TUNISIAN REPUBLIC

Ministry of higher education and scientific research

University of gabes

National School of Engineering of Gabes

Department of Communication &Network Engineering

المدرسة الوطنية للمهندسين بقابس

قسم هندسة الاتصالات والشبكات

THESIS OF THE END OF STUDIES PROJECT

Submitted for the

National Diploma of Engineering in communications & Network Engineering

Realized by :

Salim Bechraoui

Subject:

Implement the CI/CD process via DevOps Toolchain

Presented in x/x/2022 in front of the jury commission

Année Universitaire : **2021/2022** GCR 2022

DEDICATION

To my father and my mother and my brother and my cousins also my dears friends who always stand beside me and hyped for me during the making of this work. Allows me to strengthen my knowledge and makes me move more forward behind my limits.
I would like to thank all those who have supported me in the moments when I needed Them the most for their enriching contribution to the beginning of my professional career. I would also like to thank my college for the especially warm welcome I received, but also for the advice, support, and help I was given and the patience and professionalism. This experience has been very rewarding and will certainly help me to upgrade throughout my career. I would like to express my deepest appreciation to all of them. I dedicate this humble work to you and I wish you the brightest future.

ACKNOWLEDGEMENTS

At the end of this work, I would like to thank all those who, without their priceless help, this project would never have been carried out.

I would like to thank in particular Mr.Abdelhakim Khlifi my academic supervisor, who directed my final project, for the interest he bought in my work he provided me with his support, his availability, and his advice that guided me throughout my internship.

I thank Mr. Talhaoui Salah Technical Director at UPTech for the quality of his assistance, for his friendly welcome, and for his professional cooperation throughout these months also my sincere thanks to all the staff at UPTech.

Finally, I would like to thank the members of the jury for their presence, for their attentive reading of this report, as well as for the remarks they will address to me during this presentation to improve my work. My teachers hoping that you will se in the work the rewards of the dedication you have shown during the lessons you have given us.

List of subjects

List of Figure

List of tables

List of abbreviations

DevOps: Development & Operation

API: Application Programming Interface

AST: Application Security Testing

CD: Continuous Deployment/Delivery

CI: Continuous Integration

SAST: Static Application Security Testing

DNS: Domain Name System

IT: Information Technology

IP: Internet Protocol

K8s: Kubernetes

OS: Operation System

HTTPS: Hyper Text Transfer Protocol Secure

NPM: Node Package Manager

SSH: Secure Shell

YAML: Yet Another Markup Language

VM: Machine Virtuelle

General introduction

Historically, developers and system and networks professionals (also called IT professionals) worked more in parallel than in concert. Thus, developers sometimes create deliverables that are poorly adapted to the company's infrastructures. With demands on time to market and the evolution of the project's increasingly complex software and infrastructure configurations, operational and planning, risks arise. The need to industrialize the tests and simplify the deployment of production has gradually imposed itself.

One of the most popular DevOps practices is continuous delivery. Continuous delivery is a software strategy that enables organizations to deliver new functionality to users quickly and efficiently. The goal of continuous delivery is to achieve a constant flow of changes in production through an automated software production line.

UpTech is an expert in the field of information systems and digital transformation for Tunisian companies, aiming to integrate the DevOps concept into the realization of these new projects in order to be competitive in the evolving market. It is in this context that the research end project that we carried out within Uptech was also constructed within the framework, Therefore the purpose of this work is to obtain a communications and network engineering engineer diploma from the National School of Engineers of Gabes, it 's about building a DevOps platform for continuous integration and deployment.

This report is organized as follows :

In the first chapter, we first present the host company, then we continue with a study of the existing situation to deduce the problems and propose the appropriate solution. Finally, We introduce the concept of DevOps and make a comparative study of the different tools used in the proposed solutions.

In the second chapter, we present state the requirement analysis, the global use case of the pipeline, and the design of the CI/CD chain.

The last chapter is dedicated to the functional and technical realization of our project.

Finally, we close this report with a general conclusion in which we evaluate the results achieved and outline the possible perspective of our project.

I. Project context

Introduction:

The objective of this section is to summarize the overall work of the project. It represents the following elements: The host company Thereafter, we will continue the study of the existing system by specifying these limitations, and then we will propose the appropriate solution to deal with these limitations. Finally, we introduce the concept of DevOps.

I.1 Host institute presentation

UPTECH (1) is a B2B technology company established in 2019. It is located in Tunisia and, given its digital role for the benefit of client companies, it claims to be a general technology provider.

Figure I.1-1UPTECH logo

I.2 Sector of activity

UPTech is a technology company that provides services in various domains related to

- Improve the internal management of companies by providing an efficient essay ERP
- implementing a secure and efficient IT infrastructure to facilitate data traffic.
- Facilitate and evolve the ways of working by offering virtualization and cloud services
- Ensure data security via antivirus, data monitoring, access control, and firewalls.
- Create a digital marketing strategy that relies on content marketing, mobile marketing, email marketing SEO, etc

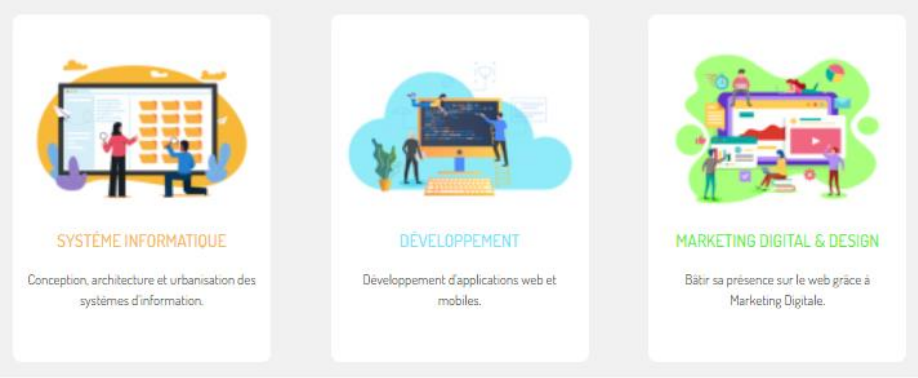


Figure I.2I.2-1Digital Transformation

I.2.1 Client file

Thank to its products and the quality of its work UPtech has become an official partner of Sophos, Divalto, and RFID Insys-

Figure I.2.1-1Company partners

I.2.2 Company chart

The functional organization chart within UPtech is shown in *Figure I.2.2-1* and we note that his internship took place in the Technical department.

The organizational structure of Uptech is composed of a CEO (chief executive officer) who is the general manager of the company and the director of the consulting department at the same time.

The technical director leads the developers and the IT security managers and development project manager, who are also consultants working for other companies and doing paid interventions for UPtech .

Figure I.2.2-2Company chart

I.3 Project Presentation

Our graduation project is part of the DevOps (2) approach at UPtech. For this part , we will make a study of the existing and its limits, and we will propose a solution to solve them.

I.3.1 Study the existing

UPTech follows a traditional project management approach. It is built around a waterfall system. Each sequence develops one after the other. When all the sequences are completed, the client will be delivered. Team and project management at UPTech are handled as follows:

- Number of teams: 3
- Development team: 5 developers
- ERP team: 6 consultants/developers
- system team: 2 system engineers
- Average project duration: 6 months to 2 years

The waterfall model as illustrated in *Figure I.3.1-1*

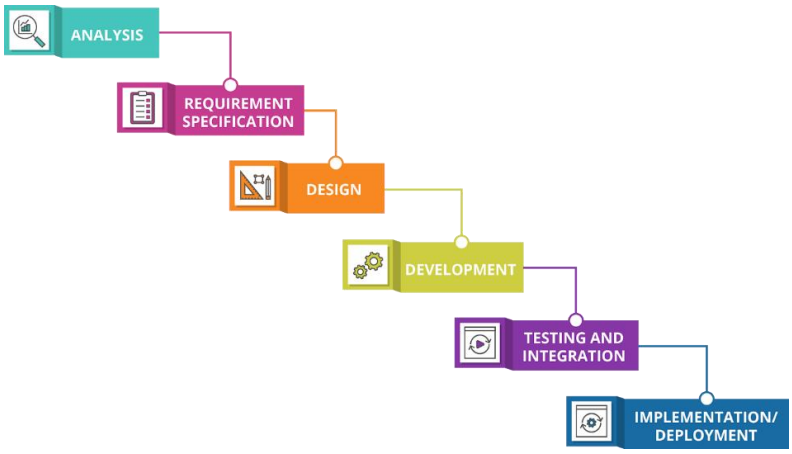


Figure I.3.1-2 Waterfall method

• **Analysis**: The purpose of a requirements analysis is to examine the functions to see if they are feasible and important. The results of these phases are specifications, or variants of them, or specifications, which provide the prerequisites for development.

• **Requirements**: This step involves understanding what is to be designed and what its function is.

• **Design**: System design helps define the overall architecture of the system and defines the hardware and software requirements of the system.

• **Development**: The development or implementation phase occurs when programmers integrate the requirements and specifications from the previous phases and produce actual code.

• **Testing and integration**: Once the coding is complete, testing is performed to ensure that there are no errors before the software is delivered to the customer.

• **Implementation**: During this phase, the production team makes the necessary corrections until the customer is satisfied.

I.3.2 Limit the existing

Within the company, the waterfall method is used to build the framework of the project and present clear and precise elements to the client. However, rigorous management can represent a constraint on rigidity, leaving no room for flexibility. As a result, it may be a small project. When it is a large project that extends over several months, it is essential to be vigilant and to take into account the evolution of needs. It can therefore be concluded that the waterfall method is not suitable for large-scale, complex projects. This method has several disadvantages

Including the following :

• The rate of customer dissatisfaction is increasing.

•Errors are often not discovered until late in the development process.

• The end-user is not integrated into the production process unit after the programming is complete.

• Conflicts, bugs, and programming errors can lead to increased costs.

With these disadvantages, the team has little choice but to complete the project, because, with a sequential process, the completion of one feature is contingent on the completion of previous ones. This makes it particularly difficult to accommodate all change requests, especially when each group is intentionally separated. knowledge sharing becomes difficult, and communication is also limited. The result is delayed due to a lack of clarity about what each team is doing, As a result, a wall of confusion appears between the two teams. These conflicts can lead to a company that was initially unforeseen and affect customer satisfaction, which is the objective of the company.

Figure I.3.2-1The wall of confusion

The IT field is changing rapidly We must continue to adapt. Whether it be to develop software or web applications, they need to be brought to market as quickly as possible.

I.3.3 Proposed solution

There is a need to apply an agile DevOps (3) methodology that matches the new projects requested to meet the needs and find solutions to previous problems. To this end, our approach is to implement a powerful infrastructure for the automation of processes and the use of sensors and technological tools to assist in the creation and evolution of applications in a fast and reliable manner. The objective of our solutions is :

• Accelerate the deployment mechanism to create faster and more efficient products and optimize them faster.

