SUP'COM

المدرسة العليا للمواصلات بتونس

ÉCOLE SUPÉRIEURE DES COMMUNICATIONS DE TUNIS

Telecommunications Engineering Degree

Major:

Multimodal Information Applications (AIM)

End of Studies Project

Retail Promotion Configuration Service

Implemented by:

Mohamed BELLAKHAL

Professional Advisors:	**Mr.Zied BELLIL, Eng**
	Mr.Ghaieth ZOUAGHI, Eng
Academic Advisor:	**Mr.Mohamed-Bécha KAÂNICHE, PhD, Eng.**

Work proposed and realized in collaboration with:

COGNIRA

Academic Year 2020-2021

Ecole Supérieure des Communications de Tunis

SUP'COM

2083 Cité Technologique des Communications - Elghazala - Ariana - Tunisie

Tél. +216 70 240 900 – site web : www.supcom.mincom.tn

I Authorize **Mohamed BELLAKHAL** to deposit his report and to assist for the end of studies project defense.

Professional supervisor, **Zied BELLIL**

Signature and stamp

I Authorize the **Mohamed BELLAKHAL** to deposit his report and to assist for the end of studies project defense.

Academic supervisor, **Mohamed Bécha KAÂNICHE**

Signature and stamp

ACKNOWLEDGMENTS

These enduring pages of work are the cumulative sequence of extensive guidance and hard work. Therefore, I would like to reserve this part of the report to express my gratitude to all of those without whom this work could not have been a reality.

I want to express my profound appreciation to Cognira, as I feel delighted and lucky to receive such help and guidance from my esteemed supervisors Mr.**Zied Bellil** and Mr.**Ghaieth Zouaghi**. I want to thank them for their valuable mentoring that led to a successful work. My two supervisors helped me unlock my potential and develop my skill-set with their priceless advice, for that I will always hold feelings of respect and regard.

I would also like to express my deepest gratitude to my university supervisor Dr.**Mohamed Bécha Kâaniche**, for his valuable orientation in the conception of the project, his constant guidance and his patience.

Moreover, my sincere thanks goes to Mr.**Elias Bouricha**, the country branch manager, for giving me this hospitable and professional environment to be able develop and work on new skills as well as expanding my knowledge horizons.

Finally, I want to thank all the interns and engineers within Cognira. I have had several special memories during these four months. I learned a lot from them and they helped me grow a lot technically and mentally. Everyone holds a special place in my heart that will never fade away. Thank you all.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

• **API**	=	**A**pplication **P**rogramming **I**nterface
• **CQL**	=	**C**assandra **Q**uery **L**anguage
• **HTTP**	=	**H**yper**T**ext **T**ransfer **P**rotocol
• **HTTPS**	=	**H**yper**T**ext **T**ransfer **P**rotocol **S**ecure
• **JSON**	=	**J**ava**S**ript **O**bject **N**otation
• **JWT**	=	**J**son **W**eb **T**oken
• **REST**	=	**R**epresentational**S**tate **T**ransfer
• **RPCS**	=	**R**etail **P**romotion **C**onfiguration **S**ervice
• **SOA**	=	**S**ervice **O**riented **A**rchitecture
• **UML**	=	**U**nified **M**odeling **L**anguage

- **WOA** = **W**eb **O**riented **A**rchitecture

GENERAL INTRODUCTION

In today's world, business is heavily reliant on data, especially for retailers. Retailers use data everyday to assess and improve their business decision-making process. One of the most used techniques that leverages data to improve the decision-making in retail is the trade promotion forecasting. This technique is the operation of attempting to discover the multiple correlations between the promotion characteristics and the past demands in order to provide accurate planning for the future promotion campaigns.

The exponential growth of the data in matters of availability and accessibility in the recent years has helped to drastically improve the promotion forecasting process. However, the classic architectural styles for retail software has always held back retailers from achieving accurate predictions of promotions using client data. When we talk about classic architectural styles for retail applications, the first architecture to be mentioned is the monolothic applications architecture. This architecture is consituted of a single application stack that contains several dependant modules within that one application. This tightly coupled architecture has proved to be unable to keep up with the fast paced increase in the amount of digital information. Since monolothic applications are less reusable, less scalable and consume more deployment and restart times. Other architectures were introduced to solve the drawbacks of monolothic applications, such as the SOA(Service oriented architecture). SOA puts emphasis on the scalability and reusability of retail applications as it allows different services to communicate through different languages and platforms. It is based on the principle of loose coupling and an open sharing architecture approach. SOA presumes sharing as much as possible between the services. This architecture offered the retailers more flexibility to run and develop their applications, optimized the deployment time and offered a more data friendly eco-system. Another breakthrough architecture was the WOA(Web oriented architecture). WOA leverages the advantages and patterns of

SOA but it also brings the SOA and the Web together. In simpler terms, the Web oriented architecture centralizes communications and deployments on the Web, not within the enterprise, which has proven successful especially in the retail business.
Still, with all its advantages, WOA was considered more of a temporary solution in retail. In fact, this architecture is multi-threaded, meaning that it favors concurrent and multiple thread of tasks execution, hence there are more overheads to manage the inputs and the outputs of each service. It also favors traditional relational databases. This approach cripples the scalability of the client data in the retail applications. Since the distribution of applications in SOA and WOA is more business oriented, these architectures favor common standards and governance. This detail hinders the creativity of the developers and that of the entreprises as these architectures brought more norms and protocols. In addition to that, do to the speedy nature of the retail industry, another drawback was the fact that continuous software delivery and DevOps pipelines are not yet as mainstream as we think they would be in a an WOA eco-system.

What if we could take all the advantages from WOA and minimize the drawbacks of this architecture in a retail context? One of the architectural styles that came to answer this question is the microservices architecture. This latter proved to be successful for data-centralized retail software. In fact, it divides the applications to a set of smaller services, but unlike the WOA, where the software is divided by entreprises and business logics. The scope of the microservices division is by application. This means that a single retail application is separated into autonomous components that are independent and loosely coupled called microservices. These microservices are single-threaded and bounded. they also don't follow common standards and protocols. On the contrary, they favor creativity and freedom of choice. The communication in this architecture is executed via a simple messaging system. Microservices guarantee the scalability of the application from data and performance perspectives, although, this architectural style has also brought its own drawbacks.
In this context, this project aims to solve one of the biggest drawbacks that came along the microservices architecture. This drawback is the constant and high need for configuration in a microservices based architecture. To carry out this project, we had to address several major difficulties when assessing the current architecture of the retail application for Cognira. These difficulties consist of handling the configuration data of the promotion management software for retailers. In order to achieve that purpose, we had to apply several optimization operations such as automating the configuration handling and keeping track on the version of the configuration data with the goal to deliver a more performant and simplified architecture.

The remainder of this report is structured into five chapters. The first chapter covers the scope of this project, such as the project description and main highlights, the owner organization of this project, and the aspects related to the work methodology during this internship.
Throughout the second chapter, we list a set of the functional and non-functional requirements for the project, along with the use case and sequence diagrams in order to describe exactly how our system works.
The third chapter contains the global design and architecture of the system that we built in this project. Then, the design is thoroughly explained in the fourth chapter which consists of the detailed design of the retail promotion configuration service.
In the fifth and final chapter, we walk through the features that we implemented and the technologies that we used in that context.
And finally, we wrap up this project report with a conclusion and then we open some perspectives about the future prospects of our work.

CHAPTER 1

SCOPE STATEMENT

Introduction

The scope statement outlines the entirety of the project, in this chapter, we define the business context and range for this project, the different responsibilities shared in the duration of this project, the set of deliverables and the features that we developed, as well as the stakeholder who will be affected by this project.
The scope statement also includes the major project objectives and goals. It will act as the primary tool for stakeholders and teammates to refer back to and use as a guideline to accurately measure the project success.

1.1 The project charter

The project charter[1] is the definition of the project scope. In this section, we introduce a general overview on the project context and objectives, the business value of the project, the participants in the project as well as a preliminary delineation of roles and responsibilities.
This section also provides a shared understanding of the project and can be considered as a contract between the parties involved in the project such as the project developers and the project owners.

1.1.1 Project context

This project was realized in the context of a graduation project at the Higher School of Communication of Tunis. This project is specified as one of the requirements for graduation and obtaining the engineering diploma. It specifies implementing the student in an internship within a professional environment from four to six months. This internship was held at Cognira from 15/02/2021 through 15/06/2020.

1.1.2 Project overview and business value

The goal of the retail promotion configuration service project is to build a microservice that automates the configuration of the promotion application suite. The former is the promotion management software offered by the software company Cognira to retailers, this software provides the Cognira clients with a better and more effective promotion planning for their set of products. The promotion planning is based on the clients behaviour, past performance analysis and accurate forecasting. The business value of the retail promotion configuration service can be regarded as the introduction of a next generation software solution of the promotion application suite. Moreover, This next generation software offers an optimized architecture, a faster development and maintenance process along with proposing a more simple approach regarding the methodology related to applying changes on the application's configuration.

1.1.3 Project roles and responsibilities

During this project, the intern was responsible for defining the scope of the project and it's concise goal, defining the technical and non-technical needs of the project, designing the solution and its specifications, building the solution and testing it against the project requirements as well as performing the necessary operational assessment and implementation activities.
In the course of these activities, the work of the intern was assisted and overseen by the school supervisor **Mr.Mohamed Bécha Kaâniche** who ensured the quality standards of the internship, along with Cognira's Software development manager **Mr.Ghaieth Zouaghi** and Cognira's Backend development manager **Mr.Zied Bellil** who were also responsible for setting the project milestones and deliverables.

1.2 The project owner and sponsors

1.2.1 Cognira presentation

Cognira was founded in 2015 by data science and retail experts. It is a self-funded company that has 3 offices located in: Atlanta (United States), Tunis (Tunisia), and Lille (France) and accounts for more than 70 employees. Cognira has been recognized by many international technology magazines and newspapers. Mentioning a few, Cognira placed 108 on INC.Magazine's 5000 List of the fastest growing companies in the USA[2] with a growth of 394 percent [3] and was recently ranked, by the Financial Times, as 9 among all technology companies in the Americas. Thanks to its strong knowledge in AI, advanced analytics and cloud-native technologies, Cognira is the leading artificial intelligence solutions provider for Enterprise retailers.

1.2.2 Cognira activities

Cognira provides 3 main services to the client: Return on Investment(ROI)-driven software implementations, Analytics consulting services, and Forecast as a service (Faas). The table 1.1 gives a brief explanation of these different services[4].

Services	**Details**
ROI-driven software implementations	Cognira is ensuring successful implementation through mainly three areas: Software expertise, Retail experience and Technology proficiency.
Analytics consulting services	Thanks to the retail knowledge and experience, Cognira applies a wide range of statistical and quantitative analysis in practical and valuable ways. In addition to that, its technical skills ensure navigating the largest and most complex data environments with the most relevant and up-to-date tools.
Forecast as a service	Retail forecasts can be challenging and time consuming to maintain, even with the most advanced systems. Cognira works with its clients' systems to diagnose and address their issues and keep their forecasts in top shape.

Table 1.1: Cognira services

1.3 The problem statement

One of the strong suits of the Cognira's software solution for managing promotions is that it offers the retailers a modular application based on microservices. A microservice can be defined as an architectural style that structures an application as a collection of services. These services are highly maintainable and testable, loosely coupled and independently deployable [5].

In retail, Cognira's solution thrived in the industry and was distinguished from its other competitors. In fact, platforms built around microservices put forward a better performance and flexibility than the traditional platforms built around one application or what we would define as a monolithic application, and platforms built and divided around business logic such as the WOA. Furthermore, microservices lower the barrier to innovation by removing any long-term commitment to a particular technology stack and instead they use a multitude of technologies in one application. These advantages make this application architecture more performant around different areas as each microservice is replaceable and scalable without affecting the day to day running of the business.

All of the advantages given by the microservices architecture to the retailers come at the price of high need for configuration. As a matter of fact, each time the client wants to affect a change on the configuration of the application, this change must pass by a multitude of microservices. This means that the developer responsible for managing the configuration will have to manually access the configuration data, import that data from the cloud infrastructure in which the data is stored, affect the necessary changes for each microservice then export the data again in the cloud. This process of editing configuration files is very time consuming and can cause conflicts in the communication between the microservices within the architecture.
So how can we offer the retailers a performant, scalable, modular and flexible application but also with a smooth configuration experience?
How can we remedy the constant need for configuration from a time and effort consuming task to a more efficient and optimized process?

1.4 The project goal and objectives

This project aims to answer these previous questions and in order to achieve that result, we opted to rely on one of the main advantages of the microservices architecture. This advantage is the fact that every component of the architecture can be independently scalable, deployable and aims to achieve a single purpose.
As a matter of fact, the goal of the retail promotion configuration service is to build a standalone microservice that answers all the needs for configuration for the retailer's promotion management software. This project will transform the configuration handling from a manual task distributed around the entirety of the application to an automated, centralized and easy to use component. All within the architecture of the application.
Another goal of this centralized configuration handling microservice is to put forward a huge improvement on the general architecture and purpose of the promotion management software offered by Cognira to the retail clients. This project optimizes the inter-communication between the microservices by playing the role of an intermediary agent or what we can also describe as a third-party. By a third-party we

mean a component which organizes the multiple requests launched by the different microservices within the promotion management application. By achieving that, we will be able to constantly get and update the configuration data within the cloud infrastructure responsible for storing this information. These requests held previously a direct nature between the microservices and the database. This added to the application complexity and created multiple performance failures.
In addition to that,The retail promotion configuration service offers traceability of the configuration by adding versioning and storage features of the data affected by each configuration process.

1.5 The project's major deliverables

After introducing the project requirements, we devoted this paragraph to mention the project deliverables. Therefore, we describe the quantifiable goods and services that must be provided upon the completion of the project.

- **Requirements specification (document)**
  This document outlines a shared understanding of the project goal and objectives as well as the roles and responsibilities for all the parties involved during the internship period.
- **Source code (document)**
  By the end of this internship, the source code of the promotion configuration service is to be delivered. This deliverable includes the source code for the application that is developed, the implementation of the testing scenarios along with the necessary files developed for the containerization and deployment for the application.
- **Source code documentation**
  The source code documentation deliverable is a fundamental engineering practice critical to efficient software development. In fact, the documentation will act as a specification of behavior for other engineers in order to fully understand the code developed behind the retail promotion configuration service. In addition to that, The documentation will help in the reusability of the solution since it will have clear and concise instructions on how to integrate the application in a new environment, package it and deploy it in a new cloud infrastructure. The documentation also contains the full explanation of the testing scenarios and will provide an understanding of the code developed for testing the application.

- **Progress reports (presentation)**

  A progress report in the form of a presentation is due every 2 weeks during the internship period which provides an overview to the Cognira's development team of the project status, the milestones achieved and the issues encountered during the progress of the project.

1.6 The project non-goals

Above and beyond improving transparency and managing the expectations for the retail promotion configuration service, setting forth the non-goals from the very beginning serves us a real sense of clarity for the work to be done.

- **User-interface**

  The configuration handling process developed in the retail promotion configuration service will not be managed and triggered through a user-interface. Instead, it will be managed through RESTful APIs requests, noting that a RESTFUL API is an architectural style for an application program interface (API) that uses HTTP requests to access and use data. These requests can be used for reading, updating, creating and deleting operations concerning the configuration resources.

- **Logging dashboard**

  Throughout the project duration, a logging feature will be implemented for the edits applied on the configuration files within the promotion management application. The logging helps to analyze and document the configuration record, with all mentioned, creating a dashboard to visualize these changes is not included in the scope of this internship.

1.7 Project management plan

After describing the scope of our project such as its goals, sponsors, requirements and deliverables we now dive into detail about the project management plan for the retail promotion configuration service. In this project, we needed to implement a testable, stable and scalable application. These characteristics need to be present at all times during the development life cycle. In addition to that, the project needs to accommodate frequent updates, especially when one or more requirements change and adapt to them. Having all of these requirements in mind, we opted for the agile methodology SCRUM as our project management methodology

1.7.1 Definition of the scrum methodology

Scrum is an agile method for developing and delivering sustainable products[6]. It relies on an incremental approach that focuses on adaptation, communication between members of the team and

constant feedback from the customer. This methodology found its way to software engineering as it allows the development of highly adaptable software. The software developed in SCRUM methodology can easily and reliably change according to emerging realities or changes in the customer's needs.
This methodology is being used at Cognira, as it allows the development teams to work as a unit towards achieving their common goal. Scrum encourages both face to face communication between team members, as well as trusting each member to be capable and independent. As such, teams are able to identify quickly and effectively, tasks to be completed and assign them to each member. The scrum methodology then defines sprints, to organize those tasks to be carried out over a specified period of time.
Each sprint has its own specifications, design and test phases. Sprints are usually 2 weeks to 1 month long, with a release planned at the end of each sprint.

1.7.2 Core roles

In a team adopting the scrum methodology, there are 3 major roles to be assumed by the members, and these are:

- **Product Owner**
  This role carries out the will of the customer. Product owners are responsible for ensuring that the scrum team contributes added value to the business, and that they comply with the customer's requirements.
- **Scrum Master**
  The scrum master is accountable for removing blockers encountered by the team that slow their ability to deliver towards the project goals. They are not just a traditional team leader, but they make sure the scrum process is being used at all times.
- **Development Team**
  The development team members are responsible for delivering shippable product increments at the conclusion of each sprint. The team is usually made up of 3 to 9 people with a varied skill set to be able to analyze, design, develop, test and document their work. The traditional role of project manager is absent in a scrum team. This is due to the fact that the developing team is self-organized, and the rest of the duties of a project manager are carried out, mostly, by the development team and the product owner. The figure 2.1 showcases the Scrum methodology cycle.

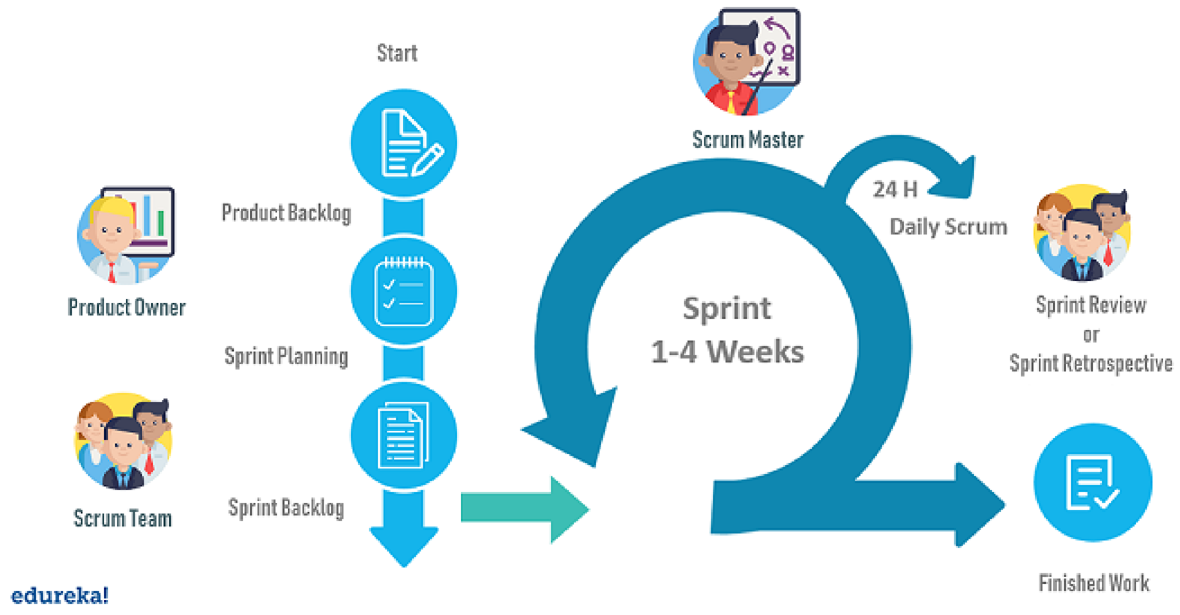


Figure 1.1: Scrum methodology cycle [6]

1.7.3 Scrum meetings

In the scrum methodology we can find 3 types of meetings:

1.7.3.1 Sprint planning meeting

At the beginning of each sprint cycle, which meets the end of the previous one, a meeting is held in order to:

- Check what work has already been done.
- Select the next product increment to implement.
- Determine the requirements.
- Detail the next sprint's tasks.
- Prepare the sprint backlog that specifies all the above, and the time it takes the work to finish.
- Determine how much of that work will be done during the sprint.

1.7.3.2 Daily scrum meeting

This meeting is held daily between the project team members during a sprint. Each meeting must follow these guidelines:

- The meeting is to be held at the same location and time of day every time.
- The meeting duration is set to 30 minutes.
- Determine the requirements.
- Each team member must have prepared the update to be shared with the others. During the meeting, each member must discuss the following points:

  - What they did yesterday.
  - What they will do today.
  - The blockers they encountered (if there's any). The blockers encountered by the team members mentioned during the meeting is documented by the scrum master. Outside the meeting, the team works towards resolving them.

1.7.3.3 End meeting

At the conclusion of a sprint cycle, a meeting is held in order to:

- Review both the work completed and not completed during this sprint.
- Present the achieved work.
- Determine the requirements.
- Detail the next sprint's tasks.
- Reflect on the past sprint and assess what went right and what went wrong during the sprint.

1.7.4 Code review

In order to reinforce the benefits of the agile method, one of the methods that we used in this project to manage the communication and ensure the product quality is the code review. In this process we allow code sharing between the team members and we emphasized the autonomy of the team members.

In a code review oriented environment, after the development of each functionality within the project, the developer shares the changes affected on the code with his colleagues. This allows them to analyze the new part developed by ensuring the quality of a diverse range of aspects within the provided solution, among which we quote:

- The code developed must be correct from a syntax point of view and completely shorn of logical errors.
- The added code must meet the requirements and the good coding practices defined by the team. As these requirements facilitate the readability and reusability of the corresponding code by another developer within the company.
- The code must have good organization, consistency and ergonomics.
- The tests of the developed code must be already implemented and functional after the modification of the code by the developer.
- This process helps the development team to avoid code regressions, wasted time for the understanding of a developed code and of course to ensure that the code and its changes go well with team goals generally and sprint goals precisely.

Figure 1.2 illustrates the code review process.

