EngD THESIS REPORT

A Solution For Configuring An Infrastructure-as-a-Service

Komang Aditya Respa Putra
October/2022
Department of Mathematics & Computer Science

EngD SOFTWARE TECHNOLOGY

EINDHOVEN UNIVERSITY OF TECHNOLOGY

A Solution for Configuring an Infrastructure-as-a-Service

Komang Aditya Respa Putra

October 2022

Eindhoven University of Technology
Stan Ackermans Instituite – Software Technology

EngD Report: 2022/076

Confidentiality Status:
Not Confidential

Partners

Steering Group	ir. H.T.G. Weffers, EngD ir. E. Algra, PDEng
Date	October 2022

Composition of the Thesis Evaluation Committee:

Chair:	Dr. T. Ozcelebi
Members	ir. H.T.G. Weffers, EngD
	ir. E. Algra, PDEng
	T. Molina, MSc

The design that is described in this report has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct.

Contact Address	Eindhoven University of Technology Department of Mathematics and Computer Science MF 5.072, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands +31 402743908
Partnership	This project was supported by Eindhoven University of Technology and ThermoFisher Scientific.
Published by	Eindhoven University of Technology Stan Ackermans Institiute
EngD-report	2022/076
Preferred reference	A solution for configuring an Infrastructure-as-a-Service. Eindhoven University of Technology, EngD Report 2022/076, October 2022
Abstract	ThermoFisher Scientific is well-known as a high-tech Microscope producer with numerous customers around the world. One of the high-end Microscopes is Electron Microscope (EM). To enable the advanced feature of the EM, ThermoFisher Scientific built an Infrastructure-as-a-Service solution for its customers called Software Delivery Platform (SDP). Configuring this solution brought additional challenges to the ThermoFisher Scientific engineers due to the highly configurable infrastructure that is depending on its customer needs. This report describes a project to design and implement a solution for configuring the Infrastructure-as-a-Service called SDP Configuration Application (SCA). The report elaborates on the analysis of the problem and the domain to understand the context of the project. It also describes the system's architecture, design, and implementation. In addition, the project management and risk analysis are explained. The system is implemented and validated by the relevant stakeholders to make sure that the proposed solution brings an added value in the context of the SDP configuration.
Keywords	EngD, Software Technology, Infrastructure-as-a-Service, Infrastructure-as-code, Configuration, Graphical User Interface, Application Development
Disclaimer Endorsement	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 ThermoFisher Scientific. The views and opinions of authors expressed herein do not necessarily state or reflect those of the Eindhoven University of Technology or ThermoFisher Scientific and shall not be used for advertising or product endorsement purposes.

Foreword

The MSD business of ThermoFisher 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 mainly 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, we need 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. Our Software Delivery Platform (SDP) infrastructure services the needs of the on-premise software as a service delivery with local tools and automation. The current configuration management interface of this infrastructure is limited in its usage: it is user-unfriendly and requires an IT skillset, whereas easy-to-use, intuitive and foolproof solution is needed that can be used by lesser trained IT service engineers also.

Respa has done a great job in filling this gap. He mastered the current state of the SDP configuration mechanism quickly and familiarized himself with the functions and limitations of the current system. He defined, designed, and implemented a nice GUI-based application that meets the new demands.

His solid structured way of working in all parts of the project (requirements definition, architecture and design choices, implementation, verification, and validation) contributed to the very usable result delivered in a timely manner, whilst making friends in the organization along the way.

It was my pleasure to coach Respa in his project, and I enjoyed the weekly moments of reflection and feedback. Looking forward to a continued working relationship as Respa accepted a permanent position within the SDP team.

Project Mentor

Egbert Algra, MSc, PDEng
October 2022

Preface

The Engineering Doctorate (EngD)[1] in Software Technology (ST) is a two-year traineeship post-master program at Eindhoven University of Technology. EngD focuses on a technological designer to prepare the trainees to become proficient in a software engineering field by participating in several workshops, such as software architecture, system design, and project management. In the second year, several projects are proposed and executed by a trainee in ten months.

This document is a technical report of *A Solution for Configuring an Infrastructure-as-a-Service* project executed by Komang Aditya Respa Putra as part of the ten-month EngD in Software Technology graduation project. The purpose of the project was to design and implement a solution that bridges the gap between a highly configurable infrastructure with an easy-to-use application that provides the overview, transparency, and guided configuration management to the lesser experienced Service Engineers in ThermoFisher Scientific.

The content of the report covers the analysis of the problem and the domain technology, elaborates on the system architecture and design, explains the implementation, presents the verification and validation strategy, and summarizes the outcome of the project.

Komang Aditya Respa Putra
October 2022

[1] The former name is Professional Doctorate in Engineering (PDEng)

Acknowledgements

EngD is a two-year post-master program that gave me a great opportunity to grow as a software designer. In the last ten months, I had carried out a challenging yet enjoyable project in the R&D group at ThermoFisher Scientific.

I would like to express my sincere gratitude to my project supervisor Egbert Algra on behalf of ThermoFisher Scientific and Harold Weffers on behalf of the Eindhoven University of Technology, who supported and guided me throughout the project. The knowledge I learned in terms of project management, software architecture, and software design during the project would be a great foundation for me to grow my career in the coming years.

Thanks to the SDP team who worked closely with me, Giovanni de Almeida Calheiros, Gang Chen, Tarkan Akcay, Tor Halsan, Olufemi S. Adeyemi, and other people that I might forget to mention for their effort in providing me with very important information so that I could understand the domain. Also, for always providing answers to my questions that were essential to tackle the challenges I faced.

During the project, we collaborated with colleagues from the Service Organisation. I would like to thank Jordy Plug, Kieran Ham, Gabriel Lee, and Ataur Rahman for their time and effort in supporting this project. Also, for the valuable feedback and suggestion they provided during the project.

I also would like to thank Yanja Dajsuren and Désirée van Oorschot for their guidance in the past two years during the EngD Software Technology (ST) program. Also, thank to my colleagues from the EngD ST 2020 generation for the fun and exciting moment during the workshops and training projects we worked on together during the program.

Finally, I extend gratitude to my beloved wife, Renisa Suryahadikusumah, for her love and support. Also, deep gratitude to my parents I Nyoman Sukerta and Sri Wachjuni, and my family in Indonesia for their generous support.

Komang Aditya Respa Putra
October 2022

Executive Summary

ThermoFisher Scientific is the major company in serving science. ThermoFisher Scientific products, such as the Electron Microscope (EM) enable various research in many different domains. During the COVID-19 pandemic, ThermoFisher Scientific's products, such as the EMs, had a critical role in supporting the scientists in analyzing the virus and leading to vaccine discovery.

The EMs can magnify the sample at a nanoscopic scale into a high-resolution image. During the process, it creates a large amount of data that require powerful computing resource and needs a massive storage capacity to store the generated data. To solve the issue, ThermoFisher Scientific built an IT solution called Data Management Platform (DMP). As part of the DMP solution, ThermoFisher Scientific developed the Software Delivery Platform (SDP), which is an Infrastructure-as-a-Service (IaaS) to provide an environment that hosts the application suites supporting the EM running seamlessly. Currently, SDP is configured using a set of scripts built in a Command Line Interface (CLI) application to guide the engineers during the installation process. However, due to the limitation of a CLI-based application, interacting with the SDP configurator was considered challenging. The project aims to design and implement a visual interface for the SDP configurator, so that opens the opportunity for the lesser experienced engineer to interact with the SDP configurator more easily.

During the project, we designed and implemented the SDP Configurator Application (SCA) to provide an easy-to-use guided step with a Graphical User Interface (GUI) to help the Service Engineer. We built SCA as a web-based application. SCA's main features are setting up an initial configuration, updating the existing configuration, visualizing an existing configuration, and providing an easy-to-use deployment template for typical configuration types. Additionally, we developed the configurator GUI in a wizard step that could guide the user and increase the usability of the configurator. The infrastructure visualization page provides information on how the current infrastructure is configured. It helps the users understand the current setting when updating the configuration.

The project fulfilled the gap between the highly configurable infrastructure and an application that provides an overview, validation mechanism, transparency, and an easy-to-use GUI. It opened the opportunity for the lesser experienced engineers to interact with the configuration management tool.

For future work, we recommend integrating the SCA with the current SDP installation workflow by migrating the process from using a CLI-based application into the SCA. In addition, features such as monitoring the execution process of applying the configuration to the infrastructure can also be part of SCA.

Table of Contents

List of Figures

List of Tables

Abbreviations

TU/e	Eindhoven University of Technology
EngD	Engineering Doctorate
PDEng	Professional Doctorate in Engineering (former name of EngD)
ST	Software Technology
DSE	Digital Service Engineer
FSE	Field Service Engineer
SE	Service Engineer
GTS	Global Technical Support
DMP	Data Management Platform
SDP	Software Delivery Platform
IaaS	Infrastructure-as-a-Service
SCA	SDP Configurator Application
EM	Electron Microscope
TEM	Transmission Electron Microscope
SEM	Scanning Electron Microscope
CLI	Command Line Interface
GUI	Graphical User Interface
MTA	Milestone Trend Analysis
WBS	Work Breakdown Structure

Glossary

Gitlab CI/CD	A tool for software development using the continuous methodologies
Ansible	An open-source IT automation tool that automates infrastructure configuration
VMWare ESXi	An enterprise-class, type-1 hypervisor developed by VMware
VMNIC	A real physical interface on an ESXi host

1. Introduction

1.1. Context

ThermoFisher Scientific is well-known as the world leader in serving science. The mission is to make the world healthier, cleaner, and safer. In ThermoFisher Scientific, there are various products and services that enable their customers to push science and technology a step beyond [1].

One of the prestigious products of ThermoFisher Scientific that enable its customer to explore many different domains of research is the Electron Microscope (EM). ThermoFisher Scientific produces several variants of EMs, for instance, Transmission Electron Microscope (TEM), Scanning Electron Microscope (SEM), and Cryogenic Electron Microscope (Cryo-EM) [2]. These types of EM are widely used in life science, material science, and semiconductor market segments.

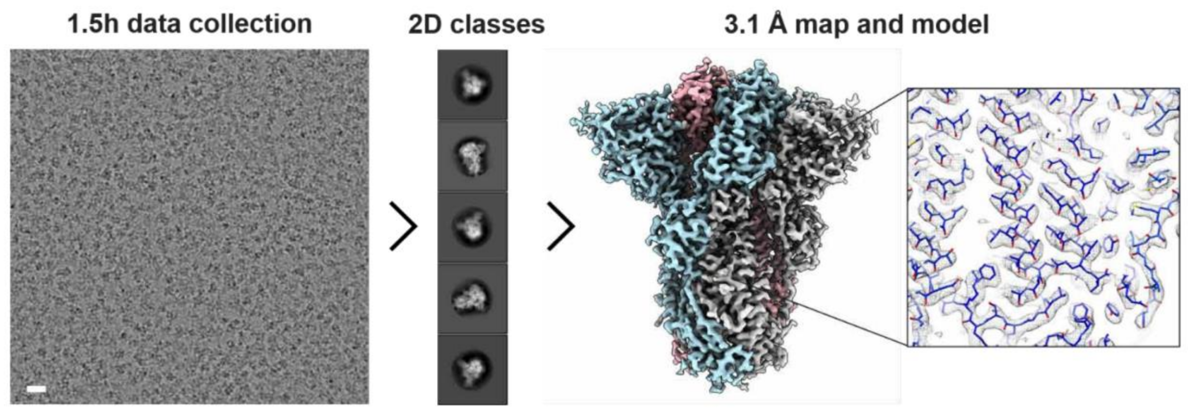


Figure 1. SARS-CoV-2 virus magnified using ThermoFisher Scientific Cryo-EM **[3]**

During the COVID-19 pandemic, scientists used Electron Microscope to analyze the SARS-CoV-2 virus spike structure. The spike structure information is essential to speed up the decision-making in selecting antibodies that eventually can help in drug discovery pipelines. Figure 1 shows the magnification of the SARS-CoV-2 virus using Cryo-EM that produced by ThermoFisher Scientific.

The EMs can magnify the sample up to 1/10,000,000,000 of a meter resolution. This operation creates a large amount of data due to rendering such a high-resolution image. The data is then processed and converted into images that scientists can analyze. This process requires a considerable computing resource and massive storage capacity. To facilitate this scenario, ThermoFisher Scientific developed an IT solution called Data Management Platform (DMP). DMP enables ThermoFisher Scientific's customers to operate the EMs seamlessly by providing the required resources including high-capacity storage and high-speed computing power.

ThermoFisher Scientific aims to provide a seamless experience to its customer while using the EMs. ThermoFisher Scientific achieved this by providing various application software suites based on the type of EM its customer used. To provide an environment for hosting the application suites in the IT solution or DMP, ThermoFisher Scientific built the Software Delivery Platform (SDP). SDP is an Infrastructure-as-a-Service (IaaS) [4] that offers the capability to scale up and down the infrastructure resources, for instance, GPU, CPU, RAM, and storage. SDP performs tasks from the very beginning of configuring the platform to installing service tools, orchestrating the services, and monitoring the resources. Currently, SDP is deployed using a set of scripts built in a Command Line Interface (CLI) application to guide the engineers during the installation process. Due to the absence of a graphical interface, interacting with the SDP tool is less interactive and error-prone since it does not have a validation mechanism.

This project is entitled “A solution for configuring an Infrastructure-as-a-Service.” It aims to extend the capabilities of the current SDP installer by providing an overview, transparency, and guided configuration management that brings a possibility for lesser experienced engineers to interact with the SDP configuration. This system is called SDP Configurator Application (SCA).

1.2. Outline

Chapter 2 describes the stakeholders, the tools used for communication, and the frequency of the communication with the stakeholders. Chapter 3 provides information about the problems in more detail and the domain analysis to understand the current situation. Chapter 4 is the requirement elicitation; the functional and non-functional requirements are described in this chapter. Chapter 5 illustrates the SCA software architecture and design. Chapter 6 explains the implementation of SCA. Chapter 7 describes the verification and validation. Chapter 0 gives an overview of how this project was managed. Finally, in Chapter 0 the conclusion including the summary and future works are mentioned.

2. Stakeholder Analysis

Stakeholder analysis is a process to identify the stakeholders and their interests in the project. Based on the interview with the stakeholders in ThermoFisher Scientific, we identified that the potential user of the SCA was the Field Service Engineer (FSE). FSE team was part of the Service Organization. FSE was responsible for configuring the EM's component and the hardware-related setup. However, FSE was not reachable by us. Therefore, the FSE was represented by the Service Organization in general. We contacted four people from this team as the representative of the potential users listed in Table 1.

The SCA was intended to extend the capability of the current SDP installer. As a result, the project activity was mainly conducted within the SDP team. The team also acted as the domain expert who provided the domain knowledge about SDP and the necessary information during the project.

To summarize, there are two groups of ThermoFisher Scientific stakeholders: the SDP team and the Service Organization. The SDP team was most interested in the technology investigation and the product solution that should be easy to maintain, extend, and integrate into the SDP release cycle. On the other hand, the Service Organization team is concerned about the prototype that should be easy to use with a GUI so that broaden the opportunity for the lesser experienced Service Engineer to be able to configure the SDP.

Table 1. Stakeholders in ThermoFisher Scientific

Name	Role in the project	Interest	Communication channel	Frequency
Egbert Algra	• Company supervisor • Architect in the SDP team	• Architecture • Technology investigation • Product solution • Final presentation/demo	• In person • MS Teams	• Weekly • Monthly
Giovanni D.A Calheiros	Domain expert of the SDP team	• Software design • Technology investigation • Product solution • Final presentation/demo	• In person • MS Teams	• Daily • Monthly • Ad-hoc
Tor Halsan	Domain expert of the SDP team	• Software design • Product solution • Final presentation/demo	MS Teams	Monthly
Gang Chen	Domain expert of the system integration and automation of the SDP team	• System integration • Final presentation/demo	MS Teams	Monthly
Tarkan Akcay	• Domain expert of the SDP team • SDP team leader	• Product solution • Final presentation/demo	MS Teams	• Monthly • Ad-hoc
Jordy Plug	• Digital Service Engineer (DSE) • SCA's potential user	• Product solution • Final presentation/demo	MS Teams	• Monthly • Ad-hoc
Kieran Ham	• Global Technical Support (GTS)			

Name	Role in the project	Interest	Communication channel	Frequency
	• SCA’s potential user			
Ataur Rahman	• Service Operation • SCA’s potential user			
Gabriel Lee	• Digital Service Engineer (DSE) • SCA’s potential user			

3. Problem Analysis

This chapter provides information on the problem analysis and domain analysis of this project. Section 3.1 describes the background of the problem in detail. Next, Section 3.2 shows the problem statement we addressed in this project. Section 3.3 explains the scope of the project. Finally, Section 3.4 elaborates on the domain analysis process to get an overview of the current situation and technology used within the SDP team.

3.1. Background

DMP which was built by the ThermoFisher Scientific R&D team provides the required resources to support the EM's workflow applications running seamlessly. As part of the DMP solution, an SDP was built containing multiple layers of technology to host the applications. Figure 2 depicts the relation between the EM, DMP, and SDP. When ThermoFisher Scientific's customers purchase an EM, it will equip with a DMP. Then, the SDP is configured on the DMP to provide the environment to run software and application suites. We discuss the detail of SDP later in this chapter.

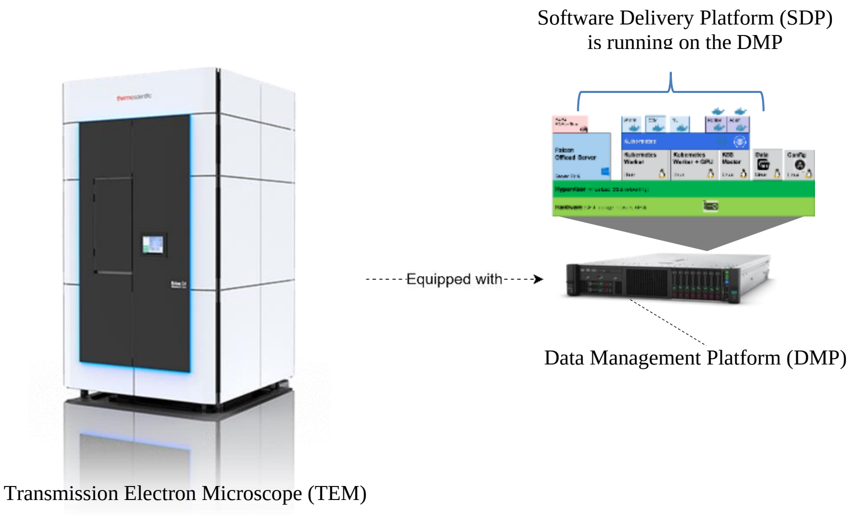

Software Delivery Platform (SDP)
is running on the DMP
Equipped with
Data Management Platform (DMP)
Transmission Electron Microscope (TEM)

Figure 2. Relationship of the EM, DMP, and SDP.

ThermoFisher Scientific's engineers from the Service Organization are responsible to configures the EM's components and the DMP on the customer site. There are two types of Service Engineer to handle this task: Field Service Engineer (FSE) and Digital Service Engineer (DSE). FSE's main task is configuring the hardware-related setup such as the EM's component and DMP server box including its basic network configuration. Once the DMP is configured and connected to the network, then the SDP installation process is carried out by the DSE. DSEs typically support through a remote VPN connection. However, in some customers, such as the semiconductor industry, direct inbound and outbound connections are prohibited. As a result, the DMP is completely in an air-gapped situation. Figure 3 illustrates the scope of the FSE and DSE when configuring DMP and SDP on the semiconductor customer site.

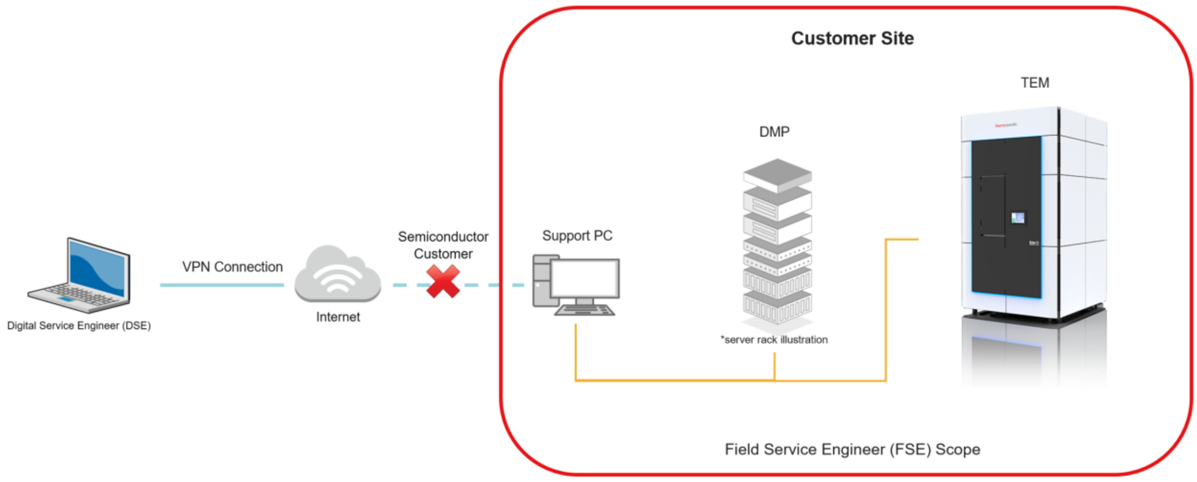

Customer Site
TEM
DMP
Semiconductor Customer
Support PC
VPN Connection
Digital Service Engineer (DSE)
Internet
*server rack illustration
Field Service Engineer (FSE) Scope

Figure 3. DSE and FSE scope in configuring DMP and SDP

In the situation illustrated in Figure 3, currently, there are two options to tackle this scenario:

1. DSEs visit the customer site to configure the SDP on-site. This increased the service cost due to the travel and accommodation expenses.
2. DSEs support the FSEs through a phone call to guide them in configuring the SDP. This approach was considered inconvenient since it was risky of mistakes and could raise errors.

SDP is configured using a CLI-based application. CLI is a text interface that is navigated by typing commands at prompts, instead of using the mouse through the visual interface. A CLI only uses the keyboard to navigate and perform actions. It does not have a GUI for the user to interact with like a typical desktop-based application [5].

ThermoFisher Scientific offers a managed service for its EM solutions to support ThermoFisher Scientific's customers conducting their research. After delivering and configuring the EM and DMP on its customer site, the process is not considered complete. Instead, ThermoFisher Scientific continuously support its customer by providing a regular update, for instance, the new version of SDP is released every three months.

From ThermoFisher Scientific's customer perspective, it is also possible for them to request support to reconfigure the SDP as needed. The common update such as changing the IP address or configuring additional storage that can be attached to the DMP server box as needed. The update is usually handled by the DSE. However, similar to the situation illustrated earlier in the semiconductor customer, the FSE could also be responsible for this task.

