Final Year Graduation Project

To obtain the State Engineering Diploma

Major: Innovation & AMOA

Green IT Integration in DevOps

Authored by :

Mr. Imad EL BOUHATI

Defended on June 26, 2024, before the jury composed of :

Mr. Nawfal ACHA	INPT	- Examiner
Mr. El Mehdi KANDOUSSI	INPT	- Examiner
Mrs. Charifa HANIN	INPT	- Supervisor
Mr. Amine AJA	Orange Business	- Supervisor

Agence Nationale de Réglementation des Télécommunications

Institut National des Postes et Télécommunications

Class of : 2023/2024

Dedicated to

I humbly dedicate this work to my dear mother, who gave me life, love, courage, and a reason to live. I also wish to express my gratitude to my father for his support and sacrifices.

A big thank you to my two dear sisters, Alae and Samia, whom I love so much, to all my friends with whom I have shared moments of joy and happiness, as well as to my entire extended family and to all those who love me.

Imad

Acknowledgments

First and foremost, I would like to express my gratitude to Allah the Almighty for giving me the strength and patience necessary to complete this project.

I would like to extend my sincerest thanks to my supervisor, Mrs. Charifa HANIN, for her competent assistance, patience, and encouragement. Her critical feedback has been invaluable in structuring my work and improving the quality of its various sections.

I also want to express my appreciation to my mentor, Mr. AJA Amine, for his immense support, the quality of his guidance, and for all the advice and information he provided with unmatched professionalism and patience.

A big thank you is also owed to Mr. MOUSTACHI Mouhsine, for offering me the opportunity to join his team and for his support.

I would also like to thank Mr. EDDAHBY Mohamed Amine, Mr. ECHARJI Abderrahman, Mr. OUCHEN Othman, Mr. EL ALJ Mohamed Taieb, Mr. EL IDRISSI Hicham, Ms. EL IBRAHIMI Dounia, Mr. OUADHDHAFE Marouane, and all the engineers in the DevOps team for their invaluable assistance, encouragement, and for making my internship at Orange Business a very enriching experience.

I wish to express my sincere gratitude to the members of the jury for the honor they bestow upon me by taking the time to read and evaluate my work.

I would also like to thank the pedagogical and administrative team at INPT for their dedication and contribution to our excellent education.

Finally, I would like to thank all the individuals who have contributed, directly or indirectly, to the completion of this work.

Résumé

Ce rapport reflète le travail réalisé chez Orange Business Maroc dans le cadre de mon projet de fin d'études pour le diplôme d'ingénieur en Télécommunications et Technologies de l'Information.

Dans un contexte où les architectures distribuées et les pratiques de développement agile dominent la conception des applications, intégrer les principes de Green IT dans les pratiques DevOps est devenu essentiel. Cette évolution témoigne de l'importance croissante accordée à la durabilité environnementale dans le secteur de la technologie.

En se concentrant sur la surveillance de la consommation d'énergie et des émissions de CO_2 de l'infrastructure, nous mettons en lumière une préoccupation croissante pour l'empreinte écologique de nos systèmes informatiques. Cette prise de conscience a conduit à l'adoption de mesures proactives telles que l'utilisation de Kepler, Prometheus et Grafana pour surveiller et visualiser ces métriques environnementales.

Mon projet de fin d'étude vise à mettre en place Kepler pour surveiller la consommation énergétique et les émissions de CO_2 de Kubernetes, et à l'intégrer dans la chaîne CI/CD.

Mots-clés : Green IT, DevOps, Kepler, Prometheus, Grafana, Kubernetes, CI/CD.

Summary

This report reflects the work carried out at Orange Business Morocco as part of my final project for the engineering degree in Telecommunications and Information Technologies.

In a context where distributed architectures and agile development practices dominate application design, integrating Green IT principles into DevOps practices has become essential. This evolution underscores the increasing importance placed on environmental sustainability in the technology sector.

By focusing on monitoring energy consumption and CO_2 emissions of the infrastructure, we highlight a growing concern for the ecological footprint of our computer systems. This awareness has led to the adoption of proactive measures such as using Kepler, Prometheus, and Grafana to monitor and visualize these environmental metrics.

My final project aims to implement Kepler to monitor energy consumption and CO_2 emissions of Kubernetes, and integrate it into the CI/CD pipeline.

Keywords: Green IT, DevOps, Kepler, Prometheus, Grafana, Kubernetes, CI/CD.

ملخص

يعكس هذا التقرير العمل الذي قمت به في أورانج بيزنس المغرب في إطار مشروعي الختامي للحصول على شهادة مهندس دولة في الاتصالات وتكنولوجيا المعلومات.

في سياق تهيمن فيه الهياكل الموزعة وممارسات التطوير السريعة على تصميم التطبيقات، أصبح دمج مبادئ تكنولوجيا الخضراء في ممارسات DevOps أمراً ضرورياً. يؤكد هذا التطور على الأهمية المتزايدة التي تحظى بها الاستدامة البيئية في قطاع التكنولوجيا.

وقد أدى هذا الوعي إلى اعتماد تدابير استباقية مثل استخدام Kepler و Prometheus و Grafana لمراقبة هذه المقاييس البيئية وتصورها.

يهدف مشروعي الختامي إلى تطبيق Kepler لمراقبة استهلاك الطاقة وانبعاثات ثاني أكسيد الكربون في Kubernetes و دمجه في سلسلة CI/CD .

الكلمات المفتاحية: Green IT, DevOps, Kepler, Prometheus, Grafana, Kubernetes, CI/CD

List of Figures

List of Tables

Contents

Acronyms

ACPI	Advanced Configuration and Power Interface
AI	Artificial Intelligence
AMOA	Assistant à Maîtrise d'OuvrAge
API	Application Programming Interface
AWS	Amazon Web Services
BM	Bare-Metal
BPF	Berkeley Packet Filter
CD	Continuous Delivery
CEO	Chief Executive Officer
CI	Continuous Integration
CNCF	Cloud Native Computing Foundation
CO_2	Carbon Dioxide
CPU	Central Processing Unit
DNS	Domain Name System
DRAM	Dynamic Random Access Memory
EC2	Elastic Compute Cloud
EIC	Enriched Interactions and Collaboration
GHG	Greenhouse Gas
GPU	Graphics Processing Unit
HTTP	Hypertext Transfer Protocol

HR	Human Resources
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KPI	Key Performance Indicator
LAN	Local Area Network
LXC	Linux Containers
LXD	Linux Containers Daemon
MSC	Major Service Centers
MTTD	Mean Time to Detect
MTTI	Mean Time to Identify
MTTR	Mean Time to Repair
NVML	NVIDIA Management Library
OBS	Orange Business Services
OCD	Orange Cyberdefense
OCWS	Orange Connectivity and Services Workspace
OINIS	Orange International Networks Infrastructures and Services
PKG	Package
RAPL	Running Average Power Limit
REST	Representational State Transfer
SCM	Source Code Management
SDWAN	Software-Defined Wide Area Network
SSH	Secure Shell
TAAS	Testing as a Service
TTM	Time to Market
URL	Uniform Resource Locator
VM	Virtual Machine
VMM	Virtual Machine Monitor
VPN	Virtual Private Network
WIFI	Wireless Fidelity
YAML	YAML Ain't Markup Language

General Introduction

Modern applications are typically designed with distributed architectures and developed using agile methodologies. These continuous integration and delivery (CI/CD) practices play a crucial role in efficiently managing the software lifecycle in terms of speed and effectiveness. However, despite this efficiency, security is often overlooked in the CI/CD workflow, particularly regarding concerns related to environmental footprint.

After deploying an application in a real production environment, additional security measures are needed to ensure its protection. The use of Green IT practices, such as integrating metrics for energy consumption and CO_2 emissions into the CI/CD pipeline, is becoming increasingly important to ensure the environmental sustainability of technological infrastructures.

My Final Year Project fits into this context, aiming to automate the monitoring of energy consumption and CO_2 emissions in DevOps infrastructures using Kepler.

This report synthesizes the work done throughout my internship period, structured into four chapters covering various aspects of the project.

The first chapter introduces the project context, objectives, and the approach and planning followed.

The second chapter focuses on the project's requirements analysis, while the third chapter addresses the technological choices.

Finally, the fourth chapter presents the project's final structure in detail, along with an overview of the achievements. The report concludes with a summary and a list of bibliographical references.

Chapter 1

General Project Context

1.1 Introduction

This chapter aims to conduct an initial exploration of the project, in order to better understand the system requirements, and to position the project within its organizational and contextual environment. It provides an overview of the hosting organization Orange Business Morocco, then reveals the general theme of the project by introducing its intentions and specifications, followed by the various approaches applied, including the recommended methodology for project management and the resulting planning.

1.2 Presentation of host organization

1.2.1 Company Overview

Figure 1.1: OBS logo

In 2000, following the opening of the telecommunications market to competition in Europe, France Télécom, the historic French operator, acquired the British brand Orange. At the same time, the Senegalese group Sonatel was created in the late 1980s, resulting from the merger of the Office des Postes et Télécommunications and TéléSenegal. Sonatel was privatized in 1997 and integrated into France Télécom's capital.

In France, France Télécom, quickly becoming a leader thanks to the quality of its networks, gradually unified all the group's subsidiaries under the name Orange. By 2009, Orange had a presence in more than 30 countries and served over 123 million customers.

On July 1, 2013, following a vote at a general meeting, France Télécom officially changed its name to Orange. This transformation marked a new chapter for the company while retaining a cultural heritage inherited from the public service. France Télécom's historic installations played an important role in the French telephone network, paving the way for the rise of mobile telephony and the beginning of a new revolution: the Internet.

Orange operates in several regions, notably in Europe, Africa, and the Caribbean. In 2019, it was considered the leader or second operator in 75

Orange Business operates in 166 countries and serves its customers in 220 countries and territories (Table 1.1).

Date of Establishment	2006
Legal Form	Legal Form
Management	Christel Heydemann (CEO, Orange S.A) Helmut Reisinger (CEO, Orange Business Services)
Headquarters	Paris, France

Parent Company	Orange
Activity	Telecommunications Information Technology Services
Subsidiaries	Orange Application for Business, Business et Decision
Website	http://www.orange-business.com

Table 1.1: Orange Business Information

1.2.2 Orange Business Morocco

Orange Business Morocco is a newly established entity in 2018 in Rabat, specializing in the design and development of application services and system integration in various domains. It works on behalf of various expertise of Orange Business Services. Several companies have provided application services for Orange Business Services, such as Almerys, a health third-party payer operator. OBS IT Morocco is founded to take control of its applications, optimize and rationalize costs, bring development teams closer to business directions, and improve IT solution delivery times for TTM. By following an Agile way of working to deliver functionalities regularly that create the most value and adopting a simple and automated development and production environment to focus on the essentials. OBS IT Morocco uses the cloud, DevSecOps tools, and automated testing and feedback utilization to enrich applications while ensuring a user-centered approach. By guaranteeing versatile, autonomous, and committed teams to manage all activities on their applications (collecting requirements, drafting user stories, development, testing, integration, deployment, support).

1.2.3 Organization and Organizational Chart

OBS Morocco, overseen by a director Mrs. Rym Sahnoun who is the highest authority, is organized into operational poles whose goal is to focus its energies on a daily basis towards customer satisfaction. Teams are organized by Orange business services. I completed my internship in the OBS IT entity managed by Mr. Aarab Abderrahim and specifically Customer Marketing Innovation (Fig. 1.6) in the DevSecOps/TAAS team attached to Mr. Moustachi Mouhcine, IT RUN Manager (see Fig. 1.7).

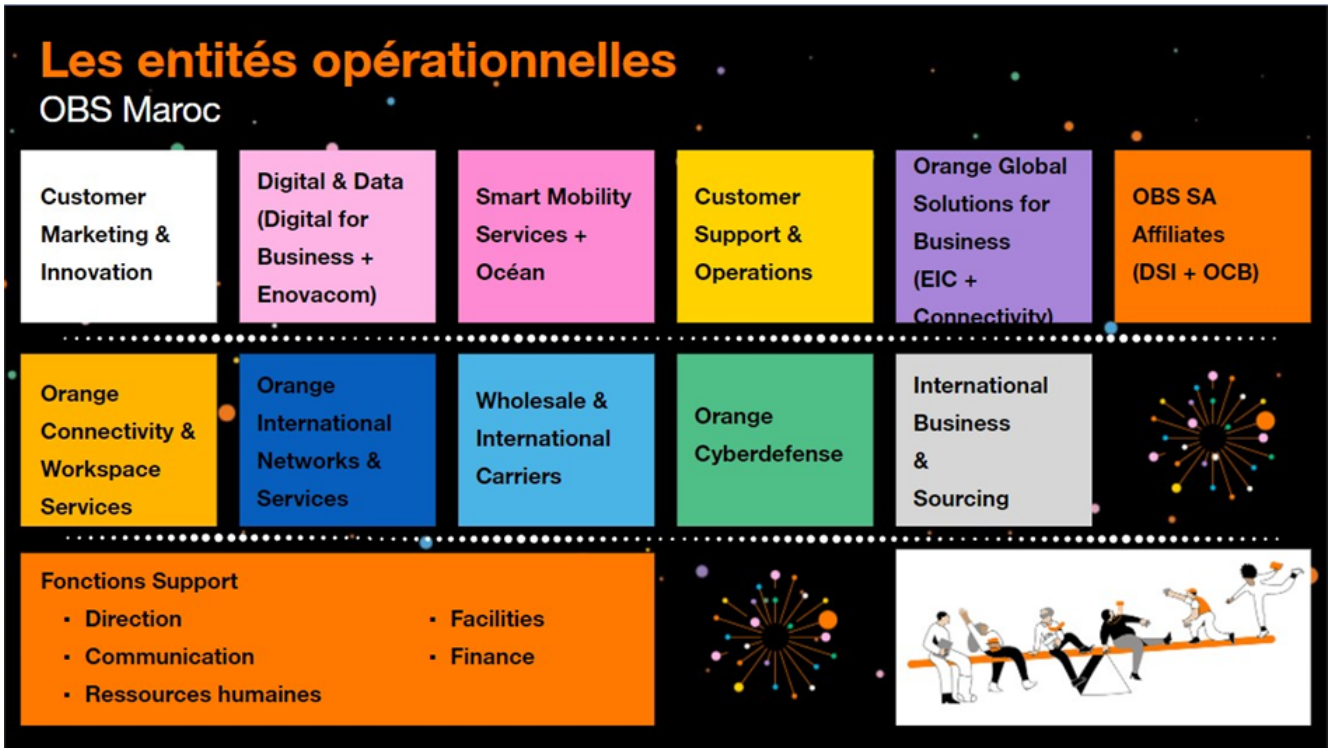

Figure 1.2: OBS IT Morocco's operating entities

CMI: This is the IT Department of the Orange Group. Its goal is to create efficient software development, entity-oriented, based on the best practices of a software factory (digital factory model). This entity adopts an agile delivery model, possesses a simple and automated development and production environment. It is a versatile, autonomous, and committed team to manage all activities related to the applications entrusted to it: from collecting customer needs and requirements, to writing user stories, through development, testing, and integration, before reaching deployment and support.

