Designing an infrastructure for on-premise software deployment

Citation for published version (APA):

Document status and date:
Published: 24/10/2019

Publisher Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 30. Apr. 2020
Designing an infrastructure for On-Premise Software Deployment

Giovanni de Almeida Calheiros
DESIGNING AN INFRASTRUCTURE FOR ON-PREMISE SOFTWARE DEPLOYMENT

A STUDY CASE FOR THERMO FISHER DATA MANAGEMENT PLATFORM

Giovanni de Almeida Calheiros

Eindhoven University of Technology
Stan Ackermans Institute / Software Technology

Partners

Thermo Fisher Scientific

Eindhoven University of Technology

Steering Group

Ir. H.T.G. Weffers PDEng (TU/e)
Ir. E. Algra, PDEng (Thermo Fisher Scientific)

Date

October 2019

Document Status

Public

SAI report no.

2019/090

The design described in this report has been carried out in accordance with the TU/e Code of Scientific Conduct.
Surrounding all Thermo Fisher Scientific equipment, there are a lot of pieces of software that need to be installed and, periodically, updated either to fix bugs or to implement new functionalities. The time required to install and update all these pieces of software is proportional to the number of pieces of software that need to be available. Finding out an approach to, automate the routine of installing and upgrading the microscope’s software is the primary focus of this project. The immediate effects of the implementation of the designed approach are reduction of the time between software release and installation, a more uniform installed base, and an easier procedure to upgrade or revert upgrades.

Keywords
PDEng, Software Technology, TU/e, Thermo Fisher Scientific, Software Delivery, Continuous Integration, Kubernetes, Packaging, Docker, Helm

Preferred reference

Partnership
This project was supported by Eindhoven University of Technology and Thermo Fisher Scientific.

Disclaimer
Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the Eindhoven University of Technology or Thermo Fisher Scientific. The views and opinions of authors expressed herein do not necessarily state or reflect those of the Eindhoven University of Technology or Thermo Fisher Scientific, and shall not be used for advertising or product endorsement purposes.

Disclaimer liability
While every effort will be made to ensure that the information contained within this report is accurate and up to date, Eindhoven University of Technology makes no warranty, representation or undertaking whether expressed or implied, nor does it assume any legal liability, whether direct or indirect, or responsibility for the accuracy, completeness, or usefulness of any information.

Trademarks
Product and company names mentioned herein may be trademarks and/or service marks of their respective owners. We use these names without any particular endorsement or with the intent to infringe the copyright of the respective owners.

Copyright
Copyright© 2019. Eindhoven University of Technology. All rights reserved. No part of the material protected by this copyright notice may be reproduced, modified, or redistributed in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the Eindhoven University of Technology and Thermo Fisher Scientific.
This PDEng thesis is approved by the supervisors and the Thesis Evaluation Committee is composed of the following members:

Chair: Ir. H.T.G. Weffers PDEng

Supervisors: Ir. E. Algra, PDEng
Ir. H.T.G. Weffers PDEng

Members: Dr. Ing. Anton Wijs
Dr. A. Radulescu
Foreword

The Materials and Structural Analysis Division (MSD) business of Thermo Fisher Scientific (formerly known as FEI) has traditionally been a global leader in the innovation of electron microscopes (EM). Software has played an increasing role in the delivery of the innovations, yet so far has focused on the instruments themselves.

The new area the company is moving towards is to deliver solutions that support the workflow of the customer using various instruments, covering data management as well as data post processing. In order to be successful, Thermo Fisher Scientific needs the ability to deliver (pure) software solutions as managed service, interfacing with customer infrastructure, and due to the nature of the customer and instruments, deployed and managed within the premises of the customer.

To support that venue, the Data Management Platform (DMP) project designs, implements, and delivers a Software Delivery Platform (SDP) as a key enabler for delivering managed software services. This is a greenfield area for the company,

Giovanni has contributed in a significant manner. His background in IT with Linux system administration, automation, and systems monitoring has jumpstarted significant parts of the realization of our ambitions. He drove the realization of the entire automated build infrastructure, facilitated the deployment automation of build results, built lightweight yet useful monitoring capabilities, and assisted in product deployments and troubleshooting that come with such first of a kind developments.

I enjoyed having Giovanni as a team member and got to know him as a modest, competent and committed colleague. As a bonus, he brings a good sense of humor and appreciation for good coffee. I am delighted to welcome him as a permanent member of our team within Thermo Fisher Scientific on the successful completion of his assignment.

Project Mentor
Ir. E. Algra, PDEng
October 2019
Preface

The technological designer programs lead to a Professional Doctorate in Engineering (PDEng) degree. These programs are given under the banner of the 4TU.School for Technological Design, Stan Ackermans Institute, a joint initiative of the four universities of technology in The Netherlands. The Software Technology (ST) program is part of the Department of Mathematics and Computer Science of the Eindhoven University of Technology. The PDEng is focused on improving the trainees’ technical and non-technical skills related to effective and efficient design and team-working.

This report describes the final PDEng project, which is entitled "Designing an infrastructure for On-Premise Software Deployment." It was proposed by Ir. E. Algra, PDEng, the Software Senior Staff Architect of Thermo Fisher Scientific and the university supervisor was Ir. H.T.G. Weffers PDEng. Basically, the goals of this project are researching automation techniques, using as the main inspiration the cloud computing environment that can be applied in the customer on-premise environment.

The content of this report covers the analysis, design, and implementation of a continuous delivery pipeline for the Thermo Fisher Scientific equipment and focuses on the High-End Transmission Electron Microscope used for life sciences.

Giovanni de Almeida Calheiros
October 2019
Acknowledgements

I would first like to thank Eindhoven University of Technology (TU/e) and Thermo Fisher Scientific for this project opportunity. I would like to express my gratitude to all the people who are involved in this research project.

I am grateful to my university supervisor Ir. H.T.G. Weffers PDEng for giving me valuable feedbacks and helping me to manage this project. I also want to express appreciation to my Thermo Fisher Scientific supervisor, Ir. E. Algra, PDEng for his encouragement support through the process of this PDEng project. During the last ten months they were more than simply supervisor and they helped me a lot in writing my thesis and during my presentations.

I would like to say thank you to Abhi Barve, Chris Schlichten, Ivanka Spee, Joren Bresser, Maarten Dekkers, Maarten Kuijper, Tarkan Akcay, and Tor Halsan for sharing their knowledge and time with me.

I am grateful to all my PDEng colleagues, especially the Thermo Fisher Scientific ones who help me to take my project easy. Beza, Karms, Kon, and Pranav the Thermo Fisher group is the best one and I wish all the best for yours future.

Gostaria de agradecer também a minha família por todo suporte que recebi durante esses meses de curso/projeto no qual fiquei bastante estressado, mas sempre pude contar com eles para conversar. Um especial muito obrigado para meu pai que me ouviu reclamando de quase tudo e de quase todos e sempre me pediu paciência; para minhas tias Lenilda, Leny, Marta, e Tânia que sempre estiveram prontas para conversar comigo quando precisai; ao tio Nilton que graças ao conselho sobre pôr uma vaca no meio da sala, me fez ver as coisas por outro ângulo; À Vó Moça, Vovó Cordélia e Vovô Edmar com quem eu ria bastante ao contar minhas aventuras aqui; sem esquecer meus outros familiares e amigos do Brasil. Back to English....

I am extremely thankful to the Eindhoven University of Technology (TU/e), for having me as a Professional Doctorate in Engineering (PDEng) trainee. In particular, I would like to thank Yanja Dajsuren, Judith Strother, Désirée Oorschot, and all the other professors for their guidance during the past two years.

Finally, I would like to express my heartfelt gratitude to Weslley who has always been by my side for the past years.

Giovanni de Almeida Calheiros
October 2019
Executive Summary

Over the years, equipment for serving science such as microscopes has evolved. Its capability to generate the data as well as its needs for supporting software has increased. Due to the amount of data generated by the equipment, problems such as data storage, data analysis, and online collaboration have become more and more important. To address these problems, Thermo Fisher Scientific created the Data Management Platform (DMP).

DMP is a digital platform to enable easier and safe data management, efficient team work, online collaboration, and live image viewing. To implement these, the DMP platform requires automated application management and packaged software ready to use.

The project described here is responsible for improving the Application Lifecycle Management (ALM) in the DMP context. Tasks such as software packaging, software delivery, and software management were addressed during this project.

Some of the major outcomes of this project have resulted in the development of Continuous Delivery (CD) Pipeline, and the tools for managing software deployment and delivery that were evaluated. The knowledge gathered during this project has become a guideline that has to be incorporated into the DMP software development process.

With the advent of these outcomes, Thermo Fisher Scientific can transform the users’ experience in many ways: using new technologies in the serving science equipment’s software, reducing the time between software releasing and deploying, and enabling the customers to decide when they want to use for the latest software available into their equipment.

In the future, the process of continuous delivery that was implemented during this project can evolve to a continuous deployment approach as long as the validation and verification of the software and the platform can be improved.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreword</td>
<td>i</td>
</tr>
<tr>
<td>Preface</td>
<td>ii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>iii</td>
</tr>
<tr>
<td>Executive Summary</td>
<td>iv</td>
</tr>
<tr>
<td>List of figures</td>
<td>vii</td>
</tr>
<tr>
<td>List of tables</td>
<td>viii</td>
</tr>
<tr>
<td>Abbreviation and Acronyms</td>
<td>x</td>
</tr>
</tbody>
</table>

1 Introduction

1.1 Context                          2
1.2 Project Scope and Goals          3
1.3 Outline                          3

2 Stakeholder Analysis               4

3 Problem Analysis                   6

3.1 Problem                          6
3.2 Solution                          6
3.3 Goal and objectives              6

4 Domain Analysis                    8

4.1 Data Management Platform         8
4.2 Containers                       9
4.2.1 Docker                         9
4.3 Cluster Orchestration            10
4.3.1 Kubernetes ................................................................. 10
4.4 Application Lifecycle Management ........................................ 11
  4.4.1 Helm ................................................................. 11
4.5 Continuous Integration ......................................................... 12
  4.5.1 Jenkins ............................................................... 12
4.6 Configuration Management Tool ............................................. 12
  4.6.1 Ansible ............................................................... 13

5 Feasibility Analysis .............................................................. 14

6 Use Cases and System Requirements ........................................... 16
  6.1 Use cases ................................................................. 16
  6.2 Requirements ............................................................. 17

7 Solution Architecture ............................................................ 18
  7.1 Packaging Phase ......................................................... 18
  7.2 Delivery and Deployment Phase .......................................... 19
    7.2.1 Package Delivery .................................................... 19
    7.2.2 Application Deployment ............................................. 20
  7.3 Monitoring phase .......................................................... 20

