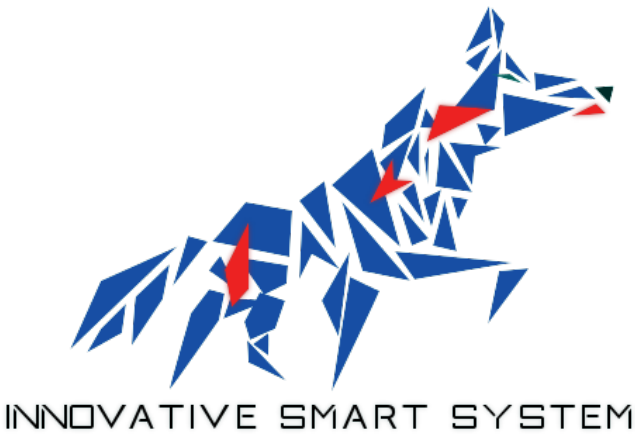

INSA Toulouse - 5ISS

Service Architecture Report

Students :
Benjamin Abonneau
Kenza Bouzergan
Lamiaa Housni
Stig Griebenow

Teacher :
Nawal GUERMOUCHE

January 22, 2024

Contents

1 Introduction

This project report documents the creation of a microservice architecture as a backend for a floating satellite beacon solution as part of the Service Architecture course at INSA. The goal was to develop and deploy a microservice architecture. Throughout the report, we describe our step-by-step approach and the technical details of this cloud architecture.

1.1 Project Background

Our project was born out of a gap in the market - a lack of cost effective floating satellite beacons. ACTIA recognized this gap and the critical need for affordable solutions, especially for applications where budget constraints are a challenge for customers. One example is the small-scale fishing sector, where individuals, such as a Thai fisherman managing his nets, yearn for a tracking system but are constrained by the cost of existing options. In essence, our project solves the problem of an untapped demand for low-cost floating satellite beacons in a variety of applications. This problem led us to the following solution.

1.2 Solution

Our solution consists of four components (as seen in figure 1), the beacon itself, an application for configuring the beacon and viewing the data, the Kinéis satellite service, and a cloud architecture for storing and managing the collected data. This report only details the development of the cloud.

2 Requirements

In the following, we will describe the requirements for the micro service architecture. The we got the following requirements regarding the cloud.

- CLOUD shall be designed in order to be cybersecure-compliant
- CLOUD should save sensor data send by BEACON though Kineis.
- CLOUD should save configuration data of BEACON.
- CLOUD should save location data send by BEACON though Kineis.
- CLOUD should be "agnostic" (ACITA should not be stuck with one provider)
- CLOUD should allow a user management.

3 Architecture

3.1 Cloud Architecture

Our microservice architecture is shown in the deployment diagram in Figure 2. As you can see, we have the microservices BeaconService, UserService, and DataService. In addition,

we have a Eureka service and a Configuration service. For the database we have chosen Redis.

Each of our microservices can communicate with the database to modify, add or delete data in the form of an API. The first microservice manages beacons, allowing you to add, delete or simply display the beacons owned by the customer. The second microservice does the same thing, but with the clients, enabling them to connect to the application. Last but not least, the Kinéis service manages data (especially sensor data). The database uses Redis technology, where each key is associated with a value. The value can be a string, an integer or an array. The array is the most commonly used form, as it enables us, for example, to store several pieces of information about a single user (login, password, email, registration date, etc.).

3.2 App

The project also includes a customer-oriented mobile application for managing beacons. The application's first interface takes the form of a login page, during which the customer is invited to connect to his account (his customer information will be stored on the Cloud). Once logged in, the customer has access to various interfaces, including one for configuring or modifying each of the beacons he owns, and another for viewing global statistics.

4 Sprints

In this chapter we will describe the 5 Sprints of the project.

4.1 Sprint 1: 12.12.2024 - 19.12.2024

The first sprint was all about laying the groundwork for our future work, so we focused on designing our microservice architecture (as described above) and getting our development environment up and running. We decided to use IntelliJ IDEA as our IDE because of its great integration with Spring Boot, which was our choice of technology for developing the services. We then set up our GitHub with 3 Spring Boot projects, 1 for each service.

4.2 Sprint 2: 20.12.2024 - 22.12.2024

The second sprint focused on the initial implementation of the services. Since we were using redis, we decided to use the "starter-redis" dependency to connect to our database. We implemented the following APIs:

- **addUser** Creates a user in the database.
- **deleteUser** Deletes a user from the database.
- **getUser** Returns a user from the database (by id).
- **createBeacon** Creates a beacon in the database and gives back a random password.
- **getBeacon** Returns a beacon from the database (by id).
- **getAllBeacons** Returns all beacons from the database.

- **getAllBeaconsByUser** Returns all beacons from the database assigned to a specific user.
- **saveData** Creates a DataPoint in the database.
- **getDataPoints** Returns all DataPoints from the database send from a specific beacon and in a specific time-intervall.
- **saveLocation** Creates a Location in the database.
- **getDataLocations** Returns all Locations from the database send from a specific beacon and in a specific time-intervall.

