Designing a solution for monitoring and managing multi-cloud on-premise deployments

PDEng THESIS REPORT

Designing a solution for monitoring and managing multi-cloud on-premise deployments

Vahe Aristakes Pezeshkian
October/2020
Department of Mathematics & Computer Science

PDEng SOFTWARE TECHNOLOGY

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Designing a solution for monitoring and managing multi-cloud on-premise deployments

Vahe Aristakes Pezeshkian

October 2020

Eindhoven University of Technology
Stan Ackermans Institute – Software Technology

PDEng Report: 2020/057

Not confidential

Partners

ThermoFisher SCIENTIFIC

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Steering Group Ir. H.T.G. Weffers, PDEng
Ir. E. Algra, PDEng

Date October 2020

Composition of the Thesis Evaluation Committee:

Chair:	Ir. H.T.G. Weffers, PDEng
Members:	Ir. E. Algra, PDEng
	Dr. R. Schoenmakers
	Dr. S. Roubstov
	Prof. dr. ir. N. Meratnia

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.

Date October, 2020

Contact address
Eindhoven University of Technology
Department of Mathematics and Computer Science
Software Technology
MF 5.080 A
P.O. Box 513
NL-5600 MB
Eindhoven, The Netherlands
+31 402744334

Published by Eindhoven University of Technology

PDEng Report 2020/057

Abstract

To support the transition process of the Electron Microscopy (EM) business of ThermoFisher Scientific (TFS) from the equipment-based model to managed services delivery, a Central Monitoring system is designed and implemented. This report elaborates on the context and technical needs for such a system by analyzing the problem domain, formulating the requirements, and describing the intended use cases. It explores the solution domain and proposes an architecture that emerges from the feasibility study, followed by the identification of components and their integration. The project management, verification, and validation processes are also described by this document. The system is implemented and deployed within TFS laboratory environment and the results and findings are presented.

Keywords

PDEng, Software Technology, TU/e, ThermoFisher Scientific, Electron Microscopy, Managed Services Delivery, Monitoring, Observability, SLA, Cloud Computing, Kafka, Kubernetes

Preferred reference

Designing a solution for monitoring and managing multi-cloud on-premise deployments. Eindhoven University of Technology, PDEng Report 2020/057, October 2020.

Partnership

This project was supported by Eindhoven University of Technology and **ThermoFisher Scientific**

Foreword

The Materials and Structural analysis Division (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 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 yet lacking the ability for our digital service organization to centrally monitor all deployed SDPs cross our customer base.

Vahe has done a great job in filling this gap. With the assignment being in-between organizational entities (various groups, both in R&D and Service), he demonstrated skills in engaging the involved parties, extracting the needs and constraints, and delivering a solution that is accepted cross the board. The SDP platform already consisted of a deep technology stack. Vahe was able to master that quickly and extend it with even more technology coming from the Kafka ecosystem. The design and implementation of the monitoring solution created by Vahe shows elegance, with good separation of concerns, appropriate technology choices and trade-offs, and good usage of common off the shelf components.

The deliverables are of good quality and completeness, demonstrated in in-house deployments today, and are ready to be deployed into customer sites shortly. Due to the circumstances of 2020, most of the work was conducted remotely. Despite these conditions, Vahe has been an involved member of the virtual team and delivered beyond expectations. I personally enjoyed the discussions we had around technology choices, architectural patterns and tradeoffs in this space around complexity, cost, technology maturity, design elegance, simplicity.

Egbert Algra, MSc, PDEng

Eindhoven University of Technology

Preface

The Professional Doctorate in Engineering (PDEng) in Software Technology (ST) at the Eindhoven University of Technology (TU/e) is a two-year technological designer program to prepare a candidate for proficiency in high-tech inter-disciplinary projects. At the final stage of the program, several design projects are proposed by various companies and candidates are elected to take on a ten-months-long project based on their interest and fitting criteria.

This report describes the final PDEng project which was proposed and supervised by Ir. E. Algra, PDEng on the behalf of ThermoFisher Scientific (TFS) and guided by Ir. H.T.G. Weffers, PDEng, as the TU/e supervisor. The purpose of the project is to design a system that can meet the TFS needs in having an infrastructure for monitoring the availability and performance aspects of various solutions provided as managed services to its customers.

The report covers the problem analysis, explores the domain where the problem is formulated, specifies the requirements of the system of interest, presents the design criteria and solution candidates, describes the implemented solution and its evolutionary development process, and summarizes the outcomes of using the implemented system in the TFS laboratories.

Vahe Aristakes Pezeshkian

October 2020

Acknowledgements

The two-year PDEng program at the Eindhoven University of Technology has been a period of learning by practicing, observing, and interacting. The final project was a unique opportunity and a major contributor to fulfilling my learning goals during the program.

I would like to express my gratitude to my project supervisors, Ir. E. Algra, PDEng, and Ir. H.T.G. Weffers, PDEng, who supported me by providing insightful feedback as well as inspiring me by being exemplary software and systems practitioners. In many years to come, I will be looking back to the PDEng project days to remember the methods and techniques that I learned, as well as to appreciate the well-brewed coffee at the office.

I had the opportunity to closely enjoy the spirit of an ambitious and rigorous team for only about two months, before the circumstances of 2020 limited our interactions within the virtual space. However, I never felt a lack of assistance from the team for which I would like to thank Giovanni de Almeida Calheiros, Tor Halsan, Gang Chen, Tarkan Akcay, and Chris Schlichten.

This project was done in close collaboration between the R&D group and Service Organization at the company. I would like to thank everyone at ThermoFisher Scientific, especially Frank van Apeldoorn, for their eagerness and support in the joint effort.

It would not have been feasible to carry on the efforts during the extraordinary conditions that the world faced during this period without the heart-warming support of my family. No word can express my gratitude for my parents' devotion to my achievements, however, this is a humble try.

Finally, I would like to thank Yanja Dajsuren, Desireé van Oorschot, Peter Heuberger, Judith Strother, and all the professors at the Eindhoven University of Technology who guided me during the past two years, as well as all my friends with whom I had the privilege to share the journey of the PDEng program.

Vahe Aristakes Pezeshkian

October 2020

Executive Summary

Scientists who research the life science, material sciences, and semiconductor fields, employ advanced equipment such as Electron Microscopes (EM) to study structures and materials at the nanoscopic scale. As the researchers contribute to the acquisition of critical knowledge, such as viral studies and vaccine development, they must have operational equipment. ThermoFisher Scientific (TFS), as a major company serving science, has several high-end EM products to offer to the scientists and help them in conducting their experiments and research for discovering new knowledge.

TFS is aiming to grow from an equipment provider to a solution provider company in the EM domain. That path includes delivering software-based solutions as a managed service to its customers, where SLAs on availability and throughput play an essential role.

As a step forward in enabling TFS to provide managed services, the Software Delivery Platform (SDP) was developed to set up an on-premise infrastructure at each customer site. The SDP elevates TFS from an equipment-producing company to an EM service provider company via various application software that integrates with the customer workflows.

This project advances TFS in providing managed services by bringing remote observability to the SDP infrastructure. A Central Monitoring (CM) system enables TFS to gain insight about the status of its infrastructure and in case of an incident have sufficient actionable information.

The CM system is designed and implemented with extensibility in mind; knowing that the set of services and applications offered by TFS will expand in the future. Additionally, The CM system is compatible with the tools and procedures in use by the Service Organization at TFS.

One of the constraints originating from the confidentiality concerns of TFS customers is the need to adapt to variant connectivity options. The CM system provides interfaces to operate with various connection mechanisms, including traditional means such as emails.

During the project, we developed a CM prototype and added it to the standard tools that are shipped with the SDP. The prototype is verified in the TFS laboratories in Eindhoven, and a set of dashboards are provided to the Service Organization. The CM prototype infrastructure is ready to be used in production; the next steps for the TFS Service Organization is configuring CM for incorporating the meaningful data and dashboards. Using the CM system in production, TFS will have the data and views available for providing SLA-compliant managed services to its customers and assisting them in using the EM equipment in the best way.

Glossary

ASG	Application Software Group
CI	Continuous Integration
CM	Central Monitoring
CMC	Central Monitoring Connect
Cryo-EM	Cryogenic Electron Microscope
D2I	Data to Information
DC	Data Collector
DMP	Data Management Platform
DSE	Digital Service Engineer
EM	Electron Microscope
MSD	Materials and Structural analysis Division
NMS	Node Management Service
NPD	New Product Development
PSG	Project Steering Group
RPC	Remote Procedure Call
SaaS	Software-as-a-Service
SDP	Software Delivery Platform
SEM	Scanning Electron Microscope
SLA	Service-level Agreement
SMTP	Simple Mail Transfer Protocol
TEM	Transmission Electron Microscope
TFS	ThermoFisher Scientific

List of Tables

List of Figures

Contents

1 Introduction

1.1 Context

ThermoFisher Scientific (TFS) is a world leader in serving science, with the mission to make the world healthier, cleaner, and safer. Various TFS products and services that are offered in several brands enable the scientists and researchers to take leaps forward in science and technology [5].

Thermo Scientific is one of the TFS brands that provides equipment, software, and services to address the needs in analyzing biological, chemical, and material-based samples at the nanoscale. Electron Microscopes (EM) are a family of equipment that are widely used in life sciences, material science, and semiconductor studies and industries. An EM is a microscope that uses a beam of accelerated electrons as the source of illumination to observe samples. By using the beam of electrons instead of light rays, the EM can capture images of nanoscopic samples, which the regular rays of photons cannot pass through.

There are several variations of EMs, such as Transmission Electron Microscopes (TEM), Scanning Electron Microscopes (SEM), and Cryogenic Electron Microscopes (cryo-EM). A TEM produces images by transmitting the electrons through the samples and generates a highly-magnified image. An SEM operates by directing the electrons to scan the surface of samples instead of traversing through them. Since a beam of electrons carries high energy, it can destroy the structure of the sample while observing it, thus creating an inaccurate image. To tackle this side effect, cryo-EMs cool and freeze the sample before targeting it with electrons.

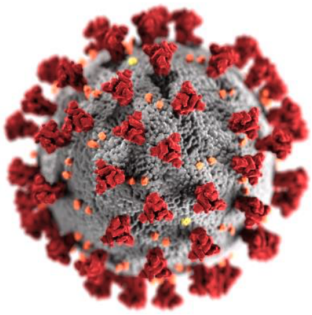

Figure 1.1: Computer rendering of a coronavirus, published by CDC [1]

The EMs that TFS manufactures are widely used in various research fields. For instance, during the COVID-19 pandemic, scientists used EMs to increase their understanding of the SARS-CoV-2 virus. Particularly, cryo-EMs were used to determine the structure of virus spike protein and its cellular receptor during infection [6]. Figure 1.1 shows a computer rendering of a coronavirus, which is

obtained using an EM manufactured by TFS.

To render images of nanoscopic samples, such as the coronavirus, the EM creates a large amount of data during the observation process. The data is then processed and turned into images that scientists use for analysis. The conversion of the data into workable images is done using high-end computing resources. To facilitate the sample observation process that the scientists perform using EMs, TFS has designed and built an IT solution that consists of an information system, called Data Management Platform (DMP), and an application suite. By using the DMP and its application software, the customers of TFS can quickly and easily obtain images of the samples that they observe in the EMs.

The DMP provides an environment for deploying the infrastructure and application software for the EMs. The deployment environment consists of computing resources such as servers and high-capacity storage volumes, virtualization technologies, a stack of utility software, and finally application software, which the end users of the EMs interact with.

The engineers at TFS are continuously working on building new products and services to bring the best possible solutions to their customers as soon as possible, which leads to the continuous release of application software. To streamline the maintenance and release of new software, TFS has developed an infrastructure, named Software Delivery Platform (SDP). The SDP defines a standard mechanism for all kinds of application software to be hosted on the DMP.

The standardization simplifies the management, extension, and scaling of the application software. Additionally, it enables the continuous support and delivery of upgraded software.

1.2 Project Motivation

While TFS used to operate on an equipment-based style, i.e., selling EM equipment, it is shifting its strategy, workflows, and toolset to become a managed services provider of EM solutions. In the managed services paradigm, TFS not only provides the EMs and the DMP to its customers, but also continuously maintains the equipment, provides software updates, and ensures that the EM services are available to the scientists.

The customers of TFS need to rely on performant and highly-available EM services to conduct their research and studies, whether in biology or materials science. TFS aims to provide the services to offer a seamless experience in using EMs to its customers. To reach this goal, the SDP is designed to support TFS to provide managed services in the EM domain. As an enabler technology, the SDP equips the EM and DMP with the infrastructure to be offered in the context of managed services delivery.

As the usage of EMs is becoming more popular, TFS expects growth in the number of customers that order EM solutions and services. On the other hand, engineers at TFS continuously design and develop new products and services to offer better solutions to their customers. As a result, TFS is looking for ways to set up an efficient and sustainable mechanism for regularly maintaining all of the EM solutions that are shipped and installed at the customer sites.