8 Deployment and Implementation ................................................. 22
  8.1 Packaging ................................................................. 22
    8.1.1 Continuous Integration Server ...................................... 23
    8.1.2 Remote Builders ..................................................... 25
  8.2 Publishing ................................................................. 28
  8.3 Delivery ................................................................. 29
    8.3.1 Ansible ............................................................... 29
  8.4 Deployment ............................................................... 30
    8.4.1 Helm ................................................................. 30
  8.5 Deployed Solution .......................................................... 31

9 Verification and Validation ...................................................... 32
  9.1 Migration from manual to automated .................................... 32
  9.2 Automated Verification .................................................. 32
    9.2.1 Platform specific .................................................... 32
    9.2.2 Application specific ................................................ 33
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.3</td>
<td>Monitoring</td>
<td>33</td>
</tr>
<tr>
<td>9.4</td>
<td>Expert Validation</td>
<td>33</td>
</tr>
<tr>
<td>10</td>
<td>Conclusions</td>
<td>34</td>
</tr>
<tr>
<td>10.1</td>
<td>Results</td>
<td>34</td>
</tr>
<tr>
<td>10.2</td>
<td>Future Work</td>
<td>34</td>
</tr>
<tr>
<td>11</td>
<td>Project Management</td>
<td>36</td>
</tr>
<tr>
<td>11.1</td>
<td>Process</td>
<td>36</td>
</tr>
<tr>
<td>11.2</td>
<td>Infrastructure plan</td>
<td>37</td>
</tr>
<tr>
<td>11.2.1</td>
<td>Communication and Information sharing</td>
<td>37</td>
</tr>
<tr>
<td>11.2.2</td>
<td>Documentation</td>
<td>37</td>
</tr>
<tr>
<td>11.2.3</td>
<td>Development</td>
<td>37</td>
</tr>
<tr>
<td>11.3</td>
<td>Milestone Trend Analysis</td>
<td>38</td>
</tr>
<tr>
<td>11.4</td>
<td>Risk Analysis</td>
<td>39</td>
</tr>
<tr>
<td>12</td>
<td>Project Retrospective</td>
<td>42</td>
</tr>
<tr>
<td>12.1</td>
<td>Reflection and Lessons Learned</td>
<td>42</td>
</tr>
<tr>
<td>Appendix</td>
<td></td>
<td>45</td>
</tr>
<tr>
<td>A</td>
<td>Self-contained docker environment</td>
<td>46</td>
</tr>
<tr>
<td>A.1</td>
<td>Self-contained docker environment</td>
<td>46</td>
</tr>
<tr>
<td>B</td>
<td>Comparative Studies</td>
<td>48</td>
</tr>
<tr>
<td>B.1</td>
<td>Continuous Integration Tool</td>
<td>48</td>
</tr>
<tr>
<td>B.2</td>
<td>Configuration Management Tools</td>
<td>49</td>
</tr>
<tr>
<td>C</td>
<td>Detailed use cases</td>
<td>50</td>
</tr>
</tbody>
</table>
# List of Figures

4.1 Containerized applications architecture vs Virtualized Applications architecture [1] . 9

7.1 Continuous Delivery Pipeline Architecture ........................................ 18
7.2 Building sub-system architecture ...................................................... 19
7.3 Monitoring architecture ................................................................. 20

8.1 High level view of the components involved in the packaging sub-system .... 23
8.2 Components of the basic builder ...................................................... 25
8.3 Components of the Athena builder ................................................... 27
8.4 Components of the DMP builder ...................................................... 27
8.5 Activities performed from the moment that one building is triggered until the artifacts become available .................................................. 28
8.6 Helm activity flow ....................................................................... 31

11.1 Summarized initial project’s milestones .......................................... 36
11.2 Detailed Milestone Trend Analyses ................................................ 38
List of Tables

6.1 Use case list ......................................................... 16
6.2 Functional Requirements ........................................... 17
6.3 Non-Functional Requirements ..................................... 17

7.1 Packaging architecture ............................................. 19
7.2 Package delivery step ............................................... 20
7.3 Application Deployment step ....................................... 20
7.4 Monitoring architecture ............................................ 21

11.1 Project’s Risks ....................................................... 39
11.2 Project’s Risk Mitigation and Contingency Strategies ............. 40

B.1 Continuous Integration Server summarized study ................. 48
B.2 Configuration Management Tool Comparison criteria .............. 49

C.1 Use Case 01: Release artifact version .............................. 51
C.2 Use Case 02: Building Artifact .................................... 52
C.3 Use Case 03: Install application ................................. 53
C.4 Use Case 04: Upgrade application ............................ 53
C.5 Use Case 05: Rollback deployment ............................ 54
C.6 Use Case 06: Delete application .............................. 54
C.7 Use Case 07: Configure host ................................... 55
Abbreviation and Acronyms

ALM  Application Lifecycle Management.
API  Application programming interface.
BATS  Bash Automated Testing System.
CD  Continuous Delivery.
CI  Continuous Integration.
DMP  Data Management Platform.
DNS  Domain Name System.
DSE  Digital Service Engineer.
FSE  Field Service Engineer.
GPU  Graphics Processing Unit.
HE-TEM  High-End Transmission Electron Microscope.
PDEng  Professional Doctorate in Engineering.
SDP  Software Delivery Platform.
ssh  Secure Shell.
TDD  Test-Driven Development.
TEM  Transmission Electron Microscope.
tls  Transport Layer Security.
TU/e  Eindhoven University of Technology.
winrm  Windows Remote Management.
1 Introduction

In this chapter, we introduce the overall problem that this project addresses as well as the context in which the solution is developed. The Thermo Fisher Scientific environment and the Continuous Delivery challenges are described as well as the project’s scope and goals. At the end of this chapter, the structure of the document is mentioned.

This project was developed with Thermo Fisher Scientific’s Advanced Technology department, and more specifically with the Data Management Platform (DMP) and Software Delivery Platform (SDP) projects. Their focus is on evolving the High-End Transmission Electron Microscope’s software architecture to achieve a more robust and reliable solution.

1.1 Context

Thermo Fisher Scientific is known as the world leader in serving science. In its portfolio, there is a lot of equipment, from plastic gloves to the High-End Transmission Electron Microscope (HE-TEM). All this equipment generates a certain amount of data that is used by other equipment or analyzed by researchers. The microscope’s capability to generate and consume data is continually increasing and the ability to analyze this data has to follow this trend. However, nowadays the data produced by the microscope remains located on it.

The DMP project includes the development of a digital platform to surround the microscope and add extra functionality to it, apart from displaying images using sub-Ångström resolution (1/10,000,000,000 of a meter). The DMP is enabling hassle free and safe data management, efficient teamwork, online collaboration, and live image viewing. Under the DMP project, there are other projects to address different aspects of the platform, one of which is the Software Delivery Platform project that is responsible for providing and coordinating the software required by the DMP.

The SDP is responsible for the infrastructure in which the DMP is installed. Tasks, such as installing software, orchestrating the services, and monitoring the platform’s resources, are performed by the SDP. Therefore, the SDP project addresses concepts such as micro-services, software delivery, manageability, and scalability for the DMP project in an on-premise environment.

This project is entitled "Designing an infrastructure for On-Premise Software Deployment” and it is part of the SDP project. It aims to add capabilities strongly related to processing automation for the higher level projects and enabling Thermo Fisher Scientific to reduce the time between software release and installation.
1.2 Project Scope and Goals

The scope of this project includes the parts of the process of artifact build, artifact delivery, application lifecycle management, and configuration management. In the context of the building process, the focus is on providing a centralized and automated solution to accommodate different building processes with conflicting requirements.

The artifact delivery process, the application life-cycle management, and the configuration management are highly connected. The first one is responsible for providing the artifacts required by the customer as soon as they are available. The second one is responsible for managing the applications deployed in the platform. The last one is controls the installation and configuration of the platform itself. The focus of this project is on evaluating possible technologies as well as implementing prototypes capable of distributing the artifacts to the customers. Besides of those, monitoring and keeping the health of the entire system are also important aspects of the project. As some parts of the project were developed independently, detailed descriptions of the scope and the goals are given separately.

1.3 Outline

Chapter 2 presents who are the stakeholders of this project, what the preferred communication media are, and where they are located. Chapter 3 describes the problem that this project aims to solve, the proposed solution, as well as goals of the project. Chapter 4 gives an overview of the projects in which this project is included, as well as the technologies that surround this project. Chapter 5 presents the feasibility analysis and some concerns for this project. Chapter 6 explains the use cases and the requirements approached in this project. Chapters 7 and 9 present the architecture of the solution and the process used to verify and validate the project respectively. Chapter 8 presents how the proposed solution is implemented.
2 Stakeholder Analysis

This chapter introduces the stakeholders of the project as well as provides further details about their relationship with this project. They mainly belong to two organizations: Thermo Fisher Scientific and TU/e. However, some of the stakeholders are remotely located and the customers’ researchers are also considered here because they interact indirectly with this project.

<table>
<thead>
<tr>
<th>Role</th>
<th>Project Supervisor from Thermo Fisher Scientific</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ir. E. Algra, PDEng</td>
<td>Successful completion of the project with a good quality prototype Spread the acquired knowledge to the team</td>
</tr>
<tr>
<td>Preferred Communication Channel</td>
<td>In Person</td>
</tr>
<tr>
<td>Update Frequency</td>
<td>Daily</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Role</th>
<th>Project Supervisor from TU/e</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ir. H.T.G. Weffers PDEng</td>
<td>Provide guidance and feedback to me Supervise the project from the perspective of the TU/e</td>
</tr>
<tr>
<td>Preferred Communication Channel</td>
<td>In Person or email</td>
</tr>
<tr>
<td>Update Frequency</td>
<td>Biweekly</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Role</th>
<th>PDEng ST Program Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yanja Dajsuren, PhD.</td>
<td>Successful completion of the project with a good quality report</td>
</tr>
<tr>
<td>Preferred Communication Channel</td>
<td>In person</td>
</tr>
<tr>
<td>Update Frequency</td>
<td>ad-Hoc</td>
</tr>
</tbody>
</table>
EPU Team

<table>
<thead>
<tr>
<th>Role</th>
<th>Internal Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interests</td>
<td>Perform integration test using the platform</td>
</tr>
<tr>
<td>Preferred Communication Channel</td>
<td>In person, Microsoft Teams, or email</td>
</tr>
<tr>
<td>Update Frequency</td>
<td>ad-Hoc</td>
</tr>
</tbody>
</table>

The following tables present the remotely located stakeholders.

