



MINISTRY OF HIGHER EDUCATION  
AND SCIENTIFIC RESEARCH  
UNIVERSITY OF MANNOUBA  
NATIONAL SCHOOL OF COMPUTER SCIENCE

## Internship Report



---

**Subject: Automation of CI/CD  
Infrastructure Setup**

---

Prepared by.....Omar MENSI

Supervised by ..... Hassen ZAOUI

**Academic Year: 2023-2024**

# Acknowledgement

First and foremost, I would like to express my deepest gratitude to God for blessing me with the opportunity and strength to complete this project. Without His guidance and support, none of this would have been possible.

I am profoundly grateful to my family, whose unwavering love and encouragement have been my constant source of motivation. I owe a special thanks to my parents for their endless support, understanding, and patience throughout this journey. Their belief in me has been the driving force behind my achievements.

I would also like to extend my heartfelt thanks to my supervisor at Sofrecom, Mr. Hassen Zaoui, for his invaluable guidance, insights, and constant support throughout this project. His expertise and mentorship have been instrumental in shaping the direction of my work.

Furthermore, I am thankful to the hosting company for providing all the necessary resources and creating an ideal working environment that enabled me to focus and achieve my objectives effectively.

Lastly, I would like to thank my friends for their assistance, encouragement, and companionship. Their support has been vital in overcoming challenges and staying motivated.

Thank you all for your contributions to this project and for being a part of this journey with me.

# Contents

<b>List of Figures</b>	<b>iv</b>
<b>1 Project Context</b>	<b>2</b>
1.1 Introduction . . . . .	2
1.2 Scope of the project . . . . .	2
1.3 Hosting Company . . . . .	3
1.4 Problem Statement . . . . .	3
1.5 Proposed Solution . . . . .	3
1.6 Conclusion . . . . .	4
<b>2 Key Concepts and Technologies</b>	<b>5</b>
2.1 Introduction . . . . .	5
2.2 CI/CD . . . . .	5
2.2.1 CI/CD pipelines . . . . .	5
2.3 Infrastructure as a code . . . . .	8
2.3.1 Ansible . . . . .	8
2.3.2 AWX . . . . .	9
2.4 Virtualisation and Orchestration . . . . .	9
2.4.1 Docker and Docker compose . . . . .	9
2.4.2 Kubernetes . . . . .	10
2.5 Registries . . . . .	10
2.6 Security and Quality Assurance . . . . .	10
2.6.1 Security Scanning and Compliance . . . . .	10
2.6.2 Quality Assurance and Code Analysis . . . . .	11
2.7 Conclusion . . . . .	11
<b>3 Requirements analysis and Design</b>	<b>12</b>
3.1 Introduction . . . . .	12
3.2 Requirements analysis . . . . .	12
3.2.1 Actors . . . . .	12
3.2.2 Functional requirements . . . . .	13

3.2.3	Non functional requirements . . . . .	14
3.3	Solution Design . . . . .	15
3.3.1	Global architecture . . . . .	15
3.3.2	AWX architecture . . . . .	17
3.3.3	CI/CD infrastructure architecture . . . . .	21
3.3.4	CI/CD pipeline . . . . .	23
3.4	Conclusion . . . . .	25
<b>4</b>	<b>Implementation and Deployment</b>	<b>26</b>
4.1	Introduction . . . . .	26
4.2	Development and Deployment Environment . . . . .	26
4.2.1	Hardware Environment . . . . .	26
4.2.2	Software Environment . . . . .	27
4.3	Implementation . . . . .	27
4.3.1	Development of Ansible Playbooks . . . . .	27
4.3.2	Setting up Kubernetes Cluster and Deploying AWX . . . . .	28
4.3.3	Setting up the Project and Installing the Infrastructure . . . . .	31
4.3.4	Creating the CI/CD Pipeline . . . . .	32
4.4	Conclusion . . . . .	33
	<b>Bibliography</b>	<b>35</b>

# List of Figures

2.1	CI/CD pipeline . . . . .	6
2.2	Ansible architecture . . . . .	9
3.1	Global architecture . . . . .	15
3.2	AWX deployment architecture . . . . .	18
3.3	AWX internal architecture . . . . .	20
3.4	Gitlab deployment architecture . . . . .	22
3.5	Jenkins deployment architecture . . . . .	23
3.6	CI/CD pipeline . . . . .	24
4.1	AWX deployment screenshot . . . . .	28
4.2	Monitoring namespace screenshot . . . . .	29
4.3	Grafana AWX dashboard . . . . .	30
4.4	AWX templates . . . . .	31
4.5	Job execution exemple . . . . .	32
4.6	Jenkins CI/CD pipeline . . . . .	32

# Introduction

In today's highly competitive and fast-paced digital landscape, companies are increasingly adopting DevOps methodologies and best practices to deliver high-quality software products with minimal time to market. This trend is driven by the need to outpace competitors, respond swiftly to market changes, and continuously deliver value to customers.

However, to fully embrace DevOps methodologies, companies must prioritize automation across all aspects of their software development lifecycle. Automation not only streamlines repetitive tasks but also reduces human error, enhances consistency, and enables teams to focus on innovation rather than mundane tasks. As a result, various tools and technologies have emerged to facilitate this automation, from version control systems to build and deployment pipelines, security scanning, and monitoring tools.

For organizations managing multiple projects or frequently initiating new projects and environments, the manual setup of DevOps CI/CD tools can become cumbersome and error-prone. This challenge underscores the need for an automated solution that can streamline the deployment and configuration of DevOps tools.

This project aims to address this need by developing a comprehensive automation solution for setting up CI/CD infrastructure. The objective is to create a solution that enables companies to quickly and reliably establish DevOps environments tailored to their specific requirements, thereby enhancing productivity, reducing time to market, and maintaining a competitive edge in the industry.

# Chapter 1

## Project Context

### 1.1 Introduction

This chapter provides a comprehensive overview of the environment and circumstances in which the project is being developed. It explores the need for automation in setting up CI/CD infrastructure within a rapidly evolving DevOps landscape. The chapter includes the project's introduction, defining the scope and objectives, and discusses the background of the hosting company, Sofrecom . The problem statement identifies the challenges faced by organizations in setting up DevOps environments, and the proposed solution being developed to address these challenges.

### 1.2 Scope of the project

The scope of this project focuses on automating the infrastructure setup and installation of CI/CD tools, providing a platform that enables developers and administrators to initiate a process to set up a specific tool in a specific environment. This automation is critical for reducing manual intervention and ensuring consistency across environments. Additionally, the project involves developing a full-stack CI/CD pipeline designed to experiment with the automatically configured environments and tools, ensuring they are fully functional and meet the organization's needs. This project aims to streamline DevOps practices and enhance operational efficiency.

## 1.3 Hosting Company

Sofrecom, a subsidiary of the Orange Group, operates in the Tunis branch and specializes in providing consulting and engineering services in the telecommunications industry. The company is recognized for its expertise in integrating complex solutions and offers a collaborative environment that fosters innovation and digital transformation. Within Sofrecom, the integration department plays a pivotal role in deploying and managing various IT and telecommunication solutions for clients, ensuring seamless integration with existing systems. As part of this department, the project benefits from access to a wide range of resources, including expert knowledge, advanced technologies, and a strong focus on continuous improvement. This environment provides the perfect backdrop for developing and testing an automated CI/CD infrastructure setup, aligning with Sofrecom's commitment to innovation and excellence.

## 1.4 Problem Statement

Many organizations, especially those managing multiple projects or frequently launching new projects and environments, face significant challenges in manually setting up DevOps CI/CD tools. This process is often time-consuming, prone to errors, and lacks consistency, which can hinder the overall efficiency of development and operations teams. The manual approach does not scale well with increasing demand, leading to delays and potential misconfigurations that affect software quality and delivery speed. Thus, there is a pressing need for a solution that automates the setup and configuration of DevOps environments, ensuring a standardized and reliable infrastructure that supports continuous integration and delivery processes.

## 1.5 Proposed Solution

The proposed solution for this project involves creating a series of Ansible playbooks to automate the setup of environments on hosting machines and the installation of various CI/CD tools. These playbooks are designed to be versatile and cover the installation of multiple tools, allowing for flexible combinations based on the specific requirements of different projects.

