TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Stanislav Nazmutdinov 163596IAPM

Automated provisioning of data analytics platforms in the Estonian Scientific Computing Infrastructure

Supervisors: Juhan-Peep Ernits, PhD
Ilja Livenson, MSc

Tallinn 2018

TALLINNA TEHNIKALIKOOL
Infotehnoloogia teaduskond

Stanislav Nazmutdinov 163596IAPM

Andmeanalüütika platvormide automatiseeritud paigaldamine Eesti Teadusarvutuste Infrastruktuuris

Juhendajad: Juhan-Peep Ernits, PhD
Ilja Livenson, MSc

Tallinn 2018

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.
Author: Stanislav Nazmutdinov

Date: _____

Signed: _____

Abstract

Data analytics has become increasingly important in the recent years. Analysis of data is expected to reveal hidden knowledge and patterns. Cloud computing paradigm provides a necessary infrastructure for executing data analytics workloads. However, to take advantage of cloud computing, it is often required to deploy and configure complex engines and development environments on remote virtual machines. The deployment complexity hinders data analysts from adopting novel approaches and technologies.

In order to overcome some of these challenges, we first explore the most widely used tools for data analytics in the industry and in Estonian academia. As a result we decide to focus on Python-based single-node processing. Afterwards we design and build services for provisioning and configuration management of Python development environment and JupyterHub deployment. The services are built in the hybrid cloud brokerage system Waldur[1] which is used at Estonian Scientific Computing Infrastructure organization [2]. Throughout the thesis, we describe the design of the most crucial parts of the proposed services as well as discuss possible ways of testing the implemented services and introduce integration tests for comprehensive quality control. Once the services are implemented and integrated into technical infrastructure of Waldur deployment at Estonian Scientific Computing Infrastructure, we perform evaluation with the potential users from academia. The gathered feedback from the users indicates that the services are indeed relevant to their everyday work and have the potential to be adopted by the university lecturers and researchers.

The thesis is in English and contains 73 pages of text, 5 chapters, 15 figures, 48 tables and 2 appendices.

[1] https://opennodecloud.com/products/waldur.html
[2] http://etais.ee/

Annotatsioon

Andmeanalüütika platvormide automatiseeritud paigaldamine Eesti Teadusarvutuste Infrastruktuuris

Andmeanalüütika on viimastel aastatel muutunud üha olulisemaks. Meetodeid mis kasutatakse andmeanalüüsis lubavad avastada andmetes peidetud teadmisi ja mustreid. Pilvandmetöötluse paradigma pakub vajaliku infrastruktuuri andmeanalüütika töökoormuse teostamiseks. Aga pilvandmetöötluse ärakasutamiseks on sageli vaja juurutada ja konfigureerida keerukaid tarkvarasid ja arenduskeskkondasid virtuaalmasinatel. Juurutamise keerukus takistab andmeanalüütikuid kasutusele võtta uudseid tehnoloogiaid.

Selleks, et lahendada mõned need probleemid me kõigepealt uurime kõige laiemalt kasutatud andmeanalüütikutega seotud tööriistad Eesti akadeemilise maailma sees ja väljaspool. Selle tulemusena me otsustame keskenduda Python-põhisele järjestikku töötlemisele. Seejärel me kavandame ja loome automatiseeritud paigaldamise ja konfiguratsiooni haldamise teenuseid Pythoni arenduskeskkonna ja JupyterHubi jaoks. Teenused on ehitatud hübriidpilvehaldamise süsteemis Waldur[3] mis kasutatakse Eesti Teadusarvutuste Infrastruktuuri organisatsioonis [4]. Antud töö käigus me kirjeldame pakutavate teenuste kõige olulisemaid disaini osasid ning arutleme rakendatud teenuste testimise võimaluste üle ja esitleme integratsiooni testisid, et kindlustada kõikehõlmavat kvaliteedikontrolli. Kui teenused on rakendatud ja integreeritud Walduri juurutamise tehnilise infrastruktuuriga Eesti Teadusarvutuste Infrastruktuuri organisatsioonis, me hindame neid teenuseid akadeemiliste ringkondade potentsiaalsete kasutajatega. Kasutajatelt kogutud tagasiside näitab, et teenused on tõepoolest asjakohased nende igapäevasele tööle ning et teenustel on olemas potentsiaal, et ülikooli õppejõud ja teadustöötajad hakkavad neid kasutada oma töös.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 73 leheküljel, 5 peatükki, 15 joonist, 48 tabelit ja 2 lisa.

[3] https://opennodecloud.com/products/waldur.html
[4] http://etais.ee/

Acknowledgements

I would like to thank my university supervisor Juhan-Peep Ernits who guided me through the writing part of the thesis. Furthermore, I want to express my gratitude to Ilja Livenson, a key person at Waldur project, who assisted me with technical questions regarding Waldur. I am also grateful to all who participated in the first survey and in the final evaluation process. Finally, I want to thank Alexander Horst Norta who provided a LaTeX template for the thesis. Last, but not least, I want to thank Victor Mireyev, a developer from Waldur for his thorough reviews of my code.

Contents

List of Figures

List of Tables

List of Abbreviations

UML ..... Unified Modeling Language

API ..... Application Programming Interface

PaaS ..... Platform as a Service

AaaS ..... Analytics as a Service

DSL ..... Domain Specific Language

ETL ..... Extract, Transform, Load

HDFS ..... Hadoop Distributed File System

HPC ..... High-performance computing

IAM ..... Identity and Access Management

GPU ..... Graphics Processing Unit

ASIC ..... Application-specific integration circuit

REST ..... REpresentational State Transfer

CLI ..... Command-line interface

PyPi ..... Python Package Index

IAM ..... Identity and Access Management

VCS ..... Version Control System

Chapter 1

Introduction

Nowadays, technology is constantly advancing and infiltrating people's everyday lives. Volumes of collected data in both industry and research grow exponentially. By analyzing large volumes of data, researchers may discover hidden knowledge and relations, which may support decision making in many areas [9].

Cloud computing paradigm provides efficient architecture, which helps overcome storage and processing issues users face when performing large-scale computations: it provides reliable storage capacities [10] and sufficient computing resources [11]. Moreover, cloud computing services free users from the burden of building and maintaining complex cluster infrastructure themselves.

Even though data analytics frameworks hide a lot of technical nuances, the main drawback of such toolkits, is that it is quite challenging and time-consuming to deploy, configure and properly manage them. These engines can also be complemented with other tools supporting data processing (for instance, interactive computing environments, visualization, monitoring, debugging tools). As the result, deployment complexity repels researchers from using cloud computational resources, since it requires additional effort and drives focus away from their main research goals.

The Estonian Scientific Computing Infrastructure is an organization that aims to provide computing and storage resources for the Estonian scientific community and R&D companies. Until recently, the Estonian Scientific Computing Infrastructure organization focused mainly on supporting High Performance Computing workloads, however, the focus was shifted towards providing cloud computing services [12]. Waldur is an open-source cloud brokerage platform and it is used by Estonian Scientific Computing Infrastructure to provide cloud services [13].

As of today, Waldur provides its users mostly only infrastructure as a service model, thus extending Waldur with services for provisioning and managing existing data analytics engines is the main goal of this thesis. Suitable tools are identified by conducting a literature analysis and a survey among industry practitioners and Estonian researches from academia.

1.1 Existing Body of Knowledge

The following subsections describe the main concepts related to the thesis. First, we give an overview of Cloud computing and its main concepts in Section 1.1.1. Afterwards, Section 1.1.2 introduces the main concepts of data analytics and last Section 1.1.3 identifies the research gap.

1.1.1 Cloud computing

Cloud computing is a novel approach for computing resources provisioning which utilizes breakthrough technologies that emerged in the past years. Cloud concept definitions may wary [14], but in a nutshell it is an architecture which provides its customers a pool of resources with on demand availability and usage-based pricing (whether it is hardware, development platforms or services). It enables high flexibility and leads to greater efficiency and cost reduction for both cloud provider and clients. Moreover, each customer is able to compose their personal cost plan and manage their resources through self-service or API endpoints. Cloud computing paradigm relies on two main techniques: Service-Oriented Architecture and Virtualization.

Service-Oriented Architecture is an architecture style that constitutes of design principles for integrating a variety of loosely coupled, independent and self-contained components that communicate with each other over a network. Collectively, all the involved services support larger business workflows [15].

Virtualization is a corner stone of each cloud environment. Virtualization is a concept of creating virtual versions of various resources, most notable examples include hardware (virtual machines creation that act as if they were operation systems running on on native hardware), memory (abstracting away internal implementation of memory), storage (composing logical storage with added value on top of underlying physical storage) and network (allowing users to configure network connectivity and addressing space in the cloud). Virtualization provides numerous benefits to both cloud provider and its users, which include increased efficiency (multiple virtual resources can be grouped on

one physical server), greater resource versatility and configurability and more reliable resource availability [16].

Overall, cloud computing brings following benefits to the users [11]:

- Automated infrastructure management: the cloud provider is responsible for managing low-level infrastructure. Moreover, all major cloud providers also offer value-added services for automated applications deployment.
- Flexibility and scalability: users can adapt their usage of computing resources at any time
- Pay per usage model: users are charged for cloud services on an hourly basis for the resources that are solely used by them. Usually, cloud providers do not require any advance payment.

1.1.1.1 Cloud deployment models

Cloud deployment strategies vary depending on the needs of the cloud provider's target customer group. We list here only well-established deployment models, without including any emerging techniques such as micro clouds/ad hoc clouds and cloudlets etc [17].

- Public Cloud: it is a deployment model when a cloud provider makes its resources available to to the general public over the Internet. Public Cloud combined with Platform as a Service offered by Public Cloud provider facilitates rapid application development and early market entry for the small and medium-sized companies.
- Private Cloud: this type of cloud is more secure than Public Cloud due to the fact that whole infrastructure is owned and accessible by a particular organization for its own purposes. The upfront cost of Private Cloud is much higher than in case of Public Cloud, since organization not only requires to have all the required infrastructure, but also qualified experts to maintain it. However, in long run, Private Cloud can pay off the investments, especially for large organization [18].
- Hybrid Cloud: such deployment strategy takes the best of two previous models: it allows users to combine Public and Private clouds and choose whichever type of deployment is more suitable for the each application: critical services with high security requirements can be hosted on Private Cloud while other less critical services can reside on Public Cloud. Waldur cloud brokerage platform belongs to the type of Hybrid Clouds. Hybrid cloud allows users to define disaster recovery

strategies involving several cloud providers in order to make users less vulnerable to data center outages and other incidents [1] [2] [3].

1.1.1.2 Service delivery models

Different types of services offered by cloud providers can be categorized into 3 main service models:

- Infrastructure as a Service: Cloud providers offering this service model provide virtualized computing resources, for instance, virtual machines and storage over the Internet.
- Platform as a Service: It is a service which can greatly simplify infrastructure operations of its customers by providing various platforms for application development, testing, deployment. Waldur currently does not posses any of PaaS offerings, thus one of the goals of this thesis is to fill this gap.
- Software as a Service: In this case cloud provider offers a software to their customers which they can access through a thin client. Customers typically use provided application on pay-per-use basis.

1.1.2 Data analytics

Data analytics is the process of inspecting, processing and transforming data. By analyzing large volumes of data, researchers are able to discover hidden patterns, knowledge, relations and other valuable information, which may bring benefits to both economy and science by suggesting conclusions and supporting decision-making in many areas.

Data analytics is composed of multiple techniques, including machine learning, data mining and visualization methods. These techniques are based on fundamental mathematics, statistics and optimization methods [9]. Data analytics can be categorized into 3 categories:

- Descriptive analytics: Data mining and visualization techniques are widely used in descriptive analytics which answers the question "what is happening"? Data mining - is a collection of methods and algorithms which are used to extract and organize valuable information from various data sets. It involves great variety of

[1] https://www.readitquik.com/articles/cloud-3/top-7-aws-outages-that-wreaked-havoc/
[2] https://cloudstatus.eu/status/google-cloud/
[3] https://cloudstatus.eu/status/windows-azure

methods including data pre- and post-processing, pattern analysis, data classification, clustering methods and data fusion techniques and others [19]. Visualization is another important approach used to create graphical representations of data that help to better understand data under analysis.

- Predictive analytics: Machine learning algorithms and tools are essential in predictive analytics which gives an answer to the questions "What may happen?", "What are possible future trends?". Machine learning - is a set of self-learning algorithms that can hidden detect patterns in data (i.e learn from the provided data) and then use this "experience" to perform some kind of decision making. [20]. Notable subset of machine learning algorithms are deep learning algorithms that proved to be very successful in many areas including image and video recognition, natural language processing.

- Prescriptive analytics: this type of systems aim to assist and even automate decision making, thus prescriptive analytics tries to answer the question "What actions should be taken?" [21]. Prescriptive analytics systems are based on previously built descriptive and predictive analytics in order to be able to come to valid conclusions.

Due to the flexibility that cloud computing paradigm brings to the users, there is an emerging type of services called Analytics as a Service model which interrelates data analytics applications with cloud computing environment. This model is considered to be an alternative to traditional on-premise data analytics. We describe Analytics as a Service model in more detail in Section 2.1.2.

1.1.3 Contribution - Detecting a Gap

The main goal of the thesis is to construct services for management of data analytics platforms within Waldur - the hybrid cloud management system. While in the essence, the idea for such platform is not new, to the best of our knowledge, there are no such platforms which would possess following attributes:

- The platform is part of hybrid cloud brokerage system.

- Complete source code of the cloud management system is distributed under open source license.

Throughout the course of the thesis we describe design and implementation of these services. Instantiation of the design within technical infrastructure of Waldur will show that such service is possible to build in hybrid cloud orchestration systems. It is one

of the first steps for Waldur to further widen its set of provided services beyond just Infrastructure as a Service. Moreover, we discuss how the logic of the built services can be proficiently tested.

1.2 Research Questions and Research Methodology

In order to structure the thesis and ensure its rigor, we follow design science research methodology. It is a formal framework which describes how to apply design-science paradigm in the context of information systems. [1].

1.2.1 Research Questions

The main research question is phrased as follows: **How to dynamically manage deployment and life cycle of data analytics platforms in the context of hybrid cloud brokerage platform?**

The main research question is divided into two subquestions:

- **RQ-1: What are state-of-the-art data analytics engines used in the industry and in Estonia?**
- **RQ-2: How to automate provisioning and management of data analytics tools through hybrid cloud brokerage platform?**

Answer to the first research question is given through literature analysis and survey, we provide an informative overview of available solutions. Based on the discovered knowledge, we make a decision which tools exactly to provision and what kind of use cases is sensible to automate. The proposed design for the second question is instantiated and evaluated with potential customers.

1.2.2 Design Science Research Framework

The core of the Design Science Research process is presented in Figure 1.1 by Hevner et al. [1]. According to this framework, the main output of the design science research process is an artifact which can be defined as methods, models, constructs or instantiations. The authors specify that methods may be algorithms and practices, models can be seen as abstractions and representations, constructs are referred to vocabulary and

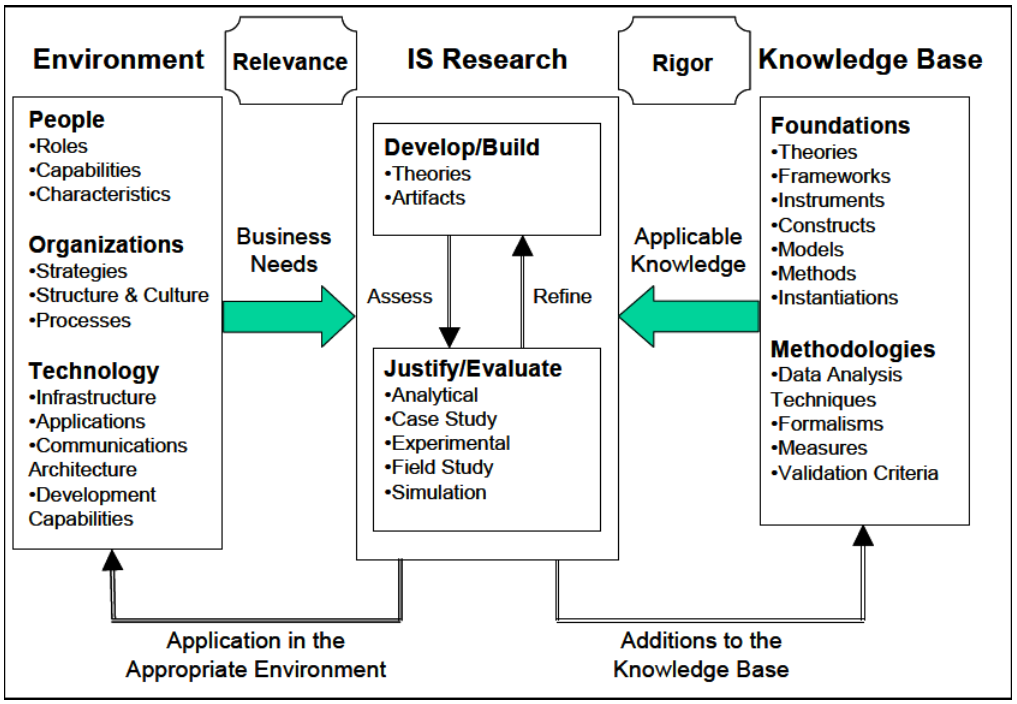

FIGURE 1.1: Design Science Research - Framework (Source: [1])

symbols. Instantiations can be implementation or a prototype [1].

The process of creation of an artifact should be based on the following aspects: firstly, it should be relevant to the environment and be applicable there. Secondly, artifact should be built on top of existing knowledge base. What is more, the provided artifact should not only be useful to the environment where it is built, but also provide contribution to the knowledge base.

1.2.3 Design Science Research - Application

The paper also provides guidelines which help researchers understands requirements of design science research. These are illustrated in Table 1.1. In this section we describe how these guidelines are applied in the context of the current thesis.

1.2.3.1 Design as an Artifact

The artifact produced throughout this thesis intends to solve the issue of provisioning and management of data analytics platforms in hybrid cloud environment. The artifact consists of a model and its instantiation. More thorough description of contributions is presented in the previous Section 1.1.3.

Guideline	Description
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

TABLE 1.1: Design Science Research - Guidelines (Source: [1])

1.2.3.2 Problem Relevance

Addressed problem is relevant especially in the cloud environment context. Even though modern data analytics frameworks hide a lot of technical nuances, the main drawback of such toolkits is that it is quite challenging and time-consuming to deploy, configure and properly manage them. As the result, deployment complexity repels researches from using cloud computational resources, since it requires additional effort and drives focus away from their main research goals.

1.2.3.3 Design Evaluation

Authors of design science research paper list evaluation methods which are summarized in Table 1.2. The evaluation in the current thesis is done with several methods.

During this thesis we instantiate the proposed data analytics tools provisioning model and integrated it with the existing technical infrastructure of Waldur cloud-brokerage system. Afterwards, a case-study evaluation with potential users is performed.

1. Observational	Case Study: Study artifact in depth in business environment
	Field Study: Monitor use of artifact in multiple projects
2. Analytical	Static Analysis: Examine structure of artifact for static qualities (e.g., complexity)
	Architecture Analysis: Study fit of artifact into technical IS architecture
	Optimization: Demonstrate inherent optimal properties of artifact or provide optimality bounds on artifact behavior
	Dynamic Analysis: Study artifact in use for dynamic qualities (e.g., performance)
3. Experimental	Controlled Experiment: Study artifact in controlled environment for qualities (e.g., usability)
	Simulation – Execute artifact with artificial data
4. Testing	Functional (Black Box) Testing: Execute artifact interfaces to discover failures and identify defects
	Structural (White Box) Testing: Perform coverage testing of some metric (e.g., execution paths) in the artifact implementation
5. Descriptive	Informed Argument: Use information from the knowledge base (e.g., relevant research) to build a convincing argument for the artifact's utility
	Scenarios: Construct detailed scenarios around the artifact to demonstrate its utility

TABLE 1.2: Design Science Research - Evaluation Methods (Source: [1])

1.2.3.4 Research Contribution

The thesis makes a contribution in a form of an implementation of services for Python development environment management and JupyterHub configuration management on a remote machine. These services make Waldur cloud services more accessible to users who would like to execute Python-based single-node workloads in the cloud. The core design of the application logic for application configuration management is general and can be reused for the purpose of provisioning other cloud applications. We show this reusability by implementing two services.

1.2.3.5 Research Rigor

Prior to selection of suitable data analysis tool for provisioning, we perform literature analysis and conduct a survey among Estonian scientific community. During this thesis, we use UML modeling language in order to document the proposed design.

As for practical part, the quality of the code is constantly controlled by the continuous integration pipelines powered by Jenkins[4]. The pipelines includes tests and static code analysis tools for identification of security vulnerabilities[5] and for checking code style[6].

[4] https://jenkins.io/
[5] https://wiki.openstack.org/wiki/Security/Projects/Bandit
[6] http://flake8.pycqa.org/en/latest/

On top of that, the code is reviewed by other peers to ensure that the implementation adheres clean-code conventions, object-oriented design principles and that testing requirements are fulfilled.

1.2.3.6 Design as a Search Process

As part of a search process, we perform a literature analysis and conduct a survey among our target user group to identify what data analytics framework are most commonly used in the industry and in the scientific community. Afterwards, we select the most promising tool that will bring the most benefit to the target users. We then come up with the set of use cases that the services should support. The appropriate services for provisioning of these tools are then designed and built within the existing technical infrastructure of Waldur cloud management system. Finally, the proposed solution is validated during case-study evaluation with the users.

1.2.3.7 Communication of Research

The thesis is presented to the university examining committee and afterwards made available to everyone on the Digital Collection of Tallinn University of Technology Library.

1.3 Thesis Structure

The rest of the thesis is structured as follows: Chapter 2 introduces state of the art applications provisioning cloud solutions and Analytics as a Service solutions. In chapter 3 we describe available frameworks and engines for data analytics and report our survey findings. Chapter 4 describes architecture of the service for provisioning data analytics tools in Waldur and its case-study evaluation. Finally, Chapter 5 concludes the thesis and provides a discussion of the achieved results.

Chapter 2

Background

The following chapter explains basic concepts used in this thesis. Section 2.1 describes service models that support data analytics in the cloud environment. Subsequently Chapter 2.2 introduces state of the art cloud applications provisioning solutions and Analytics as a Service platforms which provide interactive computing capabilities to perform data analytics in the cloud.

2.1 Cloud computing service models

In this Chapter we will further elaborate main concepts described in Sections 1.1.1, 1.1.2 and show how they are interdependent.

2.1.1 Platform as a Service

Nowadays Platform as a Service model is becoming increasingly significant to operators and developers. Platform as a Service model is a middle ground between low-level Infrastructure as a Service and high-level Software as a Service models. Cloud computing has been initially introduced to free developers from burden of infrastructure issues, PaaS, in turn, provides means to build and deploy cloud applications in more efficient way. As the result, PaaS is widely used by start-ups and medium-sized companies who strive to reduce time to market as much as possible [18].

According to Costache et al. [23] Platform as a Service model posesses three distinct features:

- Deployment automation: meaning that PaaS is able to deploy supplied application, properly configure runtime environment for it and upgrade the environment if necessary.
- Monitoring: in other words, PaaS should be able to collect and retrieve both application-level as well as PaaS-level logs to be able to conclude regarding current state of the system.
- Resource provisioning: PaaS should be able to allocate required resources to the running applications.