In summary, configuring such a complex on-premises cloud infrastructure has its own challenges. The FSE who are responsible for configuring the hardware-related component of the EMs and the DMP were less experienced to work with CLI. Therefore, they need to communicate with the DSEs, who supports them remotely. However, the current approach is considered inefficient because it could cause delays since there are multiple escalations in supporting the customer.

In the long term, as the number of ThermoFisher Scientific's customers grows, the transition from a CLI-based into a visual-based configurator could also open the opportunity for lesser experienced Service Engineers to interact with the SDP configuration. As a result, the services offered by ThermoFisher Scientific to its customer become more reliable and faster.

3.2. Problem Statements

Based on the background information we concluded that interacting with the SDP configuration is technically too difficult for the lesser experienced Service Engineer. The features that could guide the Service Engineers to prevent mistakes are missing, for instance, the form validation, wizard form navigation, and overview of the configuration result. In addition, extending the configuration with additional configuration items is considerably challenging. The R&D engineers need to develop a script to modify the Ansible inventory file to update or add the configuration items, which is considered inconvenient by the stakeholders.

3.3. Project Scope

The project's goal was to design and implement a system with a GUI that can be used to configure the Infrastructure-as-a-Service (IaaS) solution. The IaaS is built by the ThermoFisfher Scientific R&D team, known as SDP. The system must produce the configuration output that the format and structure must be compatible with the one produced by the existing CLI-based application. As a result, the visual configurator can be integrated with the current SDP release cycle.

3.4. Domain Analysis

The EM's component and the DMP are configured by the FSE on the customer site. After the FSEs configure the Microscope components and the basic network configuration of the DMP, DSE continues the setup. The SDP is usually configured remotely by the DSE or can also be on-site, depending on the customer network policy.

The software packages required to configure SDP are encapsulated in an ISO file. There are two ISO files, the first ISO file contains the Operating System installer, and the second ISO file contains the SDP-related packages. When the EM and DMP are shipped to the customer, the Service Engineers need to bring these ISO files to be able to configure SDP. Figure 4 shows the high-level flow of the SDP installation. The process starts when the DSEs have access to the DMP. Then, the DSEs mount two ISO files containing the necessary software packages required to install the SDP. Once the SDP ISO file is mounted, the DSEs then execute the initial setup script to start the configuration process. After that, the DSEs need to fill in the configuration item through the prompt dialog. These steps (Steps 5 and 6 in Figure 4) are the scope of the project. The complete step of this process can be accessed in Appendix A.

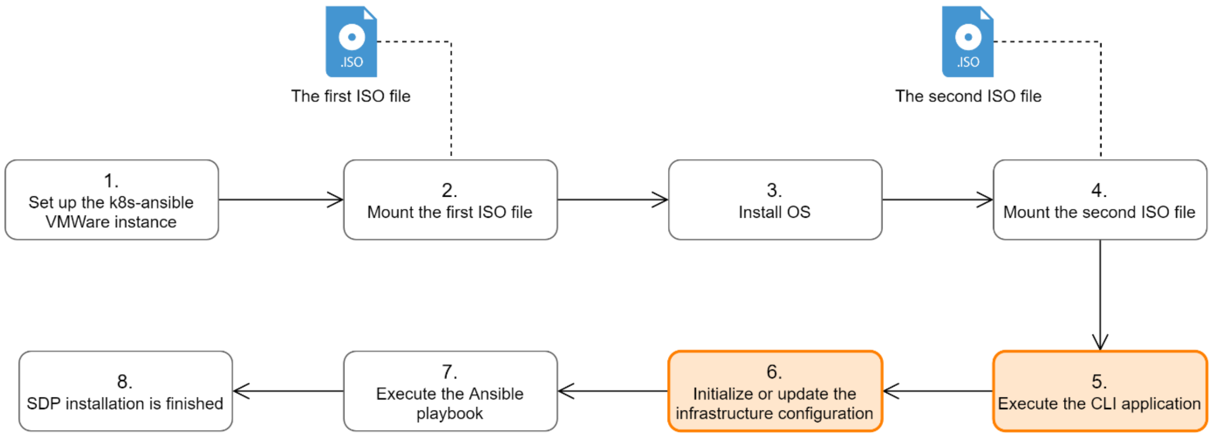

.ISO
The first ISO file
.ISO
The second ISO file
1.
Set up the k8s-ansible VMWare instance
2.
Mount the first ISO file
3.
Install OS
4.
Mount the second ISO file
5.
Execute the CLI application
6.
Initialize or update the infrastructure configuration
7.
Execute the Ansible playbook
8.
SDP installation is finished

Figure 4. High-level flow diagram of SDP installation

SDP uses Ansible [6] technology to set up multiple nodes in the infrastructur. A detailed reasoning and the comparative study of this decision are reported in [7].

To understand the current system so that we could achieve the goals of this project, we conducted further analysis on the following topics:

1. Containerization
2. IT Automation with Ansible
3. Current SDP Configurator
4. SDP deployment type

3.4.1. Containerization

Virtualization is the closest analogy we could use to understand containerization. A virtual machine is an abstraction of the physical hardware. VMWare and VirtualBox are examples of tools to run several virtual machines in a single physical server. Unlike virtual machines, container is an abstraction of the application layer that packaged the source code and its dependency together [8]. The package is called a container image. One of the examples of containerization technology is Docker. We also adopted Docker to containerize the project. Figure 5 illustrates the differences between containers and virtual machines.

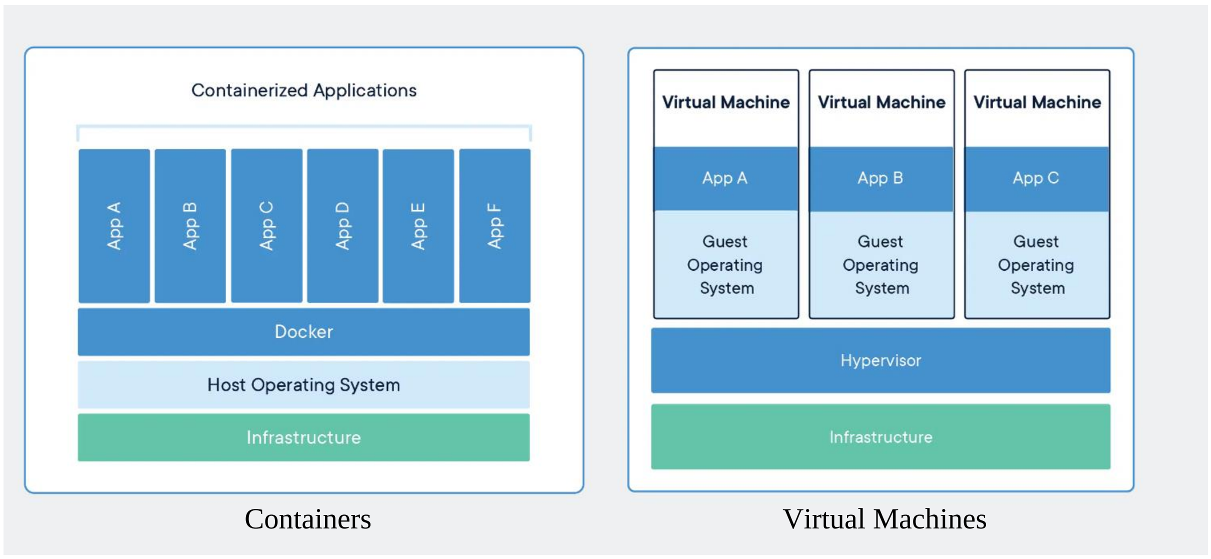


Figure 5. Comparing containers and virtual machines **[8]**

3.4.2. IT Automation with Ansible

Ansible is an open-source IT automation tool that automates provisioning, configuration management, application deployment, orchestration, and many other manual IT processes [9]. We can use Ansible to automate various tasks such as installing software, automating daily tasks, and provisioning infrastructure. Ansible works by sending a set of instructions that usually have been executed manually to the target hosts using the Secure Shell (SSH) protocol.

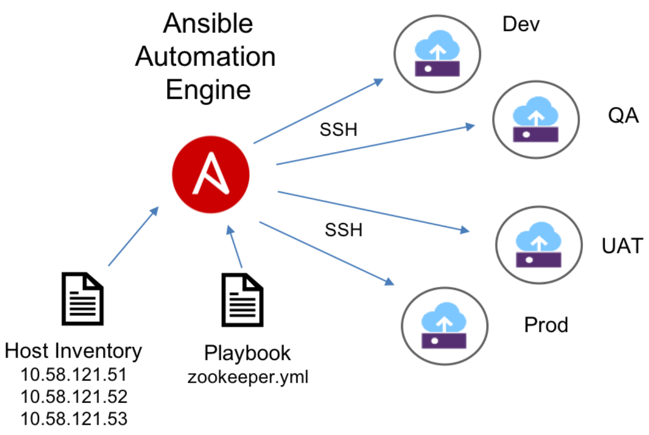


Figure 6. Configuring multiple environments using Ansible **[10]**

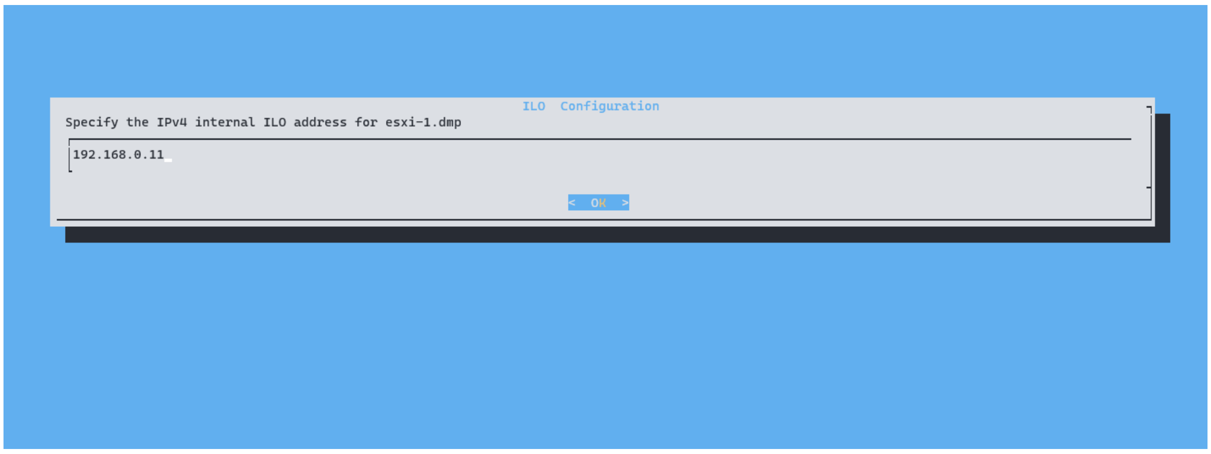
Figure 6 illustrates how Ansible applies an infrastructure configuration to multiple hosts. Ansible provides a mechanism to group a list of nodes we want to configure called an inventory. In addition, the list of instructions is also grouped in a file known as a playbook. In the remaining paragraph of this section, we describe in detail about the Ansible inventory and Ansible playbook.

Ansible Inventory

Ansible works against multiple nodes or hosts through an SSH connection unlike any other tool, which is typically using agent software installed on each node. Ansible groups the list of nodes in a file known as inventory [6]. A list of nodes is specified in the inventory including its configuration property such as the IP address, MAC address, memory, and storage size.

Ansible supports several file formats for its inventory file. The common formats are INI, YAML, and JSON. In the context of this project, we focused on analyzing the inventory in JSON format since it is used by ThermoFisher Scientific.

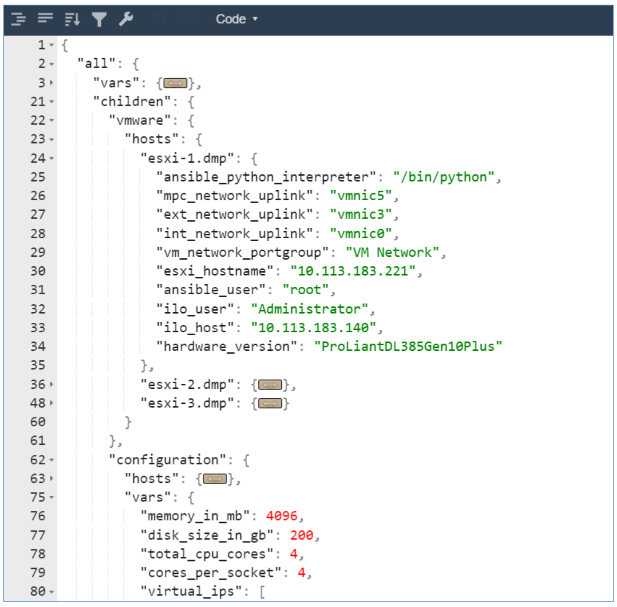
Currently, ThermoFisher Scientific uses a CLI application to construct the inventory. Figure 7 shows the example of the SDP installer interface in a Bourne Again Shell (BASH).



```
ILO  Configuration
Specify the IPv4 internal ILO address for esxi-1.dmp
192.168.0.11
< OK >
```

Figure 7. Example of IP address configuration using BASH prompt dialog

To complete the inventory structure construction, the Service Engineers need to fill in the necessary information into the BASH prompt dialog. Figure 8 shows the output of the Ansible inventory constructed using the CLI application. The complete example of the Ansible inventory can be found in Appendix G. The content of the Ansible inventory file in Appendix G was altered due to confidentiality information. However, the structure of the inventory file is maintained. The inventory structure was useful to get an overview of the target output of the SCA. In addition, the inventory structure influenced the design of the SCA which is explained in the later chapter of this document.



```
{
  "all": {
    "vars": {↔},
    "children": {
      "vmware": {
        "hosts": {
          "esxi-1.dmp": {
            "ansible_python_interpreter": "/bin/python",
            "mpc_network_uplink": "vmnic5",
            "ext_network_uplink": "vmnic3",
            "int_network_uplink": "vmnic0",
            "vm_network_portgroup": "VM Network",
            "esxi_hostname": "10.113.183.221",
            "ansible_user": "root",
            "ilo_user": "Administrator",
            "ilo_host": "10.113.183.140",
            "hardware_version": "ProLiantDL385Gen10Plus"
          },
          "esxi-2.dmp": {↔},
          "esxi-3.dmp": {↔}
        }
      },
      "configuration": {
        "hosts": {↔},
        "vars": {
          "memory_in_mb": 4096,
          "disk_size_in_gb": 200,
          "total_cpu_cores": 4,
          "cores_per_socket": 4,
          "virtual_ips": [
```

Figure 8. SDP Ansible inventory structure snippet

Ansible Playbook

In the earlier section, we discussed an Ansible inventory, which is the list of hosts or nodes Ansible configured. Once the inventory is defined, we can use the Ansible playbook to execute a set of instructions to the corresponding nodes. This set of instructions is stored in an Ansible playbook also known as a blueprint of the automation task [11]. Ansible playbooks are prewritten by the developers and they can be executed with limited or no human involvement. As mentioned earlier that ThermoFisher Scientific adopted Ansible technology to automate the IT infrastructure configuration with the SDP. However, the implementation detail of the Ansible playbook is not in the scope of this project but is useful to understand the SDP as a whole system.

3.4.3. The Existing SDP Configurator

SDP was built by adopting Ansible technology to orchestrate the IT infrastructure, such as defining the number of hosts, configuring the network, installing service tools, and deploying the applications suites. In Section 3.4.2 we explain the Ansible inventory and its structure. In Addition, we showed an example of the current prompt dialog to configure the IP address of a node. In this section, we analyzed further the most important component of the current CLI-based SDP configurator. Figure 9 describes the component of the SDP configurator in the form of a domain model.

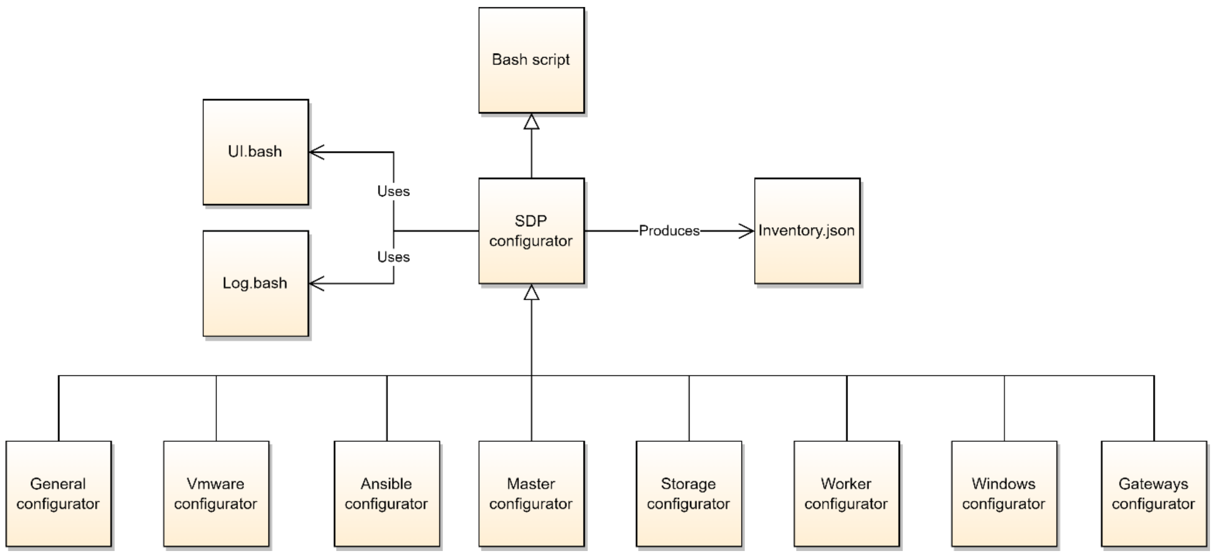


Figure 9. Software Delivery Platform (SDP) domain model

For the readability of the diagram, we only illustrated the most relevant component of the SDP configurator. As shown in Figure 9, several components extend the SDP configurator, namely *General*, *VMWare*, *Ansible*, *Master*, *Storage*, *Worker*, *Windows*, and *Gateways*. Each of those components is responsible for configuring the corresponding node. For instance, the Worker component is containing the script used to configure the worker nodes. The configuration of the workers such as the number of the worker node, the network configuration of each node, and the storage. In addition, each component uses *UI.bash* and *Log.bash* to construct the prompt dialog on the Command Line Interface and save the logs of the configurator respectively.

The current SDP provides the installation forms using prompt dialog. The users could follow the installation step and fill in the necessary information accordingly. However, with the limitation of the BASH prompt dialog, some essential features are missing, such as there is no overview of how many steps are required to complete the installation, navigating between the step is not possible, and the summary of the installation result is limited. However, the business logic was already in place whereas also needed to consider whether the component that was written in BASH script can be reuse or completely re-write it into a new technology during the implementation of the prototype.

The current SDP configurator guides the Service Engineer in such a way as to complete the Ansible inventory construction. Starts from the basic or general configuration, then the VMWare, and continued with the Ansible and so on. At the end of the process, the SDP configurator exports the Ansible inventory in a JSON format. Figure 10 shows a visual representation of the inventory file produced by the CLI-based SDP configurator.

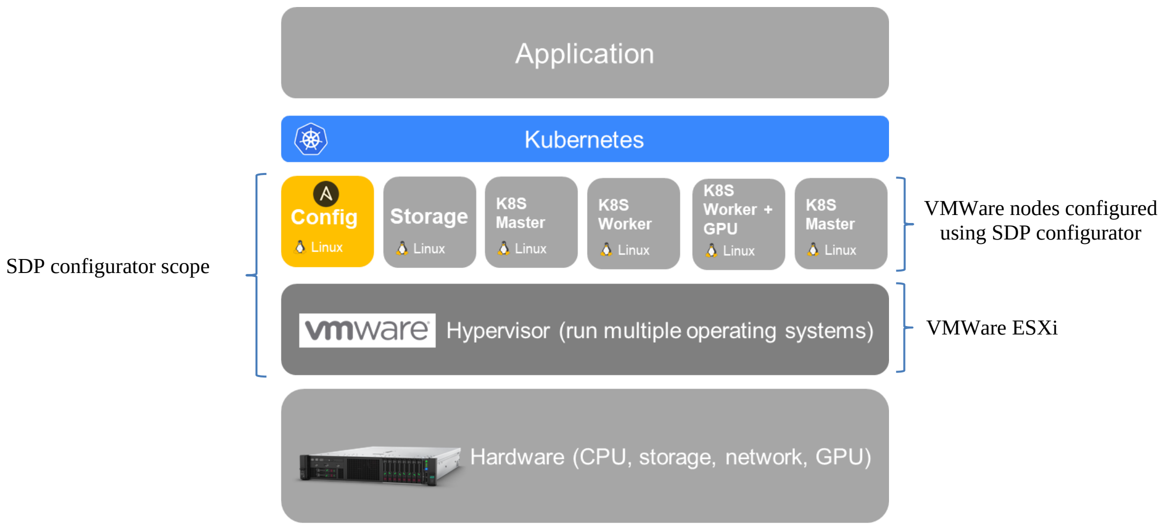


Figure 10. SDP infrastructure layer

3.4.4. SDP Deployment Type

For different Microscopes, the ThermoFisher Scientific engineers have to set up a different configuration. The ThermoFisher Scientific engineers can select this configuration as a deployment type. By selecting the right deployment type, the engineers only need to fill in any custom configuration; the rest is already provided in the deployment template. This template provides the default values. Therefore, the value may be changed depending on the needs.

Currently, the SDP configuration supports seven different deployment types: *Advanced*, *CoreImageFacilities*, *FleetManager*, *HeliosISS*, *Lifescience*, *MetriosML*, and *SDB*. The deployment types are defined based on what application is deployed on the cluster. The architect of the SDP application team decides how many resources they need to have these applications running correctly on the cluster. After that, the SDP infrastructure team creates the deployment template with the default values in it, for instance, the number of Kubernetes workers, the number of Graphical Processing Unit (GPU) available, and the number of ESXi servers.

4. Requirements Elicitation

This chapter explains the functional and non-functional requirements of the system. Section 4.1 describes the requirement overview. Section 4.2 provides an overview of the system's functional requirements and how it is developed and prioritized. Finally, Section 4.3 shows the non-functional requirements of the system.

4.1. Requirement Overview

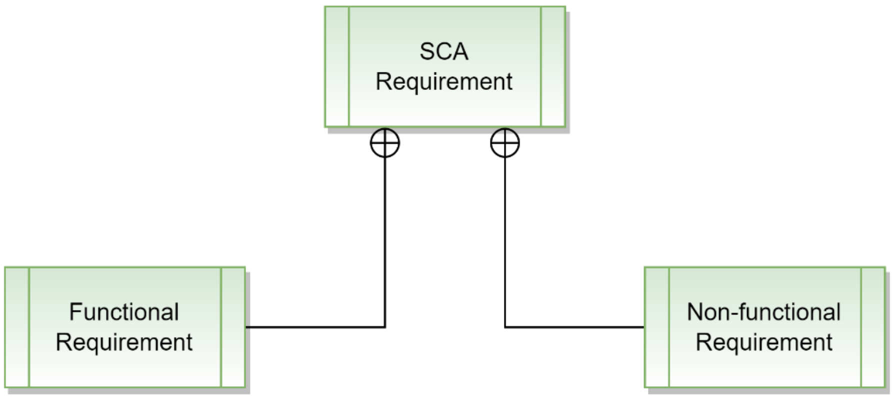


Figure 11. Requirement overview

There are two categories of requirements for this project. The functional requirement is related to the functionality that needs to be developed. The non-functional requirement is the requirement that defines the system attributes. It also directs the design criteria of the system to satisfy these non-functional requirements.