The process begins with the development of Ansible playbooks that facilitate the deployment of essential CI/CD tools on designated environments. These playbooks will be capable of installing a range of tools, such as GitLab, Jenkins, SonarQube, and others, tailored to the needs of the development and operations teams. The flexibility of these playbooks ensures that they can be adapted to



different environments and tool combinations, providing a robust solution for managing CI/CD infrastructure.

To enhance usability and accessibility, the solution also involves deploying the AWX platform, an open-source robust web application that provides a user interface, REST API, and task engine for Ansible. By integrating the playbooks into AWX, developers and administrators can easily execute these playbooks to set up environments and install necessary tools without requiring deep knowledge of Ansible. This approach ensures that the process of setting up and managing CI/CD tools is streamlined, efficient, and user-friendly.

Finally, to validate the effectiveness of the automated setup and the proper installation of tools, the project includes creating a full CI/CD pipeline for a sample project. This pipeline will be used to experiment with and test the various tools deployed through the automated process, ensuring that they are correctly configured and fully operational. This end-to-end testing will help verify the reliability and functionality of the automated CI/CD environment, ensuring it meets the desired quality standards and operational requirements.

By automating the setup and deployment processes and providing a platform for easy management, this solution aims to significantly reduce the time and effort required to establish and maintain CI/CD infrastructure, thereby enhancing productivity and fostering a more agile development environment.

## 1.6 Conclusion

This chapter has established the context for the project's aim to automate CI/CD infrastructure setup, highlighting the need for streamlined processes in a DevOps-driven environment. By addressing the challenges faced by organizations, this project seeks to enhance operational efficiency and support continuous integration and delivery efforts. The following chapter will delve into the key concepts and technologies utilized in achieving these goals.

# Chapter 2

## Key Concepts and Technologies

### 2.1 Introduction

This chapter explores the fundamental concepts and technologies that form the backbone of the project. Understanding these concepts is crucial for grasping the project's scope and implementation. The chapter delves into continuous integration and continuous deployment (CI/CD) methodologies, infrastructure as code, virtualization, orchestration, registries, and security and quality assurance practices. Each section provides an overview of the tools and technologies used, explaining their relevance and application in the context of this project.

### 2.2 CI/CD

Continuous Integration (CI) and Continuous Deployment (CD) are essential practices in modern software development that help teams deliver high-quality software faster and more reliably. CI/CD pipelines automate the processes of integrating code changes, running tests, and deploying applications, reducing manual effort and minimizing errors.

#### 2.2.1 CI/CD pipelines

A CI/CD pipeline is a sequence of automated processes that software undergoes from development to deployment. It facilitates the continuous integration of code changes, ensuring that new code is seamlessly integrated, tested, and delivered to production with minimal manual intervention. This process is crucial for maintaining high code quality and accelerating the delivery of software. The typical CI/CD pipeline consists of several key stages, each serving a specific purpose in the software delivery lifecycle:

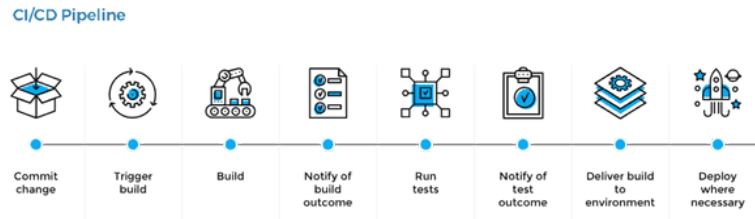


Figure 2.1: CI/CD pipeline  
[1]

1. **Code Commit:** The pipeline begins with the code commit phase, where developers submit their changes to the version control system (VCS). This action triggers the pipeline to start the automated process.
2. **Build:** In the build stage, the committed code is compiled into executable artifacts. This stage often involves converting source code into binary files, creating application packages, or generating other deployable outputs. The build process ensures that the code is correctly compiled and prepared for the next stages.
3. **Unit Testing:** Once the build is complete, automated unit tests are executed to verify the functionality of individual components or modules. Unit testing helps identify bugs early in the development process by testing small, isolated units of code.
4. **Integration Testing:** Following unit testing, integration testing checks the interaction between different components or services. This stage ensures that the integrated modules work together as expected and that the system as a whole functions correctly.
5. **Code Analysis:** Code analysis involves static code analysis tools that review the code for potential issues such as code smells, vulnerabilities, and adherence to coding standards.
6. **Security Scanning:** Security scanning tools are used to identify vulnerabilities in the code or container images. This step is critical for detecting potential security risks before the code reaches production.
7. **Packaging:** After successful testing and analysis, the application is packaged into deployable artifacts, such as Docker images or deployment bundles. This packaging step prepares the software for deployment to different environments.

8. **Pushing artifacts or images to a registry:** In this stage, the packaged artifacts or Docker images are pushed to a container registry or artifact repository. This registry serves as a centralized location for storing and managing these deployable components. Pushing artifacts to a registry ensures that they are easily accessible for deployment in various environments and facilitates version control and traceability.
9. **Deployment:** The deployment phase involves deploying the packaged application from the registry to a staging or production environment. This step can be automated to ensure that deployments are consistent and repeatable.
10. **Monitoring and Logging:** Once the application is deployed, monitoring and logging are essential for tracking its performance and behavior in the production environment. Continuous monitoring helps detect issues in real-time and provides insights into the application's health and performance.

By automating each of these stages, CI/CD pipelines enhance the efficiency, consistency, and reliability of the software delivery process. They enable teams to detect and address issues early, reduce manual effort, and accelerate the time to market for new features and improvements. Many tools are present on the market these days to achieve this automation in the scope on this project we will cover two of them: jenkins and gitlab CI/CD

## Jenkins

Jenkins is an open-source automation server that facilitates continuous integration and delivery. It enables developers to automate the building, testing, and deployment of applications, making it easier to integrate changes to the project. Jenkins is highly extensible, with numerous plugins available to support various stages of the CI/CD pipeline, including integration with source control, building environments, and deployment tools.

## Gitlab CI/CD

GitLab CI/CD is a built-in tool provided by GitLab, a web-based DevOps lifecycle tool. GitLab CI/CD integrates seamlessly with GitLab repositories, enabling automatic testing and deployment of code changes. It offers a robust set of features, including pipeline visualization, integrated security scanning, and support for various deployment environments, making it a popular choice for DevOps teams.

## 2.3 Infrastructure as a code

Infrastructure as Code (IaC) is a key DevOps practice that involves managing and provisioning computing infrastructure through machine-readable configuration files, rather than physical hardware configuration or interactive configuration tools. IaC allows for consistent, repeatable, and automated deployment of infrastructure.

### 2.3.1 Ansible

Ansible is a powerful, open-source automation tool designed to simplify the management of complex IT systems and workflows. It is widely used for configuration management, application deployment, and task automation across various environments. One of Ansible's key strengths is its simplicity and ease of use. Unlike other automation tools, Ansible does not require an agent to be installed on the target systems. Instead, it operates over SSH, allowing it to manage remote systems efficiently without the need for additional software on the client side.

Ansible utilizes YAML (Yet Another Markup Language) to define automation tasks in a human-readable format. This approach makes it easier for users to write and understand automation scripts, known as playbooks. Playbooks are a central component of Ansible, allowing users to define a series of steps and tasks that are executed in a specific order. These tasks can include installing software, configuring services, and managing files, among other operations. The modular nature of Ansible's playbooks ensures that they can be reused and adapted for different scenarios, enhancing the flexibility of automation processes.

In addition to playbooks, Ansible supports roles and collections. Roles are a way to organize playbooks into reusable units, allowing users to encapsulate and manage related tasks, variables, and files. Collections, on the other hand, are a distribution format for roles, modules, and plugins, providing a way to share and distribute automation content across different environments. Ansible's architecture is designed to be both scalable and efficient. It operates using a master-slave model where the master node, or control node, executes tasks and communicates with managed nodes, or target systems. This design minimizes the load on the control node while ensuring that tasks are executed reliably across multiple managed nodes. The idempotent nature of Ansible ensures that tasks are executed in a manner that achieves the desired state without unintended side effects, making it a reliable choice for automation.