- Enrich the relationship between companies and their customers through a digital approach and a 360° customer journey.
- Exploit machines and communicating objects that transmit real-time data to create new sources of revenue and differentiation.
- Make use of the data produced by companies and their customers to innovate and personalize their products and services.

OCWS: This entity is composed of 3 poles with different missions. The first center, internationally / in Morocco, manages the operational part. A pre-sales team provides technical assistance to commercial engineers to help them negotiate contracts. A BUILD team's missions include deploying network

solutions for customers and offering them the best possible connectivity experience through various technologies such as SDWAN, WIFI, LAN, etc.

OCD: This is the entity that designs cybersecurity services that support customers in the Maghreb and West Africa, throughout the life cycle of threats that can impact client companies. They have 5 main missions:

- Anticipate the latest threats.
- Identify the assets and critical data of customers, prepare the security strategy, and ensure its proper functioning.
- Deploy appropriate technology to defend the organization and manage it continuously.
- Monitor, qualify, and analyze security alerts to confirm if there has been an incident.
- Qualify, contain, and remedy attacks. Thanks to their sectoral expertise, they can offer tailor-made services to meet customer challenges.

Orange Connectivity: Entity responsible for marketing OBS Connectivity data offers (Internet, Ethernet, VPN).

Missions in Morocco: Management of pricing for offers marketed in France and internationally (definition and update of standard prices, calculation of custom prices).

OINIS: Orange International Networks Infrastructures & Services' mission is to design, deploy, and operate reliable, secure, and high-quality international network infrastructures and services for businesses, wholesale customers (operators), and subsidiaries.

WIN IC: This is the commercial arm of OINIS. This entity provides international connectivity services to operators and internet service providers worldwide.

The Business Unit EIC (Enriched Interactions and Collaboration): Straddling digital & data and connectivity, its goal is to design and bring to market communication and collaboration offerings from Orange Business Services: voice solutions, conference solutions, unified communications suites, and contact center solutions.

1.3 Project presentation

1.3.1 General project context

In today's world, ensuring the sustainability of IT practices is paramount. When it comes to IT professionals, it is imperative to prioritize eco-friendly strategies from the outset of the development process.

As a developer, you may not be required to master the intricacies of environmental impact assessments, as there are several reliable tools available to assess the carbon footprint of IT solutions. By integrating these assessments into the CI/CD pipeline and implementing effective remediation strategies, organizations can significantly reduce the environmental cost associated with IT operations.

This includes not only the direct costs of energy consumption but also the broader environmental impact of resource depletion and pollution. Consequently, IT teams have embraced a new approach known as Green IT, which promotes the integration of sustainability principles into IT practices and encourages close collaboration between environmental experts, developers, and IT engineers.

This approach ensures that environmental considerations are addressed at every stage of the development and delivery process, thereby fostering a more sustainable IT ecosystem.

The project is set within this broader context of optimizing Green IT practices at Orange Business Services Morocco. To support this initiative, the DevSecOps team has been recently created within the IT RUN entity to fulfill two main missions:

- **Cloud DevSecOps**: Design, build, and implement Cloud services and DevSecOps tools.
- **REaCT**: Implement the DevOps mindset within the company and support development projects.

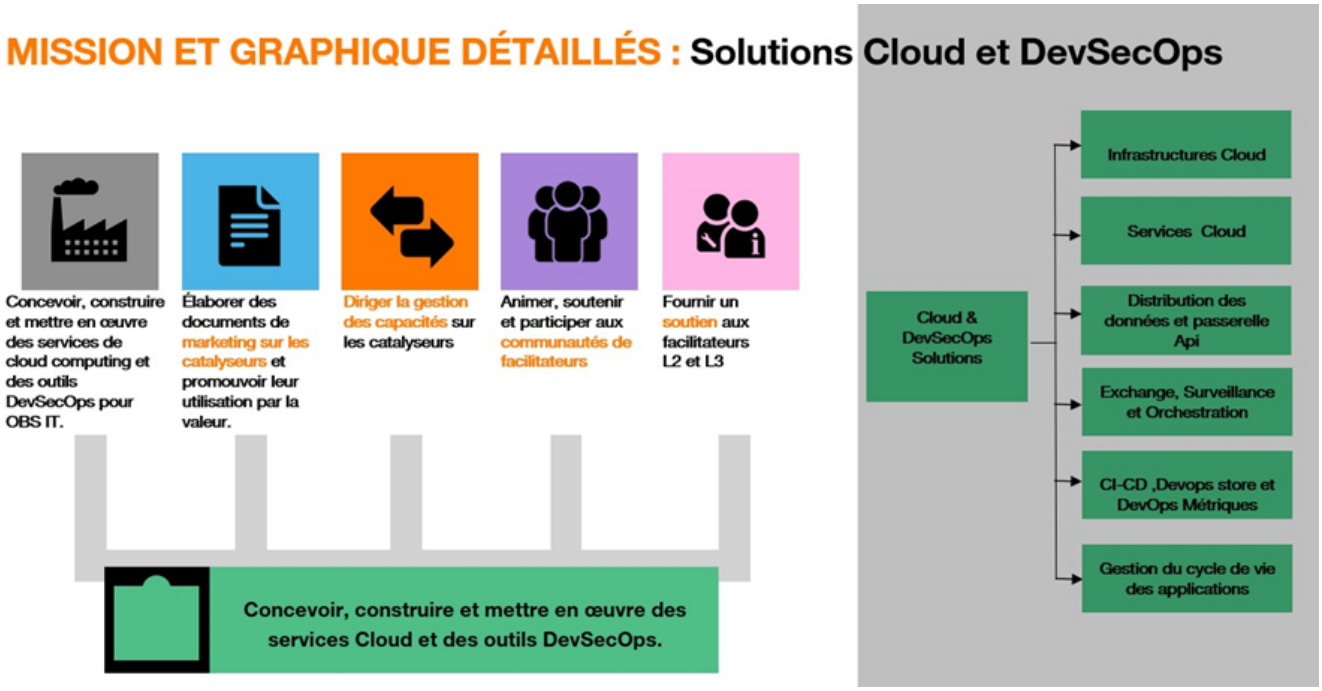

Figure 1.3: Cloud & DevSecOps missions

1.3.2 Problematic

While Orange Business Services Morocco has established efficient CI/CD pipelines using GitLab, there is a lack of focus on environmental sustainability and energy efficiency within their DevOps processes. The current approach prioritizes swift software delivery but overlooks the environmental impact of development and deployment activities. This neglect of Green IT principles hinders efforts to reduce energy consumption and carbon emissions associated with the organization's IT infrastructure, particularly its Kubernetes clusters. The challenge lies in integrating Green IT considerations seamlessly into existing DevOps workflows without compromising efficiency or productivity. Thus, the problematic revolves around balancing the need for rapid software delivery with environmental responsibility, requiring the development of strategies and tools to monitor and optimize energy usage and carbon footprint throughout the CI/CD pipeline

1.4 Project planning and management

1.4.1 Agile methodology

To ensure optimal progress of our project, we have chosen to use the agile Scrum method (illustrated in Fig. 1.10).

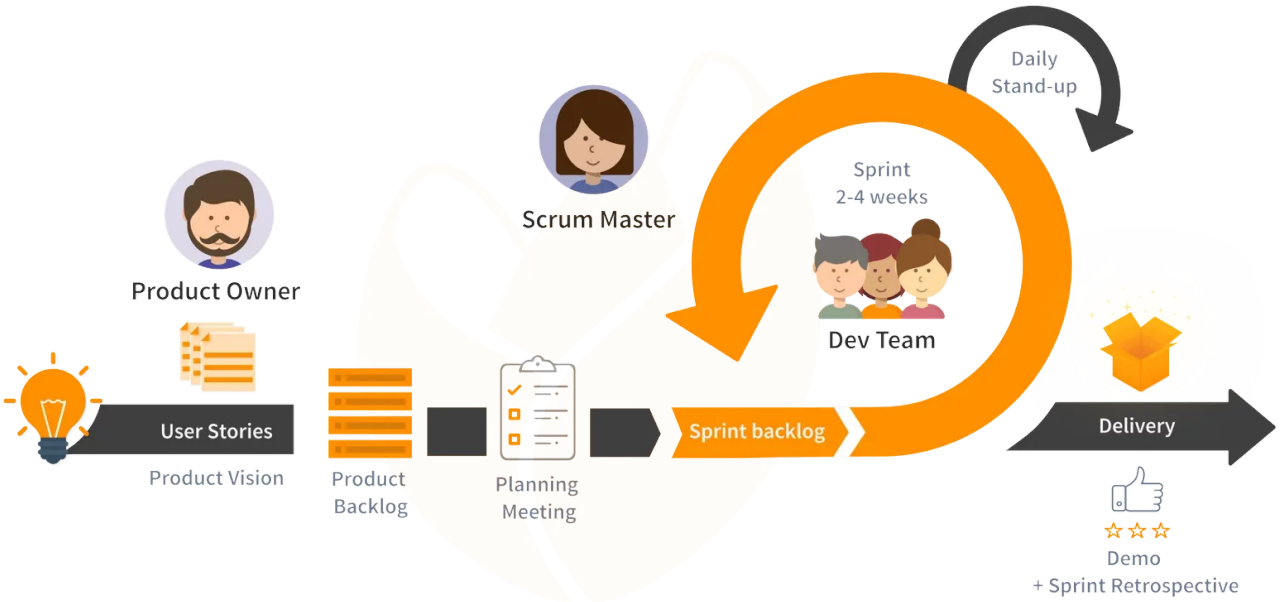

Figure 1.4: Scrum methodology

This approach will enable us to best meet the needs of the company while minimizing the risk of delays and ensuring the satisfaction of all stakeholders.

Scrum is an agile project management method that was first introduced in an article by Hirotaka Takeuchi and Ikujiro Nonaka in 1986, published in the Harvard Business Review under the title "The New Product Development Game". It was developed in the 1990s and formalized by Ken Schwaber and Jeff Sutherland in 1995.

The fundamental principle of this method is to iteratively focus on the features to be implemented. The project is divided into functional modules that are developed, tested, and delivered in iterative sequences called "sprints". Each sprint aims to achieve a specific goal, from which the features to be implemented are chosen.

We have adapted the Scrum method to our project by dividing it into sprints of two to three weeks. At the end of each sprint, a meeting is organized with the Scrum Master and the Product Owner to review the work done during the period and set the tasks and objectives for the next sprint. If differences are noted during the inspection, we adapt the process in question. To do this, we have set up weekly meetings led by the Scrum Master, during which each team member presents the status of their tasks, reports any encountered blockers, and outlines the actions to be taken for the next period as well as future prospects. This approach has allowed us to clearly define the objectives for each increment and adapt them as needed, while also allowing us to complete or modify the list of features to be implemented for the upcoming sprints.

1.4.2 Project Planning

Planning is an essential step in project management, as it allows defining the tasks to be performed, setting objectives, coordinating actions, mastering resources, reducing risks, monitoring ongoing actions, and reporting on the project's progress. In our case, we began by breaking down the project into tasks, then proceeded with a risk assessment before establishing a Gantt chart to have an overview of the project's progress.

1.4.3 Project Breakdown into Sprints

The project is divided into sprints, with each sprint being a crucial step in the project's completion. We used Jira for managing and tracking the progress of these sprints, ensuring that all tasks were clearly defined, assigned, and monitored efficiently.