Tor Halsan

<table>
<thead>
<tr>
<th>Role</th>
<th>Linux and Security Specialist from Thermo Fisher Scientific</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interests</td>
<td>Develop a stable and scalable solution</td>
</tr>
<tr>
<td>Preferred Communication Channel</td>
<td>Microsoft Teams or email</td>
</tr>
<tr>
<td>Update Frequency</td>
<td>Daily</td>
</tr>
<tr>
<td>Located at</td>
<td>Trondheim / NO</td>
</tr>
</tbody>
</table>

Chris Schlichten

<table>
<thead>
<tr>
<th>Role</th>
<th>Windows, Hardware, and Hypervisor Specialist from Thermo Fisher Scientific</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interests</td>
<td>Develop a stable and scalable solution</td>
</tr>
<tr>
<td>Preferred Communication Channel</td>
<td>Microsoft Teams or email</td>
</tr>
<tr>
<td>Update Frequency</td>
<td>Twice a week</td>
</tr>
<tr>
<td>Located at</td>
<td>Hillsboro / US</td>
</tr>
</tbody>
</table>

Athena Team

<table>
<thead>
<tr>
<th>Role</th>
<th>Internal Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interests</td>
<td>Uses an stable platform</td>
</tr>
<tr>
<td></td>
<td>Have an easy procedure to deliver and deploy software</td>
</tr>
<tr>
<td>Preferred Communication Channel</td>
<td>Microsoft Teams or email</td>
</tr>
<tr>
<td>Update Frequency</td>
<td>ad-Hoc</td>
</tr>
<tr>
<td>Located at</td>
<td>Bourdeaux / FR</td>
</tr>
</tbody>
</table>

Fleet Manager Team

<table>
<thead>
<tr>
<th>Role</th>
<th>Internal Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interests</td>
<td>Uses an stable platform</td>
</tr>
<tr>
<td></td>
<td>Have an easy procedure to deliver and deploy software</td>
</tr>
<tr>
<td>Preferred Communication Channel</td>
<td>Microsoft Teams or email</td>
</tr>
<tr>
<td>Update Frequency</td>
<td>ad-Hoc</td>
</tr>
<tr>
<td>Located at</td>
<td>Hillsboro / US</td>
</tr>
</tbody>
</table>

5 Designing an infrastructure for On-Premise Software Deployment
3 Problem Analysis

Nowadays, the HE-TEM needs a lot of pieces of software to work. Even though the microscope’s hardware and software are similar across machines, in the installation process, they are frequently tweaked by the Field Service Engineer (FSE). The tweak made by the FSE consists of manually edit application files in order to fill in some parameters for that specific deployment. As a consequence, during the lifetime of that deployment, whenever an upgrade needs to be performed, the same parameters have to applied in the new version of the software. With the introduction of the DMP platform, there are more several parameters to be filled in the each application deployment and keeping track of any tweak made as well as which parameters were used are challenges faced by the FSE.

3.1 Problem

The current situation presents two problems:

- The FSE needs to know beforehand the previously used parameters. Only after studying the current installation and gather all parameters, the FSE can start updating the application files and upgrading the software.
- As the FSE changes the application files directly, there is no easy way to revert the application to the previously deployed version.

3.2 Solution

A solution to the problems presented in the above paragraphs is divided in two parts. The first part consists of changing the application in order to accommodate any deployment-specific configuration in a reusable way. The second part is to automate the management of the lifecycle of any application. Separating the deployment-specific configuration from the application definition removes the complexity of changing any part of the application before upgrading it. This separation happens before the first installing and after that it is simply updated, if needed, and reused. Automating the management of the lifecycle of the applications minimizes the possibility of error at the same time that enables an easier way to revert any unexpected behavior after upgrading.

3.3 Goal and objectives

The goal of the project are to

6 Designing an infrastructure for On-Premise Software Deployment
• Research technologies for automating Docker/Kubernetes application lifecycle management.

• Define the approach and create the implementation to roll-out software products, accommodating customer deployment configurations and cluster topologies.

• Automate the deployment (install, upgrade) of on-premise application deployments using cloud technologies such as Kubernetes and Docker.
4 Domain Analysis

This chapter includes information about the domain in which the project is located. The DMP is the software stack that surrounds the HE-TEM. In addition, the roadmap is presented.

4.1 Data Management Platform

Among all products manufactured by Thermo Fisher Scientific the HE-TEM can be considered the most impressive one. The HE-TEM can capture images in sub-Ångström resolution[2]. This resolution gives to the researcher the ability to look at the details of the matter. However, in order to reach this resolution a big part of the process of using a microscope was automated. This automation process has three direct effects:

• The process of automating the operation of a TEM implies that a large amount of code needs to be deployed together with each piece of equipment;

• Each image acquired in high resolution results in a file with several megabytes or gigabytes that is stored in one piece of equipment close to the microscope. Those images are generated at a frequency of seven images per minute;

• Frequently, during the installation process of a TEM, some parameters of the system (in general, configurations or installed software) need to be tweaked, since the hardware configuration of each piece of equipment can be slightly different.

Another interesting point is related to the way the TEM has been used. In general, some domain expert operates the microscope to acquire images. After the acquisition process has been finished, the researcher has to copy the data from the microscope’s local storage to another device in order to start the analysis process.

To solve some of the above challenges, Thermo Fisher Scientific created a project named DMP. Some of the goals of the DMP project are to enable a better integration across Thermo Fisher Scientific equipment as well as improve the aggregation of the generated information. As DMP is a new concept inside Thermo Fisher Scientific, DMP is allowed to introducing modern technologies. For example, instead of a large-specialized software for each TEM, the microscope’s surrounding software must use a more flexible architecture, such as micro-services architecture. Instead of deploying the software components directly in the machine, these components are packaged using container technology and managed in a cluster way.
4.2 Containers

Traditionally deployed application software has its software requirements and files that can be spread across the operating system where the application is installed. Placing dependencies in the operating system can lead to some issues: conflicting dependencies exist among different applications, installing process requires knowledge about the application’s requirements and about the current environment, and installing process in a new environment increases in complexity.

A container is basically a different way of thinking about the creation, design, and deploying of one application. Instead of installing the application’s dependencies directly into the operational system, the application with all of its dependencies are packaged together into a single unit (container image) that can be run in a sandboxed environment. This sandbox environment allows applications with conflicting requirements to be deployed side-by-side on the same machine without any side effects.

In a way, a container is bit like a virtual machine. Unlike a virtual machine, rather than creating a whole virtual operating system for each deployed system, a container allows applications to use the same Linux kernel as the system that they are running on and only requires applications be shipped with things not already running on the host computer, as presented in Figure 4.1.

![Containerized Applications architecture vs Virtualized Applications architecture](image)

Figure 4.1: Containerized applications architecture vs Virtualized Applications architecture [1]

There are several container technology providers such as: Docker, CoreOS rkt, Mesos Containerizer, LXC Linux Containers, Windows Server Containers, and Hyper-V Containers. This project uses Docker because it is open-source software, free of charge even for commercial uses, and widely used.

4.2.1 Docker

Docker was released in 2013 by Docker, Inc. [1] and it is considered the world’s leading software container platform. Some drivers for this technology are available in the following list and they were extracted from [3] [4].

- Uniform and stable environment across developers, testing, and production environments
• Easier deployment process: As the Docker image contains everything that is required by the contained application, the deployment process is simplified to run that container

• Easier working in teams: As the Docker image can be run without the need of installing extra software, multiple teams can use a similar application and environment for development and debugging purposes

• Resource usage improvement: Many applications can run simultaneously in the same environment without interference

4.3 Cluster Orchestration

Containers are a good way to bundle and run applications. Keeping everything that is required by the application as a single unit makes the deployment of that specific application much easier. In production environments, as applications must be available to the customers, they are deployed in several connected computers and this set of computers is called cluster. In one cluster, there are several applications deployed that can lead to some questions: How should the containerized application be handled to avoid downtime? Where should each application be deployed? What should happen if the application is not working?

To respond to the above questions, in general, automated tools are used to managing the environment where the applications are deployed. It takes care of the application scaling requirements, failover, and deployment patterns. One example of this class of tooling is described below:

4.3.1 Kubernetes

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services that facilitates both declarative configuration and automation[5, 6].

Kubernetes brings some out-of-box capabilities:

• Service discovery and load balancing: Kubernetes can expose containers’ services using DNS names or their own IP address. The traffic to a service is load-balanced by Kubernetes and distributed across the containers that have the service so that the deployment is stable.

• Storage orchestration: Kubernetes allows automated-storage provisioning for the application. It can automatically mount a chosen storage system, such as local storage, remote storage, and public cloud providers.

• Container orchestration: Each container can have defined how much CPU and memory (RAM) it needs. When containers have the resource requests specified, Kubernetes can decide in a more appropriate way how to manage the containers.

• Service High-availability: The idea of High-availability here comes from the fact that Kubernetes, in general, coordinates more than one machine and can deploy services in any machine whenever it is required. For instance, if one of the Kubernetes nodes is not available, all services that used to be deployed there will be migrated automatically.
• **Services self-healing and monitoring**: Kubernetes validates the health of a container using a user-defined health check, the container status, and the resource usage. Based on that information, it can restart containers that fail, replace containers, and kill containers that do not respond to the user-defined health check. It prevents Kubernetes from exposing a service that is not working properly.

• **Secret and configuration management**: Kubernetes has a safe place to store and manage sensitive information, such as passwords, tokens, and tls certificates. It can deploy and update secrets and application configuration without rebuilding the container images and without exposing these to the configuration stack.

### 4.4 Application Lifecycle Management

The Application Lifecycle Management (ALM) is a process to take care of the specification, design, development and testing of a software application until the application is discontinued. ALM covers the entire lifecycle from the idea conception, through to the development, testing, deployment, supporting and ultimately retirement of systems.

The primary focus of the ALM implemented in this project is to manage the application in the deployment environment and make sure that it is correctly installed and available to the customer.

#### 4.4.1 Helm

In this project, Helm is a package manager for Kubernetes and, in this project, Helm was used as an application lifecycle manager. Helm enables a more efficient way to group Kubernetes resources and makes the deployment process easier. Just as important as this package management, Helm also has other capabilities:

**Application Deployment Management**

Helm helps the management of Kubernetes applications and Helm also keeps track of all actions performed by itself on the Kubernetes applications. Helm can present information such as when the application was installed, when the application was upgraded, the current application’s configuration, and when the application was deleted.

This feature is especially useful in a disaster-recovering situation. At any point in time, the application can be restored using a previous working version and reducing the downtime from the customer perspective.