One of the essential components to operate as a managed services provider is a central monitoring and management system. A Central Monitoring (CM) system will help TFS to be aware of the status of the EM and DMP systems that it has shipped and installed. It increases the observability of the infrastructure. As a result, TFS support and engineering teams can make informed decisions to fix the problems in a preventive manner.

By utilizing a CM system and adopting the managed services workflows, TFS will have the tools at hand to ensure that the solutions that are provided to the customer are always in good condition. As a result, the customers of TFS, who use EMs to move the science forward, can have equipment always ready in their laboratories to contribute to their research.

1.3 Project Scope

The goal of this project was to design and develop a prototype of a system that demonstrates the value of observability for the DMP solution to achieve the quality and availability of managed services delivery.

As an outcome of the project, we provided a prototype of a CM system to TFS that enables the DMP product to report its status, provides data as insights, and gives TFS Service Organization the possibility to observe several remote DMP instances from a central viewpoint.

The system serves as an infrastructure for the Service Organization at TFS to visually observe the performance and status of the customer systems and relate them to the Service Level Agreements (SLAs). This sets a foundation for taking maintenance actions proactively to ensure that the customer systems function properly.

The project intended to enable the SDP engineers and application developers to gain historical data-backed insights from the DMPs and each software application that they build. The data helps the engineers make decisions on how to design and build the next TFS application software to help customers achieve their goals.

2 Problem Analysis

While TFS Materials and Structural analysis Division (MSD) has traditionally been a major manufacturer of EMs, it has mainly operated with an equipment-based model: When a customer orders an EM, TFS ships the equipment to the customer's site, sets up the required infrastructure, and prepares it for the customer's use. The transaction is considered complete at this stage. However, since the EMs are complex devices, its users rely on the support from TFS and its experts' guidance to fix the problems and to use the microscopes efficiently. Therefore, the transaction of selling and shipping the EMs is followed by several on-demand maintenance processes, which are also known as the break-fix service pattern.

The customers use the product until a problem occurs and they contact the Service Organization at TFS. Afterward, Service Organization staff members visit the customer site or set up a remote call to fix the problem. Depending on the type of the problem, the problem-fixing process takes from a few hours to days.

As TFS is moving toward offering EM solutions as managed services, the company needs to take a proactive approach in assuring that the delivered products operate according to the SLAs and deliver the expected service to the customers. To be able to provide managed services for TFS EM and DMP products as a single system, the system has to be remotely observable, manageable, and complaint with the confidentiality requirements of the customers.

However, observability and manageability contradict with the confidentiality concerns that TFS customers have, since they work with valuable research assets, such as experiment data. Since the EMs are used for innovative undertakings in laboratories, customers impose constraints on the data that they can share with TFS for maintenance purposes.

The mentioned properties (observability, manageability, and confidentiality-complaint) are the key principles that guide the design and evolution of a CM system in the lifetime of this project. In the upcoming section, we define the problem that the project addresses, in more detail.

2.1 Problem Study

In studying the problem, we look at the problem from business, technical, and realization aspects and describe the significant properties and principles in each. We selected this framework, based on the observation stated in [7] that software architecture is a result of technical, business, and social influences. Therefore, to propose a solution that fits the problem space, it is important to understand the problem from different aspects.

2.1.1 Business Aspect

In analyzing the business aspect of this project, we take a look at the key business drivers that initiated the project.

As stated earlier, TFS is transforming its service and maintenance processes from an on-demand pattern to a managed services delivery model. The transformation, although not prevalent in the EM domain, has been widely adopted in the software delivery domain, being the Software-as-a-Service (SaaS) model.

TFS pursues a model comparable to SaaS for delivering managed services. The benefits of adopting a SaaS-like approach are:

- Becoming competitive in providing a well-integrated EM and application suite ecosystem
- Securing recurrent revenue streams by selling subscription licenses to TFS products and services

In the SaaS model, the service provider company manages the product centrally and the services, software, and upgrades are provided over the Internet. There are two properties of the SaaS model that are beneficial to the EM domain [8]:

- The application software repository is hosted centrally, so an update is applied by the provider, not by the customers.
- The solution provider can use diagnostics data such as metrics to continuously improve the service delivered via software.

The SDP has been added to the stack of technologies that TFS provides to customers to realize adopting the SaaS model. The SDP also helps the company provide confidentiality-compliant services. In the prevalent SaaS model, the entire system is deployed centrally and accessed by all the customers. Whereas in the model that TFS pursues, each customer has an on-premise dedicated system. Since the EM and its companion DMP are located at the customer's site, the customer has administrative rights to control the data that the system produces and processes. This is an example of an on-premise cloud deployment.

However, the on-premise deployment of the SDP poses challenges to TFS for monitoring and managing the deployments. The Service Organization at TFS needs to have access to the diagnostics data from the DMP to monitor the status of the services and take necessary steps to fix the technical issues.

A CM system plays a crucial role, especially when TFS deploys a growing number of SDP instances. Since all the SDPs follow a similar architecture and use a standard toolset, it will be convenient for the Service Organization to have a single view across all the customer sites.

In the long term, having access to the diagnostics data of various customers allows TFS to draw data-driven conclusions about the effectiveness of their services and make better strategic choices in developing new products.

2.1.2 Technical Aspect

The project can be divided into three main subsystems from the technical point of view. When combined, the subsystems deliver the system's functionality.

This section provides an overview of the goals of the subsystems and the environment that surrounds each. The detailed design description of the subsystems and their comprising components is provided in Chapter 6 (Solution Description).

The main roles of the subsystems are:

1. Reporting the Diagnostics Data
2. Acquiring the Reported Data
3. Using the Diagnostics Data

Reporting Diagnostics Data

As discussed earlier, the DMP is using the SDP to make software services available to the end users.

The SDP is a layered stack of technologies that creates an on-premise cloud infrastructure for the customer. This enables the customer to scale up and down the computing resources that analyze the microscope's observations or to run other application software.

The extensive set of technologies used in the SDP enables customers to run application workloads and system engineers to perform essential log and metrics monitoring within a single on-premise cloud.

The CM system must operate in the technical environment that we described above and must extend the existing infrastructure to report the diagnostics data, which is collected on-site, to TFS.

Acquiring the Reported Data

The customers of TFS are globally distributed. Therefore, the reported diagnostics data must be transferred from the customer's site to a central location, accessible to TFS.

It might seem straightforward to use a high-capacity Internet connection to transfer the data between the customer and TFS sites, as it is practiced in the SaaS domain. However, in our problem domain, there are limitations in extracting and sharing data from several aspects:

- Confidentiality: The customer is in control of the data that must not leave their organization's boundaries
- Networking: The customer does not provide a direct inbound or outbound connection
- Bandwidth: In case of an outbound connection, it is not feasible to share the entire data over the network

As a result of the constraints, the CM system must use the connection channel efficiently and if no connection is allowed, it must provide an on-site digest view so that TFS Service Organization staff can remotely use the diagnostics data via a special permit or while visiting the customer's site.

Using the Diagnostics Data

The primary users of the CM system are the TFS Service Organization staff, who will use the diagnostics data to keep the services available for the customers, by proactively taking maintenance actions.

In the SaaS industry, there is a wide range of technologies and toolset in the monitoring domain for building dashboards using the diagnostics data. The dashboards visualize the underlying data in a way that the service staff can relate the graphs and statistics to the SLAs and understand if the system performance is acceptable or not.

In the SaaS monitoring domain, the key elements of the diagnostics data are divided into four categories:

- Events
- Metrics
- Alerts
- Configuration

Each set of elements is displayed in one or more views in the dashboards that address an important aspect of the SLA.

For example, a service expert with the *Hardware Manager* role is interested in managing the disk usage of the system, which according to the SLA, must be less than 80%. The Hardware Manager needs to know if the disk space is running low and as a maintenance step, he must add more storage. The metric that the Hardware Manager should monitor is *Free Disk Space*.

Similarly, a *Software Release Manager* should know if the server has the latest version of the Operating System (OS) installed, and must receive an alert if it is not the case so that it will be upgraded. In this case, the diagnostics data of interest is *OS Version Number*, which is an example of configuration data.

Therefore, the process of building monitoring dashboards is a dynamic one. As TFS extends the set of products and services, new use cases emerge that involve specific diagnostics data, targeted at specific groups within the Service Organization at TFS. From the technical point of view, the CM system provides an integration point between the SDP and dashboard systems so that each group within the Service Organization can derive the dashboard views that address their needs, using the diagnostics data that the system provides.

2.1.3 Realization Aspect

This section describes how the activities for implementing the project are related to the goals of different organizational units at TFS.

New Product Development (NPD)

NPD group has designed and built the necessary infrastructure that TFS uses to become a managed services provider in the EM domain. The DMP and SDP are the artifacts of the NPD group.

The CM system is an extension of the SDP. Therefore, the main project activities are conducted in the NPD group. The group has the knowledge, vision, and expertise about the SDP and is the owner of the technical and architectural decisions that affect the evolution of the system.

Application Software Group (ASG)

ASG is a division of the company that develops software that is used by the end users of the microscope in the laboratories where the EMs are deployed. Since the software is designed for a specific target group, the ASG can provide the diagnostics data that one needs to verify whether the software meets the SLA.

The ASG is one of the stakeholders of the CM system. The group provides input to the requirements regarding the diagnostics data of interest and provides examples of the data which are significant for the purpose of SLA monitoring.

Service Organization

The Service Organization is the primary user of the CM system. During the lifetime of the project, they provide requirements on how the system will be integrated into the service infrastructure and workflows.

2.2 Stakeholder Needs

The three organizational units describe in Section 2.1.3 comprise various groups of stakeholders at TFS:

- The NPD group builds and maintains the CM system since the system will become part of the SDP.
- The ASG provides application-specific data to the system, therefore it is one of the users of the system.
- Service Organization is the primary user of the system at TFS since its different teams and individuals will rely on the system to implement managed services delivery processes.

The needs and interests of the stakeholders are listed below:

1. The NPD group needs to implement monitoring solution into SDP and hand it over to internal TFS users, ASG and Service Organization.
2. The NPD group needs to have a standard mechanism to define the infrastructure-level diagnostics.
3. The NPD group needs to maintain the consistency of SDP architecture and design and verify its quality while adding the CM system.
4. The ASG needs to have a standard mechanism to define the application-level diagnostics.

5. The ASG needs to obtain application-level diagnostics data to improve their products.
6. The Service Organization needs a mechanism to extract diagnostics data from the DMP, to make data-driven decisions.
7. The Service Organization needs to monitor the compliance of the DMPs with the SLAs by using a central dashboard.
8. The Service Organization needs to have a set of useful diagnostics data to implement the managed services processes efficiently.
9. The Service Organization needs to be notified when a defect happens at the customer's site, preferably earlier than it affects the customer's workflows.
10. The Service Organization needs to assure TFS customers of protecting their confidential data while obtaining DMP diagnostics data.

2.3 Solution Direction

The solution to address stakeholder needs listed in Section 2.2 is aimed at:

- Implementing an SDP-compatible component that collects, aggregates, and reports diagnostics data to a centrally managed destination
- Implementing a mechanism for defining diagnostics types and controlling their reporting mechanism
- Adopting a flexible method for the connectivity of customer's DMP and the centrally managed monitoring platform
- Providing an interface for integrating with data visualization systems, e.g., dashboards

3 Domain Analysis

As described in Chapter 2 (Problem Analysis), TFS is implementing managed workflows for delivering EM services to is customers. Figure 3.1 demonstrates the role of CM as a concept for providing EM services in a managed way.

When a customer purchases an EM solution from TFS, the equipment and infrastructure of the solution are shipped and installed at the customer's organization. The provided solution by TFS is composed of the EM and its supporting DMP system. The DMP is powered by a powerful computing server, high-capacity storage disks, one or more Graphical Processing Units (GPU), and a secure network infrastructure that isolates the EM's operational environment from the public Internet.

The DMP system intends to support operations of EM via the application software. The SDP, on its turn, is responsible for providing an environment where the application software suite is hosted. Eventually, the software that is running on SDP, is exposed to the end users of the EM, who are typically scientists and researchers.