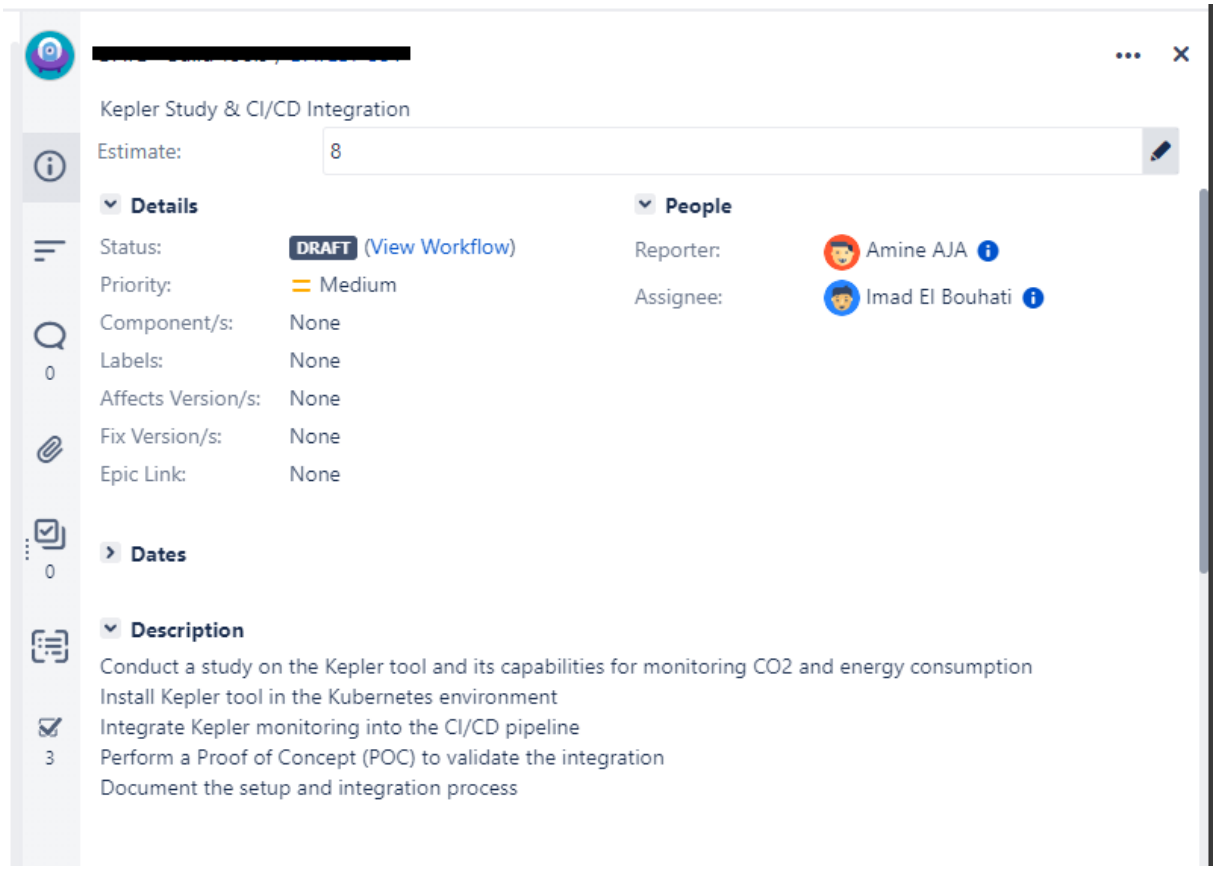

Figure 1.5: Performed tasks in sprint 10

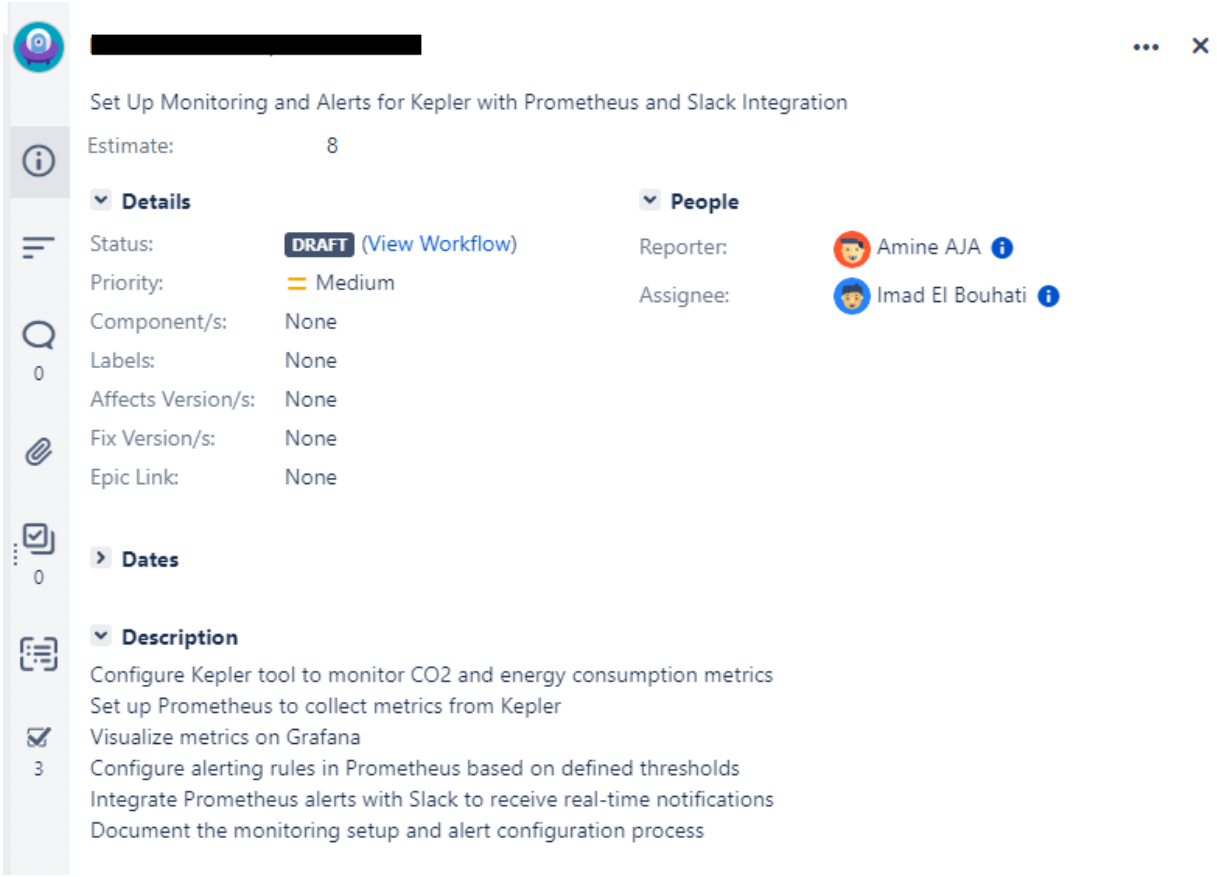

Figure 1.6: Performed tasks in sprint 11

1.4.4 Communication Tools

In the dynamic environment of modern project management, effective communication is paramount to ensure seamless collaboration and timely decision-making. Utilizing robust communication tools can streamline interactions within teams and enhance overall productivity.

Figure 1.7: Microsoft Teams Logo

Microsoft Teams serves as a cornerstone for communication within our DevSecOps team. This platform offers a comprehensive suite of features, including instant messaging, video conferencing, file sharing, and integration with various productivity tools. Team members can collaborate in real-time, hold meetings, and access project-related resources, fostering efficient communication and collaboration.

1.4.5 Internship Progress

The Gantt chart is a visual presentation method that allows positioning in time the different stages, activities, tasks, and resources involved in a project. Tasks are listed on the rows, while the columns represent days, weeks, or months. The bars symbolizing each task have a length proportional to the expected duration.

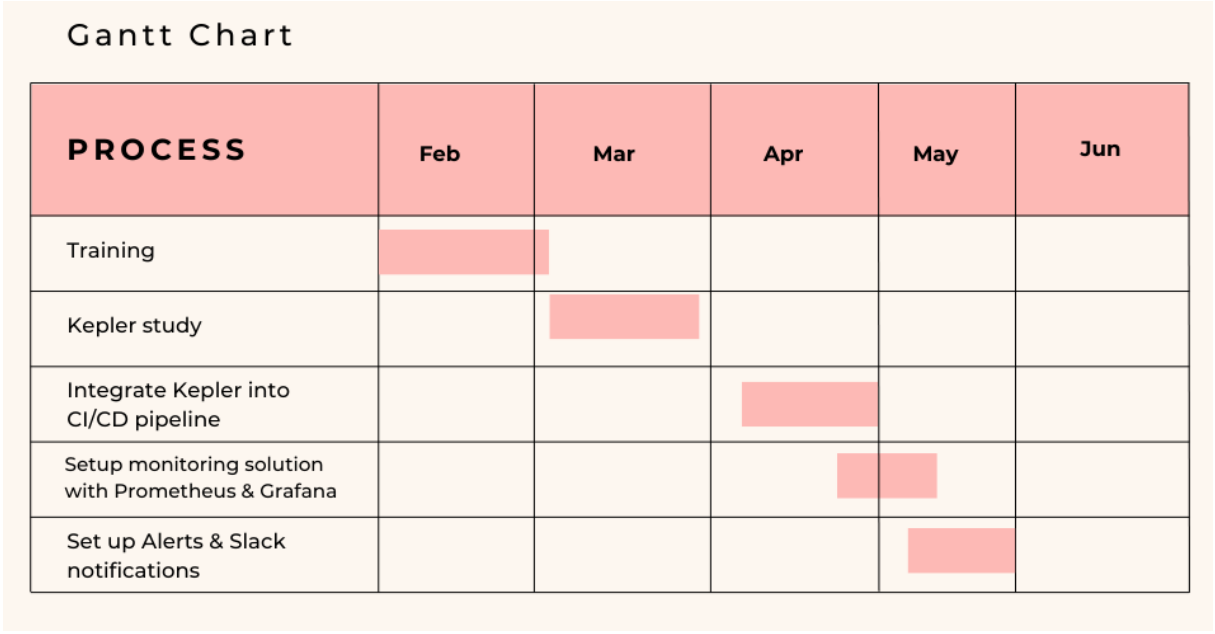

Figure 1.8: Gantt Chart

1.5 Conclusion

In this chapter, we first introduced Orange Business Morocco, the organization involved in the project. Then, we established the general framework of the project before presenting the specific problem to be addressed. We also explained the working method we adopted and provided an overview of the schedule that was followed throughout the project. In the next chapter, we will move on to the functional and non-functional analysis of our project, as well as detailed design. This phase will define the specifications necessary for the development of the overall project architecture.

Chapter 2

Specification and Requirements Analysis

2.1 Introduction

The success of a project heavily relies on the quality of its initiation. Therefore, the functional study phase is crucial for a successful start to the project. This chapter encompasses requirement specifications, including needs analysis involving stakeholder identification, system use cases, and the textual scenarios associated with these use cases.

2.2 Study and analysis of the current state

Orange Business Services Morocco has embraced an approach known as CI/CD to efficiently manage the development and delivery of its projects. They utilize the GitLab platform, which facilitates swift creation, compilation, testing, delivery, and deployment of software.

In their existing solution, they have established continuous integration and delivery pipelines using GitLab CI, Nexus, Docker, and Kubernetes. Here's how it operates:

- Code management and updates are performed using GitLab. Developers push their code changes to the GitLab repository.
- The application is containerized using Docker, a technology enabling the creation of isolated and portable environments for applications.
- The GitLab Runner conducts a check to ensure the YAML code is correct and clean. Then, it triggers pipeline steps such as Docker image generation and their submission to the Nexus repository.
- Kubernetes, aided by Helm (a package manager for Kubernetes), handles the application deployment across various environments. It follows a "Push" deployment model.

This approach enables Orange Business Services Morocco to efficiently manage the development and delivery of their projects by automating processes using GitLab, Docker, Nexus, and Kubernetes. It empowers them to deliver high-quality software more rapidly.

2.2.1 Criticism of the current state

While utilizing GitLab accelerates software development and deployment, there's a clear need for enhancements focused on sustainability and reducing environmental impact. An exhaustive study of the existing system was conducted to refine our project's scope and expected functionalities, aiming for a more reliable system than the previous one. Identified areas for improvement include

- The current CI/CD approach relies heavily on resource-intensive technologies like Docker and Kubernetes, which contribute to increased energy consumption and carbon emissions.
- While effective for streamlining development workflows, the pipelines lack mechanisms for tracking and optimizing energy usage.
- Without visibility into the energy consumption and carbon footprint of the Kubernetes cluster, targeted strategies for reducing environmental impact are challenging to implement.

2.2.2 Functional requirements

Functional requirements gathering is a crucial step in the project. This stage produces the functional specifications document, during which the expected functionalities are formalized along with all governing management rules.

To address the issues identified in the existing study, the principle is to integrate monitoring tools such as Kepler (*Kubernetes-based Efficient Power Level Exporter*), Prometheus, and Grafana into the CI/CD pipeline. This integration aims to measure the energy consumption and CO_2 emissions of Orange's Kubernetes cluster, providing valuable insights for optimizing Green IT practices within the DevOps workflow.

Integration of Monitoring Tools

The integration of these monitoring tools is essential. Kepler will monitor energy consumption and CO_2 emissions, Prometheus will collect and store metrics from Kepler, and Grafana will visualize the collected metrics for better insights. Automating the deployment and configuration of Kepler, Prometheus, and Grafana within the CI/CD pipeline is also a key requirement, ensuring that monitoring is an integral part of the continuous integration and deployment processes.

GitLab Template Creation

Creating a GitLab template for GitLab CI/CD pipelines will streamline the process of integrating and deploying Kepler, Prometheus, and Grafana configurations. This template will include automated steps for deployment, configuration, and monitoring setup, ensuring consistency and efficiency across projects.

Data Collection and Visualization

Collecting real-time data on energy consumption and CO_2 emissions from the Kubernetes cluster is critical, along with storing historical data for trend analysis and reporting. This data will be visualized using Grafana dashboards, providing customizable views to monitor both real-time and historical data.

Alerting and Notification

Additionally, setting up alerting rules in Prometheus to monitor key metrics such as high energy consumption or CO_2 emission levels is vital. Defining thresholds for these alerts will trigger notifications based on predefined conditions.

Integrating Prometheus Alertmanager with Slack to send notifications ensures that alerts are promptly sent to specific Slack channels or user groups for immediate action. This notification system will help in taking timely measures to address any issues related to energy consumption and emissions, thereby supporting the goal of optimizing Green IT practices within the DevOps workflow.

2.2.3 Non-Functional requirements

Non-functional requirements play a crucial role in system design, because they define the attributes, constraints and restrictions to be taken into account.

These requirements, also called system qualities, guarantee the user-friendliness and overall efficiency of the system. If any of these requirements are not met, the system risks not meeting the internal needs of the company, users or the market.

