INSTITUTO SUPERIOR DE TECNOLOGIAS AVANÇADAS DE LISBOA

Mestrado em Informática

Dissertação para obtenção do Grau de Mestre em

Informática

The Role of Platform Engineering in Digital Transformation

Tiago Filipe Martins Arrulo

Presidente: Professor Doutor Pedro Ramos dos Santos Brandão

Arguente: Professor Doutor Filipe Montez Coelho Madeira

Orientador: Professor Doutor Paulo André Reis Duarte Branco

Lisboa

Julho 2025

ISTEC

INSTITUTO SUPERIOR DE TECNOLOGIAS AVANÇADAS DE LISBOA

Campus Académico do Lumiar, Lisboa

Dissertação

Mestrado em Informática

Tiago Filipe Martins Arrulo

Dissertação de Mestrado apresentada para cumprimento dos requisitos necessários à obtenção do grau de mestre em Informática, realizada sob a orientação científica do Professor Doutor Paulo André Reis Duarte Branco.

Lisboa, 2025

"*Veni, vidi, vici*."

Julius Caesar

Resumo

À medida que o desenvolvimento de software se torna mais complexo e competitivo, as organizações deparam-se com *toolchains* fragmentados, complexidade operacional e fluxos de trabalho inconsistentes, dificultando a sua jornada DevOps. Como resultado, as organizações lidam com ciclos de entrega mais longos e a obtenção de um menor valor comercial. Platform Engineering, através da implementação de Internal Developer Platform (IDP), surgiu como uma abordagem promissora para mitigar estes problemas. Estas plataformas foram concebidas de modo a abstraírem a complexidade, aumentarem a produtividade dos programadores e simplificar os fluxos de trabalho.

Esta dissertação tem como objetivo documentar a elaboração, implementação e avaliação de um protótipo de IDP, como forma de validar a utilização de Platform Engineering como resposta aos desafios de desenvolvimento de software. Este protótipo integra ferramentas e fluxos de trabalho numa plataforma *self-service*, explorando como as IDPs podem uniformizar processos, reduzir a complexidade operacional e utilizar *golden paths* para fornecer aos programadores fluxos de trabalho claros e otimizados. A investigação utiliza a abordagem *Design Science Research Methodology* (DSRM) para assegurar um processo iterativo, focado no desenvolvimento e avaliação de uma IDP, através de ciclos de identificação de problemas, criação de soluções, implementação, avaliação e aperfeiçoamento.

Complementarmente ao desenvolvimento do protótipo, foram conduzidas duas rondas de entrevistas com profissionais da área das tecnologias. Estas entrevistas forneceram visões qualitativas sobre os benefícios, limitações e aplicabilidade prática de Platform Engineering e dos IDPs, permitindo validar os resultados empíricos do protótipo e reforçar a compreensão do seu impacto na experiência do programador, na produtividade das equipas e na entrega de valor ao negócio.

O objetivo desta investigação é analisar o potencial de Platform Engineering e das IDP para melhorar a velocidade na entrega de software, a fiabilidade e a eficiência operacional. O estudo procura conceber conhecimentos que contribuam para uma compreensão teórica de Platform Engineering, que ofereça orientações práticas para as organizações que estejam a considerar a sua adoção de modo a obterem uma vantagem competitiva no panorama tecnológico e adicionalmente, espera contribuir para a comunidade científica.

Palavras-chave: Cloud Computing, Internal Developer Platform, DevOps, Platform Engineering

Abstract

As software development becomes increasingly complex and competitive, organizations struggle with fragmented toolchains, operational complexity and inconsistent workflows, hindering their DevOps journey. These challenges prevent teams from achieving the full potential of DevOps. As a result, organizations experience longer delivery cycles and lower business value. Platform Engineering, through the implementation of an IDP, has emerged as a promising approach to mitigate these issues. These platforms are designed to abstract the underlying complexities, enhance developer productivity, and simplify workflows.

This dissertation aims to document the design, implementation, and evaluation of an IDP prototype, as a way to validate the use of Platform Engineering in addressing software development challenges. This prototype integrates tools and workflows into a self-service platform, exploring how IDP may standardize processes, reduce operational complexity, and utilize golden paths to provide developers with clear and optimized workflows. The research uses the DSRM approach ensures an iterative process, directing the development and evaluation of the IDP, through cycles of problem identification, solution design, implementation, evaluation, and refinement.

To complement the practical component, two rounds of interviews were conducted with IT professionals, which provided qualitative insights into the perceived benefits, limitations and real-world applicability of Platform Engineering and IDPs. These perspectives helped validate the empirical findings from the prototype, providing a broader understanding of its impact on developer experience, productivity and value delivery.

The objective of this research is to investigate the potential of Platform Engineering and IDP to enhance software delivery speed, reliability, and operational efficiency. The study seeks to produce insights that contribute to both theoretical understanding of Platform Engineering and offer practical guidance for organizations considering its adoption to gain a competitive advantage in the technology landscape and hopefully provide a contribution to the scientific community.

Keywords: Cloud Computing, Internal Developer Platform, DevOps, Platform Engineering

Acronyms

Meaning	Acronym
Internal Developer Platform	IDP
Design Science Research Methodology	DSRM
Digital Transformation	DT
National Institute of Standards and Technology	NIST
Elastic Cloud Computing	EC2
Application Programming Interface	API
Infrastructure as a Service	IaaS
Platform as a Service	PaaS
Software as a Service	SaaS
Simple Storage Service	S3
Amazon Web Services	AWS
Google Cloud Platform	GCP
Small and Medium Business	SMB
Capital Expenditure	CAPEX
Operational Expenditure	OPEX
Continuous Integration	CI
Continuous Delivery/Deployment	CD
Artificial Intelligence	AI
Internet of Things	IoT
Machine Learning	ML
Open-Source Software	OSS
Infrastructure as Code	IaC
Cloud Native Computing Foundation	CNCF
Proof of Concept	PoC
Version Control System	VCS
Google Kubernetes Engine	GKE
Virtual Private Cloud	VPC
Classless Inter-Domain Routing	CIDR
Chief Technology Office	CTO

Index

List of Figures

List of Tables

1. Introduction

1.1. Context

The increasing complexity of modern software development demands innovative solutions capable of overcoming inefficiencies, reducing cognitive loads for developers and managing operational complexity at scale. Organizations are adopting microservices architectures, containerization and multi-cloud strategies, but these advancements often burden development teams with a complex technological landscape [1]. This results in longer delivery cycles, duplicated work and increased maintenance load, delaying agility and innovation.

Platform Engineering, with its IDPs, offers a compelling solution to these challenges. IDPs act as a central platform that provides self-service access to tools, services and knowledge, enabling developers to provision infrastructure, manage deployments and access standardized environments, with minimal friction [2], [3].

1.2. Motivation

Despite the rising interest in this type of solution, Platform Engineering is a relatively new concept, with a demonstrated lack of comprehensive academic research regarding the topic. With the accelerating adoption of Cloud technologies and the increasing need for organizations to deliver software and updates faster, the need for Platform Engineering seems imminent [1], [4].

As a result, there is a clear motivation to bridge this research gap and contribute to the scientific knowledge of Platform Engineering, particularly its role in improving developer experience and organizational efficiency. To address the absence of dedicated research on the topic, this dissertation seeks to combine professional practices with scientific literature, aiming to provide insights that are both practical and academically valuable.

1.3. Research Questions

In the context of modern software development, Platform Engineering emerged as a vital discipline, with the goal of simplifying software delivery processes, improving developer experience and enhancing overall productivity. Despite the growing interest in Platform Engineering, there are still questions regarding its practices that can address challenges faced by teams and organizations.

To investigate this, the following research questions were asked:

1. In what way does Platform Engineering improve the developer experience?

2. How can Platform Engineering improve productivity and deliver more value?

Answering these research questions will provide an understanding of the benefits of Platform Engineering, its impact on developer workflows and its role in enabling organizations to deliver software more efficiently.

1.4. Objectives

In this dissertation, the objective is to explore and analyze the emerging concept of Platform Engineering, focusing on its potential to improve the efficiency and effectiveness of software development and delivery processes. The research combines theoretical insights with a practical implementation of an IDP, along with two rounds of interviews to gather professional feedback. As such, the following objectives were defined:

- To provide an overview of Platform Engineering.
- To design, develop and evaluate a prototype of an IDP.
- To analyze the impact of Platform Engineering through experimentation.
- To gather perspectives from IT professionals via two sets of interviews.

This dissertation aims to provide a structured study that demonstrates the feasibility of Platform Engineering and Internal Developer Platforms, while validating its potential to transform software development and delivery processes. By achieving these objectives, the research aims to serve as a foundation for future exploration and practical adoption.

1.5. Research Methodology

The methodology adopted for this research is the Design Science Research Methodology (DSRM), a well-established approach particularly suited for designing and evaluating IT artifacts. DSRM provides a structured yet flexible framework for solving practical problems, through the development and evaluation of innovative solutions. Its iterative nature supports continuous refinement, making it appropriate for the evolving requirements of Platform Engineering environments [5].

As highlighted in Figure 1, the DSRM consists of six stages, each contributes to a structured approach for solving practical problems:

1. Problem Identification: Define the problem and the importance of addressing it.
2. Definition of the objectives of a solution: Identify the goals of the solution and highlight its importance.
3. Design and Development: Create the artifact that will address the problem.

4. Demonstration: Apply the artifact in relevant context to demonstrate its usefulness.
5. Evaluation: Measure and observe how the artifact performs in addressing the problem.
6. Communication: Document and communicate the problem, the artifact, and the results.

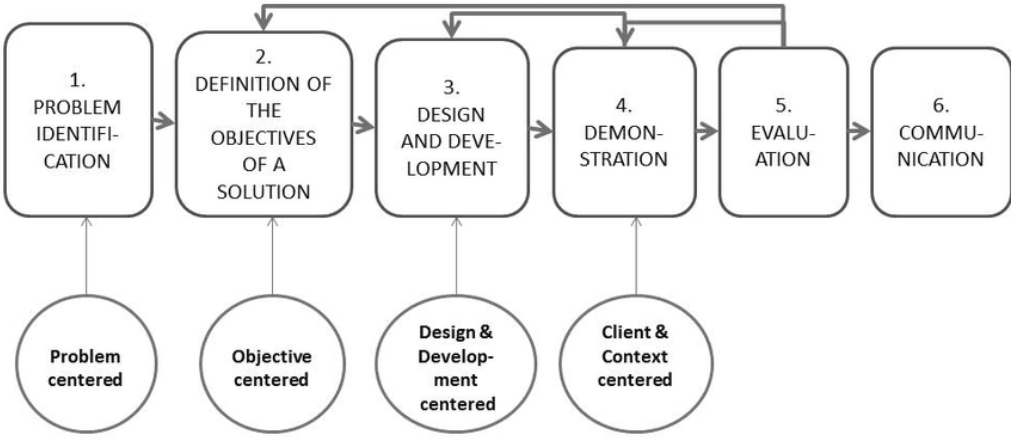


Figure 1 – Design science research method (DSRM) process model [6]

This structure was first formalized by Peffers, Tuunanen, Rothenberger and Chatterjee and has been widely applied in IT research to guide practical innovation [5]. For instance, Adwan and Alsaeed applied DSRM in the development of a Cloud Computing Framework for retail banks in Bahrain. Their research confirmed the relevance of DSRM in real-world enterprise settings, particularly by highlighting its flexibility. As the authors emphasize, the methodology does not require a strict sequential progression through its phases. Researchers may enter the process at different stages, such as starting directly when the problem and requirements are well understood, and the initial phases may be streamlined or combined if the research context is already well established. This adaptability strengthens the practical utility of DSRM in projects where responsiveness to evolving technical constraints is critical, allowing iterative refinement and stage-skipping when justified, DSRM ensures both methodological rigor and operational relevance [7].

In this dissertation, the DSRM framework was followed systematically:

- Problem Identification: The study began by identifying current inefficiencies in cloud-native software development processes, particularly the fragmentation of tooling and lack of standardization.
- Definition of the objectives of a solution: Objectives were defined to improve developer workflows, standardize infrastructure provisioning, and introduce an Internal Developer Platform (IDP) as a strategic enabler.
- Design and Development: A technical artifact was created in the form of an IDP prototype using Backstage, GitHub, GitHub Actions and Terraform. This artifact was

designed to orchestrate cloud resource provisioning workflows and reduce developer cognitive load.

- Demonstration: The artifact was demonstrated through controlled use cases that replicated common developer tasks, allowing observation of its benefits in standardized conditions.
- Evaluation: This dissertation applied empirical evaluation through performance metrics and qualitative assessment of user experience.
- Communication: The outcomes of the research are consolidated in this dissertation and may be shared with the academic community.

1.6. Document Structure

This document is composed of five chapters and is organized in the following structure:

- **Chapter 1 - Introduction**: This chapter sets the stage for the research, by providing the context and highlighting the importance of Platform Engineering in modern software development. It provides the research problem, defines the scope of the study and research objectives.
- **Chapter 2 – Theoretical Background**: This chapter establishes the foundational knowledge necessary for understanding this research. It explores key concepts related to Digital Transformation, Cloud Computing, DevOps and their association to Platform Engineering. This chapter is crucial to provide the reader with a theoretical basis for comprehending the following chapters.
- **Chapter 3 – Literature Review**: Reviews the State of the Art, presents a literature review and discusses the current research on Platform Engineering.
- **Chapter 4 - Project**: This chapter details a practical project developed using DSRM to address the research objectives. It details the problem and motivation, defines the solution objectives and describes the design and development processes. The chapter includes a demonstration of the project, including the preparation, implementation and results. Finally, an evaluation of the project is presented alongside with the findings.
- **Chapter 5 - Conclusion**: This chapter concludes the thesis summarizing the key findings and insights obtained from the research project. It discusses the implications of the findings for practitioners and academics.

2. Theoretical Background

2.1. Digital Transformation

With the emergence of new digital technologies like social media, mobile and big data, has motivated companies to explore the potential benefits of Digital Transformation (DT), making DT a central focus for both organizations and researchers, given its potential to revolutionize businesses and reshape the competitive landscape [8], [9].

While no universal definition has been established, DT has been broadly understood as a strategic integration of new digital technologies (such as mobile, artificial intelligence (AI), cloud computing, blockchain and Internet of Things (IoT)) across various business areas, leading to improve customer experience, streamline operations and create new business models, through an implementation that often includes the transformation of products, practices and organizational structures, to address the increasing customer expectations, maintain a competitive advantage and promote an innovative-driven culture within the organization [8], [9], [10], [11], [12], [13].

However, the process of DT extends beyond simply adopting new practices and tools, it requires a shift in the organizational mindset, the operational processes and how value is delivered to stakeholders, introducing this shift is identified in several surveys as a challenge, with many DT initiatives failing to meet the objectives set, due to the difficulties of aligning organizational mindset and practices with the possibilities offered by the digital technologies, as the lack of cultural and operational shift, often leads to an enhancement of existing inefficiencies instead of an improvement [12], [13], [14].

2.1.1. Drivers for Digital Transformation

In today's rapidly evolving market, DT extends beyond keeping pace with industry trend, it has become a crucial strategy for organizations seeking to remain competitive. However, embracing DT is a complex process that requires companies to adopt innovative digital solutions, rethink traditional business models and ensure operational continuity [15]. Identified below are the two main key drivers, external and internal, that motivate DT initiatives [16], [17]:

- External Drivers (Market and Environment-Driven)

External drivers originate from outside the organization and are influenced by market dynamics, industry shifts and social trends.

- Changing Customer Behaviors and Expectations: Customers demand personalized, transparent and seamless experiences, obliging organizations to adapt to the evolving needs to remain relevant.
- Market Pressure and Competition: The competitive landscape is shaped by digital disruption, from companies leveraging innovative technologies to rapidly gain market share, demanding others to quickly adjust or become obsolete.
- Digital Shifts in the Industry: Industries are being reshaped by digital technologies, forcing organizations to adapt to new standards.
- Technological Progress: The rapid advancement of technologies such as AI, Cloud Computing, IoT and Big Data is creating opportunities for innovation and efficiency.
- Regulatory and Legal Frameworks: New regulations and legal frameworks can motivate organizations to adopt digital technologies and transform their processes to ensure compliance.
- Supply Chain: The increasing complexity of supply chains motivates organizations to improve their processes by leveraging digital technologies.

- Internal Drivers (Organization-Driven)

Internal drivers originate within the organization and are related to strategic goals, operational efficiency and organizational culture.

- Process Improvement: Organizations seek to optimize internal processes, reduce inefficiencies and enhance productivity. DT may offer tools to automate tasks, streamline workflows and improve operational performance.
- Workplace Improvement: Creating more engaging, efficient and safe work environment is a key driver for DT, as technologies can be leveraged to improve collaboration, automate routine tasks, enhance safety and support employee development.
- Cost Reduction: DT initiatives can lead to cost savings through automation and increased efficiency.
- Management Support: Leadership and vision from management are essential for driving a successful DT.
- Employee Support: Digital technologies and systems assist employees to perform their work and may simplify and automate their tasks.
- Vertical Integration: The need for data flow and communication between different levels of the organization is driving the adoption of digital systems.
- Horizontal Integration: This driver refers to the seamless integration of various IT systems and processes across the value chain. The goal is to breakdown silos

between departments and create a unified system that enables data-sharing, collaboration and process optimization.

2.1.2. Impact of Cloud, DevOps and Platform Engineering in Digital Transformation

DT is proving crucial for organizations striving to meet evolving customer needs in a competitive landscape, leveraging several technologies that enable this transformation.

Cloud Computing is a current part of the DT drive in today's market, by offering scalable, flexible and agile solutions, with its on-demand and self-service models that enable rapid deployment of services, by minimizing the need for substantial infrastructure investments, unlock cost-effectiveness and agility, ultimately promoting innovation and competitiveness [18], [19], [20].

DevOps plays another key role for accelerating DT initiatives by fostering collaboration, automation and continuous improvement across development and operation teams. Through this collaboration it is possible to accelerate delivery cycles, enhance reliability and responsiveness to market shifts. By integrating modern technologies such as containerization, Cloud Computing and microservices, further enhancing agility and operational efficiency [21], [22].

Platform Engineering complements these efforts by establishing a solid and adaptable foundation that allows to seamlessly integrate several technologies while simplifying management through scalable and cloud-native architectures. This ensures that applications can evolve to shifting demands, improving operational efficiency and market responsiveness. Through cross-team collaboration and workflows for development, testing and innovation, Platform Engineering ensures swift adoption of transformative technologies, helping organizations remaining competitive in an increasingly fast-paced digital landscape [23], [24], [25].

2.2. Cloud Computing

Cloud Computing, as defined by the National Institute of Standards and Technology (NIST), is a continuously evolving paradigm characterized by a model that allows users to flexibly provision and adjust computing resources to meet their requirements. These resources, ranging from servers and storage to applications, are drawn from a shared pool, allowing efficient, on-demand allocation and scalability [26]. This definition is widely recognized and has been adopted by many researchers, serving as a foundational reference point that guides both academic research and industry practices [27], [28].

Fournier and Nowland's observations underscore the importance of simplifying the developer's interaction with complex infrastructure. Their argument aligns with the need for Platform Engineering to streamline and centralize cloud resources in a way that minimizes operational complexity [29].

2.2.1. Cloud Computing Core Concepts

The concept of Cloud Computing was first introduced in an academic context by Ramnath Chellapa during the 1997 INFORMS Annual Meeting in Dallas [30]. It is aligned with the ideology of Utility Computing, a term created by Professor John McCarthy in 1961, who envisioned computing power as a public utility [31]. The rise of Cloud Computing practices can be attributed to telecommunication companies offering cost-effective, dedicated point-to-point Virtual Private Network (VPN) services [32].

The first major technological breakthrough in the development of Cloud Computing, occurred with the introduction of Salesforce platform in 1999, which allowed customers to access business applications directly through the company's website, an early demonstration of on-demand, internet-based service delivery that would later become standard in Cloud Computing [33]. These advances encouraged other technology giants, such as Amazon, to play a central role in the development and adoption of the Cloud. Amazon chose to modernize its own infrastructure and invest in offering computing and other services to end users. In 2006, Amazon launched one of its first services, Amazon Elastic Compute Cloud (EC2), which provides remote virtual machines. These developments motivated competitors such as Google and IBM to begin investing in Cloud Computing research projects in 2007, collaborating with various universities [32], [34], [35]. Subsequently, in 2008, Microsoft and Google released their first Cloud services, Azure and Google App Engine, respectively [36].

In addition to its conceptual and historical groundwork, the architecture of Cloud Computing environments can be divided into four different layers: the hardware (or datacenter) layer, the infrastructure layer, the platform layer and the application layer. Each layer addresses different aspects of Cloud Computing service delivery.

- **Hardware Layer**: The hardware layer manages the physical infrastructure of Cloud Computing, including services, storage systems, routers and the associated power and cooling systems. Typically, these resources are hosted in datacenters.
- **Infrastructure Layer**: Sitting above the hardware is the infrastructure (or virtualization) layer. Here, virtualization technologies (such as Xen, KVM, VMware) pool and abstract physical computing and storage resources, turning into a scalable, flexible resource pools.
- **Platform Layer**: On top of the infrastructure layer there is the platform layer, which includes operating systems, development frameworks, and runtime environments. By providing standardized Application Programming Interface (API) and managed environments, this layer simplifies the development and deployment of applications.
- **Application Layer**: At the top of the stack is the application layer, providing the end-user applications and services that leverage the underlying Cloud Computing infrastructure. In

contrast to traditional applications, they can automatically adjust to the demands, maintaining a high availability, a better performance and cost [37].

Building on this understanding, there are five essential characteristics that describe Cloud Computing. These include:

- On-demand self-service, which allows users to independently provision resources without requiring provider interaction.
- Broad network access, enabling services to be accessed through any device over the network.
- Resource pooling, where resources are shared among multiple users in a multi-tenant model.
- Rapid elasticity, allowing resources to scale dynamically based on demand.
- Measured service, which ensures that the resource usage is monitored, controlled and reported to provide transparency for both providers and customers.

These characteristics highlight the efficiency, flexibility and scalability that make Cloud Computing a transformative paradigm in IT [26], [27].

2.2.2. Service Models

Expanding on the characteristics of Cloud Computing, the concept is supported by three core service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). These models provide varying levels of control, management and flexibility to the end-users, enabling them to choose the solution that best suits their needs. As illustrated in Figure 2, the responsibilities of managing different layers of infrastructure and software vary among the Traditional environment (On-Site), IaaS, PaaS and SaaS. While IaaS leaves several upper layers, such as operating system and applications, in user's hands, PaaS and SaaS shift more management tasks to the service provider, enabling users to focus on their core activities rather than on maintaining the infrastructure.

- **IaaS**: IaaS enables users to acquire computing resources, such as processing power, networking, memory and storage, and use it to deploy and run their own applications. Compared to PaaS and SaaS, IaaS provides users greater flexibility, as it allows users to install software on top of the operating system. However, the users are responsible for maintaining, updating and patching the operating system and all the applications. Examples of IaaS include AWS EC2 and AWS Simple Storage Service (S3).
- **PaaS:** PaaS offers users the ability to deploy, test and host their applications, using a set of programming languages and tools that are supported by the PaaS provider, without the need to manage the infrastructure behind the solution. PaaS provides its users a high

level of abstraction, allowing them to focus on development of their applications. Hence, the users do not have control or access to the underlying infrastructure. Examples of PaaS include Google App Engine and Microsoft Azure.

- **SaaS:** SaaS allows users to take advantage of software applications over the Internet, without the need to manage or control the infrastructure, or even install or execute applications. SaaS provides an even higher level of abstraction, as the user does not have control or access to the underlying infrastructure, that is being used to host the software. Examples of SaaS include Salesforce Customer Relationship Management (CRM) and Google Docs [26], [27], [38].

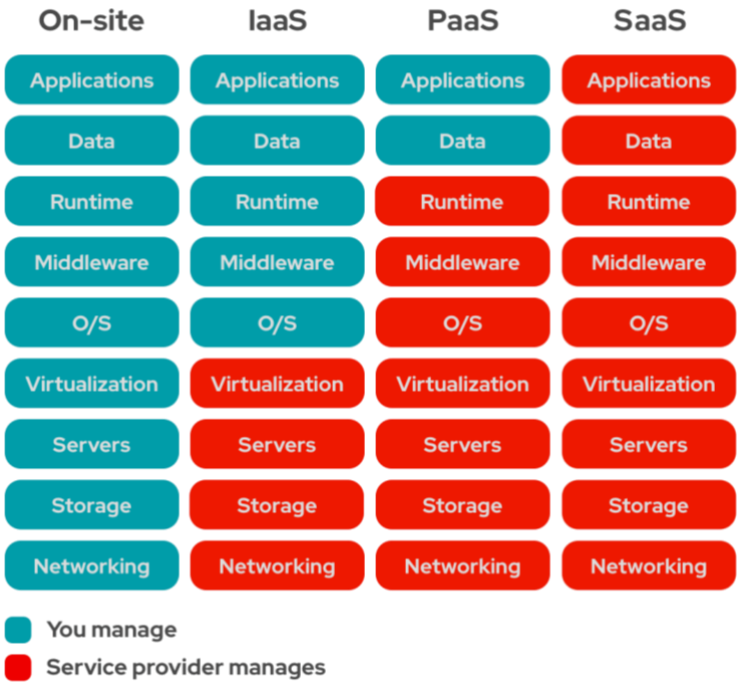


Figure 2 - Comparison of Traditional, IaaS, PaaS and SaaS Models [39]

2.2.3. Deployment Models

Peter Mell and Timothy Grance from NIST have categorized the delivery of services in the Cloud into four deployment models, each addressing different organizational requirements: Community Cloud, Hybrid Cloud, Private Cloud and Public Cloud. Community Cloud environments are intended to be shared and used by specific organizations or communities of users and is typically setup to address their specific requirements. Hybrid Cloud combines the elements of Community, Public and Private Cloud deployment models. Each Cloud environment can be independently managed, but applications or data can move across the Hybrid Cloud. This deployment model offers companies a balance between flexibility, scalability and security, as it allows them to leverage the unique strengths of each Cloud type. Private Cloud is intended for the exclusive use of individual organizations, providing enhanced

control and security. The Cloud may be managed by the organization or a third-party provider. Public Cloud, exemplified by platforms such as AWS, Azure and GCP, provides services that are available to the public via the Internet [26], [27].

Figure 3, adapted from [27], visually represents these four deployment models and how Hybrid Cloud combines the characteristics of the other three. This illustration helps to clarify the distinctions and intersections between Community, Private, and Public Cloud.