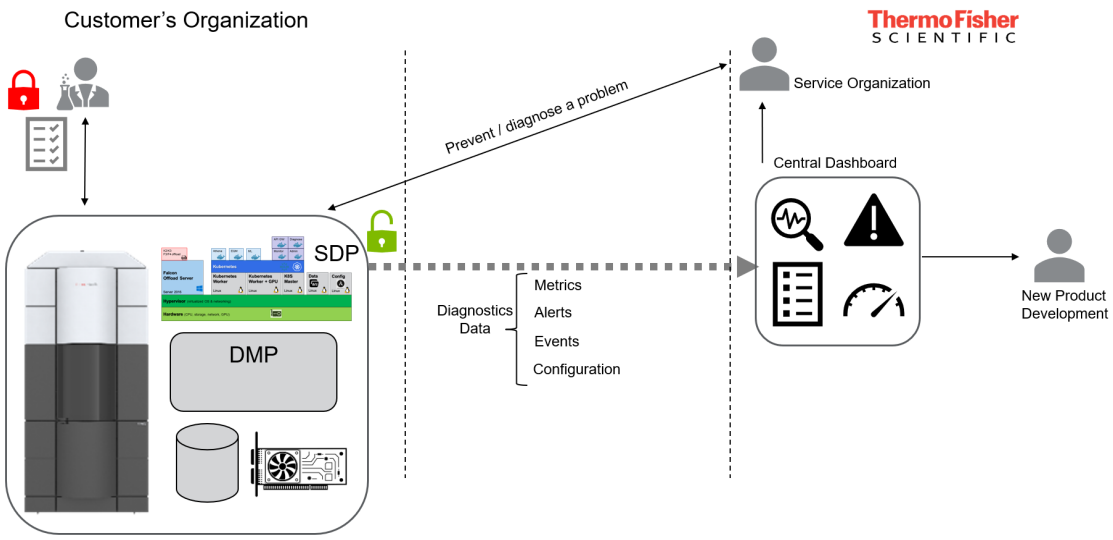

Figure 3.1: Managed services workflows

Researchers of the customer's organization use the high-end EM and conduct experiments that involve observing nano-scale samples. The users of the EM are interested in keeping their experiments and results confidential, as they contribute to the development of innovative products that contain new knowledge, such as a vaccine. However, in order to enable TFS to remotely support the operations of the EM, they allow certain diagnostics data to cross their organization boundaries.

A central dashboard at the TFS site receives the diagnostics data and displays them to two main groups of users:

- Service Organization: The Service Organization staff use the diagnostics data in order to understand the status of the EM and DMP systems at the customer's end. They receive alerts and metrics to take timely actions for preventing or diagnosing a problem that might interrupt the customer's experiments.
- New Product Development: NPD staff use the historical data in order to evaluate the quality aspects of their products, such as the amount of used and available disk space and the mean duration for rendering an image of a virus. The accumulated data over time enable NPD staff to gain insights and ideas for improving their hardware, software, and algorithms.

The described workflow in this project combines concepts from three interconnected disciplines:

- Managed services delivery
- Cloud computing
- IT infrastructure monitoring

3.1 Managed Services Delivery

TFS traditionally ships EMs to the customers and sets up the microscope and the required infrastructure at the customer sites. However, the interaction between the company and the customers does not terminate once the solution is shipped and configured. Since the products are continuously upgraded, the company staff must frequently visit the customer sites to update or install new software. Additionally, when an incident happens, specialists must urgently travel to the customer site to investigate the issue and fix the problem.

The described scenario is known as the on-demand service model, using which the provider of the service addresses the problems after they arise, in a discrete manner. The on-demand service model imposes high maintenance costs mainly due to the necessity of traveling or dedicating several service staff for supporting each customer. Given the fact that TFS plans to increase the number of deployments, the traditional, on-demand, service model cannot scale to meet the expected growth requirements.

An alternative method to the on-demand model for providing services is the managed services delivery model. In the managed services delivery model, TFS Service Organization staff take the maintenance actions without the need to travel to the customer site, cutting maintenance time and costs. In this model, the service is streamlined from TFS to the customers. It opens the opportunity for preventing potential problems before they occur, or before the customer notices them, by actively monitoring the systems and predicting an upcoming failure based on the status and usage of the system. In addition to improving service quality, adopting the managed services delivery model helps the experts share knowledge and experience, since they share the service infrastructure.

Once adopting the managed services processes, there has to be an agreement between the service provider and the recipient. The SLA defines the terms of service delivery and the guaranteed quality

and availability that TFS commits to its customers. The SLA relies on a quantitative description of the quality of services, such as the percentage of service uptime and the number of image acquisitions per day.

In order to deliver managed services, it is crucial to have a mechanism for measuring the parameters of the system concerning the SLA and reporting them to the service managers and the customers, to improve visibility, which leads to having a better customer relationship.

The CM system is critical in ensuring the realization of the SLA in order to deliver managed services.

3.2 Cloud Computing

The National Institute of Standards and Technology (NIST) defines cloud computing as a model for enabling ubiquitous, convenient, and on-demand access to a shared pool of computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction [9]. The ability to scale resources up and down automatically or semi-automatically is called rapid elasticity.

TFS leverages the cloud computing paradigms in developing the SDP in order to achieve rapid elasticity and resource pool sharing to operate EMs and its application software.

The infrastructure for cloud computing is called a cloud and according to [9], there are multiple cloud deployment types, one of which being the on-premise cloud deployment. An on-premise cloud is hosted physically at the customer's site and not shared by any other organization. Given the confidentiality constraints of TFS customers, the preferred cloud type that TFS pursues is the on-premise cloud.

The New Product Development group at TFS has used toolset and technologies for building on-premise clouds and deploying them at the customer sites. The utilized technologies include:

- VMware vSphere Hypervisor [10], as the virtualization layer on a computing server
- CentOS Linux [11], as the operating system of choice across the on-premise cloud
- Docker Containers [12], as a mechanism to deploy infrastructure and application software
- Kubernetes [13], as the container-orchestration system of choice to manage micro-service application deployments
- Elasticsearch [14], Fluentd [15], and Kibana [16] stack (EFK), as storage for searchable application and system logs
- Prometheus [17], as an on-premise monitoring system for application and hardware metrics
- Grafana [18], as a tool for creating dashboards within an on-premise cloud

3.3 IT Infrastructure Monitoring

The on-premise cloud, which supports operations of the customers, is a system that contains an elastic infrastructure that hosts numerous applications. The complexity of such a system can grow notably

when more services are provided via the infrastructure. Therefore, the observability of the system is essential for its continuous usability. According to [2], observability is the property of being able to describe or reconstruct the internal state of the system by using its output.

According to the modern IT infrastructure observability practices, it is advised that the system provides dedicated types of output data for diagnosis.

The guidelines in [2] define three main categories of the diagnostics data:

- Metrics, in order to capture a numeric value about the system at a specific point in time – for example, the number of image acquisitions currently in progress by the EM
- Events, in order to describe a discrete occurrence that provides crucial context for understanding changes in the system's behavior – for example, an upgrade of the operating system
- Alerts, in order to communicate a specific problem in the system – for example, "The image database disk is full"

Another practice which is highly recommended by [2] and [19] is tagging of the diagnostics data. Tagging is a mechanism for assigning meta-data to the metrics, events, and alerts, in order to specify additional information that although are not part of the main data, are important for analysis. Examples of tags are the name of the customer, the IP address of the server running the application, and the version of the operating system that runs the application. Since the cloud infrastructure and the application software running in the cloud are elastic, the diagnostics data that they output also have different form factors at different points in time. Leveraging the tagging technique allows ingesting new meta-data without limiting the data structure while designing the system, when the format of the data that will flow in the system is unknown.

In the monitoring domain, the metrics are visualized by tools such as time-series graphs, single numbers, and charts. A dashboard is a view consisting of one or more graphs that are designed to describe the historical and current status of the system and its components at a single glance. For example, the dashboard displayed in Figure 3.2 provides insights about the performance of a web application.

The tagging model suggests to keep the meta-data in the diagnostics data and assumes that the displaying graphs and dashboards will aggregate or separate the data and construct a view based on the metrics and events dynamically. This method of processing is known as slicing and dicing.

The SDP is equipped with state-of-the-art open-source monitoring tools such as Prometheus, Elasticsearch, and Grafana that make the SDP system observable.

- Prometheus is an open-source software application that collects metrics and provides alerting functionalities based on the evaluation of the metrics and comparing them against certain thresholds [17].
- Elasticsearch is a search engine that is used for storing and querying schema-free text documents. The SDP uses the Elasticsearch engine for storing and processing events that the infrastructure and application components produce [14].
- Grafana is a platform for visualization of metrics, events, and alerts that integrates with multiple data sources including Prometheus and Elasticsearch [18]. Grafana is designed as an extendable system, using graphs and charts that can be added as plugins, and the open-source community provides a rich pool of plugins for dynamically creating views based on monitoring data.

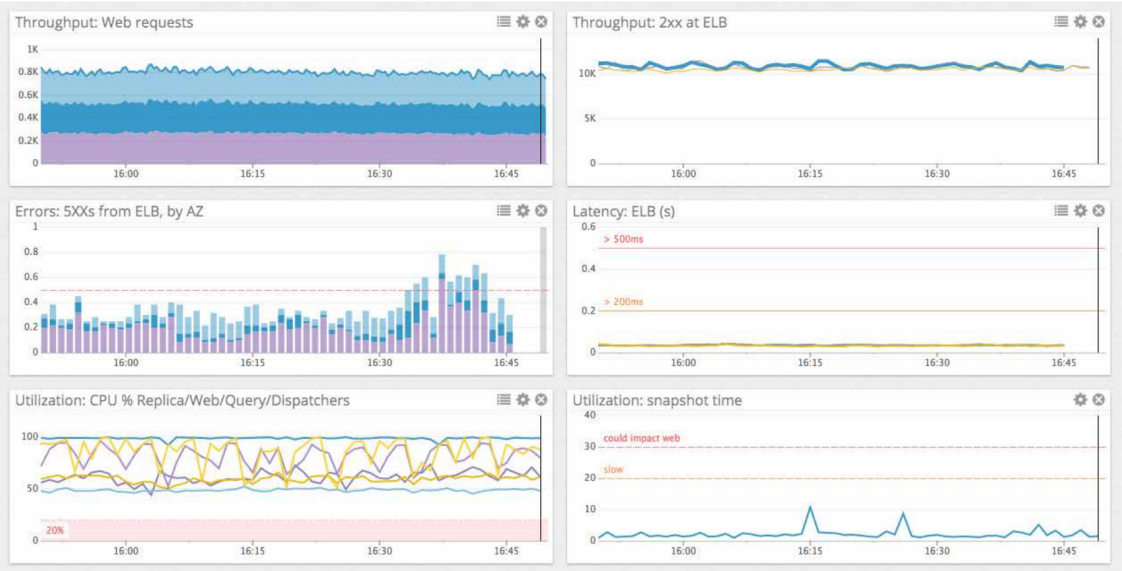

Figure 3.2: A dashboard describing performance of a web application [2]

3.4 Summary

The problem defined in this project combines paradigms from three domains: Managed services delivery, cloud computing, and IT solutions monitoring.

The central monitoring system helps TFS monitor the realization of SLAs in order to fulfill the commitments of service quality. The set of metrics, events, and alerts that make the SLAs measurable are the main data that fuel the central monitoring system.

The system is an extension of the existing SDP. Therefore, it must be compatible with the family of technologies that SDP relies on. Specifically, Kubernetes [13] is the enabler platform for deploying any software application on the SDP and the CM system must use its mechanisms and patterns to bind existing and new components.

The SDP provides an on-premise monitoring solution composed of Prometheus, Elasticsearch, and Grafana. This toolset allows TFS to support staff to use a web interface by accessing the on-premise SDP deployment at the customer's site. However, the SDP, at its current state, does not permit the staff to be aware of the problems before they cause an incident and are not sufficient for the realization of managed services delivery.

4 Requirement Analysis

The requirements describe properties of the CM system that must fit in the problem space that TFS is facing. Therefore, they serve as key specifications for the selection or development of the system.

In order to develop the requirements specification document, we used a techniques that consisted of:

- Stakeholder interviews: During interviews and brainstorming sessions with the stakeholders, they mentioned their interests and concerns, which helped to identify the needs that the system must address.
- Rapid prototyping: We developed prototypes using Components Off The Shelf (COTS) and used the prototypes during the interactions with the stakeholders to demonstrate possible solutions. The Visual elements in our discussions helped us choose the preferred solution for the system.
- Comparative study: The comparison between existing systems in the monitoring domain helped to discover varieties of the systems and while comparing the alternative systems, we discovered the favorable requirements for the business needs of TFS.

We also selected a template for documenting the requirements. The method used for textually describing the requirements is based on the Easy Approach to Requirements Syntax (EARS) guidelines [20]. In addition to the requirements text, the following meta-data is assigned to each requirement:

- Unique Requirement ID, or order to refer to each requirement
- Importance, categorized as must-have and should-have
- Rationale, the reason that justifies the existence of the requirement

Additionally, the requirements are divided into functional and non-functional categories.

This chapter describes the functional and non-functional requirementes of the system, defines the main actors of the system and their roles, and provides an overview of the use cases that the CM system must provide to the actors.