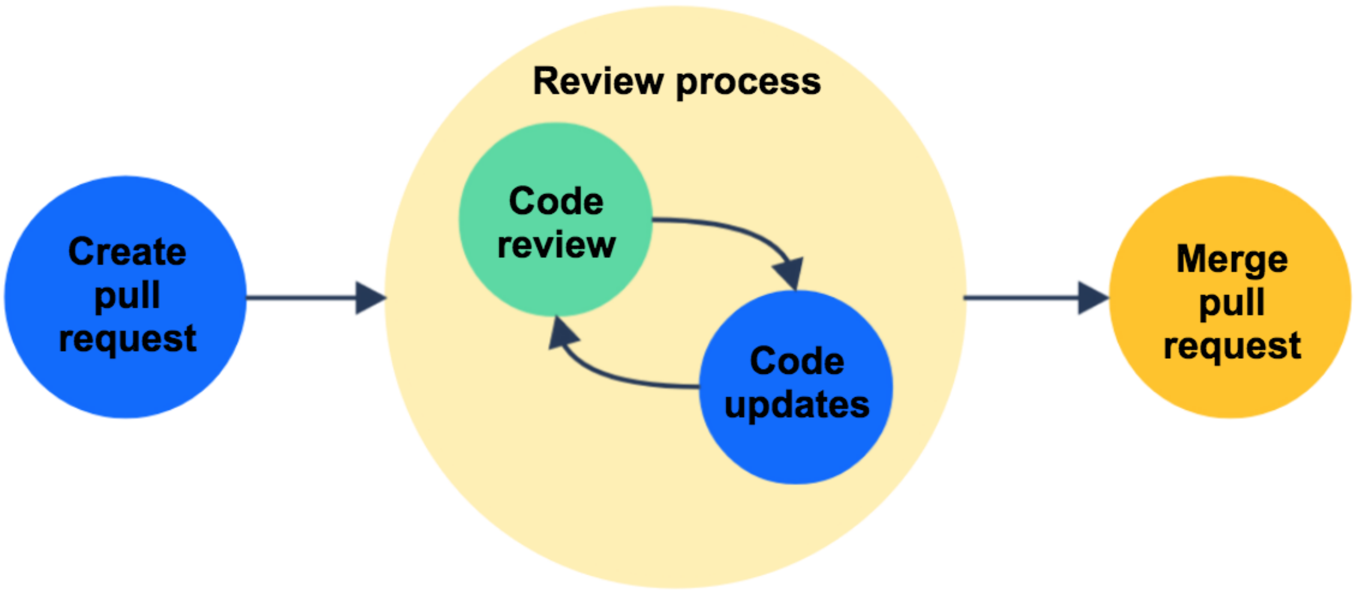


Figure 1.2: Code review process [7]

1.8 Project plan

In this section, we break down our project into a set of milestones that need to be achieved according to the Scrum method. In order to do that, the first step is to define the "Product Backlog", then in a further step we identify the project tasks based on the product backlog that we implemented.

1.8.1 Product backlog

In this step, we define the expected features in our project. More precisely, the product backlog covers all of the functional characteristics that constitute the desired product. These functional characteristics can be labeled as the User stories. Table 1.3 summarizes our product backlog with the estimated number of days for each feature to be developed.

Features	**Estimate(d)**	**User Stories**	**Priority**
The capability to automatically edit configuration files. (add,update or delete configuration data)	20	As a user of the retail promotion configuration service (RPCS), I can edit the data within a configuration file with a simple HTTP request.	High

The ability to handle the exceptions raised in the application.	1	As a member of the quality assurance team, I can identify the exception that caused the RPCS to fail when editing a configuration file.	Medium
The ability to handle the exceptions raised in the application.	2	As a user of the RPCS, I expect to get a clear error message if my operation to edit configuration data is not successful. (validate or reject a request with a human-friendly response)	High
The capability to handle the requests for configuration data within the promotion management application.	10	As an enterprise using the promotion management software of Cognira, the configuration data will be automatically uploaded into my platform and every update applied on the configuration data should be synchronized and fed to my application.	High
Authorization check.	1	As a user or an enterprise, my requests to edit or retrieve the configuration data from the RPCS will be processed by Cognira's security microservice.	High
The capability to automatically add version to the configuration files and store them into cloud storage.	20	As a user of the RPCS, I can keep track of the changes applied on the configuration data and I can retrieve versioned configuration files from a cloud storage.	High

The capability to deploy the RPCS.	5	As a cloud or software engineer working in Cognira, I expect to find the RPCS as a packaged and re-deployable microservice that can be reusable and integrable in any microservices architecture.	High

Table 1.2: Product backlog

1.8.2 Project flow

Before starting to carry out the tasks to build the promotion application microservice, the Cognira software development team set up an integration phase for the interns. The integration consisted of a 3 weeks software development competition. The goal of this competition was to get comfortable with the technology stack and the retail business aspects. Following this was a 2 weeks of deployment and integration tasks for the microservices within the promotion management software solution offered by the company.This step aims to get used to the promotion management software' architecture and the cloud operations related to it and also to gain a better understanding of the Cognira product.

This phase was beneficial and essential to allow us to get to know the team members and gain a better understanding of Cognira's services, business aspects along with the base code and best practices.

After this integration phase, the work is planned according to the scrum methodolgy in order to achieve the features that we set out in the "Product backlog" paragraph. The work was broken down into three main phases:

- Phase 1: the planning phase, the preparation of the technical documents and the conception of the project.
- Phase 2: the development phase during which we develop the project features, this is the phase where we launch our sprint.
- Phase 3: quality assurance phase which is used to correct the bugs identified as a result of the different testing scenarios for the developed solution.

Conclusion

In this first chapter, we provided a detailed description of the project and the work that must be done in order to carry it out in the best possible way. In the next chapter, we move on to analyzing and detailing the set of requirements in order to achieve this solution.

CHAPTER 2

REQUIREMENT ANALYSIS AND SPECIFICATIONS

Introduction

The specification phase is a crucial part in a software project. Actually, specification aims to determine the quality of the final product. By analyzing those requirements, we make sure that we're able to reach our final goals. In the specification phase, we set a crystal clear understanding of how we want the final product to be. This is done by specifying the various requirements and analyzing the functionalities. In this chapter, we go through both the functional and non-functional requirements of our application. Then, we will present the use cases and specify the actors to analyze those requirements.
Finally, we demonstrate how these actors communicate together and how our system works from a general point of view using an analysis sequence diagram.

2.1 Requirements analysis

In this section, we analyze the expected functionalities of our system. This step is necessary before starting the development process.

2.1.1 Functional requirements

The functional requirements offer us an insight in detail about the set of features the application will be able to provide. Our project, the promotion configuration service is expected to offer these specific functionalities:

- **Configuration automation**
  Develop a feature within the microservice which will be responsible for automating the process of applying modifications on the rows of the configuration files. These operations will consist of adding, editing and deleting rows. This functionality allows the Cognira development team and the retail clients to apply changes on these files within the promotion application architecture.
- **Configuration versioning**
  Each time a modification is to be applied on the configuration files, The microservice will be responsible for mapping the edited file to a unique version. Later on, this file is pushed to a cloud infrastructure. This functionality provide the ability for the traceability of the configuration files. Furthermore, it makes the configuration files reusable for other use-cases and adds a reverting feature to the configuration of the promotion application's architecture.
- **Configuration handling**
  The microservice is required to have a feature responsible for handling requests for configuration from the other microservices. as the developed microservice will constantly feed them with the necessary configuration information.
- **Configuration logging**
  The microservice is responsible for creating logs for all of the changes applied on the configuration files. The logs can be later on consumed as a dashboard to visualize the different changes applied on the promotion application architecture.
- **Testing**
  Create testing scenarios and implement unit tests for each of the developed features to ensure the quality of the solution provided.
- **Containerization and deployment**
  During the period of the development of the promotion configuration service, the microservice needs to be continuously containerized, which means that the code and the dependencies of the application developed need to be packaged into a standard unit of software called a container[URL6]. The containerization offers the ability to run our microservice quickly and reliably from one computing environment to another.

This container is to be deployed in the cloud infrastructure provided by Cognira within the promotion management software architecture.

2.1.2 Non-functional requirements

The non-functional requirements present the features and criteria to be used to judge the operating routine of the entire system. These requirements are essential to make sure the project behaves as initially planned. When implementing our solution, we had to bear these following attributes:

- **Security**

  Having an application that is going to serve a large amount of requests and configuration's data sharing, a secured system is a vital part. We need to make sure that our resources are directed to serving each user with their corresponding retailer data and configuration all the time. To carry that out, we have chosen JWT (JSON Web Token) authentication which encodes then stores the user data in a token. The token is sent with each request the user makes and it can be decoded on the server side[8]. This ensures each user will only be able to access the resources they're allowed to.

- **High data fidelity**

  Our application handles configuration data on which the promotion management application is reliant for each retail client. As a result, we need to ensure that all the data provided does mention the retailer in question and also includes the correct and full configuration.

- **Performance**

  The various components of the system need to maintain an acceptable level of performance. This means that every component will need to be optimized especially when we take into account the various data processing operations and the various requests that we will be conducting all the time.

- **Reusability**

  The code for the entire application will need to satisfy certain requirements such as readability, respecting indentations, documentation and modularity.

- **Ergonomics**

  The system must provide a uniform design across the application through implementing user-friendly and easy-to-understand endpoints that allow the user to manage the configuration files.

2.2 Requirement specifications

After presenting the various requirements of the application. In this section, we give a detailed study of those requirements.

2.2.1 Modeling language

For object modeling in our project, we used the UML "Unified Modeling Language", UML is intended for the architecture design and implementation of software systems[9]. It consists of different types of diagrams that describe the boundary, structure and behavior of the system and the objects that constitute the system.

2.2.2 Actor identification

An actor can be an external element or a human that interacts with the system[9]. An actor is supposed to interact with the system use-cases, make decisions and take initiatives. We consider as actors in our system:

- **The end-user**
  Our users can be either a Cognira developer or a configuration handling personnel in one of cognira's retail clients. These users can interact with the system in order to edit the configuration files in the promotion management application, the interactions are made by the intermediary of our developed solution, the promotion configuration service (PACS).
- **The cloud platform**
  The PACS is responsible for processing the edited files by the actor, adding a version to these files and then storing them in the cloud platform used by Cognira.
- **The promotion management application**
  The microservices within the architecture of the promotion management application are going to constantly request the configuration data from the PACS. The requests are handled by our microservice, this latter sends a response to the promotion management application for each request to be made.

2.2.3 Use case diagram

In UML, use case diagrams describe the general functions and behaviour of a certain system[9]. These diagrams also identify the interactions between the system and its actors (what the system does and how the actors use it). Each use case describes a particular goal for the actor and explains how this latter interacts with the system to achieve this goal. The diagram in Figure 2.1 describes the different use cases of our system.

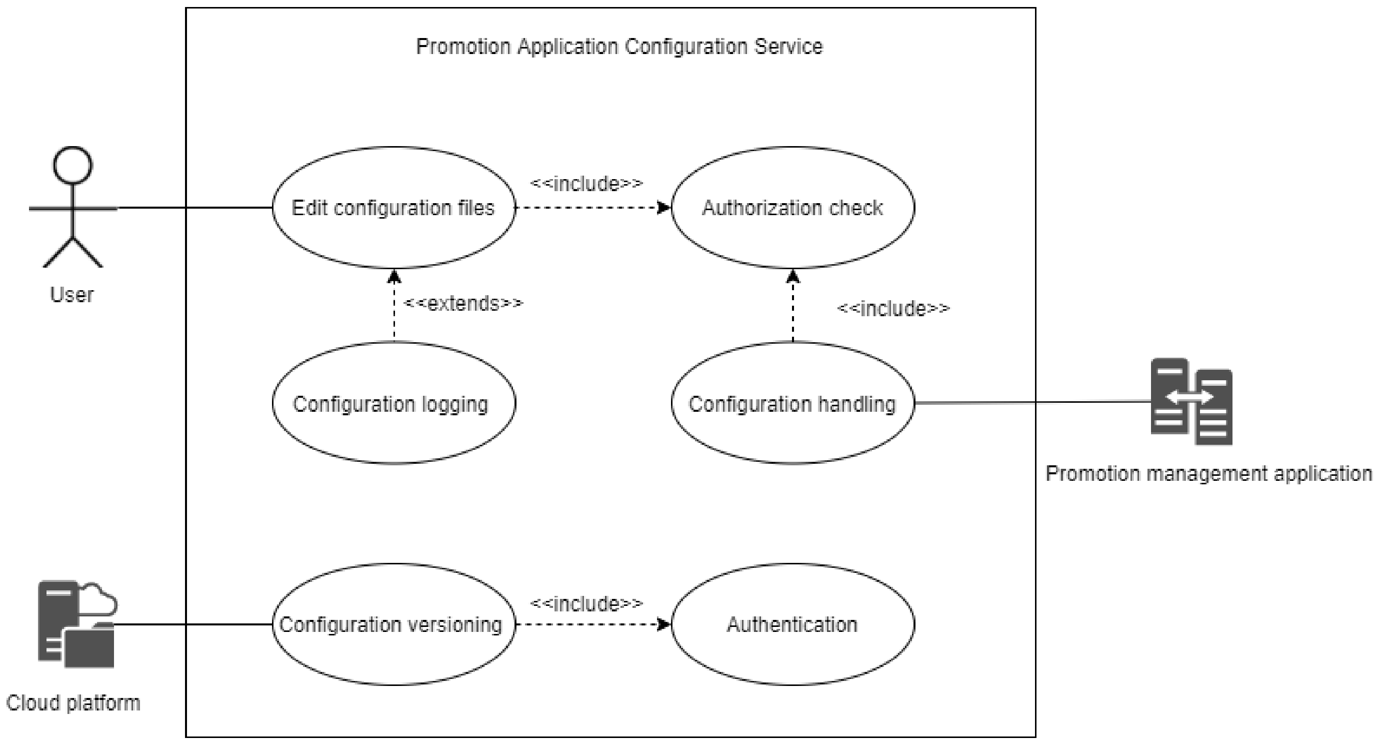


Figure 2.1: Use case diagram

2.2.4 Use case textual description

The textual description of use cases is a description of the chronology of actions within our system. It provides a clarification in regard to the course of the functionality. In the following tables, we present the textual descriptions of the use cases presented in the use case diagram, the textual descriptions serve the purpose to have a global vision of the functional behavior of our system.

2.2.4.1 Use case: edit configuration files

Table 2.1 resumes the functionality of editing a configuration file in the promotion configuration service.

Use case description: Edit configuration files
Identification Title: Edit configuration files Goal: Apply modifications on the configuration files. Principal Actor: The end-user. Summary: The end-user can request to either add, update or delete configuration data from the database of the promotion configuration software through our system.

Sequencing The use case starts when the user sends a request to edit the configuration data
Pre-conditions The user is authorized to send requests to the system.
Nominal scenario 1. The user chooses the type and the content of the request to send to the system. 2. The user sends the request to the system. 3. The system accepts the request. 4. The system connects to the database of the promotion management software. 5. The system applies change on the database. 6. The system sends the result of the operation as a response to the user.
Exception scenarios **E1: invalid response format** 3. The system throws a request format exception. 4. The system responds with the correct request format template. **E2: invalid data type in database fields** 5. The database driver throws a data type exception 6. The system returns a response specifying the wrong data field.
Postconditions The configuration files are updated within the database of the promotion management software.

Table 2.1: Use case : edit configuration files

2.2.4.2 Use case: configuration versioning

Table 2.2 resumes the functionality of applying versioning on configuration files in the PACS.

Use case description: Configuration versioning
Identification Title: configuration versioning Goal: Add a version on the modified configuration files andv store them in the cloud. Principal Actor: The cloud platform.. Summary: Based on a scheduled timeline, the system will add a version to the configuration files then push them into the cloud platform used by Cognira(Azure cloud) in order to keep track of the modifications on these files.
Sequencing The use case starts when the timer for this task is triggered.
Pre-conditions The system is authenticated to the cloud platform.
Nominal scenario 1. A timer is launched for this scheduled use-case. 2. The system connects to the database of the promotion management software. 3. The system connects to the cloud platform 4. The system gets all the configuration files from the database. 5. The system adds a version to each configuration file. 6. The system pushes the versioned configuration files into the cloud storage.
Exception scenarios **E1: a configuration file is empty** 5. The system throws an exception for a missing configuration file. 6. The system logs the exception in a file 7. The system pushes the file to the cloud infrastructure.

Postconditions
The configuration files are versioned within the system. The versioned files are pushed into the cloud platform

Table 2.2: Use case : configuration versioning

2.2.4.3 Use case: configuration handling

Table 2.3 resumes the functionality of handling requests from the microservices within the promotion management software for configuration data.

Use case description: Configuration handling
Identification Title: configuration handling. Goal: Handle requests for configuration data from the promotion management application. Principal Actor: The promotion management application. Summary: The system is responsible for handling the promotion management application's requests for configuration data and feeding the microservices within this application with the necessary data as a response.
Sequencing The use case starts when the back end microservices in the promotion management application request configuration data
Pre-conditions The microservices within the promotion management application are authorized to make the request for the configuration.
Nominal scenario 1. A microservice launches a request to the system in order to get the configuration data.

2. The system connects to the database of the promotion management software. 3. The system gets the requested configuration data from the database 4. The system feeds the configuration to the microservice and thereby to the application.
Exception scenarios **E1: Wrong user/retailer id** 5. The system throws an exception for a mismatching id. 6. The system sends a response to the microservice to verify the id
Postconditions The microservices within the application are fed with the configuration data.

Table 2.3: Use case : configuration handling

2.2.5 Analysis sequence diagram

The use case diagram describes the major functions of a system from the point of view of the actors. But it does not reveal the dialog between actors and use cases. For that, we use the analysis sequence diagram. An analysis sequence diagram in software engineering is a sequence diagram that shows the set of events created by external actors.[10] In addition to that, this diagram respects the chronological order of these events. An analysis sequence diagram also illustrates the possible inter-system events, for a specific scenario of a use case.

In the sequence diagram below, we start with detailing the interactions of the user with our system during editing the configuration data. Moreover, we illustrate the set of events needed in order to version and export the configuration files to the cloud infrastructure. Finally, we demonstrate the main events that occur when a microservice requests configuration data from our system, the promotion configuration service.

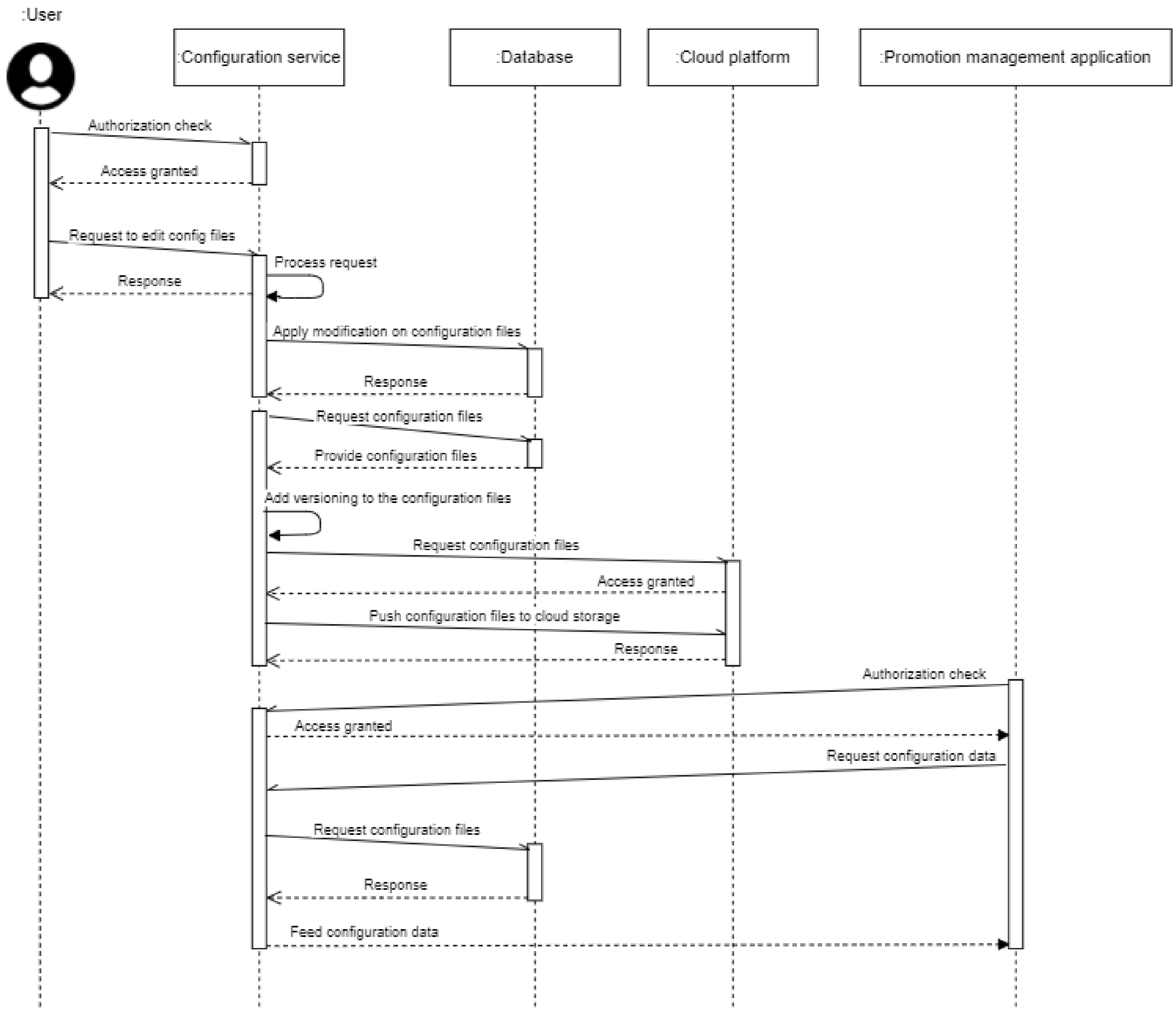

:User
:Configuration service
:Database
:Cloud platform
:Promotion management application
Authorization check
Access granted
Request to edit config files
Process request
Response
Apply modification on configuration files
Response
Request configuration files
Provide configuration files
Add versioning to the configuration files
Request configuration files
Access granted
Push configuration files to cloud storage
Response
Authorization check
Access granted
Request configuration data
Request configuration files
Response
Feed configuration data

Figure 2.2: Analysis sequence diagram

Conclusion

In this chapter, we highlighted the requirements of our system in a first part then we examined in depth these requirements through UML diagrams.

CHAPTER 3

GLOBAL DESIGN OF THE CONFIGURATION SERVICE

Introduction

To make sure that we'll be able to reach the project goals that we have set in the previous chapter, we need to satisfy certain requirements and features. They summarize our application's functionalities.
In this chapter, we detail the global design of the promotion configuration service. In order to achieve that, we present the solution architecture from a broad point of view. After that, we proceed with the physical and logical architectures. Finally, we conclude with the deployment architecture.

3.1 System design

The promotion configuration service is segmented into 3 main components. The first part of the application consists of a back end server. This server is responsible for automating the process of editing configuration data by the users via a set of REST APIs. noting that a REST API(REST stands for Representational state transfer and API for Application programming interface) is a collection of rules and definitions for creating and integrating software applications[11]. In simpler terms, APIs will enable our microservice to interact with the users without the need to understand how it works. This type of interaction will be through a simple type of request called HTTP request. In an HTTP request, the user should only specify the configuration file that he wants to edit, the content of the edited configuration data and the type of edit. this latter can range from an insertion of configuration data(Translates to a POST request in HTTP requests), an update in configuration data(Translates to a PUT request in HTTP requests) or a deletion of configuration data(Translates to a DELETE request in HTTP requests).

One of the advantages of using REST APIs for our application is that it also helped us to fulfill the second major component of our application. In fact, we also used REST APIs when implementing the configuration requests handler from the promotion management application used by Cognira. Thus, we avoided elevating the complexity of our application. In fact, these requests can also be translated to HTTP requests. Through these requests, the microservices within the promotion management application can acquire all the data they need with a simple GET request.
These two components proved to be crucial to our project. Actually, they enable us to simplify the complexity of the architecture of the promotion management application. Since they optimize the performance of the application by making it drastically faster to the promotion management application to acquire the configuration data as well as encapsulating the process of editing the configuration data for our user in a simple automated request.

