

Optimization Framework with Spring Boot Java and MiniZinc

Mariia Muzykantova

Bachelor's Thesis
Degree Programme in Business
Information Technology
2019



Author(s) Mariia Muzykantova	
Degree programme BIT	
Report/thesis title Optimization Framework with Spring Boot Java and MiniZinc	Number of pages and appendix pages 47
<p>Optimization is the process that we enter into a relationship with every day. We are constantly facing and trying to solve optimization problems; whether we are planning a trip abroad or buying a car. In the case of companies, optimization problems are more complex and more time-consuming, e.g. find the optimized location for a radio-based station receiver considering there are several data centers with different latency and capacity.</p> <p>The main goal of this thesis is to study the optimization process and produce practical results through the development of an optimization engine for the company Nokia. The idea of an optimization framework is to fully automate a complex calculation in the case of a constraint optimization problem.</p> <p>The application was defined by a team of Nokia SW architects and required getting a result within 400 hours by using Scrum methodology. The work started with learning the terminology and an investigation of the tools to be used.</p> <p>The theoretical framework was composed by focusing on studying the optimization process overall, making highlighting the constraint optimization type. The theory part covers also MiniZinc optimization language and its work as a tool at the application and client interface. During the work on the optimization engine, the use case was specified, the API definition was defined, the asynchronous running process was implemented, and the database connection was established.</p> <p>The project took place in the fall 2019. Despite the obstacles encountered, the project was completed according to the schedule. Scrum methodology was used to achieve the goal. This helped to keep the work in order. Among other methods employed, there were also lectures and brainstorming led by the team of Nokia architects.</p>	
Keywords Optimization, MiniZinc, Models, Cassandra, Postman, Swagger, API, Gerrit	

Table of Contents

1	Introduction	1
2	Theoretical background	3
2.1	Spring Boot	3
2.2	Gerrit	3
2.2.1	History	3
2.2.2	What is Gerrit?	4
2.3	Optimization framework	5
2.3.1	Optimization	5
2.3.2	Constraint optimization	7
2.3.3	Optimization in the Networks	7
2.4	ONAP	8
2.5	MiniZinc	9
2.6	Database	11
2.6.1	SQL	11
2.6.2	NoSQL	12
2.6.3	Cassandra	12
2.7	API	13
2.7.1	REST API	14
2.7.2	Swagger	14
2.7.3	Postman	15
2.8	Docker	17
2.9	Network terminology	18
3	Design and planning	20
3.1	Project architecture	20
3.2	Use case	22
3.3	Database	25
3.4	Version Control System	26
3.5	API design	26
3.5.1	Model	27
3.5.2	Model/{modelName}	28
3.5.3	Task	28
3.5.4	Task/{taskId}	30
3.5.5	API documentation	30
4	Project Implementation	34
4.1	Prerequisites	34
4.2	Setting up the environment	35

4.2.1	MiniZinc IDE.....	35
4.2.2	Docker	35
4.2.3	Cassandra database	36
4.3	Running the project.....	39
5	Discussion.....	40
5.1	Review.....	40
5.2	Challenges and solutions	41
5.3	Future development.....	42
6	Bibliography	43
7	Table of figures	46

ACKNOWLEDGMENT

I would like to express my special thanks of gratitude to my family. The reason for what I became today. I am grateful to my husband for his continuous care, patience, and encouragement.

I would like also to extend my gratitude to my thesis advisor Kasper Valtakari for his guidance and great support in completing the Bachelor thesis. Thank you for his time, suggestions and feedback.

Symbols and abbreviations

5G	Fifth generation
AOP	Aspect-Oriented Programming
BSD	Berkeley Software Distribution
CMSO	Change Management Scheduling Optimizer
CU-UP	Centralized Unit User Plane
EPL	Eclipse Public License
gNB	next-generation NodeB
HAS	Homing and Allocation Service
IoT	The Internet of Things
JDBC	Java Database Connectivity
MPL	Mozilla Public License
MVC	Model View Controller
NFVI-PoP	Network Function Virtualization Infrastructure Point of Presence
NoSQL	No Structured Query Language
OF	Optimization Framework
ONAP	Open Network Automation Platform
OOF	ONAP Optimization Framework
OSDF	ONAP Optimization Service Design Framework
PaVaROt	Parameter Value Resolving and Optimization
PCI	Physical Cell ID
POJO	Plain Old Java Object
Rest	Representational State Transfer
SDN	Software Defined Networking
SQL	Structured Query Language
VM	Virtual Machine
VNF	Virtual Network Function

1 Introduction

A team at Nokia SW set the problem of creating an optimization framework (OF) that would meet the company's needs and have enough expression power to provide solution options to complex problems and increase service delivery. In the first stage, the following main issues were set to identify the next steps and finally get this project completed:

- Investigate the Open Networking Automation Platform (ONAP)
- Decide whether to utilize the ONAP optimization framework (OOF) or create the company's own OF following similar logic.

The first step of this study was to analyze open source ONAP code and match the possible use of OOF for our own needs. The main role of OOF is to solve optimization problems. Moreover, each problem in OOF is solved differently and depends on the use case by different sub-components. Sub-components are not covered in this study as they are case-specific to ONAP use.

After the examination of OOF open-source code, it was learned that the OOF framework is written in Python and contains a MiniZinc library. OOF utilizes the open-source project MiniZinc, which has a solver-independent modeling language and has interfaces to various open source and commercial solvers. A solver-independent language indicates a high-level of abstraction in the problem specification. In other words, MiniZinc supports over two dozen of solvers. Where a solver is a tool which contains constraint solving modules. It runs the process of solving a system of constraints. MiniZinc lets the user define a constraint problem with their own model and solve the optimization issue using different solvers. Taking this into account, Nokia's team of architects decided to investigate MiniZinc further.

Finally, having learned the ONAP code, it was discovered that we had to create our own framework/engine that we would call Parameter Value Resolving and Optimization (PaVaROt). The new level of framework enables complex problem resolution using a configuration rather than the implementation process. In other words, the framework is needed to automate and simplify the computing process by avoiding the logic of coding customs for each use case.

With PaVaROt, we precisely define a model, input the attributes to be used, provide configuration for the engine, and the rest of the code will execute the program with given parameters. The result should print out for us the optimization capability as a service, such as minimize or maximize metrics. The example of a problem to be solved can be 5G gNB

placement optimization. Where we want to find the best location by minimizing the cost and maximizing the customer satisfaction. We devote more time to this example in Chapter 3.2 “Use case”, going deeper into details.

After examination of ONAP we planned to use MiniZinc as the basis of the optimization engine in our PaVaROt project. In addition to this, the statement was made to use Java programming language, while OOF uses Python.

The main advantages of this project are:

- An automated system that integrates with MiniZinc modeling language in runtime
- The client gets an output as it is after all the complex calculations done by the program
- The defined API definition simplifies the process of posting and getting needed model files, data files, and input and output parameters
- The client can follow the status of the running process.

We create a comprehensive platform for automation of the calculations by having certain data, the defined model, and its use cases. To understand the project concept, the study contains a theoretical framework, a list of abbreviations, and illustrations. The reader is also provided with a use case to draw a clear picture of the constraint optimization use in the Network lifecycle. The thesis provides the reader with the project implementation steps, including design, planning, and setting information.

2 Theoretical background

This section of the study covers the definition of tools, software, and languages. The chapter concentrates on describing the main terminology that will help the reader understand the project work better and put the acquired knowledge into practice. First, we will familiarize ourselves with the programming language, its framework, editor, and other details to demonstrate the advantages of those over different technologies. Second, we will look at the code controlling tool, which is also used for team collaboration. Third, we analyze the definition of optimization, its types, why and where we use it, and how we use optimization in the project. In addition, we take advantage of using optimization in a network system. Finally, we parse a database, interfaces, and platform we used in our project.

2.1 Spring Boot

Spring boot is a Java-based framework extensively used by developers. Spring Boot simplifies coding by using “sprint-boot-starter-web” / ”sprint-boot-starter-jdbc” dependencies to generate the project for us. Spring Boot is the system that allows you to reach all business purposes. Spring Boot’s functionalities include the following:

- POJO
- Dependency injection
- MVC
- Data
- Rest
- AOP
- Security etc.

With Spring Boot, it is easier to generate production-ready software. All the standard configurations are conducted by Spring Boot only if a developer needs their own features, they need to create some small configuration, such as in “application: properties” file. Spring Boot supports and provides the embedded server (Tomcat, Weblogic, Websphere, etc.), which means it makes the server act as a part of the application (Walls, 2016).

2.2 Gerrit

2.2.1 History

Gerrit code review was created back in 2008 to support the Android Open Source projects. Since that time Gerrit became popular with enterprise companies. Gerrit has the license of Apache 2.0 and 93 included dependencies (mostly Apache 2.0, BSD, MPL1.1,

EPL). Originally it was written in Python and later was recast into Java and SQL. (Gerrit-Forge Blog, 2017)

2.2.2 What is Gerrit?

Gerrit is a web-based code review tool, which purpose is to make the code cleaner and better to read. Gerrit is the team code collaboration system that by the cooperation of several developers makes the developing less demanding and improves share project inventions. Reviewing and learning the code of other developers one upgrade the programming skills. (Milanesio, 2013)

Gerrit features:

- Reads code
- Enable a user to leave the comments on the code and every file that was pushed for review
- Acts as a repository, both a developer and reviews can go back and check previous code view, commits and changes

The example of how the review commit looks like in Gerrit is demonstrated in Figure 1, where area marked in green is the code was added since the last patch of the project and a yellow box shows the commit message with the information of who made this commit, what time and the content of the commit. (Milanesio, 2013)

```
63 private static final long serialVersionUID = 1L;
64 private static final Logger log = LoggerFactory.getLogger(LogServlet.class);
65
66 private static String getTemplateName(GitilesAccess access) throws IOException {
67     return Objects.firstNonNull(
68         access.getConfig().getString("command", "log", "soyTemplate"),
69         "gitiles.logDetail");
70 }
71
72 static final String LIMIT_PARAM = "n";
73 static final String START_PARAM = "s";
74 private static final int DEFAULT_LIMIT = 100;
75 private static final int MAX_LIMIT = 10000;
76
77 private final GitilesAccess.Factory accessFactory;
78 private final Linkifier linkifier;
79
80 public LogServlet(GitilesAccess.Factory accessFactory, Renderer renderer, Linkifier linkifier)
81     super(renderer);
```

	Shawn Pearce	We should get logDetail worked into this somehow. logDetail for the ...	03/10 17:34
	Stefan Zager	I defer to your judgement. Changing the subsection to "logDetail" me...	03/11 21:32

Figure 1. Gerrit workspace sample (Gerrit, 2019a)

Additionally, to the above properties, with Gerrit, a user can select the people that will review the project, assign the project to yourself or someone else. One can also specify the branches that others are allowed to see, fetch or clone from the repository. This makes it good for working in environments where one needs to hide a security branch that has a

bug fix in it until one has a release binary available to hand out to the customers or when a developer does not want people to see a certain experiment that he/she is working on. Another use case can be a situation when a software engineer collaborates with different partner companies and he/she cannot share code with all of them. (Milanesio, 2013)

Figure 2 demonstrates the easy way to find the created repository.

