Paving the path towards platform engineering using a comprehensive reference model

Ruben van de Kamp
ruvdkamp@gmail.com

July 28, 2023, 94 pages

Academic supervisor:	Dr. Zhiming Zhao, z.zhao@uva.nl
Daily supervisor:	Kees Bakker, k.bakker@hostedsolutions.nu
Host organisation/Research group:	Wehkamp, wehkamp.nl

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Abstract

Amidst the growing popularity of platform engineering, promising improved productivity and enhanced developer experience through an internal developer platform (IDP), this research addresses the prevalent challenge of a lack of a shared understanding in the field and the complications in defining effective, customized strategies. Introducing a definitive Platform Engineering Reference Model (PE-RM) based on the Open Distributed Processing reference model (ODP-RM) framework to provide a common understanding. This model offers a structured framework for software organizations to create tailored platform engineering strategies and realize the full potential of platform engineering. The reference model is validated by conducting a case study in which a contextual design and technical implementation guided by the reference model is proposed. The case study offers guidance in designing platform engineering in the context of a software organization. Furthermore, it showcases how to construct a technical platform engineering implementation, which includes experiments exposing the productivity improvements and applicability of the implementation. By facilitating a shared vocabulary and providing a roadmap for implementation, this research aims to mitigate prevailing complexities and accelerate the adoption and effectiveness of platform engineering across organizations.

Keywords: *Platform engineering, reference model, system modeling, cloud infrastructure, Platform as a Service, development lifecycle*

Contents

Chapter 1

Introduction

The digital age has initiated substantial transformations across all industries, making digitalization an imperative for survival in today's volatile market landscape [1]. In this rapidly growing software-driven world, the need for organizations to be innovative and improve their technological infrastructure, efficiency, and overall performance becomes more essential. This transition has amplified the importance of software development and deployment, with DevOps and Agile acting as instrumental drivers of this digital revolution [2–4]. Agile, advocating adaptability and customer collaboration [5], and DevOps, fostering cooperation between development and operations teams, enhance the efficiency and quality of software delivery [6]. Amid increasing technological complexity, platform engineering[1] is emerging as a promising discipline. It focuses on altering the engineering culture and creating an engineering platform, e.g., an Internal Developer Platform (IDP), to offer self-service capabilities for software development teams in the cloud-native era [7, 8]. Platform engineering refers to the design, development, and management of an engineering platform that supports the integration and interoperability of multiple components, applications, or services [2,3]. It involves the creation of robust, scalable, and modular platforms that facilitate streamlined collaboration, data sharing, and communication across various systems and teams. By adopting platform engineering principles, organizations can harness the potential of their technological assets and resources to deliver more value to their customers and stakeholders. Moreover, it includes organizational changes to enable development teams to work more efficiently, with the introduction of a platform team responsible for the engineering platform and related tooling. Yet, interpretation variations tied to personal perspectives and organizational contexts have rendered the definition ambiguous.

1.1 Problem statement

One of the key challenges software organizations face is the proliferation of tools, frameworks, and libraries, which leads to fragmentation and difficulty in selecting the most suitable technology stack. This problem has been exacerbated by the increasing adoption of methodologies such as DevOps, which emphasize rapid iteration, continuous integration, and continuous delivery [9]. With this culture, each team works independently, reducing the standardization across teams. The absence of clear adoption and integration guidelines intensifies this issue, contributing to an accumulation of technical debt that impacts system maintainability, scalability, and performance [10]. The increased tool diversity further compounds labor costs and hinders effective knowledge sharing and innovation. Furthermore, deploying applications consistently and reliably across diverse environments is another challenge within the current software engineering field. Although DevOps advocates for seamless integration between development and operations [9], managing complex dependencies and configurations across various teams and environments is demanding [11, 12]. Inconsistent environments pose a high risk of deployment failures that could result in service outages, lost revenue, and dissatisfied customers. The lack of standardization further exacerbates operational costs and impedes the ability to invest in strategic initiatives.

The current software landscape necessitates collaboration between different teams. This is further amplified by the adoption of methodologies such as DevOps and Agile, which advocate for seamless communication, continuous feedback, and joint decision-making among development, operations, and quality assurance teams [3, 9]. However, this can be challenging due to information silos and inconsistent pro-

[1]`https://platformengineering.org/blog/what-is-platform-engineering`
[2]`https://www.gartner.com/en/articles/what-is-platform-engineering`
[3]`https://www.liatrio.com/blog/what-is-platform-engineering-the-concept-behind-the-term`

cesses, like deployment and development, which result in miscommunication, duplicated efforts, and errors [12, 13]. This can stall an organization's ability to deliver quality products and services on time, leading to inefficiencies that heighten project costs and decrease productivity. Ineffective collaboration may also lead to increased system complexity, longer development cycles, and inflated operational costs, particularly visible within microservice architectures. Lastly, governance and resource management pose significant challenges, particularly for the focus on cost management. The ease of resource provisioning with cloud providers can result in over-allocation or under-utilization of resources, leading to unnecessary spending [14]. The blurred boundaries between traditional roles introduced by DevOps practices complicate getting an overview and accountability. Organizations must carefully monitor their cloud resources and associated costs to prevent overspending and unnecessary costs, a particularly pressing concern in the era of Cloud computing and budget cuts [15].

DevOps enables companies to shift their software development to an agile and lean approach, wherein software can be released early and with higher frequency[16–18]. This gives great advantages to organizations, but with these advantages, the described challenges are still present or even aggravated. The proliferation of tools could be the result of the adoption of DevOps and Cloud computing since each team is responsible for its entire lifecycle, including infrastructure, tools, and resources, resulting in an increase in complexity and overlap across different teams. Moreover, without an operations team maintaining the environments, the risk of inconsistency between environments can increase, resulting in more deployment failures and maintainability issues. Since DevOps and Agile focus on the project lifecycle and not on the organizational level, the communication challenges across the whole organization are left behind, resulting in disparate processes and workflows. Different methodologies have been introduced to reduce these challenges, like Site Reliability Engineering (SRE) [19], to reduce the time to resolve production issues. However, it does not solve these challenges other than reducing the effects.

Based on the latest development in the software engineering field, we can conclude that there is a need for a solution to these challenges stated above, and platform engineering is a candidate. However, platform engineering adoption faces challenges such as inconsistencies in understanding across different stakeholders and the interchangeable use of terminology like "Internal Developer Platform" and "DevOps", which create communication hurdles and compromise the efficiency of technological strategies [4,5]. Moreover, a clear understanding of the separation of accountability between different teams is essential. Further addressing the team types differentiated by team topologies [20]. Hence, standardizing the platform engineering terminology is critical to establishing a shared understanding and fostering efficient industry practices, including the added value of platform engineering. Another significant challenge involves crafting a tailored platform engineering strategy, which requires a deep understanding of the organization's processes and culture. It also necessitates comprehending how implementing platform engineering could alter these facets. The complexity in this paradigm shift to platform engineering extends to organizational changes, architectural design, technology selection, and operational management [6,7].

To address these challenges, this research proposes a reference model for platform engineering called the Platform Engineering Reference Model (PE-RM). The framework seeks to clarify the nature and scope of platform engineering, bridging comprehension gaps among various stakeholders. It dissects platform engineering elements and provides a precise, standardized model, mapping the interconnected components, processes, and roles. In addition, it includes a case study to validate the reference model. It offers a guideline and small proof of concept on effectively designing and implementing platform engineering guided by the PE-RM in a software organization.

1.1.1 Research questions

To tackle these issues, we investigate the following research question;
Main Research Question: How can a software organization effectively integrate platform engineering using a comprehensive reference model?
In order to give a detailed answer to this question, we have decomposed it into several other research questions:
RQ1: How to model platform engineering in the context of a software company?
By creating a reference model, we will provide a common understanding of platform engineering from

[4]https://www.gartner.com/en/articles/what-is-platform-engineering
[5]https://www.liatrio.com/blog/what-is-platform-engineering-the-concept-behind-the-term
[6]https://www.infoq.com/news/2023/02/platform-engineeringchallenges
[7]https://thenewstack.io/platform-engineering/platform-engineering-challenges-and-solutions

different viewpoints. Expert interviews and observations, in combination with the ODP (open distributed processing) reference model, will be used to create this reference model.

RQ2: How to define a customized platform engineering design tailored to a specific organization?
A case study will be conducted to create a conceptual design guided by the reference model. Validating the reference model and offering a guideline for designing a tailored platform engineering approach in the context of a software organization.

RQ3: How to effectively construct a technical platform engineering implementation?
With the reference model and conceptual design, we will create a small prototype that contains the basic elements of platform engineering. The engineering platform implementation will be used as an example of a technical implementation and validation of the reference model.

1.1.2 Research method

In this project, we will first try to extract most of the information related to platform engineering from scientific sources, such as papers (journal papers and conference papers) and theses. However, to get a clear picture of the state-of-the-art of this new discipline, we will also be looking into industry-related sources such as interviews, encyclopedias, whitepapers, and forums. All the scientific sources used will be documented in the bibliography section of this thesis, using the plain Bibtex style. Some other sources, such as references to forums and articles, will be documented using footnotes. To build the Platform Engineering Reference Model, we want to conduct a conceptual analysis during this research, including many expert interviews. The reference model will be created iteratively to validate and improve the model based on new observations and information. Moreover, we will conduct a case study focusing on our host organization to provide a conceptual design and technical implementation of platform engineering. The case study will also be combined with controlled experiments based on the reference model proposed in this research. By designing a conceptual design and implementing a basic technical engineering platform, we are able to validate the Reference Model. The validation and evaluation of the proposed technical implementation will be done by conducting various experiments with different stakeholders.

1.2 Contributions

Our research makes the following contributions:

1. **Platform Engineering Reference Model (PE-RM)**
   The first result is the PE-RM modeled with the use of literature and conceptual analysis. This reference model is based on the ODP-RM, and an iterative approach is used to improve the applicability and completeness of the model and is validated by a case study.
2. **Conceptual design of platform engineering within an organization**
   Based on the reference model, a case study on migrating to platform engineering is conducted. A thorough analysis of the case company is performed, and from these observations, a conceptual design guided by the PE-RM is proposed.
3. **A technical platform engineering implementation**
   To validate the reference model and outline the added value of platform engineering, a proof of concept (PoC) is created guided by the technical viewpoints of the reference model. This PoC serves as validation and technical contribution to platform engineering using open-source tooling. It provides proof and experiments demonstrating its value to a software organization.

1.3 Outline

In Chapter 2, we describe the related work of this thesis. In Chapter 3, we present the Platform Engineering Reference Model (PE-RM) and its methodology. In Chapter 4, a case study will be presented, which includes a conceptual design, technical implementation, and experiments of the implementation. Chapter 5 will contain the discussion and threats to validity. Finally, we present our concluding remarks in Chapter 6 together with future work.

Chapter 2

Related work

In this chapter, we will discuss work that is related to our research. Given that platform engineering is an emergent field characterized by a relatively light academic literature base, there is also an emphasis on industry-sourced literature and state-of-the-art tooling. Additionally, various studies that touch upon platform engineering conceptually or through proposed solutions have been investigated, which proved instrumental in enhancing the understanding of scientific contributions. Moreover, this section also looks into the Open Distribution Processing Reference Model (RM-ODP) and other alternative reference models to model platform engineering. Finally, different reference models and architecture of related methodologies have been investigated.

2.1 Platform engineering

Platform engineering is getting more popular in the industry, and different types of resources on platform engineering have been published, including reference architectures and open-source tooling that can help to implement an engineering platform to support platform engineering adoption. In this section, we will discover platform engineering-related initiatives relevant to this research.

2.1.1 IDP Reference Architecture

The essential component of platform engineering is an engineering platform e.g., Internal Developer Platform (IDP), which will allow teams to be more productive and have a better developer experience by offering self-service capabilities. It will take care of the tools needed to build an application and the infrastructure to run applications. An IDP will be different for each organization; therefore, creating a general reference architecture for such IDP can be difficult. There are many ways to create this engineering platform. Nevertheless, Humanitec proposed a reference architecture that can be used as a foundation to implement an IDP within an organization. Multiple variations of this reference architecture are created with different cloud providers (AWS, GCP, Azure)[8]. In figure 2.1, the reference architecture with AWS as a cloud provider is shown.

Although this is a technical implementation of the IDP, it can help understand how a potential platform could be implemented with some specific tools. It helps us separate the components of an engineering platform i.e. Internal Developer Platform (IDP), and what is essential. However, this reference architecture delves into specific tool choices, which reduces the applicability of this architecture since it can depend on individual preferences and pre-existing toolsets. Moreover, it ignores the organizational and cultural changes necessary to adopt platform engineering and the IDP to be effective. A technical understanding is not enough to implement platform engineering within an organization. Therefore, we believe this reference architecture is not complete but valuable as input for the technical viewpoints of our reference model.

[8]`https://humanitec.com/reference-architectures`

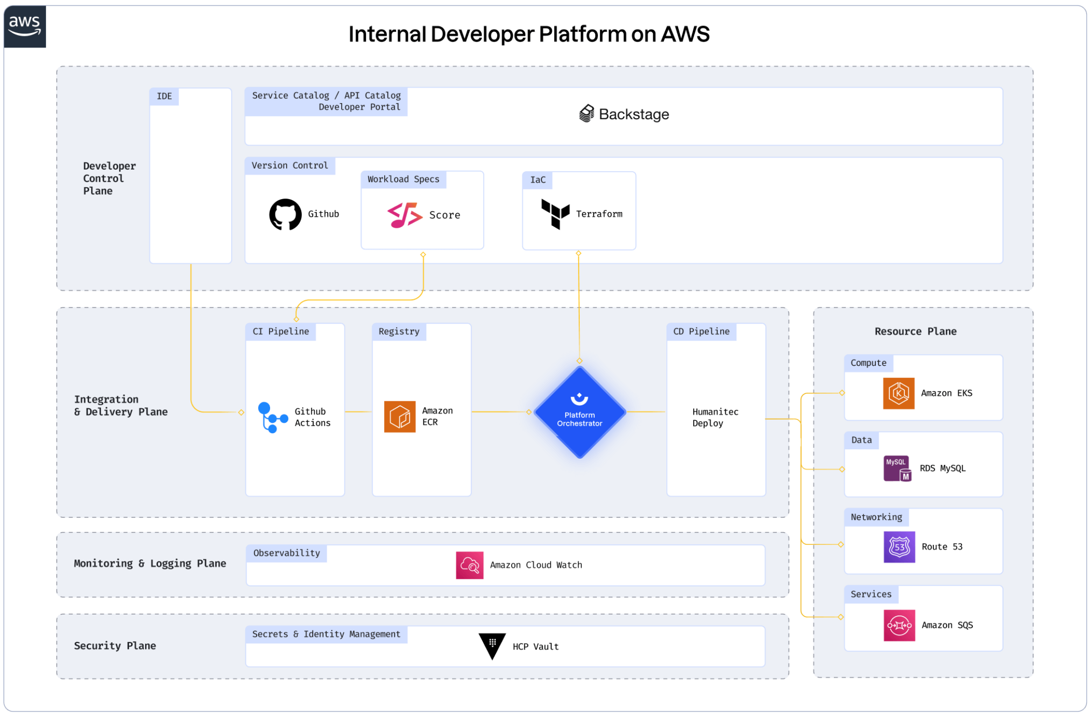


Figure 2.1: Reference architecture Internal Developer Platform (IDP) [8]

2.1.2 Platform tooling

In this subsection, we want to discuss the existing platform tooling related to platform engineering.

Otomi

Otomi[9] is a self-hosted PaaS (Platform as a Service) and adds developer- and operations-centric tools, automation, and self-service on top of Kubernetes, offering a multi and hybrid platform experience out-of-the-box. Otomi is placed in the Cloud Native Computing Foundation (CNCF) landscape [10] under the PaaS/Container Service section and attempts to connect many of the technologies found in the CNCF landscape to provide direct value. Otomi is built on top of Kubernetes. It can be helpful as it eliminates the need to re-inventing the wheel when building and maintaining your own Kubernetes-based (developer) platform or bespoke stack. Otomi is open source and can be deployed and used in your infrastructure, which enables organizations to take control of the tool and stack it supports.

For building a small proof of concept, this can be valuable as it offers out-of-the-box tooling. Still, as a state-of-the-art, it can be a valid option when building an engineering platform implementation with out-of-the-box functionality and configurations. It gives us more customization since we are not forced only to use the tools shipped with this PaaS solution; therefore valid to quickstart our PoC. However, for large organizations, it could be more beneficial to build your own platform based on already used tools.

Backstage

Backstage[11] is an open-source developer portal platform built by Spotify that enables developers to manage the software development lifecycle (SDLC). It's a centralized developer portal where developers can discover services, provision applications, manage versions, documention, and much more. It offers tools for technical and non-technical team members to perform software management tasks efficiently. The main purpose of Backstage is to streamline the software development process by integrating functionality into a single user interface, thereby simplifying the work of developers and increasing productivity.

Backstage is relevant to platform engineering because it essentially serves as a self-service developer portal for software engineers, automating many aspects of the software development process. In a company with a platform engineering team, these engineers would typically be responsible for building

[9] https://otomi.io/
[10] https://landscape.cncf.io/
[11] https://backstage.io/

and maintaining the underlying technology platforms that the developers use to create and operate their applications. Backstage simplifies and accelerates this work, as it provides a unified interface for developers to interact with the different platforms. This enables platform engineering teams to focus on building robust, scalable technology platforms. At the same time, developers can more easily leverage these platforms for their work, leading to a more efficient and productive engineering organization.

Azure DevOps

Azure DevOps[12] is a suite of development tools, services, and features provided by Microsoft that helps teams plan smarter, collaborate better, and ship faster with a set of modern dev services. It offers an end-to-end DevOps toolchain for developing and deploying software. Azure DevOps includes Azure Boards for work tracking, Azure Repos for source control, Azure Pipelines for CI/CD, Azure Test Plans for testing, and Azure Artifacts for managing packages. It plays an integral part in the platform engineering realm as it helps automate the software delivery process and reduces the complexity of managing the lifecycle of applications, from planning, development, and testing, to deployment. By integrating all aspects of the software development lifecycle into a single toolset, Azure DevOps allows platform engineers to establish a unified and automated environment. However, since this tool is universal, not much customization can be done, which could be essential if your organization requires specific functionality in order to adopt platform engineering. depending on the organization. Therefore less suitable to fully adopt platform engineering with this platform.

Heroku

Heroku[13] is a cloud-based platform as a service (PaaS) that simplifies the deployment, scaling, and management of applications. Developed by Salesforce, Heroku supports several programming languages, including Java, Python, Ruby, Node.js, and more. With Heroku, developers push their application code, and the platform takes care of the deployment and production environment – enabling developers to focus on writing code without worrying about the hardware or IT infrastructure. In the context of platform engineering, Heroku abstracts away many of the lower-level infrastructure management tasks. This allows platform engineers to focus on more strategic, high-value areas such as optimizing application performance, managing data services, or improving development processes. However, with the use of Heroku, the organization is limited with the tools and processes they want to integrate.

SAP business technology platform

SAP Business Technology Platform (BTP) [14], formerly known as SAP Cloud Platform, is an integrated offering from SAP that provides a suite of services, tools, and technologies to enable businesses to build, extend, and run their applications in the cloud. It is a PaaS that provides capabilities for integrating SAP and non-SAP applications, building custom applications, extending SAP cloud and on-premise applications, and leveraging advanced technologies like machine learning and artificial intelligence. The platform supports a range of programming languages and frameworks, including Java, Node.js, Python, and more. For platform engineers, SAP BTP is significant as it simplifies the process of developing and managing business applications. It provides a unified environment for deploying, managing, and scaling applications, helping platform engineers build robust, enterprise-grade applications efficiently. However, with the adoption of SAP, you as an organization are limited by the capabilities of SAP and required to work within their boundaries, limiting specific functionalities.

2.2 Related research

Since platform engineering is new and mainly pushed by the industry, there is not a lot of scientific research focused on platform engineering. Therefore we focussed on research related to platform engineering by the proposed tools or platform engineering characteristics. Moreover, these observations can conclude that the need for scientific evidence for platform engineering could be valuable.

[12]`https://azure.microsoft.com/en-us/products/devops`
[13]`https://www.heroku.com/platform`
[14]`https://www.sap.com/products/technology-platform.html`

2.2.1 Platform engineering in enterprise application development

Prior to the rise of platform engineering in the tech industry, the research by Zhou *et al.* delved into this field, particularly in the context of enterprise application development [21]. The study focuses on the need for systematic and reusable software development methodologies to counter IT projects' historically high failure rate. It introduces critical practices like software reuse, component-based software engineering, model-driven development, and software product line engineering. These methodologies help enhance software development's efficiency, reusability, and flexibility. Furthermore, the paper discusses the power of software platforms in product development, underlining how they streamline the development process and enhance product diversity. It explores the role of components in web applications and the necessity of a generalized component model. The author identifies a gap in academic research on platform-based enterprise application development. They encourage more academic work in this area and propose closer collaboration between academia and the industry. We believe this can be achieved since there is more adoption in the industry, and the need for a general understanding of platform engineering is increasing.

However, this paper, written in the early 2000s, fails to account for more contemporary developments in platform engineering. Given its timeline, certain aspects of the paper are now outdated. Although it offers an in-depth theoretical framework, it lacks empirical substantiation of the proposed ideas and methodologies. While discussing elements that influence success and failure in software reuse, the paper does not thoroughly analyze the obstacles impeding the adoption of platform engineering. The adoption of platform engineering can vary based on factors such as industry type and organization size. An in-depth comparative study or exploration into industry-specific challenges and solutions could yield valuable insights.

2.2.2 Composable DevOps

Given its overlap with other methodologies, platform engineering has been the subject of related research, such as the concept of composable DevOps, as discussed by McCarthy *et al.* [22]. The paper focuses on the development of a composable DevOps solution. It uses an ontology-based approach, with tools such as the Web Ontology Language (OWL) and Semantic Web Rule Language (SWRL), to model, understand, and automate the complex processes in the DevOps pipeline. The authors stress that successfully adopting DevOps practices requires a cross-functional team. Ideally, This team would understand and speak the same language, promoting more effective collaboration and problem-solving. The paper introduces the Metadata Provenance Framework for automating DevOps maturity path decisions. The framework engages in complex interactions between actionable policies and processes, generating metadata that can be analyzed to comprehend process integrity, behavior, and the precision of decision-making.

The approach to composable DevOps elaborated in this paper has significant implications for platform engineering. It presents a method to evaluate and enhance DevOps adoption, a critical facet of platform engineering. By highlighting the role of shared understanding and team dynamics, the paper lends insights applicable to our research context. The automated decision-making framework could also provide valuable insights for comprehending and improving decision-making in platform engineering adoption.

Although the paper primarily concentrates on DevOps, platform engineering is a more expansive concept that includes other elements like API management, Microservices, and cloud platforms. The paper lightly touches on cultural aspects concerning team understanding and collaboration. Still, a deeper investigation into the influence of organizational culture and change management is crucial for successful platform engineering adoption.

McCarthy *et al.* [22] acknowledged that there are legitimate barriers and concerns from IT and "governance" teams when they bring ideas to the table. The groups responsible for creating and supporting applications and solutions are chartered. Therefore, they propose a Composable DevOps solution framework that enables iterative collaborative upskilling and collaboration value measurement. This solution architecture consists of components including a composable DevOps runtime, composable DevOps controller, monitoring framework, and DevOps portal. Many of these components have characteristics that can be related to platform engineering.

2.3 Reference model frameworks

In this section, we will discuss the different reference model frameworks that can be used to model various project lifecycles, focusing on the Open Distributed Processing (ODP) reference model. We will explore existing research, examine examples of its applications, and compare it with other reference models.

2.3.1 Zachman framework

The Zachman Framework for Enterprise Architecture is a conceptual structure that provides a systematic approach to defining, organizing, and managing enterprise architectures. Proposed by John A. Zachman in 1987, the framework has since become a widely recognized tool for organizing the multiple dimensions of an enterprise's information systems [23, 24]. The framework is based on a matrix with six rows representing different perspectives (e.g., planner, owner, designer, builder, implementer, and worker) and six columns representing various aspects of the architecture (e.g., data, function, network, people, and motivation).

The Zachman Framework helps organizations ensure that all relevant aspects of their enterprise architecture are considered, leading to more coherent, integrated, and maintainable systems. By systematically addressing each cell in the matrix, stakeholders can communicate and collaborate on the design, implementation, and management of the enterprise architecture. The framework has been influential in developing other reference models and frameworks. It is widely used for guiding organizations in their efforts to align their information technology (IT) systems with their business objectives and strategies.

2.3.2 Reference model for Service-Oriented Architecture (SOA-RM)

The Reference Model for Service-Oriented Architecture (SOA-RM) is a high-level framework for designing and implementing service-oriented architectures. Developed by the Organization for the Advancement of Structured Information Standards (OASIS), the SOA-RM provides an abstract and technology-agnostic foundation for understanding and creating service-oriented solutions [25]. The model consists of three main components: the service ecosystem, which includes the providers, consumers, and brokers of services; the service-orientation design paradigm, which promotes modularity, reusability, and loose coupling of services; and the service-oriented enterprise, which represents organizations adopting this.

The SOA-RM provides a set of core concepts, relationships, and principles that guide the design and implementation of service-oriented systems. By following these guidelines, organizations can create flexible, scalable, and maintainable systems that adapt to changing requirements and business needs. The SOA-RM has been applied in various domains, such as e-government and telecommunications, to facilitate interoperability, reduce complexity, and improve IT systems' overall agility and responsiveness. Its use promotes effective communication and collaboration among stakeholders in designing, implementing, and managing service-oriented architectures.

2.3.3 Open Distributed Processing reference model

The ODP reference model [26], as defined in the joint standard ISO/IEC 10746 and ITU-T Recommendation X.901-X.904, has been widely studied and applied in the design and implementation of distributed systems [27] origin book. Open distributed processing (ODP) describes systems that support heterogeneous distributed processing both within and between organizations through the use of a typical interaction model. The reference model ODP (RM-ODP) carefully describes its components without prescribing an implementation. In this case, we are interested in part 3 of RM-ODP which prescribes a framework using viewpoints from which to abstract or view ODP systems.

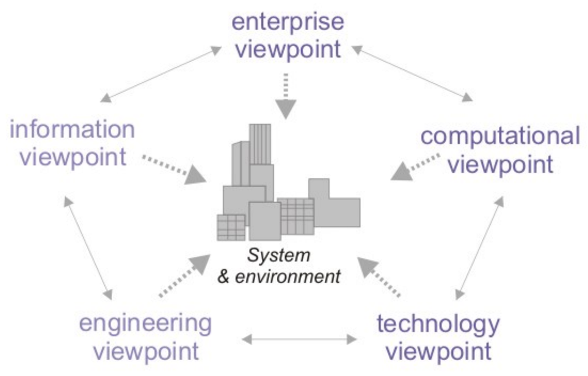


Figure 2.2: RM-ODP view points

RM-ODP defines the following five viewpoints:

- **Enterprise viewpoint:** is used to organizational requirements and structure. From the enterprise viewpoint, social and organizational policies can be defined as objects, communities, and roles. The enterprise language is concerned explicitly with performative actions that change policy, such as creating an obligation or revoking permission. Especially roles and lifecycles will be of value to model platform engineering from the enterprise viewpoint.
- **Information viewpoint:** which defines the types of information to be exchanged between systems. If all the interacting subsystems involved use the same set of information types, we can achieve a basic level of consistency in the way information is interpreted and used and avoid problems that would arise from there being divergent interpretations of it. The objects are divided into information and action objects.
- **Computational viewpoint:** which expresses the operations of the system, drawing on the information types where appropriate to ensure consistency. This design describes the system's software operations and interfaces between components, expressed in a platform-independent manner.
- **Engineering viewpoint:** which provides a set of generic middleware concepts and solutions, providing templates for transforming the computational specification into a concrete implementation. It is responsible for the management of distribution and concrete representations. The use of tools to select appropriate templates for each environment in which the system is to be deployed is one of the keys to the effective reuse of engineering solutions.
- **Technology viewpoint:** which expresses the resources available to support the resulting system and policies for selecting suitable resource components and technologies, including the declaration of standards to be used throughout the system implementation. Bringing these aspects together is essential for the management of conformance.

Kilov *et al.* [28] provided an in-depth analysis of the ODP reference model, its foundations, and its practical applications in various domains. The authors discuss the experiences of researchers and practitioners who have used the ODP reference model in real-world scenarios, as well as the challenges and lessons learned from these experiences. It can be concluded that since the publication of the RM-ODP, it has become a mature and widely appreciated framework for distributed system design.

2.3.4 Comparison

The ODP reference model, the Zachman Framework for Enterprise Architecture, and the Reference Model for Service-Oriented Architecture (SOA-RM) address different aspects of system organization, management, and technical implementation. The ODP reference model primarily focuses on distributed systems, emphasizing the separation of concerns, reusability, and interoperability. It provides a systematic approach that enables stakeholders to understand and address various aspects of a distributed system through five distinct viewpoints. In contrast, the Zachman Framework offers a comprehensive, matrix-based structure that organizes an organization's information systems architecture based on six perspectives and six aspects. This framework ensures that all relevant aspects of an enterprise's architecture are considered, resulting in more coherent, integrated, and maintainable systems. The SOA-RM, on the other hand, is a high-level framework for designing and implementing service-oriented architectures. It provides an abstract, technology-agnostic foundation for understanding and creating service-oriented solutions, promoting modularity, reusability, and loose coupling of services.

In the context of platform engineering, the ODP reference model would be a better choice for creating a reference model because of its focus on distributed systems, separation of concerns, and reusability. These characteristics align well with the challenges typically encountered in platform engineering, such as the need to support scalable, modular, and interoperable solutions that can evolve. Additionally, the ODP reference model's five viewpoints offer a structured approach that facilitates communication and collaboration among stakeholders, which is crucial for the successful design, implementation, and management of platform engineering projects.

2.4 Project lifecycle reference models

2.4.1 Adaptive enterprise project management (AEPM)

The adaptive enterprise project management (AEPM) capability reference model, created by Gill [29] is an interesting reference model for understanding how Agile can be adopted within an enterprise organization. AEPM uses a comprehensive approach that focuses on the continuous adjustment and improvement of project management processes in response to changes in the business environment. This approach emphasizes flexibility, agility, and adaptability in managing projects. AEPM facilitates the rapid identification and mitigation of risks, promotes efficient allocation of resources, and enables organizations to respond effectively to new opportunities and challenges. By incorporating feedback loops and iterative planning, AEPM ensures that project goals and objectives remain aligned with the evolving needs and priorities of the organization.

Integrating AEPM into the platform engineering reference model creation helps to ensure that project management processes are agile and responsive to changing business needs. This adaptive approach allows organizations to prioritize and allocate resources more efficiently, leading to more effective decision-making and better alignment between technology platforms and business objectives.

2.4.2 DevOps reference architecture (DRA)

Ghantous and Gill [30] has done research into creating a DevOps reference architecture (DRA) to deploy IoT into the cloud. Although this paper is focused on IoT, it can be related to platform engineering based on points made in this reference architecture. This DRA consists of four different models: DRA contextual model, conceptual model, logical model, and physical model. The contextual and conceptual models are abstract, and to achieve an optimal system, DRA relies on the Cloud to provide an abstract infrastructure layer. The next layer is the Logical model, which states the logical architecture details on five models: repository, CI-broker, CD platforms, Monitoring and communication, and database. The final model is the DRA physical model and is the integration of DevOps tools to fulfill the logical model. These tools can differ for each organization, but generally, they consist of a list of tools.

The integration of these tools is the key factor of the DBA; therefore, platform engineering can help achieve this integration. Based on this reference model, we can conclude that some interesting points were made about the logical and physical model of this DBA, which can be used during the creation of the platform engineering reference model.

2.4.3 Software architecture framework for quality-aware DevOps

Di Nitto *et al.* did research into a software architecture framework for quality-aware DevOps [31]. In this research, they stated that many stakeholders are involved in DevOps but do not have a direct relation to the product but account for the product's organizational stability. This is also the case for platform engineering and, to some extent, explained by team topologies [20]. Di Nitto *et al.* found that the 4+1 software architecture views framework required specialization and additional re-elaboration and therefore created a refinement of that framework. Based on this research, it can be said that their proposed SQUID framework offers a valuable basis for describing and achieving quality in DevOps but has limitations in achieving quality-aware DevOps, especially in terms of tooling, applicability, and user-friendliness.

2.5 Summary

In summary, there is a marked scarcity of research in the scientific and industrial domains on platform engineering, and the sparse literature available presents varied perspectives on the topic. This underlines the limited support in identifying and elucidating what platform engineering entails and its potential applications. The current reference architecture for platform engineering appears to inadequately incorporate cultural understanding, offering a limited explanation of the adoption process of platform engineering within organizations. Since the methodology is different from existing ones the reference models for other methodologies cannot be applied to platform engineering. Therefore, this research offers significant value by providing a comprehensive reference model of platform engineering, integrating both technical aspects and cultural dimensions.

Moreover, there are no guidelines or evidence of platform engineering applicability and effectiveness in a software organization that could further emphasize the importance of platform engineering. There are no known articles or scientific publications that showcase how to design and implement platform engineering within a software organization. In addition, there is no evidence of the effectiveness of integrating a platform engineering approach. Recognizing these gaps, this research aims to contribute by offering a case study that will propose a conceptual design and technical implementation of platform engineering guided by the PE-RM.

Chapter 3

Platform Engineering Reference Model

In this chapter, we will introduce the Platform Engineering Reference Model (PE-RM). Based on what we observed in the existing related work, a general understanding of platform engineering is needed and will be done by creating a multi-viewpoint reference model. The reference model is based on the Open Distributed Processing Reference Model (RM-ODP) [26, 27] and is an ensemble of viewpoints that illustrate the principal objects within each viewpoint, their organization by the proposed platform engineering lifecycles, and their correlations with objects defined in other viewpoints. The viewpoints suggested by the PE-RM aim to be as loosely coupled as possible to enable parallel design and development efforts across various sectors of an organization. The PE-RM outlines five viewpoints (figure 2), and the following subsections explain the methodology and elaborate on these five viewpoints.

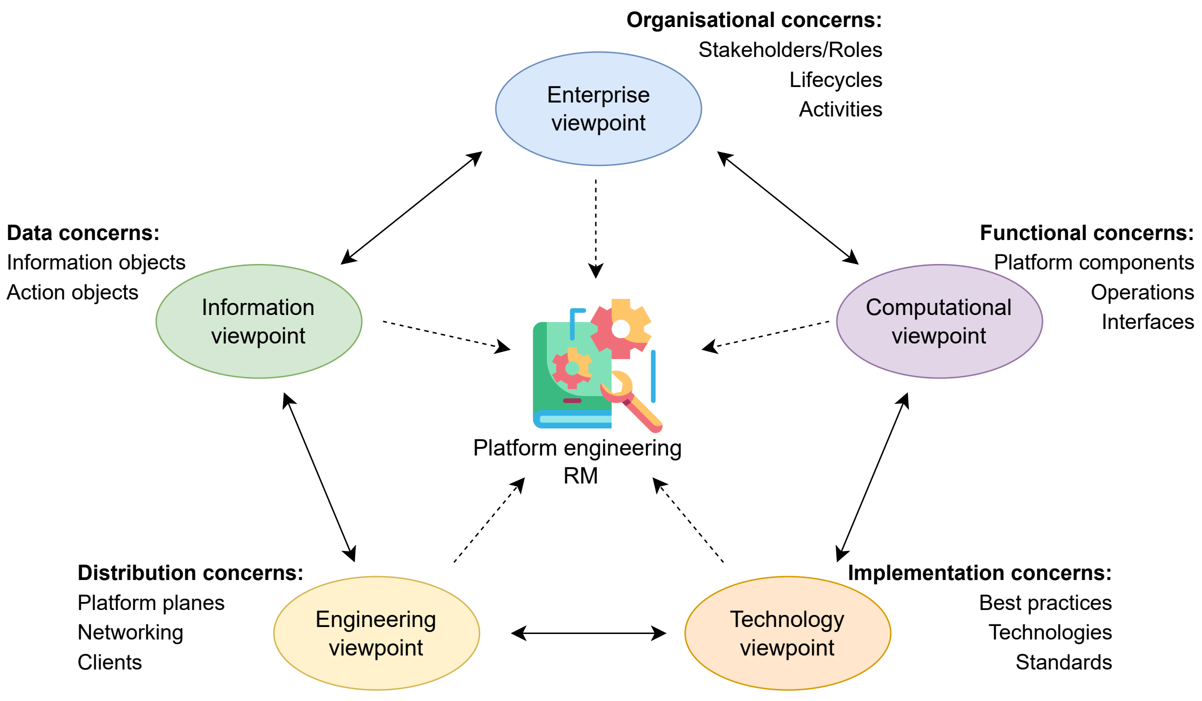


Figure 3.1: The five viewpoints of the Platform Engineering Reference Model.

3.1 Methodology

An iterative approach was used to acquire a comprehensive understanding of platform engineering by conducting a series of interviews with numerous experts, each from diverse professional backgrounds and roles. The organization investigated already manifested some aspects of platform engineering, and a need for greater standardization of terminology was apparent to enhance its applicability within this and other organizations, supporting a general vocabulary. Therefore, we interviewed industry experts outside the organization for a more holistic understanding. Moreover, external resources were conducted and integrated into the reference model. In section 2.1.1, we outlined the IDP reference architecture of Humanitec. However, since this only covers the technical aspect of platform engineering, we used it to model the engineering and technology viewpoints. Consequently, the RM-ODP framework was leveraged to express the platform engineering terminology, refining its interpretation and application. With the formulation of the reference model, we conducted a case study that involved constructing a conceptual design and technical implementation, all of which were guided by the Platform Engineering Reference Model. This aimed to validate its applicability and provide a concrete blueprint for future utilization.