Ansible also integrates seamlessly with other tools and platforms, such as cloud providers, container orchestrators, and continuous integration/continuous deployment (CI/CD) systems. This integration capability allows users to automate the deployment and management of infrastructure and applications in diverse

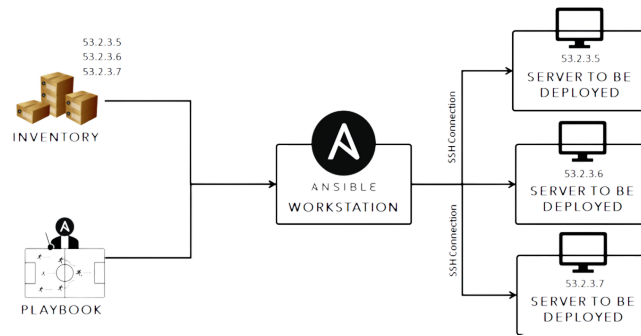


Figure 2.2: Ansible architecture  
[2]

environments, from traditional data centers to modern cloud environments.

Overall, Ansible’s ease of use, flexibility, and broad integration capabilities make it a valuable tool for automating IT operations, streamlining workflows, and ensuring consistent and reliable configuration management across various systems.

### 2.3.2 AWX

AWX is the open-source project from which Red Hat Ansible Tower is derived. It provides a web-based user interface, REST API, and task engine for managing Ansible playbooks. By leveraging AWX, organizations can centralize their automation activities, streamline workflow orchestration, and enhance collaboration among teams. AWX makes Ansible more accessible by providing a visual dashboard, role-based access control, and auditing capabilities.

## 2.4 Virtualisation and Orchestration

Virtualization and orchestration technologies enable efficient resource utilization and management of applications across multiple environments. They provide the foundation for containerization and scalable, resilient application deployment.

### 2.4.1 Docker and Docker compose

Docker is a platform that enables developers to create, deploy, and run applications in containers—lightweight, stand-alone, and executable packages that include everything needed to run the application, including the code, runtime, libraries, and system settings. Docker Compose is a tool for defining and running multi-container Docker applications. It uses a YAML file to configure the application’s services, making it easier to manage and orchestrate complex environments.

### **2.4.2 Kubernetes**

Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It provides a robust framework for running distributed systems resiliently, managing container lifecycles, networking, storage, and security policies. Kubernetes is designed to handle complex, multi-container workloads, making it ideal for large-scale deployments.

## **2.5 Registries**

Container registries are repositories that store and manage container images. They are essential for distributing images within an organization and across multiple environments.

### **Harbor registry**

Harbor is an open-source container image registry that provides security, identity, and management capabilities for Docker and Helm charts. Harbor extends the open-source Docker distribution by adding features such as role-based access control, vulnerability scanning, and image signing, making it an ideal solution for enterprises looking to manage container images securely and efficiently.

## **2.6 Security and Quality Assurance**

Security and quality assurance are critical components of the software development lifecycle, ensuring that code is both secure and meets the required standards of quality.

### **2.6.1 Security Scanning and Compliance**

Security scanning tools help detect vulnerabilities and compliance issues in code and container images, allowing teams to address security risks before they reach production.

#### **Trivy**

Trivy is a comprehensive, easy-to-use open-source vulnerability scanner for container images, file systems, and Git repositories. It identifies vulnerabilities in operating system packages and application dependencies, providing organizations

with detailed reports on potential security risks. Trivy integrates seamlessly with CI/CD pipelines, enabling continuous security checks throughout the software development lifecycle.

### **2.6.2 Quality Assurance and Code Analysis**

Quality assurance and code analysis tools help ensure that code adheres to best practices and organizational standards, reducing the likelihood of defects and improving maintainability.

#### **SonarQube**

SonarQube is an open-source platform for continuous inspection of code quality. It performs static code analysis to detect bugs, code smells, and security vulnerabilities in over 25 programming languages. SonarQube integrates with various CI/CD tools, providing real-time feedback to developers, and helping maintain high standards of code quality throughout the development process.

## **2.7 Conclusion**

This chapter provided an overview of the essential concepts and technologies used in DevOps, including CI/CD pipelines, Infrastructure as Code, containerization, orchestration, registries, and security and quality assurance tools. Understanding these technologies is crucial for implementing a robust, automated CI/CD infrastructure. In the next chapter, we will focus on the requirements analysis and design needed to effectively automate this setup.



# Chapter 3

## Requirements analysis and Design

### 3.1 Introduction

This chapter outlines the requirements analysis and design process for automating the CI/CD infrastructure setup. It begins by identifying the functional and non-functional requirements necessary for the project to meet organizational needs. The chapter then presents the solution design, detailing the global architecture, AWX deployment, CI/CD infrastructure architecture, and the design of the CI/CD pipeline. These elements form the foundation for building a scalable, efficient, and automated CI/CD environment.

### 3.2 Requirements analysis

The purpose of the requirements analysis is to define the necessary criteria and conditions needed to implement an effective automated CI/CD infrastructure. This analysis includes identifying the key actors involved, outlining the functional and non-functional requirements, and ensuring that the system meets the needs of both administrators and developers while supporting organizational objectives for security, scalability, and usability.

#### 3.2.1 Actors

- **Administrator:** The administrator is responsible for managing the CI/CD infrastructure, including the setup, configuration, and maintenance of CI/CD tools. Administrators are tasked with ensuring that the system runs smoothly, securely, and efficiently. They have elevated permissions to modify system settings, manage user access, and perform updates. Administrators also

oversee the integration of various tools and services within the CI/CD pipeline to ensure a seamless and cohesive workflow.

- **Developer:** Developers are the primary users of CI/CD tools. They rely on these tools to automate the build, test, and deployment processes for their software projects. Developers need easy access to these tools and the ability to trigger builds, tests, and deployments without extensive manual intervention. They benefit from a CI/CD environment that is intuitive, responsive, and supports their need for rapid and continuous delivery of high-quality software.

### 3.2.2 Functional requirements

- **Easy initiation of tool installation:** The system must provide a straightforward method for initiating the installation of various CI/CD tools. Users should be able to start the installation process with minimal effort, reducing the complexity of setting up new environments.
- **Configurable installation using abstracted variables:** The installation processes should be highly configurable, allowing users to easily modify settings through the use of abstracted variables. This flexibility ensures that the system can be tailored to meet specific project needs and accommodate different environments without requiring deep technical knowledge.
- **Triggering installations through a user interface (UI):** A user-friendly interface should be provided to allow administrators and developers to trigger the installation and configuration of CI/CD tools. This UI should simplify complex tasks and make the deployment process accessible to users with varying levels of technical expertise.
- **Integration of installed tools:** The system should ensure that all installed CI/CD tools can be seamlessly integrated with each other. This integration is crucial for creating cohesive workflows that automate the entire software development lifecycle, from code integration to deployment and monitoring.
- **Authentication and execution history tracking:** The system must include robust authentication mechanisms to verify user identities and maintain a secure environment. Additionally, it should provide detailed logs of execution history to track actions performed by different users, ensuring accountability and traceability.

### 3.2.3 Non functional requirements

- **Secure storage of host machine credentials:** Credentials for accessing host machines should be stored securely within the system. This measure prevents unauthorized access and protects sensitive information, aligning with best practices for information security.
- **Role-based access control (RBAC) and credential management:** The system should support role-based access control to restrict or permit actions based on user roles. This approach enhances security by limiting access to sensitive operations and data to only those with appropriate permissions. It should also provide functionality for adding, viewing, and managing credentials securely, ensuring that only authorized users can access or modify them.
- **Effective distribution and separation of projects and users:** The system must facilitate the organized distribution and separation of projects and users. This requirement ensures that different teams can work independently on their projects without interference, while also enabling centralized management and oversight by administrators. Clear boundaries between projects and users help maintain data integrity and prevent conflicts or unintended interactions.

## 3.3 Solution Design

In this section, we present the solution design, detailing the architectural frameworks and deployment strategies that form the backbone of our automatically installed CI/CD infrastructure. We begin by outlining the global architecture, offering a comprehensive overview of the system's components and their interactions. This is followed by an in-depth exploration of the AWX deployment and internal architecture, highlighting how AWX integrates into the larger infrastructure. Additionally, the deployment architectures for GitLab and Jenkins are examined, showcasing the general configurations setups employed. Finally, the CI/CD pipeline design is illustrated, demonstrating the workflow and stages involved in automating the software development lifecycle. Each subsection provides a visual representation through detailed diagrams, ensuring clarity and aiding in the understanding of the complex architectural designs.

