UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

DevOps Monitoring in a Development Team

Bruno Miguel Pereira Susana

Mestrado em Engenharia Informática

Trabalho de Projeto orientado por:
Prof. Doutor Pedro Alexandre de Mourão Antunes

2025

Acknowledgments

To those who have played an important role in my academic and professional journey.

To my family, whose unconditional love, support, and encouragement have always been my foundation. Your belief in me has been a constant source of strength.

To TrustSystems, for providing me with the opportunity to grow and apply my knowledge in the real world. Special thanks to Ana Guimarães for the opportunity, and whose guidance and leadership have inspired me greatly.

To my advisor at TrustSystems, Artiom Andronic, for your feedback, patience, and for always challenging me to do better.

To Professor Doctor Pedro Alexandre de Mourão Antunes, as faculty advisor, for your invaluable mentorship and academic guidance. Your wisdom and dedication have greatly influenced the direction of this work.

To my friends and colleagues, for your continuous support and encouragement throughout this journey.

This work is a reflection of the collective effort and belief of all those who have supported me along the way. I am deeply grateful.

Thank you.

Resumo

No atual cenário tecnológico, marcado por rápidas transformações e por uma competitividade crescente, as organizações enfrentam desafios que vão muito além da simples criação de software. Hoje, não basta desenvolver soluções funcionais, é também necessário garantir que estas sejam entregues de forma ágil, com qualidade, segurança e capacidade de adaptação a contextos em constante mudança. A pressão para inovar é constante, e cada atraso ou falha pode representar a perda de oportunidades valiosas num mercado global altamente competitivo.

A competitividade não é mais medida apenas pela inovação tecnológica, mas também pela capacidade de transformar ideias em soluções funcionais de forma confiável e contínua. Assim, empresas que conseguem alinhar velocidade, qualidade e segurança nos seus processos de desenvolvimento conquistam uma vantagem estratégica significativa, consolidando-se num mercado que valoriza a adaptabilidade e a excelência operacional constante.

Durante décadas, os processos de desenvolvimento seguiram modelos rígidos e pouco flexíveis, como o tradicional Waterfall, onde cada fase era concluída antes do início da seguinte. Este tipo de abordagem mostrou-se insuficiente à medida que os sistemas se tornaram mais complexos e que as necessidades dos clientes passaram a evoluir com maior rapidez. A ausência de comunicação eficaz entre desenvolvimento e operações criava barreiras difíceis de ultrapassar, levando a atrasos, retrabalho e, muitas vezes, a um desalinhamento entre o que era produzido e o que era realmente necessário.

Foi neste contexto que surgiram metodologias mais ágeis, capazes de reduzir distâncias entre as equipas e encurtar ciclos de entrega. O movimento Agile abriu portas para uma maior flexibilidade e colaboração, mas rapidamente se percebeu que era necessário ir ainda mais longe. A partir desta necessidade nasceu o conceito de DevOps, uma filosofia que procura unir desenvolvimento e operações num único fluxo integrado, promovendo não apenas processos mais eficientes, mas também uma cultura de colaboração, automatização e melhoria contínua.

O DevOps trouxe uma nova visão para as empresas de tecnologia: ao invés de olhar para desenvolvimento e operações como áreas separadas, passou a encará-las como partes complementares de um mesmo ecossistema. A integração contínua, a entrega contínua e a automatização de testes tornaram-se práticas comuns, reduzindo falhas e acelerando a disponibilização de novas funcionalidades. Mais do que ferramentas ou pipelines, DevOps representa uma mudança cultural que valoriza a comunicação entre equipas, a responsabilização partilhada e a criação de soluções orientadas ao utilizador.

Neste contexto, um dos pilares que mais se destaca é a monitorização contínua. Se antes monitorizar era visto como uma tarefa reativa apenas para identificar falhas quando algo corria mal, hoje a monitorização assume um papel estratégico. Em sistemas cada vez mais complexos, compostos por microserviços, ambientes cloud e múltiplos utilizadores simultâneos, não basta reagir a problemas; é preciso antecipá-los, preveni-los e aprender com os dados.

A monitorização contínua fornece às equipas visibilidade em tempo real sobre o desempenho das aplicações, o uso de recursos e a saúde geral dos sistemas. Com dashboards claros, alertas configuráveis e análises históricas, é possível não só resolver incidentes de forma mais célere, mas também identificar padrões, prever picos de utilização e planear o crescimento de forma sustentável. A informação recolhida deixa de ser apenas técnica e passa a ser estratégica, influenciando decisões de negócio e sustentando a inovação.

É neste ponto que se enquadra o caso da TrustSystems que sempre teve como objetivo usar a tecnologia e a inovação para dar resposta às necessidades dos seus clientes, construindo parcerias de longo prazo baseadas em confiança e qualidade. Contudo, como muitas empresas que lidam com ecossistemas complexos, a TrustSystems deparava-se com uma limitação significativa: a ausência de uma solução de monitorização eficaz que acompanhasse a evolução do seu pipeline DevOps. Esta lacuna representava riscos reais para a organização. A falta de visibilidade dificultava a detecção precoce de falhas, aumentava o tempo de resolução de incidentes e, consequentemente, impactava a produtividade das equipas e a experiência dos clientes. Sem dados centralizados e estruturados, tornava-se também difícil implementar uma cultura de melhoria contínua, já que as análises eram incompletas e baseadas em percepções parciais.

Face a este cenário, a implementação de uma solução robusta de monitorização deixou de ser apenas uma necessidade técnica e passou a ser um objetivo estratégico. O presente trabalho surge precisamente dessa necessidade: desenhar e integrar uma solução de monitorização que ofereça à TrustSystems uma visão centralizada, clara de todo o seu ecossistema DevOps.

- Maior visibilidade e transparência: permitir que todas as equipas envolvidas no ciclo de desenvolvimento tenham acesso a dados fiáveis em tempo real.
- Resolução proativa de problemas: identificar falhas e anomalias antes que atinjam os utilizadores finais, reduzindo riscos e tempo de inatividade.
- Eficiência operacional: diminuir o esforço manual de diagnóstico através da automatização de alertas e relatórios, libertando as equipas para tarefas estratégicas.
- Cultura de melhoria contínua: analisar dados históricos para identificar padrões e oportunidades de otimização.
- Escalabilidade e flexibilidade: garantir que a solução acompanha o crescimento da organização, integrando novos serviços e ambientes de forma ágil.
- Reforço da segurança: centralizar logs e eventos para identificar comportamentos suspeitos e fortalecer práticas de cibersegurança.

Além dos benefícios técnicos, a integração da monitorização com ferramentas já presentes na organização, como o Jira, cria um fluxo de trabalho mais ágil e eficiente. Problemas detetados pela monitorização podem ser automaticamente registados como tarefas no sistema de gestão de incidentes, garantindo rastreabilidade e facilitando a comunicação entre equipas.

Para além da componente tecnológica, é fundamental reconhecer que a adoção de práticas DevOps e de soluções de monitorização implica também uma transformação cultural dentro da organização. As equipas deixam de atuar de forma isolada, passando a partilhar responsabilidades e a colaborar de forma mais próxima. Esta mudança promove um ambiente em que o conhecimento é disseminado, os erros são encarados como oportunidades de aprendizagem e a comunicação aberta torna-se parte essencial do dia a dia. Assim, a monitorização não é apenas um recurso técnico, mas também um catalisador de uma cultura mais colaborativa e transparente.

Outro ponto relevante prende-se com os benefícios de longo prazo. Uma solução de monitorização bem implementada contribui para reduzir custos operacionais, uma vez que diminui o tempo gasto em diagnósticos manuais e previne falhas graves que poderiam causar interrupções prolongadas. Além disso, ao aumentar a confiança nos sistemas e na capacidade de resposta da empresa, cria-se um diferencial competitivo que reforça a credibilidade perante clientes e parceiros. Em mercados cada vez mais exigentes, esta confiança traduz-se em relações comerciais mais sólidas e sustentáveis.

Sublinhar que a integração desta solução na TrustSystems não representa apenas uma melhoria operacional imediata, mas também um investimento estratégico no futuro da empresa. Ao alinhar tecnologia com objetivos de negócio, a organização reforça a sua capacidade de inovação, resiliência e adaptação a novas realidades. A monitorização contínua passa, assim, a ser não apenas um mecanismo de controlo, mas um verdadeiro motor de crescimento e diferenciação, posicionando a TrustSystems para enfrentar com sucesso os desafios de um setor em constante transformação.

É importante destacar que esta abordagem não pretende apenas resolver problemas imediatos, mas sim preparar a empresa para os desafios futuros. A evolução do setor caminha para conceitos como observabilidade e AIOps, onde a inteligência artificial e a análise avançada de dados assumem um papel central na gestão de sistemas complexos. Ao investir numa solução sólida de monitorização, a TrustSystems posiciona-se de forma a integrar estas tendências, garantindo que os seus serviços se mantêm competitivos, resilientes e inovadores.

Em suma, mais do que implementar uma ferramenta, trata-se de impulsionar uma mudança estratégica que reforça a qualidade e a fiabilidade dos processos de desenvolvimento, ao mesmo tempo que promove uma cultura de colaboração e inovação. Numa era em que a capacidade de adaptação é sinónimo de sobrevivência empresarial, investir em DevOps e monitorização contínua é investir na sustentabilidade e no futuro da organização.

Palavras-chave: DevOps, Desenvolvimento, Operações, Monitorização contínua, Desempenho

Abstract

In today's fast-paced software world, agility and adaptability are key to staying competitive. Companies are under pressure to release products and updates faster, without sacrificing quality or security. This is where DevOps really shines, bridging the gap between development and operations, helping the entire software life-cycle.

But DevOps is more than just a methodology or set of tools. It's a culture that encourages collaboration. By working closely together, development teams can focus on innovation, while operations ensure stability and performance in production. Continuous integration and delivery, boosted by tools like Jenkins, speed up testing and deployment, reducing delays.

Monitoring, meanwhile, plays a critical role. It gives teams a clear view of how systems are performing, letting them catch problems early on—before they escalate. Continuous monitoring ensures that software remains reliable and secure, from development all the way to production.

Choosing the right monitoring tools is vital. Prometheus and Grafana are popular for system and real-time monitoring. The ELK Stack (Elasticsearch, Logstash, Kibana) is widely used for log analysis, while tools like New Relic and Datadog also offer comprehensive solutions, especially for cloud environments like AWS, Azure, or Google Cloud Platform.

For monitoring to work effectively, it must be proactive. This means identifying potential issues before they affect users. Setting clear metrics, such as response times and resource usage, helps teams stay on track. It's important to regularly adjust these metrics to fit the project's evolving needs.

Incorporating continuous monitoring into daily routines helps teams stay on top of incidents, fostering a culture where problems are solved quickly, and reliability is consistently improved.

Keywords: DevOps, Development, Operations, Continuous Monitoring, Performance

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Companies must adapt quickly to today's evolving IT business and technological innovation pace in order to remain relevant and competitive. Failure to adapt to these changes quickly creates a substantial risk, endangering a company's ability to improve processes, develop creative solutions, and maintain a robust position in the volatile IT ecosystem. As a result, organizations need sense & respond systems and processes in their operations not as a merely strategic decision, but as a fundamental requirement for long-term success.

The advantages of having a DevOps pipeline are significant, as they greatly reduce the software delivery time and also have the effect of reducing potential conflict among development team members. Over time, it became necessary to evolve these pipelines so that they not only deliver software quickly but also deliver quality, stable, and reliable software. Additionally, a well-structured DevOps pipeline allows for better continuous integration and continuous delivery (CI/CD), facilitating the early detection of bugs and issues, resulting in less rework and greater efficiency in the development cycle. The automation of tests and constant monitoring are crucial elements that contribute to the continuous improvement of the product that is being developed.

This way, teams can focus more on innovation and less on repetitive and manual tasks, promoting a more collaborative and productive work environment. Currently, the organization TrustSystems is dealing with an inefficient monitoring solution that fails to provide complete insights into the complexities of its DevOps operations. The limits of the existing solution threaten a fast diagnosis and treatment of the operational issues, resulting in an accumulation of possible issues affecting software delivery and overall system stability. Regarding this, to address these challenges, an investment must be made in a more advanced monitoring solution capable of delivering real-time, comprehensive insights. This upgrade is essential for identifying and resolving these issues as soon as possible, thereby ensuring software delivery and maintaining system integrity.

1.1 Objectives

The present study suggests the implementation of a DevOps monitoring tool. This solution intends to improve different metrics and logs tracking, to improve coordination between the development and operations teams, optimizing the work of the operational ecosystem as a whole. This imple-

mentation is to be integrated in the company's DevOps pipeline leading to improvements in:

Increase Visibility and Transparency: Adopting this tool, the entire development life cycle will become more visible. This includes real-time monitoring of the application's performance, resource usage, and system health. This ensures both development and operations teams a clear view of what's happening.

Promote Proactive Problem Solving: With these advanced monitoring features, the tool will help teams spot potential issues before they become critical. This early detection and proactive problem-solving will help maintain system stability and reduce downtime.

Boost efficiency and Productivity: With all the monitoring processes being done automatically, this process will minimize the manual effort that is needed for tracking and troubleshooting. Automated alerts and detailed insights allow teams to focus on more strategic tasks, increasing the overall productivity.

Support Continuous Improvement: The data that is collected through the monitoring processes can be analyzed to identify trends and areas for improvement. This feedback will support continuous optimizations and the implementation of the best practices within the pipeline.

