DISTRIBUTED SYSTEMS 2020

Assignment 3

Remote Procedure Call

Pill Dispenser Application

Rusu Andrei Paul, 30441

2020

Contents

Introduction

The purpose of this assignment was to develop a distributed system using remote procedure call mechanisms.

The assignment is based on the domain of the previous assignments and has as users Doctors, Patients and Caregivers, whose roles have been described in the previous assignment. The main goal was to design a Pill Box application that would keep track of a patient's daily medication intake and display the prescribed medication that has to be taken at each point in time.

The main components of the system will be shown and described below.

Conceptual Architecture

The solution is comprised of 4 different modules, as seen in the conceptual architecture below.

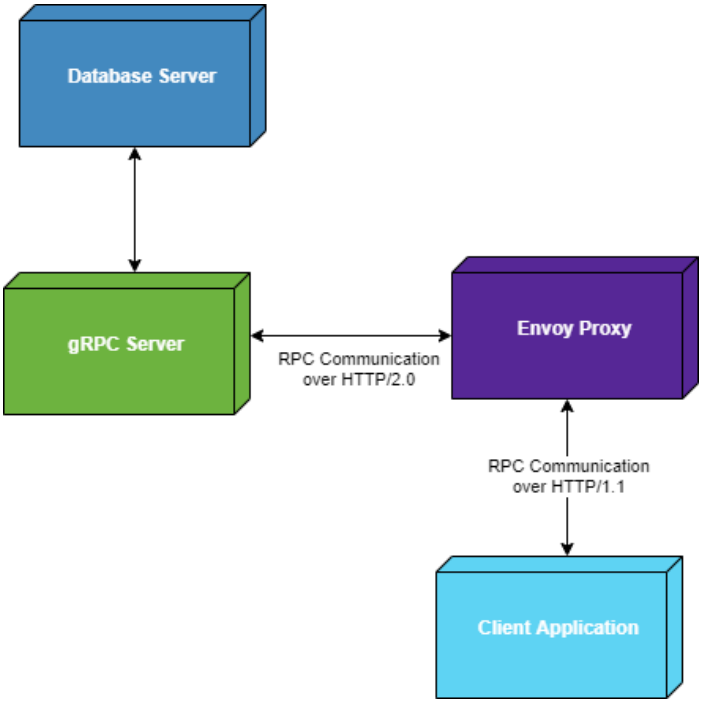

FIGURE 1: CONCEPTUAL ARCHITECTURE

There is Client application that runs in a browser and displays, in real time, a counter that the patient can see. Also, it displays the list of medication that has to be taken at that certain time, with a button next to each one, in order to mark it as taken. If the allocated time passes, the medication is marked as not taken. This system uses gRPC for communication over HTTP/2.0. Since browsers don't support this protocol, an additional component is needed, which translates HTTP/1.1 to 2.0, in order for the message to be received by the gRPC server. For this purpose, an Envoy Proxy is used, that acts as forwarding middleware. Finally, the gRPC server interacts with the database on its own.

Each component is further detailed below:

- **gRPC Server:** a gRPC server running alongside a regular Spring REST server, that intercepts gRPC requests that follow a defined service interface defined in special ProtoBuff (Protocol Buffer) format. These services define the methods and variables used in the RPC calls;
- **Envoy Proxy:** a required middleware solution for translating HTTP/1.1 data to HTTP/2.0 (and vice-versa), in order for the gRPC server to receive it correctly and for the client browser to be able to receive the response correctly;
- **Database Server:** the main datastore of the application, this is where the patient's activity is recorded. PostgreSQL is used as the database engine;
- **Client Application:** the front-end of the system, where caregivers are notified of the misbehaviors of their patients. This is a ReactJS application.

Deployment

The entire system runs within Docker, with each component having its separate Docker container.

The previously deployed applications to the Heroku Cloud remain in place, while the Envoy Proxy container is added and interacts with the local browser. The database contents carry on from the previous assignments and provide the medication data that is displayed to the patient.

The gRPC calls that are executed are defined in the PillBox.proto file and contain definitions for methods that are called when the patient has taken the medication in time, late, or when the need arises, to download a patient's medication plan.