### 3.3.1 Global architecture

The diagram illustrates the global architecture of a CI/CD infrastructure deployment process using AWX hosted on a Kubernetes cluster.

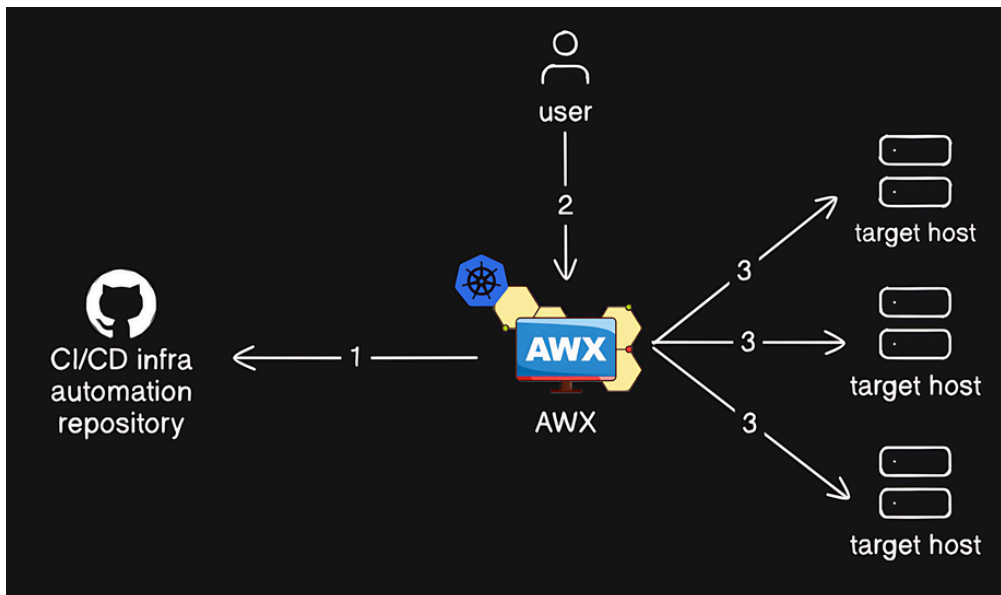


Figure 3.1: Global architecture

Here's a detailed explanation of the components and the flow represented by the numbered arrows:

1. **AWX Syncs with GitHub Repository:** The AWX instance, which is hosted on a Kubernetes cluster, periodically syncs with a GitHub repository that contains the Ansible playbooks necessary for deploying the CI/CD infrastructure. These playbooks are developed and maintained in the repository, ensuring that the latest configurations and scripts are always available for execution.
2. **User Configures and Executes Playbooks:** Users interact with AWX to configure the target hosts. This involves specifying the IP addresses and SSH credentials of the machines where the infrastructure will be deployed. Once the target hosts are configured, users can select and execute the relevant playbooks. AWX fetches the playbooks from the GitHub repository during this process, ensuring that the most up-to-date version is used.
3. **AWX Deploys Infrastructure to Target Hosts:** AWX takes the selected playbooks and executes them on the specified target hosts. This step involves installing and configuring the necessary CI/CD tools and components on each target machine. During and after the execution, AWX provides detailed logs, allowing users to monitor the progress and troubleshoot any issues that arise during the deployment.

#### **Benefits of This Solution:**

- **Centralized Automation:** The use of AWX provides a centralized platform for automating the deployment of CI/CD infrastructure, making it easier to manage and scale as the environment grows.
- **Version Control and Consistency:** By syncing with a GitHub repository, this architecture ensures that all deployments are based on version-controlled playbooks, reducing the risk of inconsistencies and errors in the infrastructure setup.
- **Scalability:** Hosting AWX on a Kubernetes cluster allows the solution to scale efficiently, handling an increasing number of deployments and target hosts without degradation in performance.
- **User-Friendly Interface:** AWX offers a user-friendly web interface that allows users to configure target hosts and execute playbooks without needing to manually run commands, thus reducing the learning curve for managing infrastructure deployments.

- **Detailed Logging and Transparency:** The detailed logs provided by AWX during playbook execution offer transparency into the deployment process, enabling easier troubleshooting and ensuring that the deployment process is fully auditable.

### 3.3.2 AWX architecture

In this section, we delve into the architecture of AWX, a powerful open-source tool for managing Ansible playbooks and automating IT processes. The architecture is explored from two perspectives: deployment and internal structure.

#### AWX deployment architecture

The AWX deployment architecture focuses on how the application is hosted on a Kubernetes cluster, detailing the components and services that make up the deployment environment. This includes the various pods, services, and other Kubernetes resources that are essential for the operation of AWX.

This deployment diagram illustrates the setup of AWX within a Kubernetes environment. This architecture ensures high availability, scalability, and efficient resource management for AWX operations.

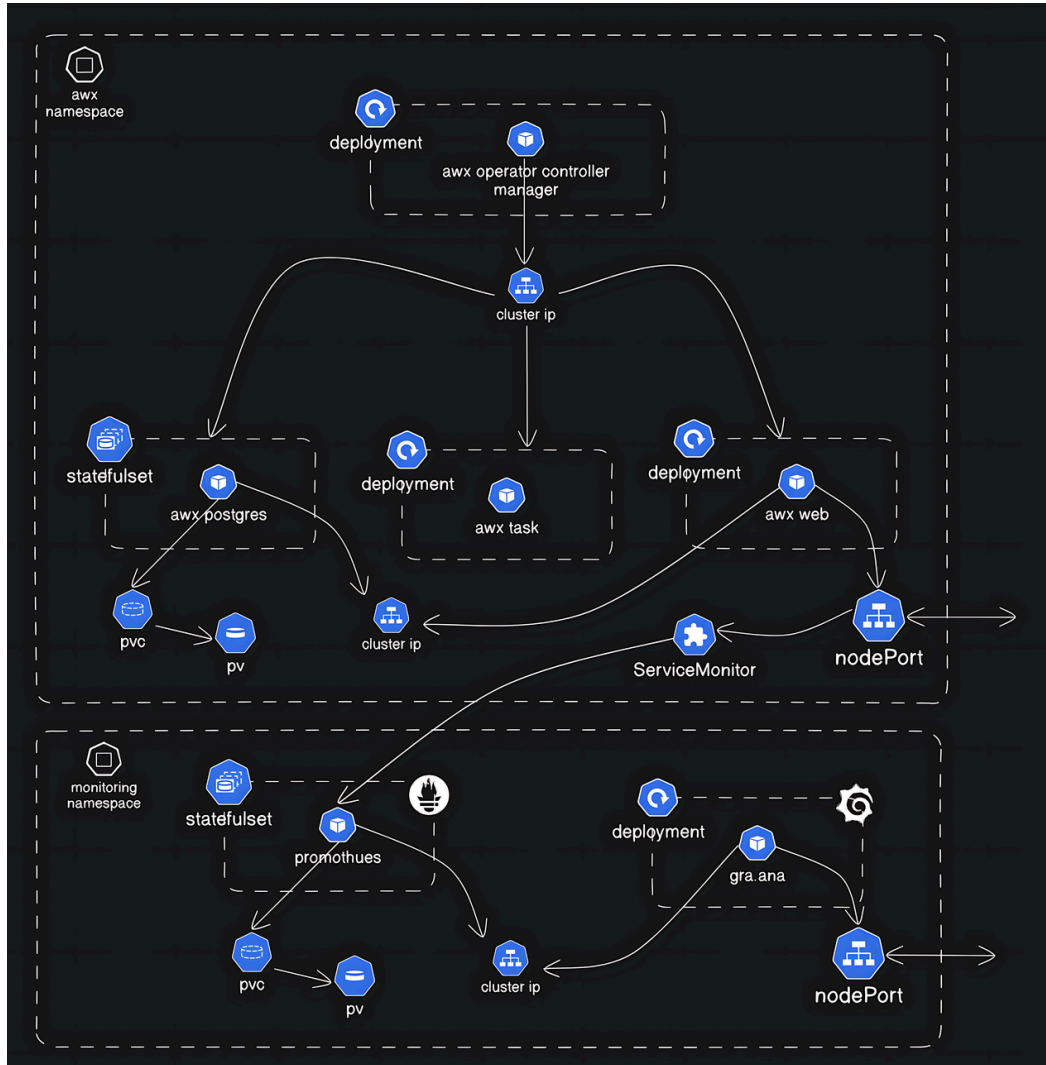


Figure 3.2: AWX deployment architecture

Here's a breakdown of the components and their interactions:

### **AWX Namespace:**

1. **AWX Operator Controller Manager:** Deployed as a controller, the AWX operator manages the lifecycle of the AWX application on Kubernetes. It automates tasks such as installation, upgrades, and scaling, ensuring that the AWX environment remains consistent and reliable.
2. **AWX Postgres StatefulSet:** The database for AWX, running as a StatefulSet, ensures persistent storage and consistent data management. It uses a Persistent Volume Claim (PVC) linked to a Persistent Volume (PV) to store data across pod restarts, maintaining the state and configurations of AWX.
3. **AWX Task Deployment:** Handles the execution of background tasks and job runs in AWX. This deployment ensures that tasks are processed efficiently and in a distributed manner, leveraging Kubernetes' scaling capabilities.
4. **AWX Web Deployment:** The web interface for AWX is hosted here. Users interact with this component for configuring and managing automation tasks. The deployment exposes a NodePort service to allow external access to the AWX interface.
5. **Service Monitor:** Integrated monitoring of AWX services, providing metrics and health checks for the application. This ensures that AWX performance and availability can be tracked in real-time.

### **Monitoring Namespace:**

1. **Prometheus StatefulSet:** Prometheus is used for monitoring and alerting, deployed as a StatefulSet to ensure persistence of metrics data. It collects and stores metrics from AWX and other services, ensuring that historical data is retained across restarts.
2. **Grafana Deployment:** Grafana provides a graphical interface for visualizing the metrics collected by Prometheus. It is deployed with access via a NodePort service, enabling users to access detailed dashboards and performance data of the AWX environment.

### **Key Interactions:**

- The AWX operator manages the deployment and lifecycle of AWX components, ensuring that each service functions as expected.



- The AWX web deployment interacts with users, allowing them to configure automation tasks and monitor execution.
- Prometheus and Grafana in the monitoring namespace track the performance and health of AWX, offering insights into the overall system status.

This architecture leverages Kubernetes' capabilities to provide a robust and scalable AWX environment, with integrated monitoring and persistent storage to ensure that automation tasks are managed efficiently.

### AwX internal architecture

The AWX internal architecture offers a deeper look into the composition of AWX itself, highlighting its core components and main objects. This section covers the internal workings of AWX, such as how it manages projects, inventories, job templates, and credentials, providing a comprehensive understanding of how AWX orchestrates complex automation workflows.

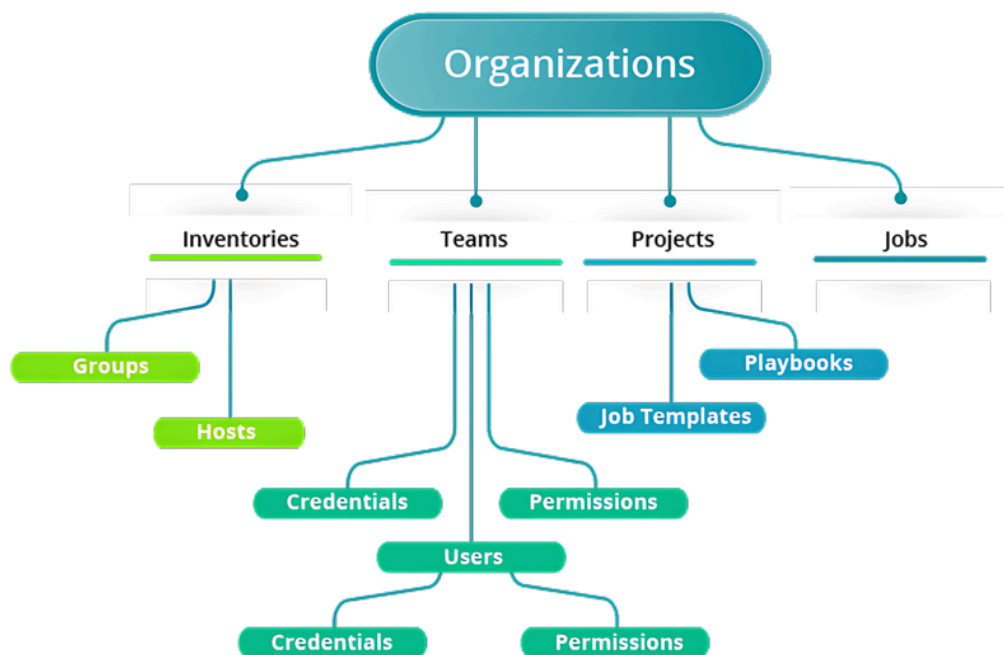


Figure 3.3: AWX internal architecture  
[3]

The AWX internal architecture is organized hierarchically, starting with "Organizations" at the top. Organizations encompass four primary components:

- **Inventories** are divided into Groups and Hosts, where Groups contain collections of hosts, which are the actual machines managed by AWX.
- **Teams** consist of Users, who are the individuals interacting with AWX. Users have associated Credentials and Permissions that dictate what actions they can perform.
- **Projects** house Job Templates and Playbooks. Job Templates are predefined workflows, while Playbooks are collections of Ansible tasks to be executed.
- **Jobs** are executed actions derived from Job Templates and Playbooks.

This structure ensures a modular and scalable approach to managing resources and permissions within an organization.

### 3.3.3 CI/CD infrastructure architecture

In this section, we will explore the architecture of the Continuous Integration/Continuous Deployment (CI/CD) infrastructure, detailing the deployment setups for both GitLab and Jenkins, as well as the overall CI/CD pipeline design. The diagrams provided will illustrate how these components are configured and interact within the infrastructure, highlighting their deployment on Docker and their communication patterns for executing automated CI/CD tasks.

#### Gitlab deployment architecture

The diagram below depicts the architecture of the automatically installed GitLab Community Edition (CE).

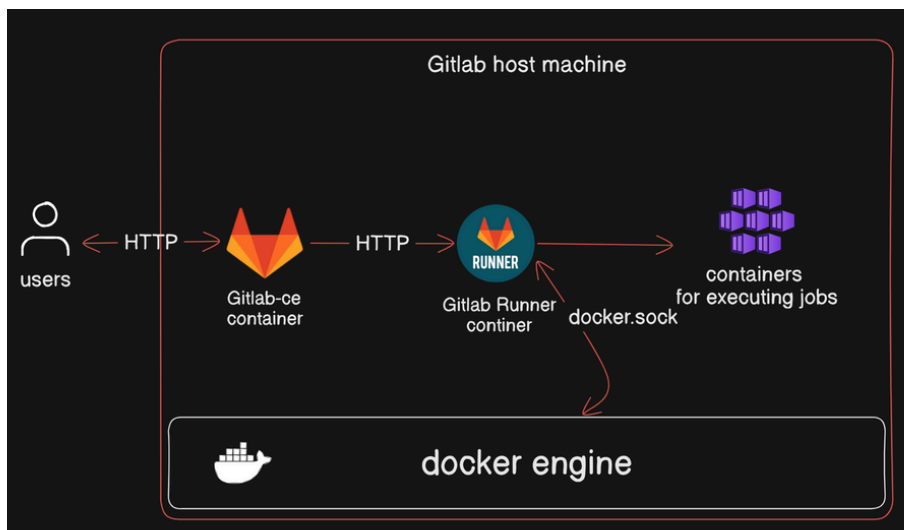


Figure 3.4: Gitlab deployment architecture

As illustrated, GitLab CE is deployed as a container running on top of a Docker Engine. Alongside it, a GitLab Runner is installed, which facilitates the execution of CI/CD jobs. The GitLab CE and the Runner communicate with each other through HTTP, while the Runner interacts with the Docker Engine to spin up containers required for executing the CI/CD pipelines. The entire application is exposed to users via HTTP, allowing for easy access and management of repositories, CI/CD pipelines, and more.