The following techniques were used to acquire requirements:

- Brainstorming
- Stakeholder interview
- Comparative study
- Prototyping

The brainstorming session was held during the initial stage of the project. We conducted the brainstorming session starting by analyzing the problem and the possible solution to solve the problem. We came up with several functional requirements at the end of the session as listed in the next section. We also interviewed the relevant stakeholders to understand their needs. To choose the possible technology to develop the prototype, we compared several technologies. The comparative study result can be found in Appendix D. In addition, to understand whether a certain technology could help in delivering the solution of this project, we developed a prototype to show and discuss with the stakeholders the candidate technology in action.

We used *MosCow* prioritization to set the priority of the requirements [12]. Table 2 describes the list of priorities we used in this project.

Table 2. MosCow prioritization

Must have	Non-negotiable product needs that are mandatory for the team
Should have	Important initiative that are not vital, but add significant value
Could have	Nice to have initiative that will have a small impact if left out
Will not have	Initiative that are not the priority for this specific time frame

4.2. Functional Requirements

Figure 12 illustrates the functional requirements with a must priority based on *MosCow* requirements prioritization. For the readability of this report, we only show the *must-have* requirement in the figure. The complete functional requirements can be found in Appendix B.

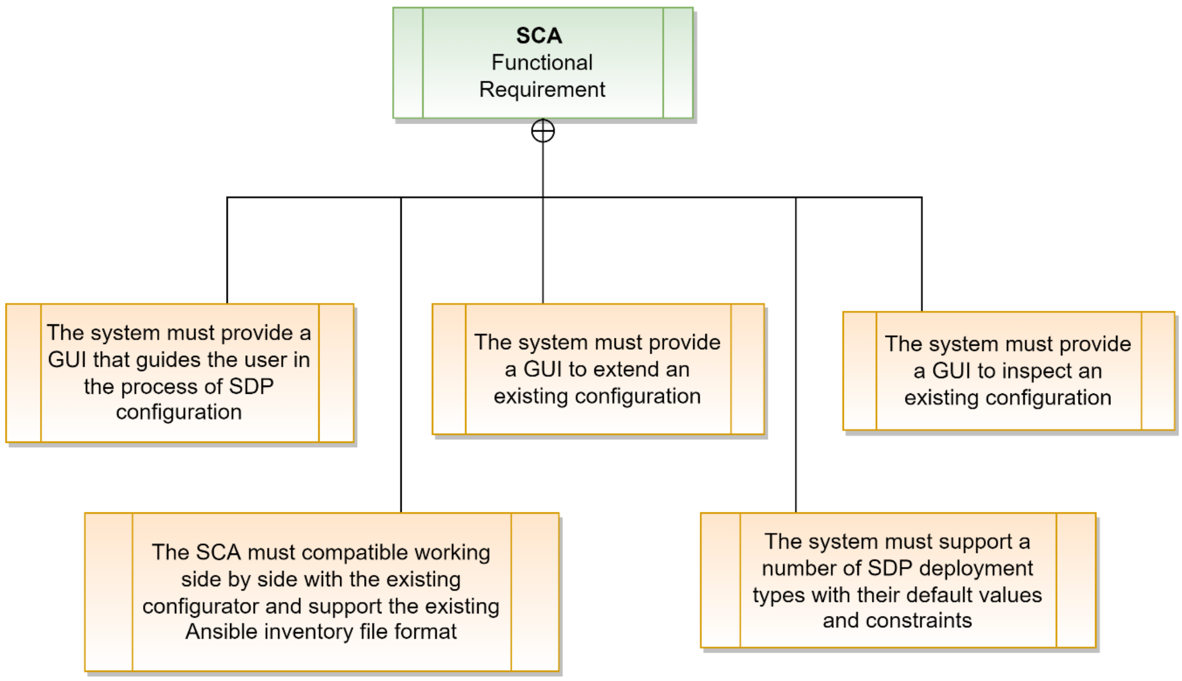

SCA Functional Requirement
The system must provide a GUI that guides the user in the process of SDP configuration
The system must provide a GUI to extend an existing configuration
The system must provide a GUI to inspect an existing configuration
The SCA must compatible working side by side with the existing configurator and support the existing Ansible inventory file format
The system must support a number of SDP deployment types with their default values and constraints

Figure 12. Functional requirements with a *must-have* priority

4.3. Non-functional Requirements

Figure 13 illustrates the non-functional requirements of the system. Prioritization of the non-functional requirements also follows the *MosCow* prioritization. The title of the red box is the category of the requirements. The detailed list of the non-functional requirements can be found in Appendix C.

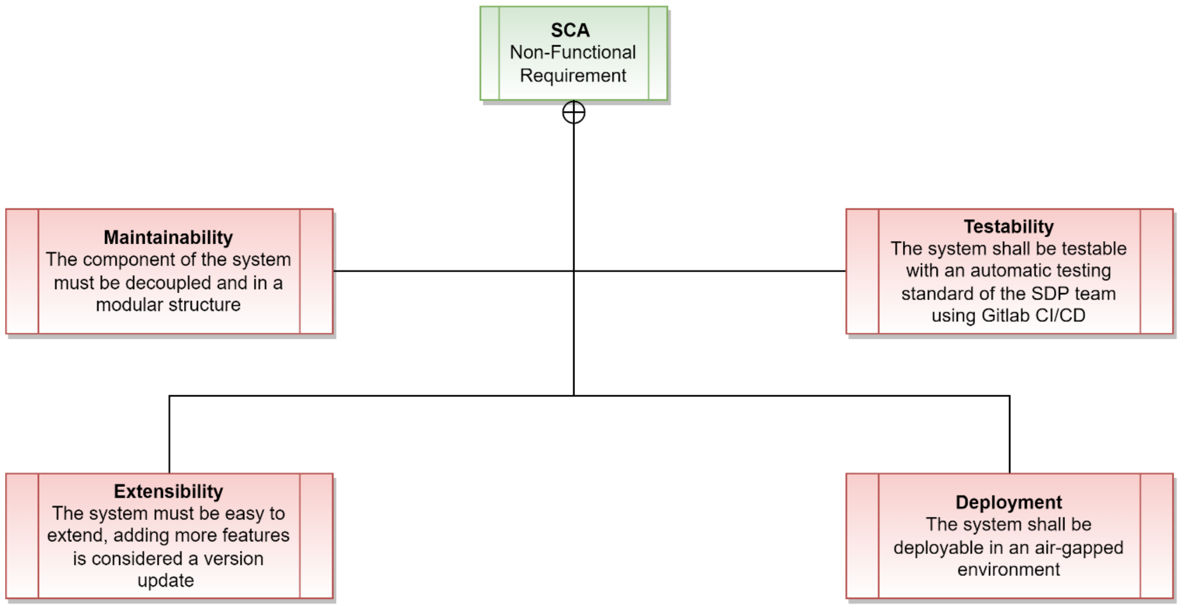

SCA
Non-Functional Requirement
Maintainability
The component of the system must be decoupled and in a modular structure
Testability
The system shall be testable with an automatic testing standard of the SDP team using Gitlab CI/CD
Extensibility
The system must be easy to extend, adding more features is considered a version update
Deployment
The system shall be deployable in an air-gapped environment

Figure 13. Non-functional requirements of the system

5. Software Architecture and Design

This chapter describes the four different views and the use cases of the project based on the 4+1 architectural concept introduced by Phillipe Kruchten [13]. The detailed explanation of each view is provided in each section of this chapter.

5.1. 4+1 Architectural View

To describe the architecture of the system, it is helpful to look at the best practices and industry standards. The well-known and widely used architectural view is the 4+1 view model introduced by Phillipe Kruchten [13]. The 4+1 architecture has multiple views to address separately the concerns of the various stakeholders: end-user, developers, and system engineers. The main goal of the 4+1 architectural concept is to easily illustrate the system in four different views to several stakeholders, which hold different roles in the company. Therefore, adopting the 4+1 architectural view in this project was useful to show the different perspectives on describing the system.

The 4+1 architecture consists of four different views, which are:

- Logical view – shows the component (object) of the system as well as the interaction. In the UML diagram, this view can be illustrated using a class diagram or object diagram.
- Process view – shows the processes of the system. This view can be illustrated using an activity diagram.
- Development view – provides building block views of the system and describes the static organization of the system modules. UML component and package diagram can be used to illustrate this view.
- Physical view – shows the installation, configuration, and deployment of the system. A UML deployment diagram can be used to illustrate this view.

The *one* from 4+1 architecture is Scenario. The Scenario is the use cases that are supported by the system, which becomes the fifth view.

5.2. Use case

Figure 14 illustrates the use cases of the system. Detailed information on each use case can be found in Appendix F. The detailed information includes:

- Actors
- Pre-condition
- Basic flow
- Post-condition
- Alternative flow

There are two actors mentioned in the diagram in Figure 14. The Service Engineer, the end-user of SCA, has three use cases: clean install SDP, update SDP configuration, and visualize the configured infrastructure from the specification in the generated inventory. The second actor is the Research and Development (R&D) Engineer who can access all features of SCA.

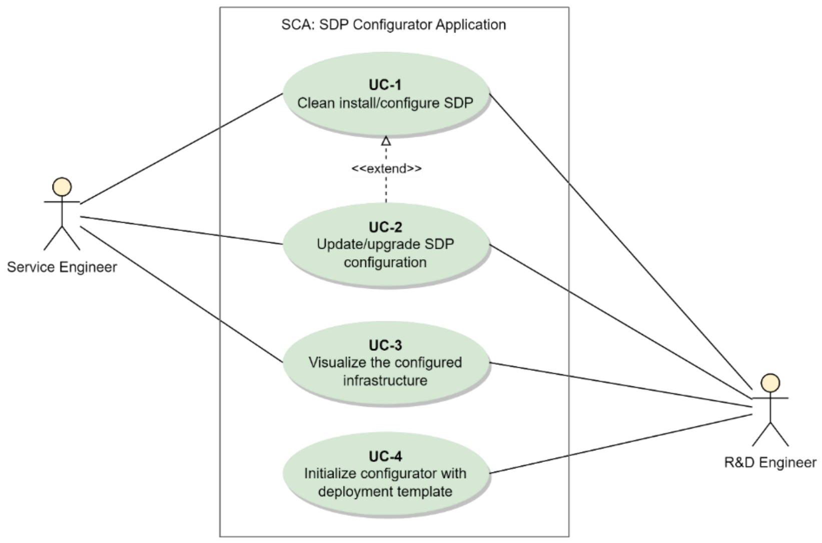

SCA: SDP Configurator Application
UC-1
Clean install/configure SDP
<<extend>>
UC-2
Update/upgrade SDP configuration
UC-3
Visualize the configured infrastructure
UC-4
Initialize configurator with deployment template
Service Engineer
R&D Engineer

Figure 14. Use case diagram of the system

5.3. Logical View

We developed the project in the modular structure to satisfy the NF-01 – Maintainability of the non-functional requirement of the project. Figure 15 illustrates the high-level logical view of the SCA from the layered architecture viewpoint. The system's main components are the presentation layer, the business logic layer, and the persistent layer. The dependency happened from the upper layer down to the lower layer as shown the Figure 15.

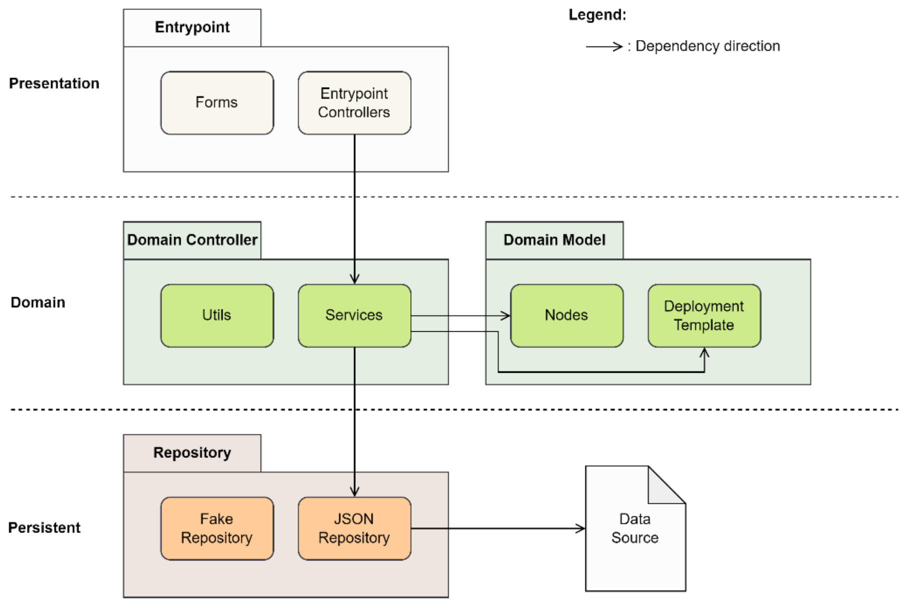


Figure 15. Logical view in the layered architecture viewpoint

Table 3 contains the explanation of each component depicted in Figure 15. The information in the table is helpful to understand the description and the functionality of each element of the diagram. The detailed interaction between components is explained in the upcoming section.

Table 3. The component description

Component	Layer	Description	Functionality
Entrypoint	Presentation layer	The entry point was developed using the Flask web application framework. The *Entrypoint controller* is responsible for handling the HTTP endpoint while the *Forms* are used to dynamically structure the HTML form and its validation.	• Providing the UI of the application • HTTP endpoints • Generating form and validation
Domain controller	Domain layer	The domain controller or usually mention as the service layer contains the business logic of the application. The main operations are stored in the *Service* Python module while some common operations are stored in the *Utils* Python module.	• Instantiating the model • Retrieving data from the entrypoint • Loading, Aggregating, and saving data to the targetted file format • Establishing a connection to an external server

Domain model	Domain layer	The domain models hold the structure of the data. The *Nodes* represent the domain object that is mentioned in the Ansible inventory structure (See Appendix G), such as the *VMWare, Ansible, Master, Worker, Gateways*, and *Windows* nodes. There is also a *Deployment Template* model that contains the predefined value loaded from the template file.	• Structuring the data • Acting as the Data Transfer Object (DTO) of the system
Repository	Persistent layer	It is specifically designed to handle data-related operations, for instance getting and saving data to the data source. The *JSON Repository* is responsible for handling JSON data format. However, the *Fake Repository* is used for testing purposes so that we could test the system without actually saving the JSON data into a file.	• Reading data from the data source • Saving data to the targeted file format

5.3.1. The Entrypoint

The Entrypoint provides direct services to the users to interact with the system. The SCA's entry point was built in a web-based application using the Flask framework. In Flask, there are three important components to explain. First, the HTTP endpoints that handle HTTP requests were handled in the controllers. This controller was different from the controller in the Domain layer. The controller in the Entrypoint handles the *GET* and *POST* HTTP methods. It passes the data to the lower layer of the system, which was the Domain layer for further operations such as aggregating and saving to the data source. The next component is the Forms component. The Forms component was used to dynamically structure the HTML form including its validation. With the Forms component, we could define the attribute of the form such as the form name, label, placeholder, default value, and the data type we wanted to validate. The Forms component could specify the validation for several criteria, such as *IPAddress*, *MACAddress*, *InputRequired*, *StringInput*, and *IntegerInput*.

5.3.2. The Domain Controller

The Domain controller is the component that handles the behavior or the business logic of SCA. The business logic for each configuration item was implemented in a separate module. We used the Python module for *VMWare, Ansible, Master, Workers, Gateways, Storage, Kubernetes*, and *Windows* to organize the code. We separated them in such a way as to increase the maintainability and the separation of concerns of each module, which was also mentioned in the non-functional requirement of this project.

5.3.3. The Domain Model

The domain model represents the entity we need to configure, such as the *VMWare, Ansible, Master, Workers, Gateways, Storage, and Windows* nodes. We used inheritance by having a parent class called Node. In the Node class, we specified the common behavior that its subclasses could use. These behaviors are the *to_inventory()* method used to export the object into JSON that comply the Ansible inventory structure. In addition, the Node class has a method called *from_inventory()* to convert the data retrieved from the Ansible inventory into a Python object. We used the *Pydantic* library to simplify the conversion process from JSON format to Python object and vice versa. There was also a need to name the field with the less technical terms (aliases) in the deployment template. These two feature is offered by *Pydantic* out-of-the-box, so that function such as mapping the classes' field and the deployment template's attribute is no longer required.

5.3.4. The Repository

The repository is the component we used to save and load the data from the JSON file. We saved the data to the JSON file directly instead of to a database. In Section 3.4 we explained that the format used for the Ansible inventory as well as the output of the SCA is a JSON formatted file. Therefore, removing the database from the design was essential to avoid unnecessary dependency.

We implemented the repository in two different classes. Firstly, the *JSONRepository* was developed to handle the JSON formatted data source. The operation includes reading and writing the data to *inventory.json.* Secondly, the FakeRepository was implemented for testing purposes. When we tested the code using the JSONRepository, we learned that there were some aspects to consider, such as removing the generated inventory after the testing or making sure that the inventory file did not exist which might affect the test. Using the FakeRepository, we no longer needed to check the state of the inventory.json file beforehand. Thus, we could test the behavior of the code without actually reading or writing the inventory file to the file system.

5.3.5. Component Interaction

Figure 16 shows how each component of the system interacts whenever there is a request from the user while accessing the web page. The process starts when the entry point receives a request from the user. The entry point controllers pass the request to the service to get the data. In the services, it instantiates the domain model class and uses the repository to read data from the data sources such as an existing Ansible inventory file or template data when instantiating the model. The instantiated domain model is then called a Data Transfer Object (DTO) and the service returns it to the entry point controller. The entry point controllers use the DTO to inject data when creating the Forms and finally show the predefined form to the user on the web-based GUI.

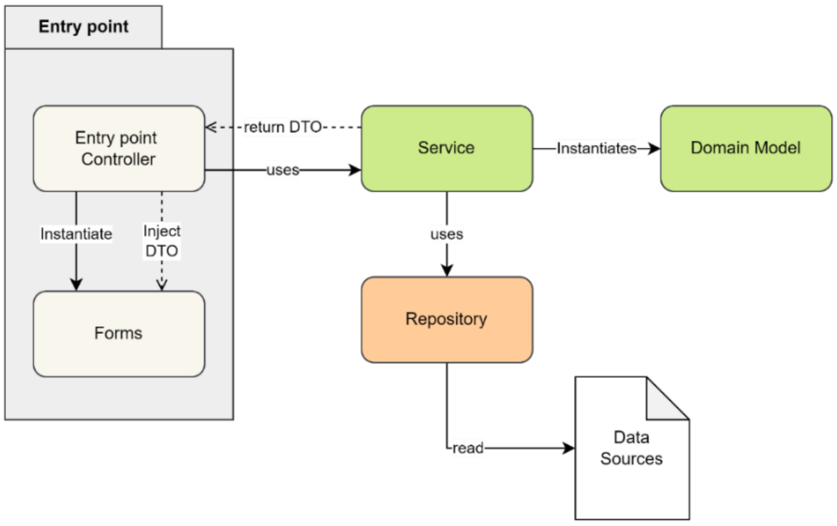


Figure 16. A high-level overview of the interaction between the Entrypoint, the Domain, and the Repository.

5.4. Process View

In the Process view, we explain several processes that are conducted by the users, such as accessing the SCA web page and submitting, validating, and saving the form data into the Ansible inventory file. There are also processes that happen in the background, for instance, the processes of SCA accessing an ESXi server and an instance of the VMWare.

5.4.1. Accessing SCA Web Page

Figure 17 illustrates the flow of accessing the SCA webpage. There are three actors shown in the activity diagrams. These actors are the user, the Presentation layer, and the Domain layer. The process starts when the users open a web browser and access the SCA webpage. The SCA Presentation layer initializes the form when the browser hits the endpoints. The Flask controller initializes the form and invokes the service in the Domain layer to load the existing data from the inventory. After that, the service checks whether the existing inventory file exists. If the inventory does not yet exist, the data is loaded from the default value and the template. The controller continues the form construction by populating the return value from the service. Finally, the controller renders the HTML template as an HTTP response to complete the process.

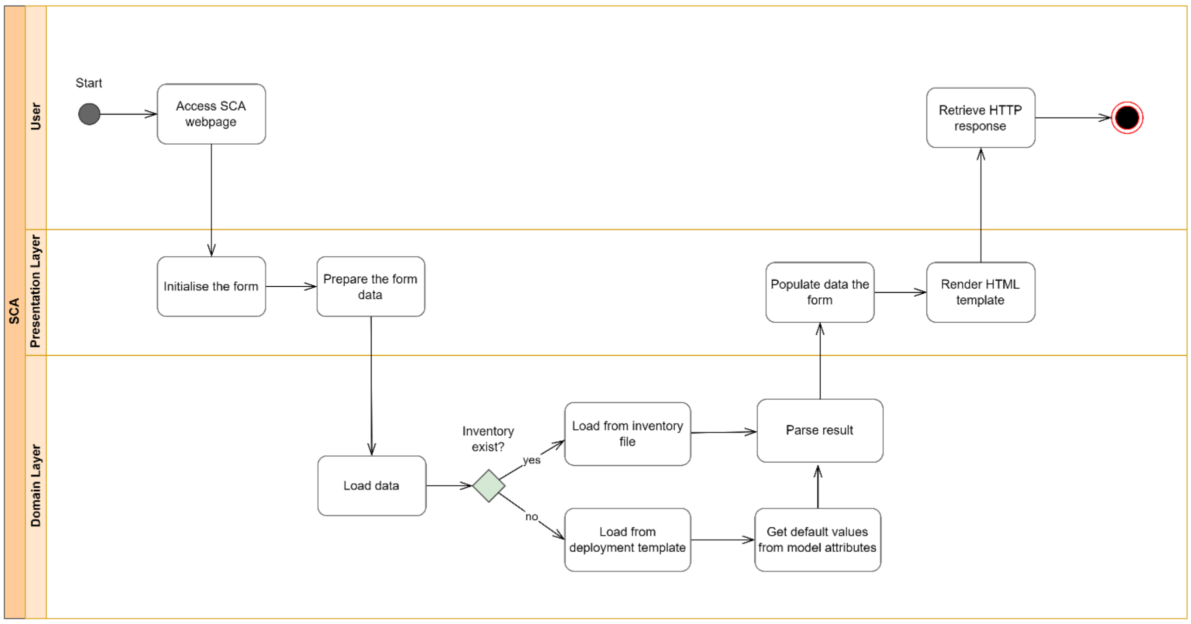


Figure 17. Process view accessing the webpage

5.4.2. Submitting and Saving the Form Data

Figure 18 illustrates the process of submitting the Form. Similarly, there are three actors in this process: the user, the Presentation layer, and the Domain layer. The starting point of this process is when the users hit submit button. This action then triggers the *POST* endpoint in the Presentation layer to validate the Form. Form validation logic was defined in the corresponding Form classes. The submitted Form is validated based on the validation criteria, for instance, *InputRequired*, *IPAddress*, *MACAddress*, *Password*, *String*, and *Integer*. Whenever the validation returns success, the data is parsed and saved into the JSON file; the controller sends a redirect response with a success message. On the contrary, the controller renders the same Form but includes the error messages on each failing field.

