Tampere University

Veikko Nieminen

INTERNAL DEVELOPER PLATFORM

Master of Science Thesis
Faculty of Information Technology and Communication Sciences
April 2024

ABSTRACT

Veikko Nieminen: Internal developer platform
Master of Science Thesis
Tampere University
Master's Degree Programme in information Technology
April 2024

In modern software development there is a high demand for better development workflows and faster delivery times. DevOps has emerged as a response to help create and unify ways of working to enable collaboration between different teams. Even still, modern cloud-native development requires increasing amount of different tools and methodologies to be able to create and publish applications. With the adoption of DevOps, every developer needs to learn all the necessary tools.

This thesis explores internal developer platform as a solution to the mentioned challenges. It integrates the development and operational tools into a unified system that simplifies software development processes. IDP should be thought of as a product that the developers are using, this way all the best product development processes can be utilized. Other key methods are promoting self-service and creating golden paths within the context. The process of creating an IDP is called platform engineering. The key benefits of internal developer platform include improved efficiency, reduced manual overhead and faster prototyping. Implementing IDP is complicated task that requires a dedication and good planning.

This thesis also presents a case study on potential benefits of implementing IDP in certain organization working in retail business. The organization in questions has some challenges utilizing DevOps ideologies and workflows, such as lack of visibility, onboarding challenges, no clear promoted ways of working, and others. Implementing IDP within the organization could solve some of these challenges and provide other benefits, but the task of implementation is technically challenging.

Keywords: internal developer platform, platform engineering, golden path, devops

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

TIIVISTELMÄ

Veikko Nieminen: Internal developer platform
Diplomityö
Tampereen yliopisto
Tietotekniikan DI-ohjelma
Huhtikuu 2024

Modernissa ohjelmistokehityksessä on jatkuva tarve paremmille ohjelmistokehitysprosesseille ja nopeammille toimitusajoille. DevOps ideologia on noussut ratkaisuksi tähän ongelmaan, auttamalla luomaan ja yhtenäistämään työskententelytapoja sekä parantamaan yhteistyötä eri tiimien välillä. Moderni pilvipalveluihin pohjautuva ohjelmistokehitys kuitenkin vaatii kasvavan määrän erilaisia työkaluja ja erilaisten työskententelytapojen käyttämistä. DevOps ideologian myötä kaikkien kehittäjien on osattava käyttää kaikkia käytössä olevia työkaluja.

Tämä työ tutkii sisäistä kehittäjien alustaa (internal developer platform) ratkaisuna ylläoleviin ongelmiin. Alustan ideana on yhdistää kehitystyössä ja tuotantotoimissa käytettävät työkalut yhdeksi järjestelmäksi, jonka avulla voidaan yksinkertaistaa ohjelmistokehitysprosessia. Alustaa tulisi ajatella tuotteena, jolloin voidaan käyttää samoja tapoja kuin ohjelmistojen tuotekehityksessä. Muita tärkeitä tapoja on itsepalveluun kannustaminen ja kultaisten polkujen (golden path) luominen. Alustan suurimmat hyödyt ovat parantunut tehokkuus, toimintojen automatisointi sekä nopeampien prototyyppien mahdollistaminen. Alustan kehitys on kuitenkin monimutkainen prosessi, joka vaatii omistautumista ja tarkkaa suunnittelua.

Tässä työssä esitetään myös tapaustutkimus sisäisen kehittäjien alustan kehittämisen mahdollisista hyödyistä tietyssä jälleenmyyntiin keskittyneessä organisaatiossa. Kyseisessä organisaatiossa on joitakin haasteita DevOps ideologian käytössä, kuten huono tiimien välinen näkyvyys, perehdyttämisen haasteet, suositeltujen työskentelytapojen puuttuminen sekä muita ongelmia. Alustan luominen organisaatiossa voisi ratkaista joitakin näistä ongelmista ja voisi tuoda muita hyötyjä, mutta alustan luominen on teknisesti hankalaa.

Avainsanat: internal developer platform, platform engineering, golden path, devops

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -ohjelmalla.

PREFACE

I would like to thank Kari Systä for the initial idea for the thesis and for the fast and great feedback that I have received along the process. I would also like to thank the anonymous organization for acceptance of the study and constant support during this process. Thank you for many inspiring conversations about the topic.

I want to say special thanks to my wife, family and friends for supporting and believing in me during my studies.

Tampere, 22nd April 2024

Veikko Nieminen

USAGE OF AI-BASED TOOLS

The AI tools used in my thesis and the purpose of their use has been described below:

Name of the tool (and version): ChatGPT (version 3.5 and 4.0)

Purpose of use and the part in which it was used: Tool was used in this thesis as a spellchecking and editing tool to improve the structure of the language.

I am aware that I am totally responsible for the entire content of the thesis, including the parts generated by AI, and accept the responsibility for any violations of the ethical standards of publications

CONTENTS

1. Introduction . . . 1
2. DevOps . . . 2
2.1 Principles . . . 2
2.2 Ways of working . . . 4
2.2.1 Continuous integration and delivery . . . 5
2.2.2 Microservices architecture . . . 6
2.2.3 Infrastructure as code . . . 7
2.2.4 GitOps . . . 7
2.3 Common challenges when using DevOps . . . 8
3. Internal developer platform . . . 10
3.1 Definition . . . 10
3.2 Product mindset . . . 11
3.3 Benefits . . . 11
3.4 Golden paths . . . 12
3.5 Core components . . . 13
3.6 Implementation . . . 14
3.7 Solutions . . . 15
3.7.1 Backstage . . . 15
3.7.2 Humanitech products . . . 17
4. IDP benefits in theory: A case study . . . 20
4.1 Organization overview and contextual challenges . . . 20
4.2 Application of IDP in organization . . . 22
4.3 Anticipated benefits . . . 22
4.3.1 Visibility . . . 23
4.3.2 Golden paths . . . 23
4.3.3 General benefits . . . 23
4.4 Implementation barriers . . . 24
4.5 Plan for implementation . . . 24
5. Discussion . . . 26
5.1 Limitations and validity assessment . . . 26
5.2 Future work . . . 26
6. Conclusion . . . 28
References . . . 29

GLOSSARY

API	Application programming interface
AWS	Amazon Web Services
CD	Continuous Delivery
CI	Continuous Integration
CLI	Command-Line Interface
CNCF	Cloud Native Computing Foundation
DevOps	Development and Operations
IaC	Infrastructure as code
IDP	Internal Developer Platform
MVP	Minimum Viable Product
RBAC	Role-Based Access Control
REST	Representational state transfer
SSO	Single Sign-On

1. INTRODUCTION

In today's rapidly evolving technological landscape, organizations face significant challenges with establishing software development workflows. The increasing demand for faster delivery times, in addition with the need for reliable operations, puts increased pressure on companies to innovate their development workflows.

DevOps has emerged as a response to these challenges, integrating software development and operations to increase agility and reliability. By adopting a culture of collaboration between previously siloed teams, DevOps practices help reduce the time-to-market for new features and ensure higher service quality [1]. DevOps provides a lot of value, but some organizations still have problems adpoting DevOps practices. Common challenges while implementing DevOps practices are missing clear defition and steps to follow, and high demand for skilled staff. Another common painpoint is that the automations required in operations and monitoring are usually insufficient.

Internal Developer Platform (IDP) presents an innovative approach to further streamline the software development lifecycle. IDPs provide a structured system that centralizes access to different tooling, reduces the cognitive load on developers, and automates complex operational tasks, increasing efficiency and enabling developer autonomy. [2]

This thesis explores what is IDP and how it can benefit the organizations that adopt it. To understand where the need for this kind of idea has emerged, brief introduction to DevOps and its methodologies is necessary. Thesis also covers some known challenges faced when organizations are adopting DevOps. To better understand the idea, a case study was performed on an organization that has challenges with the existing development workflows. The case study examines the potential benefits that implementation of IDP could provide to the organization. The original thesis topic was provided by the examiner.