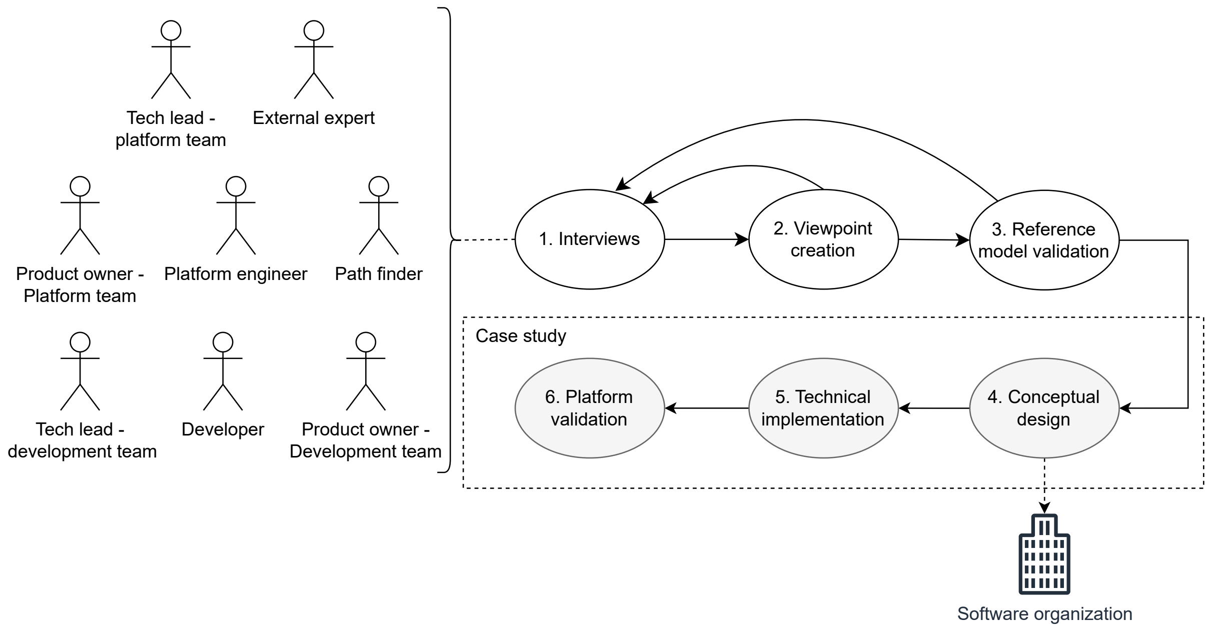

Tech lead - platform team
External expert
Product owner - Platform team
Platform engineer
Path finder
Tech lead - development team
Developer
Product owner - Development team
1. Interviews
2. Viewpoint creation
3. Reference model validation
Case study
6. Platform validation
5. Technical implementation
4. Conceptual design
Software organization

Figure 3.2: Methodology Platform Engineering Reference Model.

The first step in developing this reference model involved interviewing eight software engineering experts of varied roles and backgrounds within and outside the organization (see figure 3.2). Their perceptions guided the formation of viewpoints in the second step. In step three, its validity was assessed via additional interviews and comparison with related work. After the validation by interviews and comparison was done, we continued with a case study. The primary validation and applicability of the reference model were done in the case study consisting of a conceptual design, technical implementation, and experiments to help showcase the added value and applicability of platform engineering. The conceptual design and technical implementation are guided by the PE-RM and showcase the applicability and validity of the reference model, together with the experiments. Notably, the methodology adopted an iterative approach, necessitating continuous refinement of the reference model in light of fresh insights and varied perspectives. This recursive cycle of refining greatly enhanced the model's maturity and reliability.

3.2 Viewpoints

3.2.1 Enterprise viewpoint

The enterprise viewpoint focuses on the organizational context of the domain in which the designed systems are intended to operate. This viewpoint concentrates on the lifecycle, stakeholders (roles), and activities that platform engineering will introduce. The enterprise viewpoint is intended to cover the process changes and how this impact different lifecycles, teams, and their responsibilities. The main modeling concept of the enterprise viewpoint is lifecycles and based on these lifecycles, roles and activities are introduced.

Lifecycle

Platform engineering can be separated into two lifecycles: the platform engineering lifecycle and the application lifecycle. With the adoption of platform engineering, the platform engineering lifecycle will be introduced, and the application lifecycle will be impacted by introducing an engineering platform.

The platform engineering lifecycle

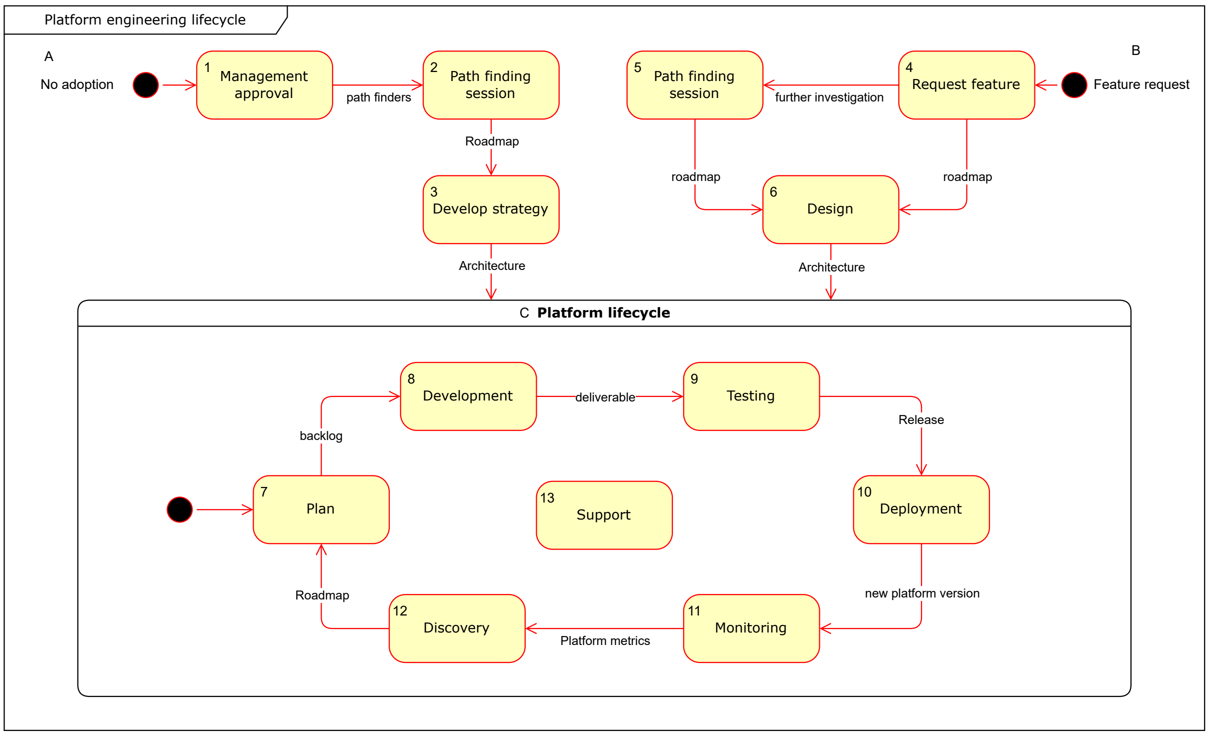


Figure 3.3: Enterprise viewpoint: platform engineering lifecycle

The platform engineering lifecycle outlines the cycle through which an engineering platform will be introduced and subsequently enhanced by a dedicated platform team. This lifecycle comprises a variety of pathways, all converging towards implementing a platform that addresses the specific needs of the development teams. Within this lifecycle, two points of inception are identified: no platform engineering adoption and platform engineering integration in which development teams can make feature requests. Stakeholders crucial to this lifecycle encompass the platform team, pathfinders, the management team, development guilds, and development teams.

A No adoption
For organizations yet to adopt platform engineering, their path aligns with the ”no-adoption” course. This path includes various phases aimed at designing a platform and roles. It is crucial to recognize that in most instances, an organization does not commence operations from a clean slate or ”Greenfield” but rather from an already developed or ”brownfield” state [32], where it is likely that some characteristics of platform engineering are already present. The ”no-adoption” path includes the following phases:

1. **Management approval:** Based on a trigger to improve the software processes within an organization, the management team has to approve this transition and assign pathfinders. Since this is an investment without a direct revenue stream, it is important to have technical expertise within this team to identify the benefits and ROI [33]. The request for the adoption of platform engineering will most likely come from senior developers or other employees with a technical background.
2. **Path finding session:** The pathfinders are responsible for getting an overview of the recurring steps and overlapping tools between different development teams. They can plan path-finding sessions to get insight into the current way of working within the organization. Based on their observations, they create the initial roadmap.
3. **Develop strategy:** Based on the initial roadmap created by the pathfinders, the management team, including the pathfinders, can develop a plan for adopting platform engineering and assigning a platform team. The pathfinders and platform team are responsible for the platform architecture.

Platform engineering often starts with a few development teams crafting productivity tools that can grow into broader systems. Dedicated teams, managerial approval, and clear roles are essential for sustainable expansion and improvement. The extent of platform engineering adoption hinges on the organization's profile, technological state, and openness to this approach. Though seen as a cost center, platform engineering should yield long-term savings.

B Feature request
Once the platform is incorporated into the organization, various routes can be explored to facilitate the implementation of new features. This could be achieved through Pathfinders instigating an architectural change that benefits the entire organization or by development teams. As these development teams utilize the platform, their requests for new features can initiate this pathway, which includes:

4. **Request feature:** The development team will have a conversation with the platform team (product owner) to discuss the need for functionality. Depending on the impact, it can be taken into the roadmap, or it needs to be further investigated. The platform team (mainly the product owner) can decide the need for further investigation.
5. **Path finding session:** Based on the decision of the platform team, path-finding sessions are done to discuss the feature request and its impact. The impact can be addressed during these sessions, and alternatives can be researched. If the feature is necessary and beneficial for the organization, it can be taken into the roadmap. In the end, it is decided by the pathfinders if a feature with a significant impact will be integrated into the platform.
6. **Design:** With a roadmap and precise requirements for the feature, the platform team designs a solution. They can identify the best practice and how it can be implemented within their engineering platform. Depending on the impact of the change, pathfinders can support the creation of this architecture.

Given the platform's size and the number of development teams, this process could occur repeatedly. Consequently, it's crucial to ensure that it doesn't affect the platform's lifecycle. Additionally, it is vital to promote an environment where teams feel encouraged to request features that enhance the platform.

C Platform lifecycle
The core of the platform engineering lifecycle is the development of the engineering platform, which is a continuous lifecycle. Based on the roadmap and architecture, the platform team works on the platform iteratively, and it includes the following phases:

7. **Plan:** The platform team works in iterations to deliver value to the development teams faster and more often. Based on the roadmap, the platform team refines and plans sprint backlogs to continue delivering value to the development teams according to the roadmap. High-priority stories are taken into the upcoming sprints and are managed by the product owner.
8. **Development:** Based on the backlog, the platform team works on improving the platform by integrating new tools and techniques. Since the platform can consist of many different components, the platform team will have to deliver different types of outputs. The most common one will be infrastructure and application code or configuration changes in a cloud provider console.
9. **Testing:** Before a deliverable can be deployed, it has to be tested to check if no bugs exist that could break the platform. Depending on the type of deliverable, tests must be done by the platform team to validate the change, resulting in a releasable platform update. Depending on the change, how the platform or separate feature is tested can be decided.

10. **Deployment:** To ensure that significant changes do not happen at any given moment, the platform team can bundle releases into one deployment. After the deployment, a new version of the platform will be available to the development teams containing new features or other improvements. Notifying the update and updating the change log is essential.
11. **Monitoring:** A new version of the platform will be monitored by automated metrics and tools to ensure the platform behaves as it should. The platform team can use these metrics to create dashboards or alerts in case something breaks that could impact the development teams.
12. **Discovery:** Based on metrics, the platform team can discover improvements to the current platform version. These improvements can be discussed with the product owner (and possibly pathfinders) and validated if they can be taken into the roadmap.
13. **Support:** The support phase will be during the entire platform lifecycle and is used to support development teams in using the platform. This includes updating documentation, blueprints for the golden paths, and other templates like dashboarding. The development guilds maintain the golden paths at any given moment and are essential to support the adoption of best practices.

Post-deployment, the platform team oversees the platform's operation, potentially identifying enhancements. Whether the organization opts to construct this platform independently, or outsource it to an external software company, depends on several factors. These include the level of customized tooling required, the development teams' willingness to adopt a more generalized work approach, and the organization's in-house expertise. The benefit of an externally developed platform lies in its potential to decrease maintenance demands.

Applications lifecycle with the use of platform engineering

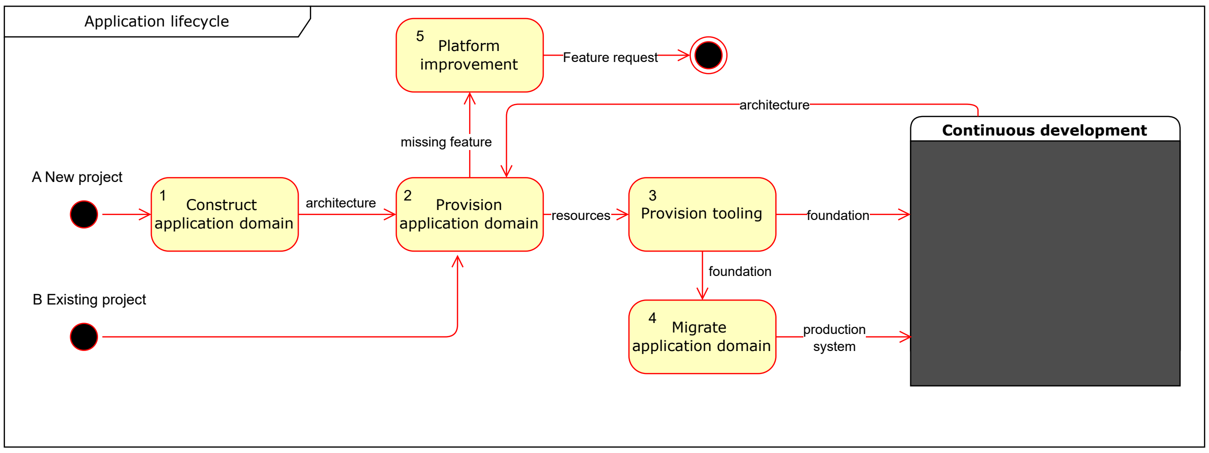


Figure 3.4: Enterprise viewpoint: Applications platform engineering lifecycle

Adopting platform engineering fundamentally alters the application lifecycle, dividing at two unique starting points based on a project's status. Platform engineering adoption will likely bolster continuous development through the platform's self-service functionalities. Despite this, it doesn't directly influence the continuous development lifecycle, which is why it is modeled as a black box. We posit that an application's design precedes continuous development, ensuring a well-defined architecture prior to development. This can also assist in identifying the platform's capabilities to support the development.

A New project
Incorporating platform engineering in a new project, the application lifecycle can leverage the platform's self-service features. The project's initial stage is key for capitalizing on these advantages. We've identified the following phases:

1. **Construct application domain:** From business problems and stakeholder requirements, the development team formulates an architecture, resulting in an application domain. This structure is fundamental and can be optimized during the continuous development phase.
2. **Provision application domain:** Utilizing an architecture that embodies various resources (the application domain), the development team can use golden paths for provisioning. Development

guilds maintain these paths, encapsulating best practices. Platform-enabled application provisioning promotes governance and standardization across development teams.

3. **Provision tooling:** After resource provisioning, resources are interconnected, and tools are configured to boost productivity and efficiency. Provision tooling ensures the creation and linking of CI/CD, services, documentation, etc., to the application domain.

Post-provisioning of a new project, the development team possesses the resources needed to create business value through continuous development. They can opt to reassess the architecture during the development phase, which enhances the application domain. This can also be influenced by platform improvements that benefit a project.

B Existing project
For an ongoing project, integrating platform engineering presents more challenges to utilizing the platform and its self-service capabilities. The existing application domain must be transitioned into the platform ecosystem, involving the following phases:

2. **Provision application domain:** Based on the existing application domain, the team can employ golden paths for application provisioning. Platform-enabled provisioning assures the use of standardized tools and proper governance, adhering to platform guidelines.
3. **Provision tooling:** After the resources are provisioned, the resources have to be connected, and tooling needs to be configured to improve productivity and efficiency. This could, for example, include CI/CD, documentation, networking, and more.
4. **Migrate application domain:** The existing application can be migrated to the newly provisioned application domain, done by the development team. It depends on the type of application and the current state of the project on how much time it takes to migrate the application. This results in the application running in production using the platform.

This path is essential to leverage platform engineering benefits for existing projects, controlling and minimizing technical debt.

Platform improvement
Throughout the continuous lifecycle, the development team may identify potential improvements for their application domain or boost productivity and developer experience. If such an enhancement is identified, they can consult the golden paths or request the improvement, leading to the following phase:

4. **Platform improvement:** The team may identify lacking techniques, configurations, or tools that could enhance their application or improve productivity. It's crucial that the team can request these features, ensuring explicit requests and detailed impact assessment.

This path will trigger the platform engineering lifecycle in figure 3.3.

Continuous development
We believe that with the adoption of platform engineering the way development teams create business value during the continuous development lifecycle will not change. It will impact the productivity and developer experience in this phase, but not the way it should be done. This way it is up to the organization and development teams how they want to use the continuous development lifecycle. However, it is important to understand that we have extracted the architecture design and provisioning of services outside the continuous development lifecycle in order to show the benefits and way of working for platform engineering within the application lifecycle.

Stakeholders

Based on the different lifecycles stated above, different stakeholders are involved with the platform engineering lifecycles. These stakeholders have different roles within this lifecycle, and we will explain why they are relevant to platform engineering.

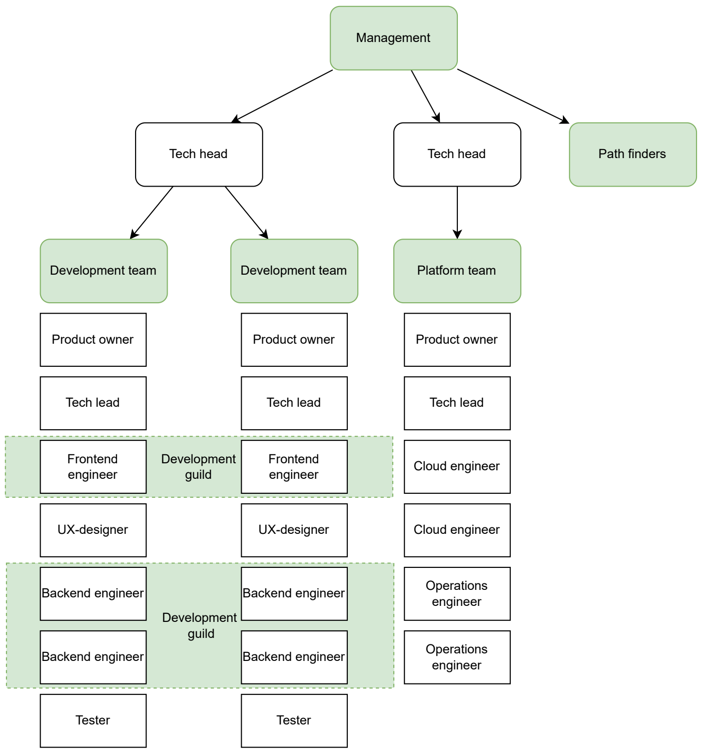


Figure 3.5: Enterprise viewpoint: stakeholders

- **Platform team:** The platform team is responsible for building, maintaining, and evolving the platform, including the core infrastructure and tools that support the development and deployment of applications. This team comprises the product owner, tech lead, and different types of developers with expertise. The platform team is also responsible for ensuring teams can adopt best practices, and they deliver blueprints for golden paths and how applications should be provisioned. Tools commonly used across development teams fall under the platform team's domain.
- **Development team:** The development teams are cross-functional teams that build business value for their customers. They consist of a product owner, tech lead, and different types of developers and testers. Utilizing the platform's self-service capabilities, they build, maintain, and deliver applications, and can request new platform features.
- **Path finders:** Pathfinders can be related to architects and are not part of a development team. They are individuals or a group that explores new technologies, tools, and practices to improve the overall software engineering processes within the organization. They evaluate the benefits and risks associated with new technology adoption, providing recommendations for integration into the platform. Pathfinder decisions significantly influence the platform's trajectory.
- **Development guilds:** Development guilds are groups of developers that share a common interest or expertise in a particular area, such as a programming language, framework, or technology. They update and maintain golden paths for their field to foster standardization and best practice adoption. Using blueprints from the platform team, they create detailed guides for implementing specific languages or frameworks.
- **Management team:** The management team consists of the CTO, the tech heads, and pathfinders and are responsible for the direction and future of the organization's technology department. Major decisions require their approval, with pathfinders providing technical input.

Interactions

Based on the two lifecycles and stakeholders identified above, we will further explain the interactions of the different phases, especially which actors are involved and what they do. To better understand the platform engineering practice, we identified key interactions based on the different paths explained in the lifecycles and proposed them as examples.

Adopt platform engineering

Prior to platform engineering adoption, a trigger often in the form of scalability issues, identified by tech leads, architects, or management, stimulates consideration. After they identified such issues, platform engineering can be considered a valid solution. Therefore a company needs to understand the added value of platform engineering. Existing tools can play an impact on the design decisions made on how to implement platform engineering.

To identify the added value of platform engineering and create estimation and design, the management team assigns pathfinders to look into the current way of working within the different development teams. They look into the productivity and efficiency issues within development teams and identify possible overlapping tools and techniques to increase standardization and centralization. Based on these observations, the pathfinders develop the initial roadmap of the platform. Based on this roadmap, a platform team can be assigned, and an architecture can be created.

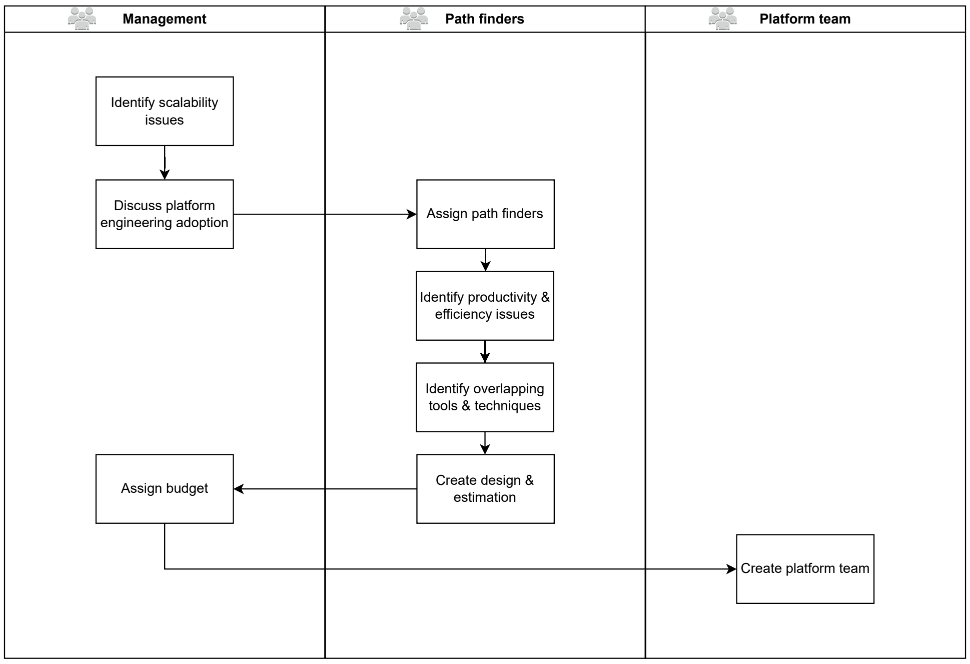

Management
Path finders
Platform team
Identify scalability issues
Discuss platform engineering adoption
Assign path finders
Identify productivity & efficiency issues
Identify overlapping tools & techniques
Create design & estimation
Assign budget
Create platform team

Figure 3.6: Enterprise viewpoint: platform engineering adoption interactions

New platform feature

New features can be requested in different ways and by different stakeholders. First, the platform team can identify new features during the continuous development of the platform and create stories to improve specific parts of the platform. Second, pathfinders can identify improvements for organizations that need a new platform feature. Finally, the development teams can request new features to be implemented into the platform to improve their productivity and efficiency, see figure 3.7.

To initiate the need for a new feature, the development team has to identify an essential improvement for their way of working. This can be as small as an extra configuration or a completely new tool that needs to be integrated. The impact of the proposed addition must be understood by the development team. Once a development team finds a feature that needs to be added to the platform, they can discuss this with the product owner of the platform team, he/she can identify the impact it has on the platform and the platform team to decide to take it into the planning, or that further investigation is needed. During path-finding sessions, stakeholders deliberate the need and impact of the feature. Once a consensus is reached, the request may be included in the roadmap or rejected. Upon approval, the platform team devises an architecture/design for the new feature.

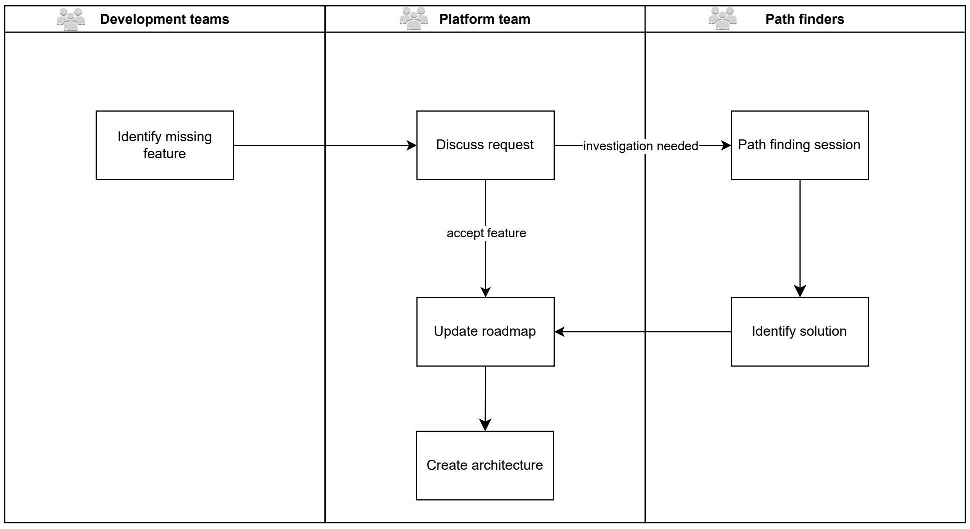


Figure 3.7: Enterprise viewpoint: new feature interactions

Platform lifecycle

The engineering platform lifecycle is a continuous cycle that begins anew each time it's completed. The initiation of this cycle necessitates an updated roadmap and a clear architectural design, equipping the platform team with direction. With these in place, the team plans and executes a sprint to deliver business value, by developing and testing new features or upgrades. However, because this continuous lifecycle can be different for each organization this will be seen as an example.

Once a feature passes testing and has a deployment date, it's released, and a new platform version becomes available. Post-deployment, the platform team provides support and potentially updates the "golden paths" based on the infrastructure setup. Deploying platform changes in a development environment first allows applications to adapt and avoid production issues. The impact of the new version can necessitate updating the golden paths, and significant changes may be announced to keep the development team informed. Following deployment, the platform team monitors the platform to ensure stability and identify potential improvements. These findings can prompt new feature requests, leading to updated roadmaps and architectures and thus restarting the cycle. The following diagram shows the interactions:

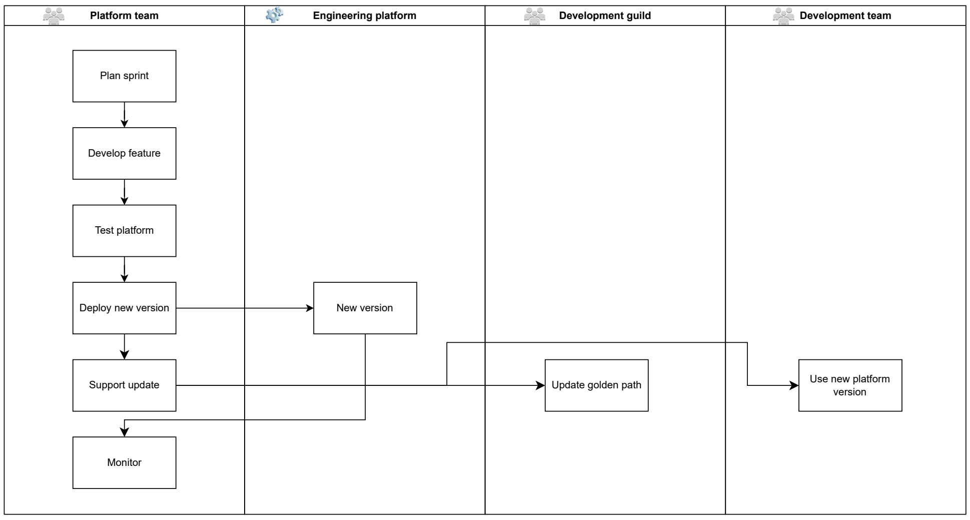


Figure 3.8: Enterprise viewpoint: platform lifecycle interactions

Provisioning project (application domain)

Prior to generating business value through continuous development, the application domain needs to be provisioned with all necessary platform integrations. This step fosters the adoption of best practices and promotes standardization within the organization. See figure 3.9 for the modeled interactions.

The first step is to look into the golden paths maintained by the development guilds to see if there is a step-by-step tutorial on how to use a particular language or framework within the platform. The platform also offers provisioning in which the development teams do not have to create or wait for the creation of specific resources or repositories. The development guilds and platform teams maintain these golden paths. As soon as the development team provisions an application, all the necessary tools and integrations are done by a provisioning tool, which makes it easier for the development team to start creating business value. For a more in-depth explanation of this process, see section 3.2.3 for an in-depth explanation of the provisioning process.

Within an application domain, the team can also create other resources, like databases, and the platform will also take care of this. This is essential to improve transparency and encourage governance. The platform team is responsible for ensuring these resources are created, and the tags are in place. As soon as everything is created, the team has the entire application domain in place and can start creating business value. During this path, the development team can identify missing features in the platform that are not supported but are needed for a specific application. In that case, they can request a feature from the platform team, and a different process will be started.

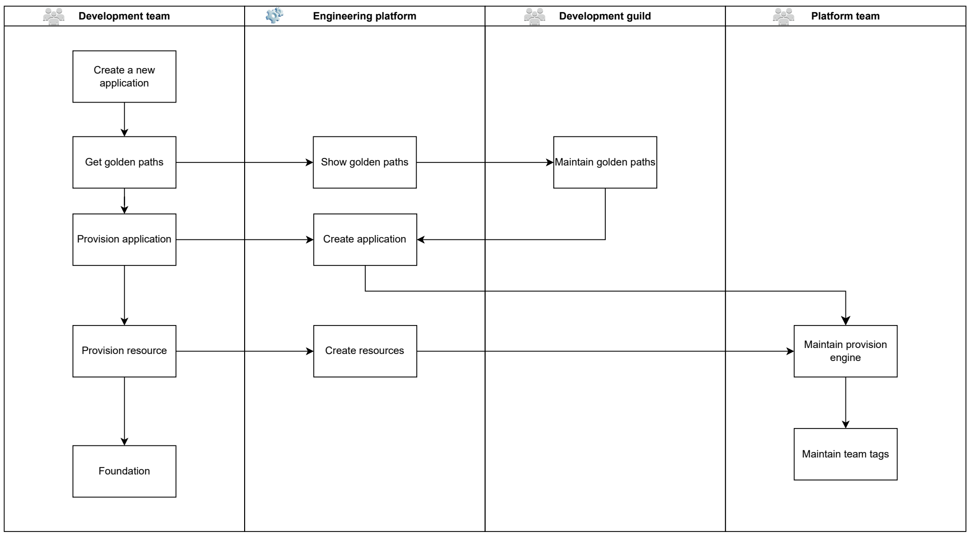


Figure 3.9: Enterprise viewpoint: provisioning service interactions

3.2.2 Information viewpoint

The information viewpoint specification enables the clear and concise representation of the information objects consumed and produced by the phases in the platform engineering and application lifecycles introduced in the enterprise viewpoint. A standard model that can be referenced throughout a complete design specification assures that the same interpretation of information is applied at all levels, including the activities related to these objects. The PE-RM information viewpoint aims to achieve a shared model for the design activity, given that it can cover a federation of tools and integration of legacy systems. The main modeling concepts of the information viewpoint are information objects and action objects, explained in Tables 2 and 3. From the information viewpoint, the emphasis is on the data, their evolution, and the activities which enable that evolution. The action objects are based on the stakeholders and roles introduced from the enterprise viewpoint. Based on the lifecycles defined in the enterprise viewpoint, the information and action objects will also have relationships since action objects cannot modify all information objects.

Platform engineering lifecycle

The platform engineering lifecycle can be separated into different phases; each phase has an input and output. The inputs and outputs, together with the information discussed in the different phases, can be translated into the following information and action objects:

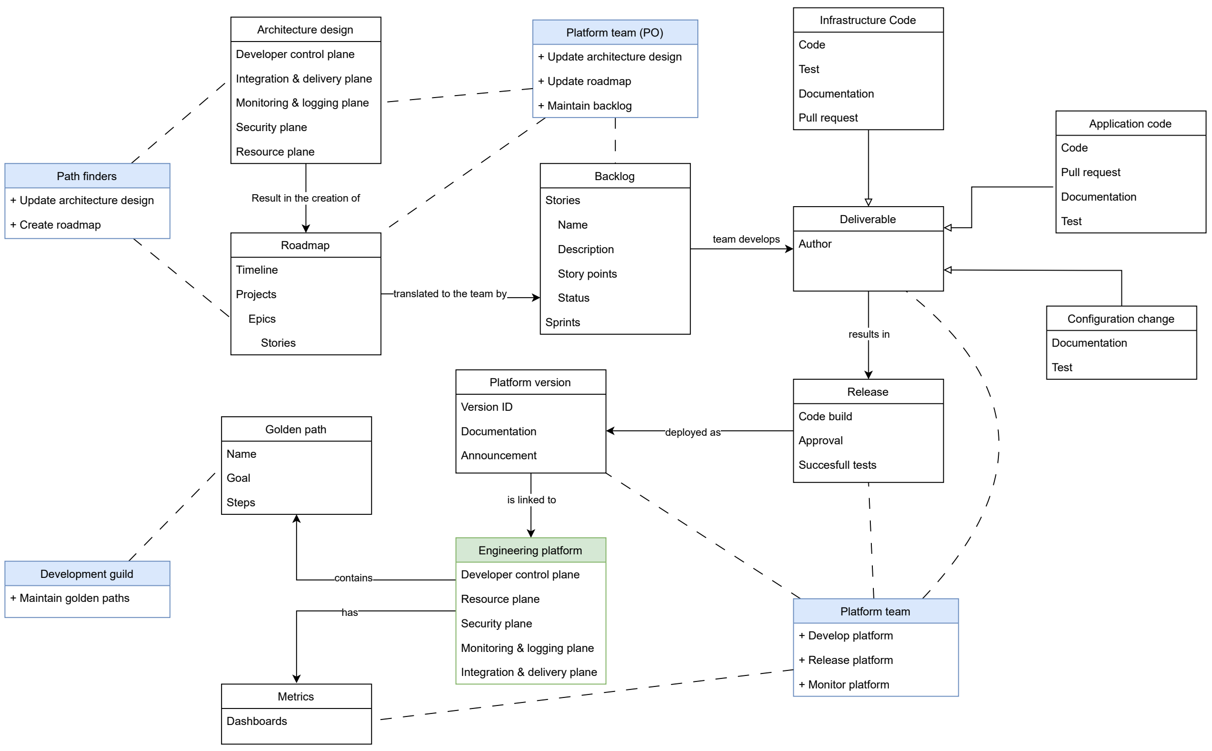


Figure 3.10: Information viewpoint: information object platform engineering lifecycle

For the platform engineering lifecycle, most objects will be created once and iterated upon during the continuous platform lifecycle. How and when these objects are updated can be found in figure 3.3. The information objects are:

Architecture design The architecture design is created based on observations of the pathfinders and is translated into an architecture that contains the five components of an engineering platform, which will be explained in section 3.2.4.

Roadmap The roadmap is a more functional component in the platform engineering lifecycle and is used to create estimation and budget by the management team. It contains the different projects, epics, and stories needed to hold the deadline. The roadmap is the source of truth for the platform. Once a feature request is approved, it most likely affects the timeline.

Backlog The backlog is used for the platform team to plan and create tasks based on the roadmap, which is understandable for the platform engineers. The stories on the backlog have at least a name, description, story points, and status. This can be different based on preferences.

Deliverable Deliverables of the platform can be different based on the type of functionality added to the platform. Most of the changes to the infrastructure are done by updating the infrastructure code, which contains tests and documentation. Moreover, the platform can also contain application code or configurations that need to be maintained by the platform team.