Naturally PaaS systems are not limited only to these features. Additionally, these systems may offer graphical user interface, failure recovery and auto-scaling features in order to ensure that service-level agreements are satisfied.

PaaS systems have found adoption in a wide range of domains, including High Performance Computing applications, Data Analytics and Web Applications [23]. In this work we will focus on data analytics workloads.

2.1.2 Analytics as a Service

Nowadays due to the great variety of the data that can be processed and analyzed, there are numerous fragmented tools available for each type of workloads. These tools run in disjointed environments, so it is very time-consuming and difficult to integrate these tools with each other.

Moreover, with ever growing volumes of analyzed data, scalability of the available tools becomes a problem. The issue is solved with parallel computing: spreading load across multiple computing nodes. Cloud environment with possibility to request seemingly limitless resources is a natural choice for this kind of workloads.

As the result, there is an emerging service model that a lot of cloud providers strive to offer: Analytics as a Service. Analytics as a Service cloud solutions provide an environment that attempts to unify numerous fragmented tools in one single environment. Additionally, AaaS provides better scalability and availability of resources for data-intensive analytical applications as well as higher cost savings by taking over low-level infrastructure management [21]. Furthermore, some Analytics as a Service services help to coordinate team activities by providing advanced IAM, project workspaces, datasets and source code versioning. Analytics as a Service solutions may either follow Platform as a Service model or can be application provisioning systems which are focused more on deployment of data analytics engines.

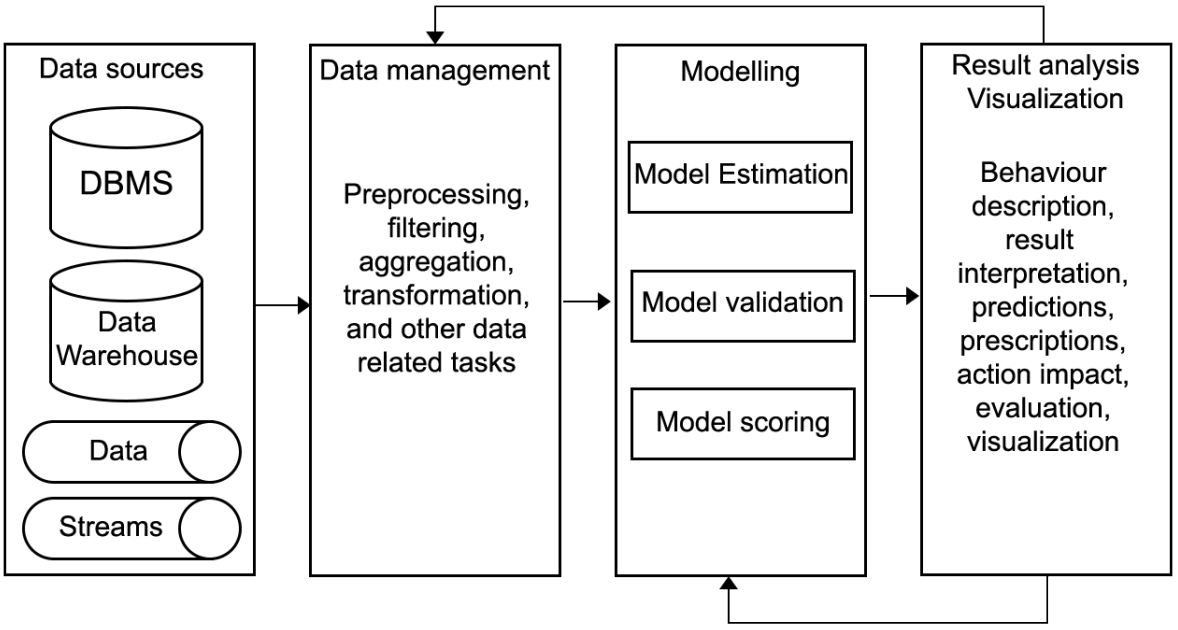

FIGURE 2.1: Data analytics workflow (Source: [2])

A general overview of the data analytics workflow is depicted in Figure 2.1. First block illustrates possible origins of data such as various database management systems (both SQL and NoSQL), data warehouses, data streams. The second block depicts various data cleaning and other transformation tasks. The processed data is used to train various machine learning models. Once a model is built, it should be validated. Usually the model is tested for various attributes using the original input and calibrated if required. Afterwards, when model reaches scoring phase, it is ready to generate predictions and recommendations. The results are then thoroughly analyzed and acted upon [2]. AaaS systems aim to support all the major steps from this workflow: offer integration with various data sources, provide effective tools for data management, model construction and visualization of data. Each part of the workflow is usually powered by the particular open-source solution. When all of them are integrated together they form unified Analytics as a Service system.

2.1.3 Application provisioning services

These are such services that provide automatic configurable deployment of application in the cloud. The deployment strategy may vary a lot: from a simple installation of the particular system packages on a single machine to a complicated cluster setup. These systems are very useful to cloud users, since they allow rapid deployment of various applications. At the same time these services leave users a possibility to access provisioned virtual machines to perform additional manipulations if needed. Many of

such systems provide automatic deployment of data analytics tools. We ourselves build a service which will offer automatic deployment and some degree of subsequent automated configuration management.

2.2 Related work

In this section we introduce available cloud application provisioning systems and Analytics as a Service solutions from major providers on a cloud market.

2.2.1 Application provisioning services for data analytics systems

2.2.1.1 Amazon AWS

One way of provisioning software to cloud users is by distributing machine images with pre-installed software packages. Amazon AWS hosts its own marketplace with a handful of virtual machine images (called Amazon Machine Images) with different pre-installed data analytics software[1] (e.g GPU drivers, deep learning libraries, interactive notebooks software, Anaconda environment, etc.). These machine images support a variety of instance types, including multi-GPU instances. What is more, Amazon has also created its own infrastructure management and provisioning engine with DSL called AWS CloudFormation. Some products available on AWS Marketplace take advantage of AWS CloudFormation templates to provision more complicated deployment topologies.

2.2.1.2 Google Cloud

Google Cloud Engine offers a service called Cloud Deployment Manager which enables users to create their own automation templates for application provisioning [2]. What is more, Google also hosts a marketplace of common application deployment templates that can be easily executed using Cloud Launcher [3]. Depending on a chosen application, it is possible to request either a cluster or a single machine deployment. Unfortunately, Cloud Launcher provides no further configuration management capabilities once the machines with the software are provisioned.

[1] https://aws.amazon.com/machine-learning/amis/
[2] https://cloud.google.com/deployment-manager/
[3] https://cloud.google.com/launcher/

2.2.1.3 IBM Cloud

IBM Cloud is a cloud platform which is integrated with formerly known IBM Bluemix - a Platform as a Service that allows users to deploy, run and manage application in the cloud environment. The service comes with a centralized catalog of all the platforms and tools that IBM has to offer, ranging from application deployments on virtual machines to requesting access to artificial intelligence and Internet of Things platforms[4]. The application deployment engine uses Cloud Foundry buildpacks to install the requested software [24]. As distinct from Google Cloud Launcher, IBM Cloud offers high-level PaaS experience to the users (due to the fact that it is based on Cloud Foundry). IBM Cloud provides extensive application management once the application is deployed. In addition to common functions such as access to log files, application lifecycle management and configuration of virtual machine computational capacity, each application management screen is specialized for the particular provisioned software.

2.2.1.4 Waldur

As of now, Waldur provides a generic service for one-time execution of Ansible Playbooks[5]. This service works in a similar fashion as Google Cloud Launcher does: users can supply values for a predefined set of parameters that are passed to the Playbook once its execution is triggered. As of today, the user interface is pretty basic. In order to make these Playbooks available to the users, administrators should upload them through Waldur admin console, regular cloud users are not allowed to upload their own Playbooks. By defualt, Waldur comes with a limited collection of Ansible Playbooks written by Waldur developers. What is more, Waldur development team provides custom Ansible modules for calling Waldur API endpoints in order to perform various actions such as provisioning of a new virtual machine, getting information about the virtual machine from Ansible Playbook logic.

2.2.2 Analytics as a Service platforms

As for AaaS solutions, in the scope of the thesis we consider noteworthy AaaS systems which provide automatic infrastructure provisioning and support interactive data analytics workloads, since our system which we built is heavily focused on those aspects. Even though modern AaaS solutions are very broad and cover all the major steps in the analytics workflow depicted in Figure 2.1, we believe that providing a brief description

[4] https://console.bluemix.net/catalog/?category=platform
[5] http://docs.ansible.com/ansible/latest/user_guide/playbooks.html

of AaaS systems is useful, since they give ideas how it is possible to further develop services built throughout this thesis into a full-blown AaaS solution.

2.2.2.1 Amazon AWS

Amazon Elastic MapReduce (EMR) is a system which enables users to process data-intensive tasks using various MapReduce-based open source tools on the Amazon infrastructure. The service allows users to create resizable cluster without bothering themselves with its low-level setup and Hadoop configuration. Furthermore, Amazon EMR deploys various supporting tools including interactive notebook servers such as Apache Zeppelin, Apache Hue and Jupyter. Needless to say, that Amazon EMR integrates with many other AWS services for data storage, cluster monitoring, identity and access management, audit and job scheduling [27].

2.2.2.2 Microsoft Azure Notebooks

Microsoft Azure Notebooks[6] is a PaaS solution which provides Jupyter notebook execution environment and a repository for sharing them. The environment is maintained entirely by Microsoft Azure development team, thus users do not have access to the underlying virtual machines and are not allowed to modify predefined set of supported Jupyter kernels and extensions. However, users are permitted to install additional libraries to the existing kernels. Unfortunately, once the notebook server instance shuts down, any additionally installed libraries are lost. Furthermore, the notebook server instances have several limitations, for example, there is a 4 GB memory limit per user and only 1 GB of accessible disk storage [28].

2.2.2.3 Google Colaboratory

Google Colaboratory [7] is another Platform as a Service system which provides Jupyter notebook execution environment. Since it is a PaaS system, it completely abstracts away underlying infrastructure. Pretty much like Microsoft Azure Notebooks, the system provides integration with GitHub and it own cloud storage solution (Google Drive). As of today, Colaboratory only supports Python 2 and Python 3[8]. A notable feature of the service is that it is possible for a user to use their local computer as an execution runtime for Jupyter notebooks managed by Colaboratory [9]. Furthermore, multiple users may

[6] https://notebooks.azure.com/
[7] https://research.google.com/colaboratory
[8] https://research.google.com/colaboratory/faq.html
[9] https://research.google.com/colaboratory/local-runtimes.html

work on the same notebook, in a same manner as in Google Docs. In contrast to Microsoft Azure Notebooks, Colaboratory provides free GPU-based computing. However, a 12 hour-limit applies to a GPU-powered working session [29]. Moreover, several users usually share a single GPU which affects the performance. As of today, the following computation and storage restrictions are applied to users: 13 GB of memory and 1 CPU core [10].

2.2.2.4 IBM Watson Studio

Similarly to Microsoft Azure Notebooks, IBM Cloud provides its own PaaS solution for powering interactive data analytics. The service is called Watson Studio. The platform strives to integrate a variety of fragmented tools in a single platform. The system provides both RStudio and Jupyter notebook environments. Moreover, Watson Studio aims to solve various issues that come with working in a team on a single project: difficulties of data and source code sharing and versioning. To further accelerate project development, it provides integration with a diverse set of datasources [30] as well as visibility into the uploaded datasets. Moreover, the system also fosters knowledge sharing across users by providing means to discover articles, tutorials and sample notebooks [31]. Naturally, it is integrated with other IBM services from IBM Watson Data Platform, such as Analytics Engine (big data processing in cluster), IBM Watson Machine learning (deep learning workloads) and others.

2.2.2.5 Databricks

Another example of Analytics as a Service proprietary product is Databricks which was founded by the creators of Apache Spark. Databricks is intended to solve several major problems that come with large-scale machine learning using Apache Spark: (i) cluster deployment complexity - the platforms enables not only on-demand cluster provisioning and scaling, but it also keeps all the required software up to date. Moreover, Databricks also provides third-party libraries installation and management in a cluster. (ii) Another issue is lack of feature-rich multi-tenant visualization environment for Apache Spark - Databricks provides its own interactive notebook environment with configurable IAM roles and integration with version control systems. (iii) Third challenge is deploying Apache Spark jobs into production which oftentimes involves writing additional scripts to deploy job to production environment. Databricks closes this gap by allowing users to seamlessly switch between production environment and interactive exploration notebooks [32]. Moreover, it is possible to execute GPU-enabled deep learning workloads in

[10] https://colab.research.google.com/notebook#fileId=1_x67fw9y5aBW72a8aGePFLlkPvKLpnBl

Databricks [11]. Databricks can also be run in heterogeneous cloud environment. As of now Databricks supports Amazon AWS and Microsoft Azure cloud providers.

[11] https://docs.databricks.com/applications/deep-learning/index.html

Chapter 3

State-of-the-art data analytics engines

3.1 Introduction

The main goal of the Chapter 3 is to answer the first research question: **What are state-of-the-art data analytics engines used in the industry and in Estonia?** In order to give a comprehensive answer to this question, we split into three subquestions, namely:

- RQ-1.1: What are general-purpose data processing engines are used in the industry? - Section 3.2
- RQ-1.2: What deep learning libraries are used in the industry? - Section 3.3
- RQ-1.3: What data analytics tools are used in Estonia: conducting a survey. - Section 3.4

3.2 General-purpose data processing engines

Nowadays there are a wide range of various tools available for executing data mining and machine learning workloads. Different engines have their advantages and downsides, many have overlapping use cases. In this section we will present an overview of available tools for data analytics.

In this section we discuss data processing frameworks which can be used either for ETL, data mining and machine learning workloads. We do not consider here any data storage

solutions as well as we do not include here deep learning frameworks due to their specific nature.

3.2.1 Python-based libraries

Python is an open source interpreted general-purpose programming language with dynamic type system and automatic memory management. Even though it is not entirely focused on data analytics workloads, it provides a very expressive syntax which appeals to many data analysts. There is available a collection of Python libraries for science and mathematics use cases called SciPy[1]. The most notable example of machine learning library is *scikit-learn* which provides implementation for many state-of-the-art machine learning algorithms. It follows imperative programming principles which makes code easier to debug and troubleshoot [33]. Due to the popularity of Python, many other data analytics engines have Python API. In addition, Python can be used in interactive development environments powered by Jupyter[2] to compose notebooks which besides code may include also formatted text, diagrams, graphics and other visualizations.

3.2.2 R

R is an open source interpreted programming language for statistical computing with built-in visualization packages. R contains numerous extensions for performing numerical computations with vectors, matrices etc. Moreover, *dplyr*[3] package provides a set of abstractions for storing data in Data frames that provide convenient and powerful API for further data manipulation. R is widely used for data mining and data analysis purposes. Unfortunately, R packages have pretty limited capacity for processing large data, because the memory is limited to only one node. Moreover, R does not have any native support for multithreading. In order to mitigate these flaws, there were developed numerous packages to support analysis if large datasets. For instance, *ff* package[4] which allows to transparent access data stored on disk as if it was stored in memory. Besides, there is also available *snow* package[5] for simple parallel computing in a cluster. Moreover, a lot of data processing engines provide API in R [6]. As for interactive computing, R is supported by both Jupyter and RStudio.

[1] https://www.scipy.org/
[2] http://jupyter.org/
[3] https://dplyr.tidyverse.org/
[4] https://cran.r-project.org/web/packages/ff/index.html
[5] https://cran.r-project.org/web/packages/snow/index.html

In subsequent sections we discuss various distributed data processing engines. The engines can be split into 2 major execution models: batch and stream processing. In case of batch workloads the processing is performed on the whole available dataset, for example, to generate various reports. The batch processing jobs may take long time to complete. Data streaming jobs, by contrast, process only the most fresh portions of data. Stream processing solutions stress the importance of being able to process constantly incoming data with low latencies. Table 3.1 provides an overview of available solutions for distributed processing.

System	Execution model	Supported languages	Available ML tools	In-memory processing	Low latency	Fault tolerance
Hadoop MapReduce	Batch	Java	Apache Mahout	no	no	yes
Apache Spark	Batch, streaming	Scala, Java, Python, R	MLlib, Apache Mahout, H2O	yes	yes	yes
Apache Flink	Streaming, batch	Scala, Java	FlinkML	yes	yes	yes
Apache Heron	Streaming	Java, Python	-	yes	yes	yes
H2O	Batch	Scala, Java, Python, R	H2O, Mahout, MLlib	yes	yes	yes

TABLE 3.1: Comparison of available distributed data processing engines (Source: [4])

3.2.3 Hadoop ecosystem

Apache Hadoop is a programming model used for distributed data processing in a cluster environment. Hadoop tools are meant to solve several issues (i) problem of slow read-/write disk operations by splitting the information across multitude of disks to achieve parallelism and therefore performance boost when working with data stored on disks. The second issue is (ii) hardware failure which is inevitable when computations are performed on large amount of nodes (which is required to solve the first problem). Thirdly, (iii) the problem of combining the results of computation which is challenging in distributed environment, thus a MapReduce model is proposed to abstract away low-level

technical details [34]. MapReduce provides unified API regardless of whether the job is executed on a local machine or in a cluster.

Multiple MapReduce-based engines and frameworks were adopted by the industry for batch analytics over the past years: they are being developed independently and all together form comprehensive ecosystem for Big Data processing.

The backbone of Hadoop stack is Hadoop Distributed File System (HDFS) which is a data storage system that can be spread across hundreds of heterogeneous nodes. It can store large volumes of structured and unstructured data. In order to ensure fault tolerance, it replicates data across the cluster. However, file lookup operations in HDFS suffer from substantial latencies which makes it not the perfect choice for real-time processing, rather than for batch processing.

3.2.3.1 MapReduce

As it was mentioned before, MapReduce simplifies processing of large data volumes which do not normally fit into a memory of general-purpose computers. MapReduce model uses two main steps during computations: map and reduce phases [5]. The MapReduce data flow is depicted in Figure 3.2.

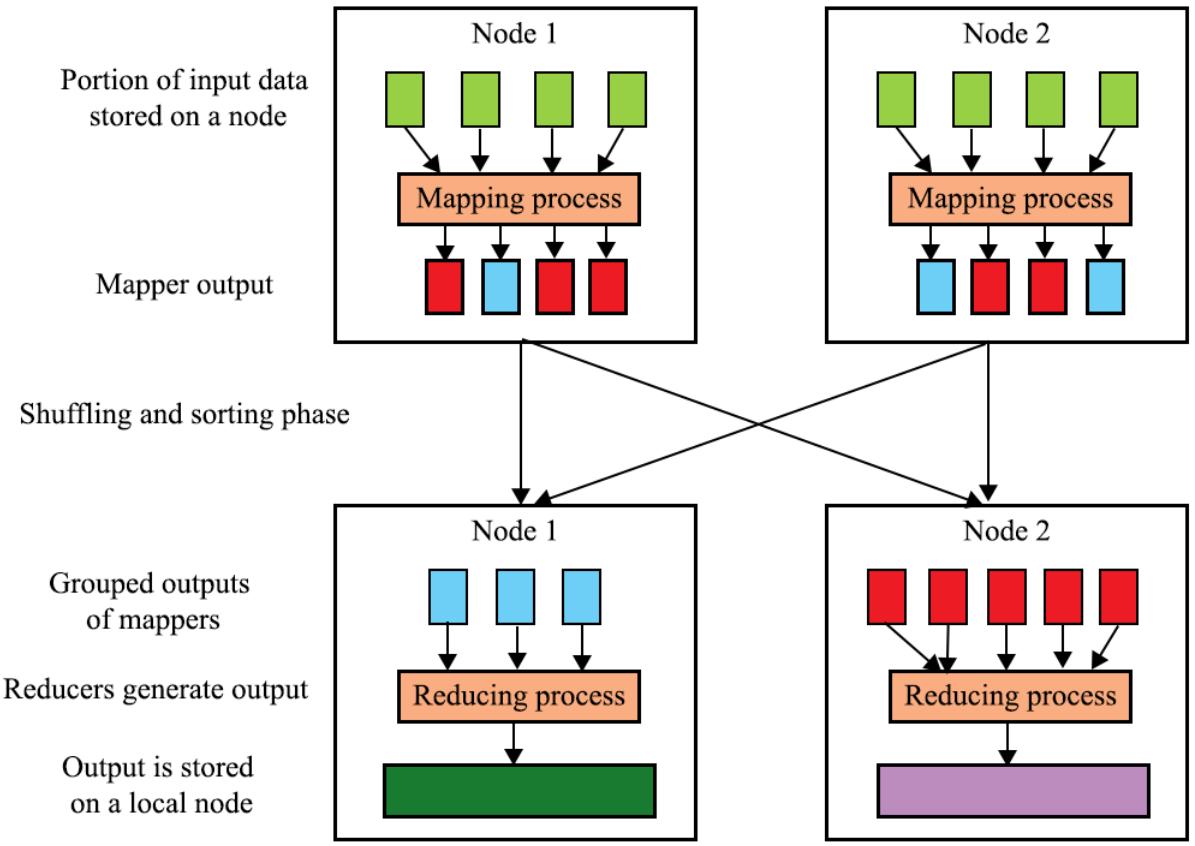

TABLE 3.2: MapReduce data flow (Source: [5])

- Firs of all, the input data is divided into separate partitions.
- Next, Map phase begins: the framework assigns each data partition to a specific node and spawns Map tasks on the nodes. During Map phase data is transformed and returned as a key-value entries.
- During Mapping phase, intermediate Shuffling and Sorting phases are continuously called that fetch Mapping functions' outputs and group key-value pairs in groups by key. As the result, Reducer function gets input data in a form of `<key, (list of values)>` [35].
- Afterwards, each group is processed by Reduce function and output is stored in an output file.

One of the main impediments to using MapReduce is that it is not trivial to implement all required algorithms in a form of MapReduce jobs. In a pursuit to overcome this issue there were developed frameworks that provide abstractions on top of MapReduce such as Apache Pig and Hive [4]. The summary of their characteristics is provided in Table 3.3.

Properties	Apache Hive	Apache Pig
Language	HiveQL (SQL-like)	Pig Latin (script-based language)
Type of language	Declarative (SQL dialect)	Data flow
Data structures	Suited for structured data	Scalar and complex data types
Schema	Enforced during read time	Completely optional
Data access interface	JDBC, ODBC	PigServer

TABLE 3.3: Comparison of available distributed data querying engines (Source: [6])

3.2.3.2 Apache Pig

Apache Pig is a framework built on top of MapReduce engine which introduces a data flow scripting language called Pig Latin which can be used to issue commands to a interactive shell. Pig Latin is meant to decrease complexity and duration of development cycle of MapReduces programs. As opposed to Apache Hive, Pig is able to process unstructured data without any schema [6]. Even though Pig Latin operators resemble SQL queries, there are several key differences. Firstly, SQL is a declarative programming language and Pig Latin is dataflow langauge, that is, it is composed of a steps where each step is a single data transformation. In case of SQL queries, user defines list of constraints that when applied together and some output is returned. Moreover, Pig Latin has better support for nested data structures as opposed to SQL which is more suitable for flat structures [34].