## Jenkins deployment architecture

The following diagram illustrates the deployment architecture for the automatically installed Jenkins environment. As shown in the diagram, Jenkins is installed

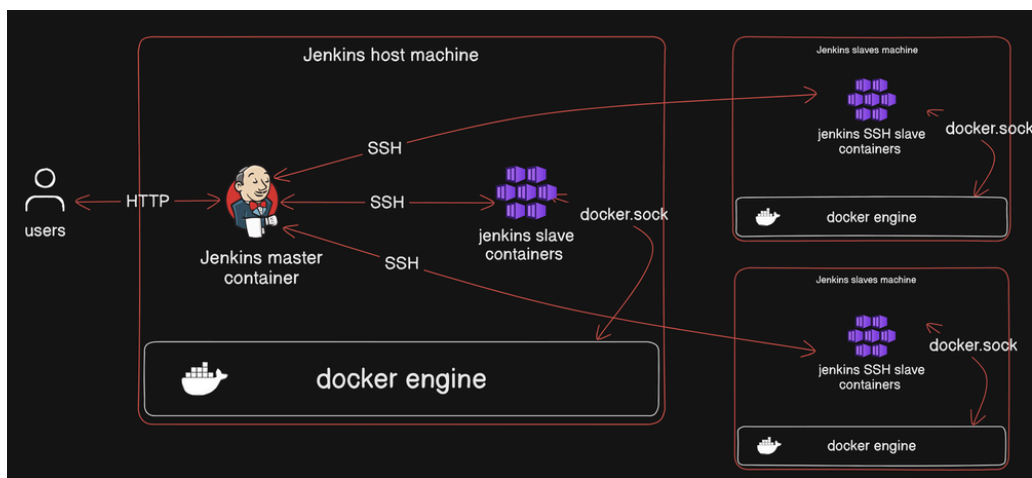


Figure 3.5: Jenkins deployment architecture

as a controller running on top of the Docker Engine. Additionally, SSH-based Jenkins slaves (also known as agents) are hosted on different machines and connected to the Jenkins controller via SSH. These slaves communicate with Docker to facilitate the utilization of Docker containers within the Jenkins pipelines, enabling efficient and scalable execution of CI/CD tasks across distributed environments.

### 3.3.4 CI/CD pipeline

The CI/CD pipeline is designed as depicted in the diagram below:

This design outlines the stages and flow of the CI/CD process, demonstrating how code changes are automatically built, tested, and deployed across various environments. The pipeline is meticulously structured to ensure seamless continuous integration and delivery, leveraging the automation capabilities provided by

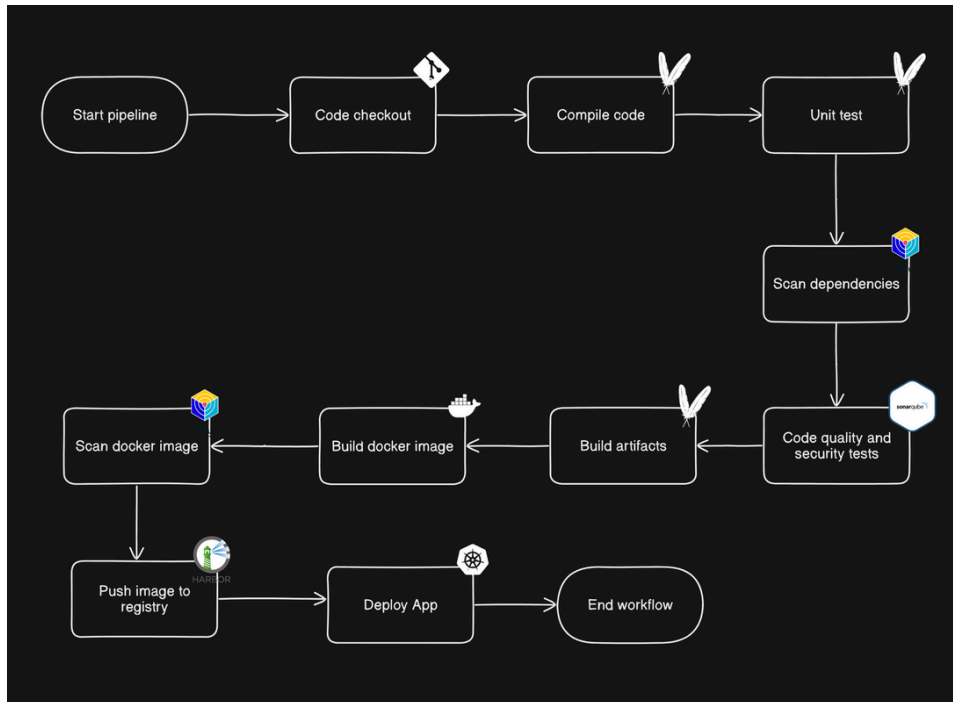


Figure 3.6: CI/CD pipeline

both GitLab and Jenkins. This automation streamlines development workflows, accelerates the release cycle, and enhances the overall efficiency of the development process.

The diagram also highlights the specific tools utilized at each stage of the pipeline:

- **Code Checkout:** Git is used for version control, allowing teams to manage code changes effectively.
- **Compilation, Unit Testing, and Build:** Maven is employed for compiling the code, running unit tests, and building the project artifacts.
- **Dependency Checking and Image Analysis:** Trivy is integrated to perform security scanning, ensuring that the application and its dependencies are free from vulnerabilities
- **Security Scanning and Code Quality:** SonarQube is used to analyze the code for potential bugs, security vulnerabilities, and code smells, ensuring high-quality code before deployment.
- **Artifact Registry:** Harbor is utilized as a secure container registry, managing and distributing Docker images efficiently.

- **Deployment:** Kubernetes is used for orchestrating the deployment of applications, providing scalability and resilience.

All the tools mentioned in this pipeline are automatically installed and configured through the AWX platform except for kubernetes cluster that has already been setup. This pipeline serves as an experimental framework to test the efficiency and effectiveness of the CI/CD infrastructure, ensuring that all components work harmoniously to support a robust and secure development process.

## 3.4 Conclusion

In this chapter, we thoroughly analyzed the requirements and presented a comprehensive design for automating the CI/CD infrastructure setup. We identified the key actors, functional and non-functional requirements, and outlined the architecture and deployment strategies for AWX, GitLab, Jenkins, and the overall CI/CD pipeline. These design considerations ensure a scalable, and efficient infrastructure that meets the needs of both administrators and developers. The following chapter, "Implementation and Deployment" will delve into the practical application of these designs, detailing the processes and steps involved in bringing the CI/CD infrastructure to life.

# Chapter 4

## Implementation and Deployment

### 4.1 Introduction

This chapter provides a comprehensive overview of the implementation and deployment process of the automation solution. It outlines the development and deployment environments, including both hardware and software configurations. Additionally, the chapter details the development of Ansible playbooks, the setup of a Kubernetes cluster, and the deployment of AWX. The final sections cover the setup of the project within AWX, the installation of the CI/CD infrastructure, and the creation of the CI/CD pipeline using Jenkins.

### 4.2 Development and Deployment Environment

This section describes the environments used during the development and deployment phases. It covers the hardware and software specifications for both environments.

#### 4.2.1 Hardware Environment

The hardware environment is divided into two parts: the development environment and the deployment environment.

##### Development Environment

The development environment consisted of a HP laptop equipped with 16GB of RAM, an Intel i7 8th generation processor, and a 500GB SSD. For testing the Ansible playbooks, I utilized AWS EC2 t2.micro free tier instances, which provided 1GB of RAM and 1 CPU.

## Deployment Environment

For the deployment environment, I utilized two on-premises virtual machines (VMs) created and managed by OpenStack private cloud infrastructure provided by Sofrecom servers. The first VM was dedicated to hosting the Kubernetes cluster and AWX, while the second was used to deploy the CI/CD infrastructure. Both VMs were provisioned with 32GB of RAM, a 300GB SSD, and 8 CPUs (Intel® Xeon® 6746E processors).

### 4.2.2 Software Environment

The software environment included Ubuntu Linux 22.04 LTS (Jammy Jellyfish) as the operating system, Python 3.11, Ansible for automation, Visual Studio Code (VSCode) for code editing, and Git for version control. Additionally, the AWS Management Console was used for managing AWS resources.

## 4.3 Implementation

This section details the implementation process, including the development of Ansible playbooks, the setup of the Kubernetes cluster, and the deployment of AWX.