Release The release is a set of deliverables that must be deployed as a new platform version. To create a release, it has to be approved, and the tests must succeed. Depending on the type of release, it will also have to build code to ensure it will be working effectively in the platform.

Engineering platform Engineering platform (i.e., IDP) is the core of platform engineering. It can be split into five different planes: the developer plane, resource plane, security plane, monitoring & logging plane, and integration & delivery plane (explained in section 3.2.4).

Platform version The platform always has a current version to keep track of the improvements and make reverting easier. This contains a version ID, documentation, a possible announcement to the stakeholders, and a change log.

Golden paths Golden paths are created based on the self-service capabilities the platform provides at that moment. It will contain different steps and functionality to help development teams achieve specific goals.

Metrics Platform metrics are available to the platform team to validate the current state of the platform by using dashboards. It can vary on which metrics you want to show for the platform that will help give insight into possible improvements. These metrics could be related to the productivity improvements of the development teams, for example.

The information viewpoint for the platform engineering lifecycle will also include the following action objects:

Path finders The pathfinders are mainly responsible for the architecture and initial roadmap and can help to update it if necessary; they can also help identify the need for a new feature and create an architecture for it.

Product owner platform team The platform team product owner is responsible for the roadmap and the backlog on which the team works. Making sure that the deadlines are being met. The platform team also maintains the overall architecture together with the pathfinders.

Platform team The platform team is responsible for everything related to the platform. They will mainly interact with the information objects in the continuous lifecycle and focus on building features for the platform.

Development guilds The development guilds are responsible for maintaining the golden paths that can be updated based on a new platform version. This includes creating new golden paths or updating existing once based on a change in the platform.

Application lifecycle

Since we believe that the continuous development lifecycle of a project can be seen as a black box from the platform engineering perspective, we focus on the information objects related to the start of a project. We believe that the information objects do not change during the continuous development of an application. The only thing that will change is the application code and the data within the information objects. Still, once the architecture has to be updated, it will go through the defined lifecycle again. Resulting in the following information objects:

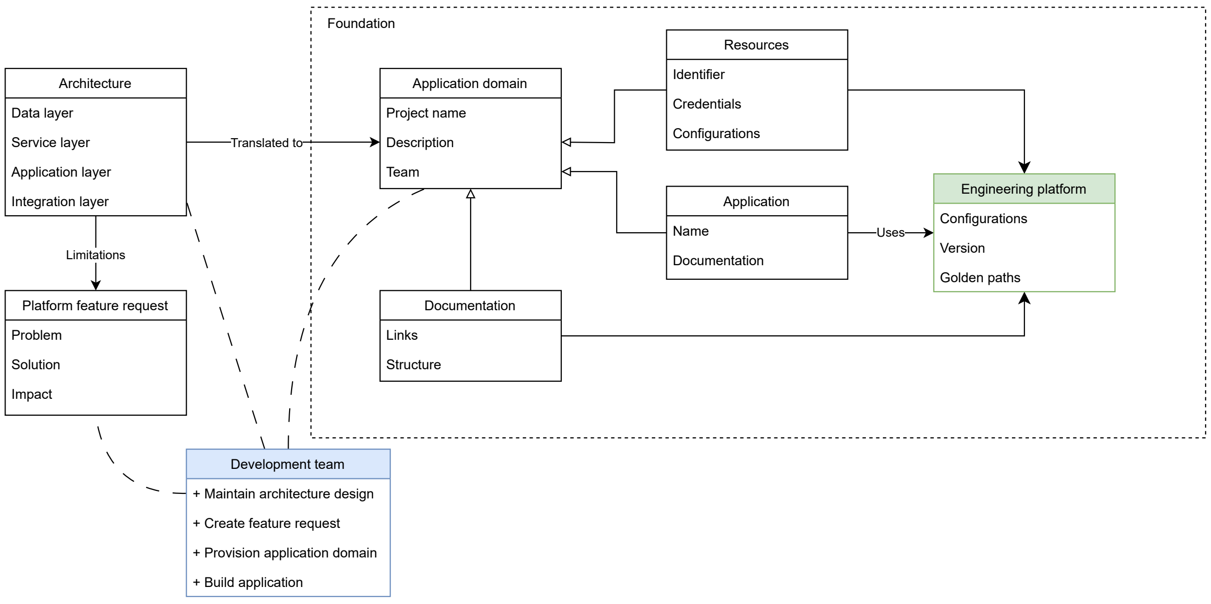


Figure 3.11: Information viewpoint: Information object project lifecycle

For the application lifecycle, the platform provides the foundation of the application domain by offering provision strategies and golden paths. This foundation is used during the continuous development lifecycle, including the self-service capabilities that improve productivity, and it includes the following components:

Architecture The architecture is the design of the application domain based on different layers. The architecture design can differ based on preferences, but overall, it is translated into resources, which the platform support. The architecture is affected by the golden paths and supported resources.

Application domain The application domain is all the applications and resources needed for a project and is maintained by the development teams. It will have a project name and description and the team responsible for this application. It translates the architecture and contains at least the following elements: applications, resources, and documentation. The golden paths are used to adopt best practices and support the integration of this application domain into the platform.

Documentation Each application domain will have the documentation necessary to support knowledge sharing and decrease knowledge siloes. This documentation must be accessible and easy to read by the entire organization. The engineering platform could help provide documentation frameworks or other supporting tools to encourage documentation.

Resources The resources are part of the application domain and can differ for each organization. It will most likely be databases or other related cloud services. The platform provides the resources and has default configurations and credentials that can be used to access the resources.

Application The application is the system built by the team; the platform and golden paths provision the foundation. The platform is responsible for provisioning and configuring at least a git repo, docker repo, and pipelines to ensure the development team can begin delivering business value. The platform also enables deployments to make the application available.

Engineering platform The application and the application domain use the platform's self-service capabilities to integrate the application into the platform entirely. This includes services and tools to get applications to production (like pipelines) and tools used in production (like networking, logging, and observability) without too much manual work.

Platform feature request A platform feature request is based on the problems the development team encounters while creating the architecture. The platform feature request must include the problem, a solution, and the impact of the change in the platform. This can include missing support or updating configurations.

Based on these information objects, the development team can work on the application domain during the continuous development lifecycle and change these objects. As soon as the development team encounters problems that require a change in the infrastructure, the application domain will be updated, and therefore, these objects will change.

The application lifecycle will also have the following action object:

Development team The development team is responsible for the entire application lifecycle and, therefore, can update the different information objects, especially the infrastructure and application domain. If they find missing features, it can be requested, which will start the other lifecycle.

The development guilds and platform teams are indirectly action objects in this lifecycle since they can change the platform, which could force development teams to update the application domain.

3.2.3 Computational viewpoint

The computational viewpoint specification models the components that provide different functionalities for processing data assets and allows the lifecycle to continue to other phases. The computational viewpoint is concerned with developing the high-level design of the processes and applications supporting the platform's self-service capabilities and activities done during the two lifecycles. The viewpoint expresses models in terms of functional components of the engineering platform and how different parts will interact with typed interfaces by performing a sequence of activities. The computational viewpoint specifications refer to specific parts of the engineering viewpoint. In this section, we have worked out several operations related to the operations with the platform.

Operations

To elucidate the various components of the platform, we've analyzed fundamental interactions and processes enabled by platform engineering. These interactions stem from the lifecycle and information objects proposed in the enterprise and information viewpoint. We've utilized UML and, specifically, sequence diagrams to model these operations. Although application deployment is outside our project lifecycle scope, modeling these interactions is crucial to understanding how the platform functions. Platform engineering serves two functions: facilitating application production deployment and maintaining applications in production. Our focus in this viewpoint will be on the first, and many of these operations are proposed as examples. Golden paths can play an important role in the translation of these operations into functionality.

Golden paths

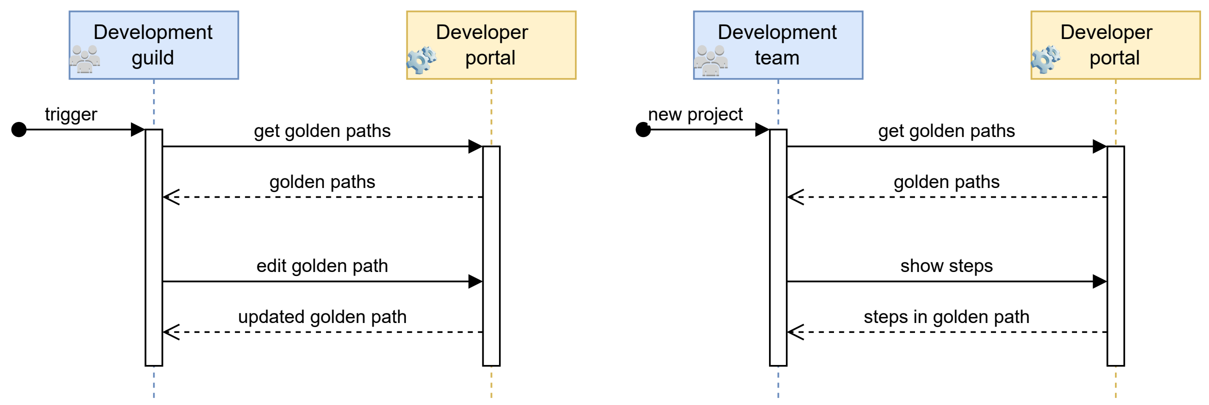


Figure 3.12: Computational viewpoint: Golden paths

Golden paths are one of the essential components to maximizing platform self-service capabilities and adopting best practices. Golden paths are part of the developer portal, a component of the Developer Control Plane. The operations of golden paths can be divided into two parts: the maintainability of the golden paths and the utilization of golden paths. The development guilds maintain golden paths, which anyone can access but can only be updated by the responsible development guilds.

For their utilization, the development team can find the needed golden paths on the developer portal. They can follow the steps in the golden path to achieve their goal. Golden paths can be used to create an application domain and migrate services to a new platform version. The golden paths also contain the provisioning service responsible for creating the application domain foundation, which we will explain in the following operation.

Provision application

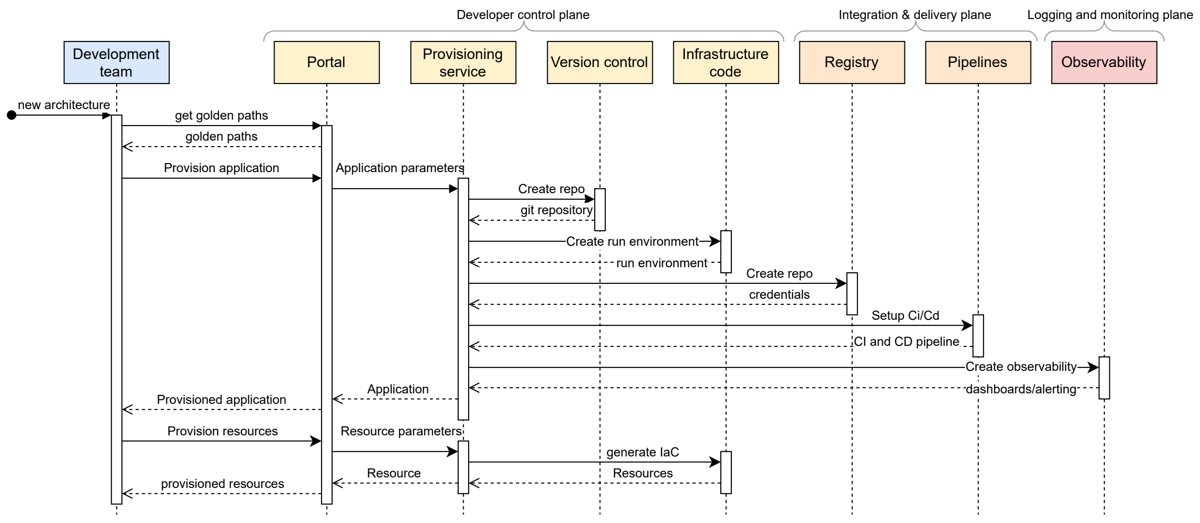


Figure 3.13: Computational viewpoint: Provision application domain (example)

In compliance with the architecture, the development team can provision the application domain via the platform and golden paths. The developer portal should offer golden paths that provide a step-by-step guide and a built-in provisioning service for application domain setup. According to the application parameters, the provisioning service generates a version control repository with the required compute resources. Embedding a provisioning service within the golden paths enhances the developers' experience by consolidating functionality and information. This operation ensures deployment tools and the infrastructure for running applications are in place.

A docker repository and necessary pipelines for building and deploying the application are created to guarantee application integration with the delivery plane. Out-of-the-box dashboards should be available to the development team for monitoring, and the provisioning service can create additional resources for the application. The provisioning service creates and deploys infrastructure code, which results in resource creation and ensures correct permissions and credentials. The standardization and centralization of tools and techniques are promoted, reducing tool proliferation within the organization.

The platform team is also responsible for maintaining applications in production. Thus, the necessary infrastructure for running these applications must be in place (i.e., networking, load balancing). To keep the reference model clean, this part is not modeled.

Deploy application

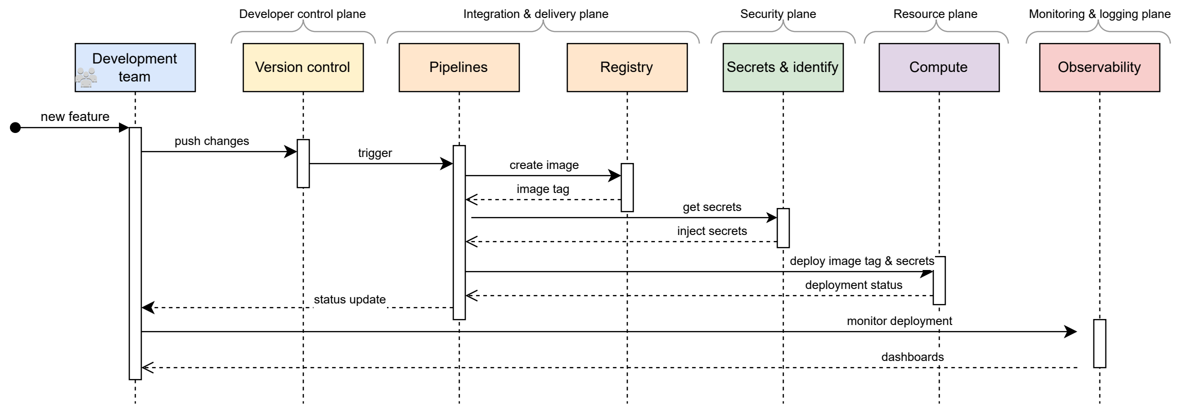


Figure 3.14: Computational viewpoint: Deploy application (example)

For application deployment, the development team interacts with the platform by pushing code to the version control. Based on their configurations, this triggers the CI pipeline, builds the application, and stores it in the registry. The registry returns the new tag and injects it into the CD pipeline, which deploys the application with the new tag. Depending on the configurations, the development teams could also interact with the developer plane or integration & deployment plane to deploy the application in a specific environment. The monitoring & logging plane can be used to monitor the application in production. Secrets needed for deployment are retrieved from the security plane and injected into the deployment.

The operations align closely with traditional lifecycles (i.e., DevOps); the only difference is the platform's automation, relieving the development team from maintaining pipelines, registry, compute resources, and observability. This is done by the platform and maintained by the platform team to lessen the cognitive load of software developers and enhance productivity.

3.2.4 Engineering viewpoint

The main goal of the engineering viewpoint is to represent the distribution of components among different software systems and tools. The engineering viewpoint tackles the problem of diversity in platform structure and gives the prescriptions for supporting the necessary abstract computational interactions in various situations, explained in the computational viewpoint. Interactions may involve communication between subsystems (components); accordingly, other engineering solutions will be used. Given that most organizations will already possess an infrastructure and a set of tools, it can be enhanced by reshaping it according to the model offered from the engineering perspective. The main modeling concepts of the engineering viewpoint are engineering objects, containers, and channels. However, to ensure that the platform engineering is correctly acknowledged, the engineering platform has been separated into different planes based on the IDP reference architecture outlined in section 2.1.1. The resource plane can be seen as the foundation on which the platform and applications will run. The engineering platform can be separated into two features: construction applications and enabling applications to be running on production, which is not explicitly modeled in this diagram.

Engineering platform components

The platform divides into five components or ”planes,” each containing various subcomponents or ”objects.” A concise overview is shown in figure 3.15. The resource plane, functioning as the foundation, supports all other planes and manages the running applications.

Our main focus lies in the platform’s functionality related to the application lifecycle and its transition to production rather than maintaining applications in production. This provides a clearer overview and sharper focus on the platform’s essence. A feature or operation only integrates into the platform once it aligns with the platform’s entire lifecycle, including all plane interactions.

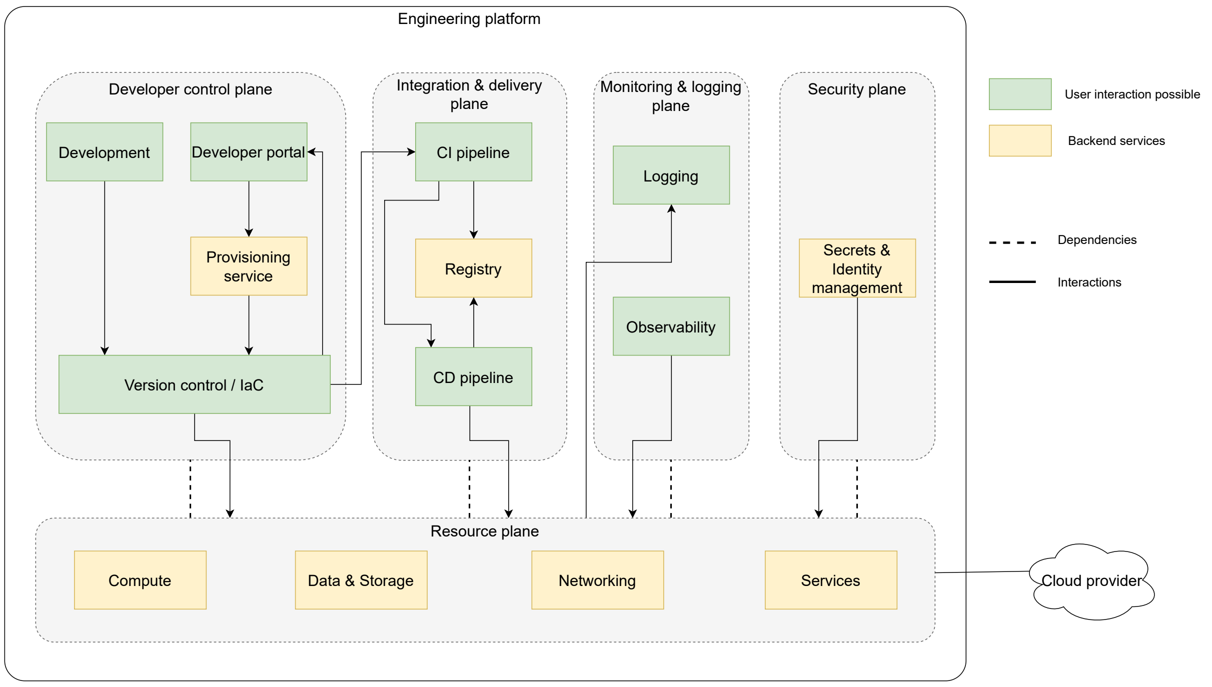


Figure 3.15: Engineering viewpoint: components of the engineering platform

The platform’s key components are identified in the diagram, which aims to display the high-level structure of a potential engineering platform, as well as interactions and dependencies among components. The platform comprises five nodes called ”planes”:

Developer control plane

Developers use the developer control plane to interact with the platform, and it contains the following components (objects):

- IDE: the local development environment of the development teams; this also includes the backlog management systems.
- Developer portal: a frontend for development teams to interact with the platform. It contains at least the following essential components: service catalog, documentation, and golden paths. It is up to the organization if more features will be integrated.
- Provisioning service: this service can be introduced to provision application domains, and best practices, and ensure governance is in place. It communicates mainly with the IaC and other planes to ensure the development team has all the tools and resources.
- Version control: this is the central location where the platform is stored with Infrastructure as Code (IaC) and contains the different applications built by the development teams. It will improve transparency, and best practices can easily be adopted.

The development teams most likely interact with the platform through code by pushing code to the version control. Moreover, the developer portal is the frontend that contains helpful information like golden paths and the service catalog in which developers can interact through a user interface.

Integration & delivery plane

The integration & delivery plane is an important platform plane since it glues the developer control plane and resource plane together by automating the build and deployment of the different applications. Still, it can also play a role in the provisioning of monitoring and logging resources. It includes the following nodes:

- CI pipeline: linked to the version control and gets triggered as soon as code is pushed. It will depend on the setup and configuration of how and when this pipeline will be triggered.
- Registry: stores all releasable artifacts like docker containers and packages, depending on the use case. Linked to the pipelines to increase automation.
- CD pipeline: updates an environment with a new artifact stored in the registry and can be triggered by the CI or manually. It could also be responsible for deployment-related tasks.

Depending on the tools used for the CI/CD pipeline, the communication between components can differ. The platform team maintains these pipelines to create a centralized and standardized way of building and deploying applications. This way, development teams do not have to worry about the CI/CD and can focus on the development.

Monitoring & logging plane

The monitoring & logging plane is offered by the platform to give the development teams more insight into their application. By provisioning the application by the provisioning service, the platform will make sure that observability and logging are available, containing the following nodes:

- Observability: metrics and dashboards created by the platform to offer insight into the applications. The observability component will pull the metrics from the different resources. The platform can offer default dashboards to provide metrics out of the box. It is up to the organization and use case what kind of observability tools will be used, like tracing or alerting, could also be included.
- Logging: logging endpoint that the development teams can use to send logs for debugging. It will have to store many logs and therefore use TCP to receive them. The platform will also offer querying tools to read logs.

It depends on the use case and organization what kind of monitoring and logging is needed and how this is configured. The platform team is responsible for these functionalities, and the development teams for using these platform features.

Security plane

The security plane is available to offer security needs for applications that require secrets or other sensitive data. It contains the following nodes:

- Secrets & identity management: manages all the secrets, including credentials used to deploy an application with the correct environment variables. It also takes care of the permissions to ensure that applications and users have the correct role to perform specific actions.

Using the platform to handle permissions and policies, the security can be centralized and standardized to improve governance. By making sure that the platform handles security and permissions, it will make it easier to guarantee strict rules and guidelines.

Resource plane

The resource plane contains all the resources necessary to run all the applications within the organization. It can depend on the type of organization in which resources are needed, but it includes at least the following:

- Compute: the applications will be running in this plane. This can also include applications and tools used in the engineering platform and, therefore, essential to set up first.
- Data & storage: resources that can store data and other artifacts like images or files. It also has to be reachable by applications running in production and can be handled by the security plane.
- Networking: enabling applications to be reachable from the outside world or internal systems, including uptime and proper DNS.
- Services: services that are used by applications for certain functionality. For example, a message broker or email service.

Depending on the use case and size of the organization, the number of resources available can differ. Moreover, the type of resources you want to make available within the platform can differ. The resource plane also contains all the cloud provider services that the organization uses; this is the part that interacts with the cloud.

The different nodes interact via API calls orchestrated by the platform maintained by the platform team. The goal is to simplify application deployment in the resource plane, minimizing overhead and manual configurations. Development teams primarily interact with the developer control plane, which communicates with the integration & delivery plane, updating the resource plane accordingly. The monitoring & logging plane receives information from the resource plane and can be adjusted by the development teams through code updates. By abstracting the underlying infrastructure and automating processes, including application provisioning and deployment, development teams can focus on coding and problem-solving, improving productivity and developer experience by reducing cognitive load.

3.2.5 Technology viewpoint

Technology Viewpoint specifications represent the concrete dependencies between design and implementation. The technology viewpoint is concerned with managing real-world constraints, such as existing application platforms, tools, or restrictions based on requirements and budget. The adoption of platform engineering will never really have the luxury of being a greenfield project [32], and this viewpoint brings together information about the existing infrastructure and technology stack. It is concerned with the selection of universal standards to be used in the system and the allocation and configuration of resources. It represents the software components and techniques of the implemented system based on the engineering viewpoint specifications. Bringing this all together, it expresses how the specifications for an ODP system are to be implemented.

This viewpoint also has an essential role in the management of testing conformance to the overall specification because it specifies the information required from implementers to support this testing. The main modeling concepts of the technology viewpoint are *components* and *standards* and, in our case, *best practices.* Because the engineering platform is a central collection of tools, services, and automated workflows and can differ for each organization and environment, this viewpoint does not focus on state-of-the-art tools. In table 3.1, the diagram represents the platform components and the best practices/standards.

Best practices

Since it can depend on the organization which technologies you want to use to create the engineering platform, we will focus on the different best practices of each component instead of state-of-the-art tooling. In table 3.1, we have stated the requirements and the best practices for each component.

Component	**Requirements**	**Best practices**
	Developer control plane	
Development	The development environment is based on the developer's preferences. The backlog management tool will most likely depend on integrations.	IDE & Backlog management
Developer portal	A user interface that contains a service catalog, golden paths, platform documentation and versioning, to reduce cognitive load for development teams.	Developer portal & ChatBot
Provisioning service	Used to provision application domains and ensure governance is in place. Depending on the use case how this can be implemented.	Microservice
Version control / IaC	Depending on the software strategy and the current technologies, platform, and configuration of infrastructure.	GIT & IaC
	Integration & delivery plane	
CI pipeline	Depending on the existing tools and the amount of customization each team needs, a pipeline can be chosen.	Containerization
Registry	Depending on the way the organization wants to deploy applications and how to store artifacts.	Repositories
CD pipeline	The CD pipeline will most likely look the same across different types of applications, and therefore, it must be centralized, depending on the CI and cloud provider.	Containerization
	Monitoring & logging plane	

Observability	Observability can be very diverse and depends on the number of services and requirements. Different types of observabilities can be considered.	Metrics Scraper & Elastic storage
Logging	Logging tools are based on an estimation of messages that will be sent to the logging cluster and requirements on querying logs.	Elastic storage & Querying tool
	Security plane	
Secrets & identity	Secret and identify will most likely be offered by the cloud provider and it depends on the organization to which degree this is needed.	Role management & secret manager
	Resource plane	
Compute	The compute will be decided based on the infrastructure the organization want to run on and what is already configured.	Docker & Kubernetes & serverless
Data & storage	This will be based on the existing way of storing data in the organization and choosing the best options.	Databases & file storage
Networking	Networking will be based on the cloud provider and the front-facing frontend that are being built.	Load balancer & gateways
Services	This completely depends on the use cases of the organizations. A possible service is a message broker.	Example: message broker

Table 3.1: Technology viewpoint: engineering platform components

3.3 Validation

3.3.1 Reference model comparison

The proposed reference model, which partially draws upon the existing IDP reference architecture, facilitates a comparative analysis between the two. Unlike the reference architecture, which mainly focuses on technical aspects such as the technologies used, the proposed reference model encompasses a broader perspective. The proposed model includes technical components and emphasizes the cultural and organizational shifts necessary for adopting platform engineering. This makes the proposed model more comprehensive than the solely technology-driven IDP reference architecture.

The existing IDP reference architecture has its merits, with specific tools and techniques designed for seamless integration with the AWS cloud provider, giving an understanding of a possible technical implementation. Compared to the proposed reference model, the engineering and technology viewpoint also focuses on the technical implementation of platform engineering, but in a more generic way. This freedom of choice, coupled with the comprehensive nature of our model, makes it versatile and easily adaptable across different organizations. Conversely, the proposed reference model remains a support tool and is not tool-specific, articulating only the best practices for distinct components.

The biggest gain of the proposed reference model compared to the IDP reference architecture is the comprehension of lifecycles and the cultural impact of platform engineering. Platform engineering will introduce different working methods which are not visible in the existing reference architecture. The proposed reference model explains the different lifecycles, the stakeholders, and the separation of accountability between different roles. In the IDP reference architecture, many claims are made about the benefits of the Internal Developer Platform and multiple concrete examples are given. Within these examples, assumptions are made which are not explained. An example can be given that when a resource is not known to the IDP, it has to be added to the general baseline, but it doesn't explain how that process works. That is one of the reasons why this reference model is more comprehensive than the existing reference architecture.

3.3.2 Expert feedback

We conducted 8 expert interviews with staff in different positions to identify the relevancy, applicability, and validity of the proposed reference model. The scripts for the interviews differ in structure, types of questions, and number of questions, depending on the role and expertise of the interviewee. Experts were selected for voluntary interviews based on their position, status, and experience. Further criteria for selecting the experts were knowledge of relevant functions, the ability to provide accurate information, and their availability to be interviewed. In section 3.1, the different experts are explained. We also interviewed an external expert to get a broader understanding. In the following, we discuss our findings based on the expert interviews.

Five experts underscored the crucial need for a distinct separation of accountability between development and platform teams. This delineation of roles and responsibilities fosters streamlined processes and enables effective operations. The development teams focus on creating and refining software products, whereas the platform team's duty revolves around constructing and maintaining the platform. This division is made clear in the reference model and paves the way for clear accountability within the organization.

Second, three experts concurred that adopting platform engineering does not inherently alter the development lifecycle. Instead, platform engineering enhances this lifecycle by introducing advanced tools, streamlined workflows, and an infrastructure that provides greater efficiency and reliability. It acts as an enabler, not a disruptor, allowing development teams to continue their lifecycle activities with increased productivity.

Moreover, five experts emphasized that the engineering platform has two key roles - guiding applications to production and ensuring their continued operation post-deployment. By elaborating on these responsibilities, misunderstandings and ambiguities can be effectively mitigated. This level of clarity promotes better alignment between different teams and aids in fostering a more efficient, collaborative environment. Consequently, this transparency contributes to improved validity and reliability of the reference model.

Finally, three experts, mainly related to the platform lifecycle, suggested that improved communication with development teams could significantly enhance the platform's applicability. This dialogue becomes even more critical in the context of platform alterations that could impact development work. Their insights imply the value of having a structured process to communicate changes, as highlighted in our reference model through implementing "golden paths" and versioning.

Chapter 4

Case study

This chapter seeks to demonstrate the practical application of the Platform Engineering Reference Model (PE-RM) to effectively design and implement platform engineering within a software organization. We will establish the credibility and relevance of this model by elaborating on its applicability by conceptual design and validating its technical implementation through various experiments. Consequently, the chapter is divided into three sections: Conceptual Design, Technical Implementation, and Experimentation.

4.1 Conceptual design

In this section, we will delve into utilizing the Platform Engineering Reference Model (PE-RM) to develop a conceptual design tailored for a software-centric organization. In doing so, we enhance our comprehension of the Reference Model and verify its applicability within an organization. To design platform engineering with specific reference to the organization, we will analyze the existing setup, extending to a functional and technical understanding of the technology department and its software processes within a software organization. Our exploration will focus on the current state and the areas of potential improvements. From these observations and contextual analysis, we will outline specific requirements that aim to enhance the existing practices with platform engineering. Consequently, leveraging the insights obtained from the analysis and defined requirements, we will construct a platform engineering design underpinned by the PE-RM. To conclude this section, we will reflect on the relevance and adaptability of the PE-RM to clarify how the model can be strategically deployed within an organization.

4.1.1 Analysis

Wehkamp is one of the leading online retailers in the Netherlands, offering a broad range of products, with more than 400,000 items and serving millions of customers. In order to serve all these customers and ship thousands of parcels daily, the organization needs to have software systems to support these requests each day. Wehkamp has over 100 employees in the technology department working on different parts of the website and related software systems.

Organizational setup

The technology department has many software development teams divided into store/platform and core/-operations (explained in figure 4.1). Every department is overseen by a Tech Head, who is ultimately responsible for their section and is also part of the Tech MT. There are different types of teams within this department, and they could be different in their team construction. In table 4.1, we explained the different types of teams and the roles they include. Table 4.2 contains a list of all the development expertise within development teams. In addition to these teams, the technology department also includes pathfinders who are not part of a team and have a head of pathfinders which, in the case of Wehkamp, is the CTO. Their primary role involves providing technical input in decision-making processes, thus, assisting in guiding the overall software processes within the organization. Moreover, they are also available to assist development teams when needed.

An essential team for this analysis is the cloud team, which is responsible for the container platform we will research, and consists of 7 full-time employees. Currently, the technology department has enough engineers for the workload they have. However, the cloud team could be seen as understaffed due to the transition to a new container platform which requires a lot of expertise and manpower.

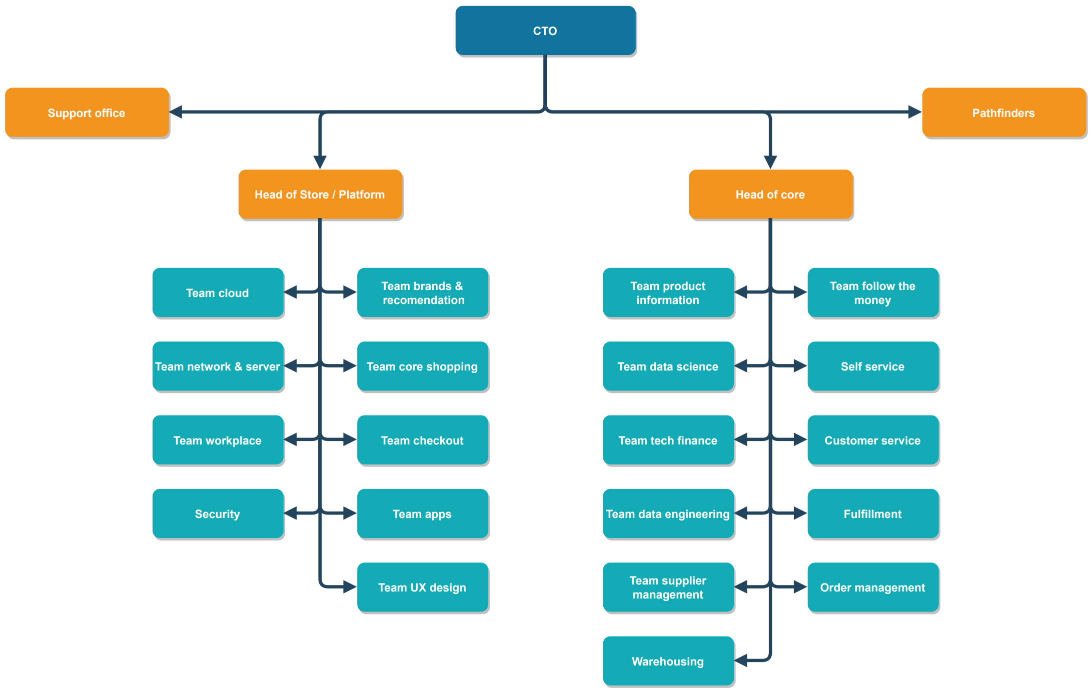


Figure 4.1: Wehkamp hierarchy: technology department

Team	**Roles**
Development team	Tech lead, product owner, scrum master, testers, backend/frontend developers
Cloud team	Product owner, tech lead, scrum master, cloud engineers
Network & server team	Product owner, tech lead, sys admin, network admin
Special teams	Product owner, tech lead, configuration engineers, information analysts
Workplace team	Service desk, sysadmins (external hired)

Table 4.1: Type of teams.

Expertise	**Description**
Frontend developers	Frontend applications build in JavaScript with Node.js and React. It also includes the maintenance of custom NPM packages.
.NET developers	Many microservices are built in .NET and running in containers. This also includes NuGet packages.
Java developers	Some older services are built in Java, and not all are running in the cloud, but on-premise. This also includes Maven packages.
Scala developers	Many services are built with Scala in a dockerized environment. However, they plan to migrate from Scala to Java or .NET.
Python developers	Python is used across different applications and services. Mainly as a supporting tool and, in some cases, as a backend service.
Data scientists	Data scientists mainly work in external systems like Databricks. They interact with different parts of the platform and develop in Python and Scala.
Data engineers	Data engineerings mainly work on the data platform (delta lake, vent ingest, reporting pipelines) and develop in Python and Scala.

Table 4.2: Type of developers.

Technology stack

Wehkamp's infrastructure comprises two key divisions: the on-premise and the cloud-based systems. Our investigation, however, will be centered primarily around their cloud infrastructure which serves as the core of their operations. Wehkamp's cloud environment allows development teams to deploy their microservices and microsites as containers. While these applications are encapsulated in containers, the choice of programming languages is subject to certain constraints, primarily due to an architectural choice since they believe limiting the number of languages will make engineers more mobile in the organization (see table 4.2). For the adoption of a new language, it's required that the development team first discuss and receive approval from the pathfinders; otherwise, they must select from the already-approved languages. At present, Wehkamp is transitioning towards a new container platform. Therefore, we found it necessary to deliver a comprehensive overview of all tools deployed across the current and new environments. This analysis of the tools was carried out using the framework provided in the engineering viewpoint of the PE-RM (proposed in section 3.2.4). The architecture of both environments, analyzed from the perspective of this model, is explained in table 4.3 and explained in more detail in appendix B.

The main driving force behind Wehkamp's decision to establish a new environment is its commitment to staying current with state-of-the-art technology. This fits the requirements of automatic scaling and cost efficiency. This shift is prominently characterized by their adoption of Kubernetes [34]. By transitioning to a new infrastructure, there will also be the opportunity to support more capabilities for the development teams, which we have stated in table 4.4. While the new environment is largely geared towards enhancing the deployment and production performance of Docker containers, it is essential to note that not all categories have been subjected to change. To fully grasp the incremental value that the new environment imparts to the development teams, we have drawn comparisons between the capabilities of both platforms in table 4.4. In appendix B, we have stated an overview of all the tools based on the architecture framework proposed in the engineering viewpoint of the PE-RM.