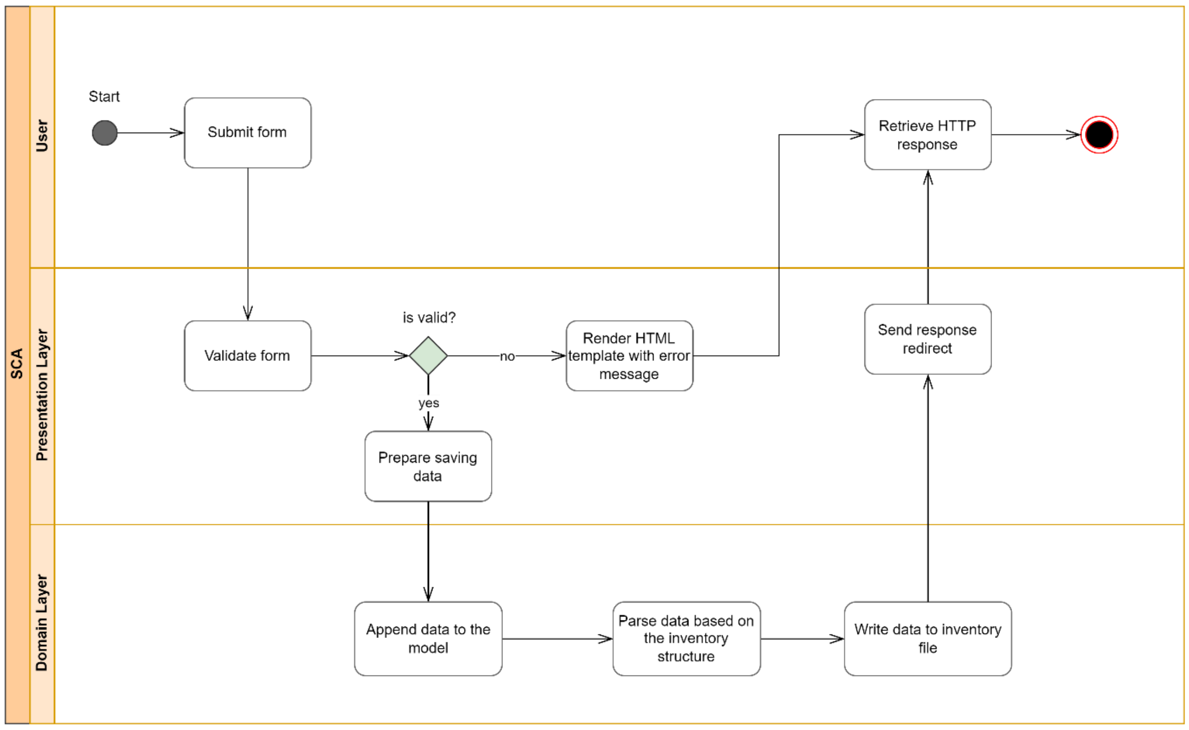


Figure 18. Process view submitting the form

5.4.3. Establishing Connection to an ESXi Server

The SCA needs to access external sources to get the data. The data are the MAC addresses of one of the VMWare instances, the hardware model of the ESXi server, and the available Drive Identifier of the ESXi server. To illustrate this scenario, Figure 19 provides information on accessing the ESXi to acquire hardware model data to provide the default value for the VMNIC (see the Glossary) for the configuration.

We leveraged the *Pyvmomi* library to establish a connection to an ESXi server through Python code. The process starts when the users access the VMWare configuration page on the SCA. Then, the users are required to fill in the ESXi hostname, username, and password as part of the configuration item. After that, the SCA retrieves the information through the HTTP handler in the Entrypoint and calls the *get_vmnic()* method in the service, and passes the hostname, username, and password as parameters. Once a connection is established, we could get data from the ESXi, for instance, the hardware model and the available drives. In this case, we needed to know the hardware model of the ESXi server and use the hardware model to get the VMNIC in the prepared configuration file. The configuration file contains a mapping between the hardware model and the default VMNIC that is used for the VMWare configuration.

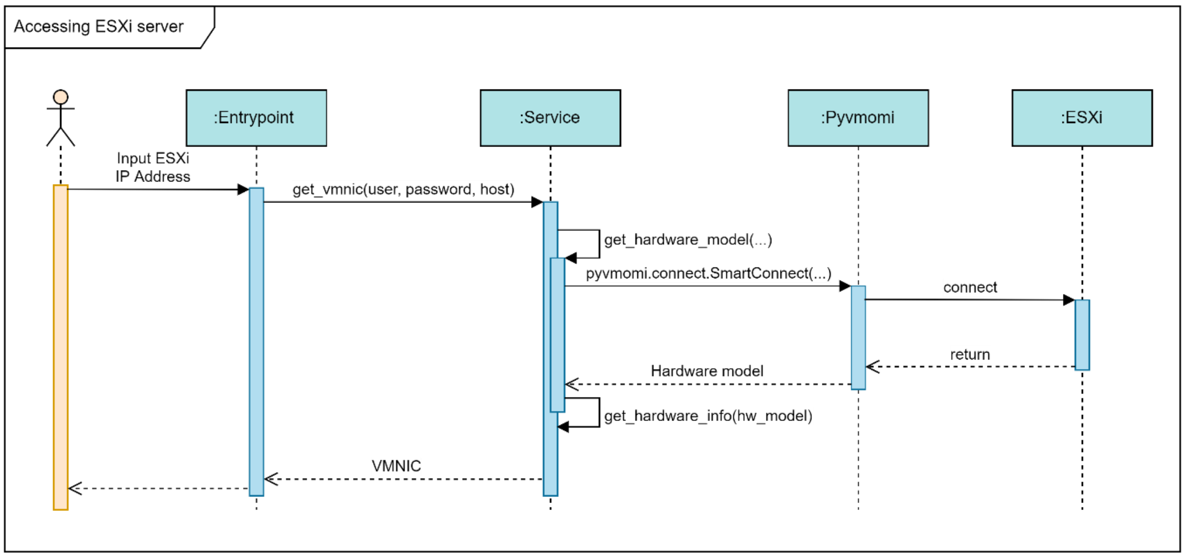


Figure 19. The Sequence diagram of accessing the ESXi server from the SCA

5.4.4. Accessing VMWare Instance Using SSH

Another process that requires and external sources is acquiring the MAC addresses of the Ansible node as shown in Figure 20. Ansible node is a VMWare instance that is pre-configured by the SDP as the host of the SDP configurator.

Whenever the users open the Ansible configuration page on SCA, the users are supposed to see two MAC addresses on the dropdown menu. These MAC addresses were not generated but should be acquired from the Ansible node using an SSH connection. After the entry point's controller receives the *GET* request, the *get_mac_addresses()* function is triggered to access the Ansible node and get the MAC Addresses of that node. We used the *Paramiko* library to implement the SSH connection mechanism because it offers a high-level implementation of establishing an SSH connection that is ready and easy to use in the SCA source code. The detailed sequence of this process can be observed in Figure 20.

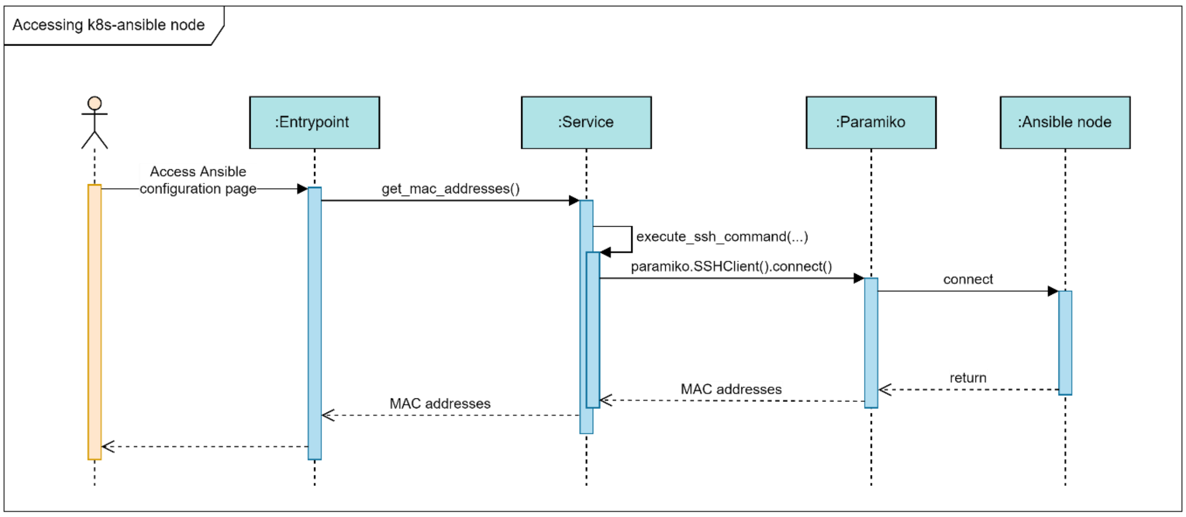


Figure 20. The Sequence diagram of accessing the k8s-ansible node server from the SCA

5.5. Development View

Figure 21 describes the component organization of the SCA. On the client side, the users could access the SCA from any browser such as Chrome, Firefox, or Edge. On the server side, the components are organized into different groups of components.

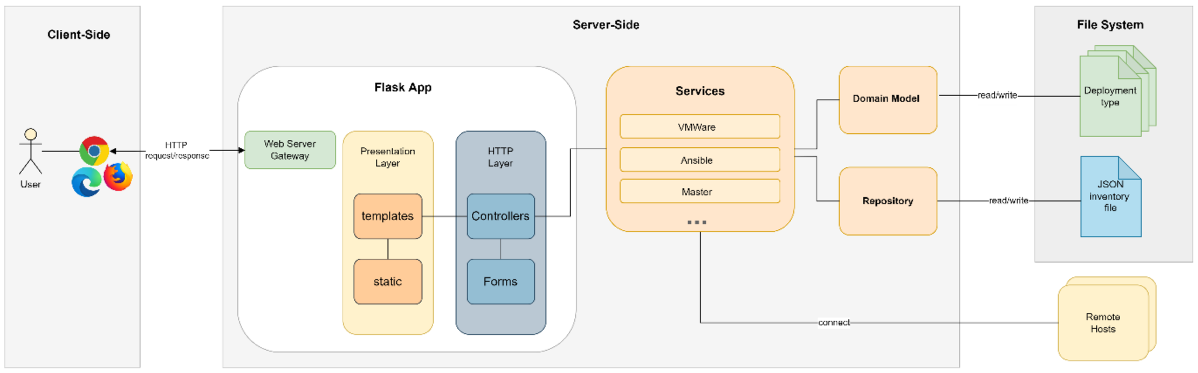


Figure 21. Development view diagram

The Flask application acted as the entry point of the system. We have the Web Server Gateway Interface (WSGI) using the Waitress Python library. This component keeps the SCA web application running. The presentation layer contains the HTML and static files such as JavaScript and CSS files. The next component is the HTTP layer, which contains several endpoints of the Flask application. The endpoints support GET and POST HTTP methods. Lastly, we used the Forms component to generate the HTML form input. The library used to create the Form component is Flask-WTF.

The Service Layer is the business logic of the SCA. The behaviors, including instantiating the model, updating the model instance attribute, and exporting the model into JSON, are stored in the Service layer. Additionally, SCA requires establishing a remote connection to remote hosts. This connection is used to acquire data during the SDP configuration process.

In conclusion, separating the entry point, business logic, and domain model improved the system's maintainability, separation of concern, modularity, and extensibility.

5.6. Physical View

Figure 22 shows the Physical view of SCA using the UML Deployment diagram. There are two devices required to deploy SCA. The first device is the Client PC. We tested SCA using three browsers, namely Chrome, Firefox, and Edge. The second one is the ESXi server. There are several instances inside this ESXi server running, including the Ansible node where the SCA is running as a Docker container. Docker is the only requirement needed to run an SCA. Additionally, we needed to ensure that the SCA container runs with an appropriate network configuration and that the volume is correctly bound.

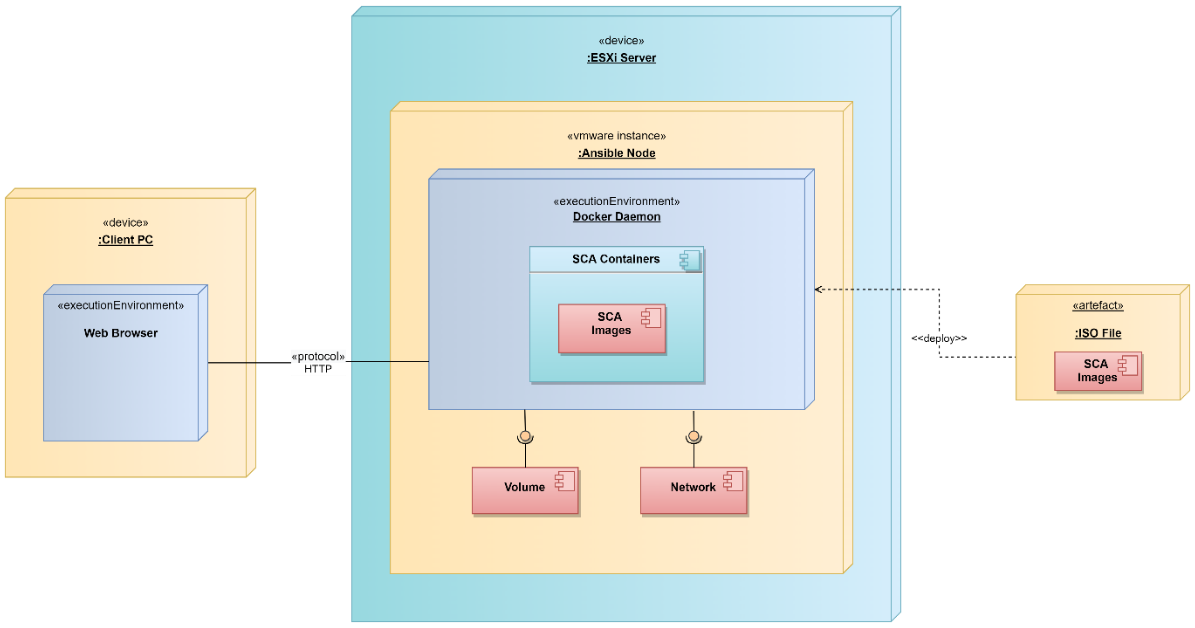


Figure 22. Physical view deployment diagram

6. Implementation

This chapter describes detailed information on SCA implementation. Section 6.1 describes the overview of the implementation of the system. Section 6.2 provides information on how we developed the User Interface (UI) mockups. Section 6.3 explains the SCA's features in detail. Section 6.4 provides information about the third-party component we used in the project. Section 6.5 illustrate the test automation and continuous integration setup of this project. Finally, Section 6.6 depicts the process of containerization of the solution.

6.1. Overview

The existing configurator was built in BASH scripts and executed as a CLI-based application. The logic of certain processes was defined in the scripts. Within the ThermoFisher SDP team, there was also a need to migrate from the procedural approach given the nature of a CLI-based application into an object-oriented way of structuring the logic of the configurator. Therefore, according to the architectural decision that we leveraged Python with Flask framework to develop the SCA, we re-wrote the logic that was written in a BASH script into Python code with an object-oriented approach. As a result, the code became easier to read, maintain, and extend due to the flexibility offered by Python with an object-oriented approach, such as modular, simpler, and minimal compared to the BASH script.

6.2. User Interface (UI) Mockups

To align with the user's expectations, we developed the mockups using Figma. Figma is a web-based application that can be used collaboratively and interactively to develop UI mockups [14]. In Figma, we can develop mockups with some behavior, such as navigating to other pages when the users click a button. Thereby, we can communicate with the end users more effectively by having mockups.
Figure 23 depicts the UI mockups used to present the idea of the SCA to the end-users.

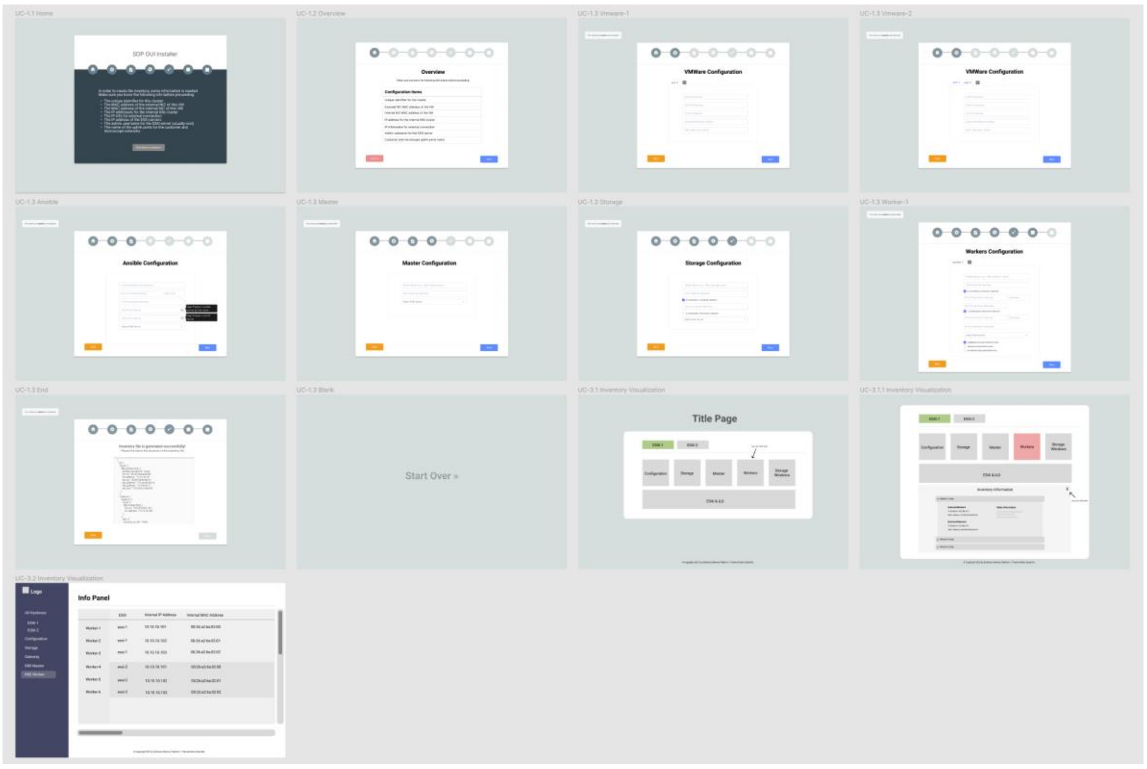
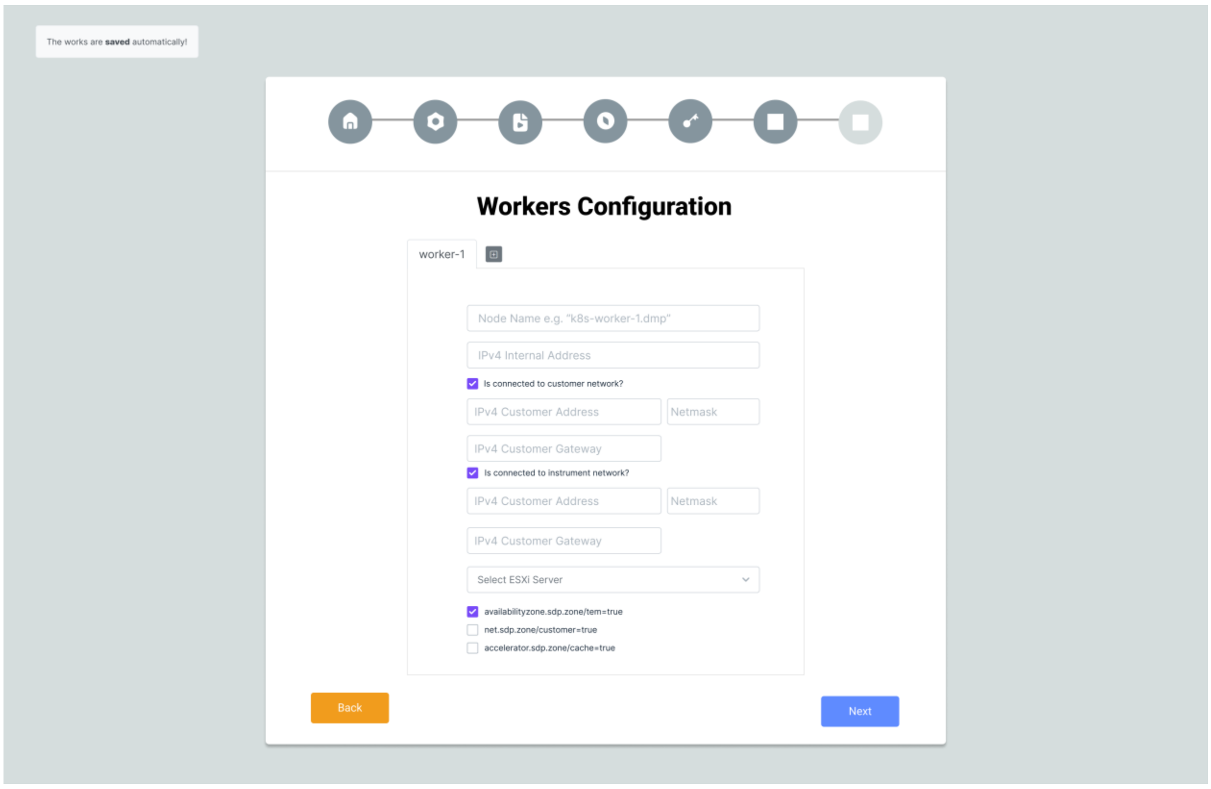


Figure 23. SCA UI mockups with Figma

Figure 24 is one example of the form UI mockup. The UI mockup illustrates how the worker page would be implemented for the end users. This approach was very effective in gaining feedback from the end users.


The works are **saved** automatically!

Workers Configuration

worker-1

Node Name e.g. "k8s-worker-1.dmp"

IPv4 Internal Address

✓ Is connected to customer network?

IPv4 Customer Address | Netmask

IPv4 Customer Gateway

✓ Is connected to instrument network?

