UNIVERSIDADE D
COIMBRA

Ana Sofia da Silva Brito de Oliveira

DEVELOPMENT OF AN ORCHESTRATION ENGINE FOR THE DATASCIENCE4NP PLATFORM

Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering, advised by Filipe João Boavida Mendonça Machado de Araújo and Professor Rui Pedro Pinto de Carvalho e Paiva. and presented to Faculty of Sciences and Technology / Department of Informatics Engineering.

July 2020

This page is intentionally left blank.

Abstract

The demand for qualified people capable of extracting value from the ever-increasing volume of data is growing. More data scientists need to be trained, but training can be a time-consuming task due to the diversity of disciplines it involves. A more gradual learning curve can be achieved by abstracting programming languages from the scientists' path. The ultimate goal of the Data Science for Non-Programmers project (DataScience4NP) is to implement data science practices rightfully without requiring programming skills, thus enabling non-programmers to be part of the data science workforce.

The DataScience4NP is a platform focused on machine learning (ML) workflows and is available through a Web User Interface. It follows a microservices architecture with multiple Docker containerized services running ML algorithms orchestrated in a Kubernetes cluster. These technologies provide great flexibility in deploying and managing applications, either on-premises or on the cloud. Nevertheless, we still need an orchestration solution to manage the execution of workflows (a technology to orchestrate the ML tasks fed to the ML microservices). Netflix Conductor was the technology initially adopted for this purpose, but, because it cannot support workflows with hundreds of tasks (such as workflows involving cross-validation with repetitions), Conductor turned out to be an unsuitable solution.

In this dissertation, we adopt a new approach to orchestrating ML workflows using Amazon Web Services (AWS) Step Functions with the final intention of executing more complex workflows.

Keyworks

Orchestration, Microservices, Cloud Computing, Amazon Web Services, Machine Learning

This page is intentionally left blank.

Resumo

Está a crescer a procura por pessoas qualificadas que sejam capazes de extrair valor do grande volume de dados gerados atualmente. Existe a necessidade de treinar novos cientistas de dados, no entanto este pode ser um processo lento e dispendioso devido às várias áreas interdisciplinares que a Ciência de Dados envolve. O tempo de aprendizagem pode ser reduzido se abstrairmos os cientistas das linguagens de programação. O objetivo do projeto Data Science for Non-Programmers (DataScience4NP) é implementar práticas usadas em Data Science de forma correta, sem serem necessários conhecimentos de programação.

A aplicação foca-se em workflows de Machine Learning e está disponível através de uma interface web. Segue uma arquitetura de microsserviços conteinerizados com Docker e orquestrados num cluster de Kubernetes. Estas tecnologias providenciam uma alto nível de flexibilidade na gestão e no deployment de aplicações na *Cloud*. No entanto, era ainda necessária uma solução para gerenciar a execução dos workflows de Machine Learning e assim orquestrar as tarefas de ML nos microsserviços. O Netflix Conductor foi a tecnologia inicialmente adoptada para esse fim mas que acabou por se revelar numa solução inadequada devido às suas limitações para executar workflows com centenas de tarefas, como por exemplo workflows que envolvam validação cruzada com repetições.

Nesta dissertação, é adoptada uma nova abordagem para a orquestração dos workflows de ML usando Amazon Web Services (AWS) Step Functions para que seja possível executar workflows mais complexos.

Palavras-Chave

Orquestração, Microsserviços, Computação na Nuvem, Amazon Web Services, Aprendizagem Computacional

This page is intentionally left blank.

Acknowledgements

Firstly, I thank my supervisors, Professor Filipe Araújo and Professor Rui Pedro Paiva, for the given opportunity and their consistent guidance and feedback throughout this project.

I would also like to thank Artur Pedroso for taking the time to help me step up the project initially and the Ph.D. Student André Bento for giving me the resources to deploy the DataScience4NP platform during the development of this thesis and for his willingness to help me whenever I encounter a problem. Additionally, I express my appreciation to Infraestrutura Nacional de Computação Distribuída (INCD) for the computational and storage resources of high capacity and performance.

I could not forget my friends, who always were by my side. Thank you for the memorable good times and for always uplift me in the not so good ones. To Jerónimo, for his love, his unbelievable understanding and for knowing how to handle my lousy temper like no one.

Finally, I would like to express my endless gratitude to my family for their unconditional support, not only during the development of this thesis but through all my whole life, and for always motivating me to do more and better. All of my accomplishments would not be possible without them.

This page is intentionally left blank.

Contents

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Glossary

AI Artificial Intelligence. 12

AWS Amazon Web Services. i, iii, x, 2, 3, 21, 22, 25, 29, 47, 50, 52, 54–57, 59, 60, 65, 81, 101, 114

CLI Command-Line Interface. 21

DEI Departamento de Engenharia Informática. 3

EC2 Elastic Compute Cloud. 21

ECS Elastic Container Service. 21

GCP Google Cloud Platform. 3

GKE Google Kubernetes Engine. 81, 100

GUI Graphical User Interface. 11, 13, 15, 22, 102, 108

HA High Availability. 10

IaaS Infrastructure as a Service. 8

INCD Infraestrutura Nacional de Computação Distribuída. 3, 44, 62, 85

IT Information Technology. 8

JSON JavaScript Object Notation. 18, 47, 53, 54, 57, 108

JVM Java Virtual Machine. 19

ML Machine Learning. x, 1, 2, 12–15, 17, 29, 50, 52, 54–56, 101, 102

PaaS Platform as a Service. 8

SDK Software Development Kit. 52

SOA Service-Oriented Architecture. 9

SVM Support vector machine. 54

This page is intentionally left blank.

Chapter 1

Introduction

This dissertation is a further work of the project Data Science for Non-Programmers (DataScience4NP) previously carried by Bruno Leonel André Lopes and Artur Jorge de Carvalho Pedroso, two former MSc students at the Department of Informatics from the University of Coimbra. The present chapter presents the motivation and scope for the project, followed by the essential objectives and chosen approaches.

1.1 Motivation and Scope

Data science is an interdisciplinary field that addresses the on-going needs and challenges emerging with the Big Data era. These challenges demand qualified people capable of extracting value and creating useful insights from the ever-increasing volume of data. Data can be generated in the most various ways. Examples of it are ad clicks, on social media, streaming content, and data pouring out of any of our connected devices such as smartphones, smartwatches, or smart TVs. According to Data Never Sleeps 7.0 [1], an infographic annually released by Domo, on each minute of every day 188 000 000 emails are sent, Google conducts 4 497 420 searches and Netflix users stream 694 444 hours of video. Education in the Data Science field is essential to expand the data science workforce successfully. However, training new data scientists can be a slow process due to the diversity of disciplines it involves.

The Data Science for Non-Programmers (DataScience4NP) project has emerged from the need to provide new data scientists with a better, easier, and less time-consuming approach to learn Data Science methods and processes. The platform is only focused on Machine Learning workflows, and it tackles the steep learning curve problem by abstracting programming languages from the scientists' path.

The application of Machine Learning (ML) entails the execution of different steps that comprise various tasks. These steps include data preprocessing, where operations such as normalization or feature selection are applied to make data more amenable to be used in the next phase, where a model is trained. After training a model, it may be tested to evaluate its predictive capacities (the model performance). In the end, tasks might be changed to attempt the achievement of better results [2].

DataScience4NP is intended for people that have a basic understanding of Machine Learning concepts/algorithms but who have no programming experience, and as already mentioned, removes the barrier between users' interest in ML and their lack of programming skills. The platform allows users to build machine learning workflows using a graphical

interface. Workflows are sequences of tasks from the different stages in ML model training, namely preprocessing, learning, and evaluation phases. The graphical interface enables the execution of ML workflows in a sequential fashion way. This approach reduces the effort imposed on users during the creation of workflows and guides them during the ML process. Users are presented with an established sequence of options and can build their models by choosing the most suitable tasks for their purpose. The sequential ML workflows composed of different ML tasks are then translated to an orchestrated representation enforcing good ML practices.

The system was built as a cloud platform which enables the remote execution of compute-intensive ML experiments in cloud provided infrastructures, without imposing efforts in the users' machines. DataScience4NP follows a microservices architecture where each ML task has a microservice responsible for its execution. This architecture allows great flexibility both in the insertion and scaling of ML functionalities to be used in ML workflows. The logical workflows that users can create in the graphical interface are intended to abstract them from the complexity of the ML process. After the creation of logical workflows, there is a need to translate them into system workflows to be understood by the orchestrator. The orchestrator is responsible for the orchestration of ML tasks to be processed in ML microservices.

To manage the execution of the workflows, Netflix Conductor was adopted as an orchestration technology. Despite being initially evaluated as fit technology, Conductor presented some drawbacks in the execution of more complex workflows, in particular, workflows that included nested cross-validation with repetitions that can easily reach thousands of tasks. This limitation imposed by Netflix Conductor motivated its replacement for a new orchestration technology. Netflix Conductor is explored in more detail in section 2.3.1.

The primary objective of this dissertation is to solve the orchestration problem and enable the DataScience4NP platform to be more efficient in the execution of workflows. For this purpose, the AWS Step Functions will be the orchestration solution adopted.

1.2 Objectives and Approach

DataSicence4NP follows a microservices architecture, with multiple Docker containerized services running the ML algorithms orchestrated in a Kubernetes cluster. This approach will remain the same as these technologies provide great flexibility in deploying and managing applications, either on-premises or on the cloud.

Nevertheless, we need a new orchestration solution to manage the execution of workflows adequately. Upon this necessity, and after considering several alternatives for the orchestration, we believe that porting our solution to AWS and replace Conductor by AWS Step Functions is our best option. Besides, AWS can simplify platform management. It also has several services that can potentially be explored and introduced in our system.

This shift to AWS Step Functions has the ultimate goal of providing users the ability to execute complex workflows by placing their potentially long experiments to run in the cloud. To do so, we need to build a better orchestration solution capable of dealing with sizable workflows.

Every computational resource has a price. AWS is no exception: the computational resources, services, and products provided by it have a price. We need to support the costs of exploring the platform by billing the users for what they spend. Thus, we need to create a billing system capable of account the resources used by each user and to charge accordingly.

The outcome of the project will be a prototype of an online platform available through a Web User Interface that precludes the need for any programming skills, enforces good machine learning practices, and can deal with sizable workflows.

1.3 Challenges

Taking over an existing software project has its challenges, and it can quickly become a cumbersome task. Even more, if the project involves a lot of different technologies (microservices architecture, Kubernetes), and being new to those technologies, the challenge was more significant.

The first goal was to step up a development environment with a running version of the project, which was not possible until the mid-time of this thesis: the free tiers offered by cloud providers such as AWS and GCP do not provide enough resources to deploy the DataScience4NP, and DEI Cloud was not functioning correctly.

Alongside the various attempts to deploy the platform, a study of the code was being conducted. The documentation (previous theses) helped to understand the big picture and how the platform works.

Besides the deployment of the platform, the greatest difficulty was to comprehend the logic behind the Workflows Service (orchestrator, responsible for the translation of workflows). Furthermore, all the other services that contact with AWS were also modified.

After having the platform deployed in INCD, the development started. Working with AWS Step Functions was tricky because some of the limitations imposed by the AWS were not expected.

1.4 Contributions

This thesis is a continuation of the project Data Science for Non-Programmers (DataScience4NP) previously developed by Artur Pedroso e Bruno Lopes. The contributions to this project are the following:

- An orchestrator compatible with AWS capable of dividing significant complex workflows into smaller ones.
- Improvements in ML services and consequently in the performance of Machine Learning tasks (workers). It improved the capacity of the system in such a way that it is now possible to handle multiple users who can also be multiple various workflows at the same time with reasonable response times.
- A simple billing service as a proof of concept to bill users for their computational usage.

Furthermore, some improvement made in the code documentation was made, and even though it is not one of the perceptible contributions, it can result in a more gentle learning curve for future developers.

1.5 Document Organization

The remainder of this document is organized as follows:

Chapter 2 elucidates the reader about some fundamental concepts in this dissertation, it overviews systems similar to DataScience4NP, and it reviews current solutions for orchestration.

The design of the system is addressed in Chapter 3, which states the requirements for the DataScience4NP system and its quality attributes.

Chapter 4 presents the original architecture of the prototype and the current architecture.

Chapter 5 details how the service that acts as the orchestrator performs its work and how Step Functions are used.

Testing phase and all of its results are documented in Chapter 6.

Chapter 7 explains the strategy used in the development of this dissertation, presents a risk assessment and the work plan for both the first and second semesters.

Finally, it can be found in Chapter 8 a reflection over the work performed during the completion of this thesis and its outcome.

This page is intentionally left blank.

Chapter 2

State of the Art

The present chapter intends to expose and review existing systems that provide machine learning functionalities similar to the ones that the DS4NP platform is seeking to deliver. Existing orchestration engines and cloud solutions are also examined and presented, given our interest in improving the DS4NP orchestrator as well as its overall performance.

2.1 Concepts

The following subsections clarify some basic concepts of this thesis.

2.1.1 Cloud Computing

Cloud computing consists of different resources and services provided through the Internet. The main goals of cloud computing are to solve large-scale computation problems and to make better use of the distributed resources [3].

Web-based applications are not the same as cloud applications. Some providers wrongly claim to offer cloud services. To be considered a cloud implementation, an application must exhibit some characteristics [4].

There are five key cloud characteristics:

- **On-Demand Self-Service**: automated provision of cloud resources/services on-demand, without support from administrators or support staff.
- **Broad Network Access**: provision of cloud resources through network connections (usually some type of Internet connection) requiring only lightweight clients or no client at all. It should be possible to access resources from various devices (laptops, desktops, smartphones, tablets, and other options).
- **Resource Pooling**: it assumes that costumers do not always need all resources to be available for them. Resource polling makes those resources not to sit idle and place them available to be used by other costumers. It allows flexibility on the provider side, enables more costumers to be served, and is usually achieved using virtualization.
- **Rapid Elasticity**: the ability of cloud environments to quickly grow to meet user demand. Even if cloud providers already have computer resources to expand their capacity, they are only used until needed.

- **Measured Service**: the ability to measure usage based on some metrics, for example, time used and type of resources used. Measured service is what enables the "pay as you go" feature.

It is possible to divide services provided by cloud computing into three categories:

- **Infrastructure as a Service (IaaS)**: the lowest level of cloud offerings that delivers computer resources such as the capacity of storage, processing, and network. IaaS is an advantageous option due to the pay per use basis, its security, and reliability.
- **Platform as a Service (PaaS)**: it offers scalable runtime environments on-demand capable of hosting the execution of applications.
- **Software as a Service (SaaS)**: the highest level of cloud offerings that provides services directly consumable by end-users. It replaces applications running on a PC, and it works based on a pay as you go policy.

The environments for deploying and access cloud computing can also be categorized into three types:

- **Public cloud**: environments where cloud resources and all the IT infrastructure (hardware, software, and other supporting infrastructures) are owned and operated by a third-party service provider and are delivered to the end-user over the Internet. Public providers use bare-metal IT infrastructure that can be abstracted and sold as IaaS, or developed into a platform sold as a PaaS. Public cloud has some advantages such as low costs (users do not need to purchase software and hardware and only pay for services they use) and maintenance being provided by service providers.
- **Private cloud**: environments exclusively dedicated to a single end-user/business or organization where infrastructure and facilities are maintained on a private network. This environment is commonly used by government agencies, financial institutions, and other organizations with business-critical operations since it provides high security (resources are not shared with others), privacy, and regulatory concerns.
- **Hybrid cloud**: environments that combine on-premises infrastructure, or private clouds, with public clouds. Its main advantage is flexibility: hybrid cloud takes advantage of the privacy in private clouds and from additional resources in the public cloud when needed.

Cloud computing, big data, and data science can be seen as concepts that are somehow related. **Data Science** is a field focused on analyzing and manipulating data with the ultimate goal of extracting value from it and gain useful insights. We have observed in the past years a massive increase in the amount of data generated, mainly due to technological advances. To this tremendous volume of generated data, we call **Big Data**. According to Gartner, "Big data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation" [5]. One innovative way of dealing with big data is by adopting cloud-based technologies. Cloud computing has changed the way computer infrastructure is abstracted and used, and is continuously becoming a robust architecture to perform large-scale and complex computing [6].

Undoubtedly, cloud computing can be beneficial in the context of the DataScience4NP. Our platform was envisioned to be deployed using an IaaS solution, and according to Magic Quadrant for Cloud Infrastructure as a Service, the Leaders in cloud IaaS are AWS, Microsoft, and Google [7].

2.1.2 Microservices

The DS4NP Platform follows a microservices architecture. Microservices are small, autonomous services that work together [8]. This architectural style emerged as a specific approach for Service-Oriented Architecture (SOA), and it holds many and varied benefits:

- Technology Heterogeneity: microservices systems are composed of small and decoupled services that permit the use of different technologies on each one.
- Resilience: when failures happen, it is possible to isolate the problem as they occur in isolated services, but this does not mean that other services will never be affected. However, the system may carry on working as the opposite of a monolithic service, where everything stops working if a service fails.
- Scaling: it is possible to scale only services that need scaling, instead of scaling a monolithic system.
- Ease of Deployment: microservices allow us to make individual changes to services and deploy them independently of the rest of the system.
- Optimizing for Replaceability: replacing services by an improved version of it is easier to manage, of it requires less effort and is less risky.

Another concept tightly associated with cloud environments and microservices is **containerization**. A set of techniques for integrating the software development process with the deployment and operations called DevOps can be the key to the overall success of cloud-based software solutions [9].

A container is a stand-alone unit of software that packages up code and all its dependencies. Code portability is a useful feature of containers: packing software along with its dependencies into a single image provides the freedom of "develop once, deploy everywhere". Containers are also helpful by providing isolation and guaranteeing the modularity of microservices. The technologies adopted for containerization in the DS4NP platform were Docker and Kubernetes.

2.2 Similar Systems

The primary goal of the DS4NP is to provide an easy way of constructing machine learning workflows without requiring any programming skills. Doing so while enforcing the best practices used in Data Science is also a significant concern that we try to address. The need for writing code and the ease of constructing an ML pipeline are some of the aspects we tried to observe in the systems presented in this section.

2.2.1 H2O

H2O.ai[1] is an open-source software company that provides various products and solutions being one a machine learning platform called H2O. This platform offers algorithms for supervised and unsupervised learning, it allows the use of some programming languages like R and Python to build models, and it supports massive datasets using in-memory

[1] https://www.h2o.ai/

processing with fast serialization between nodes and clusters [10]. It has a new key functionality named AutoML, whose purpose is to automate the machine learning workflow by minimizing the amount of human effort. AutoML allows the performance of a large number of modeling-related tasks in a simple wrapper function. H2O also provides a user interface called H2O Flow, without requiring any programming experience to run it since it combines command-line computing with a point-and-click type of graphical interface.

In Figure 2.1 we can see an example of one of the interfaces, in this case, the outline to build a model where it is possible to select an algorithm from the drop-down menu.

Whenever someone starts a new project, an Assistance outline with possible outlines is prompted. H2O has an Admin tab where it is possible to choose the option to consult a few features such as cluster status and perform network tests. H2O has good documentation[2] on how to use the graphical interface.

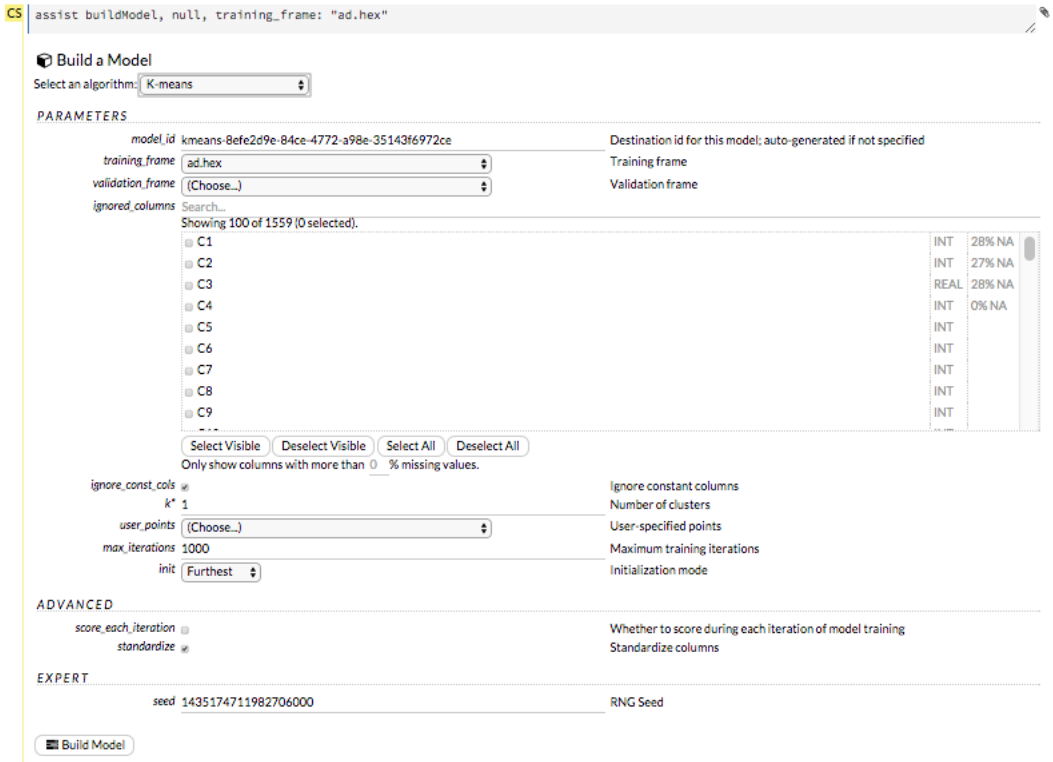

Figure 2.1 – H2O Flow: Build a Model.
Source: Using Flow - H2O's Web UI.

Gartner, the world's leading research and advisory company, creates annually "Magic Quadrant" reports for diverse kinds of IT products. In January of 2019, they published the *Magic Quadrant for Data Science and Machine Learning Platforms*, and H2O is present in the Visionary quadrant meaning that H2O is a recent vendor who has a strong vision and a reliable supporting roadmap with the potential to shape the market. Even though H2O.ai established a premier position in the automated ML domain, it still lacks some product capabilities. A shortage of features for data access and preparation is one example of it. H2O components are highly optimized and parallelized for CPU multicore and multinode configurations, but AutoML does not benefit from it since this solution seems far less powerful [11]. Another disadvantage of H2O is that the platform does not support high availability (HA) [12].

[2] http://docs.h2o.ai/h2o/latest-stable/h2o-docs/flow.html

Besides allowing users to train/test models in H2O Flow without requiring programming skills, there is no option to perform feature selection or data preprocessing before model creation.

H2O can integrate with Hadoop tools, thus supporting big data. It is also supported in different cloud environments, including Amazon AWS, Google Cloud Platform, and Microsoft Azure. In AWSMarketplace[3], for example, it is possible to consult the pricing, usage, and reviews for each H2O solution. AutoML is only in version 0.2.

2.2.2 RapidMiner

RapidMiner[4] is a code-optional platform for data science that offers a GUI with drag and drop functionalities and optional integration with various programming languages like R and Python. This platform also appears in the *Magic Quadrant for Data Science and Machine Learning Platforms*, as H2O, but in the Leaders quadrant meaning that RapidMiner has a strong presence in the data science and ML market and it provides outstanding service and support. To use RapidMiner, one has to install it locally but, through the GUI workflows can execute in a cluster. A free version is available but with several limitations: it is restricted to 10000 data rows and one logical processor. Just like H2O, RapidMiner has good documentation. It provides sample datasets and templates. The model of one of these samples can be seen in Figure 2.2. RapidMiner may not be a feasible solution to non-experts, but it remains a Leader among experienced data scientists by striking the right balance between ease of use and data science sophistication [11]. Some favorable aspects worth mentioning about RapidMiner are their attention on how to properly validate machine learning models[5], and the optional setup to ensure high availability.

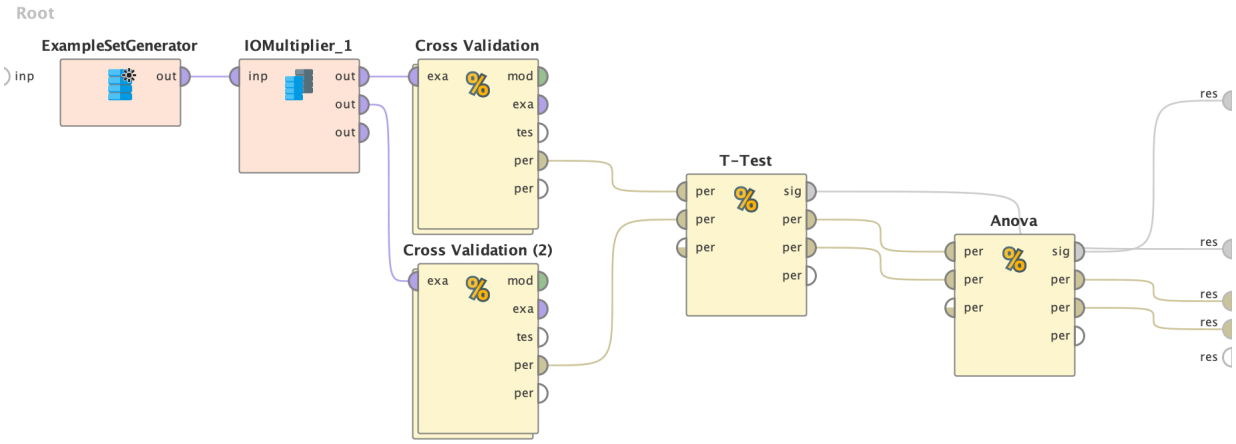

Figure 2.2 – RapidMiner Sample - SignificanceTest
Source: RapidMiner Samples

[3]https://aws.amazon.com/marketplace/seller-profile?id=55552124-d41b-4bad-90db-72d427682225
[4]https://rapidminer.com
[5]https://rapidminer.com/resource/correct-model-validation/

2.2.3 Azure Machine Learning Studio

Azure Machine Learning[6] is one of the Azure AI products from Microsoft. It can be used through a web studio where users can build, train, and deploy models with prebuilt building blocks that can be customized and connected in a drag and drop manner. Users can also upload data locally, using sample datasets or using the Reader Module, which allows the import of data from different data sources (for example, from Azure Table or Web URL via HTTP). It is possible to convert data formats and to visualize the chosen dataset at any time. Diverse ML tasks, such as preprocessing, feature selection, model creation, and model evaluation, can be selected to construct workflows, but there is also the possibility to introduce and execute R and Python scripts. The created models can be deployed as a web service and become accessible for future use with few clicks [13].

Azure AI has a Gallery where its community can share its analytics solutions, including solution templates, machine learning models, and predictive analytic experiments. In Figure 2.3, it is possible to observe a sample that demonstrates how to use Cross Validate Model with regression models.

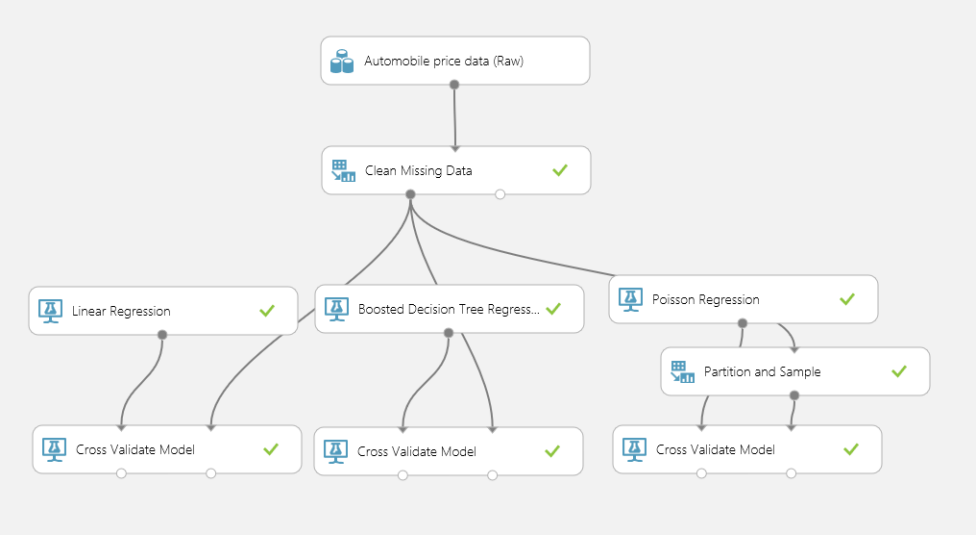

Figure 2.3 – Azure Experiment: Cross Validation for Regression.
Source: Sample 4: Cross Validation for Regression.

Azure Machine Learning applies the preprocessing and feature selection phases to the entire dataset before training/testing the final model with cross-validation, which is an incorrect practice and can provide overly-optimistic error estimates for the produced models. Besides not employing cross-validation correctly, this platform also does not support nested cross-validation, and the solution is proprietary.

2.2.4 Amazon SageMaker

SageMaker[7] is a service launched in 2017 by Amazon that covers an entire machine learning workflow by providing the capacity to build, train, and deploy ML models in the cloud. They answer to the lack of existing built-in tools for an entire machine learning workflow by providing all the necessary components in one tool to reduce effort and time. The platform offers a broad range of built-in ML algorithms, and it gives the possibility for developers

[6] https://studio.azureml.net
[7] https://aws.amazon.com/sagemaker/

to create their algorithms from scratch. SageMaker does not have a GUI where users could visually create their workflows, but it includes an integrated development environment (IDE) called Amazon SageMaker Studio, where users can perform all ML development steps by coding them. This platform is equipped with more features such as Amazon SageMaker Notebooks and Amazon SageMaker Experiments but is only suitable for experienced data scientists. Figure 2.4 represent the different components that compose this single toolset.

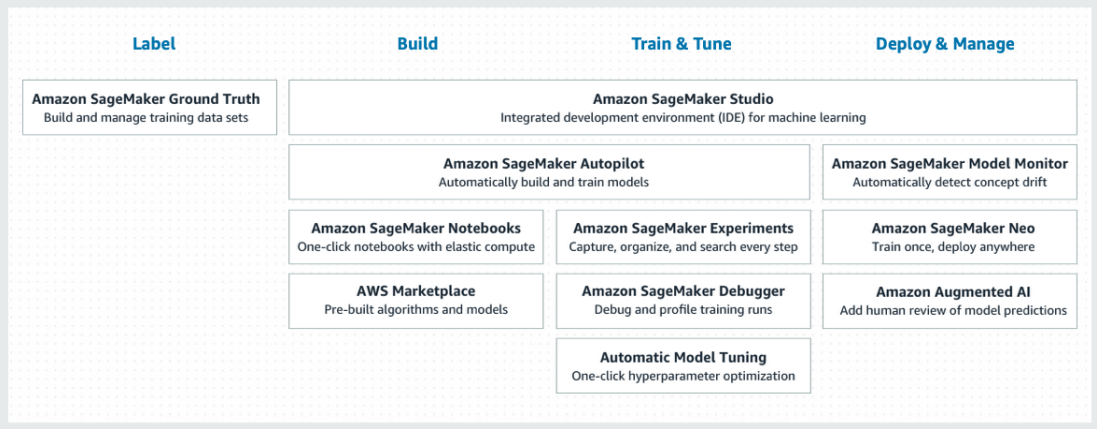

Figure 2.4 – SageMaker and AWS components.
Source: AWS SageMaker page.