Category	Technology current environment	Technology new environment
Developer control plane		
IDE	JetBrains, Visual Studio (Code)	JetBrains, Visual Studio (Code)
Portal	SlackBot, Confluence, Jira, Trello	SlackBot, Confluence, Jira, Trello
Version control	GitHub, Terraform, *Ansible*	GitHub, Terraform
Integration & delivery plane		
CI pipeline	Jenkins	Jenkins
Registry	AWS ECR, JFrog, NPM, NuGet, PyPi, Maven	AWS ECR, JFrog, NPM, NuGet, PyPi, Maven
CD pipeline	Jenkins, Cypress, Postman	Jenkins, *ArgoCD*
Monitoring & logging plane		
Observability	Thanos, Grafana, Prometheus, AlertManager, PagerDuty	Thanos, Grafana, Prometheus, AlertManager, PagerDuty, *Jaeger*, *Kiali*
Logging	Elastic Search, Kibana, Kinesis, *Fluentd*	Elastic Search, Kibana, Kinesis, *Fluent-Bit*
Security plane		
Secrets & identify management	GitHub, IAM, *Consul*	GitHub, IAM, *Secret Manager*
Resource plane		
Compute	Docker, *Mesos*, *Marathon*, *Consul*, *ZooKeeper*	Docker, *Kubernetes*, *Keda*, *Karpenter*
Database	PostgreSQL, Redis, Elastic Search, DynamoDB, S3	PostgreSQL, Redis, elastic storage, DynamoDB, S3
Networking	Cloudflare, Elastic Load Balancer, VPC, *HAProxy*, Certificate Manager, *Ngnix and Lua*	Cloudflare, Elastic Load Balancer, VPC, Certificate Manager, *Istio*
Messaging	Kafka	Kafka (MSK)

Table 4.3: Technologies.

Capability	**Current platform**	**New platform**
Configuration management	A custom repository with CD job to push secrets from GitHub to consul that can be injected into the containers. In combination with docker labels	Secrets are stored in the secret manager, and the argo applicationSet is stored in a different workload file.
Kafka message broker	Self-managed Kafka server.	Managed Kafka server.
Terraform state	One big terraform state in a GitHub repository.	Three stages of terraform states: (1) account seeding: including networking and vpc. (2): platform: including guard duty and EKS clusters. (3) Application provisioning: terraform code for an application.
Policy management	Services have access to all resources.	An application has, by default, no access to resources, and policies have to be added and approved.
Application scaling	Is possible in Mesos/Marathon but is never configured.	Based on metrics and Keda, the application can be scaled automatically to manage the workload.
Platform scaling	In Mesos/Marathon, they used a headroom function to scale, but it is not very cost-effective.	With the use of Karpenter, the platform is able to buy spot instances and add them, making it more cost-effective.
Cron job	Chronos is installed in Blaze to run a container.	The new platform offers cron job support with the needed observability.
Running a job	Jenkins is used for some maintenance jobs; application-like jobs are in the container, which could be a huge risk.	Creating a job (K8s jobs) on Kubernetes and terminates once done.

Table 4.4: Platform capabilities.

Development and deployment process

In addition to the technological stack, it's also crucial to assess how the development teams operate and the kind of processes within the organization. To provide clarity and structure, we have divided the processes into distinct categories:

- *Development Methodologies:* Development teams adopt Scrum with two-week sprint cycles and a DevOps approach allowing independent application deployment. Platform teams manage networking and cloud operations. ArgoCD with GitOps is utilized in the new environment, and a chatbot, SlackBot, is used to interact with the platform by provisioning functionalities.
- *Version Control Practices:* Teams use a Slackbot (together with a provisioning service) to create GitHub repositories, Docker registries, and CI/CD pipelines. The configuration of GitHub repositories is at the discretion of each team. The Git strategy uses master and feature branches, with automatic deployment upon code push to the master branch.
- *Pipelines:* Creating a GitHub repository via Slackbot triggers the creation of ECR repositories and related CI/CD pipelines. Builds are done in Docker containers with Jenkins jobs. Continuous scanning of GitHub enables automatic trigger of Jenkins CI jobs, which has too many permissions. A GitOps approach with ArgoCD is implemented in the new environment.
- *Deployment Strategies:* Feature branches aren't deployed by default, but a parameter can allow it. Master branch code is auto-deployed to development, then production. Unique Docker images simplify rollbacks. In the new environment, batch deployments smooth out the process.

Based on these development and deployment processes, many significant initiatives already help increase the productivity of the development teams. However, with these practices, there are also some downsides. For example, it can take some time for Jenkins to visualize the Pipelines necessary to deploy applications since they are centralized and not easily customizable by development teams. Each development team has a lot of freedom in adopting its guidelines and rules related to its git strategy and coding standards, making standardizing more difficult.

Productivity and performance metrics

To evaluate the current workflow, we identified key performance metrics that could be influenced by platform engineering, such as onboarding times for developers and applications and deployment rate. Developer onboarding time signifies how long a new engineer takes to deploy their first application. In contrast, application onboarding time measures the time needed to deploy a new application with all required tools. Deployment rate tracks the average number of weekly deployments.

To measure these, we analyzed three new developers for developer onboarding time, conducted an experiment to measure application onboarding time, and used various sources for deployment rate. These metrics are stated in table 4.5, and we elaborate on these metrics in more detail in appendix B.2. However, these metrics showed that the organization lacks a way to gather this information, hindering performance improvement.

Metric	**Description**	**Result**
Developer onboarding time	The time it takes for a new engineer to deploy an application to production.	Average: 15 working days
Application onboarding time	Time to deploy an application to production.	Total time: 185 minutes
Deployment rate	The average amount of deployments done for an application to production.	Current environment: 153 per week, new environment: 84 per week

Table 4.5: Performance metrics.

Despite existing tools aiding developers in specific tasks, much work remains manual. For example, developers can use a Slack bot for creating repositories and pipelines in application provisioning. However, they must create metrics exposure manually, follow best practices, and set up database policies. This increases the onboarding time and error rate, especially with a GitOps approach where Yaml files are essential. In addition, this will also increase the developer onboarding time since it can be difficult and confusing to get started.

Maintenance and support

From a platform perspective, providing platform feedback primarily involves direct messaging to the platform team to initiate a discussion. When platform issues arise, the cloud team generally undertakes repair efforts promptly. However, there is a formalized method for requesting new platform features or notifying development teams about platform malfunctions. However, this is not generally adopted and used, making feature requests less frequent. The primary mode of communication between the platform/cloud team and the development teams is Slack, where an announcement can be made on a specific channel.

Slack also serves as the principal communication channel for support, assisting developers with resolving specific platform-related issues. Whenever the platform undergoes an update, this information is communicated through Slack. Platform documentation pages are available, but they are located at different locations. However, this is not a structured means of communicating new platform features.

Conclusion

While Wehkamp already demonstrates several characteristics of platform engineering, both organizationally and technically, numerous unexplored aspects of platform engineering could be pivotal for a more efficient implementation. The transition to a new container platform has resolved many technical challenges; however, fundamental issues remain. Therefore, we can infer that significant enhancements could be achieved by implementing platform engineering on both technical and organizational levels to refine the overall processes.

A primary area of improvement will be the onboarding time of applications and employees. The time required to provision and deploy an application to production, equipped with the necessary tools and observability, is relatively extensive. Additionally, there's an abundance of manual configuration that could disrupt a centralized and standardized workflow, but more importantly, it is prone to errors. Wehkamp's usage of a Slackbot, which solely creates repositories and pipelines, does not address the creation of code or ArgoCD manifests needed for application deployment. This lack of automation could lead to time-consuming, non-standardized processes across different applications when performed manually by developers. For new developers, it could be daunting to understand the existing applications and their functionalities, thereby increasing onboarding time. Furthermore, the lack of standardization in observability and logging could extend the time to resolve production issues.

Organizationally, the informal communication between the cloud and development teams could pose challenges. The absence of formal opportunities to provide feedback or request new features could impair communication, resulting in reduced alignment. The platform lacks official documentation that could aid developers in understanding changes made by the platform team or the appropriate usage of tools.

4.1.2 Requirements

From the insights gathered during our case study analysis, we will formulate specific requirements on which a potential platform engineering initiative should be based. These requirements are exclusively derived from our observations, and will subsequently be translated into a functional and technical design, in line with the reference model. Since these requirements are based on our observations, we can not make them measurable, and they will be used more as guidelines.

Organizational requirements

Here, we detail the organizational requirements of platform engineering and the criteria it needs to meet. These requirements express the desired organizational changes that the adoption of platform engineering should bring about. Given that this organization has already incorporated some aspects of platform engineering, we will only lay out new requirements that arise from potential improvements compared to the current operational methodology.

1. Improve communication and elaboration between the platform and development teams to increase the effectiveness of the platform features
2. Introduce formal accountability rules, who is responsible for maintaining a tool, and when will it become part of the platform team or development teams.
3. Improve the onboarding time of developers.
4. Integrate policies on when to use tools; they can only be used once they are fully integrated into the platform.
5. Formal communication channels between developers and the platform team to increase elaboration.

Technical requirements

Here, we outline the technical requirements of platform engineering and the standards it needs to fulfill. These requirements represent the expected technical enhancements the engineering platform should introduce to the organization. As Wehkamp has already embraced some elements of platform engineering, we will focus on presenting the new requirements and technological modifications based on possible enhancements when compared with the existing mode of operation. We only focussed on the technical requirements that platform engineering could bring to the organization.

1. Measure productivity and performance metrics to track progress over time to see the impact of platform improvements.
2. Improve onboarding time of applications to make it easier to deploy to production.
3. Information about applications should be easier to find. This includes documentation and other relevant information.
4. The platform needs to have versioning to get more control and formality on new changes.
5. The platform should offer tools to help with deployment configurations
6. The CI needs to be more secure by ensuring it only runs in a sandbox.
7. Reduce logging costs by introducing rules on when to log certain levels and what handles these logging requests.
8. Services do not have access to other services by default. This has to be defined to increase security and an overview.

4.1.3 Design

In this segment, we will construct a platform engineering design drawing on the analysis conducted and the requirements compiled throughout the case study. Leveraging the Platform Engineering Reference

Model, we propose a platform engineering methodology designed to address particular challenges and enhance the overall software procedures within the organization. It will also serve to verify the validity of the reference model crafted in chapter 3. To provide deeper insights into the utility of the reference model, this section will be organized according to different viewpoints.

Enterprise viewpoint

Implementing platform engineering involves two lifecycles within the organization: platform and application (detailed in section 3.2.1). Communication between platform and development teams needs to be formalized to streamline the platform lifecycle, transition from ad hoc Slack messages to a structured approach, and facilitate feature requests. This also includes better support from the platform team to the development teams. Since we treat the application development lifecycle as a black box, we focus on the initial state, which can significantly impact development. This lifecycle can be enhanced by introducing methods to support application provisioning through a centralized, standardized tool that supports golden paths.

Various stakeholders play essential roles in platform engineering integration, as explained in section 3.2.1. These roles must be clearly defined for the entire technology department. In table 4.6, we outline these roles, responsibilities, and interactions with different lifecycles. A meaningful rule of thumb is the accountability of the platform team - if two or more development teams utilize a tool, it should fall under the platform team's responsibility.

Stakeholder	**Responsibilities**	**Interactions**
Platform team	The organization already has a platform team, which is responsible for the platform	They will improve the platform, create new releases and support development teams by offering easy-to-use tools.
Development teams	They are the users of the platform.	These development teams are the stakeholders of the platform and could request new features.
Development guilds	They have to be formed to offer golden paths to developers to push best practices and enable integration with the platform	They will offer support to development teams by golden paths
Pathfinders	Pathfinders are already within the organization and overlook the entire software process.	They help with platform adoption, feature requests, and infrastructure design of the platform.

Table 4.6: Stakeholders.

With the introduction of stakeholders and lifecycles, we formalize many stakeholder interactions, making it easier to leverage platform engineering. We focus on three key interactions: **(1)** Make feature requests more accessible by designating the platform product owner as the primary contact. **(2)** Focus on deployment when developing new platform features and bundle deployments into releases. **(3)** Introduce rules for project provisioning through the platform to increase standardization and adherence to best practices. Further details on these interactions can be found in section 3.2.1 of the Platform Engineering Reference Model.

Information viewpoint

Right now, the organization is missing some key information and action objects that could help adopt platform engineering (for reference, see section 3.2.2). The following information and action objects need to be introduced:

- *Platform version:* This would improve documentation and facilitate communication between the platform and development teams. Bundling deployments into a single release enhances the change log clarity and helps development teams understand what has been changed, promoting tool adoption and standardization.
- *Golden paths:* Implementing golden paths on the engineering platform would boost productivity by enabling the automatic creation of new applications with all necessary tools included. This could reduce onboarding time and decrease code or configuration errors.

- *Metrics:* The platform currently lacks detailed productivity metrics and other data that could enhance understanding of the platform. Such metrics are vital for identifying areas of improvement.
- *Platform feature requests:* These would allow development teams to create new applications using preferred tools. With the enforcement of golden paths, it would become easier for teams to make platform requests for unsupported functionalities. Especially a formal process to investigate features could benefit from a bigger discussion.

Computational viewpoint

From the computational viewpoint, we will focus on the operations that will most impact the organization related to the analysis and requirements. The main improvement is the provisioning of an application domain, which would also result in the introduction of golden paths. This would introduce new interactions between the development teams and the platform. With the introduction of golden paths, the translation of these operations can become easier since each operation can be created in a golden path.

(1) Golden paths are the first gain to improve standardization across the organization by developing golden paths for development teams to provision an application. Therefore the organization introduces a developer portal to make these golden paths possible and a development guild to manage these golden paths. Without these components, it will be challenging to create the golden paths to offer advantages to development teams in the provisioning of their application domain. In section 3.2.3, we explained this operation in more detail. For these golden paths to be useful the development guild needs to work together with the platform team to offer full integration with the platform. Although the implementation of golden paths is up to the organization, these are possible options based on our observations:

- Provisioning of react application: this will reduce the onboarding time of new applications as it takes a lot of time to provision applications in the current way of working.
- Provisioning of ArgoCD applicationSet for deployment: this will reduce errors and increase standardization for deployments, reducing maintainability.
- Migration of application from current to a new environment: help developers reduce the configurations they have to do to migrate their applications to the new environment.

(2) Provisioning of applications is the second improvement that has to be done and can be achieved with the use of golden paths, a developer portal, and a provisioning service. Within Wehkamp, they already have a provisioning service. However, this service only provisions the basic resources and does not take care of scaffolding an application. In order to fully take advantage of platform engineering, scaffolding has to be done. This will include exposing the correct metrics across all applications and adopting best practices. In section 3.2.3, we explained this in more detail. However, the sequence could depend on each organization. In the case of this case study, we have worked out two golden paths. For more details see figure 4.2 for the sequence of provisioning an application and figure 4.3 for provisioning the deployment for an application.

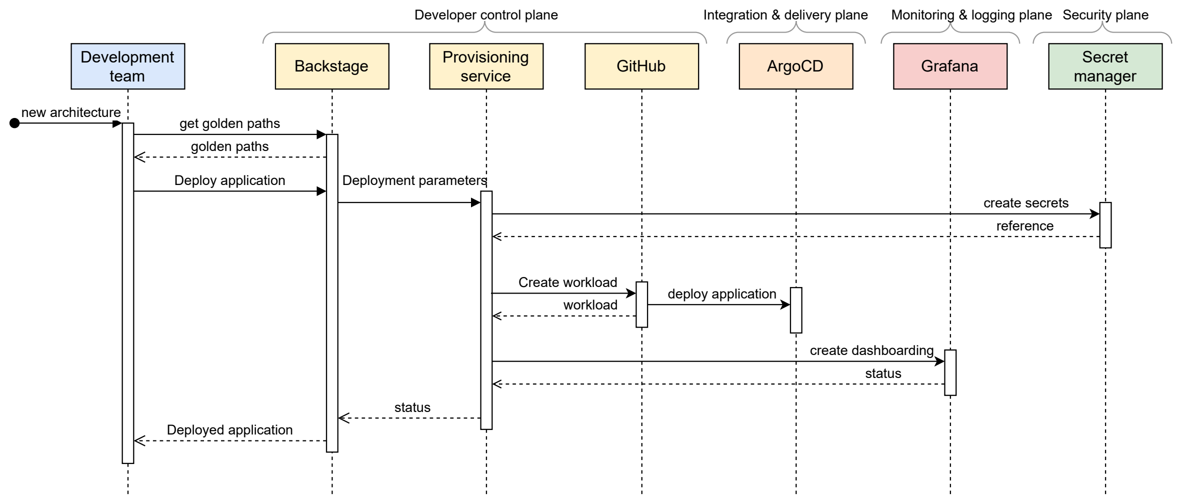


Figure 4.2: Golden path: provision application

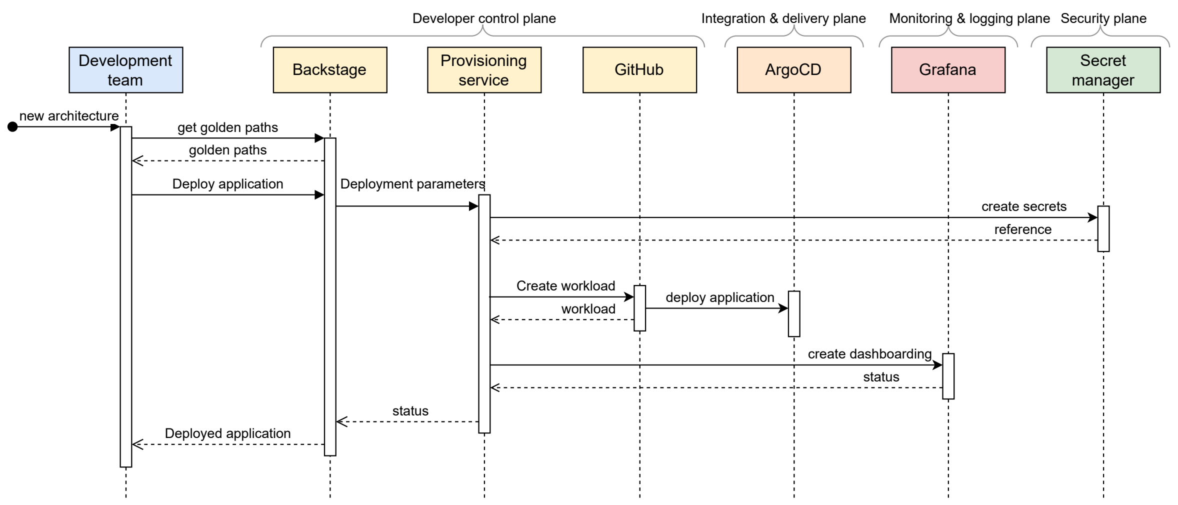


Figure 4.3: Golden path: provision application deployment

Engineering & technology viewpoint

Next to the organizational changes and operations that must be introduced, we also need to develop the technical implementation to support these changes. This will be done with the engineering viewpoint (explained in section 3.2.4) and the technology viewpoint (explained in section 3.2.5). Since many of the technical implementations are based on preferences and existing tools, the focus was not on changing the entire technology stack but on the main improvements that can be done to support the other viewpoints but also comply with the requirements and observations. Figure 4.4 provides a possible architecture, the technology stack beforehand can be found in appendix B.

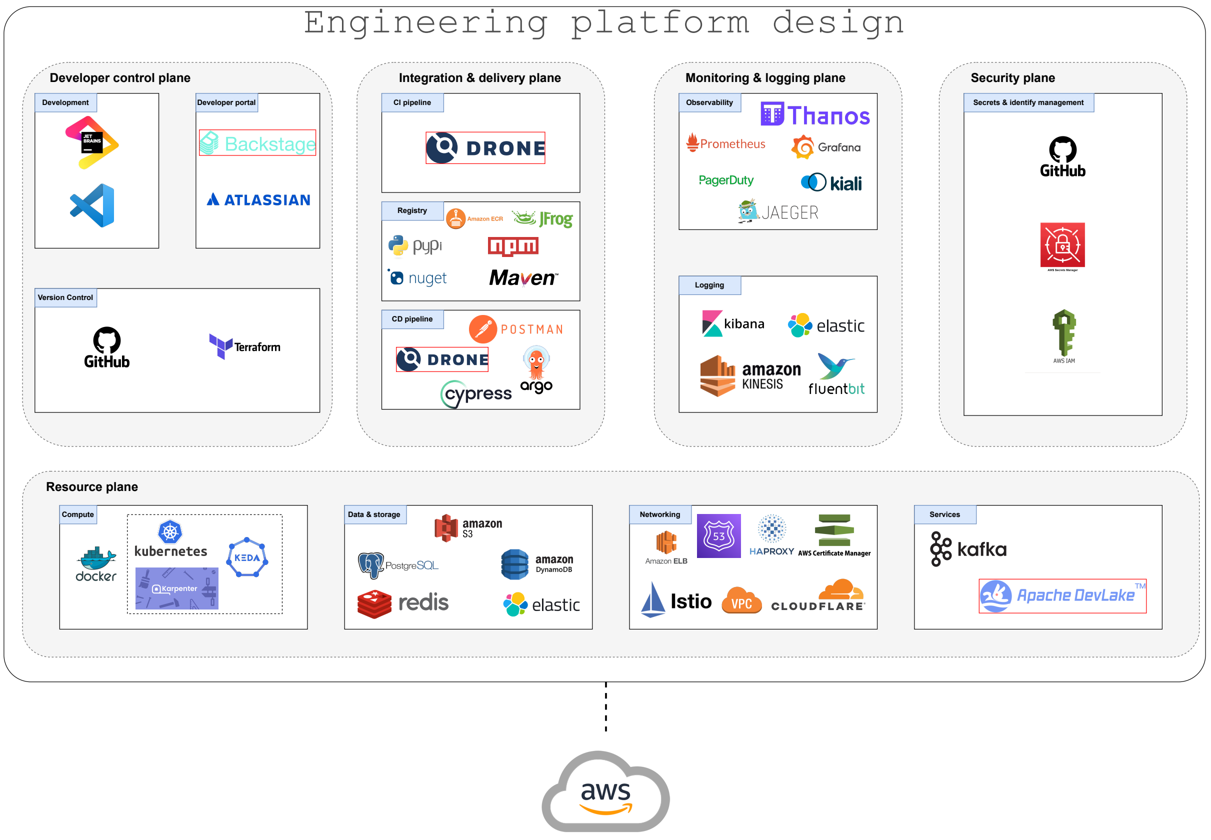


Figure 4.4: Engineering platform technical implementation

In this architecture, you can see three main changes done, which focus on achieving better integration of platform engineering and complying with the requirements and observations. Moreover, this technical implementation will support the creation of platform engineering according to the enterprise, information, and computational viewpoints.

(1) Backstage is the tool that all the development teams will use to communicate with the platform. Backstage could have multiple functionalities, giving the platform versioning and documentation. But also offer golden paths and service discoverability. Therefore the part that will play the most crucial role in the transition to a successful platform engineering strategy is Backstage.

(2) DroneCI can be a potential new tool to help give developers more control of their pipelines. Moreover, this way, the golden paths for the provisioning of specific tools can be easily customized. Each language could have its pipeline configuration. Moreover, this way, the organization can give the CI fewer privileges and be more sandboxed. However, the choice of which tool this should be can be different based on preferences or other factors.

(3) Apache DevLake is a new service that can be introduced due to the lack of metrics about performance within the organization. By looking into the performance metrics, we found that getting those is very difficult. Therefore, apache Devlake can be used to give more insight into the DORA metrics of the organization [35].

4.1.4 Summary

From our analysis, it's evident that the organization exhibits certain platform engineering attributes. However, this doesn't necessarily indicate a seamless integration of platform engineering practices. Notably, there's an apparent communication gap between the platform and development teams. This is exacerbated by the absence of a developer portal, making the support process from the platform team to the development teams more difficult and complex. The organization lacks standardized practices such as golden paths, platform versioning, and formal channels for feature requests, which play a vital role in platform engineering and the PE-RM. Additionally, critical performance metrics remain hidden, making it difficult to validate the current productivity. There's an evident overreliance on tools, as each team can independently create their resources. This freedom, while possibly beneficial in some contexts, has led to discrepancies in code quality and deployment practices, decreasing standardization and amplifying communication challenges.

In designing the platform engineering approach, our focus was not only on the technical aspects but more holistically on understanding platform engineering from the organizational point of view. By categorizing these challenges into the different viewpoints, we were able to design actionable recommendations. On an organizational level, the establishment of development guilds and clear role definitions can make a significant difference. From the information and computational perspectives, the introduction of golden paths and platform versioning will be beneficial. Hence, implementing a developer portal, specifically "Backstage", stands to be a transformative step for the organization. Not only would it provide platform versioning and golden paths, but it would also streamline application and developer onboarding by offering comprehensive information and support. Further, the incorporation of tools like DroneCI will empower developers with more pipeline control. Simultaneously, utilizing Apache DevLake can ensure that crucial metrics are easily accessible.

We leveraged the Platform Engineering Reference Model (PE-RM) to architect this platform engineering design, facilitating a customized implementation. It underscored the requisite organizational and technical shifts to fully embrace the platform engineering paradigm. By integrating new roles, information constructs, and technologies, these viewpoints collectively form a cohesive and interrelated structure.

4.2 Technical implementation

Now that we have given a guideline on how the Platform Engineering Reference Model can be used to design a tailor-made platform engineering design for an organization, we will focus on the technical implementation. Since implementing an engineering platform based on the design is too big for this short time, we have focussed on a small implementation with open-source tooling to show how to implement a primary engineering platform. This section will outline the different requirements used to build this solution, the architecture and components, and the unique functionalities and added value.

4.2.1 Requirements

This section will explain the requirements used to implement the engineering platform. The requirements are based on the observations in the case study and the context of this research. These observations are translated into users' stories and thereafter into functional and technical requirements.

User stories

Since we are building an engineering platform that should provide value to development teams, we need to identify what features should be included from their point of view (in order to add value). This can be achieved by identifying user stories, which we will list in this section.

- As a developer, I want the platform to generate my applications so I can focus on creating business value with the correct configuration and best practices.
- As a developer, I want the platform to generate the deployment configurations so I can deploy my application to a development environment.
- As a developer, I want the platform to generate the deployment configurations so I can deploy my application to a production environment.
- As a developer, I want to get the current version of the platform so I get insight into the changes and the documentation.
- As a developer, I want to see logs of my application by default so I can focus on creating the application and retrieving logs.
- As a developer, I want to have default dashboarding so I can get metrics of my applications without creating dashboarding myself.
- As a developer, I want to expose my application to the outside world without manually adding gateways so my application is reachable.
- As a developer, I want to be able to inject secrets into my application by default so I can store and inject secrets securely.
- As a developer, I want to have an overview of all the applications within the organization so I can gather information about the architecture.
- As a developer, I want the platform to scan my docker images for security issues, so I get security checks by default.
- As a platform engineer, I want to have a separation in development and production, so development teams can deploy in both development and production independently.

Based on the user stories, we collected a list of requirements to grasp better what our proposed system should be able to do. We have split these requirements into functional and non-functional requirements.

Functional requirements

Here we list the functional requirements that we want our framework to fulfill. These requirements describe the desired behavior of our system and are transformed to be more understandable in the context of platform engineering.

- The platform must have a scaffolding functionality to provision two types of applications: React and NestJS.
- The platform must configure the docker registry and pipeline for building and pushing docker images when provisioning an application
- The platform must have a scaffolding functionality to create a workload for an application's production and development environment.
- The platform must show the platform version, including the change logs, to the developers.
- The platform must offer logging and observability by default.
- The platform must offer a way to expose applications to the outside world.
- The platform must include secure ways to store and injects secrets.
- The platform must have discovery functionality to get an overview of all the applications.
- The platform must offer image scan functionality to validate the security of docker images.
- The platform must separate a development and production environment for deployed applications.
- The platform should allow developers to change configurations manually.

Non-functional requirements

Aside from the functional requirements, we also designed some non-functional requirements for our system. These requirements focus on the technical aspects of the platform.

- The graphical user interface should focus on the functionality above the design.
- The platform should be protected with authentication.
- The platform should be accessible by a DNS.
- The platform should only use open-source tooling.
- The platform must use open-source tooling to increase transparency.
- The platform must have a web interface that developers can use.

4.2.2 Architecture

We have constructed an architectural design for our engineering platform, as depicted in figure 4.5. This design, founded on the engineering and technology viewpoints of the Platform Engineering Reference Model (chapter 3), showcases the platform's components, their purposes, and design decisions.

Our prototype's goal was to showcase essential platform engineering functionalities, not tool selection. We utilized Otomi, an open-source Platform as a Service system, to fulfill this aim (detailed in section 2.1.2). This platform integrated various tools, which we configured to meet requirements, while customizations enabled new functionalities, including the integration of additional tools such as Backstage (section 2.1.2). This section will detail the main design and implementation decisions.

We chose Otomi and Backstage for their flexibility in creating tailor-made solutions, a cornerstone of platform engineering. This choice bypassed the constraints of other Platform as a Service (PaaS) solutions like Azure DevOps and Heroku.

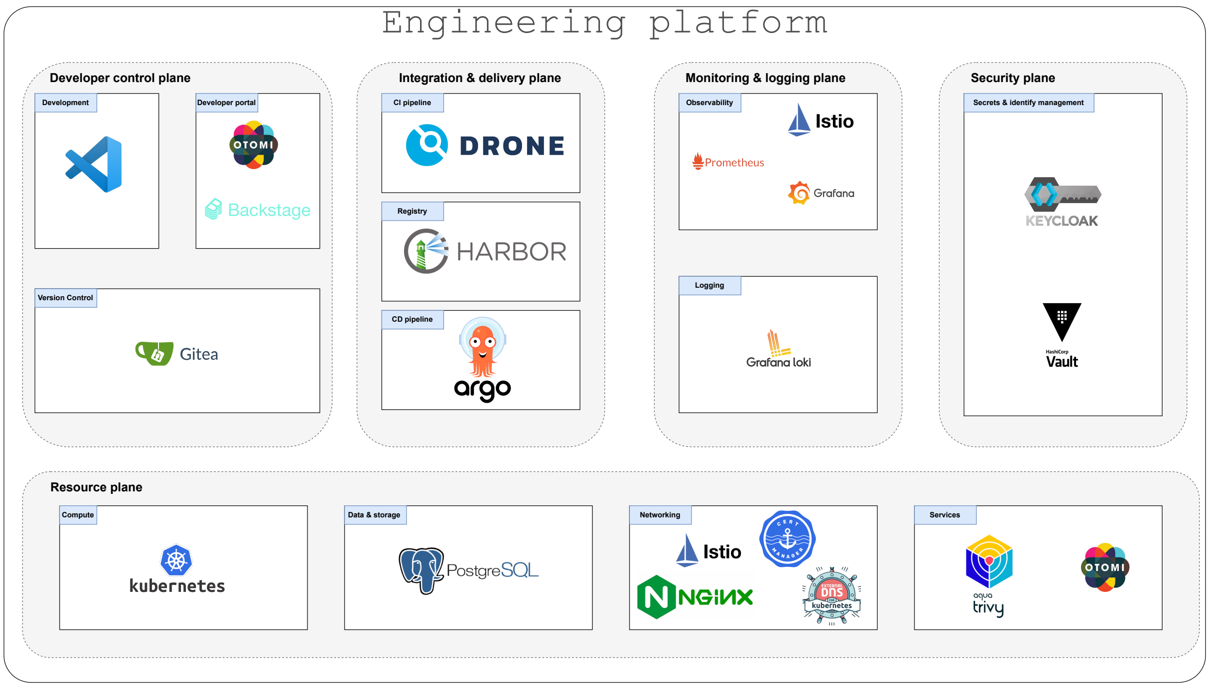


Figure 4.5: Implementation: engineering platform infrastructure

Developer control plane

In creating the developer control plane, we utilized Otomi as a central point for platform setup. Primarily, this interface is navigated by the platform team. In contrast, Backstage serves as the developer portal, providing facilities for provisioning and application discovery and serving as a repository for platform versioning and documentation. Several golden paths, have been established within Backstage. These golden paths, which are outlined in more detail in Appendix C, include the following:

- Create React application.

- Create NestJS application.
- Create development workload for application.
- Create production workload for application.

To execute these golden paths, Backstage interfaces with Gitea via an API, managing a variety of configurations. This includes creating repositories with templating, configuring pipelines for DroneCI, and establishing ArgoCD workload files. Depending on the specific golden path, it can modify a file to add distinct configurations. We've embedded custom functions within Backstage, which trigger HTTP requests to Gitea, updating repositories with code generated in Backstage. Additionally, developers leverage Otomi for tasks like creating DNS records or injecting secrets. For version control, we elected to use Gitea, largely due to its integration with the Otomi PaaS solution. We established a dedicated repository, *argo-workload*, which maintains the record of applications as applicationSets deployed through ArgoCD on Kubernetes. This repository is interconnected with ArgoCD. To ensure the smooth operation of Backstage in conjunction with Gitea, it was necessary to create custom functionalities. These are documented extensively in Appendix C.4.

Integration & delivery plane

The tools incorporated within the integration & delivery plane were part of the Otomi installation, although with modifications to fit our framework. We streamlined DroneCI by integrating Harbor secrets into the organization, automatically injecting necessary secrets to push Docker images to Harbor, thereby simplifying configuration for new applications. The DroneCI configuration pipelines were auto-provisioned via a custom function in Backstage (see Appendix C.3).

For ArgoCD, a repository, *argo-workload*, was created in Gitea. This repository houses various YAML files, including application sets. Utilizing golden paths for file creation enhances error minimization and boosts developers' productivity. Both development and production versions of the application are contained within each application set, and they run on a single Kubernetes cluster. The application set refers to specific files located in a named folder tagged as either "dev" or "prod". These folders contain three files: *development.yaml*, *service.yaml*, and *servicemonitor.yaml* (details are in Appendix A). Each application maintains a unique name and namespace for effective monitoring and logging. ArgoCD updates Kubernetes deployments automatically following repository updates.

Configurations of DroneCI pipelines, found in the repository, are provisioned by golden paths, which can append new deployment pipelines based on the path selected. Similarly, the ArgoCD repository is auto-configured using golden paths, ensuring accurate namespace and port specifications in Kubernetes. Manual adjustments are possible but typically unnecessary.

Monitoring & logging plane

For observability, we use a combination of Istio, Grafana, and Prometheus to gather metrics of applications. By default, Prometheus is connected to Istio in a service mesh to gather metrics about a container and store it. Grafana is used to display those logs in dashboards. Since we deploy applications with a unique namespace, we can easily query logs based on the namespace. This makes it easier for developers to get the logs and metrics for their applications without having to worry about the configurations.

In terms of monitoring, we provide default dashboards for developers, empowering them with direct access to metrics. For the prototype, we've expanded this offering by devising a custom dashboard applicable for each application. These customized dashboards reveal unique metrics specifically exposed by their corresponding applications. By implementing applications via the 'golden paths', developers automatically receive this expanded functionality. This enhanced dashboard can then be used to identify HTTP errors originating from their applications, facilitating prompt issue resolution. By standardizing this approach, all applications will emit identical metrics, consequently enabling universal dashboards. This provides a coherent view across all applications and fosters an environment of standardization, enhancing the overall efficiency of our monitoring and observability architecture.

Security plane

To secure the platform with role-based access control (RBAC) and identity management, we've incorporated several tools to enhance security. Initially, our implementation only established a default team endowed with full rights; however, this arrangement allows for future modifications. To engage with the platform, users are required to sign in through an account managed by Keycloak, a step that grants access to all tools deployed within the platform. This system ensures that we maintain control over user management and settings, boosting overall security. For application-related matters, we rely on HashiCorp Vault for secret injection into pods. This is particularly useful when deploying in a Kubernetes environment, as it ensures that secrets are readily accessible. This strategy promotes secure secret storage, eliminating the risk associated with storing sensitive data in plain text. Furthermore, we've established a connection between Vault and ArgoCD to facilitate smooth integration, minimizing the need for manual intervention. An illustrative example of this can be found in the *argo-workload* repository, where Backstage retrieves its secrets from HashiCorp Vault. This particular arrangement underscores our commitment to creating a secure and user-friendly platform.

Resource plane

The compute components of this platform, which fundamentally operate on Kubernetes, are central to our platform, largely due to our decision to utilize Otomi. This strategy allows for significant vendor lock-in reduction, even enabling custom hosting. For this particular implementation, we've chosen Digital Ocean as the hosting provider, favoring its easy configuration and provision of large clusters, which contributes to expedited development time. Numerous pods and deployments are initialized within the Kubernetes environment to ensure the platform's functionality.

While the emphasis of this implementation doesn't fall on data and storage—resulting in their exclusion—there exist many possibilities for database integration. Given that the platform resides on a cloud provider, we could introduce Terraform into Gita to provision databases or create a new 'golden path' that empowers developers to provision Postgres databases within Kubernetes.

As for networking, we employ a suite of tools that facilitate network connectivity, ingress, and load balancing for all tools and applications housed within the platform. Given that both custom applications (developer-built) and platform tooling operate on Kubernetes, we've enabled various ingress controllers to manage traffic flow. We utilize Nginx as an ingress controller, while Istio takes charge of networking and request management. Additionally, we employ external DNS to assign a default DNS to our application, ensuring that applications are accessible via specific DNS. Most of which is set up by Otomi.