The thesis is structured as follows: Chapter 2 describes DevOps and its key principles. It also discusses some common problems with utilizing DevOps. Chapter 3 explains what the term IDP means and how one can be implemented. Chapter 4 presents a case study on organization that has some problems with current workflows and how IDP could solve those problems. Chapter 5 has discussion of the limitations and the possible future works on this topic. Chapter 6 has the conclusion for the thesis.

2. DEVOPS

Development and Operations (DevOps) is a set of practices that aims to streamline the software development, release, and operations processes. The concept of DevOps is driven by the need to reduce the time it takes to deploy changes to a system while ensuring high quality. This approach recognizes that inefficiencies in these processes are often caused by an organizational split between development and operations teams. DevOps promotes a collaborative approach that focuses on communication and integration between these teams to achieve faster and more reliable software delivery. By breaking down silos and fostering a culture of collaboration, DevOps enables organizations to overcome the challenges associated with traditional development and operations models and deliver high-quality software faster and more efficiently. [1, 3, 4, 5, 6]

DevOps is a relatively new approach that has emerged in response to the challenges faced in modern software development and operations processes. As a result of its rapid evolution, there is still much debate and discussion surrounding the actual definition of DevOps. Some argue that it is a philosophy that emphasizes collaboration, automation, and continuous improvement [5], while others view it as a set of practices that focus on integrating development and operations teams to streamline the software delivery pipeline [1, 3]. Despite the lack of a clear and consistent definition, DevOps has gained widespread acceptance as a way to address the inefficiencies and bottlenecks that often arise in software development and operations. [3]

DevOps can provide several benefits to organizations, including improved software quality, faster time-to-market, increased agility, and better customer satisfaction. By emphasizing collaboration and testing, DevOps enables organizations to produce higher-quality software with fewer defects, leading to cost reductions and increased customer satisfaction. Additionally, the focus on automation and streamlined processes results in faster time-to-market and increased agility, allowing organizations to respond quickly to changing market conditions and customer needs. [6, 7]

2.1 Principles

While there is no single, definitive set of principles for DevOps, there are several commonly accepted principles that guide organizations in their DevOps initiatives [3]. These

include building a culture of collaboration and communication between development and operations teams, using automation to streamline and accelerate software delivery processes, and implementing continuous testing and monitoring to ensure high-quality software. Other key principles of DevOps include breaking down silos and promoting cross-functional teams, embracing change and experimentation, and focusing on the end-to-end customer experience. By following these principles, organizations can achieve greater agility, faster time-to-market, and improved software quality, ultimately driving greater value for their customers. [1, 3]

Gall and Pigni propose conceptual framework for DevOps which consist of three main categories, continuous collaboration, continuous automation and continuous monitoring [3]. This framework will be used as a source for DevOps principles in this thesis with support from other sources. The main principles have been pictured in the figure 2.1.

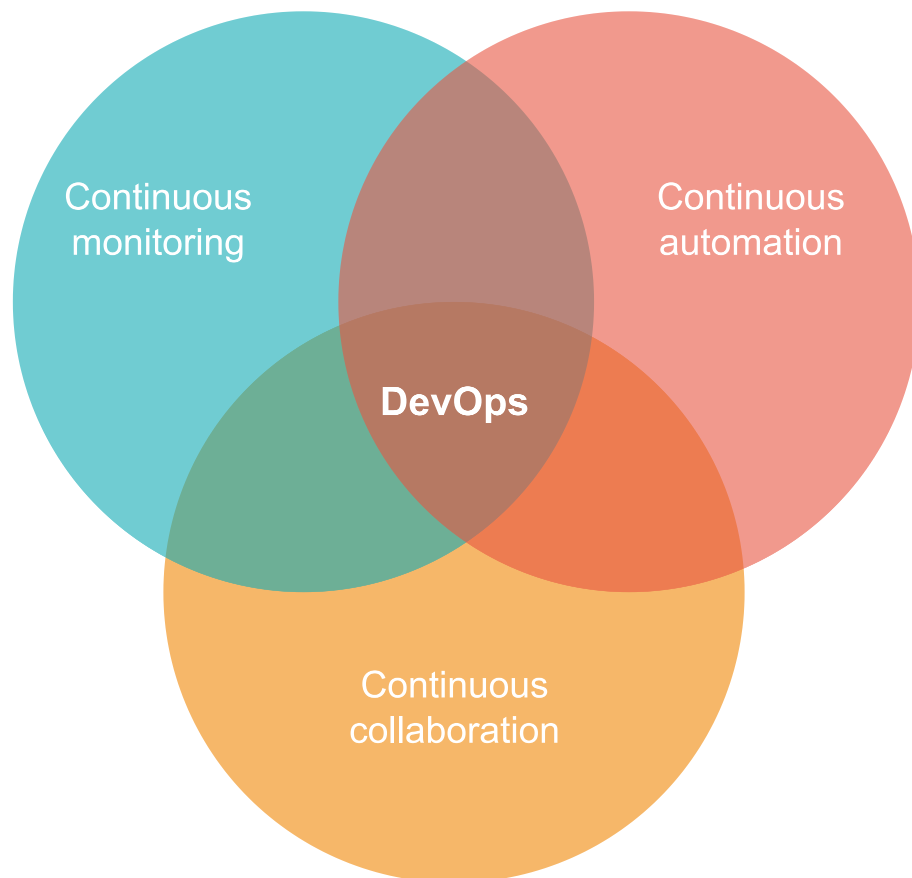

Figure 2.1. *Key principles of DevOps. Adapted from [3]*

Continuous collaboration is the first principle of DevOps that Gall and Pigni present, focusing on the need for co-operation and teamwork between developers and operations teams [3]. Collaboration promotes a culture of shared responsibility and accountability, breaking down silos and promoting cross-functional communication [3, 7]. It encourages the adoption of agile methodologies, where feedback is incorporated throughout the de-

velopment process, promoting continuous improvement. Collaboration also requires the use of tools and technologies that enable seamless collaboration, such as version control systems, Continuous Integration (CI) and Continuous Delivery (CD) pipelines, and agile project management tools.

Second principle is continuous automation. It involves automating manual processes to increase efficiency, reduce errors, and eliminate repetitive tasks. Automation plays a vital role in achieving fast feedback and fail-fast approaches [3]. Fail-fast is the principle of the earlier the bug or problem is noticed, the cheaper it is to fix. Through CI and CD, developers can quickly detect and fix issues in the application, reducing the risk of defects in production. Testing is also an essential aspect of automation, allowing for extensive testing throughout the development cycle. The deployment process can also be automated, resulting in faster and more reliable releases. [1, 7]

Final principle that Gall and Pigni present is continuous monitoring. Continuous monitoring is an essential practice in the DevOps methodology that involves the continuous gathering and observation of metrics throughout the software development and operation lifecycle [3]. Metrics provide insights into the performance and health of the software system and enable teams to detect and resolve issues proactively. Transparency is a crucial aspect of monitoring while it ensures that all team members have access to the same information and can collaborate effectively to maintain the system's health. [7]

2.2 Ways of working

The field of software development has evolved over time, the need for more efficient and effective ways of working has become more important, and the need for collaboration between teams has become apparent. Organizations are now focused on delivering high-quality software quickly and continuously while also managing complexities related to rapid growth and change. The core concept of DevOps revolves around a set of practices and tools that enable development and operations teams to work together more closely, thereby enhancing productivity, reducing time to market, and improving overall software quality. [1]

However, achieving the desired level of collaboration and efficiency in DevOps requires adopting new ways of working. These working methods are designed to facilitate the seamless workflow across the entire software development lifecycle, from planning and coding to deployment and monitoring. This section will dive into four key ways of working that have proven to be in key roles in the successful implementation of DevOps: CI and CD, Microservices, Infrastructure as code (IaC) and GitOps.

