



FEDERAL UNIVERSITY OF SANTA CATARINA  
TECHNOLOGY CENTER  
AUTOMATION AND SYSTEMS DEPARTMENT  
UNDERGRADUATE COURSE IN CONTROL AND AUTOMATION ENGINEERING

Gustavo Schmitz Albino

**Development of a DevOps infrastructure to enhance the deployment of machine learning applications for manufacturing**

Aachen  
2022

Gustavo Schmitz Albino

**Development of a DevOps infrastructure to enhance the deployment of machine learning applications for manufacturing**

Final report of the subject DAS5511 (Course Final Project) as a Concluding Dissertation of the Undergraduate Course in Control and Automation Engineering of the Federal University of Santa Catarina.  
Supervisor: Prof. Jomi Fred Hübner, Dr.  
Co-supervisor: Hendrik Mende, M.Sc

Aachen  
2022

Ficha de identificação da obra elaborada pelo autor,  
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Albino, Gustavo Schmitz

Development of a DevOps infrastructure to enhance the deployment of machine learning applications for manufacturing / Gustavo Schmitz Albino ; orientador, Jomi Fred Hübner, coorientador, Hendrik Mende, 2022.

98 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2022.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Qualidade da produção. 3. Aprendizado de máquina. 4. Integração contínua. 5. Diagramas UML. I. Hübner, Jomi Fred. II. Mende, Hendrik. III. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. IV. Título.

Gustavo Schmitz Albino

**Development of a DevOps infrastructure to enhance the deployment of machine learning applications for manufacturing**

This dissertation was evaluated in the context of the subject DAS5511 (Course Final Project) and approved in its final form by the Undergraduate Course in Control and Automation Engineering

Florianópolis, March 07, 2022.

Prof. Hector Bessa Silveira, Dr.  
Course Coordinator

**Examining Board:**

Prof. Jomi Fred Hübner, Dr.  
Advisor  
UFSC/CTC/DAS

Hendrik Mende, M.Sc.  
Supervisor  
Fraunhofer Institute for Production Technology IPT

Prof. Leandro Buss Becker, Dr.  
Evaluator  
UFSC/CTC/DAS

Prof. Eduardo Camponogara, Dr.  
Board President  
UFSC/CTC/DAS

## **Acknowledgements**

Foremost, I would like to thank my fiancée, Lara Dutra. It would not be possible to carry out this project without the support you provided. You are a very special person with huge contributions to this work. Furthermore, thanks to my parents for always motivating me to invest in my education.

I would also like to express my sincere gratitude to the Fraunhofer Institute for Production Technology and the colleagues at the production quality department for the opportunity to explore this topic in such a trailblazer institute for applied science. Special thanks to the company supervisor M.Sc. Hendrik Mende, who was always available to provide any help and acted as a true mentor. The knowledge acquired in this experience had an important impact on my formation.

Finally, I would like to thank Professor Dr. Jomi Fred Hübner for the given support as the project advisor. I could not forget to thank the Federal University of Santa Catarina for all the amazing opportunities provided during the whole Control and Automation Engineering course.

## Abstract

Among the production quality department's attributions at the Fraunhofer Institute for Production Technology IPT is the development of solutions for manufacturing using machine learning models. Many of these projects are carried out in partnership with third-party companies. The institute has well-established processes for the research phase where the problem in the partner's production line is explored together with the available static data following the CRISP-DM methodology. These procedures have a machine learning model that satisfies the metrics specified and the necessary data treatments to feed it as its main results. However, the deployment step where the model is embedded in a robust application capable of being implemented within the manufacturing environment took more time than expected and resulted in significant project delays in past experiences. The department analyzed that this happens partially due to the lack of automation on integration steps and development guidelines. Therefore, the solutions proposed by this FPW involve the creation of a continuous integration pipeline capable of building, testing, and releasing the application automatically during its development, and a series of general UML diagrams that can be adapted to different machine learning solutions built for manufacturing. A data science project previously elaborated by the institute was converted into an application developed and deployed in two different simulated environment architectures using these solutions. The results achieved indicate an increase in the development agility and the software structure robustness. The applications were deployed following basic steps and integrated correctly in the environments, which demonstrates the solution benefits and motivates its usage in non-simulated environments in future projects.

**Keywords:** Production quality. Machine learning. Continuous integration. UML diagrams.

## Resumo

Entre as atribuições do departamento de qualidade da produção do Instituto Fraunhofer para Tecnologia da Produção IPT está a resolução de problemas de manufatura através da implementação de modelos de aprendizado de máquina. Muitos desses projetos se dão em parcerias estabelecidas com empresas terceiras. O instituto contém processos e fases bem estabelecidas para as etapas de pesquisa onde o problema e o conjunto de dados estáticos são explorados seguindo a metodologia CRISP-DM até se obter um processo de tratamento de dados e um modelo que satisfaça as métricas projetadas. No entanto, o processo de incorporação do modelo em uma aplicação robusta capaz de ser implementada junto ao ambiente de manufatura mostrou-se demasiadamente lento em projetos passados devido, em parte, à falta de automatização de processos de integração e definições de desenvolvimento. As soluções propostas por este PFC constituem a criação de um *pipeline* de integração contínua capaz de construir, testar e disponibilizar a aplicação automaticamente durante seu desenvolvimento, e uma série de diagramas UML genéricos que podem ser adaptados para diferentes soluções de aprendizado de máquina voltado para manufatura. Um projeto de ciência de dados elaborado pelo instituto foi convertido em uma aplicação desenvolvida e implementada em dois tipos de ambientes simulados utilizando-se tais soluções. Os resultados alcançados sinalizaram maior agilidade de desenvolvimento e robustez satisfatória da arquitetura de *software*. As aplicações foram implantadas com facilidade e integraram-se corretamente aos ambientes, o que motiva o uso das soluções em projetos futuros com a implementação em ambientes não simulados.

**Palavras-chave:** Qualidade da produção. Aprendizado de máquina. Integração contínua. Diagramas UML.

## List of Figures

Figure 1 – Simplified workflow of the institute projects with companies . . . . .	18
Figure 2 – Implementation approach overview used on precedent project . . . .	21
Figure 3 – Solution overview . . . . .	23
Figure 4 – CRISP-DM diagram . . . . .	26
Figure 5 – DevOps cycle stages example . . . . .	28
Figure 6 – Square/Rectangle problem that illustrates LSP . . . . .	31
Figure 7 – Apache Kafka components overview . . . . .	38
Figure 8 – Docker containers architecture vs virtual machines architecture . . .	40
Figure 9 – Data source package on the general class diagram . . . . .	45
Figure 10 – Model package on the general class diagram . . . . .	46
Figure 11 – Configuration package on the general class diagram . . . . .	47
Figure 12 – Main controller and data preparation classes . . . . .	49
Figure 13 – General sequence diagram . . . . .	50
Figure 14 – Architecture patterns . . . . .	51
Figure 15 – Stream package in the general class diagram adaptation for stream- ing architecture . . . . .	52
Figure 16 – Stream controllers in the general class diagram adaptation for stream- ing architecture . . . . .	53
Figure 17 – Test pyramid of the machine learning applications to be developed .	59
Figure 18 – Tests report page inside the GitLab platform . . . . .	60
Figure 19 – Releases page inside GitLab platform generated by the CI pipeline .	63
Figure 20 – Resulting pipeline sequence to achieve the release stage . . . . .	64
Figure 21 – Flowchart containing the CI pipeline execution logic . . . . .	64
Figure 22 – Relational database diagram of the simulated environment . . . . .	68
Figure 23 – Shared database environment . . . . .	70
Figure 24 – Streaming environment . . . . .	70
Figure 25 – Kafka topics distribution . . . . .	73
Figure 26 – Pipelines executed in the released tags of the shared database appli- cation . . . . .	76
Figure 27 – Pipelines executed in the released tags of the streaming application	76
Figure 28 – Messages left behind in each consumer according to the Confluent control center after the execution of the worst case scenario . . . . .	78
Figure 29 – General class diagram for production applications . . . . .	85
Figure 30 – General class diagram for production applications with streaming architecture . . . . .	87
Figure 31 – README file of the shared database application for the use case validation . . . . .	97



Figure 32 – README file of the streaming application for the use case validation 98

## List of Tables

Table 1 – Most relevant available data for the tool condition prediction . . . . .	66
Table 2 – Configuration parameters of the shared database application . . . . .	71
Table 3 – New configuration parameters used in the streaming application . . . . .	73
Table 4 – Pipelines executed during the applications development . . . . .	75

## **List of abbreviations and acronyms**

AI	Artificial Intelligence
CD	Continuous Delivery
CI	Continuous Integration
CRISP-DM	Cross-Industry Standard Process for Data Mining
DB	Database
DSP	Dependency Inversion Principle
FPW	Final Project Work
IPT	Fraunhofer Institute for Production Technology
ISP	Interface Segregation Principle
IT	Information Technology
JSON	JavaScript Object Notation
LSP	Liskov Substitution Principle
ML	Machine Learning
OCP	Open-Closed Principle
ORM	Object Relational Mapper
OS	Operational System
R&D	Research and Development
SOLID	SRP, OCP, LSP, ISP, and DIP
SQL	Structured Query Language
SRP	Single Responsibility Principle
UML	Unified Modeling Language
XML	Extensible Markup Language

## Contents

<b>1</b>	<b>INTRODUCTION</b>	<b>13</b>
1.1	OBJECTIVES	14
<b>1.1.1</b>	<b>Main objective</b>	<b>14</b>
<b>1.1.2</b>	<b>Specific objectives</b>	<b>14</b>
1.2	METHODOLOGY	15
1.3	DOCUMENT STRUCTURE	15
<b>2</b>	<b>SCENARIO DESCRIPTION</b>	<b>17</b>
2.1	FRAUNHOFER-GESELLSCHAFT	17
2.2	INSTITUTE FOR PRODUCTION TECHNOLOGY	17
<b>2.2.1</b>	<b>Business model</b>	<b>17</b>
<b>2.2.2</b>	<b>Production quality department</b>	<b>18</b>
2.3	PROBLEM DESCRIPTION AND MOTIVATION	19
<b>2.3.1</b>	<b>Precedent project</b>	<b>20</b>
2.4	PROPOSED SOLUTION	21
<b>3</b>	<b>BACKGROUND</b>	<b>24</b>
3.1	MACHINE LEARNING	24
<b>3.1.1</b>	<b>Logistic Regression</b>	<b>24</b>
<b>3.1.2</b>	<b>Random Forest Classifier</b>	<b>24</b>
3.2	CRISP-DM	24
3.3	SOFTWARE ENGINEERING	27
<b>3.3.1</b>	<b>DevOps</b>	<b>27</b>
3.3.1.1	Continuous integration and continuous delivery	28
<b>3.3.2</b>	<b>MLOps</b>	<b>29</b>
<b>3.3.3</b>	<b>SOLID Principles</b>	<b>30</b>
<b>3.3.4</b>	<b>Software tests</b>	<b>32</b>
3.3.4.1	Test doubles	32
3.3.4.2	Unit tests	33
3.3.4.3	Integration tests	34
3.3.4.4	Contract tests	34
3.3.4.5	End-to-end tests	35
3.4	SOFTWARE ARCHITECTURES AND TOOLS	36
<b>3.4.1</b>	<b>Event stream processing</b>	<b>36</b>
3.4.1.1	Apache Kafka	37
<b>3.4.2</b>	<b>Docker</b>	<b>38</b>
<b>4</b>	<b>SOLUTION DESIGN</b>	<b>41</b>
4.1	SYSTEM REQUIREMENTS	41
4.2	GENERAL CLASS DIAGRAM	43

4.2.1	<b>Data source package</b>	43
4.2.2	<b>Model package</b>	44
4.2.3	<b>Configuration package</b>	46
4.2.4	<b>Controller and data preparation classes</b>	47
4.3	GENERAL SEQUENCE DIAGRAM	49
4.4	DEPLOYMENT ARCHITECTURE	50
5	<b>DEVELOPMENT</b>	<b>54</b>
5.1	CI PIPELINE	54
5.1.1	<b>Build stage</b>	<b>56</b>
5.1.2	<b>Test stage</b>	<b>58</b>
5.1.3	<b>Release stage</b>	<b>61</b>
5.1.4	<b>General aspects</b>	<b>63</b>
5.2	VALIDATION USE CASE	65
5.2.1	<b>Environment simulation</b>	<b>67</b>
5.2.2	<b>Shared database application</b>	<b>70</b>
5.2.3	<b>Streaming application</b>	<b>72</b>
6	<b>RESULTS</b>	<b>74</b>
6.1	CI PIPELINE	74
6.2	DEPLOYMENT APPLICATIONS TESTS	77
6.3	SOLUTION OVERVIEW	78
7	<b>CONCLUSION</b>	<b>79</b>
7.1	FUTURE PROJECTS	80
	<b>BIBLIOGRAPHY</b>	<b>81</b>
	<b>APPENDIX A – COMPLETE GENERAL CLASS DIAGRAM</b>	<b>85</b>
	<b>APPENDIX B – COMPLETE CLASS DIAGRAM FOR STREAMING ARCHITECTURE</b>	<b>87</b>
	<b>APPENDIX C – NEW PROJECT CREATION GUIDELINES</b>	<b>88</b>
	<b>APPENDIX D – DEVELOPMENT GUIDELINES</b>	<b>90</b>
	<b>APPENDIX E – PROJECT MANAGEMENT GUIDELINES</b>	<b>93</b>
	<b>APPENDIX F – TESTING GUIDELINES</b>	<b>94</b>
	<b>APPENDIX G – APPLICATIONS' README FILES</b>	<b>97</b>

## 1 Introduction

When dealing with complex tasks inside a company the idea of bringing agility and automation to the processes is usually stimulated. Making things faster and with better quality will consequently save time for the company and result in money savings, sometimes even revealing new business lines.

In this spectrum, artificial intelligence (AI) plays an important role to bring agility to manufacturing processes. Modern production facilities tend to produce a considerable amount of data and the ability to convert them into useful key performance indicators it is already an essential management activity. Moreover, the usage of business intelligence software to exhibit the business data into convenient dashboards, charts, and graphs guide the decision-making process in companies of different segments (IBM, 2021). However, with machine learning (ML) applications the analysis of data goes beyond visualization. It makes it possible to develop models that can predict failures and undesired behaviors, automate tasks, and optimize processes.

The path to building these models usually includes having knowledge in the domain, data science proceedings, and software engineering. Thus, the first model prototypes are developed in experimentation environments with researches that go from data understanding to the tuning and evaluation of the best performing machine learning algorithms. For instance, the CRISP-DM methodology explained in section 3.2 breaks ML projects down into analytical steps necessary to achieve the best results.

Nonetheless, reaching a suitable model is not the final step of the AI implementation in production. It is still necessary to perform the deployment stage, where, in general terms, the models have to be integrated into the daily activities on the production line accordingly with its specifications. For example, if a company wants a ML model to constantly monitor data from manufacturing to generate a warning when a failure is predicted, the deployment stage will probably include the development of an application that can consume data directly from the plant environment, make the necessary transformations, feed the model, and perform actions based on the predictions or make them available to the responsible team.

Since the ML model is sometimes seen as the final product of such an application, the deployment stage is occasionally ignored in the first phases of development. This results in many difficulties for the team to properly implement the project on the company wasting much more time than expected. On the other hand, using techniques that integrate development and operations steps from the beginning of the project is a difficult task when no standards or procedures for the development are defined.

However, the lack of a development culture is not the only problem. A slight change in the focus of the solution implementation can culminate in the usage of a completely different method. The existing processes to deploy ML projects tend to

investigate general procedures that do not consider the nuances of each application type. As an emerging technology, the architectural patterns and operational definitions are still being deeply explored, which makes it necessary to consider the company's specific needs when seeking a solution to this stage.

The Fraunhofer Institute for Production Technology, a reference in production solutions, has also identified these problems in some of their machine learning projects. Therefore, this project will explore the proceeding of such applications and propose a cycle that properly integrates development and operations turned to AI in production considering the existing process and tools used by the institute. The focus is on automating the development steps where possible, having the deployment as the main project's motivation, even though it will not limit itself to this stage.

## 1.1 Objectives

This project aims to explore the whole lifecycle of machine learning models in production, having the objectives described in this section.

### 1.1.1 Main objective

Define and automate a continuous integration pipeline suitable for the ML applications turned to production, developed by the Fraunhofer Institute for Production Technology, to accelerate the path to deployment and enhance the software quality.

Furthermore, validate the built pipeline developing a deployment application for a use case scenario in production, where the needed improvements must be observed and refactored back on the pipeline.

### 1.1.2 Specific objectives

- Build a repository on a git platform with the continuous integration pipeline stages defined and configured.
- Define a set of code best practices for deployable AI applications.
- Build general code structure diagrams to be a reference for the development.
- Define a testing pipeline that fits the most into the deployment applications, explaining how to structure and develop the software tests that will compound it.
- Provide integration between project management best practices and the git platform.
- Define the rules that will guide the management of branches during the development of the deployment applications.

- Integrate adequate tools that can bring agility and automation to different processes of the continuous integration pipeline.
- Make it possible to run the application regardless of the environment.
- Provide orientation for the code documentation.
- Write guideline documents for all the defined standards and configurations. Having easily readable material available to serve as a reference for the development.

## 1.2 Methodology

The early stages of this project were composed of a series of meetings with the institute presenting the main problem and the areas that it embraces. After performing research on that areas, a discussion on the solution fields to explore and to gain a common understanding was held. Especially to ensure that the student and the institute had the same comprehension of the project's goals and steps.

After this research exploration, meetings with other colleagues from the department were crucial to limit the scope of the project to the most important and urgent points. The focus on building a solid base from which future research activities can be derived was also defined.

Regarding the ML projects that are the base for this work, CRISP-DM is the methodology followed to describe the data science lifecycle. The goal of this project includes giving better directions on how to properly implement some of its stages.

Finally, a schedule of the activities was made and, during the whole project, an agile methodology was followed based on sprints of one week. In each weekly meeting, the previous tasks were reviewed and the new sprint was planned. Furthermore, the challenges found on a precedent project inside the institute were one of the main motivations for this work. Therefore, several meetings were scheduled with the team that has worked on it to ensure that the development was covering most of the challenges and avoiding wasting time with definitions that had already been explored.

## 1.3 Document Structure

In chapter 2, the scenario where the project took place is described. More details about the IPT and its business model are provided together with the problems that motivate the work performed. Moreover, the proposed solution is described for the first time in this document.

Chapter 3 contains the background concepts necessary to understand the solution details and the path to achieve it. Research fields, such as DevOps and event streaming, are explained inside this chapter, as well as the operation of some of the tools used, such as Docker and Apache Kafka.



In chapter 4, the solution design is described. The requirements of the systems to be implemented are defined and the general Unified Modeling Language (UML) diagrams for ML applications built for manufacturing, one of the main results of the project, are presented.

In chapter 5, the solutions' implementation is described. The applications elaborated and how they were applied in the institute's architecture, the technologies used, and the environments simulated are exhibited.

Chapter 6 evaluates the solution's impacts and the results that it brought to solve the problem described in chapter 2.

Finally, chapter 7 makes a conclusion analyzing the relation of the project with the Control and Automation Engineering course and bringing suggestions for future projects.

## 2 Scenario Description

In this chapter, the scenario that embraces this project will be clarified. It will first give an overview of the business model and the guidance of machine learning projects inside the partner institute. Then, the problem that motivates this work will be better described followed by the planned solution.

### 2.1 Fraunhofer-Gesellschaft

The Fraunhofer-Gesellschaft is the organization leader in applied research in Europe (FRAUNHOFER-GESELLSCHAFT, 2020). Founded in 1949, the non-profit organization has been playing an important role in the worldwide innovation sector configuring itself as a key bridge between the research and the industrial environment in several fields.

The structure of the organization is divided into 75 institutes and research institutions located in Germany, the country in which stays its headquarters, and other affiliations throughout the globe.

### 2.2 Institute for Production Technology

The Fraunhofer Institute for Production Technology (IPT) is among the eleven institutes that compose the Fraunhofer-Gesellschaft's Group for Production (Fraunhofer IPT, 2020). Located in Aachen, Germany, IPT has already developed several research and development projects within important partner companies to explore different trends in the manufacturing industry.

Focused in the fields of process technology, production machinery, technology management, production quality, and metrology (Fraunhofer IPT, 2019b), the institute seeks, since the beginning of its activities in 1980, to combine knowledge of different areas to come up with new products and solutions that can help to improve the manufacturing sector worldwide. Therefore, IPT has "transfer research findings into economically viable and unique innovations in the field of production" (Fraunhofer IPT, 2019c) as one of its mission statements.

Moreover, the institute acts, since 1994, to promote technology transfer between the European and American markets in partnership with the Fraunhofer Center for Manufacturing Innovation in Boston, USA (Fraunhofer IPT, 2019a). Thus, configuring itself again as an important international actor for the production technology improvement.

#### 2.2.1 Business model

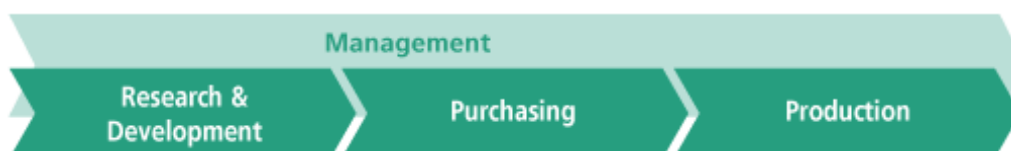
IPT develops many projects with different clients from the manufacturing industry usually intending to optimize processes or develop new products for modern produc-

tion facilities. These projects seek to analyze all the production line steps and their interactions to have a result that fits properly on the client's business reality.

To explore the clients' needs, the institute's big projects are usually covered by three main stages. First, the research and development (R&D) phase will identify new technologies, make conceptual models and develop prototypes that will fill the main gaps found in the client's process. This stage focus on optimizations that can lead the customer to achieve higher competitiveness in the market. Then, the purchasing phase will make a diagnosis of the company's relation with their suppliers trying to find improvement points since this is considered a crucial sector by the institute. Finally, there is the production phase which emphasizes the institute's focus on developing groundbreaking technologies that guide their clients and, therefore, the manufacturing sector as a whole, to be constantly evolving their production systems to an optimal stage. (Fraunhofer IPT, 2019d)

During all the previously mentioned phases, the institute also analyzes the management of operations and technologies applied by their clients. If necessary, IPT uses its expertise in this field to propose changes and improvements also on the business management level. This simplified stages workflow when working with clients can be visualized in Figure 1.

Figure 1 – Simplified workflow of the institute projects with companies



Source: (Fraunhofer IPT, 2019d)

These projects in cooperation with partner companies of industry represent a core part of the institute's business model. However, it is important to mention that IPT also receives funds from public organizations, such as the European Union and different German federal ministries (Fraunhofer IPT, 2022), especially for R&D projects. This collaboration with research funding institutions also configures itself as a key factor in the institute's activities.

## 2.2.2 Production quality department

To reach the objectives established by the project's phases in section 2.2.1, the institute is divided into different departments that focus their expertise on specific research areas. Among them, there is the production quality department in which this project took place.

The institute's work guidelines are naturally valid for this department, however, the focus is directed to projects that aim to improve the quality of manufacturing products and processes. To achieve that, the division has focused on having a data-driven approach building through the last years a broad knowledge especially on the data acquirement and availability from production, the analysis of this data, and how to implement actions based on these analyses (Fraunhofer IPT, 2021).

Another subdivision inside the department is the Automated Machine Learning group, where the main mission is to develop automation solutions for the whole ML pipeline. To achieve that, the team leads partnerships that pursue solving production quality problems with machine learning models capable of making predictions of anomalies or undesired behaviors in manufacturing.

The projects of this subdivision are the ones on which the solution to be described by this report aims to focus. In these cases, the manufacturing company usually presents the idea of the problem they have on their production line, sometimes after making a detailed exploration of it, and at other with this exploration to be made by the institute. After discussing the problem and defining the objectives, the department's team, following the Cross-Industry Standard Process for Data Mining (CRISP-DM) steps, better described in section 3.2, receives some static data files from the real plant and goes through a long data analysis work and many experimentations to come up with an adequate machine learning model. Then, it can generate predictions that fulfill the established goals and make it possible to act on the production before the undesired behavior happens, therefore enhancing its quality and saving money.

With this study, the institute can then provide this know-how or the ML models to the client which can build the best structure to implement it in its daily activities. As mentioned, the process to achieve these machine learning models and knowledge about the production facility is very research-centric embracing many experiments. To have an adequate environment for this type of operation, the department usually works with Jupyter Notebooks <sup>1</sup>.

### 2.3 Problem description and motivation

With the general process of machine learning projects inside the institute described, it is possible to relate it with the problems briefly introduced in Chapter 1. In many partnerships, the main goal is to properly identify a problem in the production line and achieve knowledge on how to solve it using artificial intelligence. In such cases, the outputs that come from the data science research - sometimes even resulting in a ML

<sup>1</sup> Jupyter Notebook is a web-based application that provides the ability to develop Python programs using interactive computing. Furthermore, it represents not only the code but also exploratory text, mathematical expressions, images, and other rich media representations of objects. (Jupyter Team, 2015)

model serialized on a file - are an adequate final solution for the partner company.

However, the institute realized that implementing this research knowledge is not a simple task. As mentioned earlier, the deployment process - described as the final stage of the CRISP-DM methodology - can have multiple operational problems. To reach an appropriate application that will be running in the production environment it will usually be necessary to deal with the communication with databases and other microservices. Furthermore, the implementation aims to achieve a software easy to understand and with a high level of maintainability, a state which demands the usage of different patterns during code development.

These characteristics are usually not deemed in programs with a relevant amount of experimentation on its development, where a considerable part of the produced code will not even be forming the final application. On the other hand, the experimentation stage is crucial to achieving the best project outcomes. Using strict code structures and architectures since the beginning of ML projects, for example, could result in a very slow process where it is really hard to reach important data and models understanding.

### **2.3.1 Precedent project**

In a specific precedent project, from which the observations made are one of the main motivations for this project, the necessity of having a better transition from the research application to one deployable in manufacturing environments became clear to the institute. This case was one of the first in which the partner company also showed interest to have the deployment stage as part of the delivered solution.

Since this was a definition that changed while the project was in course, the team had to take the developed research application and try to adequate it to production without having time to define work guidelines and consider integration aspects.