- **Accuracy**: The system must provide accurate measurements of energy consumption and CO_2 emissions to support informed decision-making.
- **Scalability**: Ensure scalability to handle increasing data volumes as the size of the Kubernetes cluster grows.
- **Reliability**: The solution should be reliable, with minimal downtime, to maintain continuous monitoring and data collection.

- **Security**: Implement robust security measures to safeguard sensitive energy consumption and emission data from unauthorized access or tampering.
- **Performance**: Ensure efficient performance to deliver timely insights and visualizations, even during peak usage periods.
- **Ease of Use**: Design an intuitive user interface that allows administrators to easily configure monitoring settings and interpret data visualizations.
- **Compatibility**: Ensure compatibility with existing infrastructure and tools used within Orange Business Services Morocco, such as Prometheus, Grafana, and Kepler.

2.3 DevOps

2.3.1 Definition

The word DevOps is a combination of the terms development and operations, meant to represent a collaborative or shared approach to the tasks performed by a company's application development and IT operations teams. The DevOps methodology aims to shorten the systems development lifecycle and provide continuous delivery with high software quality. These characteristics help ensure a culture of building, testing, and releasing software that is more reliable and at a high velocity.

2.3.2 CI/CD

Continuous Integration (CI) is a software development practice where developers frequently merge their code changes into a central repository, typically multiple times a day. Each merge triggers an automated build and testing process. The primary goal of CI is to detect and address issues early in the development cycle, which helps in maintaining code quality and reducing integration problems. By integrating frequently, developers can identify bugs early, facilitating easier and quicker fixes.

Continuous Delivery (CD) extends CI by automatically preparing code changes for a release to production. It ensures that the software can be reliably released at any time, with the deployment process being automated but requiring manual approval. Continuous Deployment is a further extension of CD where every change that passes all stages of the production pipeline is automatically released to customers without human intervention. This practice reduces the lead time for delivering new features and fixes, thus providing rapid feedback to developers and stakeholders.

2.3.3 Green DevOps:

Green DevOps extends DevOps principles, focusing on resource efficiency while staying agile. It emphasizes sustainability, reducing environmental impact, and integrating eco-friendly strategies into software development. The goal is to balance technological innovation with ecological responsibility.

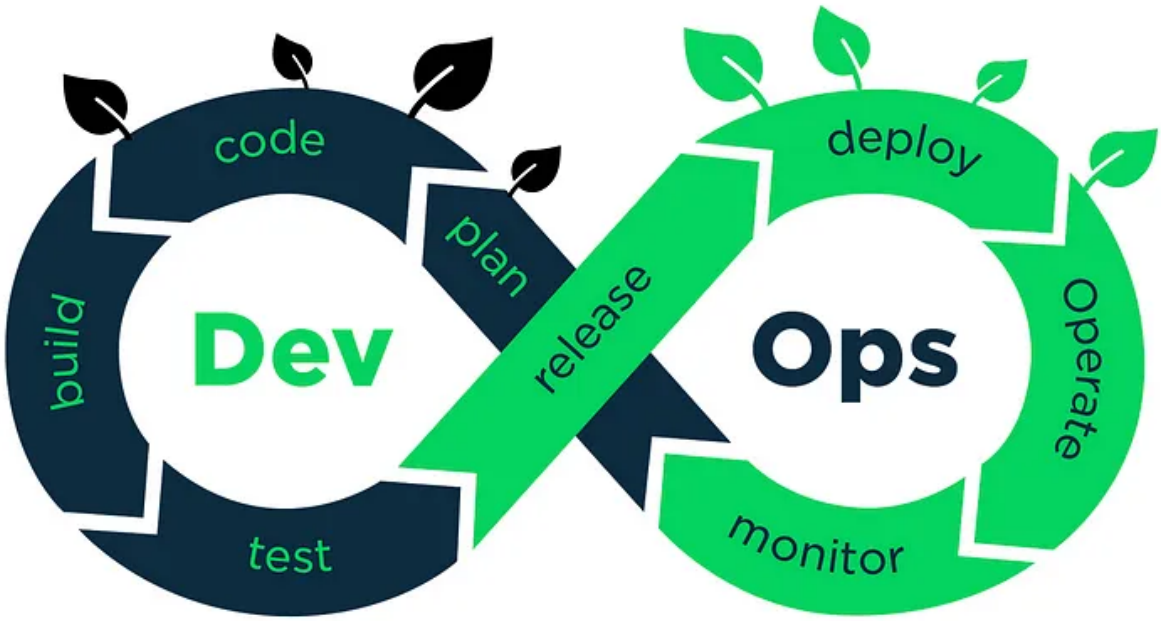

Figure 2.1: Green DevOps

The Fundamental Principles of Green DevOps

Green DevOps is nothing more than an extension of DevOps. It shouldn't be viewed as an upheaval of all DevOps principles. We retain all DevOps principles but adjust them as finely as possible to optimize resource usage.

Green DevOps relies on the following concepts:

- **Infrastructure Optimization:** Choosing eco-friendly data centers, energy-efficient hardware, and effective resource utilization are pillars of Green DevOps. The goal is to reduce energy consumption while maintaining performance.
- **Reduction of E-waste:** This involves minimizing over-provisioning of IT resources, properly managing electronic waste, and promoting hardware component recycling.
- **Automation and Process Optimization:** Automation is at the core of DevOps, but it holds even greater importance in the context of Green DevOps. Automating tasks reduces energy consumption by minimizing unused resources and enabling fine-grained resource management.

- **Carbon Impact Assessment:** To truly become "green," measurement is necessary. Green DevOps integrates tools and metrics to continuously assess and monitor the carbon footprint of projects. This helps identify areas needing improvement and track progress.
- **Reduction of Cycle Times:** Shorter development cycles mean less energy consumption and faster time to market. By reducing delays, Green DevOps contributes to the competitiveness of the business while minimizing its environmental impact.
- **Awareness:** DevOps teams need to be aware of environmental issues. Employee training and engagement are essential to fostering the adoption of sustainable practices.

The Benefits of Green DevOps

- **Cost Reduction**: Less wasted resources, time saved, and efficient resource utilization result in a significant reduction in operating and infrastructure costs.
- **Quality Improvement**: Green DevOps practices encourage automation, monitoring, and rigorous testing, leading to better software quality, fewer bugs, and an enhanced user experience.
- **Social Responsibility**: Companies adopting Green DevOps demonstrate their commitment to environmental sustainability, which can enhance their brand image and attract new customers and talents.
- **Regulatory Compliance**: With an increasing number of environmental regulations in place, Green DevOps helps companies comply with standards and avoid potential penalties.

2.3.4 DevOps Monitoring

DevOps monitoring is the process of tracking and measuring the performance of applications and systems in order to help software development teams identify and resolve potential issues more quickly. This is typically done via a manual or automated DevOps monitoring solution or a collection of continuous monitoring tools that gather data

DevOps Monitoring Use Cases

The main benefit of DevOps monitoring is its ability to define, track, and measure KPIs across all aspects of DevOps. Here are some specific use cases of DevOps monitoring:

- **Detect and Report Errors Earlier:** Flagging issues to DevOps teams more quickly means they can resolve them before they impact user experience. Early detection and reporting of errors allow for prompt resolution, minimizing disruptions and maintaining a smooth user experience.
- **Reduce System Downtime:** DevOps monitoring tools provide continuous oversight of databases, applications, and networks, enabling teams to resolve issues before system downtimes occur. By proactively identifying potential problems, teams can take preemptive actions to avoid outages and ensure system reliability.
- **Enhance Observability of DevOps Components:** Easily identify when various systems and applications in your DevOps stack degrade in performance, cost, security, or other factors to avoid problems down the road. Enhanced observability enables teams to maintain optimal performance and security by monitoring and analyzing system behavior continuously.
- **Uncover Root Cause of Issues Faster:** Continuous tracking of logs and metrics helps teams identify the root cause — where a problem started or occurred. This allows engineers to detect patterns in system behavior to anticipate and prevent future issues, improving mean time to detection (MTTD), mean time to repair (MTTR), and mean time to isolate (MTTI).

2.4 Benchmark

2.4.1 Scaphandre vs. Kepler

In the landscape of power consumption monitoring tools, Scaphandre and Kepler stand out as prominent contenders. As organizations increasingly prioritize energy efficiency and sustainability, understanding the capabilities of these tools becomes crucial. In this benchmark, we compare Scaphandre and Kepler to provide insights into their strengths and suitability for diverse computing environments.

Aspect	Scaphandre	Kepler
Scope and Granularity	Provides per-process power usage monitoring. Collects real-time power consumption metrics from system components using sensors like RAPL.	Offers estimation of power consumption at the process, container, and Kubernetes pod levels. Utilizes various APIs and sensors to collect real-time power metrics, including RAPL for CPUs and DRAM, NVML for GPUs, ACPI for platform power, etc.
Environment Support	Primarily focused on monitoring power consumption in the host environment.	Supports both bare-metal and virtualized environments. Provides ways to estimate power consumption in VMs on public clouds where direct power metrics may not be available.
Modeling and Estimation	Utilizes a ratio-based model to estimate power consumption per process based on resource utilization metrics collected from the system. Relies on sensors like RAPL to measure absolute power consumption.	Employs various models, including trained power models for VMs in public clouds and regression-based models, to estimate power consumption at different levels within the environment.
Training and Optimization	Does not involve explicit training of power models. Relies on direct sensor readings and resource utilization metrics to estimate power consumption.	Includes a Model Server component for training power models using Prometheus metrics from bare-metal nodes. The models are continuously optimized for accuracy and can be shared publicly for use in various environments.
Limitations and Future Enhancements	Acknowledges limitations such as the accuracy of pre-trained power models, challenges in estimating power in virtualized environments, and overhead issues.	Acknowledges similar limitations and actively works on enhancements, including broader support for power data sources, vendor-agnostic support, performance optimization, and multi-level deployment strategies for accurate power estimation in VMs.
Best Choice: Kepler stands out as the best choice due to its comprehensive approach to power consumption monitoring and estimation. With support for multiple environments, advanced modeling techniques, continuous optimization of power models, and active development for future enhancements, Kepler offers a robust solution for accurate and efficient power management in modern computing environments.		

Table 2.1: Comparison of Scaphandre and Kepler

2.4.2 Kepler

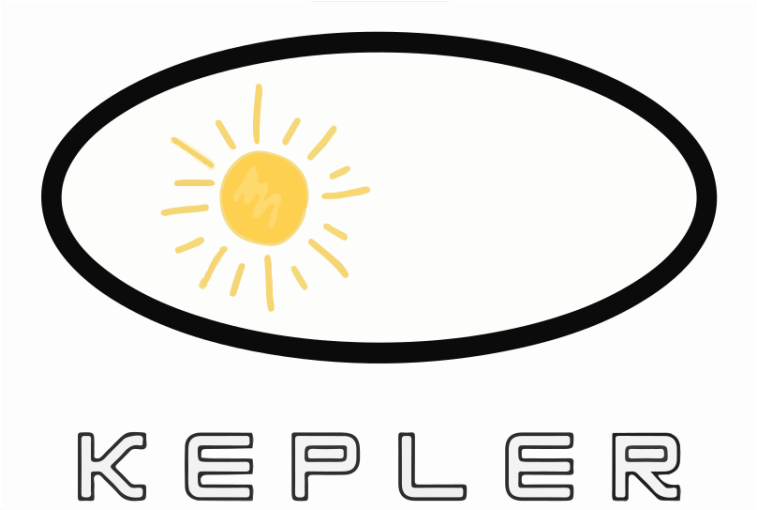

Figure 2.2: Kepler-logo

Kepler, short for Kubernetes-based Efficient Power Level Exporter, is a sophisticated tool designed to measure and estimate power consumption at various levels within a cloud infrastructure. It aims to provide detailed insights into the energy usage of processes, containers, and Kubernetes pods, making it an invaluable resource for both industrial and research projects. Kepler's architecture is extensible, allowing for the integration of novel power models that cater to diverse system architectures. This tool is essential in an era where efficient energy consumption is critical to minimize environmental impact and manage operational costs effectively.

2.4.3 Kepler Functionalities

Kepler offers several key functionalities that make it a powerful tool for monitoring and optimizing energy consumption in cloud environments:

1. **Real-Time Power Metrics Collection:** Kepler can gather real-time power consumption metrics from various hardware components, such as CPUs, GPUs, and DRAM. This is achieved through the use of APIs like Intel Running Average Power Limit (RAPL) and NVIDIA Management Library (NVML). These metrics provide an accurate and immediate view of the power usage of different system components.

2. **Power Consumption Estimation:** Kepler estimates power usage at multiple levels, including processes, containers, and Kubernetes pods. By analyzing resource utilization metrics, Kepler

can determine the energy consumption of individual components and aggregate these to provide a comprehensive view of the system's power usage. This granular estimation is crucial for identifying power-hungry processes and optimizing resource allocation.

3. **Extensible Power Models:** One of Kepler's strengths is its extensibility. It allows for the integration of custom power models, which can be tailored to different system architectures and hardware setups. This adaptability ensures that Kepler can provide accurate power consumption estimates across a variety of environments, making it a versatile tool for both industrial and research applications.

4. **Data Aggregation and Storage:** The power consumption data collected and estimated by Kepler are stored in Prometheus, a powerful monitoring and alerting toolkit. This integration facilitates easy access, analysis, and visualization of power metrics. Users can query Prometheus to gain insights into energy consumption patterns and identify opportunities for optimization.

5. **Support for Bare-metal and Virtual Machines:** Kepler is designed to operate in both bare-metal and virtual machine (VM) environments. In bare-metal nodes, it uses direct power metrics from hardware components. In VM environments, where direct power metrics are not available, Kepler employs trained power models to estimate power consumption. This dual approach ensures that Kepler can provide valuable insights regardless of the underlying infrastructure.

2.4.4 How Kepler collects metrics?

The method for collecting system power consumption metrics in Kepler varies based on the deployment environment—bare-metal (BM) or virtual machines (VMs).