• Automate the processes between the development teams and the teams responsible for maintaining the developed application in an operational situation.

• A deployment technique based on continuous software delivery to ensure faster execution time and better code quality.

• Quickly adapt to a customer's changing needs on a regular basis to better meet their needs and gain their trust

•Reduce the risk of errors and failures in deployments by shortening production times.

• Reduce implementation costs.

The goal of adapting to the DevOps culture is to remove the barriers between two teams that have traditionally been isolated from each other. With this culture, we will restructure the organization and break down the communication barrier that exists between the development and operations teams, resulting in a more productive and confident work environment

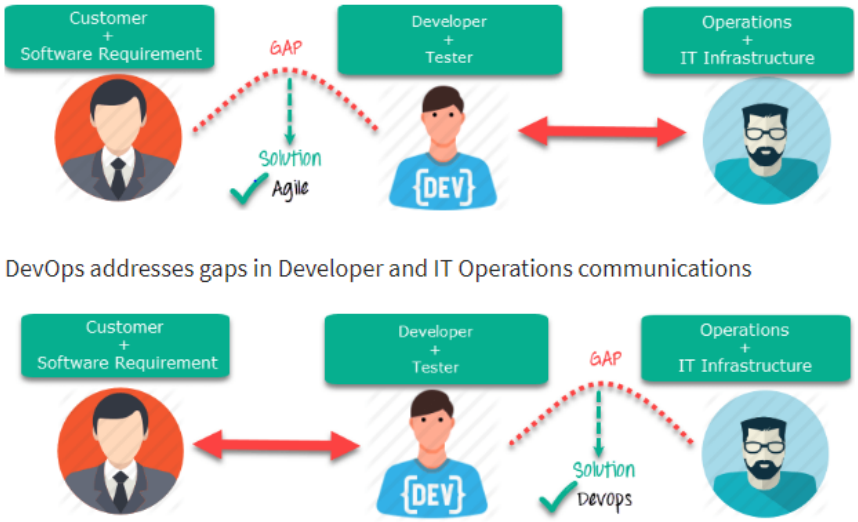


Figure I.3.3-1Agile DevOps

UPtech tasked us with merging the development part and the deployment part into a more streamlined exercise by automating the application life cycle stages. Our solution is called, in the DevOps concept a chain of continuous integration/continuous deployment (CI/CD pipeline).This the key to a successful DevOps transformation.

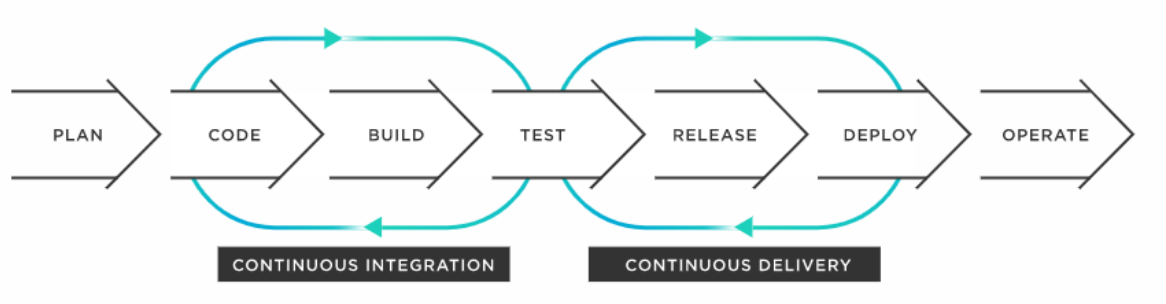


Figure I.3.3-2Continuous integration and continuous deployment

I.4 Principles and Tools used

The DevOps model is an approach to IT culture, automation, and platform design that aims to optimize business productivity through faster and more efficient service delivery. DevOps is a set of best practices for the industrialization of information systems and a strategy to shorten the time to market. For these reasons we are faced with the need to improve production efficiency through the automation of the various stages of the product life cycle

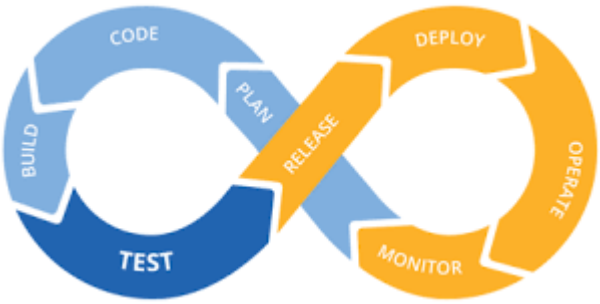


Figure I.4-1DevOps Cycle

I.4.1 Code

The first step in DevOps collaboration is to integrate the development and operations teams into a common source code management tool. In fact, this allows the developers to know the different modifications of the code and the authors of these modifications.

Definition: This phase includes software design and the creation of software code using GitHub or GitLab and Bitbucket for example

Tools Based on our research, we found menu development solutions, but we noticed that GitLab and Bitbucket are the most used tools.

• **GitLab:** is a release management software. It works in a distributed form, each person can develop their own repository.

• **Bitbucket:** is a flexible and evolutive web hosting service for software development teams and in particular provides source management with Git.

Table I.4.1-I-1:Comparaison between Gitlab and Bitbuck

Tools	Advantages	Disadvantages
GitLab	• Easy to manage and configure. • Open source. • Works with Git import	• The interface is relatively slow • There are often regular problems with the repositories.
Bitbucket	• Availability of private repositories and freepipelines • The ability to import other Git projects from Excel, GitHub, etc.	• Not an open source • Bitbucket is extremely slow, in particular the slowest git alternative available. Pages refresh in seconds and merge requests can last up to several minutes.

Summary:

This comparison was made for academic purposes, the choice of the solution was made by the host company, which is GitLab

I.4.2 Build

The building phase is where DevOps kicks in Once a developer has completed a task, he validates his code in a shared code repository

Definition: This phase consists of managing software versions and using automated tools for the compilation and integration of code for production. Source code or package repositories are also used to ”bundle” the infrastructure required to deliver the product using Docker, Ansible, Puppet, Chef, Gradle, Maven, or JFrog Artifactory, for example.

Tools: Based on our research we found many build solutions and the most important ones are Maven and Gradle:

• **Maven :**
Apache Maven (4) is a tool for managing and automating the production of Java software projects in general and Java EE in particular. It is used to automate continuous integration during software development.
• **Gradle:**
Gradle (5) is a build automation software known for its flexibility in creating software. The creation process includes compiling, linking, and packaging the code.

Table I.4.2-I:Comparaison between Maven and Gradle

Tools	Advantages	Disadvantages
Maven	• Automatically add all required dependencies tothe project • Easily build its project (jar, war, etc.)	• Maven must be installed in the system to operate a Maven plugin for the editor.
Gradle	• Gradle supports multi-project builds. • Free and open source • Allows using Maven repositories	• IDE integration is not asgood as Maven

Summary :

Based on our review of the various build tools, we decided on maven It is a complete solution that responds to the needs of the host organization.

I.4.3 Test

This phase includes continuous testing, whether manual or automated to ensure code quality.

Definition: There are many types of software testing. Each has its own attributes that can be tested to ensure better quality. The following pyramid shows the different levels of testing.

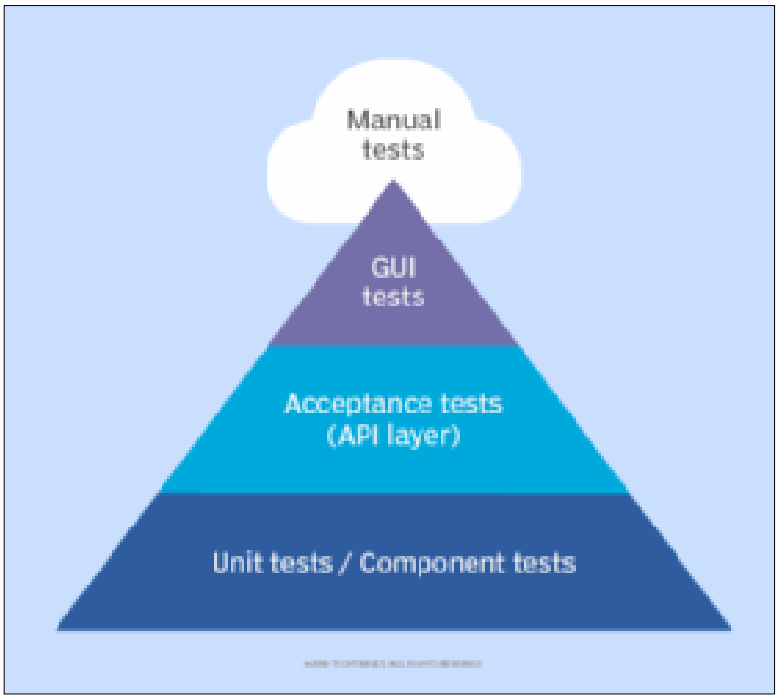


Figure I.4.3-1 Test levels pyramid

For each software, some of these tests will be more important than others, depending on why and by whom the system will be used. In our case we have chosen to work on analysis test

Tools: A code analysis test is a set of methods to get information about the program behavior without actually running the program. It is used to find programming errors

- **SonarQube:**
  Sonarqube (6) is an open-source platform to perform automatic reviews with static code analysis to detect bugs, code bugs, and security vulnerabilities

- **SonarLint:**
  SonarLint (7) is a plugin to perform SonarQube code analysis during development, directly in the environment (IDE).

Table I.4.3-I:Comparaison between Sonarqube and SonarLint

Tools	Advantages	Disadvantages
SonarQube	• Quality profiles for each project. • History of modifications is stored in any database . • Support for integration of personalized static analysis programs.	• Plugins for some languages are only available in the business versions. • Extra time for configuration and resources
SonarLint	• IDE integration.	• No reports available.

Summary:

According to Table, we will work with SonarQube as a code analysis testing solution because it handles complete analyses and gives a global view of the code quality.

I.4.4 Release

The release phase is an important step in the DevOps pipeline. This is the point at which we indicate that a release is ready to deploy to the production environment.

Definition: This phase includes the management of changes, version approvals, and release automation.

Tools:

Many continuous integration tools are currently available, with their qualities and shortcomings. Among them are Jenkins and GitLab CI:

• **Jenkins:** Jenkins (8) Is a continuous integration server that allows the whole team to have access to all the data, but also the history and the automation of all the production phases(Compilation, tests, deployment...)