These ways of working are not exclusive to DevOps but have been widely embraced by the software development community due to their ability to streamline processes, improve

collaboration, and increase the quality of delivered software. Instead, they complement each other and, when used together, can significantly enhance the overall effectiveness of a DevOps environment. By understanding and implementing these practices, organizations can unlock the full potential of DevOps and create a more agile, flexible, and resilient software development environment. It is important to note that these practives are not always the correct solution, they can be adapted and modified to fit the specific needs and requirements of a given organization or project. [1]

2.2.1 Continuous integration and delivery

Continuous Integration (CI) and Continuous Delivery (CD) are closely related but distinct practices that play a critical role in the successful implementation of DevOps principles [1]. While they share common goals, CD extends CI by automating further stages of the deployment process. CI involves frequently integrating code changes into a shared repository and automatically testing those changes, while CD focuses on automatically deploying those changes to production in a safe and efficient way. Together, CI and CD enables rapid iteration and experimentation with fast and easy way of deploying, which is essential for successful microservices architecture. [8]

To implement CD, organizations must first establish a strong foundation of CI. This involves using a shared and centralized repository where an automation is set up which automatically test and validates all the changes. Usually a successful CI run is needed before changes can be merged to main branch. CI is usually thought of as a pipeline that consists of different steps or jobs. These steps can include static file checks, unit or integration tests or end-to-end tests. Ideally the result of the pipeline tells if the code functions properly and if the code is up to set standards. [1] The pipeline should be designed to provide feedback as quickly as possible, as catching errors early in the process reduces the time and effort required to fix them [8].

CD is the process of automatically deploying certain changes to production. Traditionally the process of deploying has been rare and slow process. CD aims to make the process easier and reproducible. CD usually consists of steps like build, verify and deploy. Actual implementation of deploy scripts depends highly on the used technologies and environments. In some cases the deploy can be as simple as just copying the files to certain location, on other occasions, there can be a multistep build process and complicated deployments required. [1]

Implementing effective CI and CD pipelines requires a focus on consistency and continuous improvement. The team needs to agree on common rules that the pipeline will enforce. It should be possible to modify the pipelines with short notice to adapt to changes in business logic. [8] Ideally CI and CD system should provide a possibility to verify and deploy any changes as the developers wants. The verifying stages should tell if the

changes would cause problems and if so, give feedback to the developer. With adequate monitoring and observability, a problem in deployments can be noticed and if need be, a rollback to known working version should be possible with short notice [1].

2.2.2 Microservices architecture

Microservices architecture is nowadays quite common approach for creating software that focuses the use of small, separately deployable services that work together to form a larger application. Microservices are often organized around business capabilities rather than technical concerns, and are designed to be easy to change and maintain. One of the key benefits of microservices architecture is improved scalability. Each microservice can be scaled independently of the others, which allows for better utilization of resources and can result in lower costs. Another benefit is better fault isolation, as failures in one microservice are less likely to impact the rest of the system. Microservices can also enable faster release cycles, when changes to individual services can be deployed independently. [8, 9]

Microservices also introduce new challenges. One of the biggest challenges is increased complexity when managing a large number of services. Distributed system management becomes more complex because communication between services needs to be carefully managed. Additionally, each microservice needs to have a well-defined responsibility and a clear service boundary. Communication between microservices should be based on lightweight protocols such as REST or messaging, and each microservice should have its own data store, with data accessible only through the service's API. Successful management of a microservices architecture requires deployment automation and testing. [9] Automation is critical for ensuring that each microservice is deployed consistently, and changes to individual services do not negatively impact the rest of the system. Testing is also essential because each microservice must be tested thoroughly before deployment to ensure that it works correctly and does not introduce regressions [9].

Organizational considerations are also important for the success of microservice architecture. Team structure and communication patterns need to be carefully considered, as a culture of collaboration and continuous improvement is necessary to adapt to changing requirements and technologies. It is also important to consider the trade-offs and challenges of microservices architecture, and it may not be the best approach for every application or organization [9]. To address some of the challenges of microservices architecture, various patterns and techniques have emerged. API gateways can provide a single entry point for all client requests, and can perform functions such as authentication and rate limiting. Service discovery can be used to automatically locate and connect to other services. Circuit breakers can be used to prevent cascading failures by isolating failing services. [8]

2.2.3 Infrastructure as code

Infrastructure as code (IaC) is an approach for managing infrastructure components and configurations as code. This allows the code and also the infrastructure to be controlled, tested, and deployed in the same way as software code [10, 11]. IaC provides several benefits, including improved consistency, increased visibility, reproducibility, and scalability. IaC also makes it easier to manage complex environments and to make changes to infrastructure while maintaining consistency and reliability. Several IaC tools are available for implementation, for example Puppet, Chef, and Ansible. In cloud-based environments, tools such as AWS CloudFormation, Terraform, and Ansible can be used to manage infrastructure. [10]

IaC should be treated as software development and best practices in development should be followed. Otherwise, changes to the infrastructure could introduce errors and inconsistencies that are difficult to detect and fix. [1]. In IaC, it's good to adopt modular design for the code because that allows the use of shared code that enables easier changes and updates. Instead of changing code for all services, one can just update the shared code and all services are updated. [11]

2.2.4 GitOps

GitOps is a development and operations methodology that utilizes version management system (usually Git) as the primary source of truth for declarative infrastructure [12]. The term was introduced by Weaveworks in 2017 as a way to manage Kubernetes clusters. Since then, the term has gained popularity as a methodology for managing complex distributed systems at scale, including cloud-native applications and microservices [13]. Simplified GitOps process described in figure 2.2. Change management presents change requests, rollbacks and other version control concepts.

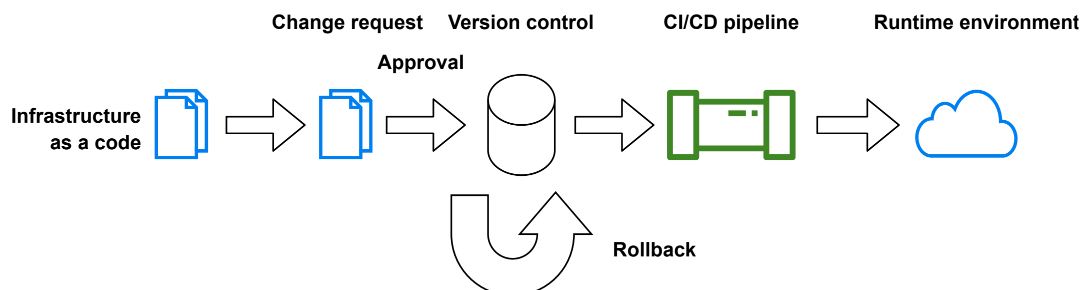

Figure 2.2. *High level overview of GitOps process [14]*

In GitOps, changes to infrastructure and applications are managed using Git best practices, which provides a structured approach to managing changes. These structures

are branches, tags and change requests with approvals. This ensures that changes are checked and approved by the proper authority before the changes are applied [15]. Automated checks can be performed on changes to ensure that they comply with predefined rules and standards, such as linting and security checks, before they are merged into the repository. Once changes are merged, automated delivery pipelines are triggered to deploy the changes to the target environment, ensuring that the actual state of the infrastructure matches the desired state specified in the Git repository. This process is observable, with monitoring and logging tools used to provide real-time feedback on the state of the infrastructure and to identify issues as they arise. These automated checks and deployments help to ensure that changes are consistent, repeatable, and auditable, promoting a culture of collaboration and transparency among team members. [13]

Benefits of the GitOps are increased speed and efficiency, greater consistency and reliability, improved collaboration, enhanced observability and it might also provide better security and compliance [13]. Collaboration is promoted through code reviews and change requests, with real-time feedback provided through monitoring and logging tools. Access controls and audit logs help to ensure compliance with security policies and regulations, and provide a record of all changes made to the infrastructure. [12]

Implementing GitOps does provide some challenges that users should consider before adopting it. These challenges include the complexity of managing multiple layers of infrastructure, the dependency on external services, and version control conflicts. Overreliance on automation can make it difficult to troubleshoot issues. Adopting GitOps requires a cultural shift towards a more collaborative and transparent approach. To mitigate these challenges, teams should establish clear ownership and review processes for changes, minimize external dependencies, and provide sufficient training and support to team members. [15]

2.3 Common challenges when using DevOps

Introducing DevOps into an organization comes with a wide range of challenges. These include missing clear steps to follow, problems with setting up automation and dealing with the high demand for skilled staff. There are often issues with managing projects and resources, improving how teams communicate and work together, and making the big cultural changes needed for succesful implementation.