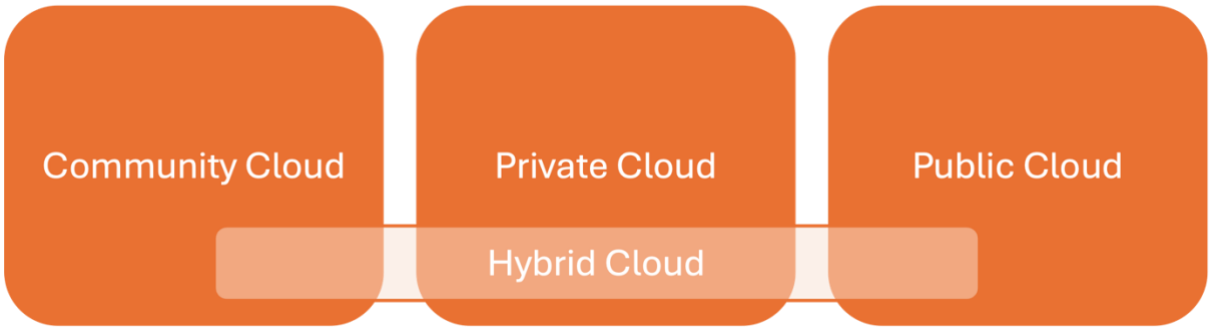


Figure 3 - Cloud Deployment Model according to NIST

2.2.4. Advantages and Challenges

Having discussed the service and deployment models, it is essential to analyze their advantages, and the challenges associated with Cloud Computing implementation. On one hand, companies can benefit from cost savings, scalability and simplified management. On the other hand, it is necessary to address concerns such as data security, service quality and interoperability.

In the digital era, Cloud Computing has emerged as a transformative force, reshaping traditional practices and processes. One of its main advantages is the potential for significant cost reduction, especially for small and medium-sized businesses (SMBs), which can avoid upfront investments while benefiting from modern technological resources. By reducing the need for large Capital Expenditure (CAPEX), companies can transition to a more flexible Operational Expenditure (OPEX) model. This shift reduces the financial burden, as well as facilitates immediate access to hardware resources, enabling companies to rapidly scale their operations.

Building on this financial flexibility, Cloud Computing lowers barriers to innovation, by providing access to advanced resources without the need for extensive investments. This inspires companies to experiment and innovate with agility, allowing them to develop and deploy new ideas and products faster. Additionally, the scalability offered by Cloud Computing is crucial, as it enables companies to dynamically adjust the computational resources based on their evolving needs. This scalability ensures that resources are optimized, avoiding the inefficiencies of underutilized capacity,

while supporting the development of new types of applications and services that were impractical in traditional IT environments.

Another benefit of Cloud Computing is its virtually unlimited storage capacity, which allows organizations to efficiently store and backup large volumes of data. Finally, while data security remains a common concern, Cloud service providers have made it a priority, continuously improving their security measures [40], [41].

However, despite these advantages, cloud adoption presents several challenges. One major concern revolves around the shift in responsibility for data security. Although cloud providers implement security measures, storing data externally raises questions about its integrity, privacy, ownership and the lack of physical control over the data locations.

In addition to security concerns, the efficient management of resources in the Cloud requires continuous monitoring to ensure optimal performance and avoid constraints such as overutilization, leading to cost inefficiencies, and underutilization, resulting in wasted capacity, are common without robust monitoring systems. Service quality is another challenge, as companies are often hesitant to migrate critical components to the Cloud without guarantees of performance and availability. The interoperability between Cloud providers also poses as a major challenge, as the lack of universal standards can complicate migrations and integration efforts. Finally, data transfer cost and potential performance bottlenecks caused by limited bandwidth remain areas of concern [40], [42].

Cloud Computing has gained popularity, since it first appeared on the market, as it offers numerous advantages to its users, including simplified resource management, fault-tolerant infrastructure, enhanced security and the quality of services associated with large-scale operations. Many IT companies have adopted Cloud solutions to develop or promote their technologies [43].

In recent years, the way companies access and utilize digital technologies has experienced changes. Traditionally, organizations were required to make upfront investments, classified as CAPEX, in hardware, software, and maintaining large IT departments. However, with the growth of Cloud Computing, companies are progressively relying on Cloud Computing to meet their processing, software and storage needs. This approach, is characterized by on-demand service availability and pay-as-you-go pricing models, has shifted IT expenses from CAPEX to more flexible, variable costs, classified as OPEX. These changes have reduced the financial IT investment and encouraged organizations to go beyond traditional IT practices, adopting innovative operational models [44].

A study conducted by Gartner researchers in February 2022, predicts that more than half (51%) of enterprise IT spending in areas such as application software, infrastructure software, business process services and system infrastructure will transition from the traditional solutions to the Public Cloud. This shift has been significantly accelerated by the COVID-19 pandemic, which forced organizations to

adapt rapidly to new business and social dynamics. Furthermore, advancements in technologies such as AI and Machine Learning (ML) have driven the adoption of Cloud Computing [45], [46]. As illustrated in Figure 4, investments are increasingly moving away from traditional IT solutions and towards Cloud services, reflecting the industry's broader trend toward more agile, scalable, and cost-effective computing models.

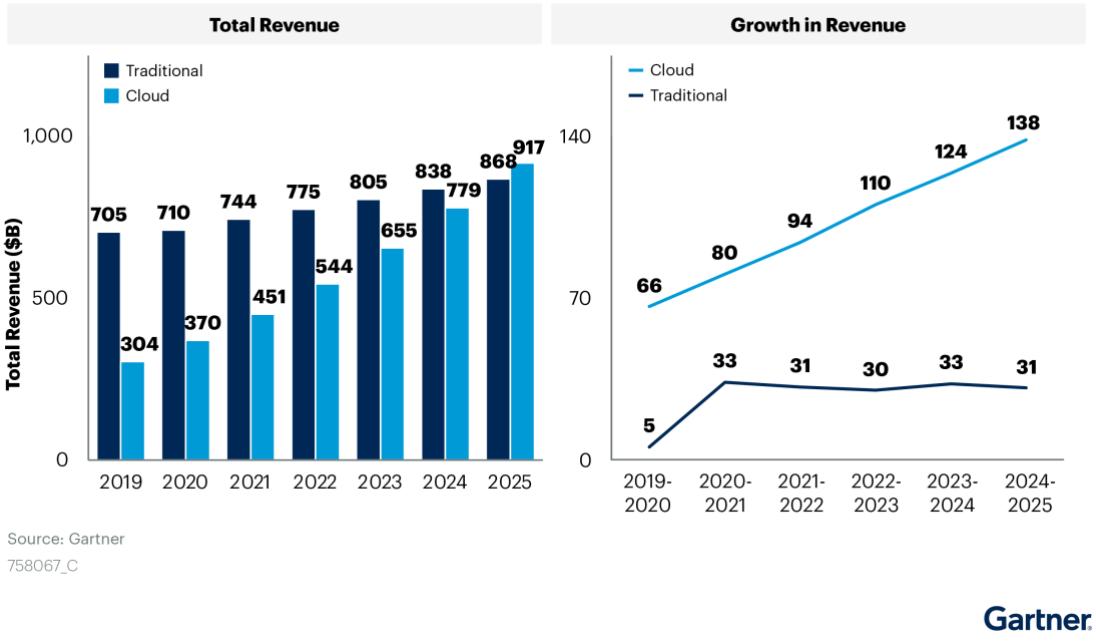


Figure 4 - Investment Shift to the Cloud [45]

2.2.5. Association with Platform Engineering

While Cloud Computing offers flexible, on-demand access to Computing resources, it does not solve the challenges of managing shared infrastructure, maintenance complexity and operational overhead across multiple teams. According to the authors Camille Fournier and Ian Nowland, "the cloud" by itself is not a platform that will cure all the issues. Instead, Platform Engineering emerges as a systematic approach to leveraging Cloud capabilities and organizing them into self-service product. By standardizing tools, infrastructure, and workflows streamlining development processes and promoting collaboration, Platform Engineering provides a critical foundation for improving agility, accelerating time-to-market, and enhancing product quality and user experience. Through this approach organizations can align Cloud services, APIs, tools and support structures into internal offerings, enabling development teams to build and deploy applications more efficiently, without being burdened by the complexity of infrastructure, allowing the Cloud's potential to fully materialize within the company [29].

2.3. DevOps

DevOps as a concept has been shaped over time with various interpretations. Based on the authors Andrej Dyck, Ralf Penners and Horst Lichter, the definition of DevOps is: "DevOps is a collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions, while guaranteeing their correctness and reliability" [47], [48]. DevOps emerged as a response to the traditional inefficiencies between Development (Dev) and Operations (Ops) teams, which often worked isolated, by combining them and enforcing a radical change in the philosophy of software development and IT operations, aiming to improve the integration of all the activities in software development and operations of an application system. This shift introduced a collaborative approach, focusing on communication, integration and automation to speed up the process and improve the quality of applications and service delivery [49], [50]. The primary goal of integrating Development and Operations is to enable IT companies to adapt quickly and efficiently to dynamic business environments, ensuring greater agility and responsiveness to change.

While DevOps highlights the integration of Development and Operations to streamline the delivery and management of software systems, Platform Engineering builds upon these foundations, by providing a curated internal platform or portal that standardize and simplify the build, deliver and deployment workflows for the development teams. Platform Engineering align closely with the objectives of DevOps, promoting collaboration, improving the developer productivity and accelerating time-to-market. This synergy positions Platform Engineering as an enabler of DevOps Practices in modern organizations [2].

2.3.1. DevOps Overview

The DevOps approach originated at the Agile Conference in Toronto in 2008, where Patrick Debois and Andrew Shaffer introduced it as a combination of development and operations. The term gained broader recognition in 2009 at the Velocity Conference, during a presentation [50], [51]. That same year, the first devopsdays event was held in Belgium, marking the beginning of what has become an acclaimed conference series, covering topics of software development and IT infrastructure operations [52].

As this concept gained traction, it became evident that its adoption was motivated by the growing need to address the inefficiencies of traditional practices and accelerate software delivery, including both development and operations, promoting collaboration and automation as principles. Thus, several companies face difficulties in successfully delivering software development projects, based on the challenges in product development and delivery. Despite these struggles, organizations recognize the

importance of software application development for their business success, while only 25% of these companies consider that their teams are efficient.

These inefficiencies were intensified with the evolution of software systems. Traditional software development systems relied on “systems of record”, which were stable, large-scale applications that changed infrequently and prioritized reliability. However, with the growth of mobile and web applications, companies were required to deliver “systems of engagement” that are customer-facing, interactive and dynamic. These systems required constant updates and agile answers to constantly changing customer requirements and market conditions, making traditional systems obsolete.

DevOps provides a solution by integrating agile principles throughout the software development lifecycle. It focuses on increasing the efficiency and speed of delivering a product or services, allowing companies to quickly respond to customer feedback, as well as to adapt to market demands. DevOps drives business value in three fundamental areas:

- Enhancing customer experience.
- Increasing the capacity to innovate.
- Reducing the time to value.

By cultivating a culture of automation, continuous delivery and quick feedback loops, DevOps enables companies to meet the growing demands of modern software systems, while maintaining quality and reliability of the systems [4].

DevOps relies on a specific set of practices and tools tailored to its methodologies. Based on an investigation carried out by a group of researchers, the three most common practices in DevOps methodologies are:

- “Automated and continuous deployment throughout entire pipeline”.
- “Make small and continuous releases”.
- “Developers get feedback based on releases” [53].

Furthermore, the tools that support DevOps methodologies are categorized into:

- Infrastructure as a Service / Platform as a Service (IaaS/PaaS), such as Amazon Web Services, Microsoft Azure or Google Cloud Platform, that provide the foundational infrastructure.
- Continuous Integration (CI) tools like Jenkins, Travis CI or Circle CI.

- Continuous Deployment (CD) tools, such as Ansible, Chef or Puppet, facilitate streamlined workflows.
- Provisioning and Configuration Tools like AzureRM, CloudFormation and Terraform help automate infrastructure management [54].

2.3.2. Cultural Transformation

The implementation and adoption of DevOps is not only a technical initiative but also a cultural transformation, promoting changes in how people think and work. At its core, DevOps aims to break down silos and remove barriers between the development and operation teams, promoting collaboration, communication and shared responsibility throughout the software development lifecycle. This cultural shift aims to reduce the delays and errors, while aligning architectural, design, development and operational needs. Achieving such transformation requires an evaluation of the organization structure and a commitment to continuous learning and adaptation [55], [56], [57].

In addition, DevOps culture promotes continuous improvement, where mistakes are viewed as opportunities to learn and grow rather than setbacks. By embracing this mindset, companies can encourage resilience and adaptability, enabling them to respond effectively to the demands of a fast-changing market [57].

The adoption of DevOps culture presents several challenges that companies must overcome to achieve a successful implementation. One principal challenge lies in changing corporate culture, by defining new values of collaboration, openness to innovation and adaptability to change often face resistance. This changes often encounter resistance from employees and leadership, who are used to traditional mindsets. Addressing this challenge require guidance and training programs, focusing on technical skills but the behavioral changes.

Another obstacle lies in preparing employees to adopt new methodologies and tools. Training programs must go beyond technical proficiency, as well as to equip staff with the knowledge and mindset to embrace automation and interactions between teams. Additionally, managing organizational structure changes required for DevOps adoption can be complex, as it may require new structures, redefining roles and integrating new workflows requires changing management strategies to minimize resistance and ensure a smooth transition.

These challenges underline the importance of leadership in driving cultural transformation. Leaders must support the shift by promoting an environment that values collaboration and continuous

improvement. With strategic planning, tailored training and leadership engagement, companies can address these challenges and realize the benefits of adopting DevOps culture [56].

2.3.3. Advantages and Challenges

Even though the adoption of DevOps has been growing in the IT, these practices have advantages and challenges, which organizations must consider when determining if DevOps aligns with their methodologies. To summarize, DevOps offers several operational advantages, such as improved efficiency, enhanced quality, stability, scalability and continuous feedback. It is also associated with disadvantages, including challenges with transitioning from legacy systems, reliance on multiple tools that complicate integration and cultural resistance due to organizational changes required [54]. To further illustrate, Table 1 highlights the operational advantages of DevOps, such as automation, continuous integration and scalability, alongside challenges like overcoming cultural resistance and transitioning from legacy systems. This balance highlights the need for careful planning and organizational alignment to fully leverage the DevOps capabilities.

Table 1 - DevOps Advantages and Challenges [54]

Advantages	Challenges
• Ability to use the Cloud and database management, integrating technologies with the Cloud. • Automated deployments to the Cloud. • Automated, real-time monitoring. • Automated, scalable and repeatable processes. • Carry out automatic Quality tests. • Carry out continuous integration (CI). • Carry out planned deployments. • Carry out planned tests. • Extremely scalable resources. • Fast delivery, using the build-test-deploy cycle. • Parallel deployments. • Perform code version control. • Real-time pipeline visibility.	• Overcoming the 'Dev vs Ops' mentality. • Migrate from legacy infrastructures to microservices, as outdated infrastructures and applications can be problematic. • Too much focus on tools, relying on various tools to build pipelines, and integrating them can be complicated. • Adopting DevOps can present changes for teams and stakeholders, as it is a drastic evolution. • Clashes in the tools used by Dev and Ops, given the differences.

• Recurring feedback from logs and dynamic learning. • Rollback of code and continuous planning. • Secure pipeline using authentication tools.	

2.3.4. DevSecOps

As companies adopt DevOps practices to streamline software delivery and efficiency, ensuring security has become a fundamental challenge. In traditional software development lifecycles, security was overlooked or treated as a non-functional requirement, typically addressed in the later stages of software development. This approach relies on time-consuming activities, which could create bottlenecks and delay the speed of deployments. According to a survey conducted by HPE Security Fortify team in 2016, even though many companies acknowledge the importance of integrating security into DevOps, only a small fraction have successfully done it. Gartner estimates that less than 20% of Enterprise Security Architects engage in their organizations' DevOps initiatives systematically incorporating security practices.

In this context of DevOps, this challenge is intensified by the perception of security as a barrier to the agility and speed that DevOps practices, such as Continuous Integration (CI) and Continuous Deployment (CD) aim to achieve. As a result, many developers, managers and operations view security measures as obstacles. [58], [59], [60].

The growing need to integrate security into DevOps has led to the creation of the term DevSecOps, combining Development, Security and Operations. At its core, DevSecOps emphasizes prioritizing security, by embedding security controls and practices into the DevOps lifecycle. As the demand for rapid deployment of secure and reliable solutions continues to rise, the interest in this concept has grown significantly [59], [60].

2.3.5. DevOps Market Analysis

As DevOps continues to evolve, its impact on the software development industry has become prominent. As it can be seen in Figure 5, recent studies estimate that the DevOps Market has surpassed $8 billion in total investments in 2022 and is projected to reach a Compound Annual Growth Rate (CAGR) of 20% in 2023. By 2032, it is expected to reach $70 billion, driven by the growing demand

for agile software delivery, automation, and the optimization of development and operations processes [61]. As companies try to remain competitive, future trends will focus in integrating advanced technologies, prioritizing security, and adapting to complex IT environments.

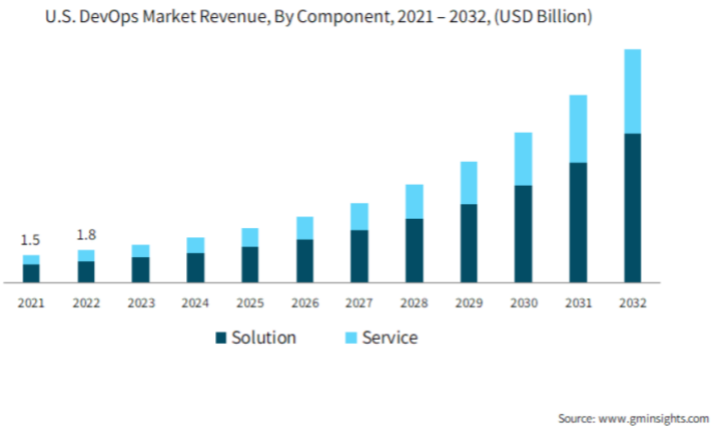


Figure 5 - U.S. DevOps Market Revenue [61]

2.3.6. Association with Platform Engineering

DevOps, as a practice, aims to bridge the existing silos and misalignments between Development and Operations, by promoting collaboration, automation, and integration. However, as organizations have scaled and systems have grown more complex, the limitations of traditional DevOps became clearer. With many organizations finding themselves "stuck in the middle of their transformation" with various tool choices and a stagnated developer productivity. These challenges leave organizations and team struggling to keep pace with evolving demands.

This is where Platform Engineering complements DevOps, by addressing the complexity and operational inefficiencies that may be encountered. Platform Engineering presents curated self-service internal platforms, that standardize workflows, enabling the development teams to deploy and manage their own infrastructure, without the unnecessary overhead of directly interacting with it. By abstracting the complexity and creating streamlined environments, Platform Engineering enhances the speed and reliability of DevOps processes, empowering the teams to focus on delivering value to the business.

While DevOps underlines collaboration and process automation, Platform Engineering operationalizes these principles through structured IDPs, that prioritize developer productivity and scalability. Together, it sets up a strategy for addressing modern software development challenges, as well as to ensure a faster delivery, enhanced efficiency and sustained innovation [2], [62], [63].

2.4. DevOps Research and Assessment (DORA)

DevOps Research and Assessment (DORA) metrics are a set of key performance indicators (KPIs) created by Nicole Forsgren, Gene Kim and Jez Humble to quantify and compare software delivery performance in DevOps organizations. These metrics were popularized through the annual State of DevOps Report and are currently widely accepted by the software industry as a powerful method to measure and benchmark DevOps performance. As a result, many DevOps-focused companies now rely on these metrics as part of their performance assessment strategies [62].

2.4.1. DORA Metrics

Research has shown that four core metrics are strongly associated with IT performance and success in software delivery. These metrics are divided into throughput (Deployment Frequency and Lead Time for Changes) and stability (Change Failure Rate and Time to Restore Service). Each metric is defined as the following.

- **Deployment Frequency**: This metric measures how often an organization deploy new code or releases to production, typically expressed as the number of deployments during a specific period. A high deployment frequency indicates an efficient and responsive delivery process.
- **Lead Time for Changes**: This metric captures the time required for code changes to be deployed to production, measured from the commit and to the release and it is typically measured in day. When lead time is long, it often signals the presence of constraints or inefficiencies in the development or release workflow.
- **Change Failure Rate**: This metric represents the percentage of production deployments that lead to failures. A low failure rate demonstrates strong testing practices and reliable implementations, whereas a high failure rate highlights potential instability or quality gaps in the deployment process.
- **Time to Restore Service**: Also referred to as Mean Time to Recover, this metric measures how quickly an organization can recover from a production failure or incident. A short restoration time underscores the effectiveness of incident response and system resiliency, on the other hand prolonged recovery times indicate opportunities for improvement in response and operational readiness [62], [63], [64].

2.4.2. The Relevance of DORA Metrics

The relevance of DORA metrics is based on its ability to enable organizations to make data-driven decisions and to provide them objective and measurable insights, which highlight areas for improvement in the software development workflows, release processes and CI/CD. By focusing on these metrics, organizations can assess the progress and success of engineering initiatives, better understand the Return on Investment (ROI) and the business impact of their projects. Additionally, DORA metrics simplify the identification of performance trends across the software development lifecycle, helping teams to discovery bottlenecks in their delivery processes and even enhance developer productivity and experience. Furthermore, these metrics offer a methodical approach to identify and promote best practices within teams, ultimately enhancing overall organizational performance [65], [66].

2.4.3. Association with Platform Engineering

The integration of Platform Engineering reveals both benefit and challenges. Research from Google Cloud DORA highlights that implementing IDPs typically results in significant benefits, including an approximate 6% increase in organizational performance, 8% higher individual productivity and 10% improved team performance. However, this progress is complemented by a potential reduction in throughput and stability, with reported decreases of 8% and 14%. These initial obstacles often represent temporary friction, common during the early adoption stages, which typically diminish as the platform matures and feedback mechanisms improve [67], [68], [69].

2.5. Platform Engineering

Platform Engineering has emerged as a discipline focused on designing, building and operating an IDP, to reduce the complexity in modern software delivery. According to the author Evan Bottcher, a platform is “a foundation of self-service APIs, tools, services, knowledge and support which are arranged as a compelling internal product. Autonomous delivery teams can make use of the platform to deliver product features at a higher pace, with reduced co-ordination” [70]. By applying this concept, IDP enable companies to overcome fragmented infrastructure, processes and tools, by providing developers a unified environment that streamlines the application lifecycle.

These platforms serve as software-based abstractions, tailored to operational needs. Platform Engineering is regarded as one of the most significant trends in software and considered as the future of software delivery. Its main goal is to manage the complexity of modern systems, by standardizing toolchains and workflows, while promoting self-service capabilities for engineering teams. In the cloud-native era, where the extensive set of available technologies can overwhelm the developers, Platform

Engineering offers more structured approach. Rather than leaving the teams to navigate an expanding range of tools, it incorporates "Golden Paths", which VMware describes as well-defined development routes to reduce complexity and speed up delivery [71]. These paths streamline the development workflows and promote consistency throughout the software delivery process [29], [72].

Figure 6 from Gartner illustrates how Platform Engineering teams abstract underlying infrastructure complexity to deliver unified IDP, consumable by both product and service teams. Starting at the bottom, the Infrastructure Platform layer lays above the infrastructural complexity, providing the foundational capabilities and standardized interfaces. Building on this foundation, the Digital Platform layer incorporates tools, platform services, reusable components and organizational knowledge. These elements appear through the Developer Portal and packaged as Anything-as-a-Service (XaaS) offerings, simplifying the consumption and integration into the delivery workflows. By structuring the platform in a layered approach, Platform Engineering ensures that engineering teams access standardized, curated resources, while remaining protected from infrastructure details.

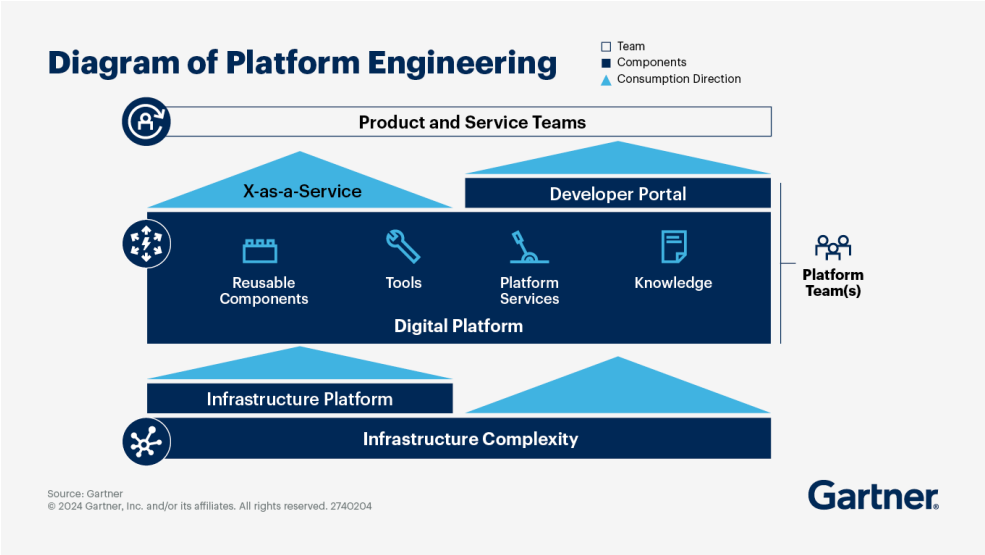


Figure 6 - Diagram of Platform Engineering [73]

To succeed, Platform Engineering relies on four key pillars, which together ensure that platforms can address complexity. These pillars are:

- **Product**: Treating the Platforms as a Product ensures that the features, UX and support align with the user (developer) needs, leading to continuous improvement.
- **Development**: Implement software-based abstractions and workflows to streamline the application development lifecycle and processes.
- **Breadth**: Serving a broad base of application developers maximizes the platform's impact by standardizing the best practices across teams and reducing duplicated efforts.

- **Operations**: Establishing a stable backbone, including consistent practices for CI/CD pipelines, observability and security integration, supporting reliability and scalability[29].

Platform architectural patterns provide a foundational blueprint for building IDP, assist teams to design and integrate components efficiently. As showcased in Figure 7, these patterns are structured based on five primary planes and each of these planes serves a unique purpose in streamlining operations:

- **Developer Control Plane**: Acts as the main interaction layer for developers, offering tools like portals.
- **Integration and Delivery Control Plane**: Focuses on CI/CD pipelines, images registries, and orchestrations tools to automate delivery workflows.
- **Resource Plane**: Incorporates clusters, storage, databases and networking resources, which are the underlying infrastructure for the applications.
- **Monitoring and Logging Plane**: Ensures real-time observability for systems and applications.
- **Security Plane**: Handles secrets management, identity and access management, and policy enforcement to protect the platform.

