interaction between the application and the MySQL database for both data storage and retrieval operations. The NumberDatabase interface extends CrudRepository<Numbers, Integer>, providing a streamlined mechanism for CRUD (Create, Read, Update, Delete) operations on the Numbers entity.

Interaction Flow

1- Data Storage:

- Clients send a POST request to the "MySQL" microservice's /addNumber endpoint, including an array of numbers in the request body.
- The NumbersDbController processes the request and saves the provided numbers to the MySQL database using the numberDatabase repository.
- Utilizing the NumberDatabase interface's saveAll method, the data is securely stored in the database.

2- Data Retrieval:

- Clients send a GET request to the "MySQL" microservice's /getNumbers endpoint.

- The NumbersDbController interacts with the NumberDatabase repository, which in turn utilizes the findAll method to retrieve all stored numbers from the MySQL database.
- The retrieved numbers are seamlessly returned to the client, ready for analysis or presentation.

Code Structure

- NumbersDbController: This controller acts as an intermediary between the application and the MySQL database. It leverages the NumberDatabase repository to handle data storage and retrieval operations, responding to GET and POST requests.
- Numbers: This class defines the structure of the numeric data to be stored in the database. It includes fields for the id and number, accompanied by getter and setter methods. The @Id and @GeneratedValue annotations facilitate the assignment of primary keys.
- NumberDatabase: The NumberDatabase interface extends CrudRepository<Numbers, Integer>, serving as a key interface for the MySQL microservice. It encapsulates the necessary methods for CRUD operations on the Numbers entity.

Conclusion

The "MySQL" microservice, powered by the NumberDatabase interface, plays an integral role in ensuring the secure and efficient storage and retrieval of numeric data. Through the NumbersDbController and underlying Numbers entity, this microservice exemplifies data persistence best practices while promoting seamless interaction with the MySQL database.

Analytics

The "Analytics" microservice is a crucial component of the microservices architecture, responsible for generating statistical analyses and insights from the collected numeric data. In this section, we delve into the key functionalities, interaction flow, and code structure of the "Analytics" microservice.

Functionality Overview

The primary objective of the "Analytics" microservice is to compute a wide range of statistical metrics and insights from the data stored in the MySQL database. The Analyzer class forms the core of this service, performing data analysis operations and producing various statistical results. Additionally, the AnalyzerController orchestrates the entire analysis process and communicates with other microservices to store and update analysis results.

Data Analysis

The Analyzer class encapsulates the logic for computing statistical metrics, including the sum, maximum, minimum, mode, average, median, range, squared sum, variance, and standard deviation. It accepts an array of numeric data and, upon analysis, produces insights that can be utilized for data interpretation and decision-making.

Interaction Flow

1- Initialization:

- Upon instantiation, the Analyzer class initializes an empty list of numbers and performs an initial analysis.

2- Periodic Analysis:

- The AnalyzerController is responsible for triggering periodic analysis.
- It retrieves data from the MySQL database using the mysql-db-service.
- If new data is detected, the controller performs analysis using the Analyzer class and updates the analysis results.

3- Data Storage:

- The AnalyzerController communicates with the mongo-db-service to store and update analysis results.

Code Structure

- Analyzer: The Analyzer class encapsulates the statistical analysis logic. It computes a variety of metrics and insights from a provided list of numbers, including sum, maximum, minimum, mode, average, median, range, squared sum, variance, and standard deviation.
- AnalyzerController: This controller coordinates the analysis process. It periodically fetches data from the MySQL database, triggers analysis using the Analyzer, and communicates with the mongo-db-service to store analysis results.
- AnalysisData: This class defines a data structure for storing analysis results, including metrics like sum, maximum, minimum, mode, average, median, range, squared sum, variance, and standard deviation.

Conclusion

The "Analytics" microservice serves as a vital component for generating insightful statistics and insights from the collected numeric data. Through the Analyzer class and the coordination provided by the AnalyzerController, this microservice empowers the microservices system with the ability to gain valuable information from the data stored in the MySQL database. By seamlessly communicating with other microservices for data retrieval and storage, the "Analytics" microservice enhances the overall functionality and decision-making capabilities of the system.

MongoDB

The "MongoDB" microservice plays a pivotal role in the microservices architecture by serving as a repository for storing and managing the analysis results generated by the "Analytics" microservice. This section delves into the essential functionalities, interaction flow, and code structure of the "MongoDB" microservice.

Functionality Overview

The primary purpose of the "MongoDB" microservice is to store and manage analysis data generated by the "Analytics" microservice. It utilizes MongoDB, a popular NoSQL database, to store structured documents representing analysis results. The AnalysisDbController class facilitates the interaction between the microservice and the MongoDB database, allowing data retrieval and storage.