The third component of the PACS will is responsible for the versioning of the configuration files, this process is periodically scheduled and fully automated, in a such way that in every period(The period will be daily in our case). Our component acquires all of the configuration files from the database, adds a version on each file and then exports these versioned configuration files into a data lake storage. A data lake storage in our use-case can be defined as a cloud pool that holds a huge amount of data in an optimized binary format until this data is needed[12]. The data lake service can be provided from a multitude of cloud providers, such as Amazon, Microsoft and Google, we decided to go with Microsoft's Azure data lake storage which is the cloud provider for our host organization Cognira.
This step will help to track the modification on the configuration files and revert back the changes if a previous version of the configuration files is needed. It will also help us to create a logging history for our data.
The figure 3.1 illustrated the design of our application.

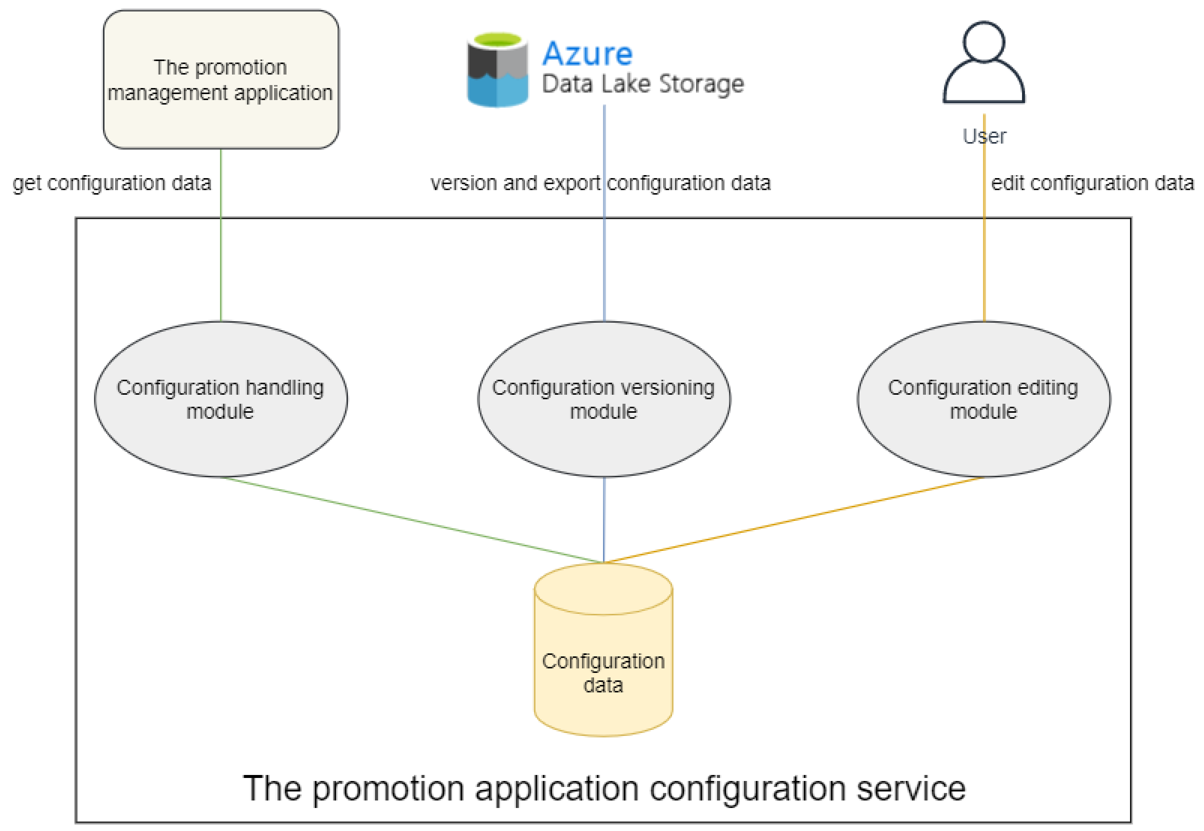

The promotion management application
Azure Data Lake Storage
User
get configuration data
version and export configuration data
edit configuration data
Configuration handling module
Configuration versioning module
Configuration editing module
Configuration data
The promotion application configuration service

Figure 3.1: Design of the promotion configuration service

3.2 Physical architecture

3.2.1 Microservices architecture

The physical structure of which the promotion configuration service is built around is a microservices architecture. To understand the physical architecture of our system, we first need to dive into the key notions of a microservices-oriented architecture.

- **Containerization**

  When using a microservices architecture, we expect several advantages[13]. These advantages are the ability to share the hardware resources between the components of our system, the flexibility to divide these resources to different components and suppliers and most of all the scalability of this architecture. In fact, a microservice takes the minimum amount of physical resources to function correctly and independently.

  This is why containerization can be considered as the fundamental block of this architecture. Actually, a container is a standard unit of software that packages up the code and all its dependencies[14] so the microservice runs quickly and reliably from one computing environment to another. This allows the developer to be certain that, with the use of a container, the developed application will be operating on any other machine regardless of any custom machine settings that may vary from the machine that is used to write and test the code.

In simpler terms, containerization includes combining an application with all of its related libraries and dependencies. Doing so, the application is able to execute efficiently in a variety of computer environments. Inside of the container, the microservice for the promotion application configuration and its dependencies sit on top of a container runtime environment. This latter can be defined as a software which runs the containers. The container runtime can work on the host operating system infrastructure or hardware equipment of choice.

In the figure 3.2 we illustrate the different components of a container.

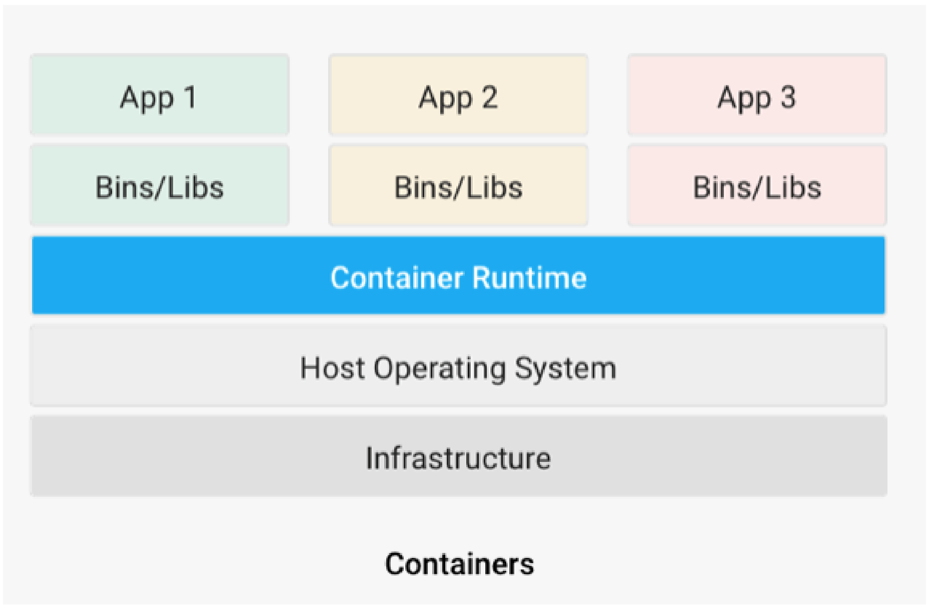


Figure 3.2: The different container layers [15]

- **Container runtime: Docker Engine**

  To build our containers, we used Docker and its tool product docker engine as a container runtime. This tool is designed to facilitate the creation, deployment, and running of containerized applications[16]. Docker shares a lot of similarities to a virtual machine. But unlike the latter, it allows the microservices to use the same operating system kernel (Linux in the case of Docker) as the system rather than constructing a whole virtual operating system.

  Using the Docker Engine to create a container gives us a lightweight, standalone, executable software package that contains all the necessary software for running our microservice. The following figure (3.3) presents the different layers of a docker container.

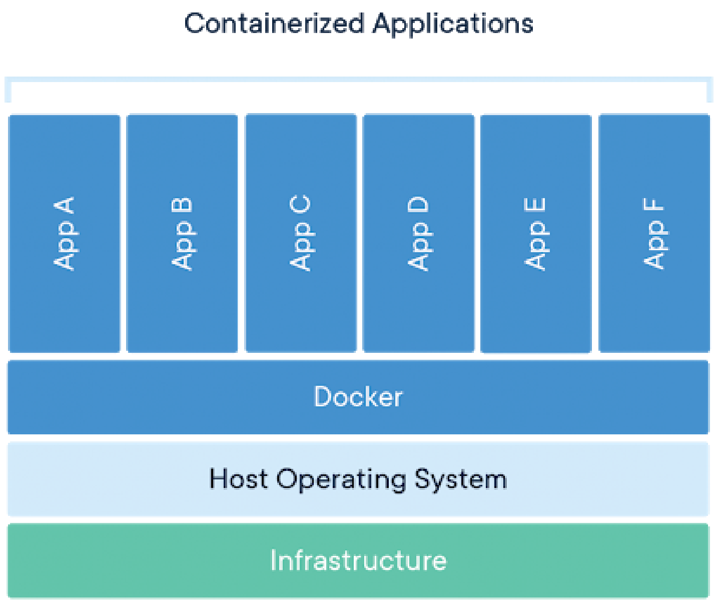


Figure 3.3: The different docker layers [17]

3.2.2 The promotion configuration service's physical architecture

The physical architecture of our project can be considered as a four-tier architecture[18], it is in fact formed by the following elements. The first component is the presentation layer. This layer is the device from which our user sends requests to the application. Moving forward, the second layer is the application server containerized as a microservice. This server handles the process of communicating with the other server that houses the data(the database server) which is our third component. The database server is also wrapped into a container. The application server runs all the operations required on the data. Besides, it will subsequently be in charge of serving the configuration data to the presentation layer.
Finally, we have our data layer in the form of an Azure cloud data lake storage. The data layer is responsible for storing multiple versions of the configuration data.

These independent yet interconnected components that we've presented, form the four-tier architecture. We opted for this physical structure because it allows us to achieve good performance on the two main aspects of our project. In fact, the user only has to wait one instance where all the data is loaded. Then the application server takes care of refreshing its output in a timely manner. Following this architecture guarantees that the user is having a fluid and smooth user experience. The second advantage is that the fourth data layer which we added comparing to the classic three-layer applications will allow us to keep track on the huge amount of versioned configuration data. Not only

this but the data layer also stores the versioned data in a different component rather than our database server. This feature helped us avoid overburdening the database server with data. In fact, when adding the data layer, we opted to avoid storing irrelevant data to the user in the database. This led to a drastically better response time for our application.
Figure 3.4 describes the physical architecture of the promotion configuration service.

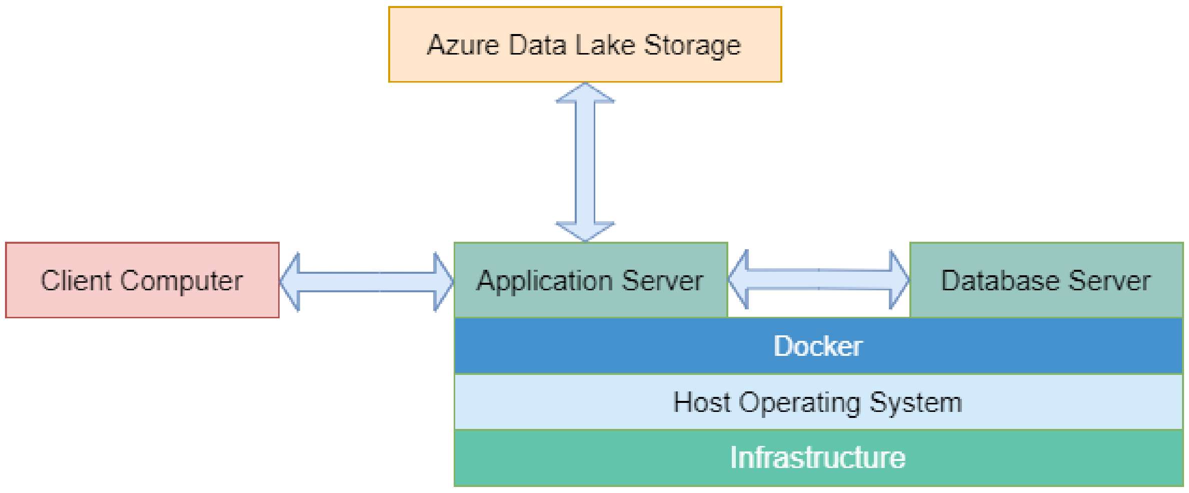


Figure 3.4: The physical architecture of the PACS

3.3 Logical architecture

As we previously stated, the primary advantage of our microservice is handling and serving the configuration data to our users and to the other microservices in the promotion management application provided by Cognira. To satisfy this need, in this section we explain the logical architecture that we used to satisfy these requirements. Starting by explaining the service-oriented architectural design that we adopted in this project.

3.3.1 Service oriented architecture

Service-oriented architecture[19] is a software design paradigm. In this architecture, the application components deliver services to other components over a network communication protocol. The key principles of this latter are the independence from suppliers, hardware, and technologies. A service is a discrete unit of functionality that can be accessed and acted upon from a remote location. In fact, we can describe the service oriented architecture as the backbone of the microservices architecture that we are working with in this project.

3.3.2 Web Services

As we mentioned in the previous paragraph, to achieve the inter-communication between the components of our application as well as the communication with the external actors, we need to follow

a network communication protocol. The web services do indeed achieve this purpose acting as a unified logical framework and a service-based one. With web services, applications can exchange data in a universal way, web services help us to avoid the dwelling on the differences of technologies used within each application.
The main goal of web services is to achieve a maximum level of compatibility between the different applications. This allows us to accomplish the requirements that we stated in the global system design. As a matter of fact, web services make us able to communicate with the set of APIs that we developed in our application through simple HTTP requests.

3.3.3 The promotion configuration service logical architecture

Following the definition of numerous concepts in the preceding sections, our solution drew inspiration based on those designs and architectures. These concepts assisted us in creating a precise design for our application. The logical architecture for the promotion configuration service can be concluded in these five main components:

- **Database**

  The database in our system represents the data model layer as it stores and organizes the large amount of configuration data in our application. The data is stored in a structured and distributed way so it can be easily queried afterwards.

- **Back end**

  This component consists of an independently deployable web service. This latter is able to communicate and exchange data with our users. Furthermore, it offers the capability to feed the other microservice in the application with configuration data via REST API calls. Moreover, this web service is able to query the database in order to modify or fetch the data that needs to be served. Some operations can also be conducted to apply transformations and format the data before making the exchange between the database and the actors of our system.

- **Schema manager**

  In order to handle and create the data model for our database, we created the schema manager service. The schema manager is responsible for creating the database and the schema of the tables within the database. This service also helps to update the schemas in the database when we want to update a configuration table or add a new one.

- **Data lake**

  One of the most important requirements of our project is exporting the versioned configuration data periodically into a cloud platform. This platform is required to be separated from the database. To achieve that, we use a data lake in our system as it gives us multiple advantages over other cloud services designed for this matter such as data warehousing. In fact, a data lake not only offers us the ability to store our structured configuration files, the ability to conserve their original structure, an unlimited scalability and the capacity to perform data analysis on our configuration data. But the data lake service also comes with a very low cost compared to a data warehouse.

- **Spark CronJob**

  In order to schedule a task on a Linux-based deployment environment, we use a software utility called cron. The purpose of cron is to run a series of commands, or scripts on a predetermined schedule [20].

  Cron has proved to be very useful in our project as the process of exporting and versioning the configuration data from the promotion configuration service to the data lake is a periodic and recurrent process. This process needs to be repeated in a constant time cycle. To achieve this purpose, we used a variation of cron called "crontab". Crontab runs as a background task in our deployment environment and provides us with the capability to run our scheduler.

  The crontab starts when the system boots, adds a version to the data and exports it to the data lake. After that, based on its predetermined schedule, our crontab can be configured to run as often as every minute or as infrequently as once a year. In our case, we are going to update the configuration data in the data lake once every twenty-four hours.

  In order to handle the large amount of configuration data to be fetched from the database and exported each time the CronJob gets triggered, we opted for Spark. As described in the Spark documentation[21], Spark is a Lightning-fast big data engine that offers multiple advantages for large-scale data processing such as its compatibility with a variety of database structures and deployment environments.

  Spark became a leader in speed-oriented big data applications thanks to its architecture that optimizes the scheduling of several workloads. It also offers optimized data queries as it uses a physical execution engine allowing it to use in parallel the maximum amount of resources available.

3.4 The deployment architecture

After developing the components of our application, the next step in our project is to deploy our microservice. In this section, we detail the design of our deployment architecture from a global perspective, then we walk-through the deployment of the components of our logical architecture.

3.4.1 Kubernetes for container orchestration

In the section “3.2.1 Microservices architecture“ we detailed how the microservices architecture relies on the containerization concept. In our project, we are going to build a microservice that consists of multiple containers for the database and the back end. What’s more is we must follow that by using these containers in a deployment environment along with triggering the Spark CronJob and the schema manager components. Not to mention that we need to integrate our solution with the promotion application used by Cognira. Since the Cognira promotion management application is also built around microservices and containers.
To pull off this complex deployment model, we used Kubernetes(K8s)[22]. Kubernetes is an open-source container orchestration system used for automating the deployment, scaling, and orchestration of containerized applications.
Kubernetes has lately become the standard and the leading software in container orchestration. In order to be so, it defines a new deployment model and uses a specific architecture and set of components. In the next paragraph we will highlight the key components of a Kubernetes deployment architecture.

3.4.2 Kubernetes architecture components

The deployment in Kubernetes is based on managing a collection of computers that operate together as a single compute unit. These abstractions allowed Kubernetes containerized applications to be deployed in a cluster without having to bind them to specific machines. To make use of this new deployment model, it is necessary to package applications. This will serve the purpose of separating them from the individual hosts. The separation leads to the compatibility of Kubernetes with the containerized applications and the microservices architecture.
The role of Kubernetes is to automate in a more efficient way the distribution and scheduling of containers throughout the cluster. A Kubernetes cluster consists of two types of resources:

- **Master node:** The master node is the main component of a Kubernetes architecture. It is responsible for providing the Kubernetes API. As well as the scheduling of the deployed containers and the management of the entire cluster. The components of the master can be run on any node in the cluster. The key components are:
  - **API server:** Since the communication between microservices is going to be through HTTP requests, the API server plays the role of an entry point to all of the REST requests, and it represents the only user-accessible element in the Master Node.

  - **etcd:** A distributed efficient key-value storage used to store the Kubernetes cluster information, API items, and service identification details.
  - **Scheduler:** The role of the scheduler in the Kubernetes architecture is to Watch and assign nodes for the newly-deployed containers.
  - **Controller:** Controls the behavior of the deployed containers in the Kubernetes architecture.

- **Worker nodes:** The worker nodes host the deployed containers and provide them with the necessary computing, networking and storage resources. Worker nodes also provide all the services required for networking between containers, for communication with the Master node, and for the allocation of resources to the containers. We find present on every node these two component:

  - **Kubelet:** Monitors the condition and ensures that the container is functioning and running. The kubelet also communicates with the data store(etcd) in order to provide information on services and writes details for the newly created ones.
  - **Kube-proxy:** Plays the role of a proxy service for the worker nodes and deals with forwarding the network traffic to the node's different resources. If a service is made accessible to the outer world, this component is the one responsible for forwarding this access to the designed container over the multiple isolated networks in the cluster.

In the next figure, we are going to display an overview of the Kubernetes deployment architecture.

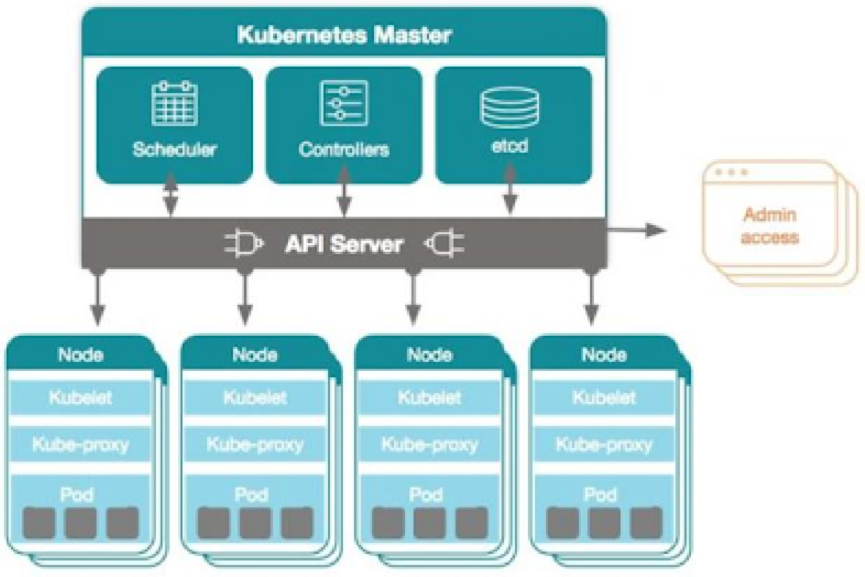


Figure 3.5: Kubernetes architecture [23]

3.4.3 Kubernetes main resources

After illustrating the architecture of the Kubernetes deployment model. In order to demonstrate the integration of our application within this architecture, we need to define the resources that we are going to use in Kubernetes:

- **Pod**

  A pod is the basic building block in Kubernetes, defined as the smallest and simplest unit that can be created or deployed in the Kubernetes object model. A Pod encapsulates a container (or several containers in some instances) and provides the physical storage resources, a unique IP network and options in which the user can govern the running of the container(s).

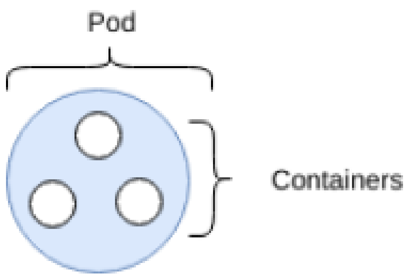
  

  Figure 3.6: Multiple containers running inside a pod [24]

- **Deployment**

  A Deployment Controller provides Pods and ReplicaSets with declarative updates. Noting that a ReplicaSet is a set of n replicated pods running at the same time. A deployment is the most common type of a Kubernetes resource and it is used as an abstraction via a single interface over replicates and pods. In simpler terms, deployment is the engine for Kubernetes in building and updating application instances. After the creation of a deployment, the Kubernetes master schedules the instances of the application to individual nodes within the cluster. Once created, the deployment controller continuously monitors these instances.