Though services were not the primary focus of this implementation, we did integrate Trivy to illustrate one potential service the platform could offer. Trivy conducts vulnerability scans on Docker containers, providing developers with valuable insights and advanced security tooling. From this implementation, we recognized in short order that we also employ Otomi as a service, as it assists in exposing applications and injecting secrets into them.

4.2.3 Functionalities

In this section, we will work on the functionalities of the platform to understand the platform implementation and added value of the platform. In order to show you what functionalities are unique to this implementation and could help the adoption of platform engineering, we have created figure 4.6. In this figure, you can see the main events of an application lifecycle, which is a simplified version of the one proposed in section 3.2.1. In the case of the implementation, we have focussed on the **(1)** application domain and **(2)** observability and logging.

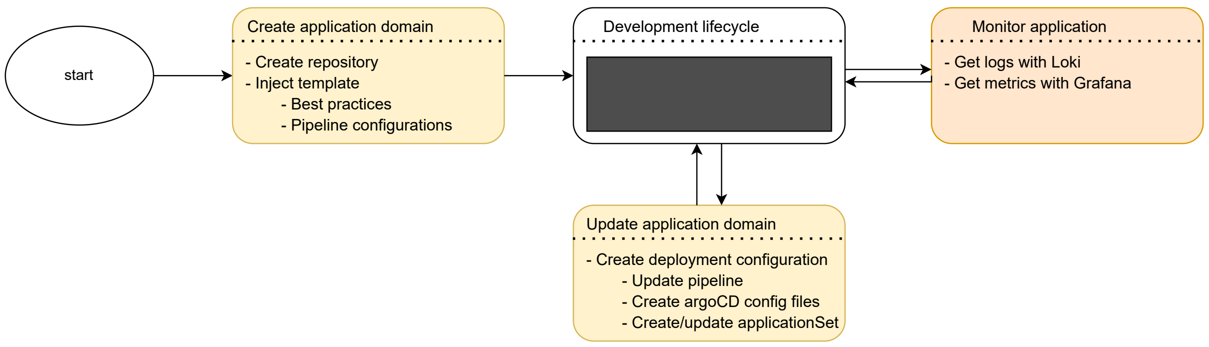

start
Create application domain
- Create repository
- Inject template
- Best practices
- Pipeline configurations
Development lifecycle
Monitor application
- Get logs with Loki
- Get metrics with Grafana
Update application domain
- Create deployment configuration
- Update pipeline
- Create argoCD config files
- Create/update applicationSet

Figure 4.6: Application lifecycle: different events

Automated provisioning of services

The primary contribution of implementing the engineering platform is the development of golden paths, which streamline the configuration processes for applications, pipelines, and deployments. This approach allows developers to prioritize the construction of an application that creates business value instead of being sidetracked by the complexities of configuration. The ability to construct these golden paths is influenced by the organization's structure and the technologies utilized. Development guilds and the platform team can curate golden paths tailored to specific actions. In the present implementation, we have engineered four distinct golden paths: *Create Nest.js application*, *Create React application*, *Create development workload for application*, and *Create production workload for application* (refer to figure C.2 in the appendix). Each golden path serves unique objectives and functions within the implementation process. All development teams can access these golden paths through the developer portal.

For instance, if there is a need to create a Nest.js application, a dedicated golden path can guide this process. The sole requirement is the application's name. The golden path then scaffolds the application, integrating universally applicable boilerplates and metrics configurations that facilitate default dashboarding (further elaborated in section 4.2.3). This process also incorporates the creation of a Gitea repository, including DroneCI configurations and a Dockerfile for building and pushing a Docker image to Harbor. Upon completion, the platform provides the developers with links to the repository, pipeline, and catalog. These links allow the developers to validate the creation process and commence with the application's development. Figure 4.7 presents a sequence diagram that outlines the steps to provision a new application. Additional resources, including links to template and example repositories, can be found in appendix A. Appendix C provides a detailed description of the implementation process.

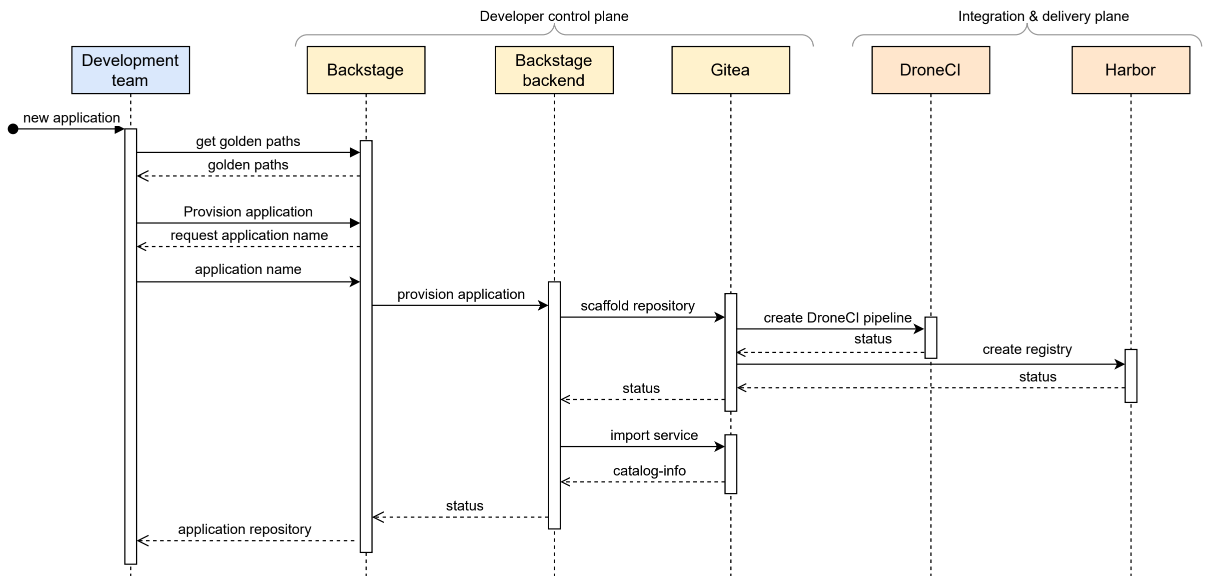


Figure 4.7: Platform implementation: provision application

We assume that the continuous development of an application can be considered a black box, given that the specific development approach can depend significantly on the organization and development team. However, once the development team is ready to deploy the application, the platform aids the deployment process by providing a separate golden path to update the application domain with workload files (refer to figure C.5 in the appendix). In our implementation, the applications are deployed using ArgoCD on Kubernetes. This deployment method, however, may vary across different organizations. Typically, organizations maintain a separation between development and production environments. Correspondingly, we have devised two distinct golden paths to cater to these environments: development workload and production workload. Once the development team initiates the deployment, they can employ the respective golden path to execute five distinct steps.

1. The first step is to update the DroneCI pipeline in the application repository to ensure that the following change of the application will be automatically deployed.
2. The second step is creating a Kubernetes deployment file that will be configured with default settings and the latest docker tag of the application.

3. In step three, the Kubernetes service configuration file is created to ensure that the deployment is created with the correct name and namespace.
4. The fourth step is creating a service monitor that will ensure that the metrics will be exposed to Prometheus that will be used for observability purposes.
5. In the last step, the golden path will create or update the applicationSet in the argoCD workload repository. If the development team creates a development workload, this applicaionSet will be created, and when it is a production workload, production will be added to the applicationSet.

Once the applicationSet is updated, argoCD will sync the repository and deploys it on Kubernetes. Using this golden path, the application will be deployed on the correct cluster with the correct settings and namespace. In appendix C, this golden path is explained in more detail.

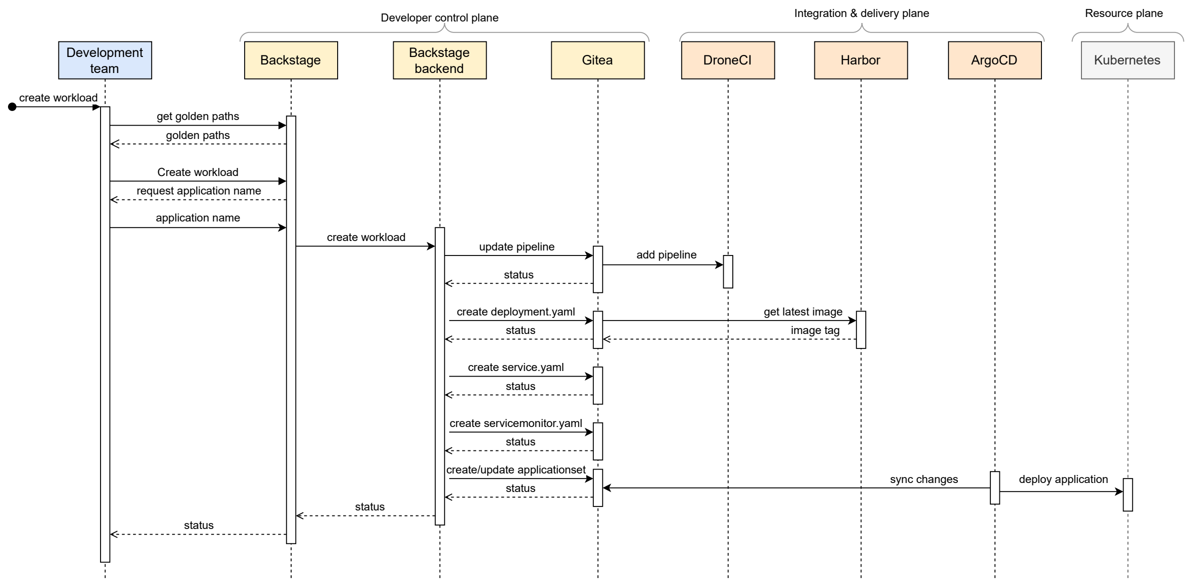


Figure 4.8: Platform implementation: provision application deployment

Employing these golden paths to provision the complete application domain enables the platform to manage the entire lifecycle, thus reducing the cognitive load on developers. This provisioning method simplifies the integration of applications with various tools, such as gateways, and secret management, among others. As a result, transitioning the responsibility of configuration from developers to the platform boosts the productivity of development teams. Additionally, it promotes a higher degree of standardization and centralization, leading to more efficient and consistent operations.

Default observability and logging

Another distinctive attribute of the platform that enhances the effectiveness of development teams is the incorporation of default observability and logging for applications. Utilizing golden paths, maintained by development guilds and the platform team, for application provisioning allows the platform to provide default dashboards and logging for these applications. This becomes particularly critical for production systems. As applications are deployed in a standardized manner onto the platform, which includes the scaffolding of metrics within the namespace that aligns logically with the platform, development teams can utilize integrated dashboards across all applications. This obviates the need for development teams to configure dashboards or logging settings individually, allowing them instead to leverage the predefined tools readily available. Figure 4.9 showcases a sequence diagram illustrating the process of obtaining logs and metrics for a specific application. Please note while the connection between Loki/Prometheus and Kubernetes may differ in practice, this representation simplifies the data source identification.

The platform's capacity to offer default dashboarding extends to creating default alerts, which can be universally applicable to all applications constructed by the golden paths initiated by development teams. This heightens the level of standardization across varied teams, simplifying the detection of production issues that impact multiple applications. Additionally, this feature eliminates the need for development teams to worry about observability as it is inherently provided. Appendix C.5 elaborates further on the platform's default observability and logging functionalities.

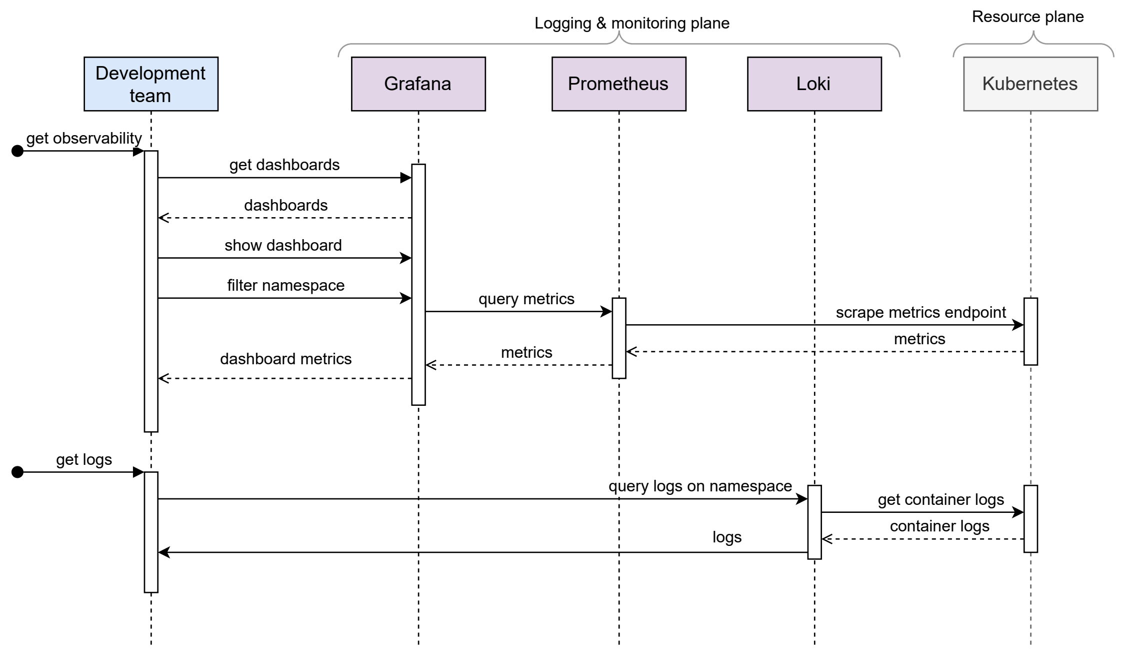


Figure 4.9: Platform implementation: default observability and logging

4.2.4 Summary

Through this technical implementation, we have demonstrated how an engineering platform can promote the adoption of platform engineering within an organization—showcasing the added value of platform engineering by creating a centralized and standardized platform with golden paths, documentation and integrated tooling. This methodology can enhance the performance and productivity of developers while fostering standardization and centralization within the organization, which consequently helps decrease maintenance overhead. When compared with Platform as a Service (PaaS) solutions like Azure DevOps, a significant advantage of our approach is its adaptability across an entire company. The high degree of customizability ensures organizations can integrate the platform into their existing technology stacks, eliminating the need for a complete migration to a new platform. Moreover, it will eliminate vendor lockin since it can run on many different providers.

The implementation process for any organization requires a dedicated platform team that collaboratively works with development guilds to construct the golden paths. Overall, this approach streamlines the application lifecycle by minimizing the configuration tasks assigned to developers. By centralizing and standardizing all tools and provisioning strategies, the organization could also enhance the migration process, as the platform offers golden paths specifically designed to facilitate this. In addition, it increases the control of the organization on the tools used, tackling the challenge of the proliferation of tools. Moreover, by using golden paths and a centralized platform, cloud spending can be monitored better to decrease costs where possible.

Although the current technical implementation serves as a small prototype and lacks certain features that could be beneficial for the organization, it serves as a powerful example. Further enhancements could include more thorough validation during the creation of a new application, along with additional parameters to enhance provisioning visibility, such as team tags. Additionally, the platform could enforce application provisioning through golden paths, increasing the organization's control and encouraging teams to utilize specific tools. However, given the illustrative nature of this prototype, these functionalities have not yet been implemented.

4.3 Experiments

In this section, we will perform a number of experiments to demonstrate the value of the platform implementation. We try to keep a uniform layout in the presented information. In the experimental setup subsections, we describe 1) the environment of the experiment, 2) possible assumptions we have in regard to the results, and 3) the goal of the experiment. Finally, we will show and discuss the results.

4.3.1 Productivity evaluation

In this experiment, we want to show the advantages of our platform implementation, focussing on the application lifecycle (for more information, see section 3.2.1 of the PE-RM). This can be validated by measuring the productivity metrics of developers, especially the onboarding time of developers and applications. In the results, we will refer to our case study performance metrics for comparison. We will gather this information in the same experiment as the usability study.

Experimental setup

The central hypothesis guiding our experiment declares that the utilization of the platform will enable participants to accelerate the deployment of their applications to production, even when operating with a minimal information base. We speculate that the developer portals and integrated tools present within the platform will facilitate a more efficient application onboarding process. Considering the participants' unfamiliarity with the platform, we also anticipate an accelerated self-onboarding process due to the ability to deploy to production within the experimental setup.

To ensure dependable test outcomes, we will divide the participants into two groups: one provided with a tutorial and the other without. This arrangement allows us to draw comparisons between onboarding times, ascertaining the impact of platform documentation on the process. Participants' current roles and experience levels will also be factored in to measure the influence of prior knowledge on their speed of adaptation. In describing the task, we will outline fundamental information concerning what the task involves and the necessary prerequisites to begin. The group granted a tutorial will receive a detailed, sequential breakdown of their designated tasks and operational principles. Observation of task execution will offer insights into whether participants can accomplish their assignments within an acceptable time frame.

Prior to initiating their tasks, we will present participants with a use case and ask them to estimate the duration it would take to deploy an application to production without the aid of a platform. They will also be prompted to identify what they perceive to be the most challenging aspect of application onboarding. Upon task completion, we will have sufficient data to delve into two key performance metrics: developer onboarding time and application onboarding time. Averaging these results and cross-referencing them with case studies will enable us to determine whether our platform significantly influences productivity, particularly in application lifecycle management and development team performance.

Results

A total of ten people participated in the experiment. Despite a limited sample size, the participant's different expertise allows us to assess whether the platform implementation is versatile enough to serve a variety of developers or if the experience differs based on their specializations. As evident in Figure 4.10, the most frequently represented role among the participants was that of a software developer, but the pool also included a number of junior and senior developers.

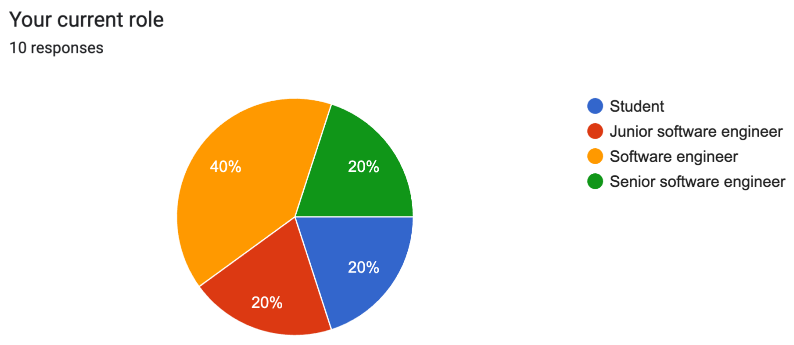


Figure 4.10: Participant roles

Our first objective was to understand the participants' initial expectations by presenting them with a use case to evaluate their experience with DevOps and onboarding an application. We offered a scenario in which they had to estimate the time required to bring an application to production under certain constraints (refer to Appendix D). As shown in Figure 4.11, the average estimation was relatively high, suggesting that the participants believed that deploying an application to production is a time-consuming process. We also asked them to identify what they would find most challenging when bringing an application to production and documented the most common responses.

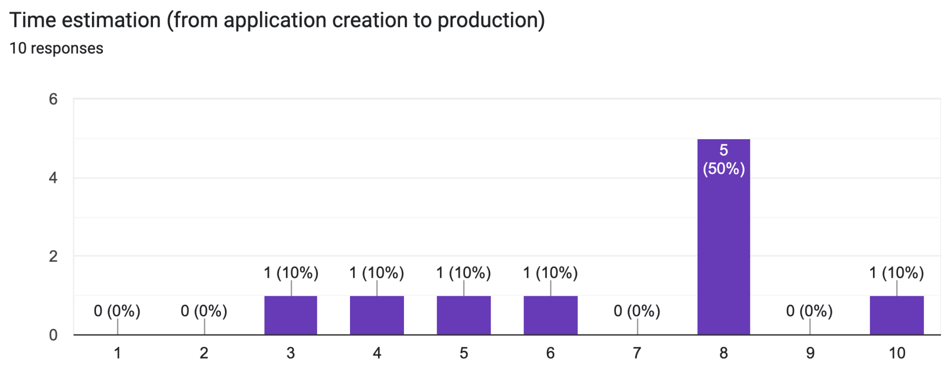


Figure 4.11: Use case estimation

```
What do you think would be the most difficult part?
     1) Setting up the different deployment settings and different environments.
     2) To have the dashboards of your application for monitoring purposes.
     3) Configuring all the tools so that they can communicate without error with
          eachother.
```

Upon task completion, we requested the participants to reassess their initial response to the first question to confirm if their perspective on the time taken to get an application to production had changed. While there were variations in the answers, the majority of the participants now believed that it would take longer than they had initially assumed (see Figure 4.12). This suggests that many participants were unaware of the complexities involved in deploying an application to production. Given the observations around the challenging parts and lack of experience, it appears that platform engineering can play a pivotal role in eliminating the uncertainties and gaps in knowledge among software developers, thereby allowing them to focus more on development rather than on configuration, infrastructure, and cloud aspects.

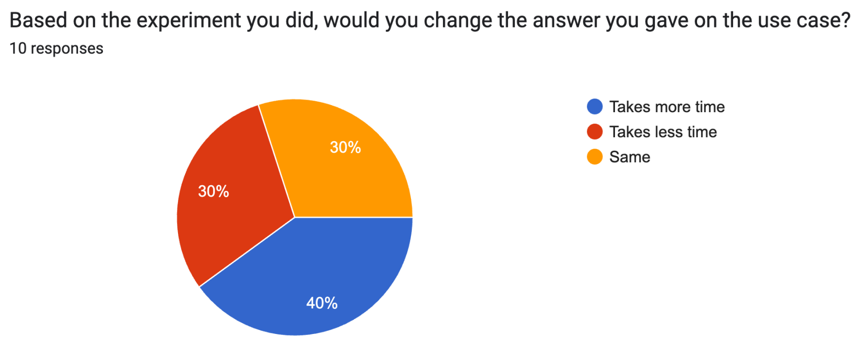
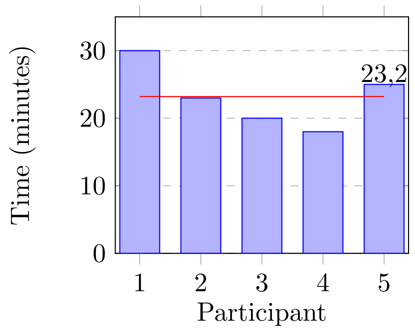
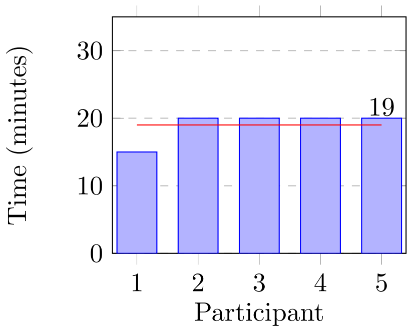


Figure 4.12: Use case estimation change

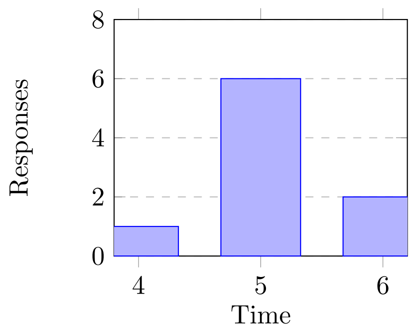
We divided the participants into two groups for the next part of our experiment: one group was provided with no documentation or tutorial on how to use the platform, while the other group was given both. This approach helped us determine the intuitive nature and ease of use of the platform. Depending on their group assignment, the participants were given different tasks (outlined in Appendix D). The group provided with a tutorial and documentation was tasked slightly differently, making their task a bit more challenging (see Appendix D). Although this makes a direct comparison of the results somewhat difficult, it does provide more insightful observations, especially given that the "no-tutorial" group needed more time to complete their task (see Figure 4.13).



(a) No tutorial



(b) Tutorial



(c) Read metrics and logs

Figure 4.13: Time to resolve experiment task

Based on these observations, we can draw the following conclusions:

- The significant time discrepancy between completing the task using the platform and the time they anticipated it would take to manually configure it is striking, averaging a 6-hour difference. This showcases the immense potential of platform engineering to enhance the application lifecycle and increase onboarding time.
- The evident difference in performance between the tutorial and no-tutorial groups underscores the value of comprehensive platform documentation, which can be instrumental in guiding developers through specific tasks, especially in the absence of assistance.
- Automatic logging and metrics can be significant time savers. Some participants expressed that configuring logging and metrics might be challenging. However, the experiment revealed that with this implementation, metrics and logs could be gathered easily.

Despite the case organization incorporating some elements of platform engineering, such as provisioning a GitHub repository, a Docker registry, and a pipeline (refer to Section 4.1.1), a significant gap exists in application onboarding time when compared to these results. The primary reason for this disparity is the efficient provisioning of the applications themselves and the deployment workload files, leading to considerable time savings.

4.3.2 Usability study

We want to conduct a usability study in the form of a survey and contextual analysis among software developers from inside and outside the organization to get feedback on the usability and effectiveness of the platform.

Experimental setup

This experiment is designed to investigate developers' interaction with our platform. By configuring a setup where developers are tasked with onboarding a new application, we can assess the platform's usability and its potential to enhance productivity and efficiency. By providing developers with a minimal information set required to deploy an application to production, we aim to test the platform's intuitiveness and ease of use while simultaneously identifying potential feature gaps.

Upon task completion, participants will be invited to fill out a survey hosted via Google Forms[15]. This process will allow us to compare our observations with those of the participants. The survey encompasses a blend of open and closed-format questions. The closed questions present either multiple-choice options for participants to select or a linear scale for participants to provide a feedback rating on a scale from 1 to 5, and in certain cases, from 1 to 10.

The survey initiates with a question concerning the participant's background to contextualize their expertise and experience. Subsequently, participants are asked to rate the platform's ease of use on a scale from 1 to 5. The ensuing question seeks to understand the proportion of manual work relative to work completed by the platform. Our fourth query is centered around the platform's usability and learning curve: is it simple to navigate or potentially complex? The fifth question invites participants to highlight their favorite features of the platform. Furthermore, we are interested in uncovering potential platform enhancements proposed by the participants. If applicable, we inquire how the platform compares to their existing workflows and the most significant differences noted. Lastly, we request general feedback about the platform and suggestions for further refinements.

Results

A total of ten people participated in the experiment. Despite a limited sample size, the participant demographic allows us to assess whether the platform implementation is versatile enough to serve a variety of developers or if the experience differs based on their specializations. As evident in Figure E.1, the most frequently represented role among the participants was that of a software developer, but the pool also included a number of junior and senior developers.

Upon task completion (refer to Appendix E for task descriptions), participants were posed several questions concerning to the platform's usability. The initial question targeted the platform's ease of use. From the gathered responses, a consensus emerged indicating the platform was easy to navigate (see Figure 4.14). However, we noted during the experiments that the multitude of interfaces could be perplexing for some, slightly reducing the overall ease of use. This observation will be taken into account in future work. The subsequent question addressed the volume of work handled by the platform in relation to task completion. This question is aimed to determine whether the platform strengthens productivity or if substantial manual work remains a requirement. Based on the responses, it was apparent that the platform performed the majority of the work (see Figure 4.15). This outcome supports our hypothesis that developers can dedicate their focus to application development rather than being bogged down by configurations.

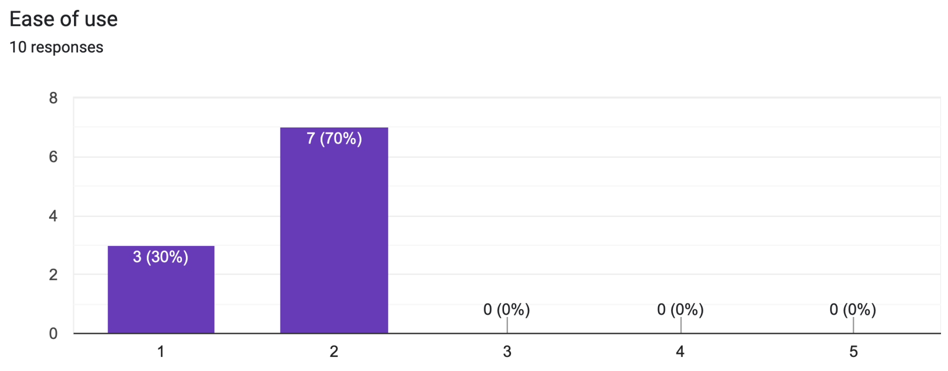


Figure 4.14: Ease of use

[15]https://www.google.com/intl/en/forms/about/

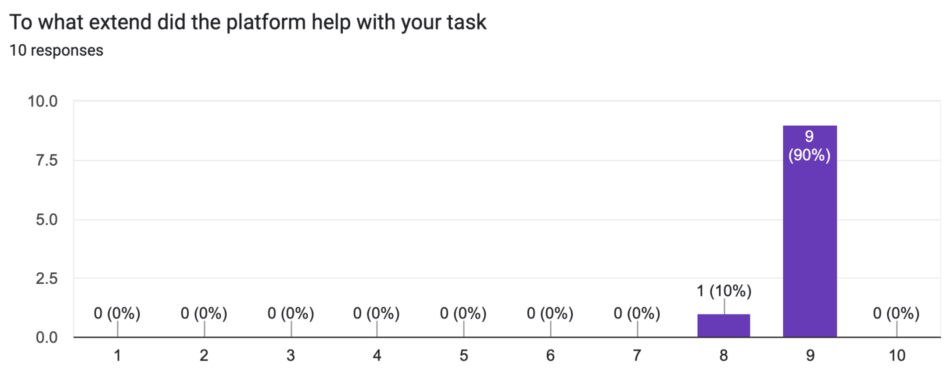


Figure 4.15: Amount of support from the platform

Regarding the platform's learning curve, a noticeable trend emerged from the responses. With the aid of documentation, participants found it easier to comprehend what tasks could be performed and how to carry them out. Without documentation, understanding the necessary steps become more challenging. This trend confirmed our observations, where guidance was crucial to initiate participant activity. Frequent switching between tools also contributed to the confusion, underscoring the necessity for the platform team to furnish development teams with robust support and thorough documentation.

```
Learning curve of the platform
    1) With the docs, it was very easy to use.
    2)Since I did this task without tutorial I needed some time to click through some of
       the tools. But after I learned which tool did what it was extremely easy to
       create, build, deploy and expose an application
    3) Pretty straightforward, although there are a lot of tools that are thrown at you
       in one go
    4) Each service inside the platform is straightforward and is very easy to use. But
       you have to switch between different service within the platform, and they all
       have different UI.
    5) It was very easy to use. Makes this task much quicker. Only thing I did not
       really like was all the switching between tools/tabs. But this is to be expected
        when trying to use lots of tools.
    6) The links between the different apps used were not always as logical or easy to
       understand. However, with a little bit of documentation and an overview of how
       the tools work together I think the relations between the different apps will be
        easier to understand and will improve the process for a first time user.
    7) Easy to use, with basic explanation about some of the tools I was unfamiliar with
    8) Rather easy. I wont be struggling on yamls configurations
    9) Yes everything was easy to use and documentation was very helpful. Only negative
       was that many applications/sites were used and it could be simpeler if it was
       all in a single flow.
    10) For someone without much knowledge of the tools that are used on the platform,
       using the set of tools can be a bit daunting. However, the actual process of
       using the tools is quite straightforward since not many manual tasks need be
       performed by the user. Exposing the site to the world was the most complex step.
```

When asked about their favorite platform features, participants' responses clustered around a few key themes. From a developer's perspective, they particularly appreciated having minimal configurations to set, thereby reducing their workload. Hence, the highly favored feature was the *golden paths*, a tool that manages all configurations with the least necessary input. The most valued features were collectively displayed in Figure 4.16 to provide an overview.

```
What improvements do you suggest for this platform?
    1) Create golden paths that will help developers to migrate applications. Not only
       the creation of new applications, but also migrations
    2) To have it in one place.
    3) Maybe there is a way to merge the content from otomi and backstage together. So
       all the information is in one place.
    4) Documentation.
    5) Good documentation about workflow and basic information of tools used
    6) hard to say at moment I used it the first time, for now looks good to me
```

```
     7) Try to include as much in a single flow.
     8) Finding a way to let the user interact with all the tools without having to
        switch between interfaces would improve the process. Something such as a wiki
        that would explain all steps in detail would make the adoption of the platform
        easier.
```

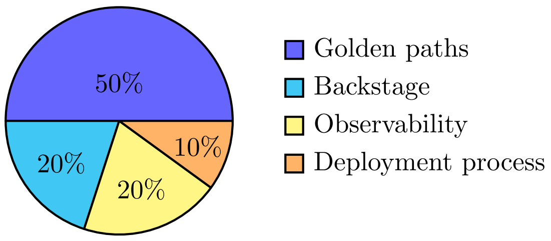


Figure 4.16: Which feature of the platform did you like the most?

Some suggested improvements align with topics previously discussed in this research, including the consolidation of all functionalities within a singular developer portal. However, due to constraints on time and resources, these features could not be realized during this experiment and will be prioritized in future work. Proposals such as 4) and 5) were indeed valuable, but these are already implemented. However, based on the version of the platform participants used, they didn't have the opportunity to utilize the documentation. Furthermore, all participants indicated they would support platform engineering within their organization, with their experiences detailed in Appendix E. In response to the query regarding whether the platform met their expectations, they provided the following feedback.

```
General feedback
     1) Yes
     2) I didnt have any expectation from the platform
     3) Yess, maybe a bit more than expected
     4) I expected the platform to bring ease of use so in that case it worked. However,
        it still requires the user to have an understanding of multiple tools. I think a
         tool like otomi already takes away a lot of the pains, but maybe having one
        tool in which you can setup an entire application without leaving the tool is,
        of course, most desirable. For example, if you look at other platforms like
        Vercel.
     5) I had little to no expectations about platform engineering
     6) sure. yaml-s to me is real place for a lot of mistakes. So this platform is very
        useful to avoid issues.
     7) yes, I did not know what to expect from platform engineering but was pleasantly
        surprised.
     8) It is not what I expected. My devops experience is not that deep, and imagining
        what to expect from a platform was therefore difficult for me. After using the
        platform, the benefits of it have become much more clear to me, and it has also
        given me a clearer view of what is meant by a platform.
```

4.3.3 Platform expert feedback

With this experiment, we want to get feedback on the platform engineering lifecycle aspect of the implementation. The focus of the other experiments was on the developer feedback on the application lifecycle but not on the technical implementation of the platform itself. In this experiment, we will give a demo to platform experts to get their feedback on the implementation. The participants were either architects, tech leads or platform engineers, which can all be related to the stakeholders introduced in the PE-RM.

Experimental setup

Our experiment will initiate with a comprehensive demonstration of our implemented design to subject matter experts within the field. This demonstration aims to provide context, clarify our design choices, and illustrate them. We plan to delve into a deeper technical discussion to enhance our understanding of the architecture and technical execution of our design. An interactive session will follow, granting experts an opportunity to pose questions for additional information or clarity.

Subsequent to this informational session, experts will be requested to complete a brief survey. The survey will consist of open-ended and multiple-choice questions designed to gather valuable feedback on the platform. The multiple-choice queries will assess the platform's practical application, whereas the open-ended questions are structured to produce suggestions for improvement and affirm areas of successful implementation. It will be highlighted that feedback should be contextualized within the modest size and scope of the platform.

The first question seeks information regarding the respondent's professional background. This approach enables us to verify the expertise of the participants and, thus, the validity of their feedback. The subsequent question assesses the overarching design of the platform, followed by questions relating to user experience, ease of use, and the platform's potential applicability within an organizational context. Respondents will be asked to provide their responses on a scale of 1 to 5. The latter section of the survey focuses on their perspectives on the most valuable features of the current implementation. Respondents will also be prompted to suggest new features they believe would enhance the platform and their reasons for the additions. In light of the platform's function as a prototype, we invite thoughts on how it could seamlessly integrate with other tools and systems. Lastly, we ask if they would endorse the platform as a model for others and how it would support technical implementation within their organizations. In this way, we aim to validate the various aspects of the platform, including technical execution, relevance, and practical application.

Results

In this experiment, we engaged with five experts related to platform engineering, all with backgrounds in DevOps or platform engineering. Using a combination of a demo and a discussion about the implementation, we posed a series of questions focused on the platform's applicability and integration potential. The initial question aimed to clarify their current roles to better comprehend their experiences and expertise, as presented in Figure 4.17.

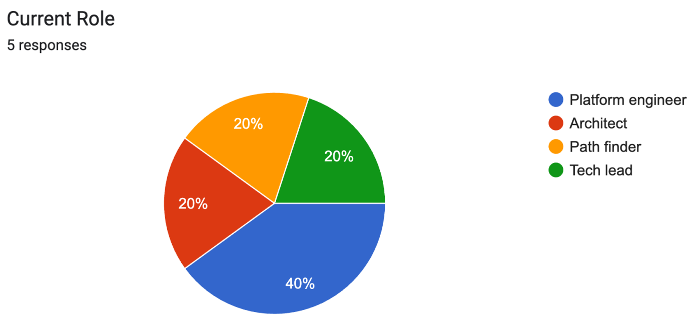


Figure 4.17: Current role of experts

The first pair of questions were closed-ended, targeting the platform's overall design and ease of use from the viewpoint of a platform engineer. The feedback collected indicated that both the design and the ease of use of the platform were well-received (refer to Figure 4.18 and Figure 4.19). However, considering the limited time and decisions made during platform implementation, certain design aspects may have drawbacks, particularly with the choice of Otomi. Nonetheless, in this project, the benefits outweigh the disadvantages.