**Application Template**

Because of the nature of the containerized application, which should contain everything that is required for its operation, many times, a containerized application contains in its code some deployment-specific configuration. This side effect of the containerization is solved by Helm using the concept of a template.
A Helm template is an elegant way to separate what is application specific from deployment specific. This technique partially simplifies the container definition only injecting the deployment-specific parameters during deployment time.

4.5 Continuous Integration

Continuous Integration (CI) is a practice of merging all developers’ working copies periodically. In general, this process happens in an isolated and automated environment. Some benefits of this practice are described below:

- Early feedback: As the integration process happens automatically, it can be scheduled to happen many times a day and each time that the process ends, it indicates if the code is still working;
- Quality assurance: The integration process can include a set of tests (unit, integration, among others) to guarantee the correctness of the code;
- Reduce time to market: The artifacts generated during the integration process have certain level of quality and, in certain cases, can be used to speed up the test phase and by consequence arrive early to a customer.

4.5.1 Jenkins

To select the more appropriate tool for the SDP building process, two different continuous integration tools were compared: Jenkins[7] and Git Lab CI[8]. The final comparative table is available in Appendix B.1.

Because there is no formal policy about which tools should be used to implement the continuous integration process, even though Thermo Fisher Scientific already uses Git Lab CI in its environment, the implementation of the continuous integration process in this project uses Jenkins. This choice was made because the implementation of the process in this specific scenario was smoother using Jenkins than it would be if the Git Lab CI were chosen.

4.6 Configuration Management Tool

Configuring different machines is a time consuming and error-prone task. This problem becomes bigger when considering also the upgrade path. So a large amount of the time is not used to update the software platform, but it is spent figuring out what is the correct path to reach the desired configuration.

The software upgrading can be considered more important than the first installation because upgrades continue to be applied until the environment is no longer in use. The upgrading process requires some study to understand and answer some questions: which are the current versions, what are the previous customizations, and what is the safe path to install the desired version?

To select the more appropriate framework for the SDP scenario, four different configuration management tools were compared: Ansible [9], Chef [10], Puppet [11], and Saltstack [12]. The final comparative table is available in Appendix B.2.
4.6.1 Ansible

After the comparative study and some experiments performed by the SDP team, Ansible was chosen. Some features were more relevant in that decision:

- **Agentless configuration**: Ansible does not require any change in the machines that are going to be configured;  

- **Plugin variety**: Ansible comes out-of-box with a lot of plugins that help in many different configuration tasks;

- **Flexibility to add new capabilities**: Ansible provides three ways to extend its capabilities: creating native plugins, injecting logic in the configuration definition, and executing scripts written in the native language of each system;

- **Easy to learn**: As the language used to define the host configuration is YAML, the same language used to define Kubernetes resources and Helm charts, the whole team already had knowledge of how to write the definitions.

---

1 Assuming that the communication protocol (ssh for Linux and winrm for Windows) is available in the destination machine.
5 Feasibility Analysis

In Chapter 3, the problem that motivated this project was presented. This chapter explains why some decisions have been taken, because the outcome of this project contains technologies that have to be added to the team knowledge and later on these technologies have to be implemented in the customer on-premise environment. Before each decision process, a feasibility study was performed and it aims to answer various questions. Some of the questions that are present in those studies are listed below and two examples of the final table creates during the feasibility study phases are available in Appendices B.1 and B.2.

Concern: Does the team have any expertise in the target tool?

**Importance**: Avoiding reevaluation of a tool and speeding up the evaluation process  
**Responsible for the Answer**: Team  
**Evidence**: Sharing knowledge about the tool

Concern: How difficult is it to learn the target tool?

**Importance**: Learning a new tool requires time and it could delay the progress of the project if the tool had a steep learning curve.  
**Responsible for the Answer**: Team  
**Evidence**: Prototype implementation, Presentation of the faced problems

Concern: What are the licensing options for the tool?

**Importance**: Open-source software\(^1\) has different types of licenses\(^2\) and each of the licenses has a different impact on the final licensing for the product\(^3\).  
**Responsible for the Answer**: Feasibility study  
**Evidence**: Licensing list available for the tool

Concern: How difficult is it to integrate the new tool into the current environment?

**Importance**: The new tool integration process can impact the architecture of the software or the used process.  
**Responsible for the Answer**: Team  
**Evidence**: Prototype;

---

\(^1\)For the DMP, we chose to use off-the-shelf software instead of implementing it internally  
\(^2\)Read more in https://opensource.org/licenses  
\(^3\)For instance if a project uses code under GPL-2 or GPL-3 licenses, this project has to become open-source as well

Designing an infrastructure for On-Premise Software Deployment
6 Use Cases and System Requirements

This chapter introduces the system requirements and use cases of the project. The use cases were elicited using the interviews with the stakeholders. The requirements were extracted from the use cases and later validated by the stakeholders.

6.1 Use cases

Table 6.1 presents the list of use cases approached by this project. These use cases can be divided in: building process Tables C.1 and C.2; Application life cycle Tables C.3, C.4, C.5, and C.6; Configuration management Table C.7.

<table>
<thead>
<tr>
<th>UC1</th>
<th>Release artifact version</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC2</td>
<td>Building Artifact</td>
</tr>
<tr>
<td>UC3</td>
<td>Install application</td>
</tr>
<tr>
<td>UC4</td>
<td>Upgrade application</td>
</tr>
<tr>
<td>UC5</td>
<td>Rollback deployment</td>
</tr>
<tr>
<td>UC6</td>
<td>Delete application</td>
</tr>
<tr>
<td>UC7</td>
<td>Configure host</td>
</tr>
</tbody>
</table>

A more detailed version of each use case listed in Table 6.1 is available in Appendix C. The tables present in that appendix include information such as

- Description of the use case
- Actors involved in the use case
- Frequency of usage
- Triggers
- Pre-conditions
- Basic flow
- Post-conditions
- Exceptions and Alternative flows

16 Designing an infrastructure for On-Premise Software Deployment
6.2 Requirements

The requirements presented in this section were extracted from the use cases described in Section 6.1. Table 6.2 presents the functional requirements developed in this project while Table 6.3 presents the non-functional requirements.

Table 6.2. Functional Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Aspect</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR1</td>
<td>The system must package the installers</td>
<td>Functionality</td>
</tr>
<tr>
<td>FR2</td>
<td>The packaging process must be triggered remotely</td>
<td>Functionality</td>
</tr>
<tr>
<td>FR3</td>
<td>The system could send notification</td>
<td>Functionality</td>
</tr>
<tr>
<td>FR4</td>
<td>The system should publish the installer in a web page</td>
<td>Functionality</td>
</tr>
<tr>
<td>FR5</td>
<td>The system could monitor the availability of the installed systems</td>
<td>Functionality</td>
</tr>
<tr>
<td>FR6</td>
<td>The system should deliver artifacts to the destination host when required</td>
<td>Functionality</td>
</tr>
<tr>
<td>FR7</td>
<td>The deployed environment must keep a local deployment history</td>
<td>Traceability</td>
</tr>
</tbody>
</table>

Table 6.3. Non-Functional Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Aspect</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFR01</td>
<td>The installers must be available to be downloaded at any time</td>
<td>Availability</td>
</tr>
<tr>
<td>NFR02</td>
<td>The installers must be built available in different flavors: nightly and release</td>
<td>Testability</td>
</tr>
<tr>
<td>NFR03</td>
<td>The building server must keep the history of the building processes</td>
<td>Traceability</td>
</tr>
<tr>
<td>NFR04</td>
<td>The building process should be able to happen in different operational systems</td>
<td>Portability</td>
</tr>
<tr>
<td>NFR05</td>
<td>Adding a new builder node to the pipeline should be an easy task</td>
<td>Extensibility</td>
</tr>
<tr>
<td>NFR06</td>
<td>The builder could notify the team if a project cannot be built</td>
<td>Transparency</td>
</tr>
<tr>
<td>NFR07</td>
<td>The building process must be robust enough to accommodate conflicting requirements</td>
<td>Robustness</td>
</tr>
<tr>
<td>NFR08</td>
<td>The building process must be triggered from three different ways: periodically, automated, and manual</td>
<td>Usability</td>
</tr>
<tr>
<td>NFR09</td>
<td>The deployed environment must be able to roll-back in case of updating failure</td>
<td>Fault Tolerance Robustness</td>
</tr>
</tbody>
</table>
7 Solution Architecture

This chapter describes the architecture defined for the Continuous Delivery (CD) pipeline in the context of the SDP project. The CD pipeline is decomposed into three phases: Packaging, Delivery and Deployment, and Monitoring as illustrated in Figure 7.1. Each phase is represented as an independent component and has its own input, behavior, and output. In some cases the output of one phase is used as input to another one.

![Figure 7.1: Continuous Delivery Pipeline Architecture](image)

7.1 Packaging Phase

The packaging phase contains three components: external repositories\(^1\), building server, and publishing server as illustrated in Figure 7.2.

There are two important interfaces exposed in the packaging subsystem, the Building Management API and the Web API. The Building Management API is responsible for exposing the services provided by the building machine, such as triggering a build process remotely, to the internet. The Web API is responsible for the communication between the publishing and destination machines. Table 7.1 presents more information about the packaging phase.

\(^1\)Repositories can store different types of artifacts such as Docker registry stores docker images; Git stores source code and Kubernetes resource definitions, and Helm repository stores helm charts.
Table 7.1. Packaging architecture

<table>
<thead>
<tr>
<th>Goal</th>
<th>Produce the artifacts to be deployed into the DMP platform by the SDP platform.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>The packaging inputs include but are not limited to:</td>
</tr>
<tr>
<td></td>
<td>• Kubernetes resource definitions</td>
</tr>
<tr>
<td></td>
<td>• Helm charts</td>
</tr>
<tr>
<td></td>
<td>• Docker images</td>
</tr>
<tr>
<td></td>
<td>• Operating system packages</td>
</tr>
<tr>
<td>Behaviour</td>
<td>Some initial behaviors presented by the packaging phase are:</td>
</tr>
<tr>
<td></td>
<td>• Code validation</td>
</tr>
<tr>
<td></td>
<td>• Artifact packaging</td>
</tr>
<tr>
<td>Output</td>
<td>SDP installers and building status</td>
</tr>
</tbody>
</table>

7.2 Delivery and Deployment Phase

The Delivery and Deployment phase is responsible for delivering the artifacts generated during the packaging phase to the customer as well as for managing the installation of them. This phase is divided into two steps: the package delivery and application deployment.

