

Educational cycle for engineers in Telecommunications

Major :

Network, Cloud, and IA (NCA)

END-OF-STUDIES INTERNSHIP'S REPORT

Topic :

**« Design and Development of a Configurable Calculation
Engine for metrics evaluation about retail promotions »**

Conducted by :

Mohamed Bilel Besbes

Supervisors :

Mr. Mohamed-Bécha Kaâniche (Sup'Com)

Mr. Youssef Fehri (Cognira Tunisia)

Mr. Housseem Nefoussi (Cognira Tunisia)

Work proposed and carried out in collaboration with :

Cognira Tunisia



Academic year : 2021/2022

Dedication

For my parents, who were always by my side, who gave me endless support, as well as endless prayers, I would like to thank you for everything that you did for me, and I do hope that you are proud of me, because that's what matters to me more than anything.

To my sister who made sure to be supportive in every step in my life, I would like to thank her for her continuous support and unconditional love.

To the souls of my paternal grandparents who died a long ago, I wish that you are proud me, may you rest in peace.

To the souls of my maternal grandparents who passed away recently, I want to thank you for being part of making me the man who I am today, may you rest in peace.

To everyone who supported me, endless love for doing so, and for everyone who held me back, thank you for letting me reveal my true potential to succeed inspite of itself.

Acknowledgements

Special thanks goes to Mr. Mohamed-Bécha Kaâniche, my academic supervisor, for his helpful support and oversight, as well as his wise remarks, suggestions, patience and mentorship throughout the synchronisation period of the internship. Also I adress special thanks to Mr. Youssef Fehri and Mr. Housseem Neffousi my professional supervisors, for their help that they have given me throughout the internship, for handing me the keys to the professional world, and for pushing me forward which lead me to reach my full potentials.

Thanks to all of my professors throughout my schooling. And my thanks goes to Cognira Tunisia team for welcoming me as a member of the team and for giving me a huge added value in such a short notice of four months.

Abstract

The objective of this work is to design and build a calculation engine for retail companies. The developed application delivers indicators like return on investment and risk margin about promotions. Such indicators help retailers to choose which promotion to go with in order to maximize profits.

The calculation engine does the computations through well-defined equations with precedence relationship between them. Each equation is associated with an indicator. The proposed solution needs to be generic, so it can compute any new formula. It also needs to be configurable, meaning that any aspect about the given promotion can be taken into account easily. The engine should deal with large volumes of data like prices and costs of huge amount of products. This is why Spark on Scala is used for implementation.

For production environment, the engine need to be containerized. Afterwards, it needs to be deployed using a container orchestration tool. That way, the solution is easily deployable on any infrastructure.

Keywords :

Calculation Engine (CE), Input Injection, Formula Parsing, Scala, Spark, Docker, Kubernetes.

Contents

Dedication	ii
Acknowledgements	iii
Abstract	iv
List of Figures	ix
List of Tables	x
List of Abbreviations	xi
General Introduction	1
1 Scope Statement	3
Introduction	3
1.1 The project charter	3
1.1.1 Project context	3
1.1.2 Project description	4
1.1.3 Project roles	4
1.2 The project owner	4
1.2.1 Presentation of Cognira	4
1.2.2 Field of expertise and solutions	4
1.2.2.1 Cognira's AI Promotion Solution	4
1.2.2.2 Forecast Service	5
1.2.2.3 Assortment and Allocation Decisions Optimization	5
1.3 Study of the existing	5
1.4 Problem statement	5
1.5 Project goals and objectives	6
1.6 Project deliverables	6
1.7 Project non-objectives	7
1.8 Project management plan	7
1.8.1 scrum methodology definition	7
1.8.2 Scrum roles	8

1.8.2.1	Scrum master	8
1.8.2.2	Development team	8
1.8.2.3	Product owner	8
1.8.3	scrum meetings	9
1.8.3.1	Sprint review meeting	9
1.8.3.2	Daily standup meeting	9
1.8.3.3	Code review process	9
1.9	Project plan	10
1.9.1	Project realization overview	10
1.9.2	Project progress chronogram	11
	Conclusion	13
2	Requirements analysis and specifications	14
	Introduction	14
2.1	Requirements analysis	14
2.1.1	Functional requirements	14
2.1.1.1	Generic calculation	14
2.1.1.2	Configurability	14
2.1.1.3	Replication handling	15
2.1.2	Non-functional requirements	15
2.1.2.1	Scalability	15
2.1.2.2	Performance	15
2.1.2.3	Reusability	15
2.1.3	Technical requirements	15
2.1.3.1	Testing	15
2.1.3.2	Containerization	15
2.1.3.3	Deployment	16
2.2	Requirements specification	16
2.2.1	Modeling language	16
2.2.2	Actor identification	16
2.2.3	Use case diagram	16
2.2.4	Use case textual description	17
2.2.4.1	Use case: provide products and promotions files	17
2.2.4.2	Use case: perform calculations	18
2.2.4.3	Use case: transform input into structured data	19
2.2.4.4	Use case: evaluate formulas	20
2.2.4.5	Use case: write output files	21
2.2.5	Activity diagram	22
	Conclusion	24

3	Design of the calculation engine	25
	Introduction	25
3.1	Retro-engineering the online engine	25
3.1.1	Definition of parse tree	25
3.1.2	Cognira's indicators' equations	26
3.1.2.1	indicator	26
3.1.2.2	dimensions	26
3.1.2.3	context	27
3.1.2.4	indicator to parse tree	28
3.1.2.5	Execution dependency graph	29
3.1.3	Replication	30
3.1.4	Computing an indicator's value	30
3.1.5	Calc Engine workflow	31
3.2	Object-oriented conception	32
3.2.1	Class diagram	32
3.2.2	Class diagram description	34
3.3	Deployment diagram	35
3.4	Conclusion	37
4	Project implementation and testing	38
	Introduction	38
4.1	The development environment	38
4.1.1	Hardware environment	38
4.1.2	Software environment	39
4.2	Technical choices	40
4.2.1	Programming Language: Scala	40
4.2.2	Data processing engine: Apache Spark	40
4.2.3	Container runtime: Docker	40
4.2.4	Container orchestration: Kubernetes	41
4.3	Implementation of the calculation engine	41
4.3.1	Input of the calculation engine	41
4.3.1.1	Formulas files	41
4.3.1.2	Prices and costs file	42
4.3.1.3	Promotions file	43
4.3.2	Used libraries	44
4.3.3	Output of the calculation engine	45
4.3.4	Packaging the calculation engine	45
4.3.5	Deploying the calculation engine	45
4.4	Testing the calculation engine	47
4.4.1	Unit testing	47
4.4.2	SonarQube analysis	49

4.5	Additional work	50
4.5.1	CI/CD pipeline	50
4.5.1.1	Definition	50
4.5.1.2	Added value	50
4.5.1.3	CI/CD pipeline implementation	51
4.5.2	Monitoring dashboard	52
4.5.2.1	Definition	52
4.5.2.2	Added value	52
4.5.2.3	Monitoring implementation	52
4.6	Validation	53
4.7	conclusion	53
General conclusion		54
Bibliography		56

List of Figures

1.1	Scrum methodology cycle [11]	8
1.2	Code review process[17]	10
2.1	Use case diagram	17
2.2	Activity diagram	23
3.1	Parse tree example	26
3.2	Dimension hierarchy configurations	27
3.3	Cognira's parse tree example	28
3.4	Part of parse tree as a string JSON example	29
3.5	Execution dependency graph example	30
3.6	Replication process demonstration	30
3.7	Demonstration of indicator computation process	31
3.8	Sequence diagram of the existing calculation engine	32
3.9	Class diagram	33
3.10	Deployment diagram	36
3.11	Deployment on Kubernetes	37
4.1	Formulas file example	42
4.2	Prices and costs file example	43
4.3	JSON promotion string	44
4.4	Indicators on the promotion dimension	45
4.5	Indicators on the promotion condition dimension	45
4.6	The job on minikube dashboard	46
4.7	PV and PVC on minikube dashboard	47
4.8	Coverage percentage of each folder	48
4.9	Scoverage coverage report	49
4.10	SonarQube report	50
4.11	Pipeline on Jenkins user interface	51
4.12	Monitoring dashboard	52
4.13	Job completion on minikube	53

List of Tables

1.1	Gantt diagram	12
2.1	Use case: provide products and promotions files	18
2.2	Use case: perform calculations	19
2.3	Use case: transform input into structured data	20
2.4	Use case: evaluate formulas	21
2.5	Use case: write output files	22
3.1	Software environment	35
4.1	Hardware environment	38
4.2	Software environment	39
4.3	The sbt libraries used in the project	44
4.4	The unit tests	48
4.5	Pipeline stages	51

List of Abbreviations

API : **A**pplication **P**rogramming **I**nterface

UML : **U**nified **M**odeling **L**anguage

JSON : **J**ava**S**cript **O**bject **N**otation

CSV : **C**omma-**S**eperated **V**alues

DAG : **D**irect **A**cyclic **G**raph

IDE : **I**ntegrated **D**evelopment **E**nvironment

JVM : **J**ava **V**irtual **M**achine

JAR : **J**ava **A**Rchive

PV : Persistent Volume

PVC : Persistent Volume Claim

PoC : Proof of Concept

General Introduction

Calculation solutions have gained more and more popularity in the recent years. A main reason for this is that a lot of software systems rely on accurate computations of formulas. Another reason is the emergence of Business Intelligence and Big Data, especially during the last decade. Therefore, performing calculations is needed to be able to extract indicators about huge scale data. These calculation engines are used everywhere. For example, it can be used to calculate precisely the amount of money to put on the paycheck of every single employee in a company. This might seem easy but there are thousands of employees, where each has his own monthly history of days off, extra working hours, and performance bonuses. Therefore, the big amount of data used for the calculation could risk putting the performance of the application into question in terms of infrastructure usage and labour need.

Another example is the case of customer retail industry. In the customer retail world, promotions take part of the main strategies to enlarge a new customer segment, keep old customers happy, and maximize profit at the same time. A key operation then is to know which promotion configuration to provide for the customer, since it is dependent on a set of factors like the products to include, their cost, their price, the timing of the promotion, and other important factors.

Often, promotions are divided into two main types. The first one is taking an amount off or a percent off the price. Anyone can show up and benefit from the promotion with no conditions. The second type is a promotion that you get only if a certain requirement is fulfilled. For example, you can get the promotion only if you have a coupon, or you can get a free product with the condition of buying three others. Different promotion types don't necessarily share the same characteristics. Therefore, choosing how to evaluate a certain promotion type is important. Retailers want to choose a promotion configuration that generates the most profit. The way for the retailer to evaluate promotions is to use indicators such as return on investment, risk margin, and maximum profit. So the retailer needs a calculation tool that receives a set of parameters around a promotion and outputs a set of indicators about that promotion. These indicators are calculated through equations.

The problem is that such computations need to happen on periodic basis. This is because

every single day for example, the retailer acquires more and more data, which will change the result of the calculation every single time. This happens especially if the equations needed for computing the promotions' indicators take precedence over each other. And the data accumulation across time creates huge data volumes. This makes it difficult to do these computations in a cost-effective fashion.

Given this context, the aim of this project is to create an offline calculation engine that does the computation of formulas of a given promotion. Also, the system should handle the introduction of big loads of data as initial input while maintaining a low computational cost. Furthermore, the equations are used to build an ordering mechanism to organize their execution so the calculation of an indicator happens exactly one time. This is because we want to avoid calculation redundancy as a computed indicator can be used in the computations of other indicators.

This report consists of four different chapters. The first one concerns the scope statement which covers the project's scope, the project's owner, and the used methodology. The second chapter presents the description of the system functionality by stating technical, functional, and non-functional requirements. The third chapter exposes the design of the calculation engine in terms of global design and architecture of the constructed system. The fourth chapter covers a demonstration of the project implementation as well as a description of the used technical tools and why they were chosen, along with testing, additional work, and validation. Finally, a general conclusion highlights a wrap-up and some perspectives of what could be done to enhance the project.

Chapter 1

Scope Statement

Introduction

This chapter contains the project charter, the project owner description, the problem statement explanation with the project objectives, and the methodology followed to accomplish the work that was done.

1.1 The project charter

The project charter contains an overview of the scope statement [1]. In this section, the project context, description, and project roles are defined and well described.

1.1.1 Project context

Cognira's clients, who are retailers, rely on promotions for several reasons. For example, promotions are used to release a new product brand, to compete with other retailers in the same area, or to destock products that are about to expire. Promotions do come at a price and attaining profit is sometimes uncertain and even if profit will be made, it should be maximised. So relying on indicators of a given promotion, like maximum possible profit or loss risk percentage can help retailers to make a decision based on concrete indicators. In order to compute the latters, Cognira provides a solution named 'Calc Engine'. It takes data of prices and costs of different products at different store locations along with parameters about promotions and perform computations on them. Upon launching the Calc Engine product by Cognira, the clients' expectations were met through it that they decided to stick with it. As their data grew throughout accumulation, the current version wouldn't be efficient to use. This is because it wasn't built to handle such big volumes of data. Also, the need for the Calc Engine is changing. Cognira's clients were using it once in a while and on demand. But now, they use it often to do the calculations all over again. So they would like to run it on periodic basis in offline mode with as little user interaction as possible.

1.1.2 Project description

Calculation Engine is an application dedicated to calculate indicators around promotions. Given a set of equations, it is able to parse them, understand precedence between them, and then build the necessary structures to evaluate and execute them. The goal of the project is to design and develop an offline version of the calculation engine. Then, a containerization process should take place followed by developing the necessary scripts to deploy the application.

1.1.3 Project roles

Throughout the entirety of the project, the developer is supposed to set its main goal, state its needs and requirements, both technical and non-technical, and make a theoretical design of the solution with great consideration to the technical feasibility, the project execution, and project documentation. Throughout the realization of these activities, the work of the developer is professionally supervised by Mr.Youssef Fehri as project manager and Mr.Housseem Nefoussi as project manager depute.

1.2 The project owner

1.2.1 Presentation of Cognira

Cognira was founded in 2015. It is a multi-national company with offices in three global locations which are Atlanta, Georgia in the United States of America, Tunis in Tunisia, Lille in France, and employees in London and Istanbul. And since opening the Tunis office in 2017, it grew exponentially in terms of number of employees to reach more than 85 employees by the end of the first quarter of 2022. Globally, Cognira made a significant growth since its foundation, so significant that it ranked as one of the top 5000 growing companies in the United States two years in a row (2019 and 2020). Cognira's clients are particularly big retailers who are looking for a solution provider that understands the technical world as well as the retail world so the overall interaction would be as easy as possible for them. [2]

1.2.2 Field of expertise and solutions

Cognira provides three different solutions for its clients depending on their need.

1.2.2.1 Cognira's AI Promotion Solution

It forecasts demand of all the products accurately. It also reduces inventory costs and decreases waste with machine-learning promotional forecasts that accurately reflect demand and continuously improve over time. [3]

1.2.2.2 Forecast Service

It provides the accurate demand forecasts desired by the customer without the complexities of managing the system and the forecasting team. [4]

1.2.2.3 Assortment and Allocation Decisions Optimization

With the assortment and allocation decisions optimization solution, retailers get the most advanced and scientific answers to the questions that concern them most - what and how much to buy, and when and where to place products - helping to reduce loss and increase sales. [5]

1.3 Study of the existing

As retailers in the United States use promotions to reach their expansion goals while making maximum profit at the same time, they need a tool to provide them with indicators telling them which promotion to choose. For example, SAP provides its clients with a tool called 'Promotion Calculation Engine'. The highly flexible solution is built to manage a large number of simultaneous pricing computations [6]. Also, IBM offers a solution for its clients to compute profit off of promotions. It also suggests on their clients pricing proposals of certain products [7]. SAP takes about 4% of the overall retail software market share, making it ranking the 5th in place, as there is serious competition from companies like Adobe. This leaves companies like IBM in a position among the top 20 companies as its market share does not surpass 1.5%[8]. These mentioned solutions are specifically targeting small to medium retailers who use it by the hour or with a Pay As You Go approach, meaning that they have to pay depending on the number of times they use the service and how much data they are processing.