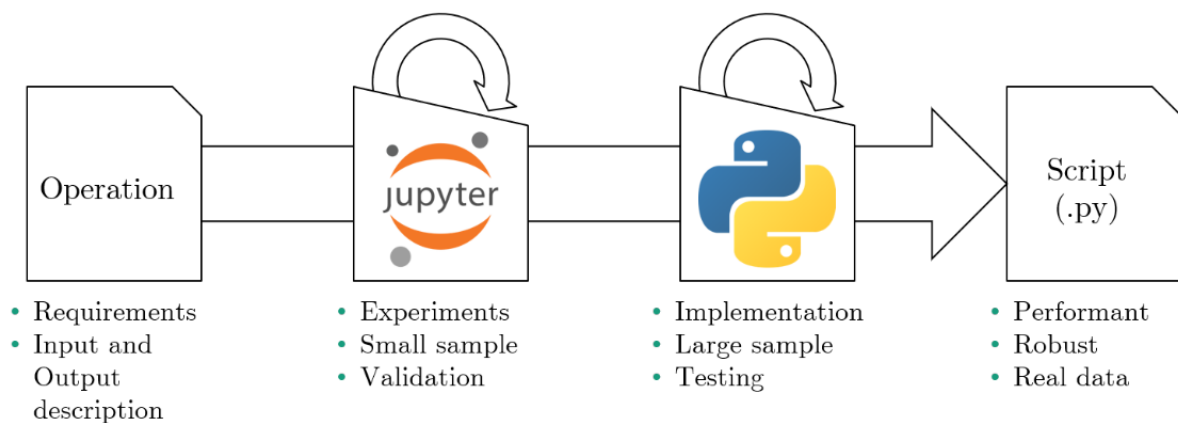
The Jupyter Notebooks were then converted in object-oriented Python scripts and the solution was correctly deployed. However, this process took a long and exhaustive period that consumed much more of the team's time than expected. To illustrate that, there were moments where the scripts were working fine inside IPT's environment but have to be sent for testing to the partner company since their environment is disconnected from the institute. Then, also taking time from the company employers, the tests showed that the scripts were not ready for production and had to come back to development. Having in mind that this was a slow process since it was necessary to send the whole program from one team to another, it exemplifies an integration problem in a scenario that could happen in future projects led by the institute.

Furthermore, new classes and packages had to be designed to transform the application. For the software to run in production, it was necessary to develop the communication with the data source and the preparation of live data while also considering maintainability and scalability aspects. Thus, the ML models that were considered the

final solution at the beginning of the project still had a long period until being deployed in production. This lack of preparation for this stage is something harmful to the institute's project since it is likely that it will consume a great part of their schedules.

Figure 2 shows the simplified implementation process used on this precedent project, going from the requirements and problems definition, passing through the experimentation phase using Jupyter Notebooks until the implementation using Python scripts.

Figure 2 – Implementation approach overview used on precedent project



Source: (BELCK, 2020)

Even though the models developed had used advanced data science skills and provided an efficient and innovative solution to predict manufacturing failures to the partner company, the project loses part of its efficiency when it is not possible to demonstrate the results integrated into a production environment.

All the problems listed in this section and the fact that they happened on a real project inside the institute enhance and motivate the implementation of a solution that can make the development and deployment process of such applications faster and more reliable, seeking to achieve the objectives already presented in section 1.1.

## 2.4 Proposed solution

Having the scenario and previous problems faced by the institute in mind, it is possible to formulate the steps to be implemented aiming to improve the integration between development and operation tasks for this type of application.

The main idea of the solution is to build a framework that can guide the development and speed up the deployment process. This can be achieved by the implementation of a continuous integration pipeline, better described in section 3.3.1.1, that will run automatically in some stages of the project. Then, integration tasks containing all the

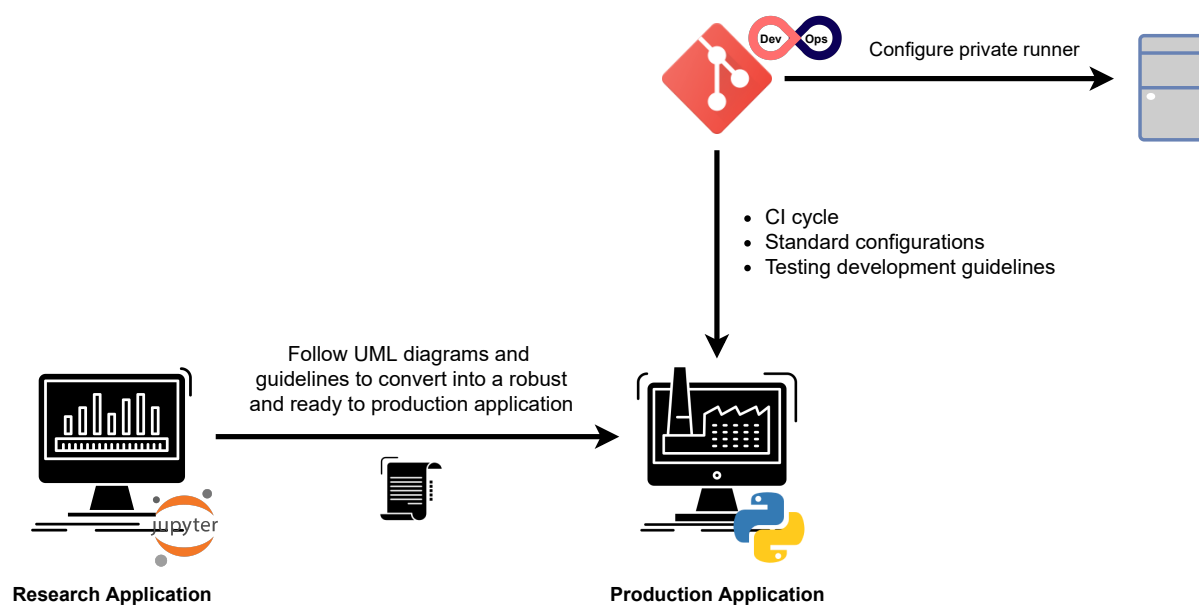
needed configuration to test and build the application will be automatically performed and will result in an instance ready to follow to production. Therefore, the majority of the operational tasks will not request manual steps, which will fulfill the lack of standards on this phase of the precedent project.

This pipeline can then be configured on a template git repository, which in turn can be imported inside another repository when starting a new machine learning project. However, the actions to be built inside the pipeline need to be executed in some machine. Thus, it is also necessary to configure an actions runner in a private computer that will be connected with our repository and will run the process when they are activated by the triggers established on the cycle.

Additionally, the research stage where experiments are led by the team to define the best solution to the problem remains an important step, just like on the precedent project. Although the freedom to try different approaches is one of the crucial characteristics of this phase, the solution to be implemented can also define important guidelines and tools to solve the most common problems found during its development, e.g. the tracking of the different experiments metrics and parameters. Therefore, the implemented actions will not only focus on the deployment but also on the entire development process.

Another important observation is that, despite the applications' peculiarities and diversity, general machine learning projects applied to production have many points in common. Hence, in addition to the integration pipeline, a crucial stage of the proposed solution is also to define diagrams and code structures that will be a reference to guide and speed up the evolution of the research application into the one to be implemented on production. In Figure 3 an overview of the solution idea is shown.

Figure 3 – Solution overview



Briefly, this project believes that these actions implemented together will form a solid structure that will guide machine learning projects to easily deploy in the client's environment, in that way enabling further exploration of this field.



### 3 Background

This chapter will introduce and explain the theories and concepts used in the solution. Many of the aspects explained here will be mentioned over the other document's chapters, therefore being crucial to understanding them. Among the topics here clarified there is the brief definition of machine learning models, the CRISP-DM methodology used in the projects for which the solutions are directed, and the explanation of software engineering concepts, architectures, and tools used in the implementation.

#### 3.1 Machine Learning

The applications that seek the deployment of ML models in a manufacturing environment are the ones to which the project's solutions are directed. Therefore, it is important to define a ML model as a file that contains the expression of an algorithm and that is trained with relevant data to find patterns or make predictions when exposed to new data (PARSONS, 2021).

However, since the best-performing models and the data preparation steps are built in the research application, it is not necessary to have deep knowledge in this field to understand the deployment of the production application. Some ML model types are briefly explained in this section to clarify the input models of the validation use case in Chapter 5.

##### 3.1.1 Logistic Regression

Logistic regression is a type of ML model commonly used when the target variable is categorical, especially in binary analysis. Also known as the *logit* model, it is based on the logistic function  $f(z) = \frac{1}{1+e^{-z}}$ , which will always return a value inside the  $[0, 1]$  interval regardless of the value of  $z$ . This characteristic motivates the model usage in cases where a probability of an event happening is the variable to be predicted. (KLEINBAUM et al., 2002)

##### 3.1.2 Random Forest Classifier

Random forest classifier is another type of ML model that aggregates different decision trees to make classification predictions. The output result is the class that appeared the most in the trees' results. Since these trees are uncorrelated, the model's result outperforms any of the constituent individual results. (YIU, 2019)

#### 3.2 CRISP-DM

As already mentioned a few times in this document, the CRISP-DM methodology (CHAPMAN et al., 2000) plays an important role in this project because it is the base

guidance used to explore the data from the manufacturing lines in most parts of the department's partnership projects.

Created with the purpose to standardize data mining processes, the methodology proposes a cycle composed of 6 main steps that became widely used on data science applications and it stands as the most popular project management framework in this field (SALTZ, 2020).

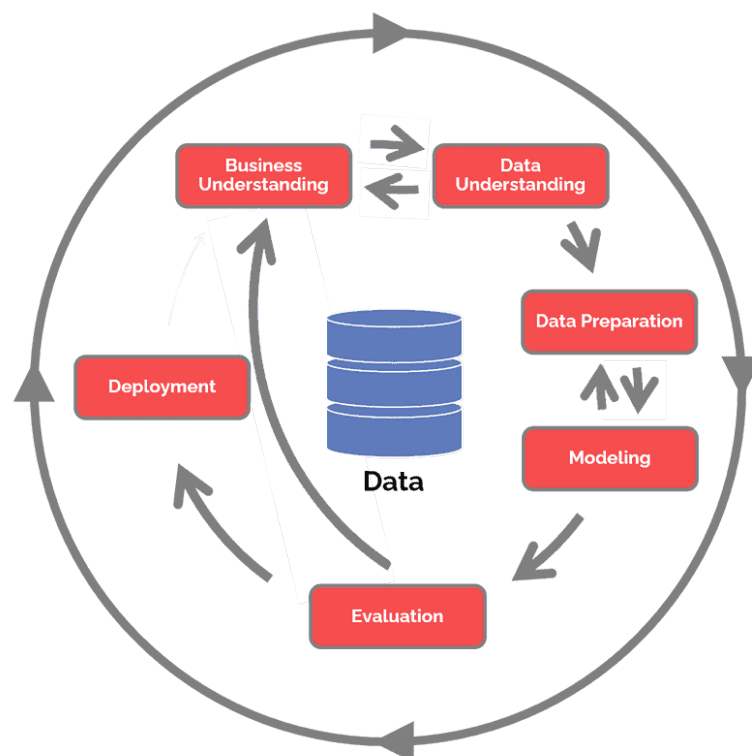
These proceedings have different workflows that aim to guide the data science team to know what are the next steps on each phase of their projects. The methodology stages are then briefly explained in the bullet points below, which are all based on (CHAPMAN et al., 2000).

- **Business understanding:** In the first stage the team must aim to understand the problem and requirements of the solution from a business perspective, usually elaborating a preliminary plan. Some project definitions are also established in this phase, such as if the application will be deployed or if it will stand just as a prototype.
- **Data understanding:** This is the stage where the first analytical steps with data are performed. Initiating with a data collection, the team should implement actions on it to properly understand the data and to be able to describe it deeper. The seeking for insights over the data, the verification of its quality, the detection of subsets, the identification of the data source in production, and other tasks with an exploration profile should also be executed.
- **Data preparation:** The main aspect that describes this stage is that its output should be the dataset with all the transformations that will be applied upon it, in other words, the dataset in the form that it will be consumed by the models. Therefore, among the actions the team should perform, there is the selection of data that will be used, data cleaning steps to enhance data quality, merge information that comes from different data sources or tables, develop the feature engineering to extract new attributes from the original raw data, and format other needed data attributes. This stage will already involve many code development to perform the listed activities.
- **Modeling:** The modeling stage will build the models and select the techniques to achieve them. This selection will imply analyzing different available approaches, searching optimal values for the models' parameters, and evaluating the technical results looking to the reached metrics. It is expected that, during the development of these tasks, the team will need to step back on the cycle and perform data preparation actions again.

- **Evaluation:** In this stage the evaluation of the models is performed considering the business objectives and restrictions. In other words, the team will ensure that the models do not have only good technical metrics but also fits properly in the considerations made on the first steps of the methodology and solves the existing problems. Then, it will be decided if the solution is ready to be deployed or if the cycle should restart with the business understanding.
- **Deployment:** The last stage will elaborate a plan to deploy the result achieved so far into the business. Usually, the customer will have active participation in this process that will also plan the monitoring and maintenance of the application, produce a final report, and make a review of to whole project listing improvement points.

In Figure 4, it is possible to see the CRISP-DM diagram which illustrates the methodology flow over the stages described. There is also an outer cycle that embraces all the steps demonstrating the cyclic nature of this type of project.

Figure 4 – CRISP-DM diagram



Source: (Data Science Process Alliance, 2018)

Having the stages description and the sequence of their execution in mind, it is possible to observe that some initial phases, especially data preparation and modeling,

will produce a lot of code that will not necessarily be on the final application. These two stages interact a lot with each other forming a mini loop inside the major cycle and, together, they create several experiments that have the metrics and parameters evaluated according to the project requirements. Therefore, they form what is defined as research application by this document and enhance the comprehension of why it is important to have a step with a flexible code structure inside the proposed solution.

Moreover, even though it has sub-processes described, the deployment stage does not state specifically how to implement the solution. This is because each data mining application will have its peculiarities and own requirements. (CHAPMAN et al., 2000) states that “the deployment phase can be as simple as generating a report or as complex as implementing a repeatable data mining process across the enterprise”. Considering that the type of applications explored here needs to be implemented in an environment different from the development one, becomes clear the importance of having a better structure that makes the deployment easier and that this solution does not come naturally from simple following CRISP-DM.

### 3.3 Software Engineering

A considerable part of this project will involve establishing and implementing definitions to guide and speed up the development, deployment, and maintenance of machine learning software applications. Software engineering can be described as the field that studies techniques to design, build, and test software aiming to satisfy the user requirements (MARTIN, 2021), which makes it important to understand some of its concepts in this section.

#### 3.3.1 DevOps

Being formed by the joint of the words “development” and “operations”, it is possible to initially define DevOps as a culture that promotes better interaction between the software development and the information technology operations teams (COURTE-MANCHE; MELL; GILLIS, 2021).

The DevOps movement started to be widely discussed around 2008 when many software companies noticed that the incorporation of agile methodologies was not enough to ensure fast implementation of the developed solutions. The structural separation between development and operations, from the different key performance indicators to the different leadership’s board, resulted in a culture where each team focuses only on their own tasks and demands. That, in turn, makes it harder to effectively deploy the solutions and deliver them to the clients (BUCHANAN, 2021).

It is easy to observe that this lack of integration among these phases relates a lot to the main problem to be explored by this project. The DevOps practices have the

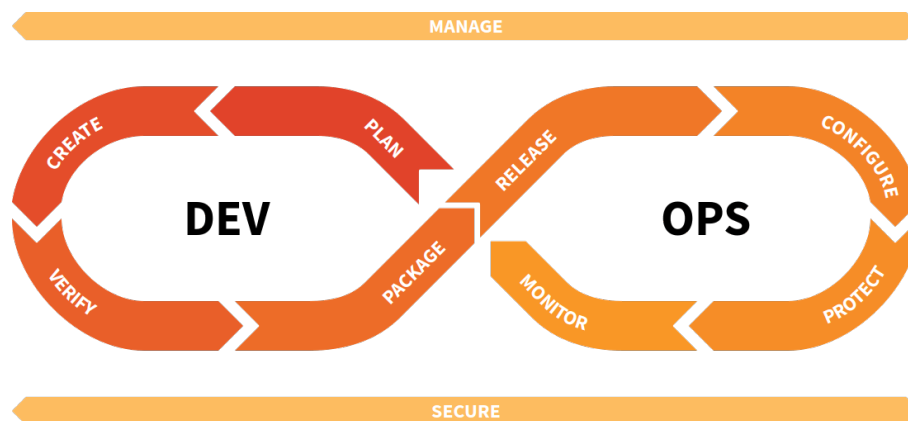
objective to fulfill this gap and that is why its definition is so crucial for the rest of this document.

To better exemplify what this concept means, an essential characteristic of its implementation is to create a DevOps pipeline: a set of automated processes that embraces both development and operations tasks executing them aiming to build the application, ensure its quality, and let it ready to be deployed on the production environment (HALL, 2021). The idea of bringing automation for this process is an important attribute of the solution proposed in section 2.4.

With this exemplification in mind, it is possible to improve the initially given definition of DevOps with the (BUCHANAN, 2021) statement that “DevOps touches every phase of the development and operations lifecycle. From planning and building to monitoring and iterating, DevOps brings together the skills, processes, and tools from every facet of an engineering and IT organization.”.

Furthermore, it is important to point out that the stages of DevOps cycles will vary from application to application. For example, a program being developed in a compiled language will have to be concerned about the compilation of the code on the build stage, while another one that uses only interpreted languages will have other concerns. Anyhow, in Figure 5 a general DevOps cycle proposal that properly illustrates the discussion so far is shown.

Figure 5 – DevOps cycle stages example



Source: (GITLAB, 2021)

### 3.3.1.1 Continuous integration and continuous delivery

To build the mentioned pipeline, one of the main DevOps best practices is the implementation of continuous integration (CI) and continuous delivery (CD). These two concepts carry the idea of automating the technical and management aspects of the whole development lifecycle.

As the name suggests, CI is the practice to be continuously integrating different versions of code inside a common repository. It allows developers to merge code changes on the same branch and also manage its different versions (REHKOPF, 2021b). Usually, a git repository provides the ability to develop these tasks, however, a solid workflow must be defined by the team.

Nonetheless, the application of continuous integration does not limit itself to having a well-configured git repository. The integration of code in real modern use cases also involves testing its quality, verifying syntax styles, and other integration tasks that will result in a much more reliable and scalable solution. These activities should be automated when possible and will usually include the use of different tools as support to speed up the development process. (REHKOPF, 2021b)

On the other hand, CD is a set of actions that aim to take the application with code integrated by CI and continuously coordinate the release of this program. Having automation again as a key value, CD just considers a new feature done when it is released to the end-user since this is the reason for the whole project. Following this concept will imply the understanding that there is no work done if it runs only on the machines of their developers. (REHKOPF, 2021a)

Again, it becomes clear how these concepts are important for the proposed solution. To be able to implement it, the particular CI/CD steps will have to be defined and configured for the general application and validated on a motivational project.

There is another concept, called continuous deployment, that embraces the standard DevOps pillars. In this process, the application is not only released automatically but also deployed to the customers without any human intervention (PITTET, 2021). Of course, the deployment stage will also be as automatized as possible in the DevOps cycle of the institute's applications, however, it will usually deal with a production environment disconnected from the development one, which can prevent the complete implementation of this last concept.

### 3.3.2 MLOps

Since the machine learning applications have a lifecycle that differs in many aspects from normal software applications, the set of aspects that can lead high-performance models to be continuously integrated and delivered to production are usually treated in a sub-field called MLOps.

Thus, the majority of the DevOps characteristics can be also considered for this sub-field, since its definition can be given by “an engineering discipline that aims to unify ML systems development (dev) and ML systems deployment (ops) to standardize and streamline the continuous delivery of high-performing models in production” (TYAGI, 2021). However, it is important to have in mind that MLOps will involve different steps and will probably form different cycles to reach an integrated result.

For instance, referring back to the proposed solution's brief explanation in section 2.4, the consideration of having a separated research application could already break some of the DevOps principles. But, since the proposal is dealing with a machine learning application, the experimentation stage is crucial for the delivery of a high-performing model.

In sum, it is important to have the MLOps concept well-defined in this document as a DevOps branch that adapts it to ML applications, differing sometimes from standards used on general software projects.

### 3.3.3 SOLID Principles

The SOLID principles are a set of development patterns used to bring a clean architecture for a software project. They will be used during the modeling of the general ML applications.

First defined by (MARTIM, 2002), the SOLID principles form an acronym with 5 major design principles that impact code architecture. Each one of them deals with a specific software modeling problem but, together, they aim to build a clean code guiding the developer on the arrangement of functions and data structures.

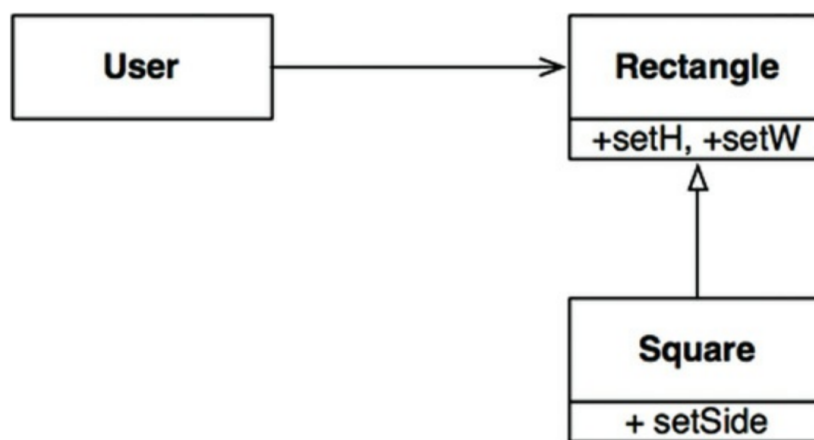
These principles are intended to be applied on the module level that will consequently guide the code implementation. Their goals are explained by (MARTIM, 2017) as the creation of mid-level software structures that can tolerate change, are easy to understand, and are the basis of components that can be used in many software systems.

Also in (MARTIM, 2017), a description of the principles focusing on their architectural implications can be found. The bullet points below will try to resume each letter of the acronym based on this reference.

- The letter S stands for the Single Responsibility Principle (SRP), which tells us that a software module should have one, and only one, reason to change. This does not mean that a function must deal with only one task, but it states that a module should have only one actor, which is its reason to change.
- The letter O stands for the Open-Closed Principle (OCP), stating that a software module should be open for extension but closed to modification. Whenever a software component is running in production without having an error, the implementation of a new task should not imply modifications on their existing structures, but the arrangement of the components should be already designed in a form that makes their extension easier. A dependency hierarchy will be crucial for preventing the high-level components from changes in lower-level components.
- The letter L stands for the Liskov Substitution Principle (LSP), which states, in an informal definition, that a sub-component should always be able to substitute

its parent-component without raising any error. This can be very related to the classes inheritance inside the object-oriented programming, as illustrated by one of the examples given by (MARTIM, 2017) where a class “Square” is created as a sub-class of “Rectangle”, shown in Figure 6. Although these forms share many aspects, the height and width of the “Rectangle” are independently immutable, which is not true to the “Square” and can confuse the user when it believes that is dealing with a “Rectangle”. The SOLID principles extend LSP to the architecture level using it as an important ally to implement a clean software design.

Figure 6 – Square/Rectangle problem that illustrates LSP



Source: (MARTIM, 2017)

- The letter I stands for the Interface Segregation Principle (ISP), stating that clients should not depend upon interfaces that they do not use. Making a class dependent on something that carries more information than what it needs will break the cohesion of the code architecture. Instead, this interface should be segregated into several interfaces that will be correlated only with objects that need to use them. Just like on the LSP, the word interface refers to any component that describes this implementation dependency and not to a specific programming language component.
- Finally, the letter D stands for the Dependency Inversion Principle (DSP), which states that software models should rely only on abstractions and not on concretions. Abstract objects will usually require fewer changes, therefore configuring itself as a more reliable structure. Together with OCP, this principle is crucial for ensuring code flexibility. With its implementation, high-level objects will never depend on low-level objects implementation but will always be opened to extend the abstract concept of the object to them.



The implementation of these principles on the modules that will compound the design components does not completely ensure a system without undesired bad practices in the code architecture. Besides SOLID, it is also important to have other software engineering principles in mind, such as the cohesion of components which will tell how to arrange program classes into different source files - in the case of interpreted languages.

However, they will serve as important guidance to achieve a clean architecture when designing the modules, classes, and components of the general application solution. Having in mind that this solution will be used as a reference to be adapted for different machine learning applications developed by different teams, it is very important to be easy to understand its concepts. This supports the necessity of the implementation of the SOLID principles for this project.

### **3.3.4 Software tests**

To establish an integrated development cycle in a software project it is very important to have a clear tests' chain. These software tests can be used to ensure different system behaviors. The analysis if a new small feature did not break old ones and the examination if the whole program's workflow is working integrated with production are some of the advantage examples.

With the dominance of agile methodologies on software projects, being able to continuously ensure that a program is working as it supposes to be turned into a crucial task that enhanced the importance of software tests automation. This relevance has even contributed to the invention of programming methodologies that have them as a base. Thus, different concepts of software tests were designed and the methods to implement them depend on the requirements and environments of each project. (FOWLER, 2019)

In this section, the most relevant tests' aspects to this project have their definition explained considering that the creation of an automated testing pipeline is one of the proposed solution objectives.

#### **3.3.4.1 Test doubles**

Before reaching the different software test types, it is important to establish the concept of test doubles to understand how these examination methods work. The objects and functions inside a program tend to be tied together in different workflows containing a correct order of how the tasks should be executed on production. Therefore, many of the actions to be tested are programmed to only initialize after a series of events using several dependencies created on previous steps. To make the verification easier and avoid executing a large sequence of operations each time, software tests create

fake objects - test doubles - to replace real dependencies which the test does not have control of.

There are different classifications of these fake objects used for testing purposes. (FOWLER, 2016) divides them into five groups according to their usage, but the classification given by (OSHEROVE, 2014) is simpler and more practical dividing test doubles mainly into stubs and mocks.

Thus, the creation of stubs and mocks is a crucial step in the development of software tests and their injection in the code can be made with the assistance of testing frameworks, which makes it important to know the definition of each one. A stub is a normal fake object created to replace an external dependency, tests can usually use several stubs. On the other hand, mocks are also fake objects but they are used to decide whether the test has passed or not. In other words, mocks are similar to stubs but it is used used to verify if the production code under test called the fake object as expected. (OSHEROVE, 2014)

#### 3.3.4.2 Unit tests

The base of tests pyramids is usually filled with this type of test. Unit tests are essentially designed inside software applications to test small units of work that execute a specific action when a method or a task is called. These tests are written in code usually on the same repository of their project, which allows their easy automation.

Furthermore, they are usually written with the support of a testing framework, and, using statements, they always check if the unit of work performed what was expected, which will decide if the test passes or fails. Thus, its structure will rarely be different from defining needed dependencies, executing the production method under test, and checking the results through assertion functions.

