Distributed Systems - Assignment 2

Asynchronous Communication

Sensor Monitoring System and Real-Time Notification

Conceptual architecture of the distributed system:

|

Message Consumer

(Spring App)

Message Producer

(Maven app)

Message broker (RabbitMQ queue)

We have a Java application with 3 modules representing the three parts of this distributed system:

-the first one is the message producer, a simple maven app that reads each activity received from the sensors from the activity.txt file, connects to the RabbitMQ Cloud message broker creating a queue and using it to store the activities one by one

-the second one is the message broker -RabbitMQ in this case- that offers theh possibility to exchange messages between clients

-it has the queue where the producer sends the messages and keeps them for

-I’ve initially used local RabbitMQ and then I switched to Cloud AMQP so that I can make the deploy on Heroku

when the consumer app consumes them

-the third one is the message consumer, a Java App built using Spring framework that also connects to RabbitMQ to the same queue that the producer app and takes the messages that get to the destination in this way.

In order to use the message-exchange application, it is necessary to run the consumer app(Spring Boot) which will be in a state of listening to any new message arrived on the declared queue and then to start also the producer app that will actually send the messages one by one from the activity.txt file populated with the data received from the patient’s sensors.

With every message that it consumes, the spring app should test if it regards the patient’s sleeping, toileting or leaving time and check if the duration of that activity is larger than the one admitted, notifying(not implemented yet) the patient’s caregiver as an alert that something could be wrong.

RabbitMQ is lightweight and easy to deploy on premises and in the cloud. It supports multiple messaging protocols. RabbitMQ can be deployed in distributed and federated configurations to meet high-scale, high-availability requirements. I used this message broker declaring the same queue in the producer and also in the consumer so that they can exchane messages.

Deployment diagram:

Configure .gitlab-ci.yml

Setting CI/CD stages

GitLab setup repository

Producer app -Maven

Message broker-RabbitMQ

Postgres Server

NGINX server(frontend)

Tomcat server(backend)

browser

Develop&push on gitlab

Deploy stage

Package stage

Codestyle stage

Test stage