In bare-metal environments, Kepler directly collects real-time system power metrics and utilizes the Ratio Power model to estimate power consumption. This model splits the dynamic power (related to resource utilization) proportionally across processes, while idle power (constant regardless of load) is divided according to the size of processes/containers as per the GreenHouse Gas (GHG) protocol. Hardware counters are used to assess CPU, memory, and GPU utilization.

In public cloud VM environments, direct power measurement isn't feasible. Instead, Kepler uses trained power models for estimation. These models are based on regression analysis of data from benchmark tests. Kepler can also help cloud providers expose VM power metrics by measuring power at the BM level and passing this data to VMs via hypervisor Hypercalls. Within VMs, Kepler applies the Ratio Power Model to estimate power usage per process, using eBPF metrics due to the absence

of hardware counters. However, Kepler does not estimate idle power for VMs due to unknowns about other running VMs on the host.

The trained power models allow for accurate power estimation despite the limitations of VM environments, and Kepler's approach ensures adaptability across different deployment scenarios.

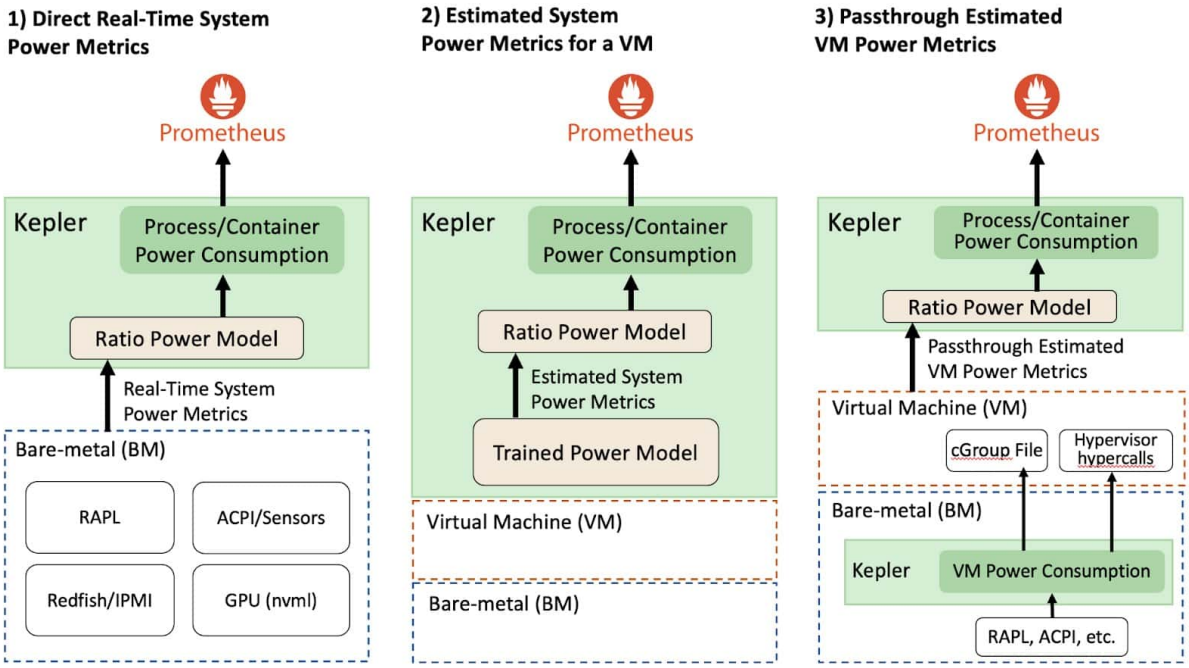

Figure 2.3: Collecting System Power Consumption – VMs versus BMs

2.5 Identification of Stakeholders

Identifying stakeholders is a fundamental step in any development project. It helps to understand the interactions and measure the influence of each group on the actions to be undertaken. But what exactly is a stakeholder? A stakeholder can be a physical person or a legal entity that participates in or is affected by the action or project in question. It is important to clearly specify the action or series of actions for which we seek to determine who the stakeholders are and what they represent.

In the context of integrating Green IT into DevOps practices at Orange Business Services Morocco, we identify the following stakeholders:

- **Platform Admin Team**: Responsible for setting up staging and production environments, managing cluster extension modules, and other cluster resources (controllers and admission webhooks), integrating development team repositories using customized GitRepository resources, and configuring how development team repositories are reconciled on each cluster.
- **Development Team**: Configures application definitions at the Kubernetes deployment level, manages Helm releases, configures how applications are reconciled across environments, and oversees the promotion of applications between environments.
- **Green IT Team**: Responsible for analyzing the energy consumption and CO_2 emissions of the Kubernetes clusters.
- **GitLab**: A secondary stakeholder that manages team access and coordinates the CI/CD processes.

By identifying these stakeholders and their respective roles, we can ensure a comprehensive approach to integrating Green IT into our DevOps practices, balancing rapid software delivery with environmental responsibility.

2.6 Conclusion

This chapter outlined the essential functional and non-functional requirements for integrating monitoring tools into the CI/CD pipeline. By leveraging Kepler, Prometheus, and Grafana, we aim to provide accurate, scalable, and reliable measurements of energy consumption and CO_2 emissions. Emphasis was placed on security, performance, ease of use, and compatibility with existing infrastructure. These requirements form the blueprint for implementation, ensuring the system meets its objectives and supports informed decision-making through effective monitoring and visualization.

Chapter 3

Used Technologies

3.1 Introduction

The adoption of the DevOps approach largely relies on the automation of communication between the various required tools, thus allowing us to benefit from the previously described functionalities. The choice of tools for each stage of our DevOps pipeline, such as continuous integration and continuous deployment, depends on specific criteria related to our needs. Consequently, this comparative study aims to reinforce the selection of tools that we have implemented for the integration of Green IT into DevOps practices.

3.2 Application Containerization

In this section, we will delve into our problem by examining the fundamental principles of traditional virtualization and containerization. Our aim is to provide a clear description of these concepts and explain the relationships between them.

3.2.1 Virtualization

Virtualization involves creating a virtual version of a device or resource, such as an operating system, server, storage device, or network resource. It allows for the abstraction of physical resources, representing them independently of their physical equivalents. Each virtual element, such as a disk, network interface, local network, switch, processor, or memory, is associated with a physical resource in a computer system. Thus, virtual machines hosted by the host machine are considered applications that require allocation or distribution of the host's resources. There are various types of virtualization, each corresponding to a specific use case:

- **System Virtualization** is a server virtualization technique that uses a hypervisor to allow the host machine to run multiple virtual instances simultaneously. These virtual instances are commonly referred to as virtual machines (VMs), while the hypervisor is known as the virtual machine monitor (VMM), responsible for managing the VMs.
- **Application Virtualization** differs from system virtualization by using a virtualization layer in the form of an application. This application creates virtual instances and isolates them from the specifics of the host machine, such as its architecture or operating system. This approach allows developers to avoid creating multiple versions of their software to run in different environments. Common application virtual machines include the JVM (Java Virtual Machine) and containers.

3.2.2 Containerization

Container-based virtualization, also known as containerization, offers an alternative to system virtualization. This approach directly leverages kernel features to create isolated virtual environments called containers. Containers use control groups (cgroups) and namespaces provided by the operating system kernel. Namespaces allow for controlling and limiting the resources used by a process, while cgroups manage the resources of a group of processes. Thus, a container provides the necessary resources to run applications in isolation, as if they were the only processes running on the host machine's operating system.

Containers offer advantages such as scalable, modular, and loosely coupled application development. The need to create portable deliverables independent of development or production environments has become a standard. The term "container" emerged to refer to an application independent of the system, represented as a box containing all the dependencies necessary for its proper functioning.

3.2.3 Containerization vs Virtualization

Containers have an intrinsically smaller footprint than virtual machines and require less startup time. This means that more containers can run on the same computing capacity compared to a single virtual machine. This improvement in server efficiency reduces costs associated with servers and licenses. In simple terms, containerization allows applications to be developed once and run anywhere. This portability is crucial for streamlining the development process and ensuring compatibility with different service providers.

Figure 3.1 illustrates the distinction between container architecture and virtual machine architecture.

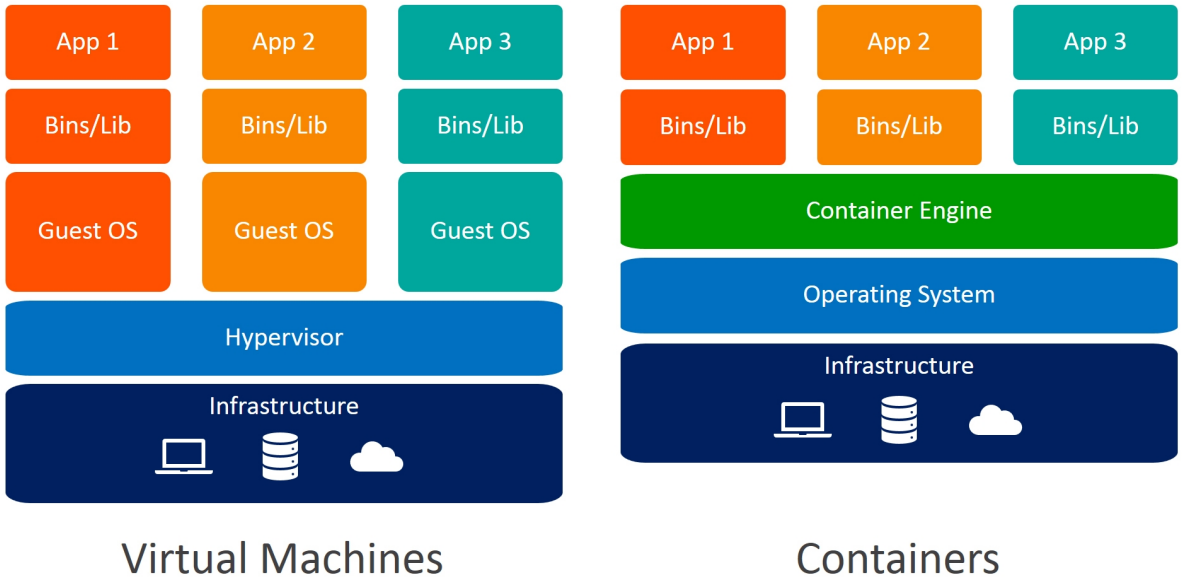

Figure 3.1: Comparison between Container and Virtual Machine Architectures

3.2.4 Containerization Technologies

There are several containerization tools available in the market. We conducted a comparison to select the most suitable one.

The analysis presented in Table 3.1 highlights the significance of Docker in the market. Docker stands out as the most universal solution in terms of image compatibility and offers numerous additional

Technology	Principle	Market Position	Documentation/ Complexity
Docker	A container contains a single process; multiple containers can be connected to form an application.	Market leader, widely used by major online service providers (e.g., Netflix, Spotify)	Official documentation is available; community-driven tutorials and resources are abundant. Known for its simplicity.
Podman	A container contains a single process; multiple containers can be connected to form an application.	Increasingly popular, supported by major enterprises like Google and Red Hat. Acquired by Red Hat in May 2018.	Limited documentation, somewhat difficult due to lack of resources but highly modular and compatible with ancillary tools.
LXC/LXD	Isolation of an entire application within a complete Linux environment (similar to a VM).	Established since 2008, the only one closely resembling a VM. Isolated but with an active community.	Limited documentation; most information is posted on Ubuntu community forums.

Table 3.1: Comparison of Containerization Tools

advantages. It simplifies the creation and management of containers, facilitates the design and construction of images, and enables easy distribution and version control of these images. Thus, Docker plays a crucial role in streamlining and enhancing containerization processes.

3.3 Docker Architecture

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.

- **Docker Daemon (*dockerd*):** Responsible for handling Docker API requests and managing various Docker objects such as images, containers, networks, and volumes. Listens for incoming requests and executes them accordingly. Can interact with other daemons to manage Docker services across different environments.

- **Docker Client (*docker*):** Primary interface for Docker users to interact with the Docker ecosystem. Users issue commands through the Docker client (e.g., `docker run`), which are communicated to the daemon for execution. Utilizes the Docker API for communication and can connect with multiple daemons for managing Docker instances.

- **Docker Registries:** Facilitate storage and distribution of containerized applications. Docker Hub is a public registry accessible to all users, while private registries can be set up for hosting proprietary or sensitive images. Commands like `docker pull` or `docker run` fetch required images from the configured registry, and `docker push` uploads images to the designated registry for sharing or backup purposes.