### 4.3.1 Development of Ansible Playbooks

I developed multiple Ansible playbooks to automate various installation and configuration tasks. These playbooks included:

- A playbook for installing frequently used packages, Docker, Docker Compose, and Python packages necessary for managing Docker.
- Three playbooks for GitLab: one for installation, one for changing the root user's password, and one for registering new runners.
- Three playbooks for Jenkins: one for installation, one for installing frequently used plugins, and one for adding a new slave.
- A playbook for installing SonarQube.
- A playbook for installing Harbor Registry with Trivy scanner.

The code for these playbooks can be found in the following repository: <https://github.com/Mensi0mar/CI-CD-infra-automation>.



### 4.3.2 Setting up Kubernetes Cluster and Deploying AWX

After developing the playbooks, I proceeded to deploy AWX. First, I set up a Kubernetes cluster manually on a VM using ‘kubeadm’ with an all-in-one architecture, where the control plane also functions as a node. This setup was intentional, as the VM was powerful enough to handle both components, and the expected usage was minimal (a maximum of 10 users).

Once the cluster was fully set up, I deployed AWX. The following screenshot shows all the components of the deployment in the AWX namespace, as discussed in the design chapter.

```
vm1@openstack:~/cicd-main$ kubectl get all -n awx
```

NAME	READY	STATUS	RESTARTS	AGE
pod/awx-operator-controller-manager-bb695bf5d-54xb7	2/2	Running	0	15d
pod/awx-prod-v1-migration-24.6.1-blqjd	0/1	Completed	0	15d
pod/awx-prod-v1-postgres-15-0	1/1	Running	0	15d
pod/awx-prod-v1-task-85477c9cfd-4sz4g	4/4	Running	0	8d
pod/awx-prod-v1-web-656d9b89c8-7fzjq	3/3	Running	0	15d

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/awx-operator-controller-manager-metrics-service	ClusterIP	10.103.183.211	<none>	8443/TCP	15d
service/awx-prod-v1-postgres-15	ClusterIP	None	<none>	5432/TCP	15d
service/awx-prod-v1-service	NodePort	10.101.219.208	<none>	80:30842/TCP	15d
service/vm-server-nlp	ClusterIP	10.109.141.109	<none>	22/TCP	8d

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/awx-operator-controller-manager	1/1	1	1	15d
deployment.apps/awx-prod-v1-task	1/1	1	1	15d
deployment.apps/awx-prod-v1-web	1/1	1	1	15d

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/awx-operator-controller-manager-bb695bf5d	1	1	1	15d
replicaset.apps/awx-prod-v1-task-85477c9cfd	1	1	1	15d
replicaset.apps/awx-prod-v1-web-656d9b89c8	1	1	1	15d

NAME	READY	AGE
statefulset.apps/awx-prod-v1-postgres-15	1/1	15d

NAME	STATUS	COMPLETIONS	DURATION	AGE
job.batch/awx-prod-v1-migration-24.6.1	Complete	1/1	4m11s	15d

Figure 4.1: AWX deployment screenshot

After AWX was fully operational and healthy, I deployed the monitoring tools and connected them to AWX. I installed the Prometheus Operator using Helm and connected the AWX instance with Prometheus using the Custom Resource Definition (CRD) ‘ServiceMonitor‘ provided by the Prometheus Operator.

The screenshots below showcase the monitoring namespace, as discussed in the design chapter.

```

vm1@openstack:~/cicd-main$ kubectl get all -n monitoring

NAME                                READY   STATUS    RESTARTS   AGE
pod/alertmanager-prometheus-operator-kube-p-alertmanager-0  2/2     Running   0           14d
pod/prometheus-operator-grafana-7df5c7bd67-fwkxz             3/3     Running   0           14d
pod/prometheus-operator-kube-p-operator-74df48c96-gmscb       1/1     Running   0           14d
pod/prometheus-operator-kube-state-metrics-86479bd8d7-75vkn   1/1     Running   0           14d
pod/prometheus-operator-prometheus-node-exporter-tzsrw       1/1     Running   0           14d
pod/prometheus-prometheus-operator-kube-p-prometheus-0       2/2     Running   0           14d

NAME                                TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
service/alertmanager-operated       ClusterIP      None             <none>            9093/TCP,9094/TCP,9094/UDP  14d
service/grafana-server-service       NodePort       10.111.85.215    <none>            3000:32678/TCP      11d
service/prometheus-operated          ClusterIP      None             <none>            9090/TCP           14d
service/prometheus-operator-grafana  ClusterIP      10.105.253.18    <none>            80/TCP             14d
service/prometheus-operator-kube-p-alertmanager               ClusterIP      10.103.15.127    <none>            9093/TCP,8080/TCP    14d
service/prometheus-operator-kube-p-operator                   ClusterIP      10.106.183.179    <none>            443/TCP              14d
service/prometheus-operator-kube-p-prometheus                 ClusterIP      10.108.108.32     <none>            9090/TCP,8080/TCP    14d
service/prometheus-operator-kube-state-metrics                ClusterIP      10.106.251.198    <none>            8080/TCP              14d
service/prometheus-operator-prometheus-node-exporter          ClusterIP      10.107.76.59      <none>            9100/TCP              14d
service/prometheus-server-service  NodePort       10.96.157.242     <none>            9090:31466/TCP       11d

NAME                                DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/prometheus-operator-prometheus-node-exporter  1          1         1        1             1           kubernetes.io/os=linux  14d

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/prometheus-operator-grafana  1/1      1             1           14d
deployment.apps/prometheus-operator-kube-p-operator  1/1      1             1           14d
deployment.apps/prometheus-operator-kube-state-metrics  1/1      1             1           14d

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/prometheus-operator-grafana-7df5c7bd67  1          1         1           14d
replicaset.apps/prometheus-operator-kube-p-operator-74df48c96  1          1         1           14d
replicaset.apps/prometheus-operator-kube-state-metrics-86479bd8d7  1          1         1           14d

NAME                                READY   AGE
statefulset.apps/alertmanager-prometheus-operator-kube-p-alertmanager  1/1     14d
statefulset.apps/prometheus-prometheus-operator-kube-p-prometheus  1/1     14d

```

Figure 4.2: Monitoring namespace screenshot

Next, I created the Grafana dashboard for AWX, and here is the screenshot:

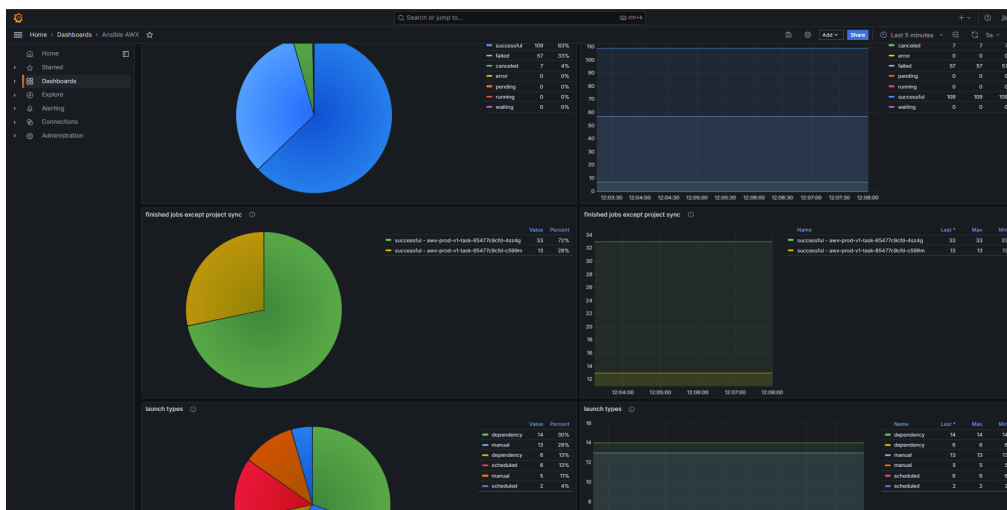
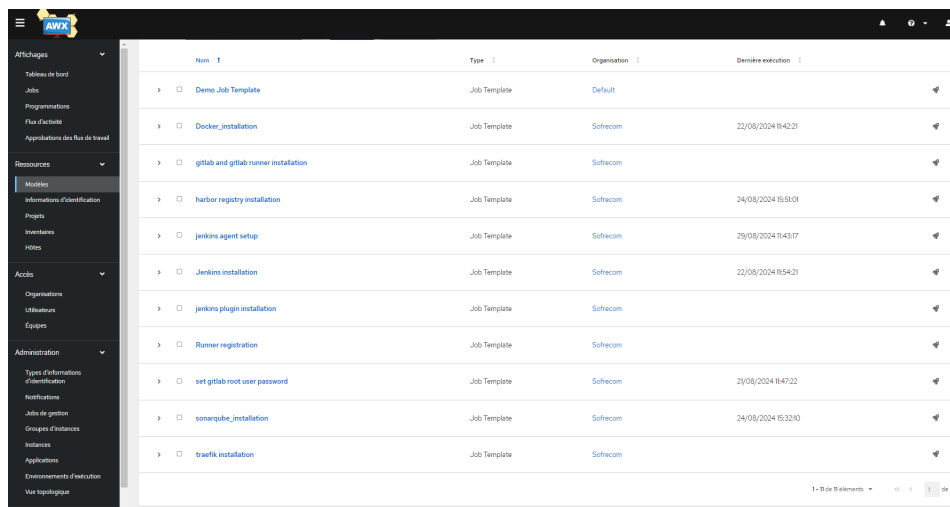