S	Project Name	Project Description
Q	Public-Plugins	Parent project for plugins*
Q	Public-Projects	Parent project enabling Anonymous Users read
Q	bucklets	Reusable building blocks for Buck build system.
Q	executablewar	Support for running code directly from WAR files
▶	gcompute-tools	Tools for Google Compute Engine
Q	gerrit	Gerrit Code Review
Q	gerrit-attic	Aborted experiments and ancient revisions of Gerrit Code Review
Q	gerrit-ci-scripts	Scripts used for continuous integration builds of Gerrit
Q	gerrit-installer	Gerrit native installation packages for Windows, Linux and Mac OSX
Q	git-repo	repo - The Multiple Git Repository Tool
Q	gitiles	A simple browser for Git repositories
Q	gs-maven-wagon	Maven wagon provider for Google Storage for Developers. Forked from https:
Q	gwtexpui	Extended UI tools for GWT. This repository is deprecated. The code has been
Q	gwtjsonrpc	JSON-RPC for Google Web Toolkit (GWT)
Q	gwtorm	Tiny ORM
Q	plugin-builder	Builder for buck-based plugins.
Q	plugins/admin-console	Provides information via SSH commands to Gerrit Administrators.
Q	plugins/approval-extension	Small example plug-in demoing extension points for manipulating approvals.

Figure 2. Gerrit repositories filter (Gerrit, 2019b)

Gerrit allows you to create a central repository for your team at one location that everybody has access to your code from code reviews or work at the commit level, not at a branch level, the way of GitHub pull request does. For example, if a developer creates five new commits locally and sends those to Gerrit code review, these five commits become five code reviews. Besides, Gerrit provides a search function and dashboards are derives from search features to let you see what changes are available for code review or what is still being worked on. This is an important view into your overall project, particularly when you have more than one git repository pull requests. (Milanesio, 2013)

2.3 Optimization framework

2.3.1 Optimization

Optimization is the mathematical process that using the idea of extrema, finding a relative or local maximum or minimum to solve real-world problems. In other words, the optimization operation is used to maximize or minimize, optimize certain things. Optimization chooses inputs that will result in the best possible outputs or making things the best that

they can be. Optimization is the core process in many social, business, economic, construction, and everyday personal decision making. Let us look at a very small sampling of the fields that make use of optimization. (Kallrath, 2004)

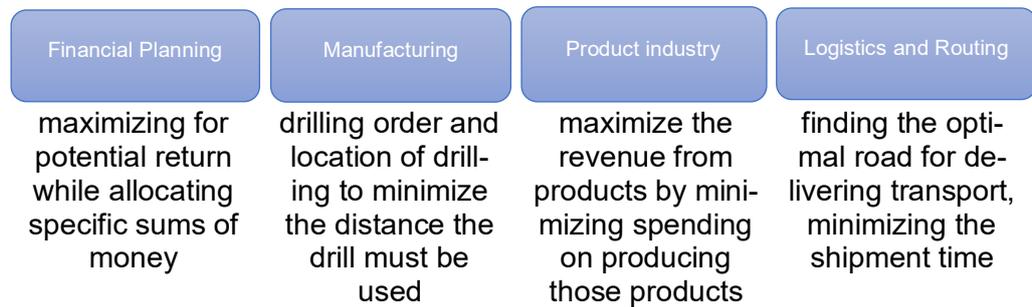


Figure 3. Optimization examples in various fields of everyday activity

Optimization problem, in this case, linear regression that has only two parameters: slope and intercept, can look like in Figure 4. Most people might see similar samples from school or university programs. This sample demonstrates the best fit line which tells you the theory of the phenomenon and the line should be close to all the experimental points. In other words, the line should be as close as possible to all the points. (Piyush, 2016)

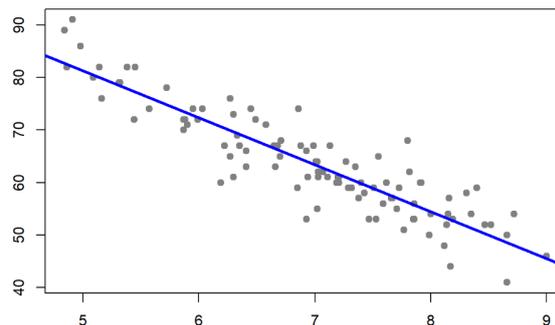


Figure 4. Regression analysis (Navarro, 2019)

Optimization can be a deciding on the most effective allocation of available resources to producing a design with the best characteristics to choosing control variables that will cause a system to behave as desired. Optimization is useful when there are limits or constraints on the resources involved or boundaries restricting the possible solutions. (Piyush, 2016)

Scientists identify the following types of optimization:

- Continuous
- Discrete
- Unconstraint

- **Constraint**
- Deterministic
- Stochastic (Neos Guide, 2019)

2.3.2 Constraint optimization

Constraint describes where the optimizer cannot go or additional conditions that must be met for a successful solution. If we make an effort to maximize the objective function in our typical life situation, classic constraints can involve a limited amount of money, time and other resources. Returning to Figure 3 “Optimization examples in various fields of everyday activity”, we might see familiar constraint problems. In other words, the key to constraint optimization is the presence of limitations. (Edelkamp, 2012)

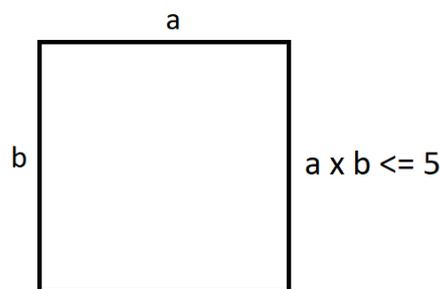


Figure 5. Constraint optimization in Math

Additionally, constraint optimization can be interpreted as the following sample: Optimize the size of the square, while the length of two sides multiplied together is less or equal 5. Where “ $a \times b$ ” is linear inequality and “5” is boundary (it represents the limits of the area).

2.3.3 Optimization in the Networks

The design and synthesis of the Network is always a relevant topic. Usually, we even do not think about how we get the Internet connection or how we get a call from someone and what stays behind it. There are a lot of research, calculations and measuring are done every time before to get the framework working. Not a long time ago humanity developed optical networks, the Internet and wireless networks, where the last generation of cellular networks is 5G. Network has become increasingly complicated in recent years. Many features have been developed to help teleoperators simplify the operations, reduce the costs and maximize the potential of the revenues. Here is where we start to talk about the optimization framework (OF). (ScienceDirect, 2018)

These newly developed technologies create new optimization problems, more complex problems. The example of the problem that needs to be solved, you can see in Figure 6, where on the very right side the connection is weak because the closest base station is not powerful enough. To solve this and other problems, we could use a framework, an OF. OF is the framework that returns the solvers based on a defined model, input attributes and provides configuration for the engine. OF is a decision doing the fulfilment orchestration process. (ScienceDirect, 2018)

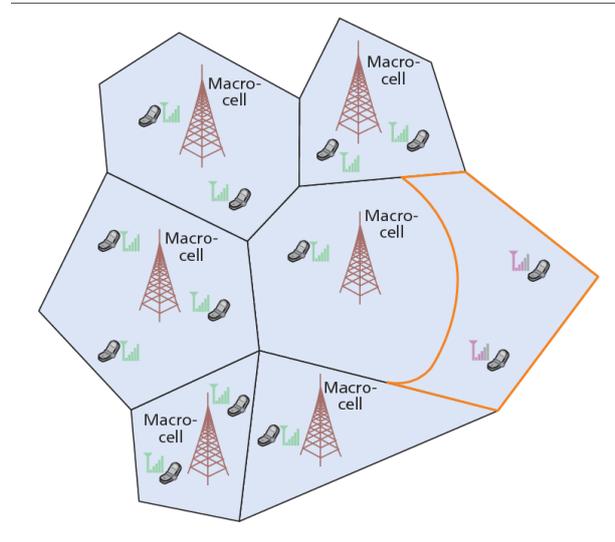


Figure 6. Map of network planning (Wang, 2016)

The first important step when a new service is introduced is in the planning phase. Optimization can find out where in the network you have available resources and where are you missing those. In turn, available resources could help to avoid unnecessary costly expenses in the telecommunication business. Optimization solver can be used also in letting a system automatically find the optimal path based on your specific criteria such as the shortest distance, number of hops, utilization of links and costs. (Techopedia, 2019)

2.4 ONAP

ONAP as a project was set first time in 2017 with the idea of creating a common platform for individual operators, among those can be e.g. cloud and telecommunication suppliers. The ONAP's main difference is profitability and competitiveness. (ONAP, 2019)

ONAP stands for Open Network Automation Platform. It is a programme above the infrastructure level that makes sure to automate the network on the whole. The platform allocates both an operator and an end-user to attach products and services through the architecture, it allows the arrangement of VNFs (Virtual Network Functions) and scales the network in an entirely self-operating mode. (ONAP, 2019)

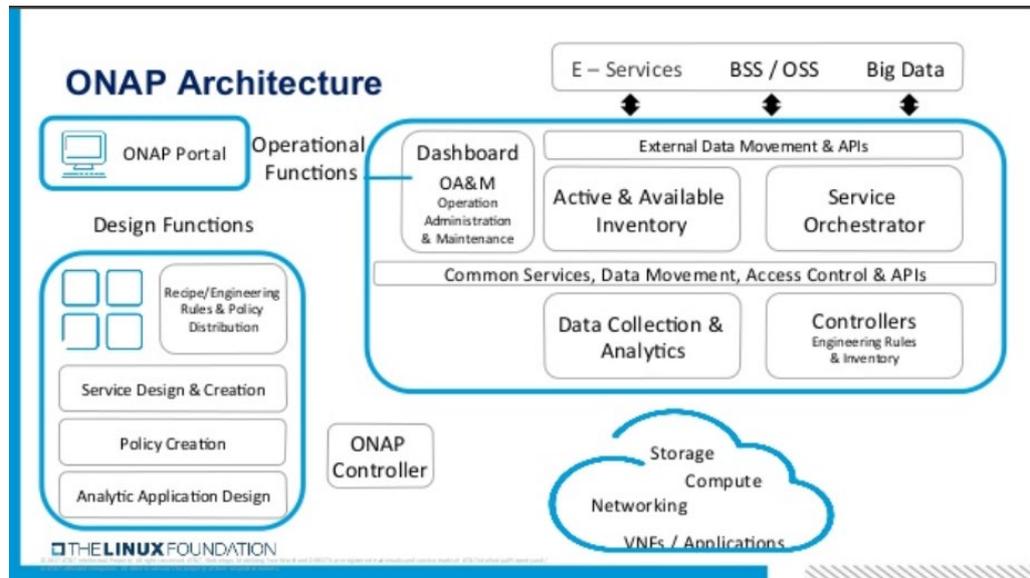


Figure 7. ONAP architecture (ONAP, 2017)

Benefits of ONAP:

- Scalable
- Fully automated
- Increase flexibility
- Increase feature velocity
- Enhances customer experience
- Supports 5G and IoT evaluation (ONAP, 2019)

ONAP is the network operating system for SDN (Software Defined Networking) and it handles the complete lifecycle associated with onboarding, orchestration, control, inventory, policy, etc. (ONAP, 2019)

2.5 MiniZinc

We face optimization problems every day playing Sudoku, planning our trip to the place of our dream, renovating an apartment. An example of a solving optimization problem can be Google Maps. When you want to get from point 'A' to point 'B', Google will show you the shortest available road or sometimes several roads. The best answer to constraint optimization problems can be given by constraint programming solvers. the majority of them are demonstrated in Figure 3. Most of these tools are open source projects, except CPLEX and Gurobi, which are commercial versions. (Bessiere et al., 2017)