However, the misunderstanding of the unit tests characteristics can lead to a code with undesired behavior. Since the testing pipeline needs to have a fluid execution, defining how a unit test should behave is also necessary. In this document, the unit test definition stated by (OSHEROVE, 2014), which also lists some properties that relevant unit tests should have, will be considered when referring to this topic. Below, there are the explanation of some of these properties, based on (OSHEROVE, 2014), considered more relevant in the context of this work.

- **It should run quickly:** one of the main advantages of this type of test is the idea of checking the quality of several parts of the production code in a few seconds. When a test takes too much time to run it is probably dealing with dependencies that unit tests should not deal with.
- **It should be automated and repeatable:** as mentioned before, these tests are written with the assistance of a testing framework. Then, their run should be

created in an automated form that can be easily repeated.

- **It should be relevant tomorrow:** the value of implementing unit tests increases as the project advances since old unit tests will always be able to run and identify if some feature is broken. If a unit test loses its relevance tomorrow, then there is no meaning in implementing it in the first place.
- **It should be consistent in its results:** this crucial statement describes very well the nature of unit tests. Since they should not rely on anything outside the source code, they should always return the same result in different executions if nothing changed between them.
- **When it fails, it should be easy to detect what was expected and determine how to point out the problem in the production code:** the statements of a unit test should always be very clear. Their main advantage is to believe that when something breaks inside the code they will always successfully catch this error. However, just knowing that something is wrong is not enough, the developer should also be able to identify where is the problem and what caused it, so a strategy to solve it can be made.

#### 3.3.4.3 Integration tests

Unit tests are not enough to cover the verification of all the actions inside a software project. Some functionalities can demand less isolation to be tested, which can break certain unit tests characteristics. Integration tests are then used to fulfill this lack of verification and are defined by (OSHEROVE, 2014) as “any tests that aren’t fast and consistent and that use one or more real dependencies of the units under test”.

With that definition in mind, it is possible to understand the necessity to implement integration tests to keep a high level of maintainability in a complex project. Many features have to use real dependencies to be properly tested and even the communication between different packages inside the program can be hard to verify using the isolation demanded by the unit test properties.

This type of test can be usually implemented using the standard testing frameworks available for different code languages and it will compound one of the CI cycle stages to be developed. But on the other hand, it is important to differentiate its definition from the other test types to ensure that their execution will not culminate in significant pipeline delays.

#### 3.3.4.4 Contract tests

It is not common to have a completely isolated program on production applications. The source code to be developed has to communicate with different external

services to get access to some data, publish information, or execute specific actions.

To have fast tests on the source code methods that deal directly with external services, it is a good approach to make a test double containing the expected response from the third-party program and use it in unit and integration tests. However, if the external service changes its operations, such as the communication protocol or the response format, it could break the application without having the error identified by the implemented tests.

Contract tests are developed to solve this problem. Their concept relies on making a real request to the external service and comparing its response with the test double created to represent it on the other tests. A failure on the set of contract tests would mean that the contract between the application and the external service is broken, in other words, the source code was expecting different behavior from the third-party program and it has to be verified. (FOWLER, 2011)

In conclusion, contract tests are used to verify if external services are still working as expected by the application and its implementation can be very useful in projects that have to communicate with other programs, which will usually be the case of machine learning applications in production.

#### 3.3.4.5 End-to-end tests

The other test types previously described in this section had different levels of isolation from the other parts of the program. In the case of end-to-end tests, the isolation will be as low as possible since their idea is to test the whole program workflow from the beginning to the end. Every action that the project can perform should be executed and the different interactions with external dependencies, real databases, and other production hardware should be tested. (BOSE, 2021)

As it is possible to observe in its definition, this type of test can be hard to be implemented. A machine learning project, for example, may involve writing back predictions inside a database (DB) and an end-to-end test could be useful to analyze its entering execution. However, making a test to write things on the real production environment can be harmful for obvious reasons, so the necessity to have an easy roll-back mechanism or an entire separated environment to perform these tests makes its maintainability difficult.

Even though some frameworks can help with its implementation and automation, end-to-end tests can be very different from project to project since it relies on the hardware architecture. Therefore, tests that manually verify the whole program workflow are also considered end-to-end tests on this document.

Since it passes through the whole program stack without using test doubles, the end-to-end test can be powerful to find particular bugs that are being missed by the rest of the testing pipeline. But, as it was described, its implementation is itself a challenge,

which will imply the usage of only a few tests of this type on most projects. (FOWLER, 2013)

### 3.4 Software architectures and tools

During the solution development, different architectures were considered to provide a wide utilization range for the CI pipeline and the software modeling. Moreover, the validation use case where were performed tests to evaluate the project's outcomes had to use different tools to correctly simulate the environments. Therefore, it is important to clarify these topics for the understanding of Chapters 4 and 5.

#### 3.4.1 Event stream processing

General applications that deal with the incoming of new data are usually designed to receive it in batches. This standard design structure makes it easier to add the data on a storage system and, when appropriate, the applications can make a single request and receive the whole batch all at once. However, this architecture can be problematic in cases where the data does not have a proper beginning or end, which would imply arbitrary decisions to separate the batches.

The storage and processing of data based on event streams is a different architecture that gives a continuous flow to never-ending data sources and provides the ability to perform actions upon data in motion. Considering the modern applications' requirements, this is a crucial data processing method that brings agility and flexibility to production. It makes it easier for edge devices to publish or consume data from the streams instead of having to centralize it on the core storage first. (Confluent, 2021)

Moreover, data can be time-sensitive in some use cases, which means that it loses its value when time passes. For example, the information of sensors in a manufacturing plant that can be used to predict errors in the final product does not have too much value if it is analyzed only after the product is already manufactured. The system can prevent results outside the desired metrics if the information is processed earlier. Data streaming technologies enable a real-time reaction to data (Confluent, 2021) which can be very useful in critical systems.

Nonetheless, some fields need more attention when developing an application inside an event data streaming architecture. The flexibility to append new devices as consumers of a stream can imply a bad data consistency, a service can consume a piece of information that was already treated and modified inside another application. Furthermore, fault tolerance and data guarantees are usually characteristics intrinsic to the database management system in batch processing applications, while it has to be ensured in streaming since the data can come from different numerous sources. Thus, it is necessary to have all the possible data flows in mind when designing the use of

this structure in a project. (Confluent, 2021)

In machine learning projects the data can have different natural behaviors. Its source can be set on a unique central system or hundreds of devices and the data creation can be very slow or it can occur in events of milliseconds period, the fact is that these projects depend on data to work properly. Therefore, it is important to have the streaming technology concept well defined to be considered while proposing a fast deployment solution in this project.

#### 3.4.1.1 Apache Kafka

To use event stream processing in a project it is necessary to have a structure that allows it to happen. Problems like data availability and consistency have to be considered to have a reliable streaming architecture and it is not feasible to build this whole system from the scratch in every project.

Apache Kafka is an open-source streaming platform that provides the needed features to properly implement this architecture. Kafka provides three key resources to drive an application based on event streaming: the ability to publish and subscribe to streams of events, store streams of events for as long as necessary, and process streams of events as they occur or retrospectively. (Apache, 2021)

Additionally, the platform is built on a distributed and elastic structure, that makes it possible for the application to scale and handle big amounts of data. Some other concerns regarding security and data replication are also dealt with by Kafka. (Apache, 2021)

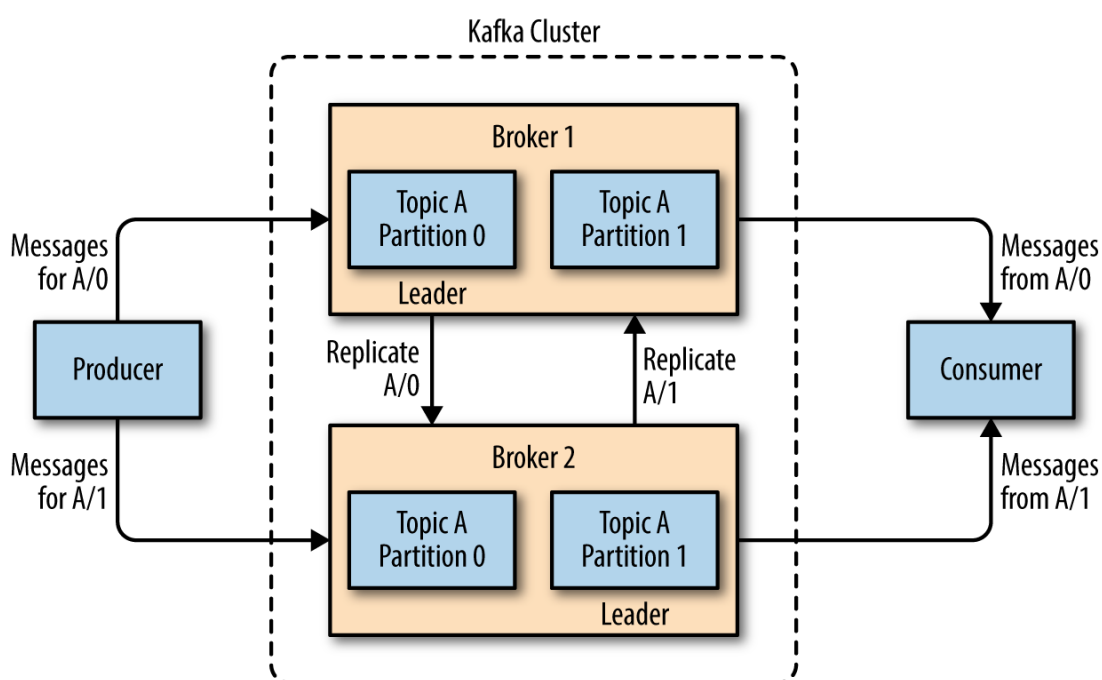
Applications and devices can communicate with Kafka in the form of clients, being them publishers (data writers) or subscribers (data readers). The interaction of these clients with the data and Kafka's simplified operation can be understood by comprehending its main objects, which are described below based on the explanations given by (NARKHEDE; SHAPIRA; PALINO, 2017).

- Messages: Since Kafka is event-oriented, messages are essentially the units of data that arrive on it. The data does not demand to be interpreted by Kafka, so it can have almost any format of information.
- Clusters and brokers: The broker is the Kafka server that coordinates the streaming operation. It is the broker that will receive the data from the different devices, make needed assignments, and store it in the host's disk. The server will also warn the consumer applications when there are new incoming data to process. In Kafka, these actions are usually performed using more than one broker that can communicate with each other and enhance the streaming quality and reliability, forming a Kafka Cluster.

- Topics: This is the way to separate and label data inside Kafka. The producers will choose a specific topic to publish a specific kind of data, which will allow other services to subscribe to that topic and read this data following the incoming order. This procedure represents the stream processing, making clear why topics are so important in Kafka's structure.
- Partitions: Topics are also divided into partitions, which is the instance where the data is written. Since a topic can have several partitions allocated in different servers, Kafka can provide data redundancy.

In Figure 7, it is possible to observe an example of a Kafka structure with data replicated in two different brokers. This simplified overview resumes how Kafka can be used to manage and stream data among different applications inside a project.

Figure 7 – Apache Kafka components overview



Source: (NARKHEDE; SHAPIRA; PALINO, 2017)

### 3.4.2 Docker

As mentioned in Chapter 2, one of the focuses of this project is to facilitate the deployment process, which can be done with the support of containers and images.

The development of an application does not have to deal only with the code but also with environment configurations. It is usually not possible to simply migrate the source code to production and run it without having to configure dependencies that were already built on the development environment. Due to the limitations of production

structures, which sometimes include not having access to the internet, this task of building a platform containing all the pre-requisites to run the application on a new computer can be difficult.

Docker provides the ability to package the application into one or several standardized units containing the source code, runtime, system tools, system libraries, and any other necessary setting to run the application. This creates a blueprint of how to build the desired system, which can then be executed to run inside a Docker Engine. (Docker Inc, 2021)

This blueprint is called Docker image and is used to solve the compatibility problem previously described. Since the system will always be entirely constructed based on its Docker image, it should be able to run exactly in the same way on every computer. When this image is set to run, it becomes a Docker container, which is the instance that will use the computer resources to perform the application actions. Again, the isolation of the container makes that every environment can run it without having to be concerned with any dependency, having in mind that all the needed configurations were correctly injected when creating the Docker image. (CHAWLA, 2020)

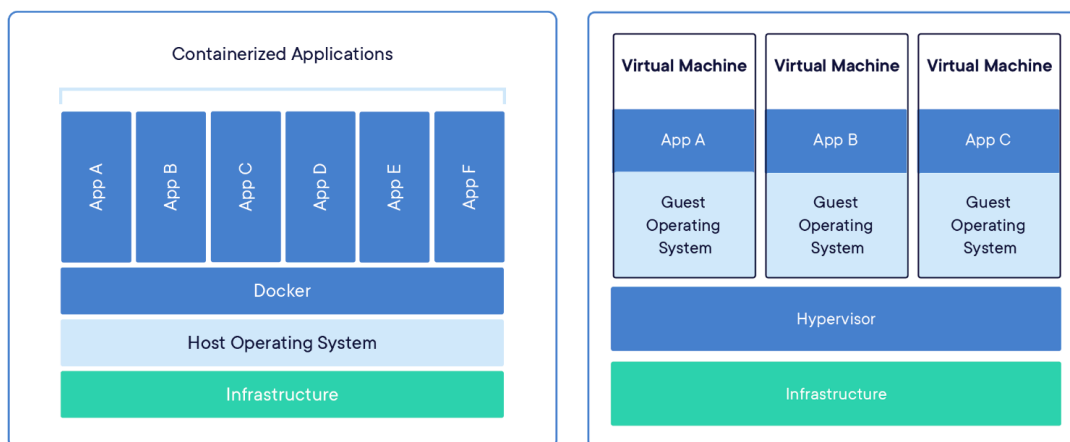
A Docker image is for its containers what a class is for its instances. The declaration of the class is what describes everything about that object, its methods, attributes, and connections, but the real world entity just exists when an instance of that class is created. In the same line, a Docker image is only used to perform real actions when a Docker container is instantiated. (CHAWLA, 2020)

The containers can have their execution compared to the usage of virtual machines due to their similar resource isolation. However, containers aim to package the application and its dependencies only, which allow them to use the host machine's operational system (OS) kernel. On the other hand, each virtual machine requires a copy of the whole OS, making containers a much lightweight and easy to run solution. (Docker Inc, 2021)

In a nutshell, Docker establishes the standards to isolate the application with all its dependencies together and make connections with other isolated systems, but also sharing important resources such as the processing unit and the host's OS. The simplified Docker architecture can be visualized in Figure 8 which also compares it with the standard architecture of virtual machines for clarification.



Figure 8 – Docker containers architecture vs virtual machines architecture



Source: (Docker Inc, 2021)

In the context of this project, Docker was used to automatically build an image of the machine learning applications directly from the CI pipeline and let them available for being deployed in the form of containers inside the production environments.

## 4 Solution design

The previous chapters formalized the problem and its relevance to the institute, the general solution, and the background concepts that enable the understanding of the implementations. Then, this chapter will now explore the solution designed to achieve the established objectives.

In this project, it is possible to separate the solution into two systems. The first one is the continuous integration pipeline which is not a conventional software application, but it has to be planned, developed, and executed independently to stand as the main solution for DevOps integration. The other system is the deployable machine learning application that will use the CI pipeline and other generic diagrams to guide its development from the data science research.

These two systems can be easily related to the main objectives of this project presented in section 1.1.1. However, it is important to clarify that the main solutions are the CI pipeline, the UML diagrams, and the guideline documents that aim to enhance the path to the deployment of any future production ML application to be implemented by the institute. Nonetheless, to properly validate these solutions it is necessary to use them in the development of an application that completed the research phase. In Chapter 5, an institute's project with two existing ML models was used as a validation use case and converted into a deployable application.

### 4.1 System requirements

Having in mind the project objectives and the proposed solution, the implementation design starts by defining the system requirements which will then guide the development. The functional requirements to be established will describe what the system does and the main features it must-have, while the non-functional requirements will state how the features should be implemented such as defining technologies and establishing desired properties (QRA Corp, 2019).

Below are presented the planned **CI pipeline functional requirements**:

1. Trigger the execution of the stages when a new instance of the application arrives on the git repository.
2. Automatically generate and publish a container with the application and all its dependencies on the same image, in order to make the deployment into another environment as easy as possible.
3. Automatically identify and execute the software tests and code best practices verification.
4. Provide standards to the applications' versions that can be deployed.

5. Automatically export simplified reports about the pipeline results.
6. It should be possible to have different stages according to the type of pipeline (e.g. the trigger branch type or if the execution was triggered by a merge request).

Next, the planned **CI pipeline non-functional requirements** are also presented:

1. It must be implemented using GitLab since this is the DevOps platform that the institute has a license.
2. Have a scalable CI structure so different machine learning applications can easily adapt it to their needs.
3. Have the possibility to choose a specific machine to execute the CI pipeline.

In addition, it is possible to list the generic requirements this project wants to achieve on the future machine learning applications turned to production that will be developed and deployed using the solutions presented in this report which, therefore, are also valid for the validation use case.

Below are presented the planned general **functional requirements for the machine learning applications** that will use the solution.

1. Import/receive information from a data source.
2. Execute a data transformation pipeline.
3. Import a machine learning model inside its structure.
4. Feed the model and collect predictions.
5. Export the predictions to an adequate monitoring system.
6. It must be adequate to the production environment and deployed simply.

Also, the planned **non-functional requirements for the machine learning applications** that will use the solution are presented below.

1. Have a robust architecture easy to understand and with a high maintainability level.
2. It must implement software tests that also enhance the reliability and maintainability.
3. It should have Python as its main language since this is the programming language that the department is more used to.
4. It must be modular and adaptable to different deployment architectures.

## 4.2 General class diagram

With the requirements listed, the design solution advanced to the development of a general class diagram for machine learning applications to be implemented in production environments. This diagram by itself is an important outcome of this project since implementing its structure will enable a robust and reliable solution for the deployment to several of the future ML projects led by the institute. Therefore, this should already bring much more agility to this development process and save time and money from these projects.

However, building a software program based on a class diagram does not ensure that it will have a clean architecture. A bad class diagram can lead the application to have a bad structure which would only make the deployment process and the maintainability even harder. To avoid that, the class diagram presented in this section sought to follow the SOLID principles described in section 3.3.3.

The focus on having a clean architecture is even more important for the kind of applications that this solution is directed to. After achieving a satisfactory program operating the models in production, the predictions made should be valid for a long period. This implies that the software will only receive maintenance when the monitoring indicates a depreciation of the model reliability or when something changes on the manufacturing line and new features wanted to be added. This in turn can imply a different development team having to deal with code developed entirely by someone else in the past, especially if the fast researchers' rotation of IPT is taken into account. Software in that situation is called legacy code and can be hard to understand and make changes due to the lack of references. Implementing a clean and maintainable architecture is a great instrument to avoid this scenario in projects with that gaps in development.

The elaborated software architecture is divided into packages and will be exhibited gradually in these sections for better visualization and understanding. However, the complete class diagram can be visualized in Appendix A and its analysis is stimulated to understand all the connections.

### 4.2.1 Data source package

As the functional requirements state, a crucial feature of a machine learning application in production is to communicate and acquire inputs from a data source. For that purpose, the data source package was developed inside the diagram, and its operation is entirely coordinated by the *DataSourceManager*. This class should establish communication with the data source and take care of the operational details as well as integrate the tasks that will be performed and ensure their correct execution.

The package diagram can be visualized in Figure 9 and its manager holds

instances of the two classes that will perform the requests to the database. The *DataBaseReader* will gather data from the data source which, in standard applications, will mean the execution of Structured Query Language (SQL) statements seeking optimal performance. On the other hand, *DataBaseWriter* is responsible for writing the predictions back into the database, which will include making data adjustments (e.g. concatenating the results) and performing the SQL post. Both classes receive the needed database information, such as the login session, directly from the manager class.

In addition, to make the correct management of the relevant database tables in a high-level approach, the class diagram proposes the usage of an Object Relational Mapper (ORM). Using this procedure, the tables and their attributes are represented inside the software as normal object-oriented classes. Then, they are all attached to the same declarative base and the other classes can perform different actions without having to be concerned with SQL requests but can do that similarly to managing any standard code object and instances.

It is generally necessary to use a library or framework to make the implementation of ORM easier. SQLAlchemy is an open-source library widely used that proved to be a great alternative for Python language.

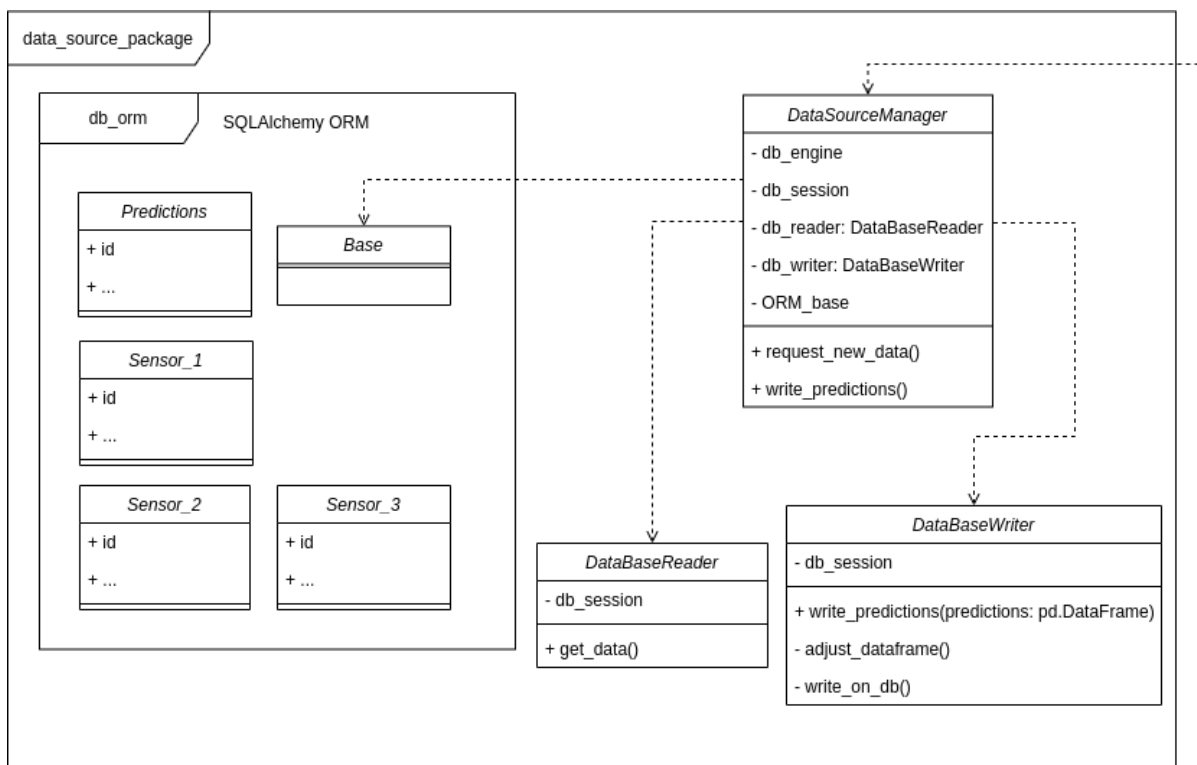
Finally, the manager class is the only interface for the rest of the software to deal with the database since the low-level methods should be protected. This is important to avoid the execution of undesired requests that could change the production database and interfere with manufacturing processes. In fact, in the ideal scenario, even this package will be capable of performing only reading commands in tables that already exist whereas the write commands should only be possible in new tables created specifically for the machine learning implementation.

### 4.2.2 Model package

Another important part of the class diagram is the model package. As the name suggests, it will be responsible for coordinating all the actions to be performed with the assistance of the machine learning model. Similar to the last package, it will also have a manager class that will gather all the information needed for those actions and will be the only interface for the models to the rest of the software.

The main methods this class has to implement are the models' import and the acquirement of the desired predictions by feeding them with the transformed data, as it is possible to see in Figure 10. For some specific applications, it can be necessary to train the model from the beginning at each execution, the precedent project that inspires this study is one example of that. In these cases, another method would be necessary and it is written in red on the diagram to signalize that it will not be present in a standard general application.

Figure 9 – Data source package on the general class diagram



An important observation is that the specific method that executes the predictions will have big variations according to the ML solution to be implemented. For instance, a use case where a single model is sent to production will result in a simple method that provides data to that model and return the predictions, whereas a use case with specialized models would demand a selection of which data goes into each model and separate their execution in threads to enable parallel programming. The same can happen with the implementation of cascaded models, competing models, and other models arrangements. However, these adaptations will only change how the method will be implemented and will not necessarily change the architecture design, therefore making the class diagram valid for different model approaches.

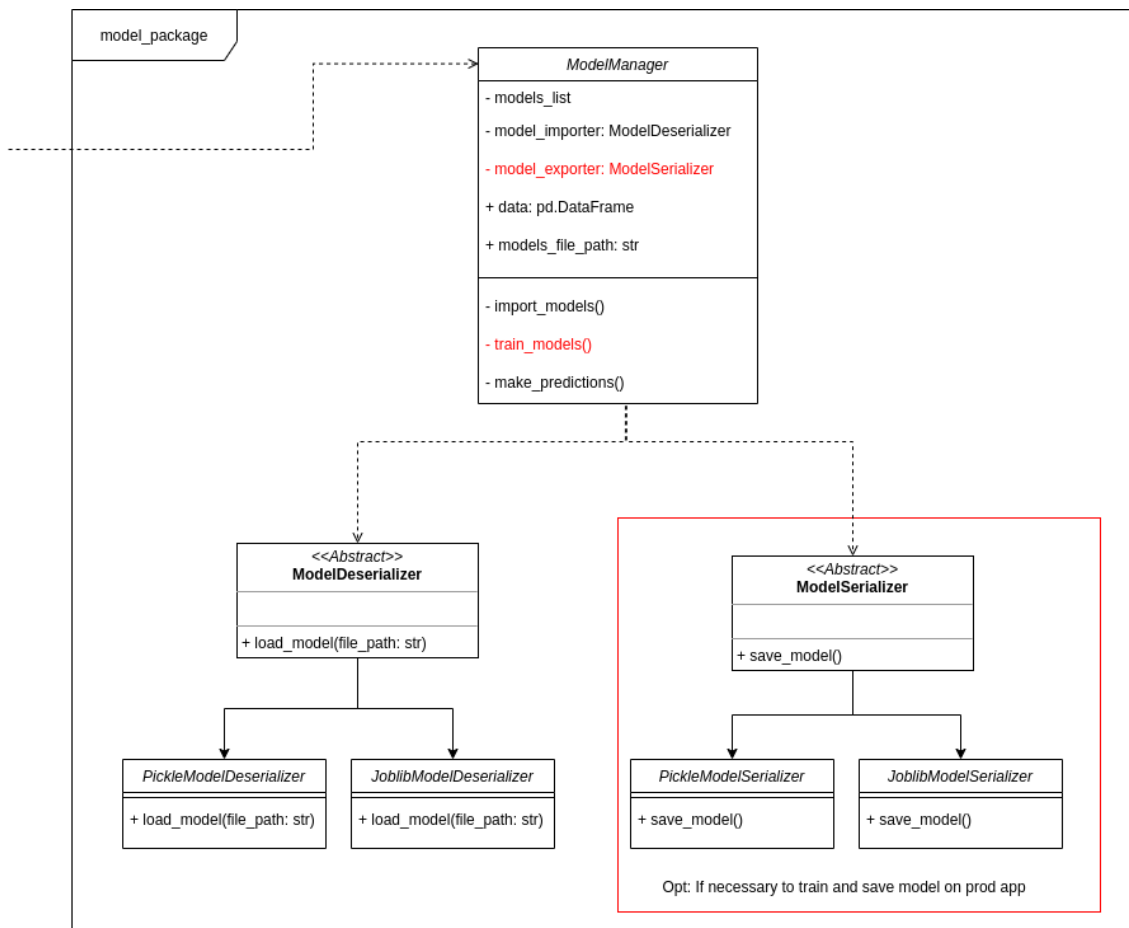
Furthermore, in the majority of cases, the model will not be embedded inside the software structure but has to be imported from a serialized file. To handle that, the package also has two other classes, one to deserialize the model into the code structure and another one to serialize it for the specific applications that build or change the model inside their executions and have to store them after that. Both classes are abstractions, which means that they describe the attributes and methods a class of this type must have like a recipe. When a child class inherits from them, they need to perform what is described in the parent class but have the freedom to do that concretely.