4.1 Functional Requirements

ID	REQ 1
Title	Unidirectional Connection
Description	The system must collect diagnostics data using only a unidirectional (from on-premise to TFS) connection.
Importance	Must-Have
Rationale	In order to meet the connection constraints imposed by the on-premise gateway settings

ID	REQ 2
Title	Connection Interface
Description	The channel of transporting the diagnostics data from the on-premise site to TFS must be decoupled from the rest of the system.
Importance	Must-Have
Rationale	In order to minimize the changes in the system when switching between the transport channels

ID	REQ 3
Title	Connection via Email
Description	The system must support SMTP as a transport mechanism for the diagnostics data.
Importance	Must-Have
Rationale	In order to include partially connected customer premises in the central monitoring system

ID	REQ 4
Title	Connection Latency
Description	When an incident happens at the on-premise site, the system must trigger the alert on the central dashboard with a maximum delay of 5 minutes.
Importance	Must-Have
Rationale	In order to provide enough time to the DSEs to react and solve the problem within 1 hour

ID	REQ 5
Title	Multi-layer Monitoring
Description	When a metric or event is recorded in the on-premise Prometheus and Elasticsearch instances from any layer of the SDP, the system must extract and transport the metric/event according to the configuration.
Importance	Must-Have
Rationale	In order to observe components from all layers of the SDP in the CM system

ID	REQ 6
Title	Filtering
Description	The system must have an interface to configure the metrics and events to selectively extract and transport to the central dashboard.
Importance	Must-Have
Rationale	In order to respect the confidentiality constraints imposed by the customers

ID	REQ 7
Title	Runtime Configuration Update
Description	When the configuration changes, the system must adapt to the configuration dynamically, during the runtime.
Importance	Must-Have
Rationale	In order to quickly create and modify dashboard views without waiting for new software release

ID	REQ 8
Title	Multi-cloud Support
Description	The system must label each diagnostics data item with the unique identifier of the SDP.
Importance	Must-Have
Rationale	In order to distinguish the originating sources of diagnostics data on the central dashboard

ID	REQ 9
Title	Diagnostics Data Types
Description	The system must provide the following diagnostics data types: Metrics, Events, Alerts, Configuration
Importance	Must-Have
Rationale	In order to be able to analyse the outcome of specific events and actions

4.2 Non-functional Requirements

ID	NFR 1
Title	Operability
Description	The operators of the system (DSE and SDP engineers) deploy, delete, and upgrade the system on a Kubernetes cluster, using Helm. The installation and configuration methods should be similar to the other existing applications of SDP.
Importance	Must-Have
Rationale	In order to assure consistency of the SDP, as a whole system.

ID	NFR 2
Title	Extensibility
Description	It should be easy to extend the sources of events, alerts, configuration, and metrics in the future releases. These changes are considered a "source code change" and end up as a new version of the system, but the method to do it should be based on well-defined conventions.
Importance	Must-Have
Rationale	In order to extend the functionality of the SDP, for example, to integrate with D2I, and to support the future applications that will run on SDP

ID	NFR 3
Title	Configurability
Description	It should be easy to extend the sources of events, alerts, configuration, and metrics while the system is operating. These changes are considered a "configuration update" and do not end up as a new version of the system.
Importance	Must-Have
Rationale	In order to extend or limit the scope and volume of the diagnostics data that CM system ingests, processes, and transports and in order to meet the limitations imposed by connection channel (confidentiality, networking, bandwidth)

ID	NFR 4
Title	Testability
Description	The system should be testable through the standard testing pipelines on Jenkins, as part of the SDP release and testing procedures.
Importance	Must-Have
Rationale	In order to assure quality of the SDP, as a whole system.

ID	NFR 5
Title	Resilience
Description	In case of recovery from a failure, the system should process the events and metrics that the SDP produced during the downtime of the CM system.
Importance	Must-Have
Rationale	Otherwise, a CM system is not usable with a lot of missing data.

ID	NFR 6
Title	Scalability
Description	The system should support the growing load and sources of diagnostics data.
Importance	Should-Have
Rationale	In order to meet the growing volume of metrics and events from the SDP components.

4.3 Actors

1. **Digital Service Engineer (DSE)**
   A Digital Service Engineer is a specialist at TFS Service Organization that assists one or more customers in keeping the DMPs in healthy condition. The DSE has the knowledge and skills to operate the DMP infrastructure, use the service tools, and delegate the issues to expert groups for further investigation and action.

2. **Application Domain Expert**
   An Application Domain Expert is a member of the NPD group or ASG that designs a new software application to be used by the customers. The expert is aware of the business and technical specifications of the software and can identify the metrics and events that are useful for monitoring purposes.

3. **D2I Team**
   The Data-to-Information (D2I) Team is part of TFS Service Organization that designs, implements, and maintains an infrastructure for extracting EM operational data such as health and algorithm precision in order to improve the observability of the EM solutions.

4. **DMP Ops Team**
   The DMP Ops Team is part of TFS Service Organization that is in charge of remotely maintaining the DMPs and performing the managed services support procedures.

5. **Customer**
   The customers use the EM and the DMP as a single solution, which is set up at their premises. The availability, health condition, and visibility of the TFS solutions are reported to the customers via visual charts on the central dashboards. A connected customer has an Internet connection between their premises and TFS site, while a disconnected customer does not provide an outbound or inbound connection.

4.4 Use Cases

Table 4.1 presents an overview of the main use cases that were defined, planned, and implemented during the project.

Table 4.1: Overview of use cases

ID	Title	Description
UC 1	Enabling the system	DSE enables TFS to acquire diagnostics data from the customer's site in a centralized dashboard
UC 2	Configuring the metrics	Application Domain Expert makes their metrics observable for the central dashboard users
UC 3	Configuring the events	Application Domain Expert makes their events observable for the central dashboard users
UC 4	Configuring the connection	DSE adapts the system to meet the connection specifications between the customer sites and TFS
UC 5	Maintaining proactively	DSE responds to an incident in a proactive way, earlier than the customer notices a problem
UC 6	Reporting to connected customers	The connected customer sees the status of the DMP in the customer portal
UC 7	Reporting to disconnected customers	The disconnected customer sees the status of the DMP on the on-premise dashboard
UC 8	Developing central dashboards	DMP Ops Team visualizes the metrics and events of choice on the central dashboard
UC 9	Analyzing diagnostics logs	D2I Team analyzes diagnostics logs to provide visibility to the customers

5 Feasibility Study

Following the requirements elicitation and analysis of the problem domain, we started exploring the solution space in the feasibility study activities. The goal of the feasibility study was to identify the general direction of the solution and propose design criteria and alternatives.

This chapter presents the process and findings that set the basis for the design and implementation of the solution.

5.1 Functional Decomposition

The CM solution must bridge the gap between several on-premise deployments of the DMP instances at the customer sites and a central dashboard, which needs to be accessible by the TFS Service Organization. The system must realize the flow of the information as described in Figure 5.1 and it must operate according to the functional and non-functional requirements, described in Chapter 4.

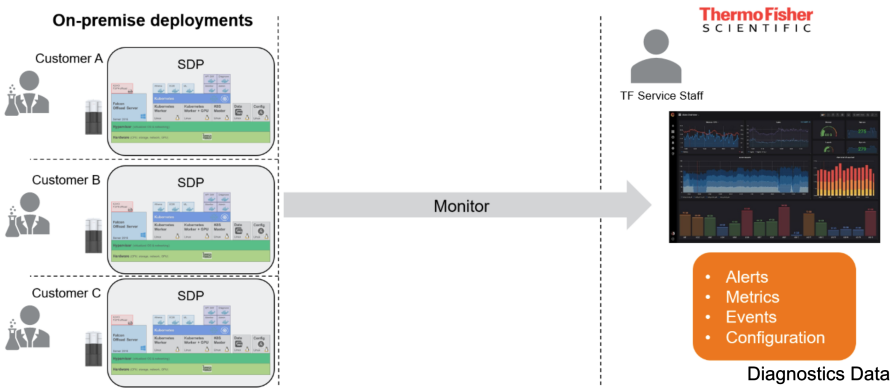

Figure 5.1: Monitoring of the multi-cloud on-premise deployments

To find a solution that can satisfy the project requirements, we compared existing systems in the monitoring domain. In a comparative study, we identified some candidate systems that could potentially fit into our problem space. Table 5.1 summarizes the candidate systems and highlights the most significant incompatibilities with the requirements of the project, which explains why the candidate system cannot be used as-is.

By summarizing the comparative study, we conclude that none of the candidate systems satisfies the requirements and constraints of the project. Therefore, we need to design and implement the CM

Table 5.1: Comparison of existing monitoring solutions

Candidate System	**Description**	**Inconsistency**
Sematext	Centralized application log and performance monitoring, alerting, and anomaly detection [21]	Proprietary software, requires direct connectivity between the premises and the TFS site, conflicting with REQ 2 and REQ 3
HPE InfoSight	Managing infrastructure performance and uptime in the data center. Powered by Artificial Intelligence (AI) platform to predict and prevent problems before they arise across an infrastructure stack [22]	Limited to server infrastructure monitoring, conflicting with REQ 201
AppDynamics	Application performance monitoring and IT operation analytics [23]	Limited to application-level monitoring, conflicting with REQ 5. Requires direct connectivity, conflicting with REQ 3
Splunk	Searching, analyzing, and visualizing the machine-generated data gathered from the websites, applications, sensors, and devices that make up the IT infrastructure of a business [24]	Proprietary software, conflicting with REQ 1 and REQ 3

system such that it fits TFS requirements.

We started the design process by breaking down the CM system into subsystems that perform parts of the system functions. The diagram in Figure 5.1 and the analysis in section 2.1.2 suggest a functional decomposition of the system.

The CM system is composed of three main subsystems, as displayed in Figure 5.2. Each subsystem can be designed and implemented separately and in parallel, provided that it adheres to the integration interfaces. Furthermore, it is possible to select different parts and components for each subsystem and connect them such that they comprise the entire system.

Table 5.2: CM subsystems

Subsystem name	**Functions**
Central Monitoring Connect (CMC)	Collecting the diagnostics data from customer's on-premise sources and preparing them for exporting
Transport Subsystem	Transporting the diagnostics data via a flexible and configurable connection channel to a central destination
Central Dashboard	Visualizing the diagnostics data and providing actionable insights to TFS Service Organization staff

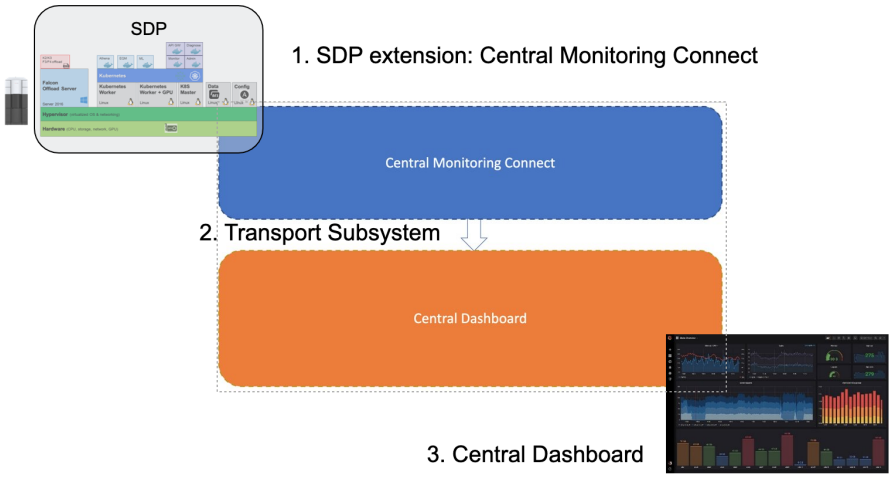

Figure 5.2: Diagram displaying the main subsystems of the system

We explain the selected strategy for the realization of each subsystem in the rest of this section.

5.1.1 Central Monitoring Connect

The Central Monitoring Connect (CMC) is the subsystem that directly connects to the SDP. It is an extension of the SDP that enables it to collect diagnostics data from the hardware, software, and applications. This subsystem also processes the data in order to prepare them for exporting to a central dashboard. The process includes the selection of important data, filtering the unneeded parts, adding important tags, adapting the data to an exportable format, and temporarily storing them until they are exported.

The SDP has a requirement on the method for publishing hardware, software, and application diagnostics data. Two databases are available for ingesting data, based on their type. Based on the requirement REQ 5, given in Section 4.1, metrics and alerts are found in the Prometheus system, and events are stored in the Elasticseach database.

Prometheus and Elasticsearch comprise SDP's on-premise monitoring facilities:

- Prometheus: Contains metrics and alerts as numerical and time-series data from several applications that adhere to the OpenMetrics standard [19]
- Elasticsearch: Contains text logs from several applications, processed by Fluentd [14], [15]

The CMC must take the data from SDP's on-premise monitoring sources and route them to the Transport subsystem.

Due to the diversity of the input sources and the variety of Transport mechanisms, the consuming and producing of the data need to be decoupled by utilizing the publish-subscribe (a.k.a. producer-consumer) design pattern. Amazon Web Services defines publish-subscribe design pattern as a form of asynchronous service-to-service communication which can be used to enable event-driven architectures or to decouple applications in order to increase performance, reliability, and scalability [25].