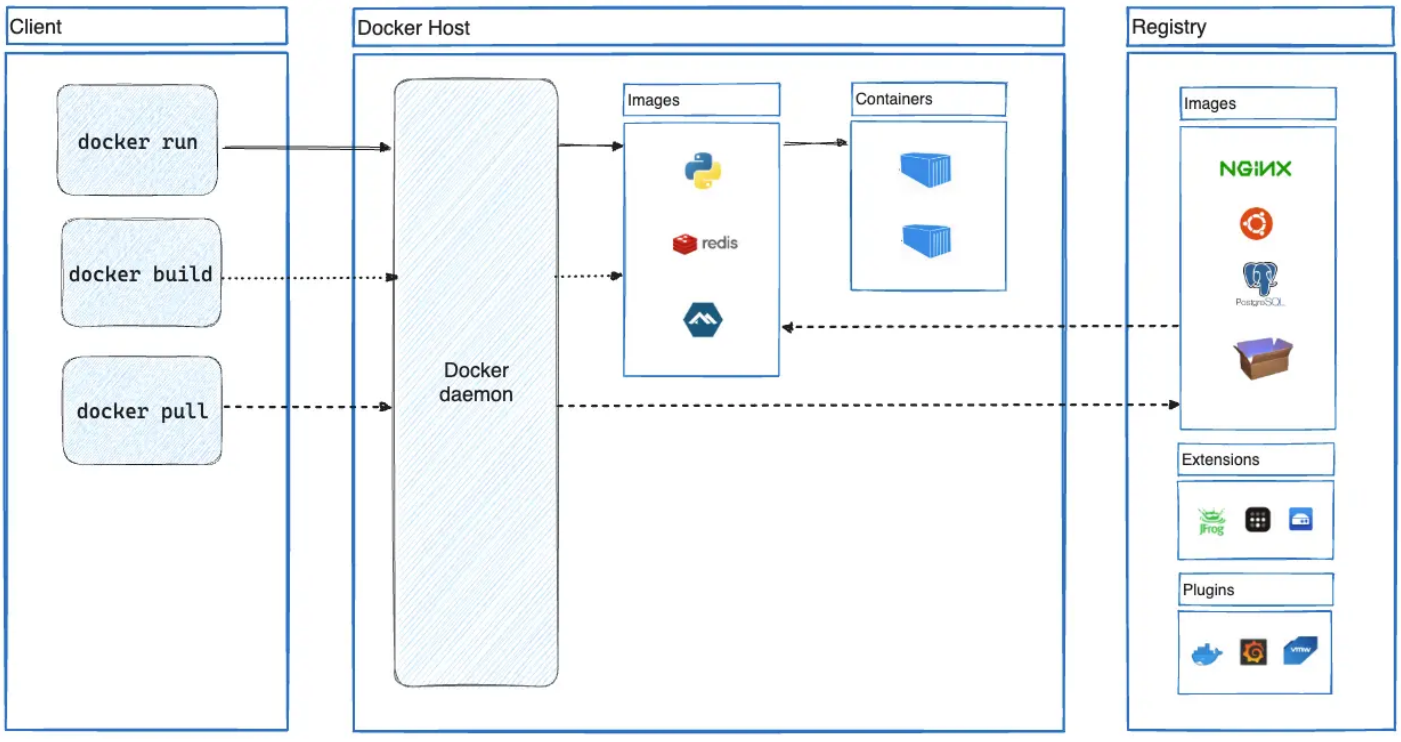

Figure 3.2: Docker Architecture

3.4 Container Orchestration

Container orchestration involves automating much of the operational tasks required to run containerized workloads and services. This encompasses various aspects that software teams need to manage the lifecycle of a container, such as provisioning, deployment, scaling (both up and down), network management, and load balancing. Container orchestration simplifies and optimizes container management by providing tools and features to effectively coordinate different container instances and associated resources. This facilitates the deployment and management of complex and scalable containerized applications in production environments.

3.4.1 Orchestration role

The ability to create containers has existed for several decades but became widely accessible in 2008 when Linux integrated container functionality into its kernel. However, widespread adoption of containers took off with the introduction of Docker, an open-source containerization platform, in 2013. Docker became so popular that the terms "Docker containers" and "containers" are often used interchangeably.

Containers have become the de facto computing units for modern cloud-native applications due to their small size, resource efficiency, and portability compared to virtual machines (VMs). Containerized microservices or serverless functions are particularly prevalent in these applications. Manually deploying and managing a small number of containers is relatively easy. However, in most organizations, the number of containerized applications is rapidly increasing, making large-scale management impossible without automation. This is where container orchestration comes into play.

Container orchestration provides essential automation for managing a large number of containerized applications. This significantly reduces the effort and complexity associated with managing this application fleet. By automating operations, orchestration facilitates an agile or DevOps approach, allowing development teams to rapidly deploy new features and capabilities.

The automation provided by orchestration enhances many inherent benefits of containerization. For example, it enables automated host selection and efficient resource allocation based on declarative configuration, thereby maximizing the utilization of computing resources. Additionally, orchestration provides automated monitoring of container health status and can move containers as needed to maximize availability.

In summary, container orchestration brings essential automation to effectively manage a large number of containerized applications. It facilitates rapid and iterative development cycles, allowing teams to

release new features and capabilities more quickly. Furthermore, orchestration can enhance and extend the benefits of containerization through advanced features such as resource management and automated monitoring.

3.4.2 Kubernetes Container Orchestration

Kubernetes is a powerful and popular container orchestration platform that enables enterprises to provide a highly productive PaaS for developing cloud-native applications. With Kubernetes, development teams can focus on coding and innovation, while container infrastructure and operations are managed automatically and efficiently. Kubernetes' advantages over other orchestration solutions (such as Docker Swarm, Apache Mesos, OpenShift, AWS Fargate) largely stem from its various more comprehensive and sophisticated features in several areas, including:

- **Container Deployment:** Kubernetes deploys a specified number of containers on a given host and maintains them in the desired state.
- **Rollouts:** A rollout is a modification to a deployment. Kubernetes allows launching, pausing, resuming, or canceling deployments.
- **Service Discovery:** Kubernetes can automatically expose a container to the internet or other containers using a DNS name or IP address.
- **Storage Provisioning:** Developers can configure Kubernetes to mount local or cloud-based persistent storage for containers as needed.
- **Load Balancing and Scaling:** When traffic to a container experiences a high spike, Kubernetes can employ load balancing and scaling to distribute it across the network to ensure stability and performance. (This also saves developers from having to configure a load balancer)
- **Self-healing for High Availability:** When a container fails, Kubernetes can automatically restart or replace it. It can also take containers out of service that do not meet our health check requirements.
- **Multi-cloud Provider Support and Portability:** Kubernetes enjoys broad support from all major cloud providers. This is particularly important for organizations deploying applications in a hybrid or multi-cloud environment.

3.5 Used tools

3.5.1 Managing Kubernetes manifests

Figure 3.3: Helm logo

Helm is a package manager for Kubernetes, facilitating the deployment, management, and scaling of applications within a Kubernetes cluster. Helm simplifies complex Kubernetes application deployments by providing a higher level of abstraction through Helm charts. These charts are pre-configured packages of Kubernetes resources.

Key Features of Helm

- **Charts:** Helm uses charts to define, install, and upgrade even the most complex Kubernetes applications. A Helm chart is a collection of files that describe a related set of Kubernetes resources.
- **Releases:** When a chart is deployed, it becomes a release. Helm manages these releases, enabling rollbacks and upgrades.
- **Repositories:** Helm charts can be stored and shared via repositories, making it easy to distribute applications.
- **Templates:** Helm charts support templates, which allow for dynamic and customizable Kubernetes manifests. Templates are written in Go templating syntax.
- **Lifecycle Management:** Helm provides commands to manage the entire lifecycle of applications, from installation and upgrades to rollbacks and deletion.

Benefits of Using Helm

- **Simplified Deployments:** Helm abstracts the complexity of Kubernetes configurations, making deployments more straightforward and repeatable.
- **Version Control:** Helm charts allow you to manage versions of your application, enabling easy rollbacks and upgrades.
- **Sharing and Reusability:** Helm charts can be shared across teams and organizations, promoting reusability and standardization.
- **Scalability:** Helm facilitates the management of applications at scale, handling multiple Kubernetes resources efficiently.

Helm significantly enhances the Kubernetes experience by providing a robust framework for managing the complexities of application deployment and lifecycle management. It empowers developers to create more consistent and reliable deployments while reducing the potential for human error.

3.5.2 Source Code Management

Source Code Management (SCM) concerns how software changes are managed. Its primary goal is to accelerate the delivery of quality code changes by development teams. SCM tools enhance tracking, visibility, collaboration, and control throughout the software development lifecycle, allowing developers working on complex projects to be more creative, have more freedom, and options.

By using SCM, source files are protected against anomalies, and all teams can identify who made which changes and at what stage of the process. This promotes transparency and facilitates collaboration among team members.

We conducted a source code manager study to consolidate the choice of the GitLab tool, which is already used by Orange Business, compared to its competitors BitBucket and GitHub.

Table 3.2 summarizes the comparison of the previously presented source code managers:

We chose GitLab without hesitation due to its open-source nature and free community version. This decision was motivated by the ability to integrate continuous delivery and issue tracking functionalities, which will allow us to expand our tool usage in the future.

Tool	Open Source	License	Version Control	CD integration
	Yes	Free for community version	Git	Already integrated or through other tools
	No	Paid	Git	Integration through other tools or GitHub Actions
	Yes	Free	Git	Integration through other tools or Jira

Table 3.2: Comparison of source code management tools

GitLab Platform

GitLab is now a comprehensive DevOps platform presented as a single application. This platform revolutionizes development, security, operations, and collaboration practices within teams. With GitLab, it is possible to create, test, and deploy software faster, using an integrated solution. GitLab includes the following features:

- **Source Code Management:** Facilitates version control, collaboration, and accelerates the delivery of high-performance software. Enables efficient task coordination, tracking, and verification of changes, as well as simplified code reviews.
- **Agile Project Management:** Improves visibility within the organization, helping to meet set deadlines and budgets. Promotes collaboration among different teams.
- **Continuous Integration and Continuous Delivery (CI/CD):** GitLab's CI/CD feature enables the creation of high-quality applications by automating integration and deployment processes.
- **Cost-Efficient Software Deployment:** GitLab enables faster software releases with reduced development costs.

3.5.3 Monitoring: Prometheus, Grafana

Prometheus

Figure 3.4: Prometheus logo

Prometheus is an open-source monitoring and alerting toolkit designed for reliability and scalability in modern, dynamic environments. It is part of the Cloud Native Computing Foundation (CNCF) and is widely used for monitoring containerized applications and microservices architectures. Prometheus provides a multi-dimensional data model with time series data identified by metric name and key/value pairs. It collects metrics from configured targets at specified intervals, evaluates rule expressions, displays the results, and can trigger alerts if specified conditions are met.

Prometheus Architecture

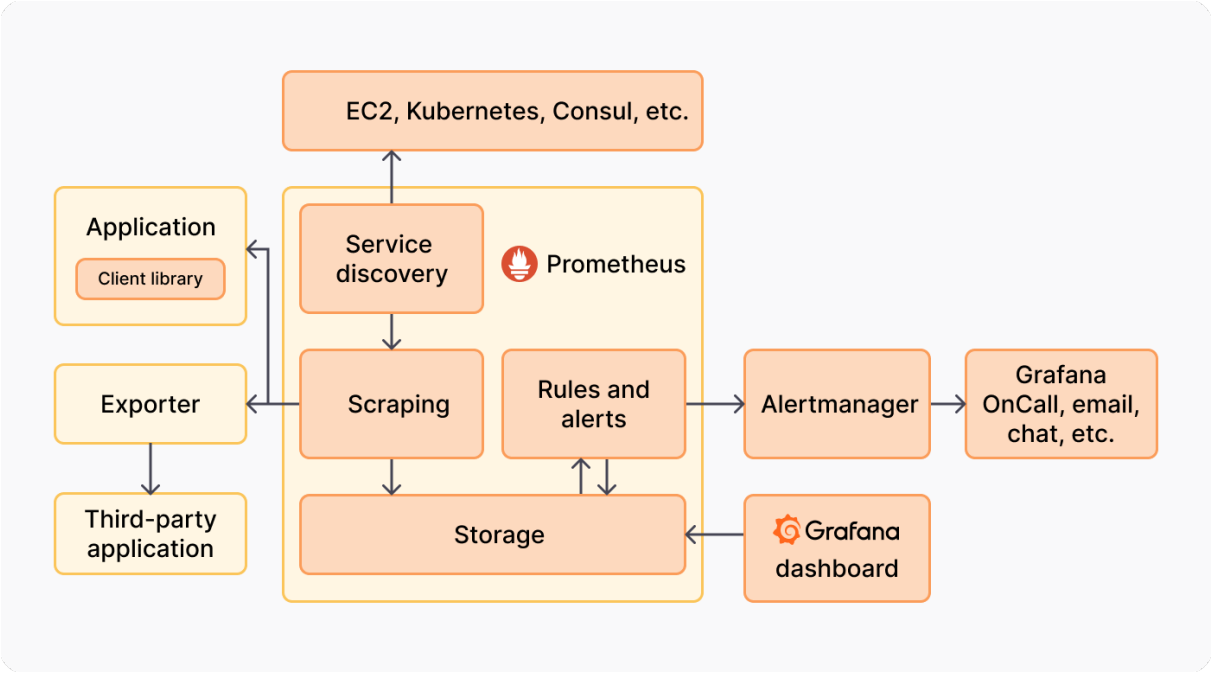

Figure 3.5: Prometheus Architecture

Figure 3.5 illustrates the Prometheus architecture. Prometheus discovers targets to scrape via service discovery, which can include instrumented applications or third-party applications accessed through exporters. Scraped data is stored and can be utilized for dashboards using PromQL or for sending alerts to the Alertmanager, which converts them into various notifications.

- **Client Libraries:** Client libraries allow for easy instrumentation of applications, typically requiring only a few lines of code. Prometheus provides official client libraries in Go, Python, Java/JVM, Ruby, and Rust, with third-party libraries available for other languages. These libraries handle thread safety, bookkeeping, and producing the Prometheus text format. They can automatically pick up metrics from dependencies and offer built-in metrics for CPU usage and garbage collection.
- **Exporters:** Exporters collect metrics from software that cannot be instrumented directly. They transform data from an application's metrics interface into the Prometheus exposition format. Exporters are widely available and can be easily extended if necessary.
- **Service Discovery:** Prometheus uses service discovery to locate and monitor applications and exporters. Integrations are available for Kubernetes, EC2, Consul, and more. Service discovery information is mapped to monitoring targets and their labels using relabeling.
- **Scraping:** Prometheus fetches metrics by sending HTTP requests called scrapes. The responses are parsed and stored. Scraping is configured to occur regularly, typically every 10 to 60 seconds per target.
- **Storage:** Prometheus stores data locally in a custom database, optimized for high performance. The storage system in Prometheus 2.0 can ingest millions of samples per second with efficient compression.
- **Dashboards:** Prometheus provides HTTP APIs for querying and displaying data. Grafana is recommended for creating advanced dashboards, supporting multiple Prometheus servers.
- **Recording Rules and Alerts:** Recording rules allow regular evaluation of PromQL expressions, storing the results. Alerting rules work similarly, generating alerts sent to the Alertmanager.
- **Alert Management:** The Alertmanager handles alerts from Prometheus servers, converting them into notifications via email, chat applications, and services like PagerDuty. Alerts can be aggregated, throttled, and routed based on different team requirements.