Data Storage and Retrieval

The AnalysisDbController class is responsible for handling the data storage and retrieval operations. It exposes endpoints to save analysis data and retrieve the latest analysis results. The analysis data, represented as AnalysisData objects, are persisted as documents in the MongoDB collection named "analysisData."

Interaction Flow

1- Data Storage:

- Upon receiving analysis data from the "Analytics" microservice, the AnalysisDbController saves the data as a document in the "analysisData" collection within the MongoDB database.

2- Data Retrieval:

- The AnalysisDbController provides an endpoint for retrieving the latest analysis data. When queried, it fetches the most recent analysis results from the MongoDB collection.

Code Structure

- AnalysisDbController: This controller facilitates the interaction between the microservice and the MongoDB database. It exposes endpoints for saving and retrieving analysis data. The class manages communication with the MongoDB repository and ensures seamless data handling.
- AnalysisDataDocument: This class defines the structure of analysis data documents stored in the MongoDB collection. It includes fields for various statistical metrics such as sum, maximum, minimum, mode, average, median, range, squared sum, variance, and standard deviation.
- AnalysisDatabase: This interface extends MongoRepository and defines the operations for interacting with the MongoDB database. It provides methods to save, retrieve, and manage analysis data documents.
- AnalysisData: This class defines a data structure representing analysis results. It contains fields for various metrics, including sum, maximum, minimum, mode, average, median, range, squared sum, variance, and standard deviation.

Conclusion

The "MongoDB" microservice assumes a crucial role within the microservices architecture by providing a repository for storing and managing analysis results. By leveraging MongoDB's flexible document-based storage, the microservice enables efficient storage and retrieval of analysis data generated by the "Analytics" microservice. Through the seamless coordination between the AnalysisDbController, AnalysisDataDocument, and the MongoDB repository, this microservice enhances the overall system's capabilities and contributes to data-driven decision-making.

Show Results (web app)

The "Show Results" microservice plays a vital role in the microservices architecture by providing users with the capability to access and visualize the analysis results generated by the "Analytics" microservice. This section provides an overview of the core functionalities, interaction flow, and code structure of the "Show Results" microservice.

Functionality Overview

The primary function of the "Show Results" microservice is to retrieve the analysis data from the "MongoDB" microservice and present it to the users in a readable format. The microservice leverages the AnalysisData class to represent the analysis results, and the ShowResultsController class facilitates the interaction between the microservice and the user interface.

Data Retrieval and Presentation

The ShowResultsController class retrieves the latest analysis data from the "MongoDB" microservice and processes it to generate a human-readable summary of the results. The summary includes metrics such as the number of numbers, sum, minimum, maximum, range, mode, average, median, squared sum, variance, and standard deviation.

Interaction Flow

1- User Authentication:

- Users access the "Show Results" microservice through the login process. The LoginController validates users' credentials against the "Authentication Service" to ensure authorized access.

2- Retrieving Analysis Data:

- Once authenticated, users can access the "Show Results" page. The ShowResultsController sends a request to the "MongoDB" microservice to retrieve the latest analysis data using the /getData endpoint.

3- Generating Summary:

- Upon receiving the analysis data, the ShowResultsController processes it to generate a summary containing key statistical metrics. The summary is constructed as a list of strings, which is then passed to the user interface for display.

Code Structure

- LoginController: This controller handles user authentication for accessing the "Show Results" microservice. It communicates with the "Authentication Service" to validate user credentials.
- ShowResultsController: This controller interacts with the "MongoDB" microservice to retrieve analysis data. It processes the data to generate a summary of analysis results, including metrics like sum, maximum, minimum, mode, average, median, range, squared sum, variance, and standard deviation. The summary is then passed to the user interface for display.
- User: This class represents the user entity, storing username and password information.

- AnalysisData: This class defines the structure of analysis data objects. It includes fields for various statistical metrics, mirroring the structure of data generated by the "Analytics" microservice.

Conclusion

The "Show Results" microservice enhances the user experience by providing an intuitive interface to access and visualize analysis results. Through the coordination between the LoginController and ShowResultsController, users can securely authenticate and retrieve analysis summaries containing essential metrics. By bridging the gap between analysis data and user accessibility, this microservice contributes to informed decision-making and data-driven insights.

Data flow

1- **Enter Data Microservice:**

Users enter data through the "Enter Data" microservice's user interface.

2- **Authentication Microservice:**

User credentials are validated through the "Authentication" microservice to ensure authorized access.

3- **MySQL DB Microservice:**

Validated data is stored in the MySQL database by the "MySQL DB" microservice.

4- **Analytics Microservice:**

The "Analytics" microservice reads data from the MySQL database and performs analysis on the collected data.