3.2.3.3 Apache Hive

Apache Hive is a data warehouse and data analytics solution that runs on top of Hadoop cluster using SQL-like language - HiveQL. It further abstracts users from underlying low-level MapReduce code. It allows users to access and manipulate data stored in HDFS or HBase. Unfortunately, Hive is not meant for real-time processing: it suffers from big latencies even in case of simple queries and small volumes of data, since it converts SQL queries into actualy MapReduce jobs. What is more, it is not suitable for building complex machine-learning algorithms. Apache Hive is built to guarantee scalability, fault-tolerance in context of big data analysis, exploration [34]. A notable feature of Apache Hive is the fact it enforces schema at the time it reads the data (when the engine loads it for further processing), not when the data is written into the store [6]. As distinct from traditional Relational Database Management Systems that enforce

schema when data is written into the store, such approach gives additional degree of flexibility which is especially relevant in the context of big data where variety of data is pretty substantial.

3.2.4 Apache Spark

Apache Spark is a general-purpose engine for distributed large-scale data processing. Apache Spark exposes a comprehensive and efficient programming model for batch and streaming analytics. It has built-in libraries for running SQL queries, machine learning and graph processing workloads along with functional programming API in several languages such as Scala, Java, Python and R. Spark has integration with several cluster management frameworks: standalone, Hadoop YARN, Apache Mesos and there is ongoing effort to provide support for Kubernetes deployments [6]. It also can run locally.

The distinct Apache Spark feature is in-memory processing: it is able to store intermediate computation results in memory which makes it efficient for iterative algorithms which are common in machine learning and graph processing. In case of MapReduce jobs, all intermediate computation results were written to external distributed file system and memory was not used to the full extent. Internally, much like Apache Tez execution engine, Apache Spark uses graph-oriented model where job is represented as directed acyclic graph where nodes are computation units or Resilient Distributed Datasets. Spark's advanced execution engine may make in memory program executions up to 100 times faster and disk executions up to 10 times faster [36].

3.2.5 H2O

H2O is an open source distributed data processing and analytics framework. H2O provides wide set of algorithms including generalized linear models, gradient boosting machine, random forest, naive bayes, deep learning and many more[7]. It provides API in Scala, Java, Python and R. Underneath, H2O uses its own Map/Reduce engine built on top of Java stack [37]. H2O can also be deployed on a Hadoop cluster, however, H2O strives to perform computations in memory as much as possible. What is more, H2O provides a library Sparkling Water which integrates H2O machine learning algorithms with Apache Spark engine which enables Apache Spark users to take advantage of having wider range of machine learning algorithms. Moreover, H2O framework comes with

[6] https://github.com/apache-spark-on-k8s/spark
[7] http://docs.h2o.ai/

web-based interactive notebook environment[8] for execution of ad-hoc scripts and data visualization.

3.2.6 Apache Mahout

Apache Mahout has been until recently Hadoop MapReduce-only machine learning framework. However, the project goals were reconsidered [9] and the project is currently in a migration phase [10] from MapReduce to the new DSL on top of Scala. The framework uses Apache Spark as a back end processing engine. The framework is focused on providing math operations for linear algebra and statistics as well as some machine learning algorithms such as collaborative filtering, classification, clustering, dimensionality reduction.

3.2.7 XGBoost

XGBoost is a distributed gradient tree boosting library. Boosting is an ensemble learning algorithm which aims to built many weak classifiers, with each new model trying to correct the flaws of the previous model. XGBoost aims to provide efficient and scalable implementation for parallel tree boosting algorithm [7]. XGBoost supports GPU-powered processing and can also run in distributed environment, for example, Hadoop YARN. XGBoost offers Python, R and Julia API. The comparison table of major gradient boosting implementations is shown in Table 3.4. XGBoost introduces its own strategy for writing data on disk and reading it in efficient manner for out-of-core computations. In order to boost computational efficiency in case of large datasets that do not fit in memory, they offer global and local approximation methods for optimal splitting point search. In addition, to XGBoost introduces its own built-in sparsity-aware approach to tackle the issue of missing data in datasets [7].

[8] http://docs.h2o.ai/h2o/latest-stable/h2o-docs/flow.html
[9] https://issues.apache.org/jira/browse/MAHOUT-1510
[10] http://mahout.apache.org/users/basics/algorithms.html

System	Parallel	Exact greedy	Approximate global	Approximate local	Out-of-core	Sparsity aware
XGBoost	yes	yes	yes	yes	yes	yes
Spark MLLib	yes	no	yes	no	no	partially
H2O	yes	no	yes	no	no	partially
scikit-learn	no	yes	no	no	no	no
R GBM	no	yes	no	no	no	partially

TABLE 3.4: Provided features comparison table of XGBoost and other gradient boosting solutions (Source: [7])

3.2.8 Data streaming

Nowadays there is a growing interest towards stream processing frameworks [38] which complement batch processing systems and form together so-called Lambda architecture. This term describes a system which processes large volumes of data using both batch and stream processing methods. The main goal of streaming layer is to make the most recent data available to users within very low time frames. While the batch layer tries to produce as accurate results as possible, the speed layer aims for performance: even though the results produced by the streaming layer may not be as precise as those produced by batch layer, they provide view on the most recent data [4]. The examples of streaming datasets include[11]:

- End users interactions with mobile or web applications
- Physical sensors providing measurements (Internet of Things data)
- Data received from financial markets
- Log data

3.2.8.1 Apache Flink

Apache Flink is a dataflow engine that supports both high-throughput streaming and batch workloads. It was designed to provide an alternative to micro-batch streaming solutions. It has functional programming API for Java and Scala. It also has integration with MapReduce stack: it can use both HDFS and MapReduce. Apart from Apache Spark, Flink does not use micro-batch approach, but it dispatches single events, as does Heron, thus achieving true real-time processing [4]. Apache Flink also provides its

[11] https://flink.apache.org/introduction.html#continuous-processing-for-unbounded-datasets

own optimizer which analyzes the code to create semantically same but more efficient execution plans. In order to achieve fault-tolerance, Flink makes consistent snapshots of the data stream as well as operator states which it can use for recovery from failures [39]. As for message delivery guarantees, Flink guarantees exactly-once delivery. Flink has integration with various messaging queues, for instance, RabbitMQ, Apache Kafka, user can also provide their own implementation.

Apache Flink has machine learning library called FlinkML. Unfortunately, the amount of supported algorithms is pretty limited [12]. It is useful for the use cases which deal with continuously evolving massive quantities of data, for instance, spam detection.

In a study, Flink was compared to Apache Spark Streaming where it turned out that Apache Spark showed better degree of fault tolerance as well as better support for iterative algorithms while Flink had better performance, however, it consumed more resources [4].

3.2.8.2 Apache Spark Streaming

Apache Spark Streaming is a stream-processing component which is built on top of the Spark SQL engine. Spark Streaming supports 2 modes of processing incoming streams: micro-batch processing and continuous processing. Apart from micro-batch processing, continuous processing mode is able to achieve end-to-end latencies of a few milliseconds [40]. Spark Streaming is exposed to the user through Spark DataFrame API, thus streaming logic is expressed in the same way as batch processing logic. The core concept of Spark Streaming relies on an unbounded table where the input data continuously arrive [41]. The developer defines queries and transformations for the DataFrame as in case of usual static data. Due to the unified API, developers have other Spark libraries at their disposal. The component is able to ensure exactly-once mode of processing.

3.2.8.3 Apache Heron

Heron[13] is a scalable stream-processing engine which was created by Twitter in order to replace their previous stream processing solution Apache Storm. Apache Storm possessed numerous drawbacks that were hard to fix without complete rewrite. However, they made an effort to keep compatibility with the existing Apache Storm API in order to reduce impact of such migration [42].

[12] https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/libs/ml
[13] https://apache.github.io/incubator-heron/

The core structure of Heron is topology, that is directed acyclic graph of spouts and bolts. Spouts are connected to data sources such as Apache Kafka or Kestrel while bolts process incoming data. These topologies are deployed to a generic service scheduler (which may run on top of YARN, Mesos and Amazon EC2 Docker Container Service). Each topology runs is composed of several containers: much like in RDBMS, before execution of a topology, engine generates a physical execution plan taking into consideration parallelism settings for spout and bolt tasks.

Another interesting feature of Heron is backpressure mechanism which dynamically adjusts the rate at which data is processed by the topology by blocking its spouts. This strategy is important to increase the computing efficiency and stability of environments where different components may process data at different speed [42]. Heron provides two modes of processing:

- At least once - engine guarantees that each message is processed at least once, however, they may also be processed several times. In this mode of processing, it is crucial to ensure idempotent message processing.
- At most once - some incoming messages may be dropped, but all of them will not be processed more than once.

3.3 Deep learning libraries

Since recently, deep neural networks have become one of the most promising areas of machine learning. The general structure of neural networks was loosely inspired by biological neurons and their connections. Deep learning has been successfully applied to a number of fields, for example, computer vision, natural language processing and speech recognition. Due to computation-intensive nature of the training process and relatively simple operations performed during this process, it is sensible to perform these computations using GPUs that are suitable for this kind of workloads. Executing deep learning workloads on GPU becomes orders of magnitude faster. Computations can be further accelerated by distributing them across GPU cluster [43].

3.3.1 TensorFlow

TensorFlow is an open-source machine learning library with focus on deep learning which was developed by Google Brain, an internal research team at Google. It is meant to replace its old predecessor - DistBelief. Besides running on both CPUs and GPUs, Tensorflow supports both single-node as well distributed workloads.

TensorFlow offers a dataflow programming interfaces using which users can compose training pipelines with various phases some of which are reading input data, preprocessing it, training and checkpointing state etc. Program execution can be divided into two major stages: (i) building whole dataflow graph based on the provided source code, (ii) building optimized version of a computation graph and its subsequent execution [44].

What is more, TensorFlow can be run in heterogeneous cluster which includes CPUs and GPUs as well as ASICs that are specialized for deep learning workloads [44]. Although, such ASICs are not yet currently commercially available, Google is in process of building TensorFlow Research Cloud which would make these ASICs available to the wide audience[14].

When it comes to fault-tolerance in cluster environment, Abadi et al. state that deep learning training process does not necessarily require strong consistency, so they provide several replication strategies each with either consistency or performance trade-offs [44].

TensorFlow provides API in wide range of languages, for instance, Java, C++, Go and Python.

3.3.2 Caffe2

Caffe2 is a successor to Caffe deep learning framework. Caffe was originally focused mostly on Convolutional Neural Networks in a single-node environment. This fact influenced design decisions of the framework and made it harder to adapt to other use cases such as distributed training, non-vision use cases, operation in mobile environment. It is stated, that Caffe2 is focused on execution on mobile environments and on supporting battle-tested algorithms [45]. Caffe2 engine is exposed through Python API. Unfortunately, as of today, Caffe2 has not yet been thoroughly studied by the academia.

3.3.3 PyTorch

Both PyTorch and Torch use same back end engine for low-level computations, however, PyTorch provides API in Python while Torch is a Lua wrapper. These frameworks have an imperative programming model which allows users to heavily optimize execution process of the programs. It is an useful tool for power users, but it lacks in terms of portability across different environments [44]. However, imperative way of programming brings its own advantages, such as more straight-forward debugging and meaningful exception stack traces. What is more, PyTorch provides greater flexibility

[14] https://www.tensorflow.org/tfrc/

in terms of neural network structure. Unlike frameworks such as TensorFlow, Theano, Caffe or CNTK, PyTorch provides an option to restructure existing neural network without starting training process from the scratch [46]. Such level of flexibility fosters experimentation and is great for trying out new architectures for neural networks [45]. As well as other deep learning libraries, it supports GPU acceleration.

3.3.4 MXNet

MXNet is a deep learning library which combines declarative and imperative programming approaches when appropriate. The creators of MXNet claim that declarative approach is suitable for describing structure of neural networks via computation graphs, whereas imperative programming brings more advantages in case of parameters updates (tensor computations). Besides, the engine also offers auto differentiation to derive gradients which allows to define a new neural network structure each iteration while running forward computation[47]. More efficient auto-differentiation is achieved with declarative approach [48]. The library is able to run on heterogeneous systems: mobile devices, CPU/GPU powered devices and distributed GPU clusters (parallelism is achieved with the generic dependency engine[15] which schedules the operations for execution if their dependencies are computed) [8].

System	Core lang	Binding lang	Devices (beyond CPU)	Distributed	Imperative program	Declarative program
MXNet	C++	Python/R/ Go/Julia	GPU/Mobile	yes	yes	yes
TensorFlow	C++	Python	GPU/Mobile/ ASIC	yes	no	yes
PyTorch	C, C++	Python	GPU	yes	yes	no
Caffe2	C++	Python	GPU/Mobile	yes	no	yes

TABLE 3.5: MXMNet comparison with other popular open-source deep learning libraries (Source: [8])

3.3.5 Keras

Some deep learning libraries have sophisticated API: even though they might be quite flexible, their API may be very low-level and tedious to work with. Therefore, there was developed a library called Keras [16] which provides high-level API on top of TensorFlow,

[15] https://mxnet.incubator.apache.org/architecture/note_engine.html
[16] https://keras.io/

CNTK and Theano. Keras enables rapid development and experimentations without going too deep into details of underlying neural network architectures. Keras provides Pytohn API.

3.4 Survey

Due to the wide variety of tools available for data analytics, it is not obvious what set of tools Waldur should provision first through its self-service. Therefore, we decided to conduct a survey to identify what data analytics tools Estonian data analysts and academia researchers use.

The questionnaire was spread among both Estonian researchers from academia and practitioners from industry. We contacted each person only once, through a personal email letter. The questionnaire was done in Google Docs. The answers were collected in November, 2017. While each question was supplied with predefined set of answers, people were given an option to write additional answers. Respondents could select multiple choices in each question. Responses are anonymous. In total, we garnered 32 responses. To report the percentages, we round values to a whole number.

The following table 3.6 shows what organization the respondents belong to. While most people belong to Estonian universities and other affiliated with the Estonian Scientific Computing Infrastructure research groups, we managed to reach also industry practitioners.

What organization do you belong to?		
Research group from the ETAIS members (UT, KBFI, TUT, HITSA)	25	78%
Private company	6	19%
Public sector (non-academic)	1	3%
Research group from non-ETAIS members public university	0	0%

TABLE 3.6: Survey responses to question 1: What organization do you belong to?

Another important fact that we should know about our target users is their programming language preferences 3.7. It turned out that majority of people (75%) have programming experience with Python. About 62% of researchers also know R. Third popular language is Java, however, it is much less popular than previous two languages.

What programming language do you feel comfortable with?		
Python	24	75%
R	20	63%
Java	11	34%
C++	7	22%
C	4	13%
C#	2	6%
Bash	2	6%
Matlab	2	6%
Scala	2	6%
Wolfram Mathematica	2	6%
Elixir	1	3%
F#	1	3%
Fortran	1	3%
Julia	1	3%
J language	1	3%
Octave	1	3%
Php	1	3%
Stata	1	3%

TABLE 3.7: Survey responses to question 2: What programming language do you feel comfortable with?

Third question 3.8 was about usage of data analytics tools. We tried to mention here mostly self-contained engines, since many of them provide API in languages mentioned in the first question 3.6.

When it comes to single-node tools, not surprisingly, Python-based tools are used by the most people. We will consider, that R-based packages are used by all respondents who mentioned R in the first question. This makes R packages second most popular tools. On the third place is Weka.

The results imply that distributed solutions are not so widely used as single-node tools. The most notable among them is Apache Spark which is used by 21% of respondents. After it goes XGBoost which is used by 4 people. Apache Hive and Hadoop MapReduce is used just by 9% of respondents.

What data analysis toolkits do you use?		
Scikit-learn	23	72%
Scipy stack	18	56%
Apache Spark	8	25%
Weka	7	22%
XGBoost	4	13%
Apache Hive	3	10%
Hadoop MapReduce	3	10%
Accord.NET	1	3%
Apache Flink	1	3%
Microsoft ML studio	1	3%
OpenCV	1	3%
Yolo	1	3%
JSoftware	1	3%
Wolfram Mathematica	1	3%
SQL Service Analysis Services	1	3%
Nothing from the list	1	3%
Apache Pig	0	0%
Apache SAMOA	0	0%
Apache SystemML	0	0%
GraphLab	0	0%
H2O	0	0%
Oryx 2	0	0%

TABLE 3.8: Survey responses to question 3: What data analysis toolkits do you use?

In the fourth question 3.9 we asked if respondents use deep learning libraries and which ones exactly. The most popular deep learning framework is TensorFlow: half of the respondents use it. It would be fair to conclude that the most researchers do not actually use TensorFlow to its full extent, but rather use Keras which is a wrapper library for TensorFlow and Torch. One third of pool respondents do not use deep learning in their everyday work. Even though Theano and Caffe are considered to be deprecated, still they reside on 4th and 5th places respectively.

What deep learning libraries do you use?		
TensorFlow	17	53%
Keras	16	50%
I do not use deep learning libraries	11	34%
Theano	8	25%
Caffe	5	16%
PyTorch	3	10%
Torch	3	10%
Caffe2	2	6%
MXNet	2	6%
PaddlePaddle	1	3%
Wolfram Mathematica	1	3%
dynet	1	3%
Deeplearning4j	0	0%
CNTK	0	0%

TABLE 3.9: Survey responses to question 4: What deep learning libraries do you use?

Finally, in the last question we asked if respondents have any suggestions or comments. By the end of the survey, we received 5 opinions. 2 persons mentioned that they heavily rely on GPU-based computing. One researcher pointed out that having the latest versions of libraries would be very nice, especially if this kind of infrastructure was kept up to date. This request probably originates from the fact that HPC cluster of the Estonian Scientific Computing Infrastructure provides Slurm[17] batch processing payloads where execution environment is not very flexible: available execution environments are defined by Environment modules [18] and these are maintained by the system administrators.

3.5 Discussion

Collected answers to the first question 3.7 prompt that it makes sense to choose first either Python or R. Furthermore, we observed that everybody who knows R also know Python. It was not the case vice versa.

As for general-purpose data analytics toolkits 3.8 we can conclude that Python-based libraries are the most widely used. The most popular distributed learning framework is Apache Spark.

[17] https://slurm.schedmd.com/overview.html
[18] http://modules.sourceforge.net/index.html

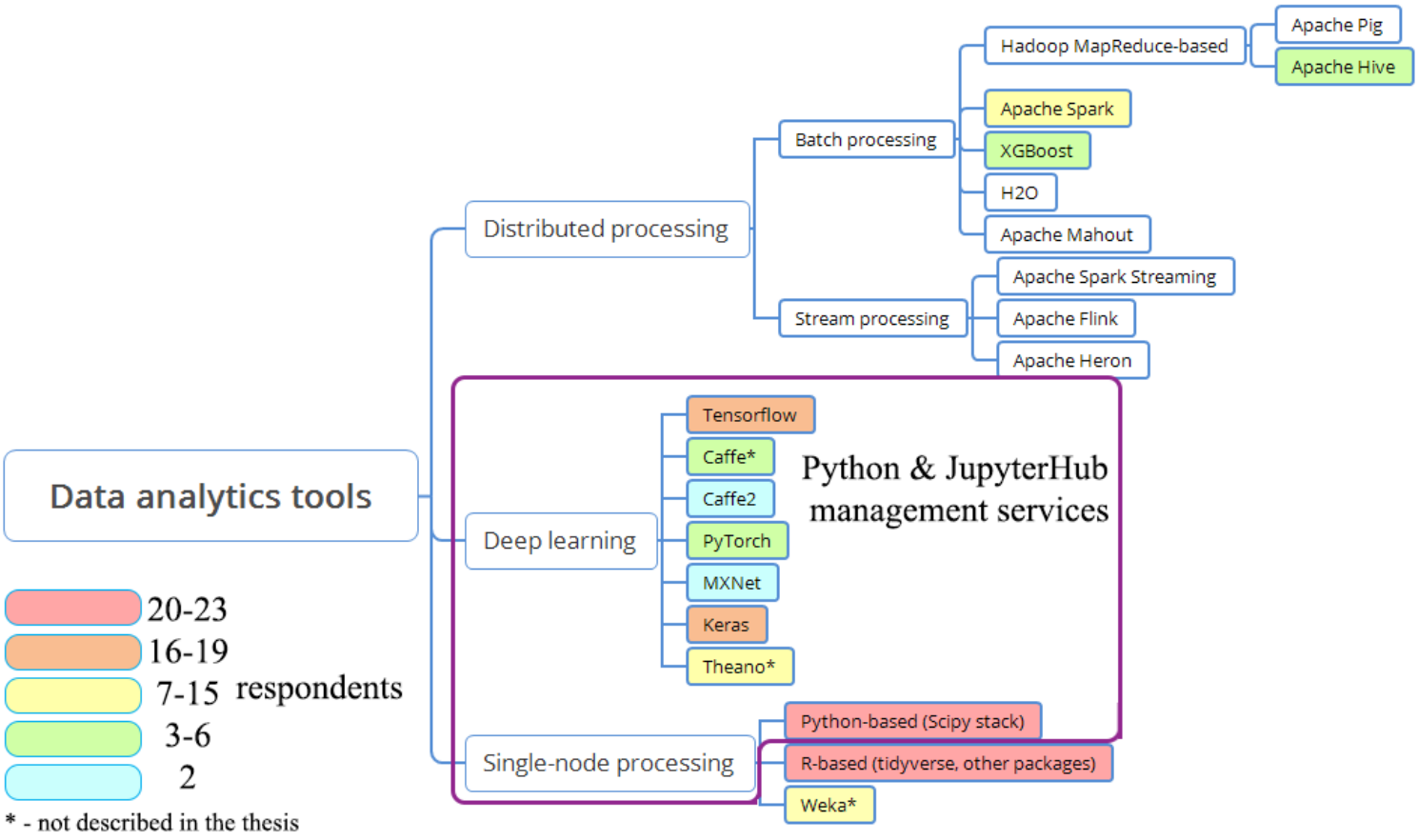

FIGURE 3.1: Data analytics frameworks with survey results

Data collected about deep learning frameworks 3.9 clearly tells us that libraries with Python API are the most numerous and at the same time the most commonly used. Very few people selected libraries that do not come with Python API (e.g Torch, Deeplearning4j).

Based on the literature analysis and on survey results we can conclude that it is makes sense to provision Python-based data analytics tools through Waldur self-service. Python is not only the most widely used programming language among researchers, but there are also a wide range of advanced and diverse set of Python-libraries that support data mining, machine learning and deep learning workloads. These aspects make Python very powerful programming language for data analytics. Such service will bring immediate benefit to all researchers who are interested in doing data analytics using Python-based tools in the cloud.

As shown in Figure 3.1, out of all data analytics tools that we described, the systems will cover Python-based single-node processing and deep learning use cases (deep learning will be even more fully supported if appropriate machine images with installed CUDA drivers are used).

3.6 Conclusion

Nowadays data analytics can be performed using great variety of tools. In this chapter we described tools which are used in different domains. These engines were classified into several categories depending on the set of features they provide. Besides, the tools were compared with each other to identify their capabilities and limitations. The choice of the appropriate engine depends to a great extend on the problem which it intends to solve. For example, for performing comprehensive analysis of large datasets it is worth to look at available batch processing engines. If there is a necessity to generate results from the incoming data flow in real-time, then one should consider stream processing tools. Deep learning libraries are positioned to be suitable for implementing computer vision and speech recognition systems. To sum up, combination of all these technologies will support more intelligent decision making in industry and academia.