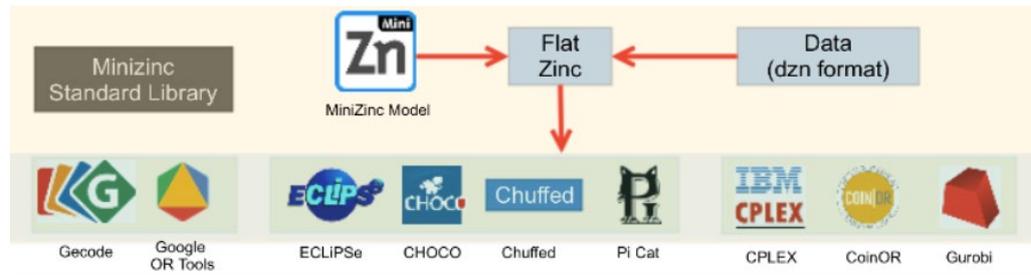


Figure 8. Constraint programming solvers (ONAP, 2019)

They work similar to translators and one of the languages that they can understand is MiniZinc. Besides, MiniZinc does not dictate what solver you need to use, it supports all the solvers. (MiniZinc, 2019)

MiniZinc is a high-level solver agnostic input modelling language, not a programming language. To be more precise MiniZinc is both a constraint language that is used to describe a model and a compiler that produces a file in a format of FlatZinc. FlatZinc is a low-level modelling language. It is simpler than MiniZinc language, it has fewer features. (MiniZinc, 2019)

Constraint optimization problems can be at different levels and have a different number of parameters. However, the concept stays the same and to solve any type of problem with MiniZinc we are required to provide at least a model file (model.mzn). (MiniZinc, 2019)

MiniZinc models can be run in two ways: by MiniZinc IDE and in the command line. The user describes the optimization problem using MiniZinc modelling language. The problem is stored in the so-called model file, which has the type of '.mzn'. In some cases, where a client wants to solve a simple constrained optimization problem, the instance data might be written to the popup fields using MiniZinc IDE or can be specified in the same file with constraints and solver. (MiniZinc, 2019)

```

%parameters definitions
int: budget = 200;

%decision variable declaration
var 0..4: Cake;
var 0..60: Cookie;
var 0..100: Lollipop;

%constraint
constraint 100*Cake + 6*Lollipop + 15*Cookie <= budget;

%solver
solve maximize 90*Cake + 10*Lollipop + 30*Cookie ;

%objective
output ["Cake = \ \(Cake), Lollipop = \ (Lollipop), Cookie = \ (Cookie)"];

```

Figure 9. Maximize sweetness satisfaction

Figure 9 demonstrates the problem where we have a budget of 200 euros, three options of sweets we can buy: cake, cookies, and lollipops. All these products have different prices and bring a different amount of satisfaction to the eaters. E.g. cake costs the most – 100 euros and brings you 90 points of eaters' happiness. We want to know how many cakes, cookies, and lollipops we should buy to make the eaters maximum happy. The last thing we specify is the way the output will look like, which is an optional feature.

As we can see from the previous example, we used only one file to define and solve the constraint problem. However, this method is not endorsed, we recommend keeping data separately from the model definition as it is better to store the model and the data in separate files to make the files more readable and the process easy to understand. The data file does not have special requirements except it should be saved within the format of '.dzn'.

To understand the operation of taking your MiniZinc model, which is written in the high-level MiniZinc, we refer to the process called Flattening. (MiniZinc, 2019)

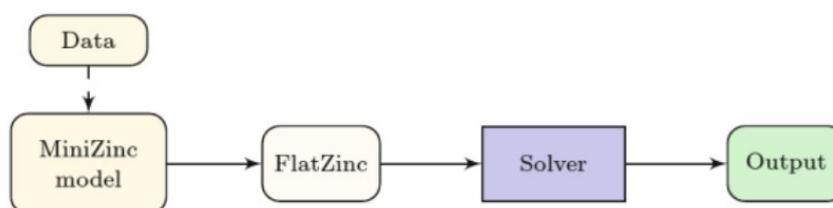


Figure 10. Overview of the MiniZinc toolchain (Bessiere et al., 2017)

Looking at Figure 10, we have two files: model.mzn and data.dzn. MiniZinc will make its first function by compiling those two files into FlatZinc format. Next MiniZinc runs an external selected solver, solver takes FlatZinc file and finds an answer for the specified unknowns. MiniZinc also translates the output of the solver into a form that specified in the model. For the convince you may create a configuration file where you specify the solver and change it whenever it is required. (MiniZinc, 2019)

2.6 Database

2.6.1 SQL

A relational database is the most widely used kind of database. It works well with applications tailored for a wide range of uses, from enterprise manipulations to science research. Many of us are aware of a relational database by knowing some of these representatives,

like Oracle, Microsoft SQL, and PostgreSQL. Although relational databases work well in many cases, some applications have requirements that are difficult to meet with relational databases. For example, some applications must write large volumes of data quickly, while other applications require extremely fast query response times. In addition to this, some applications need to have a high level of accessibility. SQL databases can achieve high read and write rates, but some of the properties of relational databases make this difficult. For example, relational databases ensure that read and write operations are acted and done in the same way over time so that all users see the latest correct version of data. Keeping track of multiple users, reading and writing data, and ensuring that data is consistent adds overhead to relational database operations. (Beaulieu, 2009)

2.6.2 NoSQL

NoSQL database is another take on data storing that takes the user away from the relational model. The most famous examples of NoSQL that are in daily use are Cassandra, MongoDB, AWS SimpleDB or Accumulo.

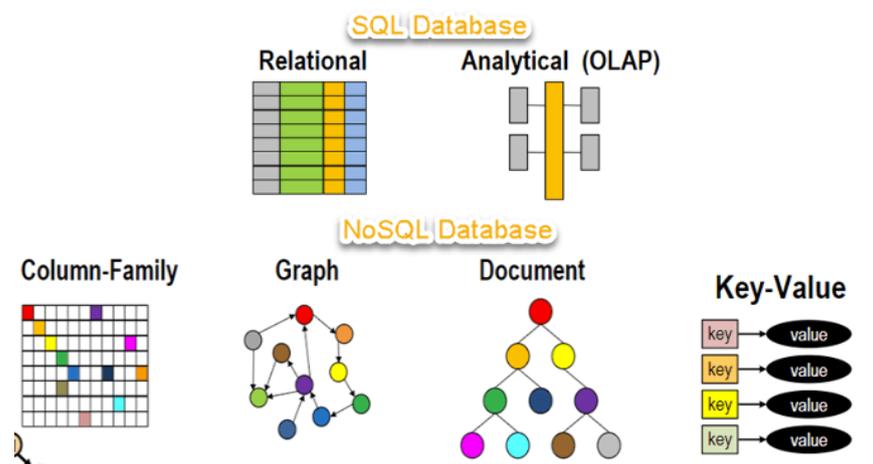


Figure 11. The difference between SQL and NoSQL (Guru, 2017)

If dealing with a server that does not promise the same level of stability, using NoSQL is the right choice. You can have much quicker server operations with NoSQL. This type of database is also characterized as a non-relational database. As the name implies, they do not use SQL for specifying and data processing. (Hills, 2019)

2.6.3 Cassandra

Cassandra is designed for availability and scale. Cassandra was written originally by Facebook for its messenger platform back in 2008 and it is one of the most popular NoSQL, enormously extendable databases today. (Hewitt, 2010)

Besides being a NoSQL database, Cassandra has many parallels to relational databases. Both use tables as a structure of basic data. Tables consist of columns in which attributes are stored. Cassandra uses data forms that a relational database programmer would be accustomed to. Cassandra tables have primary keys (one or more). They mark unique rows in a table. In Cassandra, primary keys are used for accessing the table's data. (Hewitt, 2010)

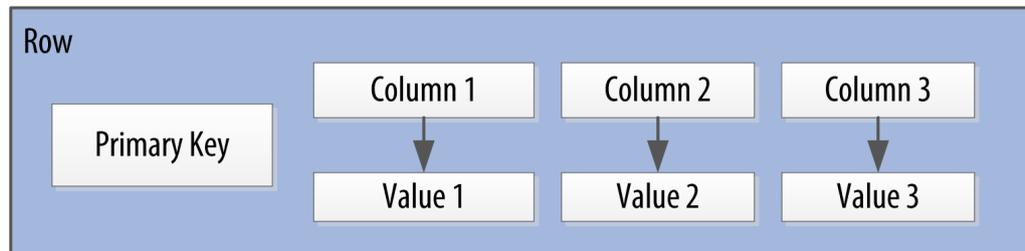


Figure 12. Cassandra row of data (Hewitt, 2010)

To enable fast access to rows within tables that span multiple servers, Cassandra tables use two additional kinds of keys. A partition key is used to determine which node in the cluster to store a row in. A clustering column defines the order in which rows are stored. Cassandra cluster servers are referred to as nodes.

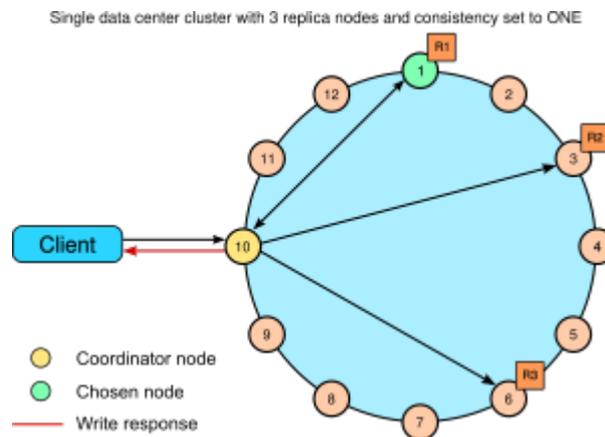


Figure 13. Cassandra architecture (Hadoop online tutorial, 2016)

2.7 API

API is the core of modern program development connecting data and services together to an expanding universe of applications across web, mobile, IoT and beyond. Everyday thousands of APIs are produced making millions of connections. (Biehl, 2019)

2.7.1 REST API

REST API is a shortening in the developers' daily routine for **R**epresentational **S**tate **T**ransfer **A**pplication **P**rogramming Interface. REST APIs are essential to the development of web-based projects and become the cornerstones of all web development. (Biehl, 2018)

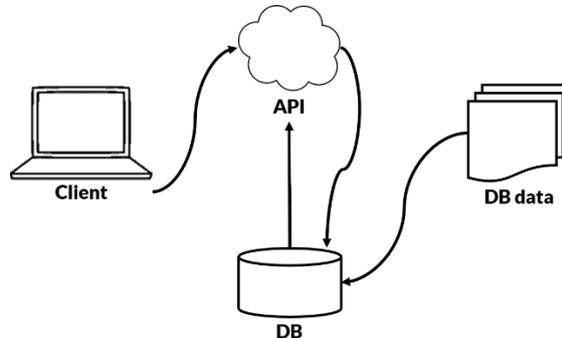


Figure 14. API calls work (Apurva, 2018)

It is a style of architecture for patterning web-applications. It works on a stateless client-server communication protocol. The important fact about REST is that it operates using just HTTP and JSON standards. This makes possible to use REST API by any programming language. Among those languages can be Python, Java, JavaScript, etc. All of them can work with a REST interface. (Biehl, 2018)

2.7.2 Swagger

Swagger is an API definition language. In other words, Swagger is a set of rules to semantically describe an API. It is in human- and machine-readable structured text format. (Swagger, 2019)