Scalability and Flexibility: The solution is designed to scale with the company´s growth, since it handles increased workloads and more complex systems. It also provides flexibility to adapt to the changing business needs and technological advancements.

1.2 Development Context

TrustSystems is integrated into the Inoweiser group, whose main objective is to use technology and innovation to meet the needs of its clients. It has over fifteen years of experience in both national and international markets, being present in more than 20 countries. The company is committed to delivering high-quality solutions and services, continuously evolving to keep pace with the latest technological advancements and industry trends. Through its team, TrustSystems aims to get long-term partnerships and drive success for its clients globally.

This necessity to implement a monitoring tool is pronounced given the company´s commitment to deliver high-quality software solutions to many different clients. This tool is a fundamental requirement to sustain and improve the current development process. As TrustSystems operates in a competitive market where the need to adapt to the client needs must always be present. With this, we can consider that this integration is not just a technological upgrade but a strategic move to improve the overall efficiency, reliability, and quality.

1.3 Document Structure

This study begins with a detailed explanation of the DevOps concept and its origins, the motivation behind its creation, and the key components that make up a DevOps pipeline. As part of this analysis, it will start with a historical context that led to the adoption of DevOps practices,

highlighting the challenges in traditional software development that DevOps aims to address and the main changes it took to get to the practices we have today.

Following this, the study will also introduce and elaborate on the concepts of Continuous Integration (CI) and Continuous Delivery/Continuous Deployment (CD). Where these practices are essential in modern software development pipelines, ensuring that code changes are systematically integrated, delivered, and deployed. Additionally, we will cover the concept of Continuous Monitoring, which will be the main focus, involving tracking possible future problems, trends in software's performance and quality at all stages.

After this, some monitoring tools will be shown addressing the main features of each alternative, followed by a brief explanation of the importance of dashboards in monitoring and their main purpose.

After laying out these theoretical concepts, the study will transition to a practical application, focusing on a specific problem associated with the company's existing architecture. This problem will be analyzed in depth, along with the current challenges. The proposed solution will be introduced, featuring a redesigned architecture to meet the company's needs. This section will present a possible solution with the chosen tool from the selected options for the solution, explaining their integration into the new architecture and justifying the choices made.

Finally, the document will provide a comprehensive description of the development process for the proposed solution, tracking its implementation from start to finish. A critical reflection on the results will be included, evaluating the effectiveness of the solution and discussing any lessons learned along the way. This reflection will serve as a guide for future improvements, ensuring that the system remains adaptable and responsive to ongoing changes.

Chapter 2

Methodology

This study has been divided into many stages that together complement the final result. The combined result is the implementation of the new monitoring tool in the company's DevOps pipeline.

Phase 1 - Literature Review on DevOps and Monitoring Tools: The first stage of the project involved conducting research on the current state of DevOps. The aim was to understand the key concepts and how a DevOps pipeline operates. This stage introduced the ideas of Continuous Integration (CI) and Continuous Deployment (CD), explaining their importance and how they correlate [38, 39]. Given the focus on code monitoring, the concept of Continuous Monitoring was explored in more detail with the comparison of some tools for the effect [40].

Phase 2 - Design the Solution: Once the state of the art was understood, the next stage was to design a possible solution in an abstract manner to first have a notion of how the whole structure works. This involved breaking down the problem into different sections such as: alerting, visual dashboards,external agents, and ElasticSearch Cluster [41, 42], this way defining the objectives of the solution. This stage is crucial as it ensures an overall overview of what needs to be done.

Phase 3 - Implementing the Solution: This third stage focused on the implementation of the solution in a concrete form. This started with researching the technologies needed for the solution together with the needs of the company, based on existing DevOps solutions, particularly in Continuous Monitoring processes [43, 44].

Phase 4 - Testing: In this stage, tests have been conducted to verify that the monitoring tools operated correctly and provided accurate, real-time insights into the pipeline's performance [44, 46].

Phase 5 - Evaluation and Optimization: The final stage involves tracking the performance of the implemented monitoring tools. Feedback from the cluster and the effectiveness of the monitoring system were analyzed. Based on this evaluation, optimizations were made to enhance the monitoring capabilities, ensuring they met the company's requirements and improved the efficiency of the DevOps pipeline [39, 47].

All this workflow through the different stages aimed at the development of an effective DevOps monitoring tool that improves monitoring, quality control, and overall pipeline performance.

Chapter 3

DevOps: State of the Art

3.1 Background on DevOps

Before DevOps, software development followed a methodology where each development phase was handled by a different team or department with almost no communication between them, also known as the Waterfall.

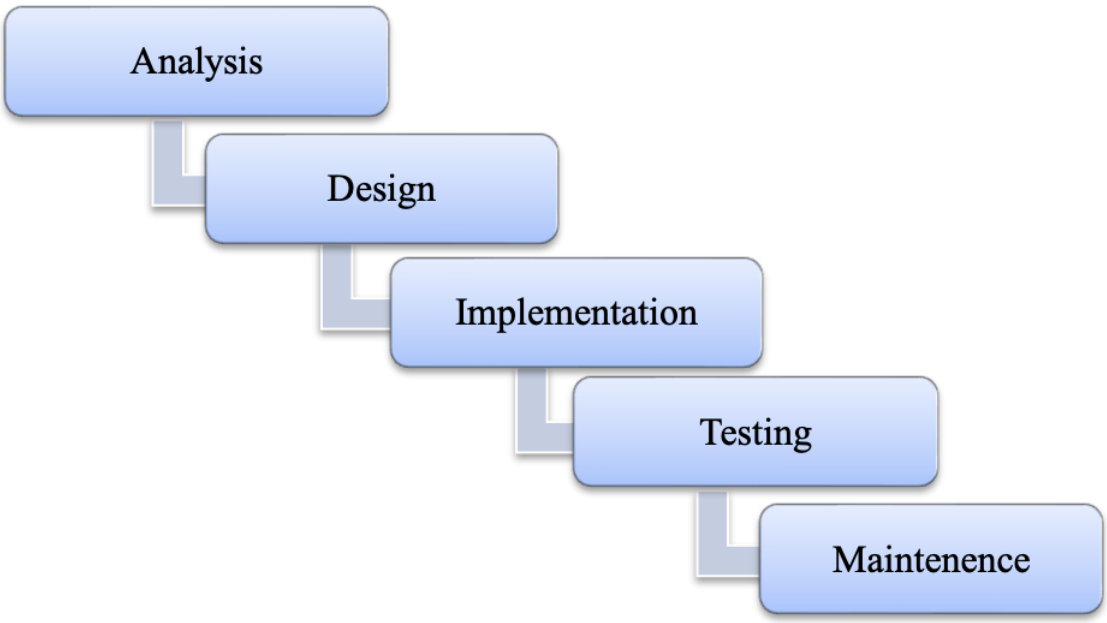

Figure 3.1: Waterfall Methodology
[1]

This methodology is characterized by ”separate and distinct phases of specification and development” [17, 19], each step must be fully completed before the next step can begin. At the end of each step, it is reviewed to ensure compliance with the requirements specified in the first step [18, 19]. This approach led to various problems as requirements and the volume of data from

products increased over the years; also, the lack of integration hindered efficiency and resulted in a lack of synergy across different aspects of the development life cycle [10, 12].

In the early 2000s, the software development industry faced challenges with traditional project management methods, which often led to delayed deliveries and budget overruns, also not meeting customer needs. In response to these challenges, a group of software developers came together in 2001 to create a new approach. This resulted in the creation of the Agile Manifesto, a document that outlined the core principles and values of Agile development [37].

The Agile Manifesto emphasized collaboration, flexibility, and customer satisfaction over rigid processes and documentation. It let teams adapt quickly to changing requirements and focus on delivering functional software in shorter cycles. This marked the beginning of Agile as a methodology, which has since become a popular approach for managing projects in various industries beyond software development [37].

As Agile techniques redefined software development by promoting flexibility, collaboration, and iterative procedures, it became clear that the entire software development life cycle needed to be automated [22]. Many issues connected with the previous Waterfall strategy were successfully solved by Agile, but the significance of integrating development and operations to achieve true efficiency and continuous delivery was also needed [22].

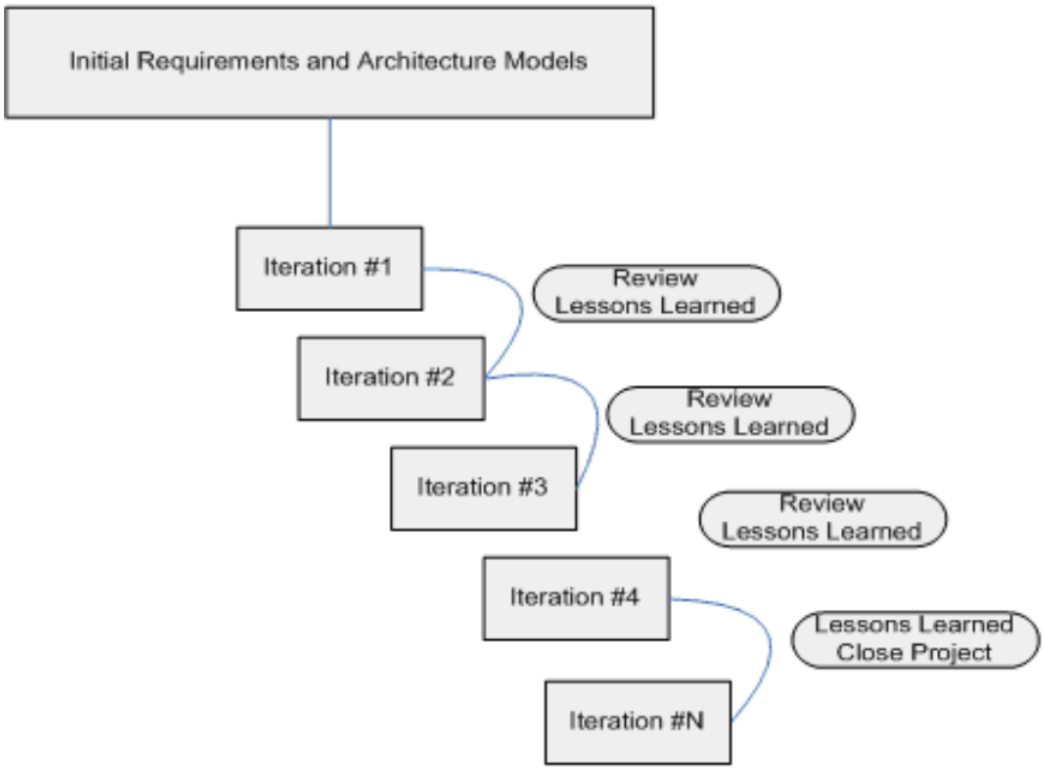

Figure 3.2: Agile Methodology
[16]

DevOps is a natural evolution of Agile, highlighting not only iterative development and collaboration, but also the integration of development and IT operations [22, 23]. DevOps seeks to break down team barriers by establishing a culture of shared help, continuous feedback, and accelerated delivery [22, 23]. Recognizing this need for a more flexible and collaborative approach, the Agile methodology emerged as a solution to address these issues of Waterfall. While Agile techniques vary in practices, they share common characteristics, including iterative development and a focus on interaction and communication. Developing in iterations allows the development team to adapt quickly to changing requirements [20, 21].

With that in mind, it was necessary to make a revolution in the way that the IT sector delivers a product to a client to fulfill its needs, and that was the turning point for DevOps, combining ”development” and ”operations.” This revolutionary approach established a collection of principles and practices aimed at improving collaboration and coordination between development and operations teams, who have traditionally been historically at odds in the IT industry. As DevOps continued to evolve, there emerged a recognition of the need to bridge not only technical teams but also business stakeholders, giving rise to what is now known as BizDevOps [48].

This approach brings business, development, and operations teams together to ensure that business objectives are tightly integrated into the development pipeline from the outset. BizDevOps extends the principles of DevOps by prioritizing business alignment, emphasizing faster delivery cycles that are driven by immediate market needs, and fostering a shared culture of accountability across all stakeholders [48]. DevOps emphasizes automation, continuous integration, and continuous delivery to streamline the software development and deployment processes [10, 12].

These different and independent domains, previously isolated, now work together to accelerate the software development life cycle [13]. This promotes a culture of shared responsibility and continuous improvement, resulting in faster delivery, greater quality, and increased satisfaction among customers. By breaking down barriers and differences, creating an environment where developers and operations teams contribute to the entire software delivery process [2].

3.2 CI/CD Pipeline

To apply this DevOps methodology described before, there’s the need to implement a CI/CD pipeline where code modifications are automatically deployed. This implies establishing a collection of processes that assist developers in efficiently building and testing their apps, thereby decreasing risk and improving quality.

The primary advantage of a CI/CD pipeline is that it enables developers to make quick adjustments while maintaining production environments [3]. From a business perspective, this procedure offers multiple advantages because it saves software delivery time, resulting in cost savings, and ensures that the software produced is stable. Additionally, CI/CD contributes to increased overall reliability and allows for more frequent and reliable releases.

In this domain, Continuous Integration can be perceived as the ability to make frequent code changes into the main branch of a shared code repository, immediately testing each change after

a commit or merge, and automatically running a build [4]. This iterative process allows issues and vulnerabilities to be recognized and fixed more quickly and earlier in the development process [14]. By integrating code changes frequently, teams can identify and address potential problems in the initial stages of development, minimizing the chances of these issues causing delays or disruptions later in the software delivery life cycle [32].