Furthermore, we conducted a survey throughout which we collected data about usage of various data analytics engines among Estonian researchers. The survey findings showed that there is a great interest towards deep learning engines, most notably, Tensorflow. The main programming languages used by Estonian researchers are Python and R. Awareness of the available data analytics tools and the gathered survey data helped us to make the informed decision about what areas our new cloud services should first cover. We decided to focus on providing support for Python-based single-node processing workloads.

Chapter 4

Services for management of data analytics tools

4.1 Introduction

In the Chapter 3 we identified state of the art data analytics frameworks and revealed what tools are used by Estonian researchers by conducting a poll. Based on this knowledge, in the Chapter 4 we intend to answer the second research question mentioned in Chapter 1: **RQ-2: How to automate provisioning and management of data analytics tools through hybrid cloud brokerage platform?** We divide the research question into three subquestions:

- What are the requirements of the application provisioning services? - Section 4.2
- How to design and implement the services for provisioning of Python development environment and JupyterHub? - Sections 4.4 and 4.5
- How to design and implement integration tests for the application provisioning services? - Section 4.6

The source code of the produced artifacts is available on GitHub in Waldur repositories. Application back end logic: `https://github.com/opennode/waldur-ansible`, Ansible playbooks: `https://github.com/opennode/ansible-waldur-module`, front end logic: `https://github.com/opennode/waldur-homeport`.

4.2 Requirements for the application provisioning services

In this section we explain what tools we intend to provision and compose the scenarios that the services should fulfill.

We decided not to build our services in accordance to Platform as a Service service model, since we place particular focus on modifiability aspect of the provisioned tools in order to ensure that users would not be strictly limited to only the provided configuration options, rather users should be able to adapt the default configuration according to their needs. As a result, we came to a decision to build the services which would focus on deployment stage of well-known open-source data analytics tools and their subsequent configuration management on a remote virtual machine.

Jupyter[1] is a environment which provides interactive computing capabilities in various programming languages. It allows to execute the programs on remote machines through convenient web-based user interface. Many researchers use interactive notebooks to present their work and share it with others. It covers a whole spectrum of what Python provides: anything from simple scripting, to more sophisticated notebooks which combine runnable code, data visualization and rich text. We believe that provisioning JupyterHub is a good choice due to its popularity and great extensibility: Jupyter supports many programming languages (Python, R, Scala) and there are many third-party extensions available for it.

JupyterHub[2] is a central server which provides authentication and multi-tenant access to Jupyter notebooks. We decided to automate provisioning process of JupyterHub deployment since it provides most needed authentication options to be able to use Jupyter either when working in a team or for organizing workshops. Moreover, deployment of JupyterHub may be quite challenging and error-prone process for people who do not have much Linux administration experience.

Furthermore, we complement JupyterHub provisioning service with Python management service which provides functionality to install Python libraries with specified versions into isolated Python virtual environments[3] through Waldur self-service. This service will allow users to conveniently manage their development environment through self-service and be able to switch between them in Jupyter notebooks. We chose not to provision any predefined list of Python libraries, since the list may become obsolete quite soon, especially the selected versions of the libraries. Instead, we decided to provide a service

[1] http://jupyter.org/
[2] http://jupyterhub.readthedocs.io/en/latest/
[3] https://virtualenv.pypa.io/en/stable/

which would allow users to configure their development environment on their own and do not depend on Waldur maintainers.

As depicted in Figure 4.1, the life cycle of a managed application can be composed of three stages:

- **Initial application provisioning:** it is the most time-consuming step since it involves required software installation. The duration greatly depends on the various factors such as network bandwidth, availability of remote installation packages and virtual machine computing resources.
- **Application management:** in this stage the application is in operational state and the system should provide an option to modify installed software configuration without a complete reinstallation from the scratch due to the time-consuming nature of the process. During this stage the system should ensure that all the necessary packages are indeed properly installed.
- **Application removal:** this is the final stage of the life cycle when the application is undeployed and removed.

To formalize scenarios and ensure that each scenario is well formed, we use Scenario Refinement template described by Barbacci et al. in [49]. However, we do not strictly stick to the template, rather use an adapted version of it which includes following scenario descriptors:

- **Relevant Quality Attributes** - related quality attributes from ISO/IEC 25010[4] standard.
- **Stimulus** - the condition that should be handled by the system
- **Environment** - the context of the stimulus
- **Artifact** - the developed artifact which instantiates response
- **Response** - what actions are taken to cover the described scenario

4.2.1 Python libraries management scenarios

In this section we identify set of scenarios that Python management service should cover. The main aim of the service is to enable users to manage their Python virtual environments through a user-friendly graphical interface.

[4] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010

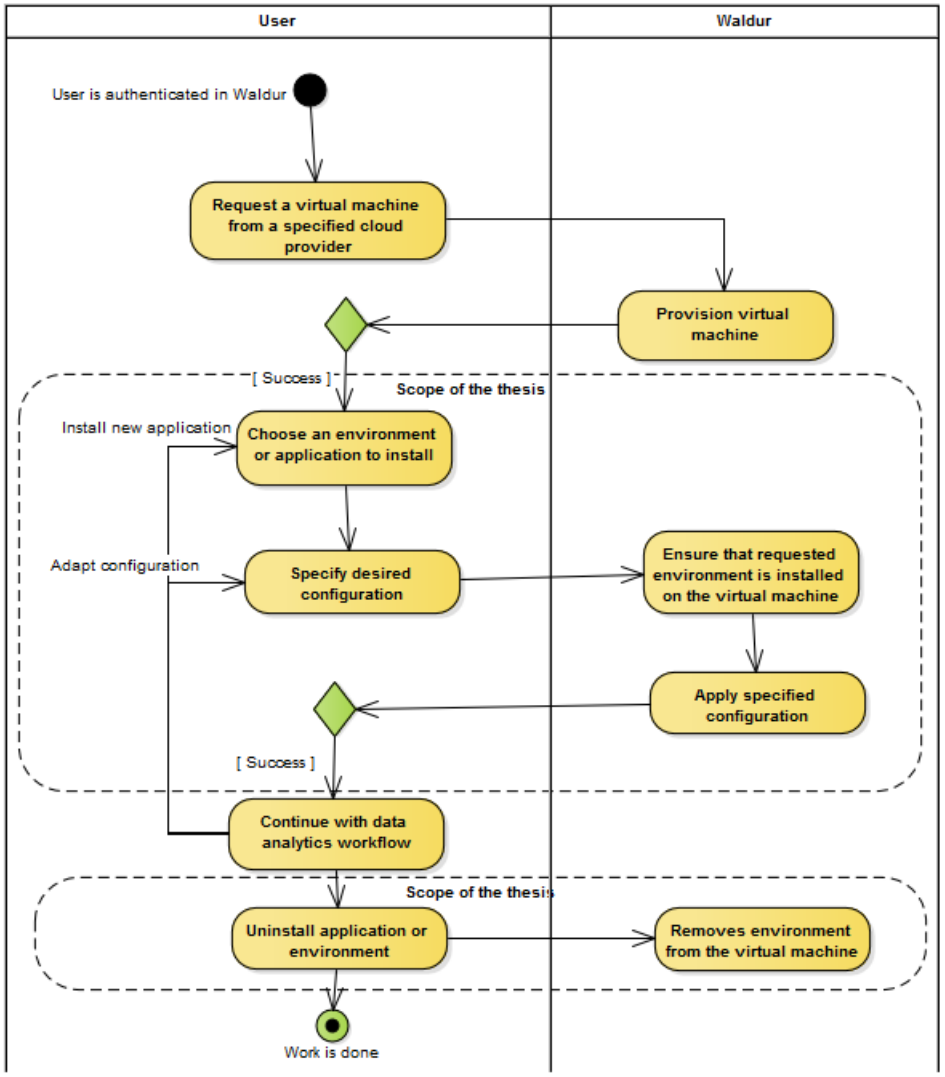

FIGURE 4.1: High-level activity diagram of data analytics provisioning services in Waldur

For the list of the scenarios for Python management service please refer to Appendix A.

It is worth noting that Scenario 2 (described in Table A.2) specifies that libraries should be installed into isolated Python virtual environments. It is important requirement, since it does not only resolve potential conflicts between various libraries (for instance, if a user works on several projects with different set of libraries), but it also provide basic multitenancy support. For example, if several researchers share same GPU-powered virtual machine: each can work in its own isolated environment without affecting other environments (however, the computing resources are shared between all the tenants). Waldur allows users to do that, since there may be multiple users assigned to a single project (in the context of Waldur), thus several users may have access to an instance of Python management service.

As for the selected package manager, we decided to use pip because we wanted to keep installation as light as possible. It means, however, that users will need to specify the libraries that they want to use. Alternatively, we could go with Anaconda distribution [5] which would free users from specifying any additional libraries. On the other hand, it would dramatically increase installation time. To validate this decision, we ask the potential users appropriate question (described Table 4.2) during evaluation stage.

4.2.2 JupyterHub management scenarios

Jupyter is an environment which provides interactive computing capabilities in various programming languages. Many researchers use interactive notebooks to present their work and share it with others. It covers a whole spectrum of what Python provides: anything from simple scripting, to more sophisticated notebooks which combine runnable code, data visualization and rich text.

As a first step towards integration Jupyter nodebooks with Waldur, we intend to ease the burden of JupyterHub server deployment and configuration management. That is, we provide automated deployment and either Linux PAM[6] or OAuth2 configuration options. OAuth2 configuration, powered by oauthenticator[7] extension, will allow researchers to integrate JupyterHub with the existing authentication methods used at their place of work. Furthermore, JupyterHub provisioning service is integrated with previously described Python management service in Section 4.2.1.

All the scenarios for JupyterHub management service are listed in Appendix B.

4.3 Waldur architecture

Before we proceed any further, it is important to describe Waldur architecture: what technologies and patterns are used there.

The Deployment Diagram in Figure 4.2 shows how Waldur components and artifacts are deployed to hardware platforms and how they are connected with each other. Component is a software artifact that works together with other components to implement required functionality which is defined by the system requirements. Please note that Monitoring solution (which collects various metrics from Waldur servers) is purely optional and can be replaced with any other platform. Waldur deployment at the Estonian Scientific Computing Infrastructure currently uses Zabbix.

[5] https://www.anaconda.com/what-is-anaconda/
[6] http://www.linux-pam.org/
[7] https://github.com/jupyterhub/oauthenticator

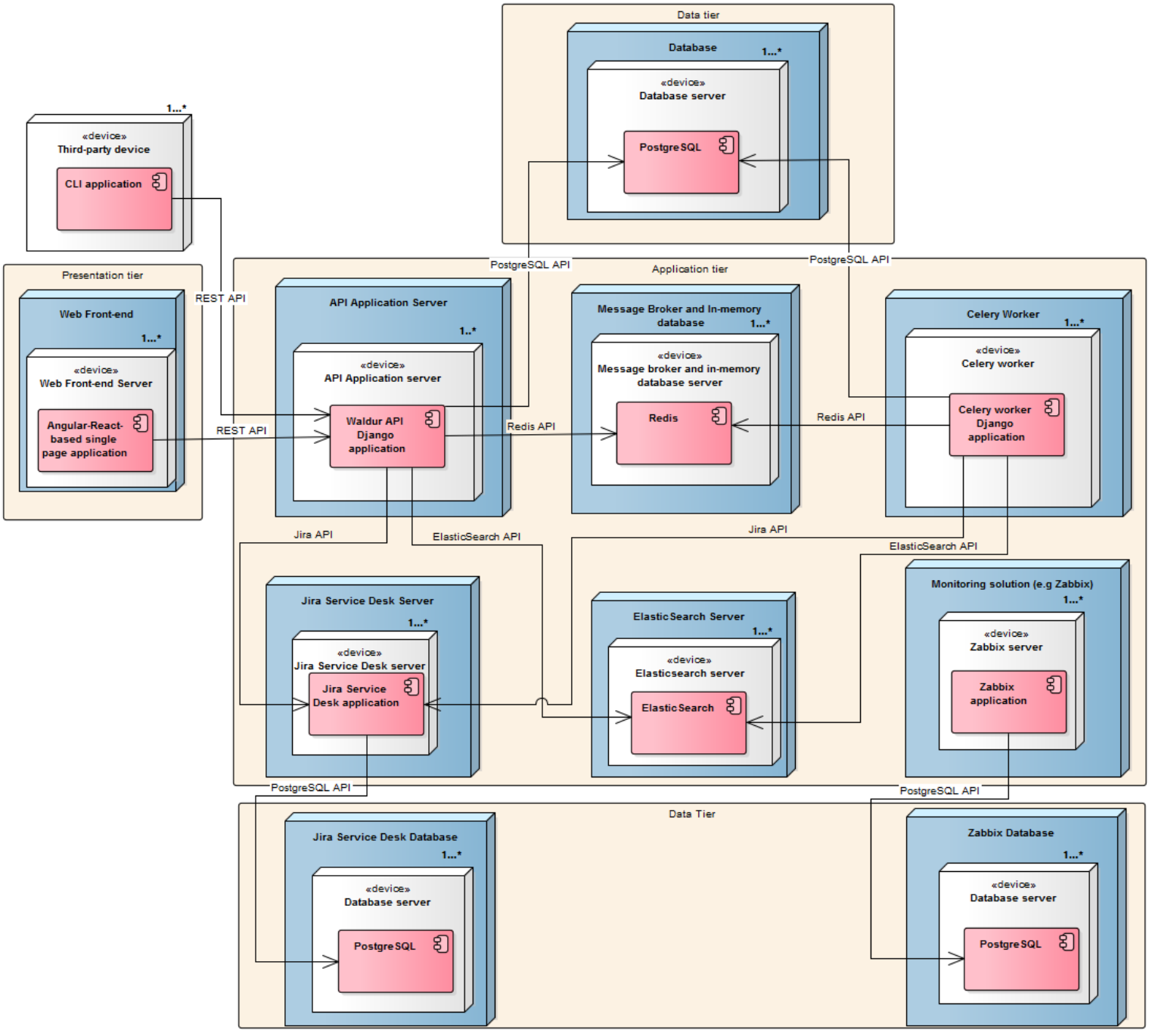

FIGURE 4.2: Waldur deployment architecture (Source: based on internal Waldur documentation)

4.3.1 Three-tier architecture

The system architecture follows 3-tier architecture pattern. Here we present a high-level overview of Waldur architecture without describing internals of each tier.

Presentation tier: It implements user interface which translates the results of backend logic into a user-friendly presentation. Presentation tier of Waldur is a single-page web application which is built on top of AngularJS framework. This tier is currently undergoing massive migration both in terms of used language and stack of technologies: not only JavaScript is being substituted with TypeScript, but also all dialogs are being rewritten on React-based stack of technologies.

Application tier: Application tier is the core of the application. It processes incoming commands, enforces authentication and validation rules, manages transactions, performs calculations and makes logical decisions in the business logic. All interactions with this layer are performed through RESTful API. Source of requests may be the presentation tier or CLI management tools. According to Richardson Maturity Model [50], RESTful API of Waldur corresponds to the second level of maturity. Moreover, application logic layer implements integration scenarios with other systems and executes regular batch jobs that run in the background. The application tier is implemented using Python technologies: the backbone of the application is Django framework. Waldur deployment depends on two applications. Jira Service Desk which is used as a customer support ticketing system.

Data tier: This layer interacts with data persistence mechanisms, for instance, database management systems, in-memory key-value store. Moreover, this layer exposes an API to the application logic tier to decouple it from the underlying storage mechanisms. While Django supports wide range of database systems, Waldur uses in production PostgreSQL as a DBMS. Data tier also contains a solution for fulfilling Waldur audit requirements. Waldur uses ElasticSearch to store history of changes of data in the specific tables of the database. The logging process is implemented using Django ORM callbacks that are called when table row is updated, created or deleted.

4.3.2 Asynchronous tasks execution

Asynchronous Programming model plays major role in Waldur backend logic. The reason is the fact that some cloud management tasks may imply high latencies, especially those workflows which deal with configuring virtual machines. Asynchronous programming model allows calling a potentially long-running logic without blocking the caller. Such strategy allows to release request threads fast and provide rapid feedback to users.

What is more, the separating server instances by different responsibilities (user requests processing, asynchronous and batch tasks execution) provides higher throughput and secures fine-granular scalability for the system architecture. Waldur application layer supports two types of asynchronous tasks:

- Ad hoc jobs - these jobs are usually triggered during processing of user requests.
- Regular batch jobs - this kind of asynchronous tasks are triggered regularly in the background according to the predefined schedule.

Execution of asynchronous tasks is achieved using message-oriented architecture. Waldur utilizes a distributed task queue called Celery. Celery is able to distribute messages across both threads and computer nodes for further processing. Celery uses a message-oriented middleware to deliver messages from clients to Celery workers. The message forwarding strategy follows Point-to-Point channel design pattern [51] (i.e each message is processed only once). Celery deployment strategy is very flexible: it allows to run the whole above-mentioned stack on a single machine which is very convenient for development purposes. What is more, if high availability and performance is of concern, Celery system can be scaled horizontally by adding more worker nodes and expanding the broker [52]. As of today, Celery fully supports RabbitMQ and Redis as message brokers. Waldur uses the latter.

Due to the fact that our services deal with software installation on remote virtual machines which is a time-consuming process, we decided to execute the installation logic asynchronously on Celery workers, rather than on Waldur nodes which serve REST API requests.

4.3.3 High availability and scalability

Every single deployment component of Waldur stack can be independently scaled depending on the amount of managed cloud resources and offered service-level agreements. What is more, if additional fault-tolerance is needed, components may even be distributed across multiple data centers.

This will require to introduce additional component to the deployment strategy between different component groups: load balancers. Load balancers make interactions with cluster of instances transparent: client components do not need to be aware of the existence of the cluster, they always interact with just single API endpoint.

4.4 Python management service architecture implementation

In this section we describe the architecture of Python environment management service and how it is implemented. This service will be afterwards integrated with JupyterHub service which we describe in more detail in Section 4.5.

4.4.1 Prerequisites

There are several important prerequisites that should be fulfilled so that this layer could function.

First of all, the service has only been tested to work with the following Linux distributions: Debian 9 and Ubuntu 16.04 LTS. It is not guaranteed that the service will properly function on any other Linux-based operating systems. However, it might work within the same families of previously mentioned Linux distributions.

As for firewall settings, a user should assign a rule to a virtual machine which allows SSH connections. Naturally, the virtual machine should be also in a security group that allows outgoing traffic, so that it would be possible to download required packages and dependencies.

Another prerequisite is that, the virtual machine should have public key of a Waldur Celery worker. It is needed so that Waldur Celery workers would be able to establish SSH connection without providing any password, and thus without storing any passwords in Waldur database.

4.4.2 Cloud application deployment and management layer

The process of automatic deployment and management of applications in the cloud environment incorporates multiple areas. First, like any unit of functionality, it is implemented using business logic which resides in the application tier. Business logic in turn stores its state in data tier. However, since the ultimate goal is to manipulate state on the virtual machine, business logic should be further enhanced with a new subprocess execution layer which has following responsibilities:

- Maps data obtained from the business logic to CLI command parameters.
- Executes a new subprocess of a configuration management tool with the built parameters.

- Acts upon the output produced by the running configuration management tool.

Artifacts involved in this process include application logic for data mapping and output parsing, the configuration management tool's DSL code files for setting up infrastructure on a target virtual machine.

By the time we started working on the thesis, this new layer had already been introduced. Its description was provided in Section 2.2.1.4. Unfortunately, it did not fully satisfy our requirements, thus we developed for our purposes a new extended layer which is based on the old one.

4.4.3 Ansible - a configuration management tool

Apart from other cloud providers that invented their own infrastructure management tools which were mentioned in Section 2.2.1 (e.g Google Cloud Deployment Manager, AWS CloudFormation etc.), Waldur team has chosen to use Ansible as an underlying configuration management tool for their pilot project of cloud application deployment which is described in Section 2.2.1.4. There are many other open source tools available that fulfill the same purpose (for instance, Chef[8], Puppet[9]): each solutions has its trade-offs and distinctive features, we, however, describe here only Ansible.

Ansible is an open-source configuration management platform which describes the infrastructure deployment process in a form of declarative YAML code. Ansible requires Python 2.7, so does Waldur. Ansible manages machines in an agentless manner: it does not require neither daemon processes, nor databases on both sides to operate: all interactions are performed via standard SSH protocol.

Ansible DSL is instantiated in a form of Playbooks. It is a simple and intuitive interpreted language built on top of YAML. Essentially, Playbooks are composed of sequence of Ansible module calls that perform certain actions on target machines. These Playbooks may be further structured into roles to increase readability and code reusability [53]. Managed machines are defined in the Inventory which can be dynamically expanded with new machines which is especially useful in cloud environment. Moreover, if Ansible does not provide some functionality out of the box, it is possible to either execute a shell command or write your own module. Ansible modules are written in Python. Waldur team has already built several Ansible modules for integration with Waldur API. Being able to describe infrastructure as code greatly reduces maintainability cost, since the

[8] https://www.chef.io/chef/
[9] https://puppet.com/

deployment and provisioning processes are completely automated. As the result, state and configuration of the servers become transparent and stable.

The main weakness of Ansible is that it is still in active development, thus a lot of modules are still not marked as stable, meaning that backwards compatibility is not guaranteed.

4.4.4 Python management service structure

Following class diagram in Figure 4.3 shows static structure of the system with the core classes and their relations. On the diagram each class represents a structure for the data that is stored in the database. These classes are used by application logic to access data.

The design decisions were influenced by the scenarios described in Appendix A: for instance, due to the fact that there may be several requests running simultaneously for the same Python Management (according to Scenario 4 in Table A.4) we decided to create for each type of request its own model, rather than limit ourselves with just PythonManagement model. It allows us to store output produced by Ansible for each possibly simultaneously running request and keep fields specific to a particular request in their own separate classes (as per Single Responsibility Principle [55]). Such design ensures extendability, clear and modular application logic. What is more, since Waldur is a hybrid cloud system, we ensured that our models depend on generic virtual machine model, rather than on concrete cloud providers' virtual machine models (in accordance to Scenario 1 in Table A.1).

We decided to store current state of Python virtual environments in the database, because of the large latencies involved in the process of obtaining installed virtual environments from a virtual machine. That is the reason why executing these search operations on the fly (each time when user visits Python Management screen) is not an option. The disadvantage of storing list of installed Python virtual environments and libraries in the database lies in the fact that this information may become out of sync with the actual state on the virtual machine. For example, if user manually installs a library through SSH. It is pretty challenging to provide a reliable solution to this synchronization problem, so we chose to let users themselves synchronize list of libraries when they think it is needed (see scenario described in Table A.5). In addition, the synchronization with the database takes place after execution of any of requests that modify state of a Python virtual environment.