The process of adopting DevOps in organization is often restricted by the lack of explicit and commonly accepted guidelines. This complicates process, making it unclear what practices to promote and how to transform the current workflows [6]. Every organization has different structure and workflows, making creation of general guidelines hard. Additionally, the lack of clear guidelines makes it hard to assess the quality increase after DevOps has been implemented.

Common measuring target is the frequency of deployments to production, but that is only one metric that does not represent the whole quality. [6] Adopting DevOps involves adopting new approach to software development flows that demands a wide range of skills from developers. Suddenly all developers need to be able to manage systems and to use specific tools and practices that are part of DevOps culture. This includes being able to automate various stages of development, testing and deployment to make the processes faster and more efficient. [4] In studies, it has been shown that many organizations lack in the automation part of the DevOps implementation [4, 6, 7]. Another common problem in the implementation is difficulties in monitoring [4, 6].

For the DevOps adoption to be successful, a significant cultural shift within organization needs to happen, moving away from traditional siloed roles to a more collaborative and integrated approach. This transformation can be challenging as it requires changes in mindset, processes, and practices across all levels of the organization. Projects often face constraints in terms of available resources and tight timelines, making it difficult to fully embrace and implement DevOps methodologies. These limitations can impede the progress and effectiveness of the initiatives. [1, 4, 7]

The need for seamless collaboration and communication between different departments, such as development, operations, and quality assurance, can be a challenge to establish and maintain. This is increased by existing communication barriers and the adjustment to new ways of working together. [4, 7]

3. INTERNAL DEVELOPER PLATFORM

Internal Developer Platform (IDP) is a tool designed to ease developer workflows with self-service capabilities [2]. It consists of a collection of different tools to create a singular system that should be easy to interact with. As internal infrastructures differ from organization to organization, the implementation of IDP varies a lot. There are a lot of tools emerging to support creation of such platforms.

The first sections presents the idea of IDP from different perspectives and describes some of the benefits that it can provide. Section 3.6 provides some common suggestions for implementation and the last section presents some existing solutions, both open-source and paid services.

3.1 Definition

Internal Developer Platform (IDP) is a collection of different tools and processes aimed to streamline and make it easier to handle software development lifecycle. It's designed to minimize the complexities that developers need to handle when utilizing DevOps ideologies, enabling developers to focus more on the business logic. An IDP serves as a way of working designed to enable developer self-service, simplifying and unifying operation related workflows. IDP should not be viewed as a yet another monolithic tool, but more of a collection of all the existing tools that the developer teams are already using. [2, 16, 17] The act of implementing these tools is called platform engineering [18].

IDP aims to reduce cognitive load of the product development team. Instead of every developer learning Terraform or Kubernetes helm charts, they can use tools to do all the operations tasks in self-service manner. [16, 19] The interface for managing and handling everything inside IDP is known as control plane. Common ways of implementing control plane is as a web application that users can access. Other common interfaces are Command-Line Interface (CLI), an API or even a chatbot. The main criteria for interface selection is ease of use. [2, 16]

IDP should provide a transparent abstraction over the existing infrastructure. Dev teams should be able to customize the underlying infrastructure if needed. Also, the adoption of IDP should be voluntary and promoted with the idea of making devs tasks easier. [19]

IDP is not suitable for every use case. For smaller organizations, it might be better to use software development PaaS (Platform as a service) instead. One well known software development PaaS is Heroku [16]. Implementing your own IDP might come into picture if the PaaS does not fit anymore in the organizational use cases. IDP is more customizable and flexible, but takes serious effort and dedicated team to implement [19].

3.2 Product mindset

IDP should be considered as a product with developers as the end user. This way all the well established best practices in product development can be applied to platform engineering. End users should provide insights for additional features and improvements for the platform. [16, 20] Strong collaboration between the platform team and the developers is required for effective implementation.

When IDP is treated as a product, its reliability should also be taken into account. It is a product that customers are using, and as such, it should be reliable, stable and usable. [19] The platform should have sufficient analytics and monitoring in place, to act on downtime and to gather usage analytics for improvements.

3.3 Benefits

Well implemented platform decreases the cognitive load for implementing applications. When IDP has templates for different types of products, creating a new service can be just a one action job, instead of spending days on setting up infrastructure, monitoring and analytic frameworks. [18, 19] This way developers suffer less from context switching, when they don't need to focus on the details of the infrastructure. Suddenly writing terraform or Kubernetes helm charts can be challenging for some developers.

IDP enables self-service. This means that developers can do certain predefined actions on their own without depending on other personnel to do things for them. This enables the operational people to focus more on developing the infrastructure, instead of doing tickets for other developers. With predefined actions, it's almost mandatory to follow the best practices set up in the platform. These functions makes creating new services easier and faster, which enables faster innovation. [2, 16, 21]

Some IDPs evolve from a service catalog. It is a tool to provide documentation of the different services that already exists within the organization. It can provide basic information of the service, it's responsible team and the dependencies it has for the different services. On some cases it can even include API documentation, and it can serve as the main place for documentation.

3.4 Golden paths

In development, there are many decisions that need to be made regarding the implementation of the application. How to structure code, which tools to use and how to use them in the most efficient way. Golden paths is a term offering a solution to this problem. [18] It presents a set of templates, standards and well known best practices to follow. Golden paths are strongly opinionated, and reflect the organizational decisions on infrastructure and preferences on tools and methodologies. [19]

IDP should support and promote following the defined golden paths [16]. Automations and actions should follow these chosen best practices. When developers follow golden paths, the different applications have the same decisions, making the change of projects easier. This also simplifies implementing the automations when projects are uniform. This way security administrators can implement policy guide rails easily and even complex task of optimizing cloud usage can become easier. [18]

There should be different golden paths for different use cases and different application stacks. High level overview of the development workflow with IDP in figure 3.1