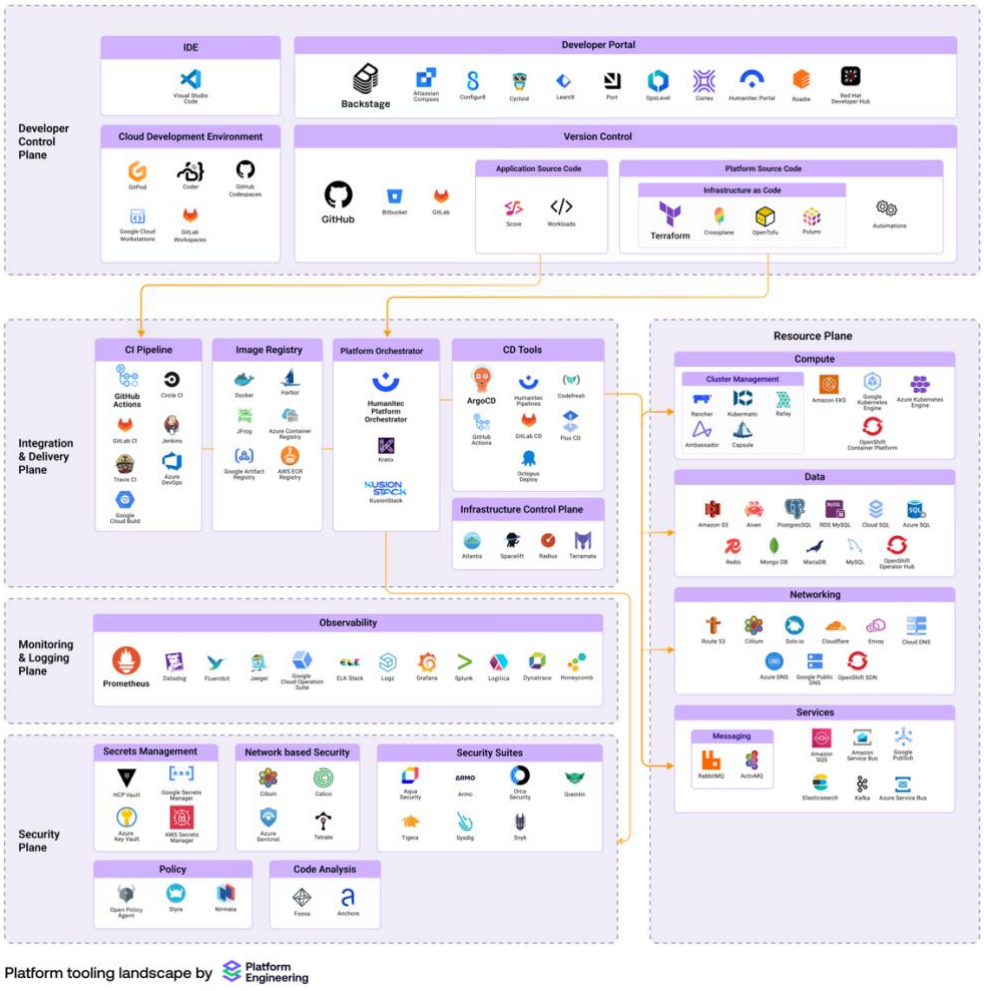


Figure 7 - Platform tooling landscape [74]

2.5.1. Growth and Adoption

Platform Engineering is facing a rapid growth and adoption, as demonstrated by various analyzes and reports. Figure 8, the Gartner Hype Cycle for Software Engineering, provides an exciting visualization of this trend. In 2022, Platform Engineering was positioned in the “Innovation Trigger” phase, representing an emerging but promising concept. Figure 9 shows that by 2023, its trajectory had boosted it into the “Peak of Inflated Expectations”, a phase that shows the increasing recognition, exploration and experimentation of this discipline within the industry [72].

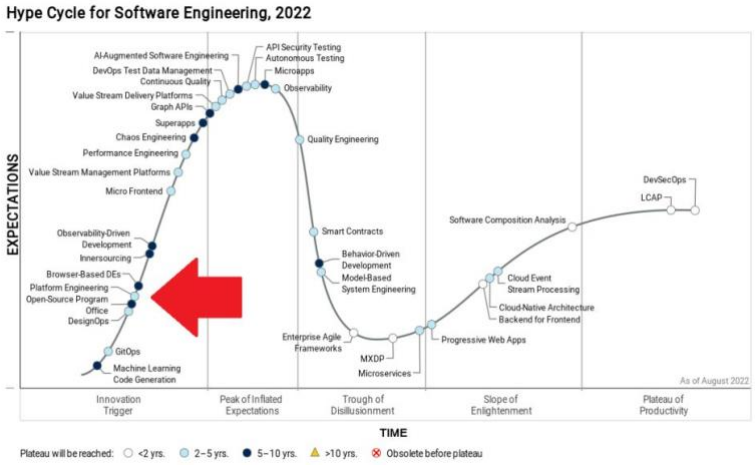


Figure 8 - Gartner Hype Cycle for Software Engineering 2022 [72]

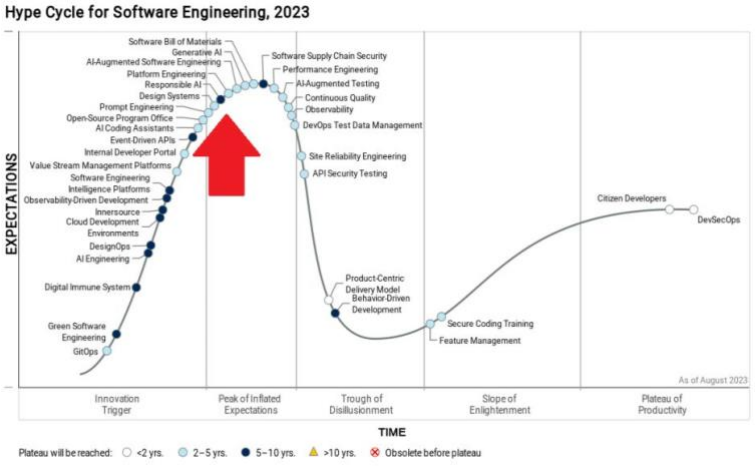


Figure 9 - Gartner Hype Cycle for Software Engineering 2023 [72]

In addition to its growing recognition, by 2024, Platform Engineering has achieved its own dedicated Hype Cycle, as shown in Figure 10, highlighting how important it has become in the technical landscape. It is not just limited to one area, Platform Engineering plays a big role in other areas such as Cloud Computing, Site Reliability Engineering (SRE), Infrastructure and Operations. This broader adoption shows how crucial it has become for solving the challenges that exist in modern software delivery and making systems simpler to manage [75].

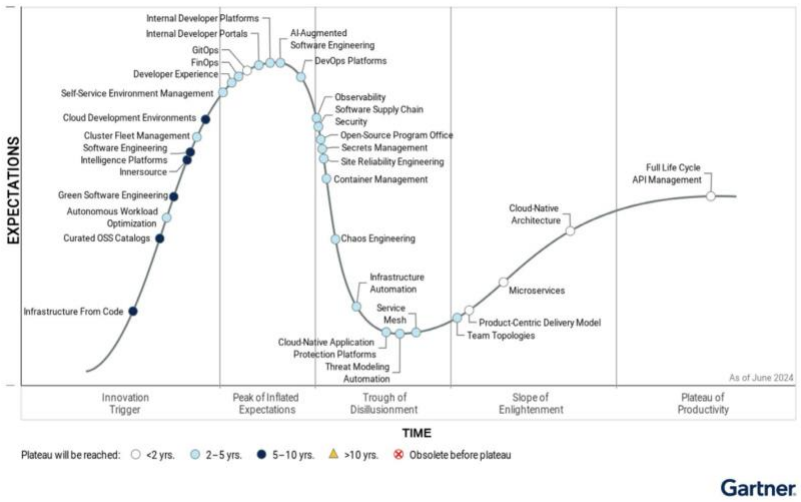


Figure 10 - Gartner Hype Cycle for Platform Engineering 2024 [75]

2.5.2. Benefits

The insights presented in the following analysis are based on a survey conducted by Puppet, with inputs from 438 participants globally. This survey served as the baseline shows the significant advantages that Platform Engineering brings to organizations, addressing critical challenges in software development and operations areas.

Platform Engineering offers a broad range of organizational benefits, beyond only improving the development speed. Key advantages described by respondents include:

- 60% of the respondents highlighted an improvement in the system reliability as a main advantage.
- Enhanced productivity and efficiency with 59%.
- Expedited delivery times with 58%.
- Improved workflows and process standards with 57%.
- Strengthened security measures with 55%.
- Reduced duplication of work through standardization with 53%.

These findings reinforce the holistic impact of Platform Engineering, extending its influence across various aspects of an organization's technical landscape [76].

Additionally, the data also emphasizes a shift in how Platform Engineering is perceived, moving beyond the traditional roots of development and operations, as 55% of the respondents cited an improved performance, aligning with the industry trend of integrating the DevSecOps principles into Platform Engineering. This evolution demonstrates that standardizing infrastructure and processes accelerate delivery but also simplify the collaboration of Security Teams in the process.

Further supporting these findings, the survey results provide insights that diverse stakeholders benefit from Platform Engineering. The findings reveal that:

- 30% identified the entire company as the primary beneficiary of Platform Engineering.
- 29% stated that Developer teams benefit the most of Platform Engineering.
- 19% highlighted that Platform Engineering serves the needs of the entire department.
- 14% indicated that Infrastructure teams are the main beneficiaries.
- 8% pointed that individual Developers are the primary group served.

These findings emphasize the extensive applicability of Platform Engineering, demonstrating its ability to address both organization objectives and team-specific objectives, while promoting collaboration and efficiency across multiple levels [76].

2.5.3. Evolution from DevOps

While DevOps has transformed the way organizations manage the development and operations, its limitations in addressing the growing complexity of modern systems have become more evident. As highlighted in previous chapters, many organizations "remain stuck in the middle of their transformation", struggling to fully embrace the benefits of DevOps, due to the overwhelming tool choice and stagnation of developer productivity. The increasing demands for scalability, efficiency and streamlined processes paved the way for Platform Engineering, as it complements DevOps, by operationalizing its principles through the creation of IDPs. These platforms abstract the system complexity, offering self-service workflows that empower the Development teams to manage the infrastructure, without the unnecessary overhead [2].

Platform Engineering appears as a natural evolution of DevOps, addressing challenges that arise as systems become more complex and scale. The book Platform Engineering: A Guide for Technical, Product, and People Leaders describes the limitations of traditional approaches, using the metaphor of "Over-General Swamp" to highlight the snowballing burden of maintaining and integrating several tools, systems and workflows. This swamp represents the challenges of DevOps, such as the inefficiency and complexity that grows over time, as applications rely on multiple open-source systems (OSS) and cloud primitives.

While these technologies and tools simplify the process of creating applications, they also increase the cost and effort required for long-term maintenance and adaptability. The "Over-General Swamp" also illustrates how, in several organizations, various teams independently select and integrate tools, resulting in divided systems held together by custom "glue" solutions. The "glue" includes integration code, automation and management tools, that lead to the creation of a fragile architecture that is difficult to scale or modify. Over time, simple updates will require significant effort in updating,

testing and redeploying across interconnected components, showcasing the complications that may delay the development and increase the risk of failures and security vulnerabilities.

Platform Engineering tackles the complexity of the "Over-General Swamp" as well as introduces a structured approach to reduce the operational overhead and technical debt, that traditional DevOps methods often struggle to address. By abstracting the layers of cloud primitives and OSS, Platform Engineering creates centralized solutions that allow for greater standardization and scalability across teams. This abstraction includes the creation of IDP, encapsulating system dependencies and workflows within reusable platforms. Contrasting with the "glue" solutions of the "swamp", these platforms are designed with a customer-focused mindset, tailored to the specific needs of the development teams. This approach helps companies to overcome the bottlenecks of isolated tool adoption and inconsistent integrations, streamlining the deployment processes and enabling smoother migrations and updates.

In addition to reducing the operational complexity, Platform Engineering empowers the collaboration between cross-functional teams, such as Infrastructure, DevTools, DevOps and SRE teams, aligning their efforts towards shared objectives. It takes inspiration from DevOps, but it evolves beyond it by embedding principles of product mindset and system reliability into platform design. The focus switches from simply automating to empowering the development teams leveraging self-service capabilities, allowing them to deploy and manage the applications autonomously, while maintaining the standards of reliability, security and performance. As a result, companies can achieve a greater agility in their development lifecycles, optimize the resource utilization, and better align the infrastructure with the evolving business objectives [29].

2.5.4. Challenges and Competitive Gains

While transformative for modern organizations, Platform Engineering presents several challenges that require careful navigation. One of the most noticeable challenges is managing the increasing complexity of platforms, which can lead to tool duplication, technical debt and difficulties in aligning workflows with evolving user needs. Additionally, gaining the buy-in from teams and developers is an obstacle, as resistance to change may conflict with leadership-driven initiatives. Limited resources, such as budget constraints and acquisition of the necessary resources may intensify the challenges. Communications gaps also represent a significant challenge, especially when platform teams lack dedicated roles, such as product owners, to communicate with users to ensure their needs are met. Furthermore, operational risks, such as system failures and cybersecurity threats, require proactive and continuous management strategies. Finally, human factors, such as skill shortage and resistance to change, reinforce the importance of promoting collaboration and adaptability within teams [77], [78], [79].

Additionally adopting Platform Engineering may result in an approximate 8% decrease in throughput compared to environments without platforms, due to an increasement in latency, caused by the introduction of multiple "handoffs" between different systems for deployment, monitoring tasks, security checks and testing tasks. Furthermore, enforcing exclusive use of the platform for executing tasks throughout the entire application lifecycle may intensify productivity challenges, with reports showcasing an additional 6% decrease in throughput when the use of platforms is strictly imposed without sufficient flexibility or suitability to users' specific needs. To address these risks, platforms should prioritize user-centric design and independent, balancing benefits with the operational tasks faced by the development teams.

Another challenge involves the increase in change instability and burnout, with platforms showcasing a 14% decrease in change stability. This instability translates into higher rates of changed failure and additional rework. Furthermore, the combined the instability and platform usage, correlates with increased levels of burnout. While platforms empower developers to confidently deploy changes through quicker remediation capabilities, this confidence can lead to more frequent and lower-quality changes, necessitating additional rework. Additionally, platforms might fail to ensure change quality, especially if teams prioritize throughput over the use of automated testing features. This scenario increases instability and rework, contributing to burnout [67].

On the other hand, Platform Engineering offers significant competitive gains that can promote organizational efficiency and innovation. By automating repetitive and low-value tasks, companies can enhance teams' productivity and reduce operational costs. The standardization and consolidation of processes and tools improve the reliability, security and reduces the duplication effort, promoting consistency across teams. Furthermore, Platform Engineering enables faster onboarding to new team members and accelerate the deployment of new features into production. Enhance developer experience (DevEx) is another benefit associated with Platform Engineering, empowering teams with self-service capabilities that can simplify workflows and boost teams' satisfaction. Finally, these efficiencies combined contribute to a faster speed to market, ensuring organizations remain agile, competitive and ensures the satisfaction of developers, business leaders and clients [77], [80], [81].

3. State of the Art

3.1. Paper Selection Process

To conduct a thorough investigation into the subject of Platform Engineering a comprehensive search strategy was deployed across academic databases. The initial phase utilized the robust search functionality of Google Scholar portal, which compiles an extensive collection of academic papers.

During the initial exploration, through all the queries created, the following filters were applied to Google Scholar search:

- Limitations: “Peer Reviewed Articles”, “Full Text”
- Language preference: “English”

The first search that was made to obtain information about Platform Engineering was searched with following query:

Contains ‘Platform Engineering’

With this query, 166 publications were made available, spanning between 1978 and 2024. Such a broad result would be hard to read and analyze properly. Additionally, several articles were not relevant to research encompassing topics such as chemical reactions, viruses, pharmaceutical production, vaccines, neuroscience and other themes related with science and chemistry.

Contains ‘Platform Engineering’ and ‘DevOps’

This query and the inclusion of publications spanning from 2019 to 2024, there was a substantial reduction in the number of articles found, with 12 publications. Following the retrieval of these publications, a multi-stage screening process was performed, by performing Title and Abstract Screening to confirm the direct relevance of the publications. The shortlisted articles were reviewed in full to evaluate their contribution and relevance to the Literature Review. Upon completion three publications were considered relevant, providing direct insights into Platform Engineering.

Given the limited number of relevant articles identified during the initial search on Google Scholar, a second exploration was conducted using ResearchGate to expand the pool of literature on Platform Engineering. The search strategy was refined and applied with the following query:

- Language preference: “English

Contains ‘Platform Engineering’

This query returned a total of 200 publications. A multi-stage screening was performed, starting with a title and abstract review to assess the relevance of the publications. Following this step, the results

were narrowed down to eight publications. Of these, only six were publicly accessible, from which three articles were considered relevant.

3.2. Literature Review

Platform Engineering is rapidly gaining recognition for its crucial role in modern technological landscapes, evolving as a refined focus within the context of DevOps. The State of DevOps Report 2023 by Puppet highlights that many organizations have moved beyond the traditional DevOps, adopting Platform Engineering as a mature discipline that enables fast-flow software delivery through the design and building of self-service capabilities [82]. This discipline focuses on creating modular and scalable solutions that promote collaboration, drive innovation and optimize supply chain processes. By integrating Platform Engineering, manufacturers can interconnect distinct systems, facilitating seamless coordination across their operations, which offers tangible benefits such as reduced product development cost and lead time, improved product quality and increased product variety. The focus on modularity and reusability provides a foundation for adaptability and rapid response to dynamic market demands.

A case study of a manufacturing company in the United Arab Emirates demonstrates the transformative potential of Platform Engineering. This organization faced challenges including managing complex data sets and adapting to an evolving a market. By adopting Platform Engineering, the organization enhanced efficiency by connecting several manufacturing systems and promoting coordination and collaboration between teams. A scalable and secure data infrastructure was established, enabling real-time analytics and empowering data-driven decision-making. Additionally, automated workflows allowed teams to focus on core expertise, accelerating software development and improving overall organizational performance. Security was strengthened through enhanced procedures, ensuring a robust protection against threats.

Beyond operational improvements, Platform Engineering offers strategic value to the manufacturing sector, enabling the creation of adaptable ecosystems that support collaboration, experimentation and innovation. Investments in technologies such as Cloud Computing and data analytics allow organizations to connect distinct systems, promote efficient coordination and decision-making processes [83].

Platform Engineering is crucial to address the complexities of multi-cloud environments, acting as a unified framework that integrates multiple cloud platforms seamlessly while ensuring interoperability, security and optimal performance. The deployment and maintenance of multi-cloud solutions require robust toolsets and methodologies to abstract the complexities of managing different cloud providers. By facilitating orchestration, resource optimization and the creation of standardized

frameworks, Platform Engineering empowers organizations to leverage the strategic advantages of multi-cloud, including enhanced flexibility, resilience and competitive edge.

An important contribution of Platform Engineering within a multi-cloud context lies in addressing integration challenges. By utilizing standardized interfaces, APIs and data formats, Platform Engineering enables seamless communication between different cloud platforms. Furthermore, tools such as Kubernetes and Terraform streamline cross-cloud orchestration and infrastructure management, significantly reducing the complexity of deploying and managing applications across multiple cloud environments [84].

Automation and security are integral components of Platform Engineering. Automation tools, such as Infrastructure as Code (IaC) implementations like Ansible and Terraform, empower organizations to efficiently design, deploy and manage resources, ensuring consistent and repeatable deployment processes across platforms, while minimizing manual intervention and human error. Orchestration tools, such as Kubernetes, provide the capability to effectively manage containerized applications. At the same time, Platform Engineering integrates robust practices, including unified identity management systems, encryption standards and compliance automation frameworks, to enforce consistent security policies across different cloud environments.

Platform Engineering enables organizations to adopt cutting-edge technologies such as AI, ML and edge computing. These technologies significantly influence Platform Engineering by introducing new practices and standards that empower organizations to leverage predictive analysis, intelligent workload distribution and enhanced resource management. Furthermore, standardization and automation are critical in overcoming the challenges of multi-cloud. Best practices such as consistent configuration management, automated scaling and centralized monitoring are fundamental to maintain operational consistency and ensuring smooth operations [84].

As emphasized by the author, Platform Engineering significantly enhances developer satisfaction and productivity. By leveraging IDPs, organizations can streamline development workflows, provide pre-configured environments and automate tasks that would otherwise require manual effort from the development teams. These practices enable organizations to simplify the setup and deployment, as well as to empower developers to focus on writing code rather than managing infrastructure components [85].

Research has demonstrated substantial productivity gains across various project types following the adoption of Platform Engineering. For instance, the projects showcased in the article, which were Web Application Development, Mobile Development, API Services, Data Pipelines and Security Tools have reported productivity improvements ranging from 40% to 50%. For reference, the Mobile Development project experienced a 50% reduction in man-hours, decreasing from 300 to 150,

highlighting the efficiency gains achieved through the automation of repetitive tasks and standardization of project setups.

Regarding developer satisfaction, research showcases that standardizing templates and automate provisioning through Platform Engineering plays a crucial role in enhancing the developer experience, as this research shows a significant improvement in satisfaction scores, ranging between 42% and 50% across different projects [85].

A SWOT analysis conducted with insights from six specialists at Amazon and Google provides insights into the potential and challenges associated with Platform Engineering. The strengths identified include capacity to accelerate time-to-market and foster innovation. Furthermore, by promoting collaboration across teams through unified frameworks, Platform Engineering helps minimize inefficiencies.

However, the analysis also highlights weaknesses that organizations must address, such as maintaining consistency across teams, balancing customization with standardization, and managing dependencies between platform components can increase operational complexity [86].

Platform Engineering presents promising opportunities, such as the increasing emphasis on automation, scalability and security, throughout the evolution of the concept. Emerging technologies, such as AI and ML, offer exciting capabilities to strengthen practices and methodologies, and Platform Engineering will continue to evolve, become more complex and with more features and capabilities.

Despite its strengths and opportunities, Platform Engineering faces external threats that could hinder its adoption and growth. The rapid pace of technological advancements and constantly shifting trends may necessitate continuous adaptation of Platform Engineering practices. Competition from other organizations developing similar or superior platforms also poses a challenge. Moreover, effectively managing the growing complexity of platform ecosystems remains a critical concern [86].

Platform Engineering role extends beyond infrastructure management, including operational efficiency, particularly for small organizations with limited resources. Efficiency is achieved by optimizing computational and human resources, minimizing cost and improving performance. This is complemented by flexibility, allowing platforms to adapt to evolving workflows, integrate with third-party services and align with shifting business strategies. Security is crucial pillar, integrating robust practices such as encryption, access control and regular audits to safeguard sensitive information.

A multi-layered architecture for an IDP is presented as a key enabler of successful Platform Engineering. This architecture contains several layers, each addressing distinct functionalities essential to development workflows. The User Interface (UI) layer provides developers with a seamless and user-friendly experience for accessing resources, tools and documentation. The API layer ensures secure and efficient communication between the frontend and backend systems. The Data Management layer

handles operations such as storage, retrieval and updates, ensuring data integrity and accessibility. The infrastructure layer supports the entire platform with robust computing, networking and security tools and mechanisms [87].

3.3. Final Considerations

Platform Engineering has emerged as a critical area of focus in modern software development, offering transformative potential in addressing the complexities of DevOps and multi-cloud environments. As evidenced by the literature reviewed, Platform Engineering facilitates the creation of modular and scalable solutions, enhance collaboration, optimize supply chain processes and empower organizations to leverage multi-cloud strategies effectively. Case studies, such as the manufacturing company in the United Arab Emirates, demonstrate the practical benefits of adopting Platform Engineering, including increased efficiency, data-driven decision-making and improved security.

The incorporation of cutting-edge technologies such as AI, ML and edge computing, further extends the potential of Platform Engineering, as these advancements enable organizations to leverage predictive analytics, optimized resource management and standardized practices across diverse cloud environments.

The discipline contributes to improving developer satisfaction and productivity, by providing pre-configured environments and automating repetitive tasks, allowing developers to focus on code development instead of managing infrastructure components. Research highlights important productivity gains when leveraging from Platform Engineering and IDPs, ranging from 40% to 50% across different project types, with significant improvements in developer experience associated to automation and template standardization.

Despite its advantages and strategic potential, Platform Engineering remains a relatively new and evolving field, leading to a limited availability of peer-reviewed research focused on Platform Engineering and a lack of in-depth studies examining its application across diverse industries and use cases. While core concepts such as DevOps, containerization, IaC and orchestration are well-established, their integration and impact within the context of Platform Engineering practices require further investigation.

Future research should aim to bridge the gaps by expanding upon the foundational work outlined in the literature. Moreover, as technological demands continue to grow, the development of standardized frameworks and best practices in Platform Engineering will be essential to ensure scalability, efficiency and relevance in rapidly growing environments. This chapter emphasizes the need for continued study and refinement of Platform Engineering, positioning it as a critical enabler of digital transformation in the modern technological landscape.

4. Project

4.1. Problem Identification

In the rapidly evolving landscape of software development, development teams are responsible not only for building applications but are also increasingly burdened by the growing adoption of CI/CD workflows, Cloud Computing, containerization and infrastructure management processes. As applications become more complex, developers often face the challenge of managing infrastructure configurations, deployments and operational processes across multiple cloud providers, such as AWS and GCP. This results in increased cognitive load, fragmented workflows and slower development cycles.

Platform Engineering has emerged as a promising solution to these challenges by providing a self-service and standardized IDP. Tools such as Atlassian Compass, Backstage, Humanitec and Port are designed to abstract the complexity of the underlying infrastructure required to host applications, facilitating autonomous development processes. By offering a user-friendly interface, these platforms enable development teams to seamlessly and efficiently provision the necessary components for deploying their applications.

With an IDP, developers no longer need to master multiple infrastructure-related technologies such as Terraform, Ansible, CI/CD pipelines and Kubernetes, tools that are traditionally required to create and manage cloud environments. Instead, they can focus on application development and deployment within a controlled cloud environment, thus reducing cognitive load and increasing efficiency.

The motivation behind this project is to implement and evaluate an IDP prototype by deploying Backstage locally and integrating it with GitHub and GitHub Actions. Backstage was selected as the central component due to its distinction within the Platform Engineering landscape, its status as an open-source solution originally developed by Spotify and currently hosted by the Cloud Native Computing Foundation (CNCF) and its high level of customizability and configurability, enabling greater flexibility in this Proof of Concept (PoC) compared to more product-centric alternatives. GitHub will serve as the Version Control System (VCS) repository, storing all components required to deploy and test the prototype, and facilitating automated infrastructure provisioning through CI/CD workflows. Additionally, Terraform will be employed as the IaC tool for deploying resources across multiple cloud providers, including AWS and GCP.

The objective of this research is to assess the impact of Platform Engineering on developer productivity and organizational efficiency. This will be achieved through experimentation and surveys conducted with technology professionals.

By combining theoretical insights with practical implementation, this research aims to bridge the gap between theory and practice by implementing and evaluating an IDP prototype. The goal is to provide empirical evidence on how Platform Engineering can optimize the software development lifecycle, reduce operational complexity and enhance developer efficiency.

4.2. Definition of the objectives

The objective of this research is to design and implement an IDP prototype to evaluate its ability to simplify and automate cloud infrastructure deployment. By leveraging Platform Engineering methodologies, this study aims to assess the impact of IDP on cognitive load, infrastructure management and developer productivity

To achieve this, the project is structured around the following key objectives:

- Develop an IDP prototype by deploying and configuring Backstage as a self-service platform for infrastructure provisioning. This includes integrating GitHub and GitHub Actions as a VCS and automation tool, as well as incorporating Terraform to manage cloud resource provisioning.
- Automate the software delivery process by defining standardized CI/CD pipelines to enable self-service capabilities, allowing developers to request and deploy infrastructure resources with minimal manual intervention.
- Evaluate the effectiveness of the IDP by measuring its impact on developer productivity. The study will compare manual and automated deployment workflows to assess improvements in time efficiency and cognitive load reduction.
- Gather qualitative insights through interviews with industry professionals, including Engineers, Pre-Sales, Delivery Managers and Directors, to understand the perceived impact of Platform Engineering on developers and organizations.