1.4 Problem statement

Big retailers who benefit from the Calculation Engine developed by Cognira use data about their products in order to get the expected results. On periodic basis, a new batch of data is accumulated by the retailer and added to the global data. This operation requires the recomputation of all indicators. Therefore, the existing Calc Engine performs the computation of all the indicators on the fly periodically. Upon launching the Calc Engine product by Cognira, the clients' expectations were met by it and they decided to adopt it. As their data grew, the current version wouldn't be efficient to use. This is because the system wasn't built to support computations on large volumes of data. This is causing quite some problems, despite using turn-arounds. Also, the client's expectations from the Calc Engine are versatile. Cognira's clients were using it once in a while and on demand. But now, they use it very often to do the calculations all over again. So we are building a batch processing version of the calculation engine so it can deal with big loads of data.

1.5 Project goals and objectives

The system to be developed must solve the discussed problems mentioned above. Therefore, the project's goals are as follows :

- **Offline-system-compliant design**

As this calculation engine is a batch processing system, the developed system must be designed to work offline. [9]

- **Support of large volumes of data**

One of the main problems that were met in earlier versions of the Calc Engine is that it cannot perform computations using large volumes of data. One main goal of the internship is to design and develop a Calc Engine that can perform calculations using big amounts of data.

- **Easy integration and deployment solution**

The developed solution need to be easily integrated or deployed on any environment.

1.6 Project deliverables

- **Source code :**

At the end of this internship, the source code of the different parts indicated is to be delivered. This deliverable includes the well reviewed and validated source code and the implementation of the test cases.

- **Requirements specifications :**

Requirements specifications must be prepared according to the project charter in order to specify the different tasks needed with an estimation of capacity and to know the functional expectations.

- **Source code documentation :**

The architecture and the implemented code must be accompanied by a conceptual documentation with all the diagrams that demonstrate the different flows.

- **Final presentation :**

During the internship, a progress report in the form of a presentation is expected on monthly basis in order to offer an overview of the project status, milestones reached, and difficulties encountered to the Cognira development team along with a group of interns.

1.7 Project non-objectives

- **A user interface :**

The input given to the calculation engine is not provided through a user interface. Instead, it will be placed in files that will be located in the indicated place through a service called the promotion planning service.

- **A RESTful API :**

Upon reading 'promotion payload', it may occur that it is provided through a RESTful API method, same assumption might be made on the output result. The input and output will not be obtained nor provided through a RESTful API.

1.8 Project management plan

After stating important definitions in the scope statement of this project like goals, sponsors, requirements and deliverables, the project organization and workflow will be described. In this project, the implementation of a testable, stable and scalable application is needed. These characteristics need to be present at all times during the development life cycle. Also, the implementation of frequent updates must be easy. Having all of these requirements to be taken into consideration, we opted that the agile methodology scrum would be the best method for managing and organizing the project.

1.8.1 scrum methodology definition

scrum is an agile methodology for development that is used in software development based on incremental and iterative processes. Scrum is an adaptable, fast, flexible and efficient agile framework. It is designed to deliver value to the customer throughout the development of the project. The scrum's main objective is to satisfy the customer's needs through an environment of transparent communication, collective responsibility and continuous progress. The development starts from a general idea of what needs to be built, by developing a prioritised list of features (product backlog) that the product owner wants to achieve [10]. Figure 1.1 represents the scrum methodology cycle.

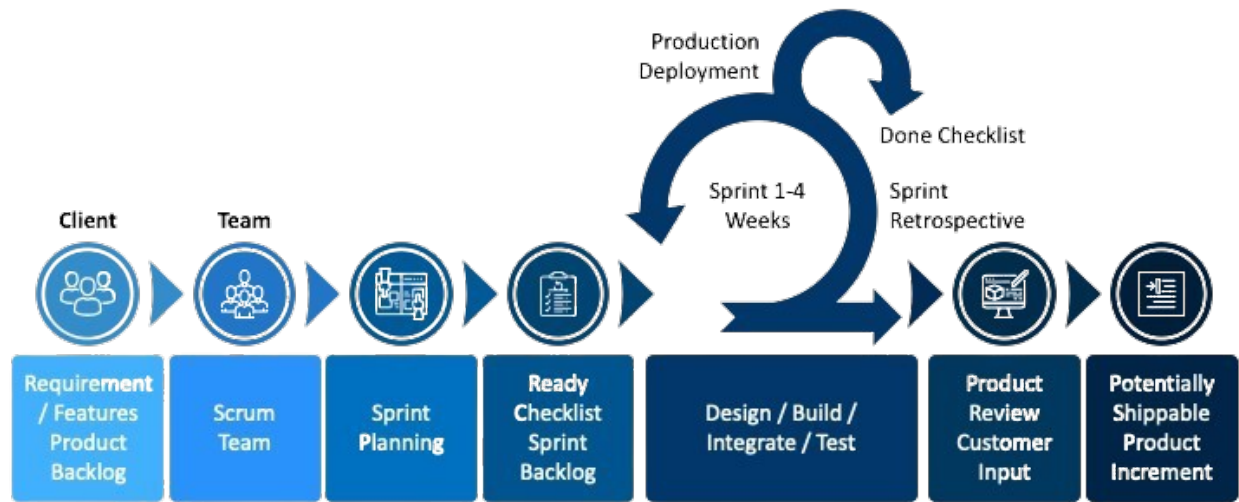


Figure 1.1: Scrum methodology cycle [11]

1.8.2 Scrum roles

Upon working with scrum methodology, a given team member should take part in one of these groups or take one of these roles :

1.8.2.1 Scrum master

The scrum master helps to facilitate the project workflow to the whole team by ensuring that the framework of the scrum is respected. The scrum master is committed to the values and practices of the scrum method, but must also remain flexible and open to the possibilities of the team and open to opportunities for them to improve the workflow. [12]

1.8.2.2 Development team

It's a self-organizing cross-functional team made up of scrum development team members. The team is in charge of constructing the actual product increment as well as meeting the sprint target. The development team's performance is critical to scrum's success. [13]

1.8.2.3 Product owner

He is a member of the Agile Team. He is responsible for defining the stories and for organizing the backlog of the team to order the execution of program precedence while keeping the conceptual and technical integrity of the components for the team. [14]

1.8.3 scrum meetings

1.8.3.1 Sprint review meeting

The scrum team and stakeholders discuss what was done throughout the sprint as well as what has changed in their environment during this meeting. The participants discuss what to do next based on these updates. The Product Backlog can be tweaked to accommodate new possibilities. The sprint review should not be limited to a presentation as it is a working session. [15]

1.8.3.2 Daily standup meeting

The daily scrum is a 15-minute meeting for the scrum team's developers. To keep things simple, it happens at the same time and place every working day of the sprint. They engage as developers if the product owner or scrum master is actively working on items in the sprint backlog. [16]

1.8.3.3 Code review process

In order to assure the good quality of the final product, a code review process occurs at the end of every sprint or following the creation of each component inside the project. As figure 1.2 demonstrates, the author submits the code to be reviewed. The reviewer have the ability to improve the code and to make remarks and comments on it. Afterwards, he can either approve it or not. In case it gets approved, the code is submitted and the author moves on to the next agile iteration. If not, further modifications are performed on the code until it is good enough to be submitted.

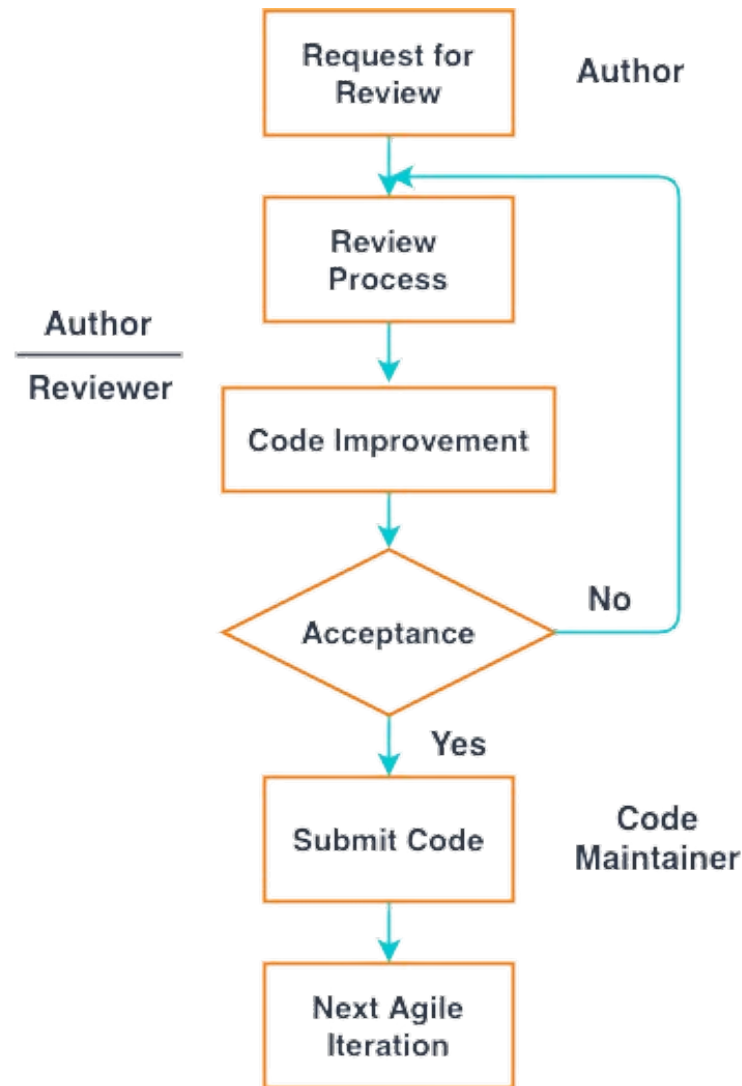


Figure 1.2: Code review process[17]

1.9 Project plan

1.9.1 Project realization overview

The workload of the implementation of the Spark Calc Engine Counterpart consists of :

- Phase 1 : Calc Engine context understanding by reading thorough documentation provided by Cognira. The main aim of this phase is to understand the syntax of the equations as well as the general workflow of the calculation engine.
- Phase 2 : Design phase since the current Calc Engine design is not useful if we want to work with Spark for data processing, which is what this project consists of.

- Phase 3 : Implementation and testing phase where the development work was done based on the design phase that came earlier.
- Phase 4 : Deployment phase where the necessary scripting is developed to be able to run the project on a standard infrastructure.

1.9.2 Project progress chronogram

The following measures were taken in order to keep track of the project execution :

- daily standup meeting so the interns could update their supervisors with the work progress
- weekly deep dive meeting every friday in order to track the overall work done for the week
- monthly work progress presentation
- code review after pushing the code of a certain sprint to the version control management tool

The timetable designed to track the project's progress is depicted in the Gantt chart[18] presented in table 1.1. In order to make sure that the project features are implemented in the best way, sticking as much as possible to this timeline was important. The work was divided into sprints. JIRA is a project management software that was used to follow the project's track. In JIRA, a sprint is called also an 'epic'[19].

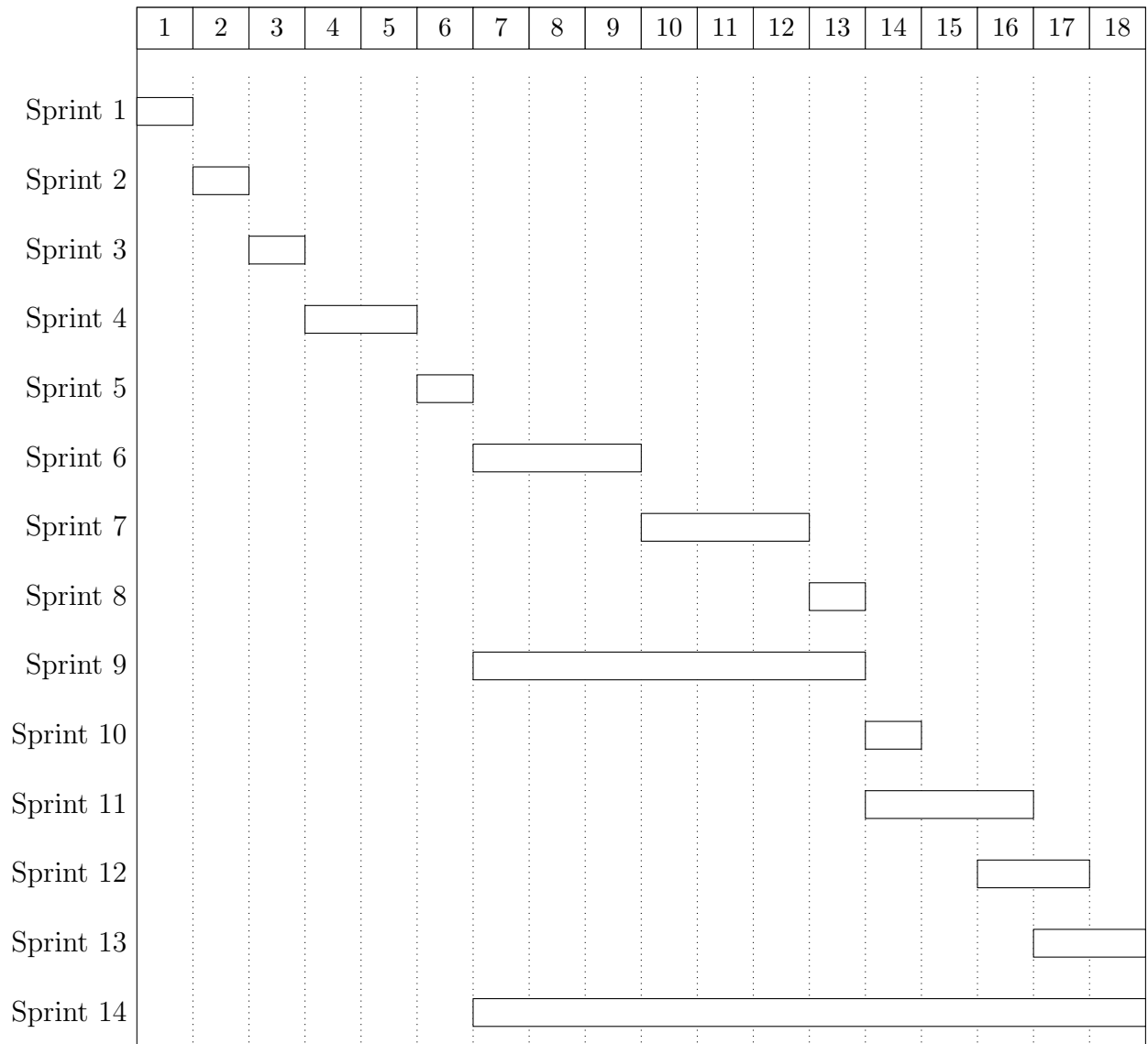


Table 1.1: Gantt diagram

These are the descriptions of each sprint :

- ✓ Sprint 1 : Training on Cognira backend development stack.
- ✓ Sprint 2 : Spark features exploration.
- ✓ Sprint 3 : Calc Engine exploration; discovering the concepts related to Cognira's Calc Engine, starting from the equations to the general structure.
- ✓ Sprint 4 : Designing the new Calc Engine.
- ✓ Sprint 5 : Implement the equations parsing from a human readable syntax into code.
- ✓ Sprint 6 : Implement the equations fetching as well as implementing the mechanism that will order the execution of the equations.
- ✓ Sprint 7 : Implement the execution of the equations.
- ✓ Sprint 8 : Implement promotions payload input to the system as well as indicators output.
- ✓ Sprint 9 : Unit test every component upon making it

- ✓ Sprint 10 : Containerize the application with docker.
- ✓ Sprint 11 : Deploy the application on Kubernetes.
- ✓ Sprint 12 : Make a CI/CD pipeline.
- ✓ Sprint 13 : Make a monitoring dashboard.
- ✓ Sprint 14 : Write the report.