• **Gitlab CI:** Gitlab CI (9) allows us to automate builds, tests, and deployments of our application

To better evaluate the solution, the tools described above will be judged according to the points mentioned in the table below :

Table I.4.4-I :Comparaison between Jenkins and Gitlab CI

Tools	Advantages	Disadvantages
Jenkins	• A large library of plugins. • Full control of the workspace. • Easy to deploy code. • Flexible and polyvalent functionality	• Complex plugin integration. • Insufficient analysis forglobal pipeline tracking.
GitLab CI	• Better Docker integration. • Easy to add tasks . • Good security and privacy policies. • Run parallel in phases	• Artifcats must be defined and uploaded for each job.

Summary :

Based on the comparison table mentioned above , we chose the GitLab CI solution because the latest features related to continuous integration make GitLab a serious to Jenkins

I.4.5 Deploy

In this phase each new change that is validated by testing is deployed to the production environment

A. Container Registry Tools:

Definition: This phase may include tools for managing, coordinating, planning, and automating the production of products that are based on containers as a registry

Tools: these are many Container Registry tools are currently available , with their qualities and shortcomings. Among them are DockerHub and Gitlab container registry

•**DockerHub**:
DockerHub (10) hosted repository service provided by docker for finding and sharing container images with your team. Key features include private Repositories: and push and pull container images.

• **Gitlab Container Registry:**
GitLab container registry (11) is a secure and private registry for docker images. Built on open source software. Gitlab Container Registry isn't just a standalone registry, it's completely integrated with GitLab.

Table A-I: Comparaison between DockerHub and Gitlab Container Registry

Tools	Advantages	Disadvantages
DockerHub	• Rapid deployment of applications • The application and its resources are secure and isolated. • Portability over multiple machines	• A container may fail at certain times. • Inability to use the different OS in the same physicalserver. • Default to the public makes it dangerous.
Gitlab container registry	• Excellent community support. • Fully integrated with GitLab • Dynamic server configuration.	• No cross-region replication • MFA for image Push/Pull • SLA Availability

B. Container Orchestration Tools :

Definition: Orchestration is the automation of the configuration, management, and synchronization of IT systems, applications, and services. Orchestration facilitates It's the ability to manage complex tasks and workflows.

Tools: To orchestrate services in production there are two basic options: use the Docker technology sack (Docker Compose, Swarm) or use an additional abstraction level provided by Google – Kubernetes (K8s).

Docker Swarm: Docker Swarm (12) enables you to manage clusters of Docker engines directly on the Docker platform. We can use the Docker CLI to create a cluster, deploy application services to the cluster, and manage the behavior of the cluster.

Kubernetes:
Kubernetes (13) Container orchestration software, commonly used to manage a high number of containers on physical infrastructure.

Table B-I:Comparaison between Docker Swarm and Kubernetes

Tools	Advantages	Disadvantages
Docker Swarm	• Quick and easy installation • No additional libraries or components and are built into any docker deployment • Provides automated load balancing within the docker containers.	• Provides limited functionality • Has a limited high availability and failure recovery capacity. • The open source community is much smaller than Kubernetes
Kubernetes	• Strong support by the CNCF • Large open source community • Convenient and powerful service management.	• Complex deployment • Requires separate administration tools • The transition from Docker Swarm to Kubernetes can be complicated and difficult to manage.

Summary:

The table below illustrates the comparison between the two deployment tools.

This comparison was made for academic purposes, the choice of the solution was made by the host company, which is the Gitlab container registry for a stage of the test environment and Kubernetes for the production environment

I.5 Conclusion:

This first chapter presents the preliminary stage of the general framework of our project. It consists of two main part: The presentation of the host company and the description of DevOps in Uptech.

II. Analysis and design

Introduction:

Through this chapter, we represent the needs analysis and the design, which is a fundamental step that precedes the realization of our project. We start with a needs analysis, . We then proceed to the definition of the actors and a global use case of the pipeline. And we finish by representing the infrastructure of the system and the interaction between these components.

II.1 Needs analysis

In this part , we start by analyzing the fundamental needs of our system through the functional needs part Then , we finish with the non-functional requirements to consider for this system.

II.1.1 Functional requirements

These are the functionalities to be provided by the pipeline:

- Automatic launch of job GitLab tasks following a merge request
- Maintaining all containerized versions of the application in a registry local
- User notification when a Gitlab-ci task fails
- Provide Feedback at each step of the automation
- Support project compilation and containerization
- Deployment of the application in a stage testing environment with docker compose
- Deployment automation with Kubernetes

II.1.2 Non-functional requirements

In addition to the functional requirements already mentioned, our continuous delivery system must respect several criteria that give a better quality of the obtained solution:

- **Performance**: the response time of the system (the different tools) must be minimal.
- **Extensibility**: the system must be able to support the evolution and the extensibility of its components, the possibility of adding nodes in case of load increase.
- **User-friendliness:** the interfaces of the tools must be intuitive and easy to understand, and allow users to achieve their objectives without ambiguity.

II.2 Global pipeline Use Case

In this part we identify the actors, then we define a use case diagram

II.2.1 Identification of the actors

- **Developer**: This is a user who can modify the source code for a given purpose (adding a feature, fixing an error, etc.)
- **System Administrator:** this is a user who can configure the environments of a project manually, In addition, he is responsible for monitoring the servers
- **Tester:** this is the user who evaluates the functionality of the application.
- **DevOps team:** this is the user who implements the CI/CD pipeline as well as the implementation of the environment and the actors who inherit the functionality of the developer also make deployment of the application to the production environment.

II.2.2 Use a case diagram

The use case diagram (*Figure II.2.2-1*) presents a global view of our continuous delivery solution.

This diagram lists the actors interacting with the system as well as the general use cases to avoid ambiguity, we detail each case in *Table II.2.2-I*

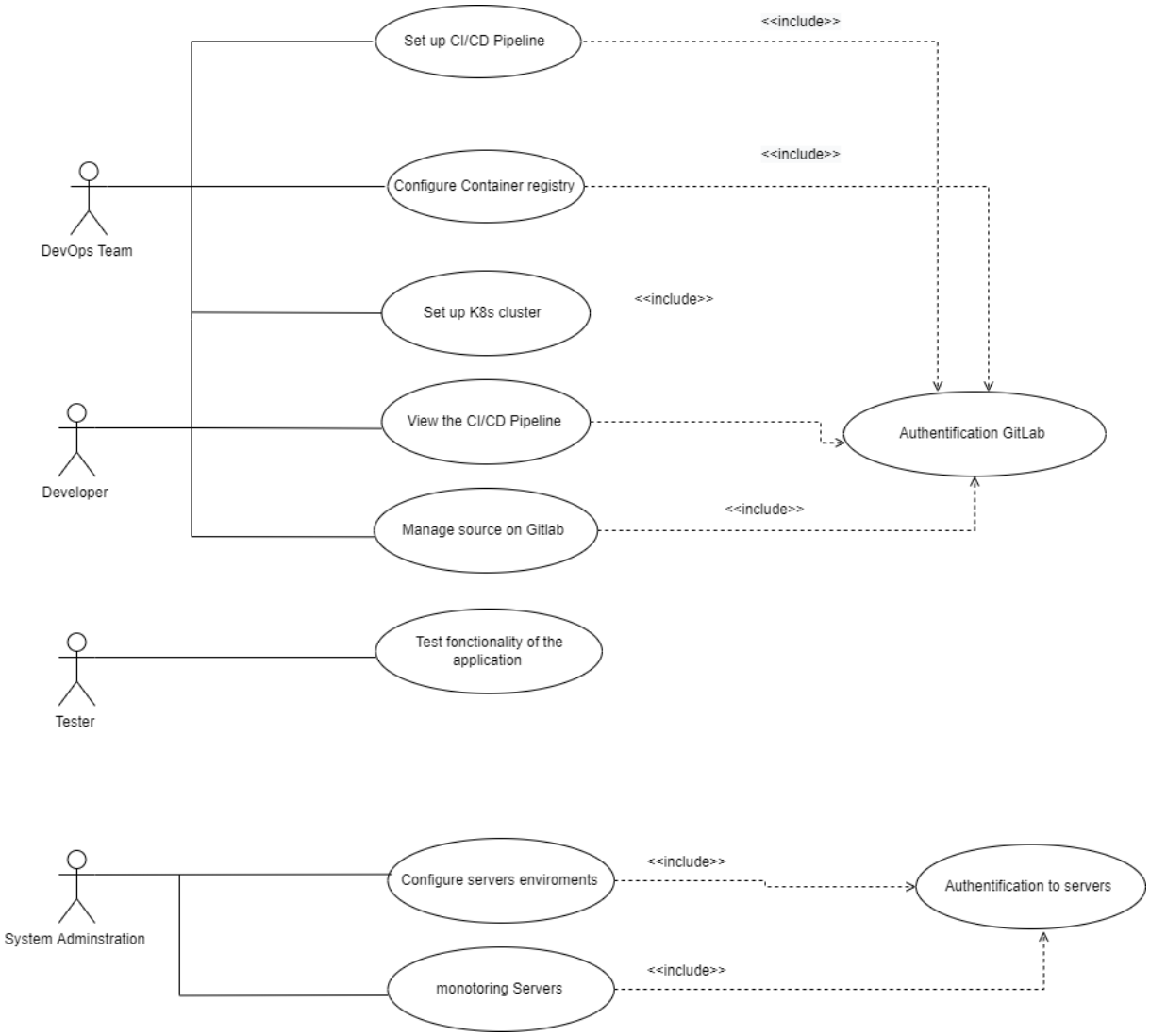


Figure II.2.2-2General use case diagram

Table II.2.2-II:Description of the Use Case

Use Cases	Description
Manage source code on GitLab	The developer can recover the source code, and push it on GitLab after modification
View the CI/CD pipeline	The developer and configurator can track the status of the CI/CD chain also can download the job artifacts
Configure Container Registry	The configurator enables the GitLab container registry to ensure the project can have its own space to store Docker images
Test functionality of the application	The tester can consult the application to test its functionality.
Set up CI/CD Pipeline	The configurator must set up a pipeline of the application

	which contains 4 steps: • Build: Building and testing application install all dependencies • Test: Go through all the unit tests as well as the sonar tests. • Package: Create a docker image for the application and send it to the registry. • Deploy: Deploy the docker image in the stage test environment and the Kubernetes cluster
Set up Kubernetes cluster	cluster configuration with all its components (installing with ansible)
Configure servers envrironment	System administrators configure and prepare the configuration of the servers
Monitoring servers	System administrators Track in-depth performance data about a server's status and health, this can include monitoring the CPU load memory utilization and disk fill levels**.**

II.3 Design of the CI/CD chain

Our conception of the CI/CD chain is based on functional architecture and physical architecture