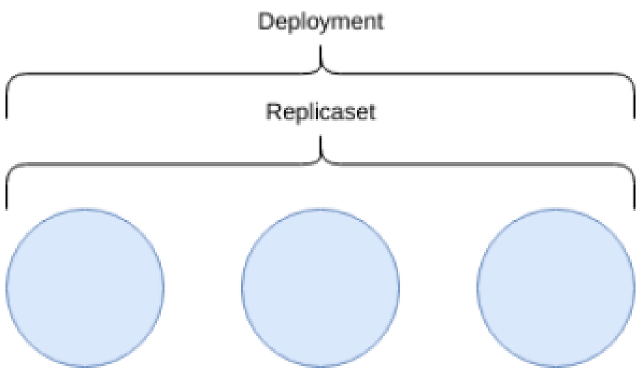
  

  Figure 3.7: Deployment wraps a ReplicaSet of pods [24]

- **Services**

  Pods are an effective way to encapsulate a container. However, one disadvantage of this resource is that it is almost impossible to reference a particular pod without complex logic to monitor the topological changes. This is in fact due to the short-lived nature of pods. Hence ReplicaSets enforce the constant deleting and creation of pods by scaling up and down the number of pods. Services solve this particular problem by providing an abstractional mapping to pods. This mapping defines a logical set of pods as well as the policy for accessing them. Moreover, services define an addressable way to communicate with the pods. The figure (1.8) showcases how services provide a way to access a set of replica pods.

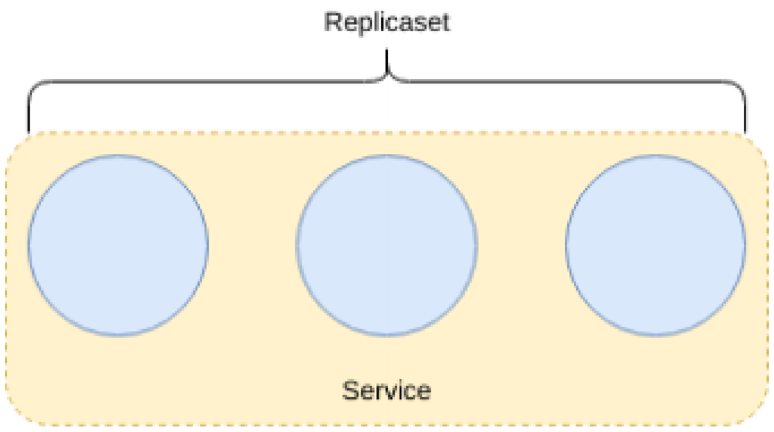


Figure 3.8: A service spanning over a ReplicaSet [24]

- **Job**

  A Kubernetes Job generates one or more pods and guarantees that at least one of these pods is terminated successfully[25]. The job tracks the successful completions of the launched pods and once the pods are completed successfully, the job is thus completed. When removing a job, the pods that were created are deleted. Generally, the purpose of using jobs is for pods that are expected to perform a specific task then get deleted.

- **CronJob**

  A CronJob is a higher level variant of a job, a CronJob in Kubernetes is an implementation of Linux's Cron as it helps create and manage jobs[26] according to a timetable. One CronJob object regularly runs a job in Cron format on a certain schedule.

3.4.4 The deployment of the promotion configuration service

After defining the architecture and the main resources of a Kubernetes deployment environment, in this paragraph we illustrate the deployment of each logical component of our microservice.

- **Database**

  The database consists of a container running within a pod that is mapped to a service(referenced as svc in the next figure). Within this container, we can access our data that is defined according to the data model of our application. The data model is created by the schema manager. In order to secure the data in the configuration service, the other components can access the database in the Kubernetes architecture through a resource called a Kubernetes secret. A secret can hold the sensitive information for our application which will be in this case the database credentials before making a connection to it and editing the configuration data.

  The next figure represents a Kubernetes deployment diagram of the database.

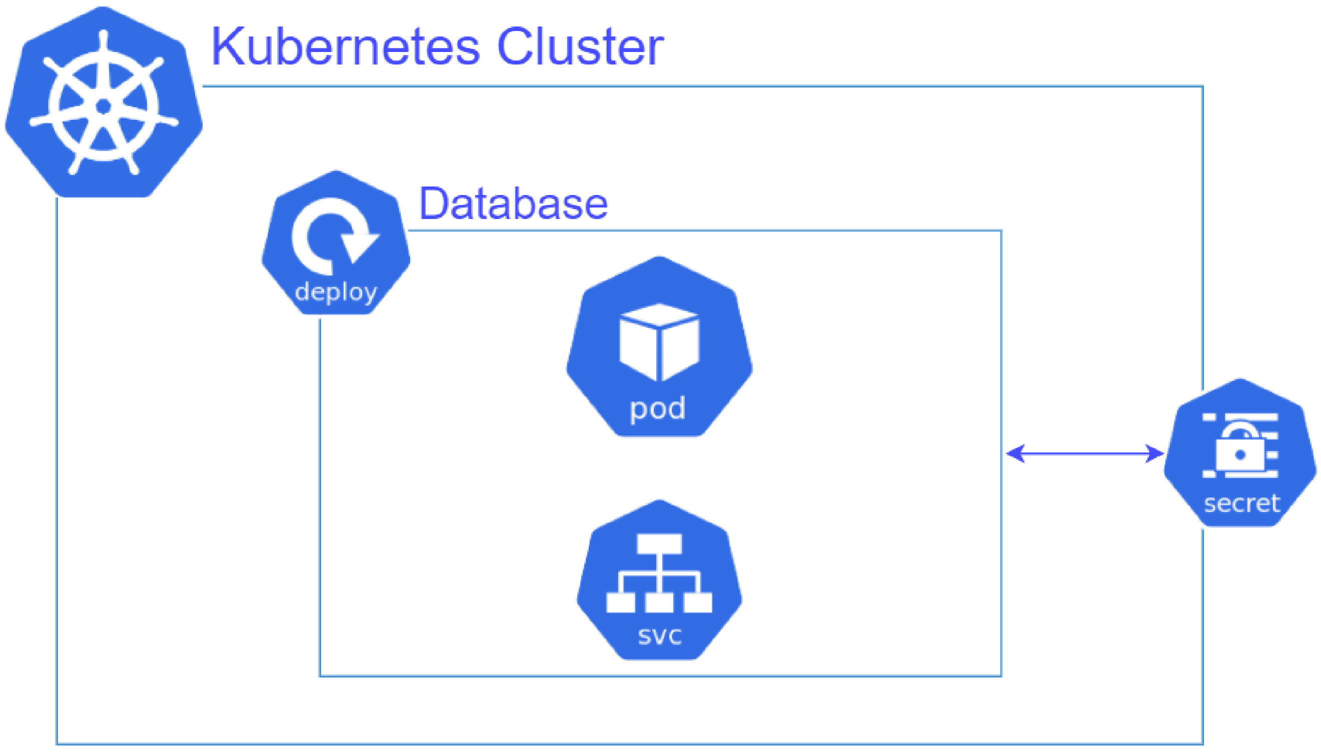


Figure 3.9: Database deployment Kubernetes diagram

- **Schema manager**

  The purpose of the schema manager is to create or update the schemas of the tables within our database. This component needs to be launched only when a creation of a new schema or an update is occurred. This led us to deploy the schema manager as a Kubernetes job in order to optimize the consumption of the physical resources. The pod inside this job is going to be triggered only when the conditions previously mentioned are met. In the next figure we demonstrate how the schema manager is deployed and how it is related to the database in a Kubernetes diagram

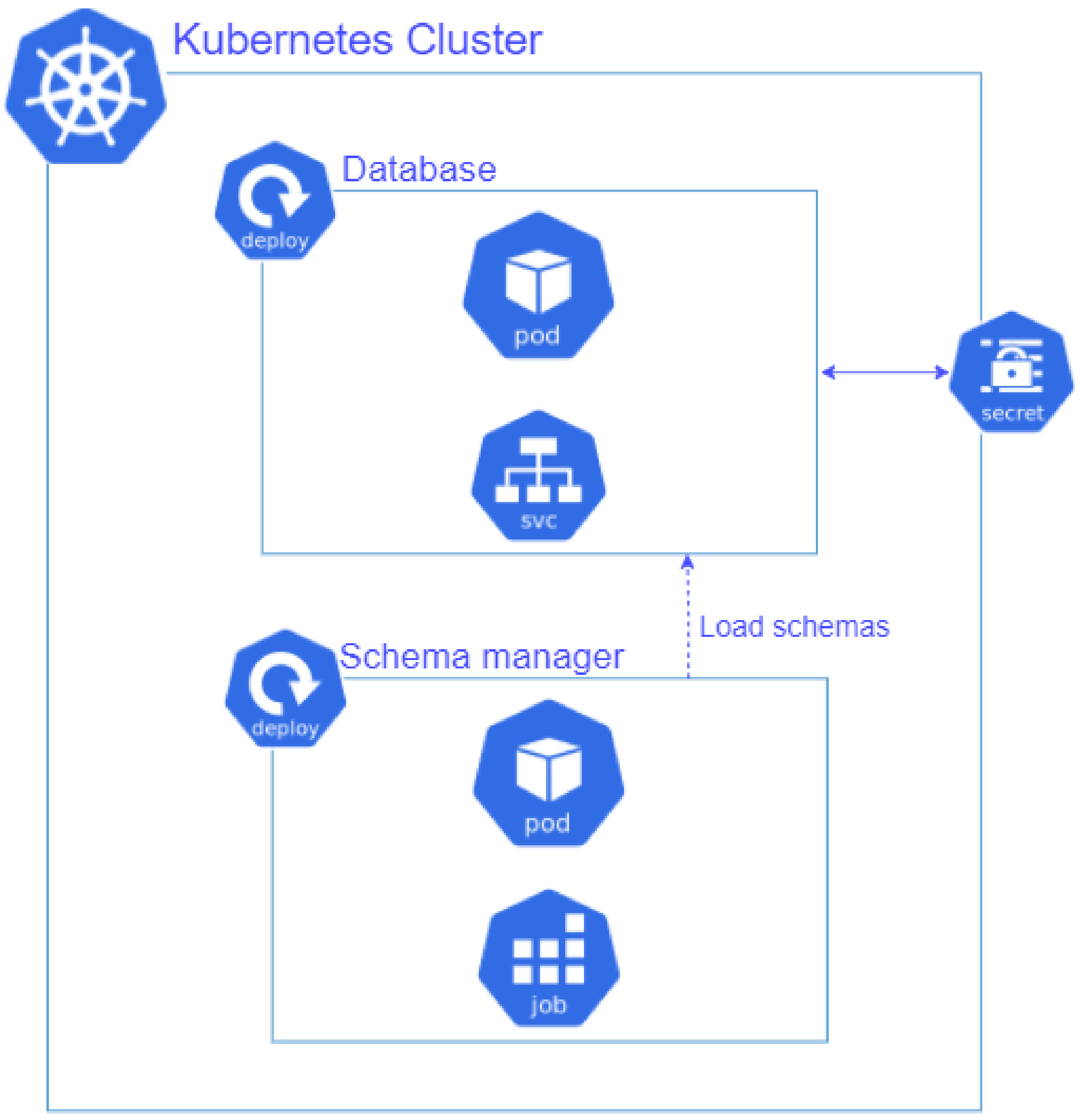


Figure 3.10: Feeding schemas to database with schema manager Kubernetes diagram

- **Back end**

  Unlike the other components, in order to package the code of our back end application and run it into a pod, we need to wrap and save the dependencies and the libraries that come with the application code. To accomplish this, Kubernetes offers a resource called a Kubernetes volume(referenced as vm in the next figure). A Kubernetes volume is a directory in the cluster that can contain the dependencies and make them accessible to containers in a given pod.

  Another complexity that we face when deploying the back end application, is that in the microservices architecture, our container needs to be independent of the original host(machine) it was first created on. This means that the environment configuration for our back end application on Kubernetes needs to be isolated. To solve this problem, we used Configmaps(referenced as cm in the next figure). These resources allow us to bind configuration files, command-line arguments, environment variables, port numbers, and other configuration artifacts to our pods containers and system components at runtime.

The kubernetes deployment diagram for our back end integrated with the database and the schema manager is described in the next figure

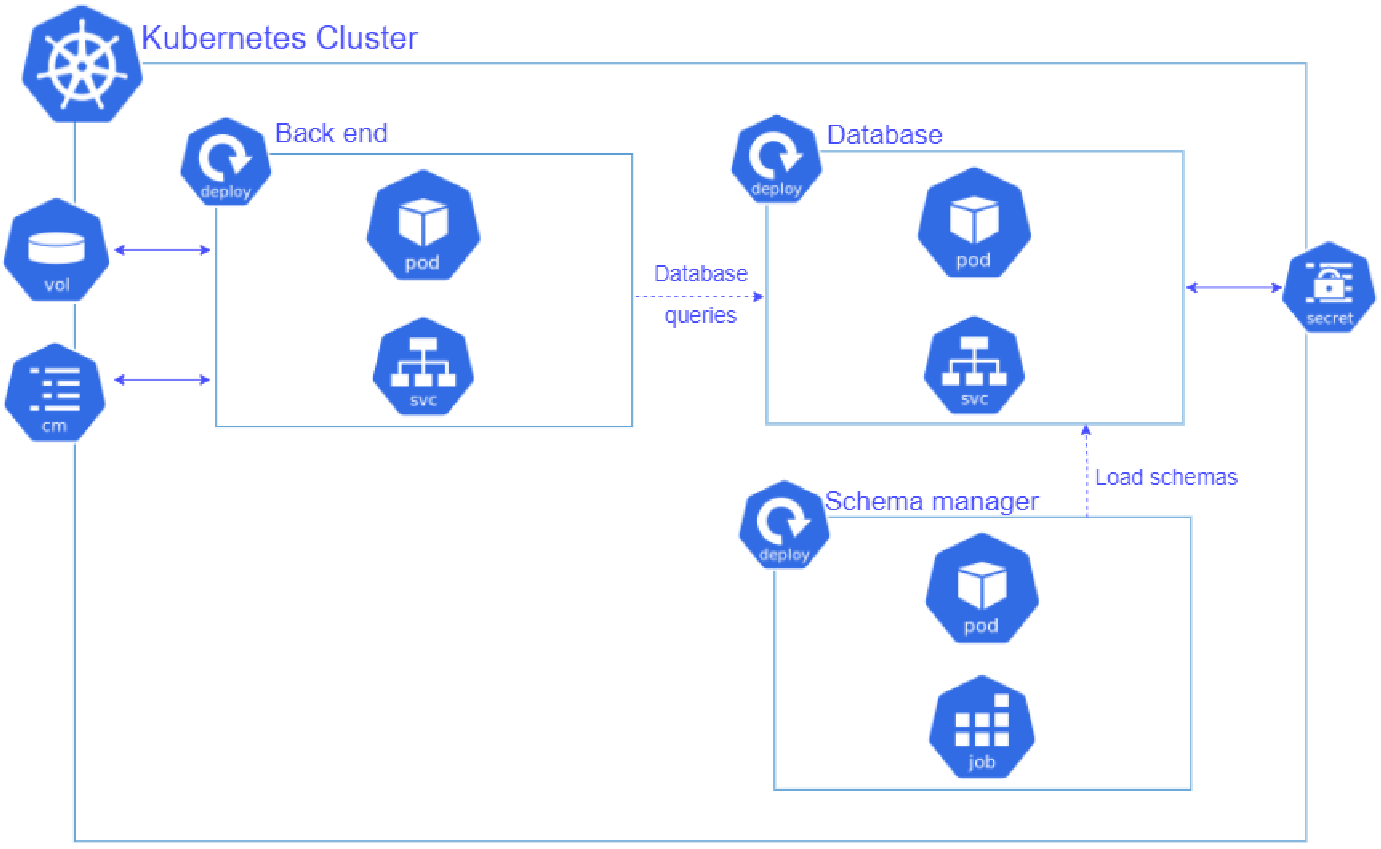


Figure 3.11: Deploying the backend application Kubernetes diagram

- **Spark CronJob**

  The CronJob software feature is already implemented as a resource in Kubernetes. In order to deploy the Spark application, we need to package the code of this latter deploy it into a pod. Subsequently, we map this pod to a CronJob. In that way, this pod will be triggered only in a certain timetable which happens to be every 24 hours in our case. After the files are versioned and exported to the data lake, the CronJob will end and the associated pod will be deleted.

  Due to the amount of configuration data that we aim to version and export each period, the Spark application responsible for this task will consume a huge amount of resources. This explains the use of a CronJob as it limits the deployment time of this application on the cluster. The CronJob leads us to avoid a huge memory and information squandering.

  In the figure below we illustrate the deployment of the CronJob integrated with the rest of the components of our system.

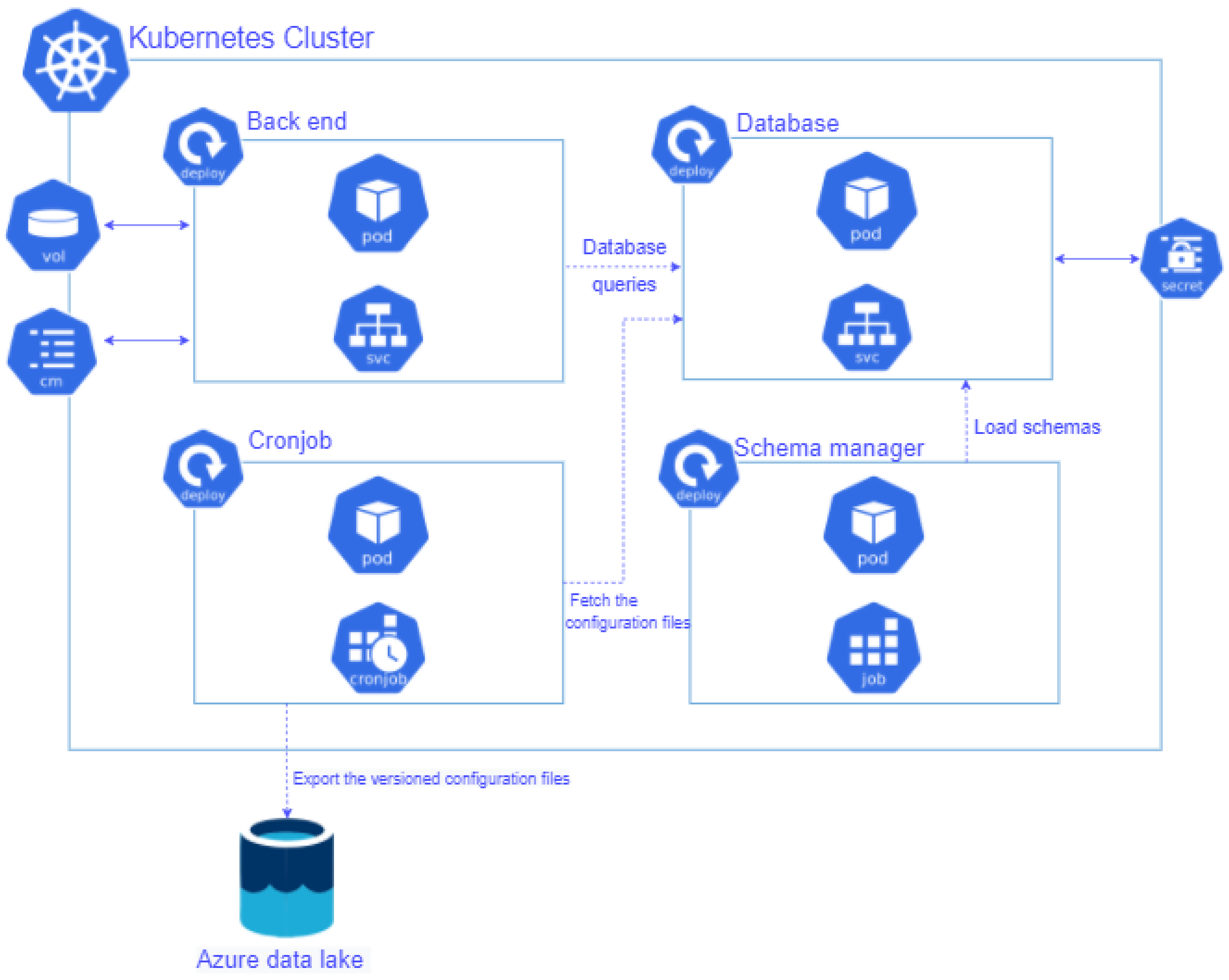


Figure 3.12: Kubernetes diagram of the promotion application configuration microservice

Conclusion

In this chapter, we surfed through the global design for the retail promotion configuration service. In the next chapter, we analyze the set of features and requirements that we defined for this project using mainly UML diagrams.

CHAPTER 4

DETAILED DESIGN OF THE CONFIGURATION SERVICE

Introduction

Throughout this chapter, we present a detailed overview of our application using activity diagrams to detail the behaviour of the promotion configuration service. We also use sequence diagrams to illustrate the various interactions the actors can have when using our solution. Then we move on to class diagram to display the logical interactions between the components of our application, followed by a deployment diagram to demonstrate the interchange between the components of our application in the Kubernetes deployment model.

4.1 Activity diagrams

Activity diagrams are important behavioral diagrams in UML modeling. They help describe the dynamic aspects of the system. An activity diagram can be considered as an advanced version of flow chart that models the flow from one activity to another. Thus, it represents the sequence of tasks to be executed in the set of processes implemented in the system.
In this part, we present the activity diagrams for editing the configuration data by the users. We also implement the diagram for the execution of the Spark CronJob that versions and exports the data to the Azure data lake. These diagrams avail to obtain a detailed view of the dynamic behavior in our system.