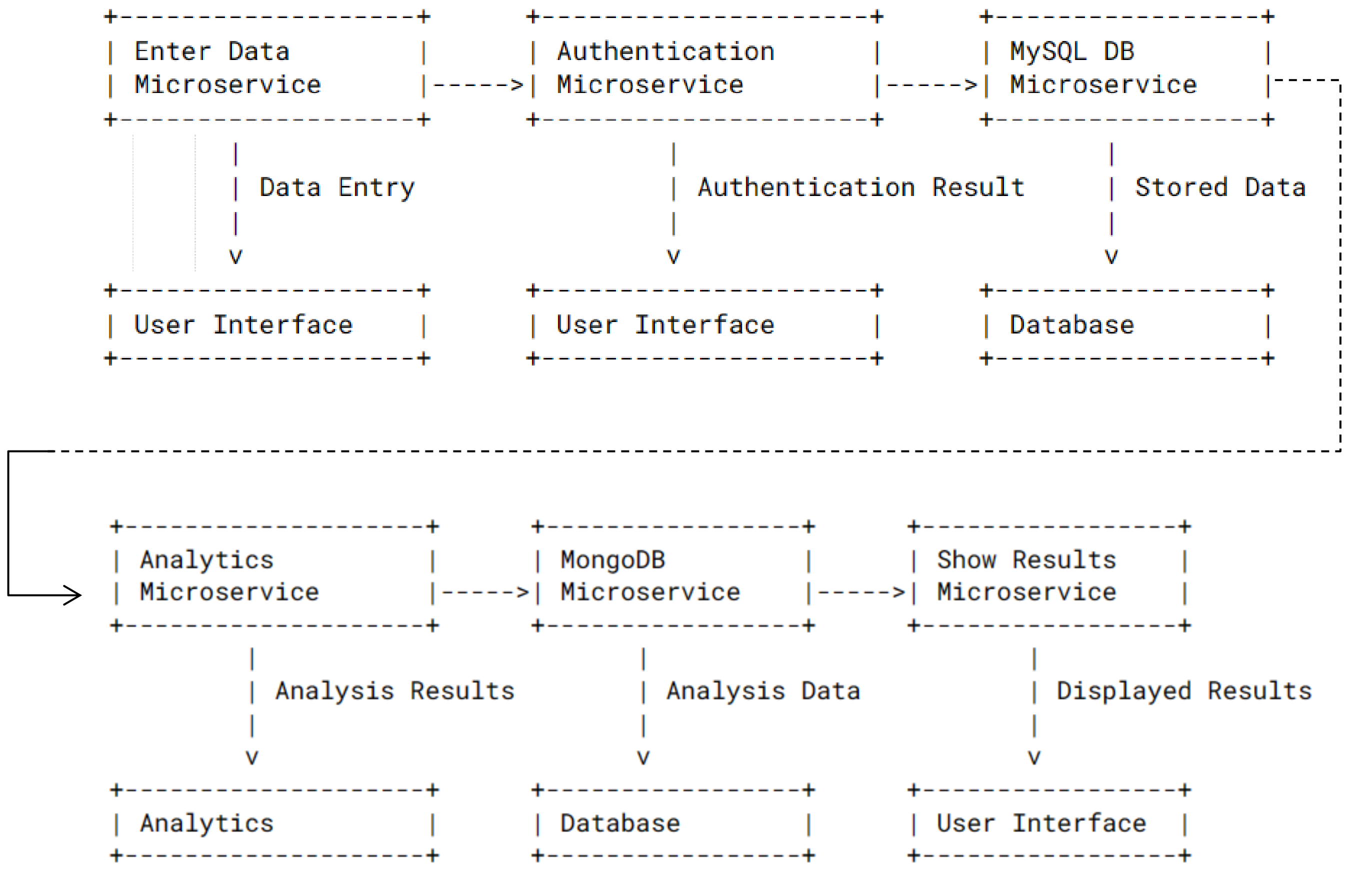
5- **MongoDB Microservice:**

The analysis results generated by the "Analytics" microservice are stored in the MongoDB database.

6- **Show Results Microservice:**

Users access the "Show Results" microservice's user interface to view the analysis results.