The usage of these abstractions is very important to ensure that the class di-

agram is respecting the SOLID principles to achieve a clean architecture. DSP is not broken in that case because the classes still rely on abstractions and not concretions. OCP, in turn, will still be respected in some future maintenance of the code since, for example, to implement the possibility to deserialize the model using another technology it is not necessary to modify the production code that is already working but an extension of the abstract class can be made.

In the class diagram, the child classes for dealing with serializations are planned to be implemented using pickle or joblib, two of the most used Python libraries for these tasks, each one with its computational advantages for specific cases. Again, if these libraries are obsolete in the future, the architecture can receive maintenance with another extension to new technology.

Figure 10 – Model package on the general class diagram



### 4.2.3 Configuration package

Some of the tasks listed on the requirements will depend on the information that can change in each application's execution. Part of this information should also

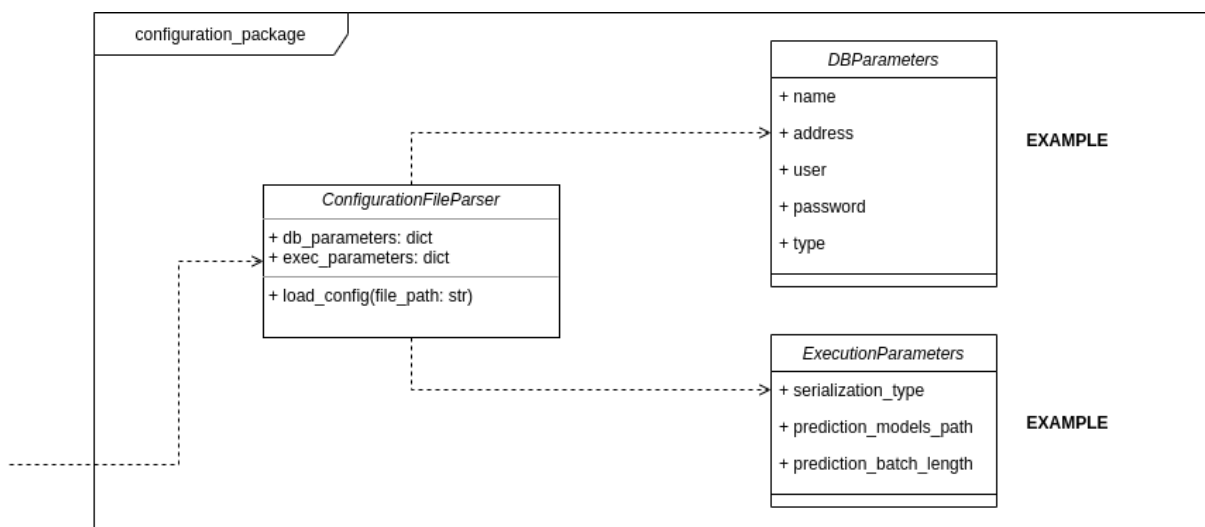
be protected and cannot be hardcoded inside the software because this would make it available for everyone that has access to the project's repository.

To handle that, a small configuration package was designed inside the general class diagram and can be visualized in Figure 11. It is compounded by a central class that must be able to load the configuration from a file put in a previously defined folder inside the application. Then, it must convert it into other structural classes that have no methods but are designed only to organize this information.

The configuration data will vary from application to application but is reasonable to assume that the database address and login credentials will be necessary. Also, other execution decisions, such as which serialization type to use to import the ML models, can be required. Therefore, the diagram separates these structural classes into *DBParameters* and *ExecutionParameters* as motivational examples, but it should be adapted to each case without representing big changes to the architecture since the classes have no methods or extra connections.

Nonetheless, the loading of these parameters can use the technology preferred by the project. However, Python applications have a built-in structure called dictionaries that are very similar to the JavaScript Object Notation (JSON) data-interchange format representation so its usage is indicated.

Figure 11 – Configuration package on the general class diagram



#### 4.2.4 Controller and data preparation classes

The last requirement of general machine learning applications to compound the design solution is the ability to execute data preparation pipelines that will be described by the research experiment. To keep the architecture cohesion, the model package should only deal with tasks that interact directly with the ML models so the data pipeline



must be executed earlier.

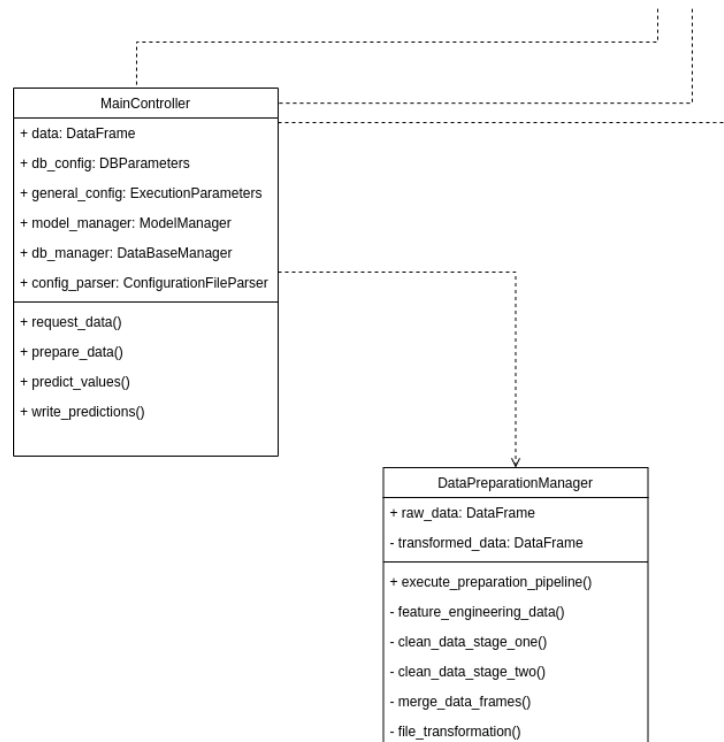
On the other hand, it is expected that the data transformation will be compounded by several complex operations but without the necessity of having different architecture components. For that reason, a single class was designed to control the sequence of preparation steps, as it is possible to see in Figure 12.

This pipeline usually contains tasks such as data cleaning, format transformations, and other feature engineering actions made during the experimentation to increase data quality and, therefore, increase the model's performance. Each one of these steps can be described as a protected method inside the class, which is called in the correct sequence and receiving the correct parameters inside the *execute\_preparation\_pipeline* method accessible from external classes. In short, the class should receive the raw data, process it in the correct order, and return the data ready to feed the model.

Finally, the *MainController* class is responsible to manage the application execution as a whole. It has an instance of all manager classes of the packages and the data preparation, which means that this class will control the workflow of the high-level methods and the inputs and outputs of each package. To do that, it will also have other attributes such as the current state of the data and the imported execution configurations. This class will contain the application's "start method" to be called in the command that will be executed inside the Docker container when running in production.

In sum, these last classes enable the execution of the remaining system requirements and connect the other packages resulting in clean and robust design architecture. This diagram will then be the base for the development of the components aiming to have a faster deployment of such applications.

Figure 12 – Main controller and data preparation classes



### 4.3 General sequence diagram

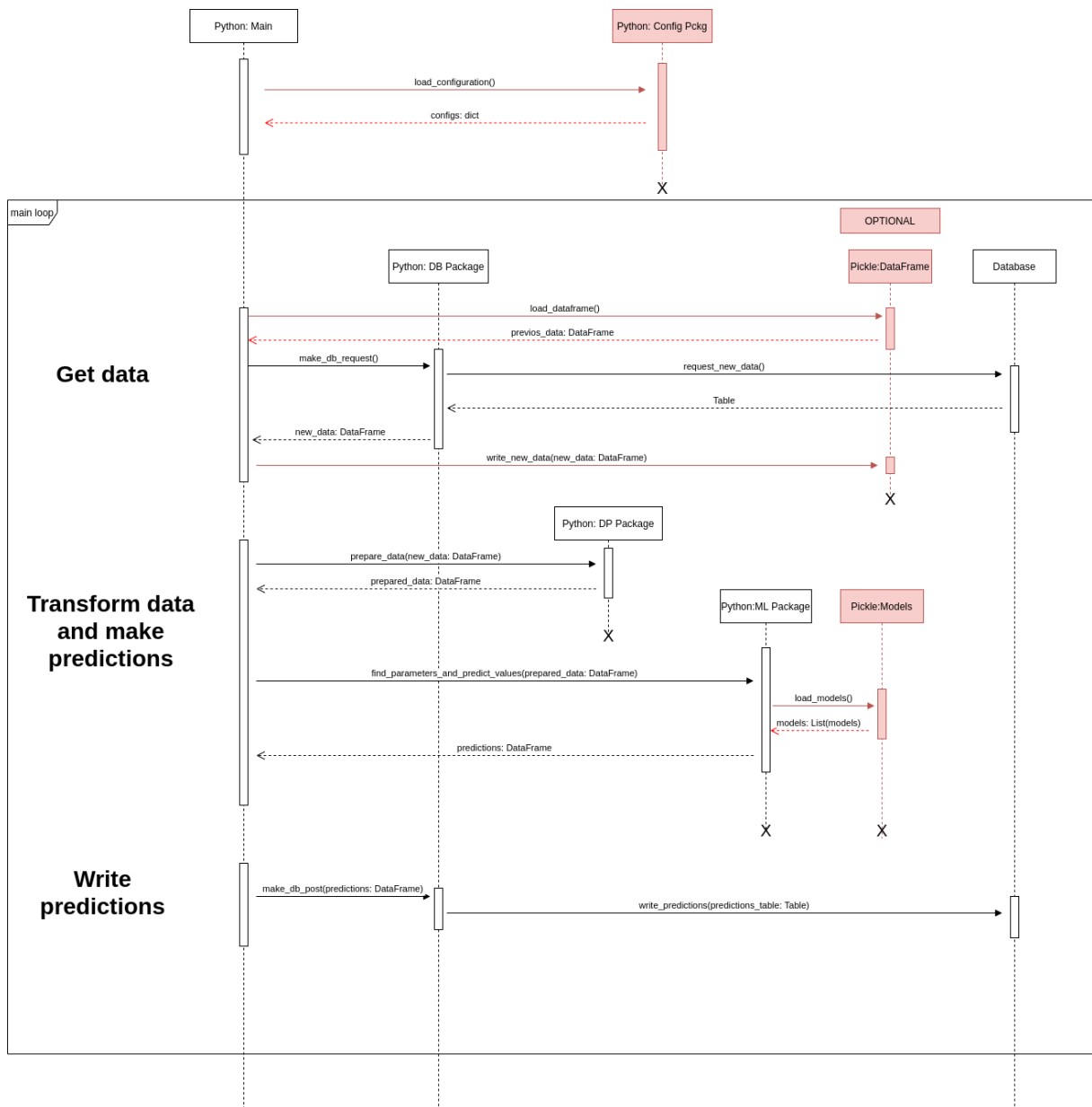
With the general class diagram constructed it is possible to build a sequence diagram to describe the application's workflow and how the classes and packages should interact. This will be important to guide the development when constructing the methods and the controller class.

In Figure 13 it is possible to see that the execution is divided into three main stages. The first one aims to get new data from the database and it initializes with an optional step where the last processed data frame is imported to define where the last loop stopped. Then, the main controller interacts with the database package to make the data request and optionally saves it back in the serialized data frame.

The next stages start interacting with the data preparation package to execute the set of transformation methods and obtain the treated data in the main controller. Next, the models' package receives this data and feeds the models, importing them from a serialized object if they are not completely built inside the method on each execution. When the predictions are done, the main controller sends them to the database package in the last stage, which will write them in the correspondent table.

The stages are repeated in the same order in the main loop. However, before the first iteration, the configuration package imports the execution parameters which can contain database credentials, batch size, a serialization library, and other details

Figure 13 – General sequence diagram



#### 4.4 Deployment architecture

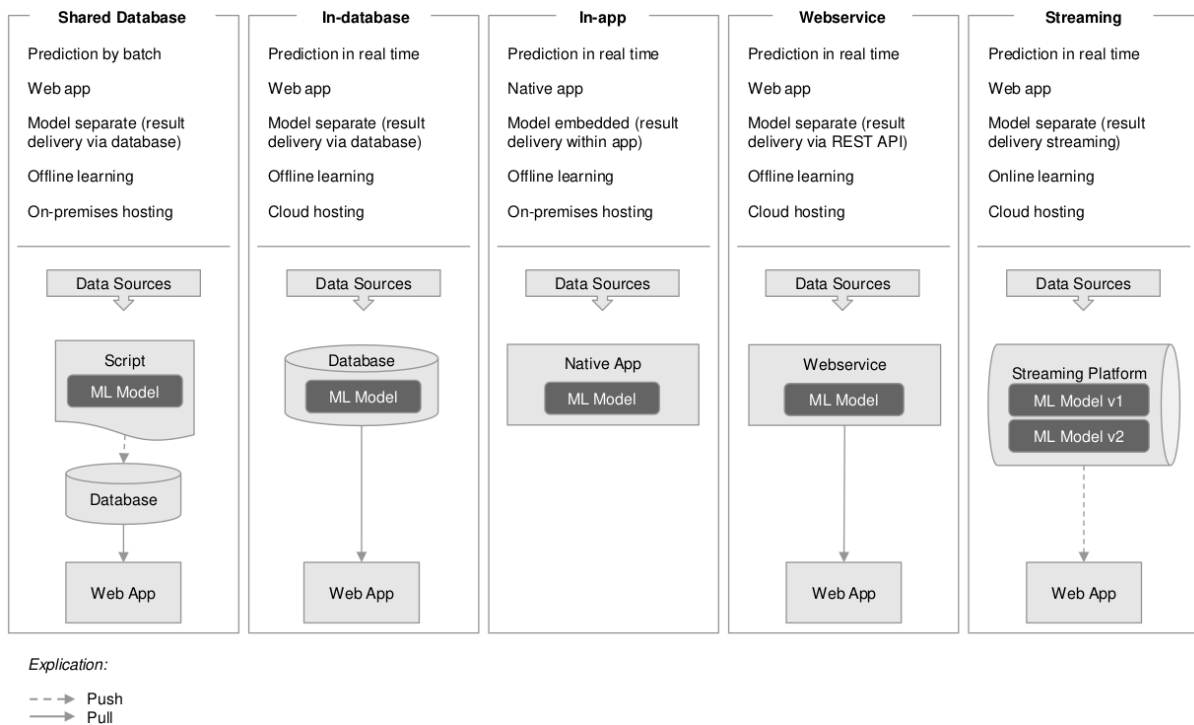
Another important concept that must be taken into account when developing the machine learning application is the architecture of the environment where it will be deployed. The necessary use of some specific technology during the implementation will result in changes on the design level as well, which implies changes in the class diagram structure.

(HEYMANN, 2021) cites some of the most common architectures to deploy ML models, as it is shown in Figure 14. The general class diagram explained in section 4.2 was developed having the shared database architecture in mind, which is the simplest

structure where the application runs according to a schedule or a human request and shares the results by publishing the predictions on the database. Then, it can be consulted by any other application that wants to make decisions driven by the predictions.

This design solution already covers a considerable amount of use cases, since this is an architecture easy to build in manufacturing lines. Moreover, it could fit well for other patterns implementing small changes in the class diagram. Since one of the non-functional requirements states the necessity to be adaptable to different architectures, it was decided to formulate the necessary changes in the original general class diagram to one of the other architectures.

Figure 14 – Architecture patterns



Source: (HEYMANN, 2021)

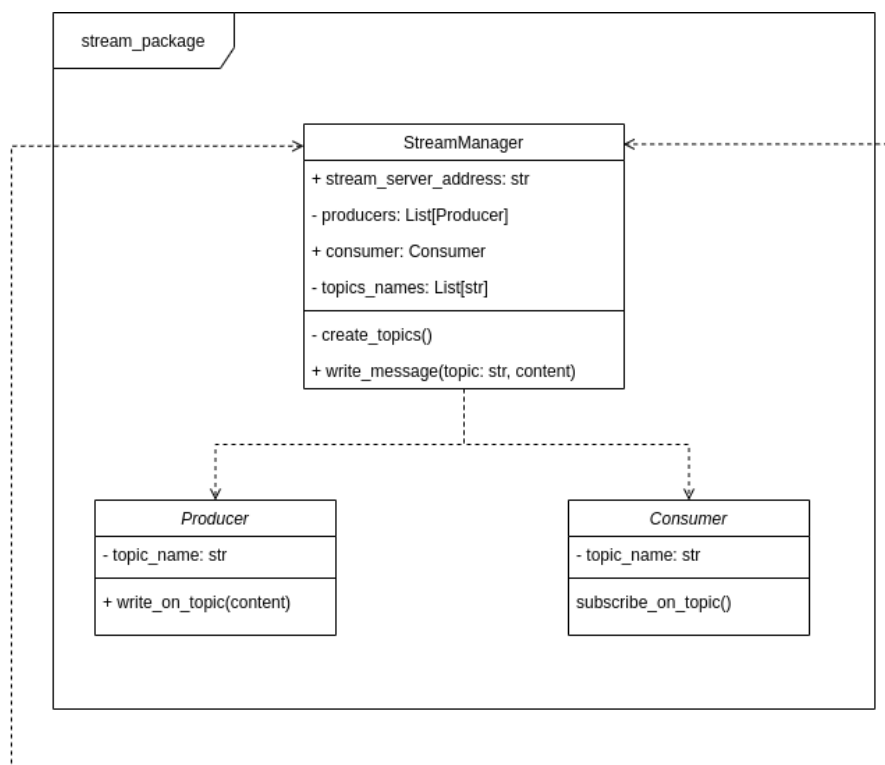
The architecture pattern that has fewer points in common with the shared database is the streaming platform. As (HEYMANN, 2021) states, it brings many advantages to the system with real-time predictions and very good scalability, but with high complexity to be implemented. For that reason, the needed changes on the design solution were elaborated considering a streaming ML application for an extra use case validation.

The first step was the creation of an extra package to handle the streams, which can be seen in Figure 15. Streaming platforms like Kafka are divided into topics that can have publishers and consumers of live data. These structures are represented in the classes *Producer* and *Consumer* which will be controlled by the *StreamManager*

following the approach used in the other packages. This last one will be the interface for writing messages on topics from the rest of the software, but the access to the consumer will still be open from external classes since they will probably need to operate in a loop subscribed to the consumer waiting for new messages to arrive.

The producer and consumer classes will usually need to use an external library to correctly implement the communication with the topics from the streaming platform since this field has considerable support from open source communities.

Figure 15 – Stream package in the general class diagram adaptation for streaming architecture



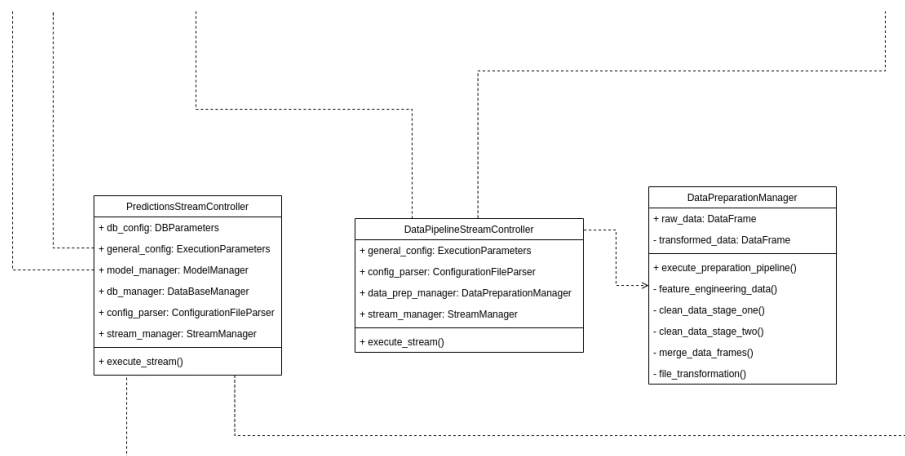
Furthermore, the class diagram based on a shared database approach has one controller class that connects all the packages and handles the execution flow. In this new scenario, each stream must execute independently, similar to having different applications making small tasks of the complete execution pipeline. To represent that in the class diagram each stream must be represented in a controller class that will be connected only with the packages that the tasks to be executed in that stream need. Then, for example, each controller class must be initialized independently in different containers inside the production environment when deployed.

One usage of this architecture is to separate the data preparation actions in one stream and the predictions execution in another one, which is represented in Figure 16. When new data arrives directly from the manufacturing sensors to one topic, the data

preparation stream can read and handle each value individually and write it back in another topic without having to wait for any prediction. In turn, the predictions stream will also receive the treated values individually and can handle them independently. This case exemplifies the advantages of this architecture, since if the predictions stream is working slower for any reason this will not prevent the preparation of new data that arrives in the streaming platform.

Finally, some elements of the first class diagram should be deleted for this new architecture. For instance, if the data will arrive in the first stream directly by a topic, then the database reader class is not necessary anymore. The complete general class diagram for streaming architectures can be visualized in Appendix B.

Figure 16 – Stream controllers in the general class diagram adaptation for streaming architecture



With the requirements established and the developed UML diagrams compounding the solution design, the machine learning applications for manufacturing inside the institute will have a solid base to guide its development. The next chapter will cover the details of practical implementations, beginning with the CI pipeline and advancing to the use case validation.

## 5 Development

The diagrams introduced in the last chapters are an essential part of the proposed solution, but they are not enough to make the path to deployment ideally fast. As explained before, the main proposal of this project is to create and set up a continuous integration environment that automatically runs important verification and integration steps.

In this chapter, the elaboration of the CI pipeline will be explained together with its stages and release to the ML projects carried within the institute. Also, an existing data-science project for a manufacturing use case developed by the institute will be analyzed and transformed into a deployable application using the built mechanism.

### 5.1 CI pipeline

As stated in chapter 3, the continuous integration pipeline aims to constantly perform automated actions in order to overcome integration barriers and enhance the path to deployment. In practice, this pipeline does not have to include any standard stages and can be developed just like any other software application. However, it is common to follow some DevOps general patterns when defining the execution sequence.

For the machine learning applications inside the institute, it is necessary to perform actions that transform the source code into something accessible and executable in different environments. Furthermore, it is necessary to ensure that each new code that arrives at the central repository is following the established standards and will not break any past feature, which will in turn result in an application that is easier to be integrated.

There are several DevOps platforms where it is possible to develop and run a CI/CD pipeline together with the software development (Katalon, 2021). However, IPT already has a partnership with GitLab and uses it as their repositories manager which limits the solution to be implemented using this platform, as stated in the non-functional requirements.

With that in mind, the standard approach GitLab uses to develop the CI actions is by creating a YAML file named “.gitlab-ci.yml” inside the application’s repository. Thus, whenever there is a new commit, this file will be read by the platform which will then define if a pipeline will run and the stages that will compound it.

Among other configurations, the CI pipeline structure inside GitLab is formed by stages and jobs. The jobs are the instances to be executed while the stages are a group of jobs that fits in the same category. The execution sequence can be explicitly defined with trigger rules created in a separated section inside the jobs’ description in the YAML file. Each job run only after all the defined rules are successfully satisfied regardless the stage in which they are configured.

Nonetheless, jobs have to be picked by GitLab runners, which are environments inside some machine, to be executed. The GitLab platform has several optimized shared runners that can be used to execute CI pipelines from different projects worldwide, but they have only 400 minutes of free execution time per month for private projects. To overcome that problem, it is possible to set up a runner instance inside a private machine and configure it accordingly. For this project, several tests were performed using a docker-in-docker structure for a private GitLab runner, which is then planned as the ideal case for running the private CI pipeline to be developed and its described step by step into the resulting guidelines that will stand as important practical documentation of the project, specifically in Appendix C.

Therefore, each job will execute in a new environment inside the private runner when triggered. The configuration of what will be performed in each job can have multiple details inside the YAML file, but they were implemented in this project starting with the definition of the images and services that will be used and consequently must be downloaded inside the environment. Then, the stage that the job belongs to is defined and a special tag can be attributed to specify in each runner the job must be executed. Other trigger rules were configured for some jobs in a separate section, such as waiting for other tasks' conclusion or only executing in specific types of commits.

Finally, the core part of the job is described in the script section. That part will contain a set of bash commands to be executed in sequence after all the configurations previously described were ensured inside the environment, representing the actions that will properly improve the integration between development and operations. These commands do not need to be described entirely inside the YAML file but a bash script can be called alternatively.

Furthermore, there are two other sections inside the YAML file that describe important configurations for this project. The first one declares global variables to be imported inside each job's environment, the other one lists bash commands similar to the script sections with the difference that these commands will be executed at the beginning of each job, avoiding the necessity to describe them inside all script sections.

Taking into account this description of how the actions are programmed inside the CI pipeline, it is important to define the stages that will compound it. The first stage will be responsible to build the application. In other words, it will transform the repository's source code into something compact and ready to run. For instance, this stage is the one responsible for the code compilation in other programming languages. Although this is not necessary for Python, the application stills have to be embedded inside an accessible container with an environment including an interpreter configured with all the needed libraries and other dependencies.