After the code has been tested and built as part of the Continuous Integration process, Continuous Deployment takes over in the final stages to guarantee it's packed with everything it needs to deploy to any environment at any time [4]. This ensures that the software is deployment-ready and can be pushed to production at any moment [32].

Teams can either manually trigger the deployments or switch to continuous deployment, where deployments are also automated [4]. Automated deployments in Continuous Deployment simplify the release process, reducing the potential for human error and enabling rapid, consistent, and reliable software releases [32].

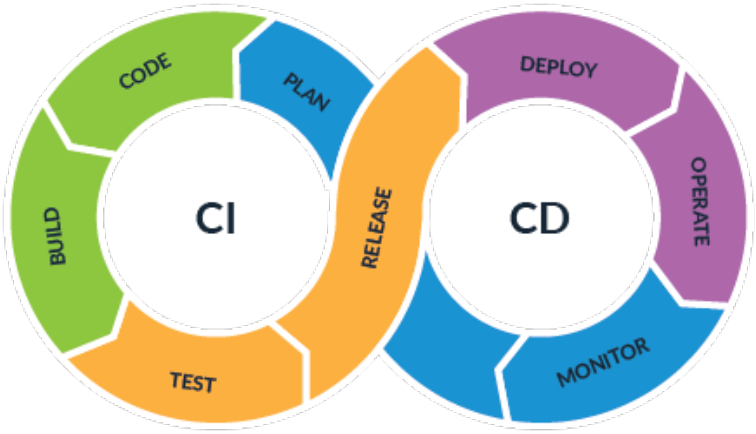

Figure 3.3: Continuous Integration / Continuous Deployment Relation [5]

3.3 Continuous Monitoring

This process of collecting, analyzing, and displaying systems, networks, and data for signs of performance degradation is commonly called continuous monitoring [15]. It promotes the identification and correction of issues as fast as possible on a system or an application, helping teams avoid degradation of service and the need to correct bigger problems with the accumulation of technical debt [6].

Continuous monitoring is an essential component in maintaining a robust and efficient software environment. This is typically done via an automated DevOps monitoring solution or a collection of continuous monitoring tools that gather data on various critical areas.

Server Monitoring ensures that servers are operating optimally, tracking metrics like CPU utilization, memory usage, and disk space to prevent downtime [8].

Cost Monitoring focuses on tracking and analyzing expenses related to cloud services or infrastructure usage, ensuring that the business stays within budget and resources are utilized efficiently.

Network Monitoring involves observing network traffic, latency, and bandwidth to ensure smooth communication between systems and prevent bottlenecks.

Application Monitoring is crucial for tracking the performance of specific applications, focusing on response times, error rates, and user satisfaction, focused mainly on the overall user experience.

Usage Monitoring focuses on understanding how users interact with an application to provide insights into user engagement, performance, and conversion patterns. This type of monitoring goes beyond tracking errors or performance issues by offering a detailed look at user activity, including page views, session durations, and event-triggered actions [49].

Finally, **Infrastructure Monitoring** provides a wider view, covering the entire IT infrastructure, including servers, databases, and networks. This approach ensures that all components of the system are working together correctly.

Regarding this, one of the key elements of this monitoring process is the collection, analysis, and utilization of logs. In this DevOps context, logs are crucial sources of information to track the behavior of software applications. As companies continue to implement DevOps principles, the importance of log management and analysis grows over time. Effective log management not only aids in troubleshooting and debugging but also plays a vital role in enhancing security by identifying and responding to potential security incidents.

This practice of obtaining logs (logging) can be applied to many places, such as CI (continuous integration), which we already discussed, where teams can check if the code builds are running successfully or if errors or warnings have occurred. Additionally, in CD (continuous delivery/deployment), monitoring failed deployments and identifying any potential problems through logs also becomes a critical aspect of maintaining a smooth, reliable, and efficient deployment process.

The image below shows a Jenkins pipeline in the build process, but an issue causes it to fail and prevents the execution of all the planned steps, resulting in an error. These types of issues are critical, particularly in a production environment, where availability is essential for end users. Such problems must be addressed immediately to prevent disruptions. When a reliable alerting system is in place to notify the appropriate personnel, issues can be resolved more quickly and effectively, giving the company more time to address the root cause and maintain its credibility with customers.

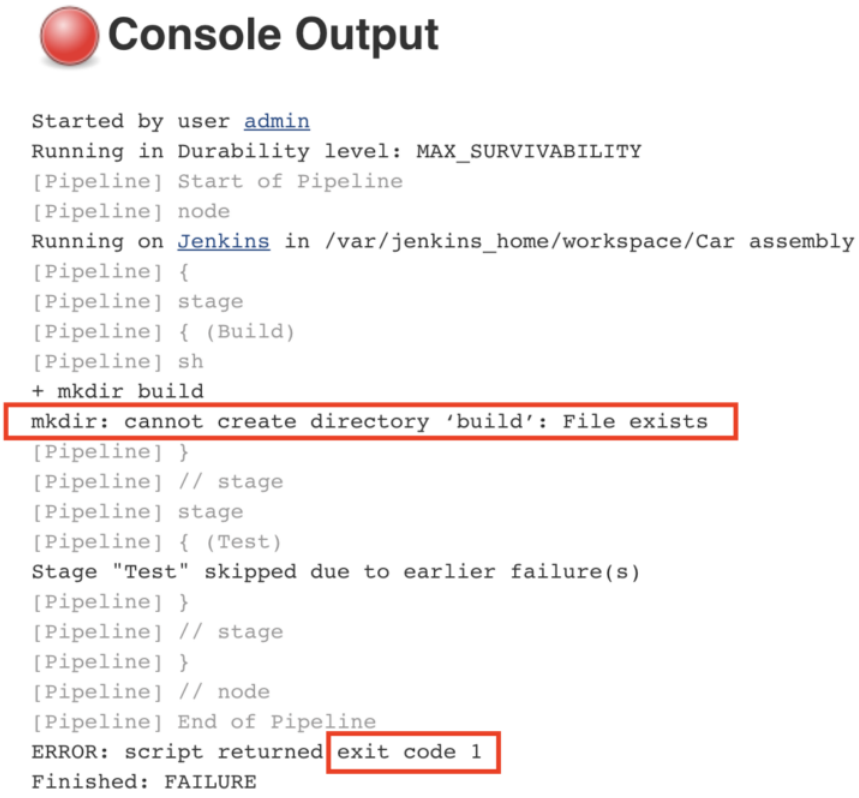

Figure 3.4: Jenkins Log Example
[9]

Monitoring software operations effectively involves gathering metrics such as API access and microservices performance. Implementing comprehensive monitoring systems provides quick identification and resolution of issues, consequently improving system reliability and user satisfaction.

This includes feedback and click-through analysis as a way to enhance the user experience as some studies emphasize the importance of user feedback in iterative design processes, leading to improved usability and user satisfaction [1]. With the help of analytics tools like Google Analytics, it can provide valuable insights into user behavior, enabling data-driven decision-making for website and application optimization.

Also, teams can be monitored in search of possible problems; for that, monitoring backlogs, feature development, and overall project progress are all important components for effective team monitoring. This improvement can be achieved by promoting agile methods, such as Jira or other agile tools [24]. Many studies demonstrate how agile approaches improve teamwork and project results [25].

3.4 Monitoring Dashboards

Dashboards play a crucial role in DevOps monitoring by providing clear and accessible insights in a visually intuitive manner. These dashboards offer a simple yet powerful way to showcase and comprehend the vast amount of data collected, enabling teams to monitor, analyze, and act on real-time data from applications, servers, and networks [11].

In the DevOps context, these dashboards process data into visually intuitive displays, enhancing decision-making and enabling users to understand the system's status effortlessly. Acting as a central hub for visualizing key metrics and performance indicators, dashboards empower users with interactive features for customizing views based on defined criteria from the user [11].

Furthermore, dashboards contribute to proactively identifying performance trends and potential issues. They provide an interactive interface for users to track and interpret data, supporting rapid decision-making processes.

This visual representation aids in troubleshooting, trend analysis, and overall system understanding. Various tools, including Prometheus, Grafana, and Zabbix, offer robust dashboard functionalities.

These tools allow users to implement personalized dashboards to their monitoring needs.

The following figure serves as an illustrative example of a Zabbix dashboard, showcasing the capabilities of these tools in presenting comprehensive data in a visually appealing format.

The dashboard is designed with customizable widgets, enabling users to display their specific monitoring needs. Widgets may include graphs, charts, maps, and other visual elements, presenting data on parameters such as network traffic, server CPU usage, memory consumption, and more. This flexibility ensures that users can easily configure the dashboard to focus on the metrics most relevant to their monitoring objectives.

The inclusion of color indicators and intuitive graphs enhances the ease of interpretation, allowing for a faster identification of trends, outliers, or potential issues that require attention.

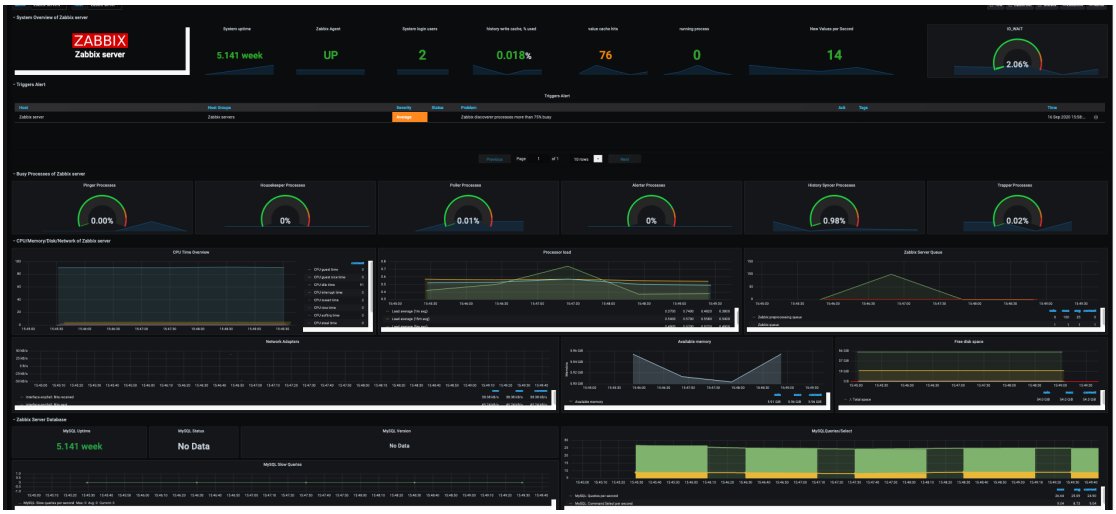

Figure 3.5: Zabbix Dashboard
[7]

3.5 Monitoring Tools

With the exponential increase in complexity over time, there has been a corresponding rise in the demand for effective and reliable monitoring solutions. Monitoring tools became an integral component of DevOps, ensuring the efficient operation of applications and infrastructure throughout the development and deployment life cycle.

These tools must be highly adaptable, capable of handling significant amounts of data, and seamlessly integrated into automated pipelines. Given their critical role, they provide valuable insights into metrics and logs, enabling teams to make well-informed decisions, address issues, and foster a culture of continuous improvement. The use of monitoring tools is paramount in identifying performance bottlenecks, detecting anomalies, and ensuring that deployed software meets the necessary standards for performance and reliability.

As an initial filtering method for all the available monitoring tools, Google Trends was used to assess global interest over time. By identifying tools that consistently ranked among the most-searched solutions, the focus was placed on those with demonstrated popularity and practical application within the industry.

This approach allowed us to prioritize tools with an established reputation, ensuring that they are widely trusted by the community and have big support.

These include Grafana, Prometheus, Zabbix, Nagios, Datadog, Elasticsearch, and Splunk. These tools not only reflect sustained interest but are also trusted by professionals for their reliability, scalability, and robust features, making them suitable for monitoring.

As a filtering method for the previously available monitoring tools, and based on TrustSystems requirements, such as being open-source, the ability to track both logs and metrics, and the capability to send alerts to responsible users were considered. These tools align with the operational

needs of TrustSystems.

————————	**Open Source**	**Log Tracking**	**Metric Tracking**	**Alerting**
Grafana	O	O (with integrations)	O (with integrations)	O (with integrations)
Prometheus	O	X	O	O
Zabbix	O	O	O	O
Nagios	O	X	O	O
Datadog	X	O	O	O
ELK	O	O	O	O (with integrations)
Splunk	X	O	O	O

Table 3.1: Comparison of Monitoring Tools

Looking at the previous table, it's clear that there are several monitoring tools that can be used for this project since many of them meet the basic requirements by TrustSystems. These tools provide the necessary features, such as being open source, supporting log tracking, metric tracking, and offering alerting capabilities.

While each of these tools could fulfill the project's functional needs, the final choice was not based solely on their technical features. Instead, it was influenced by the tool's ability to integrate in a good way with the existing systems, a critical factor that will be discussed more in detail in Section 4.3.

Other important criteria included ease of use, community support, and the availability of customization options, which are essential for ensuring long-term scalability and efficiency.

The selected tool not only met the technical requirements but also proved to be the most suitable for aligning with the current operational ecosystem of TrustSystems, balancing functionality with practical application.

Chapter 4

Solution Design

4.1 The Need for Monitoring: Problems and Objectives

4.1.1 The Problem

Maintaining a robust and reliable DevOps pipeline is crucial for ensuring the continuous delivery of high-quality products. Currently, TrustSystems, a company specializing in the IT sector, is facing a significant challenge in its DevOps processes. Despite implementing various automation tools and practices, one critical component remains unaddressed, which is monitoring. At present, there is no monitoring system in place to track the performance, health, and efficiency of the DevOps pipeline. This lack of monitoring leaves TrustSystems vulnerable to undetected issues, potential system downtimes, and inefficient resource utilization, ultimately impacting the quality of their services and products.