Table 5.3 lists and compares the advantages and disadvantages of several approaches for designing the CMC. As a result of the analysis and comparison of the possible technology choices, the Kafka broker is selected as the backbone of the CMC.

Table 5.3: CMC design choices

Approach	Description	Pros	Cons
Ad-hoc application	Developing an application that directly routes the input sources to the Transport subsystem, without using the publish-subscribe design pattern	Simplicity, Rapid Development	Violating NFR 2 (Extensibility) and NFR 3 (Configurability)
Kafka broker	Using a Kafka Cluster and developing distributed producer and consumer applications that rely on it for coordinating the data exchange	Supporting NFR 2 (Extensibility), NFR 3 (Configurability), NFR 5 (Resilience), and NFR 6 (Scalability)	The complexity of deploying a Kafka ecosystem, Learning curve
RabbitMQ	Using a RabbitMQ messaging server and developing producer and consumer applications that rely on it for coordinating the data exchange	Supporting NFR 2 (Extensibility), NFR 3 (Configurability), and NFR 5 (Resilience)	Learning curve, Difficulty of incorporating multiple consumers of a single message

5.1.2 Transport Subsystem

The Transport subsystem bridges the gap between the CMC and the Central Dashboard. This subsystem provides an abstraction of connection mechanisms that link TFS infrastructure and that of customers at their sites.

During the lifetime of a DMP, the connection capabilities might change. Therefore, it is important to design the system to be able to switch between connection mechanisms – even if there is only one connection type at a given moment.

We consider the following two connection methods that the system must support:

- Real-time connection: A connection type that delivers one message at a time within seconds from the customer's site to a central destination, such as Remote Procedure Call (RPC)
- Batch-mode connection: A connection type that delivers a group of messages at variable intervals, such as Simple Mail Transfer Protocol (SMTP)

In addition to the two main connection types, the qualities of the connection channel might also vary over time. These variables include factors such as the bandwidth and the availability window

of the connection channel. The handling of all the variabilities of the connection channels is the responsibility of the Transport subsystem.

TFS Service Innovation group has developed a toolset, Data Collector (DC), that has similar functions to those of the Transport subsystem for delivering microscope data from the customer's site to a central destination. Given the comparison and trade-off described in Table 5.4 we decided to reuse the DC component for addressing the connectivity concerns.

Table 5.4: Transport subsystem design choices

Approach	Description	Pros	Cons
Ad-hoc application	Developing an application that supports real-time and batch-mode connection types and provides a unified interface	Fully customized	Long development time
Data Collector	Adapting the DMP diagnostics data to the Data Collector API and using its pipeline to route the data to the existing visualization platforms	Low development effort, no additional cost for provisioning infrastructure, possibility to integrate with Service Organization workflows	The complexity of adapting to the Data Collector API, dependency with the Data Collector development team

5.1.3 Central Dashboard

The Central Dashboard visualizes the diagnostics data and displays them as charts, diagrams, and tables that ThermoFisher's DMP Ops Team can use in order to perform maintenance or use the historical data to make a decision.

With the growing trend of the microservices and cloud computing paradigms, numerous dashboarding tools and services have emerged in the monitoring domain. There are many open-source products, as well as several proprietary cloud-hosted services that enable IT service departments to visualize data with simple operations.

Since the Central Dashboard is the most visible part of the system and the component which the most stakeholders interact with, the decision must be accepted in NPD as well as Service Organization. Aspects to consider for deciding on this area are not only related to the technical specifications and user experience but are financially and organizationally motivated as well.

Historical analysis is one of the major benefits that a central dashboard offers. It implies that the data must be kept for a long period and must sustain incidents such as software and hardware upgrades and service outages. Therefore, the decision about whether TFS should implement and maintain a central dashboard infrastructure or reuse an existing one is a key factor that determines the project development direction.

Table 5.5 presents the two directions and their significant impacts on the project.

Table 5.5: Central Dashboard design choices

Approach	Description	Pros	Cons
In-house central dashboard	Deploying open-source components such as Grafana on ThermoFisher Scientific's infrastructure and maintaining the historical data using the company's resources	Rapid development using existing components, no cost for purchasing open-source components	Additional costs for maintaining the system, additional development expertise for future support
Central dashboard as a service	Adapting the DMP diagnostics data to the Data Collector API and using its pipeline to route the data to the existing visualization platforms	Low development effort, no additional cost for provisioning infrastructure	Complexity of adapting to the Data Collector API, longer development time for development

During the feasibility study of the project, we decided to use both approaches in the project for the following purposes:

- Central dashboard as a service: This is the solution that TFS will eventually need. It is important to research and prove the possibility of integration of existing infrastructure with the DMP monitoring solution.
- In-house central dashboard: This is the solution to use for the demonstration and proof of concept, without waiting for the possible dependencies. It also helps to build prototypes quickly and to collect feedback in interaction with the stakeholders, by using a visual aid.

5.2 Design Criteria

During the feasibility study, the general approach for driving the project was chosen and the first functional decomposition identified the subsystems that comprise the CM system. Additionally, some alternatives were listed and evaluated for each subsystem component.

This section summarizes the design criteria that we developed during the feasibility study, listed below:

1. Decoupling of the producing data sources
2. Extending in order to incorporate future data sources
3. Adapting to variable connection constraints

4. Reusing company's infrastructure as much as possible
5. Integrating with Service Organization's tools and processes

5.3 Summary

The result of the feasibility study identifies the general approach for developing the solution. It concludes that the CM system is functionally decomposed into three main subsystems and within each subsystem, there are design alternatives to choose from. As a result of the study and analysis of the alternatives, we specified the strategy for each subsystem. The key decisions for each subsystem are the cornerstones for further detailed design and implementation of the entire system.

The outcome of the feasibility study is presented in Table 5.6.

Table 5.6: Selected approaches for each subsystem

Subsystem	**Selected design approach**
Central Monitoring Connect	Developing data pipelines by using the Kafka ecosystem and producer-consumer design pattern
Transport Subsystem	Reusing the Data Collector toolset and adapting diagnostics data to its API
Central Dashboard	1. Reusing a central dashboard service as a long-term solution 2. Developing an in-house dashboard as a short-term solution for the proof of concept

6 Solution Description

Following the analysis presented in Chapter 5 (Feasibility Study), we continue by describing the CM solution, starting from the high-level decisions and finishing with the selection and implementation of the components.

This chapter also describes the integration and deployment of the system and the achieved results in TFS laboratories, where the EMs and DMPs are installed and in operation.

6.1 Architecture

The high-level architecture of the system, based on the functional decomposition, is displayed in Figure 6.1.

The CMC is the main component that enables an existing SDP instance to join the CM domain. Once the component is installed, configured, and enabled, it reports the diagnostics data that flow via pipelines of the infrastructure and finally reaches the central dashboards.

In order to skip the installation of the infrastructure separately, the SDP main package includes the CMC as a default component. As a result, once the SDP is installed, the DSE can decide whether to enable the CM as a feature or not.

The diagram in Figure 6.2 depicts the system design of the infrastructure with its significant internal components and data pipelines.

Each component of the infrastructure is realized by choosing one of the reuse, adapt, and implement strategies in order to complete the design described in Figure 6.2.

- **Reuse** a component when its specification, interfaces, and functions match those required by the system design
- **Adapt** a component when it contains most or similar functionalities required by the system design but adjustments are needed in order to fit in the design
- **Implement** a component when the required functions are not available in an open-source component or it does not match the design criteria

Each component's role description and its technical properties are included in Section 6.2.

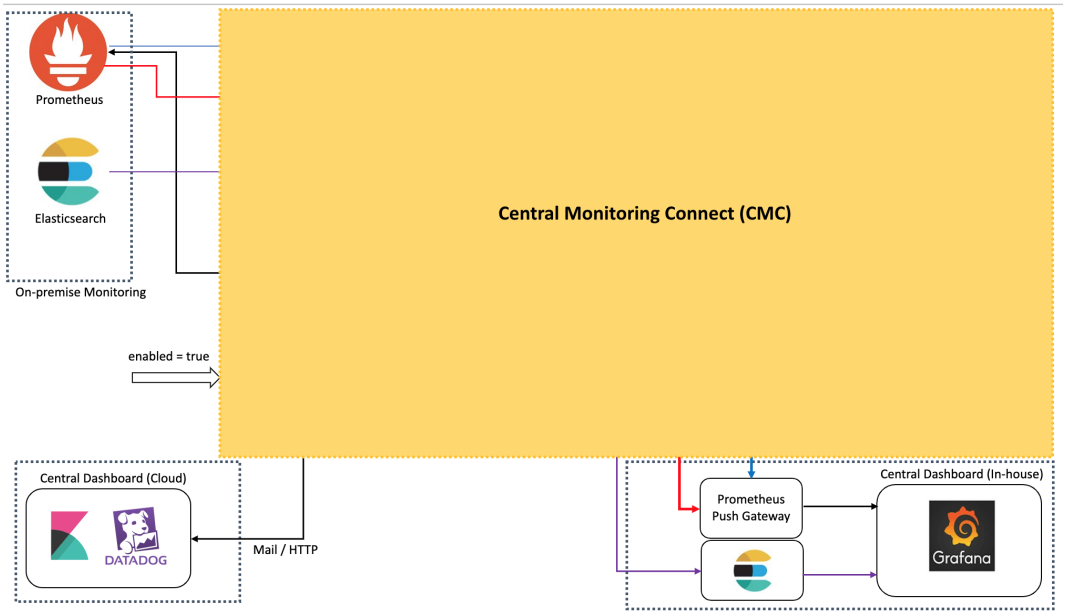

Figure 6.1: High-level components of CM system

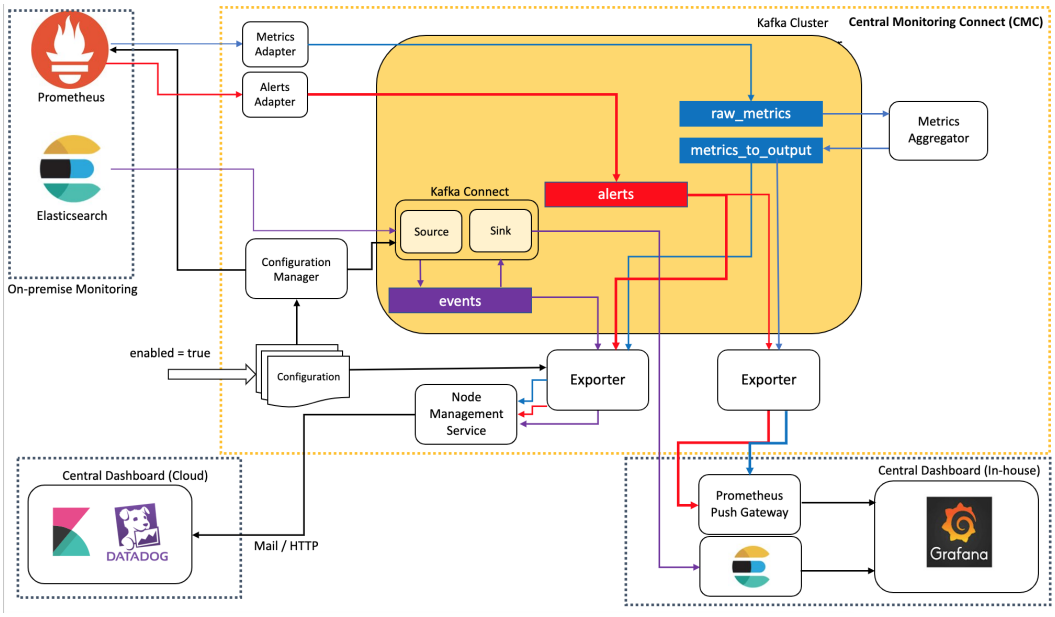

Figure 6.2: Internal components of CMC

6.2 Components

The CMC consists of the components described in Table 6.1.

Table 6.1: Components of CMC

Component name	Realization strategy
Kafka Cluster	Reuse
Kafka Topics	Reuse
Prometheus Kafka Adaper	Adapt
System Configuration	Implement
Configuration Manager	Implement
Kafka Connect	Reuse
Metrics Aggregator	Implement
Exporter	Implement
Node Management Service	Reuse

6.2.1 Kafka Cluster

Section 5.3 states that we selected the Kafka broker technology as the backbone of the CMC. Apache Kafka is a distributed streaming platform that is generally used for two classes of application [3]:

- Building real-time streaming data pipelines that reliably get data between systems or applications
- Building real-time streaming applications that transform or react to the streams of data

Figure 6.3 shows the anatomy of a generic Kafka Cluster setup. Each role of an entity in the anatomy corresponds to a component in our system design, which is described in the upcoming sections.

In the realization of this project, we have use cases that fall within both classes of applications that are introduced in [3]. The properties of the Kafka platform cover several requirements that originate in business needs. Table 6.2 describes the correspondence of Kafka platform features and the requirements of the CM system.