The second stage was defined as the testing one. Its task is to find and execute programmed software tests to validate the application's features and ensure code main-



tainability. This process usually needs to make some integration actions and must also return the tests' results as understandable as possible to make the errors correction easier for the developers. After having the application built and verified, the last stage must make it available to define the release strategy.

Another stage common to find in CI/CD pipelines is responsible for directly deploying the application inside the environments where it will run. This is a really helpful step easy to be implemented when the team has total control of the production environment. In the ML uses cases carried by the institute, the application is developed inside their environment while the manufacturing is controlled entirely by the client company. For that reason, the continuous delivery approach used by this solution limits itself to releasing a ready-to-run instance of the software, which should be available to the production environment but must be imported and started manually.

In a nutshell, the CI pipeline will be compounded by a series of integration actions programmed in bash scripts and organized into stages. Instead of performing these actions inside a developer's machine from time to time, it is configured to run automatically in an environment prepared and optimized for that, also ensuring no biases. Additionally, the execution flow can change from situation to situation, which is also automatically treated by the pipeline's configuration together with the Docker images and services that need to be downloaded for each action. The implementation of the mentioned stages is explained separated in the next topics, describing the CI pipeline development concretely.

### 5.1.1 Build stage

The GitLab repository stores code files and other documentation related to the development of the application that can be downloaded by any user with access. However, having a folder containing all the Python scripts and packages is not the same thing as being able to run the application anywhere. An environment with a Python interpreter containing all the dependencies installed is necessary together with an operational system that can execute them. For that reason, this stage aims to build a ready-to-run application instance, therefore configuring itself as a crucial step inside the pipeline.

Using a Docker container to achieve that is one of the widest used approaches and it will also be the most important action performed on this stage, as mentioned earlier. Thus, the container has to be constructed and published to be accessible. Even though this stage is very important to solve the integration problems faced by the applications, it can be entirely described in a single task so its division will contain only one job named "build-docker".

The GitLab runner will already be executing inside a Docker container, so to run the build commands and create another Docker image it was necessary to use

a docker-in-docker approach. This means that the Docker container of the runner will also download and use an image of Docker to perform Docker commands. To achieve that, some special configurations were performed inside the GitLab runner turning the privilege mode on and setting Docker as the runner's executor, just like it is described in the commands used for the new project creation guidelines in Appendix C.

Hence, when the runner instance triggers the build job, the first action it has to do is to download the latest image version of Docker and the service called `dind` (docker-in-docker). Then, since the environment will have Docker imported, the job's script session can use it to build the new application image following the steps listed inside the repository's Dockerfile. This file was created using a series of Docker commands that will serve as a receipt for the application build.

Similar to how the YAML file describes the CI pipeline, the Dockerfile starts by listing the other images that will be imported together inside the image to be generated. For this application, only a Python interpreter with version 3.6 is embedded. Then, a working directory is defined inside the environment and the first file to be transferred from the source code is the "requirements.txt", a text file containing the name of general standard libraries that will be necessary to run the application. The central idea is that this file will be incremented during the application's development adding new libraries being used, in that form the CI pipeline will automatically consider these new dependencies when running a new cycle.

Next, the build process runs a bash command to install the listed libraries using `pip`, a package installer that comes together with the Python image, also ensuring that no cache files are left by `pip` in order to have an image as small as possible. Then, the configuration, database, and models packages are inserted inside the image in their respective folders as well as the python files located in the repository's root.

The last step in the Dockerfile is the definition of the image's commands. These bash lines will be performed whenever one container is executed inside a machine using the built image. In this application, the first command changes the value of the environment variable `PYTHONPATH` to the Docker root to enable a standard use of relative paths inside the Python scripts and the second one executes the "main.py" file which will initiate the program's execution.

With the application transformed into a Docker image, it has to be published so other environments can access it. GitLab automatically has a Docker container registry integrated with the project's repository, which provides a space to store and access Docker images using the same GitLab credentials. Thus, the "docker-build" job finishes by login into the registry using the protected CI predefined variables that make temporary credentials, only valid while the job is running, and publishes the image with the shortened branch name.

To acquire a specific image of an application's version, the user can explore the

container registry page and directly copy the download tag. Then, any computer with Docker installed and access to the internet must be able to import and run the container regardless of the machine's configuration and operational system. Moreover, other jobs of the CI pipeline can import the application to perform tests and any other necessary action.

In resume, the build stage contains only one job that will download the docker-in-docker service, consult a Dockerfile, generate an image of the application, and publish it into the project's registry. If this job succeeds, the stage will finish and the test one will be triggered being available to be picked up by the GitLab runner.

### 5.1.2 Test stage

Making the application accessible and executable is not enough to ensure its integration. It is normal to expect that at some point a method or a class will contain some bug while the application grows, which can culminate in serious errors in production. However, during software development, it is not feasible to manually execute and test each feature whenever there is a new implementation. Alternatively, software tests can be developed to ensure a fast verification from time to time.

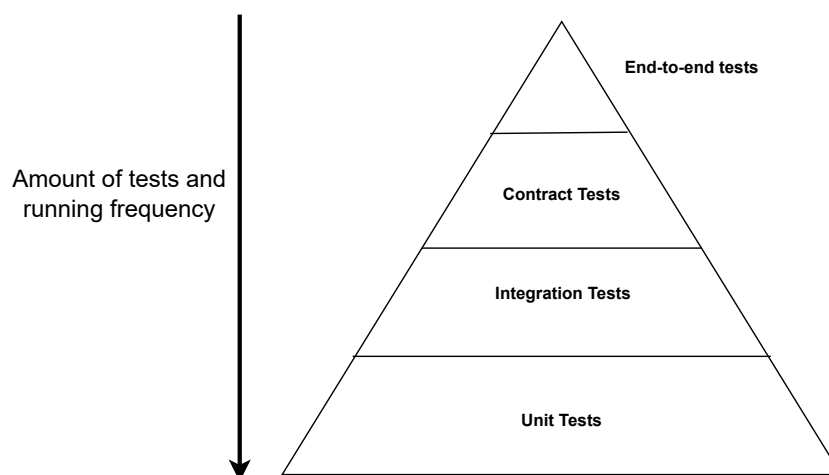
The objective of this stage inside the CI pipeline is to automatically execute and evaluate the software tests to be implemented for the ML applications, but before elaborating its creation in the YAML file it is necessary to define which types of tests will exist in these applications and how they should be developed. Intentionally, the tests explained in section 3.3.4 were the ones considered relevant for the type of application for which the project's solution is intended.

Nonetheless, unit and integration tests build together a solid verification of software applications and were chosen to be the base of the testing stage. This decision is based on the principle that small and concise well-implemented tests directly indicate what feature is broken in the application and where to find it inside the code, while tests with a wide execution stack demand an interpretation of the problem from the beginning making it harder to take value from failing tests (WACKER, 2015). Therefore, having an application well covered by unit tests and complemented by integration tests should be the highest priority when regarding verification of the solutions to be deployed.

Contract and end-to-end tests should also have their implementation considered. The first one will secure that the program still communicates properly with the microservices and third-party applications that it uses, whereas the other one makes a test similar to the final usage of the product considering the whole execution stack. Even though wide tests are not the best ones to clarify failures, it is recommended to write the sequences to execute at least one end-to-end test that validates that the application is correctly implemented in production. With these tests in mind, the test pyramid of this project was formulated as shown in Figure 17, where the path from the top to the bottom

increases the number of tests to be implemented, their desired running frequency, their execution time, and their level of isolation.

Figure 17 – Test pyramid of the machine learning applications to be developed



Taking the description of the software tests into account, it is possible to realize that the integration of this stage does not rely only on the CI pipeline but also on the continuous development of new tests while the project advances. The implementation of bad tests or bad management of them can lead to a poor verification of the features and ruin the entire test process. To avoid this situation, one of the guideline documents is entirely dedicated to this topic, it clarifies the importance of each test type and concretely shows how to implement the base tests in Python with code examples and best practice explanations, as it can be seen in Appendix F.

Regarding the execution of the tests, the unit and integration ones are also ideal to automate. Their implementation tends to be more isolated and test doubles are used to replace real-world dependencies. On the other hand, the objective of the other test types is to validate real dependencies, which makes it necessary to have access to the production environment or a copy of it implemented with the same parameters. As explained earlier, the projects carried within the institute will usually involve a production entirely controlled by the client company, which implies that the contract and end-to-end-tests can not be automatically performed by the CI pipeline, being recommended to run them manually from time to time.

Considering these points, two jobs were initially defined inside the testing stage: “unit-tests” and “integration-tests”. Instead of performing the tests using the branch files inside the repository, these jobs download the application image generated in the previous stage and validate the features directly in the instance that will be used in production. In that way, the verification is as similar to testing inside the production environment as possible, also helping to ensure that the build step was executed cor-

rectly. To make that, it is also necessary to use the docker-in-docker service, login into the project's container registry, and then download the recently published application image.

Next, the script section of both jobs calls a python command to search for unit and integration tests inside the container's tests folder using xmlrunner, an open-source Python library that executes the tests and generates Extensible Markup Language (XML) files reporting their results following the JUnit XML format. At the end of this execution, the report file is extracted from the container to the job environment and an artifact section is created to make it available inside the GitLab platform which, in turn, understands the file's format and exhibits its information on a user-friendly page. Without this feature, the developer would need to search inside the job's console the failing tests together with many other secondary commands' logs. Having this separated reports section inside the pipeline's page - shown in Figure 18 - is crucial to ensure the application scalability to projects that can have thousands of tests.

Figure 18 – Tests report page inside the GitLab platform

Pipeline Needs Jobs 7 Tests 31

**Summary**

31 tests      0 failures      0 errors      100% success rate      133.00ms

---

**Jobs**

Job	Duration	Failed	Errors	Skipped	Passed	Total
integration-tests	12.00ms	0	0	0	5	5
unit-tests	121.00ms	0	0	0	26	26

Two other jobs - “build-contract-tests” and “build-end2end-tests” - were defined inside the test stage to deal with the other test types. However, they are never executed in the standard pipeline unless the GitLab variables “CONTRACT\_TEST” and “END2END\_TESTS” have their values explicitly set to “TRUE”. As mentioned, these test types need to be executed in the production environment, so their jobs does not perform any test but export them inside a separated Docker image that can be used to manually execute them, similar to the build job.

Besides performing software tests, an extra job was configured inside the stage to validate the code quality. Each programming language and project usually have their code standards and best practices principles which when not followed will not represent syntax errors or make the program break, but will certainly affect the code understanding and maintainability. The process to verify these undesired characteristics is called linting and is common to be automated in CI pipelines.

The lint job starts by downloading the application image, just like in the other test

jobs. Then, it uses the `pycodestyle` library to verify all code lines that break some of the PEP-8 guidelines, which is a wide style guide that establishes coding conventions focusing on the code readability for Python. Also, a configuration file is added to the repository to personalize the lint verification, such as raising the maximum amount of characters in one line and enabling the statistics option to count errors and warnings.

Even though the code should be as readable as possible, specific cases can require the intentional break of some code best practices. Therefore, the objective of the lint job is to identify and warn the development team about the code quality deprecation but it should not prevent the release of the production application. As a result, this job is the only one in the whole CI pipeline that is allowed to fail without breaking the pipeline execution.

In sum, the standard execution of the testing stage executes three verification jobs. Two of them perform the developed software tests and export their results to a readable page inside the GitLab platform. The other one verifies bad code implementation but does not interrupt the pipeline execution. With the Docker image published and tested, the last step to ensure better integration between development and operations is to release it and establish a standard access approach.

### 5.1.3 Release stage

The goal of the CI pipeline is not to only verify a large batch of changes in the application all in once, the idea is to execute its actions at each program's increment to be continuously integrating, as the name suggests. On the other hand, a new deployment in production will usually be performed only when relevant new features were added in a new stable version of the code, which occurs when the development branch is merged into the master one in the git repository.

As explained in section 5.1.1, the build stage will always generate a Docker image which is already possible to be accessed in production. However, this would have to be done using the feature branch name which is not the ideal scenario. With that in mind, the release stage aims to separate the versions to be deployed providing control over which program state is running in production and the possibility to perform rollbacks. Thus, the jobs of this stage will only be executed in CI pipelines running in the master branch.

First, three project variables were created inside the GitLab platform named MAJOR, MINOR, and REVISION. Every application release must follow the pattern “{MAJOR}\_{MINOR}\_{REVISION}” in their names and the development team should always update the variables' values in the platform to the next release numbers. Nonetheless, when there is a new release without a change on the variables, the CI pipeline should consider that the new deployment only contains a revision from previous versions.

Then, the “release” job was created inside the stage where the first step per-

formed is the download of the release client image, an API created by GitLab to facilitate the management of releases inside the platform. The job's main action is to receive the release name and perform a command to the API generate it. As a result, it will add an instance in the platform release page that can be accessed and managed by the developers and create a tag in the git repository. This tag creation will trigger the execution of a new CI pipeline that will build and publish an application's image with that name, enabling easy and standard access from the production.

However, since the update of the version name relies on the manual increment in the platform, the pipeline must ensure that no releases will be generated with the same name. To perform that, the "validate-tag" job was created and set to be run before the "release" job. It aims to check the tags that were already created inside the git repository, so it starts by importing a git image to perform this command. Then, the verification logic was separated in a bash script which is called inside the job receiving the GitLab email of the user who started the pipeline, the MAJOR, MINOR, and REVISION variables as parameters.

The bash script starts by setting the user email in the git configuration. Then, it begins a loop increasing the revision variable and verifying if a tag with these MAJOR, MINOR, and REVISION values already exists in the repository. When the values do not exist it is returned as the new tag name to be released. For example, in the scenario where the variables inside the platform have the values MAJOR = 2, MINOR = 1, and REVISION = 1, the script will keep the first two values and, if necessary, will increase the revision until a satisfactory value. Supposing that the repository already has the tags 2\_1\_1, 2\_1\_2, 2\_1\_3, and 2\_1\_4, the program will return the value 2\_1\_5.

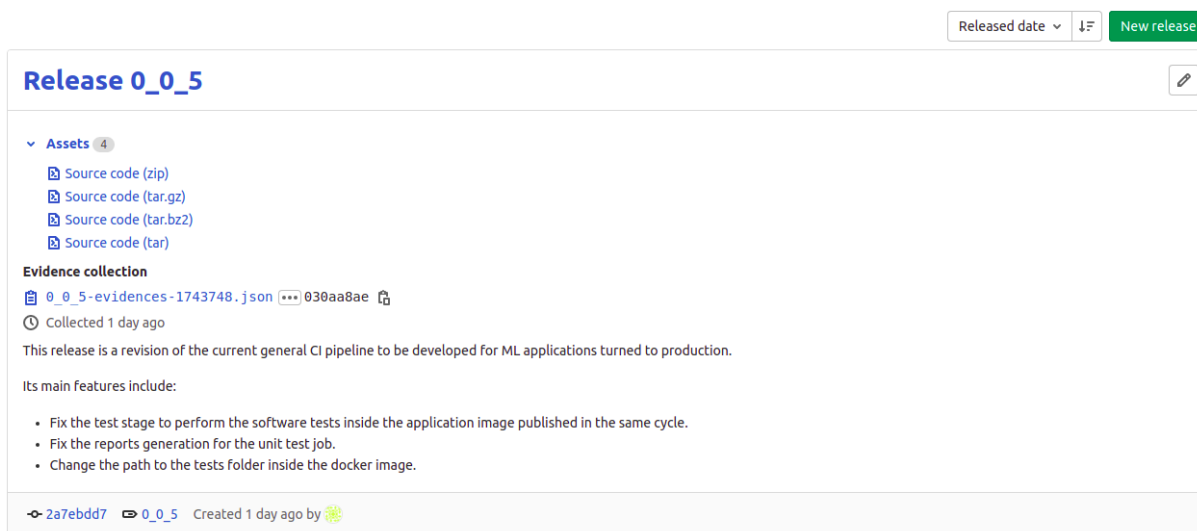
This tag value is recorded in a text file and exported as an artifact to be accessible on the platform. The "release" job imports this artifact and uses it as an input for the tag and release name to be sent to the API. With all these jobs being successful, the CI pipeline finishes its execution and new deployments can be performed inside the production structure by running these simple Docker commands, the first one to import the latest state of the application and the other one to import a specific version:

```
docker run --name <CONTAINER_NAME> registry.gitlab.com/<PROJECT_URL>:master
docker run --name <CONTAINER_NAME> \
    registry.gitlab.com/<PROJECT_URL>:<MAJOR>-<MINOR>-<REVISION>
```

The releases pattern provided by the two jobs stage is not only important to deploy the versions that should run in production, but also to have an easy way to understand what are the changes in each version. With the git tags, the developers can efficiently change the version of their local repositories and directly analyze the differences in the code. Moreover, the GitLab releases page provides the ability to give

a rich description to each version and download its entire content instead of only what was embedded in the Docker image, as it is possible to see in Figure 19.

Figure 19 – Releases page inside GitLab platform generated by the CI pipeline



#### 5.1.4 General aspects

The developed stages and jobs form the CI pipeline execution that will accelerate the path of transforming data science models into deployable applications. It is worth mentioning that the YAML file also has a section declaring bash commands to adjust the runner environment into the correct repository folder that will be executed at the beginning of each job. Nonetheless, variables such as the repository name and the application Docker image name are defined separately and are available to all jobs.

Furthermore, the GitLab runner was defined here as being only one instance for explanation purposes. The configuration steps described in Appendix C can be executed more than once and in different machines providing a net of runners. It is also possible to make advanced configurations inside their environments to optimize their execution for some specific jobs that can carry the runner tag inside the YAML file.

To clarify the pipeline execution sequence, Figure 20 shows the resulting jobs and the dependencies they have highlighting the path to the release. Also, the detailed execution sequence of the CI pipeline can be observed in the elaborated flowchart shown in Figure 21. It is important to note that, when executing in the master branch, the release stage will generate a new code instance by creating the release tag, which will start the execution from the beginning.



Figure 20 – Resulting pipeline sequence to achieve the release stage

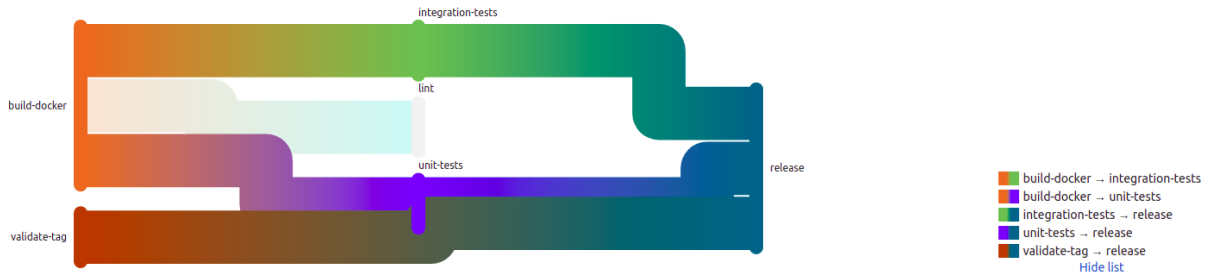
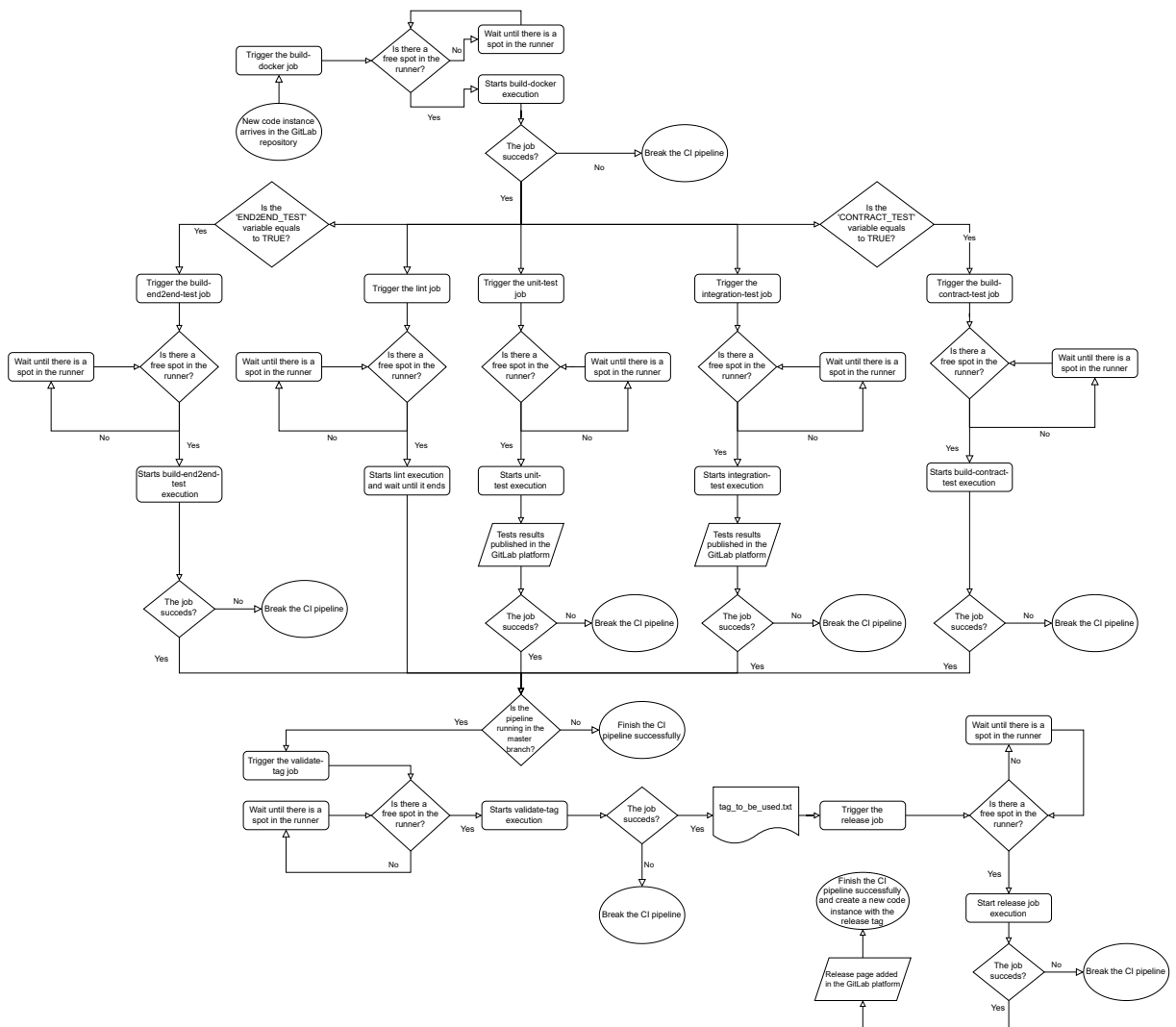


Figure 21 – Flowchart containing the CI pipeline execution logic



## 5.2 Validation use case

To test and validate the DevOps solutions, an important stage of this project lies in applying the CI pipeline and the solution design developed in a machine learning project turned to manufacture that represents the institute's needs.

The fact that the majority of projects carried by IPT with partner companies were in different development phases during the elaboration of this study and that they usually involve confidential information do not configure them as appropriate validation cases. However, to be able to present the institute's know-how in fairs and carry out data science seminars, the Automated Machine Learning team developed a machine learning project template providing a solution for the problem published in (SUN, 2017). This project performed all the CRISP-DM phases until the deployment, which makes it an ideal validation for this project's purposes.

The problem takes place in the System-level Manufacturing and Automation Research Testbed (SMART) at the University of Michigan where a series of 18 experiments were run in their CNC milling machine to produce an "S" shape figure on 2" x 2" x 1.5" wax blocks. One of the solutions proposed to be implemented is the detection of the tool condition based on the data acquired during the machining process classifying them as "worn" or "unworn".

Two types of data sets are provided by (SUN, 2017), one with time series values from 48 columns that come directly from the production and another one containing general constant data about the 18 experiments, such as the material used and the tool condition output. To illustrate the problem and the available data, Table 1 details some of the input columns considered with the highest importance by the data science project previously carried out by the institute.

As mentioned, the department's team imported these data sets and made different data science analyses using Jupyter Notebooks. Firstly, the data is merged and transformed into a data frame object that is used as input for different methods of Python libraries such as sklearn <sup>1</sup>, pandas <sup>2</sup>, NumPy <sup>3</sup>, matplotlib <sup>4</sup>, and seaborn <sup>5</sup>. These analytical steps will result in the features distribution, the clusterized correlation matrix, and other information that is crucial to understand the data and how its features affect each other, which will guide the correct implementation of the best ML models. However, the further explanation of these topics will be suppressed in this document since the focus of the solution is to get the outputs provided by the research phase and efficiently convert them into a deployable application with a high-quality degree.

After the first analysis, some operations were made inside the data frame to

<sup>1</sup> Further information in (PEDREGOSA et al., 2011).

<sup>2</sup> Further information in (The pandas development team, 2020).

<sup>3</sup> Further information in (HARRIS et al., 2020).

<sup>4</sup> Further information in (HUNTER, 2007).

<sup>5</sup> Further information in (WASKOM, 2021).

Table 1 – Most relevant available data for the tool condition prediction

	Data set <sup>a</sup>	Details
<b>ActualPosition</b>	1	Four input variables containing the actual position of the tool divided in the axis X, Y, Z, and the position of the spindle.
<b>CommandPosition</b>	1	Four input variables containing the reference position of the tool divided in the axis X, Y, Z, and the position of the spindle.
<b>OutputCurrent</b>	1	Four input variables containing the output current values in parts X, Y, Z, and the spindle.
<b>OutputVoltage</b>	1	Four input variables containing the output voltage values in parts X, Y, Z, and the spindle.
<b>M1_Current_Feedrate</b>	1	Input variable containing the instantaneous feed rate of the spindle.
<b>Machining_Process</b>	1	Input variable containing the current machining stage being performed.
<b>clamp_pressure</b>	2	Input variable containing the pressure used to hold the workpiece in the vise.

<sup>a</sup> 1 = Time series experiments' data set. 2 = Data set with experiments' constant values.

acquire new relevant features, such as subtracting the actual position of an axis from the command position to result in a column with the positions' difference. This process is called feature engineering and it will result in the data frame that will be used in the first model experiments.

Then, the drop of useless columns was performed and a series of experiments were conducted using Random Forest Classifier and Logistic Regression. For the two model types the hyperparameters were tuned using random search and considering the model's accuracy, F1 score <sup>6</sup>, and MCC <sup>7</sup>. Satisfactory values were reached and with 0.994 of accuracy, 0.994 F1 Score, and 0.988 MCC the Random Forest constituted the best model. It was serialized in a binary file to be used in the deployment.