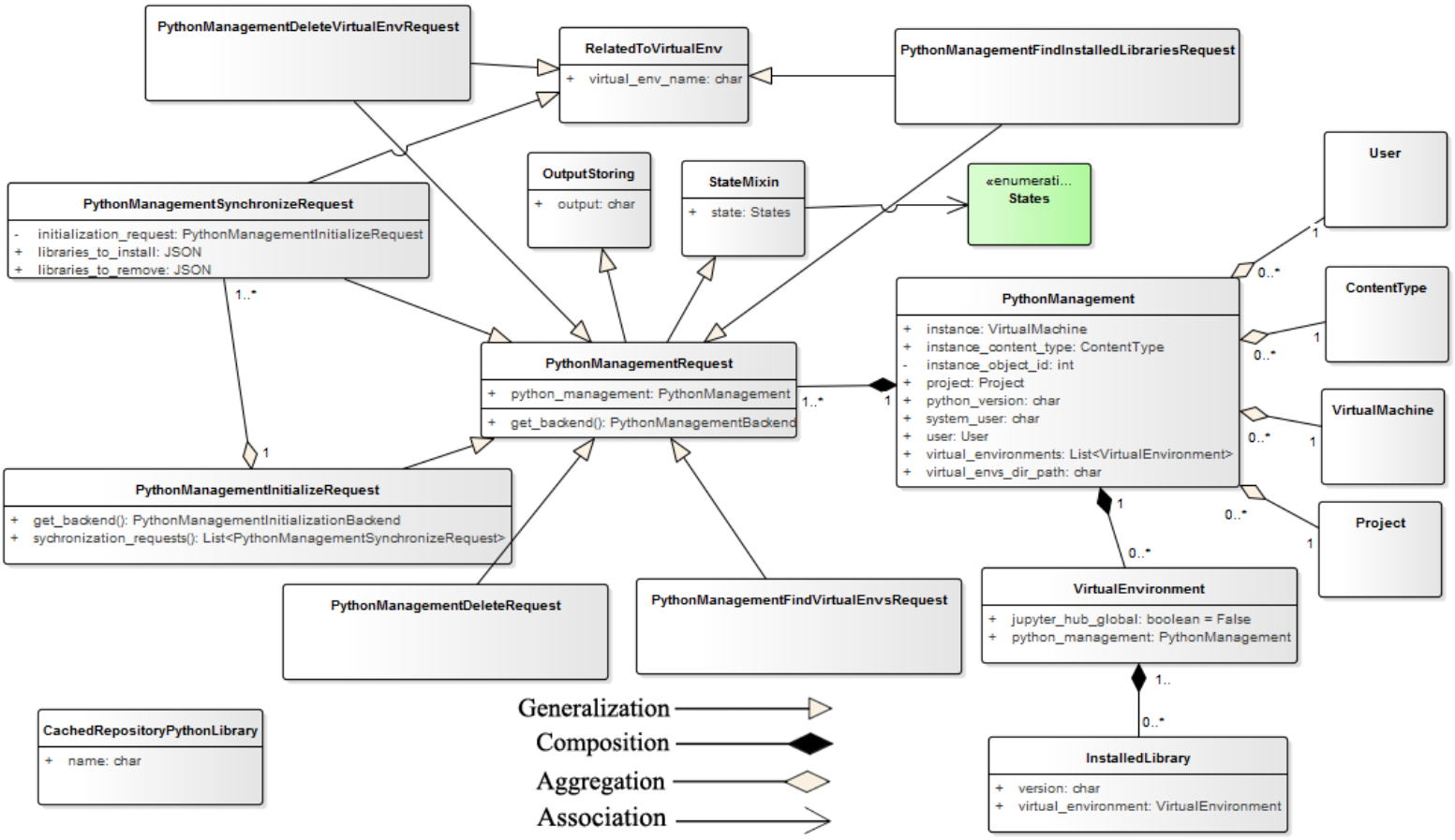

FIGURE 4.3: Python Environment management service class diagram

We chose not to introduce additional batch jobs in Waldur application tier which could regularly scan all virtual machines and bring the data stored in the database into consistent state. There still will be a time frame when the state is inconsistent, furthermore these jobs may take a long time to complete. In fact, the performance of application provisioning process depends on how much computing and memory resources allocated to a virtual machine. This approach may significantly increase hardware requirements of Waldur system.

It is also important to provide the structure of Python Management environment on the virtual machine, so we present a deployment diagram of Python Management environment in Figure 4.4. The diagram shows which tools are installed on the virtual machine and what components are managed upon the environment deployment. The modeling constructs of the following structure diagram are loosely based on ideas of TOSCA standard [25].

As it can be seen in Figure 4.4, Ansible Playbooks called by Waldur application, interact with virtualenvwrapper[10] and local Pip 3 of the particular virtual environment. Python

[10] https://virtualenvwrapper.readthedocs.io/en/latest/

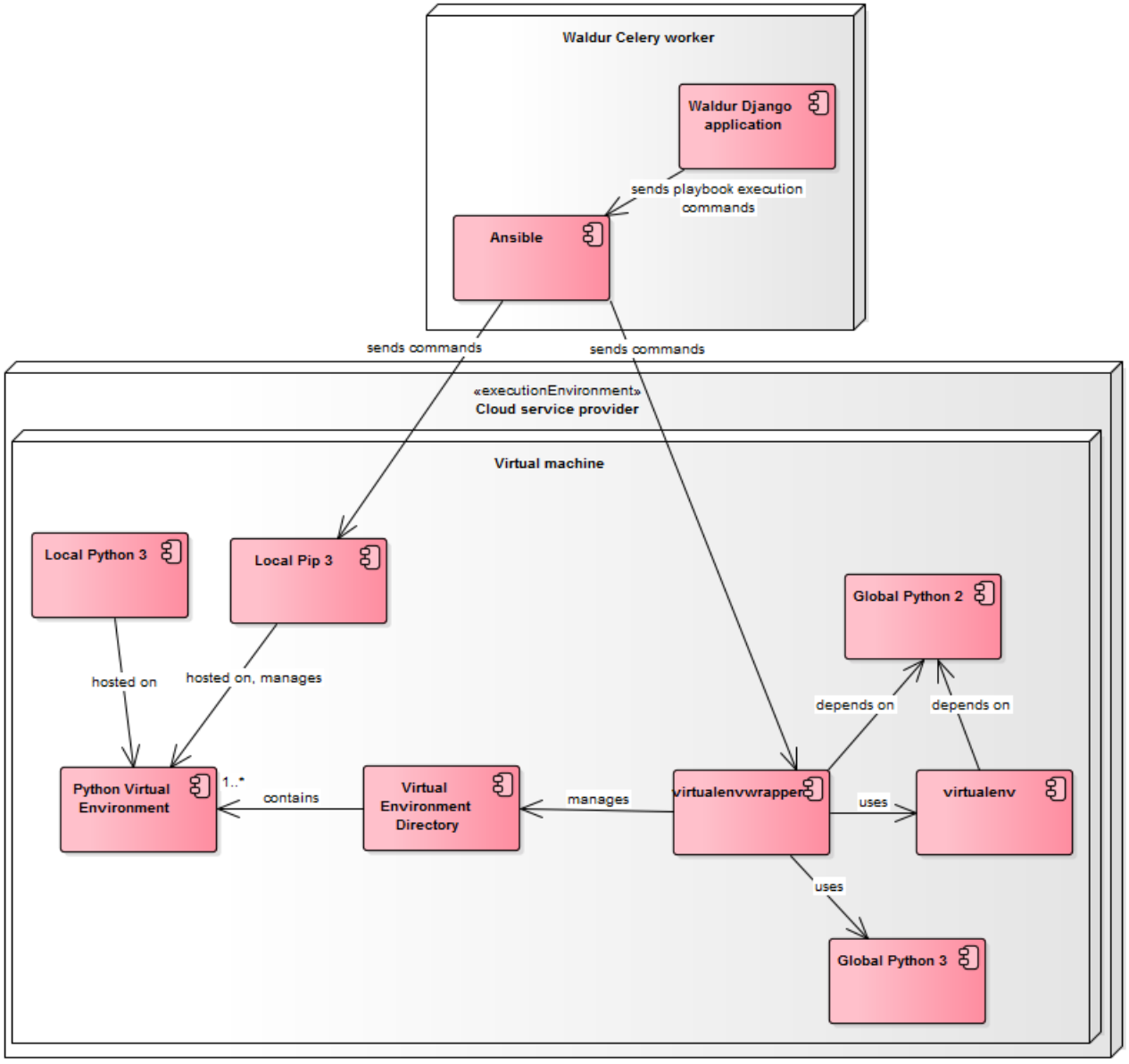

FIGURE 4.4: Python environment management after provisioning on the virtual machine

virtual environment is essentially a directory which holds local Python and Pip binaries. These binary files have access only to the locally installed libraries (we disallow access to globally installed libraries to provide higher degree of isolation). Virtualenvwrapper is an utility program which provides a set of convenient commands to manage Python virtual environments (under the hood it delegates commands to virtualenv[11] tool).

Figure 4.5 shows a Python management service details screen with scenarios described in Appendix A. Not all scenarios are depicted in the figure, since some of them are purely technical and do not correspond to any graphical user interface element.

[11] https://pypi.python.org/pypi/virtualenv

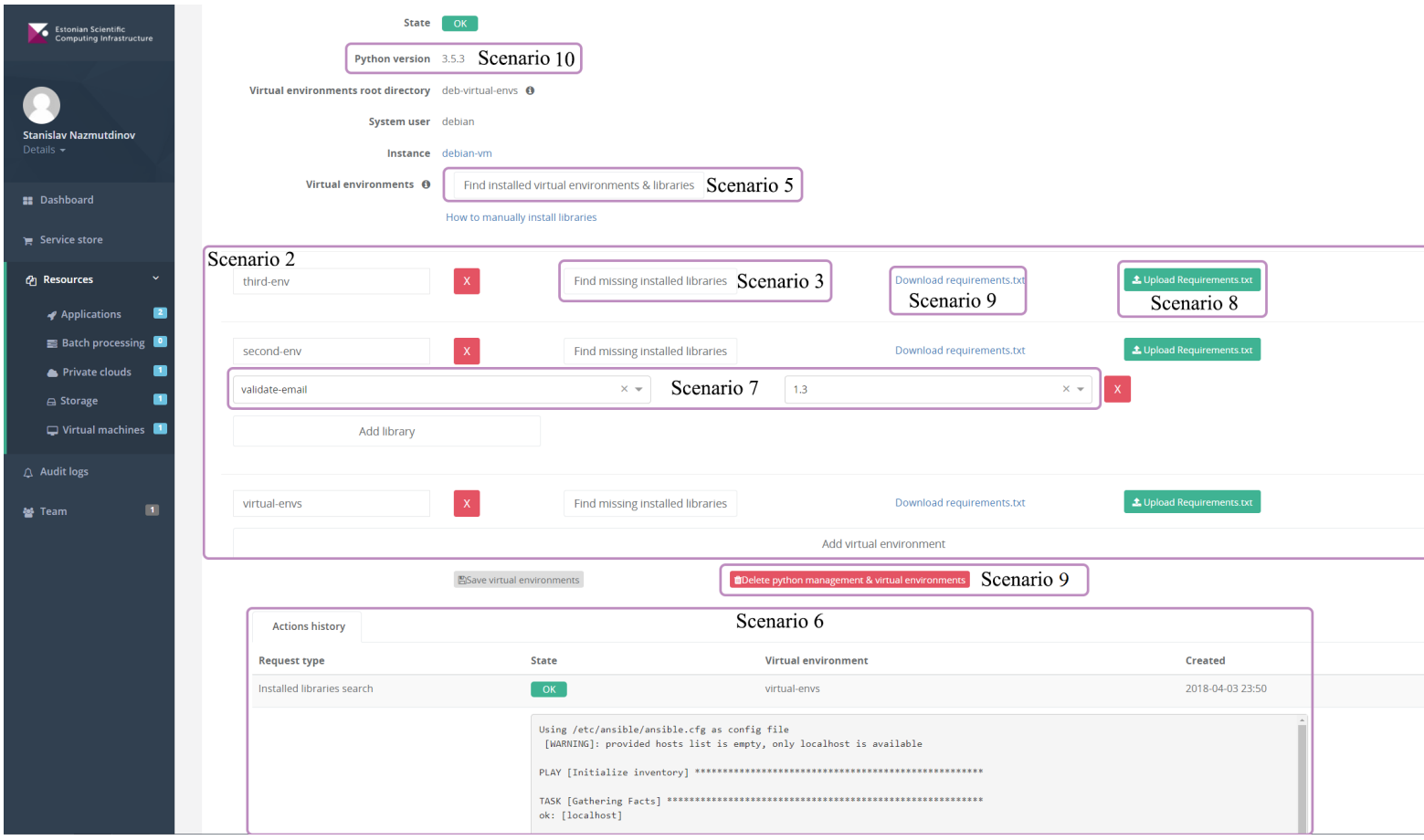

FIGURE 4.5: Python environment management details screen. Scenarios are described in Appendix A

4.4.5 Cloud application management process

The application provisioning process is composed of four major stages:

- **Provisioning request creation and task submission to Message Broker**: Celery task can hold any serialized information. In our case, we store there name of the request's Django model (which corresponds to the actual table in the database) and its primary key (steps 1.0-1.2 in Figure 4.6).
- **Request deserialization and acquiring coarse-grained pessimistic write lock**: Once the task is received by Waldur Celery worker (step 1.3), the request instance is fetched from the database (steps 1.4-1.5). Before proceeding any further, it is crucial to ensure that no other conflicting request is running (steps 1.6-1.8 in Figure 4.6). For that, we use coarse-grained pessimistic write locking mechanism[56]. The lock itself is represented as a key-value pair which is stored in Redis. They key is a string which represents the particular subset of functionality (for instance, related to a whole particular Python Management environment or

to just a single Python Virtual Environment) and value a boolean flag. The lock is stored in Redis only for specific time in order to ensure that if anything goes wrong and the lock is not discarded at the step 1.19, it will be eventually released.

- **Request processing**: As described in Section 4.4.2, this part of functionality executes external process, in our case Ansible (steps 1.10-1.12). Each output line is persisted in the database (step 1.15) and the structured information is retrieved from it (step 1.14). Once the process ends, additional callbacks are executed (steps 1.17-1.18).

- **Task status update**: Depending on the result of the previous stage (step 1.16: whether the process completed successfully or an exception was thrown), Celery worker updates request state in the database (step 1.18) and sends task status to the broker (step 1.20) which afterwards stores it in its queue. Naturally, the lock is also released (step 1.19).

The sequence diagram depicted in Figure 4.6 does not include all the involved Python classes, rather provides an overview of the main stages of cloud application provisioning process.

Same above-described strategy is used in both Python management and JupyterHub management services. Different types of requests (depicted in Figures 4.3 and 4.8) only redefine certain steps of the algorithm, for example, steps 1.6, 1.9, 1.14, 1.15, 1.17, 1.19. Such code reusability is achieved using Template method pattern [12].

4.4.6 Library name autocomplete functionality

In order to assist users with library selection in Scenario 2 (see Table A.2), we decided to introduce a library name autocomplete feature. There are several possibilities how to implement it.

- Make a Remote Procedure Call to PyPIXmlRpc service[13]. Even though this approach is the simplest, it has two drawbacks. Firstly, the query latency is unacceptable for this kind of use case and there is no way to decrease it, since the package search operation does not even provide a possibility to limit the search result size. Secondly, this PyPI XML-RPC interface is considered to be deprecated and not recommended to use[14].

[12] https://en.wikipedia.org/wiki/Template_method_pattern
[13] https://wiki.python.org/moin/PyPIXmlRpc?action=show&redirect=PyPiXmlRpc
[14] https://wiki.python.org/moin/PyPIXmlRpc

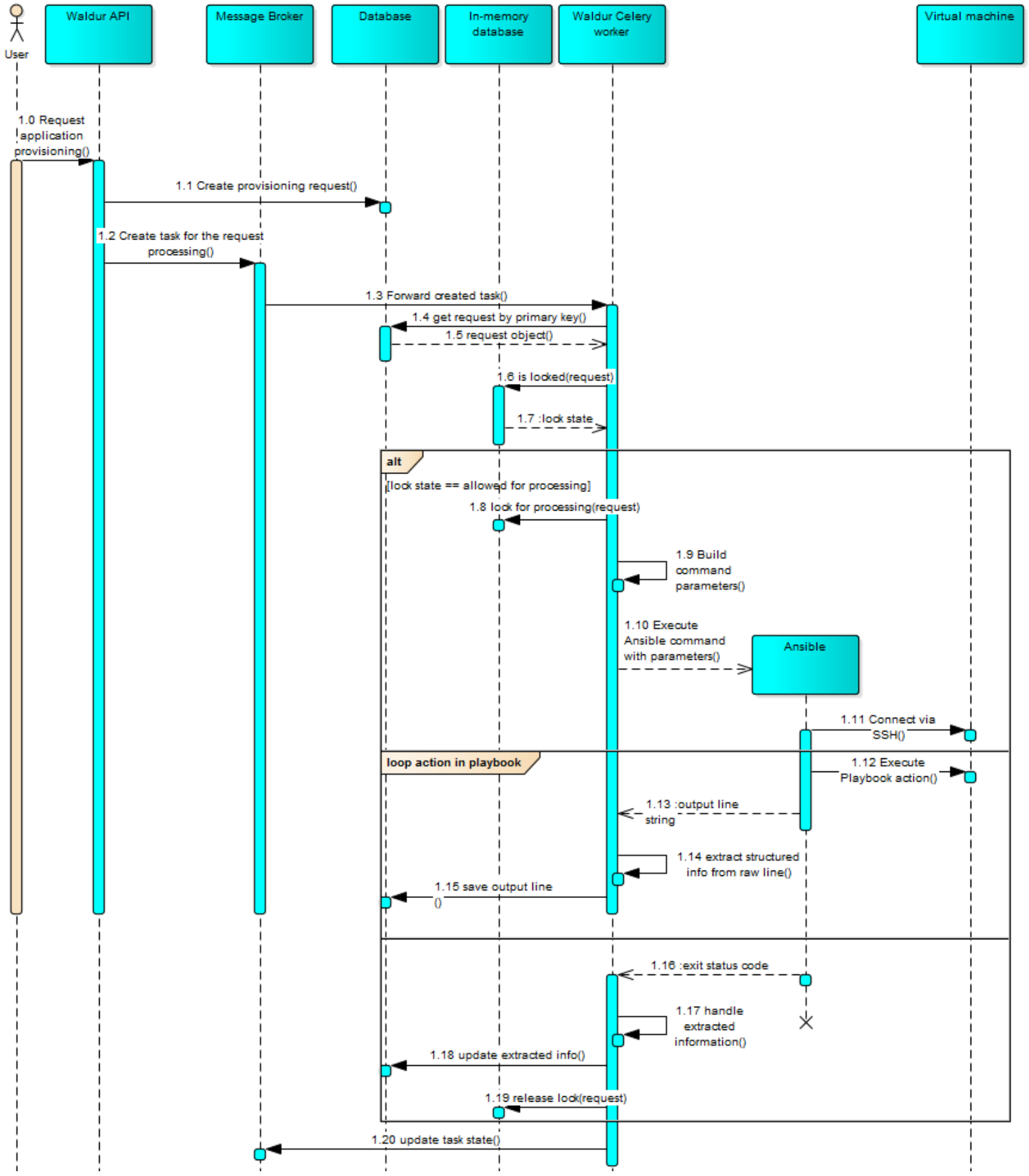

FIGURE 4.6: General application provisioning and management process

- Another approach is to regularly index PyPI package list and persist it either in the SQL database or in the in-memory database. Despite the fact that this approach requires more development effort and introduces pretty heavy batch task, user experience will improve substantially. Another design decision that should be made is where the indexed data should be held. Due to the fact that autocomplete feature requires rapid response from the server, initially the batch task was storing PyPI packages in Redis. The drawback of its approach is in its development and maintenance complexity, since Redis offers very simplistic API which does not support searching for data in SQL LIKE query fashion. In the end, we decided to store data in SQL database which would provide decent performance if indices are assigned to appropriate table columns.

High-level overview of the Python packages indexing batch task is depicted in the sequence diagram in Figure 4.7. As of March 20 2018, there are 132831 unique Python packages in Python Package Index repository. Due to this large amount of data, there were implemented following strategies in the batch logic:

- **Autocommit mode** of operation of a database connection. We decided not to use transactions for several reasons. Firstly, the batch task does not need any atomicity guarantees: if it is impossible to persist a package name in the database, we just log the failure without reverting previously inserted entries. Besides, there is no need to ensure isolation: we are perfectly fine with committing intermediate results to the database making them instantly visible to all database users.
- **Bulk inserts**: inserting and committing CachedRepositoryPythonLibrary (see Figure 4.3) entries one-by-one is clearly not the most optimal approach we can take. Thus we implemented an algorithm which accumulates configurable amount of package names in memory and afterwards performs bulk insert to the database until all packages are persisted.

4.5 JupyterHub provisioning service

In this section we describe how we implement JupyterHub server deployment and configuration management service. The new service is integrated with previously developed Python Management service to enable users to fully prepare their programming environment through Waldur self-service.

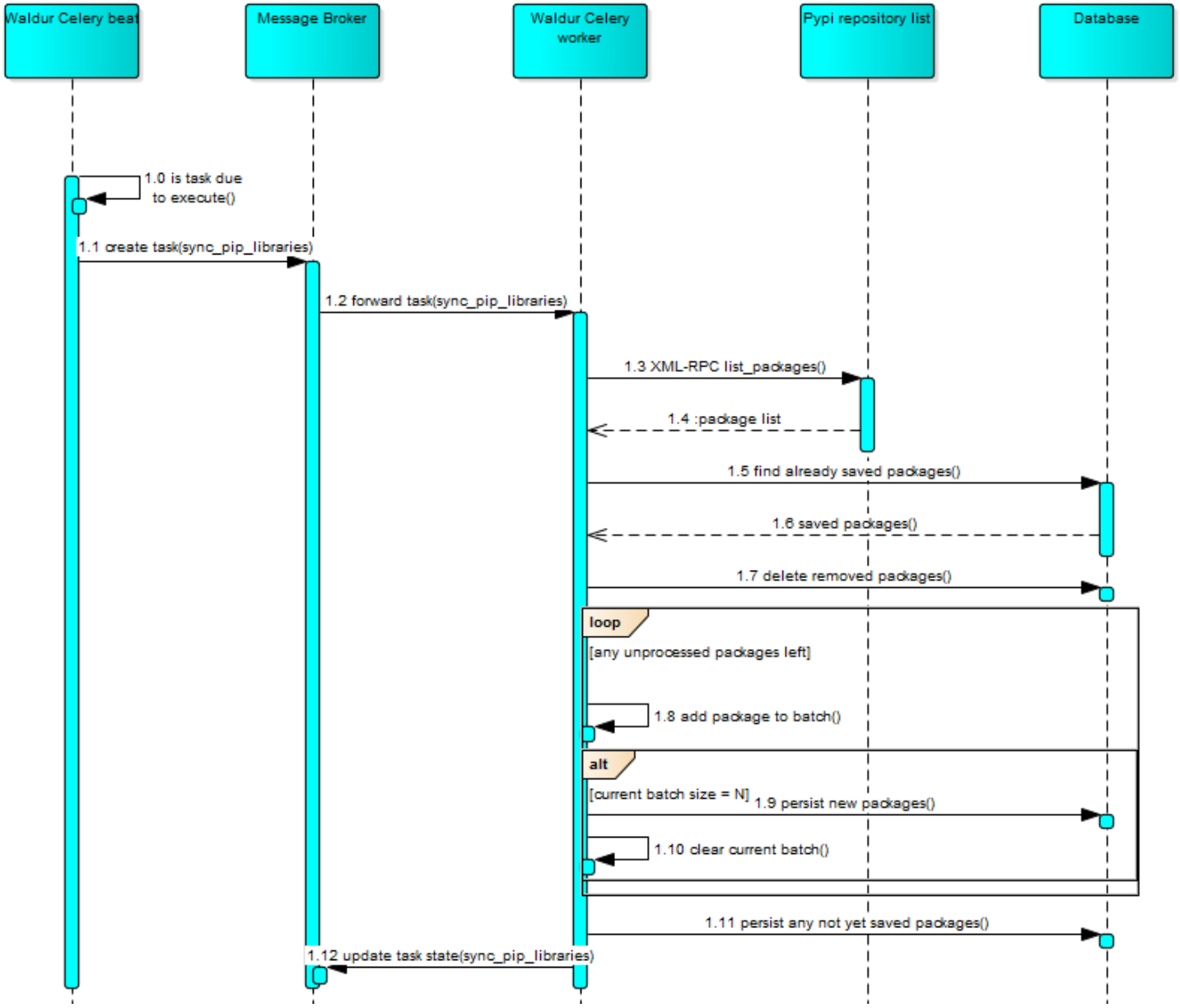

FIGURE 4.7: Python packages indexing batch task

4.5.1 Prerequisites

Same prerequisites are applied to virtual machine configuration as in case of Python management service which are described in Section 4.4.1. Additionally, an extra security rule should be assigned to a virtual machine which permits access to 80 and 443 ports. Furthermore, in order to use JupyterHub management service, first, there should be created Python management environment for the given virtual machine.