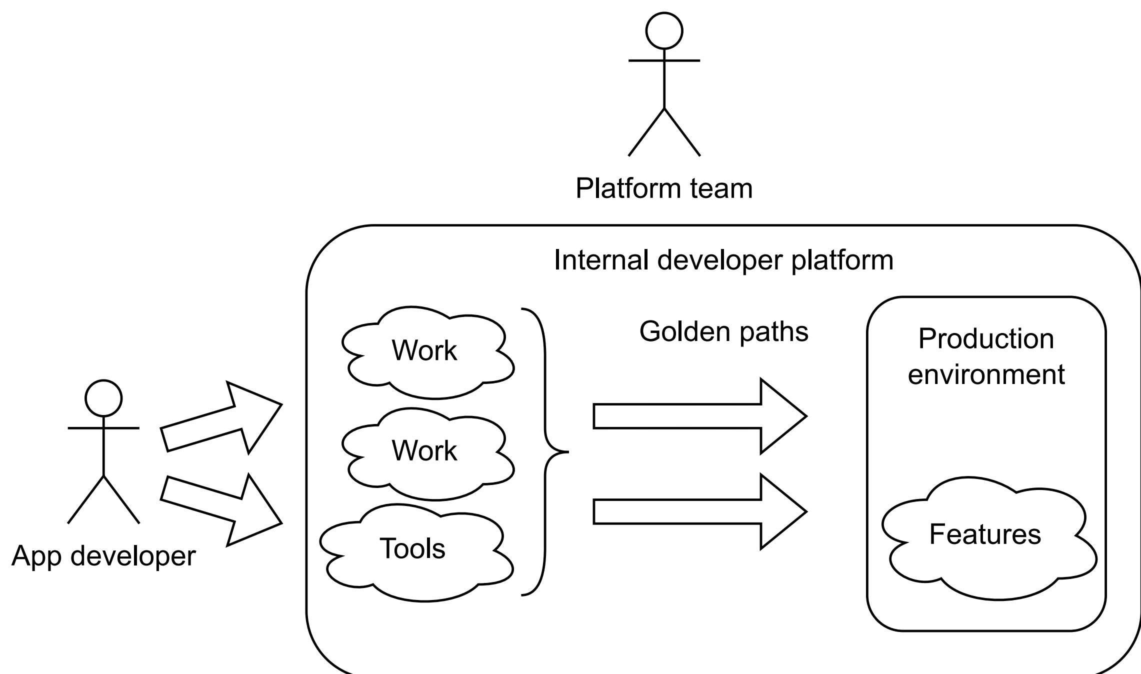

Figure 3.1. *Overview of IDP*

In the figure above, platform team is managing the whole IDP. App developer follows predefined golden paths to make changes to the production environment. In IDP they can also access other tools such as monitoring or service catalog.

3.5 Core components

One definition of IDP is done by defining the core components that it should have. According to source [2], essential components of IDP are:

- **Application configuration management**: Managing application settings and configurations involves handling settings specific to applications and their operating environments. While managing a few settings manually is feasible, complexity increases with dynamic configurations, such as feature flags. Challenges like version control and secret handling also complicate the setup even more. An IDP simplifies this by offering a centralized system for managing all configuration needs, ensuring flexibility and security.
- **Infrastructure orchestration**: Ideally IDP should take charge of setting up and managing the infrastructure that applications run on. This includes everything from cloud services to physical servers. The goal is to make managing this infrastructure straightforward, enabling developers to easily set up and adjust their operational environments without getting slowed down by technical details.
- **Environment management**: With IDPs, developers can quickly create and manage their environments for each service. This self-service approach reduces delays, as developers don't need to wait for someone else to set up their environments for them.
- **Deployment management**: IDP should enhance continuous deployment by offering automation, easy rollbacks, and robust troubleshooting support. These features ensure smoother, more reliable updates with less risk, and better experience compared to basic deployment tools. IDPs help deployment management by streamlining processes and increasing team capabilities, making continuous delivery more efficient and safe.
- **Role-Based Access Control (RBAC)**: IDPs use RBAC to define what each user can and cannot do within the platform, based on their role. This ensures that team members have the access they need to be productive while maintaining security and control. For instance, a service owners might have extensive access compared to others who only need to view basic information about the service.

While implementing a platform with all these components is already a very complete IDP, there are other aspects also to compliment the developers experience within the workflows. For example, a service catalog feature can be simple to setup and can bring a lot of value for users by increasing visibility and knowledge. [2]

3.6 Implementation

Implementing IDP is not an easy task, one needs to take many different use cases and users into account. The technical requirements for implementation can vary a lot, but generally they are extensive. Managing a lot of different technologies is necessary when connecting different systems and services together.

When designing IDP, a lot of different things need to be taken into account to make it user-friendly. First and upmost, the platform should be easy to use. This includes having clear instructions, simple interfaces, and a design that makes sense for the tasks developers use the platform for. It should help them do their work without causing confusion or requiring a lot of extra learning. Other factor to keep in mind is the purpose of the platform. It should be built for the job that it is meant for. It should offer the tools and features that developers actually need to complete their projects. This could mean anything from making it easier to initialize new projects to integrating with other services within the context. [2, 19]

Another factor is the reliability and stability of the platform. Developers should be able to count on the platform to be available and working correctly whenever they need it. If the platform is often down or not working correctly, it can slow down their work and be a source of frustration. When technology changes, the platform should be able to adapt without making things difficult for its users. This means it should be able to handle more users or new kinds of projects as the organization grows and changes. [19]

Implementation of IDP should start small. Common place to start is by automating the project creation workflow as that usually has clear workflow to be followed. Finding and automating the most common part of the developer workflow can provide the best value to the end users [2]. These automations and support functions should be implemented well and should maintain certain quality to give developers the best experience.

A common trend is to have a full team dedicated in developing and maintaining the platform. This team makes platform-wide decisions with collaboration with the development teams, cloud provider experts and operations teams to create the tools used in the development workflows [17]. Ideally the team should regard themselves as service provider and focus purely on the platform development. In large organizations there can be multiple teams working on different parts of the platform. [19]

Adopting new technologies is generally slow. Introducing IDP can also be slow and face some resistance. [19] These problems can be mitigated by establishing a fast feedback loop with the users. Fast feedback loops with the users of the platform can help to determine if the technologies presented are beneficial or not. The feedback is crucial also for general platform development to understand the use cases and the pain points [16]. With the whole platform and with new features, the platform team should use their own tools

also themselves.

Many of the measurable benefits of the IDP relate to the actions it can perform and the ways it can benefit service management and as such the benefits are more prominent when IDP is used with microservices architecture. In contrast, its advantages might not be fully realized with a monolithic application, as the amount of actions within the IDP would be less common.

If organization has already established some IaC practices and uses containerized deployments or even Kubernetes, the IDP implementation will be easier. Many of the existing products for IDP prefer Kubernetes as the runner platform as it already provides a layer of abstraction. [2, 17]

IDP implementations vary a lot from organization to organization because the operational contexts are vastly different. As such, the best practices for implementation also vary a lot. In the following subsections , some known solutions are presented.

3.7 Solutions

There are many different tools available to create IDPs, for example, Massdriver, Cortex, Port, Compass and Crossplane This section presents two most common tools used, Backstage and Humanitech products. Tools designed for other relating technologies are also in key roles when doing the implementation. [2]

The optimal tools depend highly on the existing infrastrucure and the existing tools that are already in use. In depth research should be done before deciding on the best tools and technologies for IDP implementation.

3.7.1 Backstage

Backstage is an open source framework for building portals for developers [22]. In itself it is not a fully fledged IDP, but it can be extended with variety of plugins to support wide range of development workflows. It is also often used as the control plane for IDP [2]. Backstage was first developed as internal tool to increase visibility at Spotify but was later open sourced and donated to Cloud Native Computing Foundation (CNCF) [23].

Backstage includes some plugins with the basic installation, such as software catalog, software templates and technical documentation plugin. A wide variety of plugins are also available to add integrations to various systems. Spotify also sells premium plugins such as Soundcheck, a tool made to ensure service quality and alignment with guidance. [22]

The figure 3.2 presents a screenshot of Backstage demo environment. On the left side, different plugins can be seen, such as catalog, docs and tech radar. Each plugin has their own configurable page. The main content on the screenshot is the service catalog entry for a service named "www-artist". The catalog presents various information about the service, for example owner, lifecycle, a short description and it shows an interactive relation diagram. Diagram shows dependencies and dependent components. From this page we can quickly determine who to contact and with what criticality if service would have a major problem. Under the header on the right, there is a navigation bar for different plugins related to the services. This provides direct access to service specific plugin pages. [24]

Figure 3.2. *Screenshot of service catalog page on Backstage demo environment [24]*

The catalog plugin uses entity declaration configuration described in program 3.1. as the data source. For the simplest implementation, for each service a file like this needs to be added to the system. The fields can vary a lot depending on the installed plugins and

configurations. On some integrations part of these fields can be automatically populated from the code management system. [22]

```
1  apiVersion: backstage.io/v1alpha1
2  kind: Component
3  metadata:
4      name: www-artist
5      description: Artist main website
6  spec:
7      type: website
8      lifecycle: production
9      owner: team-a
10     system: artist-engagement-portal
11     consumesApis: ['component:artist-lookup']
```

Program 3.1. *Example service declaration. This is the same service as presented in figure 3.2*

3.7.2 Humanitech products

Humanitech is a company that provides help for teams to build their own IDP with the products that they provide [25]. Many of the online sources describing term Internal developer platform can be traced back to Humanitech [2, 16]. They support many of the industry standard methods for IaC but their main products are commercial.