- **Long-Term Storage:** Prometheus stores data locally, limiting retention to available disk space. For long-term storage, remote read and write APIs enable integration with external systems for extended data retention.

Grafana

Figure 3.6: Grafana logo

Grafana is an open-source analytics and visualization platform commonly used alongside Prometheus for monitoring and observability purposes. Grafana provides a rich set of features for creating interactive dashboards and visualizations from various data sources, including Prometheus metrics, logs, and other time series databases.

- **Dashboards:** With Grafana, users can easily create and customize dashboards to visualize key performance indicators, monitor system health, and track application metrics in real-time.
- **Visualization Options:** Grafana supports a wide range of visualization options, including graphs, tables, heatmaps, and histograms, allowing for flexible and insightful data analysis.
- **Integration:** Grafana offers extensive integration capabilities with numerous data sources, including Prometheus, Elasticsearch, Graphite, and InfluxDB, making it a versatile tool for consolidating and visualizing data from different sources in a single interface.
- **Advanced Features:** Grafana provides advanced features such as alerting and annotations, allowing users to set up alerts based on predefined thresholds or conditions and annotate dashboards with contextual information for improved situational awareness.

3.5.4 Notifications: Slack

Slack is utilized specifically for receiving alerts and notifications from Prometheus, our monitoring and alerting tool. Through seamless integration, Prometheus sends real-time alerts to designated Slack channels, ensuring timely response to any issues or incidents detected in our systems.

Figure 3.7: Slack Logo

3.6 Conclusion

Throughout this chapter, we have synthesized the selection of various tools necessary for the implementation of our project. By comparing the tools that correspond to our needs, we can detail the phases of implementing these solutions in the next chapter with the goal of achieving our objectives.

Chapter 4

Implementation

4.1 Introduction

The implementation chapter is a crucial step in our project. In this chapter, we will present in detail the various steps we followed to implement our solution. Overall, this implementation chapter will allow us to document in a detailed and transparent manner the process of executing our project. It will highlight the efforts made, the choices made, and the results obtained.

4.2 Configuration & Deployment

4.2.1 Creating a GitLab Template for Kepler

To successfully integrate Kepler with our GitLab CI pipeline and ensure a seamless, efficient, and automated workflow, we followed a series of steps that included configuring the necessary environments, creating precise configuration files, and integrating essential monitoring tools

Configuration of the Environment

The initial step was to set up the execution environment for GitLab CI. We configured the runners, which are the virtual machines or containers responsible for executing the CI jobs. It was crucial to ensure that all dependencies required to run Kepler, Prometheus, and Grafana were installed on these runners.

Creation of the CI Configuration File

We added a `template-ci-kepler.yml` file. This file plays a role as it defines the various stages of the CI pipeline and specifies the actions required to integrate Kepler into our CI process. By using this configuration file, we were able to clearly and systematically outline the tasks necessary for seamless integration into our pipeline.

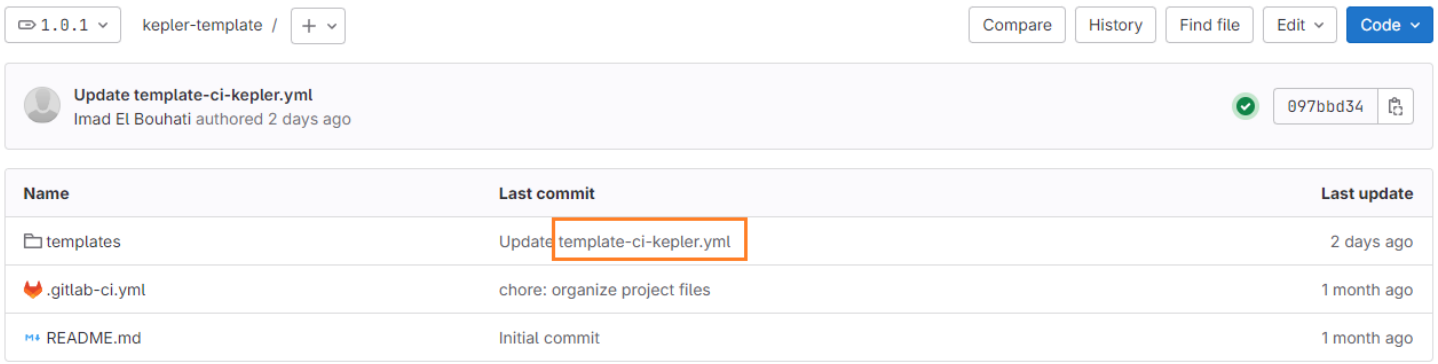

Figure 4.1: Kepler template repository

Integration of the Kepler Template

After creating the template, it can be used and integrated directly into a project. To achieve this, a few lines must be added to the `.gitlab-ci.yml` file.

First, we add the `include` directive followed by the path to our directory containing the Kepler template into an existing pipeline.

Figure 4.2: Integration of the Kepler template

Next, we specify the stage name as `"deploy"`.

Figure 4.3: Stage `"deploy"`

We define the following variables for the deployment:

- `HELM_RELEASE_NAME` as the name of our Helm release.

- `HELM_NAMESPACE` as the namespace for our Helm deployment.
- `KUBECONFIG_CONTENT` to store cluster authentication information for kubectl.

```
deploy-kepler:
  stage: deploy
  extends: .deploy_kepler_chart
  variables:
    HELM_RELEASE_NAME: kepler
    HELM_NAMESPACE: kepler
    KUBECONFIG_CONTENT: $KUBECONFIG_CONTENT
```

Figure 4.4: Deployment variables for Kepler

This configuration allows us to include the steps for deploying Kepler in our CI/CD pipeline using Helm and Kubernetes.

4.2.2 Deploying Prometheus & Grafana on Kubernetes

To deploy Prometheus and Grafana in our Kubernetes cluster as part of the existing CI/CD pipeline, we have followed a structured approach. This ensures that both monitoring and visualization tools are set up correctly and efficiently.

First, we create a new job in the `.gitlab-ci.yml` file named `deploy-prometheus-grafana`. This job will run after the `deploy-kepler` job, ensuring that Kepler is deployed before Prometheus and Grafana.

```
deploy-prometheus-grafana:
  stage: deploy
  image: alpine:3.14
  needs:
    - deploy-kepler
  before_script:
    - echo "$MIRROR/alpine/edge/community" >> /etc/apk/repositories
    - apk update
    - apk add wget helm kubectl
    - mkdir -p /tmp/.kube
    - echo "$KUBECONFIG_CONTENT" | base64 -d > /tmp/.kube/config
    - export KUBECONFIG=/tmp/.kube/config
  script: |
    helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
    helm repo update
    helm install prometheus prometheus-community/kube-prometheus-stack \
        --namespace monitoring \
        --create-namespace \
        --wait

    KPLR_POD=$(
      kubectl get pod \
          -l app.kubernetes.io/name=kepler \
          -o jsonpath="{.items[0].metadata.name}" \
          -n kepler
    )
    kubectl wait --for=condition=Ready pod $KPLR_POD --timeout=-1s -n kepler

    wget -O kepler_dashboard.json https://raw.githubusercontent.com/sustainable-computing-io/kepler/main/grafana-dashboards/Kepler-Exporter.json

    GF_POD=$(
      kubectl get pod \
          -n monitoring \
          -l app.kubernetes.io/name=grafana \
          -o jsonpath="{.items[0].metadata.name}"
    )
    kubectl cp kepler_dashboard.json monitoring/$GF_POD:/tmp/dashboards/kepler_dashboard.json
```

Figure 4.5: Job `deploy-prometheus-grafana`

Next, we define the necessary configurations and scripts for deploying Prometheus and Grafana. This script accomplishes the following:

- Adds the Prometheus Helm chart repository and updates Helm repositories.
- Installs the Prometheus stack in the `monitoring` namespace.
- Waits for the Kepler pod to be ready.
- Downloads the Kepler Grafana dashboard JSON file.
- Finds the Grafana pod and copies the Kepler dashboard JSON file to it.

4.2.3 Configuring Rules and Alerts in Prometheus

In this section, we will cover the steps involved in configuring Prometheus to monitor specific metrics and set up alerting rules to notify us of any anomalies. Prometheus allows us to define custom rules for generating alerts based on our metrics, which can be integrated with various notification channels like Slack.

4.2.3.1 Creating Custom Alerting Rules

To monitor the energy consumption of our nodes, we have defined several custom alerting rules. These rules are specified in the `additionalPrometheusRulesMap` section of our Prometheus configuration. Here is an example configuration:

```yaml
additionalPrometheusRulesMap:
  custom-cpu-rule:
    groups:
      - name: total-nodes-energy
        rules:
          - alert: NodesEnergy
            expr: sum(irate(kepler_container_package_joules_total{container_namespace=~".*", pod_name=~".*"}[1d])) > 1
            for: 1h
            labels:
              severity: warning
            annotations:
              summary: "High Energy Consumption"
              description: "Energy consumption has exceeded the threshold for more than one hour."
              energy_consumption: "{{ $value }} watt"
          - alert: NodesEnergyCritical
            expr: sum(irate(kepler_container_package_joules_total{container_namespace=~".*", pod_name=~".*"}[1d])) > 4
            for: 6h
            labels:
              severity: critical
            annotations:
              summary: "Critical Energy Consumption"
              description: "Energy consumption has reached dangerously high levels, indicating a potential system overload. Immediate action is re
              energy_consumption: "{{ $value }} watt"
          - alert: NodesEnergyInfo
            expr: sum(irate(kepler_container_package_joules_total{container_namespace=~".*", pod_name=~".*"}[1d])) > 0.5
            for: 12h
            labels:
              severity: info
            annotations:
              summary: "Informational Energy Consumption"
              description: "Energy consumption is above normal levels for more than six hours."
              energy_consumption: "{{ $value }} watt"
```

Figure 4.6: Prometheus rules

This configuration defines three alerts:

- **NodesEnergy**: Triggered when the energy consumption exceeds 1 watt for more than 1 hour. It has a `severity` label set to `warning`.
- **NodesEnergyCritical**: Triggered when the energy consumption exceeds 4 watts for more than 6 hours. It has a `severity` label set to `critical`.
- **NodesEnergyInfo**: Triggered when the energy consumption exceeds 0.5 watts for more than 12 hours. It has a `severity` label set to `info`.

Each alert includes annotations providing a summary and description of the alert, along with the actual energy consumption value.

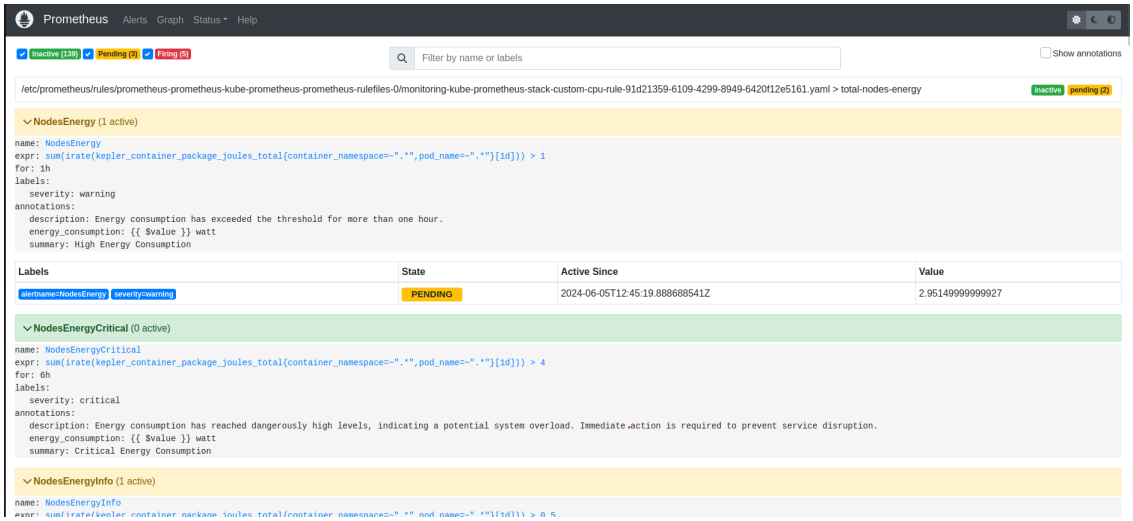

Figure 4.7: Alerts dashboard in Prometheus

4.2.3.2 Integrating Alerts with Slack

To ensure that we receive notifications for these alerts, we integrate Prometheus with Slack using Alertmanager. The following configuration sets up Alertmanager to send alerts to a Slack channel:

```yaml
## Configuration for alertmanager
alertmanager:
  config:
    global:
      resolve_timeout: 5m
    route:
      group_by: ['job']
      group_wait: 30s
      group_interval: 1m
      repeat_interval: 1m
      receiver: 'null'
      routes:
        - match:
            alertname: NodesEnergy
          receiver: 'slack'
          repeat_interval: 1m
        - match:
            alertname: NodesEnergyCritical
          receiver: 'slack'
          repeat_interval: 2m
        - match:
            alertname: NodesEnergyInfo
          receiver: 'slack'
          repeat_interval: 3m
    inhibit_rules:
      - target_match_re:
          alertname: '.*Overcommit'
        source_match:
          alertname: 'Watchdog'
        equal: ['prometheus']
    receivers:
      - name: 'null'
      - name: 'slack'
        slack_configs:
          - api_url: 'https://hooks[redacted]Uo5'
            send_resolved: true
            channel: '#alerts-monitoring'
            title: '[{{ .Status | toUpper }}{{ if eq .Status "firing" }}:{{ .Alerts.Firing | len }}{{ end }}] Monitoring Event Notification'
            text: |-
              {{ range .Alerts }}
                *Alert:* {{ .Labels.alertname }} - `{{ .Labels.severity }}`
                *Description:* {{ .Annotations.description }}
                *Energy Consumption:* {{ .Annotations.energy_consumption }}
                *Details:*
                {{ range .Labels.SortedPairs }} • *{{ .Name }}:* `{{ .Value }}`
                {{ end }}
              {{ end }}
```

Figure 4.8: Alert Manager Configuration

This configuration does the following:

- Sets global configuration options such as `resolve_timeout`.
- Defines routing rules to determine which alerts should be sent to Slack. Alerts named `NodesEnergy`, `NodesEnergyCritical`, and `NodesEnergyInfo` are routed to the Slack receiver.
- Sets up inhibition rules to prevent redundant alerts.
- Configures the Slack receiver with a webhook URL, ensuring that alerts are sent to the `#alerts-monitoring` channel. The `title` and `text` fields format the alert message.