In sum, the research solution was developed in a permissive environment with static data and has the data transformation steps and a high-performing model as the main outputs. To validate the project results a deployable application must be developed using these outputs, the CI pipeline, and the solution design.

<sup>6</sup> F1 score combines the precision and the recall of the predictions to result in a metric more reliable than the accuracy for imbalanced data. It is calculated with the formula  $F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$ . (KORSTANJE, 2021)

<sup>7</sup> MCC stands for Matthews correlation coefficient and is given in the interval [-1, 1]. It has a high value when the model can predict both the majority of positive data instances and the majority of negative data instances. (CHICCO; JURMAN, 2020)

### 5.2.1 Environment simulation

One of the main characteristics that leads to unexpected problems when deploying a machine learning project in the institute's experience is dealing with a restricted environment different from the one used to generate the models and other transformations. Therefore, it is important to properly simulate a manufacturing environment as close to reality as possible in the validation case.

As previously mentioned, the selected machine learning project comes from a motivation example published by Michigan University describing the behavior of one of their milling machines. The environment specifications are not described, but since the data available is in a table format it is reasonable to assume that it can be published in a relational database. Thus, the first step to simulate the environment where it is possible to deploy the application to be developed was the establishment of a database as a central data source.

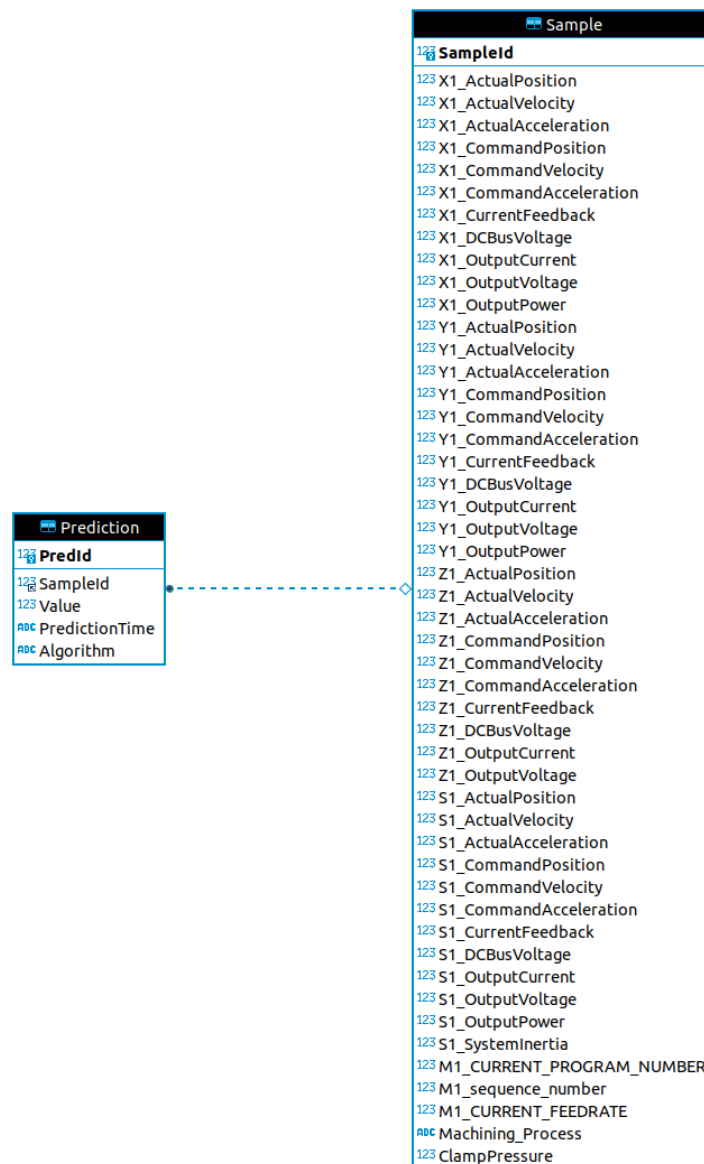
The resulting DB model contains only two relational tables, as it is possible to see in Figure 22. The "Sample" table describes the manufacturing information to be published by the machinery following a time-frequency, whereas the "Prediction" table represents the predictions to be made by the application and published back into the database indicating the tool condition. The relation between the tables is established by a foreign key in the Prediction table containing the identifier of a Sample table which means that all predictions are made based on only one sample, whereas a sample can have many predictions.

The Sample table contains all the 48 columns of the time series data set that will be published together in the simulation since they are presented in the same table by (SUN, 2017). The data set with constant values about all the 18 experiments had some of its columns manually filled, such as the output "Visual Inspect" that indicates if the workpiece passed the visual inspection after the machining process. However, the research phase concluded that the "ClampPressure" input column that indicates the pressure used to hold the workpiece in each experiment is the only data relevant for the machine learning models. Since this is a simple input and adding this value repeatedly together with the other manufacturing data should not be a problem in a real environment, it will be published in the same table here in this simulation.

In contrast, since the value to be predicted has a boolean characteristic that will only classify the tool condition in "worn" or "unworn" during the machining process, the "Prediction" table has few columns indicating the prediction value, the system time when the prediction was made, the algorithm used, and the identifier of the sample for which the prediction is referring.

After the modeling, the database was implemented using a container instance of MySQL engine, which was selected for being one of the widest used relational databases in the industry (DB-Engines, 2021). However, it is important to mention that

Figure 22 – Relational database diagram of the simulated environment



the deployable solution must work regardless of the selected data source if it is possible to guarantee that the application can access it. Furthermore, following the example of other institute's projects, the environment dependencies and the deployable application were organized in Docker containers and Docker Compose was used to coordinate them.

The first diagram in Figure 14 shows that the configured database and a script simulating the data income are enough to represent the shared database architecture. But, the solution design in Chapter 4 also implements the application class diagram for a streaming architecture. Therefore, an extra environment has to be implemented in this use case.

Apache Kafka was the selected streaming platform for this architecture approach

since it is an open source application with great documentation support. Zookeeper is another necessary instance to be implemented in this environment to ensure that Kafka messages, partitions, and topics in the cluster are being tracked. Both services will also be used in the form of Docker containers managed by the Docker Compose, which will make it simple for the application containers to register as publishers or subscribers and consume the data from Kafka providing high scalability.

Besides that, another main difference between the shared database and the streaming environment is the service from which the applications should consume the data. The shared database program can consume as much data as necessary making SQL requests directly to MySQL, whereas in the streaming architecture the arrival of new data from the manufacturing should generate an event that will trigger the subscribers to consume it through Kafka. Hence, it is necessary to establish a connection between the database engine and the Kafka cluster to stream the data whenever a new row is written in one of the database tables.

To solve that problem, the Debezium Kafka MySQL Connector was implemented in the environment. The service is designed around the continuous stream of event messages and it monitors the database tables to generate a data change event whenever an INSERT, UPDATE or DELETE operation is performed (Debezium, 2021). When integrated with the other services, the connector creates one topic for each database table in the Kafka cluster. Therefore, two topics were created for the streaming environment of this project, one that will receive the streams from modifications in the “Sample” table and another one that receives the modifications in the “Prediction” table. As a result, all the applications that need to consume from the database can subscribe to these Kafka topics and receive the data in event streams.

Another service necessary to form the environment is an application that simulates the manufacturing publishing data using the MySQL engine. The data provided by (SUN, 2017) separates the 18 experiments into different excel files. An SQL script was created for each experiment where the excel file is imported and inserted into the database in the table’s format. Then, different bash scripts were developed to execute these SQL files providing the database credentials and designing different scenarios. Since this environment will be used for validation purposes, the main bash script developed publishes all the 18 experiments, being possible to configure a time gap between the writing commands. This will configure the maximum stress scenario, especially if the time gap is small or null.

These services integrated can simulate the manufacturing environments necessary to validate the creation of a deployable application for the selected machine learning project using two different architectures. Figure 23 shows the shared database environment structure and its data flow, whereas Figure 24 illustrates the same for the streaming environment.

Figure 23 – Shared database environment

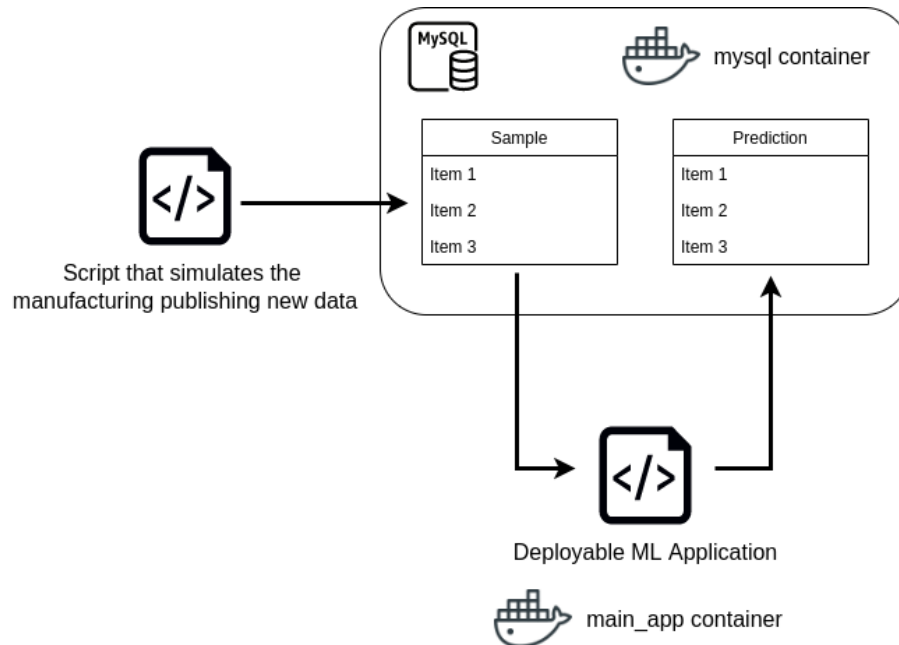
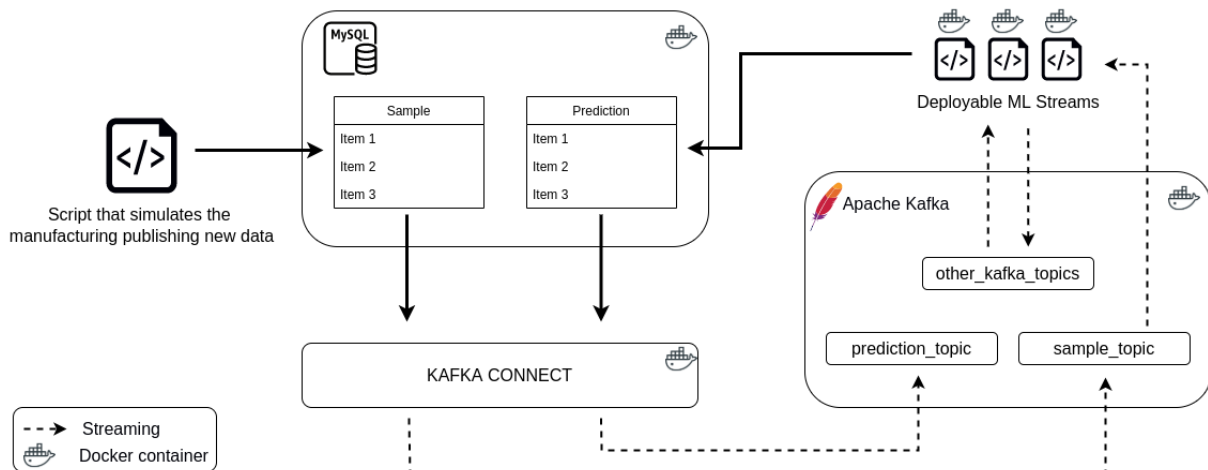


Figure 24 – Streaming environment



### 5.2.2 Shared database application

After the environments' configuration, the deployable application for the shared database was developed using the integration solutions established by this project. The outputs from the data science project were analyzed and embedded in the development following the UML diagrams presented in Chapter 4. Moreover, a Gitlab repository was used with the CI pipeline integrated and running at each new push or merge command.

The class diagram established itself as an important tool to speed up the development of the Python application that successfully performed the predictions using the

Random Forest serialized model. Its methods execution followed the sequence diagram and was developed using the project guidelines, such as the implementation of unit and integration tests that have quickly indicated errors in new code instances uploaded to the repository.

As indicated in section 4.2.3, the configuration package has its attributes changed in the development according to each projects' needs. In this implementation, a JSON file was created containing the information described in Table 2. A template file was added in the Git repository, but the real configuration file must always be filled locally and embedded into the application's container directly since it contains sensible information.

Table 2 – Configuration parameters of the shared database application

Parameter name	Group	Details
serialization_type	Execution parameters	Indicates the type of serialization used, e.g. pickle.
prediction_models_path	Execution parameters	Indicates the path of the ML model inside the application's container.
last_sample_id	Execution parameters	Indicates from which sample id the predictions should start.
db_type	DB parameters	Indicates the type of DB used, e.g. MySQL.
name	DB parameters	Indicates the DB name inside the engine.
address	DB parameters	Indicates the DB address in the environment.
user	DB parameters	Indicates the user to be used in the login credentials.
password	DB parameters	Indicates the password to be used in the login credentials.

Finally, a YAML file similar to the one used to orchestrate the CI pipeline in the Git repository was developed and used by Docker compose to manage the application. The images used were an instance of the MySQL engine and the developed application which uses the GitLab container registry with the version tag automatically generated by the CI pipeline. An SQL file that initiates the manufacturing database according to the modeling of Figure 22 is embedded in the DB container together with the SQL scripts and the excel files necessary to simulate the gradual data addition. Also, the serialized ML model and the JSON configuration file are embedded in the application's container to be used in its execution. Then, the instructions described in the README file presented in Appendix G are used to run the application correctly deployed.



### 5.2.3 Streaming application

Another deployable application was created using the research phase outputs and the solution design. The streaming application followed the UML diagrams created for this type of environment which culminated in the fast development of a robust Python application.

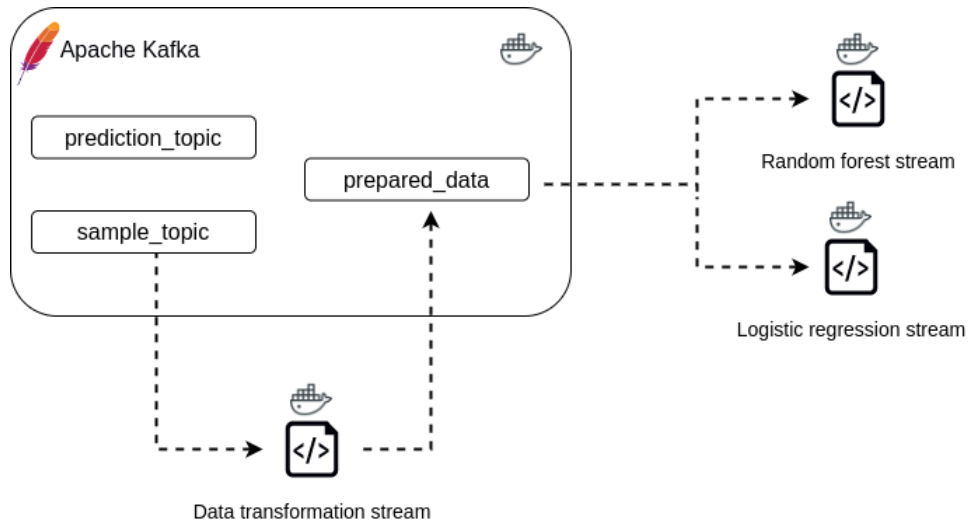
In this scenario, two controller classes were initially developed. One is responsible to process the raw data making the necessary transformations and publish it back in Kafka. With the prepared data available, the other stream feeds the random forest ML model and writes the prediction results into the database. Using that approach, which is also presented in the general streaming class diagram, any delay in getting the results from the model will not prevent the preparation of new raw data that arrives in the DB, since the processes configure two different streams being executed in different containers.

This implementation satisfies the same tasks implemented in the shared database application. However, it is possible to add other features to illustrate the scalability of the CI pipeline and UML diagrams developed to deal with the streaming scenario. Then, the usage of the logistic regression model was added to provide different predictions. The application packages remained the same and another controller class was developed to also consume from the prepared data topic. Therefore, the data transformations are performed in the same way but it is consumed by different streams. If any other model or application wants to consume the processed data in the future, it can subscribe to the desired topic without causing any impact on the running application.

Nevertheless, during the development, it was necessary to adapt some of the CI pipeline stages to the streaming approach. The job that builds the Docker images and publishes them into the container registry was divided into four jobs. Each stream is transformed in a separate application image that shares some of its packages and is built using different Dockerfiles. Furthermore, an extra image containing all packages is generated for testing purposes and it is imported only in the test stage. Hence, this last image is not released at the end of the pipeline.

Figure 25 illustrates the final topics distribution and their usage by the streams containers. A YAML file was again developed and used by Docker compose. A different configuration file is embedded in each stream container including the information described in JSON files that use some of the fields described in Table 2 together with new parameters from Table 3. The remaining environment containers indicated in Figure 24 also have important initialization configurations described in the YAML file, such as the creation of the “prepared\_data” topic which contains two partitions so it can have two subscribers consuming the duplicated data.

Figure 25 – Kafka topics distribution



It is worth mentioning that processing each new input using streams can lead to unnecessary SQL commands to write predictions that could be published together. For that reason, it is possible to configure how many predictions the stream must perform before writing them back into the database. This feature is important to have fast prediction streams when dealing with new data arriving too frequently in the DB.

Table 3 – New configuration parameters used in the streaming application

Parameter name	Group	Streams	Details
kafka_address	Execution parameters	All	Indicates the Kafka address in the environment.
predictions_buffer	Execution parameters	Random forest and logistic regression streams	Indicates the option to define how many predictions will be made by the stream before performing an SQL command to write them back into the database.

## 6 Results

In this chapter, the project's main results are exhibited and evaluated. Firstly, the CI pipeline outcomes and the tasks automated are analyzed. Then, the execution tests performed in the applications developed for the motivational use case are evaluated considering the benefits brought by the software architecture modeled in Chapter 4. For both cases, the fulfillment of the solutions' requirements and the project's objectives are verified. Finally, a summarized overview of the project results is made.

### 6.1 CI pipeline

The main objective of this project establishes the creation of a general CI pipeline that can bring agility for the deployment of ML applications turned to manufacture and enhance its software quality. The validation test explained in section 5.2 used the pipeline in the development of two different applications that converted the know-how of a data science project into deployed Python systems.

The actions performed by the pipeline were crucial to ensure its fast development and correct operation. Together with the guidelines for the software tests elaboration, the pipeline's test stage secured that every new code instance that arrived in the repository did not have broken any past feature and prevented the release of undesired behaviors. Furthermore, the lint job indicated the use of code sentences that were not following the recommended standards and what the developer should do to fix it.

With the tests ensuring the software's correct performance, the build and release stage created a controlled environment for the application in the form of Docker images. Although the pipeline stages are designed to be useful when executed in the configured order, having the image automatically built at each release containing all the necessary dependencies was the main advantage in terms of enhancing the path to deployment. Whenever a new release was performed, the only necessary step to deploy it was to change its tag in the docker-compose file following the instructions of Appendix G. Therefore, the problem identified in precedent projects of having a development environment with different characteristics from the one where the application will be deployed will not happen again, which stands as a valuable improvement for the institute's future partnerships.

Besides the qualitative analysis of the CI pipeline main objectives, it is possible to evaluate its benefits by analyzing the fulfillment of its functional and non-functional requirements listed in section 4.1:

- The pipeline was created and successfully configured inside the institute's Git-Lab platform. It is activated whenever there is a new code instance inside the repository (commits, merge requests, and tag creations).

- Regarding the software verification, unit and integration tests are automatically executed in each CI cycle. The guidelines on how to develop and maintain the software tests are also output from the CI pipeline creation. Their results are automatically transformed into artifact objects inside the platform and exported in the form of reports where is possible to visualize the failures and which commands generated them. Furthermore, the code best practices are verified in the lint job using the PEP8 standards.
- As mentioned, every cycle builds and publishes a Docker image containing the application and every dependency necessary for its execution. Moreover, when the cycle results from a merge request in the main branch, a release is performed and the image name follows a pattern with three numbers indicating major features, minor features, and revisions. These images are available inside the project's container registry which made the deployment as easy as adding the image name inside the tool used to orchestrate the environment (e.g. Docker compose).
- Different cycles are executed depending on the branch where the merge request is being performed as indicated in Figure 21. Other execution rules can be easily added in future projects following the same pattern in the YAML file. The CI structure also proved to be very scalable, since it was possible to adapt it for the development of two applications for completely different environment structures.
- Finally, it is also possible to select the machine where the CI pipeline will be executed. Since it was configured using the docker in docker approach to run in private GitLab Runners, the platform allows the connection with different Runners that can be configured in any machine following the guidelines presented in Appendix C.

Another method to evaluate the CI pipeline results is to observe the cycles performed during the applications development. As previously mentioned, it was used during the whole development for both the shared DB and streaming applications. Table 4 shows the number of pipelines executed and their results. Each failure or warning was useful to avoid bad behaviors in the releases and to promptly correct them, whereas each approved cycle culminated in the Docker image being published and becoming available to simplify the deployment.

Table 4 – Pipelines executed during the applications development

<b>Application</b>	<b>Passed</b>	<b>Passed with warnings</b>	<b>Failed</b>	<b>Total</b>
Shared DB	30	1	17	48
Streaming	28	5	18	51

Since it is necessary to correct the cycle failures before merging the repository branches, every deployable release of the applications has to correctly execute the software tests and embed the necessary dependencies. Figures 26 and 27 show the pipeline executed in each tagged release created automatically by the CI pipeline with merge requests in the master branch. There is no failed job and, therefore, no adjustment is needed after the tag is released, which also proves the benefits that the pipeline brings to the application development.

Figure 26 – Pipelines executed in the released tags of the shared database application











































Status	Pipeline ID	Triggerer	Commit	Stages	Duration
	#587271 latest		0_1_0 -> 1b7fc494 Merge branch 'develop' i...		00:23:39 13 hours ago
	#566542 latest		0_1_0 -> efc07dcc Merge branch 'develop' i...		00:59:30 2 weeks ago
	#565192 latest		0_0_4 -> 3a6f0f59 Merge branch 'develop' i...		00:20:41 3 weeks ago
	#533262 latest		0_0_3 -> a552b554 Merge branch 'develop' i...		00:53:32 3 weeks ago
	#531235 latest		0_0_2 -> a686ee1f Merge branch 'develop' i...		00:25:01 1 month ago
	#528523 latest		0_0_1 -> 840a3474 Merge branch 'develop' i...		00:43:34 1 month ago
	#524959 latest		0_0_0 -> fa4a8e3a Initial commit		00:30:36 2 months ago

Figure 27 – Pipelines executed in the released tags of the streaming application

Status	Pipeline ID	Triggerer	Commit	Stages	Duration
	#587186 latest		1_0_0 -> eae1186c Merge branch 'develop' i...		00:56:31 15 hours ago
	#566537 latest		0_1_2 -> f40b8469 Merge branch 'develop' i...		01:37:29 2 weeks ago
	#539475 latest		0_1_1 -> 043d41bd Merge branch 'duplicate_...		01:07:45 1 month ago
	#537174 latest		0_1_0 -> aaaad946 Merge branch 'develop' i...		00:32:36 1 month ago
	#533290 latest		0_0_2 -> d15d0a7f Merge branch 'develop' i...		00:58:10 1 month ago
	#519795 latest		0_0_1 -> 89062d77 Merge branch 'develop' i...		00:29:17 2 months ago
	#518735 latest		0_0_0 -> 621dcbc7 Merge branch 'develop' i...		00:41:44 2 months ago

## 6.2 Deployment applications tests

The results acquired with the applications deployed are also an important project evaluation. Their correct operation validates not only the CI pipeline but also the general UML diagrams and development guidelines elaborated.

The requirements of the applications were also listed in section 4.1 and it guided the modeling of the solution design. The majority of the requirements were accomplished in the applications' execution, which illustrates the benefits that the general software modeling can bring to future projects. Because the fast deployment of the solutions is one of the main objectives, it is worth mentioning that the functional requirement number six and the non-functional requirement number four were accomplished with the applications being deployed following simple steps in both environments.

On the other hand, functional requirement number five states that is necessary to have an adequate monitoring system to observe the prediction results. In both applications the results are published back in the relational database and, although other monitoring tools can access the data through SQL requests, this does not configure a scalable monitoring method. For the streaming application, the predictions can be easily published in a new Kafka topic from which other tools can consume, but, especially for the shared database solution, the structure to share the models' results could be improved in the general UML diagrams.

Besides the requirements evaluation and other qualitative analysis, it is important to evaluate if all the adequate samples published by the manufacturing in the data source have generated a prediction value made by the application. Being able to develop and deploy the solutions fast and with high maintainability is only a good indicator if the system's main task is accomplished.

Hence, after performing tests with the worst scenario possible - all available data published at the same time in the database - for both deployed applications, two SQL queries were performed. The first one requested the count "SampleId" in the Samples table where the "MachiningProcess" column has values for which the predictions should be performed. The other one selected the count of distinct "SampleId" values in the Prediction table. Both queries returned the value of 17520, which indicates that all sample published by the script that simulates the manufacturing was read by the applications, transformed, feed into the machine learning models, and got the result published back into the DB.

Furthermore, a Docker image of the Confluent control center was used to monitor Kafka during the execution of the streaming environment. In this tool, it is possible to check all the consumers of the cluster topics and how many event messages were published but not consumed. Figure 28 shows an image of this data taken after the worst scenario simulation. It is possible to observe that there was no message left behind in any topic, which also contributes to the understanding that the deployable

solution for this environment performed as expected in each execution step.

Figure 28 – Messages left behind in each consumer according to the Confluent control center after the execution of the worst case scenario

<b>Consumer group ID</b>	<b>Messages behind</b>	<b>Number of consumers</b>	<b>Number of topics</b>
<u>data-preparation</u>	0	1	1
<u>random-forest-consumer</u>	0	1	1
<u>logistic-regression-consumer</u>	0	1	1

### 6.3 Solution overview

In a nutshell, the solution provided the needed structure to convert procedures and machine learning models acquired in research projects into solid deployable applications. The CI pipeline developed provided automation to important operational steps, whereas the tests performed using the applications validated the benefits of the general software modeling.

The guideline documents elaborated configure themselves as important and practical instructions on how to use the project's solutions in future ML applications inside the institute. Moreover, managers from another ML project inside the department have demonstrated interest in using the study outcomes and meetings to adapt the solution have been made.