II.3.1 Functional architecture of the CI/CD chain

- The developer will proceed to his commit: This includes checking the latest source code updates. These modifications are made locally, so you have to export the code to a source management server.
- Gitlab notices the new version available, and it launches the pipeline.
- Gitlab starts the build job and installs all dependencies and prepares the artifacts to download.
- Gitlab start unit tests and ensures some safety measure are used during the code, build, and development phases.
- Sonarqube analyzes the code and provides a detailed report of bugs, code smell, and vulnerabilities.
- Gitlab build docker image then pushed to Gitlab container registry
- Gitlab deploys the containerized image to the server test environment and the application will run with docker-compose
- Gitlab deploys the containerized images to the Kubernetes cluster

- In case of failure a notification will be generated to the developer's team and DevOps team , Developers affected by the error will update the repository and configuration management and fix the exception

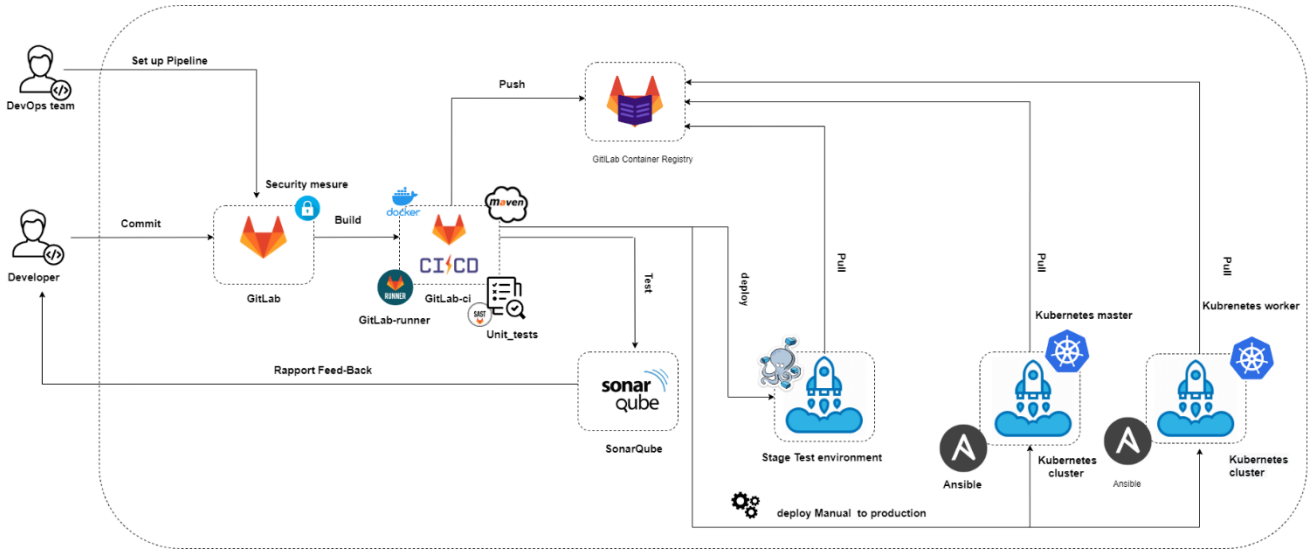


Figure II.3.1-1 Functional architecture of the CI/CD chain

II.3.2 Physical architecture of the CI/CD chain

The *Figure II.3.2-1* decribes the physical and virtual nodes of the system and their relationships

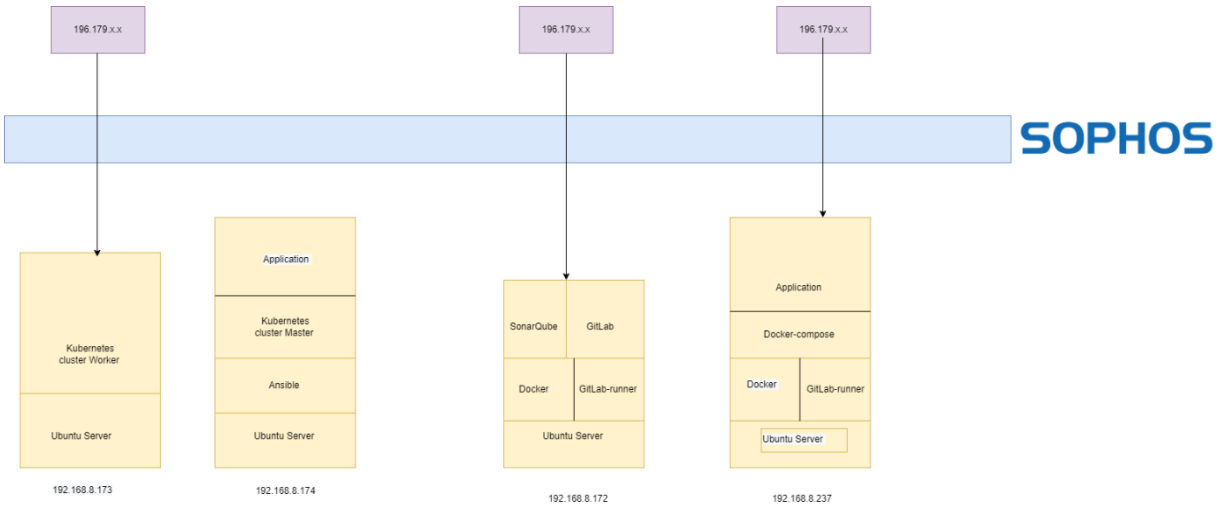


Figure II.3.2-2Physical architecture of the CI/CD chain

In the following we present the characteristics of the tools in the chain and how each one of them works and relate to other components.

II.4 Microservice for CI/CD pipeline:

In this part to ensure our pipeline corrects correctly we DevOps Teams with practicing with the developers team we make an application for testing the CI/CD Pipeline

II.4.1.1 Application Testing

Our Application will have one actor which is called " admin" who will take this takes

- Add Employees to the management system employees
- Edit and update Employee
- Delete Employee

II.4.1.2 Use a case diagram

This use case diagram will show more simplicity the role of actor

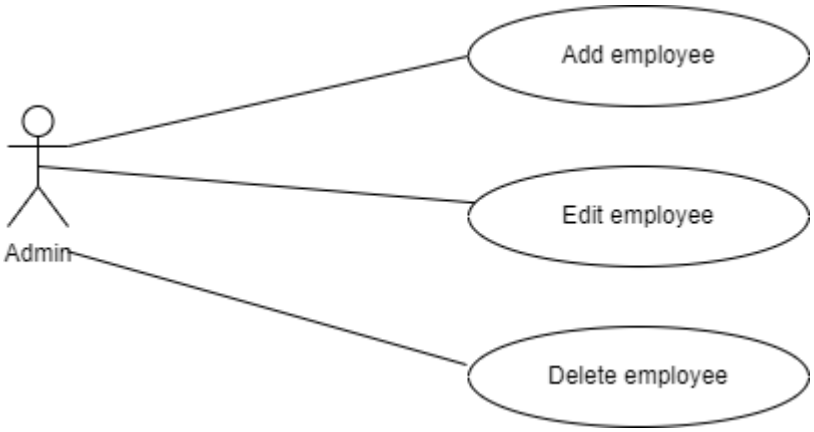


Figure II.4.1.2-1 Use Case diagram management system employees

II.4.1.3 Sequence Diagram

Our Sequence Diagram will explain more functionality of our application

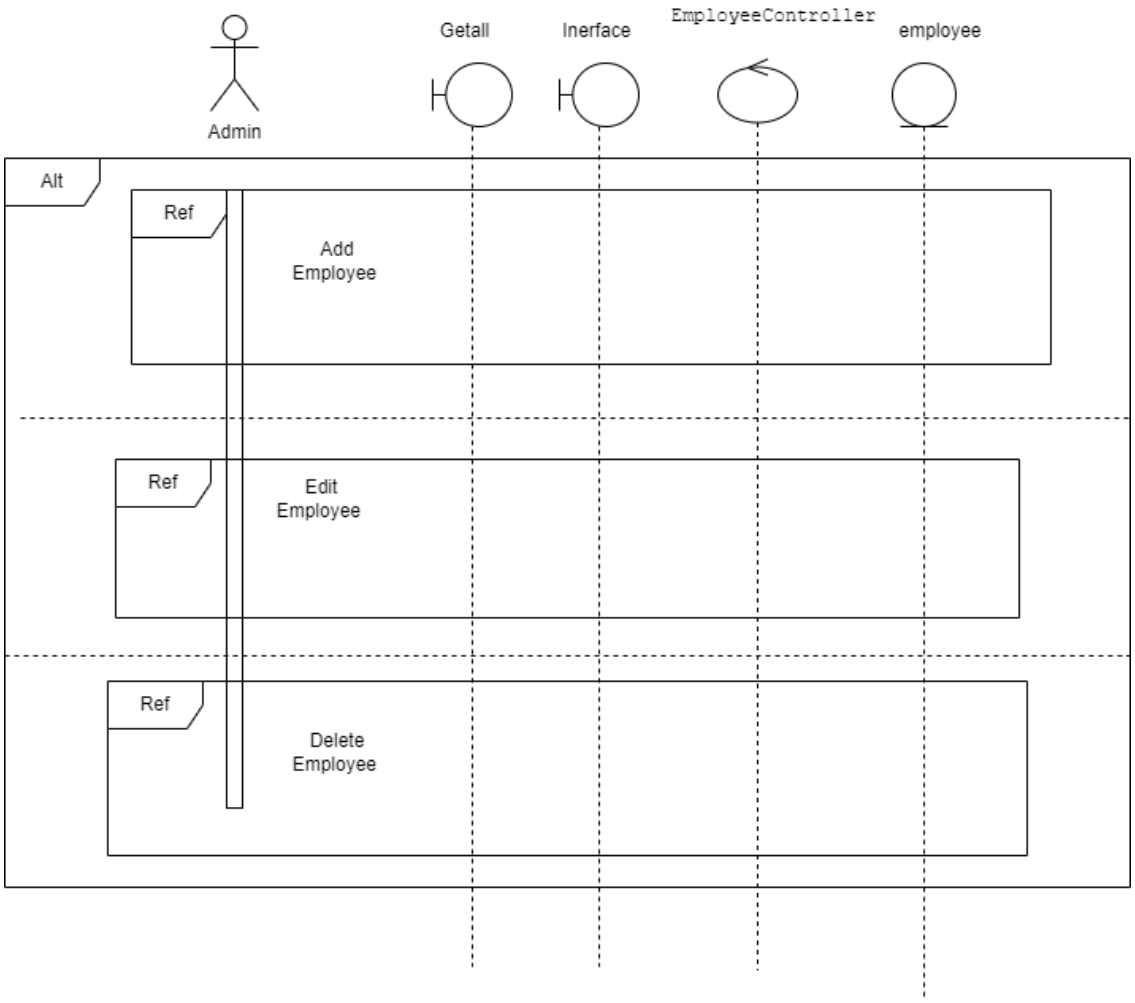


Figure II.4.1.3-1Sequence diagram management system employees

For more information about the sequence diagram of Add employee,Edit employee and delete employee check the Annexe

II.4.2 Microservice notion

Microservice (14) become the standard for modern cloud applications before microservice the application was packaged and deployed as monolithic was the standard architecture

II.4.2.1 Monolithic and microservice application

A monolithic (15) application is simply deployed on a set of identical servers behind a load balancer. In contrast, a microservice application typically consists of a large number of services. Each service will have multiple runtime instances. And each instance needs to be configured, deployed, scaled, and monitored. In monolithic, all components are part of a single unit .