2.2.5 ClowdFlows

ClowdFlows[8] is a cloud-based web-application with a graphical user interface that supports the construction and execution of data mining workflows [14]. It is possible to manage the workflows by editing, deleting, exporting, or making them public. Public workflows are available in the "Explore" page and can be accessed by everyone. The graphical interface works in a drag and drop manner, where workflow components (widgets) can be added and connected. In Figure 2.5, we can observe a workflow to build, display, and cross-validate a J48 tree. This workflow was provided by another user and accessed through the "Explore" page. The output for the execution of this workflow with the default input file is presented in Figure 2.6.

Very well-known ML libraries are integrated into ClowdFlows such as Weka, Orange, and scikit-learn. The processing is performed in a cloud of computing nodes. The platform can quickly scale horizontally and execute worker nodes in parallel on multiple machines [14]. The platform is also extensible since the widgets can be added in two ways: they can either be implemented as a python function and included in a ClowdFlows package or be imported as a web service in the GUI.

2.2.6 DataScience4NP

Before comparing the systems presented in this chapter, it makes sense to introduce the DataScience4NP.

DataScience4NP can be described as a cloud application accessible by its users on the Internet. Its architecture is based on microservices and can run on a Kubernetes cluster that has a minimum of 16GB of RAM. DataScience4NP is a platform that offers the execution

[8] http://clowdflows.org

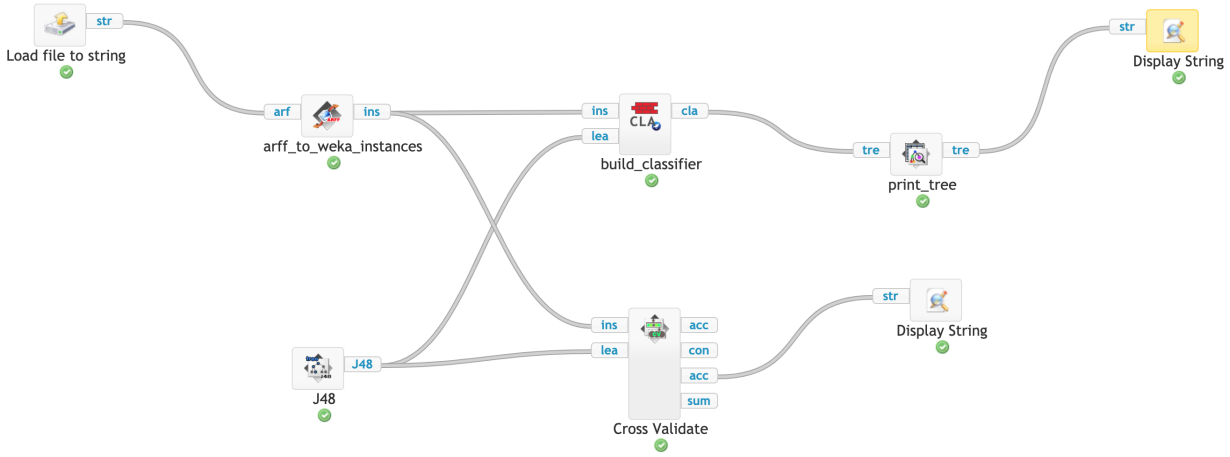

Figure 2.5 – Build and display a J48 tree and cross validate it.

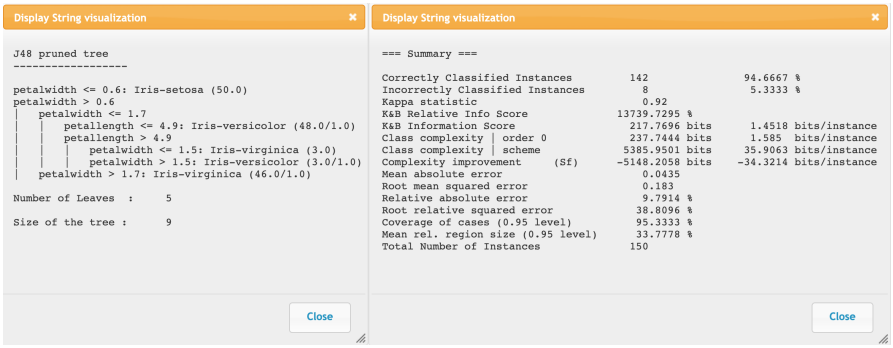

Figure 2.6 – Output of a J48 tree and its cross validation.

of ML services through a graphical interface. Contrary to most systems previously stated in this chapter (that have visual ways of building workflows like drag and drop), this platform allows users to construct workflows sequentially by choosing from a set of options that are presented to the user as they are building it. This form of creating workflows gives the user a sense of guidance during the process.

Our platform offers tasks that allow data insertion, preprocessing, feature selection, and feature projection, model creation, and model evaluation. It addresses cross-validation and nested cross-validation while ensuring that a proper model assessment and selection practices are being enforced.

In Figure 2.7, it is presented the DataSience4NP interface, where we can see a cross-validation task (a validation procedure task) and the tasks that users can choose after. The various options to create a ML workflow are presented to users in a sequence of steps. In each step, users are prompted with possible next steps that can be followed by the current one. DataScience4NP assures usability by providing multiple tasks in a sequence manner without scarifying flexibility since users have the freedom to choose from various options to build their workflows.

2.2.7 Comparison of Systems

To properly do an assessment and comparison of similar systems, we took into consideration several attributes, namely:

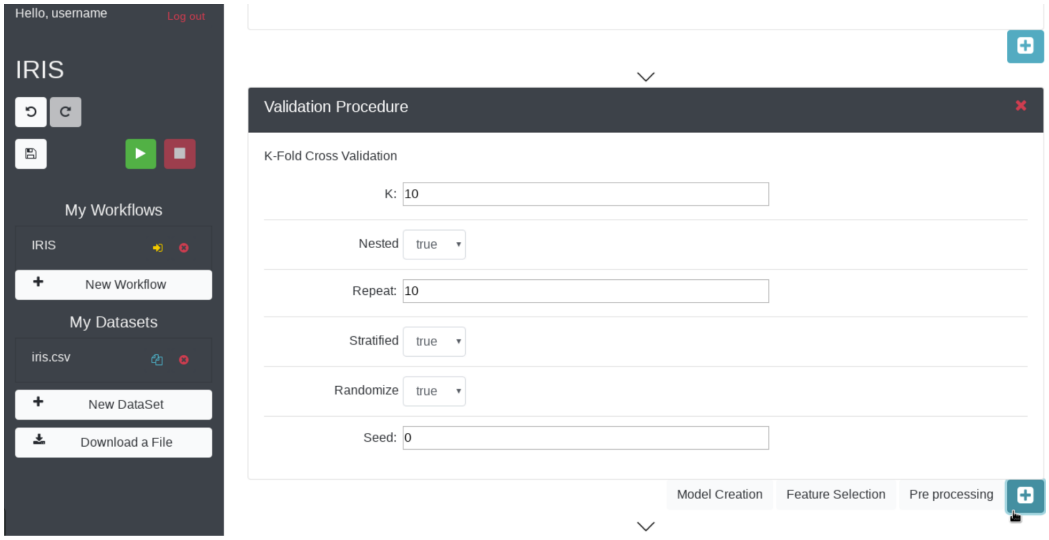

Figure 2.7 – DataScience4NP Interface.

- **Open Source**: indication if the solution is open source;
- **GUI**: indication if the solution has a Graphical User Interface;
- **Big Data**: indication if the solution has support and capability to handle large datasets;
- **Cloud**: indication if the solution is hosted in the cloud or if it has any functionalities being computed in the cloud;
- **Data Access**: indication if the solution supports different types of data and/or data sources;
- **ML practices**: indication if the solution implements good practices of Machine Learning especially when it comes to train/test models using cross-validation and nested cross-validation when preprocessing tasks are also included in the process;
- **Flexibility**: indication if the solution has a broad array of different tasks that the user can choose to implement or if it is a straightforward solution (on a scale from 1 to 5, where 1 is the minimum flexibility and 5 the maximum flexibility);
- **Usability**: GUI attractiveness, intuitiveness and complexity (on a scale from 1 to 5);
- **Pricing**: indication of tabulated values to use the solution.

These attributes will be somehow compared in the following of this section and can be found in Table 2.1 where all the systems aforementioned are present as well as the system to be.

The attributes unable to be determined were labelled with a question mark ("?"). The attributes that are not applicable to a solution were labelled with "N/A".

H2O is open-source, has a graphical interface, supports Big Data, and can be used in different cloud environments. It supports different file types (CSV, ORC, SVMLight, ARFF, XLS, XLSX, Avro, Parquet), and it can ingest data from various data sources (Local File System, Remote File, S3, HDFS, JDBC, Hive). However, there is no option to perform

Table 2.1 – Comparasion of machine learning platforms.

System	Open Source	GUI	Big Data	Cloud	Data Access	Flexibility	Usability	ML Practices	Pricing
H2O	x	x	x	x	x	3	3		
RapidMiner		x	x	x		5	3	x	$5,500 - $39000
Azure ML		x	x	x	x	4	3		Pay as you go
SageMaker			x	x	?	N/A	N/A	?	Pay as you go
ClowdFlows	x	x	x	x	?	4	2	?	
DataScience4NP AS-IS	x	x		x		4	4	x	
DataScience4NP TO-BE	x	x	x	x	x	4	4	x	N/A

features selection or data preprocessing before the model creation. Furthermore, H2O does not provide High-Availability.

RapidMiner reads data from different file types, databases, and even cloud storage. This platform ensures good practices of Machine Learning, but its free version is limited, and the annual plans vary from $5,500 to $39000 per user, per year.

Only expert data scientists can use SageMaker to leverage from the many services provided by Amazon because there is no graphical interface.

Platforms such as RapidMiner and ClowdFlows provide great flexibility by presenting users with a huge variety of tasks that can be used. However, this implies a decrease in its usability because connecting the different components to form a proper ML experiment can be quite hard. There is no such trade-off in DataScience4NP, which is one of our advantages over the other platforms.

Besides the ability to easily create ML workflows with a reasonable degree of flexibility, DataScience4NP has another significant advantage: the correct application of cross-validation processes, including preprocessing and feature selection tasks before train/test a model. DataScience4NP also addresses nested cross-validation.

2.3 Orchestration

Microservices cooperate in providing more complex and elaborated functionalities, which is the case of the DataScience4NP platform. In our solution, a machine learning workflow is composed of several tasks that have input/output dependencies. We call workflow to the specification of the tasks, their behavior, and coordination. Generally, each microservice performs a single task, and therefore it is crucial to assure communication and coordination between services. Orchestration and choreography are two approaches to establish this cooperation.

Choreography assumes no centralization and uses events and publish/subscribe mechanisms to establish collaboration. **Orchestration**, on the other hand, requires a central service to send requests to other services and monitor the process by receiving responses. [15]. While orchestration provides a centralized way to coordinate a workflow when there is synchronous processing, choreography imposes logic to be built into each service instead of having a conductor that controls the logic of each step that has to be performed in a workflow [16].

The way services collaborate, and the nature of communications (synchronous or asynchronous) are aspects to have in consideration when choosing between choreography or orchestration. There are two styles of collaboration: request/response or event-based. In request/response, a client initiates a request and waits for a reply. Synchronous communication, which is more associated with orchestration, is usually aligned with this type of collaboration (but can also be used asynchronously). Event-driven communications are asynchronous by nature, without blocking, and are associated with choreography.

Some defend that orchestration leads to service coupling and favors the adoption of choreography as it could be more flexible, amenable to change, and provide a higher degree of independence [15] [8]. However, there are good reasons to use orchestration rather than choreography.

In the DS4NP platform, machine learning workflows are created following synchronous patterns, which is why orchestration is a more suitable solution for us. Users can build their workflows by choosing from multiple options that comprise data insertion, prepro-

cessing, feature selection, model creation, and model evaluation. Among the many possible combinations to construct a workflow, there are situations in which a second task must always follow a first and only run if and when the first succeeds because of input/output dependencies. It also exists situations where multiple tasks need to be executed in parallel, and the combined result must be fed to another subsequent task. A centralized way of coordinating the services allows us to maintain fine-grained microservices since it would require a great deal of code to be written to choreograph the interaction of the microservices.

With orchestration, a conductor is responsible for the execution of workflows, while microservices are only concerned with their tasks. It allows greater reuse of existing microservices providing a more natural path for integrating new microservices.

Another significant reason for choosing orchestration over choreography is the visibility and traceability into the state of workflows. Even though choreography makes it easy to build decoupled systems (there is no central service that can fail and compromise the entire workflow execution as it happens with a conductor when we use orchestration), there is a significant risk in losing sight of larger-scale workflows. Thus visibility and traceability are key aspects we want to ensure in our platform. The high-level understanding of what is happening in the system also provides better monitoring and error handling strategies.

A disadvantage of choosing orchestration is that the conductor can be a single point of failure. If the conductor becomes unavailable, tasks will not be delegated, and the workflow will not execute.

2.3.1 Netflix Conductor

Conductor[9] is an orchestration engine developed by Netflix, open-sourced in 2016, written in Java, and designed to orchestrate microservices-based process flows at Netflix.

When considering the growing business needs and complexities of the company, developers became aware that it would be hard to scale with peer to peer task choreography. The publish/subscribe model proved to be problematic with complex flows for several reasons. Peer to peer choreography implies the insertion of process flows into applications' code, which makes it hard to adapt to changing needs due to the tight coupling around input/output, and it does not provide a systematic approach to track the status of a process [17].

Conductor allows the creation of workflows by connecting different types of tasks. These workflows are defined using JSON. Tasks are either System Tasks (such as fork, join, decision, sub-workflow), which are executed by the orchestration server or Worker Tasks that are implemented by microservices (workers) and run in a separate environment from Conductor. Just as workflows have a blueprint that defines the execution flow, tasks must also have a template to specify their behavior. These templates are known as Task Definition and allow us to attribute a unique name, to define timeouts, retry policies, inputs/outputs, etc.

Netflix Conductor offers a user interface to visualize, monitor, and troubleshoot workflow executions. Figure 2.8 shows a sample workflow with one fork, two worker tasks, and one join.

The API and storage layers are pluggable, enabling developers to use different queue and storage engines. Netflix is currently using Dynomite for storage, but it is possible to switch to another solution. Workers communicate with Conductor via the API layer to poll tasks for execution and to update their status. Figure 2.9 demonstrates a high level view of Conductor's architecture. At the core of the engine is the "Decider Service", a state machine

[9] https://netflix.github.io/conductor/

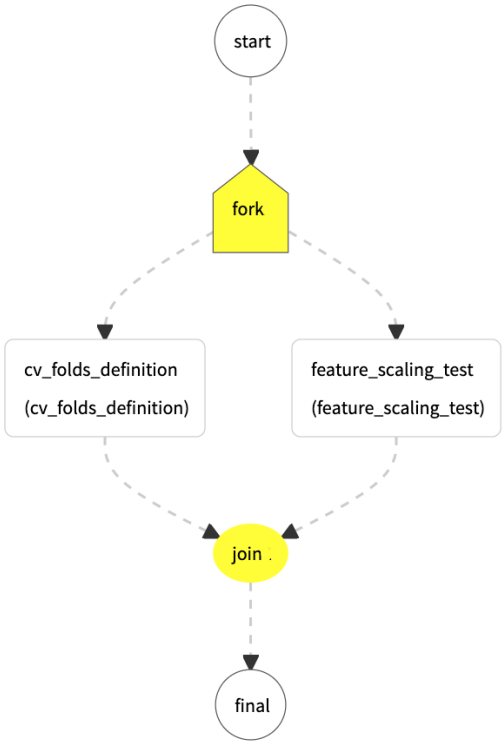

Figure 2.8 – A simple Conductor workflow.

service responsible for maintaining the workflow, i.e., it determines the state of the workflow centrally by comparing the progress of each worker against the workflow blueprint, making it possible to assess the state of the application for any moment in time. The Decider Service uses the Dyno Queues, a queue solution that runs on top of Dynomite, to manage scheduled tasks.

Some tests were performed to identify the bottleneck in Netflix Conductor that precludes the execution of complex workflows. We execute different workflows and observed the behavior of the Conductor server. At the same time, code inspections to the source code were made to understand how exactly does Conductor work. These tests are detailed in Appendix A.

2.3.2 Zeebe

Zeebe[10] is a free and source-available workflow engine for microservices orchestration. It is built to run on Kubernetes, and it is language, hardware, and cloud provider-agnostic. Zeebe provides visibility into the state of end-to-end workflows, workflow orchestration based on the current state of a workflow, and monitoring for timeouts or other process errors [18].

In workflows orchestrated by Zeebe, each task is usually carried out by a different microservices just as in DataScience4NP platform. It is an engine designed to solve the microservices orchestration problem on a considerable scale.

Zeebe uses a client/server model. The server, also called "broker" is the distributed workflow engine that runs on a JVM, and it orchestrates the workflow. The broker is responsible for

[10] https://zeebe.io

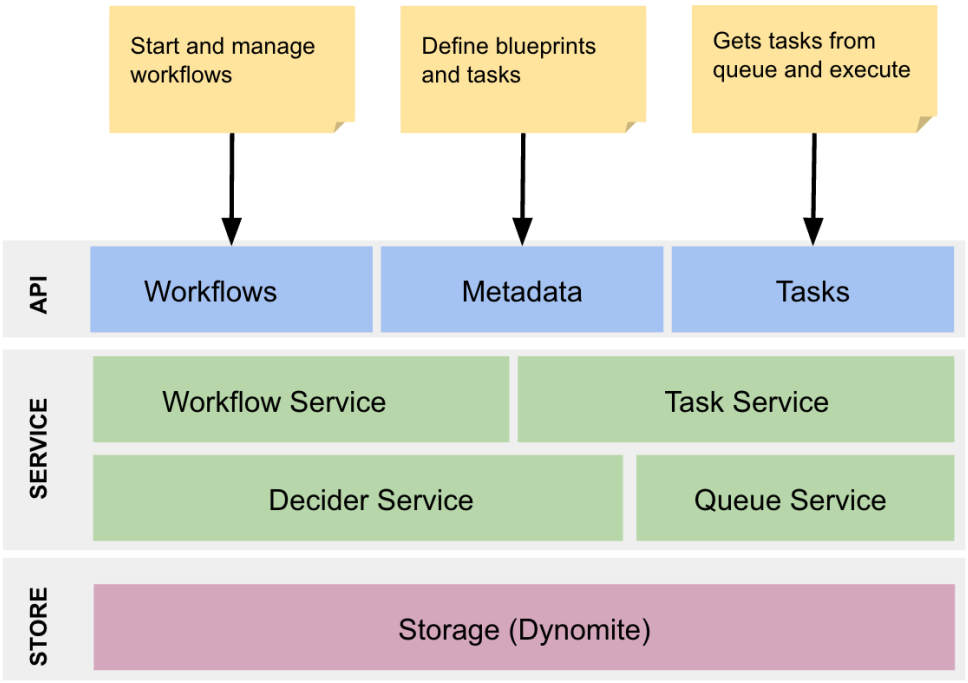

Figure 2.9 – Conductor's High Level Architecture.
Source: Conductor Documentation.

keeping the state of active workflow instances and for exposing workflow event streams to Zeebe clients through pub/sub. Clients (job workers) are libraries embed in applications (e.g., a microservice that executes your business logic) to connect to a Zeebe cluster. Workers poll for work from the worker and execute it. [19]

Zeebe captures all the changes to a workflow's state as events and stores it in an append-only log called "topic" that is written in the filesystem on the server where Zeebe is running. This is performed by the Zeebe broker.

Partitioning Brokers provides horizontal scalability, and replicating them ensures fault tolerance. It does not exist a single point of failure because, in a peer-to-peer network formed by brokers, brokers perform the same kind of tasks, and if one becomes unavailable, its responsibilities can be transparently reassigned in the network.

Zeebe implements orchestration as a state machine while maintaining the principles of loose coupling and independent deployability.

Even though it offers appropriated functionalities for our platform and it also provides horizontal scalability and fault tolerance, Zeebe is not be a better alternative to Netflix Conductor because it is still in "developer preview", i.e., it is not yet ready for production and is under substantial development.

2.3.3 Uber Cadence

Just like Netflix, Uber's platform consists of many microservices to support all the features that compose global services. Cadence[11], from Uber, has some use cases and can be seen as an orchestration engine to execute asynchronous long-running business logic in a scalable

[11] https://cadenceworkflow.io

and resilient way. It was open-sourced in 2017, it is written in GO, and it has Java and GO client libraries. It consists of a programming framework (or client library) and a managed service (or backend) [20].

Cadence has a Web UI[12], just like Netflix Conductor, to view workflows see scheduled, running, and completed activities (tasks), and explore and debug workflow executions. These two solutions have a few similarities regarding implementation details but a very different way to define a workflow. While Conductor defines workflows through JSON, Cadence defines workflows as code. A CLI that supports most of the API features is also available.

The deployment topology consists of a Cadence service that exposes its functionalities through a Thrift API, workflow and client workers, and external clients. Workflow Workers are external processes, implemented in any workflow definition language, that execute workflow code.

Cadence also uses Elasticsearch to list workflows like Conductor, and it has a persistent store. Apache Cassandra and MySQL are currently supported for storing purposes.

Scalability, durability, and high-availability are assured in Cadence.

2.3.4 AWS Step Functions

Step Functions[13] was first introduced in 2016 as one of the products offered by Amazon Web Services (AWS). It is an orchestration solution that allows the design and implementation of complex workflows. Similar to every AWS solution, Step Functions follows a pay-as-you-go approach: users only pay for what they execute.

The implementation of the orchestration logic is done by defining a JSON object using JSON-based Amazon States Language.

Tasks and State Machine are the two underlying concepts of Step Functions. A State Machine is the same as a workflow blueprint, it specifies the communication between states and how data is transmitted from state to state using the JSON. States can either be a Task, a Choice, a Fail, Succeed, Pass, Wait, or Parallel state. Each move from one state to the next one is called a state transition. Tasks perform the work declared in the state machine, which can be done by using an activity[14] (when a task is performed by a worker hosted on Amazon EC2 or Amazon ECS), an AWS Lambda function, or passing parameters through an API to other services.

An event log of data is stored and passed between application components. If some error occurs, such as a network failure or bugs in components, the workflow can be pick up at the last task completed instead of executing the whole workflow once again.

Step Functions frees microservices from excess code, so the application is easier to maintain. It allows the graphical visualization of our workflows. The graphical console also delivers real-time diagnostics and dashboards. The charging for this AWS service is based on the number of state transitions required to execute a workflow, including retries.

Scalability is one primary concern if we want to manage complex tasks and workflows. Step functions have a limit on child workflows (1000 per workflows), but it supports nested workflows. We can create as many nested workflows as we want, and we can start its execution directly from task states. Reusable workflows are quite useful as well as the

[12] https://github.com/uber/cadence-web
[13] https://aws.amazon.com/pt/step-functions/
[14] https://docs.aws.amazon.com/step-functions/latest/dg/concepts-activities.html

modularity provides by Step Functions.

Even though recent, Step Functions are quite mature. This solution provides all the necessary features to execute our machine learning workflows in a scalable and reliable way. The only downside for users is that they have to pay for the service. However, it will be possible to execute complex workflows without spending too much. Amazon Step Functions is charged per state transition and offers a Free Tier with 4,000 state transitions per month. After that, it costs $0.025 per 1,000 state transitions. These charges are applicable for Standard Step Functions, which are the default workflow type.

AWS also offers the option to choose Express Step Functions to run high-volume, event processing workloads. Standard Workflows respond to our need by allowing the execution of long-running, durable, and auditable workflows.

Step Functions support Nested Workflows[15] and dynamic parallellism[16].

The adoption of Step Functions gives us the possibility to the advantage of other AWS services, for example, Lambda Functions and Amazon Simple Storage Service (Amazon S3).

2.3.5 Full Development

An alternative option could be to develop our orchestration solution from the ground up. The DataScience4NP platform is very modular, and no significant architectural modifications are required to change the orchestration solution.

However, reformulating the orchestrator can be a time-consuming task, regardless of the technologies or approaches that can be chosen. DataSicence4NP is designed to provide sequential workflows to its users, but sequential workflows do not have a direct correspondence with system workflows, and there will always be the need to translate logical workflows to system workflows.

Users are presented with logical workflows that are then sent from the GUI to a service called "Workflows Service", one of the microservices in our architecture. This service is responsible for the translation of logical workflows to system workflows in order to be understood by the technology being as an orchestrator and then executed. When a workflow ends, system workflows are translated back to logical workflows to retrieve the results to the users.

Figure 2.10 shows the most straightforward translation that can occur: when the validation procedure is chosen is "useEntireData", meaning that all tasks will use the entire dataset given as input. Nevertheless, the translation of workflows can be more complicated, just as Figure 2.11 demonstrates.

Currently, the Workflows Service is designed to generate system workflows understood by Netflix Conductor. The adoption of a new orchestrator implies the reformulation of the some of its components.

Besides having to do this reformulation to integrate the GUI with a conductor, implement our engine would also require all the conductor development, which can be an even hard and time-consuming task.

[15] https://aws.amazon.com/about-aws/whats-new/2019/08/aws-step-function-adds-support-for-nested-workflows/

[16] https://aws.amazon.com/about-aws/whats-new/2019/09/aws-step-functions-adds-support-for-dynamic-parallelism-in-workflows/

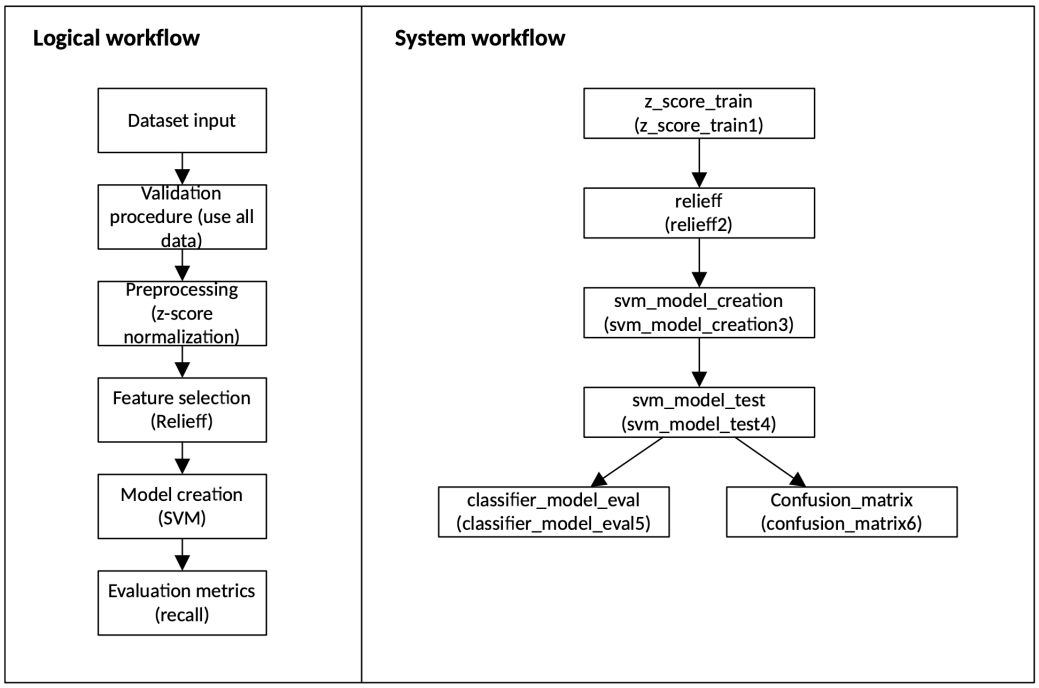

Figure 2.10 – Translation of a workflow where tasks are applied to all input data.

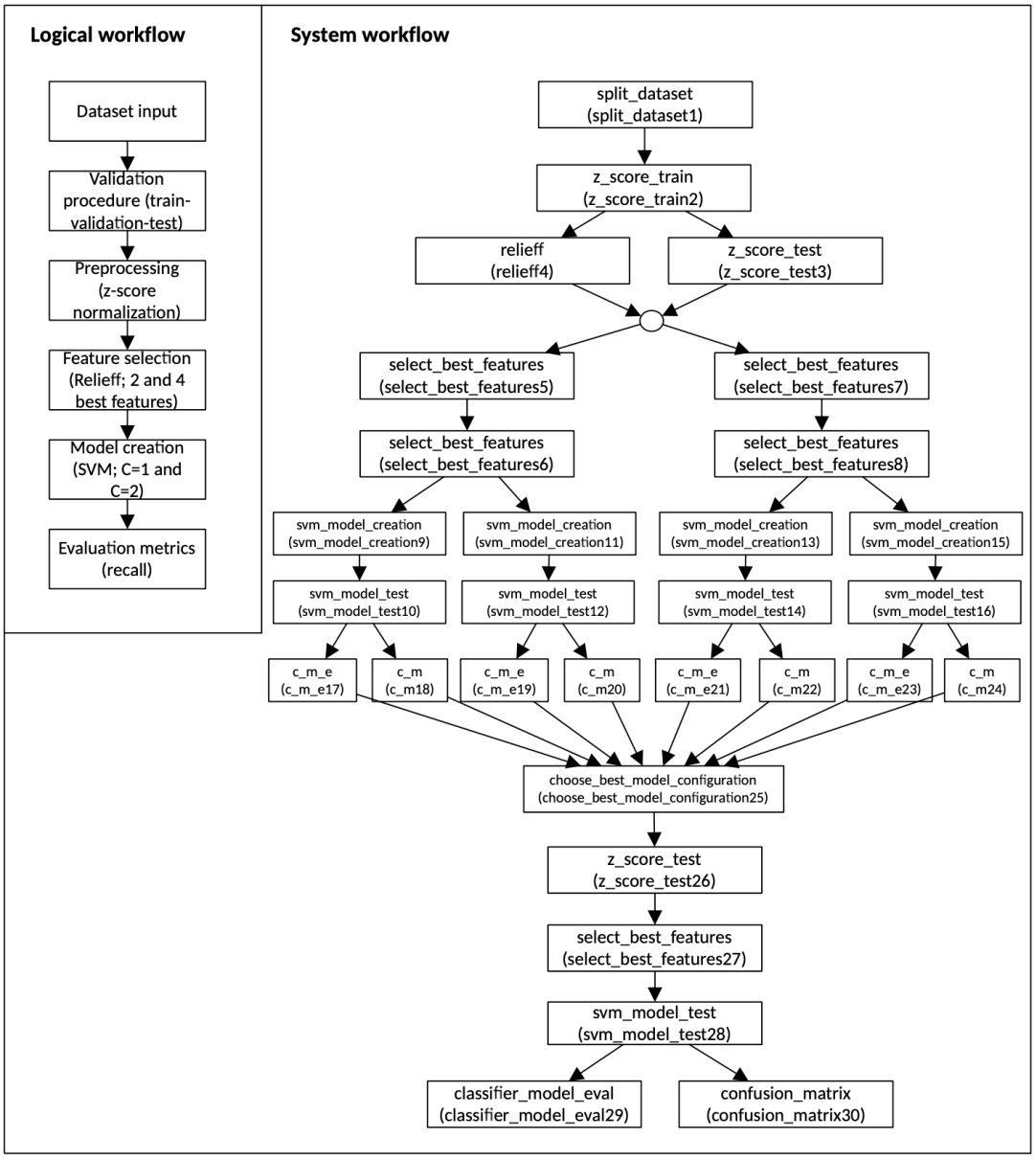

Figure 2.11 – Translation of a workflow where tasks are applied to according to a train-validation-test procedure.