Subsequent questions delved into the platform's applicability within an organization and its compatibility with different tools. This allowed us to assess the platform's potential usability in real-world scenarios and its value proposition to other organizations seeking to implement a similar platform. The feedback here was mixed; while two experts found it applicable and integrative, others suggested that it would require additional work to be fully applicable. Yet, given the bespoke nature of platform engineering, we can conclude that it could serve as a useful example, even if it is not directly transferable to all organizations.

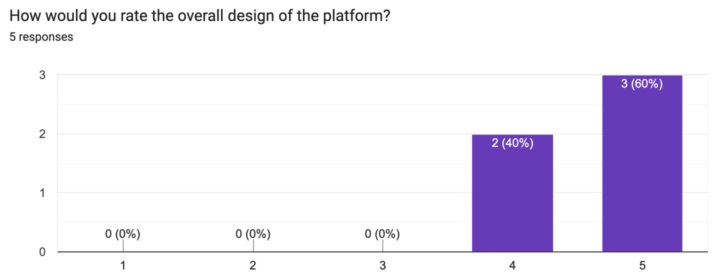


Figure 4.18: Overall design of the platform

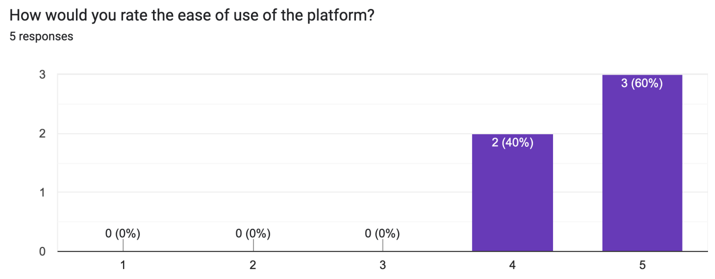


Figure 4.19: Ease of use of the platform

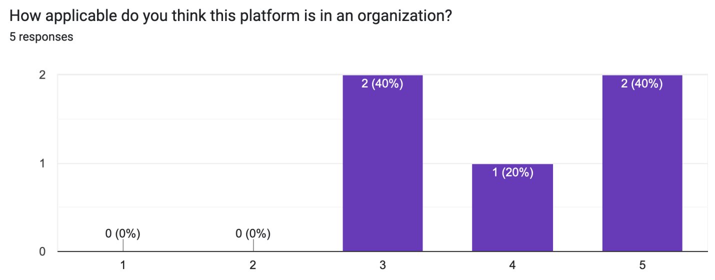


Figure 4.20: Applicability of the platform in an organization

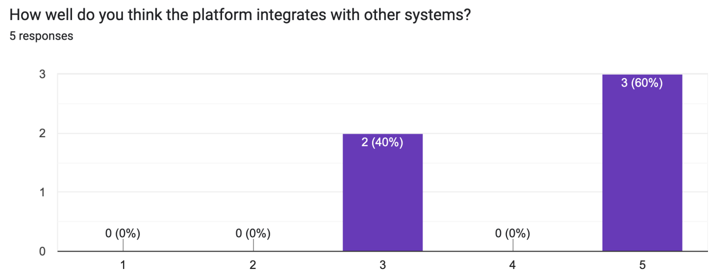


Figure 4.21: Platform integrations with other systems

Following the closed-ended questions, we asked the experts several open-ended questions to gain more nuanced feedback on the platform's implementation. These responses facilitated some valuable conclusions and observations. For the complete set of answers, refer to appendix F.

```
Which features did you find most valuable?
    1) Catalog and Click-through detail views
    2) Backstage and the integration of all the components.
    3) The IDP, providing golden paths for the development of new applications,
       provisioning workloads, etc. Specifically for the onboarding of new developers
       and getting people up to speed when switching teams, etc.
    4) Golden paths & documentation
    5) The "connectiveness" in the platform, everything is connected and by it, it helps
       the developer as it lessens the burden of getting applications ready for
       deployment.
```

```
Are there features that you think should be added to the platform?
    1) Mesh and Trace correlation to graph views
    2) Policy agents (Gatekeeper) can be helpful to avoid mistakes. Enforcing guard
       rails can help the security and stability of the platform.
    3) Definitely the IDP, maybe even the PaaS aspect of your platform, because; why re-
       invent the wheel?
    4) More elaborate and integrated golden paths. Also looking for upgrade paths, as we
       now favor constructing new things over maintaining / upgrading the old one.
```

```
Does this implementation help you with your technical implementation at your
    organization? If yes, how?
    1) It would help for demonstration purposes, especially Backstage; but due to the
       IAM integration and HA constraints it would not work in this configuration out
       of the box.
    2) For Otomi, it is very good to have a showcase of what is possible and it is easy
       to PoC some tools. Using it as a platform could be more difficult because you
       have to "play" by their rules. You give away some flexibility (their pace of
       development is slower then from the components itself ) and tailor-made usage we
       now have (Integration between our IaC and Kubernetes clusters for example)
    3) I really liked the implementation and possibilities of Backstage, impressed by
       the demo and I am convinced we need to spend time to implement this in our
       platform.
    4) I think the work for the developers will be much easier this way. For the
       platform team managing and standardizing will take off. Also onboarding of new
       team members (or applications) will be much easier than it is now.
    5) It shows what can be done (het maakt je lekker). It shows the power. But it does
       not help us to actually get there, as the tools and integration differ per use
       case.
```

```
Additional comments or suggestions
    1) This is a strong demonstration due to the "batteries_included" approach instead
       of leaning on a specific service vendor. On the other hand, this has the
       downside of being reliant on a single VM image vendor, which abstracts away some
       of the component installation.
    2) Great work. We have been talking about implementing something like this for a
       year now. Great to see a working example. Keep up the good work.
```

Based on the feedback, we were able to make several significant conclusions regarding our small-scale prototype. The most valued feature appears to be the developer portal, which development teams can leverage to carry out platform-related tasks. Through the demo and discussions, we observed that these *golden paths* could help create more standardized configurations and reduce errors since the developer portal handles these. The absence of certain features is due to the project's limited scope and duration. This is also compounded by the fact that such features are typically organization-specific and may vary between organizations. The feedback related to the use of Otomi is valid, particularly for large organizations that wish to avoid vendor lock-in. However, for the purposes of a small prototype, Otomi can streamline the setup process and deliver value quickly. Overall, the feedback was positive, suggesting that this implementation could serve as a beneficial example for organizations considering the implementation of an engineering platform and wishing to understand the value of such an endeavor.

In conclusion, we asked if the platform experts would recommend this platform as an example to other organizations. The responses were unanimous: all agreed that this could serve as a valid example for other organizations, reflecting the core aim of this implementation. For a more detailed overview, all questions and responses from this experiment are documented in Appendix F.

Chapter 5

Discussion

In this research, we have established a Platform Engineering Reference Model (PE-RM), based on the ODP-RM reference model [26], designed to assist organizations in embracing the discipline of platform engineering. With five distinct viewpoints, our model explores platform engineering from various perspectives, encompassing both technical and organizational aspects. Despite the increasing popularity and relevance of platform engineering in today's digital landscape, the field remains underrepresented in scientific literature and inadequately explored in industrial articles and whitepapers. This often results in difficulties for organizations aiming to comprehend and embrace the discipline fully. Our research aims to fill this gap, providing a comprehensive reference model that enables a more profound understanding of platform engineering and facilitating its adoption. To enhance the practical relevance of our reference model, we conducted a case study, presenting both conceptual and technical implementations of platform engineering guided by the PE-RM. Notably, we found that existing reference architectures, such as the Internal Developer Portal proposed by Humanitec [16], often fail to consider the organizational aspects crucial to platform engineering. Additionally, the relatively dated existing scientific literature [21, 22] fails to encapsulate the platform engineering discipline's scope fully.

Our research, enriched by an in-depth case study and a series of validating experiments, has resulted in key conclusions highlighting the effectiveness and adaptability of the PE-RM. The reference model encapsulates all recognized aspects of platform engineering. This comprehensive scope positions the model as a robust tool capable of enhancing software processes and overcoming organizational challenges. Its applicability extends far beyond addressing technical challenges. It also provides concrete solutions to a wide array of issues that software organizations frequently struggle with. The practical application of the model demonstrated in the case study serves as a demonstration of its efficacy. Another significant finding of our research is that the reference model can be successfully adapted into a contextually relevant technical implementation. This adaptation proves helpful in tackling a broad range of software challenges specific to an organization. The demonstrated adaptability in the case study shows the maturity of the model's technical viewpoints and their interrelation with other aspects. In addition, our research reveals that the platform implementation resulting from the reference model can serve as a valuable guide for organizations. It offers a concrete demonstration of adopting a platform engineering approach, emphasizing its unique features and added value. This practical illustration can significantly benefit organizations aiming to adopt platform engineering by providing a reference for their implementation journey. Lastly, our research affirms the potential of the Platform Engineering Reference Model to bring about significant improvements in productivity, expressed in performance metrics. Through a detailed conceptual analysis during the case study and subsequent experimental results, we observed that organizations adopting a platform engineering approach delineated by the model could experience substantial gains in productivity, as showcased in the experiment results. In conclusion, these key findings underscore the vast potential and practical implications of our reference model for platform engineering. The model shows the promising capacity to address existing challenges in software organizations while optimizing their performance metrics, making it an essential tool in the landscape of platform engineering.

Our implementation of the engineering platform demonstrated within the case study utilized open-source tooling. It resulted in the development of custom tools and applications that are open-sourced for use within any organization. By validating it within Wehkamp and their industry experts, we can assume that industry professionals have validated the implementation in combination with the experimental results.

[16]`https://humanitec.com/reference-architectures`

During the development of our reference model and its associated conceptual and technical designs, we discovered that methodology modeling is inherently complex. We observed that introducing new roles within an organization necessitates clear accountability structures, which initially we had not considered. Additionally, validating the applicability of our reference model to a diverse range of organizations was challenging, given the unique nature of each organization. Despite these hurdles, discussions with experts outside the organization enabled us to refine our model. Furthermore, our research highlighted the necessity of considering greenfield and brownfield projects [32], which present unique considerations in implementing platform engineering. In addition, during the creation of the reference model, we experienced difficulty modeling different operations and interactions since they differ for each organization. This is also the case for the technology viewpoint, which focusses more on examples, rather than state-of-the-art, since it can be challenging to model all the different integrations.

Although our research is grounded in a single case study, which may appear to threaten validity, consultations with external experts and experiments with developers outside the organization imply broader applicability. Since we already used a PaaS solution that takes care of many of the technical challenges related to integrating tools into a single platform, we still had to find a way to make it easier for developers, therefore introducing a developer portal and configuring the tools so it would all be working as one platform. One of the challenges was the integration of many interfaces into one tool, which was also one of the main feedback points from the experiments. Moreover, it's worth noting that our technical implementation, while basic and straightforward, lacks some fundamental features such as form validation and improved user experience. As our primary focus was to validate the reference model and provide an example, these aspects were not prioritized but should be considered in further research. In addition, the performance metrics used to compare the technical implementation to the contextual analysis and participants' experience could be expanded to cover more metrics to give a broader view of platform engineering's feedback.

Chapter 6

Conclusion

In this research, our primary aim was to develop a reference model that elucidates platform engineering within a software organization, thereby facilitating the adoption of platform engineering. Our proposed Platform Engineering Reference Model (PE-RM) serves as a valuable tool for organizations seeking to transition from existing methodologies to platform engineering or for those initiating their journey in this field. Our conceptual design serves as a roadmap for organizations keen on using the reference model to integrate platform engineering into their processes. In addition, we created an illustrative platform implementation following the guidelines of the reference model, which enables us to substantiate the efficacy of the reference model. Upon conducting extensive research, it becomes clear that platform engineering is a specialized field. It involves the unification of a technological engineering platform with specific lifecycle stages, roles, and information objects to form a unique methodology that can be applied to the entire software organization. The principal aim of platform engineering is to streamline, standardize, and centralize the tools and best practices across all development teams. Unlike DevOps, platform engineering significantly reduces the cognitive load on developers. Instead of requiring developers to configure the entire application lifecycle individually, platform engineering introduces standardization and centralization. This approach not only simplifies processes but also promotes consistency and efficiency across the board, ultimately leading to higher productivity and better-quality outcomes. We initiated this study with several research questions, to which we can now provide comprehensive answers.

Research Question 1: How to model platform engineering in the context of a software company?
In chapter 3, we introduce the PE-RM, which elucidates platform engineering in the context of a software organization. It encompasses five viewpoints: enterprise, information, computation, engineering, and technology. The enterprise, information, and computation viewpoints primarily focus on the organizational aspect; this includes the suggestion of new roles, lifecycles, and operations. In contrast, the engineering and technology viewpoints provide a framework for modeling the technical implementation of the engineering platform. The validation of this reference model was achieved through expert feedback, comparison with an alternative reference architecture, and a case study, which operationalized the PE-RM, thereby proving its applicability. The case study also includes a technical implementation demonstrating the technical feasibility of the reference model.

Research Question 2: How to define a customized platform engineering design tailored to a specific organization?
In chapter 4, we present a conceptual design guided by the PE-RM developed in chapter 3. This design includes distinct sections outlining the analysis, requirement analysis, and design. Serving as a practical guide for organizations, it elucidates how platform engineering can be operationalized with the assistance of the PE-RM. We extracted crucial information through the analysis, enabling us to develop a platform engineering design based on the PE-RM. Simultaneously, the design provides validation and guidance for other organizations intending to utilize the Platform Engineering Reference Model.

Research Question 3: How to effectively construct a technical platform engineering implementation?
In chapter 4, we outline a technical platform implementation guided by the PE-RM. Drawing from the conceptual design and reference model, we conducted a requirements analysis, designed an engineering platform architecture, and implemented a basic engineering platform. This implementation, which includes the design choices and unique features of the platform, serves as an exemplar for operationalizing platform engineering at a technical level. We conducted numerous experiments to validate the

implementation, illustrating the advantages of platform engineering within a software organization.

Main research question: How can a software organization effectively integrate platform engineering using a comprehensive reference model?
The answer to the main research question is synthesized from the responses to the subsidiary research questions. Besides these answers, we developed an encompassing Platform Engineering Reference Model, modeling platform engineering from various perspectives to grasp the entire methodology holistically. We also proposed a conceptual design guided by this reference model to provide a practical guide for designing platform engineering within an organization. Furthermore, we created a basic engineering platform implementation serving as a proof of concept demonstrating the technical implementation process, including experiments to validate the applicability and added value of the implementation.

6.1 Future work

The Platform Engineering Reference Model (PE-RM) can be further enhanced by validating its applicability by conducting case studies across various organizations. Examining its efficacy across organizations of varying sizes and sectors would augment the model's validity. Such an approach would also enable the evaluation of the model's universality and adaptability to a broad range of contexts. The technical implementation currently serves as a concise and effective prototype that showcases the potential benefits of platform engineering. However, it remains basic. The engineering platform could be further augmented by incorporating additional functionalities that span across more stages of the application lifecycle. For instance, provisioning an application could include features such as tagging or enabling applications to be exposed simultaneously. Usability could be amplified by integrating all functionalities into a singular, user-friendly interface. There also exists potential for constructing an engineering platform from scratch or utilizing different tools, which could yield insightful comparisons. However, this lies beyond the scope of the current research. Lastly, given a more expansive timeframe and an additional case study, the PE-RM could be further refined. For instance, observing an organization with fully integrated platform engineering, as per this reference model, would offer valuable insights into the model's organizational benefits in greater detail, including more performance metrics.

Acknowledgements

We would like to thank Dr. Z. Zhao because he has had a massive influence on this research via his insightful scientific recommendations. With his guidance, we were able to bring this research to a high level with papers publishable to high-level conferences. Also, we want to thank K. Bakker for his support during this research with insightful meetings and discussions on implementing platform engineering within the organization. We also want to thank all the people we have spoken to within Wehkamp and outside the organization. Finally, we want to thank Wehkamp for hosting this project.

Bibliography

[1] A. Bharadwaj, O. A. E. Sawy, P. A. Pavlou, and N. Venkatraman, “Digital business strategy: Toward a next generation of insights,” *MIS Quarterly*, vol. 37, no. 2, pp. 471–482, 2013, ISSN: 02767783. DOI: 10.25300/misq/2013/37:2.3.

[2] B. Fitzgerald and K.-J. Stol, “Continuous software engineering: A roadmap and agenda,” *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017, ISSN: 0164-1212. DOI: 10.1016/j.jss.2015.06.063.

[3] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe, “A decade of agile methodologies: Towards explaining agile software development,” en, *Journal of Systems and Software*, Special Issue: Agile Development, vol. 85, no. 6, pp. 1213–1221, Jun. 2012, ISSN: 0164-1212. DOI: 10.1016/j.jss.2012.02.033.

[4] L. E. Lwakatare, P. Kuvaja, and M. Oivo, “Dimensions of devops,” in *Agile Processes in Software Engineering and Extreme Programming*, C. Lassenius, T. Dingsøyr, and M. Paasivaara, Eds., Springer International Publishing, 2015, pp. 212–217, ISBN: 978-3-319-18612-2.

[5] M. Fowler, J. Highsmith, *et al.*, “The agile manifesto,” *Software development*, vol. 9, no. 8, pp. 28–35, 2001.

[6] G. Kim, J. Humble, P. Debois, J. Willis, and N. Forsgren, *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations*, en. IT Revolution, Nov. 2021, ISBN: 978-1-950508-43-3.

[7] Humanitec, “State of platform engineering report,” Tech. Rep., 2022, p. 18.

[8] N. Kersten, *The Top DevOps Trends from Our 2021 State of DevOps Report*, en. [Online]. Available: https://www.puppet.com/blog/devops-trends.

[9] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, en. Pearson Education, Jul. 2010, ISBN: 978-0-321-67022-9.

[10] P. Kruchten, R. L. Nord, and I. Ozkaya, “Technical Debt: From Metaphor to Theory and Practice,” *IEEE Software*, vol. 29, no. 6, pp. 18–21, Nov. 2012, Conference Name: IEEE Software, ISSN: 1937-4194. DOI: 10.1109/MS.2012.167.

[11] H. H. Olsson, H. Alahyari, and J. Bosch, “Climbing the ”Stairway to Heaven” – A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software,” in *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, ISSN: 2376-9505, Sep. 2012, pp. 392–399. DOI: 10.1109/SEAA.2012.54.

[12] L. E. Lwakatare, P. Kuvaja, and M. Oivo, “Dimensions of DevOps,” en, in *Agile Processes in Software Engineering and Extreme Programming*, C. Lassenius, T. Dingsøyr, and M. Paasivaara, Eds., ser. Lecture Notes in Business Information Processing, Cham: Springer International Publishing, 2015, pp. 212–217, ISBN: 978-3-319-18612-2. DOI: 10.1007/978-3-319-18612-2_19.

[13] D. E. Strode, S. L. Huff, B. Hope, and S. Link, “Coordination in co-located agile software development projects,” en, *Journal of Systems and Software*, Special Issue: Agile Development, vol. 85, no. 6, pp. 1222–1238, Jun. 2012, ISSN: 0164-1212. DOI: 10.1016/j.jss.2012.02.017.

[14] R. Makhlouf, “Cloudy transaction costs: A dive into cloud computing economics,” *Journal of Cloud Computing*, vol. 9, Jan. 2020. DOI: 10.1186/s13677-019-0149-4.

[15] I. Matt Campbell, “The Platform Engineering Guide: Principles and Best Practices,” en, *InfoQ*, 2023. [Online]. Available: https://www.infoq.com/minibooks/platform-engineering-guide/.

[16] F. M. A. Erich, C. Amrit, and M. Daneva, "A qualitative study of DevOps usage in practice," en, *Journal of Software: Evolution and Process*, vol. 29, no. 6, Jun. 2017, ISSN: 2047-7473, 2047-7481. DOI: 10.1002/smr.1885.

[17] R. W. Macarthy and J. M. Bass, "An Empirical Taxonomy of DevOps in Practice," *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 221–228, Aug. 2020, Conference Name: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) ISBN: 9781728195322 Place: Portoroz, Slovenia Publisher: IEEE. DOI: 10.1109/SEAA51224.2020.00046.

[18] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," en, *Information and Software Technology*, vol. 50, no. 9, pp. 833–859, Aug. 2008, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2008.01.006.

[19] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., *Site reliability engineering: how Google runs production systems*, First edition. Beijing ; Boston: Oreilly, 2016, ISBN: 978-1-4919-2912-4.

[20] M. Skelton and M. Pais, *Team topologies: organizing business and technology teams for fast flow.* It Revolution, 2019.

[21] J. Zhou, Y. Ji, D. Zhao, and J. Liu, "Platform engineering in enterprise application development," in *2010 International Conference on E-Business and E-Government*, 2010, pp. 112–115. DOI: 10.1109/ICEE.2010.36.

[22] M. A. McCarthy, L. M. Herger, S. M. Khan, and B. M. Belgodere, "Composable DevOps: Automated Ontology Based DevOps Maturity Analysis," in *2015 IEEE International Conference on Services Computing*, Jun. 2015, pp. 600–607. DOI: 10.1109/SCC.2015.87.

[23] J. A. Zachman, "The zachman framework for enterprise architecture," *Primer for Enterprise Engineering and Manufacturing.[si]: Zachman International*, 2003.

[24] C. M. Pereira and P. Sousa, "A method to define an enterprise architecture using the zachman framework," in *Proceedings of the 2004 ACM Symposium on Applied Computing*, ser. SAC '04, Nicosia, Cyprus: Association for Computing Machinery, 2004, pp. 1366–1371, ISBN: 1581138121. DOI: 10.1145/967900.968175.

[25] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, R. Metz, and B. A. Hamilton, "Reference model for service oriented architecture 1.0," *OASIS standard*, vol. 12, no. S18, pp. 1–31, 2006.

[26] K. Raymond, "Reference Model of Open Distributed Processing (RM-ODP): Introduction," en, in *Open Distributed Processing: Experiences with distributed environments. Proceedings of the third IFIP TC 6/WG 6.1 international conference on open distributed processing, 1994*, ser. IFIP — The International Federation for Information Processing, K. Raymond and L. Armstrong, Eds., Boston, MA: Springer US, 1995, pp. 3–14, ISBN: 978-0-387-34882-7. DOI: 10.1007/978-0-387-34882-7_1.

[27] P. F. Linington, Z. Milosevic, A. Tanaka, and A. Vallecillo, *Building Enterprise Systems with ODP: An Introduction to Open Distributed Processing*, en. CRC Press, Sep. 2011, ISBN: 978-1-4398-6625-2.

[28] H. Kilov, P. F. Linington, J. R. Romero, A. Tanaka, and A. Vallecillo, "The Reference Model of Open Distributed Processing: Foundations, experience and applications," en, *Computer Standards & Interfaces*, RM-ODP: Foundations, Experience and Applications, vol. 35, no. 3, pp. 247–256, Mar. 2013, ISSN: 0920-5489. DOI: 10.1016/j.csi.2012.05.003.

[29] A. Q. Gill, *Adaptive Cloud Enterprise Architecture*, en. World Scientific, Jun. 2015, ISBN: 978-981-4632-14-0.

[30] G. B. Ghantous and A. Q. Gill, "DevOps Reference Architecture for Multi-cloud IOT Applications," in *2018 IEEE 20th Conference on Business Informatics (CBI)*, ISSN: 2378-1971, vol. 01, Jul. 2018, pp. 158–167. DOI: 10.1109/CBI.2018.00026.

[31] E. Di Nitto, P. Jamshidi, M. Guerriero, I. Spais, and D. A. Tamburri, "A software architecture framework for quality-aware DevOps," in *Proceedings of the 2nd International Workshop on Quality-Aware DevOps*, ser. QUDOS 2016, New York, NY, USA: Association for Computing Machinery, Jul. 2016, pp. 12–17, ISBN: 978-1-4503-4411-1. DOI: 10.1145/2945408.2945411.

[32] C. D. Sousa, "Brownfield redevelopment versus greenfield development: A private sector perspective on the costs and risks associated with brownfield redevelopment in the greater toronto area," *Journal of Environmental Planning and Management*, vol. 43, no. 6, pp. 831–853, 2000. DOI: 10.1080/09640560020001719.

[33] R. van Solingen, “Measuring the roi of software process improvement,” *IEEE Software*, vol. 21, no. 3, pp. 32–38, 2004. DOI: 10.1109/MS.2004.1293070.

[34] D. Bernstein, “Containers and cloud: From lxc to docker to kubernetes,” *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014. DOI: 10.1109/MCC.2014.51.

[35] N. Forsgren, M. C. Tremblay, D. VanderMeer, and J. Humble, “Dora platform: Devops assessment and benchmarking,” in *Designing the Digital Transformation*, A. Maedche, J. vom Brocke, and A. Hevner, Eds., Cham: Springer International Publishing, 2017, pp. 436–440, ISBN: 978-3-319-59144-5.

Glossary

Agile A project management and product development methodology that emphasizes flexibility, collaboration, customer satisfaction, and delivering working software frequently. It's often used in software development, with methods including Scrum and Kanban. 4

API A set of rules and protocols for building and interacting with software applications. APIs define the methods and data formats that applications can use to communicate with each other. 10

CI/CD Continuous Integration/Continuous Deployment. 9

cloud computing A technology that uses remote servers on the internet to store, manage, and process data, rather than a local server or personal computer. It allows on-demand access to a shared pool of computing resources, which can be rapidly provisioned and released with minimal management effort. 5

DevOps A software development approach that combines software development (Dev) and information technology operations (Ops) to shorten the system development life cycle while delivering features, fixes, and updates frequently in close alignment with business objectives. 4

greenfield A project that lacks constraints imposed by prior work. 16

microservices A software development architecture where an application is structured as a collection of loosely coupled, independently deployable services. Each service, or "microservice", runs a unique process and communicates through a well-defined, lightweight mechanism (often HTTP-based APIs) to serve a specific business goal. 10

NestJS NestJS is a popular open-source, back-end framework for Node. js and TypeScript-based, server-side applications. It is intended to provide a solid foundation for developing server-side applications by leveraging well-known patterns and the best practices from other frameworks.. 44, 46

ODP-RM Open Distributed Processing Reference Model. 6

PaaS A cloud computing model that provides customers a platform to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app. 8

React React is a free and open-source front-end JavaScript library for building user interfaces based on components. It is maintained by Meta and a community of individual developers and companies. React can be used to develop single-page, mobile, or server-rendered applications. 44, 45

ROI Return of Investment. 17

Site Reliability Engineering A discipline that incorporates aspects of software engineering and applies them to infrastructure and operations problems. The main goals are to create scalable and highly reliable software systems. It was developed by Google and is often considered a specific implementation of DevOps. 5

Appendix A

GitHub links

Backstage: `https://github.com/RubenvdKamp08/backstage`

ArgoCD workload files: `https://github.com/RubenvdKamp08/argo-workload`

React template: `https://github.com/RubenvdKamp08/react-template`

Nest.js template: `https://github.com/RubenvdKamp08/nest-template`

Example site: `https://github.com/RubenvdKamp08/example-site`

Example service: `https://github.com/RubenvdKamp08/example-service`

Appendix B

Case study

B.1 Current tech stack

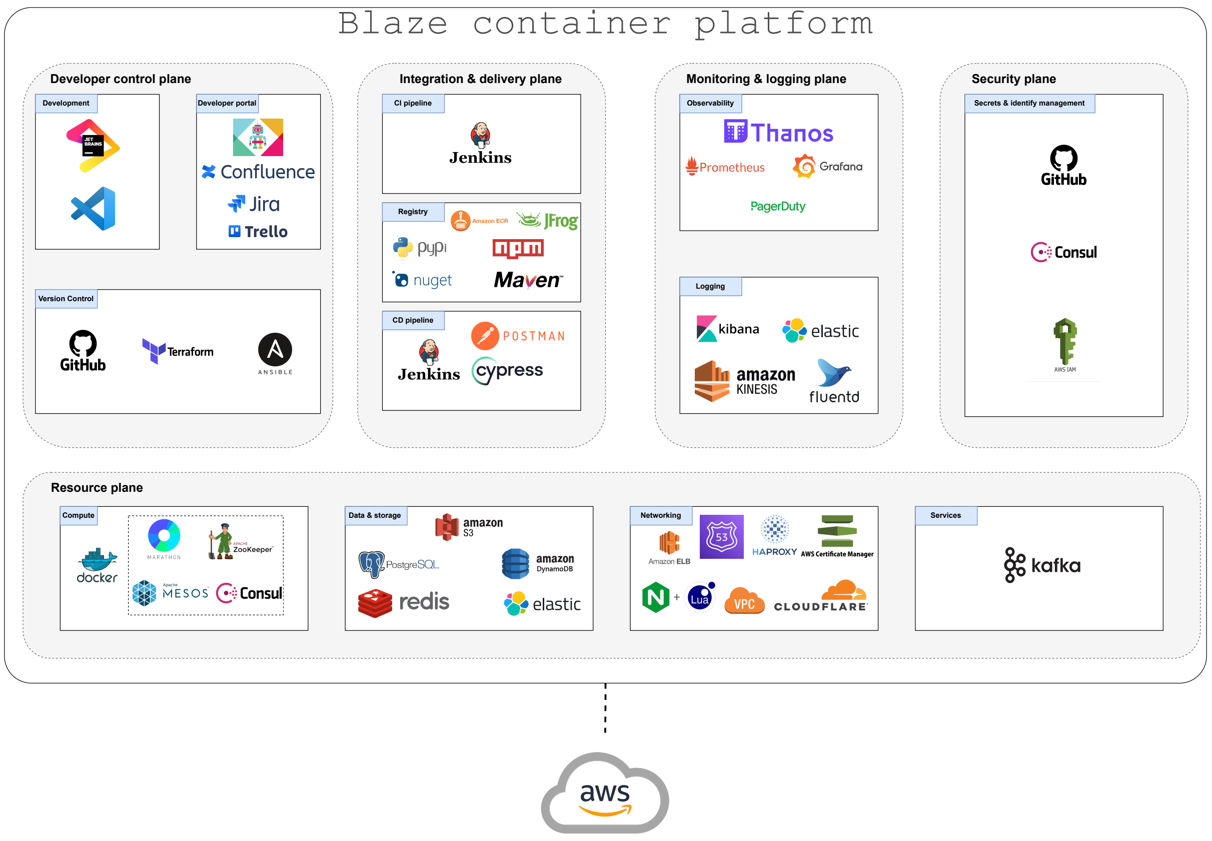


Figure B.1: Technology stack current environment

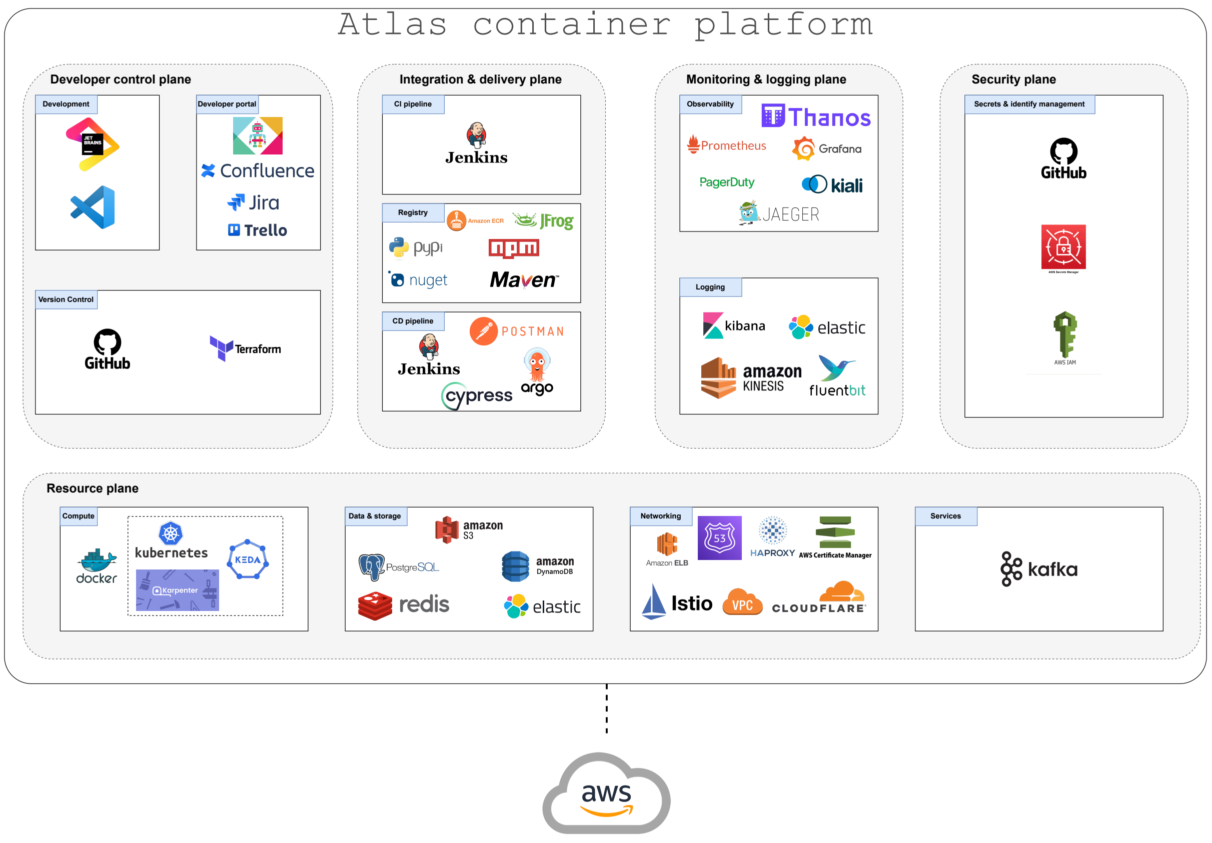
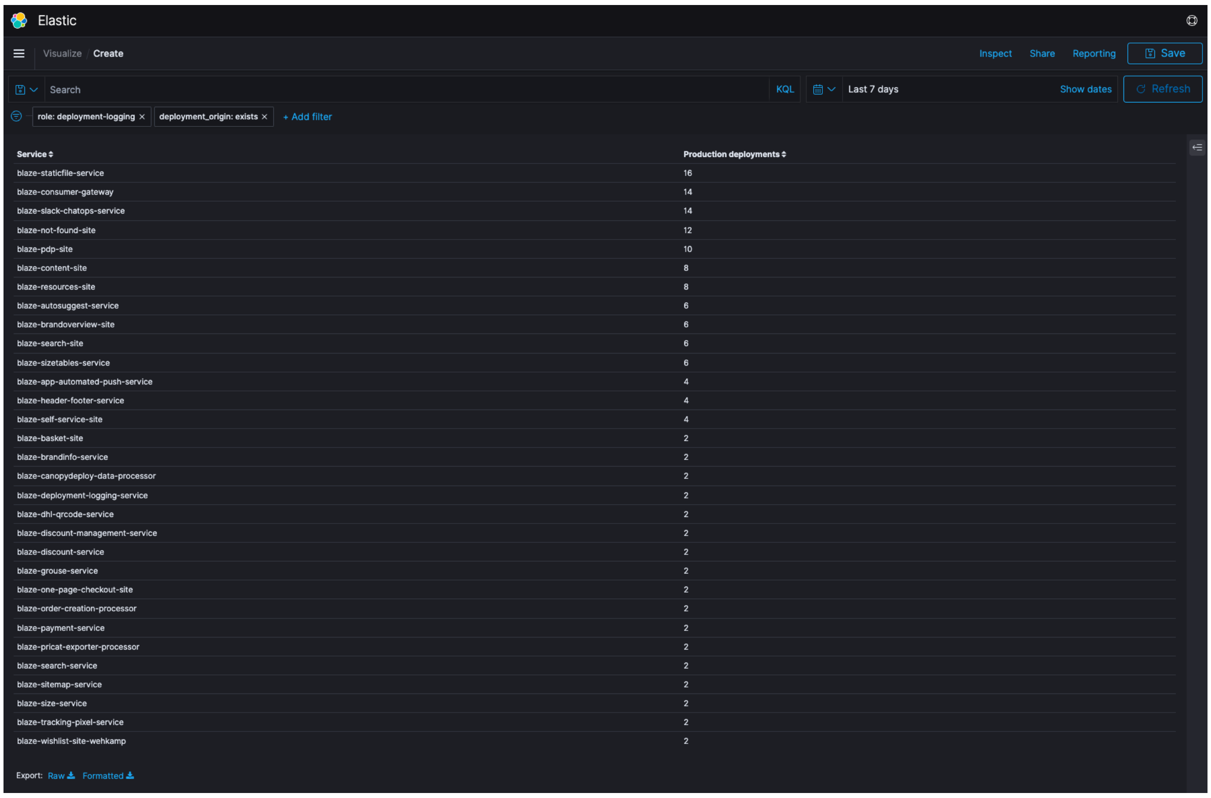


Figure B.2: Technology stack new environment

B.2 Performance metrics

Employee	**First day**	**First commit master**	**First story finished**
Employee 1 / team product information	10 April	April 26	2 May
Employee 2 / team customer service	1 January	10 January	11 January
Employee 3 / team fulfilment	1 June	19 June	3 July

Table B.1: Onboarding time developers.