4.1.1 Activity diagram: edit configuration data by the user

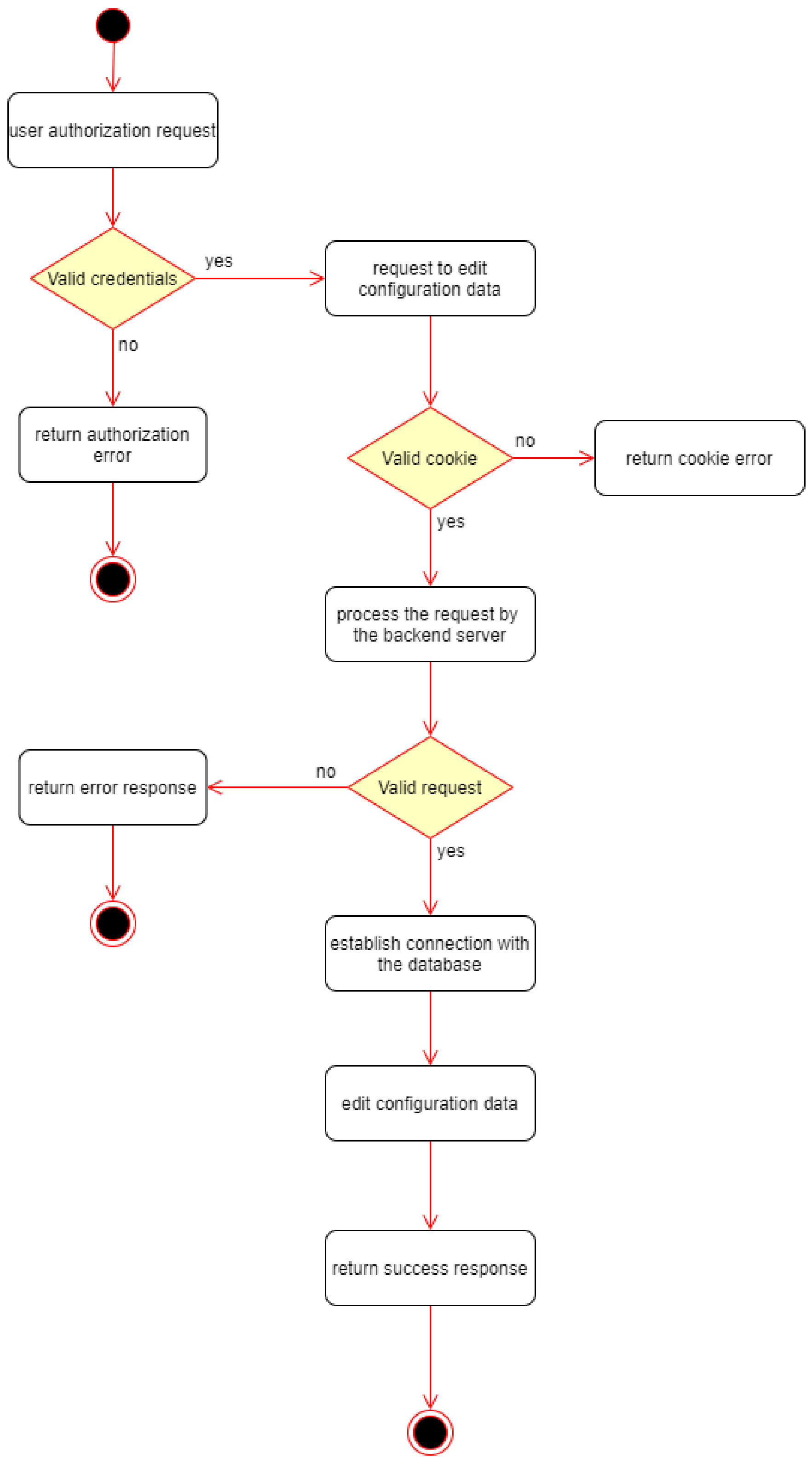

user authorization request
Valid credentials
yes
no
request to edit configuration data
return authorization error
Valid cookie
no
return cookie error
yes
process the request by the backend server
return error response
no
Valid request
yes
establish connection with the database
edit configuration data
return success response

Figure 4.1: Edit configuration data by the user activity diagram

4.1.2 Activity diagram: version and export the data to Azure data lake

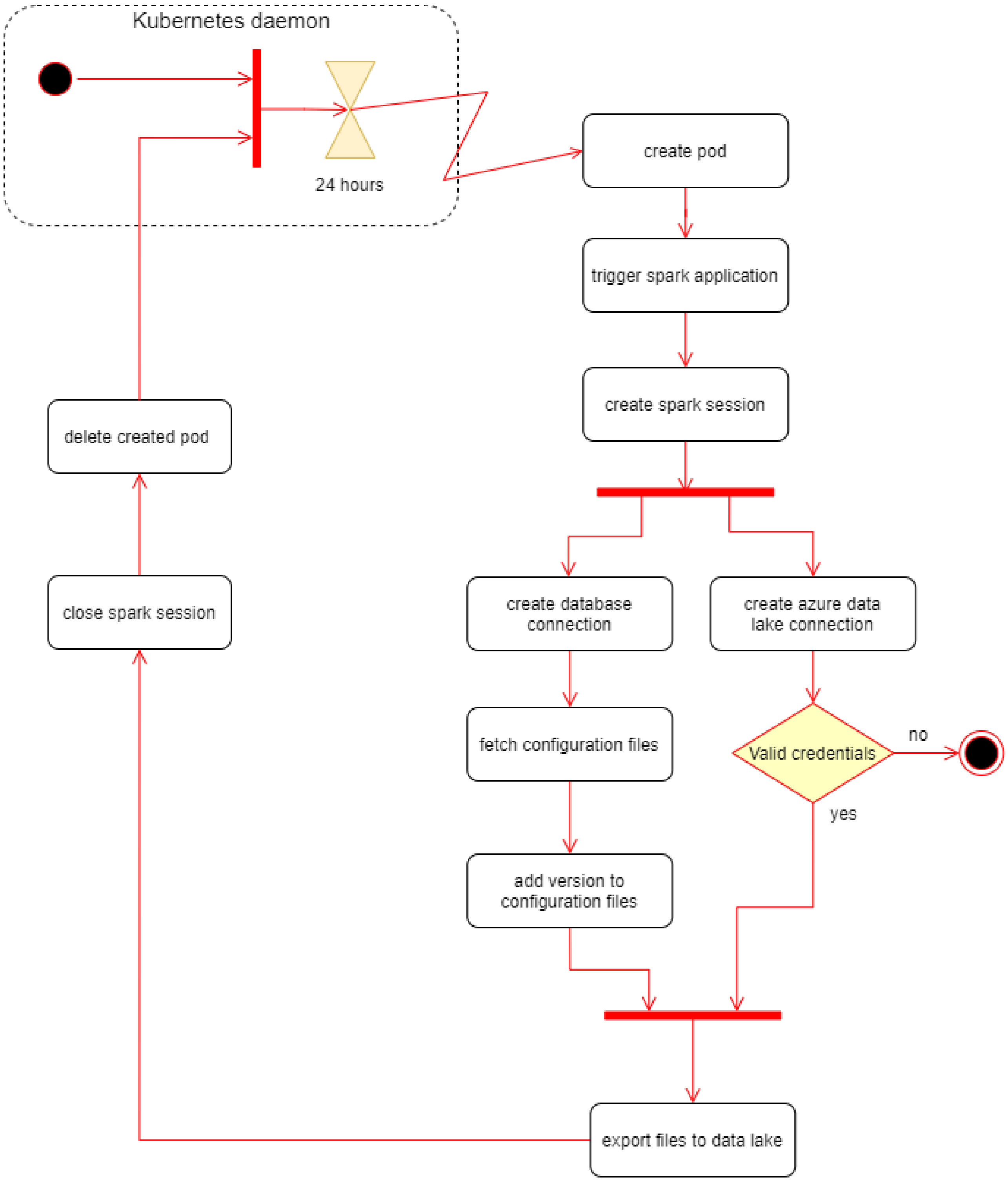

Kubernetes daemon
24 hours
create pod
trigger spark application
create spark session
create database connection
create azure data lake connection
fetch configuration files
Valid credentials
no
yes
add version to configuration files
export files to data lake
close spark session
delete created pod

Figure 4.2: Version and export the configuration data to Azure data lake activity diagram

4.2 Detailed Sequence diagrams

When used in the conception of our application, UML sequence diagrams can help us specify in detail the various interactions between the application components in each use case. They also showcase the interactions the user can have with those components.

4.2.1 Authorization request sequence diagram

In order to be able to edit the configuration files in the database, the user needs to be authorized to send HTTP requests to the backend server. This latter forwards the authorization request from the user to the security microservice. The security microservice is the microservice responsible for verifying the user credentials and generating the Json Web Token(JWT). The JWT can be served as a cookie for the user in order to make future requests. These requests can be rather getting, updating, deleting or inserting configuration data.
The Json Web Token (JWT) is in fact an Internet standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed[27].
Below we illustrate the sequence diagram for the authorization request.

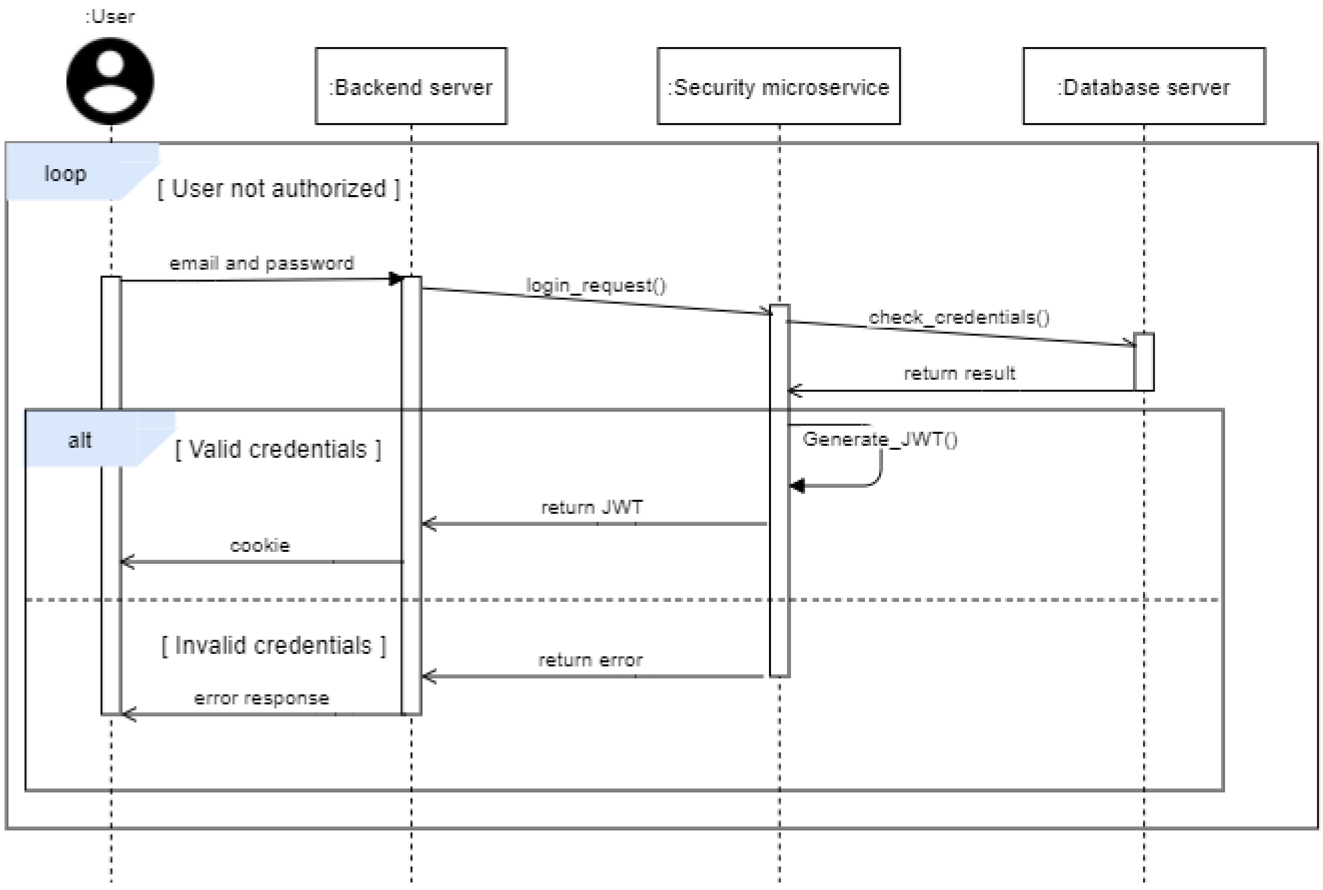


Figure 4.3: Authorization request sequence diagram

4.2.2 Edit configuration data sequence diagram

When editing the configuration data, the user of our application needs to get the authorization from the security microservice, then with every request to the backend server the cookie in which the user's JWT is held is decrypted and verified.

In case the user credentials are valid, the backend server is responsible for querying the database and applying the necessary changes on the data then calling back the user with the appropriate response.

The next figure represents the sequence diagram of the configuration data editing operation.

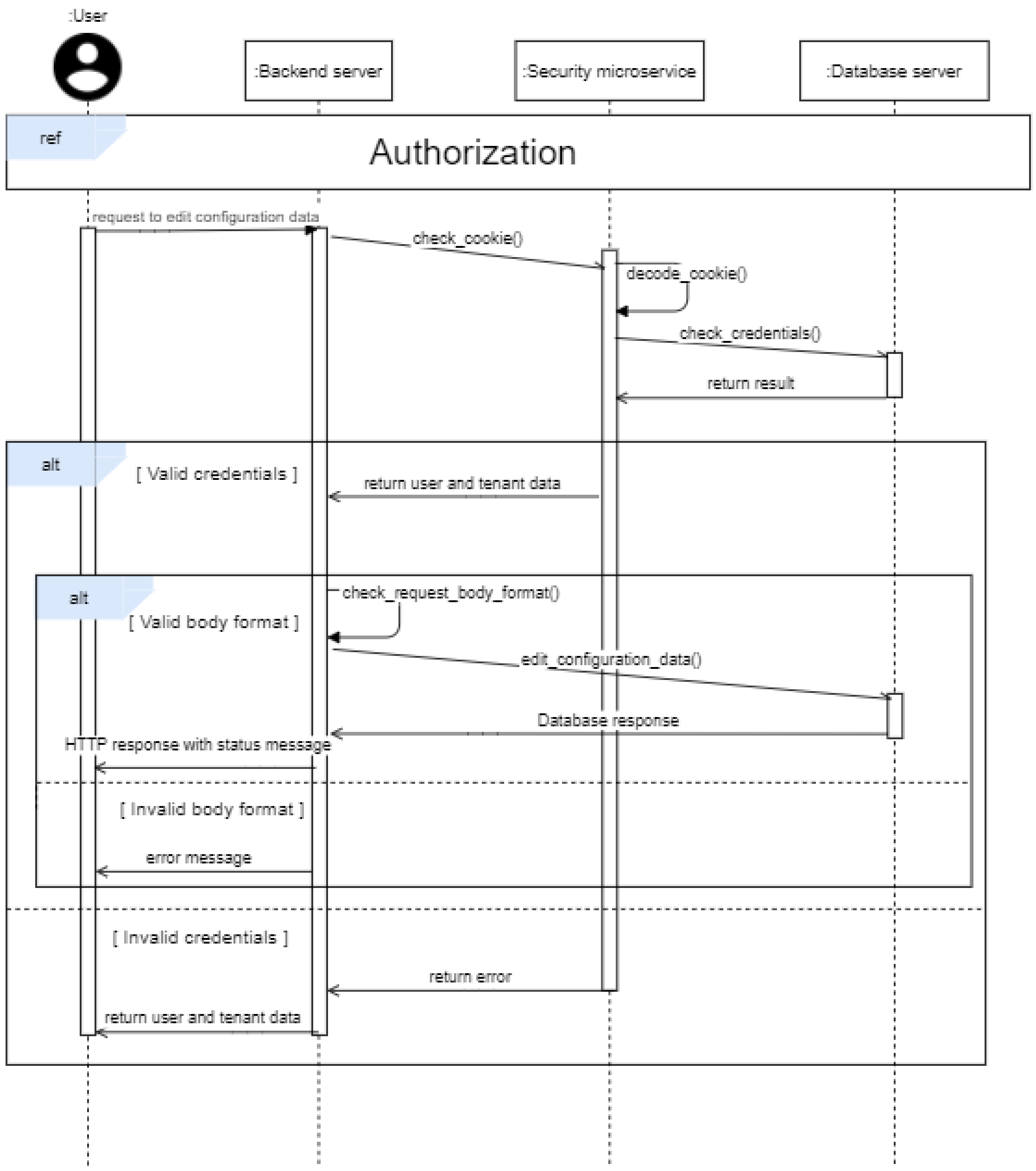


Figure 4.4: Edit configuration data sequence diagram

4.2.3 Version and export configuration files to Azure data lake with Spark sequence diagram

In order to export the configuration files to the Azure data lake, we are going to launch a Spark application to fetch these files from the database, process the data and finally export it to the Azure data lake.

In the next sequence diagram we explain the lifecycle of our Spark application.

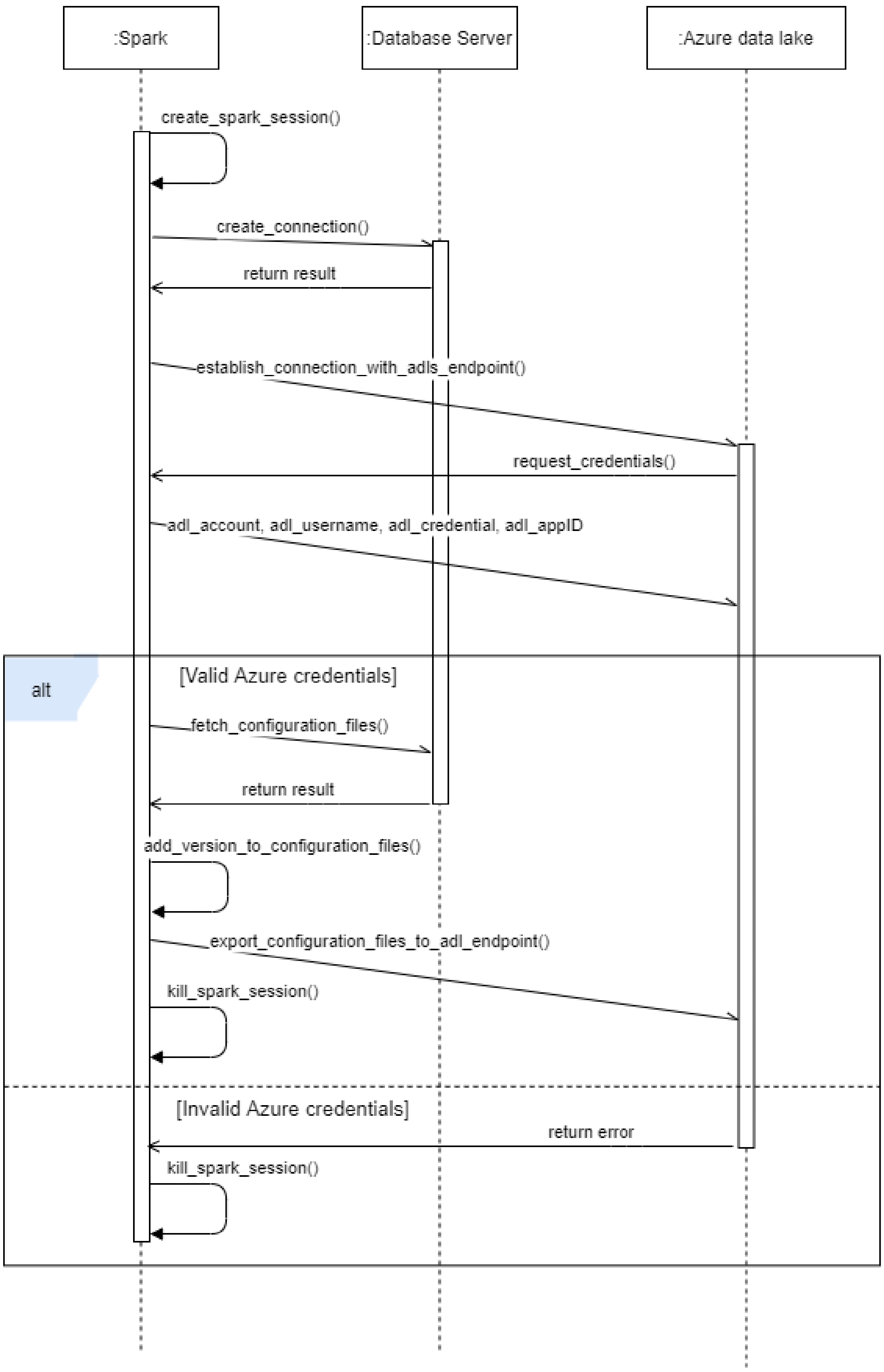

:Spark
:Database Server
:Azure data lake
create_spark_session()
create_connection()
return result
establish_connection_with_adls_endpoint()
request_credentials()
adl_account, adl_username, adl_credential, adl_appID
alt
[Valid Azure credentials]
fetch_configuration_files()
return result
add_version_to_configuration_files()
export_configuration_files_to_adl_endpoint()
kill_spark_session()
[Invalid Azure credentials]
return error
kill_spark_session()

Figure 4.5: Version and export configuration files with Spark sequence diagram

4.3 Class diagram

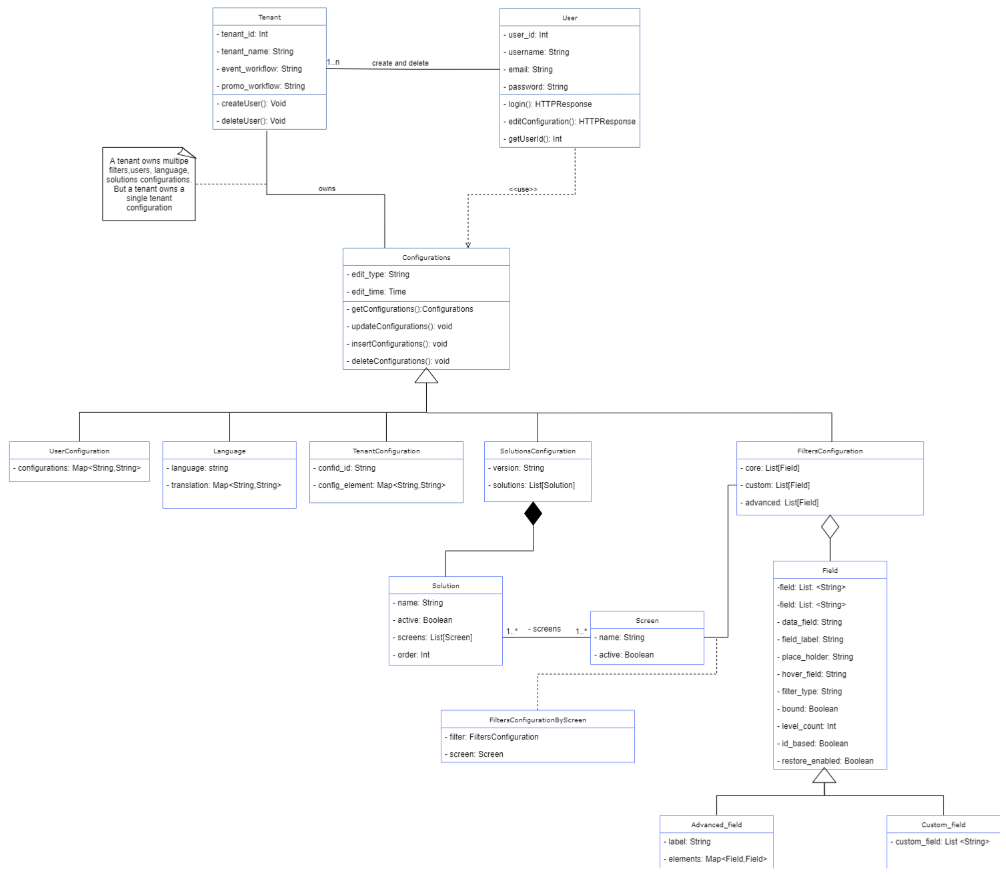


Figure 4.6: Class diagram

4.4 Deployment diagrams

Deployment diagrams are an important asset in the design of the promotion configuration service, as in UML modeling, these latters give the necessary insight for visualizing, specifying, and documenting the client/server interaction. Deployment diagrams also document the interaction between the several deployment nodes of the system. In the next paragraphs, we break down the deployment architecture of the back end application in a deployment diagram and we follow that with a diagram that details the deployment of the Spark CronJob.

4.4.1 Key concepts

Before illustrating the deployment diagrams for the components of our application, we define the key concepts that we used in these diagrams:

- **TCP/IP protocol**
  TCP and IP are two different protocols that work together to ensure that data reaches its intended destination within a network[28]. IP collects and defines the address of the application or device to which data must be transmitted (the IP address). TCP is then in charge of carrying and routing data throughout the network architecture, making sure it reaches the intended application or device.
  In the deployment architecture of the promotion configuration service, we used TCP/IP in the communciation between the user and the backend application. We also used TCP/IP between the Azure data lake and the Spark CronJob.