## 7 Conclusion

With the results presented in the previous chapter in mind, it is possible to state that the project's goals were achieved. The objectives were fulfilled and the solution will provide integration skills to the development and deployment of future ML applications on the institute. The CI pipeline and the configuration files can be imported directly from a template repository, whereas the UML diagrams together with the guideline documents provide direct instructions on how to build a robust and maintainable application for the use cases range.

Regarding the relation of the Final Project Work (FPW) with the Control and Automation Engineering course, the mandatory subjects Introduction to Computer Science for Automation (DAS5334), Information Structure Fundamentals (DAS5102), and Systems Development Methodology (DAS5312) provided to the student the needed basis of the software field that was crucial to enable the development of the general UML diagrams and the YAML file that describes the CI pipeline.

Although the project's outputs focus on the operations field, it was important to have a solid knowledge of how artificial intelligence applications operate to provide DevOps solutions. Hence, the Artificial Intelligence Applied to Control and Automation (DAS5341) subject also relates to the implementations performed.

Furthermore, it was crucial to understand how operating systems work to properly configure some of the GitLab platform features, such as the usage of the GitLab Runner in a docker-in-docker approach, and to model the streaming environment. In this regard, the student considers that the Concurrent Programming and Real-Time Systems (DAS5306) subject provides important knowledge on the course's path.

The worldwide health crisis and the global measures taken against it imposed additional challenges to the FPW elaboration. Since it was conceived in a partnership with a German organization, the necessary travel restrictions established for the second semester of 2021 due to the COVID-19 cases raise in Brazil implied considerable delay in some of the scheduled tasks. The support provided by IPT and UFSC was essential to overcome this hurdle.

It is worth mentioning that the student acquired different knowledge about the challenges for the production quality enhancement in modern industrial facilities and how to overcome them during the project development. The opportunity to explore relevant software operation fields such as DevOps and continuous integration at an institute that is at the forefront of applied research was a unique experience that contributed to the student's formation.



## 7.1 Future projects

Finally, having in mind the ideas that came up during the project evolution and the fields that were not covered considering the whole life cycle of machine learning models in production, the following topics are suggested as future projects that can aggregate value to the elaborated solution:

- Implement and automate monitoring tasks for the machine learning models after they are deployed. This can provide the ability to perform continuous training tasks that evaluate when the models inside the application are decaying and automatically retrain them with new data.
- Provide general continuous deployment guidelines. As mentioned in section 5.1, the implemented solution automates continuous integration and continuous delivery tasks releasing a new version of the application when there is a merge in the master branch. However, the Docker container must be started in the production environment manually. Therefore, defining how to link the environments and adapt the CI pipeline to automatically deploy the releases without any human intervention would be an interesting improvement of the solution.
- The guidelines of Appendix D propose the automation of some tasks in the research phase where the ML model is acquired through a series of experiments. Test and validating these suggestions in a project that does not have the research phase finalized could bring more useful insights to speed up data science actions.
- Add jobs in the CI pipeline that make security tests in the application and the release of the containers, since the use cases can deal with sensible manufacturing data. The DevSecOps is the field that incorporates security actions in the development cycle following some agile framework and has great potential to extend the elaborated solution (MATTHEWS, 2018).
- Test the solution in a real use case scenario in one of the institute's partnerships to validate the deployment in a non-simulated environment.
- As mentioned in section 6.2, the requirement to easily monitor the prediction results was not completely fulfilled. Therefore, an adaptation of the UML diagram to provide a standard package that deals better with the predictions monitoring could represent an important improvement.

## Bibliography

Apache. *Introduction: Everything you need to know about Kafka in 10 minutes*. 2021. Available from: <<https://kafka.apache.org/intro>>. Visited on: 06/10/2021. Cit. on page 37.

BELCK, L. A. Internship Report, *Development of an automated pattern detection algorithm using machine learning*. 2020. Cit. on page 21.

BOSE, S. *End To End Testing: A Detailed Guide*. 2021. Available from: <<https://www.browserstack.com/guide/end-to-end-testing>>. Visited on: 30/09/2021. Cit. on page 35.

BUCHANAN, I. *History of DevOps*. 2021. Available from: <<https://www.atlassian.com/devops/what-is-devops/history-of-devops>>. Visited on: 04/09/2021. Cit. 2 times on pages 27 and 28.

CHAPMAN, P. et al. *CRISP-DM 1.0 Step-by-step data mining guide*. [S.l.], 2000. Cit. 3 times on pages 24, 25, and 27.

CHAWLA, D. *Difference between Docker Image and Container*. 2020. Available from: <<https://www.geeksforgeeks.org/difference-between-docker-image-and-container/>>. Visited on: 30/09/2021. Cit. on page 39.

CHICCO, D.; JURMAN, G. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, v. 21, n. 6, 2020. Cit. on page 66.

Confluent. *Streaming Data: How it Works, Benefits, and Use Cases*. 2021. Available from: <<https://www.confluent.io/learn/data-streaming/>>. Visited on: 06/10/2021. Cit. 2 times on pages 36 and 37.

COURTEMANCHE, M.; MELL, E.; GILLIS, A. S. *What is DevOps? The ultimate guide*. 2021. Available from: <<https://searchitoperations.techtarget.com/definition/DevOps>>. Visited on: 04/02/2021. Cit. on page 27.

Data Science Process Alliance. *What is CRISP DM?* 2018. Available from: <<https://www.datascience-pm.com/crisp-dm-2/>>. Visited on: 03/09/2021. Cit. on page 26.

DB-Engines. *DB Engines Ranking*. 2021. Available from: <<https://db-engines.com/en/ranking>>. Visited on: 12/01/2021. Cit. on page 67.

Debezium. *Debezium connector for MySQL*. 2021. Available from: <<https://debezium.io/documentation/reference/stable/connectors/mysql.html>>. Visited on: 13/01/2021. Cit. on page 69.

Docker Inc. *Use containers to Build, Share and Run your applications*. 2021. Available from: <<https://www.docker.com/resources/what-container>>. Visited on: 30/09/2021. Cit. 2 times on pages 39 and 40.

FOWLER, M. *Contract Tests*. 2011. Available from: <<https://martinfowler.com/bliki/ContractTest.html>>. Visited on: 30/09/2021. Cit. on page 35.

- FOWLER, M. *BroadStackTest*. 2013. Available from: <<https://martinfowler.com/bliki/BroadStackTest.html>>. Visited on: 30/09/2021. Cit. on page 36.
- FOWLER, M. *Test Double*. 2016. Available from: <<https://martinfowler.com/bliki/TestDouble.html>>. Visited on: 09/09/2021. Cit. on page 33.
- FOWLER, M. *Software Testing Guide*. 2019. Available from: <<https://martinfowler.com/testing/>>. Visited on: 09/09/2021. Cit. on page 32.
- FRAUNHOFER-GESELLSCHAFT. *Fraunhofer Institutes and Research Units*. 2020. Available from: <<https://www.fraunhofer.de/en/institutes.html>>. Visited on: 30/08/2021. Cit. on page 17.
- Fraunhofer IPT. *A brief history of the Fraunhofer IPT*. 2019. Available from: <<https://www.ipt.fraunhofer.de/en/Profile/History.html>>. Visited on: 06/09/2021. Cit. on page 17.
- Fraunhofer IPT. *The Fraunhofer IPT*. 2019. Available from: <<https://www.ipt.fraunhofer.de/en/Profile.html>>. Visited on: 30/08/2021. Cit. on page 17.
- Fraunhofer IPT. *Mission Statement*. 2019. Available from: <<https://www.ipt.fraunhofer.de/en/Profile/MissionStatement.html>>. Visited on: 06/09/2021. Cit. on page 17.
- Fraunhofer IPT. *Our remit: to reflect on entire processes*. 2019. Available from: <<https://www.ipt.fraunhofer.de/en/Profile/Remit.html>>. Visited on: 20/08/2021. Cit. on page 18.
- Fraunhofer IPT. *Partners within the Fraunhofer-Gesellschaft*. 2020. Available from: <<https://www.ipt.fraunhofer.de/en/Profile/cooperation/fraunhofer-gesellschaft.html>>. Visited on: 30/08/2021. Cit. on page 17.
- Fraunhofer IPT. *Production quality*. 2021. Available from: <<https://www.ipt.fraunhofer.de/en/Competencies/Productionqualityandmetrology/Productionquality.html>>. Visited on: 03/02/2022. Cit. on page 19.
- Fraunhofer IPT. *Projects at a glance*. 2022. Available from: <<https://www.ipt.fraunhofer.de/en/projects.html>>. Visited on: 03/02/2022. Cit. on page 18.
- GITLAB. *DevOps*. 2021. Available from: <<https://about.gitlab.com/topics/devops/>>. Visited on: 04/09/2021. Cit. on page 28.
- HALL, T. *DevOps Pipeline*. 2021. Available from: <<https://www.atlassian.com/devops/devops-tools/devops-pipeline>>. Visited on: 04/09/2021. Cit. on page 28.
- HARRIS, C. R. et al. Array programming with NumPy. *Nature*, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. Available from: <<https://doi.org/10.1038/s41586-020-2649-2>>. Cit. on page 65.
- HEYMANN, H. *Deployment of Machine Learning Models for Predictive Quality in Production*. Master thesis, 2021. Cit. 2 times on pages 50 and 51.
- HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007. Cit. on page 65.

IBM. *What is Business Intelligence? Analyze business data to gain actionable insights and inform decision-making*. 2021. Available from: <<https://www.ibm.com/topics/business-intelligence>>. Visited on: 14/12/2021. Cit. on page 13.

Jupyter Team. *The Jupyter Notebook*. 2015. Available from: <<https://jupyter-notebook.readthedocs.io/en/stable/notebook.html>>. Visited on: 03/10/2021. Cit. on page 19.

Katalon. *Best 14 CI/CD Tools You Must Know*. 2021. Available from: <<https://www.katalon.com/resources-center/blog/ci-cd-tools/>>. Visited on: 03/02/2021. Cit. on page 54.

KLEINBAUM, D. G. et al. *Logistic regression*. New York: Springer, 2002. Cit. on page 24.

KORSTANJE, J. *The F1 score*. 2021. Available from: <<https://towardsdatascience.com/the-f1-score-bec2bbc38aa6>>. Visited on: 12/01/2021. Cit. on page 66.

MARTIM, R. C. *Agile Software Development, Principles, Patterns, and Practices*. [S.l.]: Prentice Hall, 2002. Cit. on page 30.

MARTIM, R. C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. [S.l.]: Prentice Hall, 2017. Cit. 2 times on pages 30 and 31.

MARTIN, M. *What is Software Engineering? Definition, Basics, Characteristics*. 2021. Available from: <<https://www.guru99.com/what-is-software-engineering.html>>. Visited on: 04/02/2021. Cit. on page 27.

MATTHEWS, K. *Understanding DevSecOps in Data Science*. 2018. Available from: <<https://towardsdatascience.com/understanding-devsecops-in-data-science-93b695a0d8ab>>. Visited on: 26/01/2021. Cit. on page 80.

NARKHEDE, N.; SHAPIRA, G.; PALINO, T. *Kafka: The Definitive Guide*. [S.l.]: O'Reilly Media, 2017. Cit. 2 times on pages 37 and 38.

OSHEROVE, R. *The Art of Unit Testing: With Examples In C#*. [S.l.]: Manning Publications Co., 2014. Cit. 2 times on pages 33 and 34.

PARSONS, C. *What Is a Machine Learning Model?* 2021. Available from: <<https://blogs.nvidia.com/blog/2021/08/16/what-is-a-machine-learning-model/>>. Visited on: 27/01/2021. Cit. on page 24.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011. Cit. on page 65.

PITTET, S. *Continuous integration vs. continuous delivery vs. continuous deployment*. 2021. Available from: <<https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>>. Visited on: 04/09/2021. Cit. on page 29.

QRA Corp. *Functional vs Non-Functional Requirements: The Definitive Guide*. 2019. Available from: <<https://qracorp.com/functional-vs-non-functional-requirements/>>. Visited on: 27/10/2021. Cit. on page 41.

REHKOPF, M. *Continuous Delivery Principles*. 2021. Available from: <<https://www.atlassian.com/continuous-delivery/principles>>. Visited on: 04/09/2021. Cit. on page 29.

REHKOPF, M. *What is Continuous Integration?* 2021. Available from: <<https://www.atlassian.com/continuous-delivery/continuous-integration>>. Visited on: 04/09/2021. Cit. on page 29.

SALTZ, J. *CRISP-DM is Still the Most Popular Framework for Executing Data Science Projects*. 2020. Available from: <<https://www.datascience-pm.com/crisp-dm-still-most-popular/>>. Visited on: 02/09/2021. Cit. on page 25.

SUN, S. *CNC Mill Tool Wear: Variational CNC machining data*. 2017. Available from: <<https://www.kaggle.com/shasun/tool-wear-detection-in-cnc-mill?select=train.csv>>. Visited on: 25/11/2021. Cit. 3 times on pages 65, 67, and 69.

The pandas development team. *pandas-dev/pandas: Pandas*. Zenodo, 2020. Available from: <<https://doi.org/10.5281/zenodo.3509134>>. Cit. on page 65.

TYAGI, H. *What is MLOps — Everything You Must Know to Get Started*. 2021. Available from: <<https://towardsdatascience.com/what-is-mlops-everything-you-must-know-to-get-started-523f2d0b8bd8>>. Visited on: 05/09/2021. Cit. on page 29.

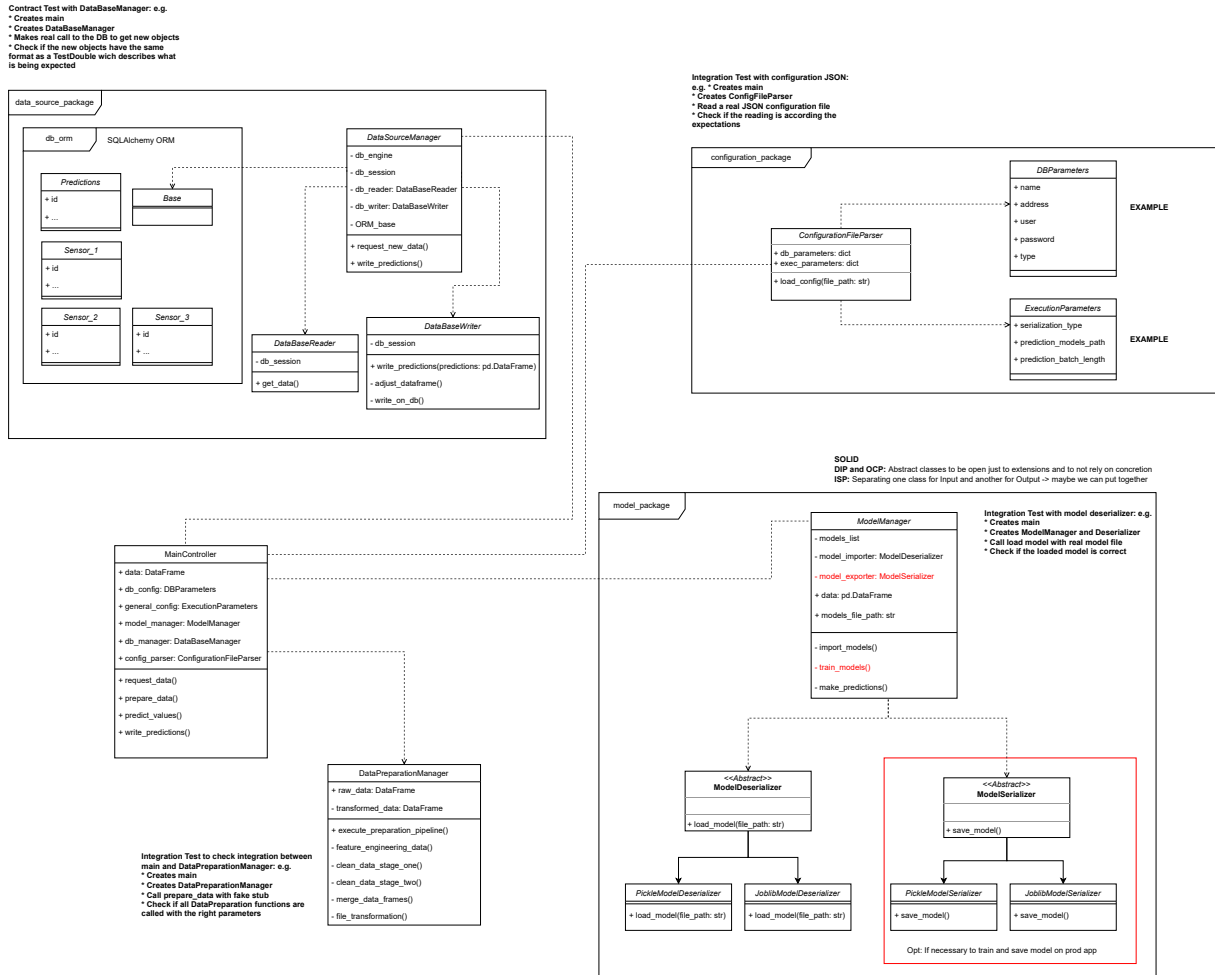
WACKER, M. *Just Say No to More End-to-End Tests*. 2015. Available from: <<https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html>>. Visited on: 16/11/2021. Cit. on page 58.

WASKOM, M. L. seaborn: statistical data visualization. *Journal of Open Source Software*, The Open Journal, v. 6, n. 60, p. 3021, 2021. Available from: <<https://doi.org/10.21105/joss.03021>>. Cit. on page 65.

YIU, T. *Understanding Random Forest*. 2019. Available from: <<https://towardsdatascience.com/understanding-random-forest-58381e0602d2>>. Visited on: 28/01/2021. Cit. on page 24.

## APPENDIX A – Complete general class diagram

Figure 29 – General class diagram for production applications

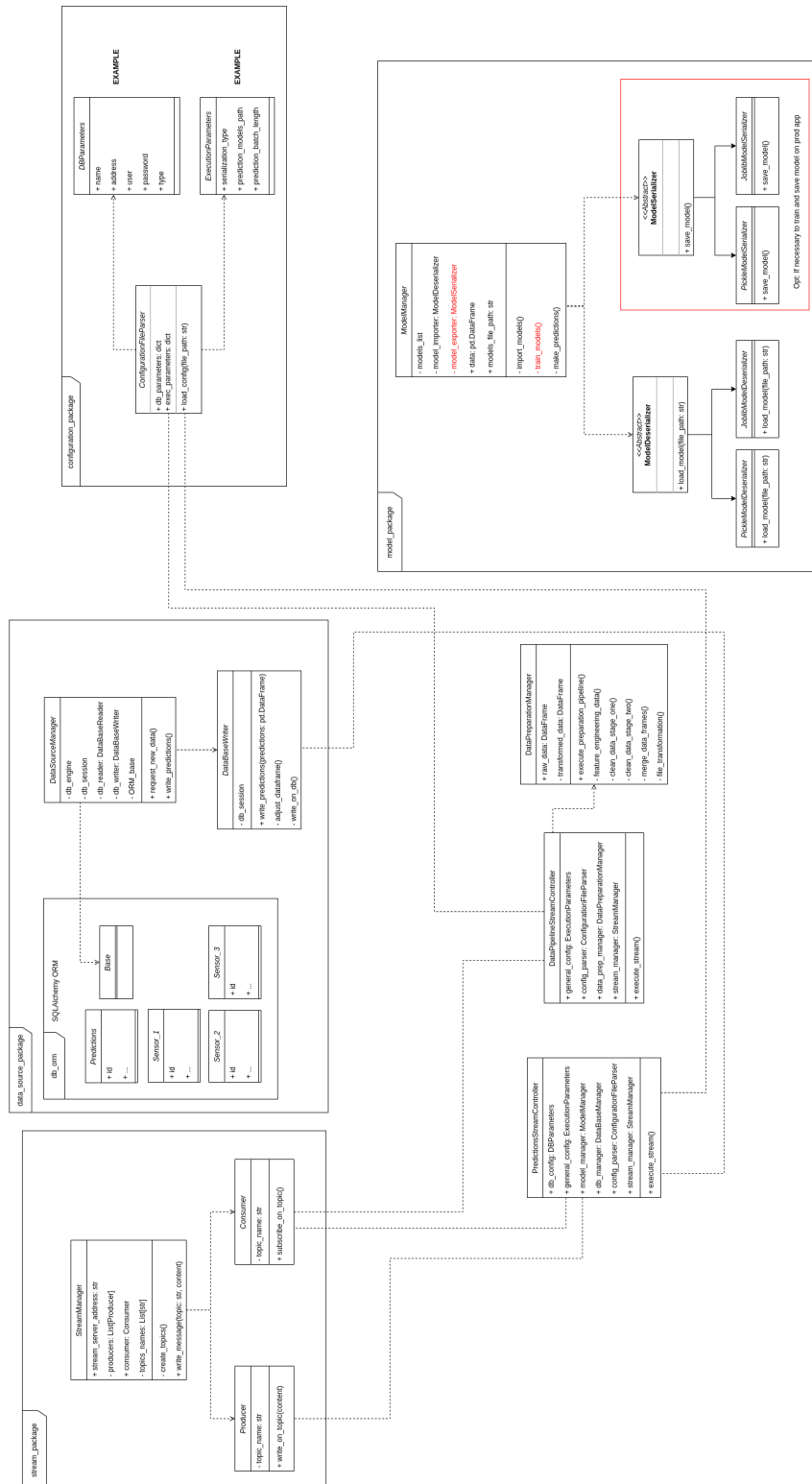


Source: Personal Archive



### APPENDIX B – Complete class diagram for streaming architecture

Figure 30 – General class diagram for production applications with streaming architecture



Source: Personal Archive



## APPENDIX C – New project creation guidelines

### New project creation guidelines

This document describes in guidelines the first steps to be done when creating a new machine learning project in order to have a Continuous Integration cycle correct configured running on your GitLab repository.

#### Create a new project on GitLab

Following the CRISP-DM methodology, the project will usually include some development stages turned to experimentation and another stage turned to a deployable application. It is recommended to divide this development in two different GitLab repositories (called here as research application and production application), especially because of the different recommended approaches of branches management. However, it is also possible to keep both stages on the same repository.

The research application does not involve many CI details, so there are no special guidelines for its creation. The next steps described on this document are regarding the production application. There is a pre-configured repository from which the production application must start.

#### Copy repository from pre-configured template

1. Go to your main GitLab page and select 'New Project' on the right-top corner.
2. Choose the 'Import Project' option and select 'GitLab export' on the next page.
3. Download the exported project template file. Add your new project's information and upload the template file.

#### Clone repository on your local machine

To edit and start developing on the project in your local machine you must [clone the repository](#) using git, for windows you can download the Git Bash from [Git for Windows](#). It is also recommended that you configure a ssh key for your user on the current project following [GitLab SSH key tutorial](#).

#### CI Cycle files

On the pre-configured template that are two main files that will ensure the correct operation of the CI cycle. The [.gitlab-ci.yml](#) contains the description of the whole pipeline with its stages and jobs, while the [Dockerfile](#) contains the instructions for the application image generation that will be executed by one of the jobs described on the pipeline. It is probably not necessary to make any changes on these files, but it might be interesting to do if you want to customize its stages or the project's development workflow.

#### Configure GitLab Runner

After cloning the template repository a CI cycle will be configured to automatically run all the CI stages. When the cycle is triggered, these jobs will be picked by a GitLab Runner instance that will effectively execute it. GitLab has some shared Runner instances, but they have a monthly usage time limit, that is why we need to configure a private GitLab Runner on a local machine. You can find detailed information about this installation on the [GitLab Runner page](#), but the following step by step already contains the installation guidelines specific to our projects.

1. We will install an instance of GitLab Runner in a docker container, so you need to have [Docker](#) installed on your machine.
2. Mount the docker volume with the runner configuration: `bash docker volume create gitlab-runner-config`
3. Start GitLab Runner container with the mounted volume: `bash docker run -d --name gitlab-runner --restart always \ -v /var/run/docker.sock:/var/run/docker.sock \ -v gitlab-runner-config:/etc/gitlab-runner \ gitlab/gitlab-runner:latest` Obs.: Now the container is running with a downloaded gitlab-runner image. You can execute any [GitLab Runner command](#) following this syntax:

```
docker run <chosen docker options...> gitlab/gitlab-runner <runner command and options...>
```

4. Go to your project's GitLab page on 'Settings > CI/CD > Runners', on the 'Specific Runners' section copy the registration URL and token. Obs.: You can also disable the option 'Enable shared runners for this project' on this same page.
5. Since we want to have a docker-in-docker service enabled to generate a docker image of our application on the CI pipeline, we will register a runner linked with our project with some extra configurations:

```
docker run --rm -it -v gitlab-runner-config:/etc/gitlab-runner gitlab/gitlab-runner:latest register -n \
--tag-list docker-runner \
--url REGISTRATION_URL \
--registration-token REGISTRATION_TOKEN \
--executor docker \
--description "Runner set on IPT's local machine number X to run the CI pipeline of projects Y and Z" \
--docker-image "docker:19.03.12" \
--docker-privileged \
--docker-volumes "/certs/client"
```

Note that the 'REGISTRATION\_URL' and 'REGISTRATION\_TOKEN' must be replaced with the data from step 4.

6. [OPTIONAL] You can make advanced configuration changes on your runner instance by changing its config.toml file ([Runner Configs](#)). To do that, you can follow these steps:

1. Enter the runner container bash:

```
docker exec -t -i gitlab-runner /bin/bash
```

2. Navigate to your config.toml folder and check the current configurations:

```
cat config.toml
```

3. Use sed to add/edit configurations. Check the changes that will be implemented on the file and add the flag -i to effectively make it, **always check and confirm the changes before implementing it**.

```
sed 's/old-text/new-text/g' config.toml  
sed -i 's/old-text/new-text/g' config.toml
```

For example, a useful configuration is to allow the runner to run jobs concurrently. This can be done changing concurrent field:

```
sed 's/concurrent = 0/concurrent = 2/g' config.toml  
sed -i 's/concurrent = 0/concurrent = 2/g' config.toml
```

7. To leave the container bash just press 'ctrl + p' followed by 'ctrl + q'.

Finishing this steps you will have the template CI cycle configured for your repository. You can then start developing your deployable application, check the [Development Guidelines](#) for references on that.

## APPENDIX D – Development guidelines

### Development guidelines

This document describes in guidelines the development structures to be used when creating a new machine learning application turned to production. To know more about how to create a new project refer to the [new project creation guidelines](#).

#### Different applications

The first thing to consider when developing such an application is that, following the CRISP-DM steps, there is one stage of development that deals mostly with data science procedures and models experimentation. This will compound a research application that later will be productionized to result in the final deployable application.