Service	Production deployments
blaze-staticfile-service	16
blaze-consumer-gateway	14
blaze-slack-chatops-service	14
blaze-not-found-site	12
blaze-pdp-site	10
blaze-content-site	8
blaze-resources-site	8
blaze-autosuggest-service	6
blaze-brandoverview-site	6
blaze-search-site	6
blaze-sizetables-service	6
blaze-app-automated-push-service	4
blaze-header-footer-service	4
blaze-self-service-site	4
blaze-basket-site	2
blaze-brandinfo-service	2
blaze-canopydeploy-data-processor	2
blaze-deployment-logging-service	2
blaze-dhl-qrcode-service	2
blaze-discount-management-service	2
blaze-discount-service	2
blaze-grouse-service	2
blaze-one-page-checkout-site	2
blaze-order-creation-processor	2
blaze-payment-service	2
blaze-pricat-exporter-processor	2
blaze-search-service	2
blaze-sitemap-service	2
blaze-size-service	2
blaze-tracking-pixel-service	2
blaze-wishlist-site-wehkamp	2

Figure B.3: Weekly deployments old environment

Service	**Number of deployments**
blaze-product-enrichment-service	1
blaze-product-onboarding-price-service	1
blaze-product-onboarding-price-site	1
blaze-product-onboarding-service	1
blaze-search-product-index-processor	1
blaze-family-generator-site	1
blaze-enrichment-processor	1
blaze-discount-management-site	1
blaze-product-overview-pinboard-management-site	1
blaze-oms-service	5
atlas-configuration-management-site	1
atlas-configuration-management-service	1
blaze-content-site-wehkamp	3
atlas-product-information-slack-bot-service	2
blaze-navigation-service	11
blaze-one-page-checkout-site-nl.wehkamp	8
blaze-header-footer-service	1
blaze-resources-site	3
blaze-brandoverview-site	1
blaze-secret-sales-sync-service	1
blaze-search-term-redirect-service	1

blaze-product-taxonomy-processor	2
blaze-basket-site-nl.wehkamp	1
blaze-wishlist-site-wehkamp	1
blaze-pdp-site	1
blaze-search-site-wehkamp	1
blaze-acount-service-com	2
blaze-financial-transaction-manager-service	15
blaze-autosuggest-service	3
blaze-search-service	1
blaze-search-product-index-processor	1
blaze-oms-order-proxy-service	1
blaze-search-enrichment-processor	2
blaze-canopydeploy-data-processor	1
blaze-staticfile-service	8
blaze-discount-management-site	1
atlas-authentication-service	1
blaze-payment-ml-service	1
blaze-product-overview-pinboard-management-site	1
blaze-combination-service	1
blaze-financial-transaction-manager-service	15
blaze-search-service	1
blaze-brandinfo-service	1
sep-copy	1
blaze-search-product-index-processor	1
blaze-payment-ml-service	2
blaze-oms-order-proxy-service	1
blaze-search-enrichment-processor	1
blaze-canopydeploy-data-processor	1
blaze-tracking-pixel-service	2
blaze-product-overview-pinboard-management-site	1
blaze-combination-service	1

Table B.2: Production deployments new environment, retrieved from Git commits

To measure the onboarding time of an application, we decided to create a new front-end application with the latest Wehkamp packages and best practices. In order to fulfill this task, we asked a frontend developer how to fulfill this task. In the table below, we have stated the different steps and the time it took to finish each task to run this application in production. The application did not contain any custom code.

Task	**Description**	**Time**	**Observations**
Provision application	Create repository, docker registry and pipeline	30 minutes	1 minute GitHub, 8 minutes ECR, 1 minute Jenkins sync and 20 minutes for CD job visibility)
Create application code	Create an application with code, metrics exposure and docker configurations	70 minutes	There is no scaffolder, you have to copy and paste from another service
Build docker image	Build docker image and store in registry	10 minutes	Sometimes difficult to see which steps goes wrong in Jenkins
Deploy to development environment	Create workload and deploy the docker image in the development environment	10 minutes	There is no scaffolder. You have to copy and paste from other configuration file.
Deploy to production environment	Update workload and Deploy the docker image in the production environment	5 minutes	If you have setup the application-Set correctly, it is very easy.
Get logs	Get the logs of the application deployed	45 minutes	Because of custom packages, logs could be difficult to setup
Get metrics	Get the metrics of the application deployed	15 minutes	Difficult to find the dashboards.

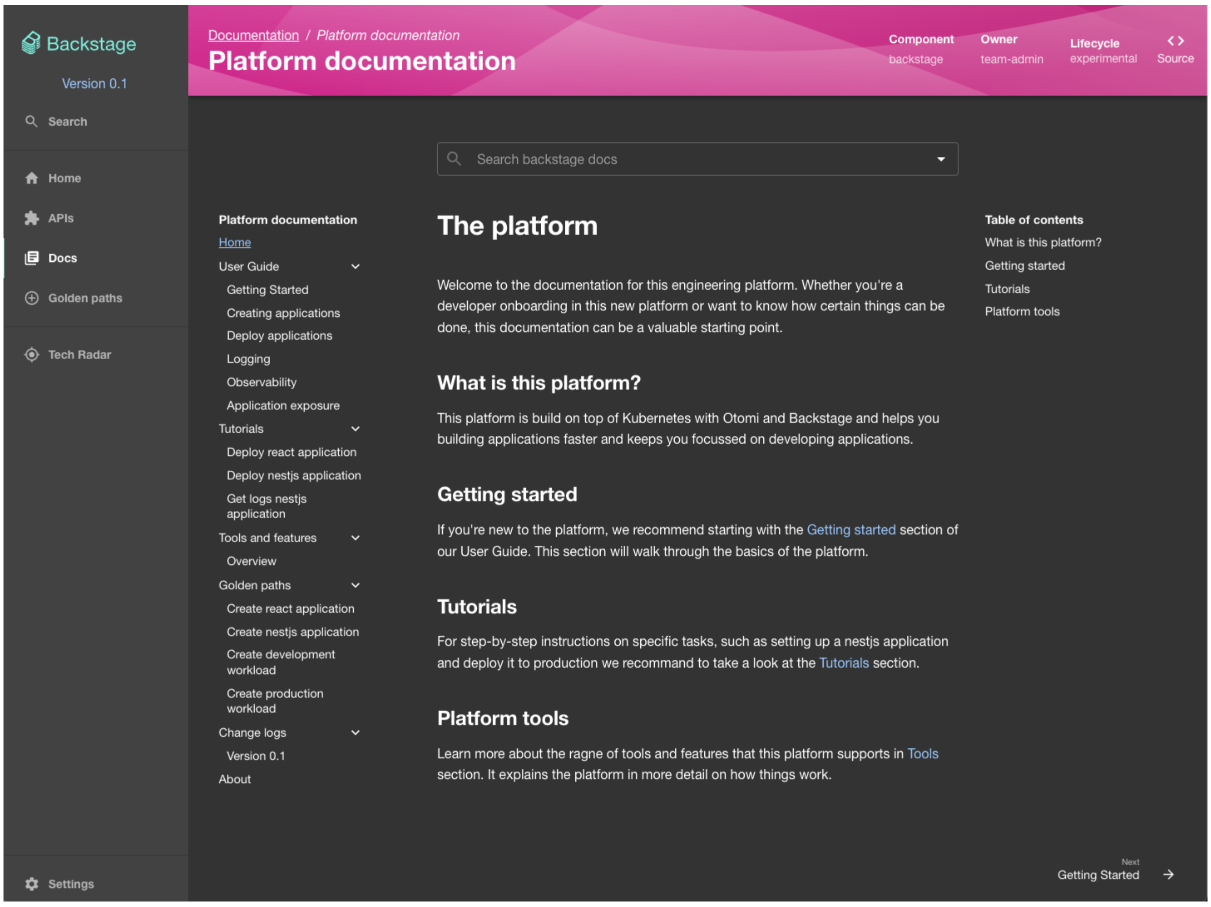
Table B.3: Onboarding time application.

Appendix C

Implementation

C.1 Platform documentation

For a detailed platform documentation on how the functionality like golden paths are working, please see the mkdocs files located in: `https://github.com/RubenvdKamp08/backstage/tree/master/docs`.
In backstage the docs were available and looked like this:



Platform documentation

The platform

Welcome to the documentation for this engineering platform. Whether you're a developer onboarding in this new platform or want to know how certain things can be done, this documentation can be a valuable starting point.

What is this platform?

This platform is build on top of Kubernetes with Otomi and Backstage and helps you building applications faster and keeps you focussed on developing applications.

Getting started

If you're new to the platform, we recommend starting with the Getting started section of our User Guide. This section will walk through the basics of the platform.

Tutorials

For step-by-step instructions on specific tasks, such as setting up a nestjs application and deploy it to production we recommand to take a look at the Tutorials section.

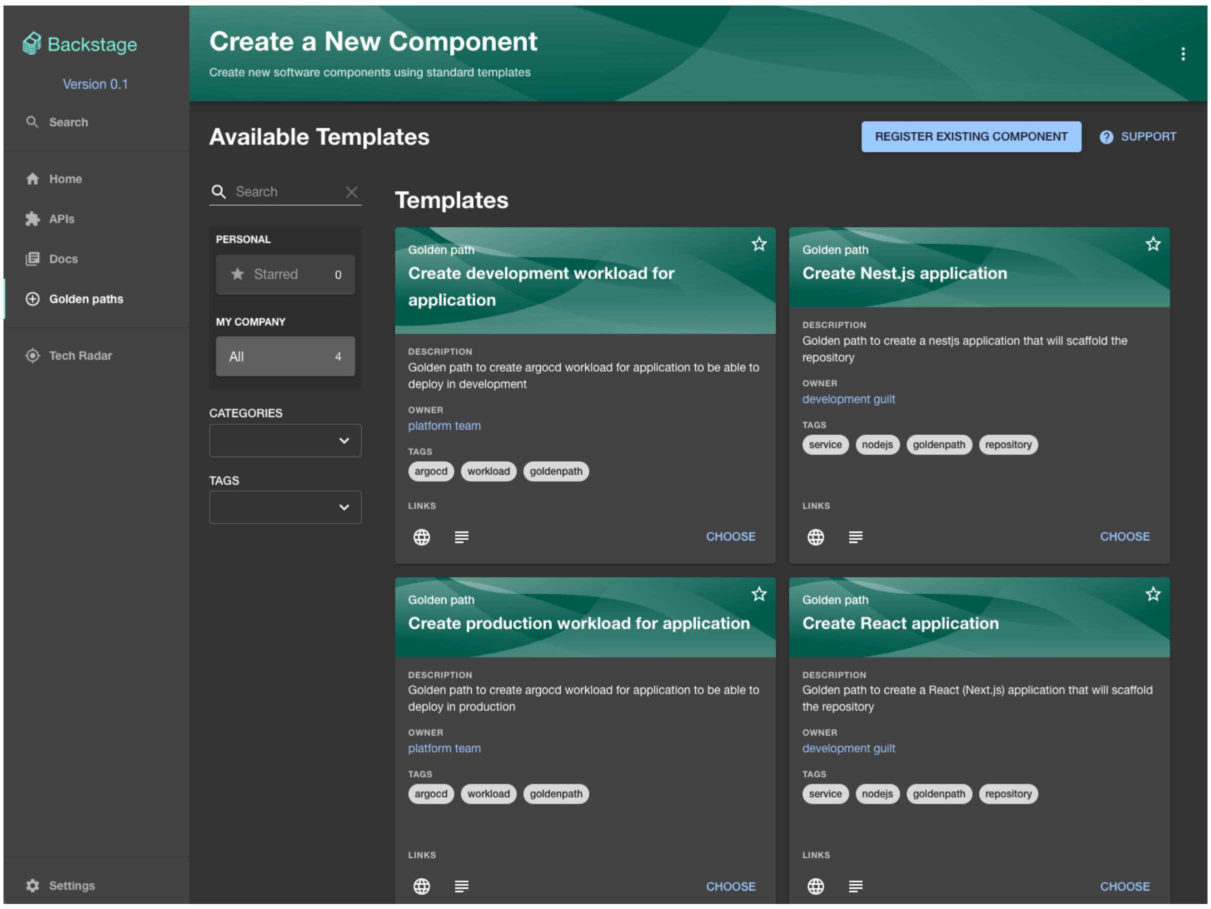
Platform tools

Learn more about the ragne of tools and features that this platform supports in Tools section. It explains the platform in more detail on how things work.

Figure C.1: Developer portal: platform documentation

C.2 Golden paths

For a more detailed explanation of the golden paths, please use the documentation defined in: `https://github.com/RubenvdKamp08/backstage/tree/master/docs/golden_paths`.



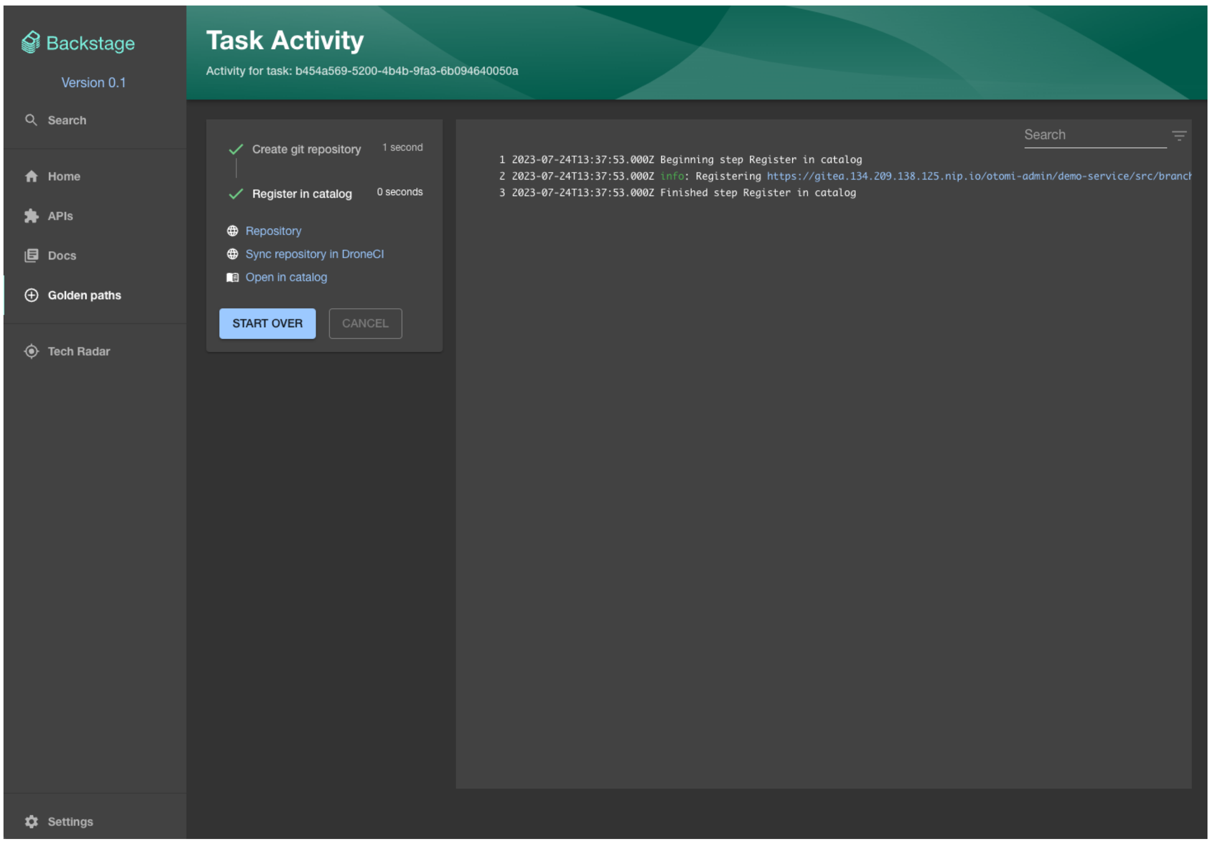
Golden path
Create development workload for application
DESCRIPTION
Golden path to create argocd workload for application to be able to deploy in development
OWNER
platform team
TAGS
argocd workload goldenpath

Golden path
Create Nest.js application
DESCRIPTION
Golden path to create a nestjs application that will scaffold the repository
OWNER
development guilt
TAGS
service nodejs goldenpath repository

Golden path
Create production workload for application
DESCRIPTION
Golden path to create argocd workload for application to be able to deploy in production
OWNER
platform team
TAGS
argocd workload goldenpath

Golden path
Create React application
DESCRIPTION
Golden path to create a React (Next.js) application that will scaffold the repository
OWNER
development guilt
TAGS
service nodejs goldenpath repository

Figure C.2: Developer portal: golden paths overview

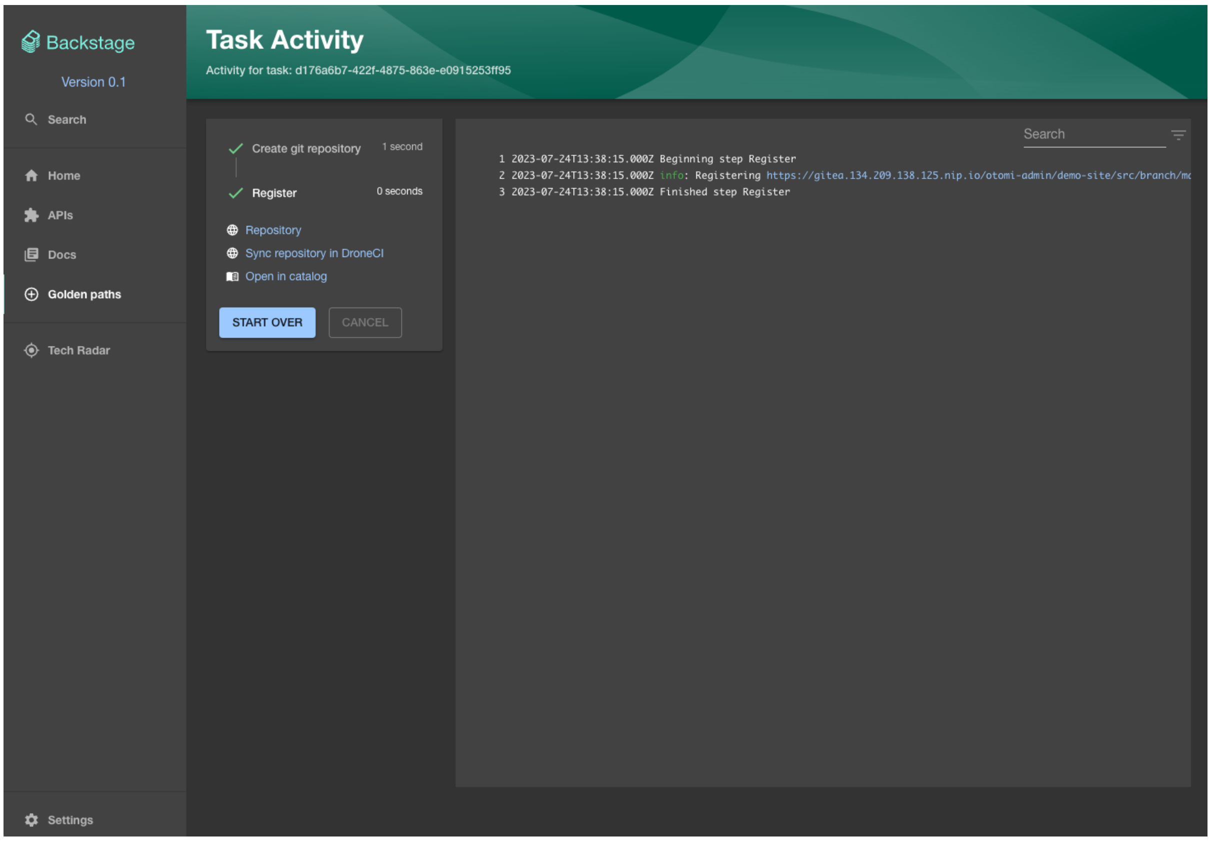


Task Activity

Activity for task: b454a569-5200-4b4b-9fa3-6b094640050a

```
2023-07-24T13:37:53.000Z Beginning step Register in catalog
2023-07-24T13:37:53.000Z info: Registering https://gitea.134.209.138.125.nip.io/otomi-admin/demo-service/src/branch
2023-07-24T13:37:53.000Z Finished step Register in catalog
```

Figure C.3: Golden path: create nest application

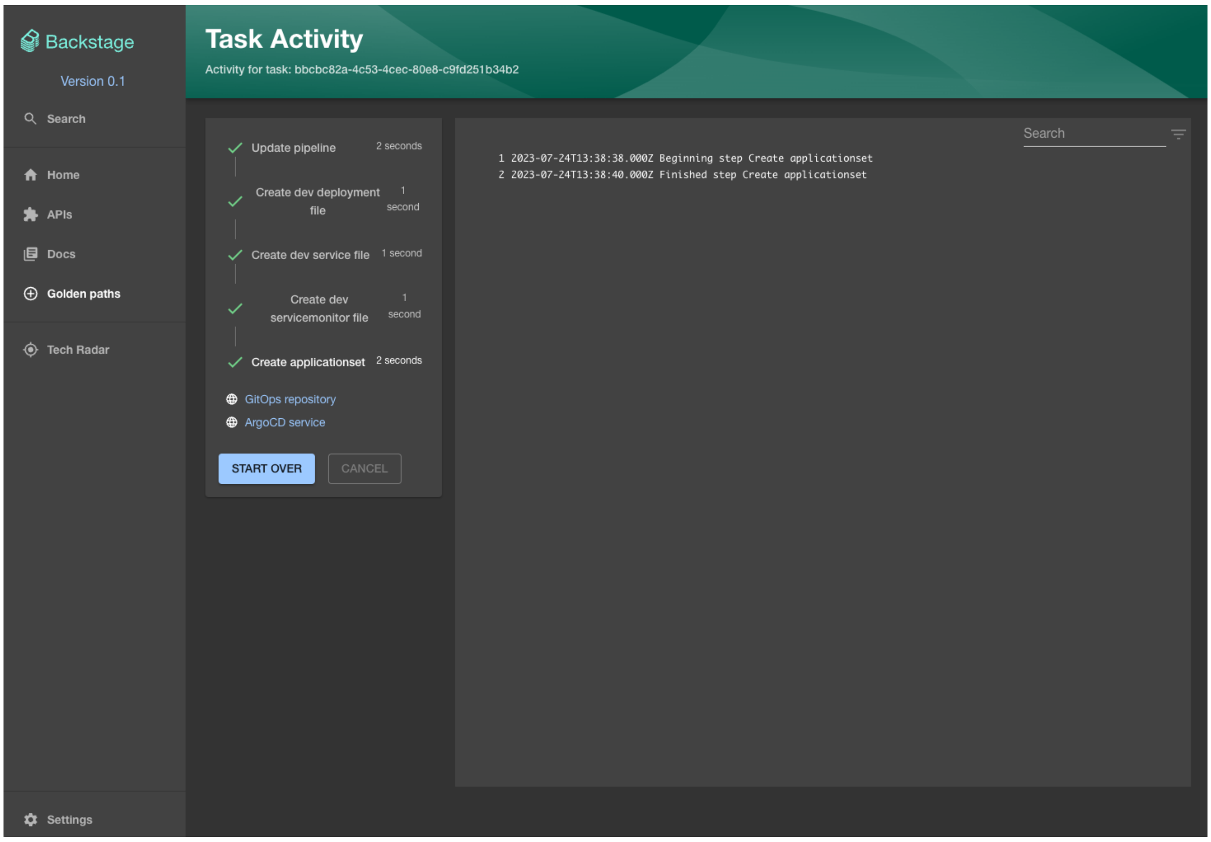


Task Activity

Activity for task: d176a6b7-422f-4875-863e-e0915253ff95

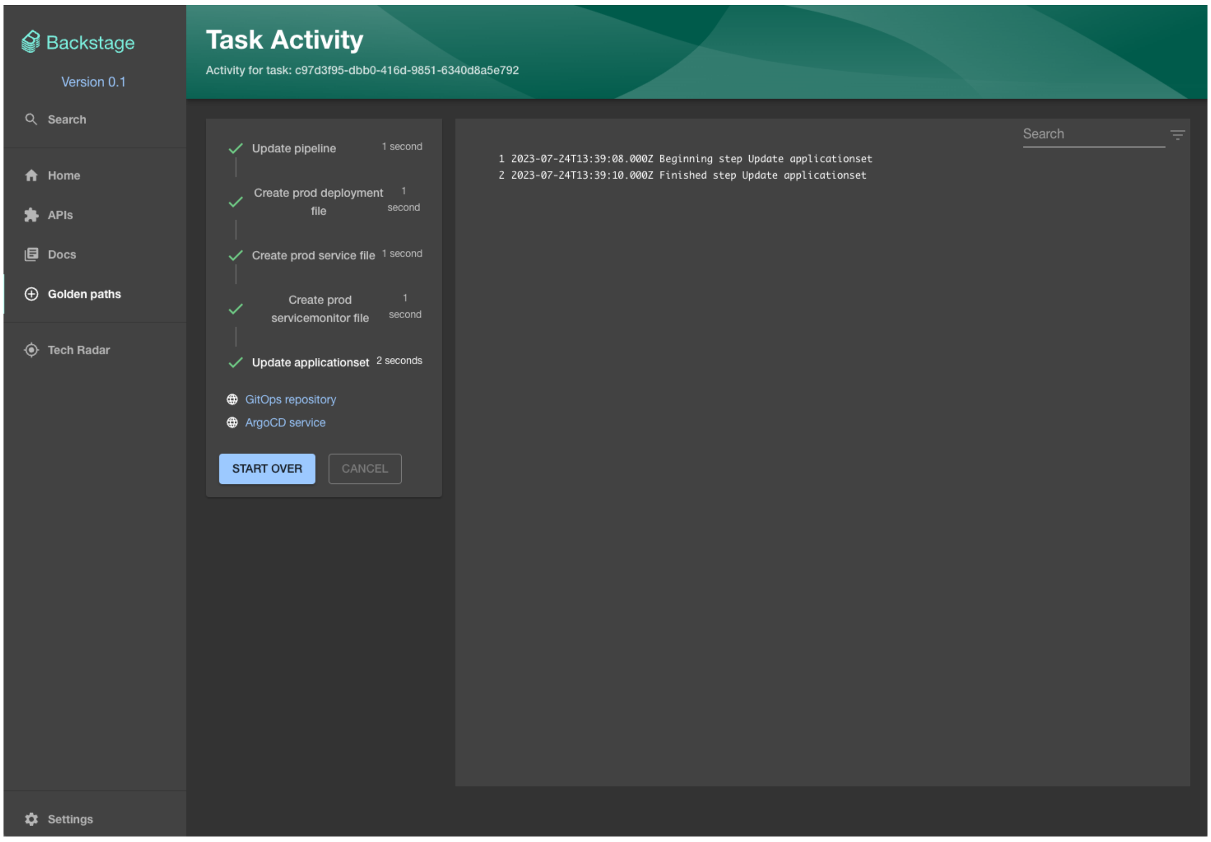
```
2023-07-24T13:38:15.000Z Beginning step Register
2023-07-24T13:38:15.000Z info: Registering https://gitea.134.209.138.125.nip.io/otomi-admin/demo-site/src/branch/mc
2023-07-24T13:38:15.000Z Finished step Register
```

Figure C.4: Golden path: create react application


Backstage
Version 0.1
Search
Home
APIs
Docs
Golden paths
Tech Radar
Settings
Task Activity
Activity for task: bbcbc82a-4c53-4cec-80e8-c9fd251b34b2
Update pipeline 2 seconds
Create dev deployment file 1 second
Create dev service file 1 second
Create dev servicemonitor file 1 second
Create applicationset 2 seconds
GitOps repository
ArgoCD service
START OVER
CANCEL
Search

```
1  2023-07-24T13:38:38.000Z Beginning step Create applicationset
2  2023-07-24T13:38:40.000Z Finished step Create applicationset
```

Figure C.5: Golden path: create development workload


Backstage
Version 0.1
Search
Home
APIs
Docs
Golden paths
Tech Radar
Settings
Task Activity
Activity for task: c97d3f95-dbb0-416d-9851-6340d8a5e792
Update pipeline 1 second
Create prod deployment file 1 second
Create prod service file 1 second
Create prod servicemonitor file 1 second
Update applicationset 2 seconds
GitOps repository
ArgoCD service
START OVER
CANCEL
Search

```
1  2023-07-24T13:39:08.000Z Beginning step Update applicationset
2  2023-07-24T13:39:10.000Z Finished step Update applicationset
```

Figure C.6: Golden path: create production workload

C.3 DroneCI pipeline configuration

```
---
kind: pipeline
type: kubernetes
name: build
steps:
  - name: yaml_validator
    image: devatherock/drone-yaml-validator:latest
    settings:
      debug: true
      continue_on_error: false
      allow_duplicate_keys: false
      ignore_unknown_tags: true
  - name: build-push
    image: plugins/docker
    settings:
      registry: harbor.134.209.138.125.nip.io
      repo: harbor.134.209.138.125.nip.io/team-admin/example-service
      insecure: true
      username:
        from_secret: REGISTRY_USERNAME
      password:
        from_secret: REGISTRY_PASSWORD
      tags:
        - ${DRONE_BUILD_NUMBER}
        - latest

---
kind: pipeline
type: kubernetes
name: update-docker-dev
clone:
  disable: true
steps:
  - name: clone-workload
    image: plugins/git
    commands:
      - git clone https://gitea.134.209.138.125.nip.io/otomi/argo-workload.git .
  - name: update-docker-tag
    image: alpine/git
    commands:
      - sed -i
        's/example-service:.*/example-service:'"${DRONE_BUILD_NUMBER}"'/g'
        ./example-service-dev/deployment.yaml
  - name: git-config
    image: alpine/git
    commands:
      - git config --global user.name "Drone_CI"
      - git config --global user.email "bot@drone.com"
  - name: git-push
    image: appleboy/drone-git-push
    settings:
      branch: master
      remote: https://gitea.134.209.138.125.nip.io/otomi/argo-workload.git
      username:
        from_secret: GIT_USERNAME
      password:
        from_secret: GIT_PASSWORD
      force: true
      commit: true
      commit_message: Update example-service-dev with docker image ${DRONE_BUILD_NUMBER}
      author_name: droneCI
      author_email: gitea@local.domain
depends_on:
  - build

---
kind: pipeline
type: kubernetes
name: update-docker-prod
clone:
  disable: true
```

```
steps:
  - name: clone-workload
    image: plugins/git
    commands:
      - git clone https://gitea.134.209.138.125.nip.io/otomi/argo-workload.git .
  - name: update-docker-tag
    image: alpine/git
    commands:
      - sed -i
        's/example-service:.*/example-service:'"${DRONE_BUILD_NUMBER}"'/g'
        ./example-service-prod/deployment.yaml
  - name: git-config
    image: alpine/git
    commands:
      - git config --global user.name "Drone CI"
      - git config --global user.email "bot@drone.com"
  - name: git-push
    image: appleboy/drone-git-push
    settings:
      branch: master
      remote: https://gitea.134.209.138.125.nip.io/otomi/argo-workload.git
      username:
        from_secret: GIT_USERNAME
      password:
        from_secret: GIT_PASSWORD
      force: true
      commit: true
      commit_message: Update example-service-prod with docker image ${DRONE_BUILD_NUMBER
        }
      author_name: droneCI
      author_email: gitea@local.domain
trigger:
  branch:
    - master
depends_on:
  - update-docker-dev
```

C.4 Custom backstage code

Location: `https://github.com/RubenvdKamp08/backstage/blob/master/packages/backend/src/plugins/scaffolder.ts`

```
...

export interface GiteaConfig {
  host: string
  password: string
}

export default async function createPlugin(
  env: PluginEnvironment,
): Promise<Router> {
  const catalogClient = new CatalogClient({
    discoveryApi: env.discovery,
  });

  const integrations = ScmIntegrations.fromConfig(env.config);

  const builtInActions = createBuiltinActions({
    integrations,
    catalogClient,
    config: env.config,
    reader: env.reader,
  });

  const giteaIntegration : any = env.config.get('integrations.gitea');
  const giteaConfig: GiteaConfig = {
    ...giteaIntegration[0],
    host: `https://${giteaIntegration[0].host}`
  }
  const actions = [
    ...builtInActions,
    publishGitea(giteaConfig), //custom actions
    updatePipeline(giteaConfig), //custom actions
    createKubeDeployment(giteaConfig), //custom actions
    createKubeService(giteaConfig), //custom actions
    updateApplicationWorkload(giteaConfig), //custom actions
    createServiceMonitorFile(giteaConfig) //custom actions
  ];
  ...
}
```

The custom functions are defined in: `https://github.com/RubenvdKamp08/backstage/blob/master/packages/backend/src/plugins/scaffolder/actions/custom.ts`.

Location: `https://github.com/RubenvdKamp08/backstage/blob/master/packages/backend/src/index.ts`

```
...
import { GiteaUrlReader } from './gitea/gitea';


function makeCreateEnv(config: Config) {
  const root = getRootLogger();
  const reader = UrlReaders.default({ logger: root, config, factories: [GiteaUrlReader.
      factory] }); // custom factory
  const discovery = HostDiscovery.fromConfig(config);
  const cacheManager = CacheManager.fromConfig(config);
  const databaseManager = DatabaseManager.fromConfig(config, { logger: root });
  const tokenManager = ServerTokenManager.noop();
  const taskScheduler = TaskScheduler.fromConfig(config);
  ...
}

...
```

The Gitea factory that was created to read docs: `https://github.com/RubenvdKamp08/backstage/blob/master/packages/backend/src/gitea/gitea.ts`

C.5 Observability & logging

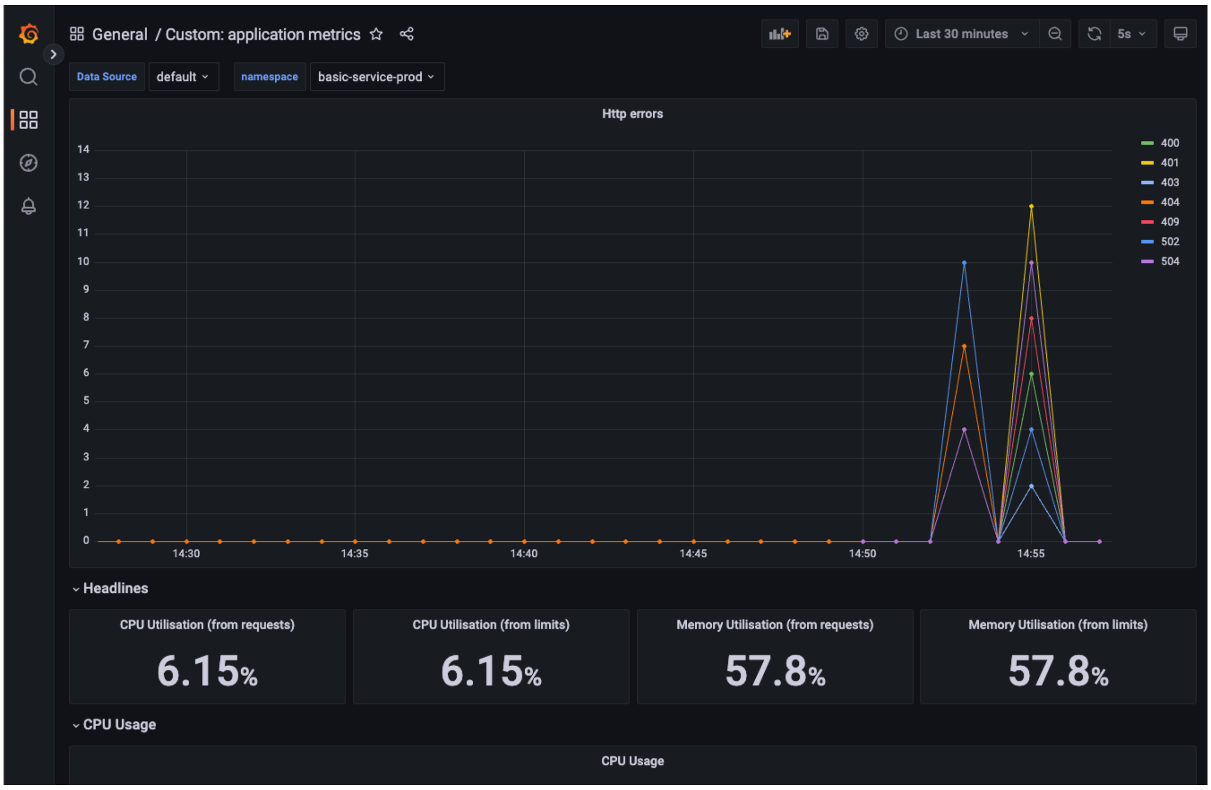
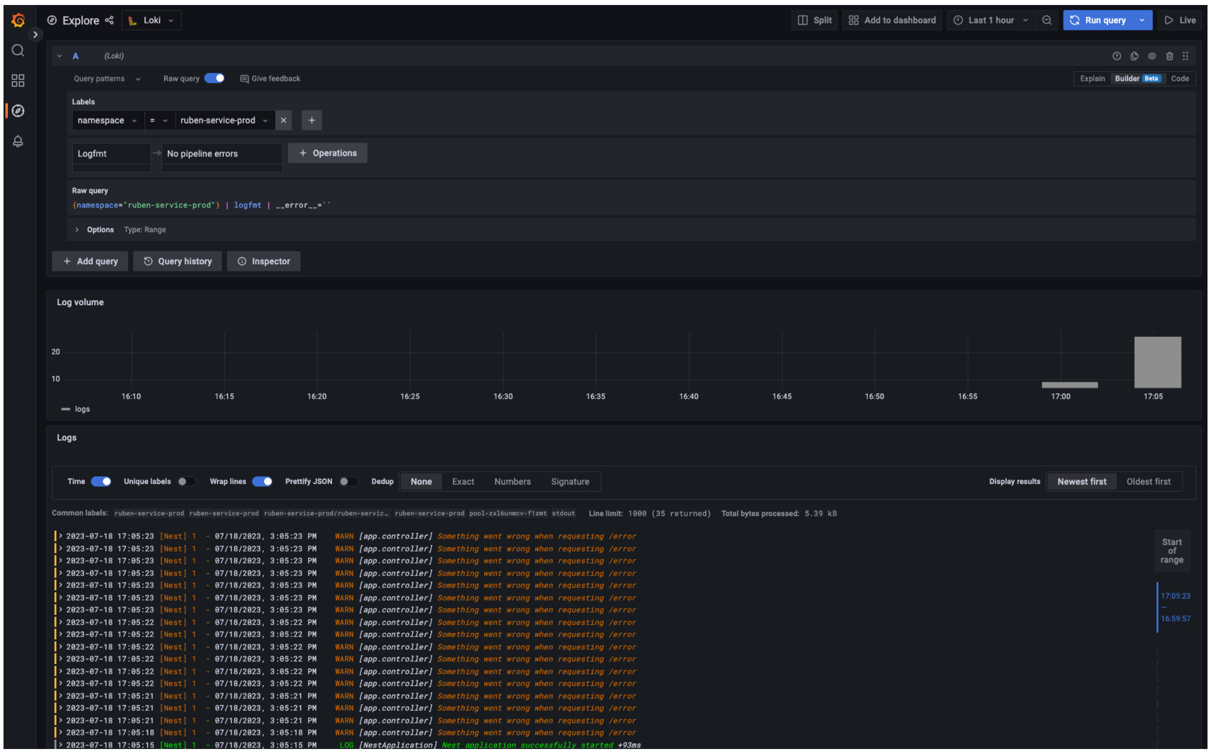


Figure C.7: Read metrics from grafana dashboards



Raw query

```
{namespace="ruben-service-prod"} | logfmt | __error__=``
```

Common labels: ruben-service-prod ruben-service-prod ruben-service-prod/ruben-servic… ruben-service-prod pool-zxl6unmcv-f1zmt stdout Line limit: 1000 (35 returned) Total bytes processed: 5.39 kB

```
> 2023-07-18 17:05:23 [Nest] 1  - 07/18/2023, 3:05:23 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:23 [Nest] 1  - 07/18/2023, 3:05:23 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:23 [Nest] 1  - 07/18/2023, 3:05:23 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:23 [Nest] 1  - 07/18/2023, 3:05:23 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:23 [Nest] 1  - 07/18/2023, 3:05:23 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:23 [Nest] 1  - 07/18/2023, 3:05:23 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:23 [Nest] 1  - 07/18/2023, 3:05:23 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:22 [Nest] 1  - 07/18/2023, 3:05:22 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:22 [Nest] 1  - 07/18/2023, 3:05:22 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:22 [Nest] 1  - 07/18/2023, 3:05:22 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:22 [Nest] 1  - 07/18/2023, 3:05:22 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:22 [Nest] 1  - 07/18/2023, 3:05:22 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:22 [Nest] 1  - 07/18/2023, 3:05:22 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:21 [Nest] 1  - 07/18/2023, 3:05:21 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:21 [Nest] 1  - 07/18/2023, 3:05:21 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:21 [Nest] 1  - 07/18/2023, 3:05:21 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:18 [Nest] 1  - 07/18/2023, 3:05:18 PM   WARN [app.controller] Something went wrong when requesting /error
> 2023-07-18 17:05:15 [Nest] 1  - 07/18/2023, 3:05:15 PM    LOG [NestApplication] Nest application successfully started +93ms
```

Figure C.8: Read application logs in Loki

Appendix D

Productivity evaluation

D.1 Introduction form

For this experiment I want to validate a platform engineering implementation and see if this way we can improve the productivity by offering a platform to handle a lot of configurations.