This data flow demonstrates how the various microservices work together to enable users to enter data, have it stored in the database, undergo analysis, and then view the analysis results. It highlights the interactions between different components and how data is passed between them to achieve the desired functionalities of your system.



```
+-------------------+     +---------------------+     +----------------+
| Enter Data        |     | Authentication      |     | MySQL DB       |
| Microservice      |---->| Microservice        |---->| Microservice   |
+-------------------+     +---------------------+     +----------------+
        |                          |                        |
        | Data Entry               | Authentication Result  | Stored Data
        |                          |                        |
        v                          v                        v
+-------------------+     +---------------------+     +----------------+
| User Interface    |     | User Interface      |     | Database       |
+-------------------+     +---------------------+     +----------------+


+-------------------+     +----------------+     +----------------+
| Analytics         |     | MongoDB        |     | Show Results   |
| Microservice      |---->| Microservice   |---->| Microservice   |
+-------------------+     +----------------+     +----------------+
        |                        |                      |
        | Analysis Results       | Analysis Data        | Displayed Results
        |                        |                      |
        v                        v                      v
+-------------------+     +----------------+     +----------------+
| Analytics         |     | Database       |     | User Interface |
+-------------------+     +----------------+     +----------------+
```

Dockerization Process

Docker is a powerful platform that allows you to package and distribute applications as lightweight, portable containers. This section outlines the Dockerization process for each microservice within the system, showcasing how Docker containers are created to encapsulate the application along with its dependencies.

Dockerfile Structure

A Dockerfile is a script used to build a Docker image. It contains a set of instructions to assemble the application's environment and configuration within a container. The Dockerfile for each microservice in your system follows a similar structure, with variations in the application name and port.

1- Base Image Selection

Each Dockerfile starts with selecting a base image that serves as the starting point for your container. In this case, the openjdk:17-jdk-alpine image is chosen, providing a lightweight Java runtime environment.

2- User Configuration

For security and isolation, a new user and group called "user" are added to the container, and the container's user is switched to this non-root user.

3- Working Directory

The working directory within the container is set to /workspace/app where the application will be located.

4- Copy Application Files

The Maven wrapper (mvnw) and application source code are copied to the container.

5- **Building the Application**

The Maven build process is initiated. Cached dependencies are utilized, and the application is built using the mvnw script. The -DskipTests flag skips running tests during the build process.

6- **Extracting Dependencies**

After the build, a new layer of the image is created to extract the application's dependencies. The application's compiled JAR file is unpacked into the /app directory within the container.

7- **Final Image**

A new Docker image is created based on the Alpine Linux image. The /app directory, containing the application and its dependencies, is copied into the final image.

8- **Exposing Ports**

The container is configured to expose a specific port. For each microservice, the exposed port corresponds to the port on which the application is intended to run.

9- **Entry Point**

The ENTRYPOINT specifies the command to run when the container starts. In this case, the Java application is launched with the required classpath to execute the microservice.

Conclusion

The Dockerization process encapsulates each microservice and its dependencies within a container, ensuring consistency and portability across different environments. By following the structured Dockerfile format, your microservices can be easily built, deployed, and managed using Docker containers.

Docker Compose Configuration

Docker Compose is a tool for defining and running multi-container Docker applications. It allows you to define all your microservices' configurations and their dependencies in a single YAML file. This report provides an overview of the Docker Compose configuration for your microservices-based application.

Docker Compose Structure

The Docker Compose configuration provided defines a set of services, each representing a microservice within your application. Each service includes the following key attributes:

- build

  This attribute specifies the build context for the service. It indicates the path to the Dockerfile and associated files for building the Docker image of the service.

- ports

  The ports attribute maps the host machine's ports to the container's ports, allowing external access to the microservices. The format is <host-port>:<container-port>.

- restart

  The restart attribute indicates the restart policy for the container. The always value ensures that the container restarts automatically in case of failure or shutdown.

- depends_on

  The depends_on attribute lists the services that the current service depends on. Docker Compose ensures that dependent services are started before the current service.

- environment

  The environment attribute sets environment variables for the service, which can be used within the application. It's often used to configure database connection details or other runtime configurations.

Microservices Configuration

The Docker Compose configuration defines the following microservices:

- **enterdata-service:** Represents the Enter Data microservice. It depends on mysql-db-service and authentication-service.
- **mongo-db-service:** Represents the MongoDB microservice. It depends on mongodb.
- **mysql-db-service:** Represents the MySQL microservice. It depends on mysqldb. It defines environment variables for database configuration.
- **authentication-service:** Represents the Authentication microservice.
- **analytics-service:** Represents the Analytics microservice. It depends on mysql-db-service and mongo-db-service.
- **showresults-service:** Represents the Show Results microservice. It depends on authentication-service and mongo-db-service.
- **mysqldb:** Represents the MySQL database service. It sets up a MySQL database container with the specified environment variables.
- **mongodb:** Represents the MongoDB database service.

Conclusion

Docker Compose simplifies the management of your microservices-based application by orchestrating the deployment of multiple services and their dependencies. This configuration file provides a clear structure for defining and running your microservices in a consistent and efficient manner.