IPv4 Customer Address | Netmask

IPv4 Customer Gateway

Select ESXi Server

✓ availabilityzone.sdp.zone/tem=true

☐ net.sdp.zone/customer=true

☐ accelerator.sdp.zone/cache=true

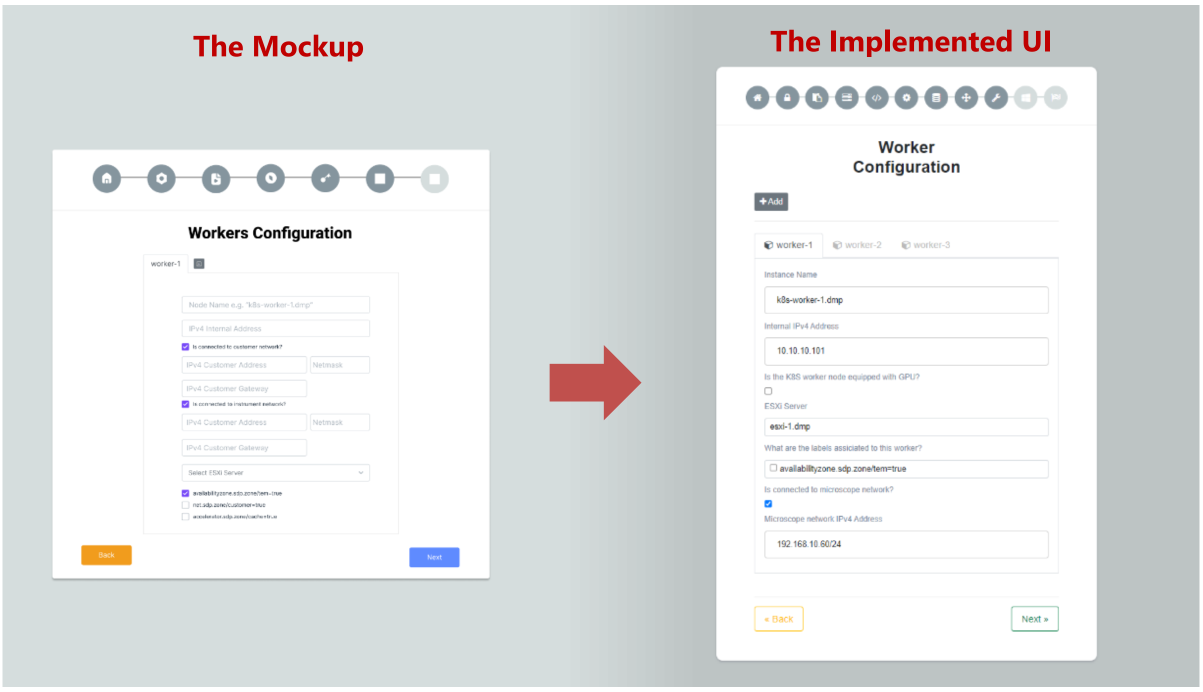
Figure 24. Worker form UI mockup

6.3. SCA Implemented Functionalities

In this section, several features of SCA are explained. Based on the use case diagram in Section 5.2, the Service Engineers have several use cases, which lead to the design decision during the project. The SCA consists of the following features:

1. Configuring SDP for fresh install
2. Updating an existing configuration
3. Visualizing the inventory
4. Configuring additional devices (NVMe)

Based on the mockups developed in Figma (see Section 6.2), we implemented the HTML and CSS code to accommodate the agreed mockups. Figure 25 shows the example of the comparison between the mockup developed in Figma and the actual UI implementation.



Workers Configuration

worker-1

Node Name e.g. "k8s-worker-1.dmp"
IPv4 Internal Address
[✓] Is connected to customer network?
IPv4 Customer Address | Netmask
IPv4 Customer Gateway
[✓] Is connected to instrument network?
IPv4 Customer Address | Netmask
IPv4 Customer Gateway
Select ESXi Server
[✓] availabilityzone.sdp.zone/tem=true
[ ] net.sdp.zone/customer=true
[ ] accelerator.sdp.zone/cache=true

Worker Configuration

worker-1 | worker-2 | worker-3

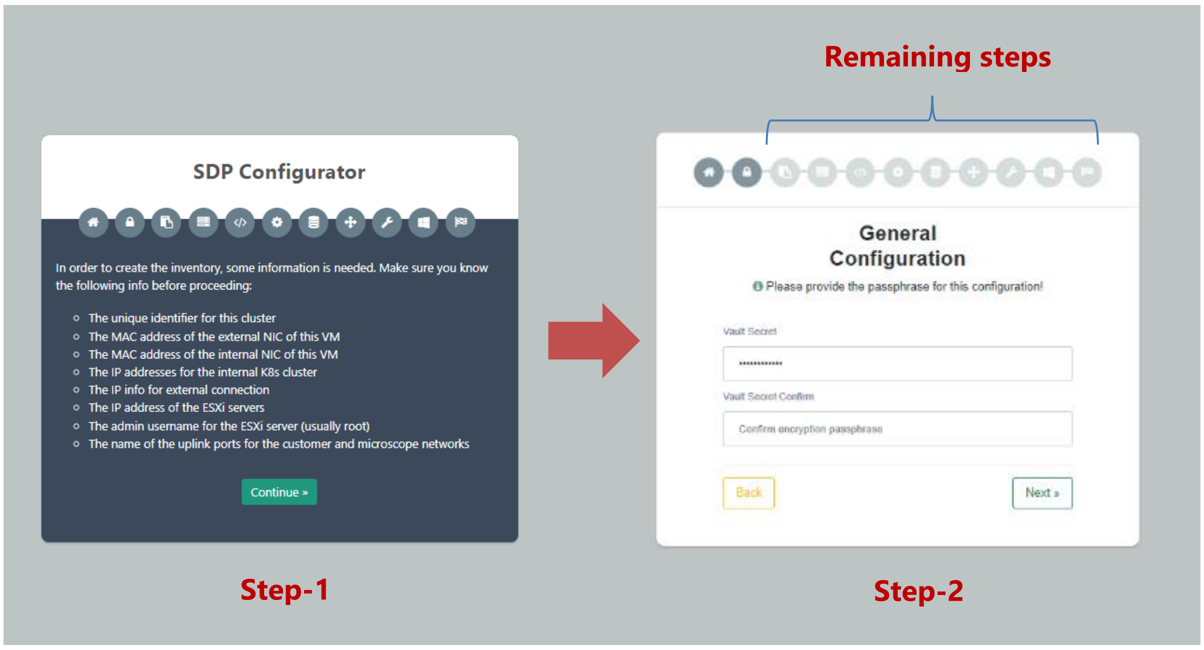
Instance Name: k8s-worker-1.dmp
Internal IPv4 Address: 10.10.10.101
Is the K8S worker node equipped with GPU? [ ]
ESXi Server: esxi-1.dmp
What are the labels assiciated to this worker? [ ] availabilityzone.sdp.zone/tem=true
Is connected to microscope network? [✓]
Microscope network IPv4 Address: 192.168.10.60/24

Figure 25. Worker configuration page mockup and actual UI implementation

6.3.1. Configuring SDP for Fresh Install

The initial configuration in the SDP CLI-based application was carried out in a guided step BASH prompt dialog. The users can follow the step and fill in the necessary information during the process. However, users cannot go back to the previous step to change the value whenever an invalid value is filled in. Also, the input value is not validated, which could lead to unintended behavior from the system whenever an invalid value is provided.

SCA was developed in a way to satisfy this requirement. The *easy-to-use* keyword was the primary purpose of this initiative. Therefore, we developed SCA, especially for the *Initial SDP Configuration* feature in a wizard step GUI. Navigating back and forth through the wizard form became an easy task. In addition, we added a validation mechanism to prevent invalid input from the users. Figure 26 depicts the wizard step GUI of SCA.



In order to create the inventory, some information is needed. Make sure you know the following info before proceeding:

- The unique identifier for this cluster
- The MAC address of the external NIC of this VM
- The MAC address of the internal NIC of this VM
- The IP addresses for the internal K8s cluster
- The IP info for external connection
- The IP address of the ESXi servers
- The admin username for the ESXi server (usually root)
- The name of the uplink ports for the customer and microscope networks

Please provide the passphrase for this configuration!

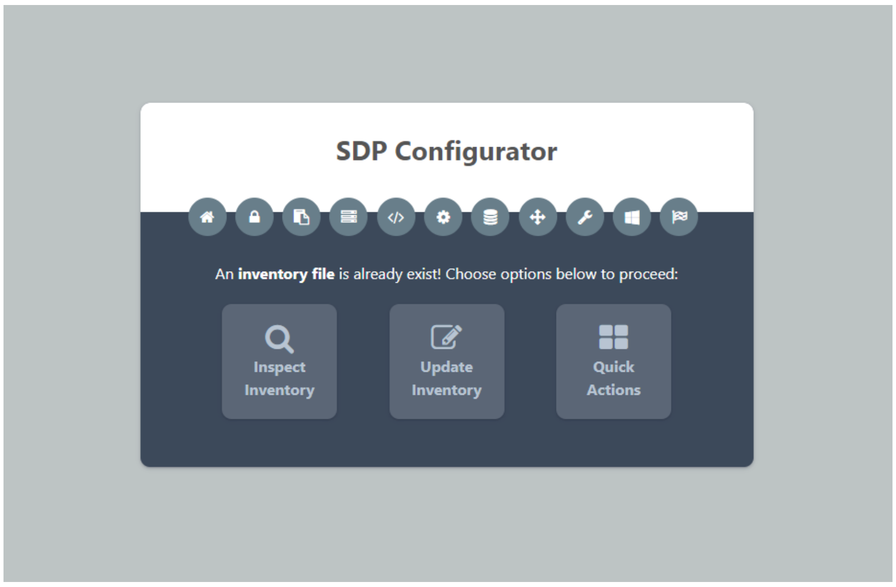
Vault Secret: ••••••••••••

Vault Secret Confirm: Confirm encryption passphrase

Figure 26. Wizard step form

6.3.2. Updating an Existing Configuration

In SCA, the user can reopen the existing Ansible inventory file to modify the configuration value, such as the internal IP Address for the worker node, GPU memory, and storage size. This feature was also developed in a wizard form similar to the initial install feature. Figure 27 shows the home page if an existing Ansible inventory file exists. The user then can choose either to update or to create a new Ansible inventory from the menu.

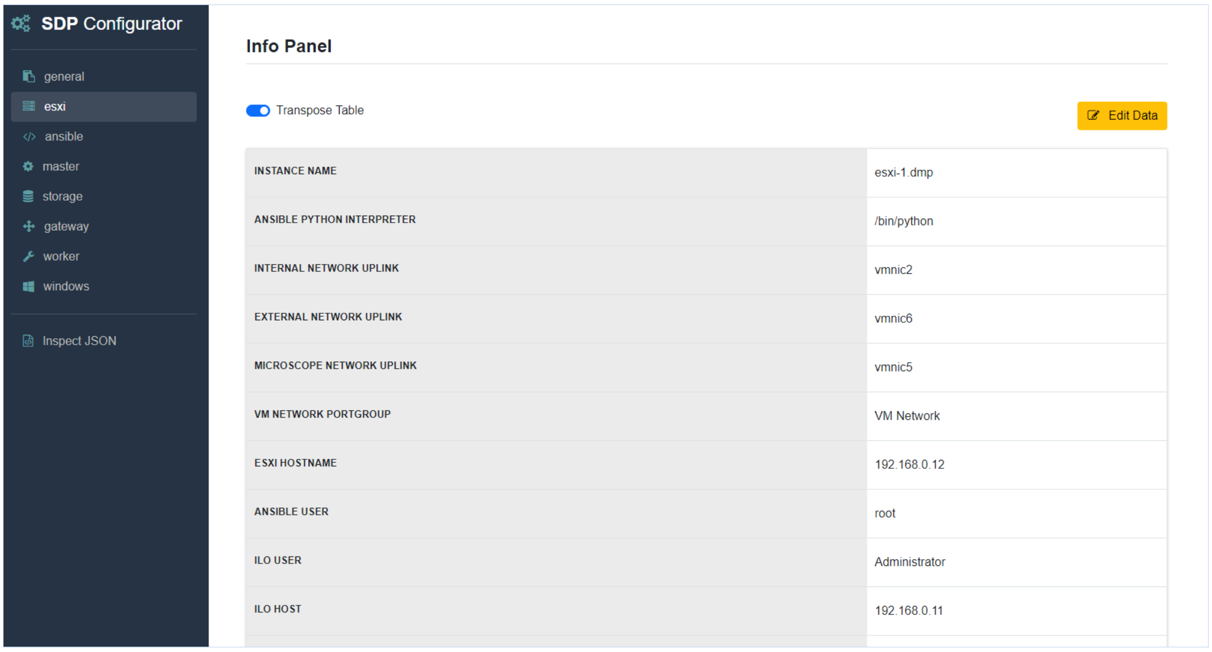


An **inventory file** is already exist! Choose options below to proceed:

Figure 27. SCA home page for update

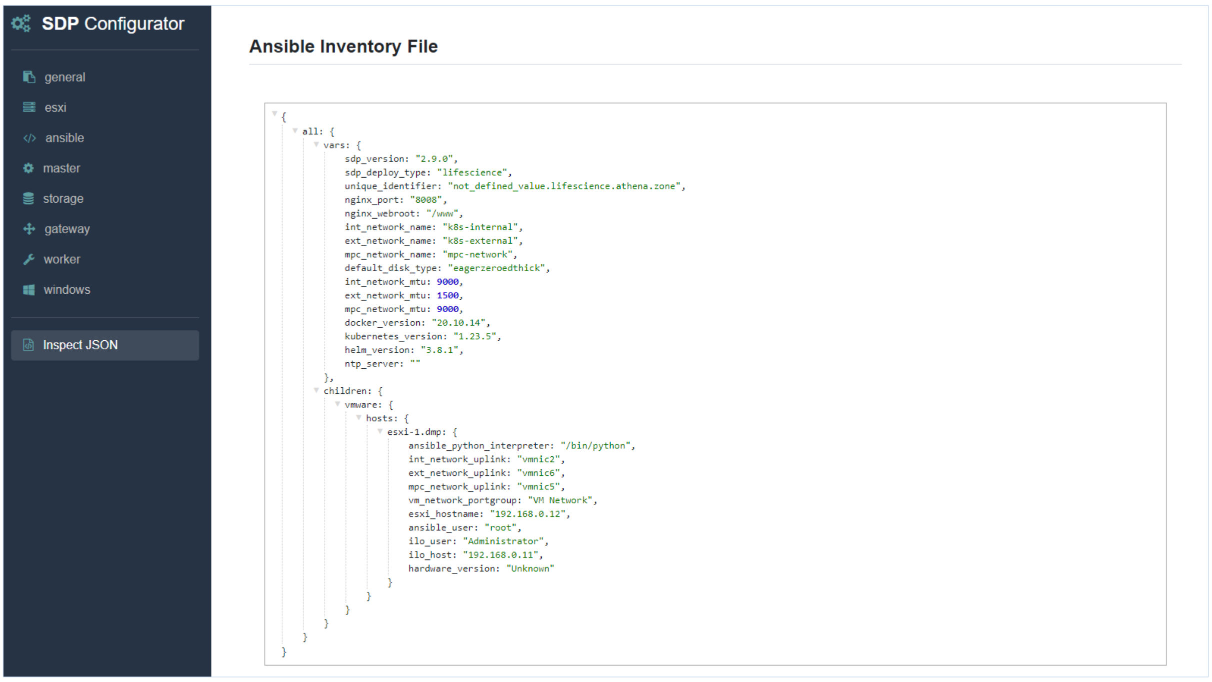
6.3.3. Visualizing Ansible Inventory

SCA presents the generated Ansible inventory file on the visualization page once the wizard step form reaches the last step. The goal of the visualization page is to illustrate the generated Ansible inventory file. This file was used to configure the infrastructure by executing the Ansible Playbook [15]. We developed two visualization options. One is in a table view as shown in Figure 28. The other one is in a tree view as depicted in Figure 29.



INSTANCE NAME	esxi-1.dmp
ANSIBLE PYTHON INTERPRETER	/bin/python
INTERNAL NETWORK UPLINK	vmnic2
EXTERNAL NETWORK UPLINK	vmnic6
MICROSCOPE NETWORK UPLINK	vmnic5
VM NETWORK PORTGROUP	VM Network
ESXI HOSTNAME	192.168.0.12
ANSIBLE USER	root
ILO USER	Administrator
ILO HOST	192.168.0.11

Figure 28. Inventory visualization in table view



```
{
    all: {
        vars: {
            sdp_version: "2.9.0",
            sdp_deploy_type: "lifescience",
            unique_identifier: "not_defined_value.lifescience.athena.zone",
            nginx_port: "8008",
            nginx_webroot: "/www",
            int_network_name: "k8s-internal",
            ext_network_name: "k8s-external",
            mpc_network_name: "mpc-network",
            default_disk_type: "eagerzeroedthick",
            int_network_mtu: 9000,
            ext_network_mtu: 1500,
            mpc_network_mtu: 9000,
            docker_version: "20.10.14",
            kubernetes_version: "1.23.5",
            helm_version: "3.8.1",
            ntp_server: ""
        },
        children: {
            vmware: {
                hosts: {
                    esxi-1.dmp: {
                        ansible_python_interpreter: "/bin/python",
                        int_network_uplink: "vmnic2",
                        ext_network_uplink: "vmnic6",
                        mpc_network_uplink: "vmnic5",
                        vm_network_portgroup: "VM Network",
                        esxi_hostname: "192.168.0.12",
                        ansible_user: "root",
                        ilo_user: "Administrator",
                        ilo_host: "192.168.0.11",
                        hardware_version: "Unknown"
                    }
                }
            }
        }
    }
}
```

Figure 29. Inventory visualization in a tree view

6.3.4. Configuring Additional Devices

SCA has additional features grouped in the Quick Action menu. This menu provides a particular use case that is frequently conducted by the Service Engineer, such as configuring Non-Volatile Memory Express (NVMe) storage for worker and storage nodes and updating the IP addresses of the gateway and storage nodes. As discussed with the stakeholders, a more specific use case might be added to the quick action menu to help the Service Engineer easily navigate to support the customer. Figure 30 illustrates the SCA quick action menu.

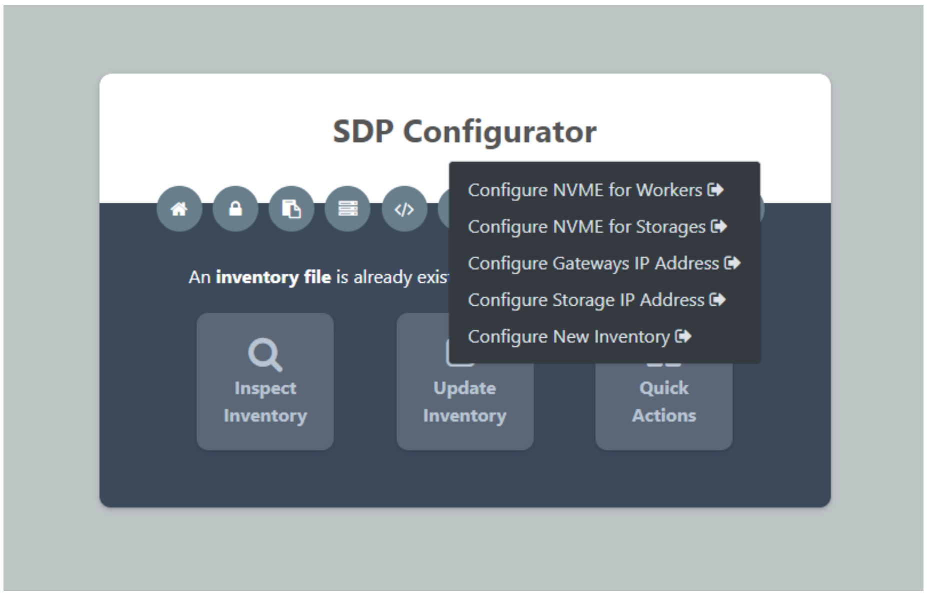
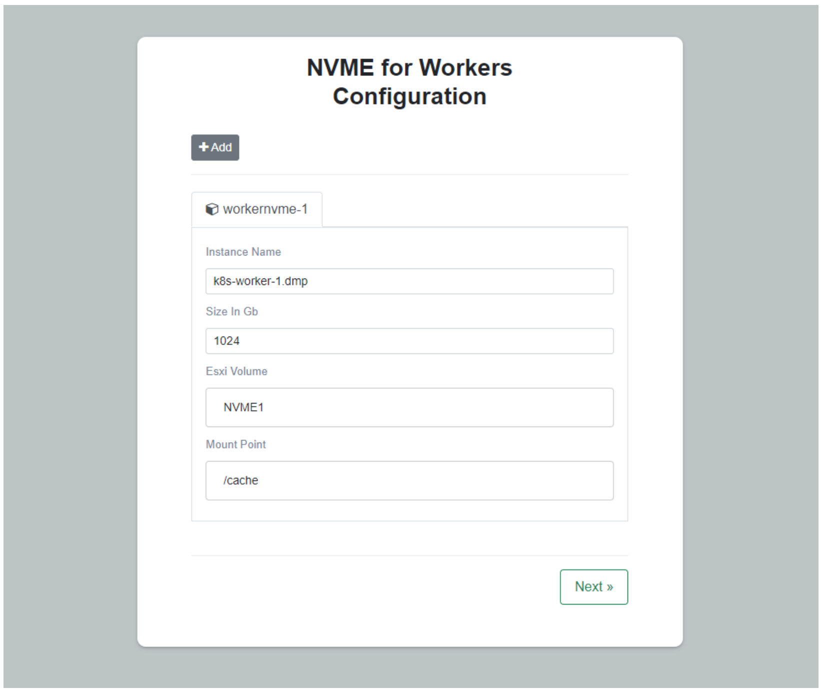

SDP Configurator
An inventory file is already exis
Configure NVME for Workers
Configure NVME for Storages
Configure Gateways IP Address
Configure Storage IP Address
Configure New Inventory
Inspect Inventory
Update Inventory
Quick Actions

Figure 30. SCA quick action menu

Figure 31 shows the NVMe storage configuration for the worker nodes. This form can be accessed through the quick action menu on the SCA home page.


NVME for Workers Configuration

+Add

workernvme-1

Instance Name: k8s-worker-1.dmp

Size In Gb: 1024

Esxi Volume: NVME1

Mount Point: /cache

Next »

Figure 31. NVME configuration for workers

6.4. Third-Party Components

The project was developed using the Python programming language. We structured the project in such a way as to improve the modularity of the system. The logical representation of the system can be found in Chapter 5 (Software Architecture and Design).

Flask is a microframework with a minimal library needed to build a web application. Therefore, we needed to install additional libraries to satisfy this project's requirements and use cases. Table 4 provides information about the additional library of this project.