This research seeks to contribute to the field of Platform Engineering by demonstrating how IDP may impact developer experience and organizational efficiency through workflow automation and self-service capabilities. The findings aim to provide empirical evidence of efficiency gains that IDP offer.

4.3. Design and Development

The design and development phase follows a structured approach to implement an IDP, integrating Backstage GitHub Actions and Terraform, aiming to automate cloud infrastructure provisioning. This phase is divided into two major stages:

1. Implementation and Integration of an IDP Prototype
2. Evaluation of the results

The initial stage consists of building a functional PoC by locally deploying and configuring the core components illustrated in Figure 11. This includes:

- Backstage, serving as the central platform that integrates all components, as well as serving as the interface to developers.
- GitHub, integrated as a VCS, ensuring traceability, version management and collaboration across teams.
- GitHub Actions, configured to establish standardized CI/CD pipelines, automating the deployment process and ensuring quick and repeatable provisioning processes.
- Terraform, responsible for provisioning and managing resources on multiple cloud providers.

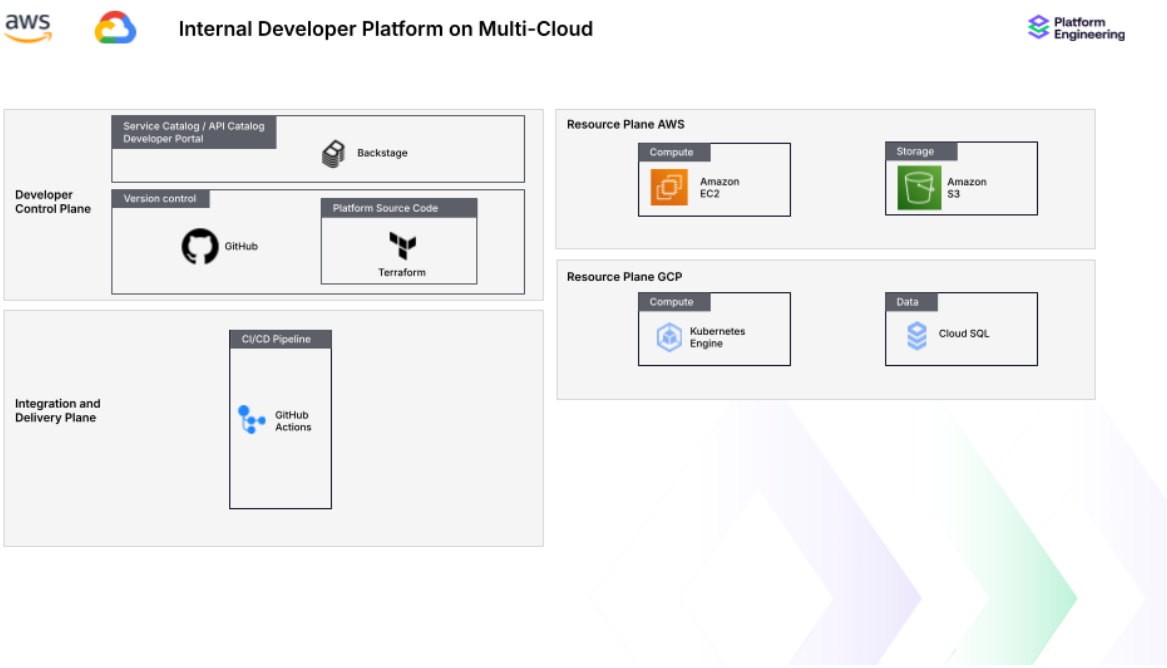


Figure 11 - Project Architecture

The primary goal of the implementation phase is to develop a solid IDP capable of abstracting and simplifying infrastructure complexities, enabling developers to autonomously provision and manage cloud resources, reducing the manual effort as well as the cognitive load which is associated with infrastructure management.

The second phase involves analyzing and evaluating the IDP prototype's success through controlled experimentation, focusing on developer productivity and operational efficiency. Comparative analysis between manual and automated deployment processes will measure factors such as cognitive load, deployment speed and error potential. Additionally, qualitative insights will be

gathered through interviews with a diverse group of professionals, including technical roles such as Cloud, Integration, QA and DevOps Engineers, as well as Cloud and DevOps Architects. Broader organizational perspectives will be collected from Sales Engineers, Delivery Managers and IT Directors to gain deeper understanding and validate perceived usability benefits and organizational impacts across various roles.

4.4. Demonstration

This chapter provides a practical demonstration of the implemented IDP, highlighting how the platform aims to automate and simplify cloud infrastructure provisioning by integrating Backstage, GitHub, GitHub Actions and Terraform. The demonstration targets deployments on multiple cloud providers, specifically AWS and GCP, detailing the necessary steps and processes to showcase the effectiveness, usability and potential benefits of the IDP prototype.

4.4.1. Preparation of the Demonstration Environment

To implement the IDP prototype, several deployment options were considered for Backstage, such as using a cloud-based environment, a dedicated Virtual Machine, containerization or a local deployment. Considering practical constraints and available resources for the PoC, a local deployment was selected. Therefore, the Backstage application was installed on the author's local machine running macOS, which complies with the requirement specified in the official documentation that Backstage only supports Unix-based operating systems.

4.4.1.1. Install Backstage Locally

Several prerequisites are required for successfully installing and running Backstage locally, as detailed in its official documentation, which include packages and tools such as curl or wget, Node.js, nvm, Yarn, Docker, and Git. For a more visual representation of the Backstage installation and configuration process, additional screenshots detailing each step are provided in Annex 1.

To initiate the installation of Backstage standalone application, the following command was executed in the local terminal:

- `npx @backstage/create-app@latest`

Running this command triggers an interactive installation wizard, prompting for the application's name, which will determine the subdirectory where Backstage will be installed.

Upon completion, the wizard creates a dedicated directory for the Standalone Backstage application containing essential files and directories. Among these, the most relevant components are:

- app-config.yaml: the primary configuration file for the Backstage application.
- catalog-info.yaml: the responsible component for defining and managing the entities catalog within the platform.
- Packages directory: which is further segregated into specific subdirectories for the application and backend components.

With Backstage installed, the next step involves running both backend and frontend components to enable local execution of the application. This is achieved using the commands `yarn start` for the frontend and `yarn start-backend` for the backend.

4.4.1.2. Backstage Authentication

By default, Backstage applications provide only a guest sign-in method. However, to enhance authentication, Backstage allows integration with third-party OAuth providers. For this PoC, GitHub was selected as the primary authentication provider due to its central role in the solution architecture. To integrate GitHub authentication, an OAuth application must be created within the intended GitHub account. During this process, the Client ID and Client Secret tokens must be generated and securely stored, as they are essential for enabling authentication integration with Backstage.

Following the official Backstage documentation guidelines, when configuring the OAuth App in GitHub, it is crucial to specify the Homepage URL and the Authorization callback URL to reflect the local deployment environment accurately.

Subsequently, within the app-config.yaml file of the Backstage repository, GitHub must be defined explicitly as an authentication provider, incorporating the previously retrieved Client ID and Client Secret, as it can be seen in Figure 12.

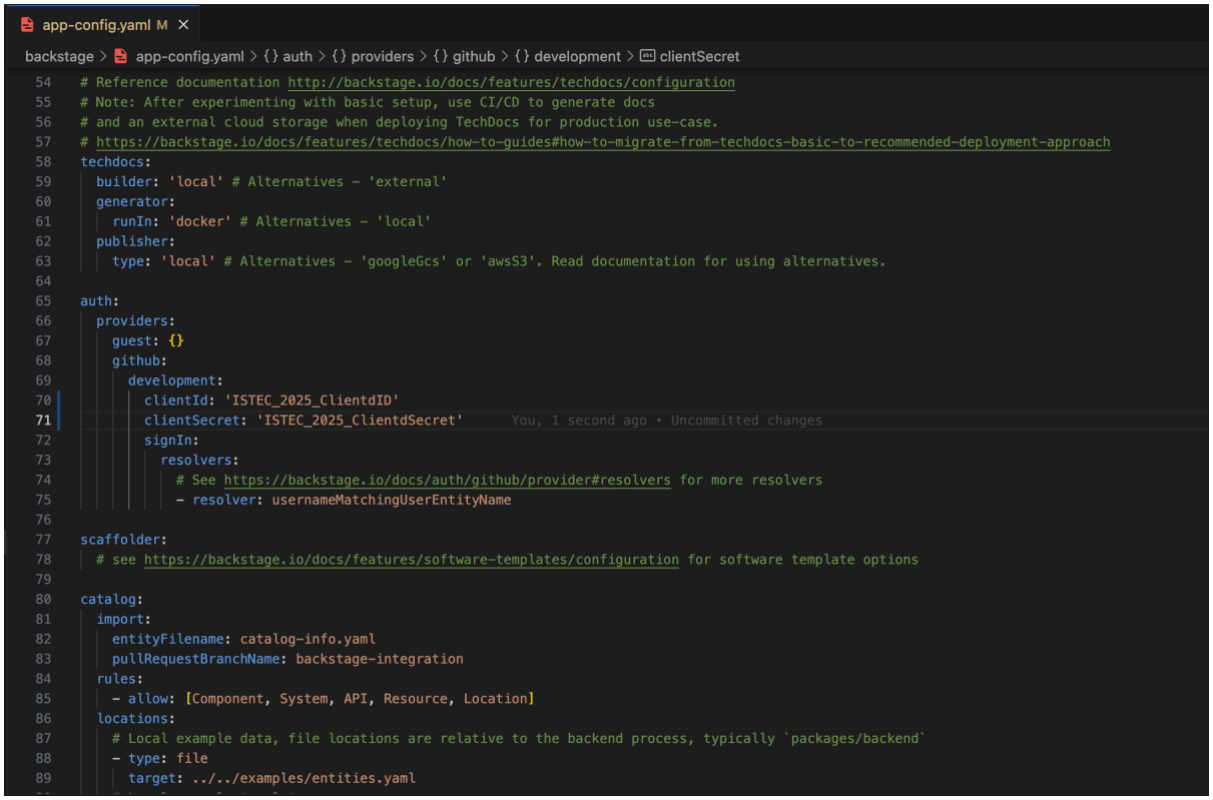


Figure 12 - GitHub OAuth Tokens on Backstage

To complete the integration, modifications are required within the frontend code, specifically in the packages/app/src/ directory. This includes importing both githubAuthApiRef and the SignInPage component as showed in Figure 13.

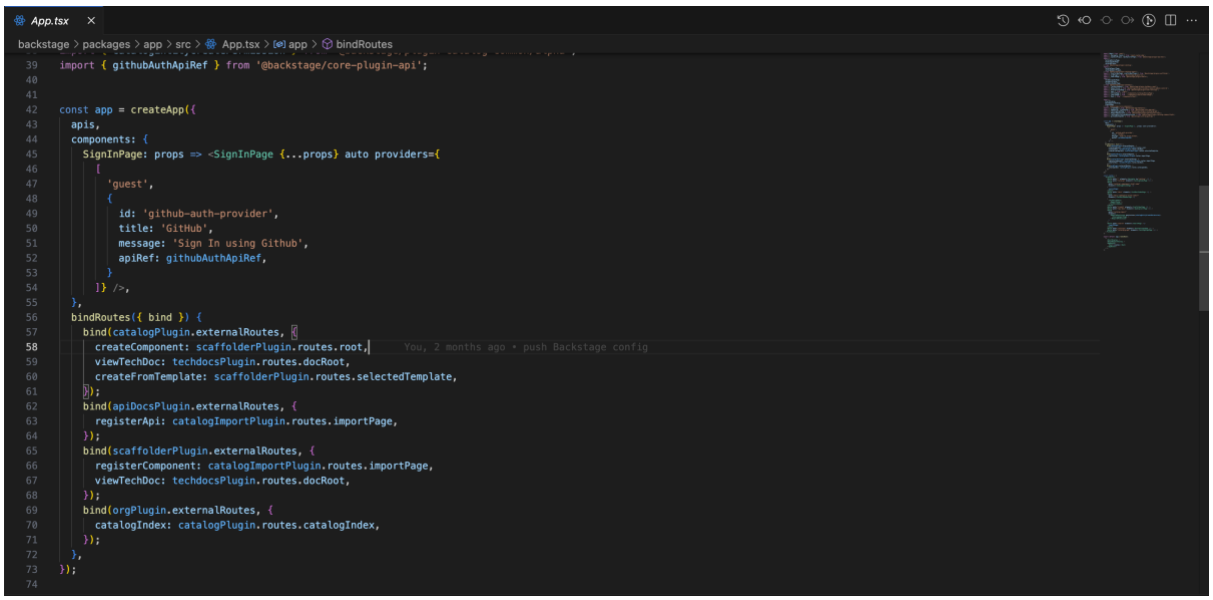


Figure 13 - Backstage Frontend GitHub Authentication

Additionally, configuring the backend is necessary through modifications to the index.ts file as showed in Figure 14. This involves adding the GitHub authentication plugin and developing a custom

authentication resolver. The resolver specifies actions upon user login, potentially including conditional checks or guardrails, ultimately leading to the creation of a JSON Web Token (JWT).

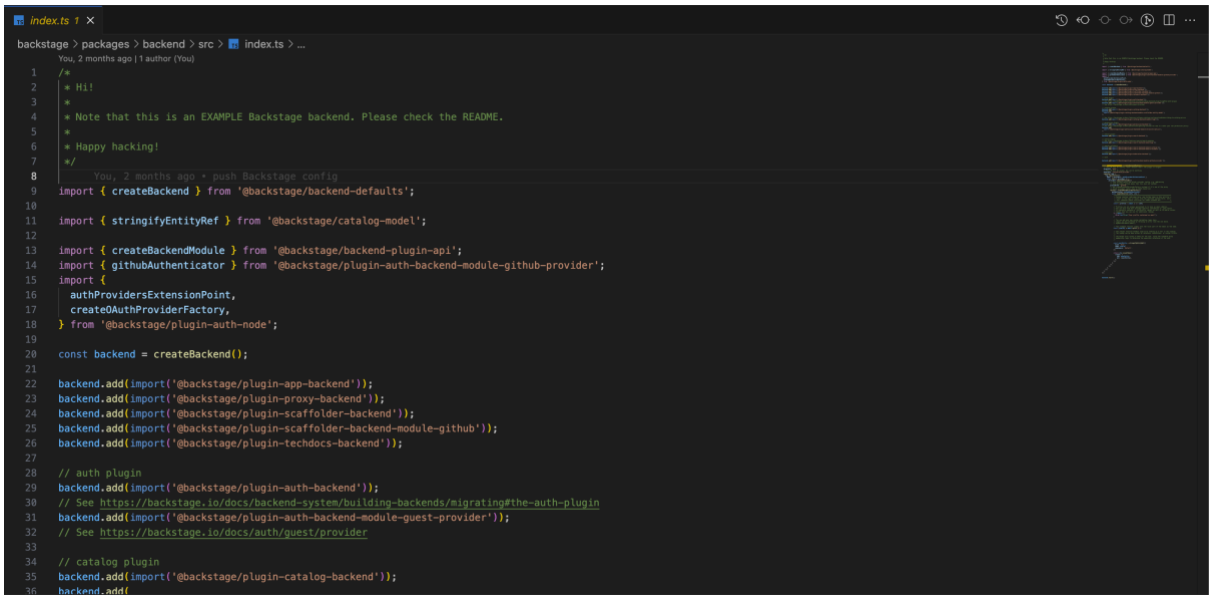


Figure 14 - Index.ts GitHub Auth Configuration

The implemented custom authentication module, retrieves the user's email address from their GitHub account, extracts the prefix (username portion), assigns it as the userEntity, and then generates the corresponding JWT. With these configurations finalized, users can authenticate themselves using either a Guest User or through their GitHub accounts within the Backstage application as it can be verified in Figure 15.

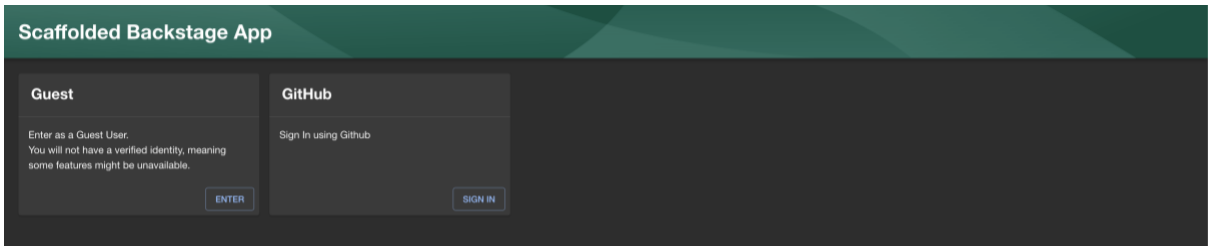


Figure 15 - Backstage UI Authentication

4.4.1.3. Integrate Backstage with GitHub and GitHub Actions

To establish Git integration with GitHub, creating a Personal Access Token (PAT) within GitHub is required. Given that the environment is controlled and locally hosted, the PAT has been assigned full permissions without an expiration date.

Once the PAT is created, it should be integrated into the app-config.yaml file as visible in Figure 16. This integration allows Backstage to interact with GitHub APIs using the PAT, enabling the

application to access and retrieve content from the specific GitHub repository containing GitHub Actions workflows and Terraform scripts.

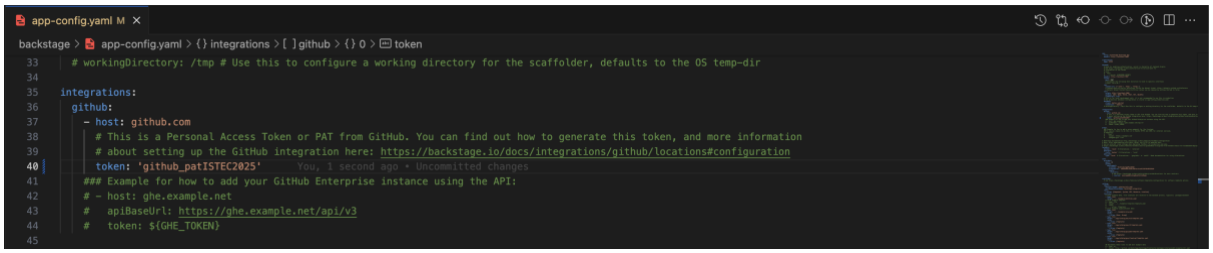


Figure 16 - GitHub PAT on Backstage

For GitHub Actions integration, the following command must be executed to install the plugin into the Backstage application:

- `yarn --cwd packages/app add @backstage-community/plugin-github-actions`

Additionally, modifications to the EntityPage.tsx file, responsible for displaying catalog information, are required. This includes importing GitHub Actions and defining the entity layout route necessary for rendering GitHub Actions within the Backstage interface.

4.4.1.4. Configuring Backstage Catalog

With authentication and integrations configured, the next phase involves enriching the Backstage catalog by providing deployable infrastructure modules for end users. For this PoC, four distinct components were developed, two targeting AWS infrastructure and two for GCP. Each component contains Terraform modules defining the configurations required for deploying specific resources and a corresponding template.yaml file as it can be seen in Figure 17.

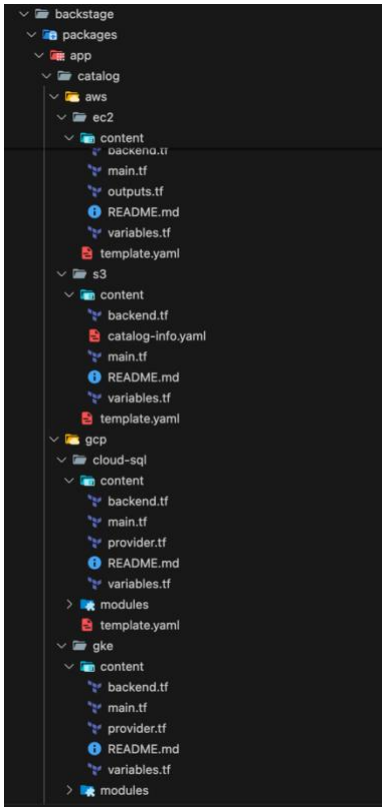


Figure 17 - Backstage Catalog Directory

The template.yaml file integrates all components into the Backstage user interface, specifying relevant information, user input fields and defining automated actions. These actions include triggering GitHub Actions workflows and registering the newly created component within the Backstage catalog.

Finally, dedicated GitHub Actions workflows must be developed and configured to automate infrastructure deployment. These workflows accept authentication details for the targeted cloud providers and execute Terraform commands (`terraform apply` or `terraform destroy`) based on user inputs received via Backstage, effectively managing the lifecycle of cloud infrastructure resources.

To enable Backstage to display the various created templates, it is necessary to reference each corresponding template.yaml file within the app-config.yaml configuration file as described in Figure 18.

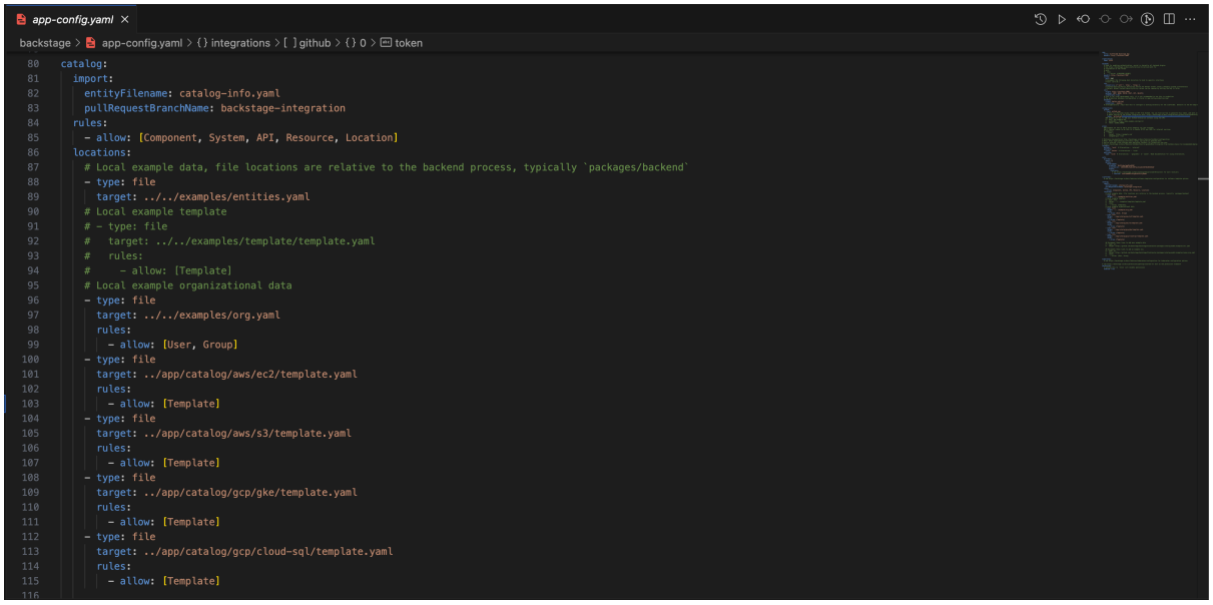


Figure 18 - App-config.yaml Catalog

4.4.2. Demonstration Workflow and Processes

This subchapter details the workflow and processes involved in demonstrating the effectiveness and usability of the implemented IDP. Specifically, it illustrates how users interact with Backstage to provision cloud infrastructure through automated workflows powered by GitHub Actions and Terraform.

Initially, users access Backstage locally via http://localhost:3000 and log in through GitHub authentication, enabled by the previously configured OAuth integration. To provision infrastructure, users can navigate to the "Create" button or side tab, which displays a catalog of available infrastructure templates. In this demonstration, available templates include AWS EC2 Instances, AWS S3 Buckets, GCP Kubernetes Engine Clusters and GCP Cloud SQL Databases, as shown in Figure 19. For a detailed visual representation of the demonstration workflow, including screenshots of the Backstage UI and the complete process for creating infrastructure components, please refer to Annex 2.

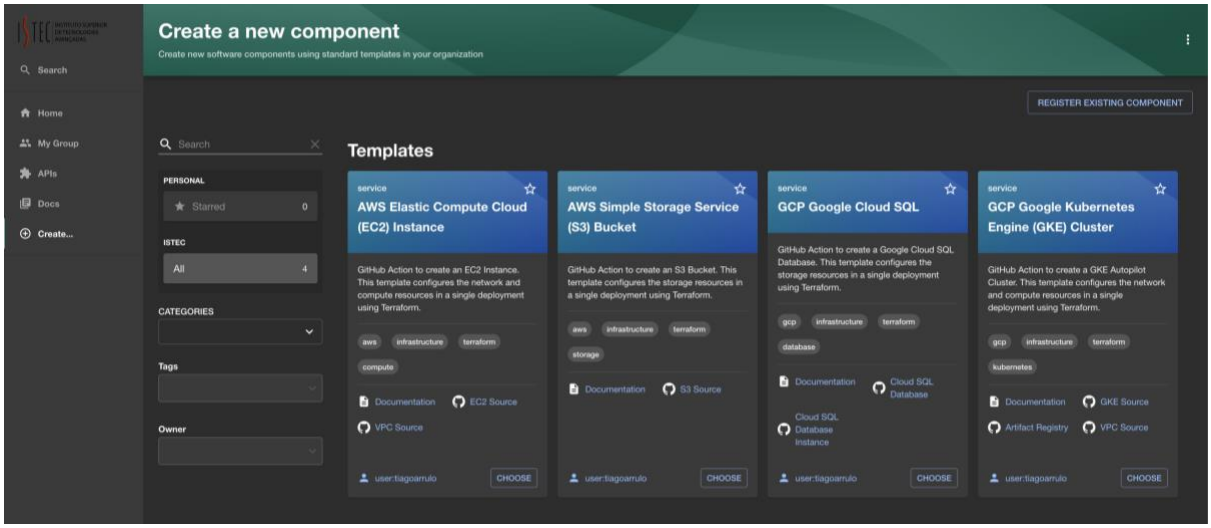


Figure 19 - Create a New Component

Upon selecting a specific template, Backstage prompts users to provide configuration details required by the selected resource type. Typical inputs include parameters such as cloud regions, resource sizes, instance types and networking configurations. For this demonstration, the creation of an AWS S3 bucket is showcased. Each component template involves a three-stage configuration defined within its corresponding template.yaml file. Initially, the user selects the desired Terraform action, either `terraform apply` to create a resource or `terraform destroy` to remove existing resources. After choosing the action, the user inputs specific resource parameters, for example, the AWS region and S3 bucket name. Before finalizing, Backstage provides a summary screen for users to review the entered configurations, demonstrated in Figure 20.