In order to do this usability study I will ask you to perform one or two tasks which are difficult at first, but with the basic documentation I will provide it should be feasible within 30 minutes.

D.2 Roles

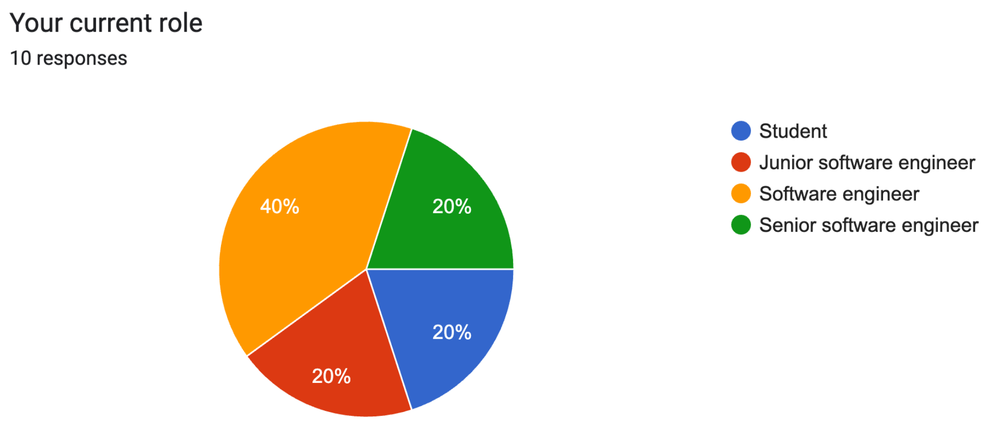


Figure D.1: Participant roles

D.3 Use case

D.3.1 Description

Before we get started with this experiment. We are wondering what your current estimations would be with your current knowledge about DevOps, platform engineering and especially on getting applications to production.

Imagine the following situation You are a software engineering within an organization which runs applications on Kubernetes. These deployments are handled with ArgoCD which automatically means they adopt a GitOps strategy. They also use a CI/CD provider like CircleCI/Jenkins/DroneCI/GitHub actions in which you can create your own pipeline configurations to build and deploy your applications. The organization also has a registry to store your docker containers. In order to get your application into production you have to make sure that your application is reachable to the outside world and it needs observability and logging in order to get information about your application. Before the application will be deployed to production you need to test your application in a development environment to ensure the quality.

In this setup, how long would it take to get your application to production? In this case you have to include the following facts:

- The CI/CD pipeline needs to build docker images
- The CI/CD pipeline needs to automatically update the deployments on dev and prod
- You need to have the dashboards of your application for monitoring purposes
- You need to have the ability to query the logs of your application

D.3.2 Responses

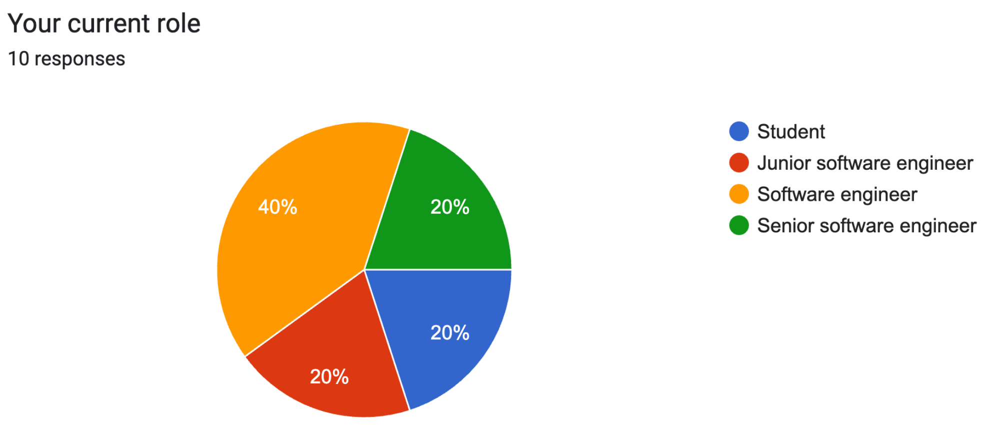


Figure D.2: Use case estimation

```
What do you think would be the most difficult part?
    1) Setting up the different deployment settings and different environments.
    2) Configuring all the tools so that they can communicate without error with
       eachother.
    3) Most difficult parts of the chain are usually mistakes in the build configuration
        and having to retry the entire pipeline.
    4) To have the dashboards of your application for monitoring purposes
    5) The pipeline configurations
    6) Depends on where the Kubernetes cluster is running and whether the cluster is
       managed or self-managed. Managed clusters are easier to integrate with than self
       -managed clusters. I think the part which will be most time-consuming is the CI
       setup. Especially the entire "DTAP" street. DTAP is a bit outdated, but the
       concepts remain. Testing should be done automatically, some systems may need a
       testing environment to be setup (think of e2e or acceptance testing). This can
       become complex quickly. Afterward, the automatic tests it needs to be deployed
       to an acceptance or demo environment where manual testing can be done (if
       necessary). After manual approval (or not) the changes can be merged to
       production. Setting up this process in alignment with business needs will be the
        most complex.
    7) Integration of building and deploying
    8) Configuration
    9) Because I have limited knowledge of dashboards and saving logs the last two parts
       :
    - You need to have the dashboards of your application for monitoring purposes
    - You need to have the ability to query the logs of your application
    would be the most difficult and timeconsuming for me.
    10) Setting up the pipeline because during pipeline configuration, trial and error
       can slow down progress
```

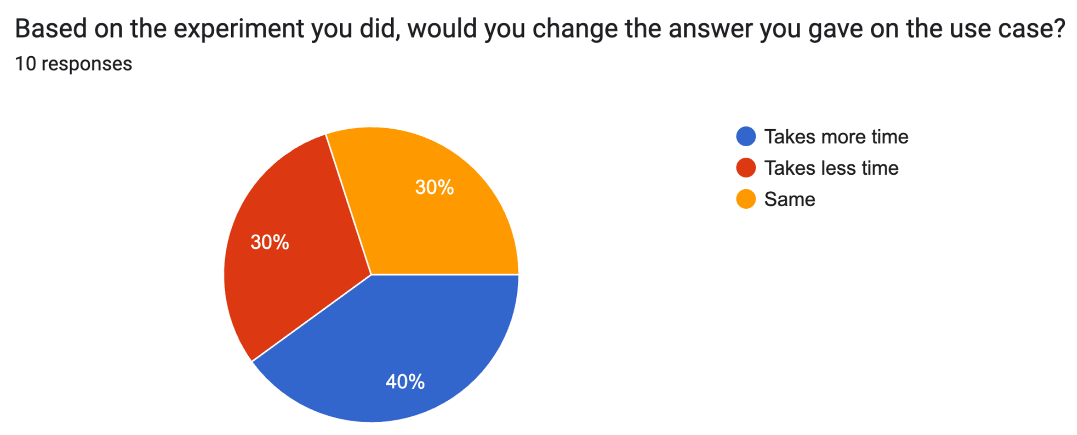


Figure D.3: Use case estimation change

D.4 Task one

In this first task you will be asked to create a react application and deploy this application to a development environment on Kubernetes. This react application and deployment needs to contain the following configuration/tooling:

- Git repository with the react code
- Pipeline setup to build the application in a docker image
- Pipeline setup to deploy the application to Kubernetes (dev)
- ArgoCD workload files to manage the deployments (dev)
- Exposure to the outside world

The deliverable are:

- A git repository with the code
- A docker image stored in the registry
- A Kubernetes deployment of your docker image
- A url to serve the application

The links required to get started are the following:

- backstage: this is the developer portal used to help you with certain tasks
- otomi: this is the developer portal used to navigate to the tools you need to use

The tools needed for this task are:

- Gitea: remote git
- DroneCI: CI platform
- Harbor: docker registry
- ArgoCD: GitOps CD tool for kubernetes
- Otomi services: expose applications

The credentials will be given in the google meet itself.

The goal of this task In this task you will need to get a react application to the development environment, you will end up with a url and on this url the site is served to the user. Example: https://example-site-dev.134.209.138.125.nip.io

Please ask questions if necessary.

D.5 Task two

In this second task you will be asked to create a NestJS application and deploy this application to a production environment on Kubernetes. This NestJS application and deployment needs to contain the following configuration/tooling:

- Git repository with the NestJS code
- Pipeline setup to build the application in a docker image
- Pipeline setup to deploy the application to Kubernetes (dev)
- Pipeline setup to deploy the application to Kubernetes (prod)
- ArgoCD workload files to manage the deployments (both dev and prod)
- Exposure to the outside world

In this experiment we have divided the groups into two categories: the one with no documentation or tutorial and the one with a tutorial to help you out. This is something that will be told before the first task.

The links required to get started are the following:

- backstage: this is the developer portal used to help you with certain tasks
- otomi: this is the developer portal used to navigate to the tools you need to use

The tools needed for this task are:

- Gitea: remote git
- DroneCI: CI platform
- Harbor: docker registry
- ArgoCD: GitOps CD tool for kubernetes
- Otomi services: expose applications

The credentials will be given in the google meet itself.

The goal of this task In this task you will need to get a react application to the development environment, you will end up with a url and on this url the site is served to the user. Example: https://example-service-prod.134.209.138.125.nip.io

Please ask questions if necessary.

D.6 Task three

In this third task you will be asked to retrieve the logs of the NestJS application you deployed in the task before.

For this situation we have enabled each nestjs application with a /error endpoint which will throw a http error message including a log and metrics. Your task is to query these logs and show these metrics.

The tools needed for this task are:

- Loki: query logs
- Grafana: dashboarding

We have created a predefined dashboard which can help you to show you the correct metrics: Custom: application metrics.

Appendix E

Usability study

E.1 Introduction form

For this experiment I want to validate a platform engineering implementation and see if this way we can improve the productivity by offering a platform to handle a lot of configurations.

In order to do this usability study I will ask you to perform one or two tasks which are difficult at first, but with the basic documentation I will provide it should be feasible within 30 minutes.

E.2 Roles

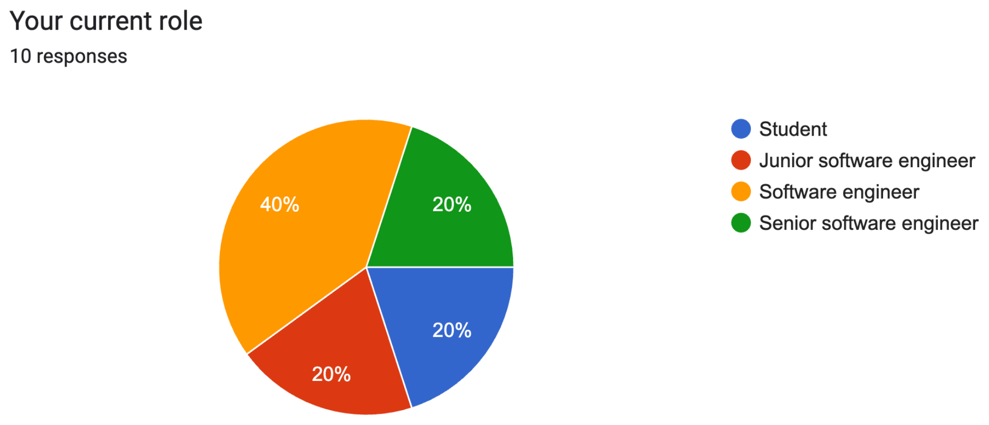


Figure E.1: Participant roles

E.3 Open questions

```
Learning curve of the platform
    1) With the docs, it was very easy to use.
    2) Since I did this task without tutorial I needed some time to click through some
       of the tools. But after I learned which tool did what it was extremely easy to
       create, build, deploy and expose an application.
    3) Pretty straightforward, although there is a lot of tools that are thrown at you
       in one go.
    4) Each service inside the platform is straightforward and is very easy to use. But
       you have to switch between different service within the platform, and they all
       have different UI.
    5) It was very easy to use. Makes this task much quicker. Only thing I did not
       really like was all the switching between tools/tabs. But this is to be expected
        when trying to use lots of tools.
    6) The links between the different apps used were not always as logical or easy to
       understand. However, with a little bit of documentation and an overview of how
       the tools work together I think the relations between the different apps will be
        easier to understand and will improve the process for a first time user.
    7) Easy to use, with basic explanation about some of the tools I was unfamiliar with
    8) Rather easy. I wont be struggling on yamls configurations
    9) Yes everything was easy to use and documentation was very helpful. Only negative
       was that many applications/sites were used and it could be simpeler if it was
       all in a single flow.
   10) For someone without much knowledge of the tools that are used on the platform,
       using the set of tools can be a bit daunting. However, the actual process of
       using the tools is quite straightforward since not many manual tasks need be
       performed by the user. Exposing the site to the world was the most complex step.
```

```
Which feature of the platform did you like most?
    1) Backstage
    2) Backstage golden paths
    3) Docs and golden paths
    4) Golden paths
    5) Observability. Logging and metrics worked out of the box.
    6) Golden paths
    7) The easy dashboarding and logging as this would be harder for me to do myself as
       I have no experience with it.
    8) Golden paths
    9) deployment process, auto yaml configuration
```

```
What improvements do you suggest for this platform?
    1) -
    2) Create golden paths that will help developers to migrate applications. Not only
       the creation of new applications, but also migrations
    3) To have it in one place.
    4) Maybe there is a way to merge the content from otomi and backstage together. So
       all the information is in one place.
    5) Documentation. See answer above
    6) Good documentation about workflow and basic information of tools used
    7) hard to say at moment I used it the first time, for now looks good to me
    8) Try to include as much in a single flow.
    9) Finding a way to let the user interact with all the tools without having to
       switch between interfaces would improve the process. Something such as a wiki
       that would explain all steps in detail would make the adoption of the platform
       easier.
```

```
Overall experience: Would you recommend using a platform in your daily tasks at your
    organization? What would be the main advantages of using this platform in comparison
     to no platform?
    1) Yes I would recommend using a platform. Because it greatly reduces the time
       needed to set everything up and troubleshoot errors.
    2) Yes. Using a platform eliminates a great deal of tasks that would normally have
       to be done manually. In a large organization that would also result in long
       waiting times for teams to resolve tickets.
    3) It can reduce deployment and maintenance part. It avoids developer to write
       different config files, but this also can be a reason the developers do not know
        how the platform works under the hood
    4) The big advantage is that it saves a lot of time setting everything up. Also
       easier for new people to use and get to know. Makes it also less error prone.
    5) Ease of use. I think the "T_shaped_developer" is a good idea, but it needs to be
       constricted to development and the tasks surrounding it. Creating and
       maintaining the entire process around the running state of an application should
        not be the main focus of a developer. I think having a dedicated platform
       engineering team makes more sense.
    6) Main advantages would be the reduction in time it would take for existing teams
       to create new applications conforming to the organisations standard, but an even
        bigger gain (I hypothesize) could be gained by allowing new teams to setup
       their applications according to organization standards.
    7) sure I would use it. main + of it - less manual work that can produce a lot of
       pain
    8) 8/10 The setup was easy and the tutorials gave a clear explanation of how
       everything was supposed to work.
    9) Overall yes, but it would depend on the type of organization. I have worked at
       smaller companies before where new pipelines would rarely be deployed as they
       had a set of small and stable services. Adopting the platform could in this case
        be too time consuming compared to the benefits. However, a large set of
       companies (which are often larger as well) frequently deploy new pipelines or
       perform related tasks. For these companies, I believe adopting a platform would
       yield many benefits in the long run thanks to the time it saves employees.

    The main advantage of using the platform is lessening the time employees have to
       spend on devops tasks and the fact that its standardizes the devops process
       within an organization.
```

```
General feedback: Is this what you expected from a platform?
    1) -
    2) Yes
    3) I did not have any expectation from the platform
    4) Yess, maybe a bit more than expected
    5) I expected the platform to bring ease of use so in that case it worked. However,
        it still requires the user to have an understanding of multiple tools. I think a
         tool like otomi already takes away a lot of the pains, but maybe having one
        tool in which you can setup an entire application without leaving the tool is,
        of course, most desirable. For example, if you look at other platforms like
        Vercel.
    6) I had little to no expectations about platform engineering
    sure. yaml-s to me is real place for a lot of mistakes. So this platform is very
        useful to avoid issues.
    7) yes, I did not know what to expect from platform engineering but was pleasantly
        surprised.
    8) It is not what I expected. My devops experience is not that deep, and imagining
        what to expect from a platform was therefore difficult for me. After using the
        platform, the benefits of it have become much more clear to me, and it has also
        given me a clearer view of what is meant by a platform.
```

E.4 Closed questions

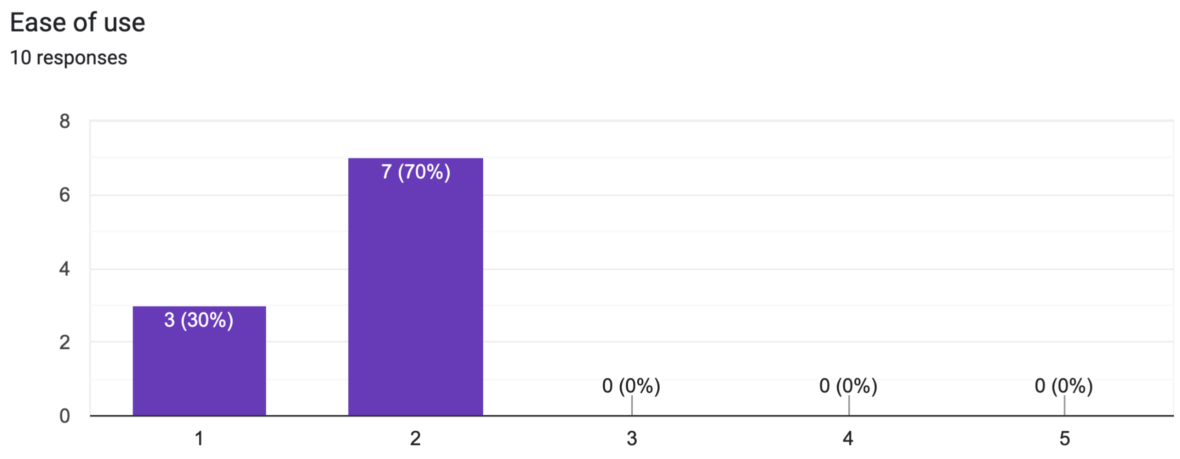


Figure E.2: Ease of use

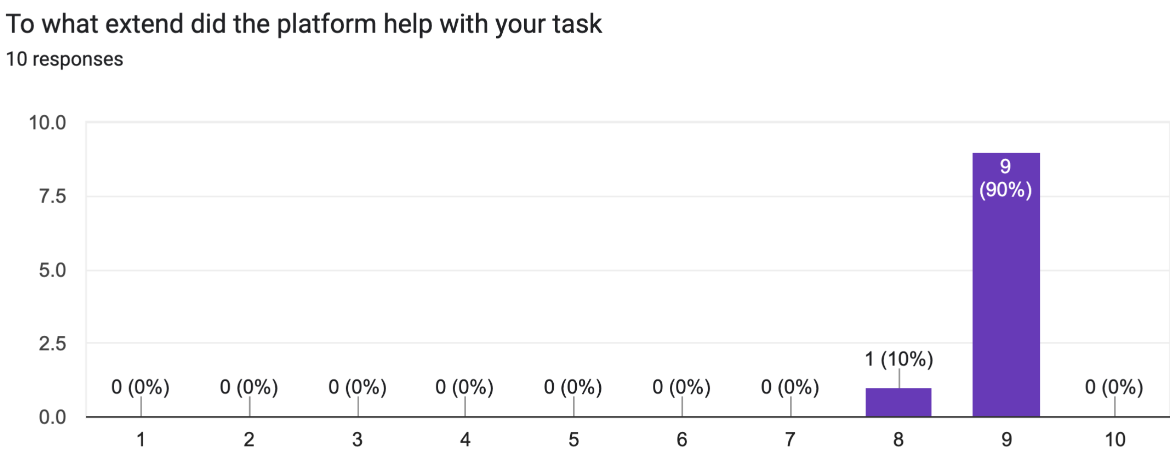


Figure E.3: Amount of support from the platform

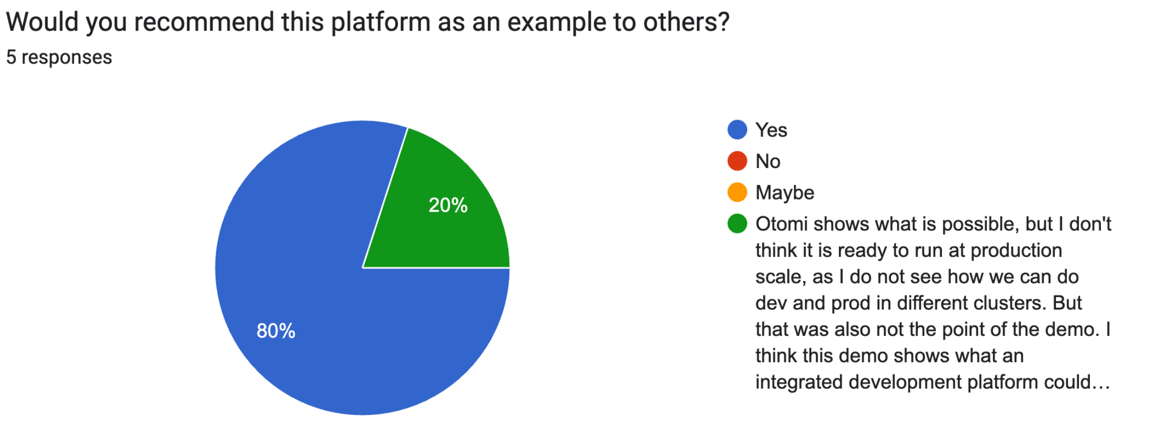


Figure E.4: Would you suggest platform engineering in your organization?

Appendix F

Platform expert feedback

F.1 Introduction text

In the past few weeks I have been working on a simple platform implementation with the help of Otomi, Backstage and other open source tools. This implementation is a compliment of the already created platform engineering reference model and can be used as an example on how you could implement a platform.

Before sending you this survey we had a discussion about the platform I have created, together with a demo of the platform. Based on this discussion and demo I would like to get some formal feedback on the implementation to see wether the platform is suitable or not.

This survey will take around 5 minutes to finish.

F.2 Roles

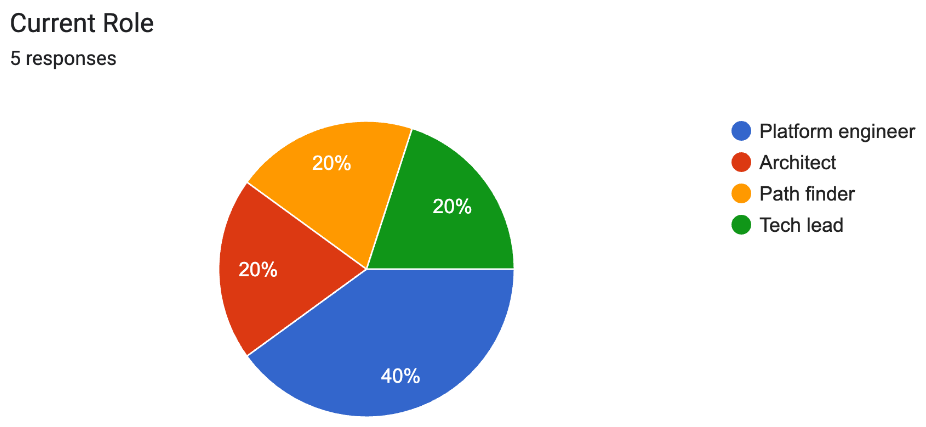


Figure F.1: Current role of experts

F.3 Open questions

```
Which features did you find most valuable?
    1) Catalog and Click-through detail views
    2) Backstage and the integration of all the components.
    3) The IDP, providing golden paths for the development of new applications,
       provisioning workloads, etc. All from a one-stop-shop. Very impressive and super
        useful. Specifically for the onboarding of new developers and getting people up
        to speed when switching teams, etc.
    4) Golden paths & documentation
    5) The "connectiveness" in the platform, everything is connected and by it, it helps
         the developer as it lessens the burden of getting applications ready for
         deployment. This gives the developer more time to focus on what he does best:
         solving business problems.
```

```
Are there features that you think should be added to the platform?
    1) Mesh and Trace correlation to graph views
    2) Policy agents (Gatekeeper) can be helpful to avoid mistakes. Enforcing guard
         rails can help the security and stability of the platform.
    3) Definitely the IDP, maybe even the PaaS aspect of your platform, because; why re-
         invent the wheel?
    4) Not right now
    5) More elaborate and integrated golden paths. Also looking for upgrade paths, as we
         now favor constructing new things over maintaining / upgrading the old one.
```

```
Does this implementation help you with your technical implementation at your
    organization? If yes, how?
    1) It would help for demonstration purposes, especially Backstage; but due to the
         IAM integration and HA constraints it would not work in this configuration out
         of the box.
    2) For Otomi, it is very good to have a showcase of what is possible and it is easy
         to PoC some tools. Using it as a platform could be more difficult because you
         have to "play" by their rules. You give away some flexibility (their pace of
         development is slower then from the components itself ) and tailor-made usage we
         now have (Integration between our IaC and Kubernetes clusters for example)

    I really liked the implementation and possibilities of Backstage, impressed by the
         demo and I am convinced we need to spend time to implement this tool in our
         platform.

    3) See previous answers regarding Backstage/IPD/PaaS.
    4) I think the work for the developers will be much easier this way. For the
         platform team managing and standardizing will take off. Also onbarding of new
         team members (or applications) will be much easier than it is now.
    5) It shows what can be done (het maakt je lekker). It shows the power. But it does
         not help us to actually get there, as the tools and integration differ per use
         case.
```

```
Additional comments or suggestions
    1) This is a strong demonstration due to the 'batteries␣included' approach instead
         of leaning on a specific service vendor. On the other hand, this has the
         downside of being reliant on a single VM image vendor, which abstracts away some
         of the component installation, which might be required knowledge during outages
         or platform debugging as well as migration to different platforms once a
         mismatch between requirements or constraints occurs.
    2) Very nice work! Keep it up. :)
    3) Great work. We have been talking about implementing something like this for a
         year now. Great to see a working example. Keep up the good work.
```

F.4 Closed questions

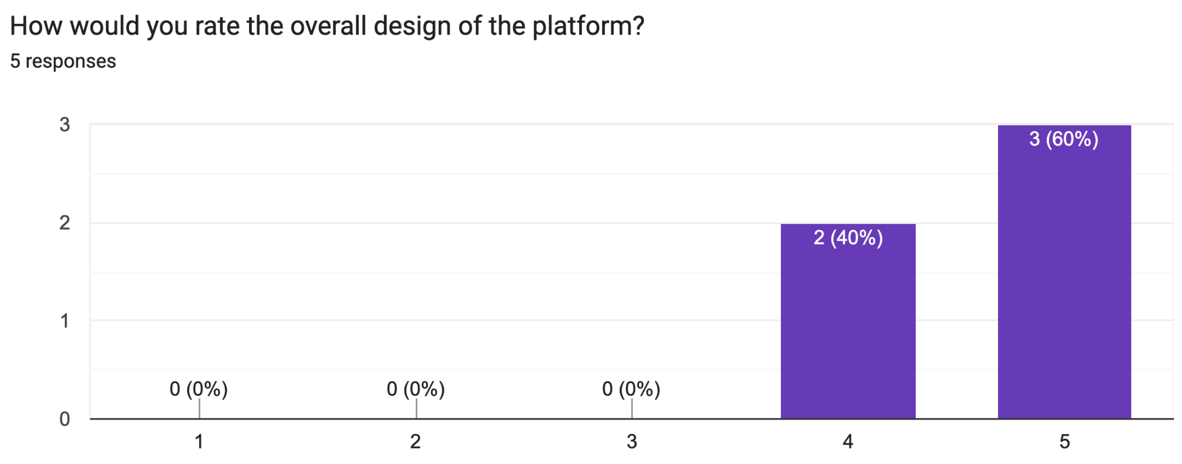


Figure F.2: Overall design of the platform

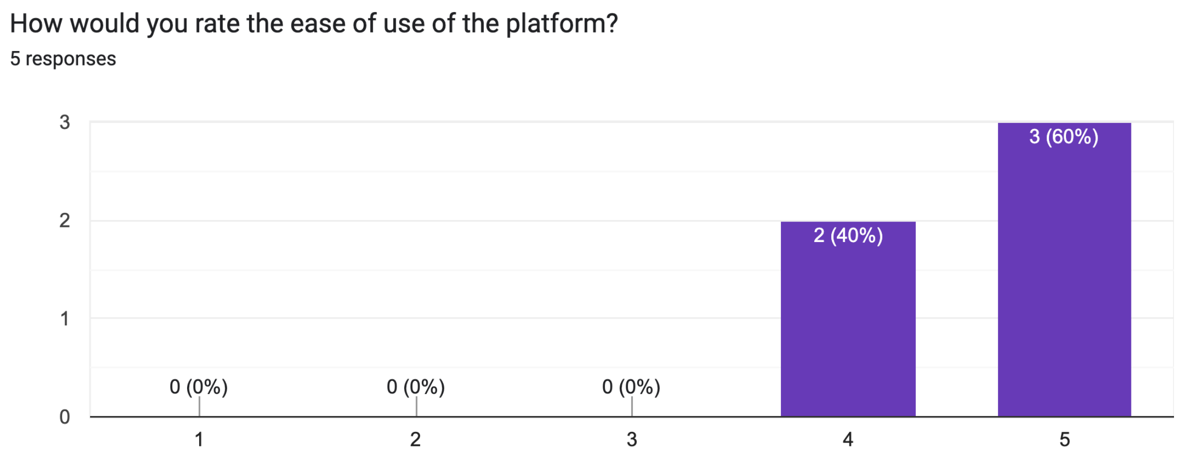


Figure F.3: Ease of use of the platform

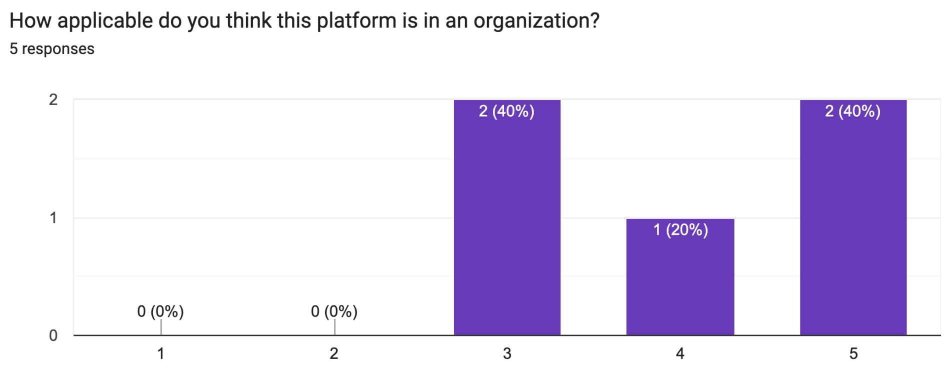


Figure F.4: Applicability of the platform in an organization

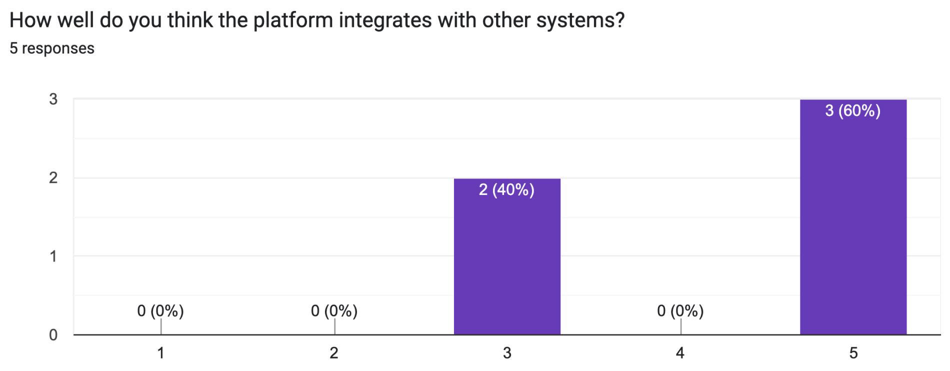


Figure F.5: Platform integrations with other systems

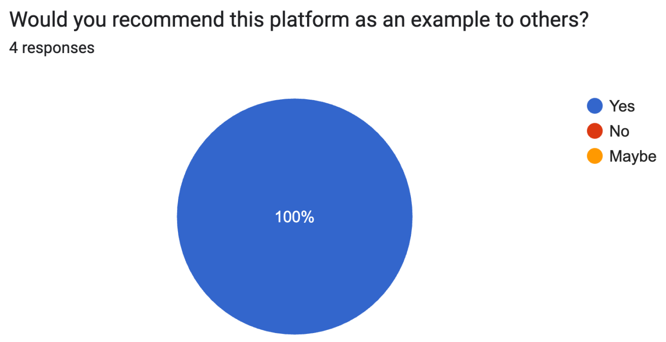


Figure F.6: Would you recommend this platform as an example to others?