Their main product is called Humanitech Platform Orchestrator. It can be used as the core of the platform to handle many of the features, such as dynamic configuration management, RBAC, infrastructure management and deployment automation [25]. The Orchestrator works by integrating with the CI and CD system. Figure 3.3 presents a reference architecture for implementing IDP using Humanitech Platform Orchestrator with Amazon Web Services (AWS) infrastructure. The Orchestrator is presented in Blue in the center of the figure, being the main contact between infrastructure declarations and infrastructure.

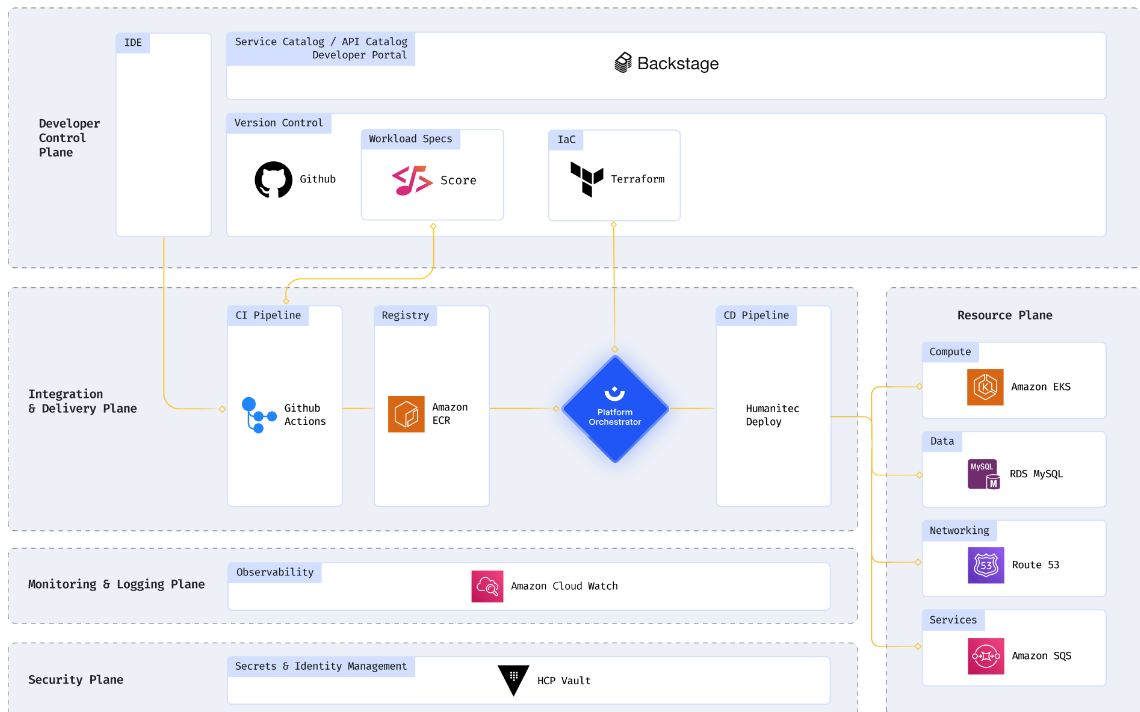

Figure 3.3. *Humanitech Platform Orchestrator reference architecture using AWS components [25]*

In the reference architecture figure 3.3, Backstage is used as one of the developer control planes. Humanitech also has their own control plane solution called Humanitech Portal. It is currently in beta stage and not much other information is available. [25]

Score is a open source workload specification. It provides abstraction for defining workloads that then can be transformed to multiple different platforms such as Helm or Docker Compose. It consists of two distinct components, specification and CLI tool that does the transformations. This means that developers can only define the specification once and then create Helm declaration for production and Docker Compose definition for local testing. [26] Humanitech Platform Orchestrator uses Score specification as the source for deployments [25].

Program 3.2 presents sample of a web application deployment Score specification. The example application consists of node service called hello-world and a postgres database. The spec also includes resource declarations for dns, route and database.

```
1  apiVersion: score.dev/v1b1
2  metadata:
3    name: hello-world
4  service:
5    ports:
6      www:
7        port: 8080
8        targetPort: 3000
9  containers:
10   hello-world:
11     image: .
12     variables:
13       PORT: "3000"
14       DB_HOST: ${resources.db.host}
15 resources:
16   dns:
17     type: dns
18   route:
19     type: route
20     params:
21       port: 8080
22   db:
23     type: postgres
```

Program 3.2. *Specification file for sample application [27]*

4. IDP BENEFITS IN THEORY: A CASE STUDY

This chapter aims to explore the fit and potential benefits of Internal Developer Platform (IDP) within one certain organization. Section 4.1 presents the organization in question and explains some of the current problems faced within the ways of working. Section 4.2 describes some current methodologies within the organization that could help with the integration. Section 4.3 outlines the anticipated benefits that the IDP could provide. Next, section 4.4 details the possible barriers for implementation. Lastly, section 4.5 theorises a possible action plan for implementing IDP in this specific context.

The observations and insights presented in this section are derived from firsthand experience from two years of working in different departments within the organization, enriched by conversations with five additional personnel within the organization. While these observations originate from specific instances, they seem to reflect the broader themes and challenges generally raised inside the organization. This analysis aims to offer a generalized understanding of the development environment challenges and operational practices.

4.1 Organization overview and contextual challenges

This case study focuses on a large organization in retail business. It has its own tech department consisting of around 500 personnel. The tech department has multiple sub debartments focusing on range of different areas needed in retail operation. On the organizational level there is a central infrastructure team that supports other DevOps roles with their efforts. This team also manages all of the external services that are being used. As with other organizations of substantial size, this one also suffers from the unique challenges caused by the size that impact its operational efficiency and team dynamics especially in the tech department.

The tech department has multiple different subdepartments that participate in wide range of different company operations. Each subdepartment has their own contextual requirements and dependencies which can be hard to take into account when discussing organizational matters in company wide context.

The technical landscape within the organization is both rich and complex. A wide range of

technologies are being used. There are some guidelines for recommended technologies used in both frontend and backend but a wide variety of other technologies is still being used. The architectures utilize microservice architecture.

Following challenges can be identified within the organization:

- **Lack of visibility**: A recurring issue is the lack of visibility across different departments and teams. This siloed nature of operations decreases cross-functional collaboration and knowledge sharing, leading to inefficiencies and duplicated efforts across projects. Lack of visibility can be seen in multiple different places such as service ownership, service dependencies and dependents and service lifecycle status.
- **Onboarding challenges**: Onboarding is challenging thing to do in any organization, and such is the case in this one. As different teams have their own methods, newcomers and team switchers need to familiarize themselves with the new ways of working. Most teams have internal onboarding documentation in the company internal document platform but those are quick to get outdated. Also, company utilizes external workforce in the tech department, causing faster rotation speeds for tech personnel.
- **No clear promoted ways of working**: Different product teams across the organization are quite seperated and have their own preferred decisions on many different topics, ranging from used tehchnologies to chosen folder structure and external tools used. While this allows more freedom, it causes differences between projects and teams. Additionally, when creating new services, there are no templates to use. Common way is to copy some existing service as a template. This can cause unwanted promotion of non-ideal methodologies and technologies.
- **No API presentation standards**: While many teams do have a some way of serving API documentation, it still varies a lot from team to team. Some use OpenAPI documentation and some teams publish their API documentation in the internal documentation platform. Manually updated API documentation has a bad habit of getting out of date fast.
- **Permission management to services**: External tools use mostly Single Sign-On (SSO) to authenticate. This means that getting access to external services requires service ticket. The existing permission system is built in a way that means that for every developer, all of the required systems need to be asked seperately. Additionally, adopting new services is more extensive because SSO needs to be implemented before the service can be used.
- **State of IaC**: In the organization, IaC and GitOps is heavily utiziled. It is used for most of the active infrastructure. For this, there are some general guidelines for DevOps personnel to follow. Self-service is endorsed but often developers needs

support from DevOps personnel. This causes unneccessary workload for them.