- **HTTPS protocol**
  Hypertext transfer protocol secure (HTTPS) is the secure version of HTTP, which is the primary protocol used to send data between web services. HTTPS is encrypted in order to increase security of data transfer.
  In the deployment of the promotion configuration service, HTTPS is used for exchanging the requests and the data between the internal microservices of the promotion management application and the developed back end application.

- **CQL protocol**
  The CQL binary protocol is a library which implements Apache Cassandra's native query language protocol[29]. With Apache Cassandra being the database used to build the promotion configuration service, we used the CQL protocol for querying the database, fetching and editing the configuration

data.

- **Java virtual machine**

  the Java virtual machine (JVM) is a software that translates a Java bytecode (intermediate language) into machine language and executes it. The JVM is the Java Platform's runtime engine. It allows any application written in Java or another language that is compiled into Java bytecode to run on any machine with a native JVM.

- **JAR applications**

  When used, a JAR file allows Java runtimes to efficiently deploy an entire application**jar**. the JAR file wraps the application including its code and the associated resources, in a single request. In our application, the code for the Spark CronJob is wrapped into a JAR file. When this file is triggered, an independent Spark application is built, this application does the job of fetching the configuration files and exports them into the Azure data lake.

- **Entrypoint scripts**

  In docker, an entrypoint script is a set of instructions that allow us to configure a container. The container then runs as an executable. An entrypoint script is exactly the tool we need to trigger our Spark application. In fact, entrypoint scripts look similar to command line instructions, they allow us to specify a command with parameters. The difference is that entrypoint scripts get triggered once and only at the creation of the docker container.

- **Environment configuration files**

  In a Kubernetes deployment model, the microservices need to be isolated from the machines used to build them, configuration files are introduced to answer this issue as they hold the environment variables for these applications such as the different hosts and ports used to access the different components in the cluster, the database and the cloud credentials along with the set of networks used in the system.

4.4.2 Deployment diagram: The back end application

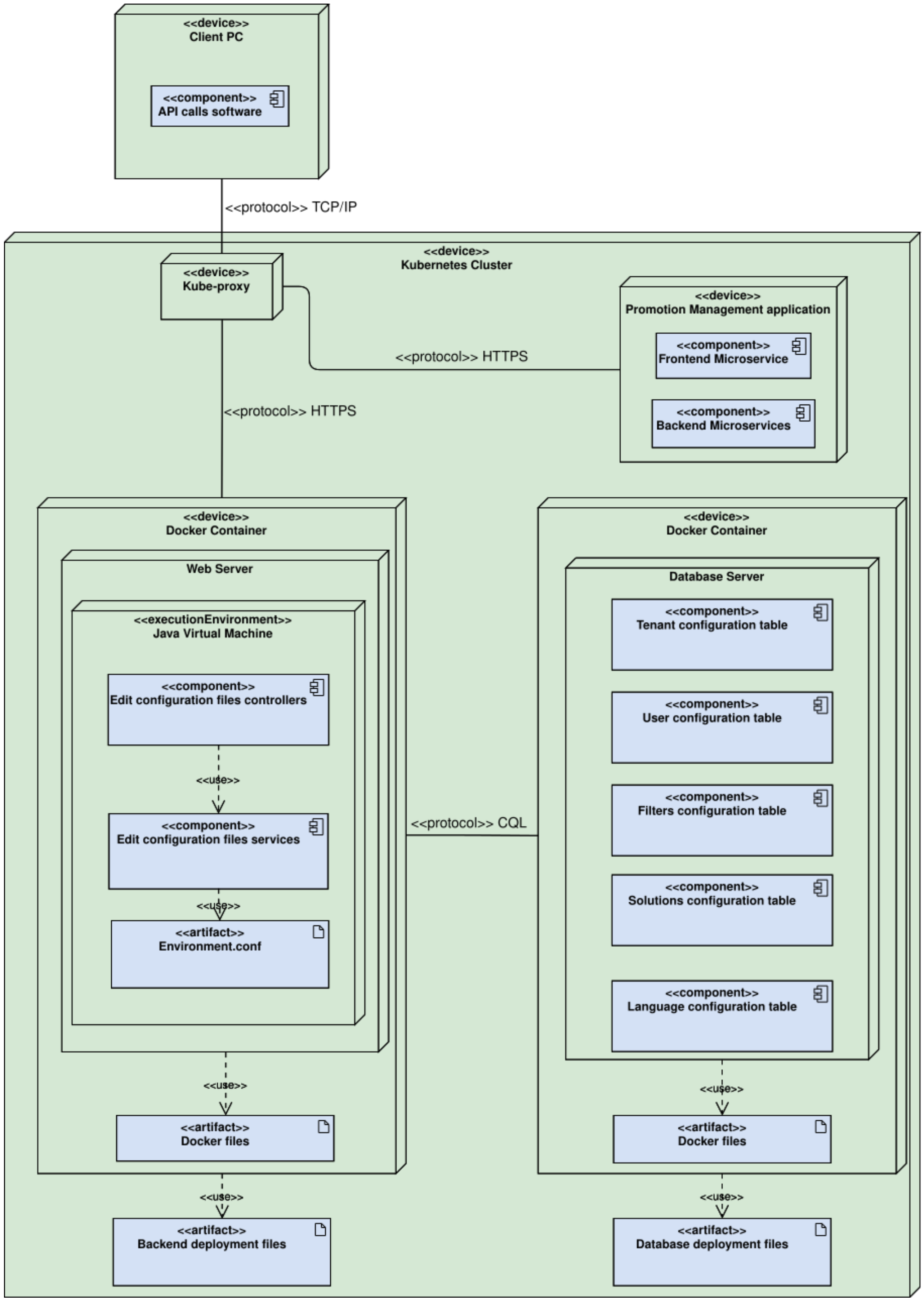


Figure 4.7: Backend application deployment diagram

4.4.2.1 Deployment diagram: The Spark CronJob

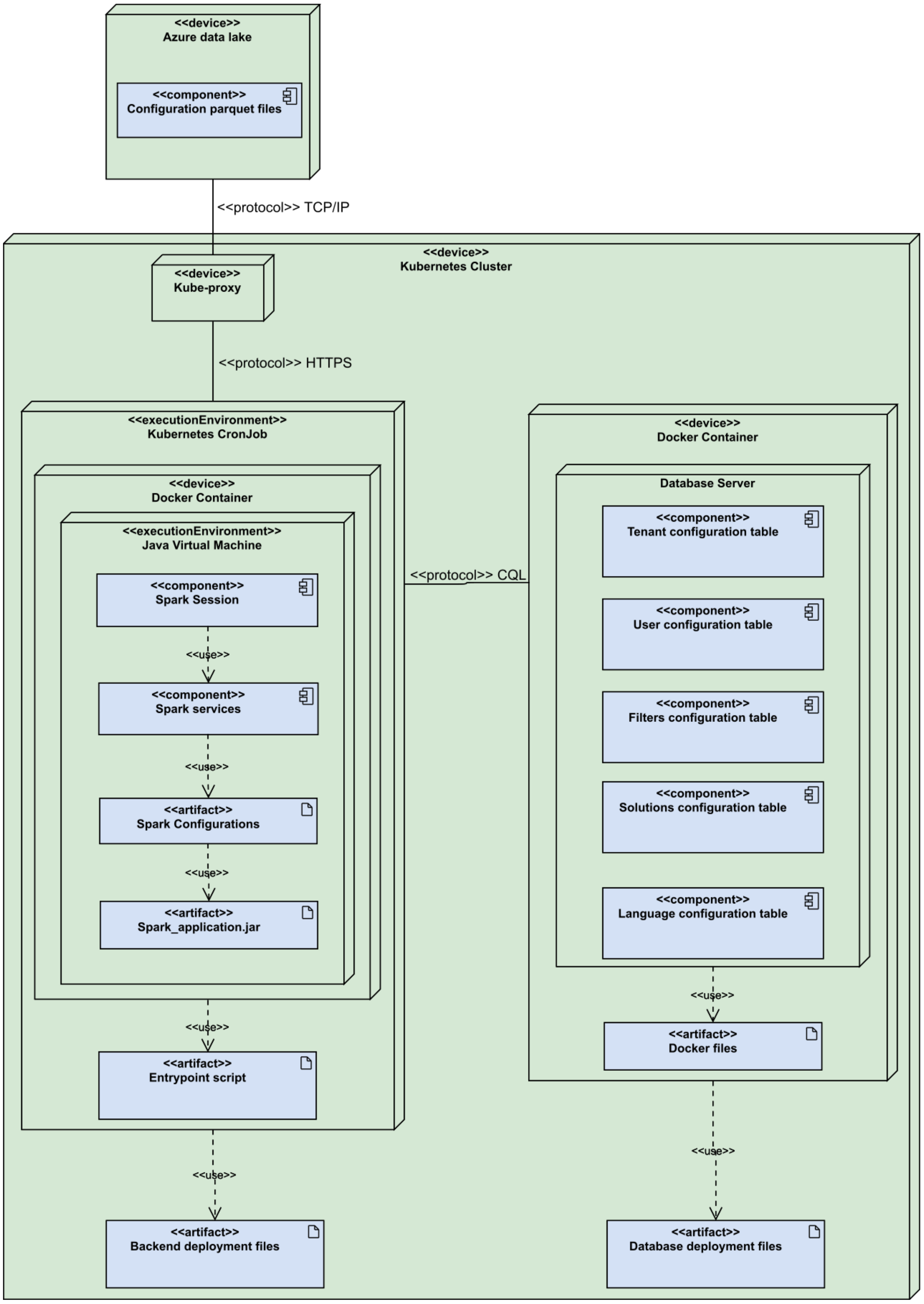


Figure 4.8: Spark CronJob deployment diagram

Conclusion

In the previous four chapters, we covered the project scope, the necessary functionalities and requirements to implement and we followed that with the global and detailed design of the promotion configuration service. What is yet to do is to describe the implementation steps in this internship project. For that, we dedicated the next and last chapter to go through the implementation details.

CHAPTER 5

IMPLEMENTATION DETAILS

Introduction

This chapter is dedicated to exposing the technical details related to the implementation of this project. We begin with describing the software and hardware environments. After that, we go through the main technologies we opted to choose to build the promotion configuration service. Finally we walk-through the main features implemented in our application.

5.1 The development environment

For the purpose of building our microservice, the working environment plays an important role. It is one of the main factors in ensuring the delivery of first quality product and rapid process of development. Our working environment is shown as below:

5.1.1 Hardware environment

Computer	Dell Latitude E5570
Processor	Intel i7-7600U 2.90 GHz x 4
RAM	16 Gb

Table 5.1: Hardware environment

5.1.2 Software environment

Operating system	Windows 10 Home Edition
IDEA IntelliJ	An Integrated development environment that supports multiple built-in languages (mainly Java and Scala). This IDE was used to write code for the back end application and also the Spark application. IntelliJ was also beneficial for code documentation and git operations.
Draw.io	An online and free diagram editor. We used this tool to draw the data model, the UML diagrams and the sketches indicated in the previous chapters.
Docker Toolbox	A command line environment to quickly and easily install and setup Docker environments on a local machine.We used this tool to build docker containers.
Minikube	A tool that runs Kubernetes locally. Minikube runs a single-node Kubernetes cluster on a local machine. We used this tool to create a local deployment environment
Postman	A collaboration platform for APIs development. We used Postamn to test the developed APIs by sending HTTP requests

Table 5.2: Software environment

5.2 Project progress chronogram

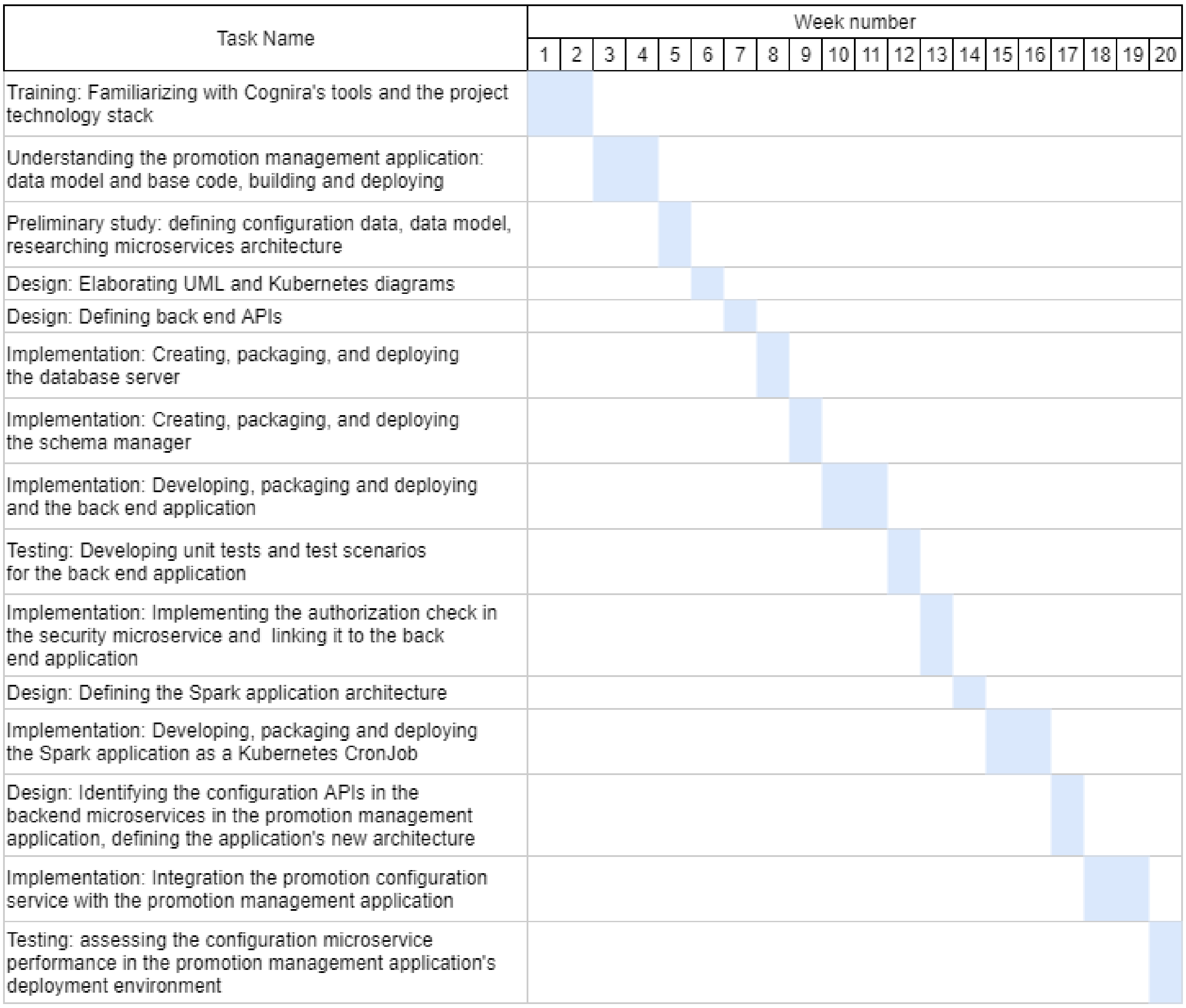
We planned The progress of this project according to the timetable presented in the table 5.3. Later on, We stick to this time organizing chart in order to make sure the project features are implemented as planned.

To make sure the project goes as planned and in order to keep treacebility on the work done during the internship, we followed these measures:

- Daily 10-15 minutes stand-ups, to catch up on the progress.
- Logging the progress on JIRA, the project tracking platform and leaving notes about the multiple steps in the project.
- Possibility to schedule meetings to discuss and work on blockers.
- Progress presentations on a bi-weekly basis.

- Scheduled meetings with the software development team for a code review after implementing each feature in order to assess the quality of the code.

In the next table, we illustrate the Gantt chart for the project progress chronogram



Task Name	Week number																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Training: Familiarizing with Cognira's tools and the project technology stack	▨	▨																		
Understanding the promotion management application: data model and base code, building and deploying			▨	▨																
Preliminary study: defining configuration data, data model, researching microservices architecture					▨															
Design: Elaborating UML and Kubernetes diagrams						▨														
Design: Defining back end APIs							▨													
Implementation: Creating, packaging, and deploying the database server								▨												
Implementation: Creating, packaging, and deploying the schema manager									▨											
Implementation: Developing, packaging and deploying and the back end application										▨	▨									
Testing: Developing unit tests and test scenarios for the back end application												▨								
Implementation: Implementing the authorization check in the security microservice and linking it to the back end application													▨							
Design: Defining the Spark application architecture														▨						
Implementation: Developing, packaging and deploying the Spark application as a Kubernetes CronJob															▨	▨				
Design: Identifying the configuration APIs in the backend microservices in the promotion management application, defining the application's new architecture																	▨			
Implementation: Integration the promotion configuration service with the promotion management application																		▨	▨	
Testing: assessing the configuration microservice performance in the promotion management application's deployment environment																				▨

Table 5.3: Project chronology Gantt chart

5.3 Technological choices

In this section, we present the various technological choices we made for this project. First we start with the programming languages, then the frameworks and libraries we made use of. The last part of this section is devoted to the project tracking and management tools.

5.3.1 Programming languages

In the following paragraphs, we illustrate the programming languages that we used to build the promotion configuration service

- **Scala**
  Scala is a multi-purpose programming language[30] that was created by Martin Odersky back in 2003. It provides strong typing, object oriented programming and functional programming. We chose Scala because it offers the same programming power as Java as it runs on Java Virtual Machine (JVM). In addition to that, Scala is a lot faster than other general purpose programming languages such as Python. Scala also offers great support for concurrency between computational tasks and it is therefore a solid choice for programming the back end application since we used asynchronous programming and a microservices architecture.
  Besides, Scala is also the native programming language for Spark applications. For that purpose, we used it for the development of our Spark CronJob.
- **CQL, Cassandra query language**
  CQL is a language used for managing and communicating with data living inside a Cassandra database. CQL follows a similar syntax to the conventional Structured Query Language (SQL). However, it offers multiple other advantages such as scalability, high availability and fault tolerance. We chose Cassandra over other SQL databases because it is more adjusted to the huge amount of configuration data. Also, we chose Cassandra over other NoSQL databases such as Graph databases(Neo4j for example), MongoDB in which the data is stored in JSON format since our configuration data is columnar[31]. This means that the Cassandra availability and data model are more suitable to our use-case.
- **Bash**
  Bash is a Unix shell and command language written by Brian Fox for the GNU Project. We used Bash scripts in our project for the Kubernetes deployment pipelines and for scripting the Docker files. This is due to the Linux-based nature of the Kubernetes environments.

5.3.2 Frameworks and libraries

These paragraphs are dedicated to presenting the various libraries and frameworks used in this project.

- **Sbt**
  Sbt is an interactive tool for building Scala projects[30]. It also offers the capabilies for running the Scala projects and testing with libraries like sbt-test. Using sbt, we can also package and

deploy Scala and Spark applications with libraries such as sbt-pack and sbt-assembly. We chose SBT over other build tools such as Maven since sbt offers the same libraries. Moreover, sbt comes with an integrated command line to assist the development process and sbt build files can be written in Scala.

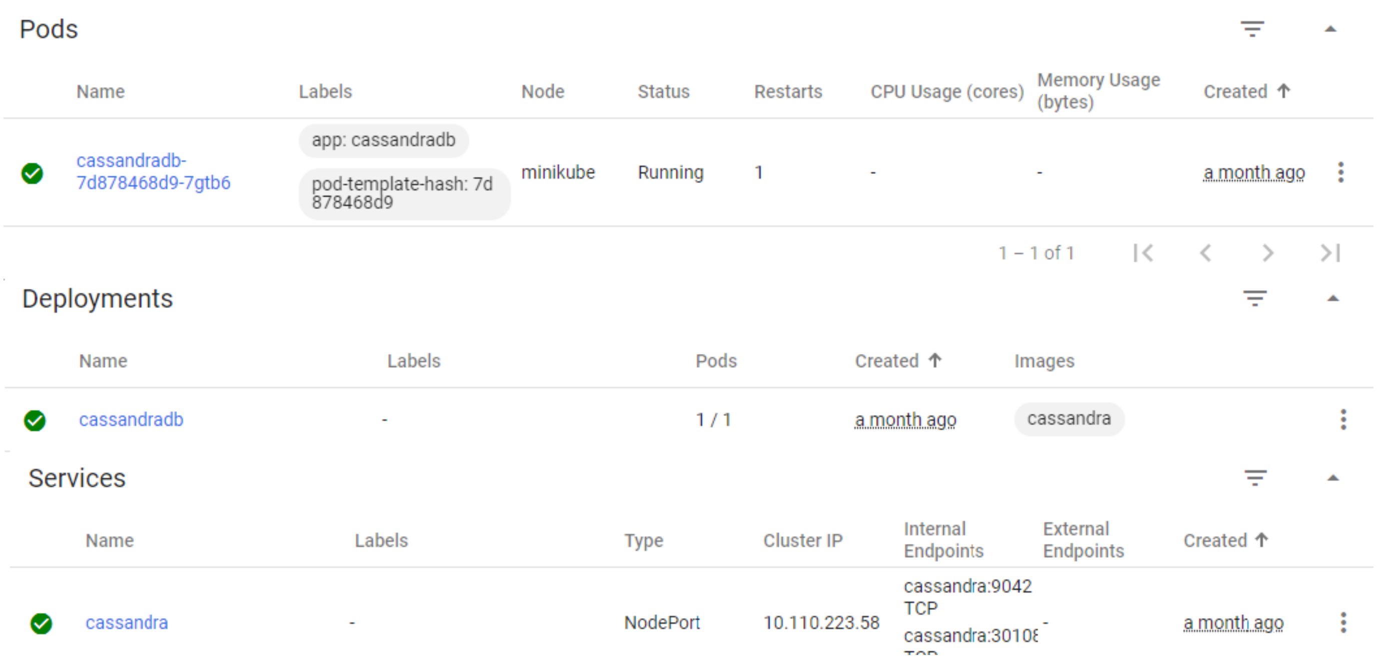
- **Akka-HTTP**
  Akka-HTTP is a library that provides both server side and client side tools to implement reliable and highly performing HTTP based services [30]. We used Akka-HTTP to implement the web server and the APIs of our back end application.
- **Datastax Cassandra connector**
  The Cassandra connector offered by Datastax is the driver that allows our back end application to connect to the database. It also allows to send requests that contain CQL statements [URL26]. This offers to the back end application the capability to interact with the database directly to perform the operations needed, such as reading and writing configuration data in Apache Cassandra.
  In this project, for fetching the configuration data with Spark from the database, we also used the Datastax Spark Cassandra connector library. In fact, the Spark Cassandra connector is the implementation of the original Datastax Cassandra library in Spark.
- **Spark core**
  Spark Core is the implementation of basic Spark functionalities in Scala[32]. This library provides operation such as distributing tasks, scheduling and the use of basic Spark data structures.
- **Spark SQL**
  Spark SQL is a Spark library used to process structured data. This library acts as a distributed SQL query engine. We used it to process the huge amount of configuration data fetched from the Cassandra database in our project.
- **Azure storage**
  This library allowed us to establish connection to the Azure data lake with Spark. It also allowed us to export the configuration data to Azure.

