MINISTRY OF EDUCATION AND RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

DS2023_30243_Assignment_3

Varvari Cosmin-Ionut

1. Introduction

1. Introduction

In the rapidly evolving landscape of modern software development, the adoption of microservices architecture has become increasingly prevalent. This document provides a comprehensive overview of our distributed system, which comprises three key microservices: UserService, DeviceService, and SDProject-Frontend. Leveraging cutting-edge technologies such as Spring Boot, Docker, RabbitMQ, WebSockets, and Angular, our distributed system is designed to deliver a robust, scalable, and efficient solution for managing users, devices, and providing an intuitive frontend interface.

1.1 Purpose

The primary purpose of our distributed system is to offer a flexible and modular approach to application development. By breaking down the application into distinct microservices, we aim to enhance maintainability, scalability, and enable independent deployment of components. Each microservice addresses a specific domain, contributing to the overall functionality of the system.

1.2 Microservices Overview

1.2.1 UserService

The UserService microservice is responsible for managing user-related functionalities within the system. Leveraging the Spring Boot framework, it ensures seamless user authentication, and authorization.

1.2.2 DeviceService

DeviceService focuses on handling device-related operations. Built with Spring Boot, it facilitates the registration, monitoring, and management of devices connected to the system.

1.2.3 SDProject-Frontend

SDProject-Frontend serves as the user interface of our application, developed using the Angular framework. This microservice provides an interactive and user-friendly platform for users to interact with the underlying functionalities of UserService and DeviceService.

1.3 Technologies Used

Our distributed system embraces the following key technologies:

Spring Boot: A powerful framework for building Java-based microservices, providing a robust and scalable foundation for UserService and DeviceService.

Docker: Containerization technology that ensures consistency in deployment across various environments, simplifying the management of microservices.

RabbitMQ: Message-oriented middleware facilitating asynchronous communication for the Monitoring and Communication Microservice in Assignment 2.

Angular: A popular front-end framework for building dynamic and responsive user interfaces, employed in the development of SDProject-Frontend.

WebSockets: Real-time bidirectional communication technology utilized in the Chat Microservice in Assignment 3.

This combination of technologies empowers our distributed system with flexibility, scalability, and the ability to adapt to the dynamic requirements of modern applications.

2. Conceptual Architecture

2.1 Overview

The conceptual architecture of our distributed system is designed to foster modularity, scalability, and efficient communication between microservices. At its core, the architecture follows a microservices pattern, allowing each component to operate independently while contributing to the overall functionality of the system.

The key components of our architecture are:

UserService: Manages user-related functionalities, ensuring authorization, and user data management.

DeviceService: Handles operations related to devices, offering features such as device registration, monitoring, and management.

SDProject-Frontend: Serves as the user interface, providing an intuitive and responsive platform for users to interact with UserService and DeviceService.

These microservices collectively form a cohesive ecosystem, each specializing in a specific domain to fulfill the overarching goals of our distributed system.

2.2 Microservices

2.2.1 UserService

Technology Stack: Spring Boot, Docker

Description: UserService is dedicated to handling user-related functionalities within our system. Leveraging the Spring Boot framework, it ensures secure user authentication, authorization, and efficient user data management.

Deployment: The UserService is encapsulated within a Docker container, allowing for consistent deployment across various environments. It is connected to the 'backendservices' network, facilitating seamless communication with other microservices.

2.2.2 DeviceService

Technology Stack: Spring Boot, Docker

Description: DeviceService is focused on managing functionalities related to devices. It provides features for device registration, monitoring, and management, enhancing the overall system's capability.

Deployment: Similar to UserService, DeviceService is deployed within a Docker container and is part of the 'backendservices' network, ensuring smooth communication with other components of the distributed system.

2.2.3 SDProject-Frontend

Technology Stack: Angular, Docker

Description: SDProject-Frontend serves as the user interface for our application, offering an interactive platform for users to engage with UserService and DeviceService functionalities.

Deployment: Encapsulated in a Docker container, SDProject-Frontend is seamlessly connected to the 'backendservices' network, ensuring efficient communication with the underlying microservices.

2.3 Communication

Our microservices communicate with each other through the 'backendservices' network, establishing a secure and reliable channel for data exchange. The communication follows RESTful API patterns, allowing for straightforward interactions between microservices. This approach enhances the maintainability and scalability of the system, as each microservice remains independent, yet interconnected.

In the subsequent sections, we will illustrate the deployment structure of our distributed system using a UML Deployment Diagram and delve into additional details to provide a comprehensive understanding of our architecture.