2.3.6 Comparison of the solutions

Table 2.2 – Comparison of workflow orchestration solutions

	Condutor	Zeebe	Cadence	Step Functions
Owner	Netflix	Zeebe	Uber	Amazon
History	2.5 years	1 year	2 years	2.5 years
Visual Workflow Display	Yes	No	Yes	Yes
Flow Definition	JSON	BPMN	GO/Java	JSON
Support for HA	Yes	Yes	?	Yes
Rest API Trigger	Yes	No	No	Yes
Scalability	Yes	Yes	Yes	Yes

Netflix Conductor handles the execution of many workflows, but not when hundreds of tasks compose workflows. By scaling the Dynomite cluster along with dyno-queues it is possible to handle scale and availability needs.

Zeebe is still in "developer preview". It is classified as a technology that allows scalability, but this is not true for complex use cases since workflows are written using visual programming.

Cadence allows workers to be implemented as entirely stateless services, which allows for unlimited horizontal scaling. It provides a web interface[17] to explore workflows, see the ones running, and debug its executions. However, Cadence defines workflows in GO or Java.

Neither Zeebe or Cadence allows the initiation of workflow executions using REST API calls.

Step Functions define workflows using JSON and can coordinate any application that can make an HTTPS connection[21]. These are two of the advantages of Step Functions over Zeebe and Cadence. The integration between AWS services allows Step Functions workflows to call those other AWS services. It scales automatically in response to changing workloads, and it has excellent documentation and support.

[17] https://github.com/uber/cadence-web

This page is intentionally left blank.

Chapter 3

Requirements

The specification of requirements is essential to software development. But, before diving into the specification of our requirements, we should first have a clear idea of what a requirement is.

IEEE Std 1233 1998 [22], defines a requirement as:

"**(A)** A condition or capability needed by a user to solve a problem or achieve an objective.

(B) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.

(B) A documented representation of a condition or capability as in definition (A) or (B)."

The DataScience4NP is an existing prototype that provides workflows for the correct application of machine learning. The functionalities to build machine learning tasks and workflows and also the features for the user interface were already outlined and put in action in the initial prototype. To understand what has been done, to successfully capture the behavior of the system, and to specify the requirements for the new solution, we outline a use case diagram for the DataScience4NP presented in Figure 3.1. This diagram only shows a high-level view of the system and its use cases, which can be decomposed in several requirements. For example, the use case "Insert a task in the workflow" unfolds for each specific task (e.g., Insert a model creation task, Insert the train-validation-test procedure). The same is true for the use case "Visualize the output".

Essentially, we are only concerned with five of the use cases (red use cases): the ones related to the translation and orchestration of workflows and the billing component. However, these use cases are extremely generic. In Section 3.1 we define the requirements that compose these high-level use cases. We also state the requirements for the execution of workflows, and the insertion and visualization of tasks (use cases in grey) since they are implicit in the translation of workflows and will allow us to verify it.

Priority

Even though there are several aspects to be improved in DataScience4NP, this dissertation focus only on the orchestration of Machine Learning workflows. For this reason, and also due to our tight schedule, it is necessary to use a prioritization technique to establish the importance of each requirement and decide which ones should be developed first. The MoSCoW prioritization technique was chosen for this purpose.

Figure 3.1 – Use Cases - Context Diagram.

According to the MoSCoW technique, requirements can be classified using the following convections:

- **Must Have (M)** - requirements critical to the delivery of the system, and that must be satisfied in the solution. The project will fail without them.
- **Should Have (S)** - requirements that represent high-priority features but that are not critical.
- **Could Have (C)** - requirements that portray desirable but not necessary features. These are included if time and resources allow it.
- **Won't Have (W)** - requirements that will not be implemented in the current release but may be included in the future.

Convention

To ensure consistency across the presentation of all requirements, we defined the following convention:

ID	Requirement	Priority
#id	#use-case name and description	#use-case priority

Table 3.1 – Requirement presentation.

The requirement identifier (ID) also follows a logical convention for a better organization:

`Context_RequirementNumber`

where the `Context` corresponds to a group of requirements within the same scope. The contexts are:

- **TKS**: ML Tasks requirements;
- **BS**: requirements concerning the billing system;
- **REQ**: other requirements.

3.1 Functional Requirements

Requirements were defined following the conventions stated before and are presented in Table 3.1. Requirements related to ML tasks (TSK_01 to TKS_41) were previously defined by the two former MCs students Artur Pedroso e Bruno Lopes. These requirements fundamentally outline the whole DataScience4NP platform and are necessary to validate the Workflows Service implemented with AWS Step Functions. Everything related to the translation of workflows is, in fact, implementation decisions to guarantee these requirements. Besides requirements related to ML tasks, simple requirements for the Billing Service were added.

Table 3.2 – System Functional requirements.

ID	Requirement	Priority
TKS_01	**Insert a data preprocessing task on the workflow** - The user shall be able to choose one of the multiple data preprocessing tasks. The system shall insert the task in the chosen position.	M
TKS_02	**Insert a (data preprocessing) min-max scaling task** - The user shall be able to insert a min-max scaling task on the workflow to scale the data used in the workflow and to insert the min and max values to apply the scaling on the dataset.	M
TKS_03	**Visualise the output of a (data preprocessing) min-max scaling task** - The user shall be able to visualize the output of a min-max scaling task (the scaled datasets, the scaled attributes, and an object to apply the scaling operation to new data) after it has been executed in a workflow.	M
TKS_04	**Insert a (data preprocessing) z-score normalization task** - The user shall be able to insert a z-score normalization task on the workflow to normalise the data used in the workflow.	M
TKS_05	**Visualise the output of a (data preprocessing) z-score normalization task** - The user shall be able to visualize the output of a z-score normalization task (the normalized datasets, the normalized attributes and the object to apply normalization to new data) after it has been executed in a workflow.	M
TKS_06	**Insert a (data preprocessing) one-hot encoding task** - The user shall be able to insert a one-hot encoding task on the workflow to convert discrete data to a numerical format to use in the workflow	M
TKS_07	**Visualise the output of a (data preprocessing) one-hot encoding task** - The user shall be able to visualize the output of a one-hot encoding task (encoded datasets and the labels included in the encoded datasets) after it has been executed in a workflow.	M
TKS_08	**Insert a (data preprocessing) discretization task** - The user shall be able to insert a discretization task on the workflow to have only discrete data to use in the workflow	S
TKS_09	**Visualise the output of a (data preprocessing) discretization task** - The user shall be able to visualize the output of a discretization task (datasets where the operation was applied) after it has been executed in a workflow	S

ID	Requirement	Priority
TKS_10	**Insert a feature selection task on the workflow** - The user shall be able to choose one of the multiple feature selection tasks. The system shall insert the task in the chosen position.	M
TKS_11	**Insert a (feature selection) Relieff algorithm** - The user shall be able to insert the Relieff algorithm in the workflow to visualize the relevance of the features of a dataset being processed on the workflow and to select the most relevant features. The user shall be able to specifiy a threshold and a number of best features to be selected according to the ranking produced by the algorithm.	M
TKS_12	**Visualise the output of a (feature selection) Relieff task** - The user shall be able to visualize the output from the application of a feature selection task that uses the Relieff algorithm (ranking and scores produced by Relieff, the number of selected features and the threshold used) after it has been executed in a workflow.	M
TKS_13	**Insert (feature selection) Info Gain algorithm** - The user shall be able to insert the Info Gain algorithm in the workflow to visualize the relevance of the features of a dataset being processed on the workflow and to select the most relevant features. The user shall be able to specify a threshold (used by the system to remove features from the dataset with information gain below the threshold), and a number of best features to be selected according to the ranking produced by the algorithm.	M
TKS_14	**Visualise the output of a (feature selection) Info Gain task** - The user shall be able to visualize the output from the application of a feature selection task that uses the Info Gain algorithm (the ranking and the computed information gain, the number of selected features and the threshold used) after it has been executed in a workflow.	M
TKS_15	**Insert a dimensionality reduction task on the workflow** - The user shall be able to insert a dimensionality reduction task in the workflow to reduce the dimensionality of the dataset used in the workflow. The system shall insert the task in the chosen position.	M
TKS_16	**Insert (dimensionality reduction) principal component analysis (PCA) task** - The user shall be able to insert a PCA task in the workflow to reduce the dimensionality of the dataset used in the workflow. The user shall be able to insert values to be used by the PCA algorithm.	S

ID	Requirement	Priority
TKS_17	**Insert a model creation task on the workflow** - The user shall be able to insert a model creation task in the workflow to create a classification model trained with the dataset used in the workflow. The system shall provide multiple tasks to create a classification model using functional, tree, rule, lazy, or Bayesian algorithms for the user to choose, and the user shall select one of the available algorithms.	S
TKS_18	**Insert a task to create a classification model using a functional SVM algorithm** - The user shall be able to insert a model creation task in the workflow to create a classification model trained with the dataset used in the workflow. The user shall be able to add the parameters to be used by the SVM algorithm (C, kernel and the parameters of the kernel)	M
TKS_19	**Visualise the output of a model creation task where the SVM algorithm was used** - The user shall be able to visualize the output of a model creation task where the SVM algorithm was used after it has been executed in a workflow. The system shall output the created model for download, the parameters used to create the model, and the predicted and expected results produced after testing the model.	M
TKS_20	**Insert a task to create a classifier of the type tree using the CART algorithm (M)** - The user shall be able to insert a model creation task in the workflow to create a classification model trained with the dataset used in the workflow.	M
TKS_21	**Visualise the output of a model creation task where the CART algorithm was used** - The user shall be able to visualize the output of a model creation task where the CART algorithm was used after it has been executed in a workflow. The system shall output the created model for download, a tree with the representation of the model, the parameters used to create the model, and the predicted and expected results produced after testing the model.	M
TKS_22	**Insert a task to create a classifier of the type lazy using the K nearest neighbors algorithm** - The user shall be able to insert a model creation task in the workflow to create a classification model trained with the dataset used in the workflow. The user shall be able to insert the parameter K to be used by the K nearest neighbors algorithm.	M

ID	Requirement	Priority
TKS_23	**Visualise the output of a model creation task where the K nearest neighbors algorithm was used** - The user shall be able to visualize the output of a model creation task where the K nearest neighbors algorithm was used after it has been executed in a workflow. The system shall output the created model for download, the parameters used to create the model, and the predicted and expected results produced after testing the model.	M
TKS_24	**Insert a task to create a classifier of the type bayesian using the Gaussian Naïve Bayes algorithm** - The user shall be able to insert a model creation task in the workflow to create a classification model trained with the dataset used in the workflow.	M
TKS_25	**Visualise the output of a model creation task where the Gaussian Naïve Bayes algorithm was used** - The user shall be able to visualize the output of a model creation task where the Gaussian Naïve Bayes algorithm was used after it has been executed in a workflow. The system shall output the created model for download, the parameters used to create the model, and the predicted and expected results produced after testing the model.	M
TKS_26	**Insert a task to create a classifier of the type bayesian using the Multinomial Naïve Bayes algorithm** - The user shall be able to insert a model creation task in the workflow to create a classification model trained with the dataset used in the workflow.	M
TKS_27	**Visualise the output of a model creation task where the Multinomial Naïve Bayes algorithm was used** - The user shall be able to visualize the output of a model creation task where the Multinomial Naïve Bayes algorithm was used after it has been executed in a workflow. The system shall output the created model for download, the parameters used to create the model, and the predicted and expected results produced after testing the model.	M
TKS_28	**Insert a task to validate the model being constructed using different procedures** - The user shall be able to insert a validation procedure task in the workflow to specify the procedure that must be used while executing the tasks that compose the workflow.	M

ID	Requirement	Priority
TKS_29	**Insert the cross-validation procedure to validate the model constructed in the workflow** - The user shall be able to insert a validation procedure task in the workflow to specify the procedure that must be used while executing the tasks that compose the workflow. The user shall be able to specify the number of folds to be used, the number of repetitions to perform the cross-validation. The user shall be able to select nested or normal cross-validation, and if the data in the cross-validation folds must be stratified. The user shall be able to specify if the data must be shuffled before the creation of the folds. The user shall be able to select an integer number to be used as a seed while shuffling and stratifying the data.	M
TKS_30	**Visualise the datasets used in a cross-validation procedure** - The user shall be able to visualize the datasets used in the different folds of an executed workflow.	M
TKS_31	**Insert the hold-out procedure to validate the model constructed in the workflow** - The user shall be able to insert a validation procedure task in the workflow to specify the procedure that must be used while executing the tasks that compose the workflow. The user shall be able to insert the proportion of data to use in the training and test partitions, if the data in the partitions must be stratified, and if data must be shuffled before the creation of the partitions. The user shall be able to select an integer number to be used as a seed while shuffling and stratifying the data.	M
TKS_32	**Visualise the datasets used in a hold-out procedure** - The user shall be able to visualize the datasets used as training and test sets in an executed workflow.	M
TKS_33	**Insert the train-validation-test procedure to validate the model constructed in the workflow** - The user shall be able to insert a validation procedure task in the workflow to specify the procedure that must be used while executing the tasks that compose the workflow. The user shall be able to insert the proportion of data to use in the training, validation and test partitions if the data in the partitions must be stratified if the data must be shuffled before the creation of the partitions. The user shall be able to select an integer number to be used as a seed while shuffling and stratifying the data.	M
TKS_34	**Visualise the datasets used in a train-validation-test procedure** - The user shall be able to visualize the datasets used as training, validation and test sets in an executed workflow.	M

ID	Requirement	Priority
TKS_35	**Insert the "use all data" procedure to construct tasks and validate the model constructed in the workflow using all data** - The user shall be able to insert a validation procedure task in the workflow to specify the procedure that must be used while executing the tasks that compose the workflow.	M
TKS_36	**Select classification performance metrics** - The user shall be able to select a classification performance metric (metrics accuracy, precision, recall and f-measure) to verify the classification performance of the classifier created in the workflow.	M
TKS_37	**Visualise the classification performance of a produced model** - The user shall be able to visualize the classification performance according to certain metrics after the execution of a workflow. The system shall output the classification performance according to the metrics previously selected and also a confusion matrix.	M
TKS_38	**Insert a feature selection task to build models using different numbers of features** - The user shall be able to insert a feature selection task to build models with different features in order to obtain the model with the best features according to a classification performance metric. The user shall be able to specify more than one value as the number of attributes to select inside the feature selection task, and the preferred classification performance metric to be used as the decider for the best model configuration.	M
TKS_39	**Visualise the configuration of features that produced the best model** - The user shall be able to visualize the configuration of features that produced the best model after the execution of a workflow. The system shall output the ranking of the features and the number of best-selected features.	M
TKS_40	**Insert a model creation task to build models using different parameters** - The user shall be able to insert a model creation task to build models with different parameters in order to obtain the model with the best parameters according to a classification performance metric. The user shall be able to specify more than one value in the parameters of the model creation task, and a preferred classification performance metric to be used as the decider for the best model configuration.	M
TKS_41	**Visualise the configuration of parameters that produced the best model** - The user shall be able to visualize the configuration parameters that produced the best model after the execution of a workflow.	M

ID	Requirement	Priority
BS_01	**Contabilize resources** - The system shall determine the amount of resources used by a user.	M
BS_02	**Determine expenses** - The system shall account for the total to be paid by a user according to the resources used.	M
BS_03	**Consult expenses** - The user shall be able to see the total of his expenses at any time.	M
BS_04	**Client billing account details** - The user shall be able to provide their bank account details.	S
BS_05	**Billing Notification** - The user shall be able to receive a notification after its bill being fetched.	S
BS_06	**Charge expenses** - The user shall be periodically charged for its expenses.	S
REQ_01	**Add sample datasets** - The user shall be able to access sample datasets and use them in his workflows.	C
REQ_02	**Add sample workflows** - The user shall be able to access sample workflows and use them.	C
REQ_03	**Add trigger alarms** - The user shall be able to choose monitoring options and to be alerted accordingly	C
REQ_04	**Recover password** - The user shall be able to recover its password.	C
REQ_05	**Create an Help Page** - The user shall be able to consult a page with indications on how to use the interface and build workflows.	C

3.2 Quality Attributes

Another architectural driver that shapes the architecture of a system, besides functional requirements, are quality attributes. Quality attributes, also called " Non-Functional Requirements ", are properties that add value to the system and consequently to its users. These properties are not directly related to the functionality of a system neither visible to users, yet condition far more the architecture of a system than functional requirements.

Quality attributes are difficult to identify and describe. However, to prove how well the system satisfies its quality attributes, they must be testable and, therefore, well stated. An efficient way to describe quality attributes is through scenarios of quality attributes, that is, circumstances in which a quality attribute can be verified and measured. A quality attribute scenario is composed of six elements:

- **Source of Stimulus**: Internal or external entity that generates the stimulus.
- **Stimulus**: Condition that requires a response when it arrives at a system.
- **Environment**: The state of the system when the stimulus occurs (e.g., under heavy traffic).
- **Artifact**: The component or group of components being stimulated.
- **Response**: The specific activity that results from the arrival of the stimulus (undertaken by the artifact).
- **Response Measure**: When the response occurs, it must be measured in some way so that the attribute can be tested.

Utility trees allow the representation of the overall usefulness of a system through the definition and prioritization of quality attributes and their respective scenarios. The utility tree for the DataScience4NP platform is presented in Table 3.3, where the first level symbolizes the key quality attributes, the second corresponds to the attributes' characterization, and the last one has the quality attributes scenarios (leaves).

Quality attributes priorities were established using a relative raking:**(H)** High, **(M)** Mid and **(L)** Low.

Two dimensions were considered for prioritizing the attributes:

- the importance of each quality to the success of the system
- the degree of perceived risk posed by the achievement of such quality

For example, a quality attribute with ranking (H, M) means that it has high importance to the success of the system and medium risk to achieve. Thus the quality attributes to be addressed first on the platform were the ones with priority (H, H).

Just like most of the functional requirements, the key attributes of the system are inherited from the previous work (outlined by the two former MCs students) and remained the same.

Table 3.3 – Utility Tree.

QA	QA Characteristics	QA Scenario
Modularity	Separation of each service in clear and well defined modules	**QAS_01**: A developer introduces a new service to the system during system development. The new service is deployed as an independent and interchangeable module and has everything required to execute its well-defined functionality. **(L, H)**
Maintainability	Capacity to adapt to new functionalities	**QAS_02**: A developer introduces a new ML service in the system during system development. The new service is deployed in the system and becomes available to other system components after registration without involving changes in more than three other services. **(M, M)**
	Capacity to make isolated changes in a service	**QAS_03**: A developer changes the functionalities of one service while the system is under development. The modified service is deployed without requiring any adjustment in the remaining system. **(M, M)**
Interoperability	Capacity of communication between different services	**QAS_04**: A developer creates a microservice while the system is under development. The new microservice becomes available in the system and can communicate with other services through technology-agnostic communication protocols (e.g., HTTP). **(M, H)**
Scalability	Ability to maintain the performance independently of the number of requests	**QAS_05**: A user places two times more requests than what is expected by the system during normal operation. The system processes this load without loss of performance. **(H, H)**

This page is intentionally left blank.

Table 3.4 – Utility Tree (continuation).

QA	QA Characteristics	QA Scenario
Usability	Responsiveness	**QAS_06**: An authenticated user or external system requests the execution of a ML workflow, while the system is in normal operation. The system starts the execution of the workflow and keeps the information about the status of every task that composes the workflow, to inform the user about the progress of execution of a workflow. **(H, L)**
	Learnability	**QAS_07**: An authenticated user executes an operation in the system, while the system is in normal operation. The user will not have difficulties learning how to use the interface or finding the required functionalities. **(H, M)**
	Efficiency	**QAS_08**: An authenticated user or external system adds a task to the workflow, while the system is in normal operation. The system inserts the task on the workflow with default values on every parameter if no parameters were provided. **(H, L)**
		QAS_09: An authenticated user creates a workflow to produce models with different features and parameters to have access to the best model configuration, while the system is in normal operation. The system executes the workflow, creating models with different features and parameters and returns the best model to the user, without requiring the user to create multiple workflows with the different features and parameters to produce the best model configuration. **(H,M)**
	Effectiveness	**QAS_10**: An authenticated user has a specific goal that is achieved through the execution of an ML workflow. The user creates the workflow and executes it, while the system is in normal operation. The workflow is properly configured at least 80% of the times, and the expected results are displayed by the system. **(H, L)**

Table 3.5 – Utility Tree (continuation).

QA	QA Characteristics	QA Scenario
Availability	System uptime	**QAS_11**: A service crashes, while the system is in normal operation. The system detects the crash and heals the service maintaining the functionalities of every service available 99.9% of the time. **(M, H)**
Performance	Execution speed of ML tasks	**QAS_12**: A ML task arrives at the system to be processed, while the system is in normal operation. The task is executed by the system with a performance comparable to running the same task on an average computer (8GB of Random Access Memory (RAM), 4 cores (3GHz each)). **(H, M)**
Elasticity	Ability to use resources efficiently	**QAS_13**: The system verifies that some modules/services are consuming memory or CPU below the average, while the system is in normal operation. The system gradually removes the modules/services that are consuming memory or CPU below the average. **(M, H)**
		QAS_14: The system verifies that some modules/services are consuming memory or CPU above the average, while the system is in normal operation. The system gradually increases the number of modules/services that are consuming memory or CPU above the average. **(M, H)**
Security	Confidentiality	**QAS_15**: An authenticated user inserts a dataset in the system, while the system is in normal operation. The system receives the dataset and makes the dataset accessible only by authorized users. **(H, M)**
		QAS_16: An unauthorized user inserts a wrong username or password to enter the system, while the system is in normal operation. The system denies the entrance to the user without exposing any data associated with the inserted username. **(H, M)**

Chapter 4

Architecture

This chapter details the architecture of the DataScience4NP platform. Since this dissertation is based on an existing prototype, we first present the original architecture outlined for the platform, and only then we reveal the architectural changes we are proposing.

4.1 Initial Architecture

The architecture is modeled following the C4 model[1]. The C4 model employs different level to abstract static structures of software systems using containers (a context or boundary where code is executed or data is stored) and components (a grouping of related functionality encapsulated behind a well-defined interface).

Level 1: Context Diagram

The context diagram is represented in Figure 4.1 where it is possible to observe the interactions of the DataScience4NP with external systems and users. Users can use the DataScience4NP platform through a graphical interface that allows them to send HTTP requests to the system. The same HTTP requests can be sent by other external systems to access the same exposed functionalities.

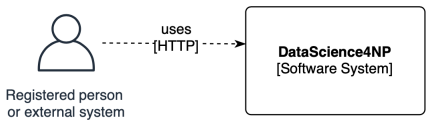

Figure 4.1 – System Context Diagram.

Level 2: Container Diagram

The container diagram intends to zoom-in into the software system to expose the system containers, which are separable runnable/deployable units such as microservices, databases, file systems, serverless functions, server-side and client-side applications.

Several containers compose DataScience4NP as we can observe from the container diagram presented in Figure 4.2. It follows a brief outline of each container.

[1] https://c4model.com

The **API Gateway** is the entry point to the system, and it is responsible for routing every request to the appropriate microservice. Based on the path in the request, the API request redirects the request to the corresponding service. For example, "index.html" redirects to GUI service, "/datasets" redirects to Dataset service, and so on.

The **GUI Service** delivers the graphical interface to users.

The **User Service** enables the registration and authentication of users. All users have access to the GUI service, but only authenticated ones can access other services accessible from the API gateway.

The **Logs Service** is currently outdated, but it was initially developed to receive logs from actions taken by users on the GUI and provide developers with behavior data, thus helping them improve the interface.

The **Tasks Service** describes data science tasks that can be used by users or external systems in the GUI to build a machine learning workflow.

The **Dataset Service** is responsible for storing datasets uploaded by users and its metadata in the system. In the initial prototype, the uploaded datasets are stored in a Distributed File System.

The **Templates Service** is currently outdated. It was developed with the intention of delivering predefined workflows to users.

The **Workflows Service** is responsible for the translation of the logical workflows to system workflows. Logical workflows are the sequential data science workflows created by users using the graphical interface. These workflows are translated to what we call "system workflows" that are in a lower-level format to be understood by the orchestration service. Each logical workflow is associated with its low-level representation and is stored in a database so that it can be retrieved later to verify its execution status.

The **Orchestration Service** manages the execution of low-level workflows translated in the Workflows Service. This service is performed using Netflix Conductor. The Orchestration Service receives workflows to be started, and it puts the tasks that form it in queues where the Machine Learning can pull them for execution. This service holds the execution status of the workflows and the outputs of each processed task on the workflow.

The **Machine Learning (ML) Services** are the different microservices (written in Java and Python) responsible for the execution of the specific tasks that compose system workflows. These services query the Orchestration Service for tasks and, upon receiving them, they proceed to its execution and return the results to the orchestration service.

A **Distributed File System (DFS)** is used to store datasets. For simplicity purposes, the initial prototype uses a normal file system that is accessed by the Datasets Service to save the datasets uploaded by users and to retrieve them when asked, and for the Workflows Service not only to read the datasets but also to store its outputs and access the output of other services (for example, models and predictions).

4.2 Current Architecture

The choice of the AWS Step Functions for the orchestrator of the DataScience4NP initially motivated a solution 100% built on top of Amazon Web Services. An application to get AWS Cloud Credits for Research[2] was submitted on December 31th of 2019, and a new architecture to build the DataScience4NP with AWS services was designed. However, we

[2] https://aws.amazon.com/pt/research-credits/

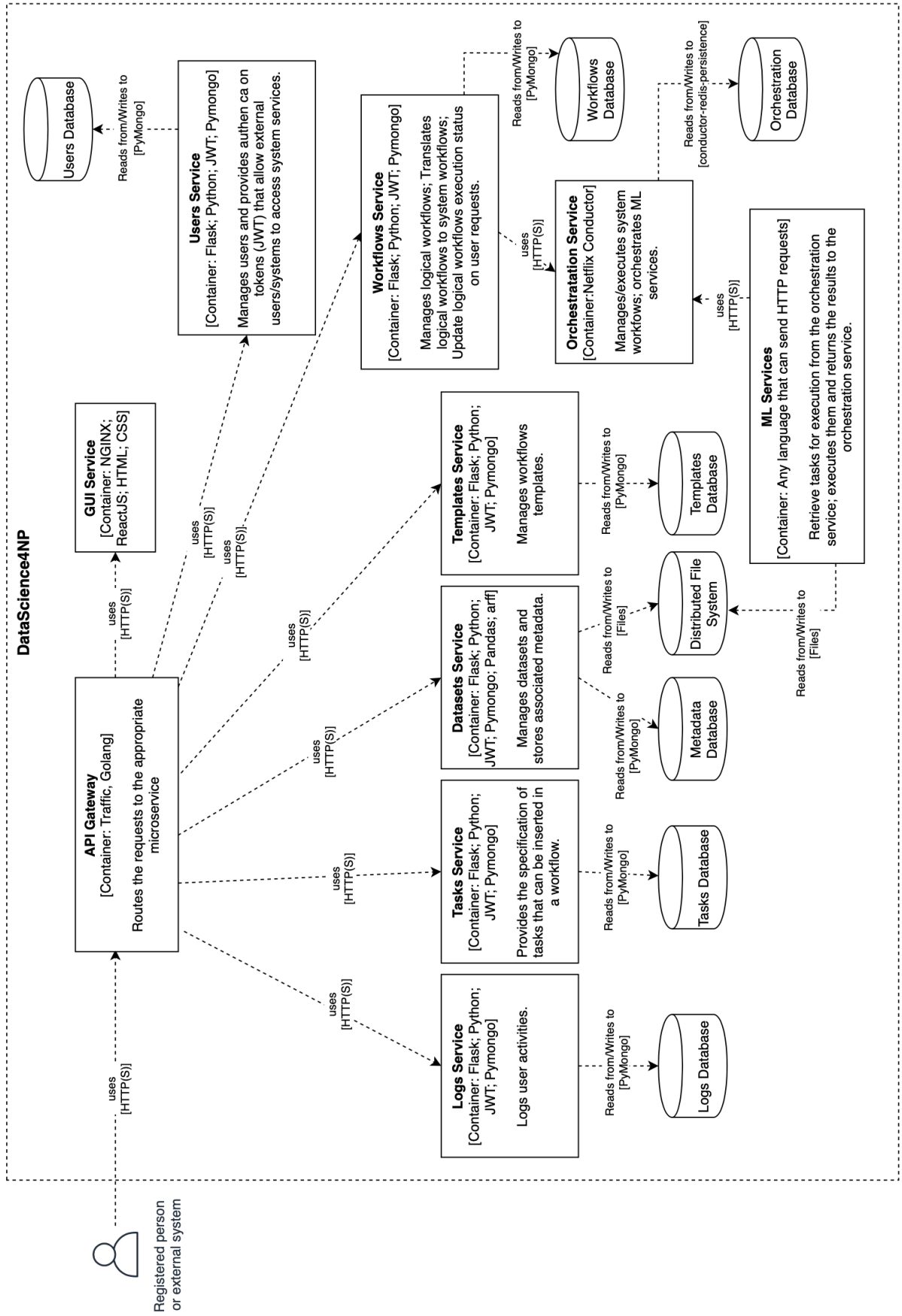

Figure 4.2 – General system container diagram (previous architecture).

did not get any response from AWS. A negative (or lack of) response to the application was a predicted risk (Section 7.2), thus the AWS architecture was set aside (Appendix B).

The integration of AWS Step Functions in the Workflows Service and on the ML Services was conducted using the AWS Free Tier[3], and the platform was deployed in an already configured Kubernetes cluster (ten machines - one master and nine workers - each one with 4vCPUs/8GbRAM/750HDD) running in INCD[4].