Figure 4.3: Grafana AWX dashboard

### 4.3.3 Setting up the Project and Installing the Infrastructure

After completing the deployment, I worked on the AWX UI to add the CI/CD infrastructure automation project, connecting it to my Git repository on the main branch. I then set up the job templates (playbooks) with the required variables and configurations.

I created an inventory for the project, containing all the required groups of hosts and added a VM as a host along with its specified credentials. Here is a screenshot of all the added playbooks:



The screenshot shows the AWX web interface with the 'Job Templates' page. The left sidebar contains navigation links for 'Affichages', 'Ressources', 'Accès', and 'Administration'. The main table lists the following job templates:

Nom	Type	Organisation	Dernière exécution
Demo Job Template	Job Template	Default	
Docker_installation	Job Template	Sofrecom	22/08/2024 14:42:21
gitlab and gitlab-runner installation	Job Template	Sofrecom	
harbor registry installation	Job Template	Sofrecom	24/08/2024 15:58:01
jenkins agent setup	Job Template	Sofrecom	29/08/2024 14:43:17
Jenkins installation	Job Template	Sofrecom	22/08/2024 14:54:21
jenkins plugin installation	Job Template	Sofrecom	
Runner registration	Job Template	Sofrecom	
set gitlab root user password	Job Template	Sofrecom	29/08/2024 14:47:22
sonarqube_installation	Job Template	Sofrecom	24/08/2024 15:32:10
traefik installation	Job Template	Sofrecom	

At the bottom right of the table, it indicates '1 - 11 sur 11 éléments' and '1 de 1 page'.

Figure 4.4: AWX templates

Finally, I installed the infrastructure using AWX, and the following is an example of an execution as a screenshot:

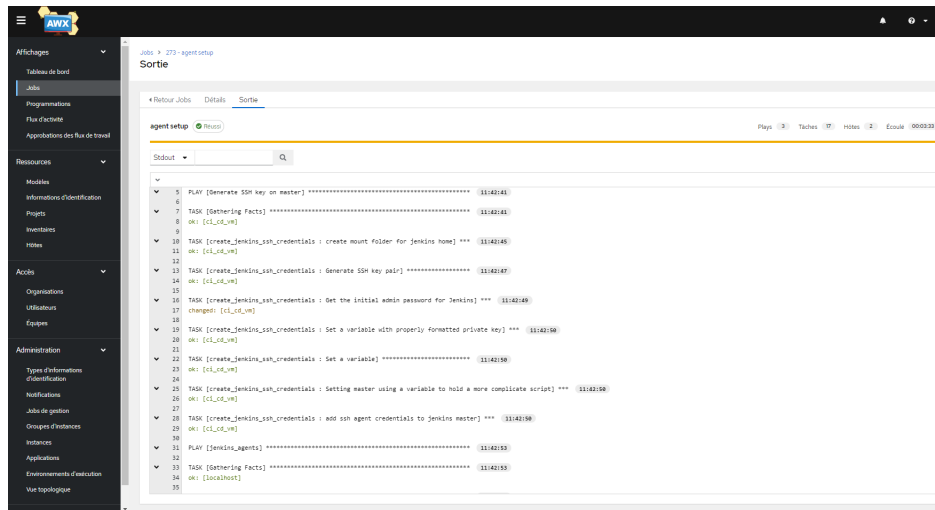


Figure 4.5: Job execution example

### 4.3.4 Creating the CI/CD Pipeline

To create the CI/CD pipeline, I used Jenkins and a random public repository containing a web application developed using the Spring Boot framework. I incorporated all the automatically installed tools into the CI/CD pipeline.

I also created a namespace in my Kubernetes cluster for Jenkins to deploy applications. For this, I created a 'ServiceAccount' named 'jenkins', created a role containing all the necessary permissions for Jenkins, and then bound the role using 'RoleBinding'. The screenshot below shows the successful execution of the pipeline:

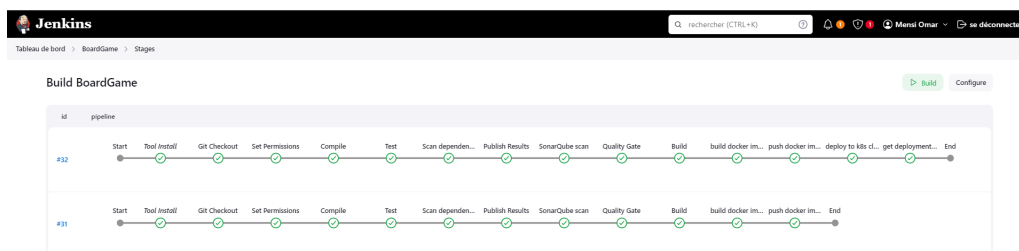


Figure 4.6: Jenkins CI/CD pipeline

## 4.4 Conclusion

In this chapter, I detailed the implementation and deployment process for the CI/CD infrastructure. From the development of Ansible playbooks to the deployment of AWX and the creation of a CI/CD pipeline with Jenkins, each step was carefully executed to ensure a robust and automated infrastructure. The successful deployment and operation of the CI/CD pipeline demonstrate the effectiveness of the automation and the environment setup.

# Conclusion

This report has provided a comprehensive exploration of the design, implementation, and deployment of a robust CI/CD infrastructure automation solution, showcasing the critical role of automation and modern development practices in creating an efficient software development pipeline. We began by establishing the context and scope of the project, identifying the challenges faced, and proposing a solution that leverages a combination of cutting-edge technologies like Ansible, Kubernetes, AWX, and Docker to address those challenges effectively.

Throughout the development and deployment phases, we meticulously detailed the hardware and software environments utilized, ensuring that the infrastructure was both scalable and reliable. The development of Ansible playbooks played a pivotal role in automating the installation and configuration processes, which are essential for maintaining consistency and reducing the potential for human error.

In deploying the solution, we successfully set up a Kubernetes cluster, which served as the foundation for deploying AWX and other essential services. The integration of monitoring tools like Prometheus and Grafana ensured that the deployed systems could be monitored effectively, allowing for proactive management and maintenance.

The culmination of these efforts was the creation of a fully automated CI/CD pipeline, which was rigorously tested and demonstrated to work seamlessly with a sample application. This pipeline represents the core of the project's success, highlighting the power of automation in streamlining software delivery and enhancing the overall efficiency of the development process.

In conclusion, this project has successfully implemented a state-of-the-art CI/CD infrastructure that not only meets the project's initial objectives but also sets a strong foundation for future enhancements and scalability. The methodologies and technologies applied throughout this project can serve as a valuable reference for similar initiatives, contributing to the advancement of automated infrastructure management in the field of software development.

# Bibliography

- [1] [www.clouddefense.ai](http://www.clouddefense.ai). How to implement an effective ci/cd pipeline.
- [2] Rajesh Kumar. Understanding ansible architecture using diagram.
- [3] [opstree.com](http://opstree.com). Features of awx.