Conclusion

In this chapter, a detailed description of the project is demonstrated along with a description of the necessary workload that should take place in order to finish the project within the required expectations. The following chapter is dedicated to the description of the requirements analysis and specifications of the project.

Chapter 2

Requirements analysis and specifications

Introduction

This chapter contains the requirements analysis section that contains functional, non-functional, and technical requirements. It also contains the requirements specifications section that contains the use case diagram as well as the activity diagram.

2.1 Requirements analysis

In this section, an analysis of the expected project functionalities is done. This step is necessary before starting the implementation process.

2.1.1 Functional requirements

Functional requirements define a set of details and features about the final version of the project. This application will provide these functionalities :

2.1.1.1 Generic calculation

Given the equations' syntax provided by Cognira, one of the main goals to attain is to write an algorithm that handles the different edge cases in the mentioned syntax.

2.1.1.2 Configurability

As Cognira provides the calculation engine service for numerous clients, each one of them has different requirements. Therefore, the final application needs to be highly configurable to fit every client's needs.

2.1.1.3 Replication handling

In the syntax provided by Cognira, some of the equations' results are the outcome of a computation between two parameters that don't necessarily share the same dimension. One particular needed functionality is to perform these types of computations through a process called 'replication' that is defined by the Cognira development team.

2.1.2 Non-functional requirements

These are requirements that the project can work without having them implemented but it is preferably to have them as they present a success measure to the work that is done. For this project, they are as follows :

2.1.2.1 Scalability

One of the main reasons of working with Scala as a programming language and Kubernetes as a container orchestrator is that they are used to build and deploy scalable applications. In the case of this project, scalability is wanted because the loads of data getting processed vary from client to client.

2.1.2.2 Performance

The application has to be performant especially when taking into account the various data processing operations that occur within it.

2.1.2.3 Reusability

The code that is written have to follow certain structures and rules that makes it easy for developers who will be working on the project to understand the code, especially while providing a suitable documentation as well.

2.1.3 Technical requirements

These are the technical requirements and their respective description. They are as follows :

2.1.3.1 Testing

In order to be certain that the computations happen in the way it is supposed to be, testing scenarios must be implemented to assure the quality of the final product.

2.1.3.2 Containerization

the code and the dependencies of the application developed need to be packaged into a standard unit of software called a container.

2.1.3.3 Deployment

The previously mentioned container need to be deployed on the deployment infrastructure provided by the host company via a tool for container orchestration.

2.2 Requirements specification

2.2.1 Modeling language

The project's objects were modelled using the UML "Unified Modeling Language." It's a modeling language for designing and implementing software architecture. It is made up of a number of diagrams that depict the system's boundaries, structure, behavior, as well as the components that make it up. [20]

2.2.2 Actor identification

There is one actor that interacts with the engine.

- **promotion planning service** : This actor is responsible for putting input files in the adequate location, starting the engine, and reading the output files. these actions happen independently, meaning that putting the input files in the adequate location does not trigger the engine to start, it is started separately. Also, when the engine is done executing and the output files are generated, the promotion planning service has to specifically request these files on the spot.

2.2.3 Use case diagram

We expose, in this section, the engine design by modeling the class diagram that will help to better understand the mechanism of the implementation of the software. Figure 2.1 represents the use case diagram of the project.

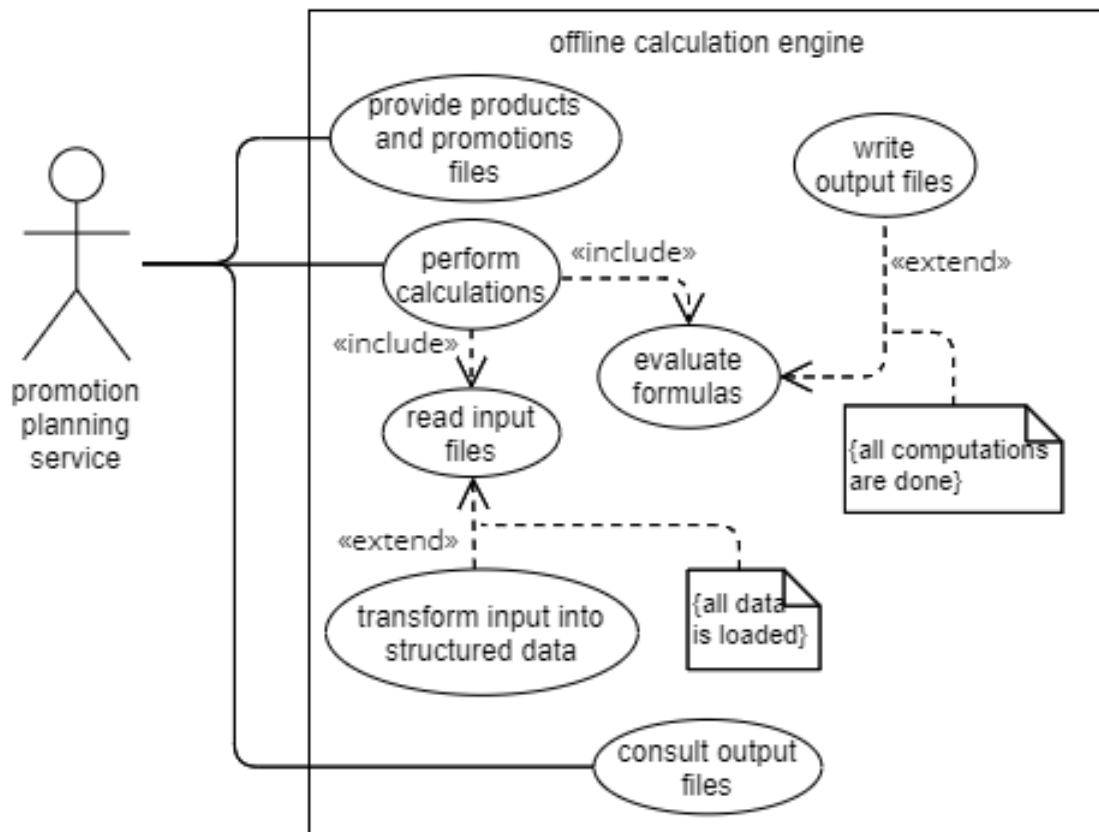


Figure 2.1: Use case diagram

2.2.4 Use case textual description

The textual description of the use cases is a description of the chronology of the actions. It allows to clarify the flow of the functionality and to indicate possible constraints. In the following, we present the textual descriptions of some use cases in order to have a global vision of the functional behaviour of our application.

2.2.4.1 Use case: provide products and promotions files

Table 2.1 resumes the use case of setting up the needed input files (products file, promotions file, and formulas files) in the adequate location on the system.

Use case description: provide products and promotions files
Identification Title: provide products and promotions files Goal : provide the engine with adequate products data and the promotions payloads to do the computation. Principal actor : promotion planning service Summary : The promotion planning service provides the files having prices and costs of products, as well as the files having the JSON promotions payloads to perform the computations on. It also sets up the formulas files.
Sequencing The use case starts when the promotion planning service proceeds to get the concerned files to be put in the designated location.
Pre-conditions No pre-conditions are required.
Nominal scenario <ol style="list-style-type: none"> 1. The promotion planning service sets up the formulas files. 2. The old promotions file get deleted. 3. The promotion planning service provides new promotions file in the adequate location. 4. The prices and costs file get updated by the promotion planning service.
Exception scenarios E1 : some or all of formulas files are named after a promotion type that does not exist <ol style="list-style-type: none"> 1. The engine does not include the formulas of these files in any computation. E2 : the old promotions file does not exist <ol style="list-style-type: none"> 2. The promotion planning service look for files with names having the pattern 'promotion' to delete them.
Post-conditions The engine will be doing the computations based on the new input.

Table 2.1: Use case: provide products and promotions files

2.2.4.2 Use case: perform calculations

Table 2.2 resumes the use case of performing the calculations.

Use case description: perform calculations
Identification Title: perform calculations Goal : perform the needed calculations. Principal actor : promotion planning service Summary : The calculation engine performs the calculations based on the parameters provided in the input files.

Sequencing
The use case starts when the promotion planning service starts the engine.
Pre-conditions
No pre-conditions are required.
Nominal scenario
<ol style="list-style-type: none"> 1. The engine reads the promotions file. 2. For each promotion, the engine gets the formulas from the input files based on the promotion type. 3. For each promotion, the graph presenting the ordering mechanism of the formulas' execution order is built. 4. For each promotion, the graph is traversed by evaluating the formulas using already calculated indicators, promotion-specific parameters, and the products' prices and costs.
Exception scenarios
E1 : The concerned promotion's type has no associated formulas file 2. Only the file that has formulas valid for all promotion types is used.
Post-conditions
The engine completed all the calculations.

Table 2.2: Use case: perform calculations

2.2.4.3 Use case: transform input into structured data

Table 2.3 resumes the use case of transforming input into structured data.

Use case description: transform input into structured data
Identification Title: transform input into structured data Goal : turn the input coming from the files provided in the designated location from string formats to instances of custom structures of data. Principal actor : The engine Summary : transform input coming from files provided by the promotion planning service from string format in CSV files into instances of specific classes so the engine can do the computations based on them.
Sequencing The use case starts when the engine is fully started.
Pre-conditions - The existence of the formulas file that is valid for all promotions types is crucial to start the processing.

<p>Nominal scenario</p> <ol style="list-style-type: none"> 1. The engine reads the promotions' payloads file as flatten JSON lines. 2. Each promotion JSON is transformed into an instance of the class that corresponds to promotions. 3. For every promotion, the equations string input is read from a specific location depending on the promotion type. 4. For every promotion, its associated string equations are transformed into instances of a specified class presenting equations. 5. For every promotion, the products data is read and transformed into dataframes.
<p>Exception scenarios</p> <p>E1 : no promotions input file is provided</p> <ol style="list-style-type: none"> 1. An exception is thrown indicating that no file for promotions is given. <p>E2 : promotion type provided in input doesn't exist</p> <ol style="list-style-type: none"> 2. Only indicators valid for all promotion types are computed. <p>E3 : no formulas file provided for the specified promotion type</p> <ol style="list-style-type: none"> 3. Only the formulas that are valid for all promotion types are read. <p>E4 : no products data file is provided</p> <ol style="list-style-type: none"> 3. An exception is thrown indicating that no products input file is provided.
<p>Post-conditions</p> <p>The engine is ready to do evaluations based on the promotions input, the products input, and the equations.</p>

Table 2.3: Use case: transform input into structured data

2.2.4.4 Use case: evaluate formulas

Table 2.4 resumes the use case of evaluating formulas of the promotions indicators based on the structured input.

Use case description: evaluate formulas
<p>Identification</p> <p>Title: evaluate formulas</p> <p>Goal : do the calculation of the indicators based on the instances of data structures presenting the promotions, the formulas, and the products' prices and costs.</p> <p>Principal actor : The engine</p> <p>Summary : the formulas associated with each promotion are evaluated based on prices and costs data and parameters from the promotion payload.</p>
<p>Sequencing</p> <p>The use case starts when the first formula in the ordering mechanism is about to be calculated.</p>

Pre-conditions - The input data should be transformed in order to do the calculations. - The ordering mechanism is already created. - The formulas to be used for a given promotion are guaranteed that they will create a DAG.
Nominal scenario 1. The context of each formula is obtained depending on the promotion parameters to obtain the formula context. 2. For each formula, the dataframes needed for the evaluation are collected. 3. For each formula, the collected dataframes are intersected together and replicated if needed. 4. For each formula, the intersected dataframes are filtered depending on the formula context. 5. The indicated formula is computed.
Exception scenarios E1 : an equation within a graph depends on an indicator that does not exist 5. The result of the concerned equation is set to 'null'. E2 : The intersection between some or all the dataframes produces empty dataframes 5. The result of the concerned equation is set to 'null' E3 : The filtering of the dataframes depending on the formula context produces empty dataframes 5. The result of the concerned equation is set to 'null'
Post-conditions The engine is ready to transform the evaluated formulas' results into output files.

Table 2.4: Use case: evaluate formulas

2.2.4.5 Use case: write output files

Table 2.5 resumes the use case of writing the wanted output files.

Use case description: write output files
Identification Title: write output files Goal : transform computation results into files. Principal actor : The engine Summary : take the results of the computations associated with each promotion, turn them into the adequate structure, and output them into files in a designated location.

Sequencing
The use case starts when the computation related to the first provided promotion is done.
Pre-conditions
All the calculations of a given promotion are already performed.
Nominal scenario
<ol style="list-style-type: none"> 1. Any folders containing old output files are automatically deleted. 2. For each promotion, the results gets transformed into a format ready to be stored into a CSV file. 3. For each promotion, the structured results are turned into one CSV file. 4. The CSV file is placed in a folder named as the promotion id and the promotion type.
Exception scenarios
E1 : No previous output folder is found <ol style="list-style-type: none"> 1. A new output folder for all promotions computations is created.
Post-conditions
The files, existing in the adequate locations, are ready to be read by the promotion planning service.

Table 2.5: Use case: write output files

2.2.5 Activity diagram

An activity diagram will be used to understand the flow of the program. It will also help to figure out constraints and conditions within the project. Figure 2.2 represents the activity diagram.