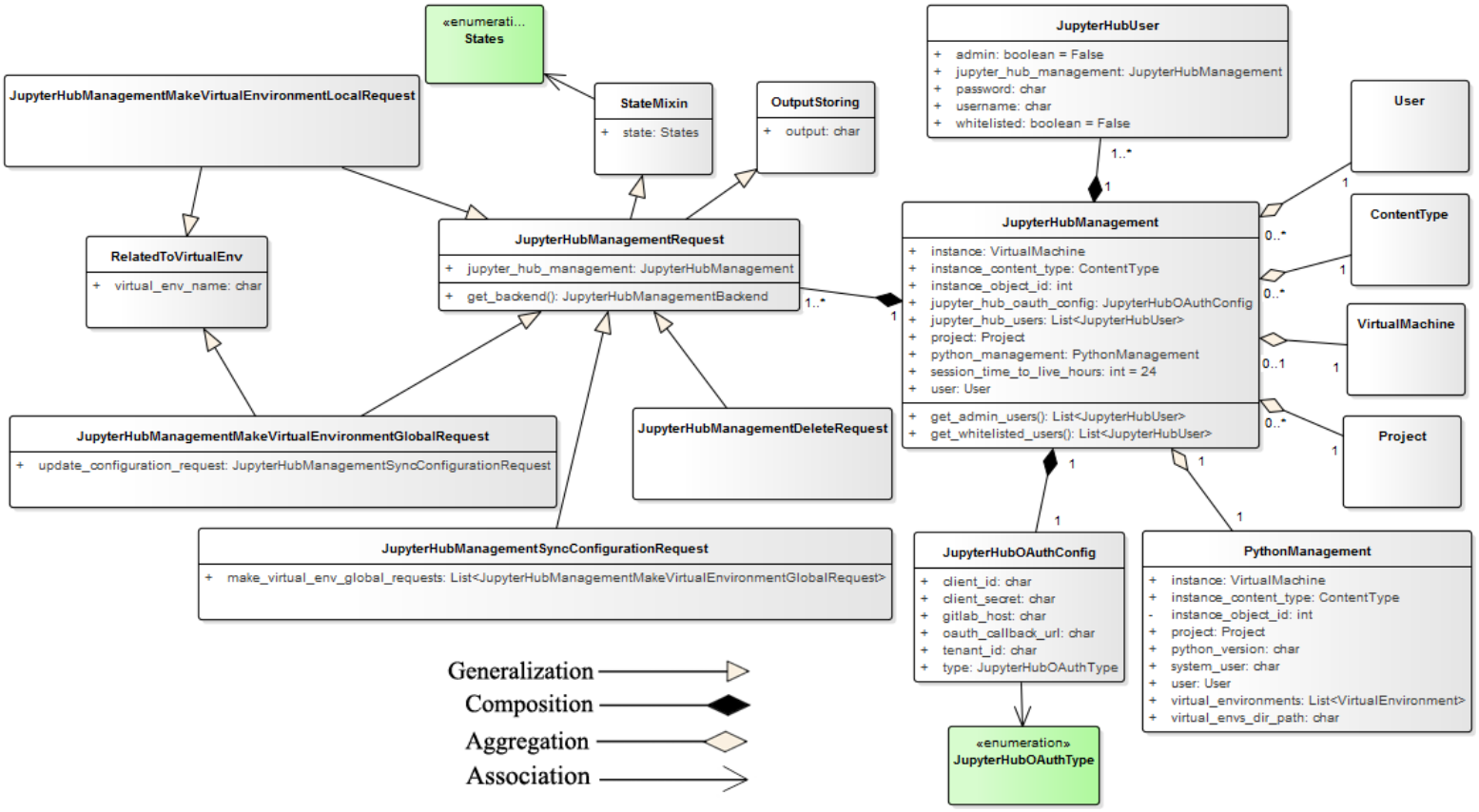

FIGURE 4.8: JupyterHub management service class diagram

4.5.2 JupyterHub service architecture

Figure 4.8 shows the class diagram of JupyterHub provisioning service. Similarly to Python Management service, each JupyterHub configurable option is stored in the database tables due to the reasons described in the Section 4.4.4. Needless to say, that this approach possesses exact same advantages and weaknesses.

We designed JupyterHub management requests in a same way, as Python management requests in Figure 4.3 to be able to reuse same code and preserve high degree of extendability, maintainability of the architecture.

As shown in Figure 4.9, according to a deployment strategy, JupyterHub is installed globally in the system. Waldur application logic ensures that only one JupyterHub is installed on a particular virtual machine. Configurable HTTP proxy is a proxy server which forwards incoming requests either to JupyterHub server or to a particular instance of Jupyter. JupyterHub maps users to local Linux system users. JupyterHub users may be either manually defined through Waldur self-service or come from OAuth2 service provider. When a new user logs in, JupyterHub creates a new system user, then spawns

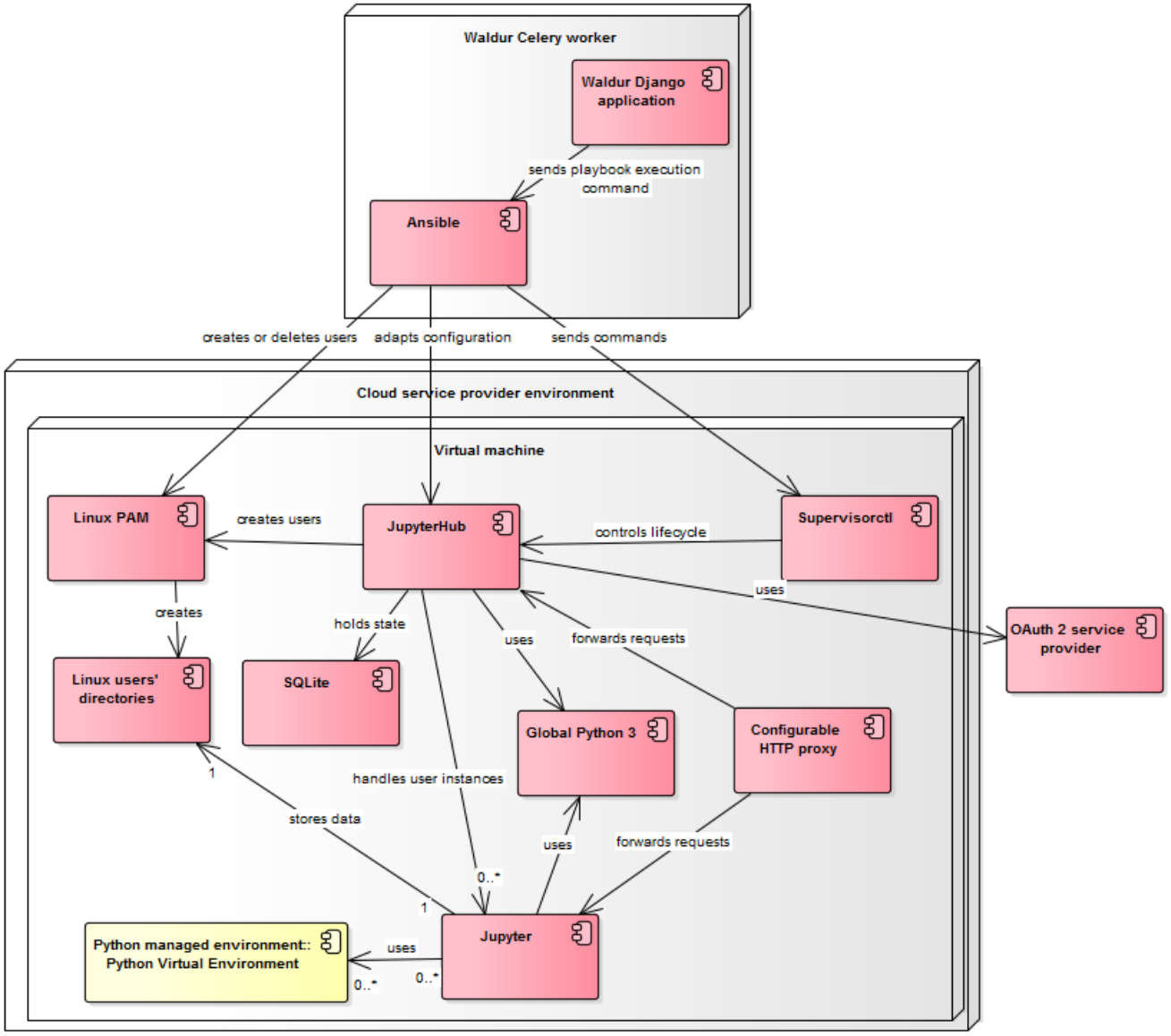

FIGURE 4.9: JupyterHub environment management after provisioning on the virtual machine

a new Jupyter process using systemd[15] and the working directory of the newly created Jupyter instance is set to the home directory of the corresponding Linux system user. JupyterHub process runs with root permissions (systemd instance spawner requires that) and separate Jupyter instances work with the permissions of the corresponding logged in user.

Furthermore, Jupyter instances have access to managed by Waldur Python virtual environments that can be marked as globally accessible to all JupyterHub users in Waldur self-service portal.

[15] https://github.com/jupyterhub/systemdspawner

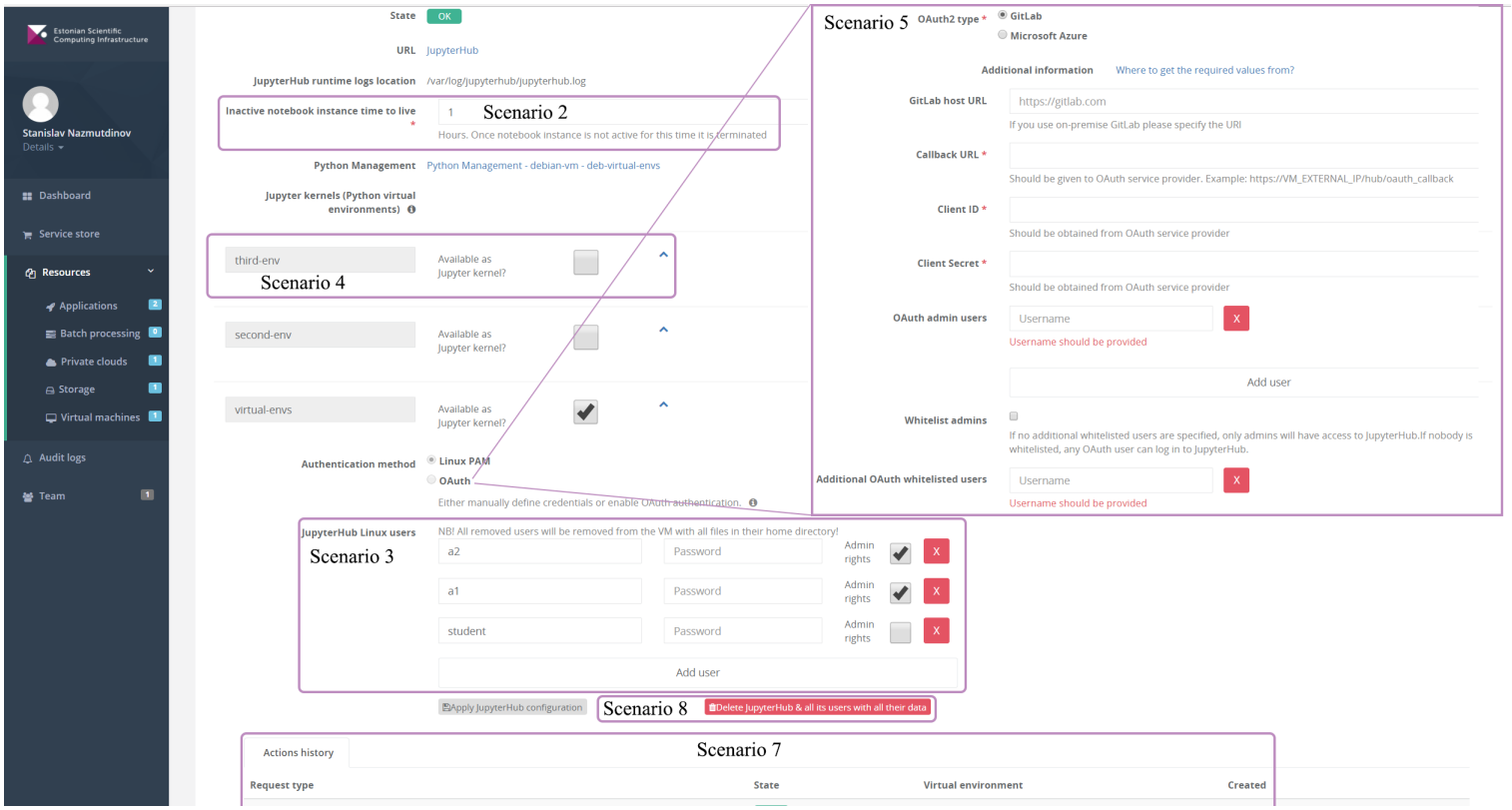

FIGURE 4.10: JupyterHub management details screen. Scenarios are described in Appendix B

The screenshot in Figure 4.10 shows a JupyterHub management service details screen with manual user definition mode. Additionally, we included a screenshot of OAuth2 form section. JupyterHub scenarios described in Appendix B are mapped to corresponding graphical user interface elements. Similarly to Figure 4.5, purely technical scenarios are not shown.

4.6 Quality control of the built services

The quality control in Waldur covers undoubtedly a wide range of aspects of the system. However, we found ourselves in a need to introduce additional type of tests for comprehensive testing of the deployment logic described in Section 4.4.5. Even though we had written component tests to test application logic and its interactions with the database, there were no tests which ensured that Ansible provisioning logic works as expected. Functionality of this type of services spans across multiple areas of responsibility (application business logic - Ansible Playbooks - target operating system), thus we argue that it is not enough to just write unit or component tests to test only application logic.

What is more, Ansible provisioning logic is tightly coupled with the application logic (application logic passes arguments to the Ansible playbooks and parses their output), so we decided not to separate them during testing. This kind of tests can be classified as integration tests and they reside on a next level after component tests in the test pyramid shown in Figure 4.11.

There is definitely a need for integration testing due to the following reasons: firstly, Ansible does not provide any means of testing Playbook logic. On top of that, it is very hard to guarantee that the provisioning logic is applicable for various Linux distributions. The reason is that various distributions come with different setups, thus some may require extra steps. For instance, Ubuntu 16.04, as opposed to Debian 9, does not come with Python 2 which is a required dependency of Ansible. What is more, the outcome of the deployment process depends not only on the particular Linux distribution but also on the tools which are used during installation and on the packages being installed. If one of these tools or packages happen to introduce breaking changes, there is a great chance that it will be discovered by the users on production and not during continuous integration process.

Needless to say, that manual testing takes a lot of effort and should be automated as much as possible. It would be very beneficial for further development and maintenance of these services to have tests for each supported family of Linux distributions. Due to the fact that integration tests cover quite a large scope of the system (application logic - Ansible playbook logic - environment of the Linux distribution), they may require substantial maintenance effort. Thus, amount of integration tests should be kept low. In our own integration tests we cover only happy paths [57].

In order to simulate a provisioning process, integration tests need an empty Linux system. For that we use Docker containerization system in order to ensure that the integration tests follow F.I.R.S.T principles [58]. Most importantly, the integration tests should be fast, isolated and repeatable. The idea of using Docker in tests was inspired by the Java library called test-containers[16]. However, we should mention that our Docker images do not perfectly reflect actual virtual machines. Even though we use same operating system images which are deployed by OpenStack cloud system of the Estonian Scientific Computing Infrastructure, due to the nature of Docker, we need to manually write container initialization logic in Dockerfiles (to run systemd, SSH server processes and to create appropriate system users).

The main entities involved into the integration tests and their interactions are depicted in Figure 4.12 can be split into following major stages:

[16] https://github.com/testcontainers/testcontainers-java

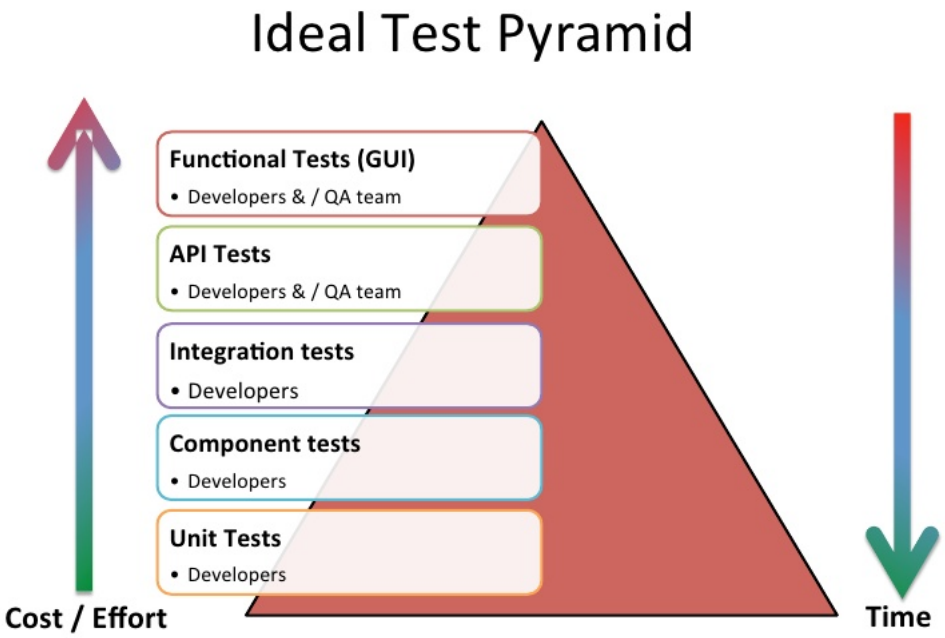

FIGURE 4.11: Test pyramid (Source: [3])

- **Docker image creation**: Each integration test suite prepares required Docker images by calling static method in Ubuntu1604Container class during setUpClass() step. It is called once per test suite (steps 1.1-1.9 in Figure 4.12). On top of that, to ensure that tests are fast, we make the process of Docker image creation lazy: if image already exists, steps 1.6-1.8 are skipped.

- **Execution of a test case**: Once the required images are created in the system, test runner executes test cases. During execution of a test case an object of Ubuntu1604Container class is created and at some point it is wrapped in "with" block (which is a statement in Python syntax). Before code enters "with" block, the wrapped object runs Docker container (step 1.12). Naturally, the logic under the test should be executed within this "with" block. The logic under the test spawns Ansible subprocess which connects to the Docker container and performs all modifications (steps 1.14-1.15). All test assertions should also be performed within the scope of "with" statement (if access to the virtual machine is needed). Once code exits "with" block, instance of Ubuntu1604Container terminates the container (steps 1.16-1.17).

For demonstration purposes we omit any additional non-related test logic executed throughout 1.0- 1.17 steps.

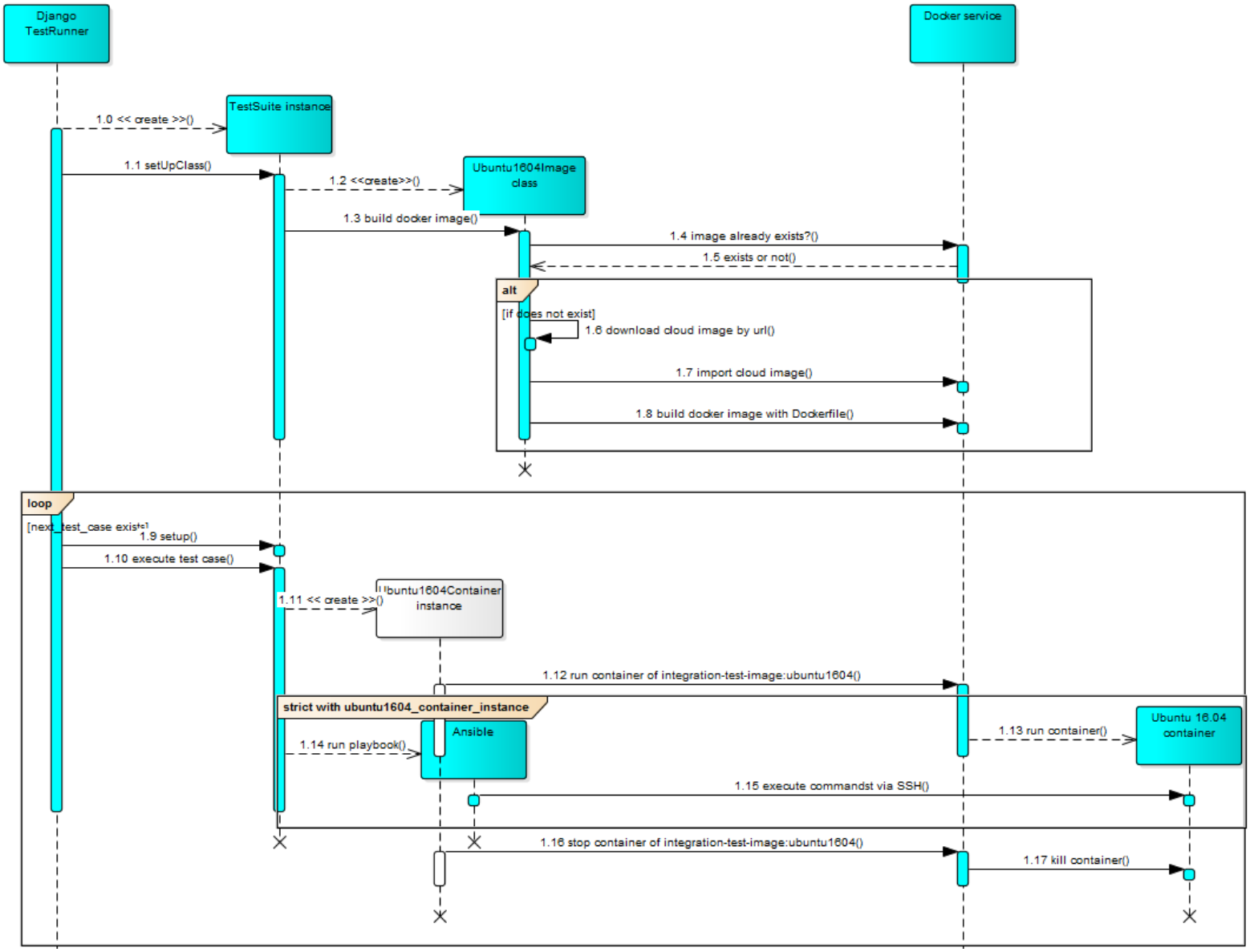

FIGURE 4.12: Integration tests execution process

Unfortunately, execution of integration test greatly depends on the host platform. For example, we discovered that Ubuntu 16.04 Docker image does not work in a stable way on CentOS 7 host. Because of the fact that Waldur team runs Jenkins server on CentOS 7, this limitation blocked us from incorporating integration tests into Continuous Integration pipeline of Waldur.