The new architecture can be found in Figure 4.3. The current architecture is similar to the former one, as the existing services continue to have the same responsibilities. Previously, the Orchestration Service (Netflix Conductor) was running in the same environment as the rest of the system. In the new architecture, we no longer have a component for the Orchestration Service. Instead, we have Step Functions. With Step Functions, everything related to the orchestration of workflows is running on the AWS side. Workflows Service create state machines and send them to Step Functions and ML Services interact with Step Functions to poll tasks and to send their result.

Step Functions coordinates workflows that are made up of a collection of steps/tasks where an ML service executes each task and where the output of one step acting as input into the next. Step Functions receives state machines from Workflows Service and automatically triggers and tracks each step, so our ML workflows can be executed orderly and as expected. When a task is reached, it is in condition to be executed by the respective ML service. Step Functions waits for the result of the task performed by the ML service and, according to it, it continues to manage the workflow (either by stoping the workflow if the task failed or by wiring the output result to the rest of the workflow).

The **Machine Learning (ML) Services** are continually polling for tasks but now from Step Functions. When a task is received, the ML service performed its work and then sends the result to Step Functions.

Billing Service is a new component in the system. It has the responsibility of, according to the time of computation used in users' workflows, determining the monthly amount to be charged. This component should interact with a payment provider to charge users their monthly bills. Easypay was one solution that was briefly explored but not implemented due to time constraints. Easypay "is a Payment Institution supervised by the Portuguese regulator Banco de Portugal and is authorized to provide its services in any SEPA country". It has a well-documented API, and through frequent payments with SEPA Direct Debit would be possible to monthly charge users.

[3] AWS Free Tier

[4] INCD - Infraestrutura Nacional de Computação Distríbuida

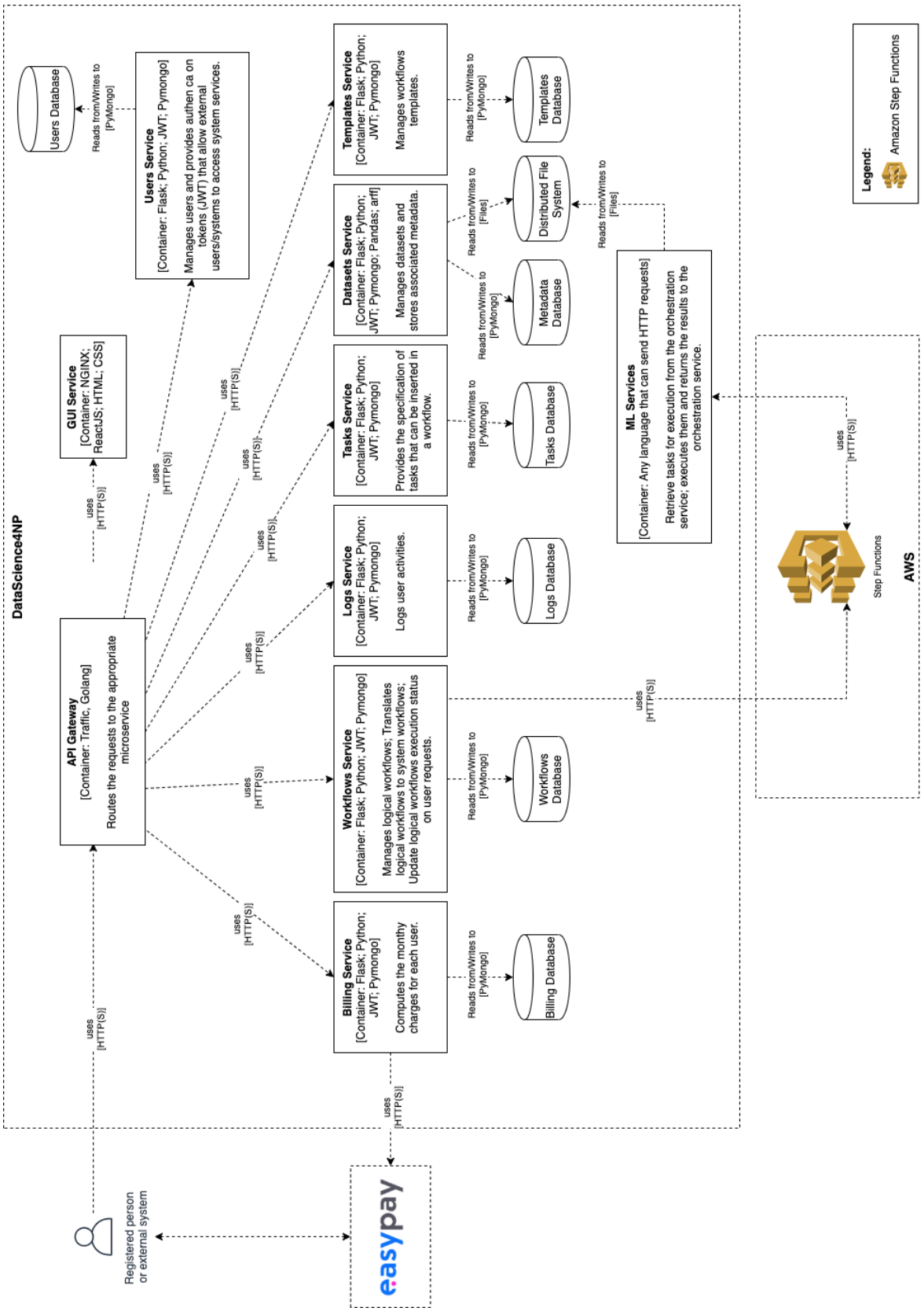

Figure 4.3 – General system container diagram (current architecture).

This page is intentionally left blank.

Chapter 5

Implementation

This chapter focuses primarily on the internal operation of the Workflows Service, the one responsible for the translation of the logical workflows to a state machine - a workflow understood by AWS Step Functions. It intends to explain how the translation is processed, how Step Functions work, its role in our implementation. It also exposes the limitations imposed by AWS and the attempts made to overcome them.

5.1 Conventions

Several terms used in this document can cause some misunderstandings. To avoid such situations, it follows a brief explanation of some terms:

- Logical Workflow: a sequential description of a workflow composed by users in the graphical interface (format: JSON).
- Logical Task: task presented to users in the graphical interface and that are added to logical workflows (format: JSON).
- System Workflow: a workflow that results from the translation process. It is a workflow understood by the orchestrator (previously system workflows were defined using JSON based DSL to be understood by Netflix Conductor and now are defined in Amazon States Language to be understood by Step Functions)
- System Task: task that composed the sytem workflow.
- State Machine: AWS term for system workflow (format: JSON-based Amazon States Language).
- State: AWS term for system task (format: JSON-based Amazon States Language).

5.2 Translation Process

From a higher perspective, what the Workflows Service does is receive a logical workflow and, given the specified tasks and its details, translated it to a system workflow and then sent it to AWS Step Functions. However, there is a great extent of steps performed (and a lot of components involved in) between the moment the Workflows Service receives the logical workflow and the moment it sends the correspondent state machine to Step Functions.

Essentially, each logical task in the logical workflows received turns into an object understood by the Translator (SystemTask), and, according to it, a corresponding strategy is used to create the AWS States correctly.

A brief explanation of the process will follow. Figure 5.1 presents the UML diagram of the core components (in the Workflows Service) used in the translation process. The three main components are the Translator, SystemTasksCreator, and TranslatorState. The SystemTask and its inherent classes also have a fundamental role in the translation but are not represented in Figure 5.1. Bear in mind that there are plenty more components, but to explain how translation is conducted, the ones mentioned are sufficient.

The Translator class holds several variables needed during the translation, for instance, the list of sequential tasks (received in the logical workflow) and queues where worker tasks and the generated tasks, objects descendent from SystemTask, are placed. Its context is shared with the SystemTasksCreator and TranslatorState classes, and its state might change during the translation.

The SystemTasksCreator and its inherent classes specify the strategies to create worker tasks. Each type of logical task has a specific strategy to be translated. During translation, worker tasks (tasks that will compose the system workflow) are created and encapsulated in objects descendent from SystemTask so the translator state can read them. The TranslatorState defines how the encapsulated tasks placed in the translator queue are inserted in the final workflow (according to the tactics implemented by the state).

The translation starts by setting the state of the Translator as "StartState," and then it begins creating tasks (using a strategy according to the logical task being processed) and puts them in the translator queue (system_tasks_queue) (responsibility of the SystemTaskCreator). The worker tasks are encapsulated in instances inherit from SystemTask class and placed with such format in the translator queue (system_tasks_queue).

Then, based on the tactic implemented on the current state of the Translator, the tasks in the translator queue are placed into another queue named wf_holding_tasks as worker tasks. At the end of the translation, the wf_holding_tasks is used to generate the final workflow.

The process of inserting tasks in the workflow continues as long as there are no tasks in the translator queue. The translator state might change according to the tasks in queue, consequentially changing the tactic implemented to insert the tasks in the wf_holding_tasks queue.

In reality, the translation process does not end after the created state machine being sent to Step Functions. Upon workflow completion (successfully or unsuccessfully), we need to return the results to the user. To achieve this, the Workflows Service needs to receive outputs from Step Functions and translate them back to a logical workflow to present the results in the graphical interface. Moreover, the graphical interface informs the user about the status progression of each task during workflows' execution. Thus, it is needed to know which system tasks have and haven't been completed and to each logical task do they correspond. The way of getting the outputs and how the states' status is determined is explained in Section 5.3.4. After having the states' output, and after all of them being completed, outputs are formated using a somehow reversed logic with the LogicTaskOutputGenerator class, which generates the output for each logical task according to a specific strategy and using the output of the associated worker tasks.

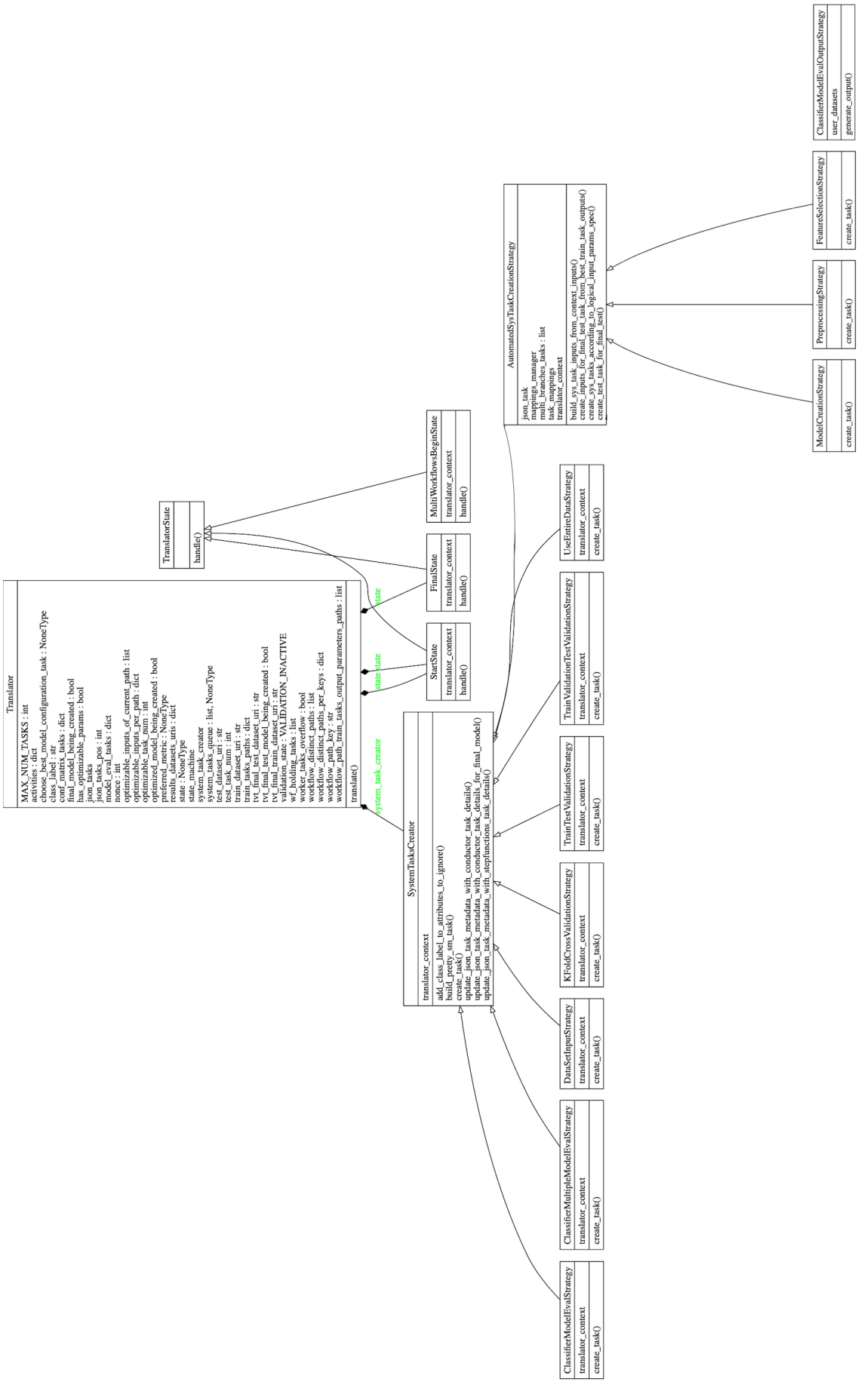

Figure 5.1 – UML diagram of the main classes used in translation

5.3 Step Functions

"Step Functions is based on the concepts of tasks and state machines."[23] Simply put, a state machine (AWS term for workflow) is a collection of states and is defined using the JSON-based Amazon States Language[1]. There are a few types of States, and each one performs a different functionality [24]:

- **Choice state**: chooses between branches of execution.
- **Fail or Succeed state**: stops execution with failure or success.
- **Pass state**: passes its input to its output or inject some fixed data.
- **Wait state**: provides a delay for a certain amount of time or until a specified time/-date.
- **Task state**: does some work in our state machine.
- **Parallel state**: begins parallel branches of execution.
- **Map state**: dynamically iterates steps.

Task and Parallel states are the two primary states types to build state machines for the DataScience4NP platform.

As the name suggests, Parallel states are used to create parallel branches of execution. We use this type of state whenever we need to parallelize tasks. For example, parallelization is present when the chosen validation procedure is cross-validation: there will be branches for each fold. If the cross-validation procedure also includes repetitions, a branch for each repetition is created using different seeds. Parallelization is also present whenever two or more parameters are specified in the model creation tasks (a branch for each model is created, and there will be as many models as there are distinct configurations of parameters and features).

A Task state "represents a single unit of work performed by a state machine"[23] and it can be an Activity, a Lambda Function, or a supported AWS service. Activities allow us to have a task where a worker performs the work, i.e., we can associate a running code (hosted anywhere) with specific tasks in a state machine through Activities. Each Activity has an Amazon Resource Name (ARN). ARNs are used by workers to poll tasks to perform their work. In the DataScience4NP platform, the Machine Learning services are the workers.

An Activity for each worker was created. Activities have a name, and the ARN mentioned above. When the Translator is initiated, the list of Activities is obtained from Step Functions, and a dictionary with Activities names and ARNs is created so we can associate the ARN to the correct Tasks during translation. Workers also get the list of Activities when they are initiated to find the corresponding ARN to its worker name tasks. When the ARN is obtained, workers begin to poll for tasks using a “GetActivityTask” call, which returns a response whenever a Task with its ARN starts. This response includes the input and a taskToken, a unique identifier for the task polled. The taskToken is then used to report the success or failure of work using either “SendTaskSuccess” or “SendTaskFailure” call (these calls also include the result of the work: either the output of the work or the reason of failure).

Listing 1 demonstrates an example of a state machine [25]. A state machine must have two top-levels fields: and object “States” and a string “StartAt”. The value of “StartAt”

[1] https://states-language.net/spec.html

must be one of the names of the "States" fields so the interpreter know where to start running the machine. States (such as Tasks and Parallel States) are expressed as fields of the "States" object. In the example of Listing 1, a state of type "Task" that executes a Lambda function is presented. All states must have a type and depending on the type it may have some required fields. For example, States of type "Task" have a field "Resource" which it is required. "Resource" is an ARN that uniquely identifies the specific task to execute. The definition of the state machine's control flow is achieved through transitions: all non-terminal states must have a "Next" field to specify its subsequent state and the terminal ones should specifiy `{ "End": true }`.

Listing 1 – State machine structure.

```
{
    "Comment": "A simple minimal example of the States language",
    "StartAt": "Hello World",
    "States": {
    "Hello World": {
      "Type": "Task",
      "Resource": "arn:AWS:lambda:us-east-1:123456789012:function:HelloWorld",
      "End": true
    }
  }
}
```

Besides Task states, we also use Parallel states to create state machines for the DataScience4NP. Its type is "Parallel" and the field "Branches", which is an array whose elements must be objects (each one must contain the fields "States" and "StartAt") is required. Listing 2 presents a Parallel state with two branches, where each branch has only one Task state, and Figure 5.2 shows the corresponding state in an AWS graphic.

Listing 2 – Parallel State structure.

```
"ParallelState": {
  "Type": "Parallel",
  "Branches": [
    {
      "StartAt": "OneBranch",
      "States": {
        "OneBranch": {
          "Type": "Task",
          "Resource":
            "arn:AWS:lambda:us-east-1:123456789012:function:AddressFinder",
          "End": true
        }
      }
    },
    {
      "StartAt": "AnotherBranch",
      "States": {
        "AnotherBranch": {
          "Type": "Task",
          "Resource":
            "arn:AWS:lambda:us-east-1:123456789012:function:PhoneFinder",
          "End": true
        }
      }
    }
  ],
  "Next": "NextState"
}
```

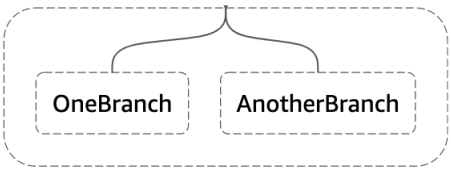

Figure 5.2 – Simple example of a Parallel State.

5.3.1 Step Functions in the code

As shortly explained before, worker tasks are created during the translation process. Previously, the creation of worker tasks and the workflow were done using Condu[2]. Condu is a python client created by Bruno Lopes, one of the students previously in charge of the DataScience4NP platform. Netflix Conductor has a Python client[3] that provides workflow management APIs (start, terminate, get workflow status, etc.) and a worker execution framework. Nevertheless, it does not offers functionalities to define tasks and workflows, and hence the need to extend the official client emerged.

AWS provides SDKs for some programming languages. One of them is Boto, the Amazon Web Services (AWS) SDK for Python. Boto[4], as the others SDKs, allows the creation, configuration, and management of AWS services, including AWS Step Functions. However, and just like Netflix Conductor, it does not enables the creating of single tasks neither the construction of states machines. We wanted to keep using the translation process aforementioned but with Step Functions.

In Step Functions, a worker task is, in fact, a State. To be capable of creating States during translation and of having a state machine defined in JSON-based Amazon States Language at the end of the process, new classes were created to abstract the AWS objects. Figure 5.3 presents these classes.

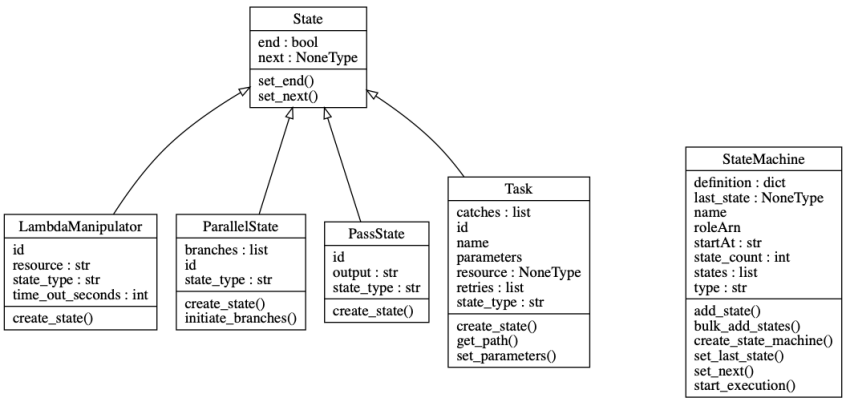

Figure 5.3 – UML diagram of the Step Functions objects.

The Translator class is initiated with a StateMachine object as a variable. During the translation, instances of the State class (or its inherent classes - Task and Parallel) are created. These are the so-called worker tasks that are added to the wf_holding_task queue. At the end of the translation process, the wf_holding_task queue turns to be the list "states" of the StateMachine class.

[2] https://github.com/bioslikk/condu
[3] https://github.com/Netflix/conductor/tree/master/client/python
[4] https://boto3.amazonAWS.com/v1/documentation/api/latest/index.htML

Methods "create_state" (example in Listing 3) transform the class to States in JSON-based Amazon States Language, ready to be added to the top-level field "States" of the state machine definition. While the states are being created, methods are called at specific times to set the "Next" or "End" fields.

Listing 3 – Method create_state from Task class.

```
    def create_state(self):
        task = {}
        task['Type'] = self.state_type
        task['Resource'] = self.resource
        task['Parameters'] = self.parameters
        task['ResultPath'] = '$.'+ self.id
        task['Retry'] = []
        task['Retry'].append(Retry())
        if hasattr(self, 'next'):
            task['Next'] = self.next if self.next is not None else None
        if hasattr(self, 'end'):
            task['End'] = self.end if self.end is True else False
        return task
```

Once the translation ends, the method create_state_machine is called (Listing 4), the states are generated as described before and sent to Step Functions.

Listing 4 – Method create_state_machine from StateMachine class.

```
    def create_state_machine(self):
        '''
        Creates a new State Machine using Boto3 (the \gls{AWS} SDK for Python).

        Returns
        -------
        response : dict
            stateMachineArn : str
                The Amazon Resource Name (ARN) that identifies the created state machine.
            creationDate : datetime
                The date the state machine is created.
        '''
        client = boto3.client('stepfunctions')
        self.bulk_add_states(self.states)
        self.definition['StartAt'] = self.startAt
        self.set_last_state()
        response = client.create_state_machine(
            name = self.name,
            type = self.type,
            definition = json.dumps(self.definition),
            roleArn = self.roleArn)
        return response
```

5.3.2 Input/Output processing

How are input and output passed from state to state? Step Functions receive JSON texts as input, which is given to the first state in the workflow. Individual states receive JSON as input and also pass JSON as output to the next state. The flow of JSON from state to state is filtered and controlled by four fields [26]:

- InputPath: selects which parts of the JSON input is passed to the task.
- OutputPath: can filter the JSON output to further limit the information that's passed to the output.

- ResultPath: selects what combination of the state input and the task result to pass to the output.
- Parameters: enables the passage of collections of key-value pairs.

The usage of paths allows the selection of JSON portions from the input or the result. "A path is a string, beginning with $, that identifies nodes within JSON text. Step Functions paths use JsonPath[5] syntax."[26]

For instance, if there is a task "ABC" that has as an output a parameter "abc" and another subsequent task "WYZ", the task "WYZ" can access the parameter of the previous one (and save it in a new parameter called "wyz") as `wyz.$ = $.ABC.abc`.

In our workflows, it may not be sufficient for a task to receive as input only the output of the previous task. Therefore, it is not in our interest to limit the information passed, and so the "InputPath" and"OutputPath" fields are not adequate. Instead, we define, for each state, the field "Parameters" and "ResultPath". State names must be unique within the scope of the whole state machine. Thus, during the translation, worker tasks are created with a unique identifier. The field "ResultPath" of each task is set with its unique identifier. This means that if a task is called "task_name" with the field `{"ResultPath": $.task_name}` then other tasks can access its output through `$.task_name.OUTPUT_NAME`.

By using "Parameters" and by defining "ResultPath" as described above, we are adding the result of each state to a global input. Thereby, states have access to the output of every precedent states and not only of the one immediately before.

5.3.3 Workers and Step Functions

ML services are individual services, and each one performs a type of task. For example, there is a service responsible for tasks related to SVM model: a task to train, a task to test, and a task for model creation. For each task, a worker exists.

ML services need to communicate with the orchestration service to retrieve tasks, inputs, and to send outputs. As the orchestration service changed, the way ML Services communicate with it is slightly different. ML services are implemented in Python and Java. Still, services can be developed in any language as far as the language provides a way for the service to communicate via HTTP, and there is an AWS SDK available to use for that language.

We are able to have tasks in our state machines where the work is executed by a worker. This is possible employing Activities, an AWS Step Functions feature. Activities have a name and an Amazon Resource Name (ARN) (Figure 5.4). In the state machine, the "Resource" field for each task must be specified, and its value must be the ARN that uniquely identifies the specific task to execute. Then, the worker responsible for that task must be polling with the same ARN.

For instance, if there is an Activity named "relieff" with a respective ARN "arn:AWS:states:eu-west-2:281404888786:activity:relieff", the worker responsible for this task must poll tasks with the same ARN, as exemplified in Listing 5. When a Task in the state machine must be performed by the relieff worker, its resource must also use the same ARN (example in Listing 6).

Listing 5 – Worker polling for a task (example in Python).

[5] https://github.com/json-path/JsonPath

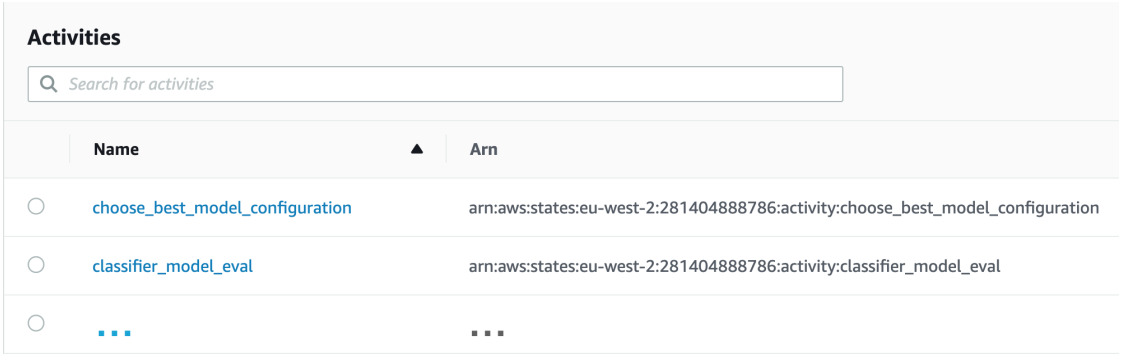

Figure 5.4 – Partial list of activities in the AWS console.

```
response = client.get_activity_task(
    activityArn='arn:\gls{AWS}:states:eu-west-2:281404888786:activity:relieff',
    workerName='relieff_worker'
)
```

Listing 6 – Task with a specific ARN.

```
"ExampleState": {
  "Type": "Task",
  "Resource": "arn:\gls{AWS}:states:eu-west-2:281404888786:activity:relieff",
  "Next": "NextState"
}
```

ARNs are not directly defined in ML services so that whenever the DataScience4NP needs to be set up, and if a new AWS account is used, it will not be necessary to change workers. The reason for this is that when the ML services are created, they obtain the list of activities and search for the corresponding ARN by name.

With ARNs set, workers can start polling. Workers were implemented in such a way that a fixed number of threads (workers) per task are initiated when the service starts. After obtaining the correct ARN, a Poller is initiated for each worker task (Listing 7), and the specified number of "pollers" continuously poll for activities and executes its work whenever it receives one.

All workers have an environment variable to set the number of workers that must be polling. Currently, there are twenty pollers per task. Workers poll for tasks with the "GetActivity-Task" call, which returns a taskToken (a unique identifier for the scheduled task) and the tasks' input whenever a task is scheduled. The worker then uses the taskToken to report the successfulness of failure of the task ("SendTaskSuccess" or "SendTaskFailure") and to send the corresponding output. This process is illustrated in Figure 5.5.

The number of pollers can be increased up to 1000, the maximum activity pollers per Amazon Resource Name (ARN) imposed by AWS, as long as the resources allocated for the modified service are adjusted accordingly.

Listing 7 – Poller interface.

```
class Poller():
    def __init__(self, client, activity_arn):
        self.activity_arn = activity_arn
        self.client = client
        self.threads = list()

    def start_pollers(self, pollers_count, exec_function, worker_name):
        for i in range(pollers_count):
            worker = worker_name + str(i)
            process = mp.Process(target=self.poll, args=(exec_function, worker,))
            process.start()

    def poll(self, exec_function, worker_name):
        while(True):
            get_activity_task = self.client.get_activity_task(
                activityArn=self.activity_arn,
                workerName=worker_name
            )

            if 'taskToken' in get_activity_task:
                activity_response = ActivityTaskResponse
                    (self.client, get_activity_task, worker_name)
                exec_function(activity_response)
```

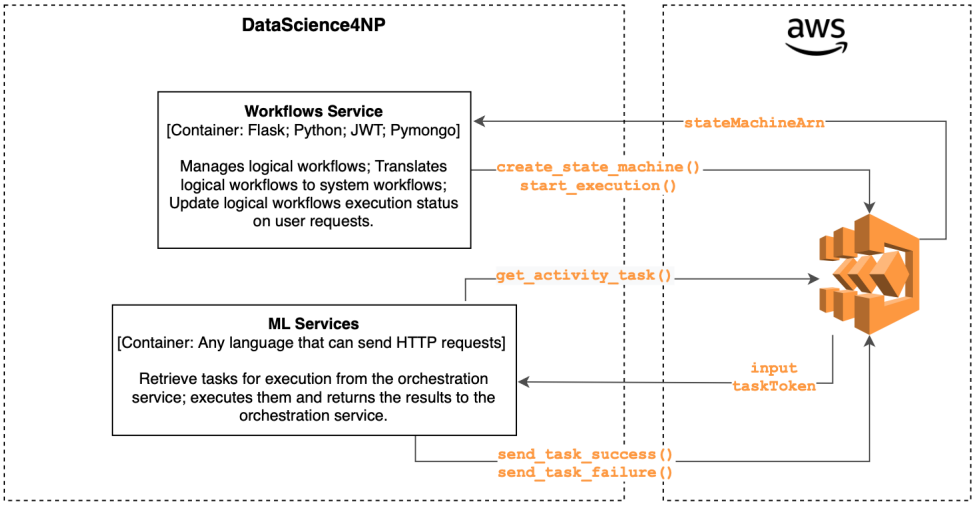

Figure 5.5 – Interaction between Step Functions and ML services.

5.3.4 Step Functions Limitations

AWS Step Functions sets quotas on several state machine parameters [27], two of them having a direct impact on our implementation. The following subsections explain these limitations, the extent of their impact on our implementation, and how they are addressed.

Maximum input or result data size for a task, state, or execution

There is a limit on the number of characters in the input/output passed between states in state machines (32,768 characters). This limitation proved out to be a bottleneck in

our state machines because each state receives input and adds their output to it, passing through the state machine bigger and bigger JSON texts.

This problem is especially observable in parallel states. A Parallel state passes to each branch a copy of its own input data, and the generated output is an array with one element for each branch (containing the output from that branch). Some of the output of these branches have duplicated data. To reduce the number of characters in the JSON text that is passed after a parallel state, two new States were introduced to our implementation: a Lambda function that receives the output of the parallel state and performs a merge of the branches and a Pass state that receives the result of the Lambda Function and passes it to the following state.