The lack of monitoring not only threatens the team's ability to identify and resolve issues promptly but also prevents the collection of valuable data that could inform future optimizations. As a result, the company is exposed to increased risks, security vulnerabilities, and delayed product releases. To address this problem, the implementation of a robust monitoring strategy within the DevOps pipeline is essential. This following work will explore this problem in the TrustSystems DevOps pipeline, analyze potential monitoring solutions, and propose a tool to address these problems.

4.1.2 Needs and Monitoring Objectives

This process started with identifying the needs from stakeholders in which, in this case, are the TrustSystems development, operations, and QA members, where conducting a discussion to understand their specific monitoring needs and expectations must be done.

This also includes gathering input for crucial metrics, performance indicators, and potential challenges that may come over time or even new ideas to be implemented. With the definition of these goals and needs aligned with the proposed objectives for the DevOps pipeline, those will contribute to the enhancement of system reliability, performance, and overall efficiency.

At this point, the discussed ideas and objectives must be crossed with the historical data analysis to highlight patterns, recurring issues, and areas that require special attention. This step is

itself really important for the final product since it allows one to define in an early stage the path to be followed and the best strategy.

With the assistance of new monitoring technologies and tools, incorporating monitoring into the DevOps cycle may be difficult and time-consuming, and there is no correct or better approach to monitor development and operations environments. The answer depends on the system and environment, but in all cases, an analysis must be performed with the goal of answering this question based on an analysis of system data. Even with the best/most used monitoring tech, adding it to the DevOps cycle can be tricky and time-consuming. There's no better solution since, as explained before, all have their own pros; it depends on the specific system and environment.

In addition, as technology keeps evolving, so monitoring practices must adapt. Regular reviews and updates are necessary to stay on top of emerging challenges. Encouraging collaboration among teams is crucial. This ensures monitoring practices stay agile, responsive, and aligned with the changing needs of the development and operations environments.

In today's software development domain, it's pretty much a given for companies to adopt a DevOps cycle. As mentioned earlier, it significantly speeds up the whole development and delivery process. This section is all about defining and explaining the problem we're aiming to solve in this project, and as described, a good and reliable pipeline's implementation is mandatory among IT companies to reduce the time to deliver software.

As previously stated, currently an IT organization relies on a DevOps pipeline to accelerate the delivery process, so in this chapter the present problem that motivated this work will be detailed, as well as a solution to that problem proposed and implemented.

Aligning the requirements from TrustSystems, the focus was on several key areas. For logging, the primary need was access to the logs and a way to save them, ensuring that all relevant data from the system, applications, and services is captured and made available for analysis. In terms of metrics, the monitoring solution must cover critical system resources such as CPU, disk usage, memory, and network performance, ensuring a clear understanding of resource consumption and potential bottlenecks.

Regarding this, there's an overall requisites list that describes what is intended from TrustSystems regarding the many aspects described before.

- **Logs**
  - Centralized Logging System: A single platform to collect, store, and query logs from all environments (e.g., development, staging, production).
  - Structured Logging: Ensure logs follow a consistent format for easier parsing and analysis (e.g., JSON format).
  - Retention Policies: Define log retention periods to balance compliance needs and storage costs.
  - Real-time Log Streaming: Immediate access to logs for debugging and monitoring.

  - Search and Filter Capabilities: Advanced query capabilities to drill down into specific log data quickly.
  - Log Anomaly Detection: Identify unusual patterns in logs that could indicate performance issues or security breaches.

- **Metrics**
  - System Metrics: Monitor resource usage such as CPU, memory, disk I/O, and network traffic for all components.
  - Application Metrics: Track application-level metrics like request rates, error rates, and latency.
  - Dashboards: Visualize metrics in real-time through customizable dashboards.
  - Historical Data: Store and analyze metrics over time to identify trends and forecast future needs.

- **Alerts**
  - Custom Alerting: Define custom alert thresholds and rules for different types of events (e.g., high latency, CPU over 90%).
  - Multi-channel Notifications: Deliver alerts through various channels like email, Slack, SMS, or incident management tools (e.g., Jira).
  - Severity Levels: Categorize alerts by priority (e.g., critical, warning, info) to avoid alert fatigue.
  - Alert Aggregation: Group related alerts into a single incident to reduce noise.
  - Automated Remediation: Trigger automated workflows in response to certain alerts (e.g., auto-scaling or restarting services).
  - Rate-limiting and Deduplication: Prevent duplicate alerts and control notification frequency.

4.2 Existing Technologies and Corporate Procedures

4.2.1 Current Situation

To develop a robust tool that fits the needs, first this process starts with identifying the needs from stakeholders in which in this case are the TrustSystems development, operations, and QA members, where conducting a discussion to understand their specific monitoring needs and expectations must be done.

This also includes gathering input for crucial metrics, performance indicators, and potential challenges that may come over time or even new ideas to be implemented. With the definition of these goals and needs aligned with the proposed objectives for the DevOps pipeline, those will contribute to the enhancement of system reliability, performance, and overall efficiency.

At this point, the discussed ideas and objectives must be crossed with the historical data analysis to highlight patterns, recurring issues, and areas that require special attention. This step is itself really important for the final product since it allows one to define in an early stage the path to be followed and the best strategy.

With the assistance of new monitoring technologies and tools, incorporating monitoring into the DevOps cycle may be difficult and time-consuming, and there is no correct or better approach to monitor development and operations environments. The answer depends on the system and environment, but in all cases, an analysis must be performed with the goal of answering this question based on an analysis of system data.

Even with the best/most used monitoring tech, adding it to the DevOps cycle can be tricky and time-consuming. There's no better solution since, as explained before, all have their own pros; it depends on the specific system and environment.

In addition, technology keeps evolving, so monitoring practices must adapt. Regular reviews and updates are necessary to stay on top of emerging challenges. Encouraging collaboration among teams is crucial. This ensures monitoring practices stay agile, responsive, and aligned with the changing needs of the development and operations environments.

Nowadays, in the IT sector, companies must adopt a DevOps cycle to boost productivity and feedback from their products. As mentioned earlier, it significantly speeds up the whole development and delivery process.

Currently, an IT organization relies on a robust DevOps pipeline to accelerate the delivery process, so in this chapter, the present problem that motivated this work will be detailed, as well as a solution to that problem proposed and implemented.

As shown in the following figure, the current TrustSystems pipeline does not include an implemented monitoring system.

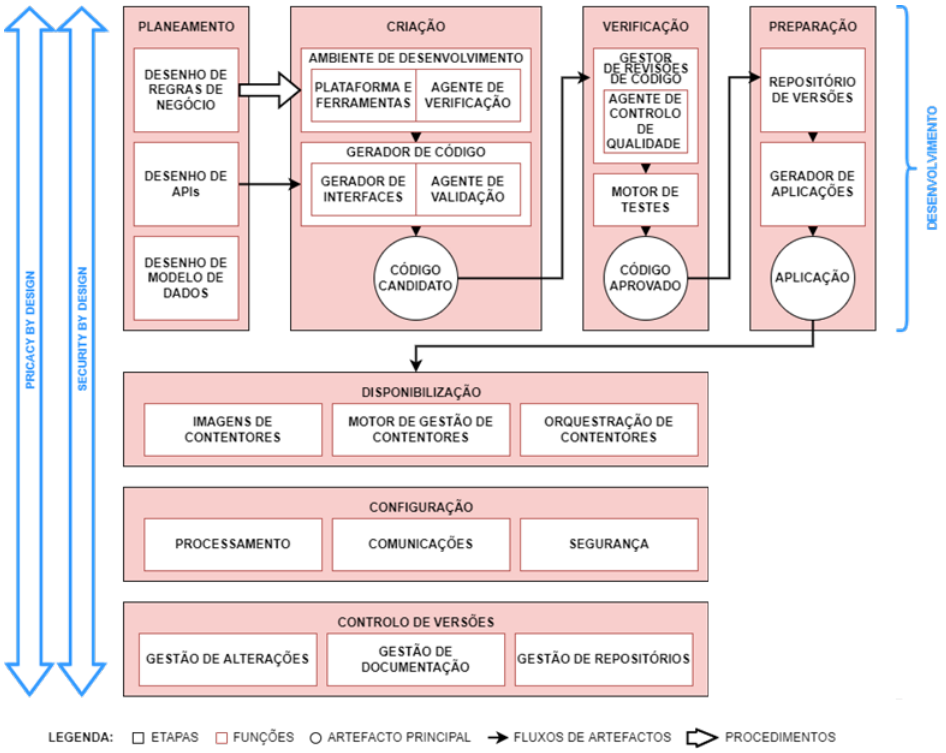

Figure 4.1: Trustsystems Pipeline
[16]

The figure 4.1 shows the current TrustSystems pipeline architecture, where every project begins with Planning, a critical stage that outlines goals, scope, stakeholders, APIs, and data models. After that, a JIRA project is created to track and handle work, encouraging openness and responsibility among team members.

Following the completion of the design and planning stage, the required scripts for a local environment are often developed along with a development environment that is created in the cloud. The steps involved in setting up a development environment are: setting up a PostgreSQL database, in this case on the cloud; setting up a BitBucket file repository controlled by the Nexus Repository tool; and using this tool to add another software build tool, in this case Jenkins [16].

Regarding the local development environment, all Docker scripts required to generate the requisite containers for this environment must be created [16].

The development process itself is the next major phase, it is split into two separate teams, the Frontend and Backend teams, that collaborate to produce the final product. The Backend team is in charge of creating all REST API calls and makes sure that the database data is used to generate the responses for Frontend requests made over the HTTP protocol [16]. On the other hand, the Frontend team is in charge of presenting those results; this is the layer that users interact with, it is imperative that they take both usability and rigor into consideration [16].

The SpringBoot Java framework is an open source platform for the development of Java applications that run on the Java Virtual Machine, and it will be used to develop the project's Backend [16]. To ensure efficiency and consistency in product development, the Backend team generates

classes in the framework using an OpenAPI file and the database. Usually there are three different layers such as Rest, Data, and Persistence that are developed utilizing this method. The remaining tasks include data entry and output, data for choices and validations, and persistence through database communication to save or retrieve data [16].

Frontend teams prefer to utilize Microsoft Visual Studio Code to build Angular (a TypeScript-based framework), CSS, and HTML, while Backend teams use JetBrains' IntelliJ concept IDE for code development. The Backend team still makes direct HTTP request calls and parses the JSON response to validate and check RESTful API results [16].

4.2.2 Company Technologies

The company has already implemented several technologies to enhance its development and deployment processes. Since our solution relies on a DevOps pipeline, Jenkins is central to the continuous integration and deployment pipeline, automating the building, testing, and deployment of code. Docker is also another technology in use, allowing the packaging of applications into containers to ensure consistent deployment across various environments. To manage the networking of these containers, the company utilizes Traefik, which works alongside Docker to handle network routing and enable communication between containers through designated URLs using DNS.

The databases utilized in this solution will be based on PostgreSQL, aligning with the company's existing standards for managing data across its projects. PostgreSQL is known for its reliability, flexibility, and advanced features, making it a robust choice for handling complex queries and data operations. In addition to PostgreSQL, Liquibase will play a vital role in managing database schema changes. This tool automates the process of applying updates and alterations to the database structure, ensuring that the correct schema is in place whenever a software service is initiated.

The foundation of the system testing project will be built in Java, using Maven as the build automation tool. Maven will manage dependencies, including JUnit, which will be crucial for supporting the creation of automated tests in Java.

Finally, to enhance active monitoring of system performance and security, the company will integrate Elasticsearch and its features into the CI/CD pipeline. Elasticsearch will be used to provide real-time search, analysis, and visualization of logs and metrics, enabling the company to proactively monitor system behavior and detect anomalies. By using Elasticsearch's capabilities, the company can gain deeper insights into system performance and security, facilitating quicker identification and resolution of potential issues before they impact production.

4.2.3 Corporate Dynamics and Procedures

When developing the solution, it's important to consider the common practices already followed by TrustSystems in its development environments. Typically, the company sets up four distinct environments for each project. Developers work in a local environment where they write and

test their code. Once the code is ready, it moves to a development environment, which hosts the latest software version for manual testing before it progresses to the Quality Assurance (QA) environment. Only after passing the QA phase does the software get deployed to the production environment. All of these environments are built using Docker containers, and they operate within a network managed by another Docker container running a Traefik image.

Also, open-source code is always a priority in choosing tools over other paid tools.

4.2.4 Corporate Interest with the new tool

The introduction of a new monitoring tool within TrustSystems DevOps pipeline is essential to meet the daily demands of this IT company. Beyond simply addressing technical gaps, the tool is designed to provide actionable insights and facilitate day-to-day operations, ensuring the reliability and efficiency of the company's systems. To understand its significance, it is crucial to explore the daily, real-world needs driving the requirements for concepts, metrics, alerts, and logs.

4.2.5 How TrustSystems Will Use the Monitoring Tool

In the daily operations of TrustSystems, this monitoring tool will serve as a critical enabler of proactive management and continuous improvement:

- **Proactive Issue Resolution:** By monitoring system health and setting appropriate alerts, teams can detect and address issues—such as resource bottlenecks or application errors—before they impact end users.