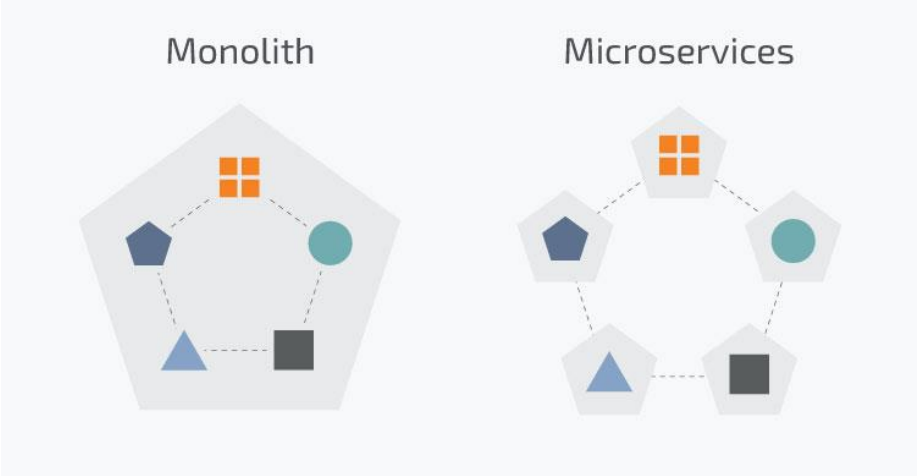


Figure II.4.2.1-1Microservice and monolith

II.4.2.2 CI/CD pipeline for microservice

We split our application into services each service had a specific job, self-contained and independent they will be developed and deployed scaled separately and each microservices has its own version also Each service has its own API by calling the respective API endpoint they can talk to each other.

The Application will be developed and deployed in a separate Git repository this notion is called Polyrepo (16).

Code is completely isolated, cloned, and worked on them separately also in Gitlab the CI/CD pipeline will be separated to better access control have complete isolation, and have smaller code bases and own pipeline.

This *Figure II.4.2.2-1* will explain more the relation between services and implementation in the pipeline

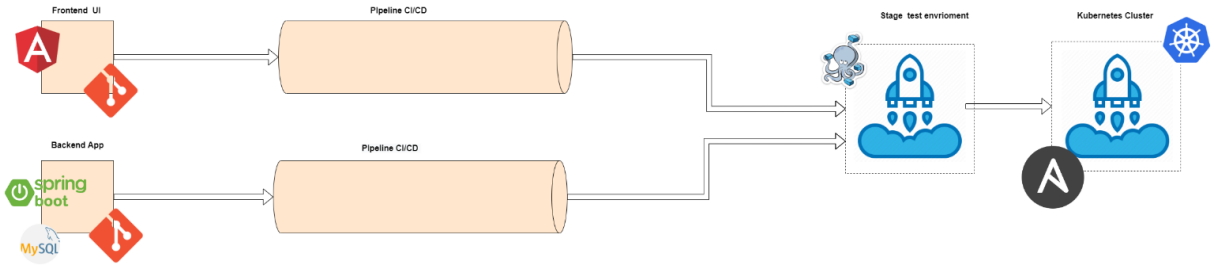


Figure II.4.2.2-2CI/CD pipeline Microservice

II.5 CI/CD pipeline steps

Developers practicing continuous integration merge their code changes into the main branch as often as possible. Changes made by the developer are validated by creating a build and running automated tests on it. This avoids the integration challenges that can occur when one waits until delivery day to merge changes into the release branch. Continuous integration focuses on automated testing to ensure that the app is working whenever new commits are integrated into the main branch.

II.5.1 Build stage:

With this single tool, we built a complete CI/CD pipeline that was both fast and reliable, We put it to use as a version control and continuous integration solution. We merge requests create code and implement CI/CD tools without having to use another tool because the Gitlab CI/CD it's really a single point of contact. Gitlab CI/CD uses Gitlab Runners to Perform builds Runners are isolated in virtual machines that use the Gitlab CI API to run commands and predefined actions compared to running on a single instance, this tool helps our projects to move through the pipeline more rapidly. All the actions that go into building our project, running linters and tests, and then deploying are all typical runner tasks.

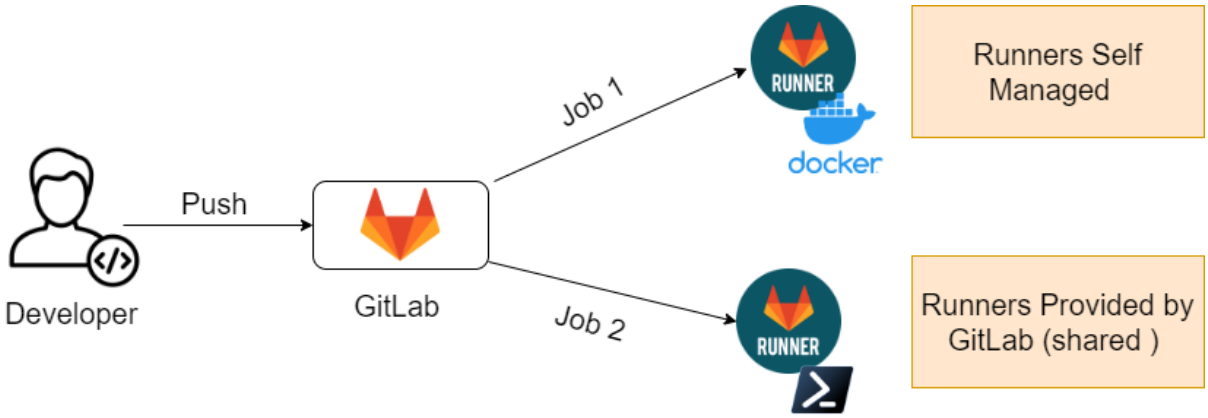


Figure II.5.1-1gitlab/runner workflow

Also to ensure safety measures we implement to the application some of the Security Testing tools (AST) (17) which is implemented in the build stage we use SAST (Static Application Security Testing) a tool in place to analyze proprietary or custom code for coding errors and design flaws that could lead to exploitable weaknesses. SAST tools are mainly used during the code, build, and development phases.

II.5.2 Test stage:

The test stage is set for code review is a software development process that measures the quality of a program's code. Code review has several benefits including detecting errors or bugs identifying security risks and ensuring that the code follows best practices. There are two forms of code review :

- Manual review is performed by a single tester
- Automated review is checks code for compliance with predefined set of rules or best practices.

Sonarqube is one of the Automated review tool gave us reports on duplicate code , coding standards , unit tests code coverage , comments bugs , code complexity and security tips among others . This figure shows the sonarqube workflow when the code is pushed the sonar analyzer is called to makes the analysis and then uploads it to Sonarqube, and then by this step the first job done by the gitlab runner it is stored in conatiners like all the jobs.

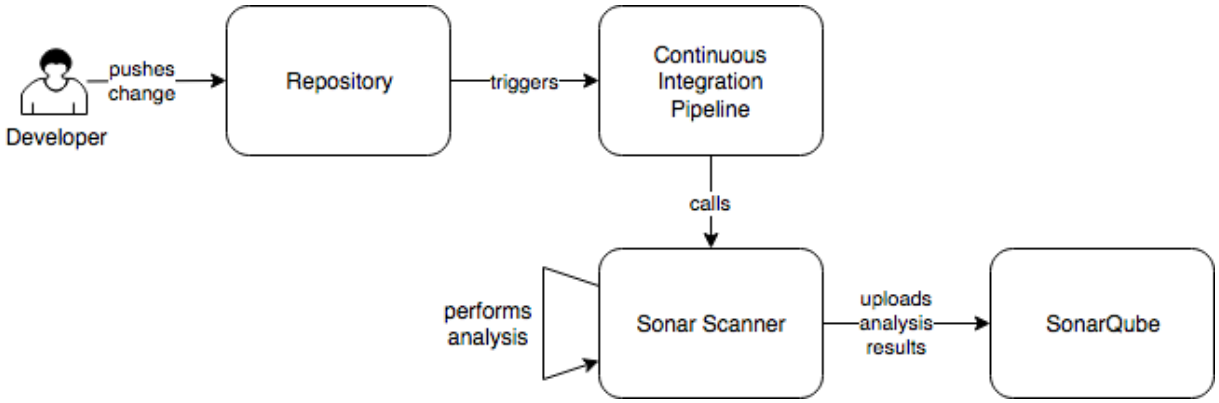


Figure II.5.2-1 Sonarqube workflow

II.5.3 Package stage:

In this stage, we will use Dockerfile to create our docker images which need interactive login to a Docker Daemon. The docker daemon takes root access on your machine to run, as you may have noticed when making images if we accomplish this inside a container, we will be our container must be run in a privileged mode in order to access a docker container of Databases in order to enter the node's filesystem and Docker daemon.

After creating Docker images w will make a pull private registry which is called Gitlab Container Registry (, Built on open-source software also it isn't a standalone registry it's completely integrated with GitLab.

Gitlab is all about having a single, integrated experience and our registry is no exception. You can now easily use images for Gitlab-CI to create images specific for tags or branches and mush more.