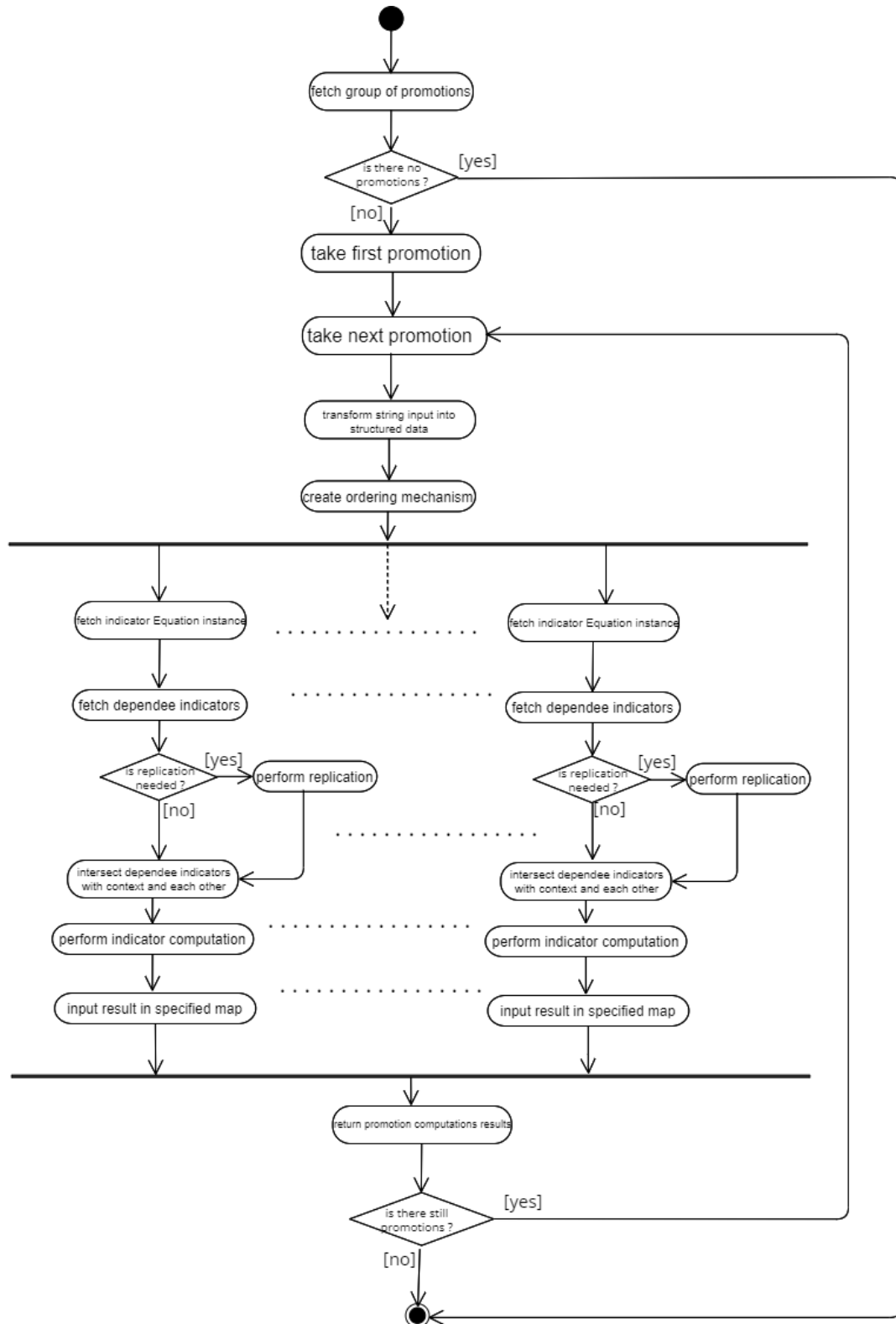


Figure 2.2: Activity diagram

Conclusion

Throughout this chapter, the functional, non-functional, as well as technical requirements are presented along with the activity diagram and the use case diagram in order to demonstrate what the software should offer. Therefore, the next chapter contains the engine design.

Chapter 3

Design of the calculation engine

Introduction

To ensure that the goals outlined in previous chapters are met, design is a necessary step before beginning development. In this chapter, we'll explain basic concepts of Cognira's calculation engine. This is done to use these latters in the design. Then, we'll be presenting different UML diagrams of the application.

3.1 Retro-engineering the online engine

3.1.1 Definition of parse tree

A parse tree is a hierarchical representation of terminals and non-terminals. These symbols indicate the derivation of the grammar for generating input strings (terminal or non-terminal). During parsing, the start symbol is utilized to begin the string. The grammar start symbol must be the root symbol of the parse tree. The leaves of the parse tree define its boundaries. Grammatical productions are reflected in each internal node. The rules for implementing a parse tree are as follows: all leaf nodes must be terminals, whereas all inner nodes must be non-terminals. The original input string is returned if the crosses are done in the correct sequence[21]. Parse trees' main advantage is that they are not restricted by a rigorous grammar, allowing them to operate as linguistically context-free equations as possible. Consider the representation of a mathematical equation in a parse tree[22]. This is the equation: $a*b+c*(d*(g-f))$, and this is its parse tree representation in figure 3.1 :

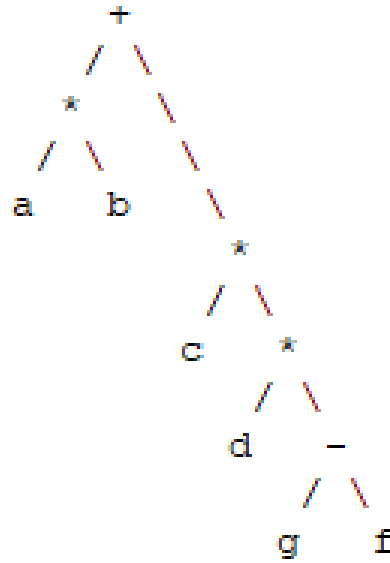


Figure 3.1: Parse tree example

Note that this representation is just an example to explain the concept of a parse tree, the implementation in our study case is not necessarily the same.

3.1.2 Cognira's indicators' equations

3.1.2.1 indicator

It is a parameter that is used to make an informative decision on which promotion to choose among several. An indicator is either associated with an equation or given directly as a parameter within the promotion.

3.1.2.2 dimensions

According to the logic proposed by Cognira, a given indicator has a certain dimension. The proposed dimensions' hierarchies are either promotion,product,store , promotion,condition, product,store , or product,store,day. For example, an indicator on the promotion,product,store dimension is a table representing the prices of the products in every store, where the products and stores are associated with the given promotion. If we want to get the maximum price of every product in the stores where it is displayed, we simply apply a MAX aggregation, so we can get the maximum price indicator that is on the promotion,product dimension. If we want to get the maximum price among all the products of the promotion, we apply a MAX aggregation again to get the maximum price indicator on the promotion dimension.

From now on, the promotion dimension shall be referenced as Pm, the condition dimension shall be referenced as C, the product dimension shall be referenced as Pr, the store dimension shall be referenced as S, and the the day dimension shall be referenced as D. Therefore,

the representation of the hierarchies is in figure 3.2. Every configuration of hierarchy has its derivative set of indicator's dimensions :

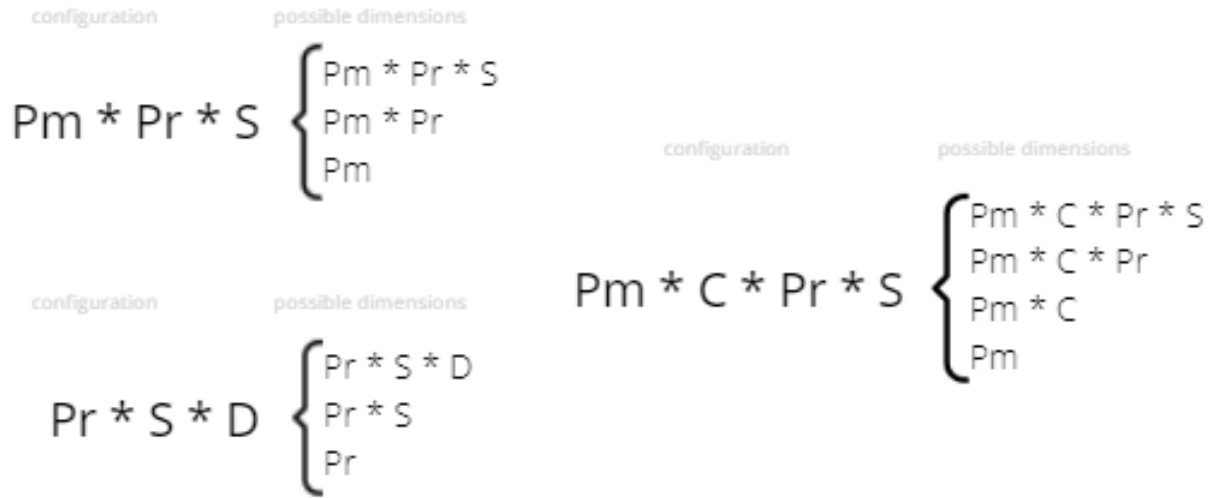


Figure 3.2: Dimension hierarchy configurations

As for the promotion,condition,product,store hierarchy, let's say that we have the cost indicator on the promotion,condition,product,store dimension, which is interpreted in this way; it is a table having the cost of a certain product from a given condition in each store in the given promotion. If we want to know the average cost of a certain product across all stores, we do an AVG operation on the store dimension, so we get the average price indicator that is on the promotion,condition,product dimension.

3.1.2.3 context

There are two kinds of contexts in the Calc Engine; the promotion context, and the indicator context.

The promotion context is the list of parameters that define a given promotion, like its type, its associated stores, its associated products, and so on and so forth. The indicator context on the other hand, represents a group of tables, like the stores table or the products table, associated with that indicator.

Using the context of the indicator is obviously crucial for its computation. As for the promotion context, it is used to avoid the concurrency problem. This is because if we use the same attributes for all the promotions being computed at the same job, two different promotions with two different products and stores could have two indicators who share the same name and dimension but not necessarily the same value. So, we are encapsulating the computations that happen for a given promotion. From now on, we will reference this by the name 'Promotion Encapsulation'.

3.1.2.4 indicator to parse tree

The indicators used in the evaluation of a given promotion are parse trees with a syntax that is specific to Cognira. This is an example of an equation that is used in the Calc Engine:

$$\begin{aligned} \text{min_reg_price}@Pm = & \min_{\substack{Pr \in \text{for_products}@Pm, \\ S \in \text{targeting_set}@Pm}} (\text{reg_price}@Pm, Pr, S) \end{aligned} \quad (3.1)$$

This equation represents the minimum regular price, which comes by doing a MIN operation on the regular price indicator that is on the promotion, product, store dimension. The computation is done given the equation context that follows: the products have to belong to the for_products list of the specified promotion, same for stores with the targeting_set list.

There are three types of equations. The first one is an arithmetic operation, which can be an addition or a subtraction between two or more indicators for example. The second type is an aggregation over an indicator. The previous equation of the minimum regular price indicator is an example of that. The last type is the aggregation over an arithmetic operation.

In order to write the equation in a way that could be interpreted by a software application, Cognira chose the parse tree logic in order to represent the equation. The parse tree representation of the previous equation is as follows in figure 3.3 :

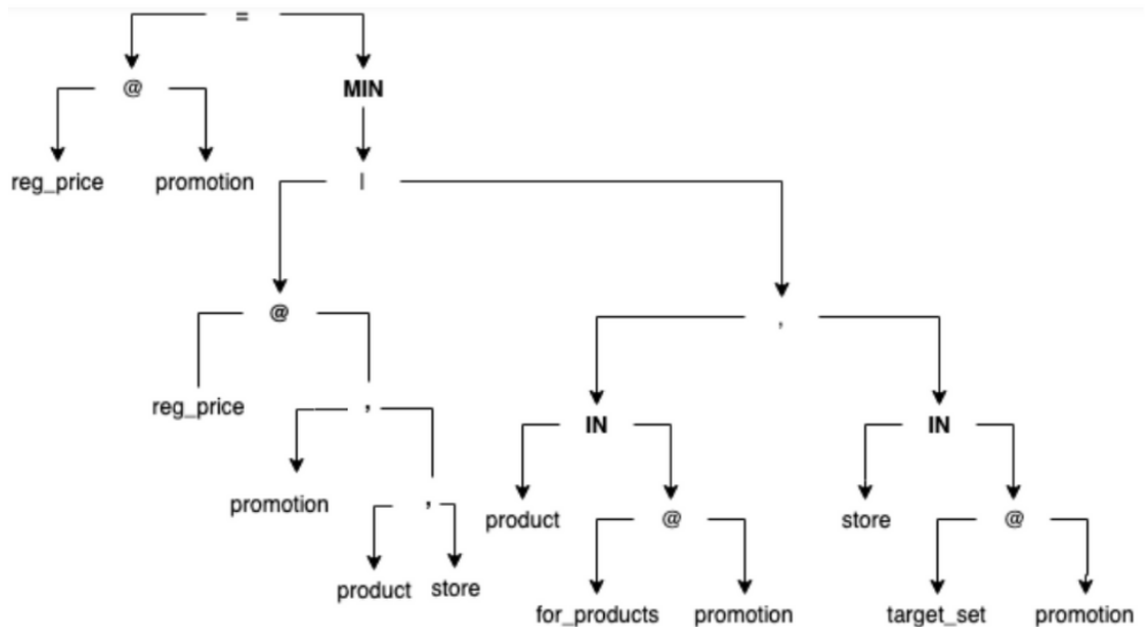


Figure 3.3: Cognira's parse tree example

In order to be stored in a machine-readable format, a parse tree is presented as a JSON string into the data warehouse. When it is fetched, it is converted into a parse tree instance. It is composed of a data_type attribute, which is a string representing the type of the label.

It is also an attribute representing the operation to be done at that specific parse tree. The parse tree itself contains children parse trees as attributes. In case where a parse tree is a leaf, its associated parse trees are null. For example, figure 3.4 shows a part of a parse tree representing an indicator that is used in the computation of an equation :

```
"left_child": {
  "data_type": "fetch",
  "label": "@",
  "left_child": {
    "data_type": "field",
    "label": "max_reg_price",
    "left_child": null,
    "right_child": null
  },
}
```

Figure 3.4: Part of parse tree as a string JSON example

3.1.2.5 Execution dependency graph

To calculate a certain indicator that is dependent on another set of indicators, we have to calculate that set first. The indicators in it can be dependant on other sets of indicators as well. Therefore, upon calculating one indicator, we will be calculating several indicators ahead. What if a given indicator is used for calculating several other indicators ? we will be calculating a given indicator several times, which is not efficient. Therefore, the execution dependency graph is built. It is a graph that has indicators as nodes. It also has edges that represent the dependence of a certain indicator on another. We start by calculating the indicator with no uncalculated dependent indicators associated with it. Then, we continue the computation of each indicator, that is represented as a node, when it has no edges coming into it. The computation continues until no indicator is still uncalculated. By using this structure to order the computation of the indicators related to a given promotion, we guarantee that each indicator is calculated exactly once. In the figure 3.5 following this paragraph, an example of an execution dependency graph is shown, where indicator A depends on B and B depends on both A and C, and finally, D depends on E. As both E and C do not depend on uncalculated indicators (meaning they come directly from the data source or the promotion payload), they should be evaluated first.