The Lambda Functions is called "Manipulator," and the Pass state is called "Pass". During translation, these states are immediately added after every parallel state.

Figures 5.6 and 5.7 demonstrates states machines with and without Lambda and Pass states and Listings 5.1 and 5.2 present the input passed to the state multiple_model_evals11 (the result of either of the parallel state or the pass state, correspondingly). This is a simple example with only one basic parallel state, but in workflows with more states and after some levels, the difference is substantial between using the Lambda function or not is significant.

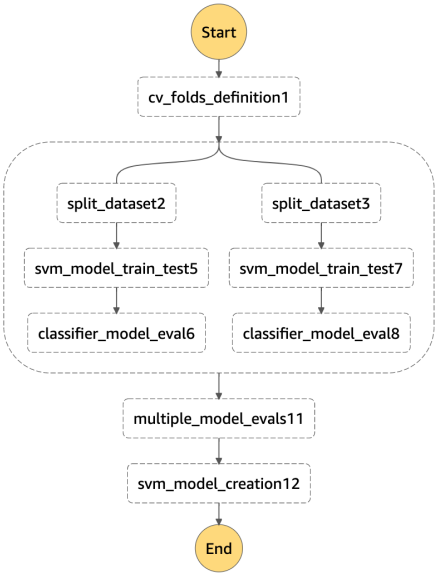

Figure 5.6 – State machine without a Lambda function and Pass state.

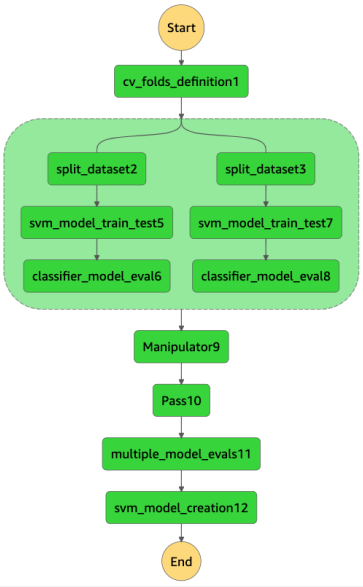

Figure 5.7 – State machine with a Lambda function and Pass state.

The Lambda function to merge the output was implemented in Python 3.8 and must be created if a new AWS account for the DataScience4NP platform is created. Its ARN must also be set directly in the Workflows Service.

This approach proved itself not to be sufficient. Hence, we proceeded to reduce the output resultant from parallel states that include multiple_model_evals tasks by setting the ResultPath of these tasks to "$", which means that multiple_model_evals states only output its result instead of adding that output to the JSON that flows between states. The Lambda function was modified to consider these states. Some other minor adjustments in the Workflows Service were also required.

This was the major issue proponent by AWS Step Functions, and the aforementioned

approach did not handle it. It was considered an unfit approach latter in the development and left no more time to considerer another one.

Paths defined for each state are defined during translation, and, as one state may need the output of another state some level above in the workflow, the ideal would be to have the JSON text with all output flowing through states, which was the reason for insisting on this approach.

The best example is the cv_folds_definition and split_dataset tasks. The first generates different training/test indices to be used while training models on different cross-validation partitions and the later breaks the instances in a dataset in training, test, and optionally validation sets. Other tasks later need these parameters. For example, model tasks need the URI to datasets with train and test sets to perform their work.

Listing 5.1 – Input of the state multiplemodelevals11 without the new states

```
[
  {
    "cv_folds_definition1": {
      "dataset_uri": "/ds4np-fs/tmpuc3yht27.csv",
      "train_idx_uri": "/ds4np-fs/tmpih_fgn9i.csv",
      "test_idx_uri": "/ds4np-fs/tmp95jcip2x.csv"
    },
    "split_dataset2": {
      "train_uri": "/ds4np-fs/tmpa4dscc08.csv",
      "validation_uri": null,
      "test_uri": "/ds4np-fs/tmpn97bo28v.csv"
    },
    "svm_model_train_test5": {
      "model_uri": "/ds4np-fs/tmp7ccgnd50",
      "results_uri": "/ds4np-fs/tmpgp6zdgs6.csv"
    },
    "classifier_model_eval6": {
      "confusion_matrix_uri":
                "/ds4np-fs/tmpd4q38up7.csv",
      "model_performance": {
        "accuracy": 1,
        "precision": 1,
        "recall": 1,
        "f-measure": 1,
        "specificity": 1
      }
    }
  },
  {
    "cv_folds_definition1": {
      "dataset_uri": "/ds4np-fs/tmpuc3yht27.csv",
      "train_idx_uri": "/ds4np-fs/tmpih_fgn9i.csv",
      "test_idx_uri": "/ds4np-fs/tmp95jcip2x.csv"
    },
    "split_dataset3": {
      "train_uri": "/ds4np-fs/tmpxsmvyc8v.csv",
      "validation_uri": null,
      "test_uri": "/ds4np-fs/tmpdo3cskyl.csv"
    },
    "svm_model_train_test7": {
      "model_uri": "/ds4np-fs/tmp0_6uy7hq",
      "results_uri": "/ds4np-fs/tmpypk9wd7x.csv"
    },
    "classifier_model_eval8": {
      "confusion_matrix_uri":
            "/ds4np-fs/tmp3oouhnn6.csv",
      "model_performance": {
        "accuracy": 1,
        "precision": 1,
        "recall": 1,
        "f-measure": 1,
        "specificity": 1
      }
    }
  }
]
```

Listing 5.2 – Input of the state multiplemodelevals11 with the new staten

```
{
  "split_dataset2": {
    "train_uri": "/ds4np-fs/tmp99avsbpg.csv",
    "validation_uri": null,
    "test_uri": "/ds4np-fs/tmpizv_fsvn.csv"
  },
  "svm_model_train_test5": {
    "model_uri": "/ds4np-fs/tmp44897rjj",
    "results_uri": "/ds4np-fs/tmpyb44hljx.csv"
  },
  "classifier_model_eval6": {
    "confusion_matrix_uri":
              "/ds4np-fs/tmp2zjfar6l.csv",
    "model_performance": {
      "accuracy": 1,
      "precision": 1,
      "recall": 1,
      "f-measure": 1,
      "specificity": 1
    }
  },
  "split_dataset3": {
    "train_uri": "/ds4np-fs/tmp9x77tnjs.csv",
    "validation_uri": null,
    "test_uri": "/ds4np-fs/tmp0_7ddbcm.csv"
  },
  "svm_model_train_test7": {
    "model_uri": "/ds4np-fs/tmp0ola0hwx",
    "results_uri": "/ds4np-fs/tmptum8pxjs.csv"
  },
  "classifier_model_eval8": {
    "confusion_matrix_uri":
            "/ds4np-fs/tmpb6ntpbhz.csv",
    "model_performance": {
      "accuracy": 1,
      "precision": 1,
      "recall": 1,
      "f-measure": 1,
      "specificity": 1
    }
  }
}
```

Maximum execution history size

"A state machine execution occurs when an AWS Step Functions state machine runs and performs its tasks." Each execution is described by its history. Figure 5.8 shows the execution history of an execution from the state machine presented in Figure 5.7.

Execution event history

ID	Type	Step	Resource	Elapsed Time (ms)
▶ 1	ExecutionStarted		-	0
▶ 2	TaskStateEntered	cv_folds_definition1	-	29
▶ 3	ActivityScheduled	cv_folds_definition1	-	29
▶ 4	ActivityStarted	cv_folds_definition1	-	74
▶ 5	ActivitySucceeded	cv_folds_definition1	-	301
▶ 6	TaskStateExited	cv_folds_definition1	-	301
▶ 7	ParallelStateEntered	dummy_fork8	-	309
▶ 8	ParallelStateStarted	dummy_fork8	-	309
▶ 9	TaskStateEntered	split_dataset2	-	410
▶ 10	ActivityScheduled	split_dataset2	-	410
▶ 11	TaskStateEntered	split_dataset3	-	419
▶ 12	ActivityScheduled	split_dataset3	-	419
▶ 13	ActivityStarted	split_dataset2	-	466
▶ 14	ActivityStarted	split_dataset3	-	480
▶ 15	ActivitySucceeded	split_dataset2	-	1457
▶ 16	TaskStateExited	split_dataset2	-	1457
▶ 17	TaskStateEntered	svm_model_train_test5	-	1465
▶ 18	ActivityScheduled	svm_model_train_test5	-	1465

Figure 5.8 – Example of an execution history in the AWS console.

Step Functions limit the number of entries in the execution history up to 25,000 events. If we examine a state of type "Task" completed with success, we can easily recognize that Task states will have at most five entries: TaskStateEntered, ActivityScheduled, ActivityStarted, ActivitySucceeded, and TaskStateExited.

Assuming that, in the worst case, each state has five entries, we can only have state machines with 5000 states. However, this limit does not cover extensive workflows that employ nested cross-validation with repetitions. The solution to this limitation is to divide state machines.

If we were running simpler workflows, maybe it would be possible to draw a strategy to split them at runtime, in a similar manner as the one suggested by AWS in the tutorial "Continuing as a New Execution" [6], but instead, workflows are divided before they are sent to AWS.

The complexity of workflows created by the DataScience4NP makes it extremely hard to

[6] https://docs.aws.amazon.com/step-functions/latest/dg/tutorial-continue-new.html

outline an efficient and advantageous way to split workflows as there are many dependencies and a lot of parallel states involved. A simple strategy based on a limit of states per workflow and that considers the size of parallel states was developed.

Before sending the workflow to AWS, its size is verified. A global count keeps the number of states in the workflow. If the global count exceeds the limit at a parallel state, then the workflow is divided, and each branch results in a new workflow. This is possible because Step Functions integrates with its own API as a service integration. Here is the strategy used to divide a workflow at a parallel state:

- For each branch of the parallel state that is to divide, a new state machine is created. The definition of the new state machine matches the definition of the branch.
- A new state that replaces the branch in the original workflow is generated. This state launches an instance of the state machine created. (Listing 8).
- The branches of the original parallel state are replaced by the newly generated tasks that launch the execution of the new state machines.

Listing 8 – Service Integration with AWS Step Functions (start of a new execution).

```
"Start an execution of another SM":{
   "Type":"Task",
   "Resource":"arn:aws:states:::states:startExecution.sync",
   "Parameters": {
        "stateMachineArn": "arn:aws:states:eu-west-2:281404888786:stateMachine:
            5ef891f06d3067c27da42a24",
        "Input": input,
        "AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"},

   "Next":"NEXT_STATE"
}
```

The size of each new state machine is verified and is also divided if needed.

Listing 8 presents a state used to associate a nested workflow execution with the parent state machine. The resource must be “arn:aws:states:::states:startExecution.sync” in order to specify that the state starts an execution (of another state machine) and waits for it to complete.

Workflows are divided considering a limit of 2000 states per workflow. Even though the theoretical value is 5000, there are also other limits, such as on the HTTP content length when sending requests to AWS (1049600 bytes), and therefore, the limit of states was adjusted.

Figures E.19 and E.20 from one of the integration tests demonstrates how divided workflows look like.

Access to states status

It is required to let users be aware of the progress of the workflow execution, which is only possible if the status of individual AWS states is known. AWS Step Functions only provides a method to get the status of a state machine execution as a whole, i.e., it is only possible to tell if the state machine execution status is one of the following: “RUNNING”, “SUCCEEDED”, “FAILED”, “TIMED_OUT” or “ABORTED”. However, there is a method

to obtain the details of an execution history (mentioned in Section 5.3.4) from which we can take advantage.

The execution history has useful pieces of information. The getExecutionHistory method returns the history of a particular execution as a list of events. Each event has an id (events are numbered sequentially, starting at one), a previousEventId, a type, and additional information according to the type.

If a user is consulting a workflow that is running (on the graphical interface), a global updatet_tasks function is continuously called. This function updates logical tasks in the logical workflow accordingly to the status of states running in Step Functions. It also updates the outputs of the logical tasks if all states completed successfully or the errors, if any.

It follows the strategy used to get the execution history and to determine the states' status.

- The list of events is obtained in reversed order using the execution ARN and the method "get_execution_history".
- The list of events is parsed: "terminating" events are looked up (ActivityFailed, TaskStateExited, TaskStateAborted), and its precedent events are found using "previousEventId" and then attached to the correspondent terminating event.
- The details of each "terminating" event are parsed: for example, "TaskStateEntered" have the name of the state executed, "ActivityStarted" has the timestamp of the initiation of the state execution and "ActivitySucceeded" and "ActivityFailed" have either the state output or the error that cause the task to fail.
- At the end of the process, a dictionary with all "terminating" events, its status, output, and timestamp are returned. For example:
  `{'split_dataset1': {'status': 'COMPLETED', 'output': 'output example', timestamp: 0.91}}`

The obtained dictionary is then used when updating tasks to deliver their status to the user.

5.4 Billing Service

Independently of the cloud computing provider, there will always be computational resources that will be used and charged. Thus, we need to determine the expenses for each user and charge them without prejudice on our part.

The giants of cloud computing offer a "pay as you go" approach where users only pay for what they execute. In the DataScience4NP, all workflows created by users run in instances of our account. It is required to determine which resource is used by each user and for how long.

In Step Functions, charges are based on the number of state transitions—each step in a workflow corresponds to a state transition. On the date of 26 June 2020, the tabulated price for 1000 state transitions is $0.025 [7].

Whenever a user starts a workflow, the corresponding state machine ARN is added to a database. Each entry has a username, the executionArn, a startDate and some other parameters to manage the database.

[7] https://aws.amazon.com/step-functions/pricing/

State machines are not immediately deleted from Step Functions whenever a user tries to delete a workflow. Instead, the state machine is flagged as deleted in the database, and it is intended to delete flagged states machines only after they were charged to the user.

The Workflows Service runs a scheduled job every day to update finished state machines. It also obtains the necessary information from the execution history (this is why we get the timestamps of the events while parsing their details). Still, the output is discarded because it does not matter for billing purposes.

Briefly explained, each user has every state machine that was ever run by him/her. With the execution history, it is possible to obtain the time used by the ML services to execute the users' workflow.

To inform users of the current bill or to charge users the monthly, a sum of all executions performed in the current month is made, obtaining in this way the total amount of seconds of computational resources used. Then, to compute a user's expenses, the amount of seconds is multiplied by our tabulated value.

DataSience4NP is currently running in a Kubernetes cluster created in INCD. The cluster is composed of ten nodes (one master and nine workers) each one with the following specifications:

- cpu: 4
- memory: 8174692Ki (8.370884608 GB)
- os-image: Ubuntu 16.04.6 LTS
- kernel-version: 4.4.0-157-generic

Figure 5.9 shows the table prices for the type of instances in AWS and Google Cloud that are more similar to the ones INCD offers.

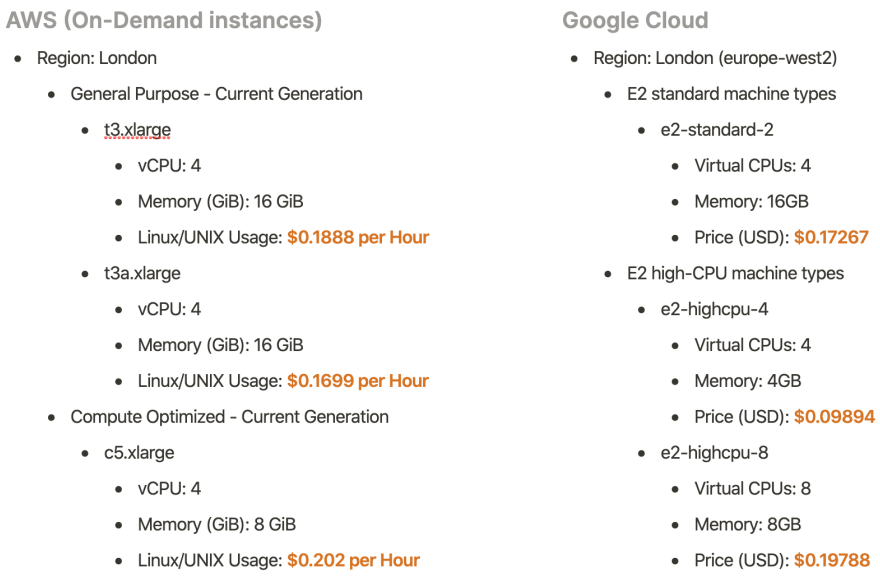

Figure 5.9 – Tabulated prices for AWS and Google Cloud instances (June 2020).

Typically, prices range between 0.17 and 0.20 dollars per hour. As a proof of concept, our price established for an hour of computation is 0.40. This is a hypothetical price at this

phase. We are interested in stipulating a value sufficient enough to cover all the resources used. Besides covering the computational time used to execute workflows, this value allows us to cover Step Functions and Lambda Functions expenses and maybe even storage.

After having the total seconds used by a user, the Billing Service calculates the total billing having our stipulated value into account.

The Billing Service also runs a scheduled job every day 2 of each month to calculate users' expenses, and it should trigger a payment request with a debit card, but due to time restrictions, there was no opportunity to implement integrations with a payment provider.

This page is intentionally left blank.

Chapter 6

Testing

This chapter describes the tests performed, its results, and the analysis. First, the usability tests already carried out by the two previous students in charge of the DataSciene4NP platform are exposed.

6.1 Integration testing

Integration testing is the most relevant type of test in the context of this thesis. It is essential to verify if the different services, especially ML services and the Workflows service, are working collectively and rightly to deliver the expected results. We can consider the Workflows Service and, consequently, AWS Step Functions, the heart of the DataScience4NP. Without it, a user can't run workflows properly.

There are 41 requirements regarding the construction of workflows, each one either states the insertion of a task or the visualization of its output. Knowing that we have many tasks and that each task may be specified with different parameters, the total number of combinations of workflows that could be created with the DataScience4NP platform grows to a point where it is no longer feasible to test all of the possibilities. We cannot thoroughly check every path in the construction of a workflow, but we can create broad integration tests to test workflows that go through all services at least once. This strategy is, somehow, the same as used in equivalence partitioning [28]. In the equivalence partitioning technique, a set of test conditions are divided into groups that can be considered the same, and it assumes that all the conditions in one partition are handled in the same way. Thus it is only needed to test one condition from each partition.

Integration tests were firstly stated: each test has an identifier, a description of the workflow used, and the use cases addressed. Then, the workflows of each test were created and started in the graphical interface. AWS has a console where it is possible to see state machines running and the inputs/outputs of the individual states. It is also possible to know the cause of error if a state fails. The results of the performed tests were observable in the AWS Step Functions console.

Tests conditions: DataScience4NP deployed with one replica per service. There are 10 pollers per each worker task.

Tests results: The result of a test can either be "PASSED", meaning that the workflow was correctly created, completed with success, and the output was presented in the graphical interface or "FAILED" meaning that its execution was not completed or the output was not presented.

The details of each test present the time it took for the workflow to complete, a reference image to the visual representation of the workflow, or other information about the test such as the reason of failure.

Test Identifier	IT_01
Workflow	dataset_input (weather.csv) use_entire_data one_hot_attribute_encoding (ignored 2 attr)
Use cases addressed	TKS_01, TKS_06, TKS_07, TKS_35
Result	PASSED
Details	1 second (Figure E.1)

Test Identifier	IT_02
Workflow	dataset_input (breast-cancer.csv) use_entire_data one_hot_attribute_encoding (1 attr encoded) info_gain (attr to select: 2)
Use cases addressed	TKS_01, TKS_06, TKS_07, TKS_13, TKS_14, TKS_35
Result	PASSED
Details	2 seconds (Figure E.2)

Test Identifier	IT_03
Workflow	dataset_input (breast-cancer.csv) use_entire_data one_hot_attribute_encoding (1 attr encoded) info_gain (threshold=0) z-score normalization (feature_standardization)
Use cases addressed	TKS_01, TKS_06, TKS_07, TKS_08, TKS_09, TKS_13, TKS_14, TKS_35
Result	PASSED
Details	3 seconds (Figure E.4)

Test Identifier	IT_04
Workflow	dataset_input (breast-cancer.csv) use_entire_data one_hot_attribute_encoding (1 attr ignored) info_gain (threshold=0) z-score normalization (feature_standardization) decision_tree (CART) classifier_model_eval
Use cases addressed	TKS_01, TKS_06, TKS_07, TKS_08, TKS_09, TKS_13, TKS_14, TKS_20, TKS_21, TKS_31, TKS_32, TKS_35, TKS_36, TKS_37
Result	PASSED
Details	8 seconds (Figure E.5)

Test Identifier	IT_05
Workflow	dataset_input (iris.csv) train_test svm (rbf) classifier_model_eval
Use cases addressed	TKS_18, TKS_19, TKS_31, TKS_32, TKS_36, TKS_37
Result	PASSED
Details	2 seconds (Figure E.6)

Test Identifier	IT_06
Workflow	dataset_input (iris.csv) train_test feature_scaling (min-max scaling) svm (poly) classifier_model_eval
Use cases addressed	TKS_02, TKS_03, TKS_18, TKS_19, TKS_31, TKS_32, TKS_36, TKS_37
Result	PASSED
Details	2 seconds (Figure E.7)

Test Identifier	IT_07
Workflow	dataset_input (iris.csv) train_test feature_scaling (min-max scaling) svm (sigmoid; C=1,2) classifier_model_eval
Use cases addressed	TKS_02, TKS_03, TKS_18, TKS_19, TKS_31, TKS_32, TKS_36, TKS_37, TKS_38, TKS_39
Result	PASSED
Details	3 seconds (Figure E.8)

Test Identifier	IT_08
Workflow	dataset_input (iris.csv) train_test relieff feature_scaling (min-max scaling) svm (linear; C=1,2) classifier_model_eval
Use cases addressed	TKS_02, TKS_03, TKS_11, TKS_12, TKS_18, TKS_19, TKS_31, TKS_32, TKS_36, TKS_37, TKS_38, TKS_39, TKS_40, TKS_41
Result	PASSED
Details	4 seconds (Figure E.9)

Test Identifier	IT_09
Workflow	dataset_input (iris.csv) train_test relieff (attr to select: 2,4) feature_scaling (min-max scaling) svm (linear; C=1,2) classifier_model_eval
Use cases addressed	TKS_02, TKS_03, TKS_11, TKS_12, TKS_18, TKS_19, TKS_31, TKS_32, TKS_36, TKS_37, TKS_38, TKS_39, TKS_40, TKS_41
Result	PASSED (Figure E.10)
Details	21 seconds

Test Identifier	IT_10
Workflow	dataset_input (iris.csv) train_validation_test z-score normalization (feature_standardization) svm (rbf, poly, sigmoid) classifier_model_eval
Use cases addressed	TKS_04, TKS_05, TKS_18, TKS_19, TKS_33, TKS_34, TKS_36, TKS_37, TKS_38, TKS_39, TKS_40, TKS_41
Result	PASSED (Figure E.11)
Details	5 seconds

Test Identifier	IT_11
Workflow	dataset_input (iris.csv) train_validation_test z-score normalization (feature_standardization) info_gain (attr to select: 1,2,3,4) svm (rbf, poly, sigmoid) classifier_model_eval
Use cases addressed	TKS_04, TKS_05, TKS_13, TKS_14, TKS_18, TKS_19, TKS_33, TKS_34, TKS_36, TKS_37, TKS_38, TKS_39, TKS_40, TKS_41
Result	PASSED (Figure E.12)
Details	12 seconds (Figure E.12)

Test Identifier	IT_12
Workflow	dataset_input (iris.csv) train_validation_test z-score normalization (feature_standardization) relieff (attr to select: 1,2,3,4) nearest_neighbors classifier_model_eval
Use cases addressed	TKS_04, TKS_05, TKS_11, TKS_12, TKS_21, TKS_22, TKS_33, TKS_34, TKS_36, TKS_37, TKS_38, TKS_39, TKS_40, TKS_41
Result	PASSED(Figure E.13)
Details	7 seconds

Test Identifier	IT_13
Workflow	dataset_input (iris.csv) train_validation_test z-score normalization (feature_standardization) relieff (attr to select: 1,2) nearest_neighbors (K=1,2,3,4,5) classifier_model_eval
Use cases addressed	TKS_04, TKS_05, TKS_11, TKS_12, TKS_21, TKS_22, TKS_33, TKS_34, TKS_36, TKS_37, TKS_38, TKS_39, TKS_40, TKS_41
Result	PASSED
Details	11 seconds (Figure E.14)

Test Identifier	IT_14
Workflow	dataset_input (iris.csv) k_fold_cross_validation feature_scaling(min-max scaling) relieff (attr to select: 2) nearest_neighbors classifier_model_eval
Use cases addressed	TKS_02, TKS_03, TKS_11, TKS_12, TKS_22, TKS_23, TKS_29, TKS_30, TKS_38, TKS_39, TKS_40, TKS_41
Result	PASSED
Details	7 seconds (Figure E.15)

Test Identifier	IT_15
Workflow	dataset_input (iris.csv) k_fold_cross_validation (K=10) feature_scaling(min-max scaling) relieff (attr to select: 2) gaussian_naive_bayes classifier_model_eval
Use cases addressed	TKS_02, TKS_03, TKS_11, TKS_12, TKS_22, TKS_23, TKS_29, TKS_30, TKS_38, TKS_39, TKS_40, TKS_41
Result	PASSED
Details	14 seconds (Figure E.16)

Test Identifier	IT_16
Workflow	dataset_input (iris.csv) k_fold_cross_validation (K=2, repeat=2) feature_scaling(min-max scaling) relieff (attr to select: 2) gaussian_naive_bayes classifier_model_eval
Use cases addressed	TKS_02, TKS_03, TKS_11, TKS_12, TKS_24, TKS_25, TKS_29, TKS_30, TKS_38, TKS_39, TKS_40, TKS_41
Result	PASSED
Details	8 seconds (Figure E.17)

Test Identifier	IT_17
Workflow	dataset_input (iris.csv) k_fold_cross_validation (K=2, repeat=2) feature_scaling(min-max scaling) relieff (attr to select: 2,4) multinomial_naive_bayes classifier_model_eval
Use cases addressed	TKS_02, TKS_03, TKS_11, TKS_12, TKS_26, TKS_27, TKS_29, TKS_30, TKS_38, TKS_39, TKS_40, TKS_41
Result	PASSED
Details	12 seconds (Figure E.18)

Test Identifier	IT_18
Workflow	dataset_input (iris.csv) k_fold_cross_validation (K=2, repeat=2, nested = True) feature_scaling(min-max scaling) relieff (attr to select: 2,4) svm (poly=1,2) classifier_model_eval
Use cases addressed	TKS_02, TKS_03, TKS_11, TKS_12, TKS_18, TKS_19, TKS_38, TKS_39, TKS_40, TKS_41
Result	FAILED
Details	Returned a result with a size exceeding the maximum number of characters service limit.

Test Identifier	IT_19
Workflow	dataset_input (iris.csv) k_fold_cross_validation (K=10, repeat=2) feature_scaling(min-max scaling) relieff (attr to select: 2,4) svm (poly=1,2) classifier_model_eval
Use cases addressed	TKS_02, TKS_03, TKS_11, TKS_12, TKS_18, TKS_19, TKS_38, TKS_39, TKS_40, TKS_41
Result	FAILED
Details	Returned a result with a size exceeding the maximum number of characters service limit.

Test Identifier	IT_20
Workflow	dataset_input (iris.csv) k_fold_cross_validation (K=10, repeat=10, nested=True) feature_scaling(min-max scaling) svm (C=1,2) classifier_model_eval
Use cases addressed	TKS_11, TKS_12, TKS_18, TKS_19, TKS_38, TKS_39, TKS_40, TKS_41
Result	FAILED
Details	Returned a result with a size exceeding the maximum number of characters service limit. The division of the workflow is correctly performed as we can confirm in Figure E.19. This figure shows the main workflow that unfolds to ten others (one for each repetition). Figure E.20 is one of the workflows within the main one and it also unfolds into another 10 workflows (one for each fold).

6.2 Quality Testing

Quality attributes were presented in Section 3.2 as quality attribute scenarios, which allow us to assess if the quality attributes are being fulfilled. The following tables present the tests performed. Tests are identified by `QT_<number>`, and the corresponding scenario is specified.

Quality attributes regarding usability (QAS_06 to QAS_10) were not performed.

Test Identifier	QT_01
Quality Attribute Scenario	QAS_01
Details	All system services must have well-defined responsibilities. The architecture of the platform proves that each service is responsible and has everything needed to perform well-defined tasks. Services can also contact other services through technology-agnostic protocols if needed. The insertion of the Billing Service serves as a test. The Billing Service was developed, registered, and inserted independently in the platform during development. The normal operation of the system was not compromised.
Result	Verified

Test Identifier	QT_02
Quality Attribute Scenario	QAS_02
Details	No more than three services can be changed to integrate a new ML Service into the system. This test was not performed in practice, but the insertion of a new ML service implies the creation of the corresponding AWS Activity/AWS Activities. It may be necessary to add a strategy in the Workflows Service. The new task(s) must also be inserted as a logical task (Tasks Service) to be available in the GUI and consequently used in logical workflows. It may be necessary to insert new fields in the GUI if the ones of the new service are not expected. At most, three services are changed to add a new ML Service.
Result	Verified

Test Identifier	QT_03
Quality Attribute Scenario	QAS_03
Details	The functionalities of one service change while the system is under development, but it does not require adjustments in the remaining system. This was tested during development, for example, when the Workflows Service was being modified.
Result	Verified

Test Identifier	QT_04
Quality Attribute Scenario	QAS_04
Details	New services may be able to communicate with other existing services through technology-agnostic communication protocols. All services expose a REST API that enables them to perform requests using the HTTP protocol. Services must be registered (in Traefik). This behavior can be verified whenever the system is in normal operation.
Result	Verified.

Test Identifier	QT_05
Quality Attribute Scenario	QAS_09
Details	Duration: 0:00:16.656000 0:00:24.817000 0:00:26.757000 0:00:25.357000 0:00:32.006000 0:00:35.709000 0:00:39.081000 0:00:37.297000 0:00:35.704000 0:00:37.195000
Result	Users must be able to do two times more requests than what is expected by the system during normal operation. Test IT_16 was executed in 10 simultaneous times. The duration of the last workflows is higher, which is expected since we only have ten pollers for each worker task (to serve all users). However, the number of pollers can increase, and the architecture allows each service to scale independently to process more requests without losing performance.

Test Identifier	QT_06
Quality Attribute Scenario	QAS_13, QAS_14
Details	This test concerns the ability to use system resources efficiently: when the system services are consuming memory or CPU below the average, they must release resources, and when system services are consuming memory or CPU above the average, they must be scaled.
Result	This test was not performed in practice, and the system does not provide an automated way to scale in/out. However, during development, it was possible to do so without affecting the regular operation of the system. This is possible because of the technology used: Kubernetes.

Test Identifier	QT_07
Quality Attribute Scenario	QAS_15, QAS_16
Details	Datasets must only be accessed by the user who inserts them, and access to the platform must only be given to registered users and with the correct credentials.
Result	The security of the system was confirmed during development.