This *Figure II.5.3-1* will explain more package stage workflow

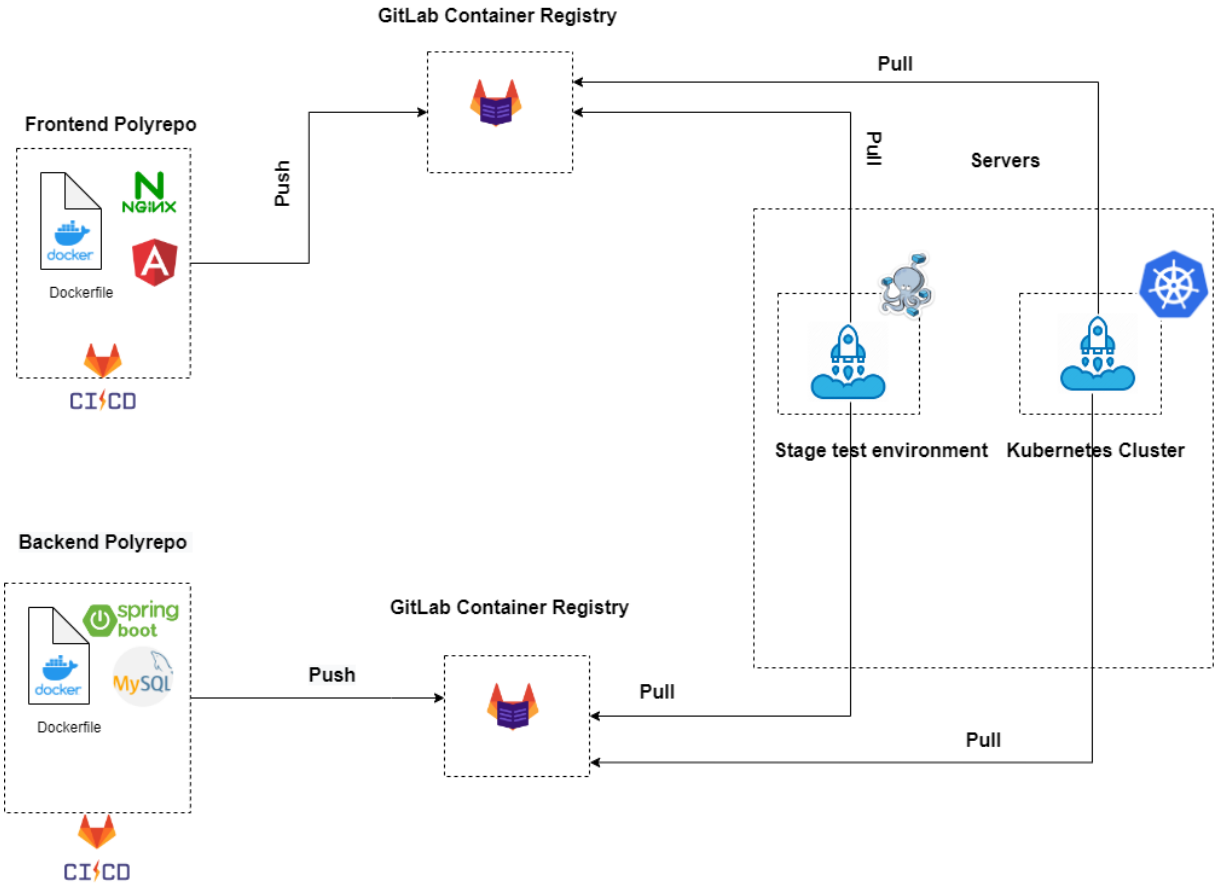


Figure II.5.3-2 Package Stage Workflow

II.5.4 Deploy stage:

In this stage we going to focus on how to deliver our code continuously to get into servers quickly and reliably . In the Deploy stage we will focus on two methods of deployment:

II.5.4.1 Continuous Deployment:

Continuous deployment goes further than continuous delivery with this practice every change that passes through all stages of the production pipeline is transmitted to clients. No human intervention is involved and only a failed test will prevent the deployment of a new change to production In order for our application to run in a stage test environment we used docker-compose (18) and pulled images of our application from the GitLab container registry; this figure will explain the workflow of continuous deployment

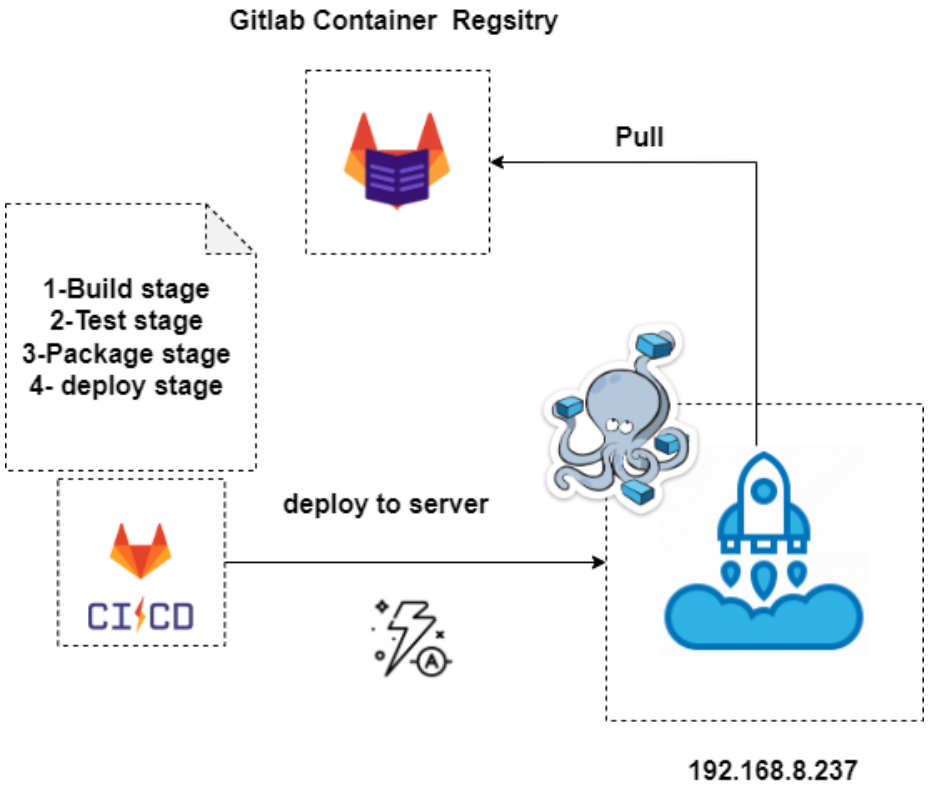


Figure II.5.4.1-1 deployment to environment test

II.5.4.2 Continuous Delivery :

Continuous Delivery is an extension of continuous integration that allows new changes to be released rapidly to customers in a sustainable way. This means that in addition to automated testing, one can also automate the release process by deploying an application at any time at the click of a button. It is partially going to environment production which is the Kubernetes cluster which is a set of nodes that run containerized applications. Containerizing applications package an app with its dependencies and some necessary services. They are more lightweight and flexible than virtual machines and in a production environment, we prepared Kubernetes cluster master and Kubernetes cluster workers using ansible. The figure above presents our proposal of the proposed environment architecture of deployment, and to obtain it it is enough to incorporate the installation of the software packages required on the declared Ansible host.

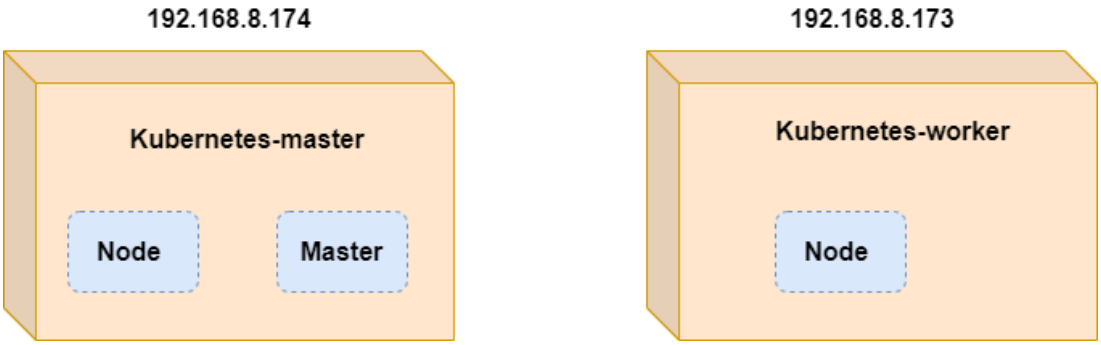


Figure II.5.4.2-1Proposed Architecture

In order to install a highly available Kubernetes infrastructure, and according to official documents, we have proposed the following architecture (*Figure II.5.4.2-2*)

Due to the limitation of temporary resources, the installation of the infrastructure is

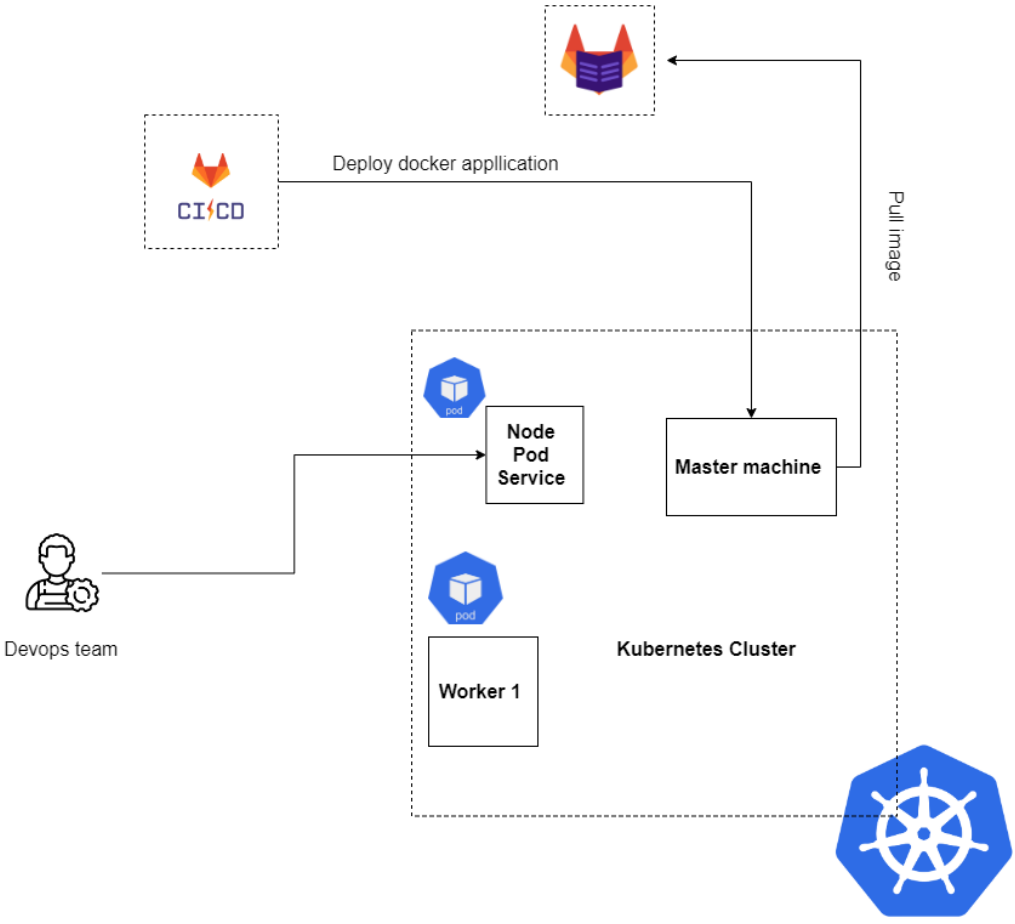


Figure II.5.4.2-3 Architecture of a highly available cluster

Table II.5.4.2-I:Description of Kubernetes components

Making up	Role
Master Machine	It is one or more physical or virtual machines with components Kubernetes allows the management of nodes and the orchestration of pods that are housed at the level of some knots. It uses the distributed directory the **etcdi** key/value for configuration sharing and service discovery.
Node	A node provides the runtime environment for containers. Each node provides the essential services so that it is visible to the master and so that it can manage
Server	Component on master that exposes the Kubernetes API. This is the front end for the Kubernetes control plane. It is designed for an upgrade horizontal scale, meaning it scales by expanding additional instances
Pod	A Kubernetes pod is a set of one or more Linux containers. It is the smallest unit of a Kubernetes application. A pod can contain multiple tightly coupled containers (advanced use case) or a single container (more common use case)
Worker	Worker nodes within the Kubernetes cluster are used to run containerized applications and manage networking to ensure that traffic between applications on the cluster and from outside the cluster can be properly facilitated.