Figure 15. Swagger Editor (BlazeMeter, 2017)

There are three tools within the open-source toolbox, the Swagger Editor, Swagger UI, and Swagger Codegen. The editor acts like an IDE for building files with API definitions. Swagger UI can consume API definition files to produce interactive API evidence. Finally, Swagger Codegen can build API clients or server stubs in several languages based on the API definition files. (Swagger, 2019)

The most important takeaway is that Swagger's tooling centres around API definition files that conform to the open API specification. Swagger's open-source tools are a great way to design, test, document and build definitions for RESTful web services. (Swagger, 2019)

- **Build.** Swagger allows you to build API you like with the help of a demo provided by default. Swagger makes the building process easy, build stable and code reusable.
- **Test.** With Swagger, every endpoint can be tested. This is useful both for API endpoints discovery and the development of the API.
- **Document.** Documentation is a vital part of any RESTful API. Swagger is used for creating API documentation for any project regardless of the computer language used to create it. Swagger automatically creates static documentation for every operation, parameters, responses, description, output schema, and other metadata. Swagger UI lists all the available REST endpoints and provides detailed information about them. (Swagger, 2019)

2.7.3 Postman

Postman is a good tool to bring order to the structured work with APIs without regard to the API and the stage of the API. When an API is first created with a specification, its design elements can be built directly within Postman or imported from a common format. When its description is ready it is time to build the API. (Postman learning center, 2019)

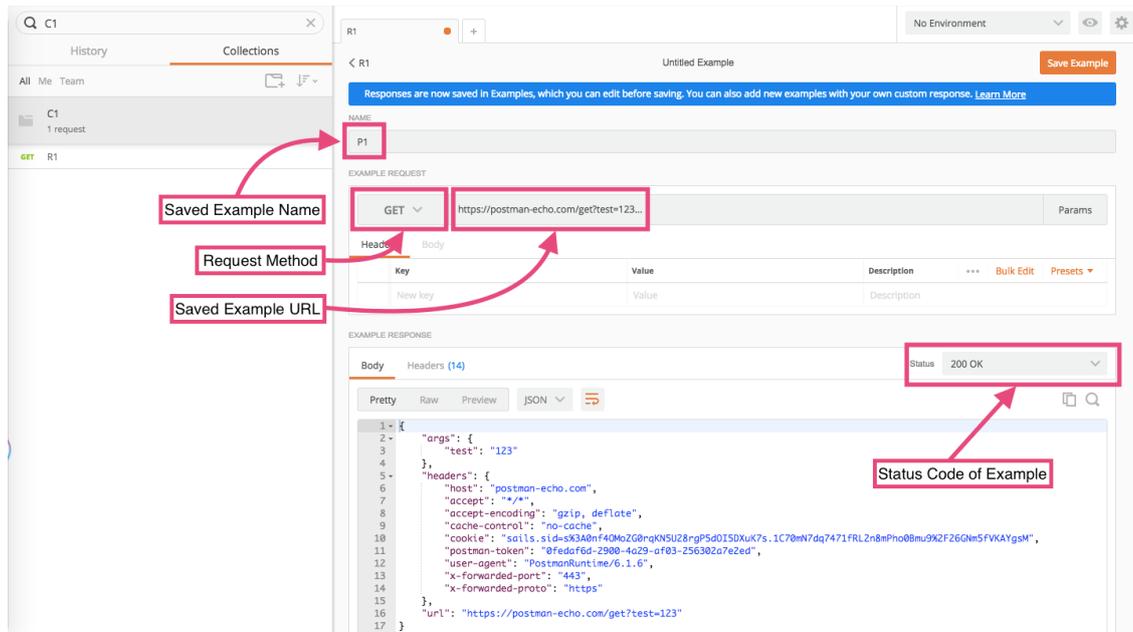


Figure 16. Workspace for the creating API with Postman (Postman learning center, 2019)

Postman is available in every operation system: Mac, Windows, and Linux. Postman is an ideal tool if you are writing your API or if you are working on a server-side code in technology like PHP, Java or Node.js. With Postman you can start interacting with the existing backend and get feedback on how things are going immediately, without building client-side or writing front-end code. In other words, Postman demonstrates to you if the API works the way you expect it, or something needs to be corrected. In the long run, Postman saves development time and helps to find bugs early in the development stage. (Postman learning center, 2019)

Postman's mock server mimics the API to show how it will run when the application's API is completed. Postman can be used to debug the API, writing a test to ensure that API works exactly the way it was created for. Postman testing is comprehensive, flexible and can be even automated. (Postman learning center, 2019)

An exceptional API also needs exceptional documentation, that others can learn all about the API, its models, methods, components, and requirements. Postman documentation is detailed, customizable and machine-readable. According to Postman tutorial documentation for the API includes:

- Requests and headers samples
- Representation associated with collections and requests
- Other metadata

Postman monitoring allows regular checking to ensure the API is behaving to spec. Postman monitoring will know right away and warn a user if anything changes. (Postman learning center, 2019)

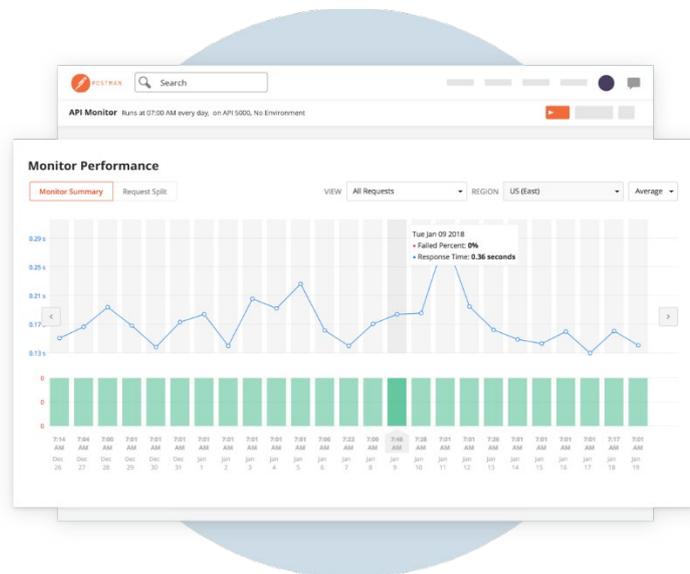


Figure 17. API Monitoring with Postman (Postman, 2019)

Postman has also publishing tools that can be used to spread the API with someone else. Others can get everything they need to know to work with the API. The API collections can be exchanged amongst the team with just one click in Postman. (Postman, 2019)

2.8 Docker

Docker is a software that helps create the wrapper around the files and the libraries that are needed for a code to work. (Poulton, 2018)

With Docker, the project is built inside a container. There is no need to configure this project multiple times on different platforms. The project can be run on any OS same way as it was run in the software engineer's OS. (Novoseltseva, 2018)

One of the biggest benefits of using Docker is that every application works inside its container and does not interfere with other applications. Numerous containers can be started on the same system without intervention. When you remove the container, it will delete all the files and will not leave behind any traces on the system. (Novoseltseva, 2018)

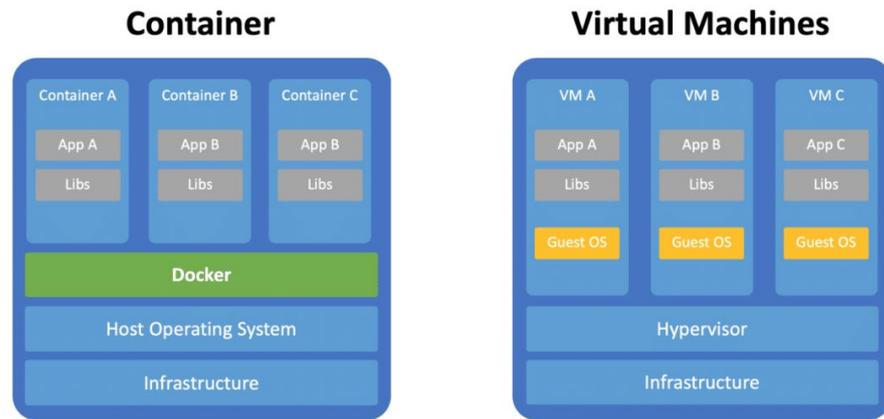


Figure 18. Containers vs. Virtual Machines (Eschweiler, 2019)

Containers are the artefacts, as you can see from Figure 18, containers are like VM, but it is a lot smaller, faster and takes up fewer resources. The way containers are deployed to make them easy to scale fast and provide elastic cloud services. The main difference between container and VM is that the container virtualizes the OS so that many workloads can be run on a single OS when VM runs a unique guest OS. Running several OS may be many GB in size. (Poulton, 2018)

With Docker, a programmer can wrap all the software, libraries and dependencies in one consistent package and Docker will make sure that this container can be deployed on every possible platform and work fine on every system. For example, a developer has specific demand dependencies for the code, after setting them up in the container, he/she can be confident that as the application moves from environment to environment it has what it needs to run. (Poulton, 2018)

2.9 Network terminology

Here we have a look at the crucial terminologies that are used in the Network study and will be experienced in the use case of the next chapter “Design and planning”.

- **NFVI-PoP** - Network Function Virtualization Infrastructure Point of Presence. In other words, NFVI-PoP is the location of NFVI. Where NFVI is an NFV infrastructure that characterizes the elements of hardware and software on which virtual networks are raised. (Office of the CTO blog, 2019)

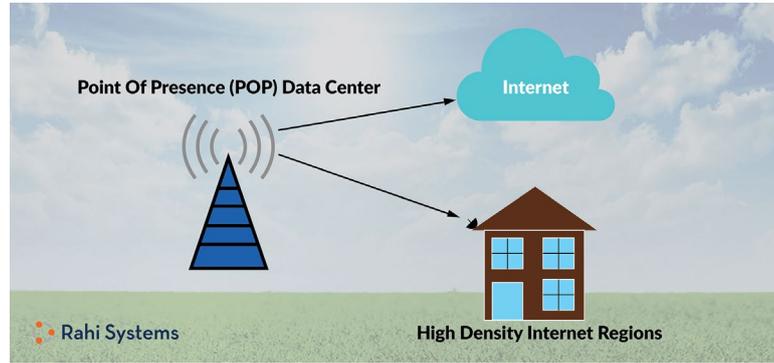


Figure 19. Point of Presence (Rahi Systems, 2019)

- **Cell** – is one part of the Cellular network revealed in Figure 20. For instance, each octagon is one of the cells. The Network is spread among over the land areas that we call “cell”. (Scribd, 2014)

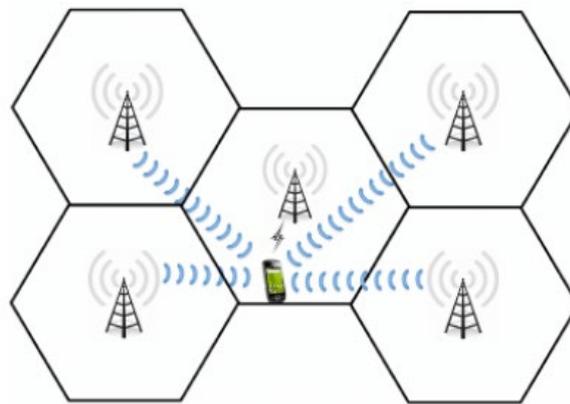


Figure 20. Cellular network (Arash, 2013)

- **gNB** – next-generation NodeB that assists 5G New Radio. The New Radio operates at much higher frequencies than the traditional eNB (4G). (National Instruments, 2019)

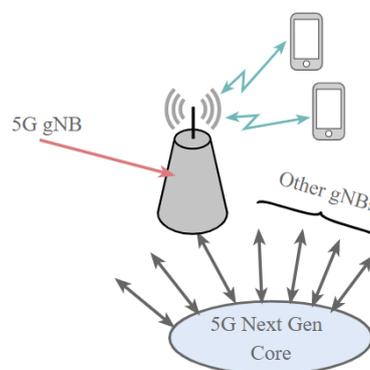


Figure 21. 5G radio access network gNB (Electronics Notes, 2018)

If to split the architecture and stand out the main components from the complete picture, we get a client, the OF that stays for the Optimization Framework, DB (NoSQL Cassandra DB) and external model and data files. Let have a closer look at OF structure. It includes defined API, Services, Task, and Model components, however, the most interesting part for us is the MiniZinc optimization plugin, which stays behind the OF logic. Let us summarize the components we have and give them a short description:

- Client - is a single user that executes the application.
- Model and Data files – are marked orange. They represent the documents with model and data parameters, respectively. Both files are required to perform the MiniZinc engine and to create a task.
- Cassandra – NoSQL database, more detailed in Chapter 3.3 “Database”.
- MiniZinc optimization plugin – is the method that runs model and data to provide a client with an optimization solution.
- Model and Task – are marked blue. Both stay for Java objects with own unique key and object’s variables.
- Services – in our case two services describe business logic in different layers. Communication between API methods and Java objects is taking place in Service classes.