6.2.2 Kafka Topics

In a Kafka Cluster, a topic is a category or a feed name to which records are published [3]. A record (a.k.a. message) represents a single piece of data that a source emits and that can be processed separately. Topics are useful for logically organizing records such that producers and consumers can coordinate their tasks using the topic.

The CM system uses four topics to organize all the diagnostics data:

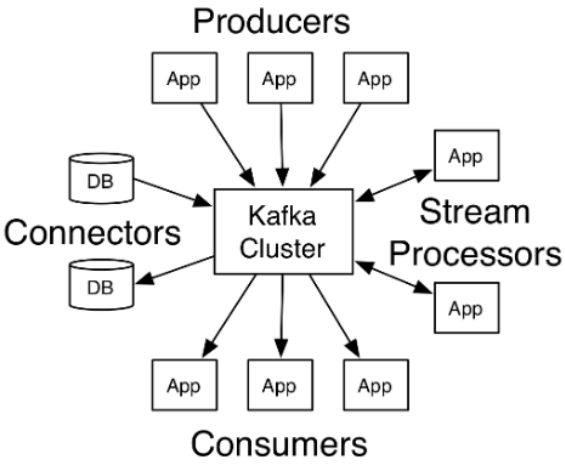

Figure 6.3: Anatomy of a Kafka Cluster and its usage [3]

Table 6.2: Correspondence of Kafka platform features and system needs

System need (Problem domain)	**Kafka feature (Solution domain)**
The need for receiving data from multiple sources	Ability to accept multiple producers
The need for the different priority of diagnostics data	Ability to collect data in different topics
The need for low coupling between system components	Inherent publish-subscribe design pattern
The need for using the same records of data for various purposes	Ability to allow multiple consumer groups per topic
The need for fault-tolerance	Possibility of a multi-node setup to achieve high availability and recovering from failures without loss of data and state
The need for scaling up	Possibility of a multi-node setup to achieve scalability

- **Raw Metrics** topic contains a subset of the metrics that the Prometheus instance emits. The Metrics Kafka Adapter is responsible for converting the original metrics data into JSON records that are collected in this topic.
- **Metrics-to-output** topic contains the metrics that represent a useful summary based on the raw metrics. The Metrics Aggregator calculates aggregated (such as maximum, minimum, and average) values of the metrics and stores them in this topic, which are finally exported to the central dashboards.
- **Events** topic contains the textual data that various SDP applications produce. The data represent a specific incident that happens in the SDP as a result of user interaction, such as an image observation or an internal action, such as a software reboot.
- **Alerts** topic contains indications of faulty incidents that happen in the SDP and Prometheus records them. The Alerts Kafka Adapter is responsible for converting original alerts into JSON records that are collected in this topic.

The reasons for using different Kafka topics in the system are:

- Organizing each data type, which results in logically distributing application roles. Therefore each application processes specific data types and promotes system decoupling.
- Storing each topic for a different duration, which helps to prioritize the data types. As a result, the important data are stored for a longer time than the less important data, which can be removed to save storage space.
- Processing each data type from a dedicated topic, which results in efficient use of the transport channel. The high-priority topics are exported earlier than the low-priority topics.

6.2.3 Prometheus Kafka Adapter

The metrics and alerts data that DMP applications produce are stored in the customer's on-premise Prometheus server. Prometheus uses an internal time-series database to store the metric and alert samples and can also use third-party databases. An integration interface, named Remote Write, allows reliable duplication of the samples from Prometheus to external storage.

The CM system leverages the Remote Write capability in order to export the samples from the Prometheus database to the Kafka Cluster, which stores them temporarily for further processing.

The adapter is responsible for adjusting the samples' data model while exporting them from Prometheus to Kafka. The Prometheus Kafka Adapter component acts as third-party storage for the on-premise Prometheus and routes the metrics and alerts to a Kafka topic while converting them to JSON records in the process.

Additionally, in order to distinguish the metric sources in the central dashboard, the adapter adds a tag that identifies the DMP that originates the metric values. Each DMP is identified by a unique name that TFS Service Organization chooses when the DMP is installed at the customer's site. By adding the DMP identifier as a tag on each metric and alert sample, it is possible for the CM system to trace its values from the central dashboard to a customer's DMP system.

The converting and tagging process that the adapter performs is displayed in Figure 6.4.

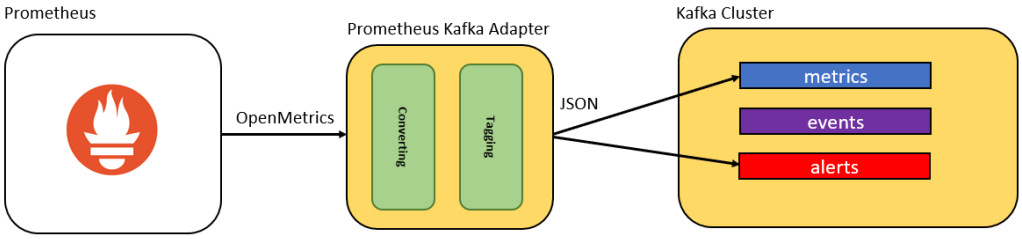

Figure 6.4: Process of adapting Prometheus data to Kafka

The Prometheus Kafka Adapter component is also used for converting metrics as well as alerts into Kafka records and routing them to the Kafka Cluster. By reusing the component, we create two instances of the component and divide the responsibility of processing the metrics and alerts to each individual instance.

6.2.4 System Configuration

As presented in Section 4.1, the system must alter its behavior dynamically. The ability to adapt the system behavior is one of the important aspects of microservices architecture. According to the microservices best practices, it is advised to disaggregate the configuration of applications from the source code, so that one can change the configuration easily and without building the source code [26].

The CM System Configuration contains a description of the metrics and events that must be exported from the DMP to the central dashboards.

Metrics configuration is composed of a regular expression, a set of aggregation functions, and an aggregation window. The system selects the metrics that match the regular expression and applies the aggregation function to all the metric samples that fall within the specified window.

Events configuration is composed of an Elasticsearch query and a target Kafka topic. The system uses the Elasticsearch query to discover the desired events and replicates the events to the described Kafka topic.

Kubernetes offers the Config Map mechanism for implementing the external configuration [27] of software applications. The Config Map is a Kubernetes resource that offers key-value mapping for storing configuration options of applications that run on Kubernetes infrastructure.

The CM System Configuration is implemented as a Kubernetes ConfigMap resource and uses a JSON syntax in order to describe the desired metrics and events configuration, as well as the option for enabling and disabling the system.

6.2.5 Configuration Manager

The System Configuration described in Section 6.2.4 is a text-based resource that embodies the desired behavior of the CM system. In order to apply the configuration, the Configuration Manager component processes the system configuration resource and updates other parts of the system to adjust their

behavior.

The configuration manager is implemented as a Kubernetes Cron Job. A Cron Job is an application that activates on specific periods and performs a single task, which is adapting of the CM system to the system configuration description [27].

6.2.6 Kafka Connect

Kafka Connect is a tool that makes it simple to define connectors that move large data sets into and out of Kafka Cluster [4]. It is a scalable and reliable tool for working with various input and output data sources in standard ways. Figure 6.5 demonstrates the relation of Kafka Connect for moving data from an external source to a Kafka Cluster and from the cluster to an external destination.

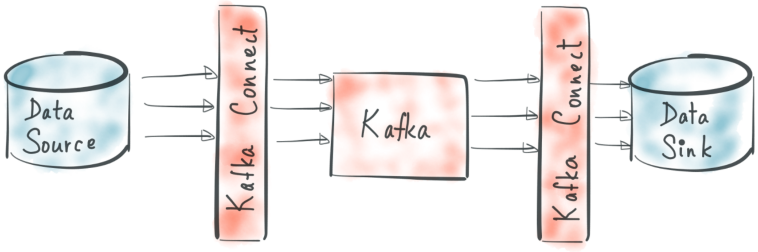

Figure 6.5: Usage of Kafka Connect for moving data to and from Kafka Cluster [4]

The input and output data plugins are called source and sink connectors, respectively. Each connector is implemented as a Java Archive (JAR) that once deployed in the Kafka Connect server, triggers a continuous task for fetching data into and out of Kafka topics.

The advantage of using the Kafka Connect is that all the connectors share a single management API and it is possible to programmatically automate the activation of the connectors. The Configuration Manager component relies on this property for applying the configuration of the event.

A source connector makes it possible to extract events data from Elasticsearch and move it to the events pipeline. Similarly, a sink connector moves the events from a Kafka topic to the in-house central dashboard.

6.2.7 Metrics Aggregator

Metrics are numerical values that describe the performance of certain software and hardware parts in the system. The generation of metrics in the DMP follows a streamlined pattern and the on-premise Prometheus instance records the metrics. A subset of the recorded metrics is forwarded to the CM system via the Configuration Manager, which opens the gateway and populates the Kafka metrics topic with the metric samples.

One of the key differences between the on-premise and CM dashboards is that while the on-premise monitoring dashboard captures all the generated metric samples, the CM system processes a subset of the samples due to limited connection bandwidth. Therefore, it is crucial to recreate a meaningful subset of metric samples and use them in the central dashboard as representatives of the original metric

samples.

The Metrics Aggregator component is responsible for subsampling the metrics and generating of the aggregated metrics that while are meaningful for the monitoring purpose, are smaller in size. The metrics aggregation process starts with reading all the incoming metrics that are available in the metrics Kafka topic and filtering only those that must be aggregated according to the configuration. Afterward, each metric sample is assigned to a bucket that collects the samples of a time window. Finally, the Metrics Aggregator applies a number of functions to the sample buckets and stores the result of the calculation in the output metrics topic of the Kafka Cluster. Figure 6.6 displays the steps for calculating the aggregated metrics.

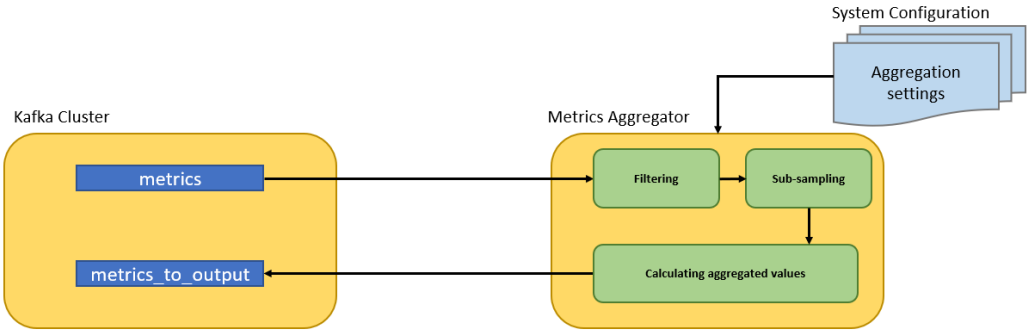

Figure 6.6: Metrics aggregation process

The Metrics Aggregator component is developed using NodeJS technology and uses the Kafka Streams API in order to continuously process metric samples.

6.2.8 Exporter

The Kafka Cluster temporarily stores metrics, events, and alerts data in dedicated topics from the moment various data sources publish diagnostics data until the data is exported to the destination. The Exporter component is responsible for reading all the diagnostics data and adapting them to the format that is required by the destination dashboard.

Figure 6.7 demonstrates the main functions of the exporter component.

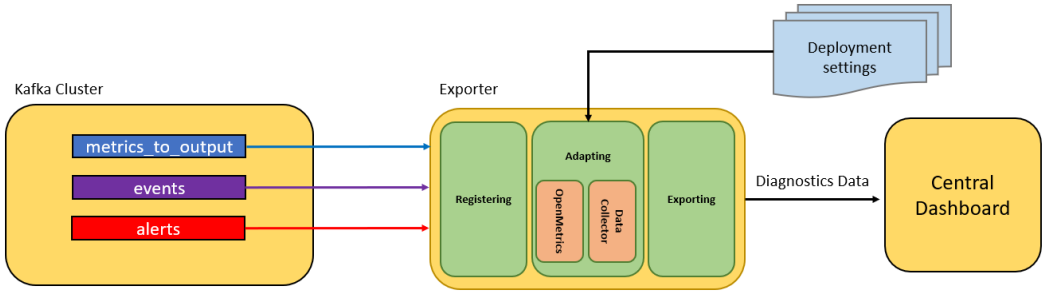

Figure 6.7: Metrics exporting process

In order to receive the diagnostics data, the Exporter subscribes to the Kafka topics and registers

the incoming metric samples, events, and alerts. The deployment settings control the behavior of the exporter component and the mechanism that adapts the data to the required data format of the destination. There are two operational modes for adapting the diagnostics data: OpenMetrics and Data Collector (DC).

Adapting to the OpenMetrics format makes the diagnostics data visible on the dashboarding systems that are based on the Prometheus system. On the other hand, the Data Collector data format is compatible with the proprietary infrastructure designed and implemented by the TFS Service Innovation Team.