II.6 Conclusion

This chapter first allowed us to propose our CI/CD architecture functional and physical and to specify the functional and non-functional needs of our system. Then, we identified the actors of our system. In the next chapter we present the realization of our project .

III. Realization

Introduction:

In this chapter , we will present the work accomplished. At first we will present material and project software environment. Next we installed and configured some of the necessary tools to prepare our environment to implement our CI/CD pipeline.

III.1 Work environment

We start by exploiting the existing infrastructure and we strengthen it by the addition of RAM and hard disks which are detailed in the table below and we four of those servers are connected to each other to make our pipeline runs

Table III.1-I:Computer characteristics

CPU	4vCPUs
Memory	8 GO
Hard disk	30 GO
Network adapter	VM Network (Connected)
OS	Ubuntu 20.04

III.1.1 Objectives

Our objective is to offer a complete solution based on DevOps concepts. Our solution translates as follows :

- Establish a stable environment in which more reliable applications are build
- Faster and more frequent delivery of updates and features.
- Process automation, continuous delivery to ensure a fluent flow of the SDLC.

In the next section , we will detail how these objective have been achieved

III.1.2 Preparation server

The first step in our project is to configure the server by installing tools and configuring a firewall to prepare our environment. Beginning with the creation of a new virtual machine VM are computing instances created by a program running on other machine that doesn't physically exist. The machine creating the VM is called the host machine and the VM is called a program, we can have many guest VMs on the host machine. A virtual server is a server-created b program. you can have many virtual servers running from one physical machine

- Choose the character of the virtual machine

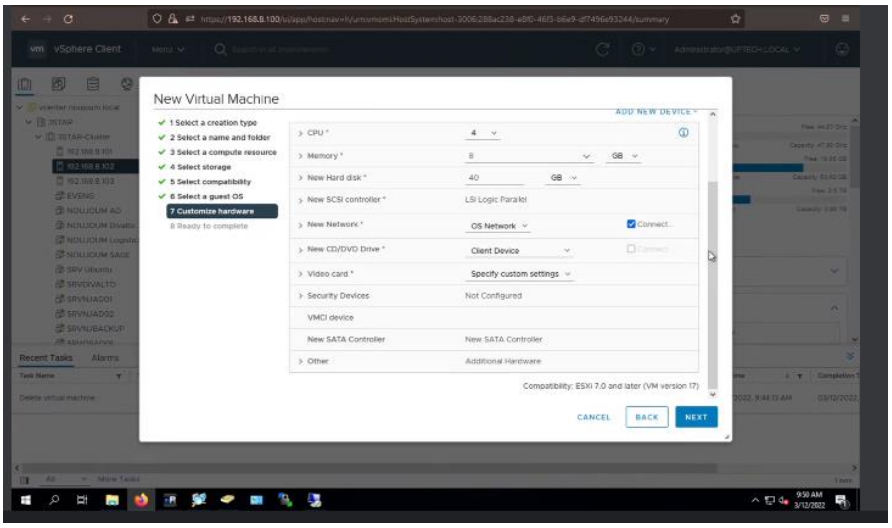


Figure III.1.2-1Creating new virtual machine

- Choose the ISO to install the system (OS)

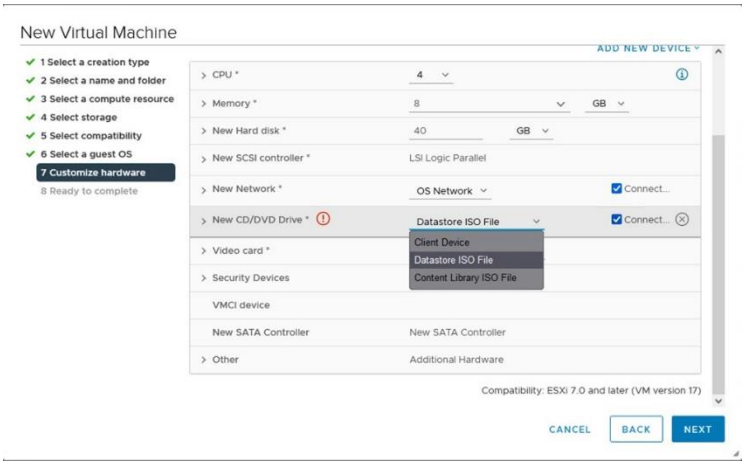


Figure III.1.2-2 Select ISO file

- Start the Virtual Machine

Figure III.1.2-3Machine Virtual ready

- Installing the system and configure the VM IP using the web console (19) , which is a web-based application that enables end-users to manage their data . the console is a self-service program that program that allows us to perform operations, including download , backup , restore and more for data management So we install web console to complete installing of the server

- Install the web console
- Install server
- Configure give a fixed IP
- Add user “ Linuxadmin “

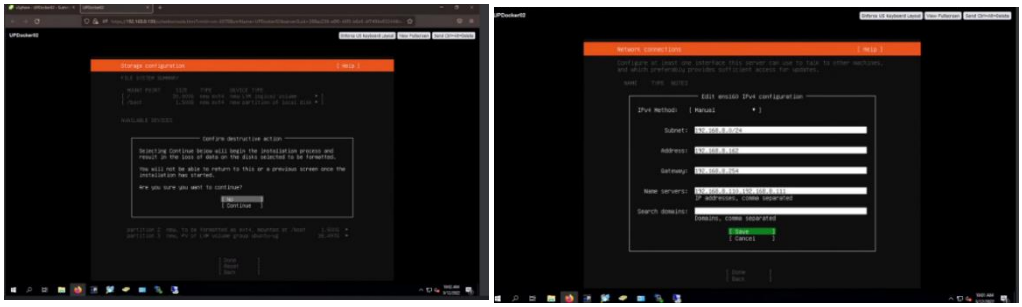


Figure III.1.2-4 Web console

- Installing docker we need to use SSH (Secure Shell is both a computer program and secure communication protocol. The connection protocol imposes an exchange of encryption keys at the start of the connection)
    - Connect By SSH
    - Use **apt-get update** to check for updates
    - Use Apt-get install docker

```
root@updocker02:/home/linuxadmin# apt-get install
Reading package lists... Done
Building dependency tree
Reading state information... Done
0 upgraded, 0 newly installed, 0 to remove and 10 not upgraded.
root@updocker02:/home/linuxadmin# apt-get install docker
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  wmdocker
The following NEW packages will be installed:
  docker wmdocker
0 upgraded, 2 newly installed, 0 to remove and 10 not upgraded.
Need to get 14.3 kB of archives.
After this operation, 58.4 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://tn.archive.ubuntu.com/ubuntu focal/universe amd64 wmdocker amd64 1.5-2 [13.0 kB]
Get:2 http://tn.archive.ubuntu.com/ubuntu focal/universe amd64 docker all 1.5-2 [1,316 B]
Fetched 14.3 kB in 4s (3,403 B/s)
Selecting previously unselected package wmdocker.
(Reading database ... 71718 files and directories currently installed.)
Preparing to unpack .../wmdocker_1.5-2_amd64.deb ...
Unpacking wmdocker (1.5-2) ...
Selecting previously unselected package docker.
Preparing to unpack .../archives/docker_1.5-2_all.deb ...
Unpacking docker (1.5-2) ...
Setting up wmdocker (1.5-2) ...
Setting up docker (1.5-2) ...
Processing triggers for man-db (2.9.1-1) ...
```

Figure III.1.2-5 install docker

- Updates the apt package index and install packages to allow apt to use a repository over HTTPS

```
Sudo apt-get install \
Ca-certificates \
Curl \
Gnupg \
Lsb-release
```

- Add the GPG key from the official Docker repository to your system.

```
$ curl – fsSL https://download .docker .com/linux/ubuntu/gpg | sudo apt -key add
```

- Configure the stable repository . to add the nightly or test repository , add the word nightly or test or both after the word stables

```
$ echo \
“ deb [ arch=$dpkg-print-architecture ) signed-by =/user/share/keyrings/docker-archive-keyring.gpg ]https://download.docker.com /linux/ubuntu  \
$( lsb release -cs) stable «  | sudo tee /etc/apt/sources.list.d/docker.list  > /dev/null
```

- Install docker

```
$Sudo apt-get update install docker-ce docker -ce -cli containered.io
```

- Verify that docker is installed correctly by running the hello-world image

```
$ systemclt start docker
$ systemclt enable docker
$ docker run hello-world
```

- Installing Gitlab on docker container by creating domain : devops.uptech.com.tn -> IP : 196.179.217.204 . when configuring the content delivery Network (CDN) for the first time , you must declare your domains from the OVH customer area and make the configurations allowing you use it optimally , the first step in this configuration is to add the subdomain of your choice to the CDN so that it accepts HTTPS (s) requests for this domain

- Install Gitlab inside a docker container

```
$ sudo docker run –detach \
    --hostname devops.uptech.com.tn \
 --publish 127.0.0.1:4443:443 --publish 127.0.0.1:4000:80 \
  --name devops \
  --restart always \
  --volume /srv/gitlab/config:/etc/gitlab \
 --volume /srv/gitlab/logs:/var/log/gitlab \
--volume /srv/gitlab/data:/var/opt/gitlab \
 gitlab/gitlab-ce:latest
```

- Install NGINX to started out as a web server designed for maximum performance and stability and configure ports of transaction that we can use with our container ports

Figure III.1.2-6 Nginx installed

- Serve Gitlab through HTTPS using the host's using Nginx and certbot (20) . Certbot is an automatic tool , compatible with major web servers (Nginx) as well as with the most common Linux Oses (including Debian Centos and Ubuntu) . It is placed under the aegis of the EFF . We need to install snapd and make sure you follow any instructions to enable classic snap support , snap are containerized software packages that are simple to create and install. They auto-update and are safe to run and because they bundle their dependencies they work on all major Linux system.