Table 4. Python libraries used in the project

Library Name	Description
Flask-WTF	Simple integration of Flask and WTForms. This library is used to generate the HTML form dynamically [16].
Pydantic	Pydantic offers data validation and setting management for the domain model. Pydantic also has some features specific to this project, such as export to JSON, field aliases, and field validation [17].
Pylint	Pylint is a static code analyzer. ThermoFisher Scientific requires any software developed in Python to convey the flake8 code style. Therefore, Pylint is one of the best choices [18].
Toolz	Toolz is a set of utility functions for iterators, functions, and dictionaries. We used Toolz library to make a query from a dictionary datatype. Toolz is a powerful utility library to support this functionality of the SCA [19].
Pyvmomi	Pyvmomi is the Python SDK for the VMware vSphere API that allows to manage ESX, ESXi, and vCenter. The SCA used Pyvmomi to acquire information from the ESXi, such as the hardware model and the available storage Drive ID [20].
Paramiko	Paramiko library enables SSH calls from the SCA Python code. SSH is used to establish a connection to the remote host, such as the k8s-ansible VMWare instance to acquire the available MAC addresses [21].

6.5. Continues Integration (CI)

We employed Gitlab development infrastructure for source code repository, configuration management, and test automation. We developed scripts that will get triggered to build and tests the code automatically when the code is pushed to Gitlab. This practice is known as Continuous Integration (CI) [22]. The process starts when the developers push the code to the repository. This action triggers the prepared scripts, such as building the docker image and running a set of test scenarios. The automation jobs were divided into stages: static analysis, unit testing, and functional testing. We considered separating the stages because we have a different strategies to test the code on a different level.

Whenever the tests are completed, the SCA Docker image is then published to the ThermoFisher Scientific Docker image artifactory. However, if the job failed due to an unsuccessful test, the developer was notified through email. The flow of this process is illustrated in Figure 32.

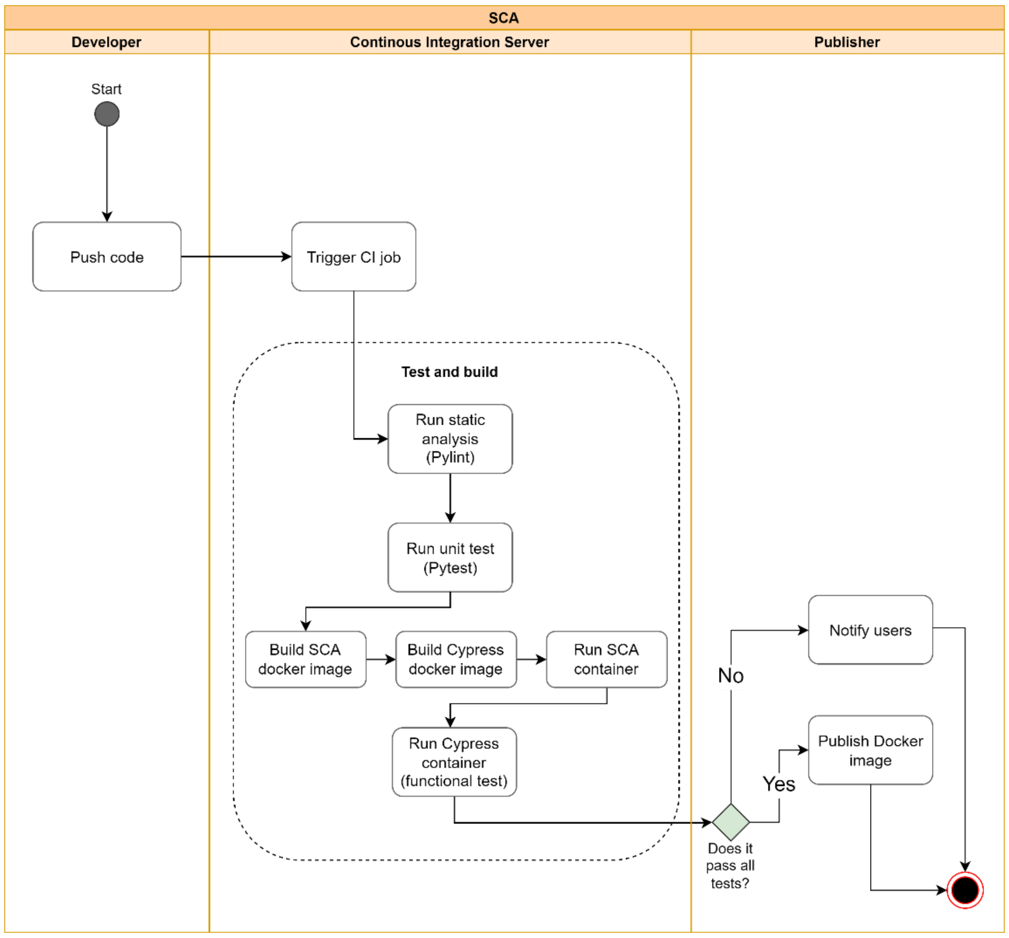


Figure 32. SCA continuous integration activity diagram

6.6. SCA Docker Container

During the SCA development process, it was required to run SCA in several environments, for instance, the developer's local computer and testing environment during the CI process on Gitlab. However, several dependencies were required to install before we were able to run SCA. Dealing with this situation, we chose Docker technology to pack the SCA and its libraries. The application's code and dependency/library are packed into a standard Docker image format [23]. This Docker image is executable into a Docker container, which is lightweight and self-contained that preserves everything needed to run the application, including libraries, system tools, code, and runtime. Since all requirements needed to run SCA are packaged together, we could run the SCA in any environment regardless of the installed library or tools in that environment.

We used Docker to package the SCA's code and dependency into a Docker image. The SCA Docker image is built in the CI pipeline and then pushed to the ThermoFisher Scientific image registry. In addition, we used Docker to run the functional testing and the VCenter Simulator (VCSim) to simulate the ESXi server. During testing, SCA is tested against this simulator to avoid external dependency, such as connecting to an external server.

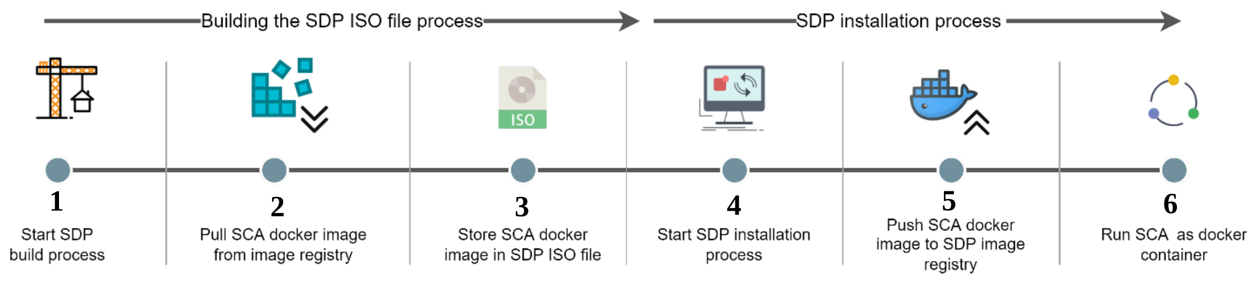

Building the SDP ISO file process
SDP installation process
ISO
1
Start SDP build process
2
Pull SCA docker image from image registry
3
Store SCA docker image in SDP ISO file
4
Start SDP installation process
5
Push SCA docker image to SDP image registry
6
Run SCA as docker container

Figure 33. High-level overview on how SCA is being part of the SDP ISO file release

Figure 33 shows the process of how SCA will be added to the SDP ISO file during the SDP build process. This process illustrates the power of Docker for shipping the SCA with less additional library needed.

When the SDP build job is triggered, it starts the process such as pulling the necessary packages and packing them into the ISO file (steps 1 and 2 in Figure 33). This build process also pulls the SCA image stored in the ThermoFisher Scientific image registry that was built and published during the SCA’s CI process (see Section 6.5). Then, the software packages including the SCA docker image are packed into an SDP ISO file (step 3 Figure 33).

Up to this step, the SCA is part of the SDP bundled in the ISO file. We could technically run SCA as a Docker container once the Docker daemon is installed. The next step is when the Service Engineers start the SDP installation using the SDP ISO file. The script will push the SCA image from the ISO file into the SDP image registry (steps 4 and 5 in Figure 33). Then, by having the SCA image in the SDP image registry, it can be run as a docker container to continue the process of generating inventory (step 6 in Figure 33).

7. Verification and Validation

Verification is a process of evaluating a system or component to determine whether the software satisfies the defined requirements [24]. In this project, we verified and validated the different levels of deliverables with different strategies. Table 5 provides information on a different levels of the artifacts and the strategies we used to perform the tests.

Table 5. Verifiable artifacts and the test strategy

Artifacts	Testing Strategy
Source code	Static analysis using Flake8 for checking the code base against standard Python coding style (PEP8) [25] [26].
Functions and classes	Unit test using Pytest library [27].
System functionalities	Functional test using Cypress with Cucumber plugin [28] [29].
Business requirements	Live demonstration with the stakeholders during weekly and monthly progress meetings. The received feedback was added to the next development cycle.

7.1. Static Analysis

Static analysis is a way to validate the source code whether it complies with the standard coding pattern of a certain programming language. In this project, we used Python programming language and leveraged PEP8 [26] as a standard reference. Our goal in conducting static analysis was to analyze the code so that we could detect vulnerabilities, detect bugs in the early stages, improve the code quality, and improve the consistency of the code.

We investigated several libraries as a candidate technology during the project, for instance, Pylint, Flake8, Black, and Mypy. Based on the discussion with the SDP team, ThermoFisher Scientific uses Flake8 as the standard library used specifically for the Python project, which is also adopted in this project.

We added a static analysis stage in our CI pipeline to make sure the code satisfied the company Python code styling standard before proceeding to the further step. We explain the test automation and CI pipeline in Section 7.4 of this chapter.

7.2. Unit Testing

Unit testing is a way to validate the smallest piece of the system. In the context of this project, we used a unit testing strategy to test the functions and classes. As mentioned in Chapter 6 (Implementation), we implemented the project in a modular structure such as separating the HTTP layer with Flask and the service layer. This approach made the testing part easier since the code was isolated based on its main behavior, for instance, there is a separate module to handle the VMWare configuration and Storage configuration.

During the project, we used the Pytest library for unit testing. We decided to use Pytest because it is a mature testing framework with good documentation. We considered writing tests in Pytest was also relatively simpler because it follows a functional programming style. We developed several test scenarios associated to its module as shown in Figure 34.

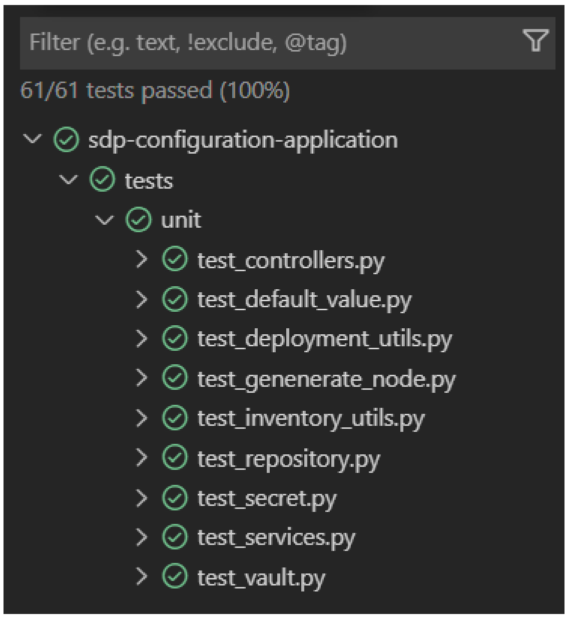

Filter (e.g. text, !exclude, @tag)
61/61 tests passed (100%)
sdp-configuration-application
tests
unit
test_controllers.py
test_default_value.py
test_deployment_utils.py
test_genenerate_node.py
test_inventory_utils.py
test_repository.py
test_secret.py
test_services.py
test_vault.py

Figure 34. Unit test scenarios implemented in Pytest

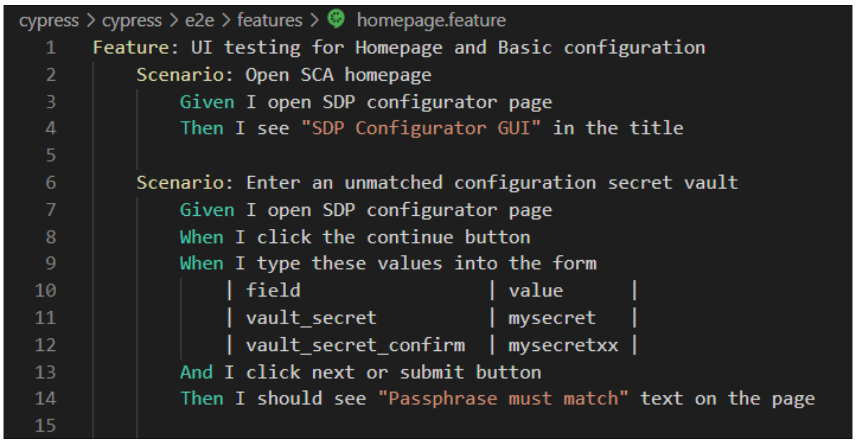
The test is fully automated once we pushed the code to the Gitlab repository, we configured a set of scripts executed by the Gitlab runner in the *gitlab-ci.yaml* file. The explanation about the test automation is in Section 7.4.

7.3. Functional testing

The goal of the project is to create a visual interface for the less-trained engineer to interact with the SDP installation. We wanted to verify each User Interface (UI) and its functionality; we developed several functional test scenarios to make sure the functionality of SCA works as intended. For automated functional testing, we adopted the Cypress UI technology as the SDP team already has good experience with the technology. We explained how we set up and run Cypress in the following section.

7.3.1. Setup Cypress

Cypress is an open-source UI testing framework. The unique characteristic of Cypress was that we were able to have a plugin called Cucumber. Cucumber is a syntax library so we were able to use Gherkin syntax for writing the test. Based on [30], writing tests is considered an expensive activity in terms of time and effort. However, after configuring the test with Cucumber using the Gherkin syntax, we could reduce the complexity of defining the test scenario. Figure 35 shows how we specify the test in Gherkin syntax of the Cucumber library.



cypress > cypress > e2e > features > homepage.feature

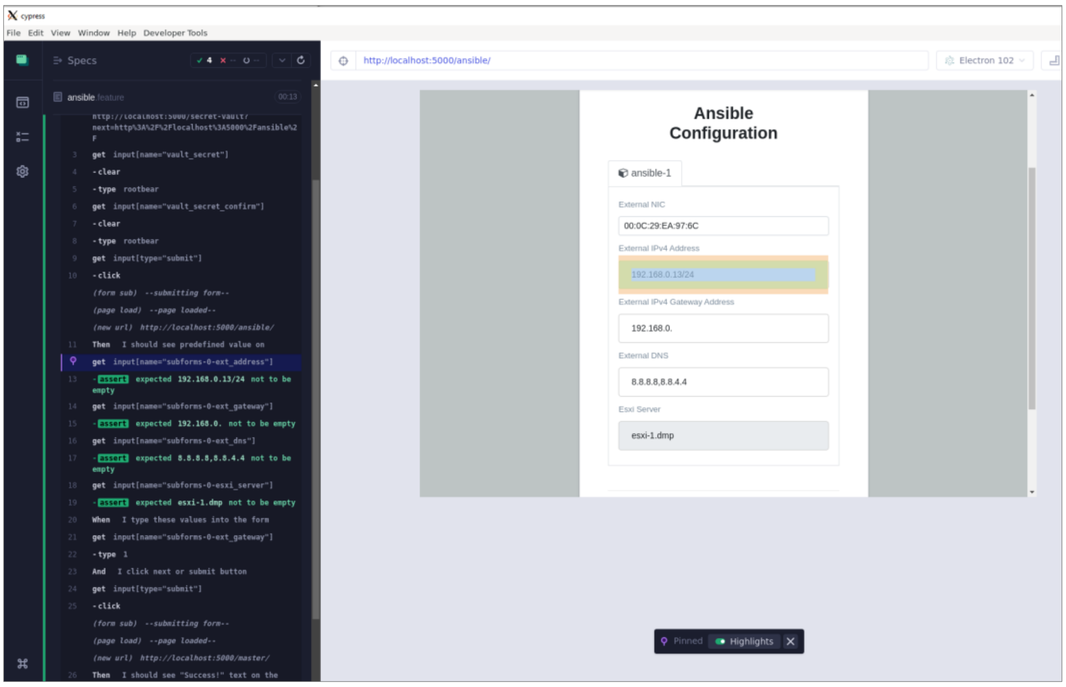
```
Feature: UI testing for Homepage and Basic configuration
    Scenario: Open SCA homepage
        Given I open SDP configurator page
        Then I see "SDP Configurator GUI" in the title

    Scenario: Enter an unmatched configuration secret vault
        Given I open SDP configurator page
        When I click the continue button
        When I type these values into the form
            | field               | value       |
            | vault_secret        | mysecret    |
            | vault_secret_confirm | mysecretxx |
        And I click next or submit button
        Then I should see "Passphrase must match" text on the page

```

Figure 35. Example of test scenario written in Cucumber Gherkin syntax

In Cypress, we separated the test as a Feature. A Feature contains one or multiple Scenarios. In the Scenarios, we defined the step to test the functionality of the application. For instance, there was a scenario to test the worker configuration. When the test was executed, it actually accessed the SCA's worker configuration page, filled in the necessary information for the worker, and submitted the data into the inventory file. This process's steps were defined in the test scenario and fully automated with no or limited human involvement.

7.3.2. Running Cypress

Cypress can be executed either with the Cypress web portal or headless run for automation purposes. When running Cypress in headless mode, it runs the test scenario in the background and only shows the result as standard output on the CLI. For local testing purposes, meaning that the Cypress was executed on the developer's local PC, we used the Cypress web portal. The Graphical User Interface of Cypress is shown in Figure 36.



cypress
File Edit View Window Help Developer Tools
Specs
✓ 4 ✕ --
ansible.feature
00:13
http://localhost:5000/ansible/
Electron 102
Pinned
Highlights

```
http://localhost:5000/secret-vault?next=http%3A%2F%2Flocalhost%3A5000%2Fansible%2F
get  input[name="vault_secret"]
-clear
-type  rootbear
get  input[name="vault_secret_confirm"]
-clear
-type  rootbear
get  input[type="submit"]
-click
(form sub)  --submitting form--
(page load)  --page loaded--
(new url)  http://localhost:5000/ansible/
Then  I should see predefined value on
get  input[name="subforms-0-ext_address"]
-assert  expected 192.168.0.13/24 not to be empty
get  input[name="subforms-0-ext_gateway"]
-assert  expected 192.168.0. not to be empty
get  input[name="subforms-0-ext_dns"]
-assert  expected 8.8.8.8,8.8.4.4 not to be empty
get  input[name="subforms-0-esxi_server"]
-assert  expected esxi-1.dmp not to be empty
When  I type these values into the form
get  input[name="subforms-0-ext_gateway"]
-type  1
And  I click next or submit button
get  input[type="submit"]
-click
(form sub)  --submitting form--
(page load)  --page loaded--
(new url)  http://localhost:5000/master/
Then  I should see "Success!" text on the
```

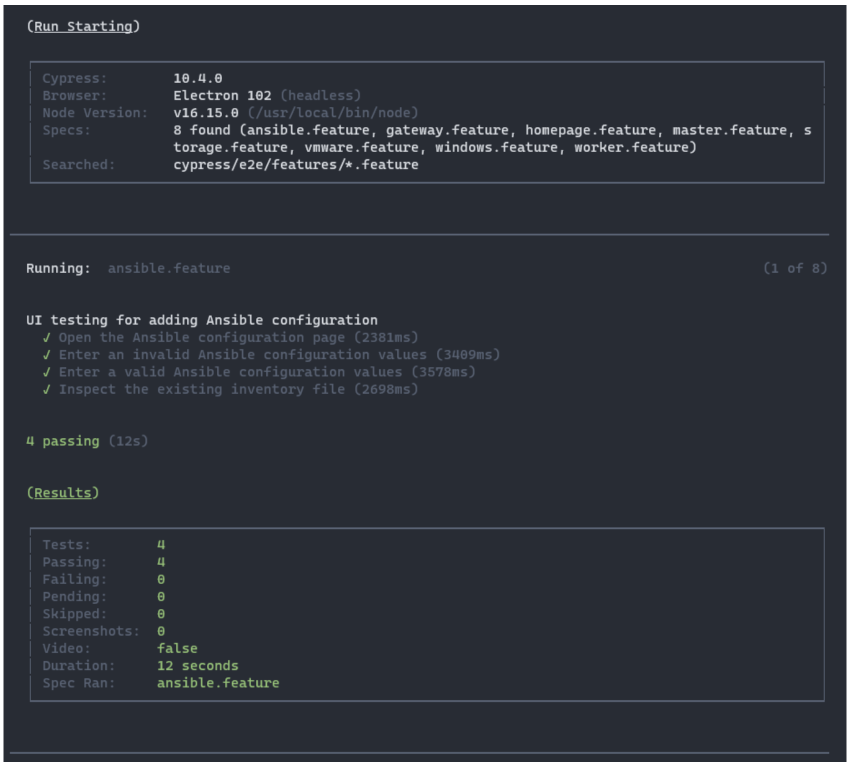
Ansible Configuration

ansible-1

External NIC: 00:0C:29:EA:97:6C
External IPv4 Address: 192.168.0.13/24
External IPv4 Gateway Address: 192.168.0.
External DNS: 8.8.8.8,8.8.4.4
Esxi Server: esxi-1.dmp

Figure 36. Cypress running in an interactive mode

For automation purposes, we run Cypress in a headless mode in the Gitlab CI. If we run Cypress in headless mode, Cypress only exports the test result as standard output on the console. Figure 37 shows how Cypress runs in a headless mode.



```
(Run Starting)

┌────────────────────────────────────────────────────────────────────────────────────────────────┐
│ Cypress:        10.4.0                                                                         │
│ Browser:        Electron 102 (headless)                                                        │
│ Node Version:   v16.15.0 (/usr/local/bin/node)                                                 │
│ Specs:          8 found (ansible.feature, gateway.feature, homepage.feature, master.feature, s │
│                 torage.feature, vmware.feature, windows.feature, worker.feature)               │
│ Searched:       cypress/e2e/features/*.feature                                                 │
└────────────────────────────────────────────────────────────────────────────────────────────────┘


────────────────────────────────────────────────────────────────────────────────────────────────────

Running:  ansible.feature                                                                  (1 of 8)


UI testing for adding Ansible configuration
  ✓ Open the Ansible configuration page (2381ms)
  ✓ Enter an invalid Ansible configuration values (3409ms)
  ✓ Enter a valid Ansible configuration values (3578ms)
  ✓ Inspect the existing inventory file (2698ms)


4 passing (12s)


(Results)

┌────────────────────────────────────────────────────────────────────────────────────────────────┐
│ Tests:        4                                                                                │
│ Passing:      4                                                                                │
│ Failing:      0                                                                                │
│ Pending:      0                                                                                │
│ Skipped:      0                                                                                │
│ Screenshots:  0                                                                                │
│ Video:        false                                                                            │
│ Duration:     12 seconds                                                                       │
│ Spec Ran:     ansible.feature                                                                  │
└────────────────────────────────────────────────────────────────────────────────────────────────┘
```

Figure 37. Cypress running in a headless mode

7.4. Test Automation

Testing is considered a very expensive process and consumes one-third to one-half of the cost of a project [30]. Also, the source code was continuously changed during the project and needed to be verified over time. To tackle these concerns, we configured a test automation pipeline using Gitlab CI/CD. In Figure 38, we set up three stages in the Gitlab CI/CD pipeline based on the testing strategies we implemented.