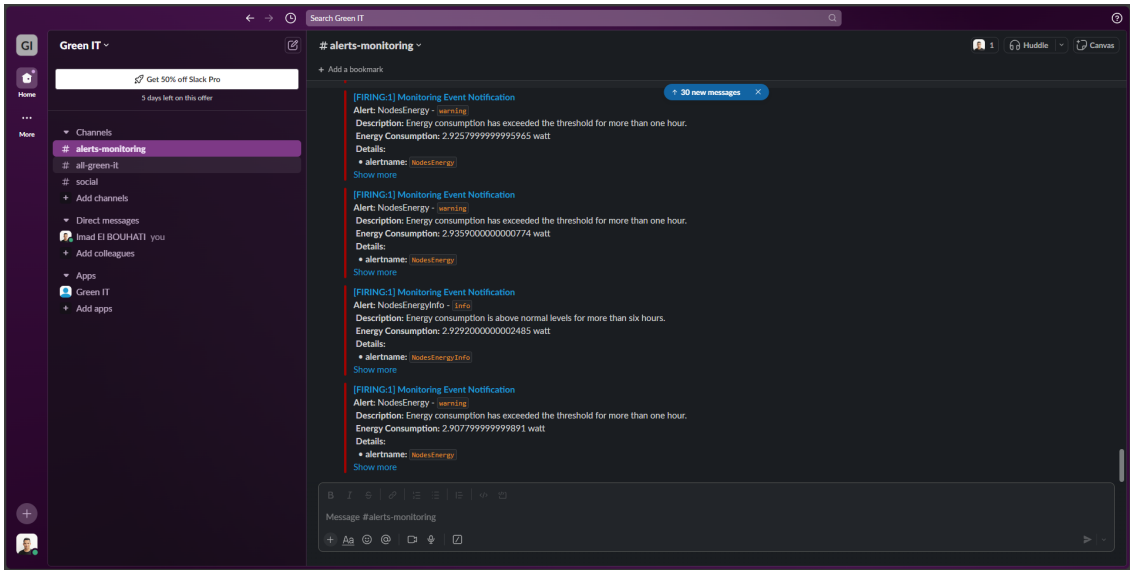

Figure 4.9: Slack alerts messages

4.3 Launching the CI/CD Pipeline

In this section, we will discuss the process of launching our CI/CD pipeline, showcasing the various stages and the resources deployed in our Kubernetes cluster. Finally, we will present the Grafana dashboard to visualize the metrics collected by Prometheus.

4.3.1 Starting the GitLab Pipeline

To start the pipeline, we push our code changes to the GitLab repository. This triggers the GitLab CI/CD pipeline, which executes the stages defined in the `.gitlab-ci.yml` file. The pipeline consists of multiple stages, including building, testing, and deploying our applications.

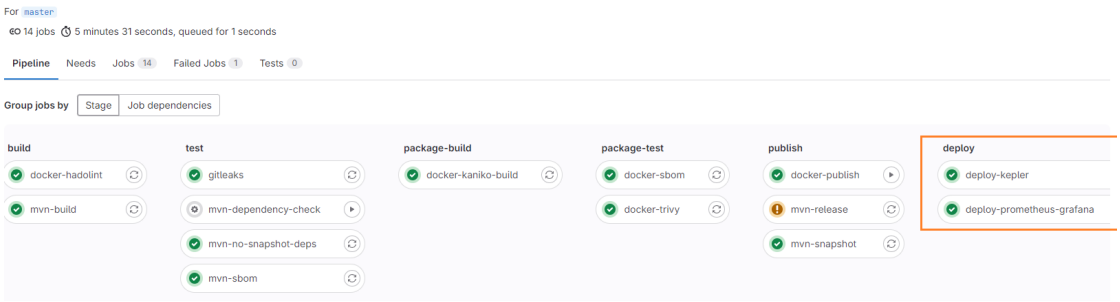

Figure 4.10: GitLab CI/CD Pipeline Stages

4.3.2 Deployed Resources in Kubernetes

Once the pipeline completes successfully, the following resources were deployed in the ‘kepler‘ and ‘monitoring‘ namespaces within our Kubernetes cluster.

4.3.2.1 Kepler Namespace

The ‘kepler‘ namespace contains the resources related to the Kepler application. These resources include:

- **Pod**: The running instance of the Kepler application.
- **Service**: Provides a stable IP and DNS name for accessing the Kepler application.
- **DaemonSet**: Ensures that a copy of the Kepler application runs on all nodes.

```
imad@ubuntu:~$ kubectl get all -n kepler
NAME                READY   STATUS    RESTARTS   AGE
pod/kepler-c5g6t    1/1     Running   0          4d3h

NAME             TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/kepler   ClusterIP   10.245.240.57   <none>        9102/TCP   4d3h

NAME                     DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
daemonset.apps/kepler    1         1         1       1            1           kubernetes.io/os=linux   4d3h
imad@ubuntu:~$
```

Figure 4.11: Deployed resources in the ‘kepler‘ namespace.

4.3.2.2 Monitoring Namespace

The ‘monitoring‘ namespace contains the resources for monitoring and alerting. These resources include:

- **Prometheus**: Collects and stores metrics data.
- **Alertmanager**: Manages alerts sent by Prometheus.

- **Grafana**: Visualizes the collected metrics.
- **Services**: Provide stable IPs and DNS names for accessing Prometheus, Alertmanager, and Grafana.
- **DaemonSet and Deployments**: Ensure that Prometheus and its components are running correctly.

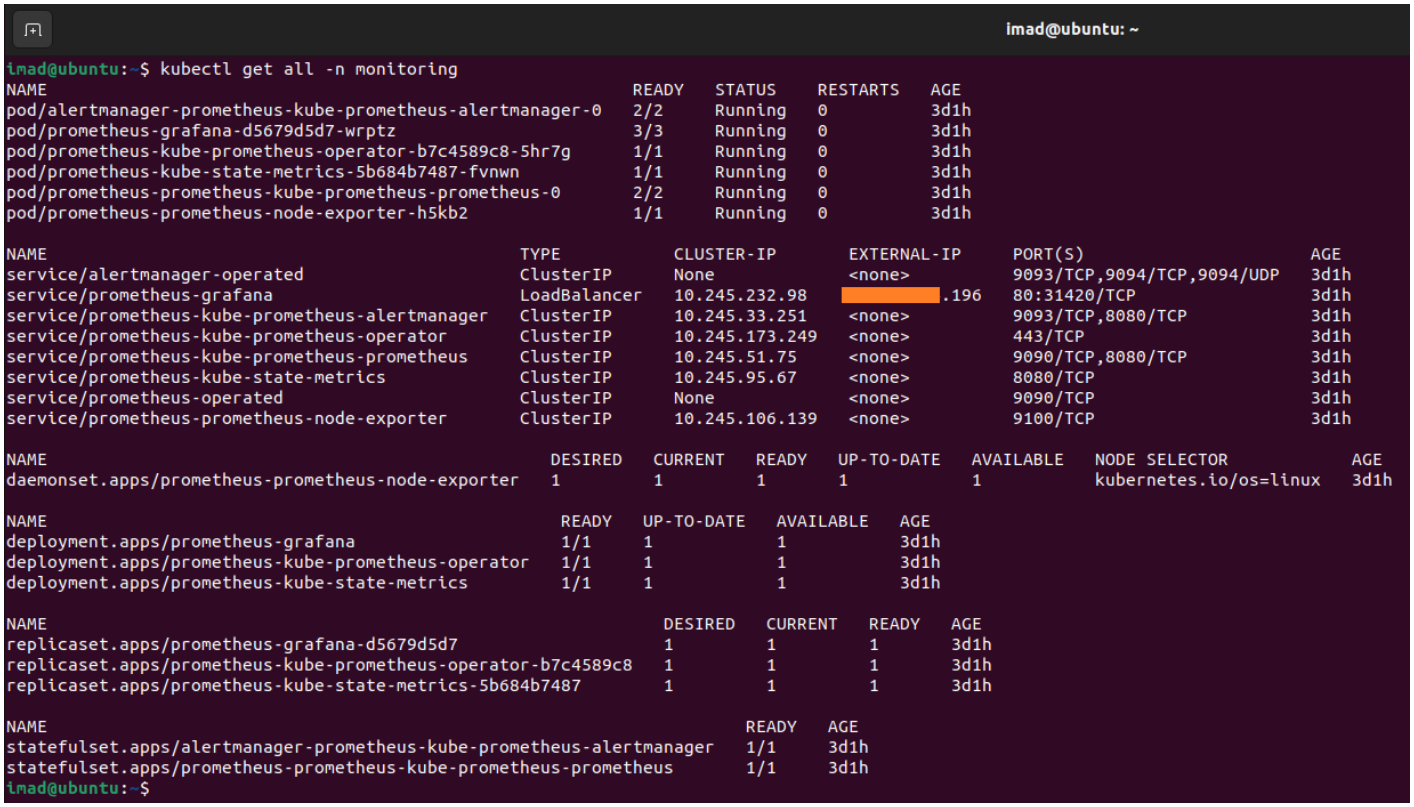

Figure 4.12: Deployed resources in the ‘monitoring‘ namespace.

4.3.3 Grafana Dashboards

We utilize Grafana dashboards to monitor the performance and resource utilization of our Kubernetes cluster. Below are the key metrics visualized:

4.3.3.1 Carbon Emissions Dashboard

Figure 4.13: Grafana dashboard showing carbon emissions based on energy consumption.

The above figure (Figure 4.13) shows the carbon emissions associated with the energy consumption in the cluster. The dashboard highlights the carbon footprint in terms of CO_2 `Coal`, CO_2 `Petroleum`, and CO_2 `Natural Gas`. This information is crucial for understanding the environmental impact of our infrastructure and taking steps towards sustainability.

4.3.3.2 Power Consumption Dashboard

Figure 4.14: Grafana dashboard showing total power consumption by namespace.

The above figure (Figure 4.14) displays the total power consumption (in watts) across all namespaces in the cluster. The power consumption is broken down into different components: `PKG` (Package), `DRAM` (Memory), and `OTHER`. The dashboard provides insights into the energy usage trends over time, which helps in optimizing resource allocation and identifying potential inefficiencies.

4.4 Conclusion

Integrating our CI/CD pipeline with Kubernetes and monitoring tools has enabled efficient deployment and management of large-scale resources. The `kepler` and `monitoring` namespaces hosted essential pods, services, and daemons, ensuring high availability and simplified maintenance. The Grafana dashboards provide detailed views of power consumption and CO_2 emissions, facilitating proactive and sustainable infrastructure management. This approach not only optimized performance but also helped minimize the environmental impact of our operations. Future improvements could include integrating additional monitoring features and extending our CI/CD pipeline with complementary tools to enhance the robustness and security of our Kubernetes ecosystem.

General Conclusion and Perspectives

In conclusion, the project focused on collecting and estimating power consumption metrics in cloud environments using Kepler to monitor and optimize energy usage. Kepler proved effective in bare-metal environments by directly collecting real-time power metrics and utilizing the Ratio Power model for accurate process-level power consumption estimation. In virtual machine (VM) environments, where direct measurement is challenging, Kepler employs trained power models to estimate energy consumption, ensuring continuous and reliable monitoring.

Integrating Kepler into our development workflow has enhanced our understanding and management of energy consumption, leading to resource optimization and reduced energy costs. This approach also promotes a more sustainable and environmentally friendly management of cloud infrastructures.

Looking ahead, we plan to further explore the advanced features of Kepler, particularly improving power models for VM environments and integrating new technologies for better accuracy and efficiency. We also aim to expand our CI/CD pipeline to include additional monitoring and optimization tools, thereby enhancing our ability to detect and address energy inefficiencies proactively.

Additionally, we seek to strengthen collaboration between development, infrastructure and Green IT teams to foster an integrated and proactive approach to energy consumption management throughout the application lifecycle. This synergy between teams will contribute to the continuous improvement of our practices and the achievement of our sustainability and energy efficiency goals.

Bibliography

[1] *Julien Pivotto & Brian Brazil, Prometheus: Up & Running, 2nd Edition*, **Book**

[2] *How Do You Go from DevOps to DevGreenOps?* **Greenspector**

[3] *Green DevOps: Poursuivre l'excellence opérationnelle tout en réduisant l'impact environnemental*, **Medium article by glalloue**

[4] *DevOps Monitoring*, **Splunk Blog**

[5] *Docker Architecture Overview*, **Docker Documentation**

[6] *What is DevSecOps?*, **Red Hat**

[7] *Exploring Kepler's Potentials: Unveiling Cloud Application Power Consumption*, **CNCF Blog**

[8] *Kepler Helm Chart Documentation*, **GitHub**

[9] *Orange Business Services*, **Orange Business**

[10] *Prometheus Documentation*, **Prometheus.io**

[11] *Slack API: Webhooks*, **Slack API Documentation**