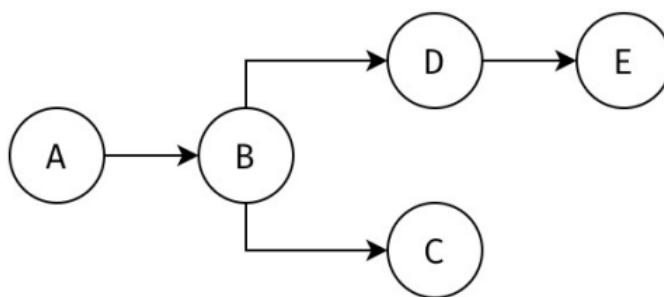


Figure 3.5: Execution dependency graph example

3.1.3 Replication

One of the main advantages of the syntax provided for the Calc engine's equations, is that an arithmetic operation can occur between two or more indicators who don't necessarily share the same dimensions. For example, an addition operation between `max_price@Pm,Pr,S` and `amount_off@Pr` is basically repeating the same value of `amount_off` on every product,store pair. The promotion id coming from the promotion context is "promo_1", the products existing in the resulting indicator's context are "Pr_1", "Pr_2", and "Pr_3", and the stores existing in the resulting indicator's context are "S_1" and "S_2". The figure 3.6 demonstrates a basic example of how the replication process occurs :

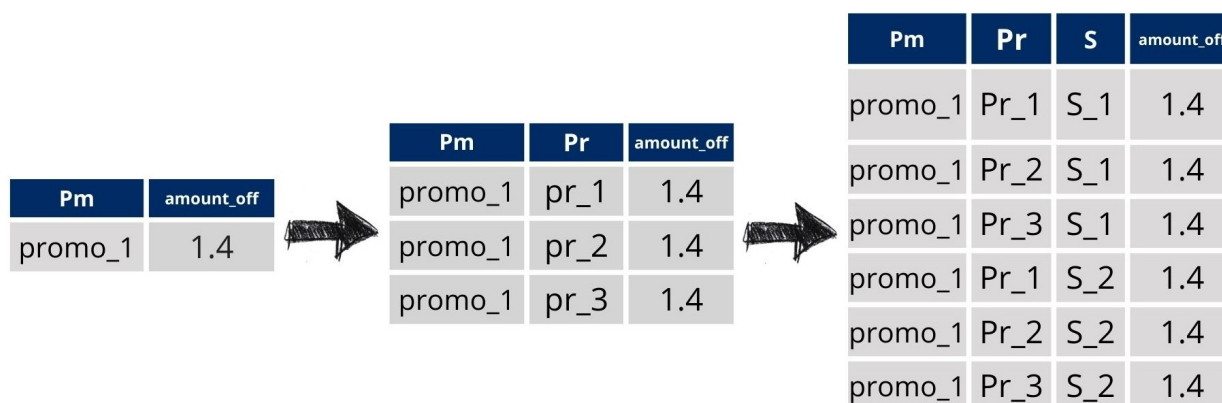


Figure 3.6: Replication process demonstration

3.1.4 Computing an indicator's value

In order to compute an indicator, its associated equation is needed, as well as its dependee indicators come from the data source or from the promotion payload. After the computation process, we get the value of the indicator as a result. The figure 3.7 demonstrates the process :

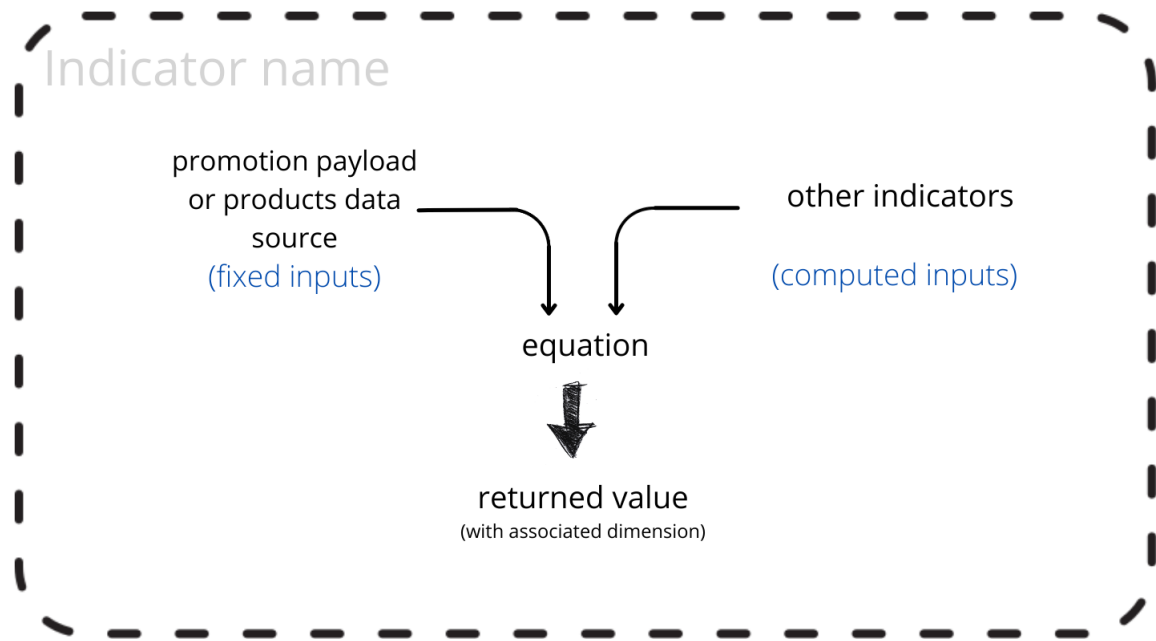


Figure 3.7: Demonstration of indicator computation process

3.1.5 Calc Engine workflow

The Calc Engine takes three different kinds of input; it takes the promotions to be computed in parallel at once from the promo planning micro service, the equations as parse trees from the database, and it takes the prices and costs of a given product at a given store at a given day from the databases.

It inputs the promotions first. Then, the equations are fetched in string format, which gets transformed into a parse tree depending on a set of attributes related to that promotion such as its type. Afterwards, the execution dependency graph of a given promotion is made where a given equation gets transformed into a 'Bound Tree', which is a structure that is composed of other bound trees, it can be made to either parse a certain indicator or to perform an operation such as an addition between its associated children bound trees. Note that if an indicator is computed, it is stored through a caching system.

After completing the computations of a certain promotion, its associated indicators are given as output. Here is the sequence diagram of the Calc Engine that is proposed by Cognira shown through figure 3.8 :

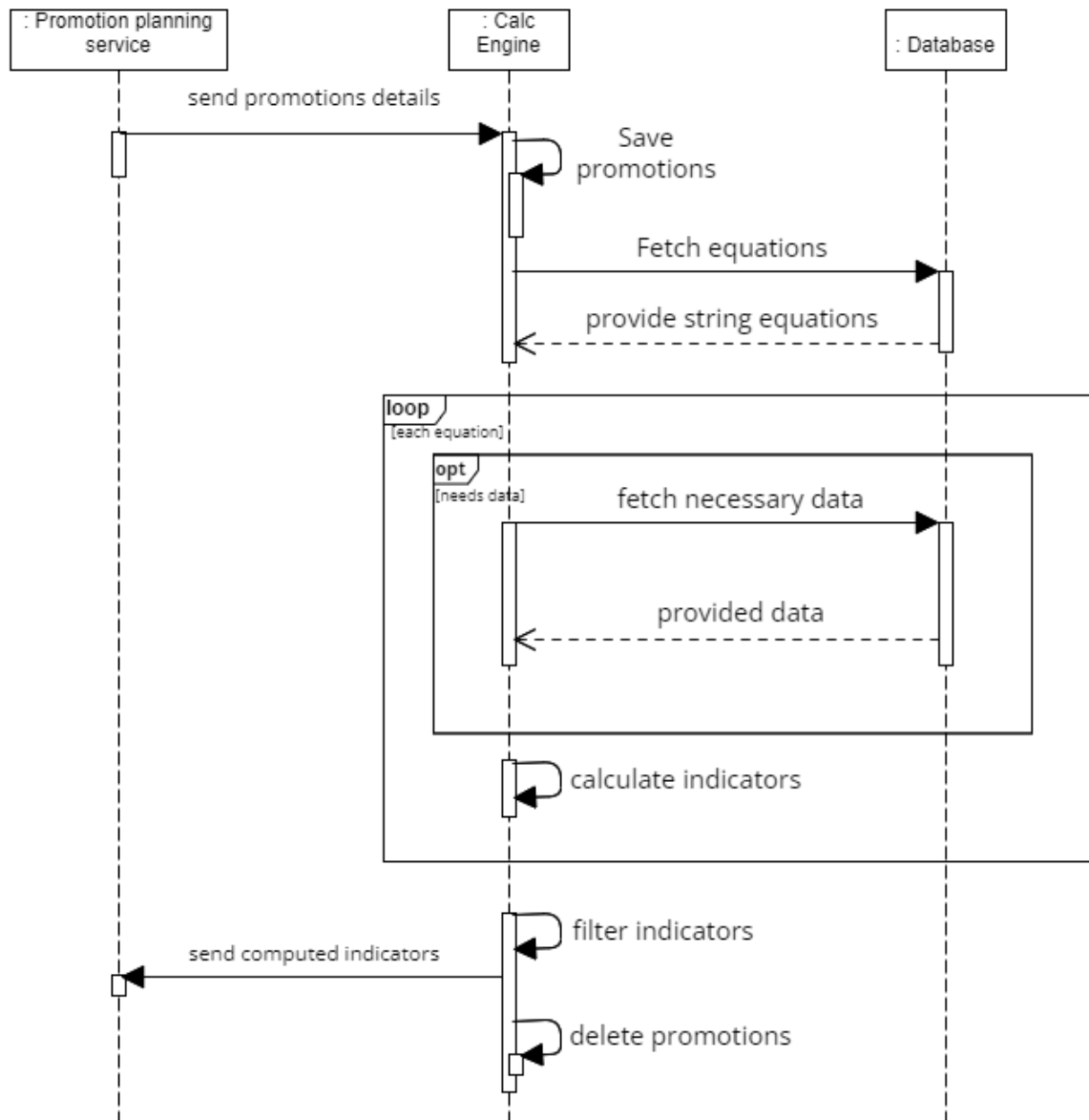


Figure 3.8: Sequence diagram of the existing calculation engine

3.2 Object-oriented conception

Object-oriented programming is a software development paradigm that prioritizes data above functions and logic. A field of data having definite qualities and behavior is referred to as an object. It was chosen because it allows you to divide down the program into smaller, easier-to-manage parts[23].

3.2.1 Class diagram

Class diagrams are one of the most useful and frequently used UML diagrams since they depict the structure of a system's classes, characteristics, actions, and object relationships[24]. The figure 3.9 demonstrates the class diagram of the whole application.

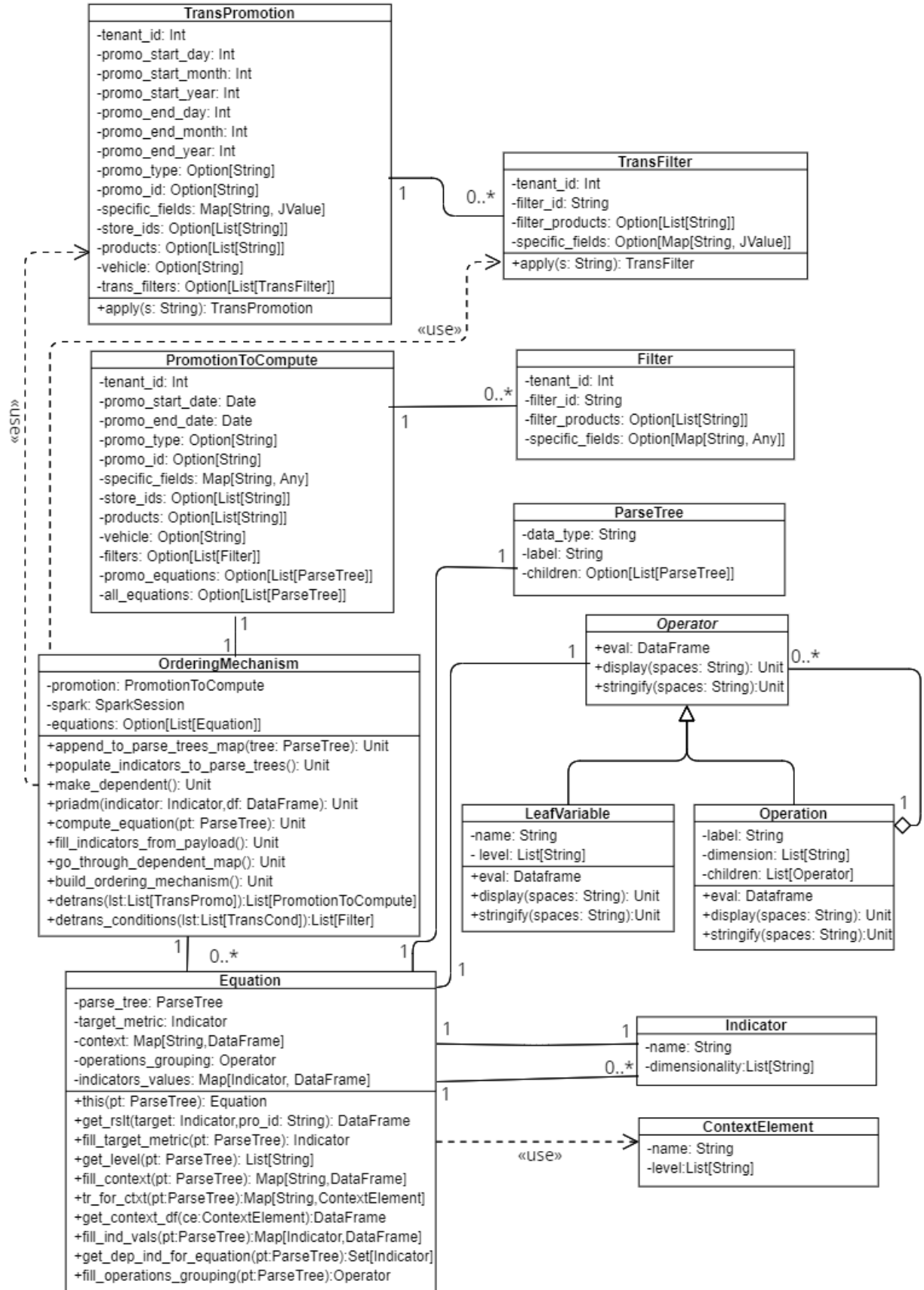


Figure 3.9: Class diagram

3.2.2 Class diagram description

Table 3.1 describes and illustrates the various classes that are used in the engine :

Class name	Class description
PromotionToCompute	It represents a given promotion. It has attributes such as the promotion id, its type, the stores and products associated with it, its start and end date, ... Concerning the promotion type, it can be one of the following : Amount Off, Price Point, Percent Off, BXGX (Buy X et X), BXGY (Buy X Get Y), EDLP (Every Day Low Price), Free,
Filter	It represents a condition of a certain promotion. It has attributes like buy_get.indicator, which is the type of the condition, it can be either 'Buy' or 'Get'.
OrderingMechanism	It represents the graph needed for the computations for a certain promotion. Thus, it has attributes like its associated promotion (PromotionToCompute class instance).
ParseTree	It represents a tree having a Cognira-specific context. It has attributes like its label and its children.
Equation	It represents all what is needed to compute a particular indicator; it has its indicator of class Indicator, its equation transformed from a ParseTree into an executable operation, as well as the needed indicators for the calculation of this particular equation. It also has the context elements needed for the computation.
Indicator	It represents a given indicator, given its name and dimension.
Operator	This is a trait used to perform the computation of a given indicator. It is used as a part of the composite design pattern.
Operation	It is a class extended from the Operator trait. It represents a part of an equation that comes through the computation of two or more indicators.
LeafVariable	It is a class extended from the Operator trait. It represents an indicator or a numeric that is used in the calculation process of another indicator.
TransPromotion	This class is a transitory class used to parse the json promotions input because parsing a date in JSON directly is not possible, and parsing an instance of the Scala type 'Any' directly isn't possible either. So the dates are divided into days, months, and years as attributes of type 'Int', and the instance that is supposed to be of type 'Any' is read as an instance of type 'JValue'.

TransFilter	This class is a transitory class used to parse the JSON filters in JSON promotions input because parsing an instance of the Scala type 'Any' directly isn't possible. So the instance that is supposed to be of type 'Any' is read as an instance of type 'JValue'.
ContextElement	This class is a transitory class that helps in building instances of class 'Equation' by parsing the elements of the context related to a certain equation and transforming it to a dataframe to be used in the computation of the equation.

Table 3.1: Software environment

3.3 Deployment diagram

The deployment step is depicted using a deployment diagram. It outlines how the components of a system are distributed and connected in the infrastructure[25]. The following diagram in figure 3.10 demonstrates the deployment of the application.