6.3 Usability Testing

Usability tests intend to verify how usable an interface is for its users. Usability concerns the ease and intuitiveness of performing any tasks in the interface.The usability of the DataScience4NP platform is an important aspect since the sequential way of building workflows is an advantage over other similar systems.

Bruno Lopes created the graphical interface, and usability tests were conducted at the time by him. Since the graphical interface did not change, it is unnecessary to repeat usability tests. Therefore, it follows a sum up of the performed tests and their results.

These usability tests were conducted on two types of users: users with no experience with ML systems and no knowledge in ML or programming languages (type A, 7 users), and users with experience in ML systems (mainly Orange), expertise users in ML but without programming skills (type B, 11 users).

Figure 6.1 presents the questionnaire results. In general, the results acquired from type A users were lower than the ones from type B, which shows that users with no experience in ML (type A) had more difficulty using the interface.

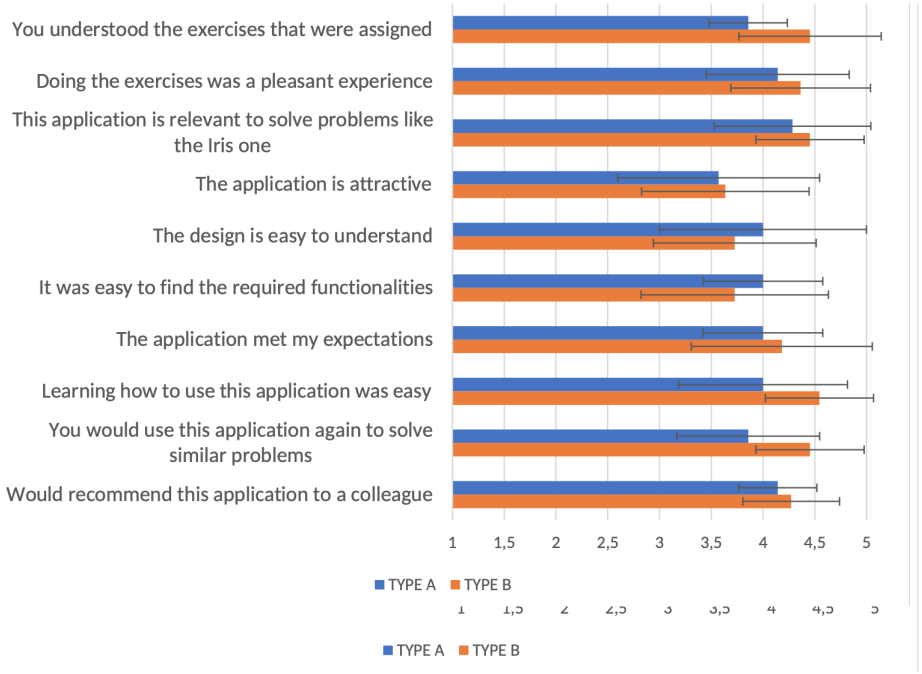

Figure 6.1 – Usability tests (performed by Bruno Lopes) results: average and standard deviation of the participants' responses.

The results were satisfactory with positive feedback and without critiques related to what was aimed to achieve.

This page is intentionally left blank.

Chapter 7

Project Management

Project management is the process concerned with the successful conclusion of a project. Success in the context of project management means that a project is completed within its time and cost constraints while meeting all the previously outlined requirements.

The present chapter uncovers all the details related to project management applied to the dissertation "Development of an Orchestration Engine for the DS4NP Platform", more specifically the methodology employed, a risk assessment, and the work plan.

7.1 Methodology

Some projects do not have clearly defined goals and/or requirements at the begging. Fortunately, there are strategies to address these types of projects.

Even though we have the specific objective of improving the orchestration solution to allow the execution of complex workflows, the decision on how to do it is almost exploratory.

Managing change is part of project work, and given the nature of this dissertation and the cutting edge technologies to be employed, requirements can change quite fast.

We need to adopt an adequate lifecycle that will allow constant monitoring due to the uncertainty associated with requirements, the unclear know-how to adopt some technologies, and the unfamiliarity related to them.

Agility will be the key to our project. "Agility is the ability to create and respond to change" [29]. Thus, we choose Scrum and will adapt it to our case. Scrum is described as a framework that allows iterative and incremental development by dividing the project into diverse time-boxes called Sprints. In Scrum, there is a prioritized list of features to be developed, called Product backlog. At the begging of each Sprint (usually two weeks), a Sprint Meeting Planning takes place to select high priority items from the backlog which the developers are committed to deliver by the end of that Sprint. This list of items for the Sprint is called Sprint backlog.

This is somehow the methodology used in the first semester. Every two weeks, an informal meeting was held with my two advisors, professor Dr. Filipe Araújo and professor Dr. Rui Pedro Paiva, where we decided that tasks be executed until the next meeting. Multiple setbacks forced us to change, in the middle of the first semester, the meetings that took place every two weeks to weekly meetings.

In the second semester, we will continue with this approach and considering requirements as the initial Product backlog and trying to fit Sprints into the envisioned plan (Figure

7.3). Scrum will enable us to accommodate change by arranging requirements and tasks according to our needs at the beginning of each Sprint.

7.2 Risk Assessment

During the development of software projects, inevitable setbacks may arise. Early identification of possible risks allows us to draft a mitigation plan and thus guarantee that even with potential setbacks, the project moves in the right direction.

Risks are classified in different ways:

- **Impact**
  (1) marginal
  (2) critical
  (3) catastrophic

- **Likehood**
  (1) low
  (2) medium
  (3) high

It follows a brief description of the identified risks and how they can occur during the project. Table 7.1 presents the correspondent mitigation strategy for each risk, its impact, and its likelihood of happening.

R_01: Lack of understanding of the initial prototype
It is required some knowledge about the core concepts underlying the DataScience4NP, such as microservices, containerization, and cloud environments, to understand how the initial prototype works. For this reason, and also due to the complexity of the solution (code structure and extensiveness), the learning curve can be steep.

R_02: Unfamiliarity with the necessary tools
To develop the DataScience4NP it is necessary to know, for example, how Docker containers works, and how deploy a Kubernetes cluster in the cloud. The initial familiarization with these technologies can be time consuming.

R_03: Overly generous time estimates
The uncertainty about the time effort of each task can lead to over-optimistic estimations regarding the time needed to perform them.

R_04: Unavailable resources
There is a minimum of 16GB of RAM needed to run the DataScience4NP platform. With the virtualization platform from the Department of Informatics Engineering[1] being deprecated, it can be hard to find a good and possible alternative to deploy and run the platform.

R_05: Application for AWS Cloud Credits for Research denied
In order to develop a AWS-enabled version of the DataScience4NP, we applied for AWS Cloud Credits for Research[2]. However, AWS can deny our proposal leaving us with no resources to develop our cloud platform.

[1] https://cloud.dei.uc.pt
[2] https://aws.amazon.com/pt/research-credits/

ID	Mitigation Strategy	Impact	Likehood
R_01	Perform code reviews; Begin by focusing on small, localized sections of the code; Build a mind map of the components; Build code dependency graph; Contact the former MSc student Artur Pedroso in case of any doubt.	2	2
R_02	Enroll in online courses; Follow tutorials; Acquire help from people who are already familiar with the technologies.	1	2
R_03	Manage expectations by updating time estimates/-work plan at the beginning of each Sprint.	2	2
R_04	Use free credits offered by cloud providers.	2	1
R_05	Design and build the solution restraining to the limits of AWS Free Tier[3]; Develop and test AWS Step Functions Workflows locally[4].	2	1

Table 7.1 – Mitigation Strategy.

7.3 Work Plan

Figure 7.1 outlines the initial plan for the first semester and Figure 7.2 show how the first semester actually was proceeded. These plans are much different mainly because of some misfortunes that happened during the semester.

There is a high complexity associated with the take over of an existing software project and it is crucial to have comprehensive documentation to understand the structure of the project and how it is implemented.

Access to the source code repository and some documentation were provided, as this dissertation is a further work of the project DataSience4NP. One essential step to be executed at the begging of the project was the deployment of the system. Given the recent understanding of the system and the lack of familiarity with the technologies used, it would be a challenge to deploy the whole system solely based on the provided setup guidelines. Therefore, after an initial analysis of how the system works, the former MCs student Artur Pedroso did a walkthrough on how to set up access to the DEI cloud, how to create new instances, and install a Kubernetes cluster. Even with the know-how, deploy the system was a hindrance task because the DEI cloud is becoming obsolete, and it will be discontinued. We then try to deploy the system in GKE and to use only the Netflix Conductor to perform some tests.

Figure 7.3 shows the envisioned plan for the second semester. The plan started with the reformulation of the Translator (Workflows Service), so Step Functions can understand the translated workflows (workflows in JSON using the declarative Amazon States Language). Then, the implementation of Step Functions would follow. After both Workflows and Step Functions were up to use, the translation of workflows would be verified to ensure it was correctly performed and if Step Functions were correctly invoked. It would follow the migration to AWS, and the billing system. The end of the semester was reserved for testing and writing both the dissertation and the scientific article.

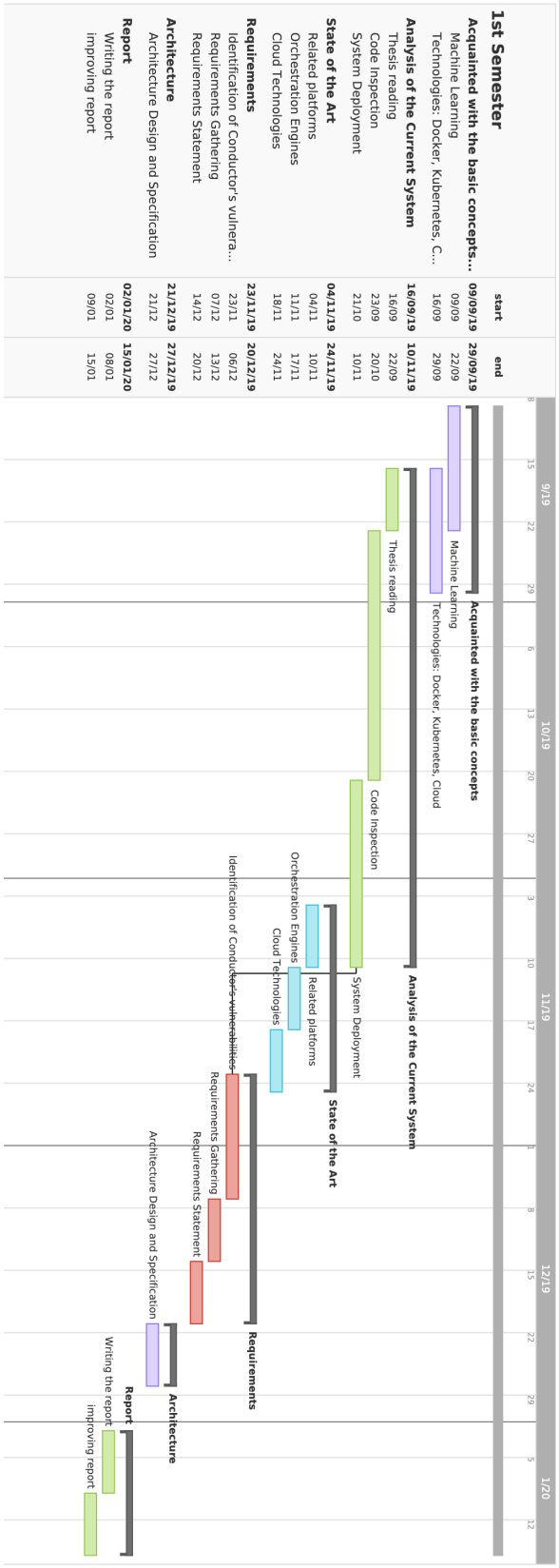

Figure 7.1 – First Semester - planned tasks.

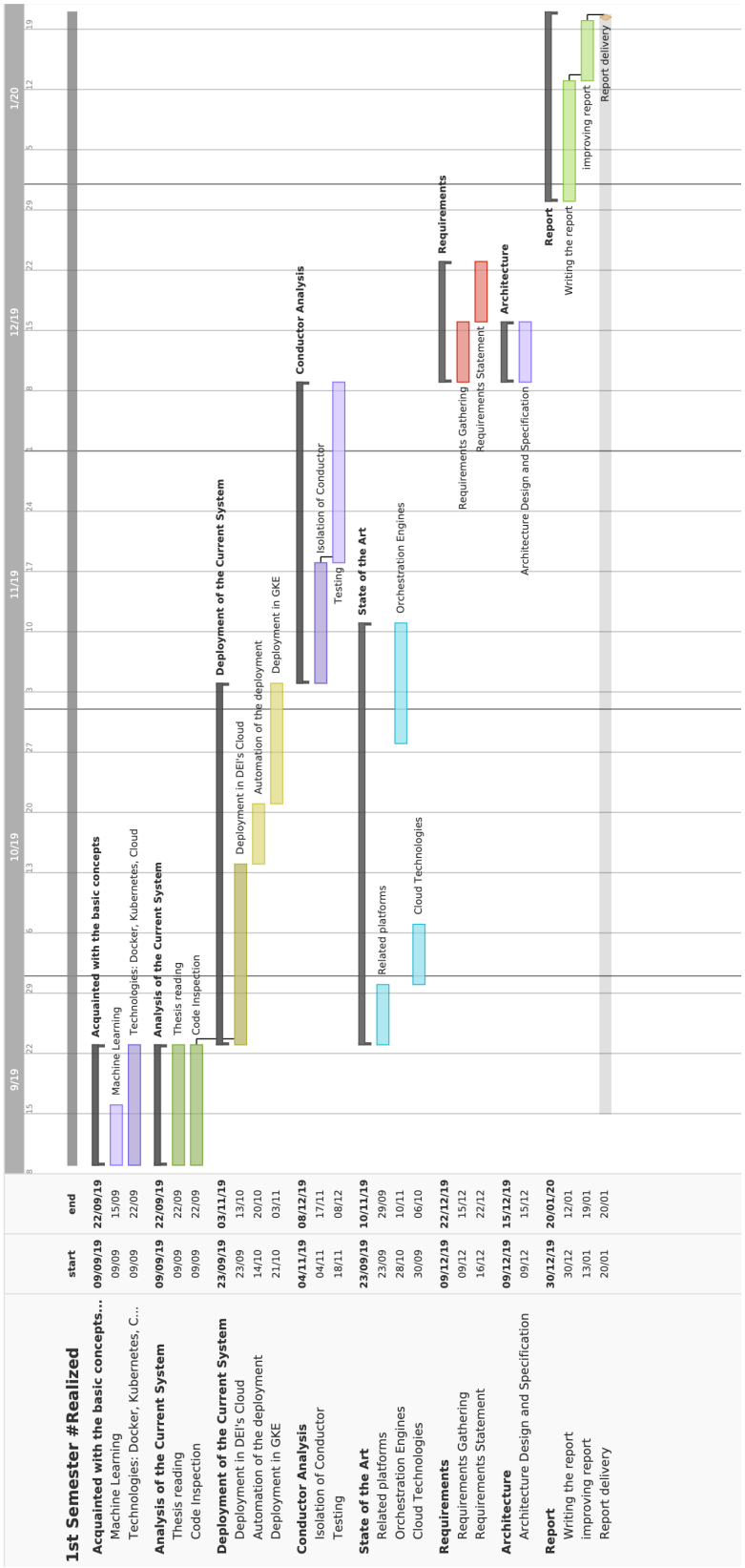

Figure 7.2 – First Semester - tasks accomplished.

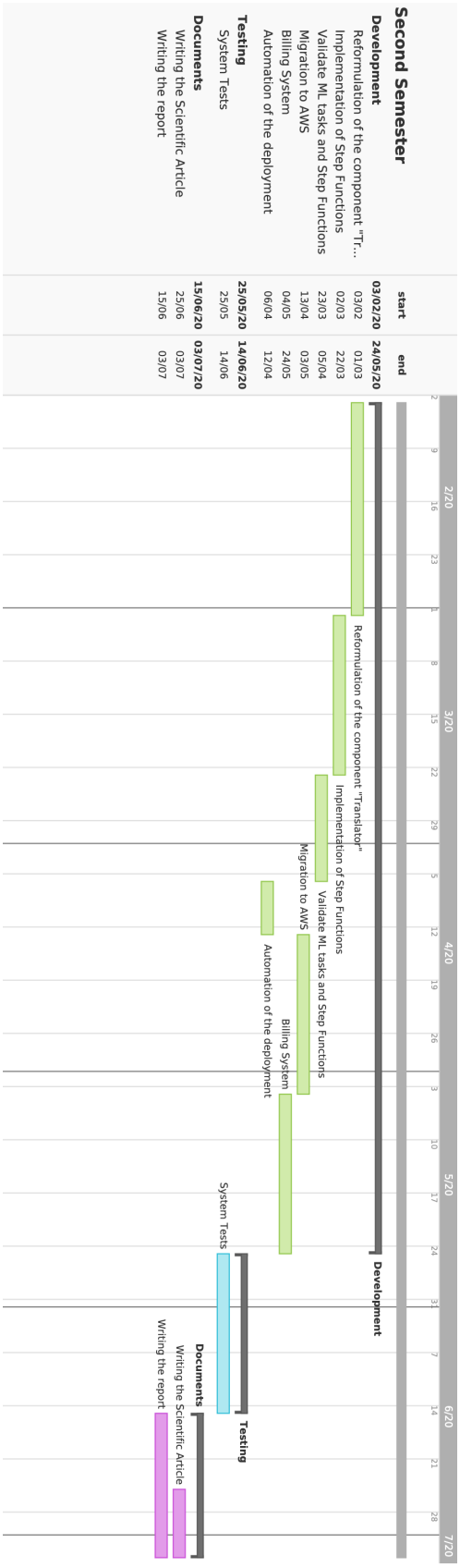

Figure 7.3 – Second Semester - planned tasks.

The real work plan was quite different (Figure 7.4). Through the first weeks of February, the Workflows Service was set up locally in order to familiarize with the code with code inspections and by debugging the service. The first changes to modify the service were made, namely the creation of the Step Functions objects to be used during translation. By the end of March was when the access to the INCD was provided and when the deployment of the platform started. The deployment was a problematic task as there were a lot of errors raised during the process. Only by the end of April, the platform was fully deployed. Once the platform was available, the workers were changed. Adjustments to the Workflows Service and Step Functions implementations were continuously made throughout the whole time, as there was always a new limitation or things to correct.

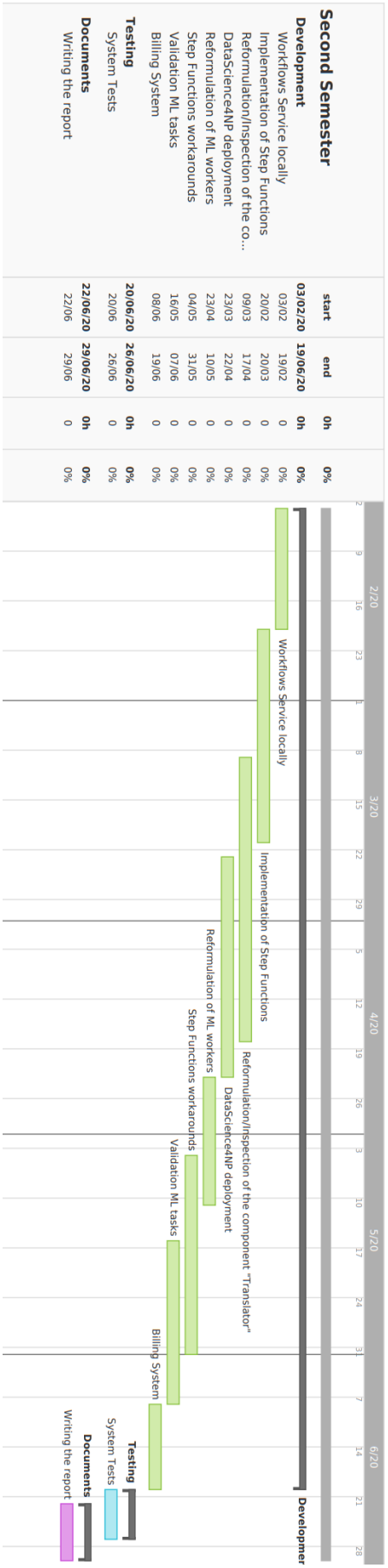

Figure 7.4 – Second Semester - planned tasks.

This page is intentionally left blank.

Chapter 8

Conclusion

This thesis was executed as a continuous work in the DataScience4NP project, previously started two former students of the Department of Informatics Engineering. The result of their theses was a platform designed in a microservices architecture to be deployed in a Kubernetes cluster. The platform allows the creation of sequential ML workflows composed of several tasks from the preprocessing, learning, and evaluation phases, where various microservices execute portions of a workflow.

However, the resulting platform had a bottleneck in its most critical component: the service responsible for the orchestration of the Machine Learning workflows. The orchestration service, Workflows Service, was initially developed using Netflix Conductor, which turn out to be an unfit solution. With Netflix Conductor, it was not possible to execute extensive workflows, in particular, the ones involving nested cross-validation with repetitions. This inability was proved by the tests performed during the first phase of this thesis. It was also not permissible to have two simultaneous users using the platform. Microservices that execute ML tasks were also not prepared to handle thousands of tasks with acceptable performance.

Thus, it was settled that the orchestration service would change from Netflix Conductor to AWS Step Functions. The introduction of AWS Step Functions in the DataScience4NP enabled that multiple users use the platform at the same time and allows the execution of various simultaneous workflows, which was not possible before.

A strategy to divide workflows before they are sent to AWS Step Functions was implemented. By verifying the size of state machines and dividing then into other state machines that correctly compose the intended workflow allows the creation of extensive workflows.

ML services were changed to start communicating with AWS Step Functions, and a poller mechanism was implemented, which grants a good performance in the execution of workflows as workers are continuously polling for tasks.

The Billing Service is a new service in the system, and it is still in an early phase. It is a service capable of determining and reporting to users their monthly bill. A users' expense is based on the time ML services spent to execute their workflows.

In this thesis, we aimed at the execution of complex workflows, which was partially achieved. The impact of the AWS limitation on the maximum input or result data size for a task was discovered, and a workaround was developed. Only when performing integration tests, when there was no more time left for development, it was realized that the workaround was far from being enough. However, ML services efficiently work with AWS and provide a satisfactory execution of workflows with good performance.

Moreover, the division of complex workflows is done correctly, and once the AWS limitation is overcome, everything points to the correct execution of those complex workflows. An alternative to avoid large payloads being passed as input/output would be to save data to another AWS service such as S3 Buckets or Amazon DynamoDB.

This page is intentionally left blank.

Bibliography

[1] Data never sleeps 7.0.
https://www.domo.com/learn/data-never-sleeps-7. [Online; Accessed: 6 Jun, 2020].

[2] Building machine learning microservices for the data science for non-programmers platform.
https://osf.io/g6s2v/.

[3] Omonowo D. Momoh Matthew N.O. Sadiku, Sarhan M. Musa. *Cloud computing: Opportunities and challenges*. IEEE, 2014.

[4] Ileana Castrillo Derrick Rountree. *The Basics of Cloud Computing - Understanding the Fundamentals of Cloud Computing in Theory and Practice*. Syngress, 2013.

[5] Gartner glossary.
https://www.gartner.com/en/information-technology/glossary/big-data.
[Online; Accessed: Dez 28, 2019].

[6] Wenming Qiu Uchechukwu Awada Keqiu Li Changqing Ji, Yu Li. *Big Data Processing in Cloud Computing Environments*. IEEE, 2012.

[7] Magic quadrant for cloud infrastructure as a service, worldwide.
https://www.gartner.com/doc/reprints?id=1-1CMAPXNO&ct=190709&st=sb. [Online; Accessed: Dez 29, 2019].

[8] Sam Newman. *Building Microservices, Designing Fine-Grained Systems*. O'REILLY, 2015.

[9] Shu Tao Hui Kang, Michael Le. *Container and Microservice Driven Design for Cloud Infrastructure DevOps*. IEEE, 2016.

[10] H2o.
https://www.h2o.ai. [Online; Accessed: Nov 11, 2019].

[11] Gartner. *Magic Quadrant for Data Science and Machine Learning Platforms*. Gartner, 2019.

[12] H2o documentation.
http://docs.h2o.ai/h2o/latest-stable/h2o-docs/index.html. [Online; Accessed: Nov 11, 2019].

[13] Azure machine learning.
https://studio.azureml.net/. [Online; Accessed: Oct 15, 2019].

[14] P. Vid L. Nada R.S. Marko K. Janez, O. Roman. *ClowdFlows: Online workflows for distributed big data mining*. Elsevier, 2016.

[15] LafuenteManuel A. Mazzara M. Montesi F. Mustafin R. Safina L. Dragoni N., Giallorenzo S. *Microservices: yesterday, today, and tomorrow*. Springer, 2017.

[16] Pethuru Raj Neha Singhal1, Usha Sakthivel1. *Selection Mechanism of Micro-Services Orchestration VS. Choreography*. IJWesT, 2019.

[17] Netflix conductor.
https://netflix.github.io/conductor/. [Online; Accessed: Dez 12, 2019].

[18] Zeebe.
https://zeebe.io/. [Online; Accessed: Nov 30, 2019].

[19] What is zeebe.
https://zeebe.io/what-is-zeebe/. [Online; Accessed: Nov 30, 2019].

[20] Cadence.
https://cadenceworkflow.io/. [Online; Accessed: Dez 12, 2019].

[21] Aws step functions - features.
https://aws.amazon.com/step-functions/features/. [Online; Accessed: Dez 29, 2019].

[22] *1233-1998 - IEEE Guide for Developing System Requirements Specifications*. IEEE.

[23] What is aws step functions?
https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html. [Online; Accessed: Jun 1, 2020].

[24] States - aws step functions.
https://docs.aws.amazon.com/step-functions/latest/dg/concepts-states.html. [Online; Accessed: Jun 1, 2020].

[25] Amazon states language.
https://states-language.net/spec.html. [Online; Accessed: Jun 8, 2020].

[26] Input and output processing in step functions.
https://docs.aws.amazon.com/step-functions/latest/dg/concepts-input-output-filtering.html. [Online; Accessed: Jun 10, 2020].

[27] Quotas - aws step functions.
https://docs.aws.amazon.com/step-functions/latest/dg/limits-overview.html. [Online; Accessed: Jun 15, 2020].

[28] What is equivalence partitioning in software testing?
http://tryqa.com/what-is-equivalence-partitioning-in-software-testing/. [Online; Accessed: Jun 20, 2020].

[29] Agile alliance.
https://www.agilealliance.org. [Online; Accessed: Jan 4, 2020].

[30] Netflix conductor issues.
https://github.com/Netflix/conductor/issues/807. [Online; Accessed: Dez 6, 2019].

This page is intentionally left blank.

Appendices

Appendix A

Netflix Conductor Tests

The orchestration solution initially implemented in the DataScience4NP platform has proved to be unsuitable for the execution of complex workflows, namely the ones involving nested cross-validation with different features and parameters.

The present dissertation aims to propose a better and more seemly solution for the communication and cooperation of microservices in the DS4NP platform. The first step to finding the bottleneck of the Netflix Conductor was to try to reproduce previously failed tests.

Tests were performed under different conditions, and for a better understanding of the results, we are going to label tests with the following terminologies:

- **Local**: the test was performed with the latest version o Conductor in a local server.

  In-Memory server running in a Macbook Pro 2016 with the following specifications:

  2.0GHz dual-core Intel Core i5;
  4MB shared L3 cache;
  256GB SSD;
  8GB of 1866MHz LPDDR3 onboard memory.

- **GKE**: the test was performed with an 11 month-old version in the Google Kubernetes Engine (Kubernetes Cluster with a total of 8 vCPUs)
- **Tasks**: specifies the number of tasks present in the workflow
- **N Types**: specifies the number of different types of tasks present in the workflow
- **N Threads**: specifies the number of threads pulling from each queue

In the first round of tests, we wanted to observe the mode of operation of the orchestrator given multiple numbers of parallel tasks. The simpler way to construct this scenario was to create workflows with a single Fork and Join and n tasks. Figure A.1 exemplifies a workflow with 10 tasks.

The next steps summarize the actions taken to execute a workflow, from a high-level perspective:

1. Register tasks (tasks must be registered before being used in a workflow);
2. Define of workflow using a JSON based DSL;

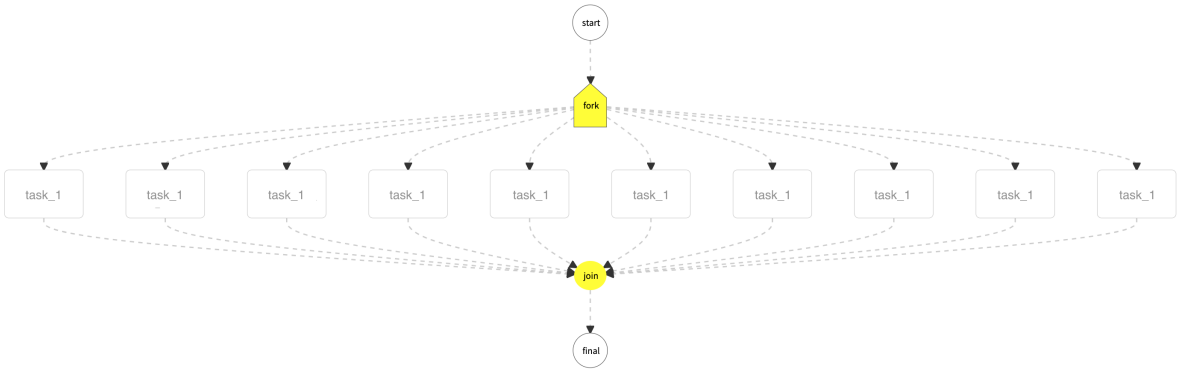

Figure A.1 – Netflix Conductor workflow with simple fork/join.

3. Start the workflow;
4. Tasks are put in the correspondent queue;
5. Workers begin the continuous polling for scheduled tasks at regular interval;
6. Workers execute its function for a task;
7. Workers update tasks' status.

Task Queues are used to schedule tasks for workers, and Netflix Conductor uses dyno-queues internally to achieve that goal.