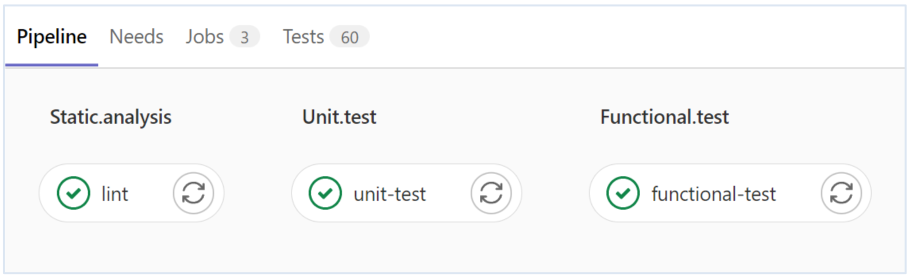


Figure 38. Pipeline stages on Gitlab CI/CD

The overview of the CI pipeline is described in Section 6.5. In Figure Figure 39, we described the detail of the functional testing stage. Whenever the static analysis and unit testing had been passed, the next step was executing the functional testing. In the functional testing stage, we built the Docker images: The VCSim Docker image, which is the VCenter/ESXi simulator, the SCA Docker image, and the

Cypress Docker image. Technically we could execute the Cypress test without building the image. However, we decided to build the Docker image because we wanted to avoid unnecessary dependency that might cause problems when executing the pipeline.

Once we have the Docker images, first we run the container of the VCSim, the SCA, and the Cypress. The Cypress container executed the test against the running SCA container through HTTP on port 5000. There was also a test scenario where the SCA was required to access data to the ESXi, here we used the simulator so that the SCA container had no dependency on an external ESXi server but the simulator.

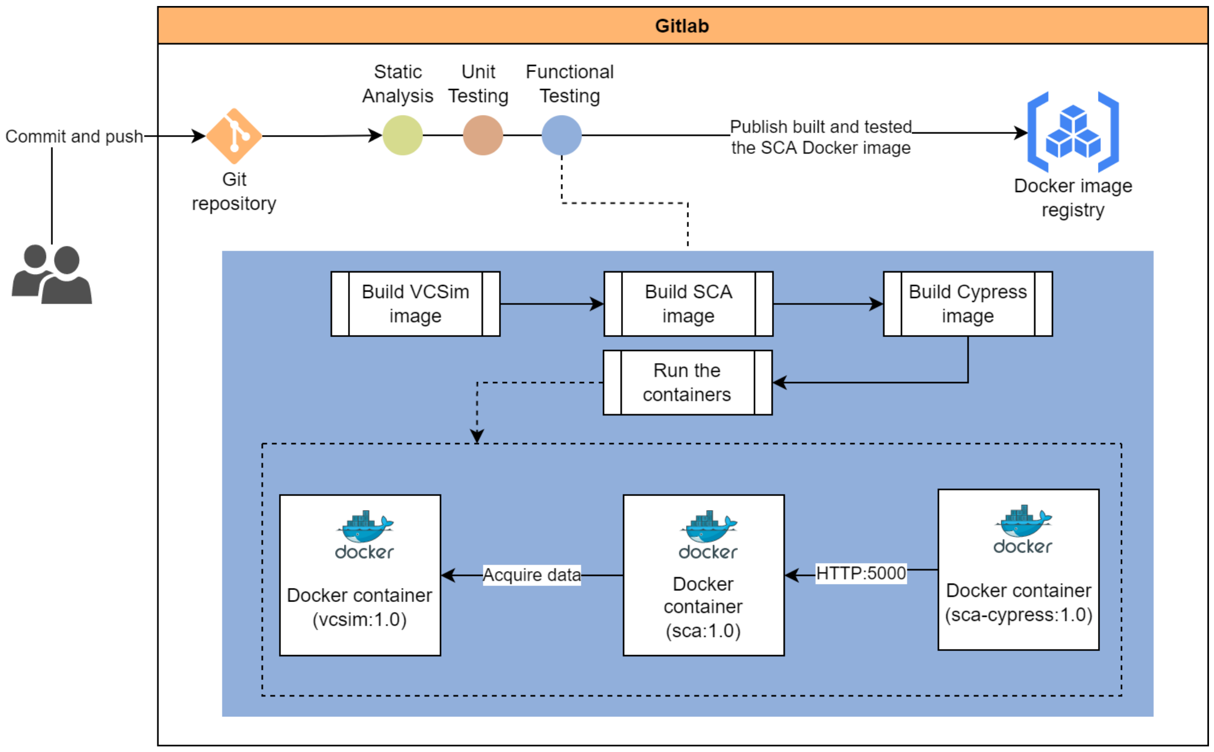


Figure 39. The detailed overview of the functional testing stage

7.5. Stakeholder Validation

We utilize weekly meetings and monthly meetings with the stakeholders to inform the progress of the project. During the meeting, we also had demo sessions with the stakeholders to show the latest version of the prototype. Thus, the SCA was validated continuously, and the received feedback was added to the next development cycle. Apart from the live demo during meeting, we deployed the SCA in the development and testing environment and shared the access to the users. Therefore, the users had more flexibility to interact with the SCA. They accessed and explore the features to see how the SCA could fulfill their needs and provided a valuable feedback to improve the SCA.

Lastly, we conducted a final check with the users, which is the Service Organization team. During the session, we demonstrated the latest version of the SCA and continued with exploratory testing conducted by the users to validate the features implemented in SCA against the requirements and the user's expectations. We provided a survey to the users to see their opinion about the prototype. The result of the survey is described in Table 6. The score ranges from one to five with the higher score showing a better result. The score indicates how confident the users are in terms of easy-to-use, usability, and the UI layout of the SCA.

Table 6. Summary of the survey result

#	Name	Easy-to-use (1 - 5)	Usability (1-5)	UI Layout (1 – 5)	Overall Experience
1	Kieran Ham	5	4	5	Very positive, it's a big improvement for somebody who is not as familiar with terminal.
2	Ataur Rahman	5	5	5	Easy to use and navigate. The visualization at the end is nice and also provides capability to edit the whole configuration if needed.
3	Gabriel Lee	5	5	5	easy to use and good to have additional feature for specific modification items, things like...modifying customer IP...etc.
4	Jordy Plug	4	5	5	Good and smooth experience. Easy to understand and work with.

From the survey result we concluded that the SCA was able to solve the problem that occurs due to the limitation of a CLI-based application. The SCA provides features that the users can confidently operate to configure the SDP. The SCA was implemented with a clear and simple UI layout, and considered easy to operate by the lesser experience engineers that were not familiar working with CLI-based application.

8.Project Management

8.1. Way of Working

The project was conducted in ten months, from January to October 2022. We divided the project period into four phases: Planning and Familiarization, Design and Implementation I, Design and Implementation II, and Finalization.

The project was managed using the Scrum approach, the progress of reaching the goal constantly relies on feedback. In addition to the Scrum process, we scheduled weekly progress and retrospective meeting with the company supervisor to keep track of project progress in a short-term period. Additionally, we conducted two regular meetings every month. The first one was held with the Project Steering Group (PSG) member, which discussed the high-level plan and progress of the project. The other one was a monthly progress meeting with the stakeholders including the potential user from the Service Organization. Nevertheless, to maintain the collaboration with the SDP team, we planned an ad-hoc meeting when needed.

The artifacts and the events of Scrum are mentioned in the next sub-section.

8.1.1. Scrum Artefact

- User story was used to define every requirement.
- All user stories were kept in the backlogs.
- Each user story was estimated using story points based on its difficulty.
- Each sprint had sprint goals and define during the sprint planning.
- Each user story had a proper, clean, clear, and concise description.

8.1.2. Scrum Events

- The project was executed in sprints, each lasting two weeks, consisting of ten working days.
- Sprint planning was conducted prior to starting each sprint.
- The story point of the user story was estimated during the sprint planning.
- The priority of the user story was estimated during the spring planning.
- The goals of the sprint, if applicable, also the deliverables of the sprint were defined during the sprint planning.
- Progress report with company supervisor was conducted in a regular weekly meeting.
- Progress report with stakeholders was conducted in a regular monthly meeting.

8.2. Work Breakdown Structure

To help the planning process, we structured the deliverable description from the higher to the lower level in the Work Breakdown Structure (WBS). Figure 40 shows the WBS used during the project to provide the detailed deliverable items of each deliverable category.

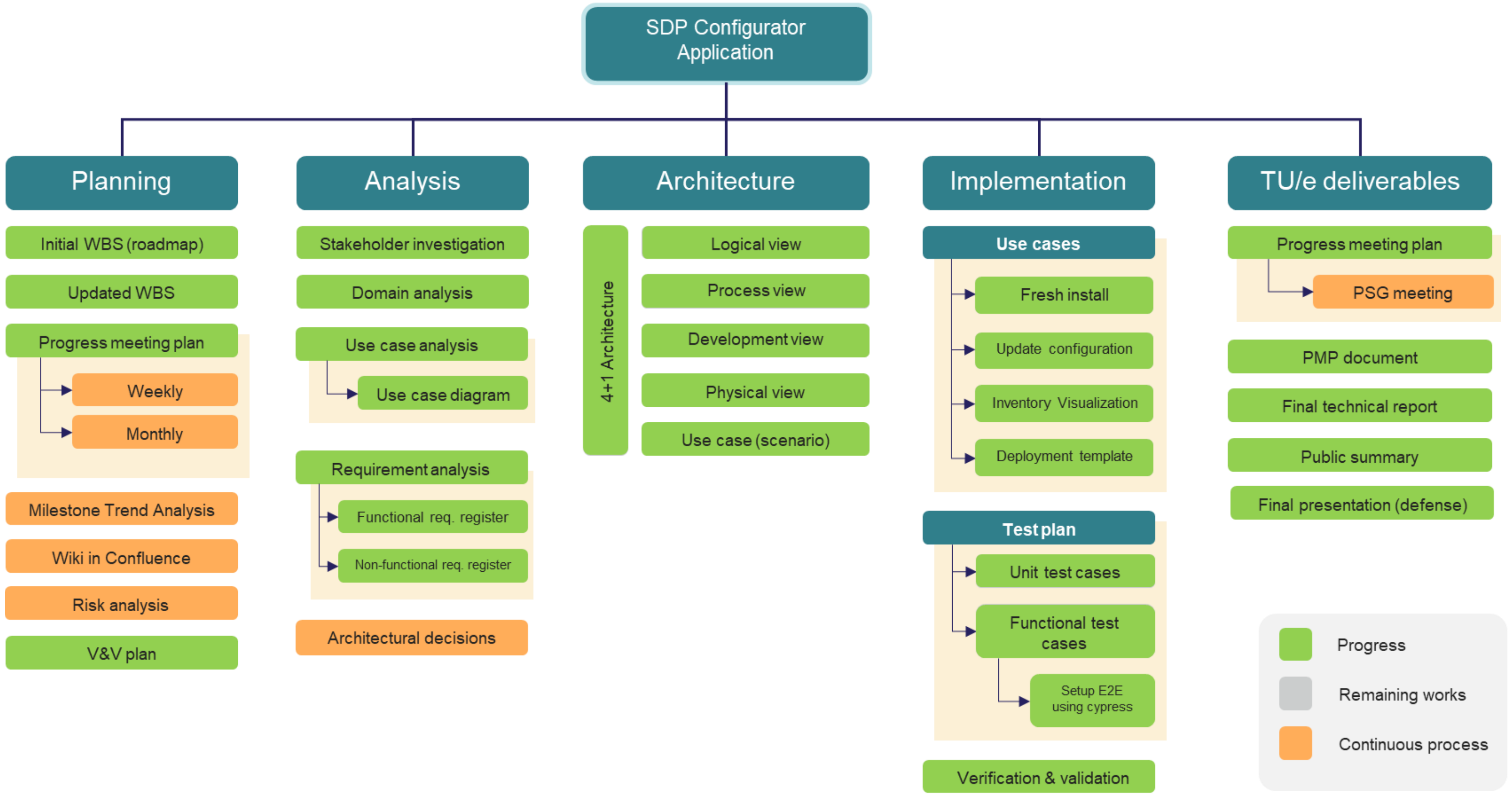


Figure 40. Work Breakdown Structure

8.3. Milestone Trend Analysis

We listed the deliverable in the Work Breakdown Structure to provide a better insight into what we were planning to deliver. However, to understand the timeline on when a certain deliverable needed to be delivered, we were using Milestone Trend Analysis (MTA). MTA is a method to keep track of milestones weekly. With MTA, we could detect a potential delay that might affect the progress of the project. Figure 41 shows the MTA used in this project.

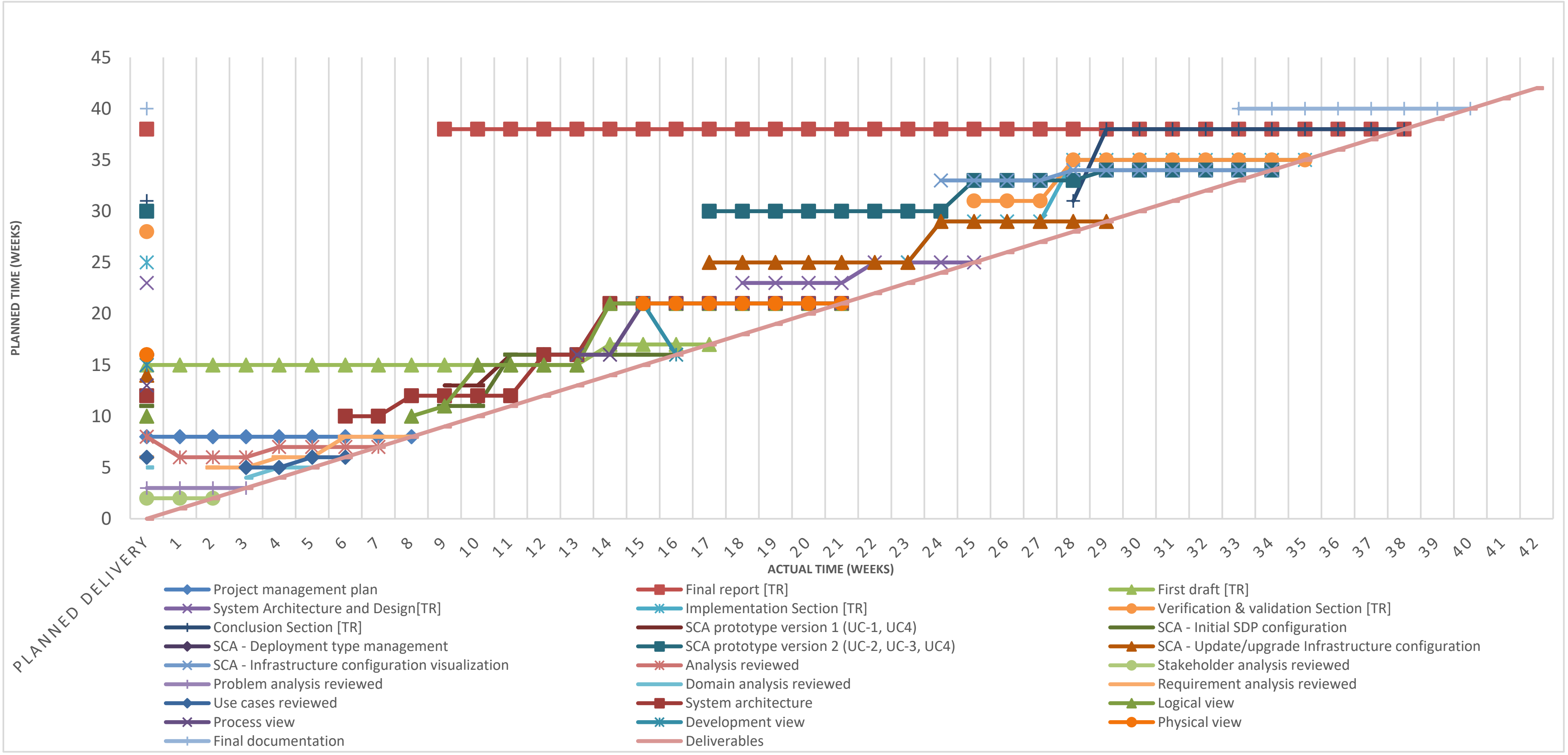


Figure 41. Milestone Trend Analysis. TR stands for Technical Report and UC stands for Use Case

8.4. Infrastructure Plan

During the project, ThermoFisher Scientific provided environments to support the project, such as a Gitlab repository, Development server, and Docker image registry. There were also several tools we used to manage the project, for instance, Confluence, Jira, MS Teams, and MS OneDrive.

All the project-related documents were stored in Confluence in the ThermoFisher Scientific server. The infrastructure used for this project is the following:

- **Confluence** was used to store the project documentation.
- **Jira** was used to monitor the SCRUM board and maintain backlog, which is hosted in ThermoFisher Scientific sever.
- **MS Teams** was used to communicate with the stakeholders and the coaches of this project.
- **MS OneDrive** was used to store the project artifact such as documents, presentation files, and notes. Some important documents were also stored/linked to Confluence page.
- **VMware ESXi** was used for the development and testing environment.

The tools used to execute this project are:

- **WSL2** was a Linux environment hosted on the local computer.
- **Visual studio code** was used to write the code of the chosen programming language.
- **Gitlab** was used to store the codebase.
- **Gitlab CI/CD** was used to make the automation pipeline.
- **Docker** was used as a local isolated environment to run the app in the development mode.

8.5. Risk Management

We keep track of the potential risk in a risk register. Table 7 shows an example of how we structured the risk register in a table format. The *ID* column is the risk identifier. The *Category* field contains the category of the risk, for instance, *Domain knowledge* means that the risk is related to understanding the domain knowledge of the system. The *Risk* column describes the detailed description of the risk. Columns *L*, *I*, and *P* are *Likelihood*, *Impact*, and *Priority* respectively. The value of columns L and I range from 1 to 5 which indicate the severity of a risk. The higher the number the more priority the risk had. To calculate the priority score, we multiplied the Likelihood and the Impact. We used color-coded ranging from green, yellow, and red to easily notice the highest priority risk.

We discussed the newly identified risk and the mitigation plan during the PSG meeting to gain feedback. The complete list of risk registers can be found in Appendix E.

Table 7. Risk Register

ID	Category	Status	Risk	L	I	P	Mitigation
R1	Domain knowledge	Mitigated	A steep learning curve of the current system	4	4	16 🟨	• Define the system of interest • Define the system boundary • Set priority

9.Conclusions

9.1. Summary

ThermoFisher Scientific is shifting into a managed services business model. When the customer purchased an EM, ThermoFisher Scientific continuously provides services to maintain the EM and its software up and running. With an Infrastructure-as-a-Service named SDP, ThermoFisher provides highly-configurable yet adjustable services according to the customers' needs. The SDP helps ThermoFisher Scientific delivers the related software to its customer seamlessly. Currently, the Service Engineers configure the SDP on the customer site using a CLI-based application. Based on the explanation in Chapter 3 – Problem Analysis, we concluded that there is a need to visualize the SDP configurator so that broadens the opportunity for the lesser experienced Service Engineers to configure the SDP.

We investigated several technologies to develop the system. The decision upon this investigation was also reviewed together with the SDP team. The comparative study result can be found in Appendix D. As a result, we designed and implemented a web-based application called SDP Configurator Application (SCA). The features of SCA (see Chapter 6 – Implementation), allow the users to configure SDP, which is a highly-configurable Infrastructure-as-a-Service for a fresh install, update/upgrade the existing configuration, visualize the existing configuration, and allow the system to pre-defined the value in a deployment template. These features fulfilled the stakeholders' needs based on the use cases that are explained in Section 5.2 – Use Cases.

To make sure that we verify the system against the software development standard and satisfied the NFR-3 – Testability, we implemented automatic testing explained in Chapter 7 – Verification and Validation. In addition, we deployed the SCA on the ThermoFisher Scientific environment and shared the access with the potential users. The users investigated and explored the features offered by the SCA. In addition, the users provided positive feedback through the prepared survey that the delivered prototype was easy-to-use and guided the lesser experienced engineer to configure the SDP.

9.2. Recommendation and Future Work

The SDP Configurator Application (SCA) offers the visualization to configure the SDP. With respect to improving the SCA, we recommend the following items to be considered in the future:

9.2.1. Integration

We implemented the SCA and its Continuous Integration pipeline. The stage in the pipeline includes the static analysis, the unit testing, the functional testing, and the building and publishing the docker image to ThermoFisher Scientific's Docker image registry. However, Continuous Deployment is not implemented in this project. Therefore, we recommend integrating the SCA with the SDP release mechanism so that the SCA can be accessible by the Service Engineers on the customer site.

9.2.2. Ansible Playbook Execution

The Service Engineers can use the SCA to configure the SDP by generating the Ansible inventory file. However, the installation of the configured item is conducted by executing the Ansible playbook (see

Section 3.4 – Domain Analysis). This process is carried out by executing the command on the CLI application. Therefore, there is a manual step to switch from the web browser to the CLI-based application to apply the configuration. We recommend adding a feature to execute and monitor the Ansible playbook execution so that the installation of the SDP can be monitored through the SCA in a web browser.

Bibliography

[1] Thermofisher Scientific, "Thermofisher Scientific - Brands.," [Online]. Available: https://www.thermofisher.com/nl/en/home/brands.html. [Accessed 02 March 2022].

[2] V. A. Pezeshkian, "Designing a Solution for monitoring and managing multi-cloud on-premise deployments," Eindhoven University of Technology, 2020.

[3] ThermoFisher Scientific, "High-throughput cryo-EM epitope mapping of SARSCoV-2 spike protein antibodies using EPU Multigrid," [Online]. Available: https://assets.thermofisher.com/TFS-Assets/MSD/Reference-Materials/cryo-em-epitope-mapping-wp0031.pdf. [Accessed 24 August 2022].

[4] J. M. Tomasz, "Chapter 1 - Introducing Oracle Cloud Infrastructure," in *Practical Oracle Cloud Infrastructure : infrastructure as a service, autonomous database, managed Kubernetes, and serverless*, Berkeley, CA, Apress L.P, 2020, p. 27.

[5] A. S., "That! Company," [Online]. Available: https://www.thatcompany.com/why-is-knowing-the-command-line-important. [Accessed 01 August 2022].

[6] Red Hat Ansible, "Ansible Documentation," [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html. [Accessed 01 September 2022].

[7] G. d. A. Calheiros, "Designing an infrastructure for On-Premise Software Deployment," Eindhoven University of Technology, Eindhoven, 2019.

[8] Docker Inc., "Docker," [Online]. Available: https://www.docker.com/resources/what-container/. [Accessed 01 September 2022].

[9] Red Hat, Inc., "What is Ansible?," 2022. [Online]. Available: https://www.redhat.com/en/technologies/management/ansible/what-is-ansible. [Accessed 01 August 2022].