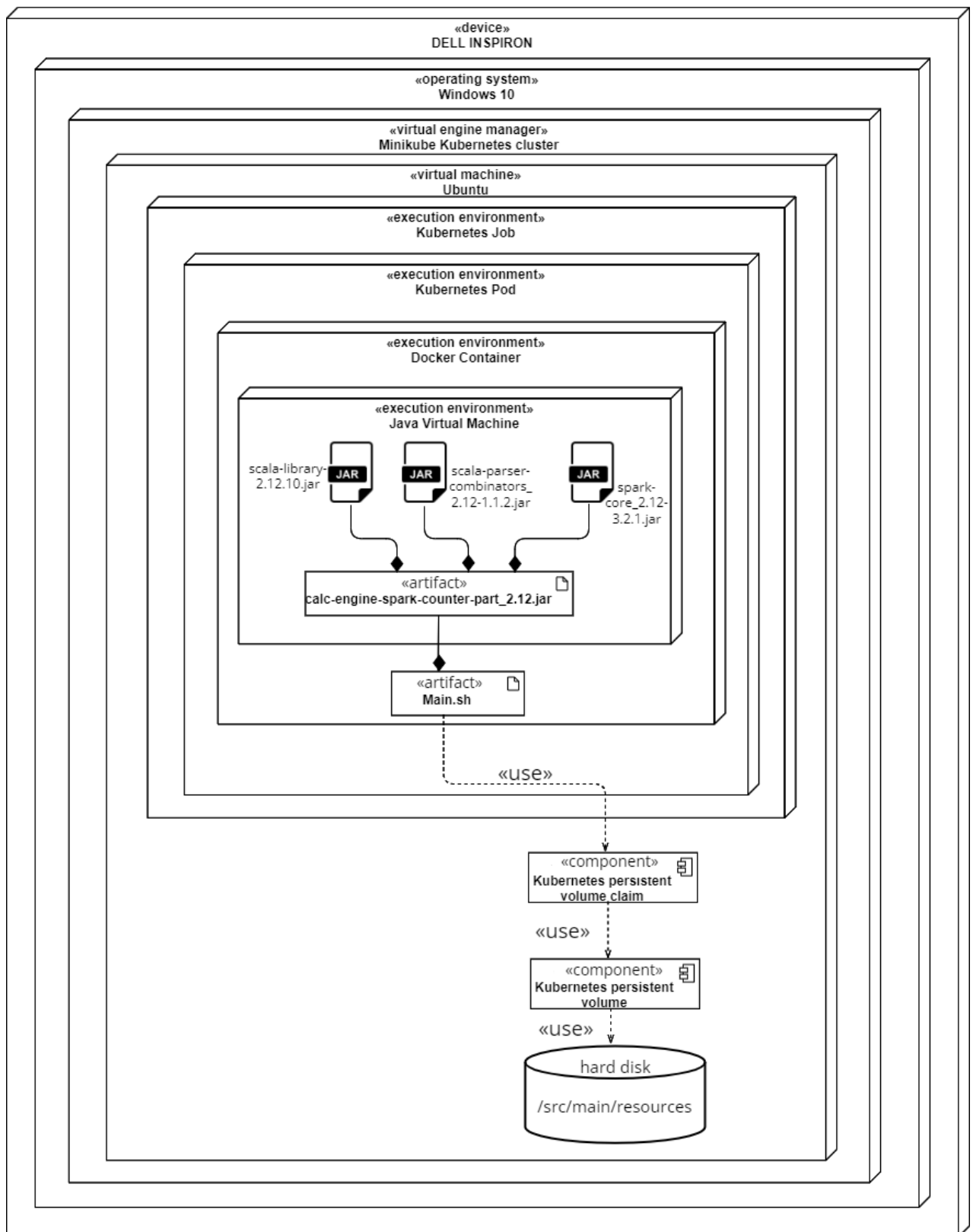


Figure 3.10: Deployment diagram

The deployment can also be modeled using Kubernetes objects' icons. This is because it makes it simpler to understand the architecture. The figure 3.11 represents the components that exist within the Kubernetes cluster using Kubernetes objects :

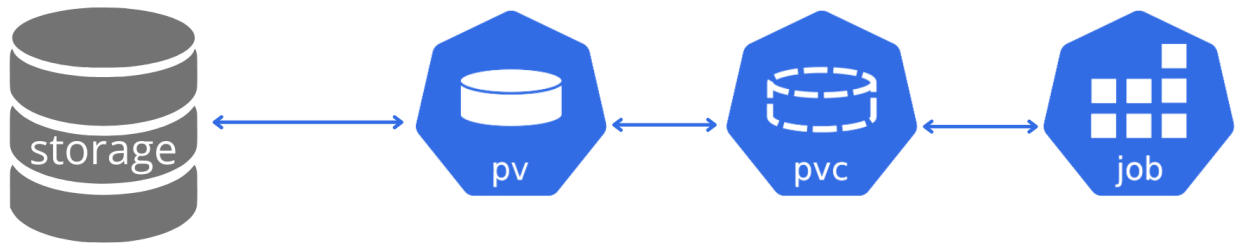


Figure 3.11: Deployment on Kubernetes

The first component is a job, a Kubernetes object that creates one pod to do the program execution and shuts down once it completes its task. The second component is a persistent volume, which is a storage object in the Kubernetes cluster that uses the local storage of the node that it is on it. The final object is the persistent volume claim which is a Kubernetes object that assures the connection between the persistent volume and the pod created by the job.

3.4 Conclusion

In this chapter, we have gone through the overall design of the application. In the next chapter, we detail the implementation phase of the project along with the testing and validation phases.

Chapter 4

Project implementation and testing

Introduction

In this chapter, technical details related to the implementation of this project are described. First, there's a description of software and hardware environments. Afterwards, the chosen technologies for the project implementation are outlined. Then, implementation details are demonstrated. After that, the testing and validation phases are showcased.

4.1 The development environment

The working environment plays an important role in order to make an application. It helps in delivering the product in good quality and helps to ensure a good work process for the developer. Our working environment is shown as below:

4.1.1 Hardware environment

As this is a software project, the only hardware that will be used is the laptop used for the development. Table 4.1 contains a description of it :

Laptop	DELL INSPIRON 3542
RAM	16 GB
Storage	1 TeraByte
Processor	Intel i5-4210U 2.7 GHz

Table 4.1: Hardware environment

4.1.2 Software environment

A wide variety of software tools were used while implementing the project. Here are the main software tools used for the project realization presented through table 4.2 :

Desired functionality	Designated tool
IDE	IntelliJ IDEA was chosen. This integrated development environment is mainly dedicated for JVM languages. It was used to develop the application as well as to write its containerization and deployment files.
Diagrams editor	Visual paradigm is an online application for creating various UML diagrams, like class and activity diagrams.
Containers setup	The containers were built locally using Docker Desktop, a platform for generating and sharing containers.
Cluster setup	Minikube is a tool that allows you to build up a Kubernetes cluster on your own workstation. It was used to run the Kubernetes deployment scripts.
Project management	JIRA is a proprietary issue tracking tool developed by Atlassian that includes bug tracking and agile project management features.
Version control	Bitbucket is a Git-based source code repository hosting service.
Templating engine	Helm chart is a tool used as a templating engine for files describing Kubernetes objects of the project.
Code quality assessment	SonarQube is a tool that delivers indicators about the code quality like the test coverage and potential bugs.

Table 4.2: Software environment

4.2 Technical choices

In this section, we define every major technological tool that was used for the project implementation, and why it was chosen along with a definition of some concepts that will be used for explanation.

4.2.1 Programming Language: Scala

Scala is a strong statically typed general-purpose JVM programming language[26]. Scalability is influenced by many factors, ranging from syntax details to component abstraction constructs. If we were forced to name just one aspect of Scala that helps scalability, though, we'd pick its combination of object-oriented and functional programming[27]. This is why we're choosing Scala. Specifically, this choice comes thanks to Scala's huge scaling capabilities when it comes to systems for handling big amounts of data, which is exactly what is needed in this project.

4.2.2 Data processing engine: Apache Spark

Batch processing is the process of handling transactions in a group or batch. There are ways for batch processing that can be done offline. As batch processing does not require extra help after it is begun, it is suitable for regularly used programs that may be performed with minimum human participation. Batch processing is distinct from transaction processing, which includes processing transactions one at a time and necessitates ongoing user interaction [9]. Note that systems that use batch processing are offline systems as well. Apache Spark is a unified engine designed for large-scale distributed data batch processing, on premises, in data centers, or in the cloud. Its design philosophy centers around four key characteristics:

- ✓Speed
- ✓Ease of use
- ✓Modularity
- ✓Extensibility

So, we are using Apache Spark because the goal of this internship is to build an offline system that can handle large amounts of data, Apache Spark is an excellent choice as a data processing engine.[28]

4.2.3 Container runtime: Docker

Docker is a free and open-source runtime for deploying and running software inside containers. On top of a virtualized container execution environment, it provides an application deployment engine[29]. A container runtime is always needed because developers are concerned with their programs running within containers, whereas operations team is concerned with container management. Although containerization and development will be done by

one person during this project, it is preferable to leave a project structure that can be handled by both developers and operations team for future re-usability of the project inside Cognira.

4.2.4 Container orchestration: Kubernetes

Kubernetes is a sophisticated open-source technology for containerized application management. It provides layers of abstracted concepts like clusters, pods, and persistent volumes that eases and standardizes the deployment planning and realization. It's made to manage distributed applications and services across a variety of platforms[30]. We are choosing Kubernetes because it manages everything through code, which is appropriate for this project because working with the deployment process is desired thanks to its capacity to contribute to the creation of reliable, consistent, and easily customizable environments[31].

4.3 Implementation of the calculation engine

In this section, we will describe the different aspects of the project implementation from building to deploying.

4.3.1 Input of the calculation engine

The input comes from three different files sources :

4.3.1.1 Formulas files

These are CSV files containing the formulas. Each file is associated with a specific promotion type. There is one extra file containing the formulas that will be computed along with each promotion regardless of its type. Figure 4.1 is a screenshot of the file containing equations associated with the "Buy X Get Y" :

tenant_id	vehicle	grouping	intersection_id	promo_type	equation_label	equation_json
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"max_buy_
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"min_buy_
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"max_get_
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"min_get_r
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"max_buy_
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"min_buy_
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"max_get_
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"min_get_c
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"sum_max
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"sum_min
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"sum_min
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"sum_max
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"sum_max
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"sum_min
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"sum_min
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"sum_max
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"max_disc
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"min_disc
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"max_disc
0	Coupon	promo_entry	promotion	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"min_disc
0	Coupon	promo_entry	promocoup	BXGY	label:"="	left_child:{"data_label:"@" left_child:{"data_label:"max_get

Figure 4.1: Formulas file example

4.3.1.2 Prices and costs file

This is a file that contains the prices and costs of different products depending on the store location and the date. Figure 4.2 showcases the price and costs file :

store_id	product_id	date	price	cost
STR_004	SKU_100008460	2020-11-01	1.6	1
STR_005	SKU_100008460	2020-11-01	1.23	1.05
STR_006	SKU_100008460	2020-11-01	1.7	1
STR_004	SKU_100009257	2020-11-01	1.44	0.9
STR_005	SKU_100009257	2020-11-01	1.21	0.7
STR_006	SKU_100009257	2020-11-01	1.28	0.6
STR_004	SKU_100009257	2020-11-01	1.44	0.9
STR_005	SKU_100009257	2020-11-01	1.21	0.7
STR_006	SKU_100009257	2020-11-01	1.28	0.6
STR_008	SKU_100009257	2020-11-01	1.28	0.6
STR_006	SKU_100009257	2020-11-01	1.28	0.6
STR_006	SKU_100009257	2023-11-01	1.28	0.6

Figure 4.2: Prices and costs file example

4.3.1.3 Promotions file

This file contains JSON strings. Each one of them represents a promotion. Figure 4.3 represents one of the JSON strings :

```

1 {
2   "tenant_id": 0,
3   "promo_start_day": 1,
4   "promo_start_month": 11,
5   "promo_start_year": 2020,
6   "promo_end_day": 30,
7   "promo_end_month": 1,
8   "promo_end_year": 2021,
9   "promo_type": "BXGY",
10  "promo_id": "74ab3675-a3f5-4af6-902e-493184e4cefe"
11  },
12  "store_ids": [
13    "STR_004",
14    "STR_005",
15    "STR_006"
16  ],
17  "products": [
18    "SKU_100009257_006",
19    "SKU_100009257_007",
20    "SKU_100008460_591"
21  ],
22  "equations_grouping": "promo_entry",
23  "vehicle": "Coupon",
24  "conditions": [

```

Figure 4.3: JSON promotion string

4.3.2 Used libraries

The first step to create the application was to choose which sbt libraries to work with. In table 4.3, we illustrate the main libraries used in the process of building the application :

Library	Purpose
Spark SQL	It is the main library to be used in the project. It will allow us to use Spark within the application
list JSON	It will parse JSON strings into instances of classes
Scalatest	It will be used for unit testing the application
Sbt pack	It is for packing the application into a distributable package and dependent JAR files
Scoverage	It is used for measuring the testing coverage

Table 4.3: The sbt libraries used in the project

4.3.3 Output of the calculation engine

The output consists of folders, where each folder represent the results of the computations associated with a given promotion. There are two CSV files for each promotion where one represents the indicators on the promotion dimension and the other represents the indicators on the promotion condition dimension. Figures 4.4 and 4.5 represent examples of the mentioned output files :

promotion	metric name	metric value
74ab3675-a3f5-4	sum_min_buy_re	1.21
74ab3675-a3f5-4	sum_min_get_co	1.8
74ab3675-a3f5-4	max_buy_cost	0.9
74ab3675-a3f5-4	sum_min_rewar	1.44
74ab3675-a3f5-4	sum_max_get_r	5.1
74ab3675-a3f5-4	min_get_regular	1.21
74ab3675-a3f5-4	sum_max_rewar	1.6
74ab3675-a3f5-4	min_discount_ar	3.63
74ab3675-a3f5-4	max_gross_marc	0.355556
74ab3675-a3f5-4	max_buy_regula	1.44
74ab3675-a3f5-4	min_discount_pe	0.715976
74ab3675-a3f5-4	sum_max_buy_r	1.44
74ab3675-a3f5-4	min_buy_cost	0.6
74ab3675-a3f5-4	min_gross_marc	0.504167

Figure 4.4: Indicators on the promotion dimension

promotion	condition	metric name	metric value
74ab3675-a3f5-4	get_1	min_discount_ar	3.63
74ab3675-a3f5-4	buy_1	min_regular_reta	1.21
74ab3675-a3f5-4	get_1	min_regular_reta	1.21
74ab3675-a3f5-4	buy_1	max_buy_regula	1.44
74ab3675-a3f5-4	get_1	max_get_regula	5.1
74ab3675-a3f5-4	buy_1	min_buy_prod_p	1.44
74ab3675-a3f5-4	buy_1	max_cost	0.9
74ab3675-a3f5-4	get_1	max_cost	1.05
74ab3675-a3f5-4	buy_1	max_buy_cost	0.9
74ab3675-a3f5-4	buy_1	quantity	1
74ab3675-a3f5-4	get_1	quantity	3
74ab3675-a3f5-4	buy_1	item_limit	1
74ab3675-a3f5-4	get_1	item_limit	1
74ab3675-a3f5-4	get_1	min_get_regular	3.63
74ab3675-a3f5-4	get_1	max_discount_a	5.1
74ab3675-a3f5-4	get_1	max_get_cost	3.15
74ab3675-a3f5-4	buy_1	min_buy_regula	1.21
74ab3675-a3f5-4	get_1	max_reward_pro	1.6

Figure 4.5: Indicators on the promotion condition dimension

4.3.4 Packaging the calculation engine

The application is packaged into executable JAR files through the sbt-pack library. It generates in /target/pack/bin directory a shell file called Mainshell that uses a JAR file called calc-engine-spark-counter-part_2.12-0.1.0-SNAPSHOT.jar that is located in /target/pack/lib directory. This JAR file itself uses other JAR files located in the same directory. These files, called 'program launch scripts', are used to run the application. This is done to be able to create a Docker image with a low size. This makes the application more performant. Otherwise, the Docker file will be loaded with the application's files as well as other dependencies.