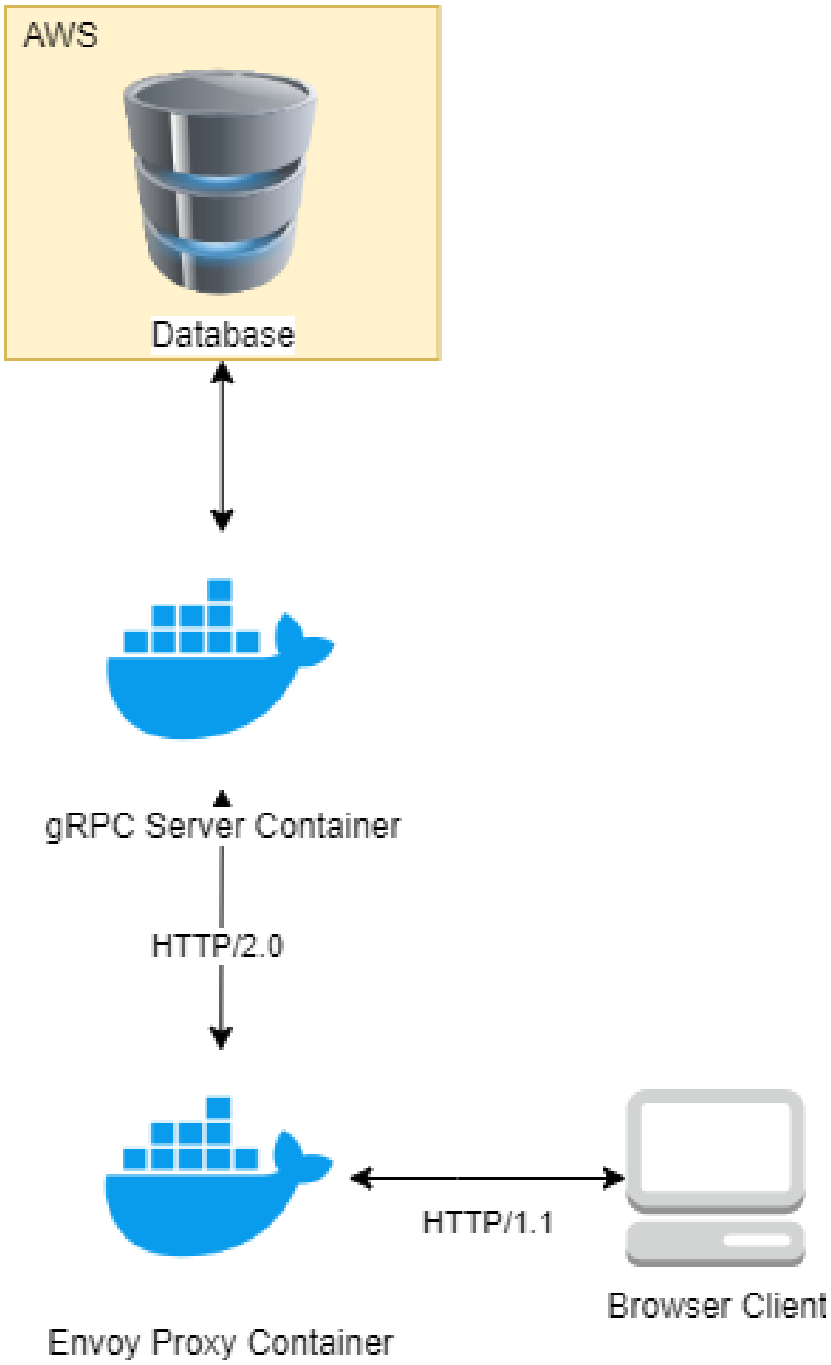

FIGURE 2: DEPLOYMENT DIAGRAM

Build and execution

The project uses Docker for both the local and the production environments. The most important file in this case is the Dockerfile, which specifies how the application should be packaged and run. Both the backend and frontend apps contain this file. Additionally, the Envoy Proxy also uses this type of running environment.

To build a Docker image from the Dockerfile of the Spring server, we run:

docker build -t spring-app .

To run the built image, we run:

docker run -p 8080:8080 --name SpringApp-docker spring-app

where 8080 is the port that will be exposed to the local development machine and will be listening for requests.