3. UML Deployment Diagram

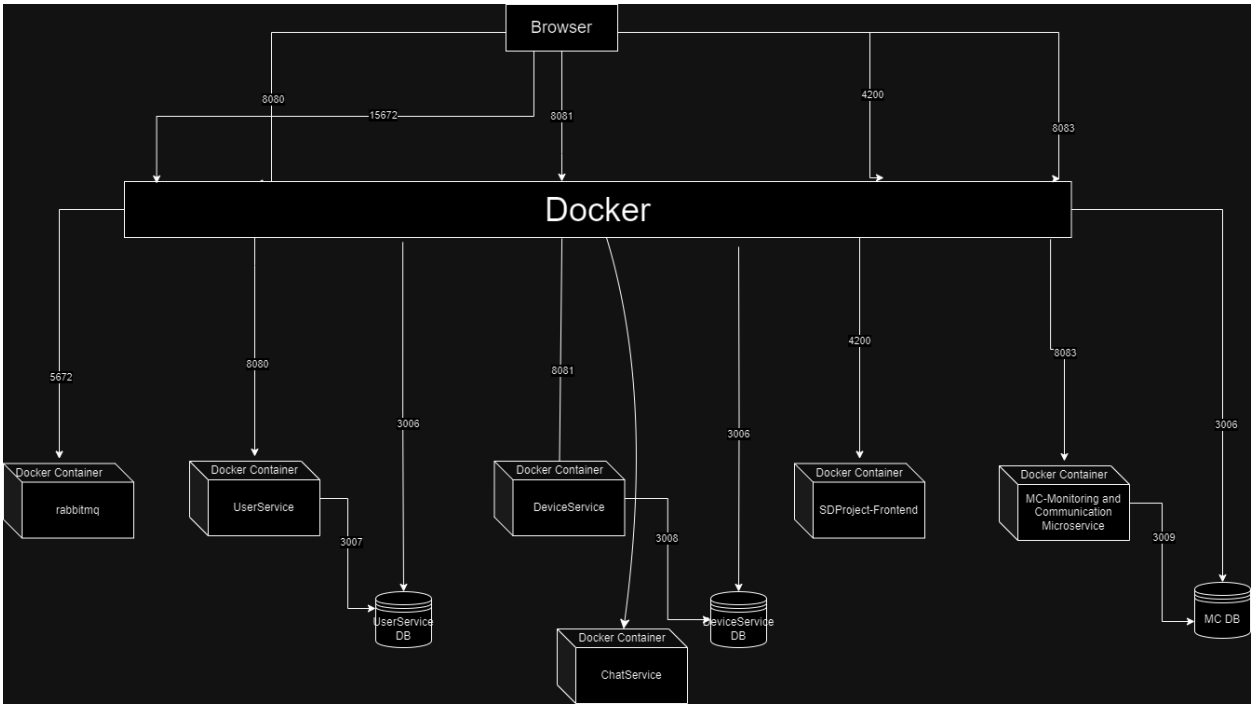
Diagram Description:

In the UML Deployment Diagram, we illustrate the physical deployment of our distributed system, using the following elements:

- Three nodes to represent the physical entities: UserService, DeviceService, and SDProject-Frontend.

- Docker containers within each node, encapsulating the corresponding microservices.

- 'Backendservices Network' connecting all nodes, facilitating communication between microservices.



4. Conclusion

In conclusion, our conceptual architecture and UML Deployment Diagram provide a comprehensive view of our distributed system, showcasing the synergy of microservices—UserService, DeviceService, and SDProject-Frontend—orchestrated through the 'Backendservices Network.' The chosen technologies, including Spring Boot, Docker, and Angular, contribute to a robust and scalable system architecture.

4.1 Key Points:

Microservices Overview:

UserService, responsible for user-related functionalities.

DeviceService, managing device-related operations.

SDProject-Frontend, delivering an intuitive user interface.

Monitoring and Communication Microservice: Processes energy data asynchronously and notifies users in real-time.

Conceptual Architecture:

Adopts a microservices pattern for modularity and scalability.

Nodes represent physical or virtual resources, each housing Docker containers.

'Backendservices Network' facilitates seamless communication between microservices.

UML Deployment Diagram:

Illustrates the physical deployment structure of our system.

Nodes and containers visually represent the distribution of microservices.

Arrows denote deployment relationships from the network to nodes and containers.

4.2 Benefits of the Chosen Architecture and Technologies:

Scalability:

Microservices enable independent scaling of components.

Docker containers provide consistency in deployment, aiding scalability.

Maintainability:

Modularity enhances maintainability, allowing updates to specific microservices without affecting the entire system.

Docker containers encapsulate dependencies, simplifying maintenance and updates.

Flexibility:

Microservices architecture allows for flexibility in technology choices for each component.

Docker facilitates portability, enabling deployment across diverse environments.

Efficient Communication:

'Backendservices Network' fosters efficient communication between microservices.

RESTful API patterns enhance interoperability and ease of integration.

Isolation and Resilience:

Containerization isolates microservices, preventing the propagation of failures.

Spring Boot's resilience features contribute to the overall robustness of the system.

4.3 Future Considerations:

As our distributed system evolves, continuous monitoring, feedback loops, and periodic architectural reviews will be crucial. This will ensure that the chosen architecture adapts to changing requirements and remains aligned with our goals of scalability, maintainability, and flexibility.

In summary, the conceptual architecture and UML Deployment Diagram lay the foundation for a resilient, scalable, and efficient distributed system, empowered by modern technologies. The selected approach positions us well for future growth and adaptability in the dynamic landscape of software development.