4.7 Evaluation

4.7.1 Feedback from the potential users

According to the design science research methodology described in Section 1.2.3 the proposed solution should be evaluated. In the scope of the thesis we perform evaluation with potential users from academia. The evaluation process is organized as follows: first, we organize one-on-one meeting with each potential user. If a user has not ever used the cloud of the Estonian Scientific Computing Infrastructure before, we show them around the self-service system. Afterwards, we give a brief informal introduction of what the new services are and what kind of problems they intend to solve. During the testing process we take the user through all the Scenarios descried in Appendices A and B. Once the testing stage is over, the user was asked a few follow-up questions about their general impression and what purposes they would use these services for. After that, the user was asked to fill a feedback form. The results and analysis of the received feedback are presented in this section.

In total, we conducted testing sessions with 5 university researchers and received feedback through survey form from 4 people. The feedback form consists of 18 questions. Even though the amount of the responses is relatively small, we managed to reach exactly the potential users who are interested in the services we built.

In the first question we asked if we made the right decision by providing the service which allows to configure a development environment through web-based user interface as an alternative to command-line execution. The following Table 4.1 shows that in general Python management service was warmly accepted by the university researchers.

The process of setting up a python development environment through self-service is more convenient than via SSH terminal.		
Strongly agree	1	25%
Agree	3	75%
Neutral	0	0%
Disagree	0	0%
Strong disagree	0	0%

TABLE 4.1: Survey responses to question 1: The process of setting up a python development environment through self-service is more convenient than via SSH terminal

The second question (Table 4.2) is meant to find out if the researchers believe that the process of installation of libraries through self-service allows them to quickly setup development environment on a virtual machine. The reason for this question was that

we were not sure if our choice to use pip as a package manager (described in Python management scenario 2 in Table A.2) was viable. The received responses show that in general, respondents do not mind specifying libraries through self-service. However, during testing session one researcher indicated that it would be nice to have already predefined set of essential data science libraries for installation.

Self-service enables me to rapidly set up a python development environment on a remote virtual machine.		
Strongly agree	2	50%
Agree	2	50%
Neutral	0	0%
Disagree	0	0%
Strong disagree	0	0%

TABLE 4.2: Survey responses to question 2: Self-service enables me to rapidly set up a python development environment on a remote virtual machine

When it comes to JupyterHub provisioning service, we wanted to know if the service allows to configure sufficient amount of aspects of JupyterHub deployment (Table 4.3). Several researchers took neutral stand in this question which indicates that further work should be carried out to identify possible configuration options to allow fine-grained customization of JupyterHub deployment.

JupyterHub management service provides sufficient amount of configuration options.		
Strongly agree	0	0%
Agree	2	50%
Neutral	2	50%
Disagree	0	0%
Strong disagree	0	0%

TABLE 4.3: Survey responses to question 3: JupyterHub management service provides sufficient amount of configuration options

The next two questions (Tables 4.4 and 4.5) are related to improving quality of courses that the researchers teach. Researchers believe that the new services may be helpful to their teaching work.

Python management service will help me to improve courses that I teach.		
Strongly agree	0	0%
Agree	3	75%
Neutral	1	25%
Disagree	0	0%
Strong disagree	0	0%

TABLE 4.4: Survey responses to question 4: Python management service will help me to improve courses that I teach

JupyterHub management service will help me to improve courses that I teach.		
Strongly agree	0	0%
Agree	3	75%
Neutral	1	25%
Disagree	0	0%
Strong disagree	0	0%

TABLE 4.5: Survey responses to question 5: JupyterHub management service will help me to improve courses that I teach

In addition to teaching, our services are also aimed to support performing data analytics in the cloud. In order to identify how well these two services actually fulfill their purpose, we asked for opinion of the researchers in the following two questions (Tables 4.6 and 4.7). The results imply that our proposed services may be of benefit to conducting data analysis in cloud environment.

Python management service will help me to solve data analysis tasks in cloud environment more efficiently.		
Strongly agree	0	0%
Agree	3	75%
Neutral	1	25%
Disagree	0	0%
Strong disagree	0	0%

TABLE 4.6: Survey responses to question 6: Python management service will help me to solve data analysis tasks in cloud environment more efficiently

JupyterHub management service will help me to solve data analysis tasks in cloud environment more efficiently.		
Strongly agree	0	0%
Agree	3	75%
Neutral	1	25%
Disagree	0	0%
Strong disagree	0	0%

TABLE 4.7: Survey responses to question 7: JupyterHub management service will help me to solve data analysis tasks in cloud environment more efficiently

The next question (Table 4.8) is an open-ended one. Due to the broad range of available JupyterHub extensions we wanted to know if researchers could suggest anything specific. These suggestions we consider as possible directions of the future work. One researcher specified that it would be nice to have an option to install R kernels through self-serivce.

What additional Jupyter extensions would you like to be part of the deployment?		
R kernel	1	25%
F# kernel	1	25%
No suggestions	2	50%

TABLE 4.8: Survey responses to question 8: What additional Jupyter extensions would you like to be part of the deployment?

So as to assess how stable the implemented services work, we asked users if they had encountered any errors while they had been using the services (Table 4.9). At the very first evaluation we encountered serious issue in provisioning logic which was manually resolved on the particular virtual machine and the provisioning process managed to complete. During another evaluation session we stumbled upon a minor bug in the user interface.

Did you encounter any errors while managing Python virtual environments or deploying JupyterHub?		
Yes	2	50%
No	2	50%

TABLE 4.9: Survey responses to question 9: Did you encounter any errors while managing Python virtual environments or deploying JupyterHub?

The next question (Table 4.10) was intended so that users could describe in more detail what kind of bugs they encountered while using the services. The first answer here is

pretty general due to the fact that the bug in the graphical user interface has already been known by the maintainer of the services. The second answer is related to bug during provisioning process.

What kind of errors did you encounter?		
Different bugs	1	50%
The errors were in an early stage and related to Ubuntu specific problems with pip.	1	50%

TABLE 4.10: Survey responses to question 10: What kind of errors did you encounter?

Provided log output of management requests helped me to resolve the issues.		
Yes	1	50%
No	0	0%
The issues did not occur during processing of management requests	1	50%

TABLE 4.11: Survey responses to question 11: Provided log output of management requests helped me to resolve the issues

After resolving the issues, Python management service worked in a stable way.		
Strongly agree	1	50%
Agree	1	0%
Neutral	0	0%
Disagree	0	0%
Strong disagree	0	0%
Issues were not solved	0	0%
There were no further interactions with the service	0	0%
Not applicable	1	50%

TABLE 4.12: Survey responses to question 12: After resolving the issues, Python management service worked in a stable way

After resolving the issues, JupyterHub management worked in a stable way.		
Strongly agree	1	50%
Agree	1	0%
Neutral	0	0%
Disagree	0	0%
Strong disagree	0	0%
Issues were not solved	0	0%
There were no further interactions with the service	0	0%
Not applicable	1	50%

TABLE 4.13: Survey responses to question 13: After resolving the issues, JupyterHub management worked in a stable way

When we asked potential users to say if they wish to use the built services in their future work or not, it turned out that respondents were in general interested in these services (as shown in Tables 4.14 and 4.15). The received responses imply that Python management service did not receive as much praises as did JupyterHub management.

I will use Python management service in my future work.		
Strongly agree	2	50%
Agree	2	50%
Neutral	0	0%
Disagree	0	0%
Strong disagree	0	0%

TABLE 4.14: Survey responses to question 14: I will use Python management service in my future work

I will use JupyterHub management service in my future work.		
Strongly agree	3	75%
Agree	1	25%
Neutral	0	0%
Disagree	0	0%
Strong disagree	0	0%

TABLE 4.15: Survey responses to question 15: I will use JupyterHub management service in my future work

The last two questions (Tables 4.16 and 4.17) are aimed to reveal how the implemented services generally appeal to the participants. The participants were asked to assess the services on a scale from one to five. Overall, the feedback was positive.

What is your overall satisfaction with Python management service?		
5	0	0%
4	4	100%
3	0	0%
2	0	0%
1	0	0%

TABLE 4.16: Survey responses to question 16: What is your overall satisfaction with Python management service?

What is your overall satisfaction with JupyterHub management service?		
5	1	25%
4	3	75%
3	0	0%
2	0	0%
1	0	0%

TABLE 4.17: Survey responses to question 17: What is your overall satisfaction with JupyterHub management service?

In the last questionnaire entry the respondents were asked to provide their feedback in free form. One researcher requested a more restrictive permissions on home directories of JupyterHub users in order to ensure that work of JupyterHub users remains hidden from other users (useful in case of teaching). What is more, the fact that user should first create Python management environment before JupyterHub deployment proved to be somewhat inconvenient for those, who do not intend to program in Python. In future this relation between Python and JupyterHub should be made optional.

Another researcher stressed the fact that degree of automation should be further increased, that is, process of fulfilling the prerequisites described in Sections 4.4.1 and 4.5.1 should be performed by the system. Another suggestion was that it would be nice if users could request provisioning of the managed tools on a new virtual machine directly via Python or JupyterHub services without having to preliminary create a virtual machine themselves.

There was also a comment regarding Python management service. A user said that the service is in general complex. He suggested that this complexity could be solved by

using Anaconda distribution instead of pip or by providing a set of libraries which are typically used for data analysis.

4.7.2 Comparison with services from other cloud providers

In order to compare our solution with other services that aim to achieve similar goals (support interactive Python computing workloads in the cloud using Jupyter notebooks), we compose following Table 4.18. First, we compare our service to Microsoft Azure Notebooks. Another solution that we selected for comparison is Amazon EMR which also provides a CLI tool for JupyterHub deployment. We are aware that Microsoft Azure Notebooks and Google Colaboratory are PaaS systems, but still our Python and JupyterHub services aim to achieve similar goals, hence we believe that it is valid to compare our services to PaaS solutions.

In summary, the Table 4.18 identifies the main weaknesses of the built services: firstly, the existing JupyterHub service should be complemented by other widely used kernels such as R, Julia. Above that, JupyterHub should come with additional extensions.

We want to elaborate a little bit more on single user environment isolation aspect of our service. We estimate it to be medium due to the following reasons: (i) Python management service does not install libraries globally, but into isolated Python virtual environments. (ii) Jupyter users do not have permissions to modify virtual environments other than their own which they can create manually through terminal if needed (for example, if they do not have access to Waldur self-service where corresponding Python and JupyterHub service instances are managed). Even though we could achieve higher level of isolation using JupyterHub's Docker spawner[17] or Kubernetes spawner[18], we decided not to introduce additional layer of abstraction in a form of containers, since we want to keep infrastructure internals accessible to the users so that they could resolve issues themselves if any occur. Docker containers add additional complexity which we would like to avoid, since our target users are researchers and lecturers who do not necessarily have advanced system administration skills.

Attributes	**Python and JupyterHub management**	**Microsoft Azure Notebooks**[19]	**Google Colaboratory**[20]	**JupyterHub CLI on Amazon EMR**[59]
Type of service	Application management service in hybrid cloud	PaaS	PaaS	Managed cluster platform

[17] https://github.com/jupyterhub/dockerspawner
[18] https://github.com/jupyterhub/kubespawner#kubespawner-jupyterhub-kubernetes-spawner

Attributes	**Python and JupyterHub management**	**Microsoft Azure Notebooks**[19]	**Google Colaboratory**[20]	**JupyterHub CLI on Amazon EMR**[59]
Supported languages	Python, additional kernels should be installed manually	Python, R, F#	Python	Python, R, Scala, Ruby, Julia
Libraries installation to Jupyter Python kernels	Using self-service, manually or through notebooks	Comes with Anaconda, other libraries are installed using notebooks or custom ssh script, all installed libraries are lost once a Jupyter instance shutdowns	Additional libraries are installed through notebooks. Comes with predefined list of data science packages	Libraries can be installed with a CLI tool, manually or through notebooks. Comes with predefined list of data science libraries
Additional kernels installation	Manual installation	Not possible	Not possible	Manual installation
Distributed processing	No, required Jupyter kernels should be manually installed	No	No	Yes, platform provides cluster management capabilities and appropriate kernels
GPU support	GPU drivers should be installed separately or come with a Linux image	No	Yes, but a single GPU is shared by multiple users	GPU instances should be used [60]
Jupyter extensions	Additional extensions should be installed manually	Comes with a predefined list of extensions	Environment is built from scratch, no Jupyter extensions are supported	Comes with a predefined list of extensions, additional extensions can be manually installed [59]
OAuth2 support	GitLab, Microsoft Azure are configurable through self-service	Authentication using Microsoft Account	Authentication using Google Account	Should be manually configured

Attributes	**Python and JupyterHub management**	**Microsoft Azure Notebooks**[19]	**Google Colaboratory**[20]	**JupyterHub CLI on Amazon EMR**[59]
Deployment and configuration management	Using self-service or manually adapting configuration files	Deployment is entirely handled by the service	Deployment is handled entirely by the service	Initial deployment via .sh script. Afterwards only manual configuration management
Notebooks sharing capabilities	Manually using VCS	Sharing capabilities come out of the box, integration with GitHub	Integration with Google Drive and GitHub	Manually using VCS
Simultaneous collaborative notebook editing	No	No	Yes	No
Single user environment isolation degree	Medium	High	High	Medium
Access to underlying infrastructure	Yes	Access only to a file system through Jupyter	Access only to a file system through Jupyter	Yes
Resource usage limits	Restricted by the particular virtual machine resources	4 GB memory, 1 GB storage per user	1 CPU core, 13 GB memory[21]	Restricted by the cluster and the particular virtual machine where JupyterHub operates

TABLE 4.18: Jupyter as a Service systems comparison

4.8 Discussion

In general, the gathered feedback from the users is positive and some of them expressed a wish to use the built services in their work. We discovered that JupyterHub management was anticipated a bit more than Python management. The probable explanation is the

[21] https://colab.research.google.com/notebook#fileId=1_x67fw9y5aBW72a8aGePFLlkPvKLpnBl

fact that there are alternative ways of installing libraries: it is possible to install Python libraries through SSH or Jupyter notebooks. Another reason is that Python management service is complicated to use. The comments received from the researchers indicate that our requirements engineering process could have been improved if we had conducted preliminary interviews with the target user group. Moreover, the best outcomes could have been achieved if the potential users had been involved into the project to ensure that all the developed functionality is appropriate and meets the expectations of the users.

Comparison with other similar systems shows that the selected model of service (application provisioning service) automates tedious deployment process of analytics software and at the same time provides high degree of flexibility to the users who would like to additionally customize the deployed software packages. In addition, due to the fact that the services are built in the hybrid cloud system, users are not bound to any specific cloud provider. Another benefit that Python management brings is that it enforces users to follow the best practices that they may not be aware of: for example, to use isolated Python virtual environments instead of installing libraries globally.

Nonetheless, there is still room for improvement. For instance, JupyterHub service should offer more kernels and extensions. Furthermore, in order to assist data analysts, Python management should provide a set of commonly used libraries for data analytics. Moreover, the overall degree of automation ought to be further increased: currently users should create virtual machines separately. The process of creation of a virtual machine should be incorporated into Python and JupyterHub management services.

4.9 Conclusion

The process of building services for automatic management of data analytics tools on a remote virtual machine was split into two phases.

First, we came up with usage scenarios. We placed particular focus on modifiability of the provisioned tools to ensure that users would not be limited to only the supported configuration options. Otherwise, the scenarios were composed based on our own usage experience of the tools. We made effort to compose scenarios in such a way, which would all together form a convenient and intuitive workflow for configuration management of the given tools on a remote virtual machine.

Afterwards we presented a design and implementation of two services for configuration management of Python development environment and JupyterHub deployment on a remote machine through cloud self-service. The new functionality made cloud services

more accessible to users with different backgrounds. Our main goal was to hide application deployment and development environment configuration complexity from the users as much as possible. We can state, that we managed to allow users to perform all the essential infrastructure configuration management through Waldur self-service for Python-based single-node general-purpose and deep learning workloads. It is possible to install libraries into isolated Python virtual environments via self-service. JupyterHub management service, in turn, allows users to deploy multitenant Jupyter notebook execution environment and use previously configured Python development environments in it.

We can conclude that the services which deal with provisioning and application configuration management are brittle since the deployment relies on a lot of external dependencies. The services ought to be maintained carefully and should be regularly checked if they still function on the given Linux distribution. For this purpose, we developed integration tests which should be configured to run as part of Continuous Integration pipeline.

The gathered feedback from the potential users indicated that the services have the potential to be adopted by the university lecturers and researchers: we managed to select the relevant to their everyday work tools for provisioning in the cloud environment and provide sufficient degree of deployment automation. At the same time, we identified shortcomings of our services some of which were taken into account and corrected.

Chapter 5

Conclusion and Future Work

In Chapter 5 we summarize all the findings of the thesis. In Section 5.1 we provide a brief recap what was achieved throughout the thesis. The answers to the main research questions are provided in Section 5.2. Furthermore, we describe limitations of the built services in Section 5.3 and outline directions for the future work in Section 5.4.

5.1 Conclusion

In the first part of the thesis we gave an overview of processing frameworks which can be used either for ETL, data mining, machine or deep learning workloads. Afterwards, we conducted a survey to discover what data analytics engines are used by Estonian researchers from academia and industry practitioners. Acquired results helped us to make an informed decision regarding what data analytics tools service Waldur should first provide.

Subsequently, we constructed a design for dynamic configuration management of two platforms. The first service is for management of Python development environments and the second service is for managing JupyterHub deployment. Both front end and back end of the services were implemented in the existing Waldur codebase and then released to Waldur deployment at the Estonian Scientific Computing Infrastructure. Furthermore, appropriate integration tests were developed for testing back end logic of the built services with the help of Docker containers that ensure fast response, isolation and repeatability of the tests. Once the design had been instantiated and released to the production cloud environment of the Estonian Scientific Computing Infrastructure, we introduced new services to university researchers and asked for their feedback. It turned out that in general feedback was positive. The services have the potential to be

adopted by the university lecturers and researchers. Moreover, the gathered feedback allowed us to identify possible future development directions.

5.2 Answering the Research Questions

The main research question was formed as follows: **How to dynamically manage deployment and life cycle of data analytics platforms in the context of hybrid cloud brokerage platform?** The main research question was split into two subquestions. In the following sections we provide the answers to each of them.

5.2.1 RQ-1: What are state-of-the-art data analytics engines used in the industry and in Estonia?

In the scope of the thesis we distinguished 3 major groups of data analytics frameworks: (i) single-node general-purpose tools, (ii) distributed general-purpose tools and (iii) deep learning tools. Distributed general-purpose engines, in turn, can be further classified into two categories: batch and stream processing. We identified the major tools for each field.

The most notable single-node data analytics tools are available in a form of Python and R packages. As for distributed batch processing, MapReduce-based tools have been dominating the area for long time. However, MapReduce does not use memory in an efficient way, so there is a growing trend towards utilization of in-memory computations for large scale processing. Most notable examples of such engines include Apache Spark and H2O. There are several options available for stream processing such as Apache Flink, Apache Spark Streaming and Apache Heron. As far as deep learning is concerned, there is a great variety of engines available each providing its own notable features: imperative or declarative programming approaches, high degree of flexibility during the training process, support for distributed processing, GPU or mobile support, etc.

The survey findings showed us that majority of Estonian researchers use single-node Python and R-based tools to perform data analysis tasks. Besides that, there is a great interest towards deep learning engines. With that knowledge we concluded that it makes sense to support first researchers that work with Python-based tools which include single-node general-purpose libraries as well as deep learning engines that provide Python API.

5.2.2 RQ-2: How to automate provisioning and management of data analytics tools through hybrid cloud brokerage platform?

Automated application provisioning service logic relies on a multitude of tools each of which is used for a particular task. All in all, the logic spans across several environments. Due to the long lasting nature of software installation process, the logic is executed asynchronously with help of message-oriented architecture. In order to improve performance when accessing the services, all the information related to the provisioned tools is held in the database. The first steps of provisioning logic are performed in an application tier: a worker receives a task from a message broker, obtains all the necessary information from the database and locks the environment for processing using coarse-grained pessimistic write lock. Next, the application tier spawns a new Ansible subprocess and delegates infrastructure configuration to it. Ansible logic connects to a virtual machine via SSH and brings the environment into a desired state. During this process the state in the database of the cloud broker is updated by parsing logs produced by Ansible.

5.3 Limitations

There are several limitations that apply to our services. Firstly, users should manually fulfill prerequisites described in Section 4.4.1 so that the services could function.

Another common limitation of these two services is that they may suffer from state inconsistency issues if a user manually installs Python libraries and does not trigger appropriate search for libraries through self-service or if JupyterHub configuration is manually changed through SSH terminal. This limitation could have been eliminated had we decided to build PaaS-type services, but we decided to provide more flexibility and expose underlying infrastructure to the users.

Concerning scalability limitations of JupyterHub deployment, the current solution supports JupyterHub deployment on a single node. We considered it to be a valid solution given the fact that OpenStack private cloud of the Estonian Scientific Computing Infrastructure can provision virtual machines with up to 16 CPU cores and 32 gigabytes of RAM. However, if it is expected that there will be hundreds of simultaneous JupyterHub users then scalability might become an issue.

Both Python and JupyterHub management services have a common constraint: they were tested only on Ubuntu 16.04 and Debian 9 operating systems, thus compatibility with other Linux distributions is not guaranteed. Neither of the services support Microsoft Windows operating systems.

As for integration tests, even though we managed to implemented them to ensure that they are fast, isolated and repeatable, they may not entirely reflect actual virtual machines provided by the cloud providers, due to the fact that developers should write initialization logic in Dockerfiles themselves. Moreover, we discovered that execution of integration tests depends on the host platform, for instance, when we tried to run integration tests using Ubuntu 16.04 image on CentOS 7 host, the container froze unexpectedly in the course of the test. Our best guess is that it happens due to the fact that Ubuntu 16.04 distribution requires more modern version of Linux kernel than CentOS 7 can provide. This prevented us from including integration tests into Waldur continuous integration pipeline.

5.4 Future Work

In this section we discuss possible directions for the future work.

Firstly, actions should be taken to automate fulfillment of prerequisites described in Section 4.4.1: it should be possible to specify that a new virtual machines should come with public keys of Waldur Celery workers. In addition, an option of automatic creation of a new virtual machine for the new deployment should be included into workflow of Python and JupyterHub management services.