- **Monitoring**: State of the monitoring in the organization varies a lot. New motions for improving monitoring across different projects and teams have been started but generally the monitoring and the general visiblity of the state of the apps is lacking.
- **Lack of skill**: Organization utilizes external workforce and usually chooses to employ junior and medior roles in the teams. Being effective with DevOps methdologies requires wide range of skills and some teams are lacking in these skills to enable effective workflows.

These challenges collectively create significant inefficiencies across daily operations, which can add up to considerable time losses. Such inefficiencies not only slow down workflow processes but also impact overall productivity, ultimately hindering the company's ability to meet targets and deliver outcomes efficiently.

4.2 Application of IDP in organization

IaC and GitOps are used in the organization for infrastructure for most of the services in active development. The chosen tool is Terraform and centrally managed custom terraform modules are used. This means that there already is widely used level of abstraction for most of the infrastructure. When implementing IDP, this makes the infrastructure automation easier as that can be used for the infrastructure management.

Another related tool utilized within the organization is GitLab. It is used as the main tool for project version management and for CI and CD. Terraform configurations are used for managing projects and groups. Most ready made tools for IDP are compatible with GitLab, which means that for project version control management integration into IDP will be easier. Also, automation in project management means that if some configuration files are needed for each repository, it can be arranged programmatically.

4.3 Anticipated benefits

If IDP would be implemented in the presented organization, certain benefits could be anticipated. In this section the potential benefits are categorized to three distinct groups: increased visibility, established golden paths, and general benefits. Additionally, some other possible benefits could be identified.

The extensivity and the potential of these benefits are highly dependent on the implementation and the decisions made around the adoption process. A key factor for the implementation success is the amount of people and workforce that is allocated to work on the adoption process. Another major factor is the planned end product and the features it should have.

4.3.1 Visibility

There are multiple different aspects of visibility in the development process that could be improved. The biggest improvement could be done to increasing visibility across teams and departments with a service catalog feature. This would provide a way to present wide range of information about each service, such as service ownership, current lifecycle state of the service, possible dependencies and dependents and other general information.

Another possible visibility benefit could be a standardized and even mandatory way of presenting and documenting internal API endpoints of the services and information how to get access to these endpoints. This way locating necessary information for new integrations would be easier. IDP could also provide a way to check service logs and monitoring data. This could be utilized in higher abstraction levels, for example to see if any services in certain domain have any critical issues.

Currently in the organization there are some budgets in place for cloud infrastructure spending. Teams get warning when they go over budget, but there is no context of explanations for the budgets. IDP could provide features for increased visibility with cost analysis and budget tracking. Additionally, it could also give context for the budgets.

4.3.2 Golden paths

Having an IDP could provide an easy way to introduce golden paths into the organization. Common way of implementing golden paths is by creating templates that follow the chosen best practices and technologies. For example, creating a suggested workflow for a commonly used Java microservice would work as a template and a guideline on how to implement certain features, such as monitoring and automated testing. Clear established golden paths can have a lot of benefits in the organization. For developers, it would present clearly the best programming languages and frameworks to use. Creating a new project would become easier when the template would provide CI and CD, monitoring solutions, and developer could be sure that the technologies selected are the correct ones.

Golden paths provide standardization that on its own has similar benefits. Standard practices across teams make changing teams easier. It also makes the development and the extension of the framework easier as implementing new integrations need to be only implemented once.

4.3.3 General benefits

While increased visibility and established golden paths already offer great benefits for developers, there are other aspects that could also be affected. With fully fledged IDP,

developers could potentially get more done than before. Prototyping and testing new ideas becomes faster to implement. With self-service, developers are able to do more instead of relying on DevOps personnel. This also frees the DevOps personnel to focus more on their own tasks. These benefits are more apparent with personnel that consist of mostly junior and medior roles.

Well implemented IDP could work as the control plane for all the development related workflows. This would make the burden of context switching smaller for developers. Depending on the security limitations, IDP could also provide an easier way to integrate new services for developers.

4.4 Implementation barriers

Implementing IDP within the organization will not be an easy task. With every organizations of this size, adoption of new technologies is slow. The platform would need to provide enough value to overcome the burden of adoption and modifying existing workflows and tasks. To succesfully adopt IDP, an organizational change is required.

One big barrier for adoption is the cost of the implementation. Even if IDP would be implemented with open-source solutions, a significant investment in workhours is needed. For fully fledged solution, a dedicated team would be necessary in organization of this size.

As there are no existing promoted frameworks for services, creating new ones can turn out to be difficult. Agreeing on technical and non-technical decisions can be challenging. Frameworks should be open for proposals and improvements. Ideally all teams across all domains should agree on certain designs but this would be hard to achieve.

Currently there are no standardized ways for handling service monitoring. Different domains have different solutions. Implementing monitoring within the framework means either supporting all used services or promoting singular monitoring within the framework. In latter situation extra effort is needed from the products teams when migrating to the new framework.

In the organization, three different cloud providers are used for infrastucture. Most of the infrastructure is within one provider, but in order to support all existing services with IDP, all three providers need to be supported. Additionally, some on-premise infrastructure is also used.

4.5 Plan for implementation

Implementing IDP in the organization would have the possibility to be a transforming step to streamline and simplify the development processes. This subchapter outlines a plan to

implement IDP in the mentioned organization taking the specific challenges and potential benefits into account.

The adoption process should be initially started with MVP platform that is initially used only by couple teams in one domain. This method has already been tested to be effective within the organization. Good approach would be to choose a domain that has both legacy and newer projects and wide range of variety in projects to first validate the use cases and benefits. The usage can start even with minimal feature support to get user feedback as early as possible. Wider adoption should not be forced, the IDP should provide enough value for the users so that they want to use it and want to include it in their workflow.

The starting focus point should be to prioritise visibility across different software development teams. There is two different aspects why this would be the first priority. Firstly, the lack of visibility was recognized as the most impactful issue currently challenging the development efficiency. As such, enabling visibility would have the most potential impact. Secondly, establishing a framework for increasing visibility is relatively simple and can serve as the foundation for whole IDP. Even if initially implemented for only limited number of teams, the improvements in visibility can provide significant benefits.

Another major focus point should be creating golden paths early on. Initially creating just a single, well thought path that outlines recommended practices, tools, and architectures could help standardizing project setups. The first path for the organization should be Java backend service following microservices architecture, as that is the most common way to do backend services. The path needs to be modified by feedback and contributions and the end goal is to have a path that all the users can agree on. Also, even if user creates a project following a golden path, the user should still be able to break the path limitations if necessary. Starting with just single path is a good way to establish practices around iterating the path that other paths for other tehcnologies can learn a lot from. Ideally there should be paths for each recommended technology stack.

One suggested way of starting implementing golden paths is by creating an internal CLI tool that other developers can use to create and modify projects. This same tool could be used to add the service catalog definitions in interactive manner. CLI works as a good interface that most of the developers are already familiar with, but comes with some problems also such as the need to support wide variety of devices and operating systems.

Even if iterative implementation is a great way of implementing IDP, planning before implementation is necessary. Having a concrete and clear plan to reference in different stages of implementation great way to stay on track. Organization needs to dedicate certain amount of workforce for the implementation. Ideally a full team that would take the product mindset and fully own the whole IDP. The initial work could be started by just a one dedicated person.

5. DISCUSSION

This chapter discusses the limitations that have emerged while writing this thesis. Most of the limitations are rooted to the fact that the term is still quite new in the industry. The chapter also proposes some ideas for future work on this same topic.

This thesis was heavily affected by the fact that the thesis topic was provided by the examiner, and that the organization in question did not have internal demand for the study. The idea for case study was raised while researching the thesis topic and noticing the similar challenges within the organization while working there.