After the diagnostics data is adapted to the desired format, the component exports data to the external destination.

6.2.9 Node Management Service (NMS)

The Node Management Service (NMS) is a component developed by the TFS Service Innovation Team. The component facilitates the delivery of the diagnostics data from on-premise deployments to central storage, in order to visualize using dashboards.

NMS has been designed to process EM structured data that can be administered centrally. However, due to its ability to use a direct connection channel as well as the SMTP email channel, it is a suitable candidate to reuse for the purpose of delivering DMP diagnostics data.

By including NMS in the CM system and using its pipeline, it is possible to integrate DMP monitoring infrastructure with the infrastructure that the Service Organization is using. This design choice eliminates the need for allocating additional human and financial resources for provisioning the infrastructure and learning the skillset for working with dashboards.

6.3 Integration

The CM system is composed of various components described in Section 6.2. Each component is a software artifact that can execute independently. Similar to all the SDP-compatible applications, the CM system is deployed and executed on the Kubernetes platform. Therefore, a Kubernetes namespace encompasses all the components of the system that are implemented and deployed as Kubernetes resources.

Table 6.3 presents the selected Kubernetes resource type for implementing each component of the system.

The various components (deployed as Kubernetes resources) are integrated and treated as one system. The SDP applications use Helm technology to manage Kubernetes applications. Helm is a package manager that helps one define, install, upgrade, and rollback complex Kubernetes application [28]. Helm packages are called Charts. The CM Helm chart contains all the components that are defined as Kubernetes resources and ties the input and output of the components in order to form an integral system.

Table 6.3: Kubernetes resources of CM system components

Component name	Kubernetes resource type	Rationale
Kafka Cluster	Stateful Set	Each broker that is a member of the Kafka Cluster is a stateful application.
Kafka Topics	Persistent Volume	The diagnostics data must persist when the system is undergoing restart and upgrade processes.
Prometheus Kafka Adapter	Deployment	Prometheus Kafka Adapter is a stateless application.
System Configuration	Config Map	The system configuration must be updated and adjusted during the run time of the system, without the need to modify the source code.
Configuration Manager	CronJob	The system configuration must be applied at regular times, as well as per demand of the DSE.
Kafka Connect	Deployment	Kafka Connect is a stateless application.
Metrics Aggregator	Deployment	Metrics Aggregator is a stateless application.
Exporter	Deployment	Exporter is a stateless application.
Node Management Service	Deployment	NMS is a stateless application.

6.4 Deployment

In order to deploy the CM System, the Digital Service Engineer (DSE) needs to perform the deployment procedure of SDP family products. The deployment procedure requires that all the resources be packaged as a single installation file (ISO image) that can be shared via network.

The installation file contains scripts that automate the process of deployment, which is consisted of the following steps:

1. Checking the system requirements
2. Preparing the environment prerequisites such as creating storage space
3. Copying the required Docker images and Helm charts to the DMP machine
4. Installing or upgrading the CM Helm charts

The deployment steps are executed by a set of Bash scripts that accompany the CM Helm charts, which make the deployment process simple for the DSE. Additionally, the test and build pipeline (CI) uses the scripts to automatically create packages and deploy them on a testing server after a version of the CM system is ready to be tested.

Once the CM system is deployed, it can be enabled by editing the System Configuration and triggering the Configuration Manager. Following this final step, the metrics, events, and alerts will appear in the central dashboards.

7 Project Process

This chapter describes the processes, tools, and methods that were used during the life cycle of the project.

The ISO 12207 standard defines a set of technical and management activities for the evolution of a software system, starting with the initiation and finishing with the disposal stages of typical software systems [29].

In order to establish technical and management processes for the project, we considered not only the ISO 12207 standard but also noted that TFS product development processes are based on Agile methodologies such as Scrum [30]. Therefore, in designing the project process, we used the standard as a guiding document, while tailoring the process to the TFS Scrum methodology in order to perform the activities in an iterative manner. The combined approach allowed us to define the long-term deliverables and estimate the plan for completing the deliverables and gain benefits from implementing short-term iterations and quick feedback loops.

7.1 Deliverable Planning

The outcome of the project is presented by three deliverables:

- Solution Prototype, in order to demonstrate the feasibility and value of central monitoring in the context of managed services delivery
- Solution Architecture, in order to transfer the project knowledge to TFS for future development
- Technical Report, in order to present the process and results

At the beginning of the project, we identified the use cases listed in Section 4.4 that the Solution Prototype must cover. Despite the ordinal listing of the use cases, the implementation process did not cover the use cases sequentially. We divided the use cases into Stories, which are units of planning in Scrum methodology, and implemented the Stories in iterations. This approach allowed us to implement the use cases in parallel and incrementally so that at the end of each iteration the system covered an extended scope of each use case.

In monthly meetings with the stakeholders, we prioritized each use case, considering:

- Significance of the use case from the proof-of-concept point of view
- Architectural impact of implementing the use case; the use cases with higher impact were given higher priority

- Dependency with other parties; to mitigate possible unforeseen risks, the use cases which involved interaction and communication with external teams were preferred to start early

The deliverables and their break-down structure were maintained in an Excel spreadsheet along with their status and envisioned completion date, used for prioritization in collaboration with the stakeholders. We used the TFS Jira platform to hold all the Stories that were derived from decomposing the Solution Prototype and in order to track their progress status during the iterations.

At the end of each iteration, the status of the Stories and use cases were reviewed by TFS stakeholders and the Project Steering Group (PSG) members.

7.2 Reviewing Process

At the beginning of the project, the following recurring review meetings were set up:

1. Weekly project sync up with TFS supervisor, in order to review product requirements, propose and refine use cases, review technical implementation, and discuss architectural decisions
2. Biweekly sprint review and planning, in order to review the progress of Stories
3. Monthly PSG meetings, in order to review the long-term planning, evaluate risks and receive feedback
4. Biweekly sync up meetings with the Service Organization stakeholders, in order to review and refine use cases, plan the integration of SDP monitoring with Data Collector tools, and propose modifications to the components, if needed

In addition to the planned review meetings, there were also ad-hoc review sessions, where a design or implementation was discussed with internal team members.

7.3 Requirement Management

The requirements are registered and maintained in the Confluent platform, in order to easily share and discuss with the stakeholders. Each requirement is described using a unique number, title, description, importance, and rationale.

7.4 Communication Plan

During the stakeholder analysis, we identified the stakeholders, their needs and interests, the deliverables that are relevant to them, as well as their preferred communication method and frequency. At the beginning of the project, most of the communication with TFS stakeholders was conducted at the company site. However, due to the circumstances imposed by the COVID-19 pandemic, we used the Microsoft Teams software for holding remote meetings.

In order to communicate architecturally significant decisions with the stakeholders, we used lightweight system diagrams similar to UML component diagrams, which express the major components, their

roles, and relations to each other, without the need to refer to the UML syntax for interpreting them. These styles of diagrams, which we used Microsoft PowerPoint to develop them, proved to be effective not only in discussions with the NPD group but the Service Organization as well.

The system diagrams are stored on the TFS Microsoft OneDrive storage system and also exported to the Confluent platform to accompany the documented pages.

7.5 Risk Management

Table 7.1 shows the risks and their evaluated likelihood and impact at the beginning of the project. Each risk item has a minimum review frequency and once activated, the mitigation or contingency actions described in Table 7.2 were taken.

Table 7.1: Risk table and their corresponding likelihood, impact, and review frequency

ID	Description	Likelihood	Impact	Review Frequency
1	The solution might not land well on the Service Organization workflows	High	High	Biweekly
2	Integration issues and delays with DataCollector infrastructure	Medium	High	Biweekly
3	Escalation of the COVID-19 outbreak and becoming a pandemic	High	Medium	Monthly
4	Inefficient communication due to remote work	Medium	Medium	Daily
5	Delay in provision of 3rd party infrastructure	Medium	Medium	Monthly
6	Technical issues with the TFS development computer, IT department not acceissble due to the remote work	Low	High	Weekly

Table 7.2: Risk mitigation and contingency plan

ID	Mitigation	Contingency
1	Engage stakeholders via interview and demos in order to gather requirements and negotiate	Provide reusable components
2	Start early, to set up a skeleton	Develop a parallel solution with well-defined interfaces
3	N/A	Adopting remote-working procedures in order to keep the productivity at a high level
4	More one-to-one communication	Use more written communication
5	Evaluation of the available infrastructure early, work with the D2I team closely to provide feedback and highlight the requirements	Reduce dependency in the design by using existing infrastructure (e.g. Grafana) and providing technology-independent interfaces
6	Follow maintenance guidelines, install VPN on another computer, use remote repositories for source code and documents	Use another computer which can access the repositories and has the necessary toolset

8 Verification and Validation

We describe the Verification and Validation (V&V) methods used during the development of the Central Monitoring System.

8.1 Verification and Validation Model

The V-Model is an approach for planning and implementing system development projects. In the V-Model, the verification and validation steps are defined at different levels of the system, as it evolves during the Software Development Life Cycle (SDLC). The emphasis is placed on verification on the left-hand side of the V and validation on the right-hand side, using test cases to ensure adherence between the development activities [31]. Figure 8.1 displays how we used the V-Model in order to define the system level and the validation methods used for each level.

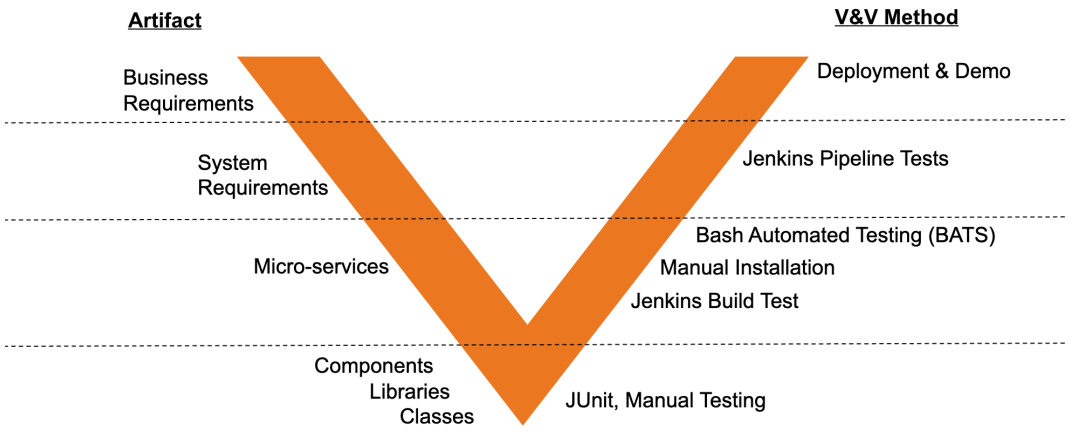

Figure 8.1: Verification and validation using the V-model

8.2 Verifiable Artifacts

During the SDLC, we verified each of the artifacts listed on the left side of the V shape by interactive reviews and discussions. Upon completing an artifact and delivering it to increment the system's version, a validation step was also taken. Table 8.1 defines each level of the artifacts and explains the V&V methods used for each.

Table 8.1: The V&V artifacts and used methods

Artifact	V&V methods
Business Requirements: The requirements and use cases that bring added value to the NPD, Service Organization, and Customers	1. The system is deployed in the TFS lab environment and made available for the employees to interact with. 2. Live demonstration sessions are organized regularly to showcase the features of the system and receive feedback for further developments.
System Requirements: The functional and non-functional requirements of the system, as described in Chapter 4	A set of Jenkins pipelines validate the compatibility of the system with the SDP in automated build, install, and upgrade scenarios.
Microservices: The set of integrated and coordinated components that perform the Central Monitoring System's functions	1. The entire system is automatically built and packaged as an ISO file. 2. The built ISO package is installed manually on lab DMP machines. 3. Bash Automated Tests validate the availability and functionality of the system once it is deployed.
Components, libraries, and classes: The software parts that implement or adapt any of the components described in Section 6.2	1. JUnit tests validate the functionality of Java-based artifacts. 2. Components are tested by running on the development computer and a lab server and the functionality is manually validated.

8.3 Validation Tools

Besides the manual validation methods such as review and manual installation, we used a number of tools in order to automate the validation processes, which are listed below:

- **JUnit** is a unit testing tool for software classes and modules developed using Java. We used JUnit for testing the Kafka Source Connector that moves data from Elasticsearch to the Kafka Cluster, to make sure that the queries are performed correctly.
- **Bash Automated Testing System (BATS)** is a testing framework that provides a simple way to verify that a UNIX program performs as expected [32]. BATS is used for validating the integration of components in order to confirm the microservices are operational and can communicate with each other in order to fulfill the defined tasks.
- **Jenkins** is an open-source automation server that helps automate the building, deploying and testing of software [33]. We used Jenkins for automating the building of the CM system Helm charts and package all its related resources into a single ISO file and publishing the results to the TFS file server. In addition to the build step, Jenkins was also used for validating the integration

of the CM system with other SDP subsystems. A set of pipelines were triggered after releasing a new version of the SDP to perform the installation and upgrade use cases and assure that the candidate version is operational.