4.3.5 Deploying the calculation engine

After the packaging process, we put the JAR files and the shell file in a deployable Docker image. This is made possible using a Dockerfile. For deploying the application on Kubernetes, we prepared a helm chart containing a persistent volume template, a persistent volume claim template, a job template, and the file containing the values to insert within the mentioned

templates. Figures 4.6 and 4.7 show the different instances of Kubernetes objects that were displayed on the minikube dashboard:

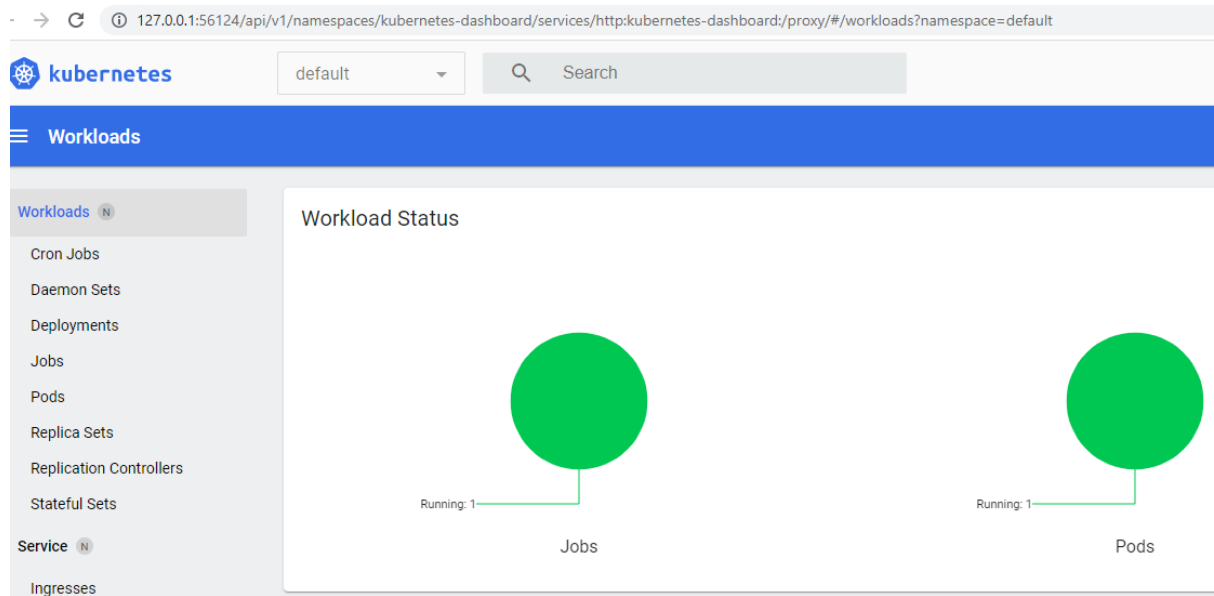


Figure 4.6: The job on minikube dashboard

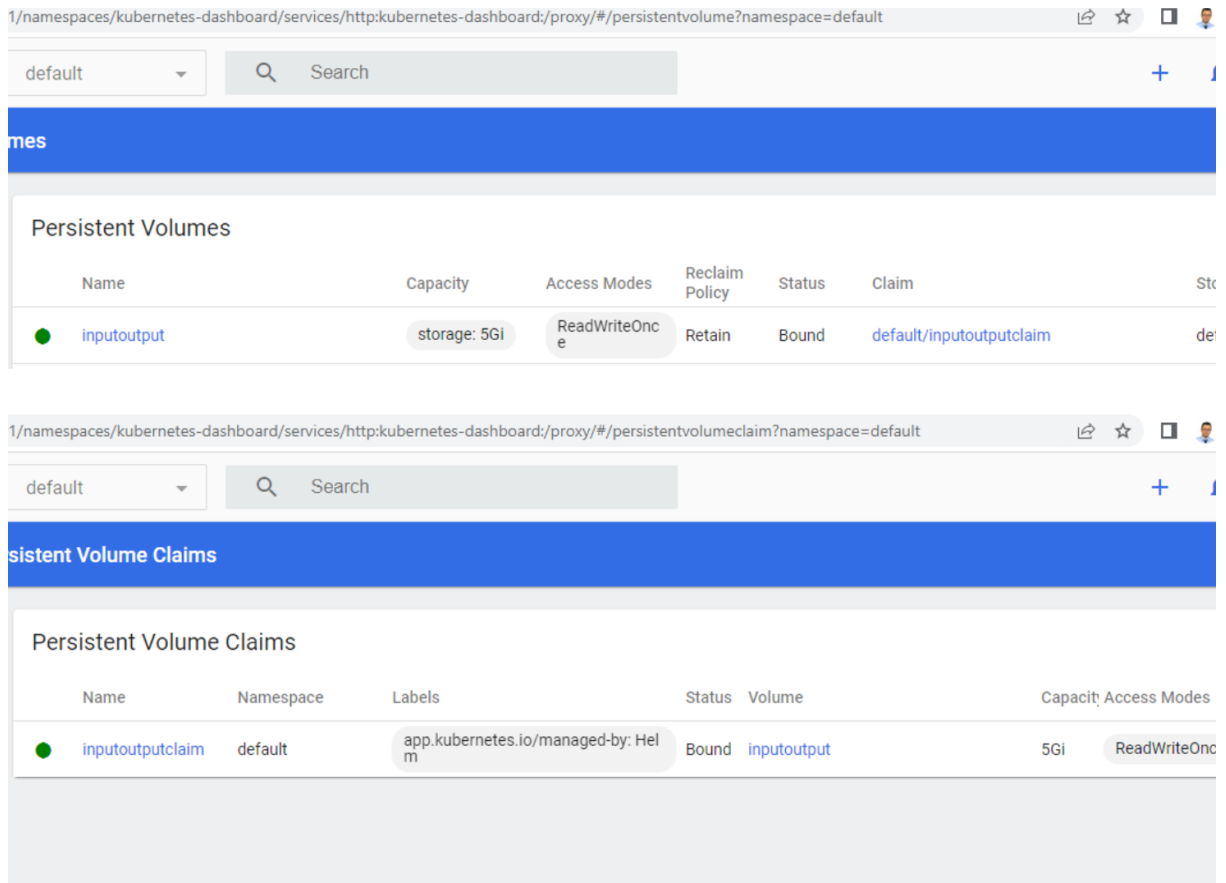


Figure 4.7: PV and PVC on minikube dashboard

4.4 Testing the calculation engine

The testing phase is crucial for ensuring the good quality of the final result. A series of unit tests were made in order to verify different edge cases of the application. Also, a code quality assessment was made using a tool just for that.

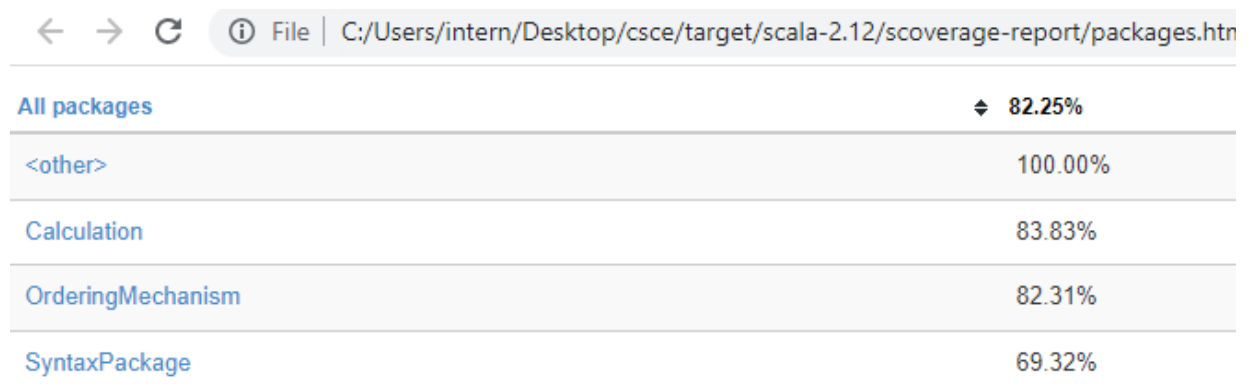
4.4.1 Unit testing

To ensure that our code works properly, we use unit testing, which is a technique of ensuring that one element of the program works correctly. Unit tests protect the developed application from potential regressions that might occur as a result of a change in the code or the dependencies. After making any changes to the code, all unit tests are run to ensure that the changes made to the code haven't harmed the functionality of other modules. The table 4.4 represents the different unit tests created during the project :

File name	Components to test	Number of tests
ParseTreeToCodeTest.sc	- Equation - Operator - Operation - LeafVariable - Indicator	6
WholePromotionTest.sc	- PromotionToCompute - Filter	12
EquationsComputationsTest.sc	- OrderingMechanism - ContextElement	32
InputOutputTest.sc	- TransPromotion - TransFilter	7

Table 4.4: The unit tests

The unit testing gave a code coverage of 82.25%. We got this percentage because the ParseTree component was already tested by Cognira’s backend team. Therefore, it wasn’t necessary to conduct the testing on the logic of the parse tree since it was taken as a postulate. The tool used for measuring the code coverage is called scoverage[32]. It has the advantage of generating a report containing details about the coverage. Figure 4.8 is the HTML file generated by scoverage displaying the report. It shows the percentage of test coverage of each folder. Note that the Calculation folder contains the Equation, Operation, and LeafVariable classes. The OrderingMechanism folder contains the OrderingMechanism class, and the SyntaxPackage folder contains the rest of the classes :



← → ↻ ⓘ File C:/Users/intern/Desktop/csce/target/scala-2.12/scoverage-report/packages.htm	
All packages	82.25%
<other>	100.00%
Calculation	83.83%
OrderingMechanism	82.31%
SyntaxPackage	69.32%

Figure 4.8: Coverage percentage of each folder

Figure 4.9 shows a report generated as an HTML page by `scoverage`. It contains information about the unit tests like the number of invoked statements and code coverage :


Lines of code:	1957	Files:	14	Classes:	14	Methods:	114
Lines per file:	139.79	Packages:	4	Classes per package:	3.50	Methods per class:	8.14
Total statements:	2631	Invoked statements:	2164	Total branches:	306	Invoked branches:	243
Ignored statements:	0						
Statement coverage:	82.25 %						

Figure 4.9: `Scoverage` coverage report

4.4.2 SonarQube analysis

In order to get a better quality assessment of the code, a tool called `SonarQube`[33] is used. It is for detecting potential bugs, analyzing security vulnerabilities, measuring the code replication percentage, and showing the code coverage. Figure 4.10 is the screenshot of the report provided by `SonarQube` about our application. Note that there is a slight difference in code coverage percentage because `SonarQube` has a slight problem in reading generated test files of `scoverage` :

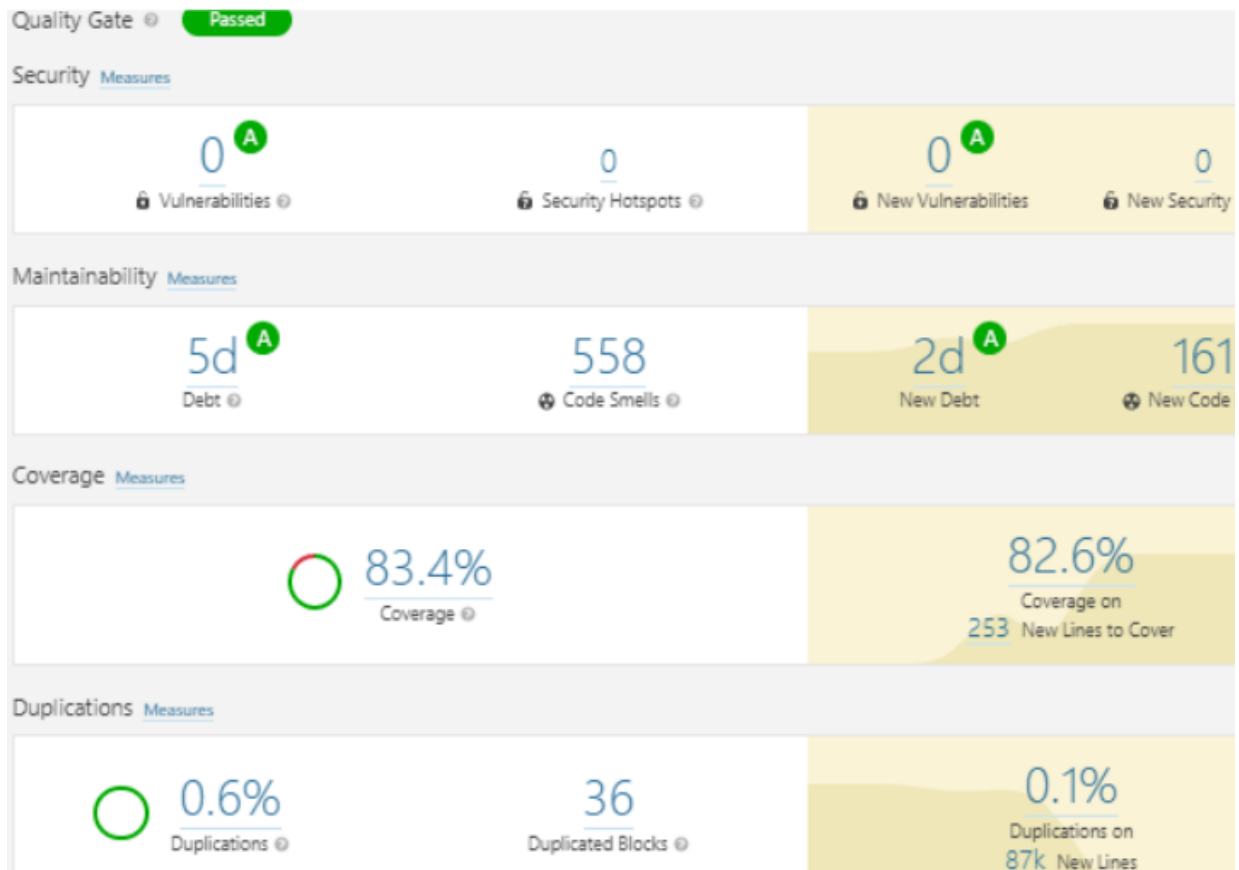


Figure 4.10: SonarQube report

4.5 Additional work

As the required work of the project was done earlier than expected, two additional parts were added to the project, which were a CI/CD pipeline and a monitoring dashboard.

4.5.1 CI/CD pipeline

4.5.1.1 Definition

Continuous integration (CI) is a software development method in which developers integrate their code changes into one central repository on periodic basis, followed by automated builds and testing. Continuous Delivery (CD) is a software development method in which code changes are continuously created, tested, and prepared for production.[34]

4.5.1.2 Added value

A CI/CD pipeline would help to automate the deployment or the delivery of a given software product. It is necessary for backend engineers to have an idea on how it works and what does it take to build it. Therefore, it was concluded that constructing a pipeline would be a good addition to the project.

4.5.1.3 CI/CD pipeline implementation

The pipeline was constructed using Jenkins which is an automation server[35].The pipeline contained six stages. They are explained in table 4.5 :

Stage name	Stage description
SCM Checkout	It checks out from which source the file containing pipeline details called "Jenkinsfile" is taken
Unit Testing	All the unit tests run. Once they finish, the code coverage is measured.
SonarQube Analysis	A new report is sent to SonarQube detailing things like new potential bugs and security vulnerabilities.
Packing	The application is packed into JAR files and a shell file.
Dockerization	The packed application is transformed into a Docker image.
Kubernetes Deployment	The old helm chart is deleted and a new one is created .