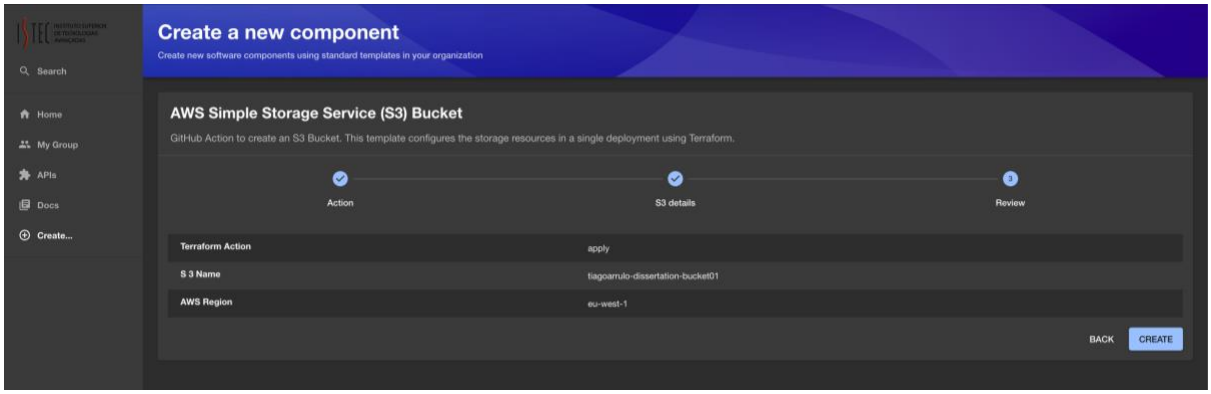


Figure 20 - Create a new component Review

Once the process is completed, the Backstage UI displays the execution workflow defined in the template.yaml file, consisting of four phases:

- Rendering Terraform Files: Downloads and populates template variables into Terraform files.

- Publishing Details: Creates a dedicated GitHub repository for the resource, enabling independent management.
- Dispatching GitHub Action Workflow: Triggers the associated GitHub Action for automated deployment.
- Registering the New Component: Adds the newly provisioned resource to the Backstage Catalog.

Following successful deployment, the Backstage interface provides the user with three actionable outputs: "Open in Catalog" redirects to the newly created component in Backstage, "View Resource in AWS Console" provides direct access to the AWS management page and "Check Documentation" offers detailed insights into the deployed Terraform templates, demonstrated in Figure 21.

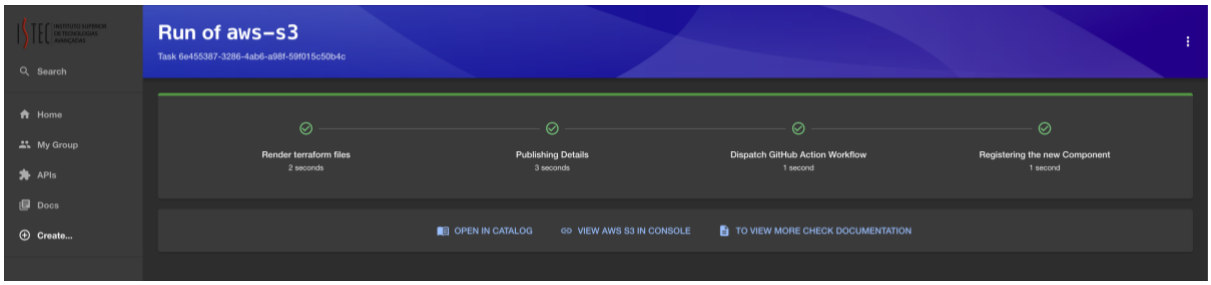


Figure 21 - Backstage Run of S3 Deployment Workflows and Outputs

Selecting "Open in Catalog" confirms the successful registration and displays the newly created S3 bucket component within the Backstage Catalog, including its associated metadata and relationships, as illustrated in Figure 22.

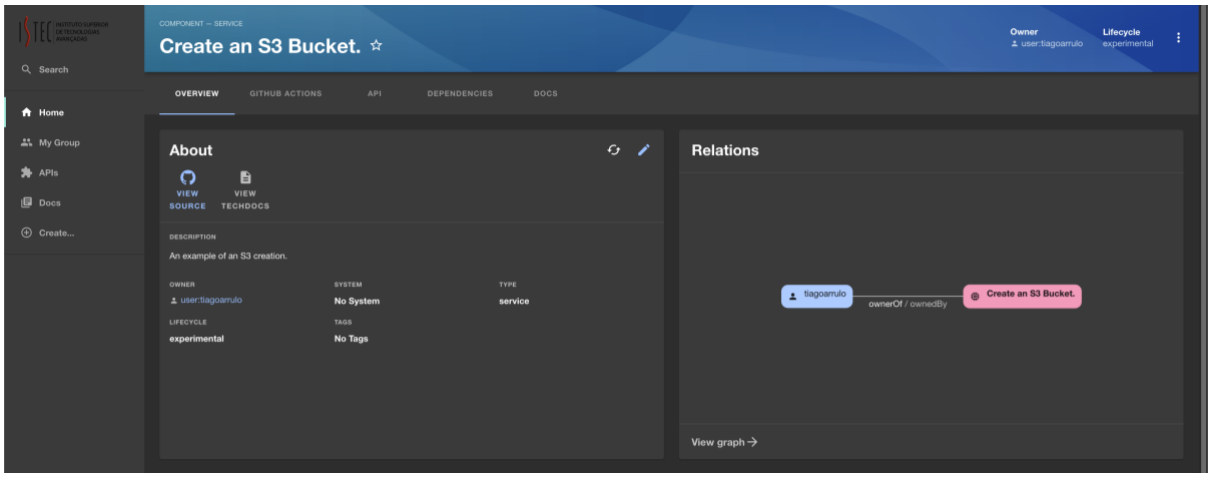


Figure 22 - S3 Component in Backstage Catalog

Within the GitHub Actions tab, users can verify the successful execution of workflows, re-trigger deployments if necessary, or directly view workflow details on GitHub, as depicted in Figure 23.

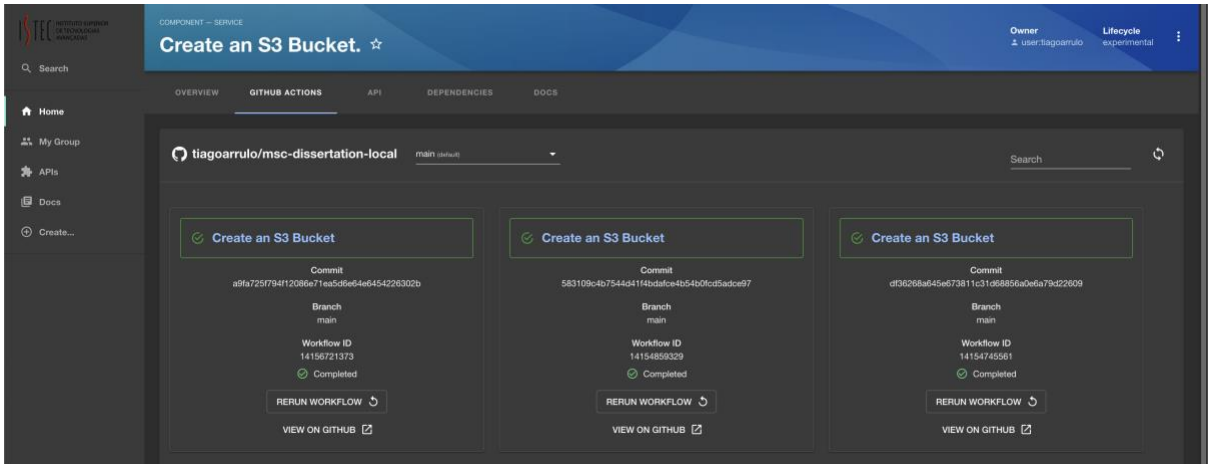


Figure 23 - Backstage GitHub Actions for S3 creation

On GitHub, users can review comprehensive workflow execution logs and Terraform configurations, ensuring transparency and traceability of automated actions performed, as presented in Figures 24.

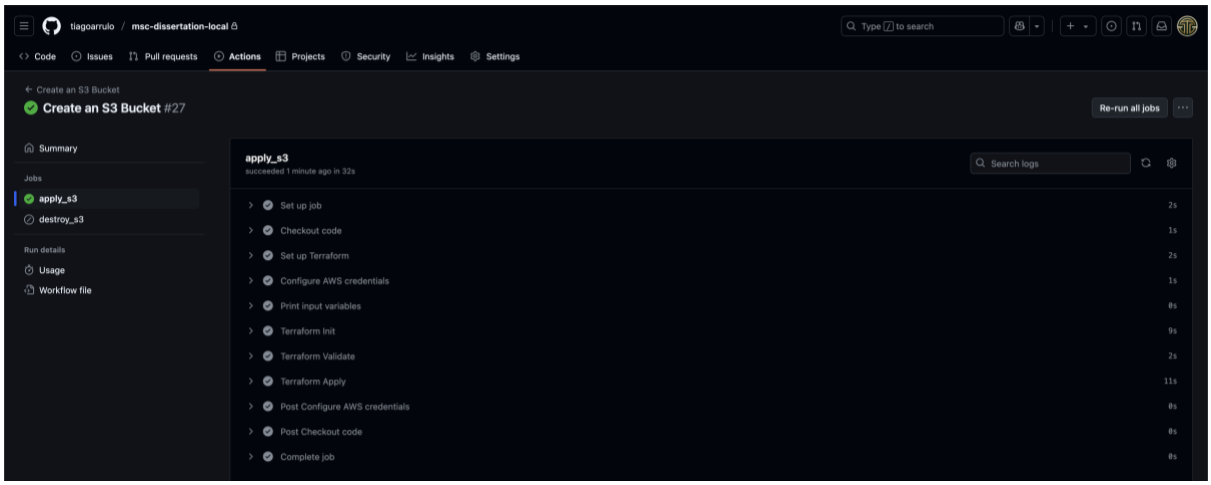


Figure 24 - GitHub Action Workflow

Lastly, by selecting "View Resource in AWS Console," users are redirected to the AWS Console, where they can verify the successful creation of the S3 bucket within their AWS account, confirming the effectiveness of the automated deployment process as it can be seen in Figure 25.

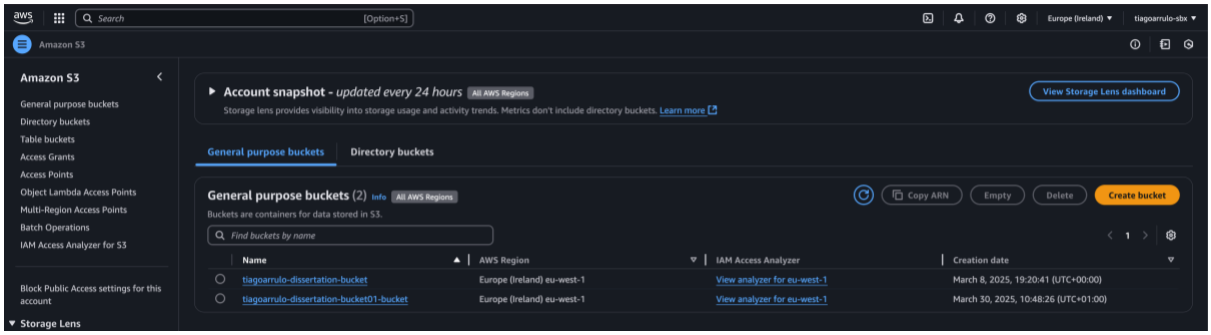


Figure 25 - AWS Console with S3 created

4.4.3. Results and Observations

During this chapter, the outputs of the implementation of an IDP prototype based on Backstage, GitHub, GitHub Actions and Terraform are presented. The main objective was to determine how effectively Platform Engineering practices streamline infrastructure provisioning tasks compared to manual methods.

Given the constraints of the practical evaluation environment, specifically that the author was the only user of the prototype, quantitative evaluation using traditional team-based metrics, such as DORA, were not considered feasible. Instead, a comparative analysis approach was adopted, focusing on operational efficiency, reduction of cognitive load, repeatability and process efficiency.

4.4.3.1. Methodology

To effectively evaluate the benefits provided by the implementation of the IDP, two deployment scenarios were defined and executed:

- **Scenario A – Manual Deployment**: In this scenario, infrastructure provisioning was conducted manually by directly using Terraform scripts. Each step, including the script preparation, provider authentication, resource provisioning and validation was documented in detail to capture the complexity and cognitive demands placed on the developer.
- **Scenario B – IDP-based Automated Deployment**: In this scenario, provisioning tasks were executed using the developed Backstage based IDP prototype. This scenario leveraged predefined Terraform templates, automated CI/CD workflows through GitHub Actions and an intuitive UI provided by Backstage. Within this scenario, it is assumed that the setup of the Backstage and all the other components were previously setup, as it is done by Platform Teams in real scenarios. Similar to Scenario A, every step from initiation through deployment and resource validation was documented.

To be able to compare both scenarios, multiple quantitative metrics were defined:

- **Number of Steps Required**: Counting explicit manual actions or inputs required by the developer.
- **Deployment Duration**: Total time taken from initiation to the resource becoming operational.
- **Complexity and Cognitive Load**: Subjective evaluation of mental effort and complexity involved in performing deployment tasks.
- **Potential for Errors**: Likelihood and identification of mistakes or misconfigurations during the deployment process. Its origin lies in the number and complexity of manual inputs or configurations required at each step.

While both Complexity and Cognitive Load and Potential for Errors involve subjective interpretations, relying partly on the researcher's direct experience, this study acknowledges the resulting bias as a limitation. Consequently, any conclusions drawn from these qualitative assessments should be interpreted with an awareness of their inherent subjectivity.

Each scenario was executed sequentially, with documentation capturing each step and corresponding metrics based on the define evaluation criteria. Observations from these executions formed the basis for comparative analysis and qualitative assessment, providing visibility into the practical impacts and benefits associated from adopting Platform Engineering principles.

4.4.3.2. Results and Observation

This section presents the outcomes of a comparative analysis conducted between manual infrastructure provisioning methods and automated provisioning methods using the developed IDP. The analysis covers three different infrastructure resources: AWS EC2 instances, AWS S3 buckets and GCP Kubernetes Engine (GKE), as it can be seen in Table 2.

The decision to analyze multiple resources across different cloud providers aims to provide robust empirical evidence, strengthening the credibility and applicability of the findings within this dissertation. For a detailed breakdown of the manual and automated provisioning processes, including step-by-step instructions, please refer to Annex 3.

Table 2 - Comparative Analysis of Manual and IDP Based Deployments

Resource	Scenario	Steps	Duration	Complexity	Error Potential
AWS EC2	Manual	14	12m 58,54 s	Medium	Medium
AWS EC2	IDP-Based	5	3 m 58,22 s	Low	Low
AWS S3	Manual	9	2 m 45,41 s	Low	Medium
AWS S3	IDP-Based	5	56,38 s	Low	Low
GKE	Manual	19	27m 34,35 s	High	High
GKE	IDP-Based	5	11m 40,45 s	Low	Low

4.4.4. Interviews and Feedback

To extend the insights gathered from the PoC execution and experimentation, two sets of interviews were conducted with IT professionals from Portuguese tech consulting companies. The primary goal was to further explore users' perceptions and understanding of Platform Engineering:

- First Interview Round (Pre-PoC): Eleven interviewees participated in this round to share their initial knowledge of Platform Engineering, how have they discovered the concept and their perspective on its impact on developer experience, team productivity and business value.
- Second Interview Round (Post-PoC): Once the PoC was completed, a smaller subset of interviewees offered feedback specific to the PoC results. This provides additional perspective based on their professional experience, highlighting how the outcomes align or differs from their expectations.

In the first interview round, participants held a wide range of roles and had 3 to 16 years of experience in the tech industry, as shown in Table 3. Their varied backgrounds, including technical, managerial and sales-oriented positions helped capture a broad spectrum of viewpoints on Platform Engineering. A complete record of all interviewee responses is provided in Annex 4.

Table 3 - First Session Interviewee List

	Role	Years of Experience in Tech Industry
Interviewee 1	Multi-Cloud Solutions Architect	6
Interviewee 2	Cloud & DevOps Architect	16
Interviewee 3	Delivery Manager	3
Interviewee 4	Cloud Engineer DevOps	3
Interviewee 5	Cloud Architect	15
Interviewee 6	QA Engineer	6
Interviewee 7	Sales Engineer	5
Interviewee 8	Integration Engineer	4
Interviewee 9	Cloud Architect	10
Interviewee 10	Chief Technology Officer	7
Interviewee 11	DevOps Engineer	4

In the second interview round, a smaller subset of six interviewees was selected to provide feedback on the results of the PoC. The objective was to gain a deeper understanding of how these professionals, based on their experience, perceived the impacts demonstrated by the prototype, especially in terms of developer experience, team productivity and value delivery in software development. Their answers offer insights into how the PoC outcomes align with or differ from their expectations. A complete record of these responses is available in Annex 5.

4.4.4.1. First Interview Round Questions

To establish a baseline regarding the participant's perceptions and general knowledge of Platform Engineering, a first round of interviews was conducted, which was designed based on the Theoretical Background research and research objectives defined for this study, aiming to gather broader insights into the participants' experience, understanding and expectations.

Five open-ended questions were formulated to cover key research dimensions, including the participants' professional background, their familiarity with Platform Engineering concepts and their perspective on its impact on developer experience, team productivity and business value. The coverage of these questions was intended to capture different viewpoints and contextualize the relevance of Platform Engineering across different professional roles and levels of expertise.

Question 1

Could you please share your current role and tell how many years of experience do you have in tech industry?

Question 2

How would you describe your knowledge about Platform Engineering, and what were the main sources for that knowledge?

Question 3

How do you think Platform Engineering impacts the developer experience?

Question 4

How do you think Platform Engineering impacts team productivity?

Question 5

In your experience, does Platform Engineering help deliver more value to the business? If so, in what ways?

4.4.4.2. First Interview Impressions

Perspective on Platform Engineering

The first round of interviews brought together professionals with widely diverse backgrounds, spanning purely technical roles, such as Multi-Cloud Solutions Architect (Interviewee 1), Cloud Architects (Interviewees 2, 5, and 9), DevOps Engineers (Interviewees 4 and 11), QA Engineer

(Interviewee 6) and Integration Engineer (Interviewee 8), to management or strategic roles, including a Delivery Manager (Interviewee 3), Sales Engineer (Interviewee 7) and a Chief Technology Officer (CTO) (Interviewee 10). While their years of experience ranged from 3 to 16, they all expressed a unified understanding that Platform Engineering is focused on standardizing environments, tools and workflows to streamline and accelerate software development.

An important insight that emerged was how participants acquired their knowledge of Platform Engineering. Interviewee 1 reported hands-on expertise in designing self-service platforms and automation pipelines, while Interviewee 2 highlighted broad practical experience supported by formal training and day-to-day application of Platform Engineering concepts. Others, like Interviewees 3, 4, 7 and 8, emphasized a theoretical foundation, citing sources like cloud provider documentation, official courses or even university resources. Meanwhile, Interviewee 5 described being in a beginner phase, learning from CNCF documentation and online articles, underscoring the newness of Platform Engineering even for experienced professionals. Additionally, Interviewees 9 and 10, combined self-taught approaches with official documentation and resources.

Impacts on Developer Experience

Regarding the impacts on developer experience, Interviewees 1, 2 and 4 emphasized how standardized tools and automation remove repetitive tasks and infrastructure overhead, enabling developers to focus on writing code. Interviewee 1 specifically referred that this approach “eliminates repetitive tasks, reduces cognitive load, and allows developers to concentrate on building and deploying code instead of managing infrastructure”. Similarly, Interviewee 2 highlighted a more satisfying user experience driven by greater focus on the app and its processes, while Interviewee 4 described Platform Engineering as a way to “focus more on coding and delivering features instead of dealing with infrastructure complexities”.

A similar viewpoint appeared in the answers from Interviewees 5 and 7, who described how self-service capabilities and preconfigured environments promote a more productive and enjoyable developer experience. Interviewee 5 called the impact on developer work “big”, although with a potential “learning curve […] for those not working in that model”. Meanwhile Interviewee 7 highlighted the convenience for developers, that “can quickly have a "blueprint" environment ready to use”.

Lastly, Interviewees 10 and 11 emphasized the consolidation of infrastructure tools and automation of repetitive tasks, while Interviewee 8 stressed the importance of freeing developers from infrastructure concerns to focus on business problem-solving. Interviewee 10 called it a “VIP treatment” for developers and pointed to the “happier, more productive” environment that emerges when teams

share the same language and toolsets. Interviewee 11 reinforced that this strategy "reduces complexity" and "streamlining workflows", enabling faster code delivery and fewer operational distractions.

The interviewees acknowledged that standardization, automation and self-service are central to enhance developer experience, freeing teams to focus on high-value tasks, boosting efficiency and reducing cognitive burden associated with managing infrastructure.

Productivity Gains and Value Delivery

Regarding the productivity gains and value delivery, a common thread across all participants was the faster time-to-market enabled by Platform Engineering, Interviewees 1, 2 and 5 highlighted its role in speeding up the delivery cycle by reducing complexities and automating operational tasks. Interviewees 4, 6 and 8 highlighted minimizing errors and accelerating time-to-market as key advantages, while Interviewee 3 cited reuse of standardized infrastructure to develop solutions quicker and consistently.

Interviewees 7 and 10 described Platform Engineering as delivering "a lot of additional value" to the business, calling it essential when managing large-scale deployment with complex governance requirements. Interviewee 9 emphasized the benefit of delivering "customer value more quickly and reliably" and Interviewee 11 noted how consolidating tools, CI/CD pipelines and monitoring solutions and centralized platform "unifies and simplifies development".

The interviewees defined Platform Engineering as central to reducing overhead, standardizing tooling and accelerating delivery schedules, enabling businesses to have a more responsive approach to market demands and customer needs.

Summary of Interview Insights

Overall, the interviewees have a unified view that Platform Engineering is more than a specific technology stack, it is a strategic approach aimed at standardizing tools, automating workflows and streamlining development. Many referenced how self-service platforms and preconfigured environments enable developers to focus on coding rather than "wrestling with infrastructure", reducing cognitive load and boosting efficiency. Some interviewees noted potential learning curve challenges, but this was surpassed by the general consensus that standardization and automation produce a "big" impact on developer satisfaction and agility.

Regarding business value, every interviewee described faster delivery cycles and improved time-to-market as core benefits of Platform Engineering, as well as its role in simplifying operational tasks and cutting costs, reusability and reliability in complex deployments. By consolidating tools and

removing manual overhead, organizations can rapidly respond to market demands, ensure consistent governance and innovate at scale, indicating that Platform Engineering supports both developer experience and value creation across diverse business contexts.

4.4.4.3. Second Interview Round Questions

Question 1

From your professional experience and considering the outcomes of the prototype, what is your perspective on how Platform Engineering affects developer experience, productivity, and overall value delivery in software development?

4.4.4.4.Second Interview Impressions

To validate the findings of the PoC against the study's theoretical framework and research objectives, a second round of interviews was conducted. Six professionals who had reviewed the prototype results were invited to reflect, through a single, open question, on how Platform Engineering influences developer experience, productivity, and business value.

Impacts on Developer Experience

In the second interview round, interviewees unanimously agreed that the prototype outcomes demonstrated positive impacts on developer experience as speculated in the research questions. Interviewee 1 emphasized how Platform Engineering alleviates developers from "tedious, error-prone manual tasks", highlighting that automation saves time and reduces developer frustration. Similarly, Interviewee 3 noted that Platform Engineering "makes life easier for developers" by streamlining processes, allowing teams to concentrate on development tasks rather than troubleshooting infrastructure.

Interviewees 2 highlighted the importance of autonomy and self-service capabilities provided by the IDP, while Interviewee 5 emphasized the efficiency gains and reduction of complexity achieved through standardized, faster deployments. Both referenced that enabling developers to independently provision resources from a standardized catalog enhances daily work satisfaction and efficiency. These views align with the findings from Interviewee 4, that described IDP environments as "intuitive, and user-friendly", simplifying previously complex and error-prone deployment processes.

This consensus reflects an affirmation of developer-centric benefits identified in the initial research questions, particularly the reduction of cognitive load and improved focus on core development tasks.

Productivity Gains and Value Delivery

Regarding productivity, the interviewees recognized improvements attributed to Platform Engineering. Interviewee 1 described the automated provisioning and consistent environment setup as fundamental to accelerate workflows and enhancing productivity, affirming that developers can simply "experiment and get features out the door". Interviewee 4 mentioned the reduction in "both […] time and effort required for deployments", underscoring that automation decreases manual intervention and associated errors, leading to an improved productivity.

Interviewees 2 and 5 reinforced how Platform Engineering promotes consistency, reduces cognitive overhead and accelerates the onboarding of new team members further enhancing productivity within teams. Interviewee 6 highlighted the rapid replicability of automated workflows, although an initial learning curve exists, it is "quickly offset by the accelerated deployment of resources", particularly for repetitive tasks.

On the aspect of delivering value, interviewees agreed that Platform Engineering accelerates time-to-market, improves software reliability and facilitates continuous delivery of business aligned features. Interviewee 4 noted that adopting IDP principles within organizations, represent a clear pathway towards increased overall efficiency and added business value. Similarly, Interviewee 1 reinforced this statement, highlighting how consistency across environments mitigates typical deployment risks, enabling teams to deliver higher-quality software more quickly.

Summary of Interview Insights

Overall, the second interview round provided confirmation of the positive impacts that Platform Engineering and the implemented IDP prototype have on developer experience, productivity and value delivery. Interviewees consistently identified automation, self-service capabilities and standardization as crucial elements that streamline workflows, mitigate risks and empower developers. These outputs affirmatively address the initial research questions by demonstrating that Platform Engineering enhances developer experience, contribute to productivity improvements and increased organizational value.

4.5. Evaluation

This evaluation phase combines both the empirical results from the PoC and qualitative insights gathered from two rounds of interviews, ensuring an assessment aligned with the DSRM. This

evaluation is structured around the research questions addressing developer experience, productivity improvements and value delivery associated with Platform Engineering.

4.5.1. Developer Experience

The comparative analysis demonstrate that the IDP-based provisioning enhances the developer experience, as the results indicated a reduction in cognitive complexity by abstracting manual configurations and minimizing context switching. The streamlined approach required fewer manual inputs, enabling developers to interact with simplified, predefined parameters without handling underlying deployment complexities directly.

Interviewees from both rounds reinforced these findings by highlighting reduced cognitive load and increased satisfaction. An interviewee noted the IDP's role in alleviating developers from "tedious, error-prone manual tasks", while other referred to Platform Engineering as providing "VIP treatment" for developers.

The second interview round has emphasized the autonomy enabled by self-service platforms, through standardized catalogs, enabling developers to provision resources independently and efficiently. Some interviewees described the IDP as "intuitive, and user-friendly", further simplifying complex deployment processes and enhancing overall developer experience.

The alignment between empirical outcomes and qualitative feedback validates that Platform Engineering positively transforms developer experience, reducing cognitive overhead and enabling developers to focus on coding and feature development.

4.5.2. Productivity Improvements

Empirical data from the PoC indicated productivity enhancements through the adoption of IDP-based automated deployments. Automated provisioning reduced the time needed across the evaluated resources (AWS EC2, AWS S3 and GKE), demonstrating efficiency gains compared to manual methods. Predefined Terraform templates and GitHub Actions workflow simplified the provisioning processes, enabling quicker deployment cycles and iterative improvements.