The application comes to life once the application is built and running. A client makes an API request by calling any of the procedures, e.g. creating a model, getting the model, deleting it or creating a task. API contains methods that are controlled and go through the created services. For the code’s clearness, we keep the models and the data’s controllers as well as services separately. So, we have a separate controller and a server for a model and a task object. The picture below illustrates the location of those files in the application structure.

```
rest
├── ModelController.java
├── TaskController.java
└── service
    ├── DataService.java
    ├── MinizincService.java
    ├── ModelService.java
    └── TaskService.java
```

Figure 23. Part of application's files

3.2 Use case

During the project work, several use cases were specified. The use cases were needed to establish clear objectives and functions to work on. To encourage a common understanding of this study we provide one of the use cases here. The use case contains a picture that represents the visual high-level schema, a few sequences UML diagrams, the list of terminologies, the defined statement, constraints, and post-conditions.

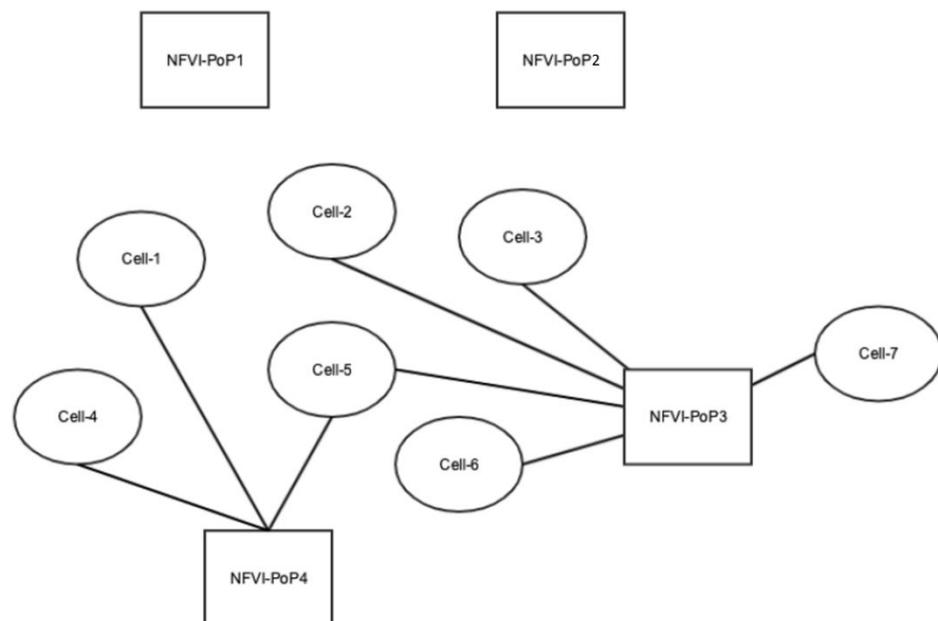


Figure 24. 5G gNB placement

<p>Terminology:</p> <ul style="list-style-type: none"> • NFVI-PoP - Network Function Virtualization Infrastructure Point of Presence (location of NFVI) • Cell – is one part of the Cellular network. • gNB – next-generation NodeB
<p>Statement:</p> <p>There are several NFVI-PoPs available for deploying gNB, that serve these Cells.</p>
<p>Requirements/constraints:</p> <p>The distance between a cell and NFVI-PoP must be within 35KM.</p>
<p>Output/Post-conditions:</p> <p>Which NFVI-PoP(s) shall be selected to deploy gNB with minimum average latency?</p>

As we can see from Figure 24, we have two presented variables, where the tracking area list has been translated, mapped to a list of Cells (Cell-1, Cell-2, ... Cell-7) and four NFVI-PoPs. The distance from Cell to each NFVI-PoP is known. We would like to find which NFVI Point of Presence to pick and do it with the minimum average latency.

To gain the optimization solution we create a model using MiniZinc modelling language.

```

1 enum NFVIPOP = {'p1', 'p2', 'p3', 'p4'};
2 enum CELL = {'c1', 'c2', 'c3', 'c4', 'c5', 'c6', 'c7'};
3
4 int: distanceLimit = 35;
5 array[NFVIPOP, CELL] of int: distances = [|15, 42, 38, 47, 37, 36, 41
6                                           |30, 43, 45, 26, 32, 41, 27
7                                           |23, 15, 31, 13, 26, 27, 11
8                                           |18, 40, 29, 18, 11, 19, 15
9                                           |];
10
11 array[CELL] of var NFVIPOP: p;
12 var int: latency = sum(i in CELL)(distances[p[i], i]);
13
14 constraint forall (i in CELL)( distances[p[i], i] <= distanceLimit);
15 solve minimize latency;
16 output ["Cell\{i} is deployed to \{p[i]\}\n" | i in CELL] ++
17 ["Average latency: \{latency div card(CELL)\}"];

```

Figure 25. A MiniZinc model file

The model starts with declaring variables. In our case, we define two enumerated types (Lines 1 and 2), a parameter in the problem which is the maximum distance between

NFVI-PoP and Cell can be taken (Line 4). We also define two arrays, where the first array describes the set of distances (Line 5) and the second array that declares the Point of Presence towards Cell (Line 11). Finally, we set the Integer variable with the variable name – latency, which calculates sum (Line 12).

Next, we determine the boundary for our optimization problem. For this, we use “constraint forall”, where “forall” stays for an aggregate expression. The constraint has a binary constraint definition – less-or-equal (Line 14), which makes sure the distance does not overshoot the specified distance limit. Next simply call “solve” item that indicates the kind of problem that needs to be solved, in our case to minimize the latency (Line 15). The last step is to specify the look of the output, MiniZinc transfers the solver output into the form specified in the model (Line 16, Line 17).

When we run this model in Minizinc IDE or using a command line the output looks as follows:

```
minizinc> minizinc model.mzn  
  
Cellc1 is deployed to p1  
Cellc2 is deployed to p3  
Cellc3 is deployed to p4  
Cellc4 is deployed to p3  
Cellc5 is deployed to p4  
Cellc6 is deployed to p4  
Cellc7 is deployed to p3  
Average latency: 16  
-----  
=====
```

Figure 26. MiniZinc printed result from the use case

The result tells us next: NFVI-PoP1 deploys Cell1, NFVI-PoP2 is not used at all, NFVI-PoP3 deploys Cell2, Cell4, and Cell7 when NFVI-PoP4 makes use of Cell3, Cell 5 and Cell6. Also, the minimum latency that was found was 16.

This example is a demonstration of one of the use cases, but the logic and the idea of the optimization engine stays the same: to get the most satisfying result for the brought problem.

3.3 Database

For data storing we chose Cassandra - nonrelational DB. Cassandra can be installed to run it locally however in our case it is not required because we use Docker to run the database. The instructions on how to configure and run Cassandra in Docker can be found in the chapter "Setting up the environment (Section of chapter 4.2.3).

After the database is set and the application is started, Cassandra's unique keyspace and tables for model and task objects are created, if they did not exist before. From the beginning the tables are empty and the additional actions need to be done to fill the tables with data.

```
@Table("model")
@ApiModel(description = "Details of table 'model'")
public class Model {

    @PrimaryKey
    @ApiModelProperty(notes = "The unique name of the model")
    private String name;
    @Column
    @ApiModelProperty(notes = "The model's content")
    private String content;

    public Model() {}

    public Model(String modelName, String content) {
        this.name = modelName;
        this.content = content;
    }

    //Getters and Setters here
}
```

Figure 27. Code sample of the Model table

The picture above demonstrates the model class that also dictates the Cassandra table's variables and their types, e.g. if the variable can be a type of NULL or not and finally the class specify the primary key for the table using '@PrimaryKey' annotation. To create a table in the database, we need to use annotation '@Table', which allows you to define the characteristics of the table that will be used to persist the entity in Cassandra DB. By default, the name of the table is the same as the name of the class. If we want to specify our table name, the name is overridden in parentheses: '@Table("model")'.

Each table has own Repository to set and get the variables we need. The Repository files look in the following way:

```
@Repository
public interface ModelRepository extends CassandraRepository<Model, String> {
}
```

Figure 28. Model's repository code

```
@Repository
public interface TaskRepository extends CassandraRepository<Task, String> {

    /**
     * Returns Task by specified unique task id
     *
     * @param taskId
     * @return Task
     */
    Task findById(String taskId);
}
```

Figure 29. Task's repository code

3.4 Version Control System

During the developing, the source code was modified many times. It was decided to use the Gerrit version control system on different stages. The source code is stored and dedicated by Git repository, however, Gerrit takes control of all made commits. The code review and developing discussions are done in the Gerrit system too. While Git is used directly from the command line, Gerrit has a user-friendly interface for team communication and code modification comments.

3.5 API design

During the project designing and planning period, it was decided to develop RESTful API as part of this work. The API is required for revival the optimization engine's functionality and connecting the application to the database. The list of the requests was changing during the discussions; however, the final version of the API definition was set and the list of requests that are used in this project can be seen below:

- **POST** /model (Create a model)
- **GET** /model (List all models)
- **GET** /model/{modelName} (Find model by a modelName and fetch the model file)
- **DELETE** /model/{modelName} (Delete a model)
- **POST** /task (Create a task)
- **GET** /task (List all tasks)
- **GET** /task/{taskId} (Find a task by a taskId)

- **PATCH** /task/{taskId} (Cancel a task)

To get a better understanding of what every API endpoint is responsible for, we illustrate each of them separately further.

3.5.1 Model

POST.

Before to create a task, the client should upload a model file to the server. The server writes the file into the user-configured directory and stores it into Cassandra DB.

We can see the interaction logic between the objects in the system in the “Model sequence diagram (POST /model).” (Figure 30). User makes a POST request, the controller calls a method “create(file)” that checks if the directory to upload a model file is provided by the user, if no, then the service creates a directory, otherwise the service uses the existing one. At this point, the client gets the model file in the filesystem and next to this model is stored in the Cassandra repository. If every step was successful, the method returns a newly created model to the user and HTTP status of “201” (Created). The model file is ready to be used for task creation.