[10] IBM, "End-to-End Application Provisioning with Ansible and Terraform," [Online]. Available: https://www.ibm.com/cloud/blog/end-to-end-application-provisioning-with-ansible-and-terraform. [Accessed 01 September 2022].

[11] Red Hat, Inc, "What is an Ansible Playbook?," Redhat, 2022. [Online]. Available: https://www.redhat.com/en/topics/automation/what-is-an-ansible-playbook. [Accessed 01 August 2022].

[12] ProductPlan, "Product Plan," 2022. [Online]. Available: https://www.productplan.com/glossary/moscow-prioritization/. [Accessed 01 March 2022].

[13] P. B. Kruchten, "The 4+ 1 view model of architecture.," IEEE software 12.6, 1995.

[14] Figma, "Figma," [Online]. Available: https://www.figma.com/. [Accessed 01 March 2022].

[15] Red Hat, Inc., "What is Ansible Playbook?," 2022. [Online]. Available: https://www.redhat.com/en/topics/automation/what-is-an-ansible-playbook. [Accessed 01 August 2022].

[16] WTForms, "Flask-WTF Documentation," [Online]. Available: https://flask-wtf.readthedocs.io/en/1.0.x/. [Accessed 01 August 2022].

[17] Pydantic, "Pydantic Documentation," [Online]. Available: https://pydantic-docs.helpmanual.io/. [Accessed 01 August 2022].

[18] Logilab, PyCQA and contributors, "Pylint Documentation," [Online]. Available: https://pylint.pycqa.org/en/latest/. [Accessed 01 August 2022].

[19] M. Rocklin and J. Jacobsen, "Toolz Documentation," 2013. [Online]. Available: https://toolz.readthedocs.io/en/latest/. [Accessed 01 August 2022].

[20] VMware, "Pyvmomi Documentation," [Online]. Available: https://github.com/vmware/pyvmomi. [Accessed 01 August 2022].

[21] J. Forcier, "Paramiko Documentation," [Online]. Available: https://www.paramiko.org/. [Accessed 01 August 2022].

[22] Gitlab , "Gitlab Docs," [Online]. Available: https://docs.gitlab.com/ee/ci/introduction/index.html#continuous-integration. [Accessed 01 September 2022].

[23] Oracle Netherlands, "What is Docker?," 2022. [Online]. Available: https://www.oracle.com/nl/cloud/cloud-native/container-registry/what-is-docker/. [Accessed 01 August 2022].

[24] IEEE, "Standard Glossary of Software Engineering Terminology," 2001.

[25] I. S. Cordasco, "Flake8 Documentation," 2016. [Online]. Available: https://flake8.pycqa.org/en/latest/index.html. [Accessed 01 September 2022].

[26] G. van Rossum, B. Warsaw and N. Coghlan, "PEP 8 – Style Guide for Python Code," [Online]. Available: https://peps.python.org/pep-0008/. [Accessed 01 September 2022].

[27] Holger Krekel and pytest-dev team, "PyTest," 2015. [Online]. Available: https://docs.pytest.org/en/7.1.x/. [Accessed 01 September 2022].

[28] Cypress.io, "JavaScript and End-to-End Testing Framework," [Online]. Available: https://www.cypress.io/. [Accessed 01 September 2022].

[29] J. Amundsen, "Cypress Cucumber Preprocessor," [Online]. Available: https://github.com/badeball/cypress-cucumber-preprocessor. [Accessed 01 September 2022].

[30] Y. Singh, Software Testing, Cambridge University Press, 2011.

[31] T. Preston and W. , "Toml Documentation," [Online]. Available: https://toml.io/en/. [Accessed 01 September 2022].

[32] H. Gao, "Desigining a Solution for Software Distribution towards Varying On-Premise Deployments," Eindhoven University of Technology, Eindhoven, 2021.

[33] IBM, "ISO 9660 - IBM Documentation," [Online]. Available: https://www.ibm.com/docs/en/i/7.2?topic=formats-iso-9660. [Accessed 01 August 2022].

[34] Teach-ICT, "Teach-ICT," [Online]. Available: https://www.teach-ict.com/gcse_new/computer%20systems/user_interface/miniweb/pg3.htm. [Accessed 01 August 2022].

Appendix A SDP Delivery Process

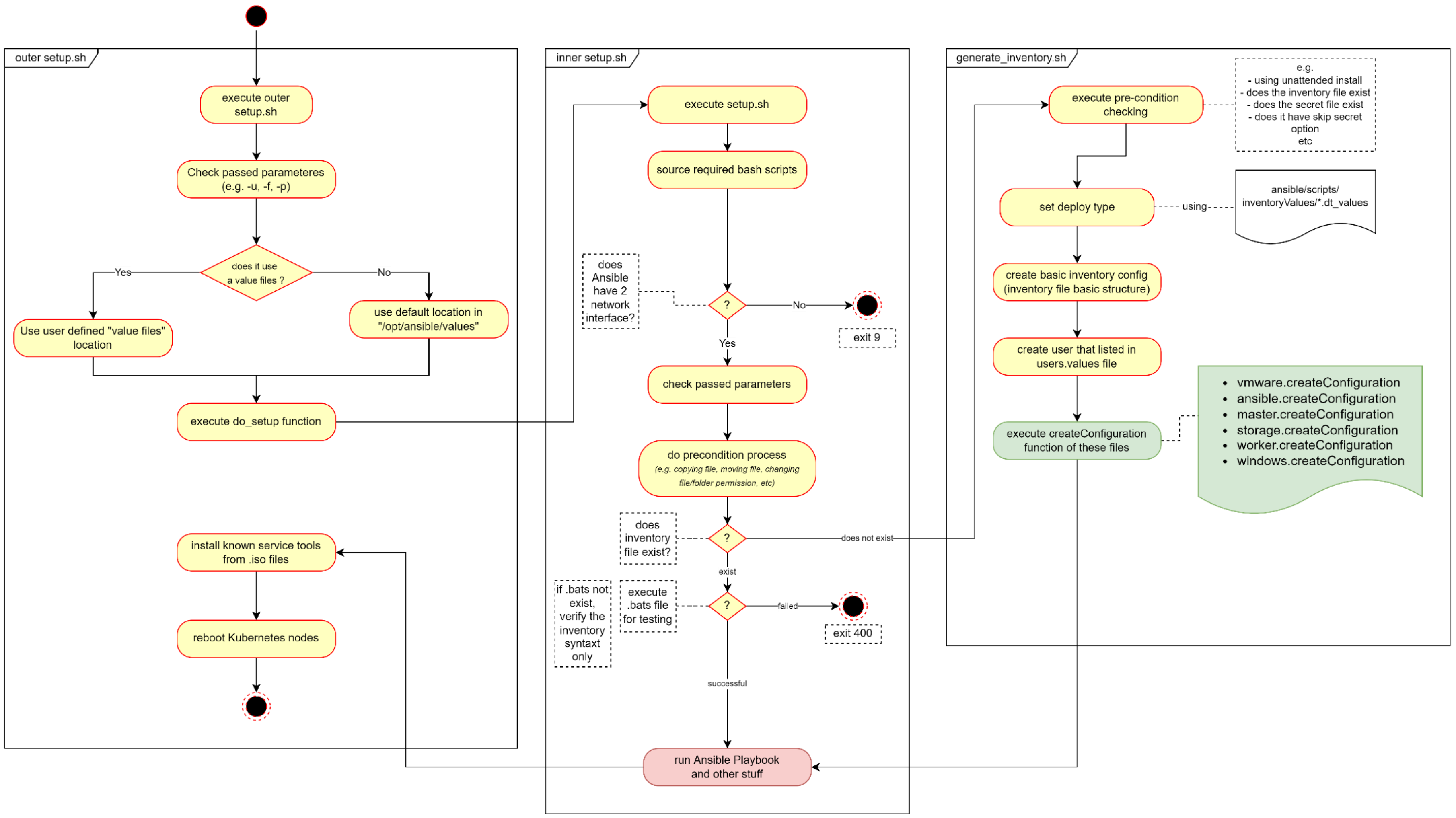

outer setup.sh
execute outer setup.sh
Check passed parameteres (e.g. -u, -f, -p)
does it use a value files ?
Yes
No
Use user defined "value files" location
use default location in "/opt/ansible/values"
execute do_setup function
install known service tools from .iso files
reboot Kubernetes nodes
inner setup.sh
execute setup.sh
source required bash scripts
does Ansible have 2 network interface?
?
No
exit 9
Yes
check passed parameters
do precondition process (e.g. copying file, moving file, changing file/folder permission, etc)
does inventory file exist?
?
does not exist
exist
if .bats not exist, verify the inventory syntaxt only
execute .bats file for testing
?
failed
exit 400
successful
run Ansible Playbook and other stuff
generate_inventory.sh
execute pre-condition checking
e.g. - using unattended install - does the inventory file exist - does the secret file exist - does it have skip secret option etc
set deploy type
using
ansible/scripts/ inventoryValues/*.dt_values
create basic inventory config (inventory file basic structure)
create user that listed in users.values file
execute createConfiguration function of these files
vmware.createConfiguration
ansible.createConfiguration
master.createConfiguration
storage.createConfiguration
worker.createConfiguration
windows.createConfiguration

Figure 42. Activity diagram of SDP delivery process

Appendix B Functional Requirement

Table 8. Functional requirements of the project

#	ID	Status	Description	Priority	Satisfy	Rationale	Notes	Example
1	FR-1	Done	The system must produce Ansible inventory file using the existing format	Must	UC1, UC2	In order to have similar ansible inventory file and keep the old mechanism as an alternative process	• The SDP configuration file creation is using GUI • The system can be used as an alternative of the current mechanism	• The file should be ex-actly the same as the current sys-tem
2	FR-2	Done	The system must visualize the SDP configuration process in a guided step GUI	Must	UC1, UC2	In order to prevent mis-take caused by the wrong input value. In addiction to have an easy to use UI for the less IT skilled people	It also should be able to go back to previous step on the wizard UI	• Wizard form • Template with pre-defined values
3	FR-3	Done	The system must have an interface to extend the SDP configuration with all the additional configuration item	Must	UC2	In order to prevent mis-take that caused by the wrong input value	• The common use case is adding cache • when extending hardware, the configurator should help the users to reconfigure the cluster (adding GPU, adding NVME)	• Update with a template configuration • Custom update configuration
4	FR-4	Done	The system must validate the form input where feasible	Must	UC1, UC2	In order to minimize mistake dur-ing the ansi-ble inventory creation time	• There will be some mandatory and optional fields on the wizard form	N/A

							• Using dropdown list, field structure • String structure e.g., IP address, mac address, hostname	
5	FR-5	Done	The system could show the progress the SDP configuration process in a progress/step bar	Could	UC1, UC2	In order to visualize how many remains step during configuration process	N/A	• Percentage • Step *0* out of *n* steps • Progress bar
6	FR-6	Done	The system could visualize the cluster topology and its resources when the configuration is finished	Could	UC3	In order to show the topology and the resources of each node in the cluster after the infrastructure is configured	N/A	• UI with layered component • Schema-liked UI • Network-liked UI
7	FR-7	Partially Implemented	The system shall generate the configuration form dynamically based on the template (deployment type)	Should	UC4	In order to have an easy to use UI to extend the configuration item based on various deployment templates now and in the future	• We implemented the behavior of the *Worker* configuration page based on the template	N/A
8	FR-8	Done	The system must allow initial configuration setup based on a deployment template, containing defaults and constraints for the configuration	Must	UC4	N/A	N/A	• Allow the user to choose the deployment template, which contains the defaults values for some of the form input
9	FR-9	Done	The system shall save work as a draft when the user interrupted the configuration process	Should	UC1, UC2	In order to save the incomplete configuration so	• This also enable the user to modify the configuration	• save to a file • save to a database
10	FR-10	Done	The system shall have user authentication/authorization	Should	All use cases	In order to secure and prevent an unintended access to the SDP configurator app		Username and password for login
11	FR-11	Not Implemented	The system could validate the number of memory/storage from the user against the hardware capability	Won't	UC1, UC2	To validate the user input in real-time	• need to interact with the machine/host to get the information	• this feature can be expand into the other aspect as well that requires interaction with the machine/host

12	FR-12	Done	The system must be easy to deploy and run on the existing SDP base infrastructure Ansible node with no or limited additional modules required	Must	All use cases	• To minimize the complexity by having an isolated environment for the system • Decouple from the rest of the system	N/A	• Run the SCA as a docker container on Ansible node

Appendix C Non-Functional Requirements

Table 9. Non-functional requirement of the project

#	ID	Status	Category	Description	Priority	Notes
1	NF-1	Done	Maintainability	The component of the system must be decoupled and in a modular structure	Must	• After the project is finished, the system shall be maintainable by the current SDP team • Modular design, separation of concern, mean-ingful abstraction • Modular struc-ture can be implemented using python package or flask blueprint
2	NF-2	Done	Extensibility	The system must be easy to extend, adding more features is considered as a version update and the method should be based on a well-defined proce-dure	Must	For the requirement that could not be delivered in this project, it should be added in the next version
3	NF-3	Done	Testability	The system shall be testable with an automatic testing standard of the SDP team using Gitlab CI/CD	Should	• Unit testing • Consider automatic testing us-ing Cypress with Gherkin syntax
4	NF-4	Done	Maintainability	The technology choices used to develop the system shall be communicated and agreed with the SDP team	Should	• Avoid a technology that give more work to the team, it should be familiar and easy to use
5	NF-5	Done	Deployment	The system shall be deployable in an air-gapped environment	Should	• Using manual dependency in-stallation (down-load/install)

Appendix D Comparative Study

We utilized a comparative study method to decide on which platform the SCA was developed. There are several needs we considered during the decision process, for instance, a need to run SCA in a limited resource and network connection, a requirement to develop SCA to be easy-to-use, easy-to-access, and easy-to-deploy, and the familiarity of the developers with the technology.

From the comparison shows in Table 10, we decided to adopt web-based platform for architectural concern. There are three main criteria to decide this concern:

1. It should be easy to execute UNIX or SSH command.
2. It should allow to structure the code in a modular fashion.
3. It should be easy to develop the Graphical User Interface.

Table 10. Application platform comparative study

Criteria/Platform	Desktop-based	Web-based
Pre-requisite packages	Light packages (e.g., Python, GNOME GUI)	Web server required to run the app
GUI development	Relatively complicated. for instance, in GTK+, it'd dependent on the standard widget gallery	Flexible to design the UI, various template can be applied for the UI
Able to execute UNIX command	Supported	Supported (using Python subprocess built-in package or PHP *shell_exec()* built-in function)
Comply the System Requirement	Difficult to implement FR-6 (Topology visualization)	Relatively easier to satisfy FR-6 (Topology visualization)
Installation or update	Manually download and update	Require manual process to update. It's not running in a central server but on each SDP infrastructure.
Modularity	Supported	Supported
Performance	Faster since it has direct access to the host	Slower
Developer experience	Low	High

After the application platform had been decided. Then we needed to consider which framework is the best fit with the objective of the project. The potential users of SCA are the Service Engineers (see Chapter 2). The SCA will be deployed on a specific customer site and the user who access SCA will be limited to the Service Engineer only. Therefore, the scalability and performance were considered less important.

Table 11 shows the comparison between Flask framework and Laravel/Lumen framework to build the web-based application. Based on the comparative study result, we concluded to use Flask framework using Python programming language to build SCA because it has minimal prerequisite to run and offers more library to support the system's funcionality. SDP team engineers are also more familiar with Python instead of PHP programming language.

Table 11. Web-based application framework comparative study

Criteria/Web-app Frameworks	Python Microframework (Flask)	PHP Microframework from Laravel (Lumen)
Deployment	Python built-in HTTP server	PHP built-in webserver
Pre-requisite	Python3, Pip	PHP7, Composer
Code structure	Very minimal and flexible structure, flexible to extend based on the needs	Fixed structure, default files come with fresh installation
Modularization	Flask Blueprint	php artisan module
Interact with UNIX command	Built-in subprocess library in Python3	*shell_exec()* available in PHP 4, PHP 5, PHP 7, PHP 8
UI template management	Jinja2 templating	Laravel blade templating
Form validation	Flask-wtforms	Built-in request validation library
SSH on the web	SSH_Client from Paramiko library	PECL ssh2 library
Caching	Flask-Caching (supports: simple, filesystem, redis, uwsgi, etc)	Two cache driver options available in Lumen: Memcache and Redis
Popularity index (programming language and framework)	Python: Rank #1 on Feb 2022 on TIOBE index	PHP: Rank #8 on Feb 2022 on TIOBE index
Benchmark	• request/sec = 21,782 • minimum latency = 284.3 ms • maximum latency = 15,522.3 ms	• request/sec = 5,401 • minimum latency = 1326.7 ms • maximum latency = 72,554.7 ms
Engineer perspective	⊕⊕⊕	⊕

Appendix E Risk Register

Table 12. Risk register of the project

ID	Category	Status	Risk	L	I	P	Mitigation
R1	Communication	Mitigated	The progress of the project might not align with stakeholders' expectations	5	5	25	• Define the scope of the project • Set a regular meeting with stakeholder to give an update about the progress
R2	Domain knowledge	Mitigated	A steep learning curve of the current system	4	4	16	• Define the system of interest • Define the system boundary • Set priority
R3	Deployment	Mitigated	The system cannot be hosted on the Ansible node	1	5	5	• Analyze the current configuration to understand how Ansible node is configured • Find possibilities on how to install the pre-requisite for the GUI app on the Ansible node (e.g., install Nginx or Docker at early step)
R4	Deployment	Mitigated	Due to an air-gapped environment, the pre-requisite module might be not downloaded	3	4	12	• Simulate the local install dependency using wheel (*.whl*) python package file • Investigate containerization using docker as an alternative
R5	Project progress	Mitigated	Delay on delivering the first prototype	3	4	12	• Focus on functionality • Prioritize the core logic rather than UI styling
R6	Project Scope	Mitigated	The implementation may not satisfy the needs because the scope is not clearly defined	3	5	15	• Conduct problem analysis and stakeholder analysis to define the scope of the project • Interview the stakeholder to understand their needs
R7	Integration	Partially Mitigated	SCA might not be able to run as part of the SDP installation	1	5	5	• Integrate the SCA with the SDP release mechanism, if possible, set up the CD pipeline

							• Make sure the component of the system is accessible in the SDP environment (e.g., running as a Docker container)
R8	User experience	Mitigated	The system may not satisfy the end user needs because it is too difficult to operate	3	4	12	• Discuss with the end users frequently to get input • Propose mockup follow with a prototype to the end users • Deploy the prototype to the development server and share the access to the users so that they can interact with it
R9	Dependency	Mitigated	Python 3 may not be available on the host where the SCA should be running	2	5	10	• Confirm with the team on when the Python3 is released • Consider running the SCA as a Docker container
R10	Delay	Mitigated	Front end styling requires much time that might cause delay	2	5	10	• Contact the engineer who has frontend skills (may contact from SPOT team who work in front-end) • Use styling template

Appendix F Detailed Use Cases

Table 13. Detailed use cases of the project

ID	Use cases	Description
UC1	Clean install/configure SDP	**Actor:** Service Engineer **Precondition:** Ansible VMware instance is up and running. **Basic Flow:** 1. Access the SDP configuration application from the local web browser. 2. Input the customer's provided information into the guided configuration form. 3. Submit the configuration. **Post-condition:** The Ansible inventory file is generated. **Alternative Flow:** 1. Installation process is interupted. 2. The configuration values are saved. 3. Once the issue is solved, continue the process using the saved configuration values.
UC2	Update/upgrade SDP configuration	**Actor:** Service Engineer **Precondition:** The infrastructure has additional capabilities or new IP addresses for the nodes. **Basic Flow:** 1. Access the SDP configuration application from the local web browser. 2. Input the customer's provided information into the guided update/upgrade configuration form. 3. Submit the configuration. **Post-condition:** The existing Ansible inventory is updated. **Alternative Flow:** 1. Installation process is interrupted. 2. The configuration values are saved. 3. Once the issue is solved, continue the process using the saved configuration values.
UC3	Visualize the configured infrastructure	**Actor:** Service Engineer **Precondition:** UC1

		Basic Flow: 1. Access the SDP configuration application from the local web browser. 2. At the SCA homepage, select the *Inspect Inventory* menu to navigate to the visualization page. 3. The SCA shows the visualization page. **Post-condition:** The current infrastructure configuration appears on the SCA visualization page.
UC4	Initialize the configurator with deployment template	**Actor:** SDP Team/DSE **Precondition:** The deployment templates are saved with the appropriate format (in TOML [31] format). **Basic Flow:** 1. Open the SCA homepage. 2. Start the SDP configuration for fresh install or update. 3. Select the deployment templates from the dropdown list. 4. Continue the configuration on the wizard step GUI of SCA. **Post-condition:** • The list of deployment type is present during the configuration process. • The predefined value that configured in the templates appears on the configuration field.

Appendix G Ansible Inventory Structure

```
{
    "all": {
        "vars": {
            "sdp_version": "2.9.0",
            "sdp_deploy_type": "lifescience",
                   …
        },
        "children": {
            "vmware": {
                "hosts": {
                    "esxi-1.dmp": {
                            "attributes": "some_values"
                    }
                }
            },
            "configuration": {
                "hosts": {
                    "k8s-ansible-1.dmp": {
                            "attributes": "some_values"
                    }
                },
                "vars": {
                            "attributes": "some_values"
                }
            },
            "masters": {
                "hosts": {
                    "k8s-master-1.dmp": {
                            "attributes": "some_values"
                    }
                },
                "vars": {
                            "attributes": "some_values"
                }
            },
            "storage": {
                "hosts": {
                    "k8s-storage-1.dmp": {
                            "attributes": "some_values"
                    }
                },
                "vars": {
                            "attributes": "some_values"
                }
            },
            "gateways": {
                "hosts": {
                    "gw-1.dmp": {
                            "attributes": "some_values"
                },
                "vars": {
                            "attributes": "some_values"
                }
            },
            "workers": {
                "hosts": {
                    "k8s-worker-1.dmp": {
                            "attributes": "some_values"
                    },
                    "k8s-worker-2.dmp": {
                            "attributes": "some_values"
                    },
                    "k8s-worker-3.dmp": {
                            "attributes": "some_values"
                    }
                },
                "vars": {
                            "attributes": "some_values"
                }
            },
            "kubernetes": {
                "children": {
                            "attributes": "some_values"
                }
            }
        }
    }
}
```

About the Author

K.A. Respa Putra received his bachelor's degree in Information System from Telkom University, Indonesia. During his study, he joined the Programming and Database (PRODASE) laboratory as a practicum assistant. In the laboratory, he pioneered and developed an online practicum system with his colleagues. In the last year of his study, he received the Global Korea Scholarship (GKS) for the undergraduate exchange student program at Kumoh Institute of Technology in South Korea for one semester. After his graduation, Respa worked at a state-owned electricity provider company named Perusahaan Listrik Negara (PT PLN Persero) in Jakarta, Indonesia. He worked there as an assistant analyst of application development for five years. Respa is interested in topics related to software architecture, software design, and software development.