5.1 Limitations and validity assessment

This work has some limitations. Most notably, the term Internal Developer Platform (IDP) seems to be mostly promoted by one company, which raises questions about its recognition within the industry. The term barely exists in peer-review publications and two of the most popular online sources are both managed by the same company. In this work, the sources [2, 16, 25, 26, 27] can be traced back to this company.

Despite this concern, it's important to acknowledge that the underlying concepts and methodologies attributed to IDPs are not exclusive to this term or company. Similar frameworks and operational strategies are evident across various organizations and have been discussed in numerous publications. This observation suggests that while the specific term might be limited in its academic and industry adoption, the principles and practices it represents are relevant and already in use.

5.2 Future work

There are many different possibilities for future work on this topic. To better understand the effect and the possibilities of the term IDP, a multiple case studies should be performed. It would provide more comprehensive and more precise picture. Also, studying organizations that have implemented similar setups would provide better use cases of the actual implementations.

Another potential topic for research would be the future evolving of the ideology. As the IDP term is still quite new in the software development industry, it will be interesting to

see how it evolves and gets adopted to wider audience. This work only focused on the term IDP. Future studies could take a broader view on the similar ideologies and tools that are already in use and provide same kind of benefits. These topics are not yet that popular within the scientific publications. This could be due to that these challenges are more common in big organizations.

6. CONCLUSION

Internal Developer Platform (IDP) integrates various development and operational tools into a unified system that simplifies and enhances software development processes. The key benefits of IDP include improved workflow efficiency, reduced manual overhead, and increased collaboration across teams. These advantages make IDP an potential solution for organizations looking to enhance their software development lifecycle and to possibly fix some of the downfalls they might have with current DevOps implementations. The success of an IDP depends greatly on its implementation. It requires careful planning, customization according to organizational needs, and active management to stay relevant with ongoing technological changes.

Thesis also presented a case study on the potential implementation of an IDP in a large retail organization. It highlighted some existing challenges within the organization and how IDP could help solve them. The study pointed out that a successful IDP implementation requires significant resource investment in both time and budget. It also highlighted the need for planning to overcome typical challenges such as organizational resistance and the technical complexities of integration. The study offers insights into the necessary preparations and considerations for organizations considering an IDP.

This thesis highlights the role that Internal Developer Platforms can play in improving software development practices. IDPs can lead to more efficient development cycles, faster time-to-market, and improved software quality. The realization of these benefits depends heavily on strategic and thoughtful implementation. Organizations must engage in detailed planning, secure enough resources, and create an environment open for changes. As technology advances, the adaptability of IDP systems will be crucial in maintaining effectiveness and ensuring keeping up with the evolving needs of businesses.

REFERENCES

[1] Len Bass. *DevOps : a software architect's perspective*. eng. The SEI series in software engineering. New York: Addison-Wesley Professional, 2015. ISBN: 978-0-13-404984-7.

[2] Internal developer platform community. *Internal Developer Platform*. Jan. 2024. URL: https://internaldeveloperplatform.org/ (visited on 19 Jan. 2024).

[3] Michael Gall and Federico Pigni. "Taking DevOps mainstream: a critical review and conceptual framework". eng. In: *European journal of information systems* 31.5 (2022), pp. 548–567. ISSN: 0960-085X.

[4] Lucy Ellen Lwakatare et al. "DevOps in practice: A multiple case study of five companies". eng. In: *Information and software technology* 114 (2019), pp. 217–230. ISSN: 0950-5849.

[5] Gene Kim. *The devOps handbook : how to create world-class agility, reliability, and security in technology organizations*. eng. First edition. Portland, OR: IT Revolution Press, LLC, 2016. ISBN: 1-4571-9138-5.

[6] Leonardo Leite et al. "A Survey of DevOps Concepts and Challenges". eng. In: *ACM computing surveys* 52.6 (2020), pp. 1–35. ISSN: 0360-0300.

[7] Nasreen Azad and Sami Hyrynsalmi. "DevOps critical success factors — A systematic literature review". eng. In: *Information and software technology* 157 (2023), pp. 107150–. ISSN: 0950-5849.

[8] Sam Newman. *Building Microservices, 2nd Edition*. eng. O'Reilly Media, Inc, 2021. ISBN: 9781492034018.

[9] Martin Fowler. "Microservices". In: *martinfowler.com* (25 Mar. 2014). URL: https://martinfowler.com/articles/microservices.html (visited on 12 Feb. 2024).

[10] Kief Morris. *Infrastructure as code : managing servers in the cloud*. eng. First edition. Sebastopol, California: O'Reilly. ISBN: 1-4919-2433-0.

[11] Indika Kumara et al. "The do's and don'ts of infrastructure code: A systematic gray literature review". eng. In: *Information and software technology* 137 (2021), pp. 106593–. ISSN: 0950-5849.

[12] Florian Beetz and Simon Harrer. "GitOps: The Evolution of DevOps?" eng. In: *IEEE software* 39.4 (2022), pp. 70–75. ISSN: 0740-7459.

[13] Weaveworks. *GitOps*. URL: https://www.weave.works/technologies/gitops/ (visited on 10 Jan. 2024).

[14] VMware. *GitOps: The Cloud Operating Model*. VMware. Feb. 2021. URL: https://blogs.vmware.com/cloud/2021/02/24/gitops-cloud-operating-model/ (visited on 12 Mar. 2024).

[15] Thomas A. Limoncelli. “GitOps: A Path to More Self-service IT: IaC + PR = GitOps”. eng. In: *ACM queue* 16.3 (2018), pp. 13–26. ISSN: 1542-7730.

[16] Jeff Doolittle and Robert Blumen. “Luca Galante on Platform Engineering”. eng. In: *IEEE software* 40.6 (2023), pp. 144–146. ISSN: 0740-7459.

[17] Mauricio. Salatino. *Platform Engineering on Kubernetes.* eng. 1st ed. New York: Manning Publications Co. LLC, 2023. ISBN: 1-63835-415-4.

[18] Megan O’Keefe Darren Evans. “Light the way ahead: Platform Engineering, Golden Paths, and the power of self-service”. In: *Google Cloud Blog* (). URL: https://cloud.google.com/blog/products/application-development/golden-paths-for-engineering-execution-consistency (visited on 24 Feb. 2024).

[19] Manuel Pais and Matthew Skelton. *Team Topologies*. eng. IT Revolution Press, 2019. ISBN: 9781098157234.

[20] “Developers - How To: Enhance Platform Engineering with an Internal Developer Platform”. eng. In: *Open Source FOR You* (2024).

[21] Zeljko Seremet and Kresimir Rakic. “Platform Engineering and Site Reliability Engineering: The Path to DevOps Success”. In: (2022). Ed. by Branko Katalinic, pp. 155–162. ISSN: 1726-9687. DOI: 10.2507/daaam.scibook.2022.13.

[22] Backstage Project Authors. *What is Backstage?* Feb. 2024. URL: https://backstage.io/docs/overview/what-is-backstage (visited on 25 Mar. 2024).

[23] Backstage Contributors. *Backstage Turns Two!* Mar. 2022. URL: https://backstage.io/blog/2022/03/16/backstage-turns-two/#out-of-the-sandbox-and-into-incubation (visited on 25 Mar. 2024).

[24] Backstage. *Backstage demo*. URL: https://demo.backstage.io/catalog/default/component/www-artist (visited on 25 Mar. 2024).

[25] Humanitech. *Humanitech - Serving Platfrom Engineers*. URL: https://humanitec.com/ (visited on 7 Apr. 2024).

[26] Humanitech. *Humanitech developer documentation*. URL: https://developer.humanitec.com/ (visited on 7 Apr. 2024).

[27] Score community. *Score spec - sample score app GitHub, Hash 67b2d04*. URL: https://github.com/score-spec/sample-score-app/blob/main/score.yaml (visited on 7 Apr. 2024).