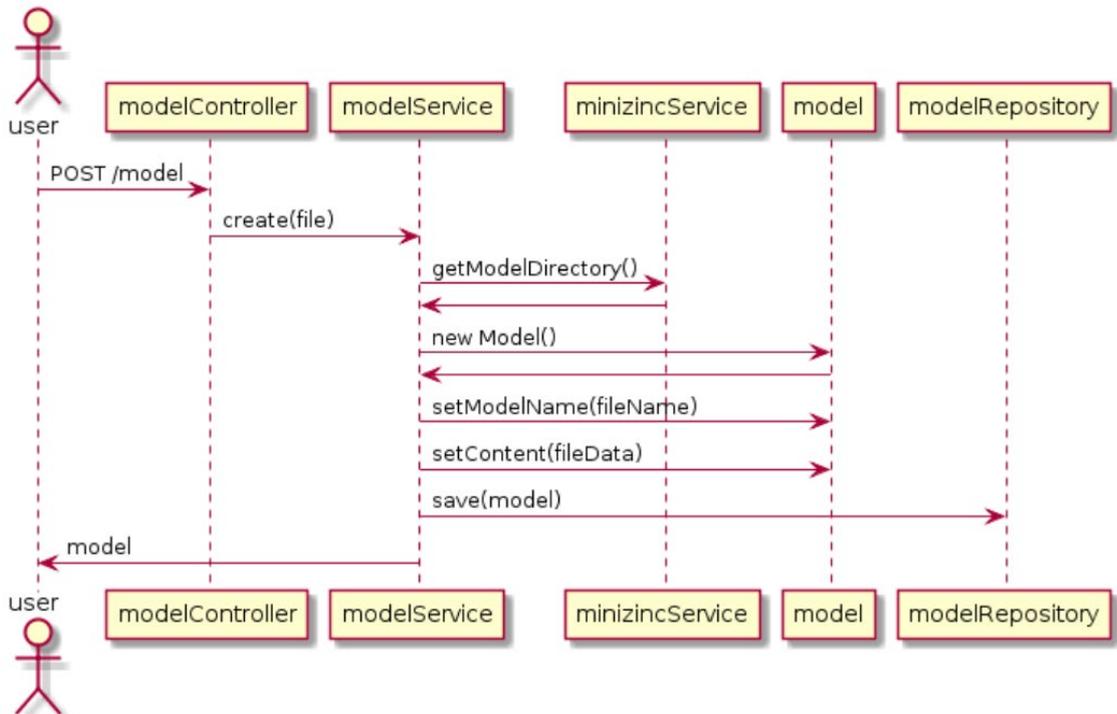


Figure 30. Model sequence diagram (POST /model)

GET.

Same endpoint ('/model') can be used for GET request. This method returns the list of all available models in the database, calling method: *findAll()* from the model's Cassandra repository.

```
public List<Model> listModels() {  
    return modelRepository.findAll();  
}
```

Figure 31. Method *findAll()* to list all models

3.5.2 Model/{modelName}

{modelName} is a sub-resource of '/model' endpoint and a unique id for a model object. The endpoint provides information and operations for a single model. This API ending is used in two requests: GET and DELETE.

GET.

The idea of this request is to find a model by a {modelName} and fetch the model file. First, we check if there is a model file in the specified by the user directory and if it is not an empty file. Secondly, we make sure that the repository has a model we request.

If every requirement is met, the method returns HTTP Status of '200' ('OK') and what is more important, GET method downloads a file to the user's machine and returns all information about the single model represented as a JSON object. The model object contains information about the model's unique name and the model's content.

In failure cases, we get a response with HTTP Status of '404' ('NOT FOUND').

DELETE.

The request deletes a single model from the database and the filesystem. Before to call a delete method we check is the model file exists in Cassandra DB as well in the directory where it was uploaded. If the file was not in the specified filesystem, the program returns message says that the model is not found with the specified name. In case the file was not found in the database, we get a response with HTTP Status of '204' ('NO CONTENT').

If both checks were done successfully, the method will delete the model file from both places: the database and the filesystem.

3.5.3 Task

The '/task' endpoint is used in two requests: POST and GET.

POST.

To be able to produce a task we also need a data file that together with the model go through MiniZinc optimization engine and generates a solution or several solutions as the result. Each task has a status, depending on the task creation stage the status is different. For example, in the beginning, every task has a status “ACKNOWLEDGED”, after the creation of the task started it gets a status of “RUNNING”. If the task is created successfully the status changes to “COMPLETED”.

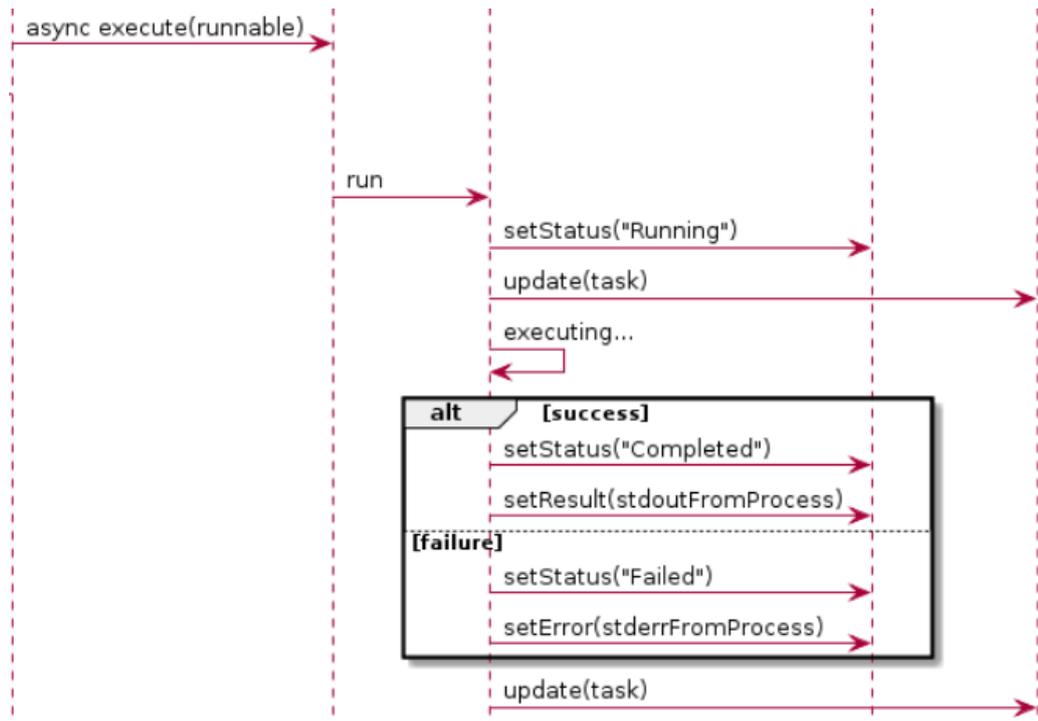


Figure 32. Part of a task sequence diagram (POST /task)

As you can see from the sequence diagram (Figure 32), the task creation process is run asynchronously, non-blocking. This means that the task creation can be annulled by any time and the process builder can be stopped whenever it is needed. If the task was cancelled, the result changes to *null* and the status to “CANCELLED” respectively. There is one more status is present, it is “FAILED”. This status is set in other cases when something went wrong, and the creation was not successful.

GET.

GET request returns the list of all tasks that are stored in the database. Even if there are no tasks, the response still has a status of '200' ('OK') but does not return any task.

3.5.4 Task/{taskId}

{taskId} is a sub-resource of '/task' endpoint, it represents the unique id of a task. The endpoint provides information and operations for a single task. This API ending is used in two requests: GET and PATCH.

GET.

The produced solution is stored in the database as the task's result. The client can print a single task's result calling GET /task/{taskId} request. The response contains a JSON object that has the task's id, a model's name, a data, a result (the result of optimization solution) and the status of task creation. The status specification was covered in the previous section of the chapter.

PATCH.

This is the last controller method in our API list. We provide 'taskId' to cancel a task creation while the creation process is still running or already finished. PATCH request has a method that checks if the task exists. If the task presents, after we call PATCH the status of the task will be changed to 'CANCELED' and the content of the task will be NULL. In case there is no task with the provided id, the method returns HTTP Status of '404' ('NOT FOUND').

3.5.5 API documentation

For forwarding development and a better understanding of the methods, resources, requests, and responses by other developers and team members, we need API documentation. For this purpose, we use a broadly used open-source tool – Swagger. To be concrete, we use Swagger UI for API documentation.



Figure 33. Swagger UI – the project's documentation

Pay attention to the web-link (in blue colour) provided by Swagger, the example we can see from Figure 33. Following this link, we get a JSON file with the full API documentation, making the understanding of requests and responses easier. The part of this JSON looks like:

```
],
  "paths": {
    "/api/v1/model": {
      "get": {
        "tags": [
          "model-controller"
        ],
        "summary": "List all models",
        "operationId": "getAllModelsUsingGET",
        "produces": [
          "application/json"
        ],
        "responses": {
          "200": {
            "description": "OK",
            "schema": {
              "type": "array",
              "items": {
                "$ref": "#/definitions/Model"
              }
            }
          },
          "401": {
            "description": "Unauthorized"
          },
          "403": {
            "description": "Forbidden"
          },
          "404": {
            "description": "Not Found"
          }
        },
        "deprecated": false
      },
      "post": {
```

Figure 34. API documentation in JSON

The list of API methods provided by UI friendly interface makes it easy to see the API calls difference. Each request can be expanded to see more detailed information, for this simply click on one of the methods.

model-controller Model Controller	
GET	/api/v1/model List all models
POST	/api/v1/model Create a model
GET	/api/v1/model/{modelName} Find model by a modelName and fetch the model file.
DELETE	/api/v1/model/{modelName} Delete a model
task-controller Task Controller	
GET	/api/v1/task List all tasks
POST	/api/v1/task Create a task
GET	/api/v1/task/{taskId} Find a task by a taskId
PATCH	/api/v1/task/{taskId} Cancel a task

Figure 35. Project's API requests

To get automatically updated API documentation, we create a Swagger configuration file (*SwaggerConfig.java*) and need to add a Swagger dependency to 'pom.xml' file. 'SwaggerConfig.java' contains only two methods: 'api()' and 'apiDetails()', where we specify the information like name of the project, name of the API documentation, version of the project. One can also set an author name and its contact details, but in our case, we specify all credential information as NULL.

```

@Configuration
@EnableSwagger2
public class SwaggerConfig {

    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2).select()
            .apis(RequestHandlerSelectors.basePackage("com.nokia.dos.pavarot.rest"))
            .paths(PathSelectors.regex("/api/v1.*")).build().apiInfo(apiDetails());
    }

    private ApiInfo apiDetails() {
        return new ApiInfo("Draft version of optimization engine", "API for Optimization Framework", "1.0", null,
            null, null, Collections.emptyList());
    }
}

```

Figure 36. Swagger Configuration file

After this simple setting is done, we can describe the API methods, give them additional information, Swagger allows us also to add the explanation to our data models (JSON

schemas). For this Swagger provide us with annotations (@ApiOperation, @ApiModel, @ApiParam, and @ApiModelProperty), you can see the examples from the code below:

```
@PostMapping(value = "/model", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
@ResponseStatus(HttpStatus.CREATED)
@ApiOperation(value = "Create a model")
public Model addFile(
    @ApiParam(value = "Model file", required = true) @RequestParam("file") MultipartFile multipartModelFile)
    throws IOException {
    return modelService.create(multipartModelFile);
}
```

Figure 37. API request to create a new model

@ApiOperation contains the value that describes the main idea of this controller method and @ApiParam clarifies if the method's parameter is required and what its value is.

```
@ApiModel(description = "Details of table 'model'")
public class Model {

    @PrimaryKey
    @ApiModelProperty(notes = "The unique name of the model")
    private String name;
    @Column
    @ApiModelProperty(notes = "The model's content")
    private String content;
}
```

Figure 38. Model class

Figure 38 demonstrates a few annotations that are used in 'Model.java' class. Among them, we can see @ApiModel, which represents the Swagger data model's description. @ApiModelProperty let us make a note to get a better idea of what one or the other variable stays for.