Interview insights supported these productivity gains, as the interviewees emphasized that automation minimized errors and reduced the number of manual steps, accelerating overall workflows. One interviewee specifically mentioned the reduction in "both […] time and effort required for deployments" and others emphasized how self-service capabilities and standardized processes further improved productivity by facilitating quicker onboarding and consistent workflows.

It was also highlighted that while there may be an initial learning curve associated with adapting an IDP-based environment, this investment is quickly counterweighed by the accelerated deployment of repetitive resources, confirming long-term productivity benefits.

4.5.3. Value Delivery

The empirical results and interview feedback jointly confirm that Platform Engineering accelerates value delivery by enabling faster and more reliable software releases. The PoC demonstrated lower error potential and higher consistency through standardized and automated deployments, enhancing overall reliability.

Interviewees across both rounds, pointed to faster time-to-market and enhanced software reliability as direct outcomes of adopting Platform Engineering practices. One interviewee has highlighted the strategic advantage of standardized infrastructure enabling quicker, repeatable deployments. Other interviewees emphasized the critical role Platform Engineering plays in delivering business value, particularly in large-scale or complex governance scenarios. These insights confirm the broader organizational impact and strategic benefits associated with implementing an IDP.

4.5.4. Summary of Insights

Overall, the integration of empirical evidence from the practical implementation of the IDP prototype and qualitative insights gathered through interviews support the theoretical assessment, outlined in the literature on Platform Engineering. Empirical results demonstrated enhancements in developer experience, primarily through the abstraction of infrastructure complexities and reduction of cognitive load. These results reinforce Camille Fournier and Ian Nowland arguments [29], highlighting how standardized and automated workflows simplify the day-to-day experience of developers, by enabling them to focus more on coding and less on infrastructure management.

Qualitative data from both interview rounds further reinforce these empirical findings, as participants identified automation, standardization and self-service capabilities as fundamental to improving developer satisfaction and autonomy, support Evan Bottcher argument [70], that platforms empower teams by providing intuitive and streamlined workflows. Interviewees described the implemented IDP as “intuitive” and “user-friendly”, further stressing an alignment with existing literature.

Finally, the assessment of value delivery highlighted how Platform Engineering contributes strategically by accelerating software release cycles, enhancing software reliability and promoting organizational efficiency. Empirical findings from the prototype demonstrated the benefits of

standardized and automated deployments in reducing manual errors. These findings align with the arguments of several authors [77], [80], [81], who recognized Platform Engineering as strategic enabler that consolidates tooling, reduces duplication and improves time-to-market through consistent and reusable processes. The interviewees reinforced these perspectives by highlighting the role of Platform Engineering in minimizing deployment risks and enabling quicker and more reliable value features.

In conclusion, the alignment between theoretical investigation, empirical outcomes from the PoC and qualitative interview insights, validate that adopting Platform Engineering practices can reduce operational complexity, enhance developer experience, improve productivity and accelerate value delivery.

5. Conclusion

This dissertation aimed to explore the impact of Platform Engineering, by designing, implementing and evaluating an IDP prototype using Backstage, GitHub, GitHub Actions and Terraform. The research was guided by two primary research questions:

1. In what way does Platform Engineering improve the developer experience?
2. How can Platform Engineering improve productivity and deliver more value?

5.1. Developer Experience and Productivity

Addressing the first research question, "In what way does Platform Engineering improve the developer experience?", the study revealed positive impacts. The IDP prototype demonstrably reduced cognitive load and minimized deployment complexities through automation, self-service capabilities and standardized workflows. For instance, the comparative analysis showed a marked decrease in the number of steps and time required for deployments, deploying an AWS S3 bucket via the IDP took approximately 56 seconds across 5 steps, compared to nearly 3 minutes and 9 steps manually. Similarly, deploying a more complex component, such as a GKE cluster the deployment duration drops from over 27 minutes manually to under 12 minutes via the IDP, with steps reduced from 19 to 5. This quantitative evidence highlights not only efficiency gains, but also to a tangible reduction in the potential for human error.

Qualitative insights gathered across seventeen interviews consistently validated the empirical findings, as the interviewed professionals highlighted that the IDP genuinely simplified daily tasks, promoting greater developer autonomy and satisfaction. Participants described the abstraction of infrastructure concerns, as liberating developers to focus on core coding and problem-solving activities, describing this experience as receiving “VIP treatment” or feeling less burdened by “tedious, error-prone manual tasks”. The streamlined, “intuitive, and user-friendly” nature of IDP approach was frequently highlighted, reinforcing theoretical claims found in literature, such as Bottcher, Fournier and Nowland [29], [70], about Platform Engineering’s capacity to enhance not just productivity, but the quality of the developer experience itself. The ability for developers to self-service resources within a standardized framework reduces friction and empowers them, contributing directly to a more positive and productive work environment.

However, despite the positive findings, the study did not directly face the challenges highlighted in the literature review, such as throughput decreases, change instability or increased burnout [67], communication gaps, budget constraints, operational risks like system failures and cybersecurity threats and skill shortage [77], [78], [79]. This absence is attributed to the controlled and single-user

environment of the prototype, hence future implementations at scale may surface these challenges, emphasizing the importance of careful management during broader adoption.

5.2. Organizational Value Delivery

Addressing the second research question, "How can Platform Engineering improve productivity and deliver more value?", the findings highlighted the strategic advantages presented by adopting Platform Engineering practices. The prototype illustrated how standardized, reusable infrastructure components, managed by IDP, accelerate software delivery cycle, enhances operational reliability and mitigates deployment risks. The observed reduction in deployment times and error potential translates directly into faster feedback loops and quicker time-to-market.

Interviewees consistently acknowledged Platform Engineering as more than an operational improvement, as they viewed it as a strategic enabler. It empowers organizations to respond with greater agility to market demands, achieve faster value realization and deliver business-aligned features with improved consistency and reliability. This aligns with existing literature emphasizing how standardized automation and reduced operational complexity strengthen organizational efficiency and agility. By providing a stable, predictable and efficient foundation for development, Platform Engineering allows teams to better align with and drive business objectives, shifting their focus from operational maintenance to innovation and value creation. The consistency promoted by the platform minimizes risks typically associated with diverse tooling and manual processes, thus enhancing overall business resilience.

5.3. Limitations

Despite these positive results several limitations were identified during the research. The prototype was tested within a controlled, single-user, local environment, which limits the generalizability of results regarding scalability, availability and security requirements pertinent to full-scale production scenarios. Furthermore, the evaluation's scope focused on a limited scope of Cloud providers (AWS and GCP) and did not extensively explore complex multi-cloud orchestration and interoperability challenges.

The interviews were conducted solely with IT professionals based in Portuguese consulting companies, narrowing the applicability of the qualitative findings to a specific geographical and industrial context. Expanding the participant internationally would offer a more comprehensive view. The interview sample, which included eleven professionals for an initial round of interviews and a follow-up with a subset of six participants, although sufficient for exploratory analysis, could be further extended in future work, to increase the reliability of findings across different roles and industries. The

prototype was also limited to a specific technology stack, exploring alternatives could further validate the universality of the benefits observed. Additionally, some of the evaluation metrics involved subjective interpretation by the researcher, hence future work should consider standardized quantitative measurement practices for greater objectivity.

Finally, the study adopted a short-term perspective, longitudinal studies are needed to understand the long-term evolution and scalability of Platform Engineering benefits within organizations. The prototype workflows were intentionally simplified for demonstration purposes and did not encompass the full complexity of real-world scenarios.

5.4. Future Work

Future research should aim to address these limitations by conducting extensive, multi-user experiments in production-like environments. Evaluations should involve diverse range of cloud providers, exploring complex orchestration and interoperability scenarios. Including international participants from various professional backgrounds would enrich the generalizability of the findings.

Further research could also explore long-term longitudinal studies, tracking the evolution of Platform Engineering practices within organizations to better understand impacts and scalability of benefits over time. Additionally, exploring alternative technological stacks and conducting comparative effectiveness studies would also provide valuable insights into different tools and methodologies. Incorporating standardized, objective metrics for evaluating developer experience and productivity would further strengthen future research.

5.5. Contributions

This dissertation contributes both to the theoretical and practical knowledge to the field of Platform Engineering. Theoretically, it validates existing assumptions found in the literature regarding the positive impacts of automation, cognitive load reduction, enhanced productivity and accelerated value delivery driven by Platform Engineering practices. Practically, the functional IDP prototype serves as a tangible demonstration of how specific tools and methodologies (Backstage, GitHub, GitHub Actions, Terraform) can be integrated to realize these benefits.

In summary, this research highlights how adopting Platform Engineering practices through an IDP can improve developer experience, boost team productivity and enhance organizational value delivery. Despite its limitations, this study establishes a solid foundation for future academic research and practical adoption, promoting broader experimentation and adoption strategies to further consolidate Platform Engineering as a crucial element in the modern software development landscape.

Bibliography

[1] P. Sivathapandi and R. Soundarapandiyan, "Platform Engineering for Multi-Cloud Enterprise Architectures: Design Patterns and Best Practices."

[2] A. Williams *et al.*, "The New Stack Platform Engineering: What You Need to Know Now," 2023.

[3] M. Salatino, *Platform Engineering on Kubernetes*. Manning, 2023.

[4] S. Sharma, *DevOps for Dummies, IBM Limited Edition*. John Wiley & Sons, 2013.

[5] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, pp. 45–77, Jan. 2007.

[6] M. Pulkkinen, *An Integrated Model for Evaluating ICT Impact in the Education Domain*. 2013.

[7] E. Juma Adwan and B. Ali Alsaeed, "Development and evaluation of a Cloud Computing Adoption Framework (CCAFF) for retail banks in Bahrain," 2022.

[8] J. Reis, M. Amorim, N. Melão, and P. Matos, "Digital transformation: A literature review and guidelines for future research," in *Advances in Intelligent Systems and Computing*, Springer Verlag, 2018, pp. 411–421. doi: 10.1007/978-3-319-77703-0_41.

[9] K. S. R. Warner and M. Wäger, "Building dynamic capabilities for digital transformation: An ongoing process of strategic renewal," *Long Range Plann*, vol. 52, no. 3, pp. 326–349, Jun. 2019, doi: 10.1016/j.lrp.2018.12.001.

[10] M. Fitzgerald, N. Kruschwitz, D. Bonnet, and M. Welch, "Embracing Digital Technology A New Strategic Imperative," 2013. [Online]. Available: http://sloanreview.mit.edu/faq/

[11] B. Solis, R. Lieb, and J. Szymanski, "The 2014 state of digital transformation," 2014.

[12] S. Kraus, P. Jones, N. Kailer, A. Weinmann, N. Chaparro-Banegas, and N. Roig-Tierno, "Digital Transformation: An Overview of the Current State of the Art of Research," *Sage Open*, vol. 11, no. 3, 2021, doi: 10.1177/21582440211047576.

[13] D. Plekhanov, H. Franke, and T. H. Netland, "Digital transformation: A review and research agenda," *European Management Journal*, vol. 41, no. 6, pp. 821–844, Dec. 2023, doi: 10.1016/j.emj.2022.09.007.

[14] B. Tabrizi, E. Lam, K. Girard, and V. Irvin, "Digital Transformation Is Not About Technology," 2019. [Online]. Available: https://hbr.org/2019/03/digital-transformation-is-not-about-technology

[15] C. Mihu, A. G. Pitic, and D. Bayraktar, "Drivers of Digital Transformation and their Impact on Organizational Management," *Studies in Business and Economics*, vol. 18, no. 1, pp. 149–170, Apr. 2023, doi: 10.2478/sbe-2023-0009.

[16] K. Osmundsen, J. Iden, and B. Bygstad, "Association for Information Systems AIS Electronic Library (AISeL) Digital Transformation: Drivers, Success Factors, and Implications Recommended Citation," 2018. [Online]. Available: https://aisel.aisnet.org/mcis2018/37

[17] K. Liere-Netheler, S. Packmohr, and K. Vogelsang, *Drivers of Digital Transformation in Manufacturing*. [Online]. Available: http://hdl.handle.net/10125/50381

[18] G. Lambropoulos, S. Mitropoulos, and C. Douligeris, “A Review on Cloud Computing services, concerns, and security risk awareness in the context of Digital Transformation,” in *2021 6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, 2021, pp. 1–6. doi: 10.1109/SEEDA-CECNSM53056.2021.9566267.

[19] M. Al-Ruithe, E. Benkhelifa, and K. Hameed, “Key Issues for Embracing the Cloud Computing to Adopt a Digital Transformation: A study of Saudi Public Sector,” in *Procedia Computer Science*, Elsevier B.V., 2018, pp. 1037–1043. doi: 10.1016/j.procs.2018.04.145.

[20] C. A. Gonçalves, C. O. de S. Messias, J. L. Soares, and M. M. da C. L. Peixoto, “Adoption of Cloud Computing in the organizations: a bibliometric analysis of this technology within a digital transformation context,” *Revista de Administração da UFSM*, vol. 16, no. 3, p. e1, Aug. 2023, doi: 10.5902/1983465975326.

[21] A. Mehta and P. Ranjan, “The Role of DevOps in Accelerating Digital Transformation.”

[22] S. AL-Zahran, “How DevOps Practices Support Digital Transformation,” *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 9, no. 3, pp. 2780–2788, Jun. 2020, doi: 10.30534/ijatcse/2020/46932020.

[23] K. Viren Ahuja and P. Pant, “ Enabling digital transformation with a platform engineering approach.” Accessed: Jan. 03, 2025. [Online]. Available: https://www.nagarro.com/en/blog/platform-engineering-digital-transformation

[24] H. Clarke, “ Unlocking Innovation: The Rise of Platform Engineering in DevOps.” Accessed: Jan. 03, 2025. [Online]. Available: https://www.harrisonclarke.com/blog/unlocking-innovation-the-rise-of-platform-engineering-in-devops

[25] “Top 5 Benefits of Enabling Digital Transformation using Cloud.” Accessed: Jan. 03, 2025. [Online]. Available: https://preludesys.com/enabling-digital-transformation-using-cloud/

[26] P. M. Mell and T. Grance, “The NIST definition of cloud computing,” Gaithersburg, MD, 2011. doi: 10.6028/NIST.SP.800-145.

[27] I. Sriram and A. Khajeh-Hosseini, “Research Agenda in Cloud Technologies.” [Online]. Available: http://aws.amazon.com/

[28] D. André Cardoso Serafim, J. Luís Brinquete Borbinha, and F. Manuel Bernardo Pereira Supervisor, “A Framework to build Information Systems using a Public Cloud Telecommunications and Informatics Engineering Examination Committee,” 2017.

[29] C. Fournier and I. Nowland, *Platform Engineering: A Guide for Technical, Product, and People Leaders*. O’Reilly Media, Inc, 2024.

[30] R. Chellappa, “Intermediaries in cloud-computing: A new computing paradigm,” in *INFORMS Annual Meeting, Dallas*, 1997, pp. 26–29.

[31] A. J. Sammes, “Computer Communications and Networks Series editor.” [Online]. Available: http://www.springer.com/series/4198

[32] S. I. Bairagi and A. O. Bang, “Cloud Computing: History, Architecture, Security Issues.”

[33] C. Computing and J. Olson, “History of the Cloud”.

[34] Jeff Barr, “Amazon EC2 Beta.” Accessed: Mar. 01, 2024. [Online]. Available: https://aws.amazon.com/blogs/aws/amazon_ec2_beta/

[35] K. Kamila, “Role of Cloud Computing in modeRn libRaRies: a CRitiCal appRaisal.”

[36] S. Bigelow and K. Marko, “The history of cloud computing explained.” Accessed: Mar. 09, 2024. [Online]. Available: https://www.techtarget.com/whatis/feature/The-history-of-cloud-computing-explained

[37] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: State-of-the-art and research challenges,” *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, May 2010, doi: 10.1007/s13174-010-0007-6.

[38] M. Vuyyuru, P. Annapurna, K. G. Babu, and A. S. K. Ratnam, “An overview of cloud computing technology,” *International Journal of Soft Computing and Engineering*, vol. 2, no. 3, pp. 244–247, 2012.

[39] “Types of cloud computing.” Accessed: Dec. 05, 2024. [Online]. Available: https://www.redhat.com/en/topics/cloud-computing/public-cloud-vs-private-cloud-and-hybrid-cloud

[40] A. Gajbhiye and K. M. P. D. Shrivastva, “Cloud computing: Need, enabling technology, architecture, advantages and challenges,” in *Proceedings of the 5th International Conference on Confluence 2014: The Next Generation Information Technology Summit*, Institute of Electrical and Electronics Engineers Inc., Nov. 2014, pp. 1–7. doi: 10.1109/CONFLUENCE.2014.6949224.

[41] M. G. Avram, “Advantages and Challenges of Adopting Cloud Computing from an Enterprise Perspective,” *Procedia Technology*, vol. 12, pp. 529–534, 2014, doi: 10.1016/j.protcy.2013.12.525.

[42] S. Chhabra and A. K. Singh, “A Comprehensive Vision on Cloud Computing Environment: Emerging Challenges and Future Research Directions,” Jul. 2022, [Online]. Available: http://arxiv.org/abs/2207.07955

[43] D. Kapil, P. Tyagi, S. Kumar, and V. P. Tamta, “Cloud computing: Overview and research issues,” in *Proceedings - 2017 International Conference on Green Informatics, ICGI 2017*, Institute of Electrical and Electronics Engineers Inc., Nov. 2017, pp. 71–76. doi: 10.1109/ICGI.2017.18.

[44] T. Destefano, R. Kneller, and J. Timmis, “Cloud Computing and Firm Growth,” 2020. [Online]. Available: www.RePEc.org

[45] “Gartner Says More Than Half of Enterprise IT Spending in Key Market Segments Will Shift to the Cloud by 2025.” Accessed: Mar. 02, 2024. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2022-02-09-gartner-says-more-than-half-of-enterprise-it-spending

[46] “Cloud Computing Market (By Deployment: Private, Hybrid, Public; By Service: Software as a Service (SaaS), Infrastructure as a Service (IaaS), Platform as a Service (PaaS); By End User: IT and Telecom, BFSI, Manufacturing, Healthcare, Retail and Consumer Goods, Media and Entertainment, Energy and Utilities, Government and Public Sector, Others; By Organization Size; By Workload) - Global Industry Analysis, Size, Share, Growth, Trends, Regional Outlook, and Forecast 2023 – 2032.” Accessed: Mar. 02, 2024. [Online]. Available: https://www.precedenceresearch.com/cloud-computing-market

[47] A. Dyck, R. Penners, and H. Lichter, “Towards Definitions for Release Engineering and DevOps.”

[48] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, “A survey of DevOps concepts and challenges,” Nov. 30, 2019, *Association for Computing Machinery*. doi: 10.1145/3359981.

[49] A. Brunnert *et al.*, “Performance-oriented DevOps: A Research Agenda,” 2015. Accessed: Feb. 21, 2024. [Online]. Available: https://ar5iv.labs.arxiv.org/html/1508.04752

[50] G. Kim, K. Behr, and G. Spafford, *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win*, 1st ed. IT Revolution Press, 2013.

[51] M. Khan, A. Khan, F. Khan, M. Khan, and T. Whangbo, “Critical Challenges to Adopt DevOps Culture in Software Organizations: A Systematic Review,” *IEEE Access*, vol. 10, p. 1, Dec. 2022, doi: 10.1109/ACCESS.2022.3145970.

[52] “About devopsdays.” Accessed: Feb. 21, 2024. [Online]. Available: https://devopsdays.org/about/

[53] T. Offerman, R. Blinde, C. J. Stettina, and J. Visser, “A Study of Adoption and Effects of DevOps Practices,” in *2022 IEEE 28th International Conference on Engineering, Technology and Innovation (ICE/ITMC) & 31st International Association For Management of Technology (IAMOT) Joint Conference*, IEEE, Jun. 2022. doi: 10.1109/ice/itmc-iamot55089.2022.10033313.

[54] G. Bou Ghantous, A. Gill, and G. Bou, “Association for Information Systems AIS Electronic Library (AISeL) DevOps: Concepts, Practices, Tools, Benefits and Challenges Recommended Citation,” 2017. [Online]. Available: http://aisel.aisnet.org/pacis2017/96

[55] A. V. Jha *et al.*, “From theory to practice: Understanding DevOps culture and mindset,” 2023, *Cogent OA*. doi: 10.1080/23311916.2023.2251758.

[56] N. O. Babenko and A. S. Shermukhamedov, “GLOBALIZATION AND ADAPTATION OF DEVOPS CULTURE IN THE CORPORATE ENVIRONMENT: CHALLENGES AND PERSPECTIVES,” *INTERNATIONAL JOURNAL OF INFORMATION AND COMMUNICATION TECHNOLOGIES*, vol. 4, pp. 66–75, Dec. 2023, doi: 10.54309/IJICT.2023.16.4.006.

[57] M. Cole, “Navigating DevOps Cultural Shifts: Challenges and opportunities.” [Online]. Available: https://www.researchgate.net/publication/383425399

[58] N. MacDonald and I. Head, “DevSecOps: How to Seamlessly Integrate Security Into DevOps,” Sep. 2016.

[59] H. Myrbakken and R. Colomo-Palacios, “DevSecOps: A Multivocal Literature Review.”

[60] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, “Challenges and solutions when adopting DevSecOps: A systematic review,” Mar. 2021, doi: 10.1016/j.infsof.2021.106700.

[61] “DevOps Market Size,” May 2023. Accessed: Feb. 21, 2024. [Online]. Available: https://www.gminsights.com/industry-analysis/devops-market

[62] B. Wilkes, A. M. P. Milani, and M. A. Storey, “A Framework for Automating the Measurement of DevOps Research and Assessment (DORA) Metrics,” in *Proceedings - 2023 IEEE International Conference on Software Maintenance and Evolution, ICSME 2023*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 62–72. doi: 10.1109/ICSME58846.2023.00018.

[63] D. Graves Portman, “Are you an Elite DevOps performer? Find out with the Four Keys Project.” Accessed: Mar. 01, 2025. [Online]. Available: https://cloud.google.com/blog/products/devops-sre/using-the-four-keys-to-measure-your-devops-performance

[64] N. Harvey, “DORA’s software delivery metrics: the four keys.” Accessed: Mar. 01, 2025. [Online]. Available: https://dora.dev/guides/dora-metrics-four-keys/

[65] “What Are DORA Metrics?” Accessed: Mar. 05, 2025. [Online]. Available: https://www.datadoghq.com/knowledge-center/dora-metrics/#what-are-the-use-cases-for-dora-metrics

[66] B. Lloyd Pearson, “The Comprehensive Guide to DORA Metrics.” Accessed: Mar. 05, 2025. [Online]. Available: https://linearb.io/blog/dora-metrics

[67] “Accelerate State of DevOps 2024,” 2024.

[68] B. Pariseau, “Google DORA issues platform engineering caveats.” Accessed: Mar. 06, 2025. [Online]. Available: https://www.techtarget.com/searchitoperations/news/366615375/Google-DORA-issues-platform-engineering-caveats

[69] “Understanding Platform Engineering Metrics for Improved Team Performance.” Accessed: Mar. 06, 2025. [Online]. Available: https://www.multitudes.com/blog/platform-engineering-metrics

[70] E. Bottcher, “What I Talk About When I Talk About Platforms.” Accessed: Dec. 15, 2024. [Online]. Available: https://martinfowler.com/articles/talk-about-platforms.html

[71] “What is a Golden Path?” Accessed: Dec. 15, 2024. [Online]. Available: https://www.vmware.com/topics/golden-paths

[72] “State of Platform Engineering Report Volume 2,” 2023. Accessed: Dec. 13, 2024. [Online]. Available: https://humanitec.com/whitepapers/state-of-platform-engineering-report-volume-2

[73] “Platform Engineering That Empowers Users and Reduces Risk.” Accessed: Dec. 14, 2024. [Online]. Available: https://www.gartner.com/en/infrastructure-and-it-operations-leaders/topics/platform-engineering

[74] “Platform tooling landscape.” Accessed: Dec. 15, 2024. [Online]. Available: https://platformengineering.org/platform-tooling

[75] “State of Platform Engineering Report Volume 3,” 2024. Accessed: Dec. 13, 2024. [Online]. Available: https://platformengineering.org/reports/state-of-platform-engineering-vol-3

[76] “The State of Platform Engineering Report 2023,” 2023. Accessed: Dec. 14, 2024. [Online]. Available: https://www.puppet.com/resources/state-of-devops-report

[77] “2024 State of DevOps Report: The Evolution of Platform Engineering,” 2024.

[78] “Challenges of platform engineering and potential risks.” Accessed: Jan. 06, 2025. [Online]. Available: https://www.stackspot.com/en/blog/challenges-of-platform-engineering-and-potential-risks

[79] M. Campbell, “Platform Engineering Challenges: Small Teams, Build Versus Buy, and Building the Wrong Thing.” Accessed: Jan. 06, 2025. [Online]. Available: https://www.infoq.com/news/2023/02/platform-engineering-challenges/

[80] " What is Platform Engineering and why adopt it in your company? ." Accessed: Jan. 07, 2025. [Online]. Available: https://blog.sparkfabrik.com/en/platform-engineering-why-adopt-it

[81] S. Fenton, " 8 Real-World Reasons To Adopt Platform Engineering ." Accessed: Jan. 07, 2025. [Online]. Available: https://thenewstack.io/8-real-world-reasons-to-adopt-platform-engineering/

[82] X. Zhao, T. Clear, and R. Lal, "Identifying the primary dimensions of DevSecOps: A multi-vocal literature review ☆," *J Syst Softw*, vol. 214, p. 112063, 2024, doi: 10.5281/zenodo.7.

[83] M. El Khatib, H. Alawadhi, and M. Al Mansoori, "Platform Engineering in Manufacturing: Role, Effect, Challenges and Opportunities," *International Journal of Business Analytics and Security*, vol. 4, no. 2, p. 2024, doi: 10.54489/ijbas.v4i2.364.

[84] P. Sivathapandi and R. Soundarapandiyan, "Platform Engineering for Multi-Cloud Enterprise Architectures: Design Patterns and Best Practices."

[85] V. Kunchenapalli, "Good Developer Experience with Platform Engineering and Devops," *Int J Res Appl Sci Eng Technol*, vol. 12, no. 3, pp. 2240–2244, Mar. 2024, doi: 10.22214/ijraset.2024.58839.

[86] M. El Khatib and A. Alzarooni, "How Technology Solutions providers are Using Platform engineering to Improve Efficiency, Agility, Performance and Responsiveness," *International Journal of Theory of Organization and Practice (IJTOP)*, vol. 3, no. 2, pp. 16–30, Jan. 2024, doi: 10.54489/ijtop.v3i2.291.

[87] S. T. Makani and S. Jangampeta, "A Comparative Study of Platform Engineering Tools: Implications for System Design and Scalability."