Table 4.5: Pipeline stages

The figure 4.11 displays the pipeline on the Jenkins user interface after running successfully:

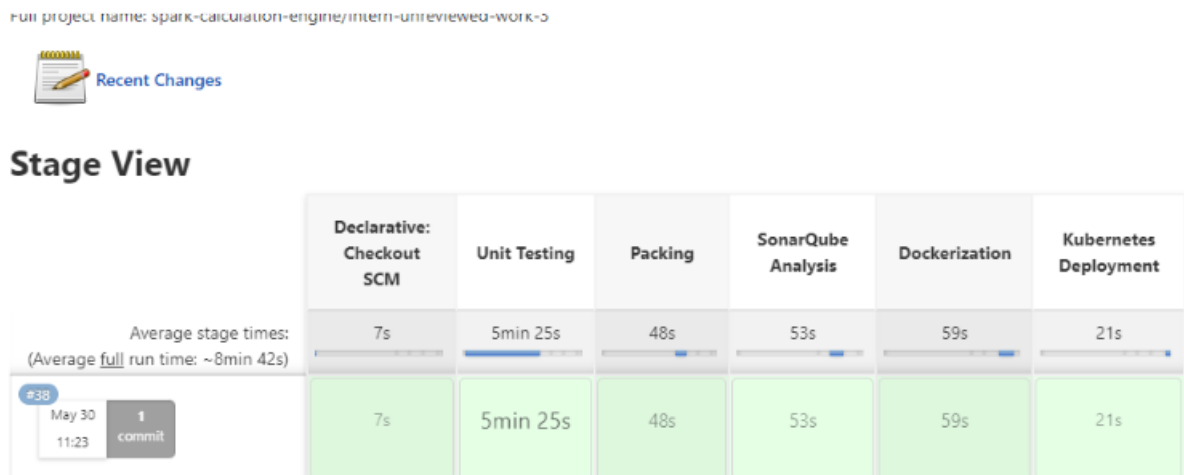


Figure 4.11: Pipeline on Jenkins user interface

4.5.2 Monitoring dashboard

4.5.2.1 Definition

DevOps' goal is to keep track of the full development cycle, from strategy to development, integration, testing, deployment, and operations. This is done via monitoring. It gives you a detailed image about the condition of your applications, services, and production infrastructure in real time.[36]

4.5.2.2 Added value

Using a batch processing system save the time and effort of managers and other important staff. This gives them more capacity to focus on their own tasks rather than managing batches. When difficulties occur, alerts are sent out[37]. An important part of IT companies' work relies on the infrastructure that they have. It is important to see how the infrastructure is being used in order to avoid any stress on the existing resources. This is why monitoring is such an important thing to tackle.

4.5.2.3 Monitoring implementation

In order to monitor the minikube cluster containing the project's resources, Prometheus, a tool to scrape metrics and KPIs from systems, was used to collect real-time metrics on the cluster such as CPU usage, memory usage, and network input/output pressure. Grafana was used to visualise these metrics. The monitoring dashboard that is presented in the figure 4.12 was built to monitor the mentioned metrics :

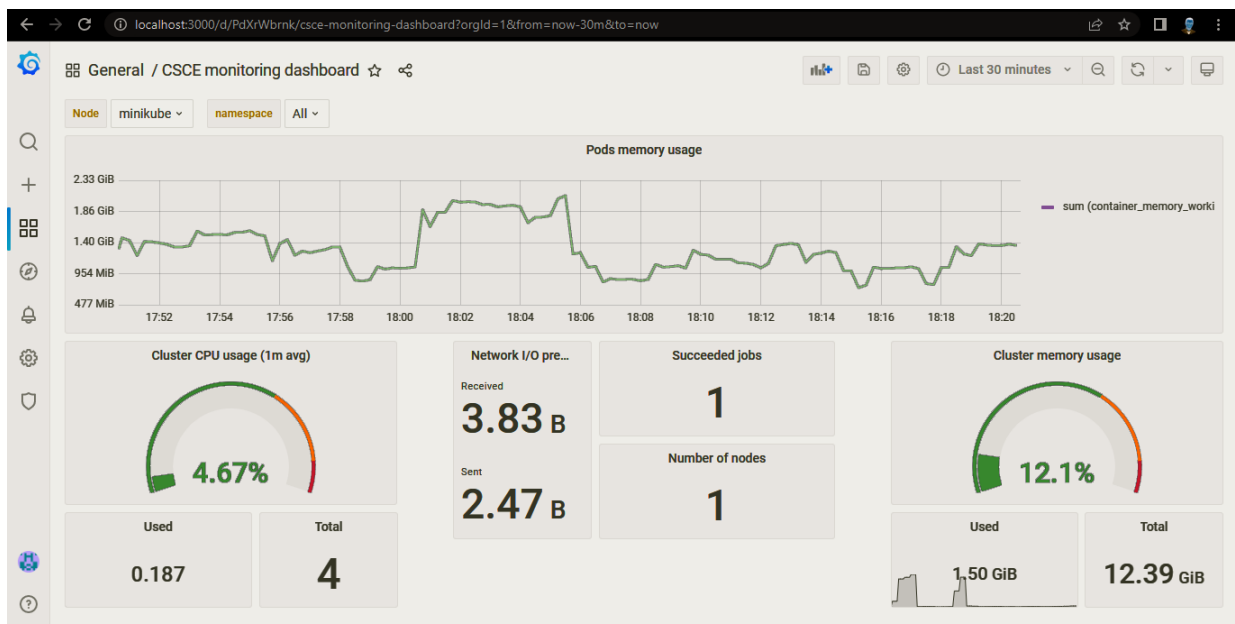
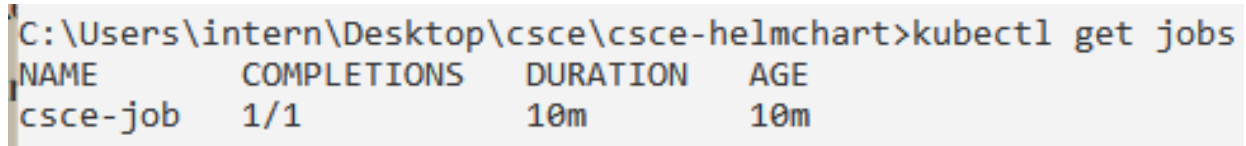


Figure 4.12: Monitoring dashboard

4.6 Validation

Cognira's analytics analysis team leader and the internship supervisors checked the solution to ensure that it delivers correct computations and that it has the accuracy of 6 digits after comma. The supervisors validated the solution by putting a prices dataset of 1.2 Million duplicated rows to compute the indicators of 6 promotions. The job finished processing this amount of data in 10 minutes when running it on a minikube environment with 4GB of RAM. The figure 4.13 displays the completion of the job in the minikube environment:

A terminal window showing the command 'kubectl get jobs' and its output. The output is a table with four columns: NAME, COMPLETIONS, DURATION, and AGE. The first row shows 'csce-job' with '1/1' completions, a duration of '10m', and an age of '10m'.

```
C:\Users\intern\Desktop\csce\csce-helmchart>kubectl get jobs
NAME                COMPLETIONS  DURATION  AGE
csce-job            1/1          10m      10m
```

Figure 4.13: Job completion on minikube

4.7 conclusion

In this chapter, we've defined the technical environment as well as the tools that are used throughout the project. In addition, we talked about the project implementation, the testing that occurred, and the additional work that was done once the mandatory work was completed.

General conclusion

This project, named Design and Development of a Configurable Calculation Engine for metrics evaluation about retail promotions, was conducted to obtain an offline version of Cognira's calculation engine. The developed application allows of calculate indicators about promotions. It takes details about promotions as input, as well as data related to products such as the prices and costs, and optionally equations in JSON format. The input comes in CSV files. The engine is a batch job that gets started by the promotion planning service. After acquiring promotions, prices, costs, and formulas as input, it turns them into structured data following a defined design. One main distinctive difference between the latter and the one already made by Cognira is that this one has the composite design pattern in it which provides structural flexibility upon transforming the formulas into structured data. This makes different components of the equations more manageable through classes and interfaces. As the formulas depend on each other, the engine creates an ordering mechanism to compute each formula exactly once, where an indicator is computed when its dependent indicators are already calculated. Afterwards, the computation takes place by going through the computed order. Finally, the results of the computations are provided in CSV files once they are ready.

Being a batch processing system is the main advantage of this version of the calculation engine. This is because it is capable of processing a significantly large data volumes without online constraints. Another advantage for this engine is its easy deployment on any infrastructure. The work has been validated as a PoC by the company managers. Even though the proposed solution is scalable, it is not auto-scalable. Auto-scaling is a technique that allows dynamic resources allocation for applications depending on their need. Also, the developed monitoring dashboard only monitors CPU usage, memory usage, and network pressure. This only gives an idea on the effect of the batch job on the infrastructure, not about the job itself. Also, the job generates a large amount of logs that are not utilized.

As a perspective, a strategy called 'vertical auto-scaling' can be done to auto-scale the application. Vertical auto-scaling is a way to modify the resources allocation to the pod associated with the job. Also, log-based monitoring, can be done by acquiring the logs generated by the batch jobs and analyzing them afterwards. This would help the development team to understand the data transformations that happen underneath the engine in order

to fix the application source code or to optimize it further. One proposed stack to be used for the log analysis is Logstash to process the data before storage, Elasticsearch to store the processed data coming from the logs, and Kibana to visualize them.

Bibliography

- [1] G. Blokdyk. *Project Charter: The Definitive Handbook*. South Carolina, United States: CreateSpace Independent, 2017.
- [2] E. Koester. *Inc. 5000*. <https://www.inc.com/inc5000/2019>. Accessed: 2022-03-27.
- [3] Amanda Hilgers. *Cognira website: promo management*. <https://cognira.com/promotion-management-solution-for-retail/>. Accessed: 2022-03-27.
- [4] A. Hilgers. *Cognira: FaaS*. <https://cognira.com/forecast-as-a-service/>. Accessed: 2022-03-27.
- [5] A. Hilgers. *Cognira: Assortment Allocation*. <https://cognira.com/assortment-allocation/>. Accessed: 2022-03-27.
- [6] C. Norris. *SAP Promotion Calculation Engine*. <https://help.sap.com/doc/7c431b12d61e41c4a5fc08a4ee80d8c0/5.0/en-US/SDKPromotionCalculationEngine.pdf>. Accessed: 2022-05-19.
- [7] H. Le Lievre. *IBM Promotion Calculation Engine*. <https://www.ibm.com/docs/en/order-management-sw/9.5.0?topic=so-overview-pricing-promotion-process-flow>. Accessed: 2022-05-19.
- [8] A. Pang. *SAP retail software market share*. <https://www.appsruntheworld.com/top-10-retail-software-vendors-and-market-forecast/>. Accessed: 2022-05-25.
- [9] A. Barone. *Batch processing definition*. <https://www.investopedia.com/terms/b/batch-processing.asp>. Accessed: 2022-05-07.
- [10] I. Nuinaja. *Scrum methodology definition*. <https://www.digite.com/agile/scrum-methodology>. Accessed: 2022-05-03.
- [11] L. Quartie. *Scrum visualized*. <https://powerslides.com/powerpoint-business/project-management-templates/agile-scrum-process/>. Accessed: 2022-05-03.
- [12] J. Sutherland. *Scrum master definition*. <https://www.atlassian.com/agile/scrum/scrum-master>. Accessed: 2022-05-03.
- [13] V. Singh. *Development team definition*. <https://www.toolsqa.com/agile/scrum/scrum-development-team/>. Reviewer: Sharma,Lakshay , Accessed: 2022-05-03.
- [14] D. Leffingwell. *Product owner definition*. <https://www.scaledagileframework.com/product-owner/>. Accessed: 2022-05-03.

- [15] R. Ming. *Sprint review definition*. <https://www.scrum.org/resources/what-is-a-sprint-review>. Accessed: 2022-05-03.
- [16] R. Ming. *Daily scrum standup definition*. <https://www.scrum.org/resources/what-is-a-daily-scrum>. Accessed: 2022-05-03.
- [17] O. Jainai. *Activity diagram of code review process*. <https://svitla.com/blog/code-review-and-its-important-role-in-software-development>. Accessed: 2022-05-18.
- [18] S.A. Mubarak. In: *Bar Gantt Charts*. 2010. URL: https://www.researchgate.net/publication/319359193_BarGanttCharts.
- [19] A. Regata. *Definition of JIRA epic*. <https://www.javatpoint.com/jira-epic>. Accessed: 2022-05-30.
- [20] K. Hamilton. *Learning UML 2.0*. VORT Yotsuyasakamachi Bldg. 1F, Tokyo, Japan: Oreilly, 2006. URL: <https://www.oreilly.com/library/view/learning-uml-20/0596009828/ch01.html>.
- [21] S. Jain. *Parse Tree*. <https://www.geeksforgeeks.org/parse-tree-in-compiler-design/>. Accessed: 2022-04-18.
- [22] R. Vincent. *Parse Tree example*. <https://stackoverflow.com/questions/24520841/how-to-build-a-parse-tree-of-a-mathematical-expression>. Accessed: 2022-04-18.
- [23] R. Lafore. *Object-oriented programming in C++*. A-27, 2nd Floor, Mohan Co-operative Industrial Estate, New Delhi, India: Vikas Publishing House, 2001.
- [24] F. Vallée. *UML 2 in action*. Livery Place, 35 Livery Street, Birmingham B3 2PB, United Kingdom: Packt Publishing, 2005. URL: <https://doc.lagout.org/programmation/Databases/SQL/UML.pdf>.
- [25] J. Osis. *Topological UML modeling: an improved approach for domain analysis and software development*. 1C, Brouwersgracht, Amsterdam, The Netherlands: Elsevier, 2017.
- [26] M. Odersky. *Scala definition*. [https://en.wikipedia.org/wiki/Scala_\(programming_language\)](https://en.wikipedia.org/wiki/Scala_(programming_language)). Accessed: 2022-05-07.
- [27] B. Venners. *Programming in Scala*. 2070 N Broadway Unit 305, California, United States: Artima Press, 2004.
- [28] M. Armbrust. *Learning Spark : Lightning-Fast Data Analytics*. VORT Yotsuyasakamachi Bldg. 1F, Tokyo, Japan: O'Reilly Media, 2020.
- [29] J. Turnbull. *The Docker Book*. 252, Astoria, Brooklyn, NY, United States: Turnbull Press, 2017.

- [30] J. Ellingwood. *Kubernetes for full-stack developers*. Hudson Square - 101 Avenue, NY, United States: DigitalOcean, 2020. URL: <https://assets.digitalocean.com/books/kubernetes-for-full-stack-developers.pdf>.
- [31] M. Weinberg. *Definition of cloud agnostic*. <https://www.cloudzero.com/blog/cloud-agnostic>. Accessed: 2022-05-08.
- [32] R. Delsalle. *Scoverage definition*. <https://github.com/scoverage/sbt-scoverage>. Accessed: 2022-06-02.
- [33] F. Langen. *SonarQube definition*. <https://docs.sonarqube.org/latest/user-guide/concepts/>. Accessed: 2022-06-02.
- [34] A. Thakkar A. Khan. *Practicing Continuous Integration and Continuous Delivery on AWS*. AWS, 2017.
- [35] A. Riglian. *Jenkins definition*. <https://www.techtarget.com/searchsoftwarequality/definition/Jenkins>. Accessed: 2022-06-02.
- [36] J. Sutherland. *Monitoring definition*. <https://www.atlassian.com/devops/devops-tools/devops-monitoring>. Accessed: 2022-06-02.
- [37] A. Barone. *Monitoring for batch processing*. <https://www.investopedia.com/terms/b/batch-processing.asp>. Accessed: 2022-06-06.