4.3 Sprint 3: 22.12.2024 - 05.01.2024

We went home to our families and celebrated Christmas, New Year's and recharged for the final sprints.

4.4 Sprint 4: 06.01.2024 - 14.12.2024

The fourth sprint was spent implementing the Azure deployment via GitHub actions, the discovery service, and externalization.

4.5 Sprint 5: 15.12.2024 - 21.12.2024

In our fifth and final sprint, we had a requirement change and needed to redevelop the data service. After requesting it 1.5 months ago, we finally got access to the Kinies API documentation and now know how the actual data will be structured. We had to make a lot of changes to transform the Kinies data format into our format. Once that was done, we cleaned up the code and merged all our feature branches.

5 Conclusion

First, we will discuss the limitations of our implementation, and then we will conclude the report.

5.1 Limitations

In this section, we will discuss the limitations of our deployed architecture.

5.1.1 Security

Our biggest limitation is that we have not implemented any security systems. the http traffic is not encrypted and could easily be read by an attacker. we also have no authentication systems. anyone could just create, change and delete data. We did not implement these measures because of time constraints. But they should be implemented if the architecture is ever used in production.

5.1.2 Post API Tests

Another limitation is that we were not able to test the Post API with the actual Kineis API and had to test it with sample post requests via postman. This was due to the fact that the Kineis API does not have a test function and only works when actual data is sent through the satellite, which we were not able to do (yet).

5.1.3 Unit Tests

Because of time constraints we only implemented Unit Tests for the DataService.

5.1.4 Externalization

Because of time constraints we only externalized the link to the discovery service.

5.2 Conclusion

We implemented a microservice architecture to manage the beacons and the data they send. We implemented three services: BeaconService, DataService, and UserService. We also deployed discovery and configuration services and implemented unit tests. Our deployed architecture works fine, but it has two limitations. It is not cybersecure, and the Post API could not be tested in a real-world scenario, only using fake post requests via postman.

5.3 Link to GitHub

Click here!

Note: We will continue to develop the architecture and created a branch Software-Architecture which freezed the code base at the date of delivery.

PS: Link to config Github

A Appendix :

Complementary information - additional resources

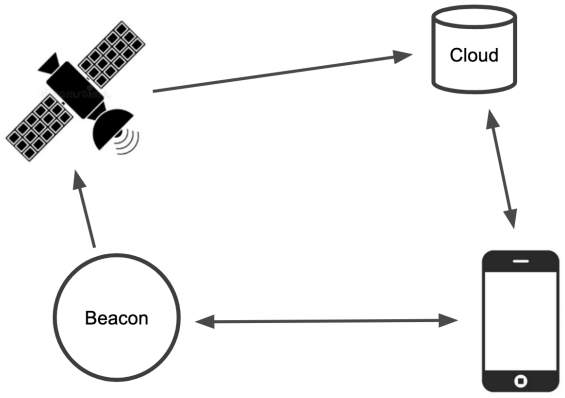

Figure 1: A high level model of our proposed solution.

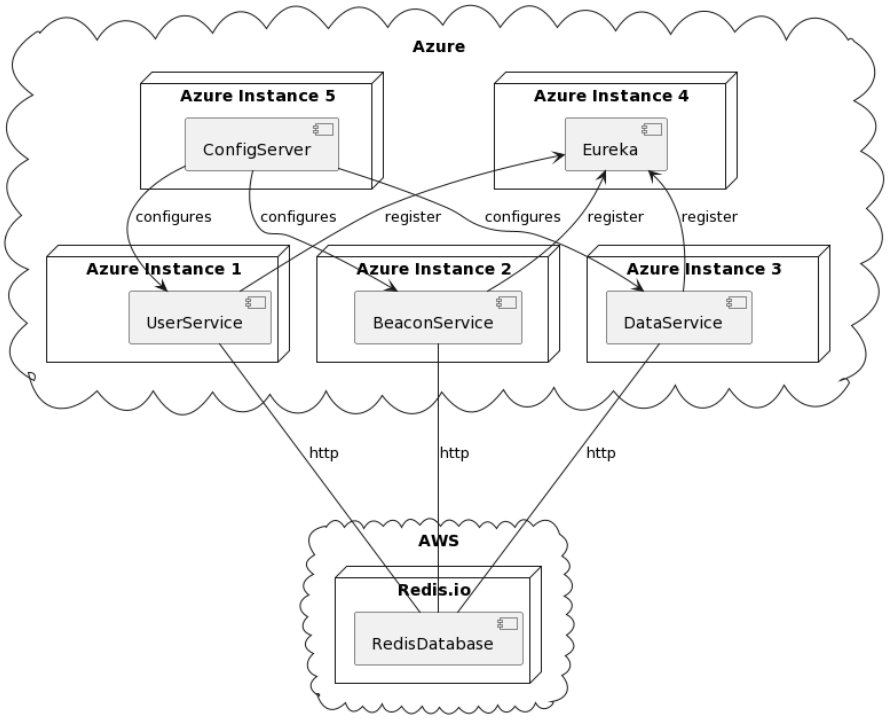

Figure 2: A UML Diagram describing the deployment of our mircoservice architecture.