7.2.1 Package Delivery

The package delivery step is responsible for the environment where the DMP is installed. It aims to guarantee that all environments have the defined configuration and the correct set of software deployed.
Table 7.2. Package delivery step

<table>
<thead>
<tr>
<th>Goal</th>
<th>Keep the correctness of the DMP environment’s configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Configuration definition</td>
</tr>
<tr>
<td>Behaviour</td>
<td>Reconfigure the DMP environment based on the defined configuration</td>
</tr>
<tr>
<td>Output</td>
<td>Configuration report</td>
</tr>
</tbody>
</table>

7.2.2 Application Deployment

The deployment phase aims to manage the application released to the customer. This management try to avoid or at least reduce any downtime faced by the customers.

Table 7.3. Application Deployment step

<table>
<thead>
<tr>
<th>Goal</th>
<th>Manage the applications installed by the SDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Application’s artifacts and configurations</td>
</tr>
<tr>
<td>Behaviour</td>
<td>Install, Upgrade, and Uninstall applications inside the DMP platform</td>
</tr>
<tr>
<td>Output</td>
<td>Application’s history</td>
</tr>
</tbody>
</table>

7.3 Monitoring phase

The monitoring phase aims to make sure if the deployed infrastructure is available to customers\(^2\). To perform this verification, periodically the system checks if the target servers expose the expected services. The result of this test is processed and published in a web page.

Figure 7.3 presents the interfaces exposed and required by the verification component and Table 7.4 provides further information about the verification process.

\(^2\)The availability in this context only considers if the service is accessible. Performance, validation, and verification of the deployed applications are taken care during the integration tests not covered in this project.

Figure 7.3: Monitoring architecture
Table 7.4. Monitoring architecture

<table>
<thead>
<tr>
<th>Goal</th>
<th>Report the status of the deployed infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Server’s address and services to be monitored</td>
</tr>
<tr>
<td>Behaviour</td>
<td>Check the availability of the deployed servers and publish a periodic report about it</td>
</tr>
<tr>
<td>Output</td>
<td>SDP infrastructure status report</td>
</tr>
</tbody>
</table>
8 Deployment and Implementation

This chapter describes the solution created in this project. To make the solution more understandable, it is decomposed in three parts: packaging, templating, and delivering.

8.1 Packaging

Packaging can be understood as an alternative way to implement continuous integration, which is described in Section 4.5 of this report. Even though the DMP and SDP projects do not have source code to be compiled in order to generate an installer, these projects have a lot of Kubernetes resource definition and Helm charts to be parsed and interpreted. Due to the fact the DMP and SDP installer is composed of different applications, the process of generating their installers requires that the building environment takes care of the following points:

- **Conflicting requirements**: It means that each built package can have its own software and configuration dependencies and these dependencies are not necessarily equal across packages;
- **Flexible builder deployment**: It means that the builder itself should not be coupled to the environment where it is deployed;
- **Ease scaling up**: Changing the number of builders available and where they are deployed must be a concise and simple process;
- **Centralized coordination**: The builders have to report to a central instance at all levels (credential managing, data loading, and exporting);

Keeping the above points as the main drivers of the packaging subsystem, the architecture presented in Figure 8.1 was proposed and implemented.
8.1.1 Continuous Integration Server

The following section provides more details about the central component of the packaging system, the CI Server.

The CI server is considered the central point of the packaging system because it is responsible for tasks such as:

- Artifact publication
- Result centralization
- Node coordination
- Credential management
- Job creation
- Web-Hook exposition

The reason for implementing the CI’s tasks are presented in the following paragraphs.

Artifact publication

The building process generates a lot of files during its execution. Some of these files have to be transferred from the remote builder to the publishing server. Which files have to be transferred, where
the destination is, and when the transferring has to be done are parameters defined inside each building job.

Result Centralization

Due to the way the Docker container works, it is not so suitable to keep data inside it. At the same time, the packaging system works with many nodes that can be available or not anytime. So, considering these constraints, we opted for having a centralized place to store all building results, which is the CI server. This architecture simplifies the backup strategy, allows analysis about building performance, and monitors the building health.

The building result transferring happens independently of the building result. It means that the node sends a building result to the CI server every time that a job was executed even if it was canceled by the user or if the building failed.

Node Coordination

In this project, we opted for a Primary-Secondary architecture (formerly known as Master-Slave) in which the CI server decides on-the-fly which node is going to be used based on the job definition. At the same time, the CI server keeps track of the remote builder’s health information.

Credential management

To simplify the credential administration, all required credentials are stored in the CI server and only when a job starts are the required credentials injected. This approach also simplifies the remote builder that can be developed only focused on the building environment.

Building Process Coordination

Each building process is mapped into the packaging system as a job. A job is a set of steps to build a project. Inside each job are defined many parameters, for instance: which node has to be used, which credentials have to be provided, a version control address and client, and a destination of the generated artifacts.

Building Triggering

All jobs inside the CI server can be started manually. However, there are two special ways of starting a building process: automated building and web-triggered building. The automated building is scheduled to be executed on specific days and times. The web-triggered building uses a resource called web-hook. Web-hook is a web-endpoint exposed in the CI server that receives messages from the version control system and, based on internal rules, it can start the building process.
8.1.2 Remote Builders

The following section provides more details about remote builders. The remote builders are one of the sub-components of the packaging system.

As described in Section 8.1, each artifact-building process has its own requirements and these requirements can be conflicting for different artifact-building processes. To deal with the possible conflicting requirements, each building process is allocated to an isolated environment named as a remote builder.

From the architecture point of view, the remote builders are the secondary ones from the "Primary vs secondary" pattern\(^1\) [13]. The implementation of this pattern basically removes the complexity of dealing with conflicting dependencies from the CI Server (primary) and delegating it to the builders (secondary).

From the implementation point of view, the remote builder is a docker container with some extra capabilities for each instance:

- Building agent
- Container runtime
- Remote connectivity
- Version Control

\(^1\)This design pattern was previously known as a Master Slave pattern, but because of the "slavery" reference, the name was updated to "Primary vs Secondary" or "Primary and Replicas." The first option is used in this report, because the term replica implies the same capabilities are available in both sides, which is not the case in here.

---

25 Designing an infrastructure for On-Premise Software Deployment
Building agent

It is the agent used by the building server to monitor the health of the remote build. This agent shares with the building server information such as building logs, system architecture, clock difference, available storage, available swap space, running builds, and response time.

Container runtime

It grants to the remote builder the ability of running containers inside a container. The ability of running containers inside a container is required because some artifacts generated by the builder are docker images and those images should not be present in the main environment. A self-contained container environment also simplifies the building process due to the fact that whenever the builder container is recreated, any previous downloaded docker image will be deleted, thus not interfering in the next building and increasing the reproducibility of the building process.

Remote connectivity

It allows the remote builder to be deployable anywhere with internet access. This ability is used by the build server to inject into the builder anything required by the building process, for instance: source code, building steps, and user credentials.

Version control

It grants to the building server the ability of changing the source code to a different commit, branch, or tag present in the code repository before building the code. In the future, the builder will be able to push back a new tag to the version control system or approve a merge request automatically based on a successful build.

The base container definition is responsible for installing the basic operating system, copying the public keys for remote access, and configuring the docker environment inside the container. Using the base container definition, each developer can create its own layer that can be used as a specific build environment.

The process used to create the remote builder is implemented using one script, which automatically checks user permissions and access keys, and creates the required port mappings. The process of extending the basic remote builder to add new capabilities was exercised in two situations and it is represented in Figures 8.3 and 8.4.

---

2Part of the process to embed the container runtime is described in Appendix A

Designing an infrastructure for On-Premise Software Deployment
The new capabilities required by the new builders were:

- **Container runtime specific configuration**: As a good practice, the container runtime should use only trusted and public resources. However, for some building processes, some locally-deployed resources are needed and the required changes for allowing their usage are considered as runtime specific configurations.

- **Cluster Administration tooling**: There is a remote builder to package the basic installer for the whole platform. This builder needs to have knowledge about the platform tool versions in order to install the correct toolset.
• **Template Tooling**: This capability enables the remote builder to read and parse the templates before building the installer. So, some part of the building process can be defined dynamically, which makes the building more flexible.

## 8.2 Publishing

As important as packaging the correct artifact is publishing as much information as possible. After each building process, the remote builder sends the building results to the CI server. When the CI server receives the results, they are evaluated and if the building was concluded successfully, some generated artifacts are copied from the remote builder to the publishing server. However, if an error occurs a notification is sent to the team responsible to inform it about errors in the process. This flow is represented in Figure 8.5.

![Figure 8.5: Activities performed from the moment that one building is triggered until the artifacts become available](image)

The publishing system can be decomposed into different steps, which do not necessarily happen in the same machine:

- **Artifact archiving**: Due to the release of a new version of the installer, the previous ones are...
periodically moved to a different server for long-term storage.

- **Artifact delivery**: In general, the artifacts generated by this CD process are large files (some gigabytes). To optimize the file transfer, the publisher is also capable of directly delivering the files that are published on it.

- **Artifact publication**: As the building results and installers must be available anytime, they are published through web pages where they can be easily accessed.

- **Artifact verification**: In order to allow the technicians to verify the integrity of the installers, besides them there is a list with the SHA-256 hashes from all artifacts generated by the continuous integration server.

- **Result Consolidation**: The aim of this process is keeping historical information related to the health of the building process. Based on this historical data, it is possible to extract metrics such as time to generate installers, cause of building failures, quantity of failures, and which step caused the failure. These pieces of information help the administrator to have new insights about the process and possible points for improvements.

8.3 **Delivery**

There are a huge number of software and configurations that have to be addressed in a single piece of hardware. Keeping them synchronized across different environments and rolling out only what is needed is a complicated and error prone task.

To deal with the complexity of managing the software and configuration of the DMP platform, in this project four tools were compared: Ansible [9], Chef [10], Puppet [11], and SaltStack [12]. These tools were compared based on criteria such as strategy used to delivering an artifact to the destination, extra software required in the destination machine, and how easy it was to implement simple tasks and structure the source code. A summarized table of this feasibility study is presented in Appendix B.2.

8.3.1 **Ansible**

At the end of the study phase, all results were presented in a meeting in which the whole team agreed to use Ansible as the official configuration tool for the DMP and SDP projects. Some important characteristics that led to that decision are listed below:

- Ansible is an agentless tool. It means that it does not require the installation of any extra software, to act like a client in the destination machine, to start to work.

- Ansible has a lot of pre-installed plugins in the basic package. It saves time during the configuration of the tool.

- Easy to extend. As Ansible is written in Python and it supports native script execution, it is easy to implement new Ansible plugins to add more features to the server;

- Easy to learn. As the language used to declare a configuration state is YAML, a language used to define Kubernetes resources, the team does not need to learn a completely new language.
8.4 Deployment

There is no limit for the number of applications that can be installed on a computer. However, managing all installed applications and reusing the Kubernetes definitions in an efficient way can become challenging.