- **Data-Driven Optimizations:** Metrics and logs will enable teams to analyze trends, identify recurring issues, and refine processes. For instance, historical data on deployment durations can guide efforts to streamline release workflows.

- **Enhanced Team Collaboration:** The centralized monitoring platform will ensure that all stakeholders—from developers to operations and QA—are working with the same data, fostering quicker resolutions and more aligned decision-making.

- **Improved Customer Satisfaction:** By minimizing downtime and ensuring optimal system performance, the tool will directly contribute to a better user experience, strengthening TrustSystems reputation and client trust.

- **Security and Compliance:** Regular analysis of security logs will help the company detect vulnerabilities, prevent breaches, and maintain compliance with industry standards.

In addition to addressing immediate operational needs, the monitoring tool will cover the needs for long-term success. As TrustSystems evolves, the tool will adapt to new challenges, such as supporting the integration of emerging technologies or scaling to meet increasing workloads. Regular reviews and updates to the monitoring strategy will ensure its continued relevance and effectiveness.

The monitoring tool is a huge increment and a fundamental necessity for TrustSystems daily operations and strategic growth.

4.3 Tools Selection and Evaluation Criteria

4.3.1 Choosing the Right Tool

When implementing the solution within a company's existing production environment, careful consideration is crucial to ensure a smooth integration with minimal disruption and fewer needed changes. A key aspect of this process is the efficient use of the systems and infrastructure already in place. Considering the existing tools and frameworks will accelerate the development and also maintain consistency, reducing the risk of conflicts or redundancies that could appear from introducing new, overlapping technologies.

To minimize the disruption to coming operations is also essential. To achieve this, this new solution should ideally be rolled out in different testing phases, allowing testing and stabilizing each component before full deployment.

Finally, an analysis to identify opportunities for optimization, with the purpose of improving efficiency. This ensures that all new tools and procedures are in accordance with the needs.

The preference for open-source technologies with free licenses is a key factor when choosing the right tool according to the organization's strategy, aiming to minimize costs while maximizing flexibility and control over the software stack. Open-source tools typically offer more opportunities for customization, which is essential when developing a solution to the specific needs.

4.3.2 Chosen Tool

When evaluating the many possible options, it became apparent that while many tools serve similar fundamental purposes, they don´t vary significantly in terms of customization capabilities, scalability, community support, and long-term viability. The ELK Stack emerged as the most suitable option due to its strong alignment with these criteria:

Customization: ELK Stack's open-source nature enables deep customization to fit the specific needs of the project. Unlike proprietary solutions, which might limit modification capabilities, ELK allows extensive adaptability, ensuring the tool evolves alongside the organization's needs.

Scalability: As the organization grows, the volume of data generated is expected to increase. The Elasticsearch component of the ELK Stack is known for its robust scalability, enabling the handling of large data sets efficiently without sacrificing performance. This makes it a reliable option for long-term use, even as demands increase.

Community Support and Ecosystem: With a large and active community, the ELK Stack benefits from continuous development, extensive documentation, and a wealth of user-generated resources, such as plugins and configurations, which can be easily accessed and implemented.

Ease of Integration: The ELK Stack was chosen in part because of its ability to integrate smoothly with the existing infrastructure. This reduces the need for significant modifications, avoiding potential disruptions and additional costs.

Long-Term Viability: Open-source projects with strong community backing, like the ELK Stack, tend to have longer lifespans and more consistent updates compared to proprietary tools,

whose future development may depend on the strategic decisions of the vendor. By choosing ELK, the organization ensures that it is investing in a solution that is likely to continue evolving and receiving updates, thus ensuring its relevance and reliability over time.

If adjustments are required, they shouldn't damage the existing system and justify the investment. As the goal is to create a straightforward integration, tools with more functionality, the preference is to use open-source technologies with free licenses, which aligns with the existing company concept.

When comparing the many different options available for monitoring and log management, it's clear that they all serve the same fundamental purpose, and the choice often comes down to specific needs and preferences. With this in mind, the final choice should be guided by how well the tool aligns with the overall requirements, the level of customization needed, and the long-term scalability and support available.

With this, the ELK Stack offers flexibility, allowing a robust solution precisely to the organization's needs. Its open-source nature provides freedom from licensing constraints and the ability to modify and extend its capabilities as required. The robust scalability of Elasticsearch ensures that it can handle increasing data volumes without a drop in performance, making it a future-proof option.

TrustSystems itself had no affiliation with a monitoring tools stack so the choice became a free choice, where the decision has been made after evaluating online documentation, reviews, and ease of use also stack forums with the amount of active users. This to select a tool from a big amount, ensuring that they were not only compatible with the existing infrastructure but also aligned with the long-term goals for scalability, customization, and support.

4.3.3 ELK Stack and Integrations

Elasticsearch is a distributed, RESTful search and analytics engine capable of addressing a growing number of use cases. As the heart of the Elastic Stack, it centrally stores data for lightning-fast search, fine-tuned relevancy, and powerful analytics that scale with ease [26, 27].

Logstash is a free and open server-side data processing pipeline that ingests data from a multitude of sources, transforms it, and then sends it to your favorite "stash" [26].

Kibana is an open-source data visualization and exploration tool designed for use with Elasticsearch. Kibana provides a web-based interface that allows users to interact with data stored in Elasticsearch, enabling them to visualize and analyze large volumes of data [26, 27].

Jira - Jira Software is the most used agile project management tool used by teams to plan, track, release and support world-class software with confidence. Empowering autonomous teams with the context to move quickly while staying connected to the greater business goal. Whether used to manage simple projects or to power your DevOps practices, Jira Software makes it easy for teams to move work forward, stay aligned, and communicate in context [28, 29].

Metricbeat - Metricbeat is a lightweight shipper that can be installed on a server to periodically collect metrics from the operating system and from services running. Metricbeat takes the

metrics and statistics that it collects and ships them to the specified output, such as Elasticsearch or Logstash.It can collect and ship system metrics, such as CPU usage, memory statistics, disk I/O, and network traffic [33].

Filebeat - Filebeat is a lightweight shipper for forwarding and centralizing log data. Installed as an agent on servers, Filebeat monitors the log files or specified locations, collects log events, and forwards them either to Elasticsearch or Logstash for indexing [34].

ElastAlert2 - ElastAlert2 is a simple framework for alerting on anomalies, spikes, or other patterns of interest from data in Elasticsearch and OpenSearch [35].

4.4 Solution

This monitoring solution uses ELK Stack (Elasticsearch, Logstash, and Kibana) to manage, analyze, and visualize logs. This also comes by the integration of Lightweight Beats agents, which specialize in efficiently gathering and transferring data from a wide range of sources directly to Elasticsearch or Logstash. Furthermore, Jira allows ticketing problems tracking, whereas Jenkins automates and streamlines continuous integration and continuous delivery (CI/CD) operations.

Beats, as a critical component, actively monitors systems to identify potential problems. Beats modules like Filebeat, Metricbeat, Packetbeat, Heartbeat, Auditbeat, and Winlogbeat are specially tuned for different use situations, ensuring a wide approach to system tracking. Each module focuses on a distinct statistic or system, all of which contribute to a robust monitoring infrastructure.

These metrics are then routed to Elasticsearch or Logstash, which serves as a central hub for ingesting data from a variety of sources. Logstash processes and filters data before delivering it to the appropriate output, and it can accept a variety of input sources. Elasticsearch's architecture is purposely built to store, search, and analyze massive amounts of data in near real-time. The subsequent transmission of data to Kibana, which has a user-friendly interface, improves the system's visualization and analytical capabilities for the user.

The solution, however, includes concerns. The combination of various technologies may add complexity to setup and configuration, requiring ample documentation and learning. Furthermore, Elasticsearch's resource requirements, particularly as data volumes increase, require careful consideration to ensure optimal performance.

Ongoing maintenance, including updates and configuration adjustments, must be managed to prevent disruptions in monitoring services. Also, challenges might appear when integrating Jira, requiring attention to communication and data flow between these components.

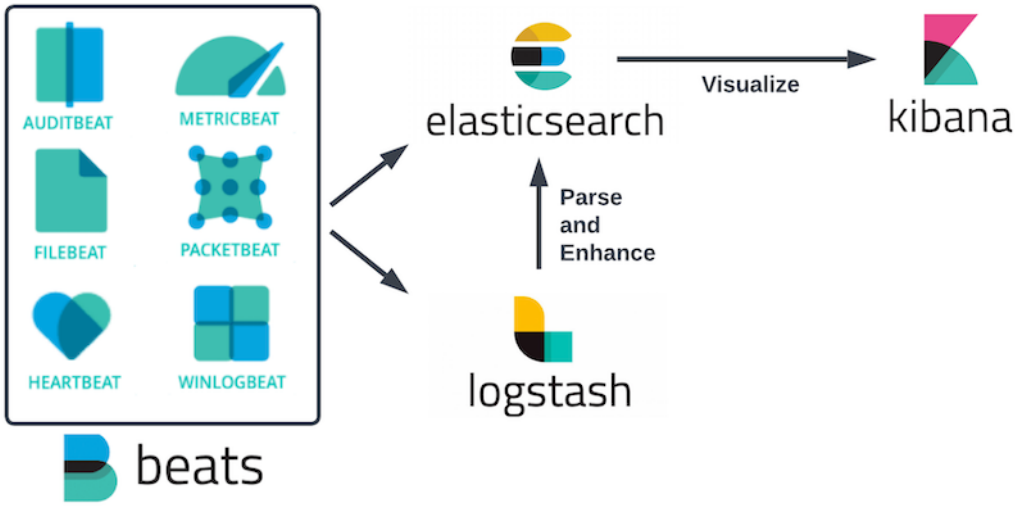

Figure 4.2: ELK Workflow
[31]

The figure 4.2 demonstrates in a generic way how the ELK Stack operates and the flow of the information from the tracked sources where the beats are installed to the elk.

Using a free license presented some challenges, particularly with certain features that were difficult to implement and utilize. A key example is the alerting system, which is a primary requirement for this project. This challenge led to further research, leading to the discovery of ElastAlert2 as a solution. ElastAlert2 is an open-source tool designed for the ELK Stack, offering users the ability to set up alerts based on patterns identified in Elasticsearch data.

By monitoring data indexed in Elasticsearch continuously, ElastAlert2 can trigger alerts when certain conditions are met, such as unusual traffic spikes, error rates, or deviations in system metrics from expected norms. These alerts can be configured to notify the team via various channels, such as email, Slack, and Jira, ensuring attention to issues.

This flexibility of ElastAlert2 in creating alert rules allows for highly customized notifications that go in accordance with the organization's needs. This customization ensures that critical events are promptly flagged and that incidents are responded to effectively and efficiently. Integrating ElastAlert2 into the existing monitoring framework enhances the overall solution by not just visualizing data but actively guarding against potential issues through real-time notifications.

Regarding this, with this implementation, the company development structure described in figure 4.1 will change with the addition of a new component.

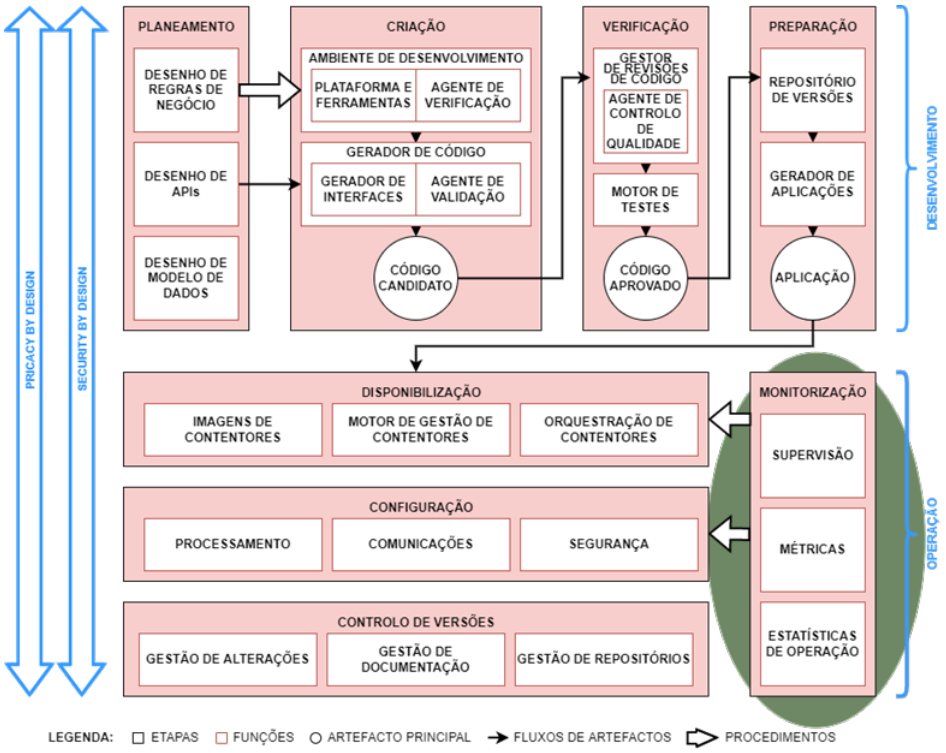

Figure 4.3: New Architecture
[16]

Figure 4.3 represents the current TrustSystems pipeline with the addition of a new monitoring function. This modification provides ongoing insight into the system's operations, helping to identify potential issues early, track performance more effectively, and support better decision-making across the software delivery process.