A worker is an instance that executes the work for a task. In our case, a worker is one of the microservices. We can have multiple instances of workers polling for tasks by specifying the number of threads. For testing purposes, the only thing a worker does is to update a task status to "Completed". None of the tasks have input or output keys. All of the parameters specified for the execution of the workflows are stated below:

- retryCount: 3
- timeoutSeconds: 3600 (s)
- inputKeys: [ ]
- outputKeys: [ ]
- timeoutPolicy: TIME_OUT_WF
- retryLogic: FIXED
- retryDelaySeconds: 600 (s)
- responseTimeoutSeconds: 1200 (s)
- concurrentExecLimit: 100
- rateLimitFrequencyInSeconds: 60 (s)
- rateLimitPerFrequency: 50

Test#	GKE	Local	Tasks	N Types	N Threads	Time (ms)
1		x	10	1	1	1329
2		x	10	1	2	388
3		x	100	1	1	13240
4		x	100	2	1	12019
5		x	500	1	1	88182
6		x	1000	1	1	277671
7		x	1000	2	1	166602
8		x	1000	2	2	90027
9		x	2000	2	2	
10		x	5000	1	1	*
11		x	5000	2	1	*
12		x	5000	100	1	*
13		x	5000	100	100	*
14	x		10	10	1	221609
15	x		100	1	1	2240177
16	x		1000	1	1	22318256
17	x		1000	100	100	*

Table A.1 – Initial tests results.

Tasks marked with * were not completed.

Each type of task has its queue: the Conductor creates a queue for system tasks (default queues are HTTP, KAFKA_PUBLISH, and deciderQueue) and for each type of task registered. Once a workflow is initiated, workers can start the continuous polling of the driver server with a specified interval.

In the workflow of test 10, tasks began to exceed the established timeout. Approximately after two hours of initiate the workflow, the Decider Service started issuing warnings about tasks that were in a pending state for longer than 3600000 ms. When checking the task queues, we noticed that, at that point, not even half of the tasks were executed since 2986 of a total of 5000 tasks were still in the queue. Eventually, the local server ran low on memory, and Conductor stopped.

The workflow from test 11 was also not completed: changing the number of type of different tasks from 1 to 2 did not prevent any memory problems. After 22 minutes, each task type had only execute 243 tasks. After 26 more minutes 228 more tasks were completed.

Trying to have a more uniform distribution of tasks per queue by increasing the number of different types of tasks (test 12) did not work, and resulted in a memory problem as well.

Test number 13 attempt to have a diverse workflow with many workers polling from the tasks queues. However, after the first worker pulled the first task, requests started to exceeded the maximum number retries for various url. The same happened in test 17 that

ran in GKE (workflow with 1000 tasks - 50 tasks of each type).

The execution of workflows was much slower when tests were performed in the cloud, as stated in the results presented in Table A.1. Increasing the number of threads from 1 to 2 improved the execution time in some cases (tests 2 and 9), but having many threads (N Types * N Threads) executing resulted in memory errors.

In Conductor, if we start two workflows WF2 after WF1 with the same tasks, WF2 has to be on hold until the tasks in the first workflow are polled. Even though this is a problem in scenarios that multiple users could be running the same workflow with the same tasks, this is not the source of the bottleneck. The workflow evaluation is the central problem in Netflix Conductor: "every time a task is updated, the workflow will be evaluated. And, every time a workflow is evaluated, all of it's tasks are loaded. And with 1000's of tasks being updated in parallel, without locking, workflows are evaluated thousands of times, which could introduce a huge load on Conductor server" [30].

Appendix B

AWS Architecture

We were committed to outlining an AWS-enabled version for the DataScience4NP. As such, the architecture presented in this appendix was designed using AWS components. Amazon Web Services provide a range of services and products that give us some options to design and build our platform.

We were already using containerized services that can run and could be managed using Amazon Elastic Container Service (Amazon ECS). We could instead use Amazon Elastic Kubernetes Service (Amazon EKS) since we already have containers running on Kubernetes, but this option has a pricing-service fee per cluster. Using Amazon ECS instead of Amazon EKS provides an additional thickened layer of security because Amazon ECS supports IAM roles. Both solutions run in Amazon Elastic Compute Cloud (Amazon EC2).

In an extreme point of view, we could discard our microservices architecture by replacing our microservices with Amazon Lambda Functions. Still, Lambda Functions can come in handy for this solution if we ever need this serverless computing service to run simple tasks that may be added to the system.

The main AWS service being used is Step Functions, which would be able to orchestrate our machine learning workflows.

To better understand how we would pour our solution in AWS, we represented the architectural transition in two different diagrams (Figures B.1 and B.2). Figure B.1 presents the same general system container diagram as Figure 4.2 but with the introduction of Step Functions. Workflows Service will remain with the responsibility to translate workflows and to send them to the orchestration solution, which is now Step Functions. Step Functions will coordinate the execution of ML tasks according to the state machine (workflow) received from Workflows Service, by assigning each one to the correspondent microservice (ML Services).

In Figure B.2, we go a step further, and we make the correspondence of each container in the initial prototype to AWS components. Our services (microservices) will run in the Amazon Elastic Compute Cloud (Amazon EC2) instances, and for the database, we will be using the Amazon Relational Database Service (Amazon RDS). The AWS equivalent to the API Gateway is the Amazon Elastic Load Balancing (Amazon ELB).

Finally, in Figure B.3, we present the real structure of our solution in Amazon Web Services. We designed two scaling groups to divide the services that perform machine learning tasks (ML Services) from other services. In this way, we will be able only to scale the services that will probably have a heavy computational load.

Amazon Elastic Load Balancing (Amazon ELB) would be responsible for redirecting traffic

to Amazon EC2 instances, the same responsibility as the API Gateway in the initial prototype. It is connected only to one scaling group, the one that has the GUI, Users, Tasks, Datasets, and Workflows Services. In this way, users could access the graphical interface and build their workflows, which are then sent to Step Functions. Machine Learning tasks would be performed by the ML Services in the second scaling group.

The databases would run in the Amazon Relational Database Service (Amazon RDS). Step Functions would be the orchestration engine, and Lambda Functions could be eventually used for several purposes. We also added Amazon Simple Storage (Amazon S3), which would allow us to collect, store, and analyze data securely and at a massive scale and can maybe improve the storage and access to datasets.

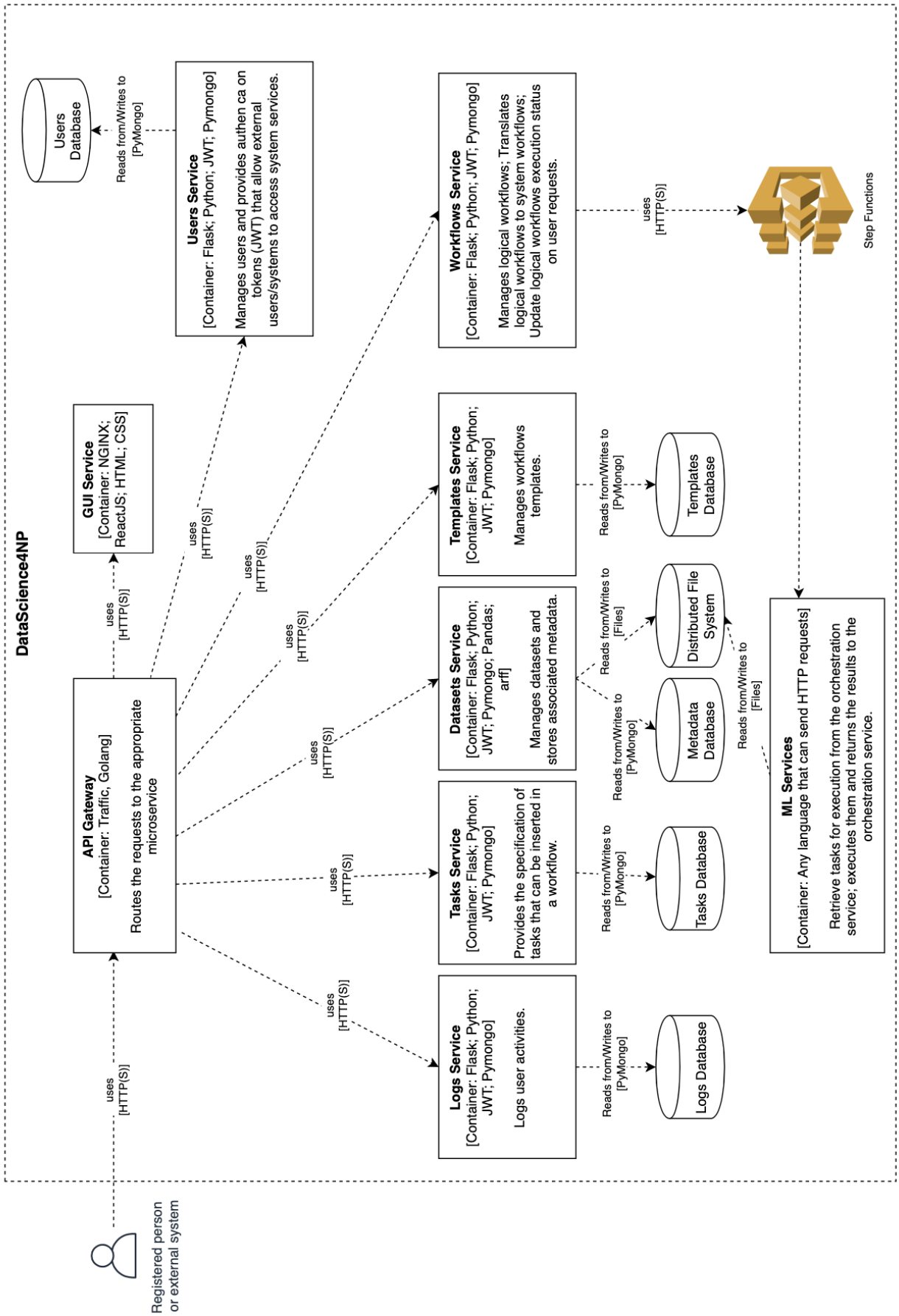

Figure B.1 – General system container diagram with step functions.

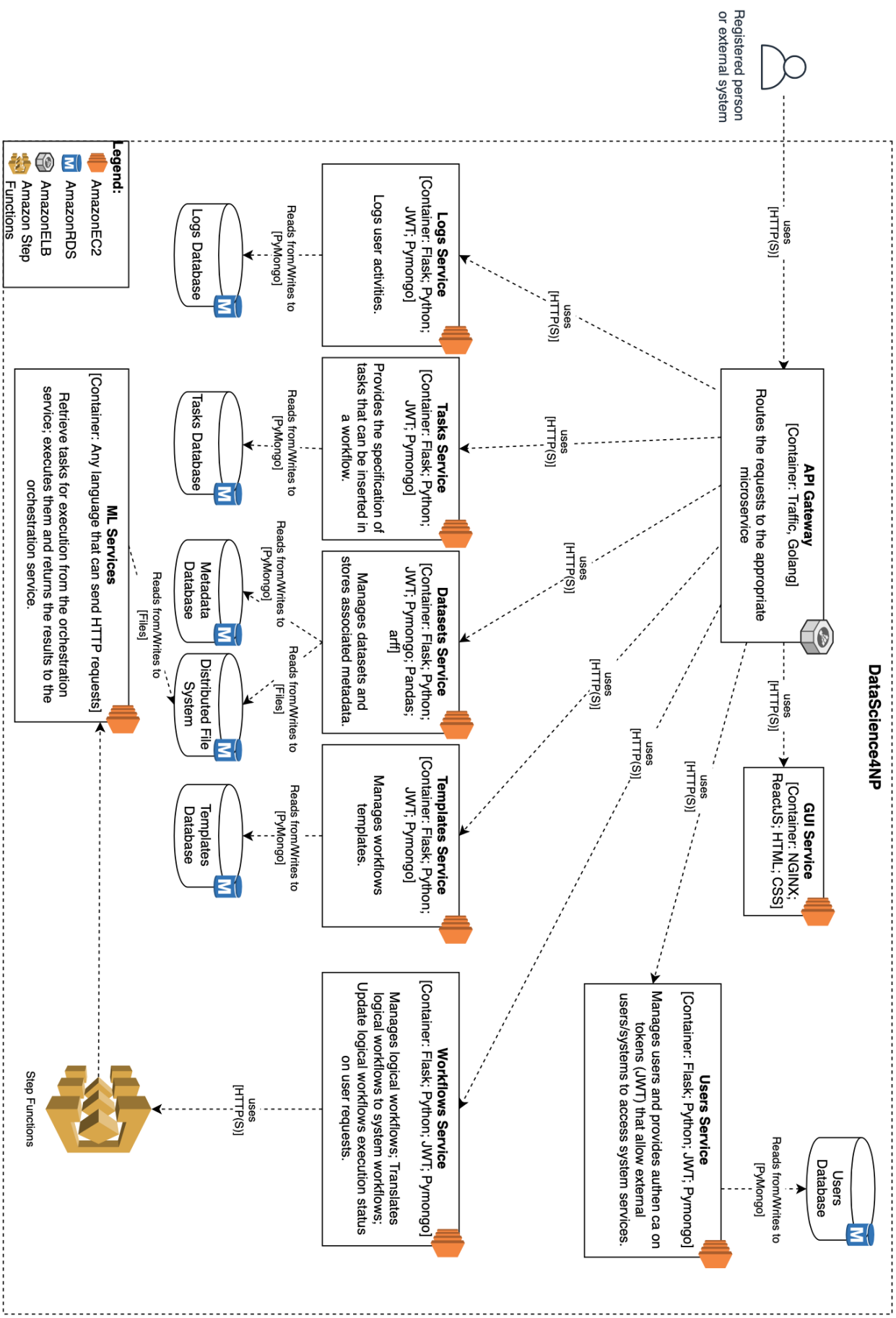

Figure B.2 – General system container diagram labelled with AWS components.

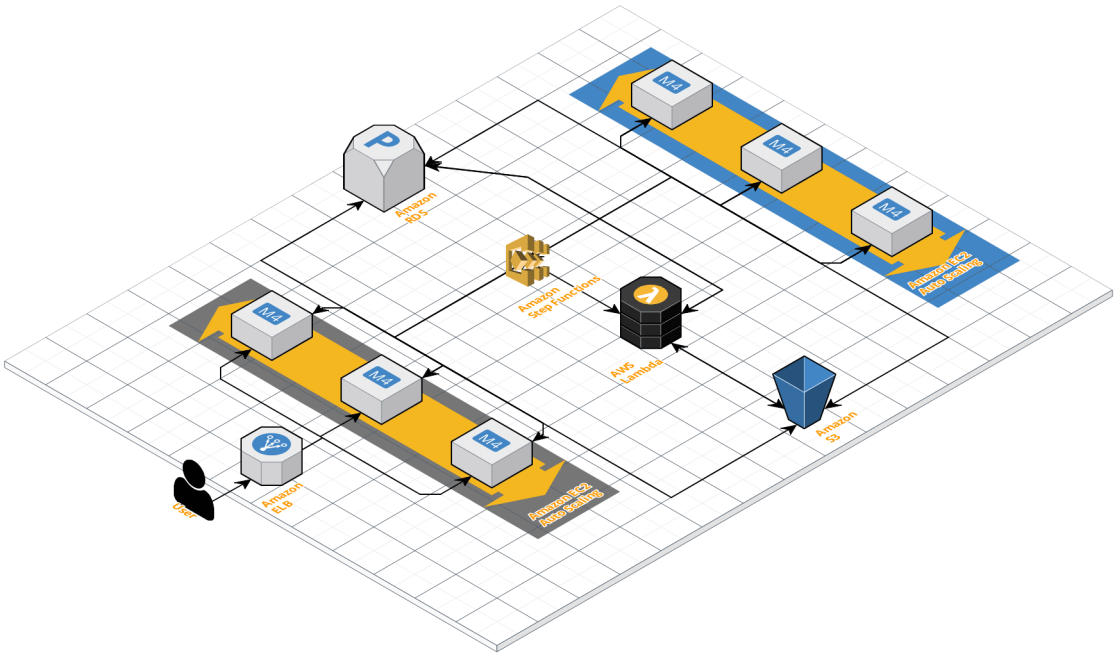

Figure B.3 – DataScience4NP Architecture.

Appendix C

Logical Workflows

Logical workflows can be defined according to a grammar depicted in EBNF notation of Listing C. This grammar is used to verify the logical workflows received by the Workflows Service. If the logical tasks preserve the rules of the grammar, Workflows Service continues with the translation process.

```
1  Start = ‘‘datasetInput’’ ValidationProcedure
2  ValidationProcedure = ((Procedure1 [(Preprocessing1 | FeatureSelection1)] ModelCreation)
3                      | (Procedure2 (Preprocessing2 | FeatureSelection2 | ModelCreation)))
4  Preprocessing1 = ‘‘aPreprocessingTask’’ [ContinueProcedure1]
5  FeatureSelection1 = ‘‘aFeatureSelectionTask’’ [ContinueProcedure1]
6  Preprocessing2 = ‘‘aPreprocessingTask’’ [ContinueProcedure2]
7  FeatureSelection2 = ‘‘aFeatureSelectionTask’’ [ContinueProcedure2]
8  ModelCreation = ‘‘aMachineLearningTask’’ ‘‘anEvalMetricTask’’
9  Procedure1 = (‘‘crossValidation’’ | ‘‘holdOut’’ | ‘‘tVT’’)
10 Procedure2 = ‘‘useEntireData’’
11 ContinueProcedure1 = (Preprocessing1 | FeatureSelection1)
12 ContinueProcedure2 = (Preprocessing2 | FeatureSelection2 | ModelCreation
```

Double-quotes represent the terminals (specific ML tasks that can be defined by the user in the workflow). Symbols between square brackets are optional, and symbols between parentheses and separated by a | represent different choices.

All workflows must start with the definition of the dataset to be used. Then, a validation procedure must be chosen. There are two different paths according to the validation procedure. If the procedure is essential for model assessment and selection (either (nested) cross-validation, hold-out, or train-validation-test), all remaining tasks will always include final tasks to create a model and assess its classification performance. If the procedure dictates that the remaining tasks to be included must use all the data (use entire data), the remaining tasks might or might not include final tasks to create a model and to assess its classification performance (always using the same data to train and to test the model).

Logical workflows are sent from the GUI to the Workflows Service in JSON. The basic structure outlined for the logical workflows can be consulted in Listing 9. Tasks are sequentially inserted in the "tasks" list as JSON objects.

Each task is identified by its logical id and its type (Table C.1).

Listing 9 – Logical workflow structure.

```
{
    "id": "workflowId",
    "name": "workflowName",
    "tasks": [
    {
        "description":"Description of the logical task",
        "id":"logicalTaskId",
        "inputParameters":[{...}, {...}., ...],
        "label":"",
        "metadata":{
            "errors":[],
            "isCompleted": False|True,
            "isRunning": False|True,
            "output":"None",
                "numTasks": 1,
                "completedTasks": 0,
                "systemTasks": {...}
            },
        "type": {
            "description":"",
            "id":"taskType",
            "label":"",
            "subsequent":[
                "name of the subsequent task",
                "...",
                ...
              ]
        }
    }, ... ],
    "user_id": "user_email",
    "errors": [],
    "isCompleted": False|True,
    "isRunning": False|True,
    "metadata": {},
    "workflowId":"arn:aws:states:eu-west-2:ABC"
    "executionArn":"arn:aws:states:eu-west-2:ABC:XYZ"
}
```

Table C.1 – Logical Tasks

Task Type	Task Id
dataset_input: defines the data source to be used in the workflow.	**dataset_input**: lets the user define the URI of the dataset to be used in the workflow, allows the elimination of attributes from the dataset and the specification of the attribute that represents the class. Output: the path to the dataset.
validation_procedure: specifies the process to be used when constructing the subsequent tasks that compose the workflow.	**use_entire_data**: defines that the tasks in the workflow should use all data. Output: none.
	train_test: defines that the tasks in the workflow will be part of a hold-out procedure. Lets the user specify the proportions to be used in the train and test sets, if the data must be stratified or randomized and what seed should be used as the random number generator for this operation. Output: the train and test sets produced in the workflow.
	train_validation_test: defines that the tasks in the workflow will be part of a train-validation-test procedure. Lets the user specify the proportions to be used in the train, validation and test sets, if the data must be stratified or randomized and what seed should be used as the random number generator for this operation. Output: the train, validation, and test sets produced in the workflow.
	k_fold_cross_validation: defines that the tasks in the workflow will be part of a cross-validation or nested cross-validation procedure. Lets the user specify the number of folds to be used, the number of repetitions, and if it should be used the nested or the normal method. It is also possible to specify if the data must be stratified or randomized and what seed should be used as the random number generator for this operation. Output: the train and test sets produced in the workflow for each fold.

Table C.2 – Logical Tasks (continuation)

Task Type	Task Id
preprocessing: applies transformations to attribute values in the dataset that being used in the workflow.	**feature_scaling**: lets users scale the dataset to min-max values specified by the user. The operations are only applied to the numerical attributes present in the dataset and never to the class. Output: the datasets produced in the workflow during train and test, the attributes used in the operation and an object to apply the scaling operation to new data (these outputs are related to the construction of the best model: in the cross-validation case, the output is presented to the folds that produce the best model; in the nested cross-validation case, only the outputs for the operations applied while building the final best model are presented).
	feature_standardization: lets the user apply z-score normalization in the dataset being used. It applies z-score normalization to all numerical attributes, except for the class attribute. Output: the datasets produced in the workflow to train and test the best model, the attributes used in the operation and an object to apply the normalization operation to new data (the outputs for the different procedures follow the same rules presented in the previous task).
	one_hot_attribute_encoding: lets the user convert feature values present in the dataset to a binary representation. Each attribute to encode will give origin to a new binary attribute that will be 1 in the rows where the value was previously present and 0 in the other rows. The user can specify the attributes to be included or excluded from the operation. Output: the names of the attributes included in the transformed dataset and the transformed datasets used to train and test the best model (the outputs for cross-validation and nested cross-validation procedures follow the same logic of the previous two tasks).

Table C.3 – Logical Tasks (continuation)

Task Type	Task Id
feature_selection: applies feature selection operations to a dataset being processed in the workflow.	**relieff**: lets the user assess the most relevant features in the dataset being processed. The Relieff algorithm is used with inputs useful for it, which are defined by the user. The user can also specify if s/he wants a threshold to be applied after running the algorithm to discard the attributes that score below such threshold. Besides these inputs, it is also possible to specify several numbers of attributes to select separated by commas (when this parameter is set with several numbers of attributes, the model being built in the workflow will be created with the different numbers of best features according to Relieff to detect what is the best combination of features that must be included in the final model). Output: the ranking produced with the algorithm using the training dataset, and the number of best features selected in the best model. In the cross-validation, the average and standard deviation of the scores produced in the different folds used to assess the best model are displayed. In the nested cross-validation, the ranking is presented only after applying Relieff using all data, and the best number of features detected before to build the best model is presented.
	info_gain: has the same utility as relieff, but instead of receiving inputs useful for the application of the Relieff algorithm, it receives inputs useful to calculate the Information Gain of the features included in the dataset being processed. Outputs: uses the same logic presented for the previous task, however, in this case, the scores will be the Information Gain calculated for each attribute.
model_creation: creates ML models using different algorithms.	**svm**: trains and tests an SVM model using the training and test data being processed in the workflow. It receives inputs useful to apply the SVM algorithm, such as the regularisation parameter C and the kernel configurations. It enables the user to set different regularisation parameters and various kernel configurations to produce different models from the different configurations, from which the best model is presented to the user. Output: an object with the created model, the parameters used to build the best model, and the predicted and expected values produced after testing the model using the test data.

Table C.4 – Logical Tasks (continuation)

Task Type	Task Id
model_creation: creates ML models using different algorithms.	**gaussian_naive_bayes**: trains and tests a Gaussian Naive Bayes model using the training and test data being processed in the workflow. It does not receive any input parameters. Output: an object with the produced model and the predicted and expected values produced after testing the model using the test data.
	multinomial_naive_bayes: trains and tests a Multinomial Naive Bayes model using the training and test data being processed in the workflow. It does not receive any input parameters. Output: object with the produced model and the predicted and expected values produced after testing the model using the test data.
	nearest_neighbors: trains and tests a k nearest neighbors model using the training and test data. It receives the parameter k that specifies the number of nearest neighbors to be used to train the model, which can have more than one value, in which case will be returned the model with the parameter k that produces the best classification performance. Output: object with the produced model, the parameters used to build the best model, and the predicted and expected values produced after testing the model using the test data.
	decision_tree:trains and tests a decision tree model that uses the CART algorithm, and the training and test data being processed in the workflow. It does not receive any input parameters. Output: object with the produced model and the predicted and expected values produced after testing the model using the test data (also outputs an SVG image with the tree representation).
model_evaluation: defines the metrics that must be used to evaluate the classification performance of a model built on the workflow.	**classifier_model_eval**: specifies the classification performance metrics to be presented according to the best model produced in the workflow. It let the user define what metric should be used as the decider of the best model configuration. Output: classification performance metrics associated with the best-produced model and also the confusion matrix associated with the tests conducted with the model.

Appendix D

Amazon Web Services Setup

The usage of AWS Step Functions implies the existence of an AWS account where the users' state machines will be executed. The account needs to have the proper IAM configurations[1].

To create and run state machines that use Lambda Functions and that launches other states machines[2][3], it is required to have a Role with Full Access policies for Step Functions and Lambda (Figure D.1).

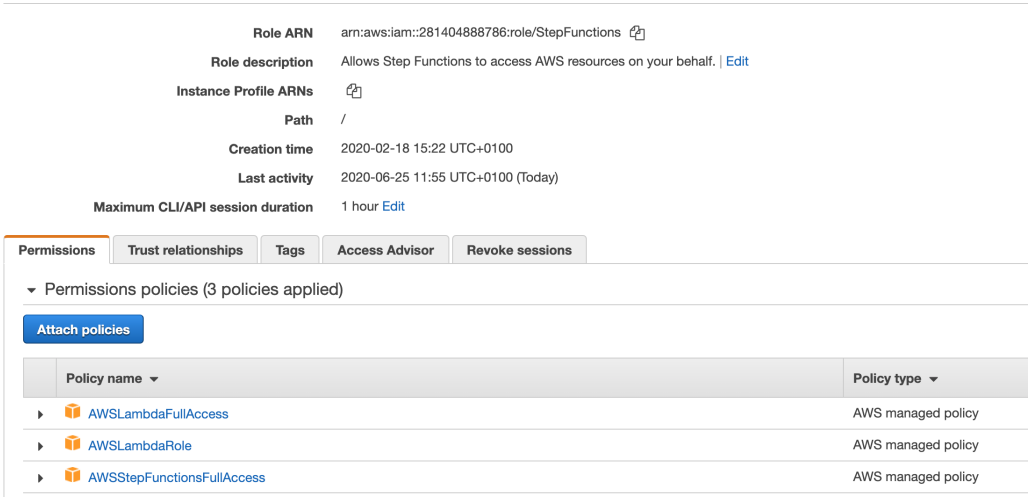

Figure D.1 – AWS Console -Identity and Access Management (IAM) (Role used during development).

Both Workflows Service and the ML Services need access to Step Functions. By setting up authentication credentials, all services are able to use SDK clients. To found these credentials, one must access the IAM Console[4] of the account to be used.

Listing 10 – AWS Authentication Credentials

```
AWS_ACCESS_KEY_ID='access_id' \
AWS_SECRET_ACCESS_KEY='access_key' \
AWS_DEFAULT_REGION='region'
```

[1] https://docs.aws.amazon.com/step-functions/latest/dg/procedure-create-iam-role.html
[2] https://docs.aws.amazon.com/step-functions/latest/dg/connect-stepfunctions.html
[3] https://docs.aws.amazon.com/step-functions/latest/dg/sample-start-workflow.html
[4] https://console.aws.amazon.com/iam/

AWS works based on Regions. The developer guide states that "A state machine or activity exists only within the Region where it was created. Any state machines and activities that you create in one Region don't share any data or attributes with those created in another Region"[5]. For this reason, we must create the necessary activities in the same Region where state machines will be created.

[5] https://docs.aws.amazon.com/step-functions/latest/dg/development-options.html

Appendix E

Integration Tests

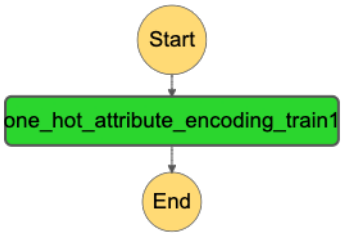

Figure E.1 – IT_01.

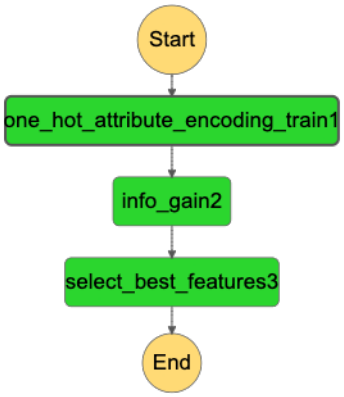

Figure E.2 – IT_02.

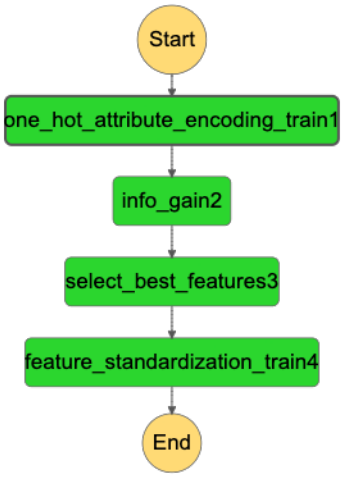

Figure E.3 – IT_03.

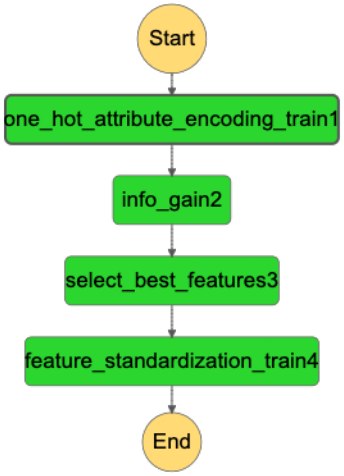

Figure E.4 – IT_03.

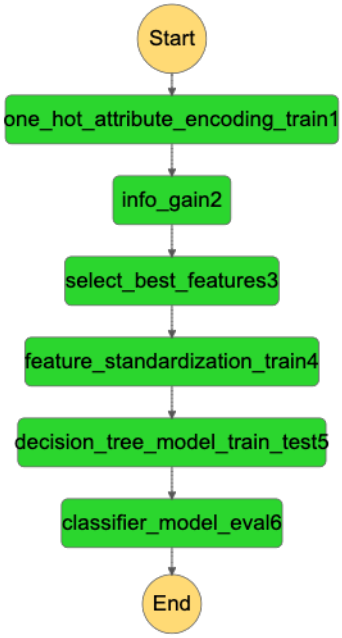

Figure E.5 – IT_04.

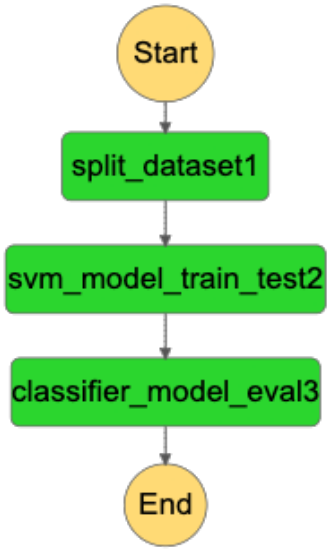

Figure E.6 – IT_05.