#### Research application

On the early stages the application will try to define the data preparation steps and make several experiments with different machine learning models. Therefore, the development should have the workflow preferred by data scientist on a very flexible environment.

#### Data and model versioning

In most programming projects the code versioning is a crucial step to make possible to keep track of the changes and enable parallel work between the members of the team. To perform that, a Git repository is used and here this is not different.

However, on machine learning projects we will also have different versions of datasets and models among all the several experiments being led. To keep track of them we could also use the git repository having different instances of these files on the different branches. On the other hand, these files can be really big and may take an undesired huge space of our repository, since Git is essentially projected to keep code files.

With that in mind, we can use a tool called Data Version Control ([DVC](#)) to keep the content of all our data and model files inside a shared storage but making only the track of them being available at the Git repository. Following you can check the main additional commands you have to use to develop with DVC. DVC runs on top of Git, so make sure you have both installed.

#### Connecting the remote storage

The first step is always to connect the git repository with a sharing storage using DVC. There are several types of storage that can be used and you can refer to the [DVC remote add](#) documentation for a detailed explanation. On the following step by step is explained how to configure DVC with the Fraunhofer OwnCloud service using WebDAV protocol.

1. Go to your OwnCloud page and get the WebDAV address of your files clicking on 'Settings' in the bottom left corner.
2. Navigate to your git repository on your local machine and execute the following command replacing the address with the one you got on step 1 and adding the path to the folder you want to use on the end of it:

```
dvc remote add -d remote_storage_name \
  webdav://example.com/owncloud/remote.php/dav/files/myuser/projetc_Y_data_dvc_folder
```

3. Add the user and password of your access to the remote storage so DVC can upload and download files from there. **WARNING: Do not forget to use the `--local` flag on these commands, in that way your user and password will be saved on a configuration file only on your machine.**

```
dvc remote modify --local remote_storage_name user myuser
dvc remote modify --local remote_storage_name password mypassword
```

Alternatively, you can set DVC to ask your password when needed instead of configuring one:

```
dvc remote modify remote_storage_name ask_password true
```

#### Adding data/model files

After connecting the repository we are now able to send and get files from the remote storage using the track files stored on git. This files management and tracking happens automatically using git and dvc commands.

To add files or folders we can follow these simple commands: 1. Add the file to DVC, which will automatically create a tracking file:

```
dvc add data/data_example.xml
```

2. Add the generated tracking file on git running the command suggested after running the command on step 1. It will be similar to:

```
git add data/data_example.xml.dvc data/.gitignore
```

3. Now you can make a commit on your git branch indicating the change:

```
git commit -m "Add new data file"
```

4. Push the changed data files to the remote repository:

```
dvc push
```

If instead of pushing a new data file to the remote repository you want to download files that you could have accidentally deleted you can use the `dvc pull` command, which will update your data files to match the tracking state that is on your current branch.

```
dvc pull
```

### Checkout to another branch

One of the main reasons to track your code project using git is that you can easily change from one branch to another visiting different states of the code. This tracking of data and model files that we just implemented with DVC uses git to give the same benefits also for these types of files. Then, whenever you change branches on your repository (`git checkout branch_name`) you should also run a `dvc checkout` command to update your local dvc tracked files.

### Experiments tracking

On the research application we are expecting to have a code development focused in make experiments to check the result of different data preparations and machine learning models. So, as mentioned before, the idea is to have a flexible structure that allows the data scientists to go deep into these different experiments.

However, it is also important to keep track of these experiments to have a central place that allows the visualization of its results. With that in mind, it is recommended to use the open source MLflow Tracking tool. You can refer to the [complete documentation](#) of MLflow, but here there are some steps that can be useful for our projects:

- You can install the mlflow running `pip install mlflow` and import it with `import mlflow`
- Then, we can use MLflow Tracking importing the library inside our Python scripts and running some commands. However, it is important to know that the tool will, by default, save the experiments tracking inside a new folder it will create on the same folder of your running script called `mlruns`. If you want to change it you can set a new tracking uri inside your python file using the following library method: `python mlflow.set_tracking_uri("file:///tmp/my_tracking")`
- The MLflow Tracking consider the following division:
  - Experiment: A label to track an experiment being made, can contain several runs.
  - Run: A label inside an experiment which represents one of the tests made. A run can log parameters, metrics and artifacts.
  - Parameter: A key-value input that will not change, e.g. ("Model type": "Decision Tree Best Estimator")
  - Metric: A key-value input that will result from the tests, value has to be numeric.
  - Artifact: Output files in any format, e.g. the model file serialized.
- Then, when starting an experiment you can create one on your Python file with the command: `python try: mlflow.create_experiment("Hiperparameter Tunning") except mlflow.exceptions.MlflowException as e: print("Exception: {}".format(e))`
- Then, you can start a run in which you will set your experiment actions and, at the end, log the relevant information. Here is an example:

```
with mlflow.start_run(run_name="Test nº 1",
                    experiment_id=mlflow.get_experiment_by_name("Hiperparameter Tunning").experiment_id):

    # Train the model with the parameters here

    mlflow.log_param("Model type", "Decision Tree Best Estimator")
    mlflow.log_param("Search", "RandomizedSearchCV")
    mlflow.log_param("n_iter", 1000)
    mlflow.log_param("cv", 4)
    mlflow.log_param("random_sate", 0)
    mlflow.log_param("n_job", -1)
    mlflow.log_metric("Accuracy", round(accuracy_score(Y_test, Y_pred), ndigits=4))
    mlflow.log_metric("F1 Score", round(f1_score(Y_test, Y_pred), ndigits=4))
    mlflow.log_metric("MCC", round(matthews_corrcoef(Y_test, Y_pred), ndigits=4))
```

- The library also have other log methods to track different objects that can be useful. For example, there is a specific [package to deal with sklearn models](#).
- After logging the desired information, you can visualize all the experiments and runs on the mlflow tracking UI running `mlflow ui` on the command line being on the directory that your tracking folder is located. It will than host the interface on `http://localhost:5000/`.

## Production application

After achieving a model and a data preparation pipeline suitable for the project requirements, the project must build a more robust application that can be deployed in production. Following the [new project creation guidelines](#), a CI pipeline will be already configured to help on this development, which should not start from zero but should take the research application core as a strong reference.

To help with the transition from research to production, some simple UML diagrams were made with the idea to guide the deployment plan and development. The team should not be limited to these diagrams but should initiate with them and make the needed changes during the deployment planning.

### Sequence diagram

The [sequence diagram](#) tries to illustrate the different execution cycles that the program will have and how is the relation between the packages. It is divided into three straightforward stages that the greatest part of our ML applications will have to have: firstly it gets the data from the data source, then it makes the needed transformations on this data (based on the data preparation pipeline that came from research) and makes the predictions (using the model that came from production), and the last stage writes back the predicted values inside the data source.

There is also an optional interaction outside the main loop where the main package reads some run configurations. This step can be used to set on a file some information that the program may need to run, such as users and passwords, path to the model, etc.

Then, the central loop of the program starts with the main package communicating with the DB one to request a new set of data. It can also use the serialized file to load and save the data in case of future error or to know which was the last value of the database to be read by the program.

Proceeding, there should be one package to deal with the data transformations and another one to deal with the models, which can be constructed on each iteration or imported from a serialized model. Finally, the main package should communicate again with the DB one to write back the values.

Sequence Diagram

## Class diagram

The [class diagram](#) aims to go further on the architecture definition and provide to the development team an initial proposal on how to organize the modules, classes, and methods. Again, it should be used as a reference guide and not as a diagram to be strictly followed.

As described on the sequence diagram, the idea is to have the main script that will contain an instance of the other packages and will coordinate the execution workflow. The model package will have a central script that will be responsible to manage all the use of the models, which will vary according to the models' approach to be used (e.g. stacked models, cascades models). The whole architecture is taking the SOLID principles as a design base and this can be observed in the abstract classes created for the models' serialization where the structure does not rely on concretions and is also open for extension but closed for modification.

Another important package is the one responsible to communicate with the database, which will have two main methods accessible for the main class. If using the SQLAlchemy, the ORM feature can be explored and the creation of classes that represent database tables can be really helpful. The correct exploration and definition of which data will be acquired from which database table and how the predictions will be written back on it is a step to be done in the planning stage.

Also, the research application considerations will form a data preparation pipeline that will compose a script and will have its methods called by the main according to the sequence diagram. Finally, the execution configuration can be described on an optional but useful package where the information can be represented on structural classes.

Class Diagram

## General python libraries

In addition to the UML diagrams, some considerations of useful Python libraries can be helpful while developing the production application.

### Dask

Regarding machine learning, NumPy, pandas, and scikit-learn are the most commonly used libraries for Python projects. [Dask](#) is a Python library that uses parallel computing to perform most of the actions usually performed by the other mentioned libraries. Its usage can improve the performance of our application, especially when we have to deal with Big Data techniques. Dask also abstracts these computational improvements and is as easy to use as the other libraries, proving to be a powerful tool to program the data preparation pipeline and the models' acquirement inside the production application.

### Model serialization

In the cases where the machine learning model will be built outside the production application, a serialization library can be used to save the model in a file and import it when needed inside the program. [Pickle](#) is maybe the most known library for this action and its usage is pretty straightforward. [Joblib](#) is another library that also deals with files' serializations, but using some computational techniques that can improve the performance when dealing with really long files, especially containing huge *numpy* arrays.

The reading of binary files using pickle is usually really fast far away from being an execution bottleneck. However, for some specific cases, joblib can provide a nice improvement. Since the deserialization can be performed with few commands, it is stimulated that both libraries are implemented inside the application relying on abstractions, as shown on the general class diagram. The user can set which library to use on the configuration file.

### SQLAlchemy

Also, our program needs a framework to help with reading and writing information on the production database's tables. [SQLAlchemy](#) is a SQL toolkit that can help to efficiently make flexible SQL requests and it differentiates itself by its object-relational mapping feature, which allows the developer to map the program's classes with tables inside the database facilitating the actions to be executed. Therefore, this Python library can be really useful when planning and developing the database package.

### Configuration file

If it is useful for the application to have a configuration package, the most common approach is to use a JSON file that maps the config parameters in a key-value format. Python has a built-in data structure called dictionary which is a representation very similar to the JSON format and very easy to work with. To translate the configuration file in one or many dictionaries we can use the native [json](#) library.

## APPENDIX E – Project management guidelines

### Project management guidelines

---

This document focus to clarify the most important project management questions for machine learning applications turned to production.

#### Different projects

As described in the [development guidelines](#), this type of application will usually be developed with two different projects, one to make data science experiments in order to get the best model and data preparation steps and another one to properly integrate the program with the production environment. Thus, each project will have its management characteristics.

#### Git branches management

Branches are one of the main features provided by Git and an important point is to define how to manage them. Creating a branch will allow the developers to have different tagged working lines inside the same project, which can then be integrated later.

#### Single trunk-based

Since the production application will have its features developed integrated in the same program, the approach to carry the project with a single main branch that accumulates all the versions that go to production works very well. Its development is based on three types of branches:

- The **master** branch: This is the main branch that will generate the version to be deployed. There is only one master branch, and it will only accept merge requests from the development one.
- The **development** branch: This is an integration branch used to merge the different features already completed on the feature branches. It should be used to test if all the functionalities are working fine together before sending it to the master. There is only one development branch.
- The **feature** branches: Whenever a new feature will be developed, a feature branch should be created having the last version of the development branch as its source. The developer will then have a separated version of the code to work in parallel to other tasks. When the feature is completed, this branch should be merged with the development one, always solving the possible merge conflicts manually.

With this approach, it is possible to have a well-versioned program making it easier to track bugs and harder to send undesired code to production. The workflow feature branches → development → master should always be respected, and it is important to have the awareness of the whole team when merging to master since it will represent a new version to be deployed. The master version should then be tagged, to make it even easier to track which version is running on production and to perform rollbacks in case of any problem.

To tag a version of the code, the following commands can be used:

1. Create the tag with its tag number and message: `bash git tag -a 1.0.0 -m "First completed version"`
2. Send the tag to the remote repository: `bash git push origin 1.0.0`

Each project can have its standard to define the version tag number. A simple approach is to do one of the changes: \* Increase the first number from left to right when the merge makes changes incompatible with the previous version. \* Increase the middle number when the merge adds a functionality compatible with the previous version. \* Increase the last number from left to right when the merge makes bug fixes compatible with the previous version.

#### Experiments based

In contrast to the production application, the research one will have its development guided by experiments and, many of them will implement code that will not be used on the final solution. For that reason, it does not make sense to have a strict branch creations methodology for this repository. In this case, it is recommended that each new experiment creates a new branch, which can be merged among each other to result in new experiments. To make a proper tracking of data and model files at this stage it is very important to use the DVC and MLFlor tracking tools described in the [development guidelines](#).

#### Sprints

Some of the most common agile methodologies are based on sprints, where a set of activities are planned to be done iteratively on the time-boxed sprint period. The implementation of these methods can be integrated together with Gitlab

First, the planned activities and new features ideas can be set on the issues board where it will stay as "open", working as a product backlog. It is a good practice to briefly explain in the issues-description for which **user** it will be useful, what is the **functionality** to be implemented, and what are its **benefits**.

Then, in the sprint planning phase, the team can create a milestone with the correct start and planned end date. Back on the issues board, the team must decide which issues will be implemented on that sprint and assign them to the milestone and to the developer that will be responsible for it. A new board will appear on the milestone page with the lists "Open", "In progress", and "Closed" that should help on the sprint's management. When a developer is free, it should take one of the issues and create a feature branch for its implementation putting the issue tag number on the name of the branch (e.g. 54-awesome-feature), which will automatically close the issue when the feature is merged into the development.

## APPENDIX F – Testing guidelines

### Testing guidelines

To achieve a well-integrated and easily deployable solution, we must establish a good testing pipeline. This stage represents an important part of the CI cycle and can usually have most of its steps automated. This document describes in guidelines the best practices to build advantageous tests on machine learning projects turned to production.

#### Test pyramid

There are several types of software tests and their application can be analyzed for each specific project. The following tests explained in this document are the ones considered most relevant for our kind of application. The simplified test pyramid exhibits these tests where the way top -> bottom increases the number of tests to be implemented, their running frequency, their execution time, and their level of isolation.

Test pyramid

#### Unit tests

The base of our test pipeline should always rely on unit tests. These simple and automated tests are very powerful when well-structured, having the objective to check if each one of the program's units of work is performing the expected actions when isolated.

In practice, whenever a developer implements a new feature, several unit tests can be developed concurrently. Then, in any other future change on the code the CI cycle will automatically run the tests and ensure that the old features still have the same expected behaviour.

Unit tests are usually developed using some testing framework, for the Python applications we can use the unittests library. Also, they tend to have a structure similar to: 1. Create needed test double objects. 2. Perform the action of the unit of work under test. 3. Assert against something to check the result.

An example can illustrate it better. Let's suppose that our program has a method that receives a list of unordered integers and organizes it in crescent order. It could result in the following unit tests:

```
# Imports unittest library
import unittest

# Imports production code to be tested
from array_manager import ArrayManager

class ArrayManagerTests(unittest.TestCase):

    # Declare test doubles to be used on more than one test
    stub_array_manager = ArrayManager()

    def test_array_crescent_order_organizer_receives_disordered_array_reorders_it(self):
        # Declare specific test doubles
        fake_array = [0, 1, 9, 3, 5, 6, 99, 8]
        reordered_array = [0, 1, 3, 5, 6, 8, 9, 99]

        # Perform action under test
        new_array = self.stub_array_manager.array_order_organizer(fake_array)

        # Make asserts to ensure expected result
        self.assertTrue(len(fake_array) == len(new_array))
        self.assertEqual(new_array, reordered_array)

    def test_array_crescent_order_organizer_receives_array_with_non_integer_item_throw_error(self):
        fake_array = [0, 2, 1, 'non-integer value']

        with self.assertRaises(ValueError):
            self.stub_array_manager.array_order_organizer(fake_array)
```

Notice on the example that the same unit of work can generate several unit tests and not only the normal workflow is tested. Check how the method deals with errors and other situations is also something important to ensure the function's right execution.

Nonetheless, the name given to the unit tests is also important. In contrast to some production code development best practices, unit tests should have a really explicit name easy to identify the method that is being tested (e.g. 'array\_crescent\_order\_organizer'), the scenario of the test (e.g. 'receives\_disordered\_array'), and the expected outcome (e.g. 'reorders\_it').

#### Test doubles

As we have seen in the example above, software tests usually have to make use of test doubles to achieve the desired degree of isolation and to specifically test what they are designed for. Test doubles are any replacement of production objects made inside a test.

There are several definitions and sub-classes of test doubles. One of the most simple and practical definitions is given by Roy Osherove in ([The art of unit testing, 2013](#)) where these test fake objects are divided into stubs and mocks:

- Stubs: Controllable object that replaces an existing dependency in the system.
- Mocks: Also a replacement for an existing dependency, but the test asserts against this fake object.

One of the most difficult steps when developing software tests, specially unit tests, is to create and inject the test doubles. The tests frameworks are a great ally that tends to have features to help in this stage. In the case of the unittest Python library, the replacement of other methods that are not under test using the 'mock' decorator is one of the most useful tools, which are exemplified below.

```
import unittest
from unittest import mock

from src.data_preparation import DataPreparationManager

class DataPreparationTests(unittest.TestCase):

    # Declare class test doubles
    raw_data = []
    data_prep_manager = DataPreparationManager(raw_data)

    data_frame_missing_only_drop_columns_preparation = # Build a fake data frame that has already passed through all
                                                       # data preparation steps except drop useless columns

    # Unit test that checks if the correct methods are called when the function "executed_pipeline()" is called.
    # Since the methods are being mocked, their call inside the pipeline will not execute the production code, but it
    # will be possible to verify if they were called. In that way, the "executed_pipeline()" method can be tested isolated
    @mock.patch("src.data_preparation.DataPreparationManager._temporary_build")
    @mock.patch("src.data_preparation.DataPreparationManager._select_machining_processes")
    @mock.patch("src.data_preparation.DataPreparationManager._feature_engineering")
    @mock.patch("src.data_preparation.DataPreparationManager._drop_useless_columns")
    def test_execute_pipeline_standard_situation_all_necessary_methods_are_called(self, mock_drop, mock_feature,
                                          mock_select, mock_temp):

        self.data_prep_manager.execute_pipeline()
        self.assertTrue(mock_drop.called)
        self.assertTrue(mock_feature.called)
        self.assertTrue(mock_select.called)
        self.assertTrue(mock_temp.called)

    # In this unit test we want to test the drop useless columns method. However, this method is protected and can only
    # be called using the execute pipeline method. Then, we will use the mock.patch decorator again to only execute the
    # production code of the method we want to test and to set one of the mocked methods to return a fake object that
    # will be the input of the method under test.
    @mock.patch("src.data_preparation.DataPreparationManager._temporary_build")
    @mock.patch("src.data_preparation.DataPreparationManager._select_machining_processes")
    @mock.patch("src.data_preparation.DataPreparationManager._feature_engineering")
    def test_drop_useless_columns_receive_prepared_dataframe_drop_all_useless_columns(self, stub_feature, stub_select,
                                          stub_temp):

        # Set return value of feature engineering method that will be the input of the drop useless columns method
        # inside the pipeline execution.
        stub_feature.return_value = self.data_frame_missing_only_drop_columns_preparation

        columns_to_be_dropped = ["Useless_1", "Useless_2"]

        data = self.data_prep_manager.execute_pipeline()

        self.assertNotIn("Useless_1", data.columns)
        self.assertNotIn("Useless_2", data.columns)
```

## Integration tests

Another layer of our test pyramid is the integration tests. This type of test has many common points with the unit tests, it is written similarly, and it generally uses the same test framework. However, its scope is not limited to test a specific method isolated, integration tests can be used to check the behaviour of the program when performing a wider action such as the integration between different modules and the use of real dependencies.

Therefore, integration tests can break some important unit test principles. Since it relies on multiple real dependencies, this type of test will not always be consistent and can also take a considerable amount of time to be executed. Nevertheless, they represent an important set of software tests that can cover complex interactions



that are not verified on the unit tests, with the decision to run them on the main CI cycle at each merge request or not being from the development team.

To better exemplify its usage, a possible integration test inside a machine learning project would be to feed the data package with a stub in a format of raw data and execute the whole program's steps until acquiring the prediction for these values using the real model and test if it was according to the expectations. This would be not only testing the interaction between different modules but also the serialized ML model.

## Contract tests

---

With the program's units of work and package integration covered by software tests, it is also relevant to test the usage of outside services on the project. Considering the example given in the last section, that integration test is already verifying a considerable part of the machine learning pipeline, however, it is still being fed with fake data object built to have the same format as a raw data coming from the data source.

Supposing that our data source is a standard local database, some of its characteristics can change and prevent the execution of our program, such as changes on the tables design, communication protocols, or login credentials. If one of these things happens, it will not be signaled by any unit or integration test, since they use test doubles to represent the raw data.

Contract tests are designed to cover these scenarios, making a real request to the external service and comparing its response with the fake object that represents what the program is expecting to get. In the previously described situation, the test could use the production code to make a request to the database to get raw data from the real environment, compare it with the test doubles used in other tests, and certify that the communication between the program and the database still work as planned.

This type of test can be designed for several third-party software and microservices being used on the project.

## E2E tests

---

Creating different unit, integration, and contract tests our program will be widely verified and will result in a robust software easy to maintain. However, some integrations can still be out of the tests' range due to the use of test doubles and their isolation from other parts of the application.

Thus, it is important to perform some E2E tests where the whole program stack is executed and verified. Since this type of test is harder to maintain and slower to run, it is recommended to not rely the project development on them, but perform them manually when a new version is going to production.

## APPENDIX G – Applications' README files

Figure 31 – README file of the shared database application for the use case validation

### Shared-DB production application instructions

This project simulates a simplified production environment to test a shared database deployable application using the outcomes of the data science exploration for the [CNC Mill Tool Wear problem](#).

To simulate the environment and run the application follow the steps below:

1. Change the application's image to the desired version.
  1. Go into the GitLab repository in the "Packages and Registry/Container Registry" section and copy the image tag of the app version you want to use.
  2. Go into the [docker-compose.yaml](#) and change the image name in the "main" section.
2. Login into the project's container registry using the password in a `local` text file: `bash sudo cat stdin.txt | sudo docker login registry.gitlab.cc-asp.fraunhofer.de/gustavosalbino/cnc-mill-tool-wear-batch-simulation --username YOUR-USERNAME --password-stdin`
3. Change the [config\\_template.json](#) file fields to set the environment's configuration.

Execution parameters: A set of parameters that configure the application execution.

"serialization\_type": The type of serialization used in the model file. Currently only accepts "pickle".

"prediction\_models\_path": The path to the model file.

"last\_sample\_id": The database id of the sample from which the predictions should start. 0 is the standard.

Database parameters: A set of parameters that configure the database used to simulate the environment. Its fields do not need to be changed if no modification was made in the MySQL database.

"db\_type": Database type string used by SQLAlchemy library.

"name": Database name.

"address": Database address in the environment.

"user": Database user.

"password": Database password.

4. Run the database by executing the command `docker-compose up mysql` inside the [environment-execution](#) folder.
5. After the database initializes properly, new experiments can be added into the Experiments table. A bash script that adds all the experiments described in the [CNC Mill Tool Wear dataset](#) was already developed and can be executed with the command `bash environment-execution/scripts/execute_adds_csv.sh`.
6. Then, the batch application that will make a prediction for each experiment line in the database can be started in another terminal with the command `docker-compose up main`, which should give new values in the Predictions table after its execution.
7. The system can be shut down with the command `docker-compose down`.

Source: Personal Archive

Figure 32 – README file of the streaming application for the use case validation

## Streaming production application instructions

This project simulates a simplified production environment to test a streaming deployable application using the outcomes of the data science exploration for the [CNC Mill Tool Wear problem](#).

To simulate the environment and run the application follow the steps below:

1. Change the streams image to the desired version.
  1. Go into the GitLab repository in the "Packages and Registry/Container Registry" section and copy the image tag of the app version you want to use. Seek for images ending with "data-preparation" and "predictions-maker".
  2. Go into the `docker-compose.yml` and change the image name in the "data-stream" and "predictions-stream" sections.
2. Login into the project's container registry using the password in a `local` text file: `bash sudo cat stdin.txt | sudo docker login registry.gitlab.cc-asp.fraunhofer.de/gustavosalbino/cnc-mill-tool-wear-stream-simulation --username YOUR_USERNAME --password-stdin`
3. Change the `data_stream_config.json`, `lr_stream_config.json`, and `rf_stream_config.json` files' fields to set the environment's configuration.

Execution parameters: A set of parameters that configure the application execution.

"serialization\_type": The type of serialization used in the model file. Currently only accepts "pickle".

"prediction\_models\_path": The path to the model file.

"kafka\_address": Kafka address in the environment.

"predictions\_buffer": Buffer of how many predictions is necessary to make before publishing them in the database.

Database parameters: A set of parameters that configure the database used to simulate the environment. Its fields do not need to be changed if no modification was made in the MySQL database.

"db\_type": Database type string used by SQLAlchemy library.

"name": Database name.

"address": Database address in the environment.

"user": Database user.

"password": Database password.

4. Run the environment related images by executing the command `docker-compose up mysql zookeeper kafka control-center connect` inside the `run-stream-arc` folder.
5. After sufficient time, open another terminal and make sure that all images are up and running properly. If some of them have failed, start them manually one by one e.g. `docker-compose up mysql`.
6. Establish the connection between the MySQL database and the Kafka running the following command: `bash curl -i -X POST -H "Accept:application/json" -H "Content-Type:application/json" localhost:8083/connectors/ -d '{ "name": "manufacturing-connector", "config": { "connector.class": "io.debezium.connector.mysql.MySqlConnector", "tasks.max": "1", "database.hostname": "mysql", "database.port": "3306", "database.user": "root", "database.password": "debezium", "database.server.id": "184054", "database.server.name": "dbserver1", "database.include.list": "manufacturing", "database.history.kafka.bootstrap.servers": "kafka:9092", "database.history.kafka.topic": "dbhistory.manufacturing" } }'`
7. Run the stream images by executing the command `docker-compose up data-stream random-forest-predictions-stream logistic-regression-predictions-stream` inside the `run-stream-arc` folder.
8. At this point the system should be already running. If you add any new experiment in the database it will be first processed by the data pipeline stream (faster) and then sent to the prediction streams (slower). A bash script that adds all the experiments described in the [CNC Mill Tool Wear dataset](#) was already developed can be executed with `bash run-stream-arc/scripts/execute_adds_csv.sh`.
9. The system can be shut down with the command `docker-compose down`.

Source: Personal Archive