Annexes

Annex 1 – Preparation of Local Environment

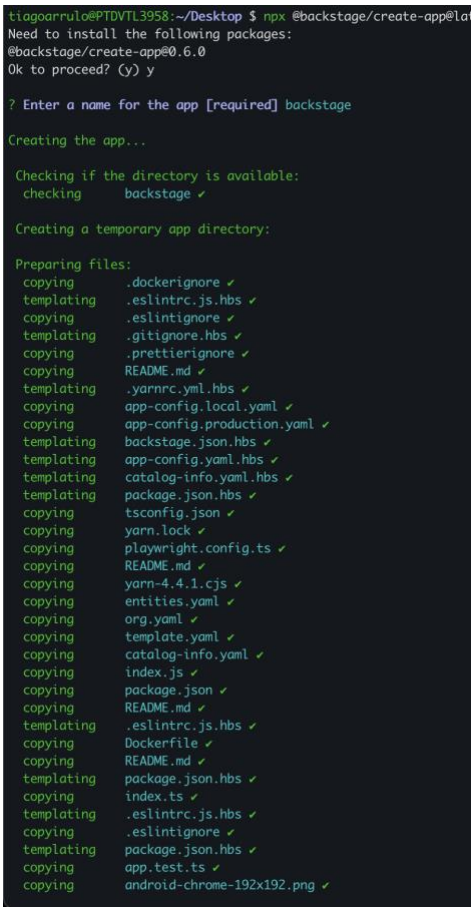


Figure 26 - Backstage Implementation Process

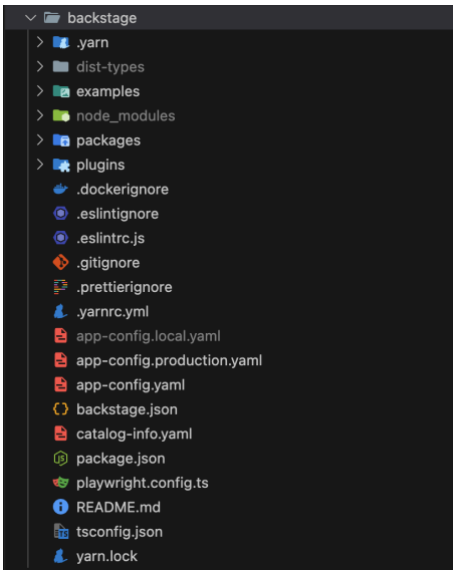


Figure 27 - Backstage Folder Structure

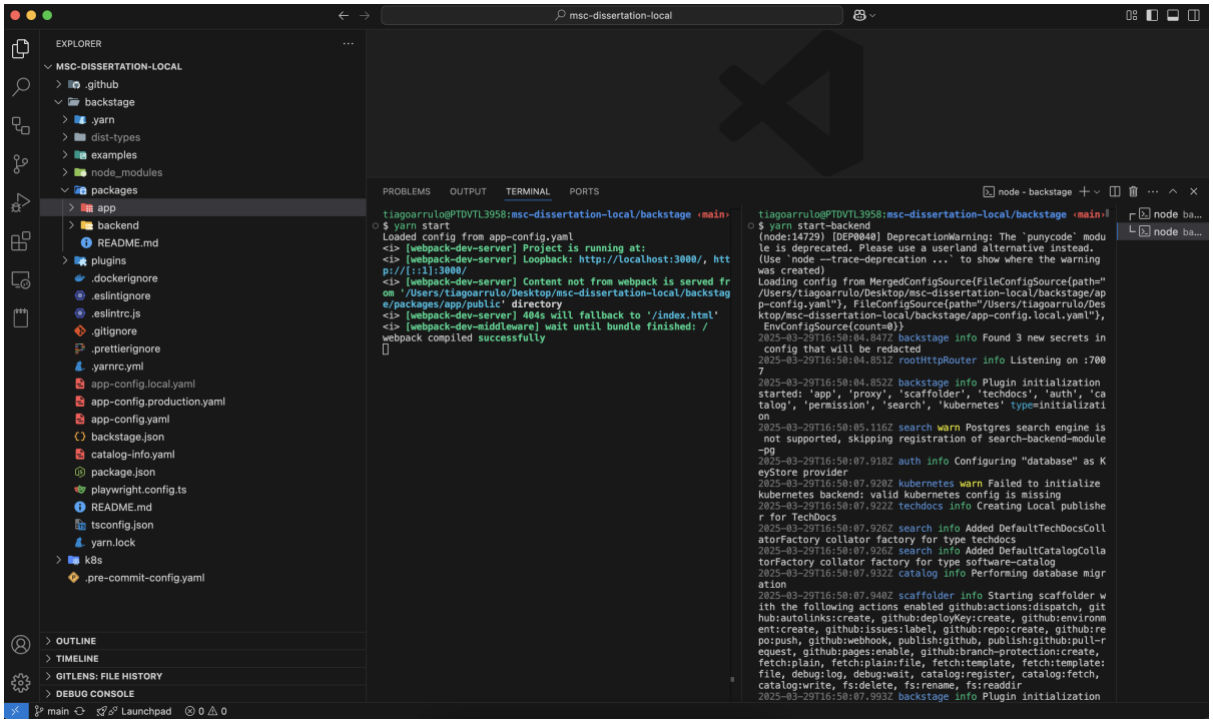


Figure 28 - Start Backstage Backend and Frontend

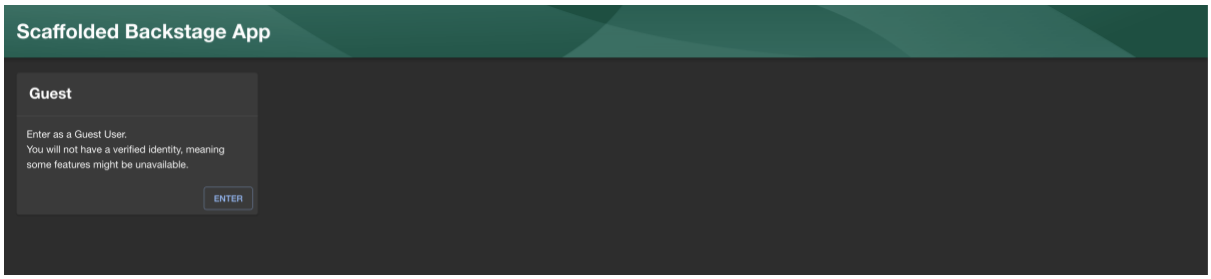


Figure 29 - Backstage Guest Authentication

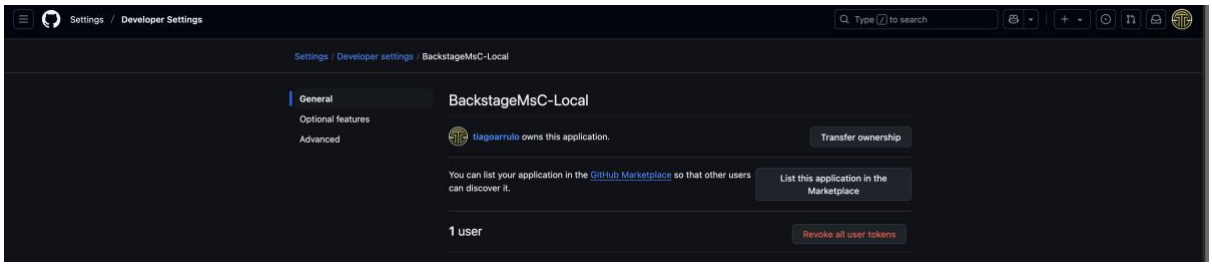


Figure 30 - GitHub OAuth App

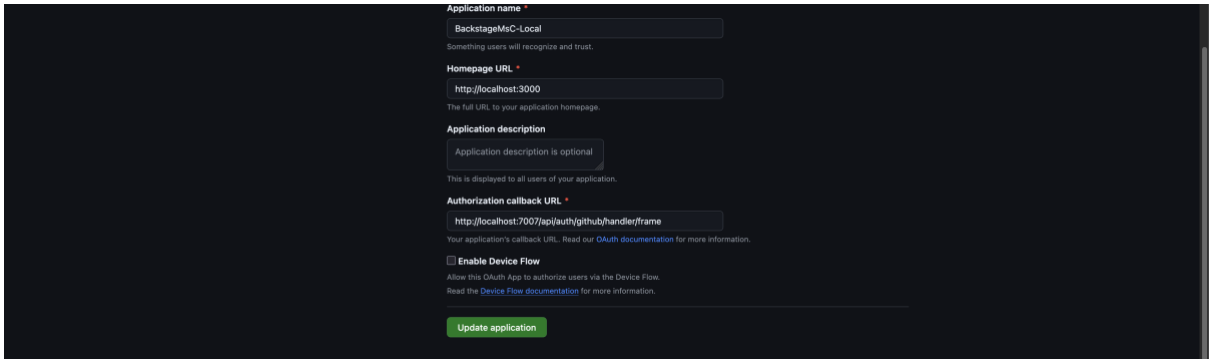


Figure 31 - GitHub Homepage URL and Callback URL

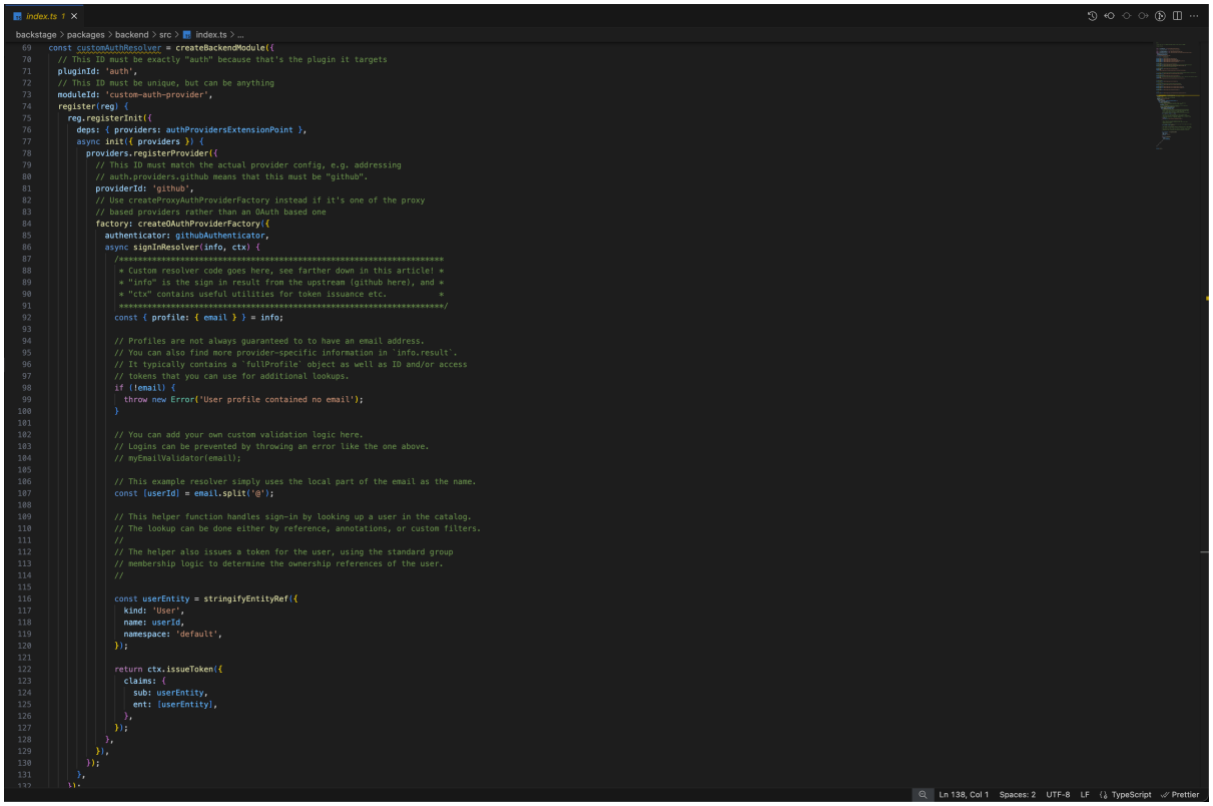


Figure 32 - GitHub Authentication Custom Resolver

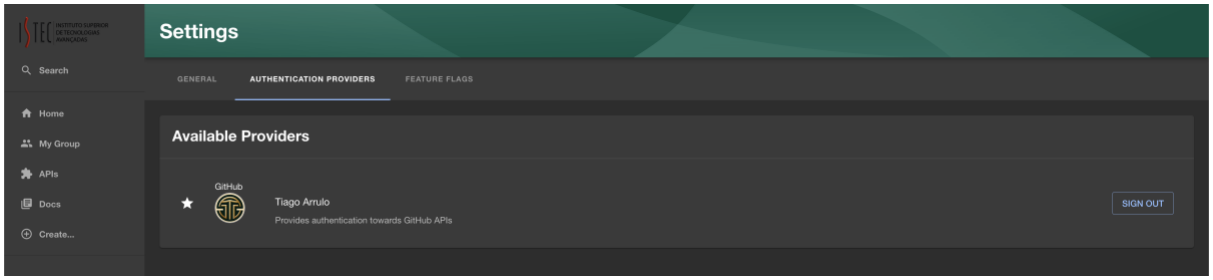


Figure 33 - GitHub User in Backstage

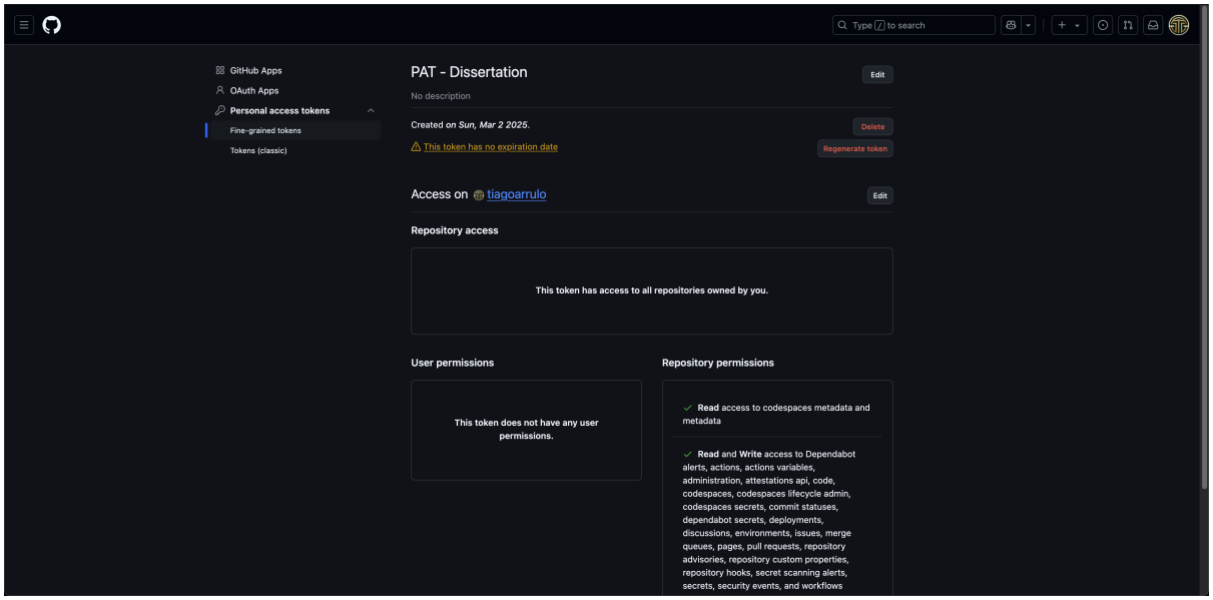


Figure 34 - GitHub PAT

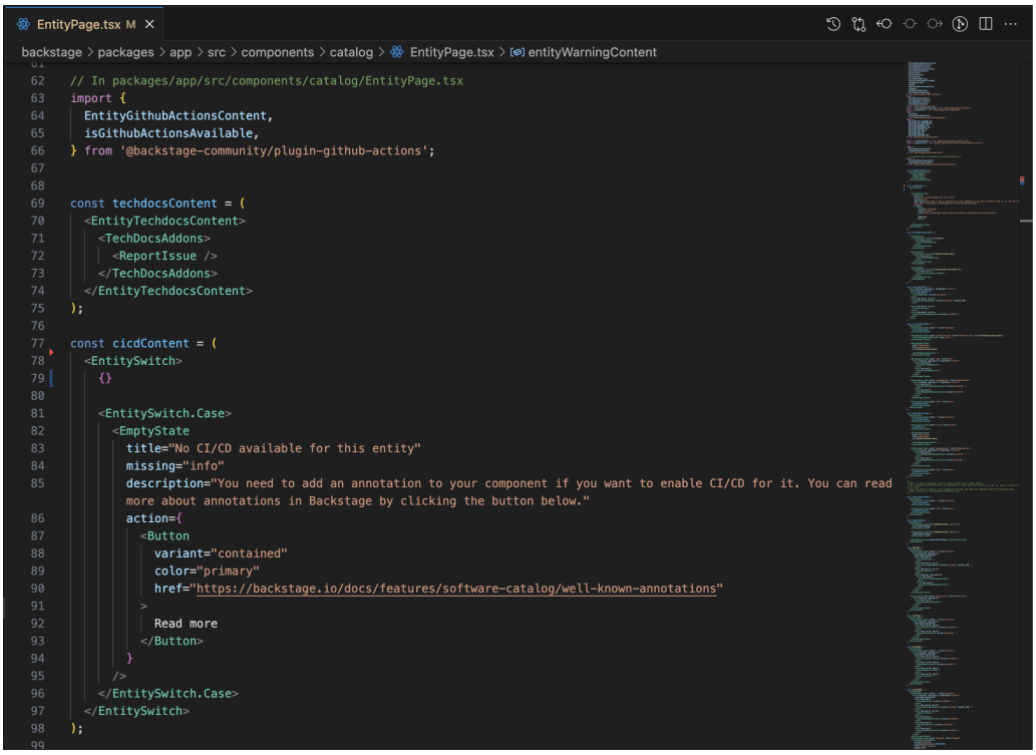


Figure 35 - EntityPage.tsx Import GitHub Actions

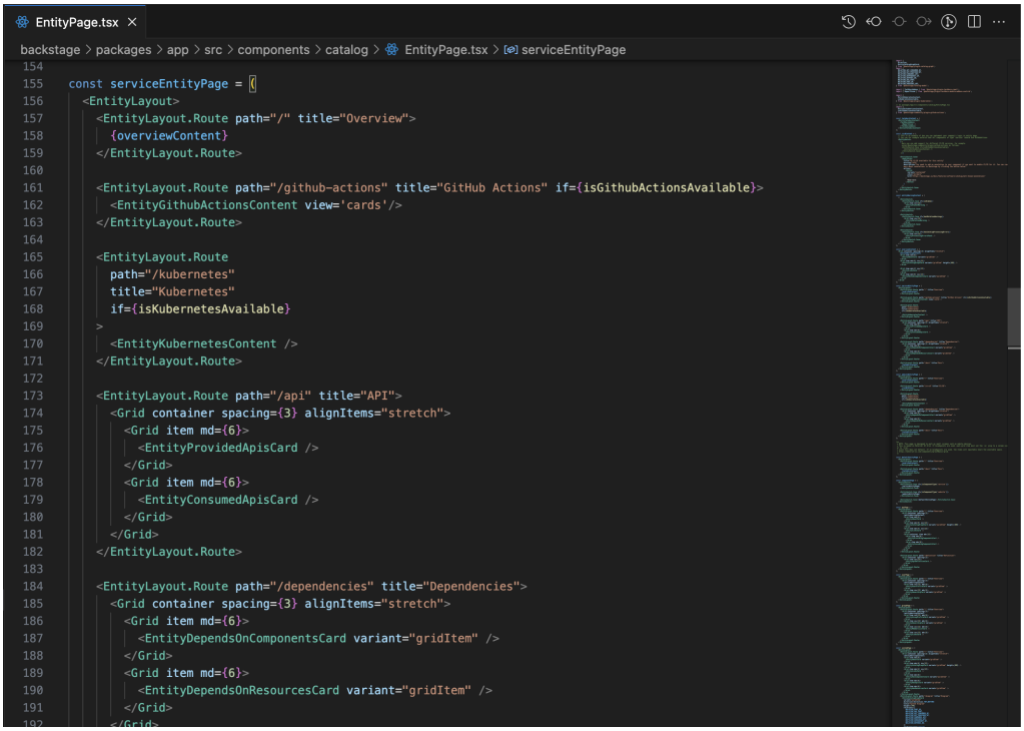
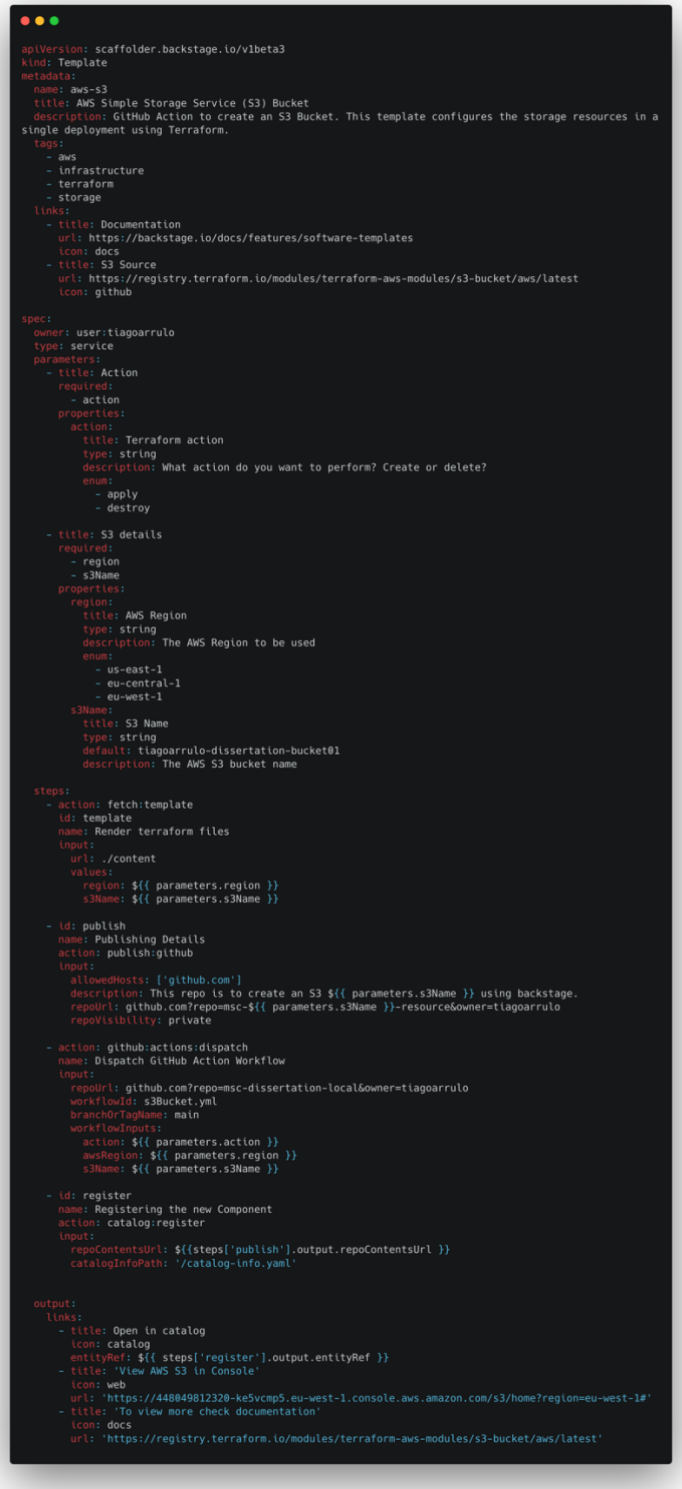
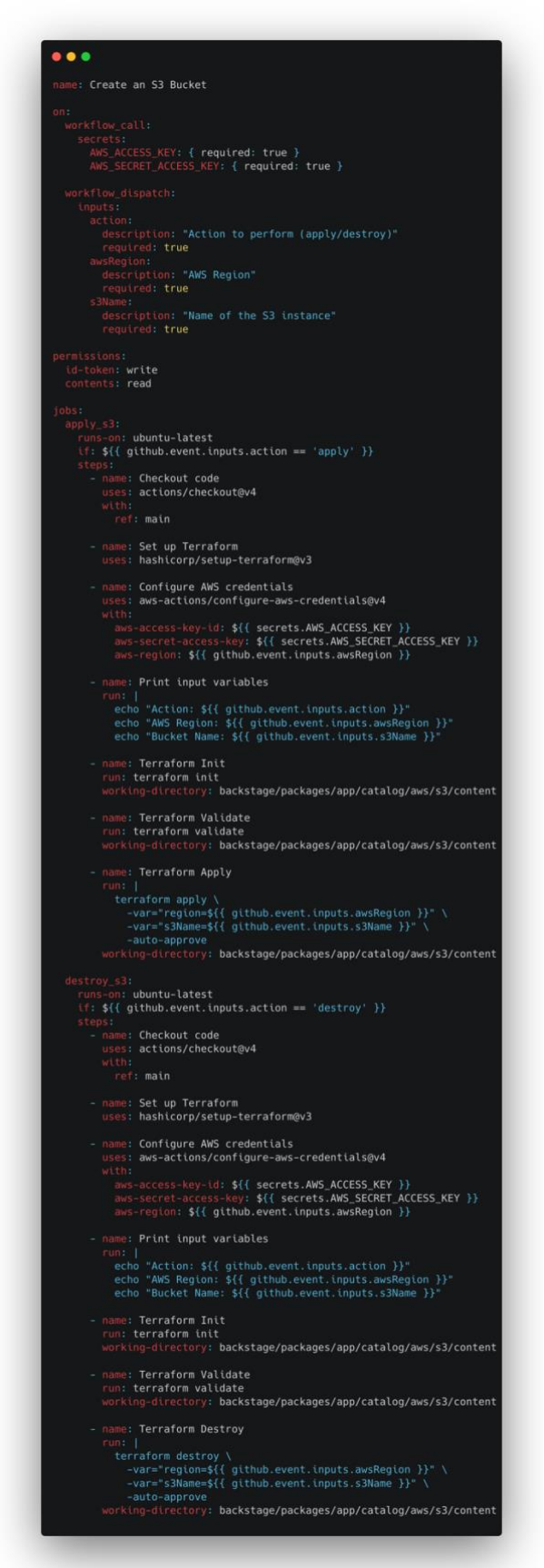