Kubernetes resources, by default, have to be completely independent of where the resource is deployed. This independence is reached by the injection of some configuration parameters inside the resource. Many times, these parameters are related only to a specific customer. To solve this problem and separate the deployment-specific definition from the Kubernetes definition, we adopted a tool for creating templates for the Kubernetes resources.

8.4.1 Helm

Helm is a standalone tool that is used as a package manager for Kubernetes. Among other features, Helm helps in the administration of the Kubernetes applications in:

- Splitting deployment-specific content from the Kubernetes definition. It gives more control and flexibility to the DSE, so that does not need to update the Kubernetes definition immediately before installing the application to fix any configurations.
- Managing the Kubernetes application life-cycle. Tasks such as installing, upgrading and rolling back the application are done using Helm using simple commands.
- Cleaning the versioned Kubernetes resource definition. As the Kubernetes resources are template, changes in the values do not necessarily require updates to the template.
- Automating the download of Helm chart dependencies. It means that Helm, on demand, downloads all sub-charts required for a specific deployment. It gives flexibility to divide the development work in different teams.

The Helm tool implemented in this project consists of two parts: Helm, a command line tool for interacting with the Kubernetes cluster environment; and Tiller, the server-side part of Helm to parse the Helm charts and manage the Helm interactions with the Kubernetes cluster. The interaction among Helm, Tiller, and the Kubernetes cluster is illustrated in Figure 8.6. This figure shows the basic flow used to deploy one Kubernetes application through Helm.
8.5 Deployed Solution

To summarize what is described in this chapter, the final proposed solution in this project is an implementation of a Continuous Delivery process which enables Thermo Fisher Scientific to deliver different software to several customers and accommodate customer’s difference at the same time.

The outcome of this project is composed of:

- Continuous integration process and infrastructure
- Remote builder concept and implementation
- Publishing server implementation
- Basic monitoring process
- Template technology implementation
- Configuration Management implementation

It is important to highlight that this technology is implemented in the Thermo Fisher Scientific’s customers’ on-premise.

31 Designing an infrastructure for On-Premise Software Deployment
9 Verification and Validation

This chapter explains the verification and validation processes adopted in this project. These two processes are divided in three phases: Migration from manual to automated, monitoring, and expert validation.

9.1 Migration from manual to automated

As one of the goals of this project is to automate a pre-existing process towards CD practice, the first approach to validate the automated pipeline was the implementation of several tests to make sure that the automated building process has a similar outcome to the manual process. The methodology applied here is similar to the Test-Driven Development (TDD) approach used to develop and refactor source-code.

The implemented tests basically checked if important files for the DMP installer are in the correct folder and in the correct version. Using TDD to verify the output of the pipeline gives more confidence about the proposed process meanwhile it helps to understand the process better and suggest improvement while keeping the same quality.

9.2 Automated Verification

As part of the quality assurance of the Continuous Delivery process some post-installation tests were defined. These test aim to verify if the minimum requirements for the platform were satisfied. These tests are divided in platform specific Sub-section 9.2.1 and application specific Sub-section 9.2.2. To support the implementation of these tests was used a framework called Bash Automated Testing System (BATS) that is available in [15].

9.2.1 Platform specific

The platform specific tests aim to verify if all the basic environment is correctly installed and available to the customer. These test are composed of:

- **Customer connectivity**: Checks if the sample application\(^1\) was correctly deployed.

---

\(^1\)There is a simple Kubernetes application called hello-world that is used to make sure that cluster is able to deploy applications.
• **Service tool availability**: Checks if the service tools\(^2\) are exposed in the correct ports and if the basic configuration of them is done.

• **Files cache**: As, in general, the cluster does not have internet access, many files are cached inside the cluster and exposed to the workers.

• **Internal connectivity**: Checks if all the internal cluster services\(^3\) are accessible from the workers.

• **Specialized node verification**: As some nodes have different capabilities, for instance GPU nodes, some specific checks are required. For the GPU nodes these checks include if the driver is loaded and if it is possible to run a test program.

### 9.2.2 Application specific

The application specific tests cover the deployment of the application’s components inside the cluster itself. The Files cache from the Platform specific test is also used in the application specific. However, the application specific tests can include:

• **Cluster application status**: The main goal of this tests is verify if the application was deployed and if all components are correctly deployed.

• **Backup creation**: Check if some components of the application are properly backed up. Application resources such as secrets, configmaps, and databases have to be exported to help in disaster recovery situation.

• **Customer connectivity**: Checks if the application was correctly deployed and it is exposed to the customer network using the correct DNS name.

### 9.3 Monitoring

A deployed system described in Chapter 8 is composed of several pieces of software that need to be exposed using some predefined configurations. However, due to the number of services exposed in a single installation, a web service was developed to monitor each instance and present the current status of each piece of software in a web page.

### 9.4 Expert Validation

The final step of the verification and validation process happens manually and is performed by a Thermo Fisher Scientific employee. During this final test, in general an FSE or DSE follows a procedure to check if all of the equipment is fully working, if the integration between systems was done, and if the quality attributes were achieved.

\(^2\)The tools considered in this test are: Prometheus, Grafana, and Kibana. They are use to troubleshoot the cluster environment.

\(^3\)File cache, API Server, and Docker internal registry
10 Conclusions

This chapter elaborates on the results achieved by this project. Additionally, it presents recommendations for future work.

10.1 Results

The scope of this project included the parts of the process of building artifacts, artifact delivery, application lifecycle management, and configuration management for the DMP and SDP projects. The developed solution created a Continuous Delivery pipeline in which the DMP’s sub-projects can be built and managed by the SDP platform.

As the first result of this project, the proposed building process was implemented, and the tasks involved in this process were divided between the Continuous Integration server and the remote builders. This division gives the flexibility to extend the building process easily because the management of the building process and the building itself are completely isolated.

Another result is the researching and implementation of tools for the Application Lifecycle Management in the DMP platform. These tools improve the automation of the management of the DMP’s deployment stack at the same time in which they reduce the effort to keep the equipment working and in the correct configuration.

The combination of these results can be considered as an initial implementation of the process of Continuous Delivery for Thermo Fisher Scientific’s equipment.

10.2 Future Work

The Continuous Delivery Architecture and proof of concept defined in this project can be used to implement a CD pipeline to deliver software in any cluster infrastructure based on Kubernetes. However, there are still some changes or extensions that can be performed to them. Some improvement points for future work are listed below.

- The configuration management software utilized in this project was installed in the default configuration and only using the currently stable plugins. Because of this, some tasks were implemented using pieces of the previous developed scripts. Creating plugins for these tasks and after that, an approach to easily generate and manage them could improve the process as well as contribute to the open-source community.

- In the current implementation, the continuous integration server needs to know, beforehand, the
remote builder’s address. It implies that any additional builder needs to be manually added to the server. Finding a mechanism to manage the builders automatically is also another interesting research area.

- Another point also related to the building process is that one builder needs to be capable of building the whole artifact. Applying distributed compilation techniques in order to have smaller builders and speed up the process can be interesting.

- As this first implementation was developed to be used inside Thermo Fisher Scientific’s R&D department, the security aspect of the solution was not further evolved. Evaluating the proposed solution and making sure that all pieces are secure would be an important asset for some customers.

- The Validation and Verification processes were no deeply explored in this project. Find out possible ways to validating and verifying whatever was delivery would be an interesting improvement for the project.

- So far, this project has considered the upgrading scenario from version X to version X + 1. Creating an upgrading path that considers a longer gap or partial upgrading of the system seems to be another improvement point for the project.
11 Project Management

This chapter presents the project management process that was adopted during the final project for the Professional Doctorate in Engineering (PDEng). It elaborates on how the project was managed during the ten-month period. Further it presents the Infrastructure plan, the milestone trend analysis, and the risk analysis.

11.1 Process

The first activity performed in this project was to come up with a project plan. This project was executed in ten months, from January 2019 to October 2019. Figure 11.1 shows the initial version of the planning and the duration of initial project’s milestones carried out during the project. Figure 11.2 presents a detailed version of the milestone trend analysis with all deliverable of this project.

![Figure 11.1: Summarized initial project’s milestones](image)

Figure 11.1: Summarized initial project’s milestones

Designing an infrastructure for On-Premise Software Deployment
11.2 Infrastructure plan

As the project is confidential, all project related artifacts are hosted on local machines available at Thermo Fisher Scientific and all team members only work on company provided computers. The data was not shared or downloaded to any other computing machine.

11.2.1 Communication and Information sharing

- **Cisco Webex**: Application used for online meetings, whiteboarding, and file sharing with the whole team.
- **Microsoft Teams**: Software used to communicate with teammates remotely located as well as for meetings.
- **Microsoft Outlook**: A personal information manager, it also includes a calendar, task manager, and contact manager.
- **SharePoint**: Website used to group, administrate, and share documents inside the company.

11.2.2 Documentation

- **Enterprise architect**: It is used for Architectural deliverable development.
- **Jira**: It is used for implementing SCRUM as our agile method and it is used as a bug tracker. The use cases are defined as stories and epics.
- **Microsoft Office**: Especially Word for producing Document, Excel for calculations, and PowerPoint for presentations.
- **Microsoft Visio**: It is used for developing diagrams related to project.

11.2.3 Development

- **CentOS 7**: It is a Linux distribution that provides a free, enterprise-class, community-supported computing platform functionally.
- **Gitlab**: This web tool is used to implement software version control, code review.
- **VirtualBox**: It is a free and open-source hosted hypervisor for x86 computers. It is used for local deployment and small-scale tests.
- **VMware ESXi**: This tool is an enterprise hypervisor for the Software Delivery Platform. Its main goal is allowing us to deploy different operational systems as well as administrate them in a centralized place.
11.3 Milestone Trend Analysis

Figure 11.2 presents the detailed list of the deliverable of this project and when their scheduled deliver week.
11.4 Risk Analysis

Table 11.1 shows the identified risks for the project as well as their impact, probability of occurrence, and the frequency of each of them are reevaluated during the project. Every risk has associated to it some mitigation strategy and contingency plane. Table 11.2 links the risks presented in Table 11.1.