5.4 Implementation of the promotion configuration service

In this section, we cover the implementation steps through this project. We start by illustrating the implementation of the database server. After that, we provide details on the implementation of the back end application. Subsequently, we go to the implementation of the Spark CronJob. And finally, we describe the steps for the integration of our project within the promotion management application.

5.4.1 Implementing the database server

In our application, we need the Cassandra database server to be running as an independent deployment component in Kubernetes. To achieve that, we need to create a pod holding a Cassandra Docker container.
To build the container, we need to create a Cassandra YAML file and execute it each time we want to deploy our application. YAML files are human-readable files destined for data-serialization. In Kubernetes, YAML files can be used deploy the containers[27] such as our database container, the back end application container and the Spark CronJob container.
The yaml file of the cassandra database controls configurations such as the name of the container, the communication protocol(TCP in our case), the port on which we intend to acces the container and the number of replicas that we intend to create for the database server. We can also configure the packaged version of Cassandra that we use. This packaged version is encapsulated within an open source Docker image provided by Apache.
After executing the yaml file, a pod for the database server is created. Additionally, a deployment is created to manage the server state along with a service that enables the other microservices to communicate with our database server. The figure taken from the Kubernetes graphical user-interface**GUI**, demonstrates the creation of these resources.



Pods

	Name	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑
✓	cassandradb-7d878468d9-7gtb6	app: cassandradb pod-template-hash: 7d878468d9	minikube	Running	1	-	-	a month ago

1 – 1 of 1

Deployments

	Name	Labels	Pods	Created ↑	Images
✓	cassandradb	-	1 / 1	a month ago	cassandra

Services

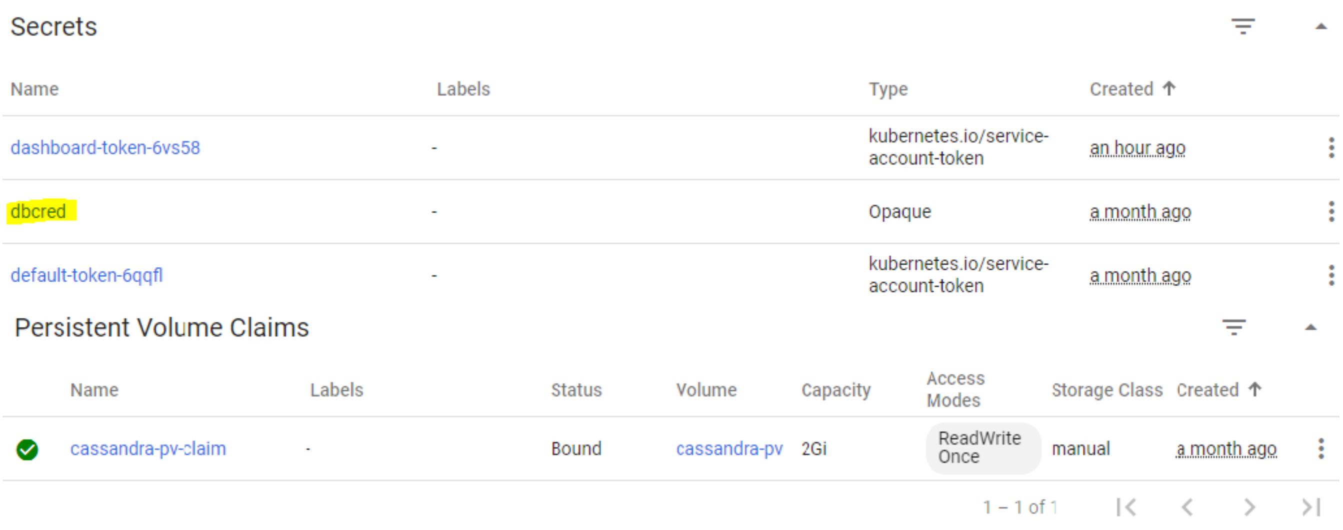
	Name	Labels	Type	Cluster IP	Internal Endpoints	External Endpoints	Created ↑
✓	cassandra	-	NodePort	10.110.223.58	cassandra:9042 TCP cassandra:3010	-	a month ago

Figure 5.1: The database server creation and deployment

After deploying the database server, we need to secure the access to it. This process is done by creating a Kubernetes secret which hold the database credentials. Furthermore, we need to allocate a specific memory to our data, this will be through a Kubernetes volume. Thanks to the flexibility of

Cassandra and Kubernetes, the memory allocated for the database can be scaled accordingly to the volume of the stored data.

In the figure below we demonstrate the creation of the Kubernetes secret and volume specific to the Cassandra database.



Secrets

Name	Labels	Type	Created ↑
dashboard-token-6vs58	-	kubernetes.io/service-account-token	an hour ago
dbcred	-	Opaque	a month ago
default-token-6qqfl	-	kubernetes.io/service-account-token	a month ago

Persistent Volume Claims

	Name	Labels	Status	Volume	Capacity	Access Modes	Storage Class	Created ↑
✓	cassandra-pv-claim	-	Bound	cassandra-pv	2Gi	ReadWrite Once	manual	a month ago

1 – 1 of 1

Figure 5.2: Configuration of the database credentials and storage on Kubernetes

5.4.2 Implementing the schema manager

The schema manager is a Kubernetes job responsible for populating the Cassandra database server with the keyspaces and schemas. The goal of the schema manager is to monitor automatically the change in the schemas within those keyspaces in a configurable fashion to increase reusability and to minimize maintenance of the schemas. The code behind the schema manager consists of a set of Bash scripts that connect to the database and load the schemas already written in the CQL language. The schema manager is deployed through helm charts. Helm charts are packaged Kubernetes YAML manifests that can be merged into a single package. This package is highly configurable. It also makes the deployment easier through running a simple helm install command.

5.4.3 Implementing the back end application

The back end application is responsible for creating the APIs for editing configuration data. The users can access these APIs in order to send configuration requests. The back end application will also be used to feed the microservices in the promotion management application with the configuration data.

5.4.3.1 Configuring the Scala application

The first step to create the back end application was to configure a Scala application using the interactive build tool sbt. In fact, sbt is responsible for downloading the libraries and loading the project settings. In the next table, we illustrate the main libraries used in the back end application.

Library	Purpose
Akka-HTTP	Building the web server.
Akka Actors	Control the state and behaviour of the server
Akka streams	Process the requests data streams
Akka-spary	Marshal data models to the JSON format used in HTTP requests and the other way around(Unmarshalling)
Cassandra driver	Create connection to Cassandra database and send queries.
JWT-core	Decode JWT for the authorization check.
Scala test	Unit testing for the APIs.
Cognira's Async HTTP	Create asynchronous HTTP requests
Cognira's Async CQL	Create asynchronous CQL requests

Table 5.4: The sbt libraries used to build the back end application

5.4.3.2 Building the web server

Now we move on to the second step which is building the web server. This latter ensures the communication with the external actors of our application. The web server is built using Akka-HTTP in top of Akka actors and Akka streams. Akka actors is responsible for the thread management in the application as it controls the application state and behaviour. As for Akka streams, it takes care of processing the asynchronous messages so we avoid worrying about setting an explicit state for our Akka actors. In The next figure, we demonstrate the basic flow that we created for an HTTP request using the Akka libraries.

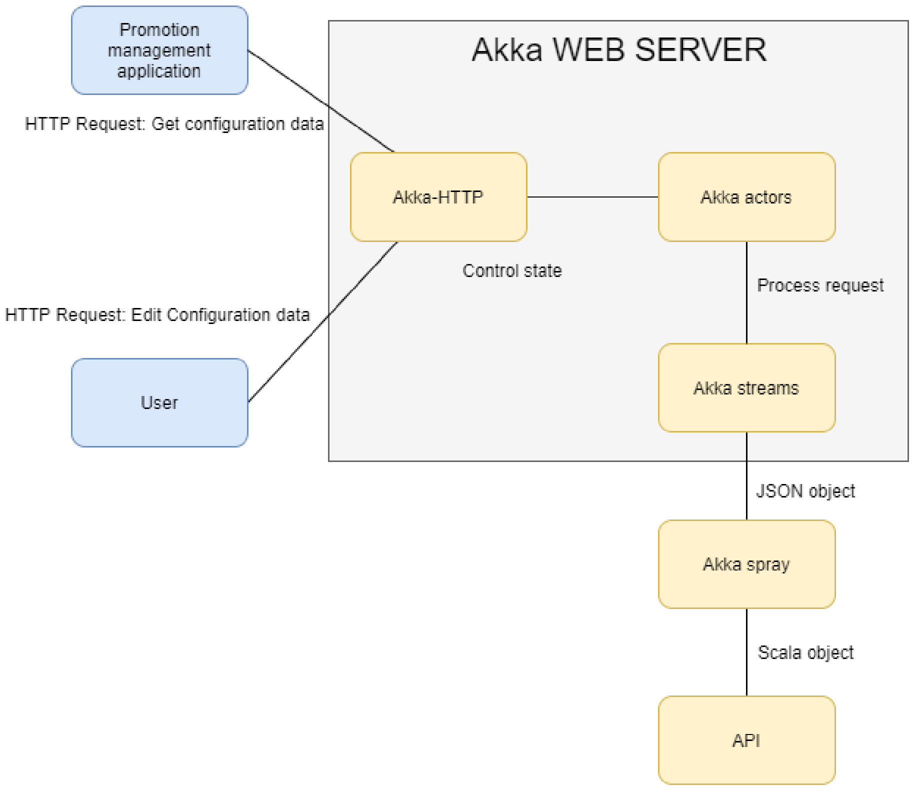
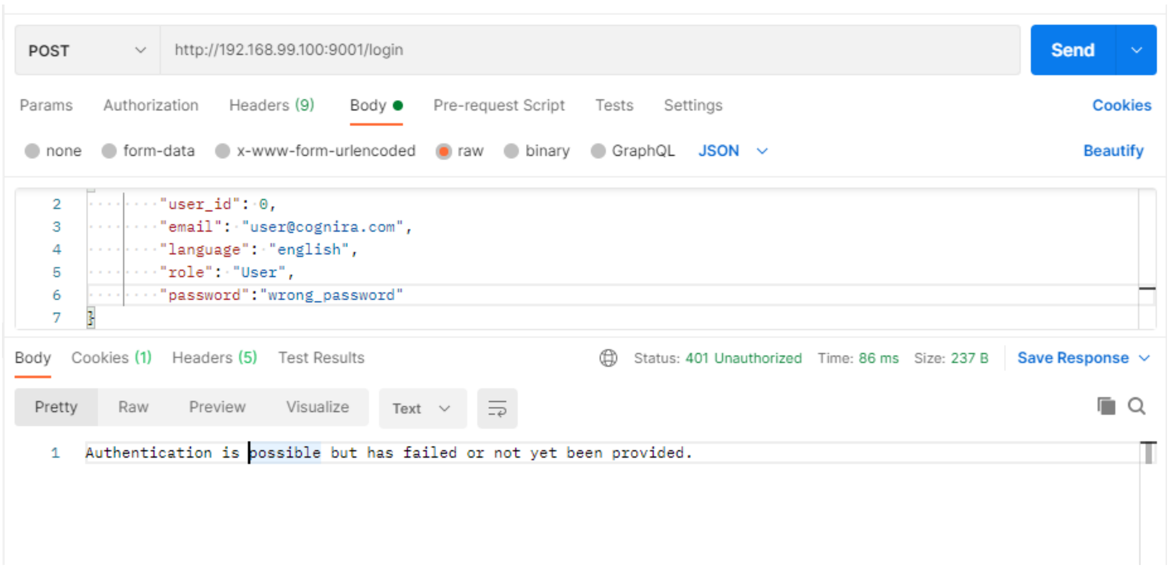


Figure 5.3: HTTP request flow in the back end application

5.4.3.3 Implementing the JWT authorization check

After building the web server, the next step is securing our application with the JWT. We implemented the authorization request in Cognira's security microservice and we linked this microservice with the back end application. Meaning that each request directed to our application is first verified by the security microservcice. In the upcoming figure we use postman to demonstrate an invalid authorization request. This type of request results in an HTTP 401 response(Unauthorized).



```
http://192.168.99.100:9001/login
```

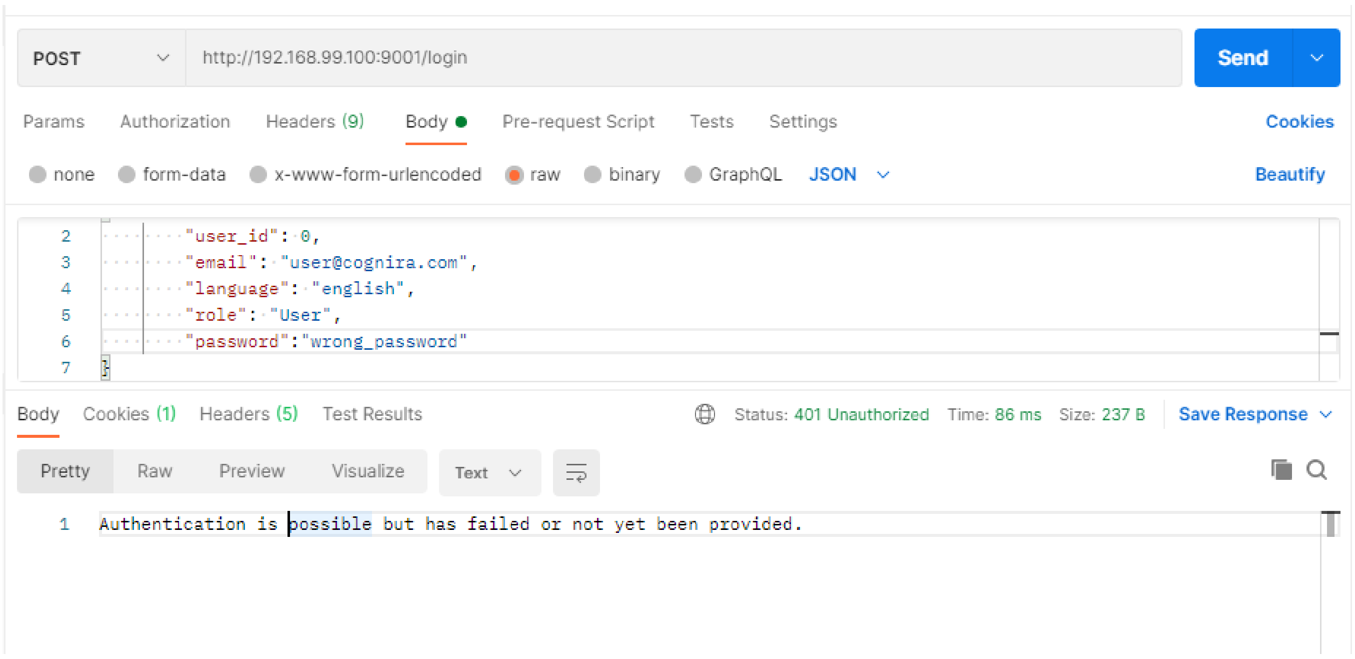
```
        "user_id": 0,
        "email": "user@cognira.com",
        "language": "english",
        "role": "User",
        "password":"wrong_password"
}
```

Status: 401 Unauthorized Time: 86 ms Size: 237 B

```
Authentication is possible but has failed or not yet been provided.
```

Figure 5.4: Invalid authorization request

In the figure 5.6 we demonstrate a valid authorization request. In this case, the server sends back an HTTP 200 response(Success). It also provides a cookie holding the JWT encrypted value. This cookie can be later on used to access the application.



http://192.168.99.100:9001/login

```
    "user_id": 0,
    "email": "user@cognira.com",
    "language": "english",
    "role": "User",
    "password":"wrong_password"
}
```

Status: 401 Unauthorized Time: 86 ms Size: 237 B

```
Authentication is possible but has failed or not yet been provided.
```

Figure 5.5: Valid authorization request

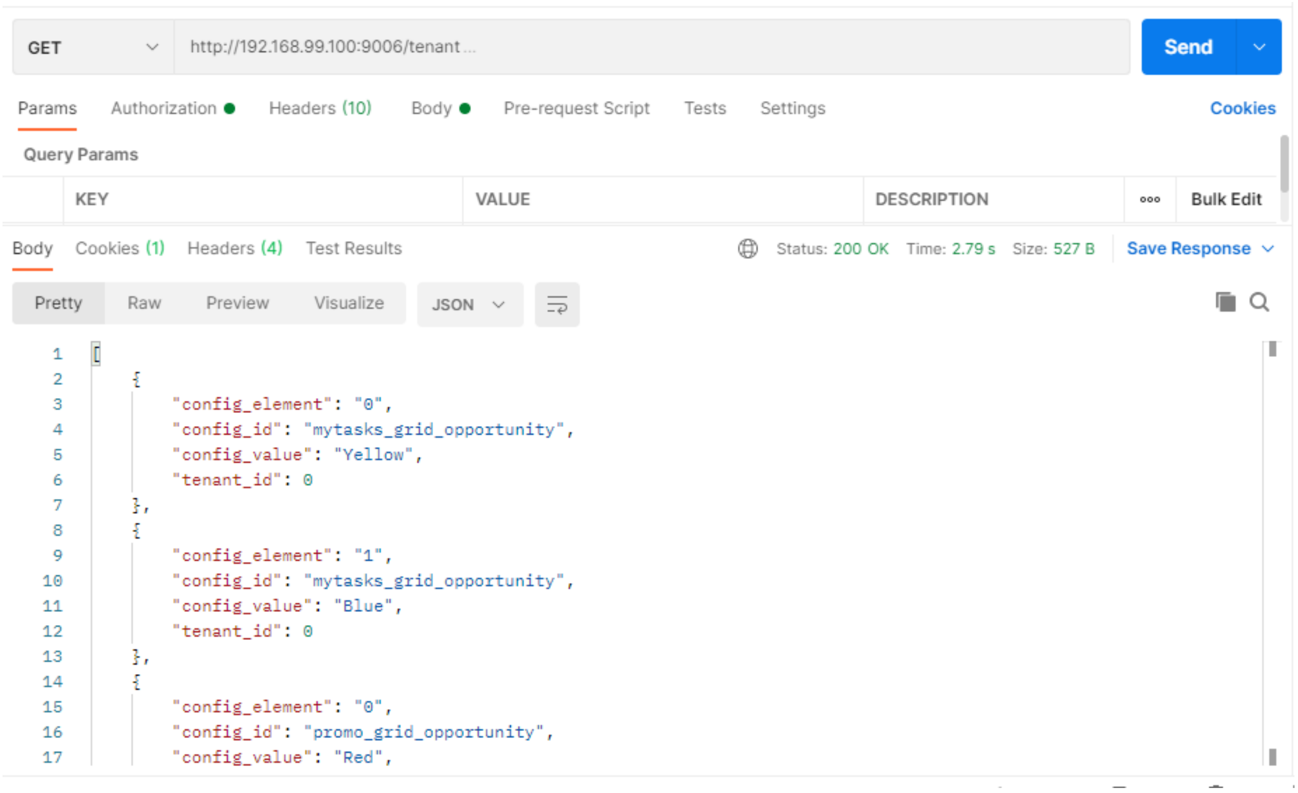
5.4.3.4 Developing the configuration handling APIs

After the creation of a secure and accessible back end application, the next step is creating the set of APIs that the external actors will be interacting with.

These APIs are the explicit implementation of the logic behind our class diagram. Actually, the APIs offer the capability to either fetch, insert, update or delete configuration data.

In the figure 5.7 we demonstrate an HTTP GET request that fetches the configuration data for the tenant by the tenant id. With the tenant being an authorized retailer client in our business case.

In the case of a valid request, the server sends back all of the tenant configuration data for that specific tenant.



```
[
    {
        "config_element": "0",
        "config_id": "mytasks_grid_opportunity",
        "config_value": "Yellow",
        "tenant_id": 0
    },
    {
        "config_element": "1",
        "config_id": "mytasks_grid_opportunity",
        "config_value": "Blue",
        "tenant_id": 0
    },
    {
        "config_element": "0",
        "config_id": "promo_grid_opportunity",
        "config_value": "Red",
```

Figure 5.6: an example of an HTTP request to the back end server

5.4.3.5 Testing the APIs

Following the APIs development, we created testing scenarios and implemented them as unit tests for each API, using the scala test library. In this step, we used the embedded Cassandra server library to create a mock-up testing database. Furthermore, we created a mock-up JWT token to emulate the authorization check. The table 5.5 contains the test scenarios and the request content for each scenario.

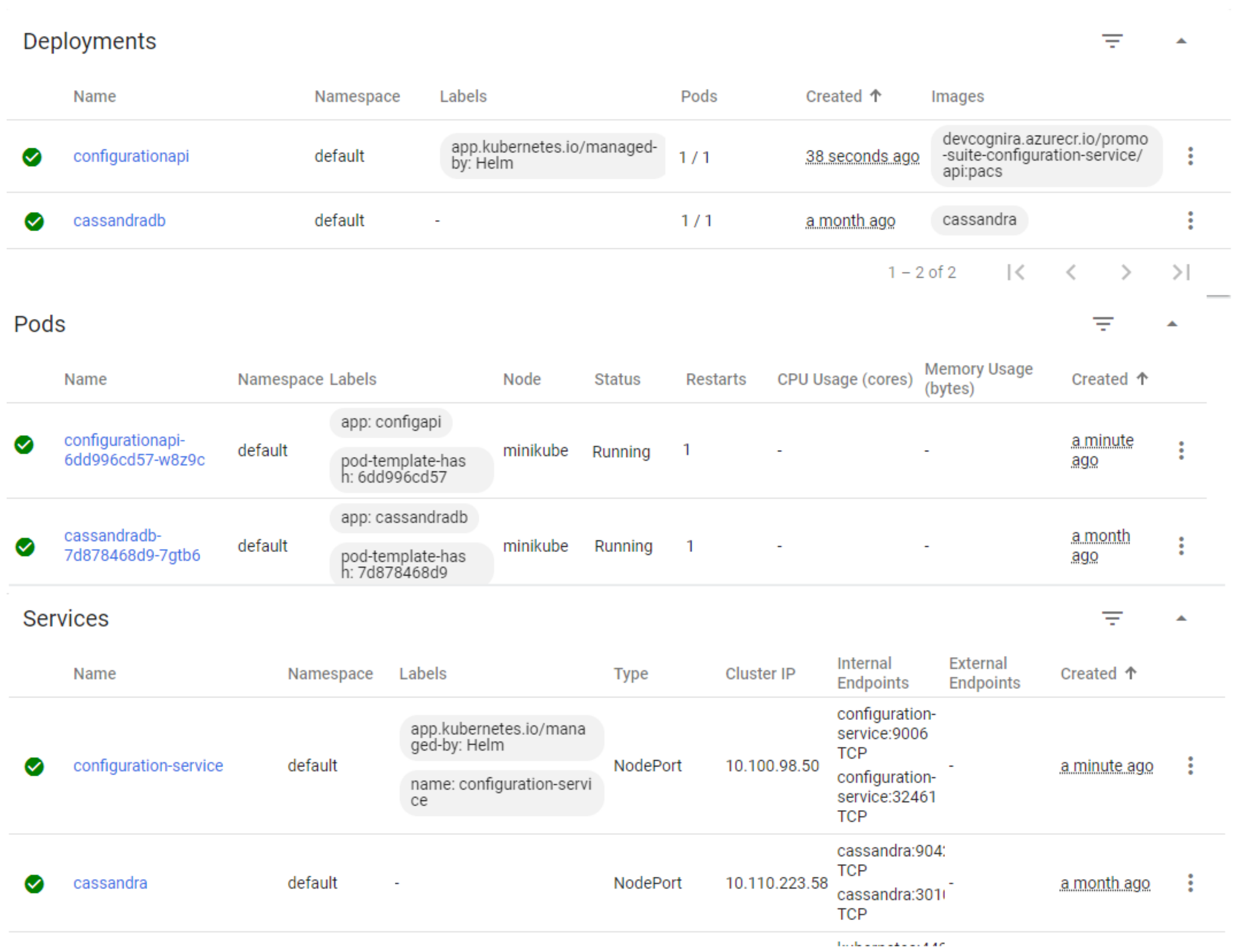
Testing scenario	**Request type/content**	**Response**
The user/microservice provides an invalid Json Web token.	Wrong credentials.	Unauthorized (HTTP 401)
The user/microservice fetches the configuration data by tenant.	Cookie + GET request	Success (HTTP 200) with configuration data.
The user inserts a valid configuration row	Cookie + POST request + valid request body	Success (HTTP 200)
The user inserts an invalid configuration row.	Cookie + POST request + invalid request body	Internal server error (HTTP 500) with the correct request format.