Figure 36 - EntityPage.tsx GitHub Actions Display



```yaml
apiVersion: scaffolder.backstage.io/v1beta3
kind: Template
metadata:
  name: aws-s3
  title: AWS Simple Storage Service (S3) Bucket
  description: GitHub Action to create an S3 Bucket. This template configures the storage resources in a
single deployment using Terraform.
  tags:
    - aws
    - infrastructure
    - terraform
    - storage
  links:
    - title: Documentation
      url: https://backstage.io/docs/features/software-templates
      icon: docs
    - title: S3 Source
      url: https://registry.terraform.io/modules/terraform-aws-modules/s3-bucket/aws/latest
      icon: github

spec:
  owner: user:tiagoarrulo
  type: service
  parameters:
    - title: Action
      required:
        - action
      properties:
        action:
          title: Terraform action
          type: string
          description: What action do you want to perform? Create or delete?
          enum:
            - apply
            - destroy

    - title: S3 details
      required:
        - region
        - s3Name
      properties:
        region:
          title: AWS Region
          type: string
          description: The AWS Region to be used
          enum:
            - us-east-1
            - eu-central-1
            - eu-west-1
        s3Name:
          title: S3 Name
          type: string
          default: tiagoarrulo-dissertation-bucket01
          description: The AWS S3 bucket name

  steps:
    - action: fetch:template
      id: template
      name: Render terraform files
      input:
        url: ./content
        values:
          region: ${{ parameters.region }}
          s3Name: ${{ parameters.s3Name }}

    - id: publish
      name: Publishing Details
      action: publish:github
      input:
        allowedHosts: ['github.com']
        description: This repo is to create an S3 ${{ parameters.s3Name }} using backstage.
        repoUrl: github.com?repo=msc-${{ parameters.s3Name }}-resource&owner=tiagoarrulo
        repoVisibility: private

    - action: github:actions:dispatch
      name: Dispatch GitHub Action Workflow
      input:
        repoUrl: github.com?repo=msc-dissertation-local&owner=tiagoarrulo
        workflowId: s3Bucket.yml
        branchOrTagName: main
        workflowInputs:
          action: ${{ parameters.action }}
          awsRegion: ${{ parameters.region }}
          s3Name: ${{ parameters.s3Name }}

    - id: register
      name: Registering the new Component
      action: catalog:register
      input:
        repoContentsUrl: ${{steps['publish'].output.repoContentsUrl }}
        catalogInfoPath: '/catalog-info.yaml'


  output:
    links:
      - title: Open in catalog
        icon: catalog
        entityRef: ${{ steps['register'].output.entityRef }}
      - title: 'View AWS S3 in Console'
        icon: web
        url: 'https://448049812320-ke5vcmp5.eu-west-1.console.aws.amazon.com/s3/home?region=eu-west-1#'
      - title: 'To view more check documentation'
        icon: docs
        url: 'https://registry.terraform.io/modules/terraform-aws-modules/s3-bucket/aws/latest'
```

Figure 37 - Template.yaml file for S3 Buckets



```yaml
name: Create an S3 Bucket

on:
  workflow_call:
    secrets:
      AWS_ACCESS_KEY: { required: true }
      AWS_SECRET_ACCESS_KEY: { required: true }

  workflow_dispatch:
    inputs:
      action:
        description: "Action to perform (apply/destroy)"
        required: true
      awsRegion:
        description: "AWS Region"
        required: true
      s3Name:
        description: "Name of the S3 instance"
        required: true

permissions:
  id-token: write
  contents: read

jobs:
  apply_s3:
    runs-on: ubuntu-latest
    if: ${{ github.event.inputs.action == 'apply' }}
    steps:
      - name: Checkout code
        uses: actions/checkout@v4
        with:
          ref: main

      - name: Set up Terraform
        uses: hashicorp/setup-terraform@v3

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ github.event.inputs.awsRegion }}

      - name: Print input variables
        run: |
          echo "Action: ${{ github.event.inputs.action }}"
          echo "AWS Region: ${{ github.event.inputs.awsRegion }}"
          echo "Bucket Name: ${{ github.event.inputs.s3Name }}"

      - name: Terraform Init
        run: terraform init
        working-directory: backstage/packages/app/catalog/aws/s3/content

      - name: Terraform Validate
        run: terraform validate
        working-directory: backstage/packages/app/catalog/aws/s3/content

      - name: Terraform Apply
        run: |
          terraform apply \
            -var="region=${{ github.event.inputs.awsRegion }}" \
            -var="s3Name=${{ github.event.inputs.s3Name }}" \
            -auto-approve
        working-directory: backstage/packages/app/catalog/aws/s3/content

  destroy_s3:
    runs-on: ubuntu-latest
    if: ${{ github.event.inputs.action == 'destroy' }}
    steps:
      - name: Checkout code
        uses: actions/checkout@v4
        with:
          ref: main

      - name: Set up Terraform
        uses: hashicorp/setup-terraform@v3

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ github.event.inputs.awsRegion }}

      - name: Print input variables
        run: |
          echo "Action: ${{ github.event.inputs.action }}"
          echo "AWS Region: ${{ github.event.inputs.awsRegion }}"
          echo "Bucket Name: ${{ github.event.inputs.s3Name }}"

      - name: Terraform Init
        run: terraform init
        working-directory: backstage/packages/app/catalog/aws/s3/content

      - name: Terraform Validate
        run: terraform validate
        working-directory: backstage/packages/app/catalog/aws/s3/content

      - name: Terraform Destroy
        run: |
          terraform destroy \
            -var="region=${{ github.event.inputs.awsRegion }}" \
            -var="s3Name=${{ github.event.inputs.s3Name }}" \
            -auto-approve
        working-directory: backstage/packages/app/catalog/aws/s3/content
```

Figure 38 - GitHub Action Workflow for S3

Annex 2 – Creation of New Components Process

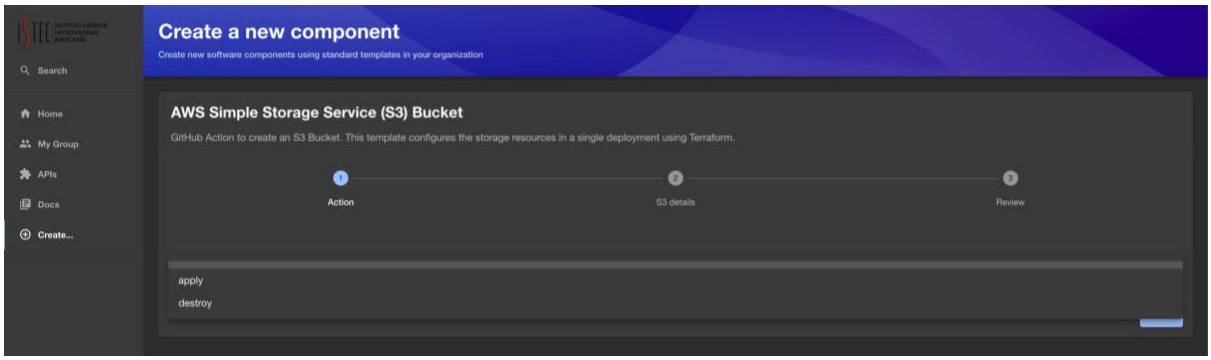


Figure 39 - Create a new component Terraform Action

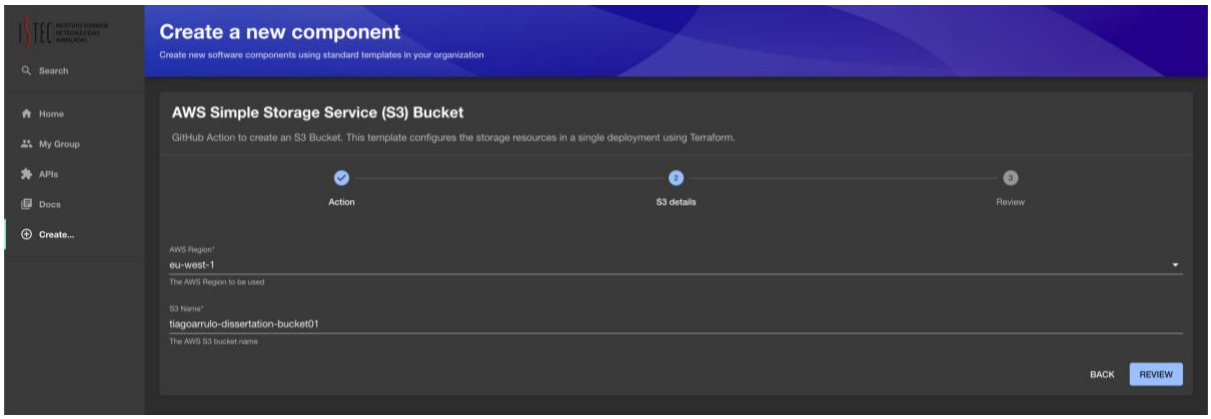


Figure 40 - Create a new component S3 details

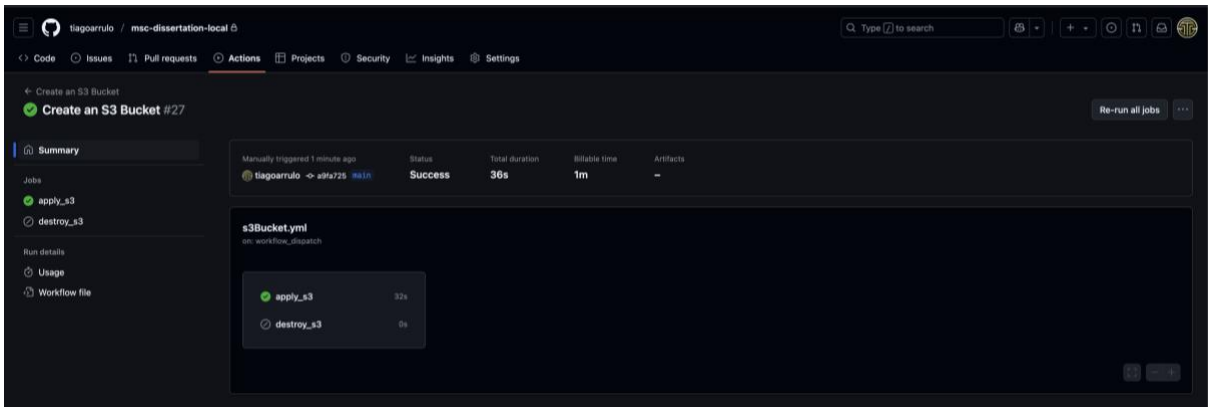


Figure 41 - GitHub Action Summary

Annex 3 – Detailed Steps for Resource Deployment

Detailed Steps for AWS EC2 Manual Deployment

Preparation:

1. Create Local Folder to Host Terraform Configuration.
2. Visit Hashicorp Terraform Registry for EC2 module.
3. Visit Hashicorp Terraform Registry for Virtual Private Cloud (VPC) Module.
4. Visit Hashicorp Terraform Registry for Security Group Module.
5. Copy contents from modules to local folder.

Customization and Configuration:

6. Adapt and customize Terraform configuration to match deployment requirements.
7. Configure and validate AWS credentials in CLI.

Validation and Deployment:

8. Execute `terraform init`
9. Run initial `terraform validate`
10. Correct minor configuration errors related to ingress Classless Inter-Domain Routing (CIDR) block.
11. Run second `terraform validate`
12. Execute `terraform plan`
13. Execute `terraform apply`

Verification:

14. Verify deployed resources via AWS console and Terraform outputs.

Detailed Steps for AWS EC2 IDP-Based Automated Deployment

1. Open Backstage console and log in.
2. Select the infrastructure template.
3. Input required fields: Terraform action, VPC region, Security Group CIDR, EC2 name and EC2 type.
4. Validate GitHub Action Pipeline status.
5. Verify successful deployment via AWS Console.

Detailed Steps for AWS S3 Manual Deployment

Preparation:

1. Create Local Folder to Host Terraform Configuration.
2. Visit Hashicorp Terraform Registry for S3 module.
3. Copy contents from modules to local folder.

Customization and Configuration:

4. Adapt and customize Terraform configuration to match deployment requirements.
5. Configure and validate AWS credentials in CLI.

Validation and Deployment:

6. Execute `terraform init`
7. Run initial `terraform validate`
8. Execute `terraform plan`
9. Execute `terraform apply`

Verification:

10. Verify deployed resources via AWS console and Terraform outputs.

Detailed Steps for AWS S3 IDP-Based Automated Deployment

1. Open Backstage console and log in.
2. Select the infrastructure template.
3. Input required fields: Terraform action, AWS region and S3 Name.
4. Validate GitHub Action Pipeline status.
5. Verify successful deployment via AWS Console.

Detailed Steps for GKE Manual Deployment

Preparation:

1. Create Local Folder to Host Terraform Configuration.
2. Visit Hashicorp Terraform Registry for GKE module.
3. Visit Hashicorp Terraform Registry for VPC Module.
4. Copy contents from modules to local folder.

Customization and Configuration:

5. Adapt and customize Terraform configuration to match deployment requirements.
6. Configure and validate GCP credentials using CLI.

Validation and Deployment:

7. Execute `terraform init`

8. Run initial `terraform validate`
9. Correct minor configuration errors related to ingress GKE subnetwork.
10. Run second `terraform validate`
11. Execute `terraform plan`
12. Execute `terraform apply`
13. Resolve Terraform actions related to Google CLI Authentication.
14. Execute `terraform apply`
15. Debug error to GKE cluster pod secondary range.
16. Execute `terraform apply`
17. Debug error with secondary IP ranges for services networking.
18. Execute `terraform apply`

Verification:

19. Verify deployed resources via GCP console and Terraform outputs.

Detailed Steps for GKE IDP-Based Automated Deployment

1. Open Backstage console and log in.
2. Select the infrastructure template.
3. Input required fields: Terraform action, Deployment Region and GKE Name.
4. Validate GitHub Action Pipeline status.
5. Verify successful deployment via GCP Console.

Annex 4 – First Interview Round Answers

Interviewee 1

A1: Im currently a senior Multi-Cloud Solutions Architect, with 6 years of experience, specializing in cloud-native infrastructure and enhancing development teams with self-service platforms With over a decade of experience in the tech industry, I have focused on deploying efficient, scalable and secure solutions

A2: My expertise in Platform Engineering roots in designing self-service platforms that optimize development and operations processes. That includes automation, continuous integration and continuous delivery (CI/CD) pipelines, Infrastructure as Code (IaC), container orchestration tools such as Kubernetes, and others

A3: Platform Engineering significantly improves the developer experience by offering standardized tools, processes, and environments. This eliminates repetitive tasks, reduces cognitive

load, and allows developers to concentrate on building and deploying code instead of managing infrastructure

A4: It enhances team productivity by promoting consistency and minimizing obstacles in workflows, it centralizes infrastructure management, automates deployment processes, and simplifies troubleshooting through shared tools and practices

A5: Yes, Platform Engineering delivers business value by expediting the time-to-market for applications, reducing operational costs through automation, and enhancing system reliability. By equipping teams with efficient and scalable platforms, businesses can accelerate innovation and effectively respond to market demands

Interviewee 2

A1: Cloud & DevOps Architect - 16 years

A2: I'm an expert in Platform engineering and my knowledge main sources are courses, training and day life experience working the a broad range of technologies

A3: Platform Engineering make developer experience more satisfying where de developer can focus on the app and processes development

A4: Platform Engineering increases team productivity

A5: Without any doubt. It enables a fast to market approach

Interviewee 3

A1: Delivery Manager - 3 anos

A2: Possuo conhecimentos teóricos e obtive este conhecimento através de pesquisas em diversas plataformas como cloud providers e outros.

A3: Platform Engineering melhora significativamente a experiência do programador ao fornecer ferramentas, infraestruturas e processos padronizados que simplificam o desenvolvimento e a gestão de aplicações. Ao automatizar tarefas repetitivas e criar ambientes self-service controlados, aumenta a eficiência e a autonomia dos programadores, permitindo-lhes focar no desenvolvimento de valor para o negócio. Esta abordagem reduz a complexidade técnica, melhora a qualidade do software e acelera os ciclos de entrega, promovendo um ambiente de trabalho mais produtivo e satisfatório.

A4: Platform Engineering impulsiona a produtividade das equipas ao fornecer infraestruturas padronizadas, automação e ferramentas integradas que simplificam processos complexos. Ao reduzir o

tempo gasto em tarefas operacionais e eliminar obstáculos técnicos, as equipas conseguem focar-se no desenvolvimento de funcionalidades de negócio. Além disso, ambientes self-service e fluxos de trabalho otimizados promovem maior autonomia e colaboração, acelerando os ciclos de entrega e melhorando a eficiência global.

A5: Sim, do ponto de vista de gestão de projetos, a Platform Engineering ajuda claramente a entregar mais valor ao negócio. Ao criar infraestruturas padronizadas e reutilizáveis, permite que as equipas desenvolvam soluções de forma mais rápida e consistente, promovendo uma maior agilidade. Uma vez criada a plataforma, o mesmo processo ou funcionalidade pode ser replicado múltiplas vezes sem necessidade de recomeçar do zero, reduzindo o tempo de entrega e aumentando a escalabilidade. Além disso, ao automatizar tarefas repetitivas e minimizar erros operacionais, as equipas conseguem focar-se em iniciativas estratégicas, acelerando o time-to-market e melhorando a capacidade de resposta às necessidades do negócio.

Interviewee 4

A1: Cloud Engineer DevOps 3 Years

A2: I have a theoretical understanding of Platform Engineering, primarily gained through books, blogs/websites and some insight provided by major Cloud Providers, but there's lack of information on this.

A3: I think that Platform Engineering will significantly improve my experience in work by providing self-service tools, standardizing workflows, and reducing repetitive tasks, with that I can focus more on coding and delivering features instead of dealing with infrastructure complexities.

A4: Platform Engineering enhances team productivity by automating infrastructure management. It reduces bottlenecks, minimizes downtime, and ensures that resources are used efficiently.

A5: Yes, Platform Engineering by providing a centralized platform that improves developer efficiency will help minimize errors and accelerates our time-to-market.

Interviewee 5

A1: Cloud Architect. 15 years.

A2: I'm in a beginner phase. The main sources are CNCF documentation and medium articles.

A3: Big impact. It requires an adaption to a microservices strategy. The learning curve can be bigger for those not working in that model. However, it can increase the capacity to deliver new features and create focus on specific tasks for each developer due to isolation in different services.

A4: The same as explain before. This model will increase the productivity.

A5: Yes. The time to market decreases using this module. It's possible to add new features more quickly and rollback faster when something is not working correctly. The capability to check the impact of the new features in isolated environments or even it Production is easier (Canary deployments for example).

Interviewee 6

A1: QA Engineer, 6 years

A2: I started in 2021, beginning with some AWS services, and from 2023, I have focused on GCP, mainly working with Apigee.

A3: Increased development agility and simplified infrastructure management.

A4: Platform Engineering enhances team productivity by streamlining workflows, automating repetitive tasks, enabling self-service capabilities, and fostering collaboration through standardized tools and processes. This allows teams to focus on delivering value rather than managing infrastructure.

A5: Platform Engineering accelerates time-to-market, reduces operational overhead, and enhances system reliability. It enables development teams to focus on business-aligned features, fosters innovation by removing bottlenecks, and optimizes resource use, driving cost savings and better IT-business alignment.

Interviewee 7

A1: Sales Engineer | 5 years

A2: My knowledge of Platform Engineering comes from my professional experience, mainly in the Cloud Computing area, where it is very common for clients to become aware of cross-cutting DevOps frameworks, where the concept of Platform Engineering is commonly debated.

I consider to have some theoretical knowledge, being able to debate the subject under discussion, however I don't have extensive experience in its practical application, taking into consideration as well that this is an emerging and recent topic.

A3: Certainly, the concept of platform engineering, the use of an IDP/Backstage solution and effectively building this type of "self-service" platform makes life easier in every way for developers, who can quickly have a "blueprint" environment ready to use, thus achieving greater security, agility, practicality, reability and even productivity with this type of practice.

A4: This type of platform and delivery solution allows development work to be disseminated quickly, integrates easily and allows hundreds of developers to work together without there being any differences between the templates used or governance and security practices.

In this sense, the productivity of working at scale and in a team is greatly affected positively, thus achieving an increase in productivity and work performance and an abysmal / easy, increase in the number of developer users, working together for the business growth.

A5: Clearly this type of practice delivers a lot of additional value to the business, I consider it to be an essential business vertical, especially when there is development activity at scale, where control and management is complex, and these platforms enable centralised governance of all development, control, security across the board and embedded in the self-service platform itself.

Interviewee 8

A1: Integration Engineer - 4y

A2: I have a good understanding of the role, having first heard of it at university, then studying for cloud certifications and often working in cross-functional teams.

A3: It allows the developer to focus on solving the business problems instead of wasting time on infrastructure setup, debugging network issues, etc.

A4: It increases efficiency by providing developers with self-service tools and pre-built infrastructure components. This removes bottlenecks and allows teams to focus on delivering value.

A5: It increases the reliability of the deliveries and reduces the time to market

Interviewee 9

A1: Cloud Architect, 10 years

A2: Self-Taught, Official Documentation

A3: A key benefit is the reduction in cognitive load - developers can focus more on writing code and delivering features rather than wrestling with infrastructure configuration, deployment pipelines, and operational concerns.

A4: The key impact comes from removing operational overhead and providing reliable, standardized tooling that teams can leverage without deep infrastructure expertise.

A5: The core benefit is enabling teams to deliver customer value more quickly and reliably while reducing operational overhead and risk.

Interviewee 10

A1: CTO Devoteam Distributed Cloud (Consulting Company); 7 years of experience

A2: I would say I have an intermediate level of knowledge regarding platform engineering contexts.

DevOps Foundation; https://platformengineering.org/; LinkedIn Posts from colleagues and authorities in the field; information from the 3 main cloud providers: MSFT Azure; GCP; AWS; lectures; conferences; books (like Platform Engineering: A Guide for Technical, Product, and People Leaders) and project and proposal experience.

A3: I think Platform Engineering is like giving developers the VIP treatment. It's like we've got a dedicated team taking care of all that tedious behind-the-scenes stuff, like setting up infrastructure, managing tools, and automating workflows. Which, in turn, frees up our developers to focus on what they love, which is usually coding and designing solutions. Ultimately, this leads to happier, more productive developers who can deliver better software faster and have access to most features they need through an Internal Developer Platform (IDP).

And it's not just about making things easier. I see Platform Engineering as fostering a more collaborative environment where developers and operations teams are on the same page, using the same tools and speaking the same language. This means less confusion, faster feedback loops, and a smoother path from idea to production.

A4: As mentioned above, I see Platform Engineering as a major productivity booster for the whole development team. We can streamlin entire development processes, removing all the roadblocks and bottlenecks that used to slow them down. By giving developers self-service access to the tools and infrastructure they need, they can work more independently and efficiently. Plus, with everyone using the same standardized environment and workflows, there's less confusion and wasted time on troubleshooting.

A5: In my experience, Platform Engineering is a game-changer when it comes to delivering value to the business. You are, essentially, building an high-speed railway for software development, allowing organizations to get new features and products to market much faster. By streamlining the development process and empowering developers to self-serve, in the speed and frequency of releases

increases, accelerating time to market roadmaps. This translates to quicker feedback from customers, faster iteration, and ultimately, a more responsive and competitive business.

Interviewee 11

A1: DevOps Engineer - 4 years

A2: I have some knowledge about Platform Engineering, as I worked in several DevOps projects and helped some clients adopt Platform Engineering practices.

My main source for that knowledge comes from reading newsletters regarding the topic, which give me a lot of insights and best practices that can be used to build seamless IDP's (Internal Developer Platforms). In addition, LinkedIn posts promoted by not only, experienced Platform Engineers but also from companies that mantain some of the most used tools in Platform Engineering, as Gitlab, ArgoCD, Kubernetes, Backstage, etc.

A3: It does impact the developer experience significantly, since it enhances it by streamlining workflows as well as reduces complexity. By focusing on building internal platforms and tools that help automate repetitive tasks and standardize processes, it allows developers to focus more on writing code and solving business problems rather than dealing with infrastructure or other issues not directly related with them. As a result, developers can deliver high-quality software more quickly and effectively.

A4: I would say Platform Engineering as a significant impact on team productivity, by simplifying processes and enabling self-service capabilities, it centralizes and standardizes the access to infrastructure, tools and workflows. This improves the developer experience and allows teams to quickly access the resources they need while adopting more effective collaboration across departments, this way reducing bottlenecks, accelerating development cycles and empowering teams to focus on delivering value.

A5: It does help deliver more value to a business, in my experience, providing a centralized platform that unifies and simplifies development tools enhances both productivity and overall business value. For example, consolidating a code repository, CI/CD pipeline and monitoring tools into a single interface, may simplify day to day development tasks and also reduce onboarding time for new team members and accelerate the release cycle, and consequently boosts product quality and team collaboration.

Annex 5 – Second Interview Round Answers

Interviewee 1

A1: From my own experience, Platform Engineering really makes a developer's life easier by taking those tedious, error-prone manual tasks and automating them, which honestly saves so much time and frustration. It's a game-changer for productivity because it speeds up things like setting up infrastructure, so you're not stuck waiting around, you can just focus on writing code.

For example, I've seen how automating environment setups cuts out all the repetitive config work, which not only reduces mistakes but also gets things moving faster. It also keeps everything consistent across dev, test, and production, so you don't run into those annoying "but it worked on my laptop" moments. Plus, with self-service platforms, devs can grab what they need and deploy quickly, which makes it so much easier to experiment and get features out the door. Overall, in my opinion it makes the whole process smoother, leading to better software and faster results for the business.

Interviewee 2

A1: My perspective on Platform Engineering is is overwhelmingly positive. It's not just a convenience; it's a fundamental shift that greatly impacts my daily experience, productivity, and ultimately, the value of my team's deliverables.

One great advantage is the autonomy and self-service that it can provide. Instead of relying on separate operations teams for every infrastructure need, I can provision these resources myself through the IDP's standardized catalog.

It also reduces cognitive load, promotes consistency and standardization and improves feedback loops. One other great advantage is it facilitates the onboarding for new members.

Interviewee 3

A1: Platform Engineering makes life easier for developers. It cuts down steps, reduces errors, and speeds things up—letting teams focus more on building and less on fixing.

Interviewee 4

A1: In my opinion, the study conducted clearly highlights the value of implementing an Internal Developer Platform (IDP). I believe that an IDP offers a self-service, intuitive, and user-friendly environment that significantly simplifies the deployment of various services. It enables faster, more automated processes with minimal manual intervention, which I see as a major benefit.

From what the study presents, it’s evident that the theoretical benefits often associated with IDPs do hold true in practice. In my view, the ability to reduce both the time and effort required for deployments, as well as the number of steps involved, is a strong argument in favour of their adoption. Furthermore, the reduced complexity compared to manual processes and especially the decrease in human error due to automation, is in my opinion, one of the most impactful advantages.

To summarise, I believe this study clearly demonstrates that using an IDP can lead to a notable increase in productivity, accelerating processes while improving overall efficiency. Based on the findings, I’m convinced that there is substantial added value in embracing the principles of Platform Engineering within any modern organisation.

Interviewee 5

A1: What your prototype shows is a clear efficiency gain by moving to an IDP. Fewer steps and faster deployments are concrete time-savers for developers. That directly translates to them pushing out code more quickly, which is what the business wants.

Beyond just speed, the lower complexity and error rates you're seeing are important for stability. Less time spent troubleshooting deployment issues means more reliable delivery. That's a tangible reduction in wasted effort and potential downtime, which has a direct cost.

From a practical standpoint, Platform Engineering, as you've demonstrated, isn't just about making developers happier (though that's a plus). It's about optimizing the software delivery process. You're getting more output, more reliably, with less overhead. That's a concrete improvement in how efficiently software gets built and deployed, and ultimately, how quickly the organization can realize value from its development efforts. The numbers you've collected make a pretty compelling case for the practical benefits of this approach.

Interviewee 6

A1: Based on my professional experience, the adoption of automation significantly reduces the implementation time for new solutions. This efficiency is achieved through the rapid replicability of automated processes and the elimination of human error, which is a common occurrence in manual operations. While the implementation of an IDP-based solution necessitates an initial learning curve and team familiarization, this investment is quickly offset by the accelerated deployment of resources, particularly those that are repetitive in nature. From my perspective, any "clickops" approach should be avoided, except in very specific scenarios such as simple Proof of Concepts or solutions intended for a single, one-time deployment.