**Table 11.1. Project’s Risks**

<table>
<thead>
<tr>
<th>Id</th>
<th>Risk event description</th>
<th>Likelihood</th>
<th>Impact</th>
<th>Review frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Technology choice may not suit the customer</td>
<td>Low-Medium</td>
<td>High</td>
<td>Biweekly</td>
</tr>
<tr>
<td>2</td>
<td>Miscommunication / misunderstanding of deliverable</td>
<td>Low-Medium</td>
<td>High</td>
<td>As soon as possible</td>
</tr>
<tr>
<td>3</td>
<td>Infrastructure available to perform automated tests</td>
<td>Medium</td>
<td>Low</td>
<td>Weekly</td>
</tr>
<tr>
<td>4</td>
<td>Stakeholders not available during critical times of the project</td>
<td>Medium</td>
<td>High</td>
<td>As soon as possible</td>
</tr>
<tr>
<td>5</td>
<td>Delay in the progress of the project</td>
<td>Low-Medium</td>
<td>Low-Medium</td>
<td>Weekly</td>
</tr>
<tr>
<td>6</td>
<td>Change plans in the middle of the project</td>
<td>High</td>
<td>Medium</td>
<td>As soon as possible</td>
</tr>
<tr>
<td>7</td>
<td>The performance of the system does not reach the required levels</td>
<td>Low</td>
<td>Medium-High</td>
<td>Weekly and mainly after every delivery</td>
</tr>
<tr>
<td>8</td>
<td>Different equipment can be, by mistake, configured using wrong parameters</td>
<td>Medium</td>
<td>Low</td>
<td>monthly</td>
</tr>
</tbody>
</table>
Table 11.2. Project’s Risk Mitigation and Contingency Strategies

<table>
<thead>
<tr>
<th>Id</th>
<th>Mitigation Strategies</th>
<th>Contingency Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Before going further with any implementation, discuss with the team and collect feedbacks</td>
<td>Work hard in the next available option.</td>
</tr>
<tr>
<td>2</td>
<td>Confirm with the stakeholder anything that we discussed</td>
<td>Negotiate with the stakeholder about the deliverable</td>
</tr>
<tr>
<td>3</td>
<td>Virtualise hardware and software for initial tests</td>
<td>Communicate to the stakeholders that the approach was tested in a different environment because was not possible to replicate the real environment</td>
</tr>
<tr>
<td>4</td>
<td>Plan meetings early. Keep the calendar updated. Confirm availability some days in advance</td>
<td>Reschedule meetings. Update stakeholders in different moments.</td>
</tr>
<tr>
<td>5</td>
<td>Use free software and virtualise any piece of hardware during the waiting time</td>
<td>Update the stakeholder with the current state. Reevaluate the task division.</td>
</tr>
<tr>
<td>6</td>
<td>Analyze any proposed change in the scope of the project and evaluate if I can accommodate it in the remaining project time</td>
<td>Negotiate with the stakeholder about the scope of the change</td>
</tr>
<tr>
<td>7</td>
<td>Evaluate the whole platform after each delivery and compare the current results with the previous ones to see if the platform presents any performance degradation</td>
<td>Find the root cause of the performance degradation and try to optimize or replace it</td>
</tr>
<tr>
<td>8</td>
<td>Use automated test tools to evaluate the performance of the deployed environment. Use automated tools, to make sure that the configuration is correct.</td>
<td>Manually tweak the deployed environment. Analyze the needing of updating the current parameters.</td>
</tr>
</tbody>
</table>
12 Project Retrospective

This chapter concludes this report and reflects on the project based on the author’s perspective. It also brings the lessons learned during the project.

12.1 Reflection and Lessons Learned

The project on which I worked during the last ten months challenged me in several ways.

My project interacts with many other projects. It means that I interacted with different teams, teams located in different places, and using different tools. These interactions were something extremely common during the whole project. All these interactions proved to me that communicating in an effective and efficient way is really important. From this point I derived two rules of thumb:

• **Even though I am responsible for a decision, I have to keep others in the loop even if they are out-of-office**: One of my stakeholders gave me a task that involved a technology choice. Between the beginning of the task and one week after the end, this stakeholder was on holidays and I took the required decisions without asking any feedback. In my mind, it was the right thing to do. However, later, I heard that my initiative/action in going further with that task was appreciated, but next time all related people have to be at least in the loop. After that conversation, I realized that it does not matter if I am responsible for one decision my teammates appreciate when I ask their opinions.

• **Be careful with everyone’s time**: As I said in the previous rule of thumb, big companies have many projects, many interactions, and many meetings. Therefore, choose an efficient communication channel and being careful with someone’s else agenda can make a big difference in the received answer.

Other challenges come from the fact that my project outcome should fit the DMP project. This interaction taught me to pay more attention and forced me to be more organized. From this point, I came up with the following rule of thumb:

• **Quality over quantity**: In general it is easier starting a task and seeing what can be done. However, this approach can lead to wasting time or to developing something wrong. Using an incremental approach to plan, execute, evaluate, and start again can use a bit more time to start the task, but can save much more time in total.
Bibliography


A Self-contained docker environment

A.1 Self-contained docker environment

The following script rewrites the configuration of the internal environment allowing a docker service to be run inside a docker container without linking internal and external docker services. The original project that developed this script is available at [16].

```bash
#!/bin/bash

# Ensure that all nodes in /dev/mapper correspond to mapped devices currently loaded by the device-mapper kernel driver
$DMSETUP=$(dmsetup mknodes)

# First, make sure that cgroups are mounted correctly.
CGROUP=/sys/fs/cgroup
LDG=/dev/log:

if [ -d $CGROUP ]; then
    mkdir $CGROUP
fi

if mountpoint -q $CGROUP; then
    mount -t tmpfs -o uid=0,gid=0,mode=0755 cgroup $CGROUP || {
        echo "Could not make a tmpfs mount. Did you use --privileged?"
        exit 1
    }
fi

if [ -d /sys/kernel/security ] && ! mountpoint -q /sys/kernel/security
    then
        mount -t securityfs none /sys/kernel/security || {
            echo "Could not mount /sys/kernel/security.
            echo "AppArmor detection and --privileged mode might break."
        }
    fi

for SUBSYS in $(cut -d: -f2 /proc/1/cgroup)
do
    [ -d $CGROUP/$SUBSYS ]; then
        mkdir $CGROUP/$SUBSYS
    fi

    mountpoint -q $CGROUP/$SUBSYS; then
        mount -n -t cgroup -o $SUBSYS cgroup $CGROUP/$SUBSYS
    fi

    # The two following sections address a bug which manifests itself
    # by a cryptic "lxc-start: no ns_cgroup option specified" when
    # trying to start containers within container.
    # Trying to start containers within container.
    # The bug seems to appear when the cgroup hierarchies are not
    # mounted on the exact same directories in the host, and in the
    # container.

    # Named, control-less cgroups are mounted with "-o name=foo"
    # (and appear as such under /proc/$ pid/gid/cgroup) but are usually
    # mounted on a directory named "foo" (without the "name=" prefix).
    # Systemd and OpenRC (and possibly others) both create such a
    # cgroup. To avoid the aforementioned bug, we symlink "foo" to
    # "name=foo". This shouldn’t have any adverse effect.
    # Named, control-less cgroups are mounted with "-o name=foo"
    # (and appear as such under /proc/$ pid/gid/cgroup) but are usually
    # mounted on a directory named "foo" (without the "name=" prefix).
    # Systemd and OpenRC (and possibly others) both create such a
    # cgroup. To avoid the aforementioned bug, we symlink "foo" to
    # "name=foo". This shouldn’t have any adverse effect.
    # Named, control-less cgroups are mounted with "-o name=foo"
    echo $SUBSYS | grep -q ^name= && {
        NAME=$(echo $SUBSYS | sed s/^name=//)
        ln -s $SUBSYS $CGROUP/$NAME
    }
    # Likewise, on at least one system, it has been reported that
    # systemd would mount the CPU and CPU accounting controllers
    # (respectively "cpu" and "cpuacct") with "-o cpuacct,cpu"
    # but on a directory called "cpu", "cpuacct" (note the inversion
    # in the order of the groups). This tries to work around it.
    [ $SUBSYS = cpuacct,cpu ] && ln -s $CGROUP/cpu,cpuacct
    done

# Note: as I write those lines, the LXC userland tools cannot setup
```

Designing an infrastructure for On-Premise Software Deployment
# a "sub-container" properly if the "devices" cgroup is not in its
# own hierarchy. Let's detect this and issue a warning.
grep -q :devices: /proc/1/cgroup ||
  echo "WARNING: the 'devices' cgroup should be in its own hierarchy."

if grep -qw devices /proc/1/cgroup ||
  echo "WARNING: it looks like the 'devices' cgroup is not mounted."
then

  # Now, close extraneous file descriptors.
pushd /proc/self/fd >/dev/null

  for FD in *
  do
    case * in
      # Keep stdin/stdout/stderr
      [012])
      ;;
      # Nuke everything else
      *)
        eval exec "$FD&-"
    esac
  done

  popd >/dev/null

# If a pidfile is still around (for example after a container restart),
# delete it so that docker can start.
rm -rf /var/run/docker.pid

# If we were given a PORT environment variable, start as a simple daemon;
# otherwise, spawn a shell as well
if [ "$PORT" ]
then
  exec dockerd -H 0.0.0.0:$PORT -H unix:///var/run/docker.sock 
    $DOCKER_DAEMON_ARGS

else
  if [ "$LOG" == 'file' ]
  then
    docker $DOCKER_DAEMON_ARGS &>/var/log/docker.log &
  else
    docker $DOCKER_DAEMON_ARGS &

  fi
  ( 
    ( timeout = 60 + SECONDS )
    until docker info >/dev/null 2>&1
    do
      if ( (SECONDS >= timeout ) )
        then
          echo 'Timed out trying to connect to internal docker host.' >&2
          break
      fi
      sleep 1
    done
    [( $1 ] &4 exec "$@"
  exec bash --login
fi

47 Designing an infrastructure for On-Premise Software Deployment
B Comparative Studies

This appendix contains the summarized tables used as one of the bases to decide if the tools fit the project or not. The criteria given in them were defined with the main stakeholders and during meeting with the SDP Team.