4.5 Solution Structure

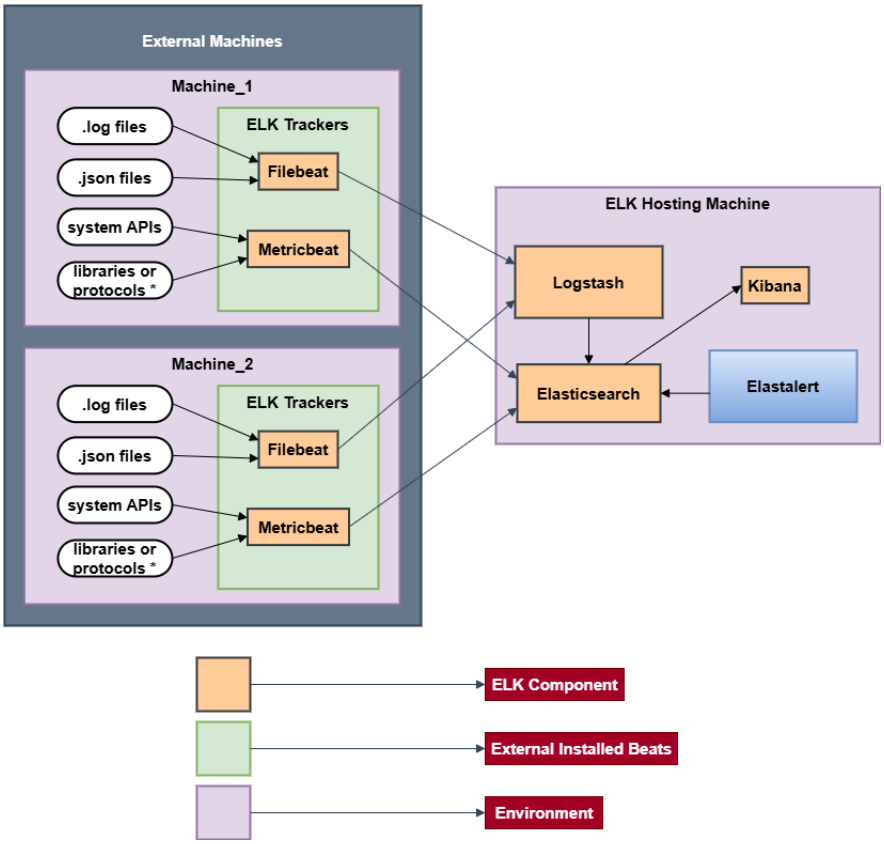

Figure 4.4: Solution Structure

** libraries or protocols are not so common as system APIs for tracking metrics, it depends on what need to be tracked.*

The figure 4.4 illustrates the overall architecture and interaction between the external machines being monitored and the server running the ELK stack.

The setup is divided into two primary domains:

1. **ELK Stack Domain:** The environment where the ELK stack is deployed and operational.
2. **External Domain:** Consists of all external machines that require monitoring for various data types, such as metrics and logs.

Within the ELK stack's environment, communication between its components occurs over localhost. Since this is a secure, internal environment, each component can directly interact with others without additional security layers.

In contrast, communication between the external machines and the ELK server is handled via Traefik, a reverse proxy. Traefik not only facilitates the connection but also secures it by masking the DNS and port details from external visibility.

The external machines utilize Beats, lightweight data shippers installed and configured through Docker. These Beats are set up to monitor and capture logs and metrics generated by the containers on their respective hosts, forwarding the collected data to the ELK stack for centralized analysis and monitoring.

Chapter 5

Solution Implementation

5.1 Implementation Details

5.1.1 Specifications and Restrictions

When developing this solution, it's essential to evaluate multiple critical aspects within the company's existing environments, processes, and technologies that are already operational.

A key requirement for this project is that everything should be done using Docker and be containerized. This ensures that the solution runs consistently across different environments, whether on a developer's machine, in testing, or in production. These Docker containers help with deployment and make it easier to manage dependencies without conflicts.

In addition, another important requirement is the use of open-source technologies wherever possible. This aligns with the company's strategy of reducing licensing costs and benefiting from community-driven innovation. Open-source tools and frameworks also provide greater flexibility and can be customized to fit the specific needs of the project. By using open-source solutions, the company can avoid vendor lock-in and stay adaptable to future changes.

Using Docker and open-source technologies not only supports the company's goal of modernizing its systems and moving toward cloud-based solutions but also ensures that the project remains cost-effective and sustainable in the long term. These requirements are mandatory and will significantly influence the design and development processes throughout the project.

5.1.2 Elk Deployed Machine

The very first step consists in searching the web for elk-stack docker images and repositories that could possibly be a good choice to be the starting point for the development. After some research, the best candidate came from a GitHub Repository [36].

In this repository, the initial setup is already defined (ports,hosts, etc.) and for its usage, the steps needed to run it already are just cloning the project, running the setup service from the compose to define the users and groups, and then running all the compose services to start the stack.

Once the configuration was in place, the Docker container was launched as this step was crucial for accessing and verifying Elasticsearch locally. The local verification ensured that the

environment was properly configured before proceeding with the integration of other components.

This compose includes already logstash, kibana and elasticsearch where Logstash was configured to handle the processing and forwarding of logs.

Meanwhile, Kibana was set up to provide a user-friendly interface for querying and visualizing the data stored in Elasticsearch.

The machine provided for this setup was named ”dev26”, and it was where the stack was deployed during the entire project duration.

Regarding the persistence of the information, it was essential to ensure that data generated and stored by the ELK stack remained intact across container restarts or system reboots. By default, Docker containers do not retain data once stopped (stateless), which posed a risk to the continuity and reliability of the project. To mitigate this, persistent storage solutions were implemented.

In this setup, Elasticsearch was configured to use dedicated volumes to persist its indices, metadata, and configuration files. This ensured that all ingested logs and search data were safely stored, allowing Elasticsearch to retain its state even if the container was stopped or removed. The persistent storage guaranteed that no data was lost, providing a reliable and stable backend for the project.

Similarly, Logstash also uses volumes to store its pipeline configurations and any necessary state information. This allowed it to resume log processing without reconfiguration, ensuring.

Kibana also leveraged volumes to retain dashboard settings, visualization configurations, and user preferences, ensuring that the interface remained consistent and accessible after restarts.

By using Docker volumes, the entire stack deployed on the ”dev26” machine maintained a high level of data persistence and stability. This approach provided a resilient environment, capable of handling interruptions or updates without risking data loss, which was crucial for the continuity and success of the project.

5.1.3 Integration with External Machines

The next procedure was to extend the system’s capabilities by incorporating Filebeat and Metricbeat for tracking and monitoring external machines and ingesting data from outside the elk machine. Both services have their own docker-compose file and their own configuration file.

In the solution architecture, external machines are used for development and play a critical role in the overall operation of providing information. These machines are not part of the ELK cluster but are essential for data ingestion, processing, and monitoring tasks. They are configured to interact with the ELK cluster, sending data to Elasticsearch for storage and analysis.

For security purposes, as explained before, all the information from outside the elk machine that is sent is through traefik, using a specific predefined DNS both for metricbeat and filebeat.

In a first attempt, the metrics and logs from the working station where the development was being made have been used to test, and all the information was sent directly to Elasticsearch with no need for Logstash. Both for metrics and logs, all the information was arriving at the elk machine and can be seen in the corresponding dashboards and Elasticsearch indexes.

The following figure illustrates how the ELK stack components work together in the context of log processing. On the left, an external machine hosts an application running two microservices, which users access via an API. Both microservices, assuming they are containerized using Docker, generate logs during their operation. These logs are collected by Filebeat, which communicates with Traefik to transmit the data.

The main difference regarding metrics is that the connection from traefik in between different machines is made directly to elasticsearch instead of logstash since there's no need to filter or rearrange the incoming metrics information.

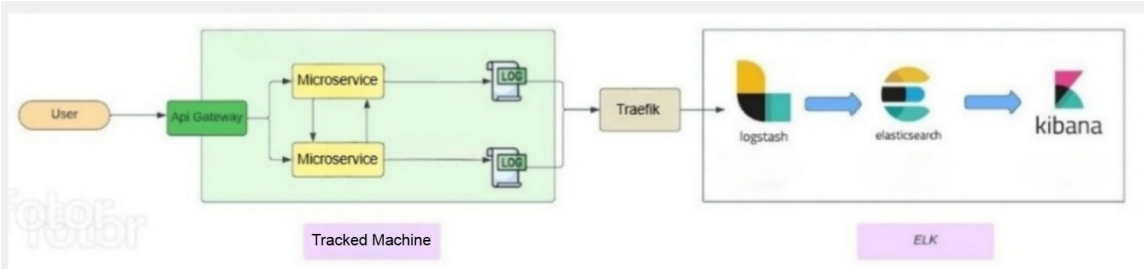

Figure 5.1: External Machine logs being tracked

The first major problem came when "dev11", which is a development machine from TrustSystems, was set to send logs and metrics to the cluster. Here for metrics everything was fine, but for logs the information that arrived at the cluster was encoded, broken, and unreadable information.

After some research the first problem was that filebeat was reading input logs with input type "log" that has already been deprecated and was not detecting everything that should. To address this issue the solution that was present at the documentation was to change the input type to "filestream" which is a new improved version.

In the other hand, the problem with the encoded and broken logs was yet happening, and after some research through forums and documentation, the issue was that the logs that were being generated from "dev11" were prepared to be stored in an AWS service used by TrustSystems, so according to that, all the generated logs were matching their encoding and file log characters, and that came to be the main issue.

To address this problem, the use of logstash was needed, and at this point, the filebeat no longer can send the information directly to elasticsearch to be directly indexed but rather to logstash. At the logstash, many filters have been implemented to address this issue such as removing white spaces, refactoring characters, and concatenating strings to form proper logs.

After all these steps, the new logs were then sent from Logstash to Elasticsearch to be indexed, and the information was now correct to be seen and analyzed.

This process can be seen in figure 5.6 where first can be seen a log with many encoded values and then after the logstash filters are applied, the new log with the relevant information.

These docker-compose.yml files were then deployed on external machines that needed to be

tracked. By doing so, both Filebeat and Metricbeat were able to capture and forward relevant data to the elk machine. To manage all external traffic and connections to the elk machine, there´s the need to use Traefik as a reverse proxy. Traefik was configured to route requests to the appropriate services within the elk machine, ensuring secure and efficient communication. This setup allows to handle incoming connections from Filebeat, Metricbeat installed outside the elk network as already explained.

5.1.4 Scaling, Security, and Performance Considerations

With the core components integrated and operational, further steps were taken to address scaling, security, and performance concerns. As the system expanded and more machines were connected to the cluster, the volume of data indexed daily grew exponentially. In the early stages, this rapid data accumulation led to memory saturation issues, posing a significant challenge to system stability.

To mitigate these issues, it became imperative to implement strategies for efficient data management and resource allocation. A solution to support horizontal scaling was then necessary, ensuring that additional resources could be integrated into the cluster to accommodate increasing workloads. This not only improved memory handling but also optimized overall system performance.

This solution consists of defining a policy that, after a defined value of days, closes the current index where the information is being written to, and then creates a new one to write to.

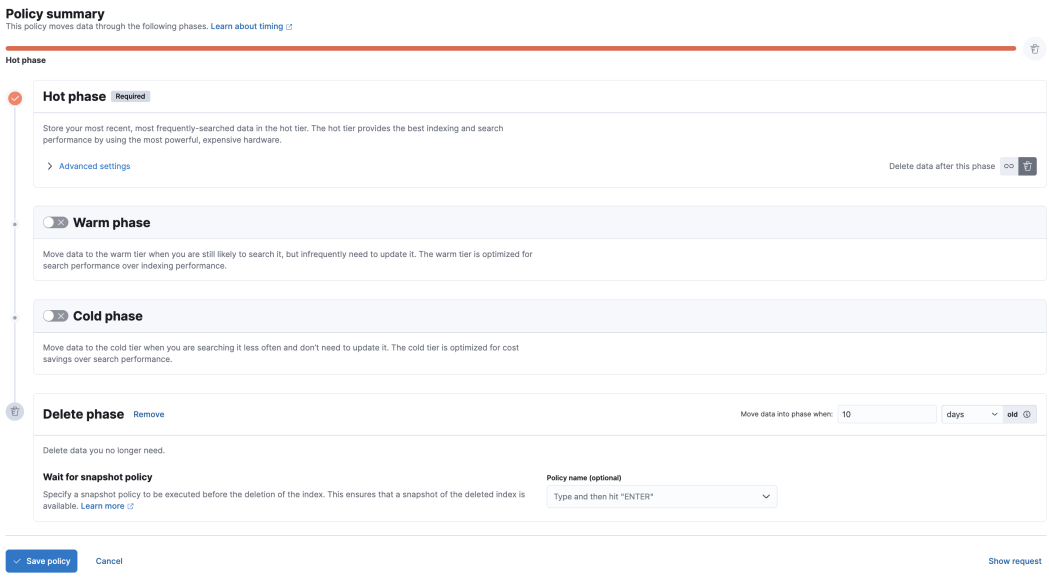

Figure 5.2: Policy Creation

The figure 5.2 shows in detail how this policy was created; first, it starts with a hot phase where the information that is accessed frequently is; this phase is mandatory, there´s also the possibility

to use the other phases (warm/cold) where data that is not frequently accessed is sent for, or it can go to delete phase if pretended. On the figure 5.2 it is defined just the hot and delete phases.

For this project, the policies have been created using the included devTools of the stack. This approach involved creating a roll-up after 5 days counting since the creation of the index, and after a fixed period of 10 days, it automatically deletes the index with its data.