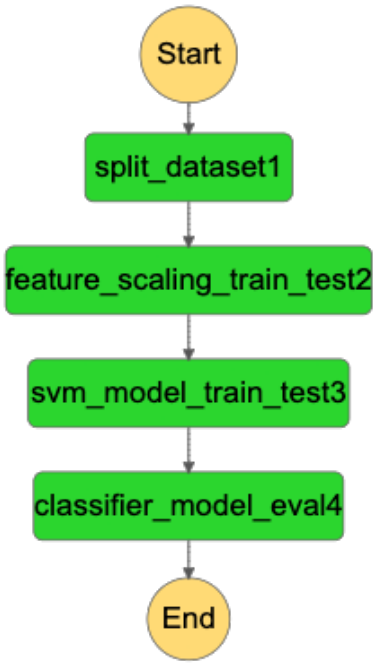

Figure E.7 – IT_06.

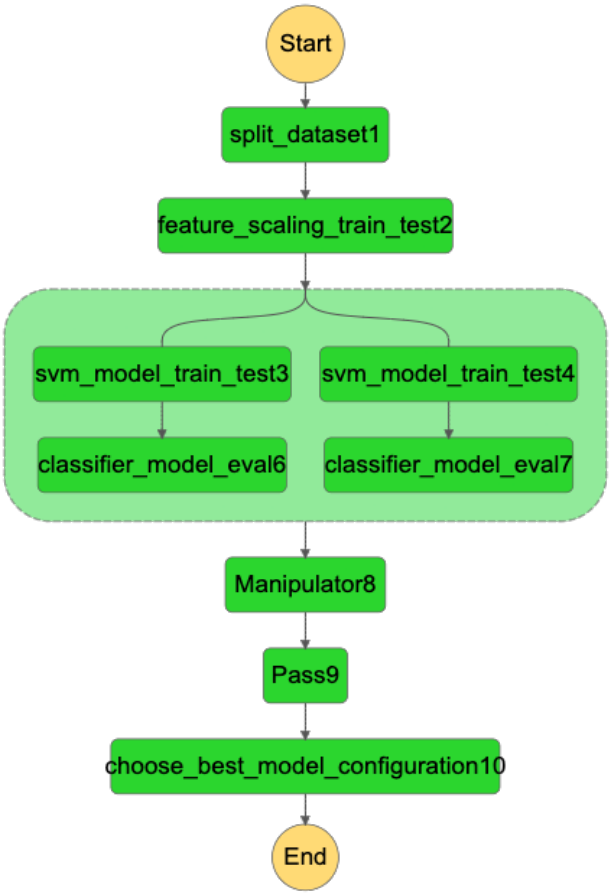

Figure E.8 – IT_07.

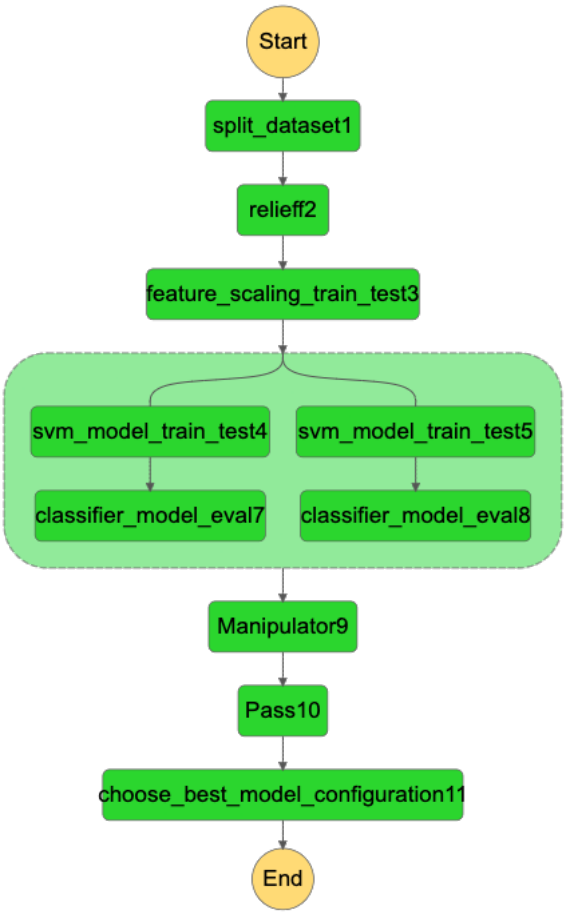

Figure E.9 – IT_08.

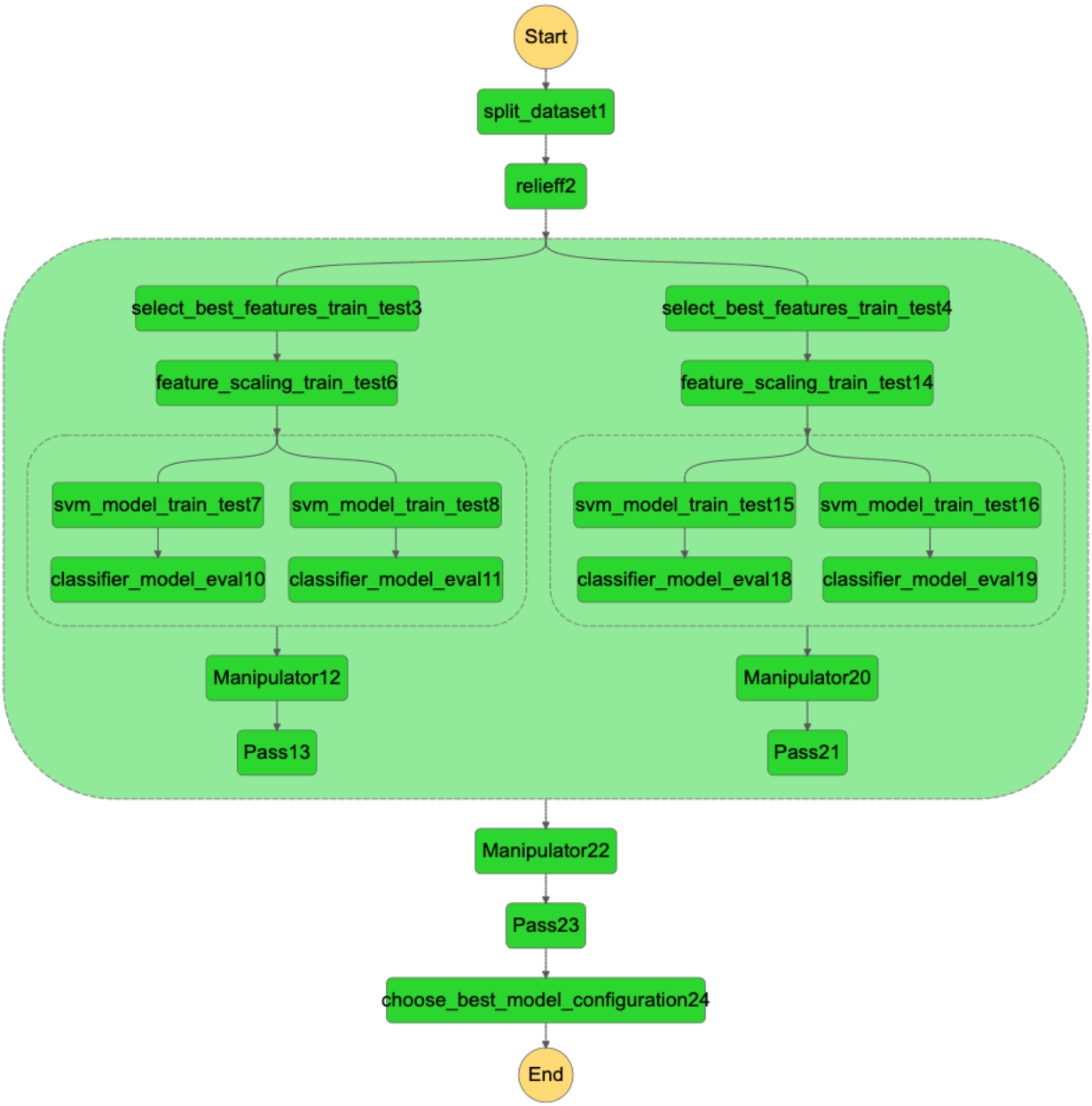

Figure E.10 – IT_09.

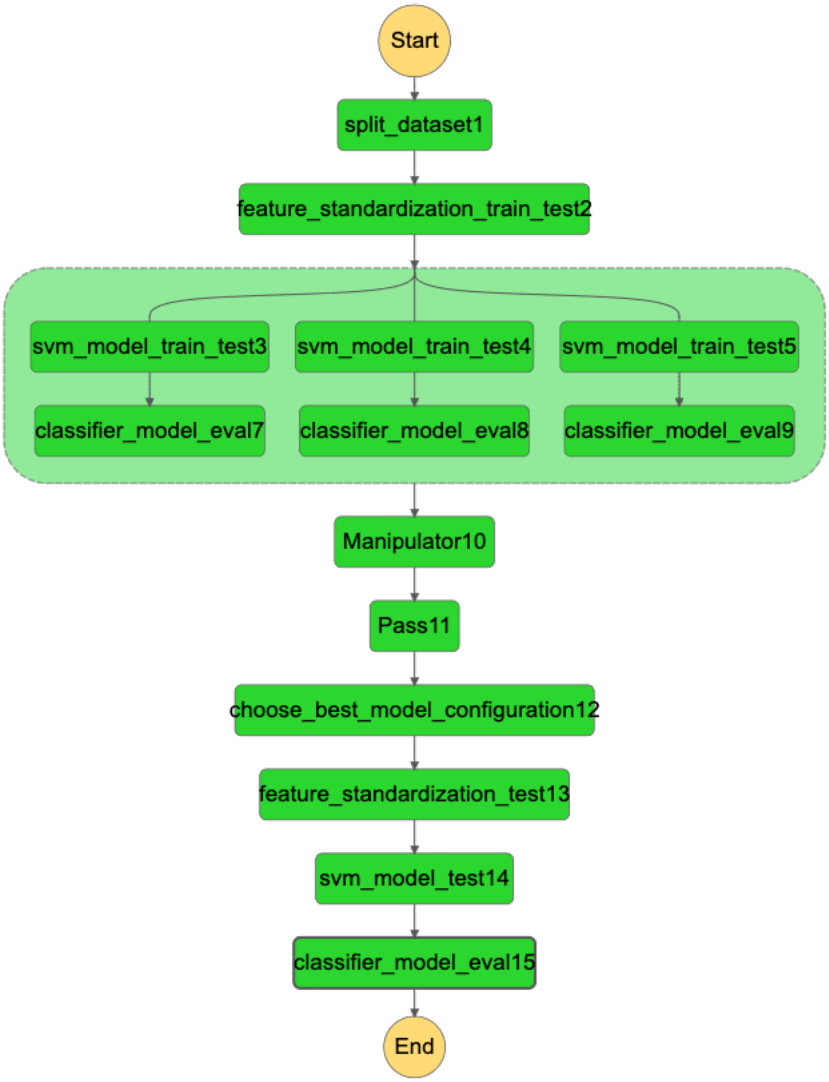

Figure E.11 – IT_10.

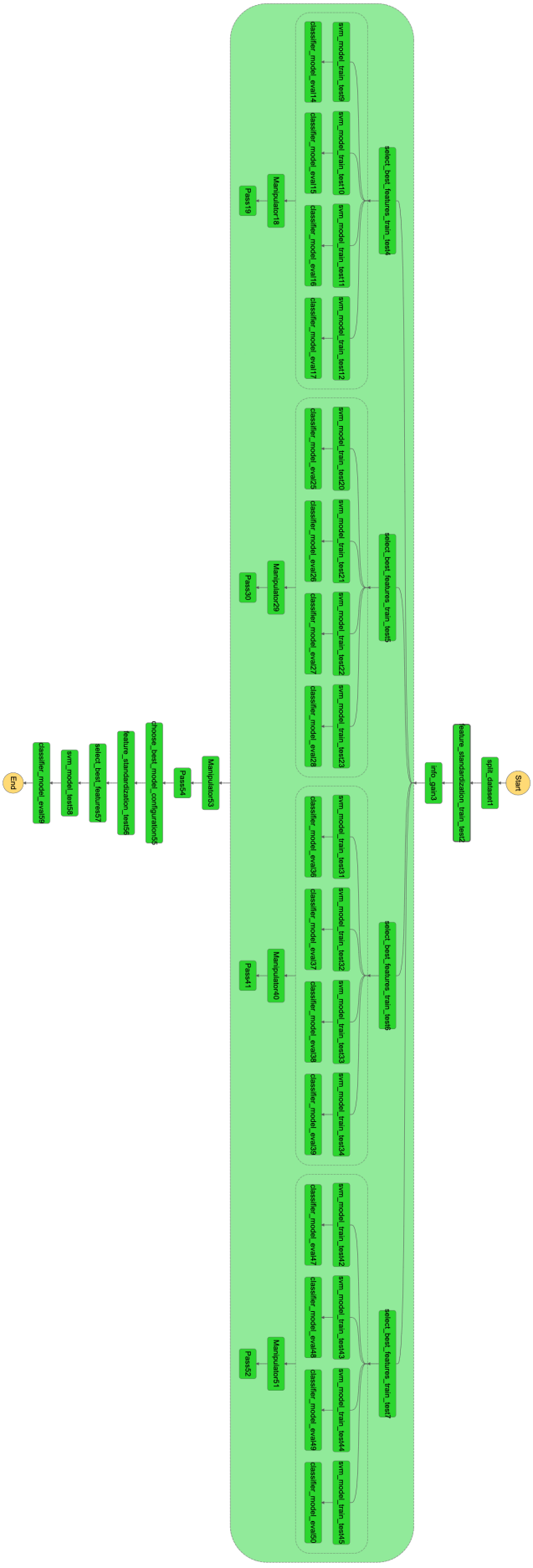

Figure E.12 – IT_11.

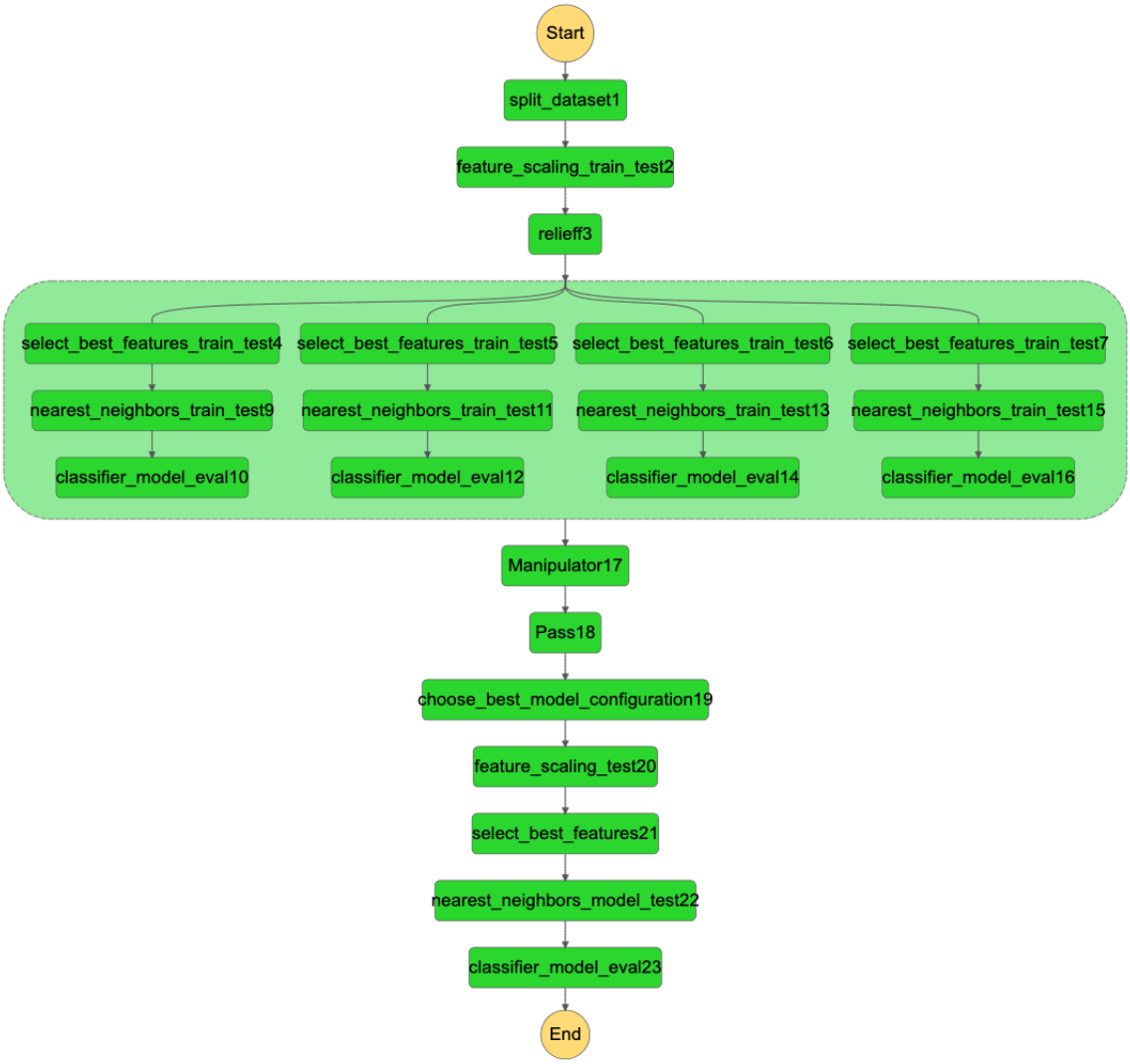

Figure E.13 – IT_12.

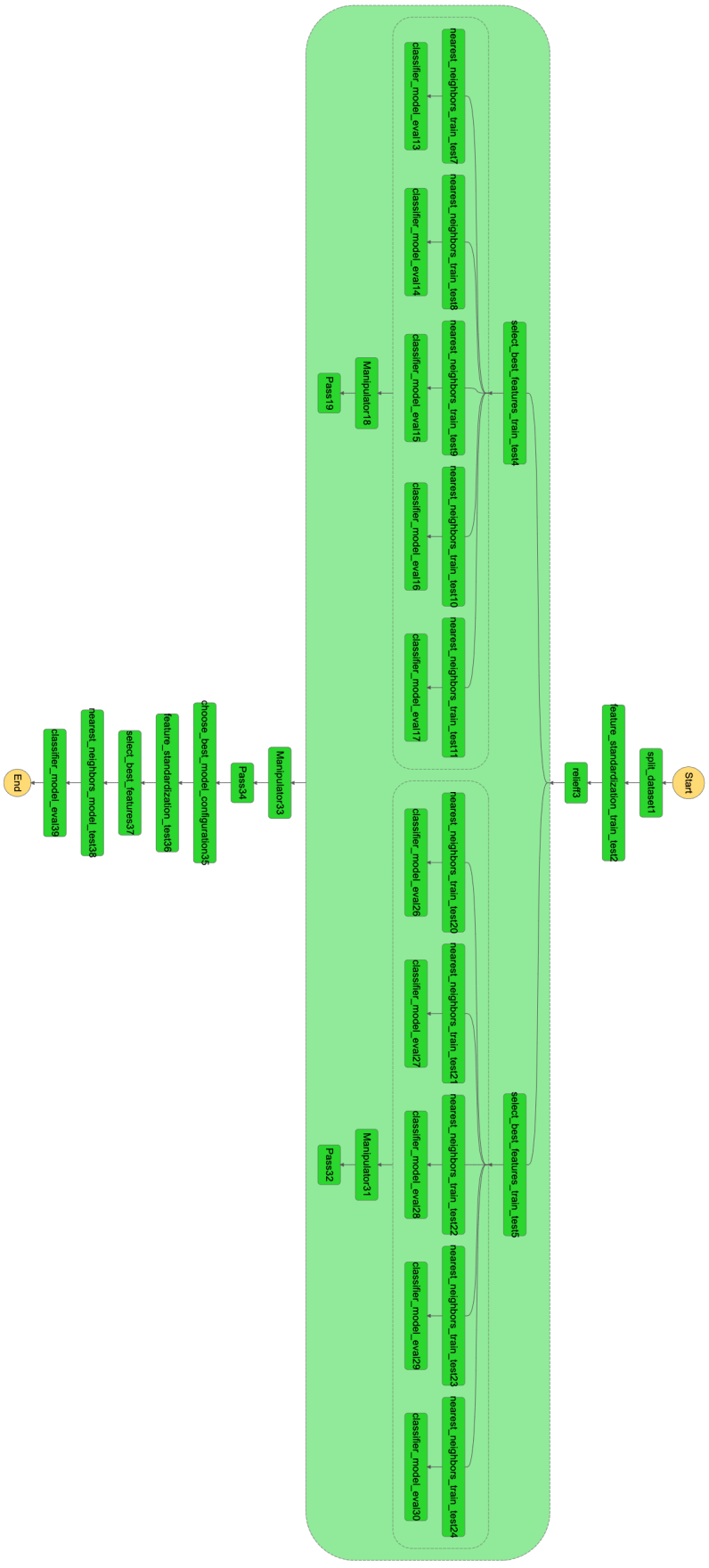

Figure E.14 – IT_13.

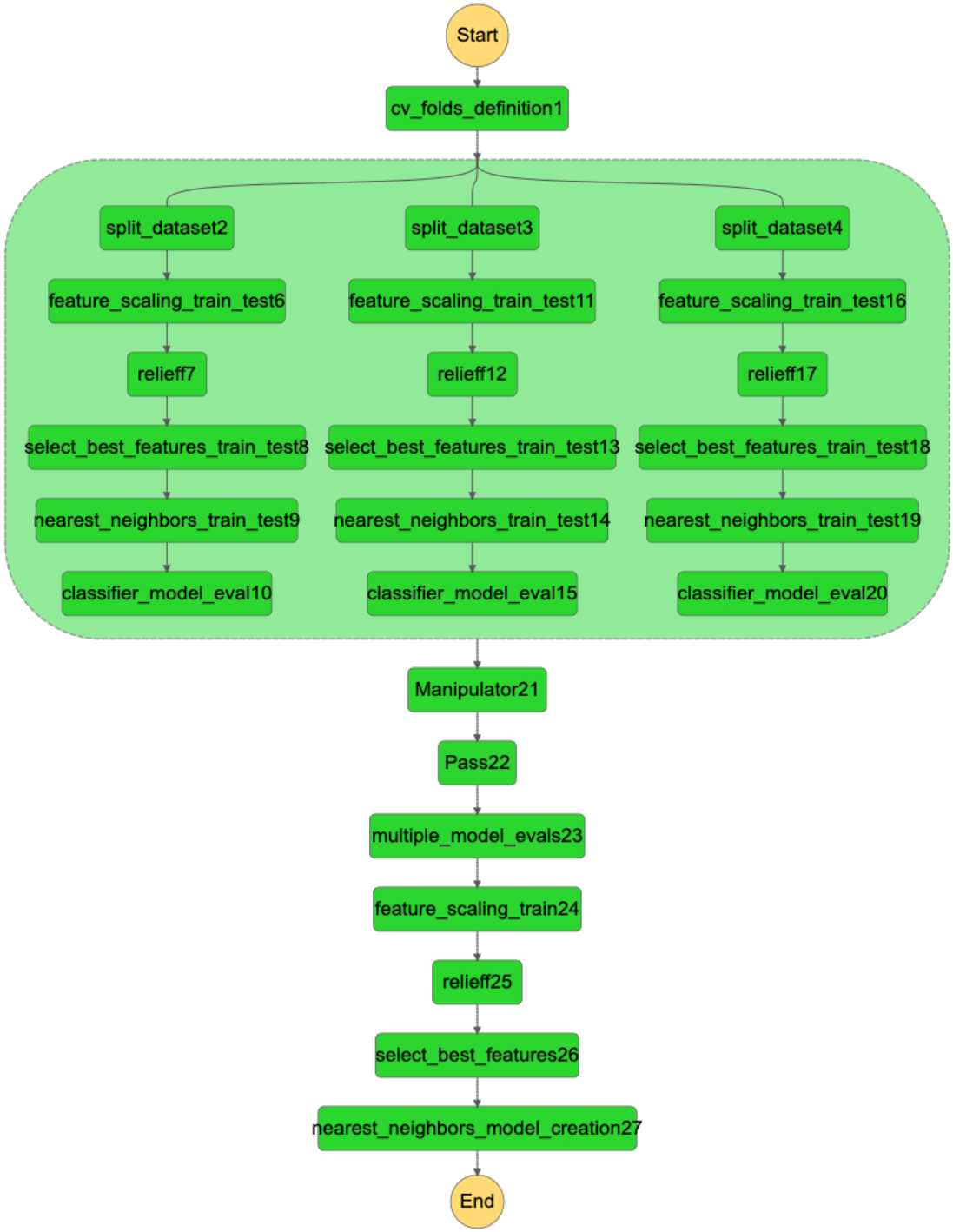

Figure E.15 – IT_14.

Figure E.16 – IT_15.

Figure E.17 – IT_16.

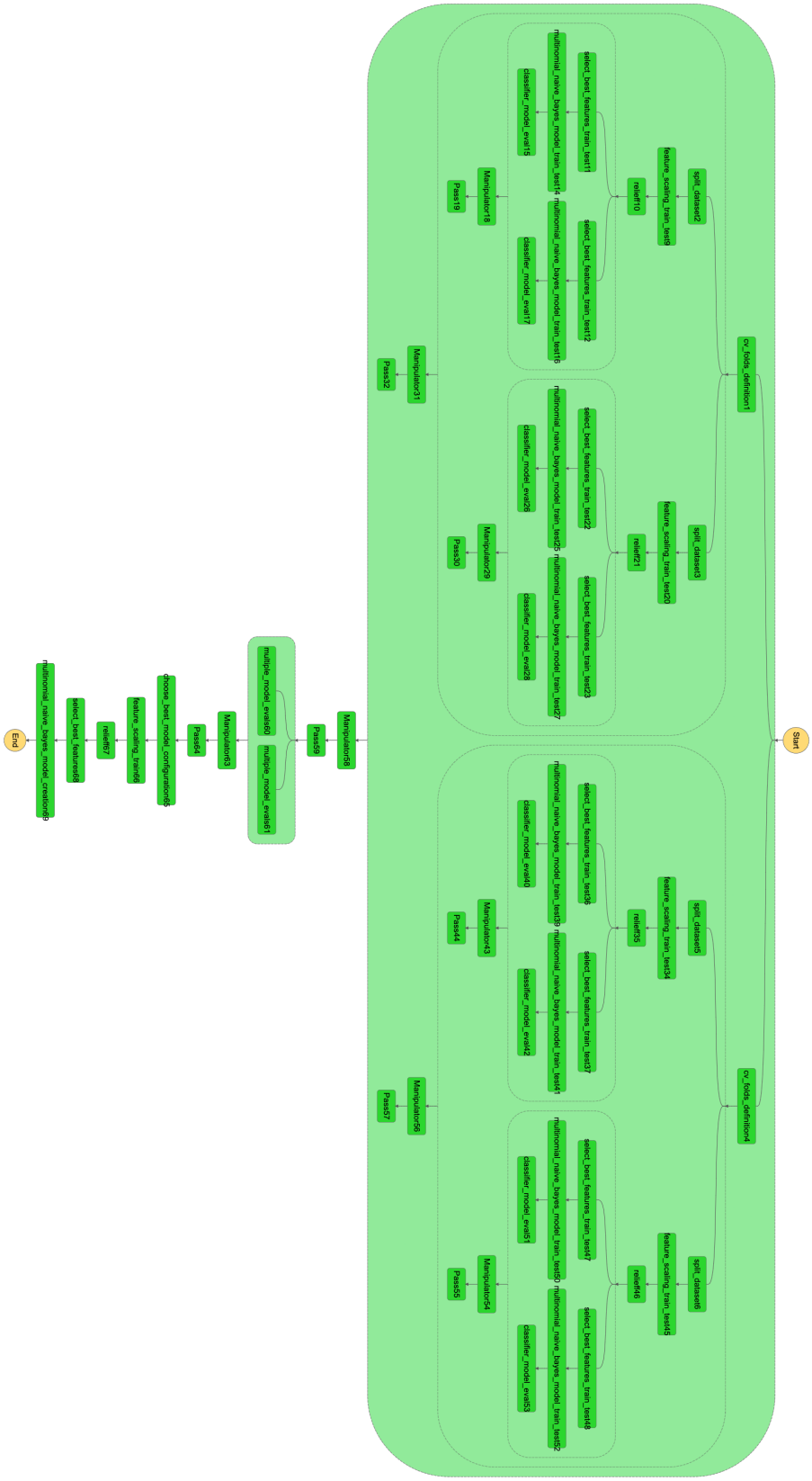

Figure E.18 – IT_17.

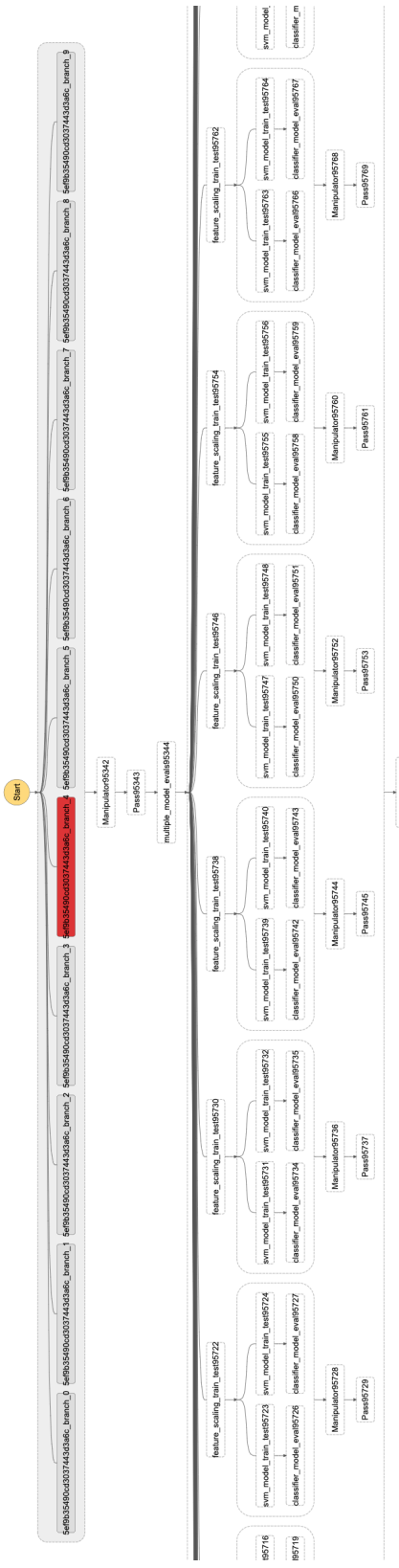

Figure E.19 – IT_20.

Figure E.20 – IT_20 (workflow within a workflow).