B.1 Continuous Integration Tool

<table>
<thead>
<tr>
<th></th>
<th>Gitlab CI</th>
<th>Jenkins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Licensing type</td>
<td>Proprietary, MIT</td>
<td>Creative Commons, MIT</td>
</tr>
<tr>
<td>Written in</td>
<td>Ruby and Go</td>
<td>Java</td>
</tr>
<tr>
<td>Language used to define jobs</td>
<td>YAML</td>
<td>Bash</td>
</tr>
<tr>
<td>Easily integrable to the available infrastructure</td>
<td>Using plugin</td>
<td>Direct copy</td>
</tr>
<tr>
<td>Cost to deploy locally</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Support by</td>
<td>Community, Paid plans</td>
<td>Community</td>
</tr>
<tr>
<td>Integration with Jira</td>
<td>Service</td>
<td>Plugin</td>
</tr>
<tr>
<td>Integration with Gitlab</td>
<td>Native</td>
<td>Doable</td>
</tr>
<tr>
<td>Paid version with more features</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Team experience</td>
<td>OK</td>
<td>++</td>
</tr>
</tbody>
</table>
## B.2 Configuration Management Tools

**Table B.2.** Configuration Management Tool Comparison criteria

<table>
<thead>
<tr>
<th>Feature</th>
<th>Ansible</th>
<th>Chef</th>
<th>Puppet</th>
<th>Saltstack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push or pull</td>
<td>Push</td>
<td>Pull</td>
<td>Pull</td>
<td>Push and/or Pull</td>
</tr>
<tr>
<td>Written in language</td>
<td>Python</td>
<td>Ruby, Erlang</td>
<td>C++, Clojure</td>
<td>Python</td>
</tr>
<tr>
<td>Configuration language</td>
<td>YAML Python</td>
<td>DSL, Ruby</td>
<td>DSL, Json</td>
<td>DSL, YAML, Python</td>
</tr>
<tr>
<td>Client required</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes¹</td>
</tr>
<tr>
<td>Communication protocol</td>
<td>SSH</td>
<td>HTTPS</td>
<td>HTTPS</td>
<td>SSH, ZeroMQ</td>
</tr>
<tr>
<td>Orchestration support</td>
<td>Yes</td>
<td>No²</td>
<td>No²</td>
<td>Yes</td>
</tr>
<tr>
<td>Windows Interaction using</td>
<td>WinRM</td>
<td>chef-client</td>
<td>Puppet Agent</td>
<td>Salt-minion Agent</td>
</tr>
<tr>
<td>Requires FQDN / DNS</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Requirements</td>
<td>NTP, MTA</td>
<td>NTP</td>
<td>ZeroMQ or RAET</td>
<td></td>
</tr>
<tr>
<td>License</td>
<td>GPL3 Proprietary</td>
<td>Apache 2.0</td>
<td>Apache 2.0</td>
<td>Apache 2.0 Proprietary</td>
</tr>
<tr>
<td>Cross platform</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Idempotent</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>VMware support</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes³</td>
</tr>
</tbody>
</table>

---

1It is possible to use agentless SSH
2It needs a separate tool (e.g. MCollective or Foreman) to achieve orchestration
3With an extra python module

49 Designing an infrastructure for On-Premise Software Deployment
C Detailed use cases

This appendix presents a detailed version of the use cases approached in this project. The content presented here is related to the Chapter 6.
Table C.1. Use Case 01: Release artifact version

<table>
<thead>
<tr>
<th>Identifier</th>
<th>UC1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Release artifact version</td>
</tr>
<tr>
<td>Description</td>
<td>It consists of packaging a specific version of an installer as a milestone for one project</td>
</tr>
<tr>
<td>Actors</td>
<td>Developer, Building Server, Remote Builder</td>
</tr>
<tr>
<td>Frequency of Use</td>
<td>Around six weeks</td>
</tr>
<tr>
<td>Triggers</td>
<td>Creation of a version in the source code repository</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The building process is the stable state</td>
</tr>
</tbody>
</table>

**Basic Flow**

1. The developer creates a tag inside the version control system;
2. The version control notify the building server;
3. The building server download source code into the remote builder;
4. The building server changes source code to the specific tag (created in step 1);
5. Remote builder follow the build steps defined by the building server;
6. The generated installer is renamed and published by the building server into the publisher;
7. The building logs are transferred to the building server.

**Post-conditions**

- One new installer is available in the publishing server;
- The Software Delivery Platform (SDP) Team is notified;
- The building server updates the building status\(^1\) process;
- The remote builder is wiped-out.

**Exception / Alternative flow**

UC1.1 Building is not concluded successfully

- Building process is stopped;
- Nothing is sent to the publishing server;
- The building server updates the building status\(^2\);
- The Software Delivery Platform (SDP) Team is notified;

---

\(^1\)The building server keep tracking of the results of the building process such as: last success, last failure, spent time, and so on

\(^2\)The building status can assume the values unstable or failed

51 Designing an infrastructure for On-Premise Software Deployment
**Table C.2. Use Case 02: Building Artifact**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>UC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Building artifact</td>
</tr>
<tr>
<td>Description</td>
<td>It consists of periodically package an installer for one project</td>
</tr>
<tr>
<td>Actors</td>
<td>Building Server, Remote Builder</td>
</tr>
<tr>
<td>Frequency of Use</td>
<td>Daily in working days</td>
</tr>
<tr>
<td>Triggers</td>
<td>Automatically triggered</td>
</tr>
<tr>
<td>Preconditions</td>
<td>-</td>
</tr>
</tbody>
</table>

**Basic Flow**

1. At a specific time, the building server download the project’s source code into the remote builder;
2. Remote builder follow the build steps defined by the building server;
3. The generated installer is published by the building server into the publisher;
4. The building logs are transferred to the building server.

**Post-conditions**

- One new installer is available in the publishing server;
- The building server updates the building status process;
- The remote builder is wiped-out.

**Exception / Alternative flow**

**UC2.1 Building is concluded with issues**

- The building is classified as unstable;
- The installer is published;
- The Software Delivery Platform (SDP) Team is notified;

**UC2.2 Building is not concluded**

- The building is classified as failed;
- Nothing is published;
- The Software Delivery Platform (SDP) Team is notified;
<table>
<thead>
<tr>
<th>Identifier</th>
<th>UC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Install an application</td>
</tr>
<tr>
<td>Description</td>
<td>Install an application</td>
</tr>
<tr>
<td>Actors</td>
<td>Digital Service Engineer (DSE)</td>
</tr>
<tr>
<td>Frequency of Use</td>
<td>-</td>
</tr>
<tr>
<td>Triggers</td>
<td>Manually</td>
</tr>
<tr>
<td>Preconditions</td>
<td>Customer survey is answered</td>
</tr>
</tbody>
</table>

**Basic Flow**

1. Write down the customer’s provided information into the configuration file;
2. Deploy the required files into the nodes;
3. Apply the customer’s provided configuration;

**Post-conditions**

Application is available to the customer.

**Exception / Alternative flow**

Table C.4. Use Case 04: Upgrade application

<table>
<thead>
<tr>
<th>Identifier</th>
<th>UC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Upgrade an application</td>
</tr>
<tr>
<td>Description</td>
<td>Install a new version of an application</td>
</tr>
<tr>
<td>Actors</td>
<td>Digital Service Engineer (DSE)</td>
</tr>
<tr>
<td>Frequency of Use</td>
<td>-</td>
</tr>
<tr>
<td>Triggers</td>
<td>-</td>
</tr>
<tr>
<td>Preconditions</td>
<td>UC3</td>
</tr>
</tbody>
</table>

**Basic Flow**

1. Write down the customer’s provided information into the configuration file;
2. Deploy the required files into the nodes;
3. Apply the customer’s provided configuration;

**Post-conditions**

A new version of the application is available to the customer.

**Exception / Alternative flow**

In case of failure the upgrading process, go to UC5
Table C.5. Use Case 05: Rollback deployment

<table>
<thead>
<tr>
<th>Identifier</th>
<th>UC5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Rollback deployment version</td>
</tr>
<tr>
<td>Description</td>
<td>Revert the deployment to a previous working version of the application</td>
</tr>
<tr>
<td>Actors</td>
<td>Digital Service Engineer (DSE)</td>
</tr>
<tr>
<td>Frequency of Use</td>
<td>-</td>
</tr>
<tr>
<td>Triggers</td>
<td>Manually</td>
</tr>
<tr>
<td>Preconditions</td>
<td>Failure in applying the UC4</td>
</tr>
</tbody>
</table>

**Basic Flow**

1. Figure out what was the previous working version based on the deployment history;
2. Revert the upgrading process.

**Post-conditions**
The previous version of the application is available to the customer.

**Exception / Alternative flow**

Table C.6. Use Case 06: Delete application

<table>
<thead>
<tr>
<th>Identifier</th>
<th>UC6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Delete application</td>
</tr>
<tr>
<td>Description</td>
<td>Removes on application from the environment</td>
</tr>
<tr>
<td>Actors</td>
<td>Digital Service Engineer (DSE)</td>
</tr>
<tr>
<td>Frequency of Use</td>
<td>-</td>
</tr>
<tr>
<td>Triggers</td>
<td>Customer’s request, replacement of a no longer supported application</td>
</tr>
<tr>
<td>Preconditions</td>
<td>UC3</td>
</tr>
</tbody>
</table>

**Basic Flow**

1. The application is uninstalled;
2. The deployed components are removed\(^3\).

**Post-conditions**
The application is no longer available to the customer

**Exception / Alternative flow**
For replacing an application uses the UC3
Table C.7. Use Case 07: Configure host

<table>
<thead>
<tr>
<th>Identifier</th>
<th>UC7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Configure a host</td>
</tr>
<tr>
<td>Description</td>
<td>Prepare the host to be deployed in the customer environment</td>
</tr>
<tr>
<td>Actors</td>
<td>Software Delivery Platform (SDP) Team and Digital Service Engineer (DSE)</td>
</tr>
<tr>
<td>Frequency of Use</td>
<td>Manually</td>
</tr>
<tr>
<td>Preconditions</td>
<td>Host has networking connectivity</td>
</tr>
</tbody>
</table>

**Basic Flow**

1. The basic configuration of the machine is done by the Digital Science Engineer to enable remote connectivity;

2. Digital Science Engineer run the configuration management tool pointing to towards the host.

**Post-conditions**
The host is ready to receive the applications

**Exception / Alternative flow**

55 Designing an infrastructure for On-Premise Software Deployment
Giovanni de Almeida Calheiros received his Bachelor’s degree in Information Technology in 2005 from Federal Institute of Alagoas. After his Bachelor’s he worked in several roles such as Software Developer, Network Administrator, as well as Database and Data Warehouse Administrator.

In 2010, Giovanni started his Master’s in Computer Science at Federal University of Campina Grande (Brazil). During his Master’s, he started working as a researcher in a project to develop a distributed and scalable system on the Internet to be used on Smart TVs for brands such as AOC and Philips. Later, this research project became a job offer that started as a Software Engineer and later evolved to Configuration Manager for the two Brazilian sites of the company.

In 2016, Giovanni started working as part of the Digital Security team at Underwriters Laboratories (UL) as a Software Engineer. Later that year, He applied for a Professional Doctorate in Engineering position at TU/e and concluded his final project at Thermo Fisher Scientific.

The Giovanni’s project at Thermo Fisher Scientific challenged him to use state-of-art technologies to adapt cloud principles to be implemented in customers’ on-premise environment.