4 Project Implementation

After going through the theory background, evaluating and planning the design of the project, we describe the technical implementation process of the application. Going through this chapter we build an understanding of the stages that need to be done and what pre-conditions are required to run the product successfully. The environment setting instructions are covered in this chapter as well.

4.1 Prerequisites

To be able to run and test the application, a user should have basic skills of programming and he or she should be familiar with basic Java, including Spring Boot Java. Knowledge of API and mathematics will be an advantage. The project does not require the user to know anything about MiniZinc and its features as the background theory covers the involved understanding.

The part of testing API methods can be done in different ways: in user-friendly Postman, Swagger tool or using Curl commands in the command line. We provide the Curl commands that can be used to try out Controller methods in the chapter “Implementation” (Chapter 4.3).

To run the application the user might have any type of OS (operation systems): Linux, Mac or Windows. The user should have any type of code editor that he or she is comfortable with, however, in the implementation stage, we use Eclipse EE. Docker should be available depending on the user’s OP. Cassandra DB can be installed to the PC (Portative Computer) or it can be run in Docker image.

The full list of technologies that are used to run the program or recommended to be used to run and test the application:

- Any editor, preferably Eclipse, Visual Studio code, NetBeans or JIDEA
- The favourite browser
- Docker (e.g. Docker Desktop in Windows)
- Cassandra DB
- PowerShell, Git Bash or other OS’s command lines
- Postman, Swagger UI
- MiniZinc IDE

4.2 Setting up the environment

Before running the application, few steps need to be done. We are required to make several environment settings, including installation and configuration parts. There are three tools that we set: MiniZinc editor, Docker, and Database. We look at each of them individually in the next sections.

4.2.1 MiniZinc IDE

Installing MiniZinc IDE is a good idea. It can be used for testing purposes, for modifying, investigating both model and data MiniZinc files. MiniZinc IDE is a straightforward Integrated Environment not only for running but also for writing new MiniZinc models and data files. The IDE is supported by all three OS (Linux, Windows, Mac). It does not have complex installation, just download MiniZinc executable file from the official MiniZinc website and run it.

4.2.2 Docker

There are two Docker editions: community and enterprise editions, we can use both, but the last one goes with a price. We go ahead with the community edition in this study (free of charge). The community edition is available on Linux, Mac, Windows, and Cloud Platforms (e.g. Azure).

The first step to do is to install Docker depending on the OS. The installation is slightly different in different OS. In this study, we look at the process of setting Docker with Linux, Mac, and Windows.

Linux.

First, we identify a system physical or virtual machine or PC that has a supported OS, e.g. Ubuntu, Fedora VM. Next, we go to the Docker Engine Community Edition page and select the OS flavour to install. Pay attention to the prerequisites and requirements that you can find on the same Docker page. Uninstall older versions of Docker if one exists. Set up the repository and install Docker using the convenience script. It is a script that automates the entire installation process:

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sudo sh get-docker.sh
```

Figure 39. Docker script

After running the script, Docker is installed to Linux and to ensure it, we check the version of Docker, running: `'docker version'`.

Mac & Windows.

If the OS is either Mac or Windows, there are two options to set Docker:

1. Install a Linux VM using VirtualBox or other virtualization platforms. Then install Docker on the Linux VM. Using this way, we cannot create Windows/Mac based Docker images or run Windows/Mac based Docker containers. In this case, we just work with Docker on a Linux VM on a Windows/Mac host, there is nothing to do with Windows/Mac applications, Windows/Mac based images or containers. However, Docker provides us with a set of tools to make this easy. This set of tools is called the Docker Toolbox. Make sure to check the requirements before installation.
2. Install Docker Desktop for Mac or Docker Desktop for Windows, which are native applications. This option is newer and uses the native virtualization technology available with Windows called Microsoft Hyper-V. Pay attention to the information that this way of running Docker is supported only for Windows 10 enterprise or professional edition, as they support Hyper-V by default.

With Docker Desktop for Mac, we take out VirtualBox and use HyperKit virtualization technology. During the installation process for Docker for Mac, it still automatically creates a Linux system underneath, but this time it is created in HyperKit instead of Oracle VirtualBox and has Docker running on that system.

4.2.3 Cassandra database

We need a database for our backend. As was mentioned previously Cassandra DB can be run in two ways: locally and with the help of Docker. Before we look at different options to run the database, we check the default values of the new installed Cassandra.

Cassandra DB runs in `'localhost'` (127.0.0.1) with port number of `'9042'` and it has the default username and password, which are equal to `'cassandra'`. Thus, the command to log into Cassandra DB looks in the following way: `'cqlsh 127.0.0.1 9042 -u Cassandra -p cassandra'`, where `'-u'` stays for a username and `'-p'` for a password. In case we run the database with default values, the command can be shortened to one word: `'cqlsh'`.

Run Cassandra locally.

To run Cassandra locally, download the latest version of Apache Cassandra. The bit version can be 32 or 64, but we recommend you use the 64-bit version. From now we can log into the database. For this we open any command line, preferably running as an admin. In the command line, we execute the command: "cqlsh". This command will work only if Cassandra configuration was not modified and contains default values, otherwise, the user should specify his/her username, password.

For security reasons, we do not recommend keeping default database credentials, but instead, change them to their ones. The user may change the password to already existed username or create a new username (there are two types of users in Cassandra: 'superuser' and 'nosuperuser'). To enable creation of a username and to change a password one need to open 'conf/cassandra.yaml' file in the Cassandra folder and replace two lines of code:

```
authenticator: AllowAllAuthenticator
authorizer: AllowAllAuthorizer
```

Figure 40. Cassandra.yaml file (original)

with:

```
authenticator: PasswordAuthenticator
authorizer: CassandraAuthorizer
```

Figure 41. Cassandra.yaml file (modified)

Save and close the file. After the restarting Cassandra database, we login again to Cassandra with default credentials:

```
PS C:\> cqlsh -u cassandra -p cassandra
```

Figure 42. Login to Cassandra DB command

After login to the Cassandra database, we should see a picture similar to this:

```
Connected to Test Cluster at 127.0.0.1:9042.  
[cqlsh 5.0.1 | Cassandra 3.9 | CQL spec 3.4.2 | Native protocol v4]  
Use HELP for help.
```

Figure 43. Login successful

Here we can change the password, running the command:

```
cqlsh> ALTER USER cassandra WITH PASSWORD 'NEW_PASSWORD';
```

Figure 44. Change the default password command

Now when we have access to Cassandra DB, we should come back to our Spring Boot Application. The project comes with few configuration files, including Cassandra configuration (CassandraConfig.java). Configuration class extends AbstractCassandraConfiguration class, which provides us with methods, e.g. to automatically configure a table if it does not exist yet, to get the entity package. Moreover, we use configuration class to get a port, host, create a keyspace, and additionally, make the connection to provided host and port by providing username and password.

```
@Override  
public CassandraClusterFactoryBean cluster() {  
    CassandraClusterFactoryBean cluster = new CassandraClusterFactoryBean();  
    cluster.setJmxReportingEnabled(false);  
    cluster.setContactPoints(host);  
    cluster.setPort(port);  
    cluster.setAuthProvider(getAuthProvider());  
    cluster.setKeyspaceCreations(getKeyspaceCreations());  
    cluster.setReconnectionPolicy(new ConstantReconnectionPolicy(1000));  
    return cluster;  
}  
  
@Override  
protected AuthProvider getAuthProvider() {  
    if (username == null || username.isEmpty()) {  
        return null;  
    }  
    return new PlainTextAuthProvider(username, password);  
}
```

Figure 45. Cassandra configuration file

Run Cassandra in Docker.

Docker has own cloud-based repository called DockerHub. Install Apache Cassandra from DockerHub, running in the command line:

```
docker pull cassandra
```

Figure 46. Docker Cassandra

This command installs the Docker image of Cassandra. This command is enough to run Cassandra with Docker:

```
docker run --name 'CASSANDRA_NAME' -d -p 9042:9042 cassandra:tag
```

Figure 47. Docker run Cassandra

Assuming that '--name' sets the unique name for a Cassandra image '-p' stays for the port, '-d' for download and tag specifies the Cassandra version we want.

4.3 Running the project

Finally, when everything is installed, all environments are set and the application is built, we can run the project. For running the Spring Boot application, we use the command line and the command looks in the following way:

```
java -jar target/pavarot-0.0.1-SNAPSHOT.jar \  
  --cassandra.username=[USERNAME] \  
  --cassandra.password=[PASSWORD] \  
  --minizinc.modelDirectory=[MODEL file DIRECTORY] \  
  --minizinc.dataDirectory=[DATA file DIRECTORY]
```

Figure 48. Run Spring Boot application command

To start the application, we run the jar file. In addition to the jar file name, we can observe '--cassandra.username' and '--cassandra.password' parameters, instead of '[USERNAME]' and '[PASSWORD]' one needs to write own username and password of Cassandra DB. The last two lines specify the directory where we want to store our files. Notice that the model and data file can as well in the same directory as in the different folders. In other words, '[MODEL file DIRECTORY]' and '[DATA file DIRECTORY]' should be replaced by PATH (e.g. "C:\User\Username\my_projects"). The project built in such a way that if the directory specified by the user does not exist, the program will create this directory.

5 Discussion

The chapter describes the project's results together with the background details that were covered in this study. We have a look at obstacles that were faced during the work on the project, what challenges were needed to be solved to fully realize the vision. In addition to this, we determine the author's thoughts and learnings that were developing altogether with project implementation. At the end of the chapter, we spend time discussing the future of the current study and what could be done to improve the project or make it easier to use.

5.1 Review

Optimization problems are always relevant. Everyone has dealt with an optimization problem at least once a life. The wish of these people who face the optimization problem is to find the solution as soon as possible with less time spending on figuring out the best answer, without going beyond the constraints, in other words, to get the optimization result minimizing costs and maximizing profit and quality.

As it was mentioned in the theoretical section there are many solvers and programs that were created to help in the situation to optimize something. Network business and everything related to it is not an exception. During the regular discussion, the decision was made to automate the system with a comprehensive platform that could make the required calculations and could find the optimal solution based on the use case.

There was an assumption that we can reuse the ONAP optimization framework to gain our objectives. Time was allocated for this purpose and the investigation of ONAP was made. During the study, we went through the ONAP idea in general, its use cases and ONAP open-source code. At the end of the examination, the conclusion was that ONAP OF is use case-specific and it is not possible to completely reuse the code. However, we noticed that ONAP OF uses MiniZinc modelling language for optimization.

After several discussions, the second decision was made. It was decided to build its own optimization engine and with the use of MiniZinc language. It was determined to use Java as the main language and Cassandra DB for collecting the data.

The biggest tasks were to work on API definition, connecting the application to Cassandra DB in a proper way and developing the Spring Boot application that is a connector between the user and the optimization language.

From the author's point of view, another objective of this thesis was to bring Java knowledge to the next level, learn more about API definition, the difference between services' and repositories' responsibility. Also, it was important to learn the proper way to write README file, that it conveys the idea of the project, how to use it and all steps to make it work. Moreover, the skills of using a version control system were improved, as well as the understanding of the significance of the code reviewing process.

The main objective was to deliver the optimization engine and present it as a demo to the commissioning part corresponding to their requirements that would replay the current long and manual process of calculation of the optimizing solution to the fast and automated one.

5.2 Challenges and solutions

During the planning, designing and developing stages there were some obstacles that must be solved or considered to move some of them for the later improvement. Many challenges existed since the knowledge of Spring Boot was not satisfied enough, and the skills of programming were limited.

One of the biggest challenges was to make a building process asynchronous, non-blocking. The problem was that this issue required the author to have any background or experience with multithreading in Java. Unfortunately, demand skills were missing, and this programming part took longer than it was expected. However, after extra time spending and having several sessions with experiences developers, the result was reached and the conclusion for further improvement was made.

Another big problem was to make a programming concept a thread safety. Even after making the program asynchronous, during testing and code review it was found that the process is not thread-safe. Commissioning party concludes that this needs to be solved but in future development as the solving operation of this issue can take too much time.

Concerning the other challenges, the time was limited. During the planning period, the project was divided into several stages with deadlines that were crucial to follow. As the thesis topic was assigned by Nokia's technical architect team and people this level run other strenuous projects same time. The part of the project that was covered by this study requires persistent participation of the architects. The problem was that the time of the architects was limited. However, this problem was solved by doing some extra work in additional time and learning the theoretical part of the author's spare time.

5.3 Future development

The current project is only the first pace of it. The purpose of this work was to see if the optimization engine can be created using Spring Boot Java, MiniZinc and Cassandra DB. We already mentioned that during testing were found a problem with implemented multi-threading and the application must be modified to make it thread safety. This is one of the first steps to take next.

The next modification that already was designated is to change String format to JSON in some parts of the application. To be precise, some variables of a model and task objects are saved to the database as a String. But in the future, it makes sense to have them in JSON format. For example, the model parameters are stored all together as a one String object when we could save them in JSON format to pick up one single parameter one by one.

Of course, the security part should be improved. For now, the application is not secured enough, both authentication and the authorization parts are missing. Also, user tests should be created.

6 Bibliography

- Apurva, K. 2018. Rest API Calls Made Easy. URL: <https://blog.yellowant.com/rest-api-calls-made-easy-7e620e4d3e82>. Quoted: 20 September 2019.
- Arash, A. 2013. A Survey on Opportunistic Scheduling in Wireless Communications. *IEEE surveys and tutorials on communications.*, p. 12.
- Beaulieu, A. 2009. *Learning SQL: Master SQL Fundamentals*. O'Reilly Media, 2nd edition.
- Bessiere et al. 2017. *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*. Springer. 2016 ed. edition.
- BlazeMeter 2017. Create Your First OpenAPI Definition with Swagger Editor. URL: <https://www.blazemeter.com/blog/create-your-first-openapi-definition-with-swagger-editor/>. Quoted 20 September 2019.
- Danielle Navarro 2019. *Learning statistics with R: A tutorial for psychology students and other beginners*. CC BY-SA. Version 4.0.
- Edelkamp, S., Schrödl, S. 2012. *Heuristic Search. Theory and Applications*. Morgan Kaufmann. Elsevier Inc.
- Electronics Notes 2018. 5G NR New Radio. URL: <https://www.electronics-notes.com/articles/connectivity/5g-mobile-wireless-cellular/5g-nr-new-radio.php>. Quoted: 29 September 2019.
- Eschweiler, S. 2019. Docker – Beginner’s Guide – Part 1: Images & Containers. URL: <https://codingthesmartway.com/docker-beginners-guide-part-1-images-containers/>. Quoted: 4 October 2019.
- Gen, M., Cheng, R., Lin, L., 2008. *Network Models and Optimization: Multiobjective Genetic Algorithm Approach*. Springer Publishing Company.
- Gerrit 2019a. Gerrit code review. URL: <https://gerritcodereview-test.gsrc.io/>. Quoted: 5 October 2019.
- Gerrit 2019b. Review UI. URL: <https://gerrit-review.googlesource.com/Documentation/user-review-ui.html>. Quoted: 6 September 2019.
- Guru99 2019. NoSQL Tutorial. URL: <https://www.guru99.com/nosql-tutorial.html>. Quoted: 5 September 2019.

- Hewitt, E 2010. Cassandra: The Definitive Guide. O'Reilly Media Inc. Sebastopol.
- Hills, T. 2017. NoSQL and SQL Data Modeling: Bringing Together Data, Semantics, and Software. Technics Publications.
- Josef Kallrath 2004. Modeling Languages in Mathematical Optimization. Kluwer Academic Publishers.
- Mark L. Braunstein 2018. Health Informatics on FHIR: How HL7's New API is Transforming Healthcare. Springer.
- Matthias Biehl 2018. Webhooks - Events for RESTful APIs. API-University Press. First edition.
- Milanesio, L. 2013. Diffy The Kung Fu Review Cuckoo. Learning Gerrit Code Review. Packt Publishing.
- MiniZinc 2019. Welcome to MiniZinc. URL: <https://www.Minizinc.org/>. Quoted: 1 September 2019.
- National Instruments 2019. 5 Things to Know About 5G New Radio. URL: <https://www.ni.com/fi-fi/innovations/wireless/5g/new-radio.html>. Quoted: 5 November 2019.
- Neos Guide 2019. Types of Optimization Problems. URL: <https://neos-guide.org/optimization-tree>. Quoted: 2 September 2019.
- NFV 2016. What Is NFV Infrastructure (NFVI)? Definition. URL: <https://www.sdxcentral.com/networking/nfv/definitions/nfv-infrastructure-nfvi-definition/>. Quoted: 20 October 2019.
- Novoseltseva, E. 2018. Top 10 benefits you will get by using docker. URL: <https://apiumhub.com/tech-blog-barcelona/top-benefits-using-docker/>. Quoted: 4 October 2019.
- Office of the CTO blog 2019. Re-Architecting Telco Networks – The NFVI Way. URL: <https://octo.vmware.com/re-architecting-telco-networks-nfvi-way/>. Quoted: 5 November 2019.
- ONAP 2017. Introducing ONAP (Open Network Automation Platform). URL: <https://www.slideshare.net/cloudifysource/introducing-onap-open-network-automation-platform-bay-area-meetup>. Quoted: 1 September 2019.

ONAP 2019. Architecture. URL: <https://onap.readthedocs.io/en/latest/submodules/optf/osdf.git/docs/sections/architecture.html>. Quoted: 1 September 2019.

Postman 2019. Postman's build-in tools are everything a developer need. URL: <https://www.getpostman.com/tools>. Quoted: 25 August 2019.

Postman learning center 2019. Mocking with examples. URL: https://learning.getpostman.com/docs/postman/mock_servers/mocking_with_examples/. Quoted: 25 August 2019.

Poulton, N. 2018. Docker Deep Dive. Zero to Docker in a single book. Leanpub.

Rahi Systems 2019, PoP: Point of Presence Micro Data Centers. URL: rahisystems.com/blog/pop-point-of-presence-micro-data-centers/. Quoted: 20 October 2019.

Rai, P. 2016. Learning as Optimization: Linear Regression. Machine Learning.

ScienceDirect 2018. Network Optimization. URL: <https://www.sciencedirect.com/topics/engineering/network-optimization/pdf>. Quoted: 4 October 2019.

Solving Algorithms for Discrete Optimization 2018. Flattering. URL: <https://www.coursera.org/lecture/solving-algorithms-discrete-optimization/3-2-5-flattening-cRDOT>. Quoted: 1 September 2019.

Stuckey P. J., Marriott, K. 2018. MiniZinc Handbook. MiniZinc. Release 2.2.0.

Techopedia 2019. Network Optimization. URL: <https://www.techopedia.com/definition/31667/network-optimization>. Quoted: 4 October 2019.

Walls, A. 2016. Spring Boot in Action. Manning Publications. New York.

Wang, S., Ran, C. 2016. Rethinking cellular network planning and optimization. URL: <https://www.semanticscholar.org/paper/Rethinking-cellular-network-planning-and-Wang-Ran/c7d198e9ba6c95e483f0c269e38ff48ef2da2368>. Quoted: 17 October 2019.

Webopedia 2019. API - application program interface. URL: <https://www.webopedia.com/TERM/A/API.html>. Quoted: 4 September 2019.

7 Table of figures

Figure 1. Gerrit workspace sample (Gerrit, 2019a)	4
Figure 2. Gerrit repositories filter (Gerrit, 2019b).....	5
Figure 3. Optimization examples in various fields of everyday activity	6
Figure 4. Regression analysis (Navarro, 2019).....	6
Figure 5. Constraint optimization in Math.....	7
Figure 6. Map of network planning (Wang, 2016).....	8
Figure 7. ONAP architecture (ONAP, 2017).....	9
Figure 8. Constraint programming solvers (ONAP, 2019).....	10
Figure 9. Maximize sweetness satisfaction	10
Figure 10. Overview of the MiniZinc toolchain (Bessiere et al., 2017)	11
Figure 11. The difference between SQL and NoSQL (Guru, 2017)	12
Figure 12. Cassandra row of data (Hewitt, 2010).....	13
Figure 13. Cassandra architecture (Hadoop online tutorial, 2016).....	13
Figure 14. API calls work (Apurva, 2018).....	14
Figure 15. Swagger Editor (BlazeMeter, 2017)	14
Figure 16. Workspace for the creating API with Postman (Postman learning center, 2019)	16
Figure 17. API Monitoring with Postman (Postman, 2019).....	17
Figure 18. Containers vs. Virtual Machines (Eschweiler, 2019).....	18
Figure 19. Point of Presence (Rahi Systems, 2019).....	19
Figure 20. Cellular network (Arash, 2013).....	19
Figure 21. 5G radio access network gNB (Electronics Notes, 2018)	19
Figure 22. The architecture of the application	20
Figure 23. Part of application's files	22
Figure 24. 5G gNB placement.....	22
Figure 25. A MiniZinc model file.....	23
Figure 26. MiniZinc printed result from the use case	24
Figure 27. Code sample of the Model table.....	25
Figure 28. Model's repository code	26
Figure 29. Task's repository code	26
Figure 30. Model sequence diagram (POST /model)	27
Figure 31. Method findAll() to list all models.....	28
Figure 32. Part of a task sequence diagram (POST /task)	29
Figure 33. Swagger UI – the project's documentation	30
Figure 34. API documentation in JSON	31
Figure 35. Project's API requests.....	32

Figure 36. Swagger Configuration file	32
Figure 37. API request to create a new model	33
Figure 38. Model class	33
Figure 39. Docker script	35
Figure 40. Cassandra.yaml file (original)	37
Figure 41. Cassandra.yaml file (modified)	37
Figure 42. Login to Cassandra DB command	37
Figure 43. Login successful	38
Figure 44. Change the default password command	38
Figure 45. Cassandra configuration file	38
Figure 46. Docker Cassandra	39
Figure 47. Docker run Cassandra	39
Figure 48. Run Spring Boot application command	39