9 Conclusion

In this project, we presented a CM system that supports TFS in delivering managed EM services. We started by analyzing the environment needs and factors (Chapter 2), explored the domain (Chapter 3), and defined the system requirements that must fit the business needs along with the constraints that arise from the TFS business environment (Chapter 4). During the exploration activities, several candidate systems were compared and potential technologies that could lead to a suitable solution were reviewed (Chapter 5). We concluded the feasibility study by selecting the Kafka ecosystem as the backbone technology for stream-processing of the metrics and events, selecting and aggregating the diagnostics data, and using the DC subsystem to address the data transfer requirements. The system was further decomposed into components that were selected and reused, adopted with minor modifications, or implemented during the project. Finally, the components were integrated as a whole system that enabled TFS to monitor its DMP systems from a central view (Chapter 6).

During the project, we followed an evolutionary software development process, as described in Chapter 7, which led to a series of system versions that we tested with the V&V approach (Chapter 9).

Each version of the system was deployed on different DMP systems in TFS laboratories to demonstrate the benefits and capabilities of using central dashboards for delivering managed services. Frequent deployments during the project helped us discover several scenarios, such as various system configurations and backward compatibility considerations, thus bringing the prototype closer to adapt to the production environment.

9.1 Results

To prepare the CM prototype for production usage, it is included in the SDP toolset. Therefore, once a new DMP is deployed at a specific site, a DSE can follow specific steps and enable the CM system for that specific site. Afterward, it is possible to monitor the DMP via the central dashboard.

Although the system is designed to be configurable, there is a set of default data that describe the health status of each DMP, such as Kubernetes node loads, storage usage status and trend, GPU usage and temperature, and the status of deployed applications. As displayed in Figure 9.1, using the central dashboard, it is straightforward to choose a DMP system, by selecting its name, to investigate directly from a single view.

The DMP Ops Team can use the provided data as the baseline for extending and adapting it to express the meaningful set of diagnostics data that describe their specific use cases. Besides monitoring the metrics, it is also possible to define acceptable thresholds for each metric that describes the compliance with the SLAs that TFS guarantees to the customers. In case of a violation, an incident is recorded by the central dashboard and routed to the specified target group or member to take an action.

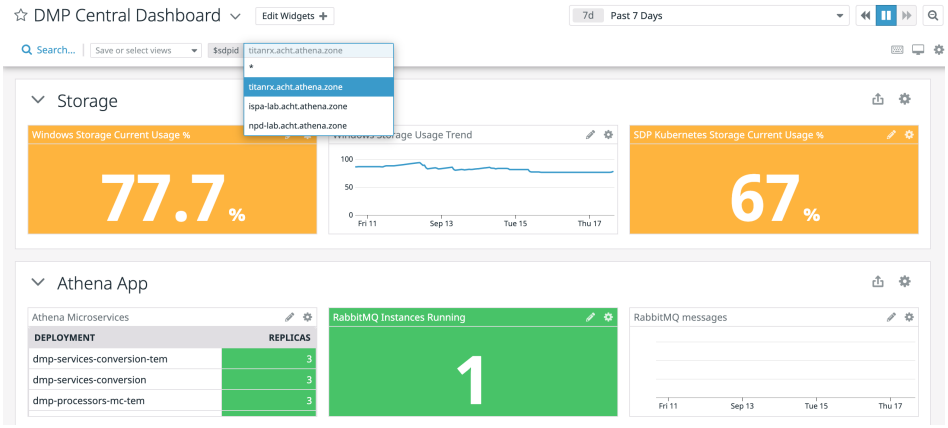

Figure 9.1: An example of DMP central dashboard

The CM system also allows a similar usage for monitoring the events, alerts, and configuration data that are extracted from the DMPs.

One of the valuable advantages of collecting the diagnostics data in a central dashboard over time is that it enables TFS to discover the usage patterns of the infrastructure resources. For instance, the DMP Ops Team can use prediction models, which are integrated into the dashboard, to estimate when the storage disk will become full and perform a cleanup procedure before it causes problems. Figure 9.2 shows an estimation of the time when the storage disk capacity is entirely consumed.

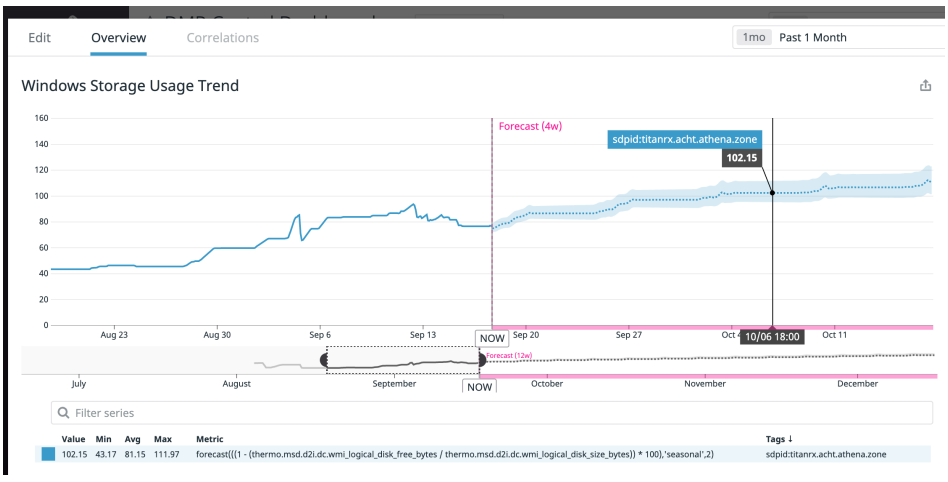

Figure 9.2: Prediction of storage disk usage

The CM system can also help NPD engineers observe the quality and status of the released SDP versions. When the NPD releases a new version of the SDP, it is deployed to a testing environment and several tests are performed or it is used by TFS staff to verify its quality attributes. NPD engineers can use dashboards similar to the examples of Figures 9.3 and 9.4 to verify that all the components of SDP are functioning correctly. A color-coded status indicator can quickly inform the engineers if there is an error or warning in a component of the system.

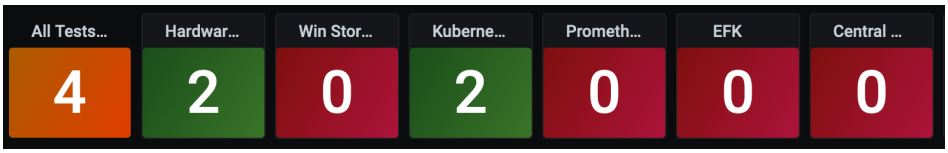

Figure 9.3: Status dashboard showing a faulty SDP deployment

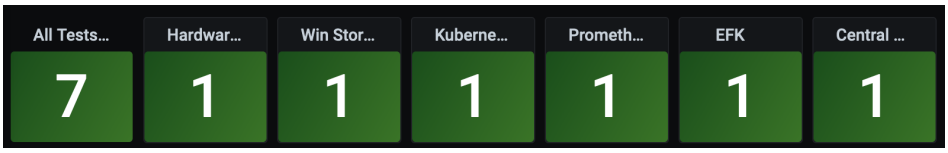

Figure 9.4: Status dashboard showing a healthy SDP deployment

9.2 Summary

The project was defined to bring TFS closer to its mission in offering managed services in the EM domain. One of the necessities to be able to offer managed services is a CM system, which brings observability to the SLA realization and sets the base for proactive maintenance. During this project, we designed and implemented a system that can address the TFS needs in the monitoring area. Using the system prototype, we reviewed the use cases and demonstrated how it can be used to fulfill them.

Using the system, it is possible to:

- Vizualise metrics and events data to overview the health status of the DMP systems as well as SLA realization
- Be notified upon a faulty incident in a remote DMP system
- Predict possible incidents based on the usage patterns and system status
- Use historical usage data to tailor the next application software products to customer needs

The CM system is an enabler infrastructure for TFS, especially its Service Organization, to depend on for laying out managed services delivery procedures. To use the CM system effectively, we recommend:

- Identifying the SLA data: SLAs are defined at the level of business requirements and describe the quality aspects such as availability and throughput that the customers expect from the EM solutions. As they are turned into system requirements and implemented in various components, several parameters and metrics evolve that can individually or collectively express whether the system is operating according to the expectations. These parameters can be identified with the help of the DSEs and ASG members who have the domain knowledge of the applications and their behavior.
- Configuring the CM system: The CM is designed to be dynamically adaptable. Several configuration parameters control the behavior of the system, such as the set of diagnostics data to be monitored as well as the connection mechanism. These parameters can be configured and applied during the runtime of the system. However, when a set of well-chosen parameters are

known and proven to be useful for the production, they can be included in the default parameter set so that the next versions of the SDP will include them by default.

Bibliography

[1] Public Health Image Library (PHIL). https://phil.cdc.gov/details.aspx?pid=23312. Accessed: 2020-09-15.

[2] K Young John Matson. *Monitoring Modern Infrastructure*. DataDog, 1st edition, 2018.

[3] Apache Kafka. https://kafka.apache.org. Accessed: 2020-09-15.

[4] Confluent Documents. https://docs.confluent.io/. Accessed: 2020-09-15.

[5] ThermoFisher Scientific – Brands. https://www.thermofisher.com/am/en/home/brands.html. Accessed: 2020-09-15.

[6] Cryo-EM Used in Novel Coronavirus Research to Support Vaccine, Treatment Development. https://tinyurl.com/y3nxhb8z. Accessed: 2020-09-15.

[7] Len Bass et. al. *Software Architecture in Practice*. McGraw-Hill, 2nd edition, 2007.

[8] SaaS vs. Software: The Release Cycle for SaaS is Usually (Not Always) Faster. https://tinyurl.com/yxhganc2. Accessed: 2020-09-15.

[9] Timothy Grance Peter Mell. *The NIST Definition of Cloud Computing*. NIST, 2011.

[10] What is a vSphere Hypervisor? https://www.vmware.com/nl/products/vsphere-hypervisor.html. Accessed: 2020-09-15.

[11] About CentOS. https://www.centos.org/about/. Accessed: 2020-09-15.

[12] Docker: What is a Container. https://www.docker.com/resources/what-container. Accessed: 2020-09-15.

[13] What is Kubernetes? https://kubernetes.io/docs/concepts/overview/what-is-kubernetes. Accessed: 2020-09-15.

[14] Elasticsearch. https://www.elastic.co/. Accessed: 2020-09-15.

[15] What is Fluentd? https://www.fluentd.org/architecture. Accessed: 2020-09-15.

[16] Kibana: Explore, Vizualise, Discover Data. https://www.elastic.co/kibana. Accessed: 2020-09-15.

[17] Prometheus. https://prometheus.io/. Accessed: 2020-09-15.

[18] Grafana. https://grafana.com/. Accessed: 2020-09-15.

[19] OpenMetrics. https://github.com/OpenObservability/OpenMetrics. Accessed: 2020-09-15.

[20] EARS – The Easy Approach to Requirements Syntax: The Definitive Guide. https://qracorp.com/easy-approach-to-requirements-syntax-ears-guide/. Accessed: 2020-09-15.

[21] Sematext: Cloud and Management Tools. https://sematext.com/. Accessed: 2020-09-15.

[22] HPE InfoSight. https://www.hpe.com/us/en/solutions/infosight.html. Accessed: 2020-09-15.

[23] Appdynamics. https://www.appdynamics.com/. Accessed: 2020-09-15.

[24] Splunk. https://www.splunk.com/. Accessed: 2020-09-15.

[25] What is Pub/Sub Messaging? https://aws.amazon.com/pub-sub-messaging/. Accessed: 2020-09-15.

[26] The Twelve-Factor App. https://12factor.net/. Accessed: 2020-09-15.

[27] Marko Luksa. *Kubernetes in Action*. Manning Publications, 2017.

[28] Helm. https://helm.sh/. Accessed: 2020-09-15.

[29] ISO 12207:2017. Systems and software engineering – Software life cycle processes. Standard ISO/IEC/IEEE 12207:2017, International Organization for Standardization, 2017. URL https://www.iso.org/standard/63712.html.

[30] What is Scrum? https://www.scrum.org/resources/what-is-scrum. Accessed: 2020-09-15.

[31] Tim Weilkiens. *Systems Engineering with SysML/UML*. Morgan Kaufmann Publishers Inc., 2008.

[32] BATS: Bash Automated Testing System. https://github.com/sstephenson/bats. Accessed: 2020-09-15.

[33] Jenkins. https://www.jenkins.io/. Accessed: 2020-09-15.

PO Box 513
5600 MB Eindhoven
The Netherlands
tue.nl

PDEng SOFTWARE TECHNOLOGY