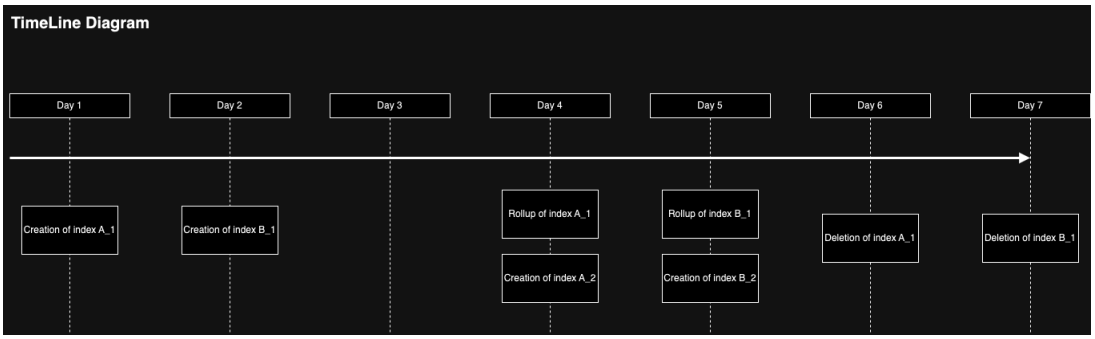

Figure 5.3: Index Management Timeline

The figure 5.3 explains this in detail in a smaller example, in day 1 there´s the creation of an index called A_1. On the second day, the same process happens at the second day but for B_1. On the following days after the creation, the received data on the cluster will be written to those indexes when the age of 3 days is reached that index is rolled up and closed and no more information is written there happening in day 4 for A_1 and day 5 for B_1. When this roll-up happens then a new index is created also and the information is written to that new place.

In general, this approach makes the information more organized in smaller indexes and easier to retrieve and analyze.

After this, the next step is to create snapshots of the information; this ensures that our older logs and metrics are preserved for more time as we want.

For this, we first create a repository; this serves as a storage location where snapshots of the cluster's data are saved. These snapshots are essential for backing up data and enable restoring it when necessary.

There are several types of repositories that Elasticsearch supports, each designed with different storage needs but as requested by the client the used was **Filesystem Repository**, where this type of repository stores snapshots on a shared file system. It's particularly useful for local backups. The shared file system can be anything from a local disk to a more complex network-attached storage (NAS) system.

To use a repository in Elasticsearch, it must first be registered with the cluster. This is done using a PUT API call using DevTools or using the interface through Kibana, where the type of repository is specified and the necessary configuration details are provided. Once the repository is registered, it is now possible to create snapshots of the data. These snapshots capture the state of

the indices at a specific point in time and can be used to restore the cluster to that state in case of data loss or corruption.

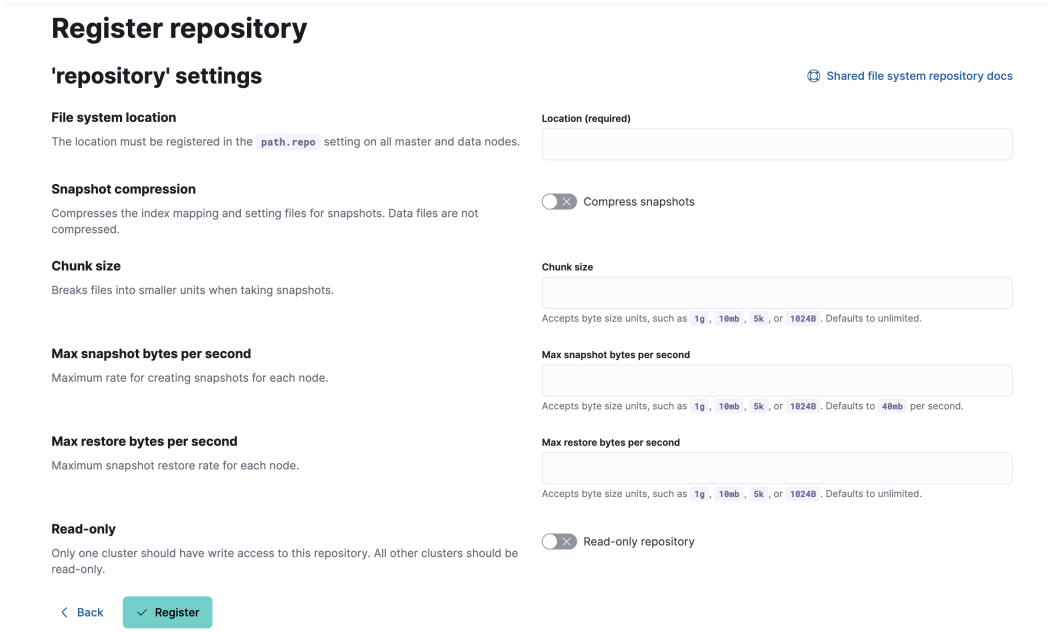

Figure 5.4: Repository Creation

Saving data to a repository offers several benefits such as reducing disk space usage, providing a backup for disaster recovery, and improving query performance by keeping active indices smaller. Additionally, repositories enable long-term data retention, easy data restoration, and efficient data sharing or migration between clusters.

5.2 Operational Results

The following section presents operational results obtained during the implementation of the Elasticsearch and ELK stack during this project. The insights include examples of log messages, system performance metrics, and workflow automation that demonstrate how effectively the stack handled the data.

The following image presents a print from a dashboard displaying CPU usage metrics. The visualization provides a clear view of CPU load trends over time. This allows for easy monitoring and quick identification of resource bottlenecks.

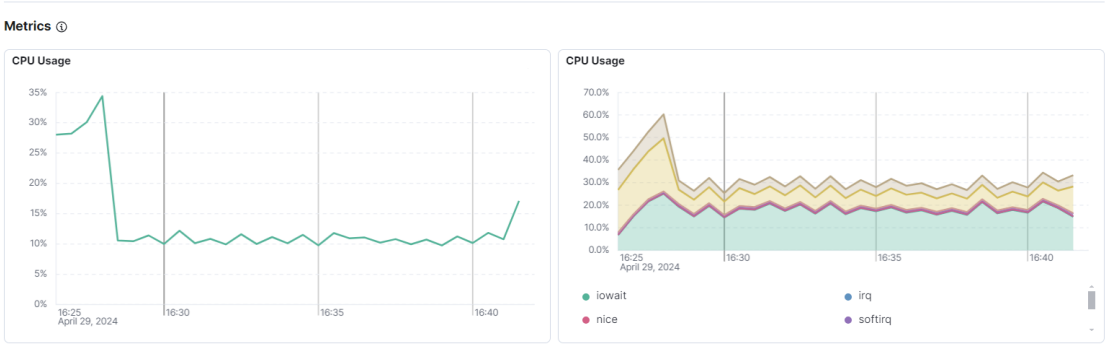

Figure 5.5: CPU metrics over time

The figure 5.6 demonstrates how Logstash was used to filter incoming logs by removing unnecessary characters and hidden ones. This allows for cleaning and transforming log data before indexing in Elasticsearch, ensuring that only relevant and properly structured data is stored, optimizing both storage space and search efficiency.

Figure 5.6: Logstash filtering image

Following, we see the machine metrics collected during the operation of the ELK stack. Monitoring these metrics was crucial for assessing system health and ensuring optimal performance. It helps in identifying when the system was under heavy load or experiencing issues like memory leaks or disk space shortages.

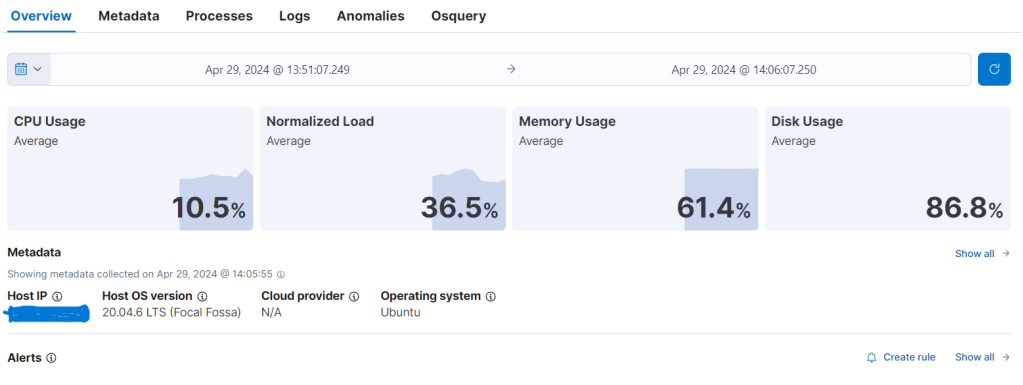

Figure 5.7: Host machine metrics

Now, regarding the alerts topic, the following result shows an example of a log message detected by ElastAlert2. This alert was triggered based on a predefined rule that continuously monitors incoming log data for specific patterns. The image presents a specifically defined timeframe where the predefined value of 0.03, which corresponds to 3% of total CPU, is surpassed 6 times and an email is sent to the respective email indicated at where the rule is defined.

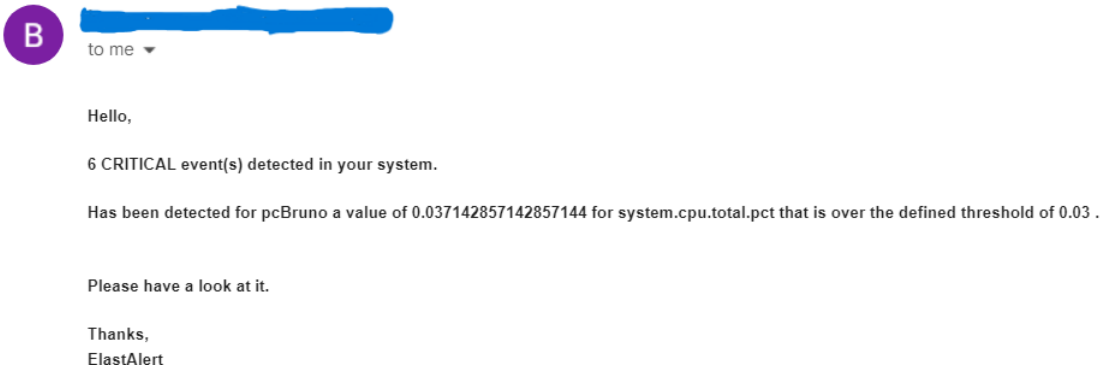

Figure 5.8: ElastAlert2 message

Finally, the last image shows a Jira issue automatically created following a triggered alert by elastalert. This demonstrates how the ELK stack can be integrated with other tools to automate workflows, improving incident management. Alerts raised by ElastAlert2 were configured to trigger Jira issues, making it easier to track and resolve incidents in a timely manner.

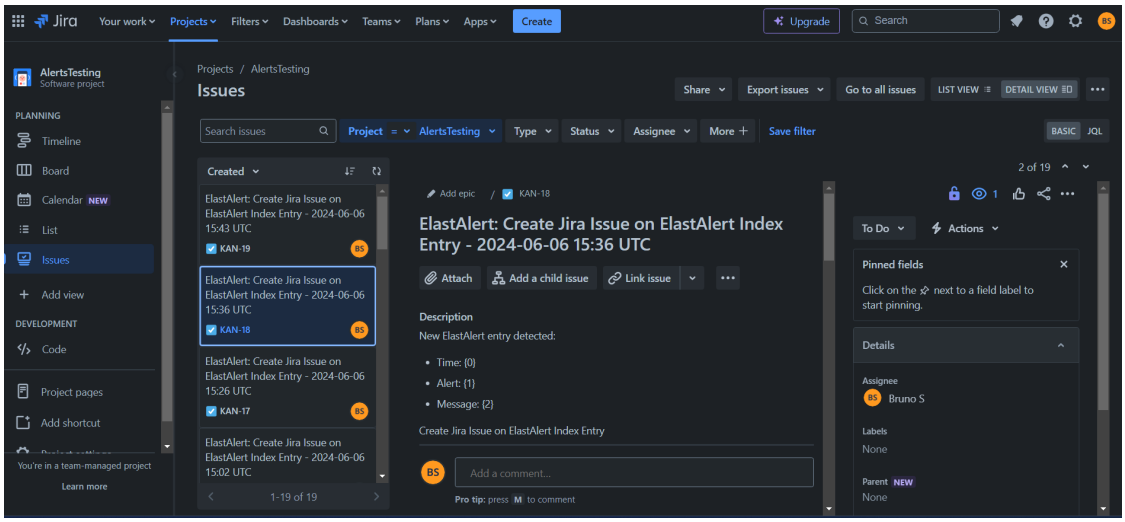

Figure 5.9: Created Jira issues

5.3 Client Feedback

In collaboration with TrustSystems, feedback was gathered to evaluate the monitoring implementation using the ELK stack. This feedback provided an operational perspective on the system's performance, usability, and its impact on both internal workflows and end-user outcomes. The feedback was crucial in identifying areas for improvement and further refinement, as well as validating the effectiveness of the overall solution.

Additionally, in the final chapter, I address some key questions answered by the company, focusing on the main points of the app and its practical application. These responses helped to understand the app's strengths and potential areas for enhancement in real-world scenarios.

5.3.1 Overall Effectiveness

Score: 4/5

From an operational perspective, the implementation has significantly improved team management. The development team no longer has to spend excessive time tracking down issues, as the triggers notify them immediately when something occurs. This has also contributed to cost savings, as we no longer rely on paid tools. With faster response times, the impact on customers is noticeable, as solutions are delivered with higher quality.