The user updates an existent configuration row.	Cookie + PUT request + request body	Success (HTTP 200)
The user updates a non existent configuration row.	Cookie + PUT request + request body	Success (HTTP 200) with message specifying that a new row is created.
The user deletes an existent row	Cookie + DELETE request + request body	Success (HTTP 200)
The user deletes an non-existent row	Cookie + DELETE request + request body	Internal Server Error (HTTP 500) with a message specifying that the row does not exist

Table 5.5: Implemented unit tests and expected responses

5.4.3.6 Packaging and deploying the back end application

The last part in the implementation of the back end application is the packaging and deployment. In fact, after creating the application, the first step is to package the application code and libraries into a JAR file independent from the development environment. This step was performed using the sbt package library.
After the packaging, we transform the JAR file to a deployable Docker container. This is made possible using docker-compose which is a tool for defining and running multi-container Docker applications. When creating the container, we push the created Docker image to the Azure container registry.
When deploying our back end application, we configured helm charts that automatically pull the Docker image from the container registry, using a Kubernetes secret to hold the Azure credentials. Once deployed, a pod containing our back end application is created. This pod generates a service and a deployment which is exactly the case with the database server. Noting that all of the steps for packaging and deploying the application are merged into an pipeline using Bash script.
The figure 5.8 shows the back end application deployed along with the database server in the Kubernetes graphical interface.



Deployments

	Name	Namespace	Labels	Pods	Created ↑	Images
✓	configurationapi	default	app.kubernetes.io/managed-by: Helm	1 / 1	38 seconds ago	devcognira.azurecr.io/promo-suite-configuration-service/api:pacs
✓	cassandradb	default	-	1 / 1	a month ago	cassandra

1 – 2 of 2

Pods

	Name	Namespace	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑
✓	configurationapi-6dd996cd57-w8z9c	default	app: configapi pod-template-hash: 6dd996cd57	minikube	Running	1	-	-	a minute ago
✓	cassandradb-7d878468d9-7gtb6	default	app: cassandradb pod-template-hash: 7d878468d9	minikube	Running	1	-	-	a month ago

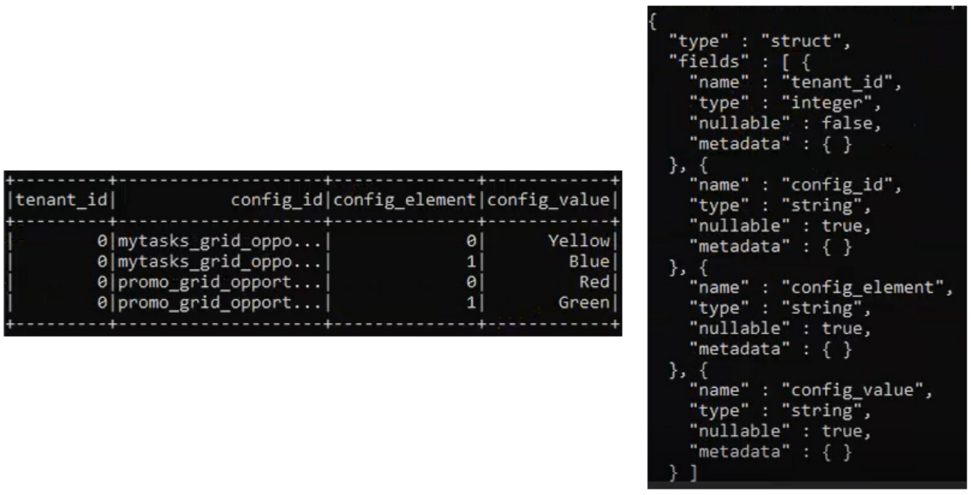
Services

	Name	Namespace	Labels	Type	Cluster IP	Internal Endpoints	External Endpoints	Created ↑
✓	configuration-service	default	app.kubernetes.io/managed-by: Helm name: configuration-service	NodePort	10.100.98.50	configuration-service:9006 TCP configuration-service:32461 TCP	-	a minute ago
✓	cassandra	default	-	NodePort	10.110.223.58	cassandra:904: TCP cassandra:301(TCP	-	a month ago

Figure 5.7: Deployment of the back end application

5.4.4 Implementing the Spark Cronjob

The goal of the Spark CronJob is to fetch the configuration files, apply versioning on the files and exporting them to the Azure data lake. In order to do so, the first thing was to create a Spark application using Scala and the Spark libraries. With the help of the Spark Cassandra connector, we were able to create a script that fetches all of the configuration files. After that, we used the Spark dataframes API to transform the data type of the configuration files to parquet. The Apache parquet file format is an open source flat columnar storage[33] format designed to be more lightweight and performant than other similar file formats such as TSV or CSV. Parquet is the file format used to store data in an Azure data lake storage. We then added timestamps to these files and exported them to the Azure data lake. In the figure 5.9, we display a configuration file as a dataframe and the structure of this file when it is converted to parquet format.

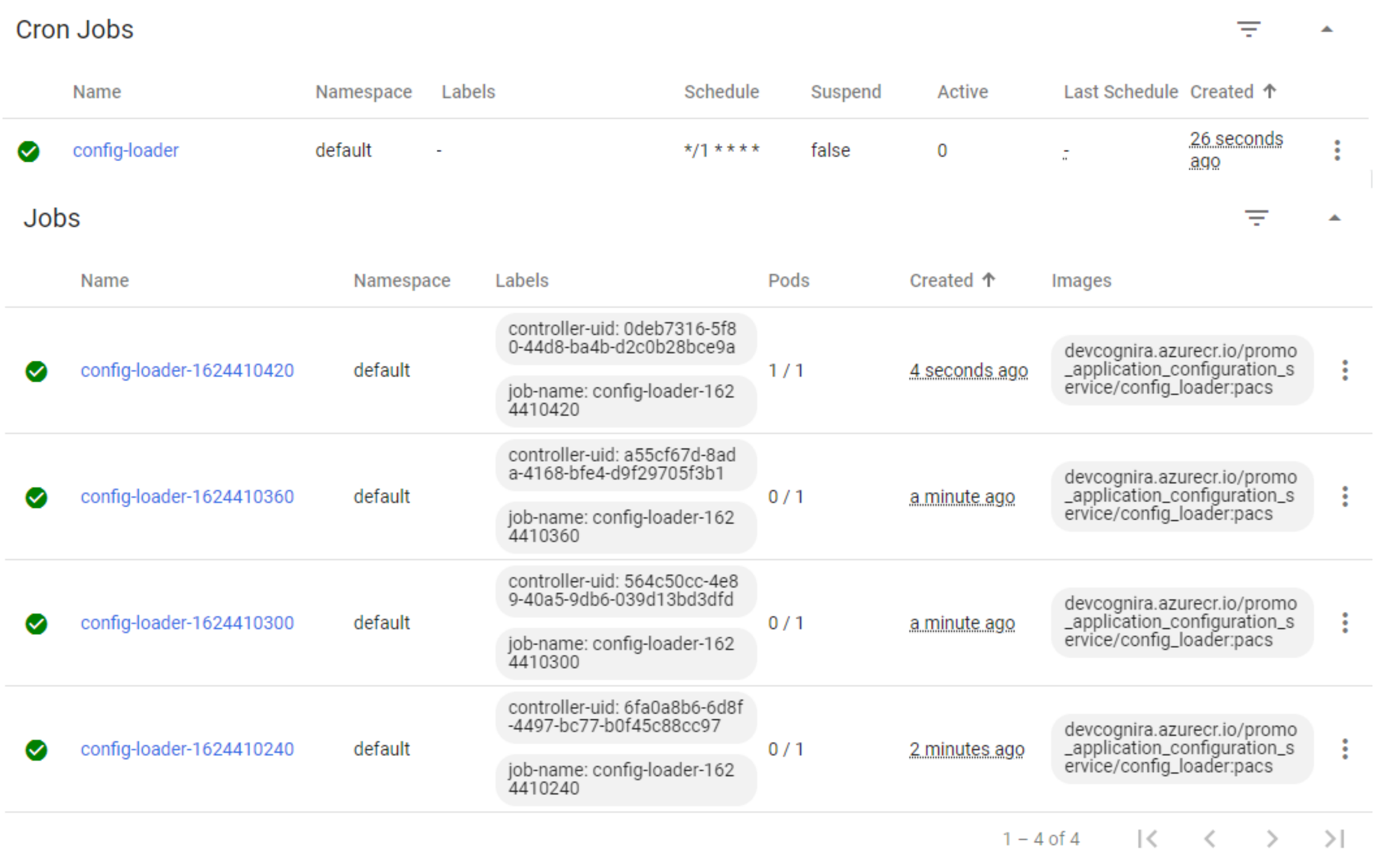


```
+---------+--------------------+--------------+------------+
|tenant_id|           config_id|config_element|config_value|
+---------+--------------------+--------------+------------+
|        0|mytasks_grid_oppo...|             0|      Yellow|
|        0|mytasks_grid_oppo...|             1|        Blue|
|        0|promo_grid_opport...|             0|         Red|
|        0|promo_grid_opport...|             1|       Green|
+---------+--------------------+--------------+------------+
```

```
{
  "type" : "struct",
  "fields" : [ {
    "name" : "tenant_id",
    "type" : "integer",
    "nullable" : false,
    "metadata" : { }
  }, {
    "name" : "config_id",
    "type" : "string",
    "nullable" : true,
    "metadata" : { }
  }, {
    "name" : "config_element",
    "type" : "string",
    "nullable" : true,
    "metadata" : { }
  }, {
    "name" : "config_value",
    "type" : "string",
    "nullable" : true,
    "metadata" : { }
  } ]
```

Figure 5.8: A dataframe configuration file(left) and its representation in parquet (right)

To deploy the CronJob, we configure a YAML file containing a timer. In fact, we need it to be automatically run each twenty-four hours to keep track of the configuration files version. In the figure 5.10, we demonstrate how once the CronJob is deployed, it automatically generates scheduled Spark Jobs. In this example We set the timer at 1 minute for demonstration purposes.



Cron Jobs

	Name	Namespace	Labels	Schedule	Suspend	Active	Last Schedule	Created ↑
✓	config-loader	default	-	*/1 * * * *	false	0	-	26 seconds ago

Jobs

	Name	Namespace	Labels	Pods	Created ↑	Images
✓	config-loader-1624410420	default	controller-uid: 0deb7316-5f80-44d8-ba4b-d2c0b28bce9a; job-name: config-loader-1624410420	1 / 1	4 seconds ago	devcognira.azurecr.io/promo_application_configuration_service/config_loader:pacs
✓	config-loader-1624410360	default	controller-uid: a55cf67d-8ada-4168-bfe4-d9f29705f3b1; job-name: config-loader-1624410360	0 / 1	a minute ago	devcognira.azurecr.io/promo_application_configuration_service/config_loader:pacs
✓	config-loader-1624410300	default	controller-uid: 564c50cc-4e89-40a5-9db6-039d13bd3dfd; job-name: config-loader-1624410300	0 / 1	a minute ago	devcognira.azurecr.io/promo_application_configuration_service/config_loader:pacs
✓	config-loader-1624410240	default	controller-uid: 6fa0a8b6-6d8f-4497-bc77-b0f45c88cc97; job-name: config-loader-1624410240	0 / 1	2 minutes ago	devcognira.azurecr.io/promo_application_configuration_service/config_loader:pacs

1 – 4 of 4

Figure 5.9: Deployment of the Spark CronJob

Conclusion

Throughout this final chapter, we elaborated on the various tools and technologies used during this project. After that, we dug deep into the implementation steps of the promotion configuration microservice.

CONCLUSION AND PERSPECTIVES

Our end of studies project, called the Retail Promotion Configuration Service, was conducted at Cognira during four months. In this internship, we adopted the SCRUM methodology, we first began by defining the scope of the project, including the project charter, the sponsor, the problem that we aimed to solve and the proposed solution.. We then focused on the specifying our requirements and specifications, which paved the way to a design study. The design study branches out to a global and detailed design approaches. In the global design, we illustrated the components of our microservice from a physical and logical point of view, then we moved on to the deployment model used in this project. Moreover, in the detailed design, we gave an in-depth study of the project features using UML diagrams. Lastly, we implemented each part of the promotion configuration service.

It is safe to say that we were able to tackle all the encountered challenges. We were able to provide our retailer client with a straightforward and effortless configuration experience. Moreover, we created the possibility to keep track on the configuration of the promotion management application provided by Cognira to the retailer clients. This created opportunities to revert the configuration for the client at any time as well as using the same set of configurations for a different client. This process also allows the Cognira team to save a huge amount of time and resources. In addition to that, we leveraged the advantages of the microservices technology. Then, we used these advantages to tackle one of its main weaknesses which is the high need for configuration. Due to this fact, when integrated to the promotion management application, our microservice contributed in optimizing the response time of the application as it relieved the pressure on the server. Since it plays the role of an intermediary agent between all the requests for configuration and the database server. Besides, we reduced the number of conflicts in the application since we centralized the configuration process in a single microservice. Actually, this project answers the needs for constant scalability in Cognira's solution. The more the Cognira software scales in terms of features and data, the more number of requests for configuration data elevates. In this context, our project plays an instrumental role in the long-term vision of this company.

Still, there is room for improvement. As we can transform the retail promotion configuration service to a full SaaS(Software as a service) solution. In the future, we will build dashboards and user interfaces to allow the retailers to configure their application and to revert the configurations without the need to directly interact with our microservice.

Many ideas about the features of this project revolve around optimizing the specific software solution of Cognira and providing the retail client with a better software in an optimized development duration. But what if we look at the solution provided during this internship from a global point of view ? In fact, several other methods can be used to optimize configuration handling in microservices. According to the RedHat corporation, one of the approaches that can be used to optimize configuration for multi-tenancy and multi-product software is not to create a microservice for that purpose. In fact, RedHat suggests to isolate the data for each tenant and set of configuration by automatically generating new databases in the architecture for each use-case**multitenant**. This will certainly give a better performance and response time than the retail configuration service that we built but it will also consume more physical resources and memory compared to the solution proposed. In the future, and since Cognira's data and set of clients are always scaling up, the RedHat approach can be merged with the solution proposed so we can take advantage of the two approaches. Another concurrent methodology to our work is to use MicroProfile Config APIs. This approach makes use of contexts and dependency injection (CDI) to inject configuration data directly into the retail application. This injection is done without requiring the user code to retrieve them. The injected values are static since they are set only at application startup. Using this approach will be completely different from the approach that we chosen to work with since in the configuration service we built a set of APIs that the user can access only whenever a specific configuration information is needed. The MicroProfile Config approach leads to a much slower application startup time since all the configuration is going to be fetched at the startup of each microservice. However, it almost eliminates the need for configuration requests within the application. Except for the situation when the configuration data is edited.
Finally, we can conclude by stating that our solution serves its purpose as it gives a better and a far

more efficient approach to handling configuration data in the Cognira solution. But still it has the ability to improve by taking inspiration from other approaches and to be bench-marked against other solutions in the future.

BIBLIOGRAPHY & NETOGRAPHY

[1] *What is a project charter in project management*, Accessed: March, 9, 2021. [Online]. Available: `https://www.wrike.com/project-management-guide/faq/what-is-a-project-charter-in-project-management/`.

[2] *Cognira recognized as one of the fastest growing private companies in the us*, Accessed: March, 11, 2021. [Online]. Available: `https://cognira.com/cognira-recognized-as-one-of-the-fastest-growing-privately-held-companies-in-the-u-s/`.

[3] *Introducing the 5,000 fastest-growing private companies in america*, Accessed: March, 11, 2021. [Online]. Available: `https://www.inc.com/inc5000/2019`.

[4] *Promotion management software solutions - cognira*, Accessed: March, 13, 2021. [Online]. Available: `https://cognira.com/`.

[5] *What are microservices ?*, Accessed: March, 15, 2021. [Online]. Available: `https://microservices.io/`.

[6] *What is scrum ?*, Accessed: March, 26, 2021. [Online]. Available: `https://www.edureka.co/blog/what-is-scrum/`.

[7] *Atlassian, use pull requests for code review*, Accessed: March, 30, 2021. [Online]. Available: `https://support.atlassian.com/bitbucket-cloud/docs/use-pull-requests-for-code-review/`.

[8] *When you should use jwt ?*, Accessed: May, 8, 2021. [Online]. Available: `https://jwt.io/introduction`.

[9] C. Larman, *Applying UML and patterns: an introduction to object oriented analysis and design and interative development.* Pearson Education India, 2012.

[10] *Chapter 7 system sequence diagrams*, Accessed: April, 2, 2021. [Online]. Available: `https://homepages.fhv.at/thjo/lecturenotes/sysan/system-sequence-diagrams.html`.

[11] *Redhat, what is a rest api?*, Accessed: April, 11, 2021. [Online]. Available: `https://www.redhat.com/en/topics/api/what-is-a-rest-api`.

[12] *Amazon web services, data lake analytics - what is a data lake ?*, Accessed: April, 11, 2021. [Online]. Available: `https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/`.

[13] *Microservices advantages and disadvantages: everything you need to know*, Accessed: April, 8, 2021. [Online]. Available: `https://solace.com/blog/microservices-advantages-and-disadvantages/`.

[14] *Docker documentation, what is a container ?*, Accessed: March, 15, 2021. [Online]. Available: `https://www.docker.com/resources/what-container`.

[15] *What is containerization.* [Online]. Available: `https://hackernoon.com/what-is-containerization-83ae53a709a6`.

[16] *Docker engine overview*, Accessed: April, 12, 2021. [Online]. Available: `https://docs.docker.com/engine/`.

[17] *Docker*, Accessed: April, 12, 2021. [Online]. Available: `https://hub.docker.com/_/mariadb`.

[18] J. Cao, J. Wei, and Y. Qin, "Research and application of the four-tier architecture", in *2013 the International Conference on Education Technology and Information System (ICETIS 2013)*, Atlantis Press, 2013.

[19] *Ibm, service oriented architecture*, Accessed: April, 18, 2021. [Online]. Available: `https://www.ibm.com/cloud/learn/soa`.

[20] R. Peters, "Cron", *Expert Shell Scripting*, pp. 81–85, 2009.

[21] *Kubernetes documentation: cronjobs*, Accessed: April, 24, 2021. [Online]. Available: `https://spark.apache.org/`.

[22] *Kubernetes*, Accessed: April, 20, 2021. [Online]. Available: `https://kubernetes.io/`.

[23] *What is kubernetes? an introduction to the wildly popular container orchestration platform*, Accessed: April, 23, 2021. [Online]. Available: `https://newrelic.com/blog/how-to-relic/what-is-kubernetes`.

[24] *Kubernetes: core concepts*, Accessed: April, 23, 2021. [Online]. Available: `https://www.yld.io/blog/kubernetes-core-concepts/`.

[25] *Kubernetes documentation: jobs*, Accessed: April, 23, 2021. [Online]. Available: `https://kubernetes.io/docs/concepts/workloads/controllers/job/`.

[26] *Kubernetes documentation: cronjobs*, Accessed: April, 23, 2021. [Online]. Available: `https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/`.

[27] *Ibm, sample files; json and yaml*, Accessed: June, 20, 2021. [Online]. Available: `https://www.ibm.com/docs/en/cmwo/4.3.0.0?topic=reference-sample-files-yaml-json`.

[28] *What is a transmission control protocol tcp/ip model ?*, Accessed: May, 8, 2021. [Online]. Available: `https://www.fortinet.com/resources/cyberglossary/tcp-ip`.

[29] *The cassandra query language (cql)*, Accessed: May, 8, 2021. [Online]. Available: `https://cassandra.apache.org/doc/latest/cql/`.

[30] A. Alexander, *Scala Cookbook: Recipes for Object-Oriented and Functional Programming.* " O'Reilly Media, Inc.", 2013.

[31] *Apache cassandra guide: data modeling*, Accessed: June, 5, 2021. [Online]. Available: `https://hackernoon.com/apache-cassandra-guide-data-modeling-i3g3ufi`.

[32] *Apache spark - core programming*, Accessed: June, 5, 2021. [Online]. Available: `https://www.tutorialspoint.com/apache_spark/apache_spark_core_programming.htm#:~:text=Spark%20Core%20is%20the%20base,of%20data%20partitioned%20across%20machines`.

[33] *Apache parquet*, Accessed: June, 23, 2021. [Online]. Available: `https://parquet.apache.org/`.

Abstract

This work is achieved in the context of a third year graduation project at the Higher School of Communication of Tunis.

This report covers the design, implementation and testing of a project that consists of building a microservice for a retail promotion management application. This microservice helps retailers to have an optimized configuration experience. To achieve this, the project relies mainly on creating a pipeline for automating the configuration of the application and encapsulating the configuration in a standalone application. Moreover, a part of this project is to create versioned configuration sets that can be logged and used for different clients and for similar business purposes.

Keywords : Retail, Microservices, Software Development, Cloud, Docker, Kubernetes, Big Data, Spark

Résumé

Ce travail est réalisé dans le cadre d'un projet de fin d'études de troisième année à l'Ecole Supérieure de Communication de Tunis.

Ce rapport couvre la conception, la mise en œuvre et le test d'un projet qui consiste à construire un microservice pour une application de gestion de promotion de vente au détail. Ce microservice aide les détaillants à avoir une expérience de configuration optimisée. Pour y parvenir, le projet repose principalement sur la création d'un pipeline permettant d'automatiser la configuration de l'application et d'encapsuler la configuration dans une application autonome. De plus, une partie de ce projet consiste à créer des ensembles de configuration versionnés qui peuvent être enregistrés et utilisés pour différents clients et à des fins commerciales similaires.

Keywords : Vente au détail, Microservices, Développement de logiciels, Cloud, Docker, Kubernetes, Données massives, Spark