A further potential direction of development is to introduce cluster provisioning and management service to support distributed processing use cases. From the survey results described in Chapter 3 we can conclude that it makes sense to build a service around Apache Spark. We believe that we have laid the groundwork for future distributed processing services. Python management service currently allows configuring development environment on a single virtual machine and it can be extended to work with multiple nodes and keep environments on all nodes in a cluster in sync. Such expansion of Python management service may make it much more useful in the eyes of users in comparison to single-node libraries installation. However, the concrete solution of course depends on a distributed processing engine and design decisions taken during implementation of a service.

As for future development directions for JupyterHub service, the survey results 4.3 imply that the provided set of configuration options should be expanded. In order to further enhance Jupyter user experience, additional Jupyter extensions and kernels should be provided (see Table 4.8). On top of that, it should be possible to request provisioning of JupyterHub without Python management - it will be especially useful when Waldur will provide in addition, for example, R or F# Jupyter kernels. Besides, when it is expected

that JupyterHub will serve large amount of users it would beneficial to offer option to scale JupyterHub deployment using Kubernetes - a cluster orchestration system. JupyterHub provides appropriate user instance spawner[1].

Furthermore, data staging functionality would be of assistance to lecturers who would like to distribute the material across the students (i.e JupyterHub users). This will be especially important in case of distributed Kubernetes deployment of JupyterHub since infrastructure complexity will be too high for users with little system administration experience. This will enrich application provisioning services with AaaS-like features which many other cloud providers offer (see Section 2.2.2).

When it comes to development and deployment process of these services, proper migration strategies should be devised to ensure backward-compatibility of new releases with the earlier deployed Python management and JupyterHub management environments. In order to make the development process of the developed services more reliable and less error-prone, there should be found a way of incorporating integration tests into a continuous integration pipeline of Waldur.

[1] https://github.com/jupyterhub/kubespawner

Appendix A

Scenarios for Python management service

Scenario 1	Python management environment should be installed on a virtual machine regardless of cloud provider.
Relevant Quality Attributes	Portability
Stimulus	Virtual machines may come from various cloud providers
Environment	Hybrid cloud brokerage system
Artifact	Backend logic
Response	Backend logic should depend on unified virtual machines abstractions, not on concrete virtual machines. Until Python management environment is set up, no actions can be performed by the user.

Table A.1: Python management scenario to ensure portability across supported cloud providers

Scenario 2	User wants to install, uninstall and reinstall (with different version) python libraries in a dedicated virtual environments. It is possible to delete virtual environments.
Relevant Quality Attributes	Operability,
Stimulus	Not user-friendly terminals and commands, conflicting Python libraries
Environment	Virtual machine with or without Python virtual environments
Artifact	Self-service, backend processing, Ansible Playbooks
Response	Appropriate logic for managing separate virtual environment using pip[1] package manager should be developed.

TABLE A.2: Python management scenario for installation and uninstallation of Python libraries in dedicated virtual environments

Scenario 3	User wants to find installed libraries in the virtual environment
Relevant Quality Attributes	Operability, fault-recovery
Stimulus	Users may install some libraries manually through terminal
Environment	Python virtual environments with installed libraries which Waldur is not aware of
Artifact	Self-service, backend processing, Ansible Playbooks
Response	User-triggered logic for indexing installed libraries in the virtual environment. All missing libraries are correctly identified, removed libraries deleted from the database.

TABLE A.3: Python management scenario for functionality to discover manually installed libraries

Scenario 4	User wants to manage each Python virtual environment independently from other virtual environments installed on the same virtual machine.
Relevant Quality Attributes	Operability, observation interval, processing rate
Stimulus	Each virtual environment is independent of other virtual environments
Environment	Waldur running in distributed computing environment with plenty of workers for executing requests, virtual machines potentially have multi-core processors.
Artifact	Self-service, backend logic
Response	Requests related to a specific environment are processed simultaneously and independently. In the meantime, user is able to issue new requests for other virtual environments. Global requests however block the whole screen.

Table A.4: Python management scenario to ensure parallel management of different virtual environments on a virtual machine

Scenario 5	User wants to find existing virtual environments and their installed libraries
Relevant Quality Attributes	Operability, fault-recovery
Stimulus	New virtual environments may be manually installed
Environment	Actually existing Python virtual environments may differ from the ones Waldur has indexed
Artifact	Self-service, backend processing, Ansible Playbooks
Response	Appropriate logic for indexing installed virtual environments and their libraries should be implemented. All missing virtual environments and installed libraries are correctly identified, removed virtual environments and libraries are deleted from the database.

Table A.5: Python management scenario for search of manually created virtual environments on a virtual machine

Scenario 6	User wants to see the history of issued requests and their outputs
Relevant Quality Attributes	Operability, fault-recovery
Stimulus	Some request may fail, some may succeed, user should be able to investigate output of the executed requests to correctly identify root cause of the problem
Environment	Several requests are running simultaneously with different outputs
Artifact	Self-service, backend processing
Response	History of requests is shown with their output. Output of the requests is updated periodically as new lines returned by the command which is being executed.

Table A.6: Python management scenario for audit functionality

Scenario 7	Libraries form should have autocomplete feature. Only compatible with current environment libraries' versions should be shown.
Relevant Quality Attributes	Operability, reliability
Stimulus	Specific Python version or operation system dependent installation packages
Environment	User entering library and selecting its version
Artifact	Backend logic
Response	Waldur should show only those library versions which have compatible with current environment installation package. Possible options of the library names and versions should be visible within 1 second.

Table A.7: Python management scenario for library autocomplete feature in graphical user interface

Scenario 8	User wants to upload Requirements file[2] which contains list of libraries to install in the virtual environment.
Relevant Quality Attributes	Operability, portability
Stimulus	Possibility to rapidly install or migrate virtual environments to another virtual machine
Environment	Python Management is in an operational state with or without virtual environments
Artifact	Self-service
Response	User interface should provide a button to upload list of libraries. Upload functionality should be performed purely on the client side, no server communication is involved. To apply modifications, user need to save Python management screen.

TABLE A.8: Python management scenario for uploading requirements file with list of libraries to install

Scenario 8	User wants to download Requirements file[3] which contains list of installed libraries in the virtual environment.
Relevant Quality Attributes	Operability, portability
Stimulus	Possibility to share or migrate the setup configuration
Environment	Exists Python virtual environment with some libraries
Artifact	Self-service
Response	User interface should provide a button to download list of libraries. The functionality should be performed purely on the client side, no server communication is involved.

TABLE A.9: Python management scenario for downloading list of installed libraries in a virtual environment

Scenario 9	User wants to delete Python environment management
Relevant Quality Attributes	Operability
Stimulus	User no longer needs to manage Python virtual environments
Environment	Python Management is in an operational state with virtual environments or virtual machine no longer exists, or Python environment initialization operation failed
Artifact	Self-service, backend logic, Ansible Playbook
Response	Waldur should trigger deletion of Python virtual environments, however, Python itself should not be deleted (other applications may depend on it).

Table A.10: Python management scenario for removal of virtual environments

Scenario 10	Python version should not be hard coded in the logic, rather the latest available version should be installed.
Relevant Quality Attributes	Adaptability
Stimulus	We want to reduce maintainability burden on developers by always provisioning the latest version of Python distribution
Environment	New Python versions are constantly being released
Artifact	Ansible Playbook
Response	During Python environment installation phase, Playbook logic should always install the latest available version of Python from APT package manager

Table A.11: Python management scenario for provisioning up-to-date version of Python environment

Appendix B

Scenarios for JupyterHub management service

Scenario 1	JupyterHub management environment should be installed on a virtual machine regardless of cloud provider.
Relevant Quality Attributes	Portability
Stimulus	Virtual machines may come from various cloud providers
Environment	Hybrid cloud brokerage system
Artifact	Backend logic
Response	Backend logic should depend on unified virtual machines abstractions, not on virtual machines of concrete cloud providers. Until JupyterHub management environment is set up, no actions can be performed by the user.

Table B.1: JupyterHub management scenario to ensure portability across supported cloud providers

Scenario 2	It should be possible to define time to live for inactive instances of JupyterHub notebook servers.
Relevant Quality Attributes	Capacity
Stimulus	In order to free up resources, it makes sense to shutdown inactive instances
Environment	Virtual machine with deployed JupyterHub
Artifact	Python script
Response	Python script which runs locally on a virtual machine with frequency specified by the user. There should be no notebook instances that are active longer than specified time to live.

Table B.2: JupyterHub management scenario to ensure release of unused computing resources on the virtual machine with instllaed JupyterHub

Scenario 3	It should be possible to define JupyterHub users and administrators.
Relevant Quality Attributes	Confidentiality, integrity
Stimulus	Multiple users working with JupyterHub, JupyterHub is exposed to the outer world
Environment	Multi-tenancy environment exposed outside the cloud firewall
Artifact	Self-service, backend logic, Ansible Playbook
Response	JupyterHub configuration is adapted accordingly, removed users are deleted from the database, new users are added.

Table B.3: JupyterHub management scenario for configuring user authentication

Scenario 4	It should be possible to make managed Python virtual environment available as Jupyter kernels to JupyterHub users.
Relevant Quality Attributes	Operability
Stimulus	Provide Python dependency management for JupyterHub projects through Waldur self-service or API
Environment	Existing virtual machine with Python management environment which contains virtual environments
Artifact	Self-service, backend logic, Ansible Playbook
Response	User is able to mark virtual environments whether they are accessible as Jupyter kernels to all JupyterHub users or not

Table B.4: JupyterHub management scenario for integration of Python and JupyterHub management services

Scenario 5	It should be possible configure GitLab and Azure OAuth2 authentication methods for JupyterHub. It is possible to define admins and whitelisted users.
Relevant Quality Attributes	Confidentiality, integrity, operability
Stimulus	It is important to reduce maintainability effort required from the owner of the JupyterHub server.
Environment	Owner belongs to an organization which is integrated with an OAuth2 provider.
Artifact	Self-service, backend logic, Ansible Playbook
Response	Possibility to configure OAuth2 configuration through a self-service.

Table B.5: JupyterHub management scenario for configuring OAuth2 authentication

Scenario 6	JupyterHub should be served over secure HTTPS connection.
Relevant Quality Attributes	Confidentiality, integrity
Stimulus	JupyterHub users may be subject to malicious attacks.
Environment	Insecure Internet environment
Artifact	Ansible Playbook
Response	Self-signed SSL certificates are generated and JupyterHub is configured to use them. User perform all interactions with JupyterHub only through HTTPs connection.

Table B.6: JupyterHub management scenario to ensure secure connection to JupyterHub

Scenario 7	User wants to see the history of issued JupyterHub management requests and their outputs
Relevant Quality Attributes	Operability, fault-recovery
Stimulus	Some request may fail, some may succeed, user should be able to investigate output of the executed requests to correctly identify root cause of the problem
Environment	Several requests are running simultaneously with different outputs
Artifact	Self-service, backend processing
Response	History of requests is shown with their output. Output of the requests is updated periodically as new lines returned by the command which is being executed.

Table B.7: JupyterHub management scenario for audit functionality

Scenario 8	User wants to delete JupyterHub management
Relevant Quality Attributes	Operability
Stimulus	User no longer needs JupyterHub and Jupyter
Environment	JupyterHub is installed on a virtual machine or virtual machine no longer exists, or JupyterHub initialization operation failed
Artifact	Self-service, backend logic, Ansible Playbook
Response	JupyterHub ans its main dependencies should be removed from a virtual machine.

TABLE B.8: JupyterHub management scenario for JupyterHub uninstallation

Bibliography

[1] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS quarterly*, 28(1):75–105, 2004.

[2] Marcos D. Assuno, Rodrigo N. Calheiros, Silvia Bianchi, Marco A.S. Netto, and Rajkumar Buyya. Big data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing*, 79-80:3 – 15, 2015. ISSN 0743-7315. doi: https://doi.org/10.1016/j.jpdc.2014.08.003. URL http://www.sciencedirect.com/science/article/pii/S0743731514001452. Special Issue on Scalable Systems for Big Data Management and Analytics.

[3] Anand Bagmar. Anand Bagmar - Behavior Driven Testing (BDT) in Agile. https://www.slideshare.net/abagmar/anand-bagmar-behavior-driven-testing-bdt-in-agile, 2012. [Online; accessed 19-April-2018].

[4] Sara Landset, Taghi M. Khoshgoftaar, Aaron N. Richter, and Tawfiq Hasanin. A survey of open source tools for machine learning with big data in the hadoop ecosystem. *Journal of Big Data*, 2(1):24, Nov 2015. ISSN 2196-1115. doi: 10.1186/s40537-015-0032-1. URL https://doi.org/10.1186/s40537-015-0032-1.

[5] Tatiana Polunina Santhosh Konda. Big Data Tutorial 1: MapReduce. https://wikis.nyu.edu/display/NYUHPC/Big+Data+Tutorial+1%3A+MapReduce, 2018. [Online; accessed 26-February-2018].

[6] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, and Samir Belfkih. Big data technologies: A survey. *Journal of King Saud University - Computer and Information Sciences*, 2017. ISSN 1319-1578. doi: https://doi.org/10.1016/j.jksuci.2017.06.001. URL http://www.sciencedirect.com/science/article/pii/S1319157817300034.

[7] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016. URL http://arxiv.org/abs/1603.02754.

[8] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015. URL http://arxiv.org/abs/1512.01274.

[9] C.L. Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275 (Supplement C):314 – 347, 2014. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2014.01.015. URL http://www.sciencedirect.com/science/article/pii/S0020025514000346.

[10] Rekha Nachiappan, Bahman Javadi, Rodrigo N. Calheiros, and Kenan M. Matawie. Cloud storage reliability for big data applications: A state of the art survey. *Journal of Network and Computer Applications*, 97(Supplement C):35 – 47, 2017. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2017.08.011. URL http://www.sciencedirect.com/science/article/pii/S1084804517302734.

[11] Sugam Sharma. Expanded cloud plumes hiding big data ecosystem. *Future Gener. Comput. Syst.*, 59(C):63–92, June 2016. ISSN 0167-739X. doi: 10.1016/j.future.2016.01.003. URL http://dx.doi.org/10.1016/j.future.2016.01.003.

[12] Ilja Livenson. Estonian E-Infrastructure Marketplace. http://e-irg.eu/documents/10920/394570/Estonian+E-Infrastructure+Marketplace+Paradigm.pdf, 2017. [Online; accessed 10-November-2017].

[13] Recent success stories - Estonian Scientific Computing Infrastructure (ETAIS)). https://opennodecloud.com/success.html, 2017. [Online; accessed 07-December-2017].

[14] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39 (1):50–55, December 2008. ISSN 0146-4833. doi: 10.1145/1496091.1496100. URL http://doi.acm.org/10.1145/1496091.1496100.

[15] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall Professional Technical Reference, Upper Saddle River, NJ, 2005. ISBN 0131858580 9780131858589.

[16] Lizhe Wang, R Ranjan, Jinjun Chen, and Boualem Benatallah. *Cloud Computing: Methodology, Systems, and Applications*. 01 2011.

[17] Blesson Varghese and Rajkumar Buyya. Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems*, 79:849 – 861,

2018. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2017.09.020. URL http://www.sciencedirect.com/science/article/pii/S0167739X17302224.

[18] Z. Li, Y. Zhang, and Y. Liu. Towards a full-stack devops environment (platform-as-a-service) for cloud-hosted applications. *Tsinghua Science and Technology*, 22 (01):1–9, February 2017. doi: 10.1109/TST.2017.7830891.

[19] Ruili Wang, Wanting Ji, Mingzhe Liu, Xun Wang, Jian Weng, Song Deng, Suying Gao, and Chang an Yuan. Review on mining data from multiple data sources. *Pattern Recognition Letters*, 2018. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2018.01.013. URL http://www.sciencedirect.com/science/article/pii/S0167865518300199.

[20] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029.

[21] Kumar Ravi, Yogesh Khandelwal, Boora Shiva Krishna, and Vadlamani Ravi. Analytics in/for cloud-an interdependence: A review. *Journal of Network and Computer Applications*, 102:17 – 37, 2018. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2017.11.006. URL http://www.sciencedirect.com/science/article/pii/S1084804517303764.

[22] Ashish Singh and Kakali Chatterjee. Cloud security issues and challenges: A survey. *Journal of Network and Computer Applications*, 79(Supplement C):88 – 115, 2017. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2016.11.027. URL http://www.sciencedirect.com/science/article/pii/S1084804516302983.

[23] Stefania Costache, Djawida Dib, Nikos Parlavantzas, and Christine Morin. Resource management in cloud platform as a service systems: Analysis and opportunities. *Journal of Systems and Software*, 132:98 – 118, 2017. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2017.05.035. URL http://www.sciencedirect.com/science/article/pii/S0164121217300845.

[24] Using buildpacks in IBM Cloud Private Cloud Foundry. https://www.ibm.com/support/knowledgecenter/en/SSBS6K_2.1.0/cloud_foundry/buildpacks/buildpacks_overview.html, 2018. [Online; accessed 23-February-2018].

[25] Chris Lauwers Paul Lipton. OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca, 2018. [Online; accessed 23-February-2018].

[26] M. Zimmermann, F. W. Baumann, M. Falkenthal, F. Leymann, and U. Odefey. Automating the provisioning and integration of analytics tools with data resources

in industrial environments using opentosca. In *2017 IEEE 21st International Enterprise Distributed Object Computing Workshop (EDOCW)*, pages 3–7, Oct 2017. doi: 10.1109/EDOCW.2017.10.

[27] Benefits of Using Amazon EMR. https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-overview-benefits.html, 2018. [Online; accessed 29-March-2018].

[28] Introduction - What is Azure Notebooks? https://notebooks.azure.com/help/introduction, 2018. [Online; accessed 25-March-2018].

[29] Manikanta Yadunanda. Fast.ai Lesson 1 on Google Colab (Free GPU). https://towardsdatascience.com/fast-ai-lesson-1-on-google-colab-free-gpu-d2af89f53604, 2018. [Online; accessed 24-April-2018].

[30] Connection types. https://datascience.ibm.com/docs/content/manage-data/conn_types.html, 2018. [Online; accessed 29-March-2018].

[31] Watson Studio overview. https://datascience.ibm.com/docs/content/getting-started/overview-ws.html?context=analytics, 2018. [Online; accessed 29-March-2018].

[32] Databricks. Get the simplifying apache spark operations whitepaper. 2016. URL http://go.databricks.com/spark-operations-with-databricks-whitepaper.

[33] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, November 2011. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1953048.2078195.

[34] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 4th edition, 2015. ISBN 1491901632, 9781491901632.

[35] MapReduce Tutorial. https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html, 2018. [Online; accessed 26-February-2018].

[36] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and*

Implementation, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2228298.2228301.

[37] Welcome to H2O 3. http://docs.h2o.ai/h2o/latest-stable/h2o-docs/welcome.html, 2018. [Online; accessed 27-February-2018].

[38] A real-time processing revival. https://www.oreilly.com/ideas/a-real-time-processing-revival, 2015. [Online; accessed 26-February-2018].

[39] Sergio Ramrez-Gallego, Alberto Fernndez, Salvador Garca, Min Chen, and Francisco Herrera. Big data: Tutorial and guidelines on information and process fusion for analytics algorithms with mapreduce. *Information Fusion*, 42(Supplement C):51 – 61, 2018. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2017.10.001. URL http://www.sciencedirect.com/science/article/pii/S1566253517305912.

[40] Joseph Torres, Michael Armbrust, Tathagata Das and Shixiong Zhu . Introducing Low-latency Continuous Processing Mode in Structured Streaming in Apache Spark 2.3. https://databricks.com/blog/2018/03/20/low-latency-continuous-processing-mode-in-structured-streaming-in-apache-spark-2-
html, 2018. [Online; accessed 31-March-2018].

[41] Structured Streaming Programming Guide. https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html, 2018. [Online; accessed 28-February-2018].

[42] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy, and Siddarth Taneja. Twitter heron: Stream processing at scale. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 239–250, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-2758-9. doi: 10.1145/2723372.2742788. URL http://doi.acm.org/10.1145/2723372.2742788.

[43] Vctor Campos, Francesc Sastre, Maurici Yages, Mriam Bellver, Xavier Gir i Nieto, and Jordi Torres. Distributed training strategies for a computer vision deep learning algorithm on a distributed gpu cluster. *Procedia Computer Science*, 108:315 – 324, 2017. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2017.05.074. URL http://www.sciencedirect.com/science/article/pii/S1877050917306129. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.

[44] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray,

Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, Berkeley, CA, USA, 2016. USENIX Association. ISBN 978-1-931971-33-1. URL http://dl.acm.org/citation.cfm?id=3026877.3026899.

[45] How is Caffe2 different from PyTorch? https://caffe2.ai/docs/caffe-migration.html#null__how-is-caffe2-different-from-pytorch, 2018. [Online; accessed 4-March-2018].

[46] PyTorch — About. http://pytorch.org/about/, 2018. [Online; accessed 4-March-2018].

[47] Automatic differentiation. https://mxnet.incubator.apache.org/tutorials/gluon/autograd.html, 2018. [Online; accessed 02-April-2018].

[48] Why MXNet? - Symbolic Programming in MXNet. https://mxnet.incubator.apache.org/faq/why_mxnet.html#symbolic-programming-in-mxnet, 2018. [Online; accessed 02-April-2018].

[49] Mario Barbacci, Robert Ellison, Anthony Lattanze, Judith Stafford, Charles Weinstock, and William Wood. Quality attribute workshops (qaws). Technical Report CMU/SEI-2003-TR-016, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2003. URL http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=6687.

[50] Richardson Maturity Model. https://martinfowler.com/articles/richardsonMaturityModel.html, 2010. [Online; accessed 12-March-2018].

[51] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. ISBN 0321200683.

[52] Introduction to Celery. http://docs.celeryproject.org/en/latest/getting-started/introduction.html, 2018. [Online; accessed 14-March-2018].

[53] Ansible Documentation - Best Practices. http://docs.ansible.com/ansible/latest/playbooks_best_practices.html, 2018. [Online; accessed 18-March-2018].

[54] Infrastructure as Code. https://en.wikipedia.org/wiki/Infrastructure_as_Code, 2018. [Online; accessed 19-March-2018].

[55] The Principles of OOD. http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod, 2018. [Online; accessed 21-March-2018].

[56] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. ISBN 0321127420.

[57] Toby Chemson. Testing Strategies in a Microservice Architecture. https://martinfowler.com/articles/microservice-testing/#conclusion-test-pyramid, 2014. [Online; accessed 31-March-2018].

[58] Tim Ottinger, Jeff Langr, Brett Schuchert. F.I.R.S.T. http://agileinaflash.blogspot.com.ee/2009/02/first.html, 2009. [Online; accessed 19-April-2018].

[59] Tom Zeng. Run Jupyter Notebook and JupyterHub on Amazon EMR. https://aws.amazon.com/blogs/big-data/running-jupyter-notebook-and-jupyterhub-on-amazon-emr/, 2016. [Online; accessed 27-March-2018].

[60] Jon Fritz. Run Deep Learning Frameworks with GPU Instance Types on Amazon EMR. https://aws.amazon.com/blogs/machine-learning/run-deep-learning-frameworks-with-gpu-instance-types-on-amazon-emr/, 2017. [Online; accessed 27-March-2018].