5.3.2 Filebeat & Metricbeat

Score: 5/5

One of the challenges in configuring the beats was fine-tuning the intervals at which they send data to the server. A shorter interval negatively impacted the client machine's performance. However, installing the beats on a new machine is a very easy process, as it's just a matter of running a pre-configured Docker Compose.

5.3.3 ElastAlert2

Score: 3.5/5

If the alerts are basic and predefined by ELK, they are relatively easy to manage. However, if more complex alerts are required, it becomes an entirely different learning curve, making the process difficult for new users. The level of alert detail also varies, depending on what we want.

5.3.4 Impact on DevOps Workflow

Score: 4/5

It adds complexity when needed but, overall, it doesn't add any unnecessary complexity. It's efficient and operates in parallel with existing workflows.

5.3.5 User Experience

Score: 3.5/5

The user experience depends heavily on the training received. It's not a simple tool to start with, but the more trained the user is, the more interactive and usable it becomes. For someone without training, it's more complex, but since this is a tool designed for developers, it's expected that the user should have some basic understanding beforehand.

5.3.6 Security & Compliance

This aspect does not apply since the level of security is the same as before. Data can potentially be intercepted in transit, but they are protected by HTTPS, as before.

5.3.7 Additional Feedback

- More training for users would be beneficial.
- The dashboard creation is not so intuitive and requires training, for someone who needs to create a dashboard from scratch, it's not intuitive at all.
- Integration with a cloud provider, such as AWS, to send data to the same place.
- Enhanced log understanding, including not just identifying errors but the type of error for better filtering.
- The ability to act, not just notify. For example, being able to restart a Docker container when it crashes, etc. (business functionality).

5.3.8 Post-Delivery Feedback

After the delivery of the project and the source code of the monitoring solution, no additional feedback was received from the TrustSystems regarding the current usage or adoption of the product to the current workflow of the company. While initial feedback during the development and testing phases was constructive and indicated alignment with the company's requirements, after delivery, the absence of follow-up communication makes it difficult to evaluate the solution's long-term impact on the company operations.

5.4 Encountered Issues

During the implementation of the ELK Stack into TrustSystems' DevOps pipeline, several challenges arose that required careful attention and troubleshooting to ensure the system worked as expected. Integrating tools like Filebeat, Metricbeat, and ElastAlert2 into the pipeline presented some challenges that had to be resolved to achieve a smooth and functional monitoring setup.

One of the first major challenges was configuring Filebeat and Metricbeat to collect and forward logs and metrics from various environments consistently. The diversity of services and infrastructure components meant that different log formats and data types were being generated, making it difficult to standardize the data ingestion process. To address this, multiple iterations of

configuration tuning were required. For example, we had to carefully define the input and output configurations for Filebeat to ensure that the correct logs were captured and forwarded to Elasticsearch without missing any critical information. Similarly, adjusting the modules in Metricbeat to accurately capture the most relevant performance metrics from different systems.

Another issue was the big volume of log and metric data being collected, especially in development environments. As the data load increased, Elasticsearch indexing became slower, affecting search performance and dashboard responsiveness in Kibana. This required a repository solution to be able to store everything. I had to adjust index patterns and implement data retention policies to avoid overloading the Elasticsearch cluster.

Integrating ElastAlert2 for real-time alerting presented its own challenges. One problem was setting the right alert thresholds. For instance, system alerts were being triggered for minor, short-lived fluctuations in performance that didn't actually indicate a problem. This led to unnecessary noise and alert fatigue. To resolve this, many rules have been tested, adjusting the thresholds and incorporating additional logic to better filter out non-critical events.

5.5 Overview

The system demonstrated its ability to effectively handle core use cases, delivering the necessary functionality and insights to meet operational needs. This highlighted the success of deploying the ELK stack in addressing the organization's primary objectives. While some minor adjustments could have enhanced the user experience, the timing of feedback did not allow for significant changes within the project's scope.

Despite this, the solution proved to be stable and functional, meeting the immediate requirements for monitoring and logging. The deployment of the ELK stack provided the team with tools to gain valuable insights, improve system visibility, and resolve issues efficiently.

At the time of writing, the project is still undergoing testing and has not been moved to production. Continued evaluations will ensure the system performs as expected once deployed.

Chapter 6

Conclusion

This chapter begins by reviewing the work carried out during the course of the project, followed by a discussion of the most significant challenges and obstacles encountered. Finally, it offers recommendations for future improvements that could enhance the system and the processes developed as part of this effort.

The primary goal of this project was to integrate and implement a new stage of automated monitoring into TrustSystems DevOps pipeline. This monitoring phase was designed using the ELK Stack (Elasticsearch, Logstash, and Kibana) to improve real-time system and application performance monitoring. The monitoring system was structured into two parts: the first focused on gathering and analyzing system and application logs to ensure performance stability, and the second aimed at providing real-time visualization and alerting, enabling proactive issue detection and resolution.

While the integration of these tools presented some challenges, particularly around the configuration and data management, the outcomes have been highly beneficial. The automated alerts and detailed dashboards have reduced manual monitoring efforts and helped prevent downtime, contributing to smoother operations across the board.

Moving forward, there is still room to expand and refine the system, especially by incorporating more advanced alerting mechanisms and broader monitoring metrics. However, the foundation laid by this project has already brought noticeable improvements to TrustSystems ability to manage its infrastructure and software with greater reliability.

In conclusion, this project has not only enhanced the monitoring capabilities within TrustSystems but has also demonstrated the value of automating key aspects of system management in a DevOps environment. The lessons learned and the tools implemented here will continue to support the company's ongoing efforts to optimize its software delivery process and ensure high performance.

6.1 Future Work

Although the integration of the ELK Stack has been a success, there are still several areas where improvements can be made to further enhance monitoring and performance tracking within Trust-

Systems DevOps pipeline.

One of the recommendations is to fully configure the APM (Application Performance Monitoring) server, which has been implemented but not yet set up. Once configured, the APM server will provide in-depth insights into application performance, such as transaction times, error rates, and slow queries. This enables the team to identify and resolve performance bottlenecks at the application level. The addition of APM will give the team a more complete picture of how applications behave in real-time.

Additionally, although Metricbeat and Filebeat are already being used to collect metrics and log data from systems, there's potential to expand the use of other Beats agents to gather even more detailed data. Agents like Auditbeat, which monitors system audit data, and Packetbeat, which tracks network traffic, could provide additional layers of insight into system security and network performance. By incorporating these Beats into the existing monitoring setup, TrustSystems could gain more granular visibility into potential security threats and network-related issues, further enhancing its ability to respond to incidents proactively.

Additionally, integrating the monitoring setup with a cloud provider such as AWS would allow for centralized data management, ensuring that logs, metrics, and other operational data from various environments can be sent to the same location.

Another potential enhancement is improving the log analysis process to not only identify errors but also classify the type of error for more efficient filtering and troubleshooting.

Finally, there is an opportunity to extend the system's functionality beyond notifications by implementing automated responses to specific events. For instance, instead of merely sending alerts when a Docker container crashes, the system could be configured to automatically attempt a restart or execute predefined recovery actions. This would reduce manual intervention, but also minimize downtime and improve overall system resilience, adding significant value to the business by keeping critical services operational.

Chapter 7

Attachments

Feedback on Monitoring Implementation in DevOps Pipeline

1. Overall Effectiveness
- How effective is the new monitoring stage in improving pipeline performance? (1-5 scale)
- What are the most noticeable benefits of this implementation?

2. ELK Stack (Elasticsearch, Logstash, Kibana)
- How user-friendly and efficient is the ELK Stack for querying and visualizing data? (1-5 scale)
- Any issues with Elasticsearch performance or Kibana dashboards?

3. Filebeat & Metricbeat
- Are Filebeat and Metricbeat collecting and delivering data effectively? (1-5 scale)
- Any challenges with configuration or reliability?

4. ElastAlert2
- How accurate and timely are the alerts generated by ElastAlert2? (1-5 scale)
- Any difficulties with configuring or understanding the alerts?

5. Impact on DevOps Workflow
- Has the new monitoring stage accelerated issue detection and resolution? (1-5 scale)
- Any impact on pipeline efficiency or added complexity?

6. User Experience
- How satisfied are you with the overall user experience of these tools? (1-5 scale)
- Any recommendations for improving the user interface or usability?

7. Security & Compliance
- Any concerns regarding data security or privacy?

8. Additional Feedback
- Any other comments or suggestions for further improvements?

Figure 7.1: Questionary

Bibliography

[1] Aroral, H. K. (2021). Waterfall Process Operations in the Fast-paced World: Project Management Exploratory Analysis. International Journal of Applied Business and Management Studies, 6(1), 91-99.

[2] Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. Ieee Software, 33(3), 94-100.

[3] Singh, V. (2022). Developing a CI/CD pipeline with GitLab.

[4] https://about.gitlab.com/topics/ci-cd/

[5] https://kylefarmer85.medium.com/what-is-ci-cd-76d71b82898d

[6] https://www.crowdstrike.com/cybersecurity-101/observability/devops-monitoring/

[7] https://grafana.com/grafana/dashboards/13023-zabbix-server-dashboard/

[8] https://www.splunk.com/en_us/blog/learn/devops-monitoring.html

[9] https://coralogix.com/blog/easily-build-jenkins-pipelines-tutorial-with-parameters/

[10] Wahaballa, A., Wahballa, O., Abdellatief, M., Xiong, H., & Qin, Z. (2015, September). Toward unified DevOps model. In 2015 6th IEEE international conference on software engineering and service science (ICSESS) (pp. 211-214). IEEE.

[11] Török, M., & Pataki, N. (2020). DevOps Dashboard with Heatmap. In ICAI (pp. 400-407).

[12] Ian Buchanan. History of devops https://www.atlassian.com/ devops/what-is-devops/history-of-devops.

[13] https://aws.amazon.com/pt/devops/what-is-devops/

[14] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering: A roadmap and agenda. 2015.

[15] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. Site Reliability Engineering: How Google Runs Production Systems. O'Reilly Media, Inc., 1st edition, 2016.

[16] Artiom, A. (2023). DevOps: Implementação do Modelo Devops em Equipa de Desenvolvimento [M.Sc. Thesis, FCUL].

[17] Ragunath, P., Velmourougan, S., Davachelvan, P., Kayalvizhi, S., & Ravimohan, R. (2010). Evolving A New Model (SDLC Model-2010) For Software Development Life Cycle (SDLC). International Journal of Computer Science and Network Security, 10(1), 112-119.

[18] Stoica, M., Mircea, M., & Ghilic-Micu, B. (2013). Software Development: Agile vs. Traditional. Informatica Economica, 17(4), 64-76.

[19] Eason, O. K. (2016). Information systems development methodologies transitions: An analysis of waterfall to agile methodology.

[20] Highsmith J., Orr K., Cockburn A., "Extreme programming", in: E-Business Application Delivery, Feb. 2000, pp. 4–17. Available: http://www.cutter.com/freestuff/ead0002.pdf.

[21] Cohen, D., Lindvall, M., & Costa, P. (2004). An introduction to agile methods. Adv. Comput., 62(03), 1-66.

[22] https://www.atlassian.com/devops/what-is-devops/agile-vs-devops

[23] Cohen, D., Lindvall, M., & Costa, P. (2003). Agile software development. Dacs Soar Report, 11, 2003.

[24] https://scrumguides.org/scrum-guide.html

[25] Dikert, K., Paasivaara, M., & Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations: A systematic literature review. Journal of Systems and Software, 119, 87-108.

[26] https://www.elastic.co/pt/elastic-stack/features

[27] https://aws.amazon.com/what-is/elk-stack/

[28] https://www.atlassian.com/software/jira/guides/getting-started/introduction#what-is-jira-software

[29] https://www.researchgate.net/publication/272506769_Measuring_and_Understanding_the_Effectiveness_of_JIRA

[30] https://www.elastic.co/beats

[31] https://lalit-soni.medium.com/what-is-elk-stack-elasticsearch-logstash-and-kibana-6b26b5b1a79c

[32] https://www.tmap.net/building-blocks/CICD-pipelines

[33] https://www.elastic.co/guide/en/beats/metricbeat/current/metricbeat-overview.html

[34] https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html

[35] https://elastalert2.readthedocs.io/en/latest/elastalert.html#overview

[36] https://github.com/deviantony/docker-elk

[37] Manifesto, A. (2001). Manifesto for agile software development.

[38] Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations.

[39] Forsgren, N., Humble, J., & Kim, G. (2018). Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations.

[40] Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (2016). Site Reliability Engineering: How Google Runs Production Systems. O'Reilly Media.

[41] Volz, B., & Davis, T. (2019). Prometheus: Up & Running.

[42] CNCF Observability Working Group. (2020). CNCF Observability White Paper.

[43] Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.

[44] Kim, G., Behr, K., & Spafford, G. (2014). The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win.

[45] Bartholomew, M. (2017). Practical Monitoring: Effective Strategies for the Real World.

[46] Garousi, V., Kähkönen, T., & Felderer, M. (2022). Continuous Testing for DevOps Professionals: A Practical Guide to Testing in DevOps.

[47] Wagner, S. (2018). Measuring Continuous Delivery: A Practical Guide for IT Leaders.

[48] Antunes, P., and Tate, M. 2024. "'What's Going On' with BizDevOps: A Qualitative Review of BizDevOps Practice." Computers in Industry (157–158: 104081).

[49] https://learn.microsoft.com/en-us/azure/azure-monitor/app/usage?tabs=aspnetcore