  - Ensure that the version of snapd is update by executing the following instructions on the command line on the machine to ensure that you have the latest version of snapd

```
$sudo snap install core ; sudo snap refresh core
```

  - Install Certbot by running this command on the machine

```
$sudo snap install –classic certbot
```

- Prepare the certbot command to ensure thet the Certobt command can be run

```
$ sudo  ln -s /snap/bin/certbot /usr/bin/certbot
$ sudo set certbot trust -plugin-with-root=ok
```

- Creating API keys for scripts using this plugin requires configuration file containing OVH API credentials for an account with the following access rules

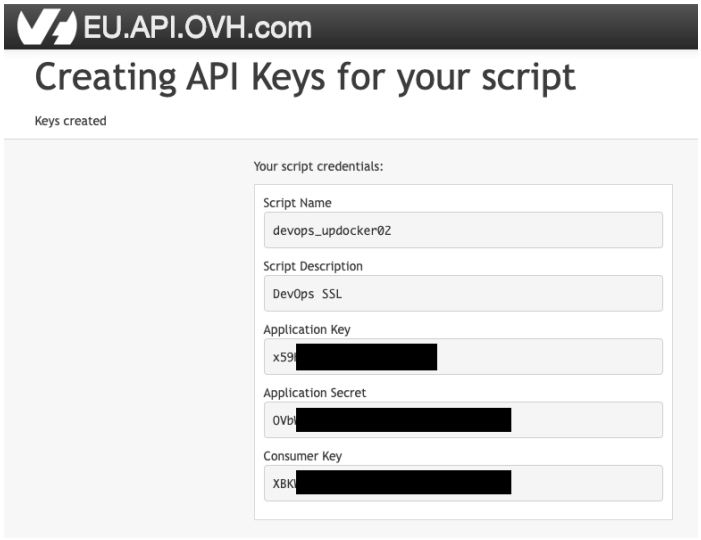


Figure III.1.2-7 API keys

- Installing Certbot

```
$ sudo snap install certbot-dns-ovh
$ nano ~/.secrets/certbot/ovh.ini
```

Add this information to this file ovh.ini :

```
# OVH API credentials used by Certbot
    dns_ovh_endpoint = ovh-eu

        dns_ovh_application_key = MDAwMDAwMDAwMDAw
        dns_ovh_application_secret =
MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
       dns_ovh_consumer_key =
MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
```

- Acquire a single cerrificate for both devops.uptech.com.tn and www.devops.uptech.com.tn

```
$ certbot certonly \
        --dns-ovh \
      --dns-ovh-credentials ~/.secrets/certbot/ovh.ini \
     -d devops.uptech.com.tn \
 -d www.devops.uptech.com.tn
```

- Run certbot by this command to get a certificate and have certbot edit nginx configuration automatically to serve it , tunring on HTTPS access in a single step

```
$ sudo certbot --nginx
```

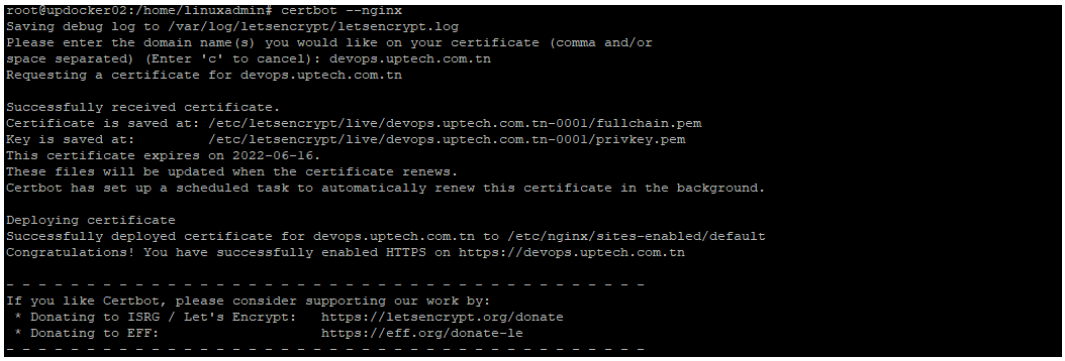


Figure III.1.2-8Certificate of our domain

- Edit the /etc/nginx/nginx.conf configuration root directory by modifying the following :

```
server {
    server_name devops.uptech.com.tn;
    client_max_body_size 256M;

    location / {
            proxy_pass http://localhost:4000;

            proxy_read_timeout 3600s;
            proxy_http_version 1.1;
            # Websocket connection
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection $connection_upgrade;
    }

    listen [::]:443;

    listen 443 ssl; # managed by Certbot
```

```
  ssl_certificate /etc/letsencrypt/live/git.domain.com/fullchain.pem; #
managed by Certbot
   ssl_certificate_key /etc/letsencrypt/live/git.domain.com/privkey.pem; #
managed by Certbot
        include /etc/letsencrypt/options-ssl-nginx.conf; # managed by
Certbot
        ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by
Certbot

}
```

- Connection header for Websocket reverse proxy

```
map $http_upgrade $connection_upgrade {
default upgrade;
''      close;
}
```

- Check our **www.devops.uptech.com.tn**

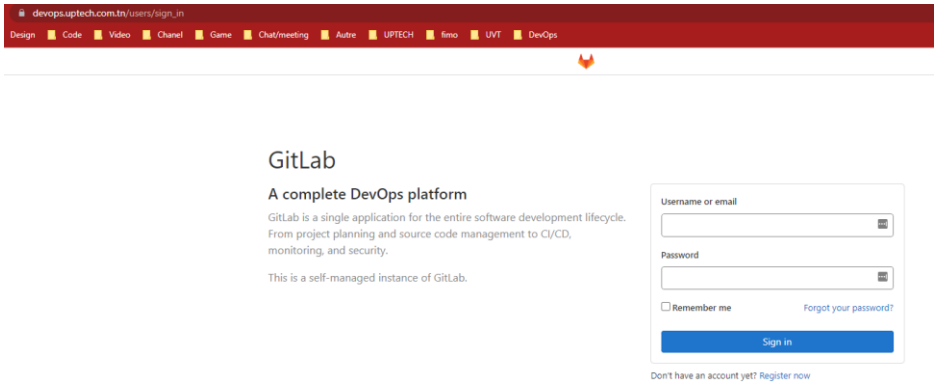


Figure III.1.2-9 Gitlab Server

- Enter to docker Container using **$docker ps** to know our container name

```
$ docker exec -it devops sh
```

- Reset the password of gitlab account

```
$ sudo gitlab-rake "gitlab:password:reset"
```

III.1.3 GitLab runner

First of all we configure the GitLab runner by installing the package on the system and starting the service that allows to running multiple runtime for different projects.

- Add the Gitlab Runner repository

```
wget -qO - https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh | sudo bash
```

- Install the GitLab runner :

```
sudo apt install -y gitlab-runner
```

- When the installation is finished, we can check the version of Gitlab Runner:

```
gitlab-runner –version
```

- Once the configuration is complete , we registered the Gitlab Runner under Linux Then we entered the URL which is the domain name of our GitLab server

```
linuxadmin@devops02:~$  sudo gitlab-runner register
[sudo] password for linuxadmin:
Runtime platform                                    arch=amd64 os=linux pid=2516122 revision=76984217 version=15.1.0
Running in system-mode.

Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
Enter the registration token:
GR1348941CWdGPkFUSo3puQzJTiAJ
Enter a description for the runner:
[devops02]: runner_app
Enter tags for the runner (comma-separated):
runner,test
Enter optional maintenance note for the runner:

Registering runner... succeeded                     runner=GR1348941CWdGPkFU
Enter an executor: kubernetes, docker-ssh, shell, ssh, docker-ssh+machine, docker+machine, custom, docker, parallels, virtualbox:
docker
Enter the default Docker image (for example, ruby:2.7):
ruby:2.6
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically reloaded!
```

Figure III.1.3-1 GitLab Runner Registration

- Shared runners are available for each project in a GitLab instance, so we've used them who have multiple tasks with similar requirements. Instead of having multiple slow runners for projects, we can have a few runners that manage multiple projects.
- we a registration token when configuring Gitlab-ci for this shared runner and enter a description and chose the docker executor then we entered the default docker image (ruby:2.1)
- now we can see the current runners for the Gitlab CI runner service

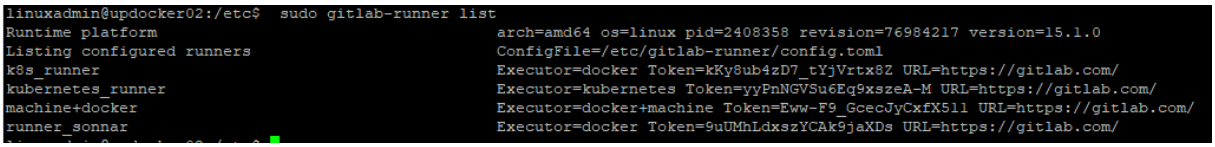


Figure 3-9: Gitlab Runner available list

III.1.4 GitLab container registry

Setting up our own registry allows us to push and pull images from our own private server, increasing security and reducing dependencies on external services in our workflow. GitLab will set up a container registry with only few configuration updates.

- We opened the file " gitlab.rb " by running this command

```
$ sudo nano /etc/gitlab/gitlab.rb
```

- In the " container registry " settings section . We expand "registry_external_url" and set it to our gitlab hostname with a port number

```
###################################################################################
## Container Registry settings
##! Docs: https://docs.gitlab.com/ee/administration/container_registry.html
###################################################################################

registry_external_url 'https://devops.uptech.com.tn:5050'
```

Figure III.1.4-1Container registry setting

- Then we enabled the registry and added the following two lines to indicate to the registry where to find our certificates:

```
### Settings used by Registry application
registry['enable'] = true

registy_nginx['ssl_certificate'] = "/etc/gitlab/ssl/devops.uptech.com.tn.crt"
registry_nginx['ssl_certificate_key'] = "/etc/gitlab/ssl/devops.uptech.com.tn.key"
```

Figure III.1.4-2Container registry activation

- Then we reconfigured our file to enable the new options with this command:

```
$ sudo gitlab-ctl reconfigure
```

- After enabling the container registry options , we can now find the private container registry in Gitlab:

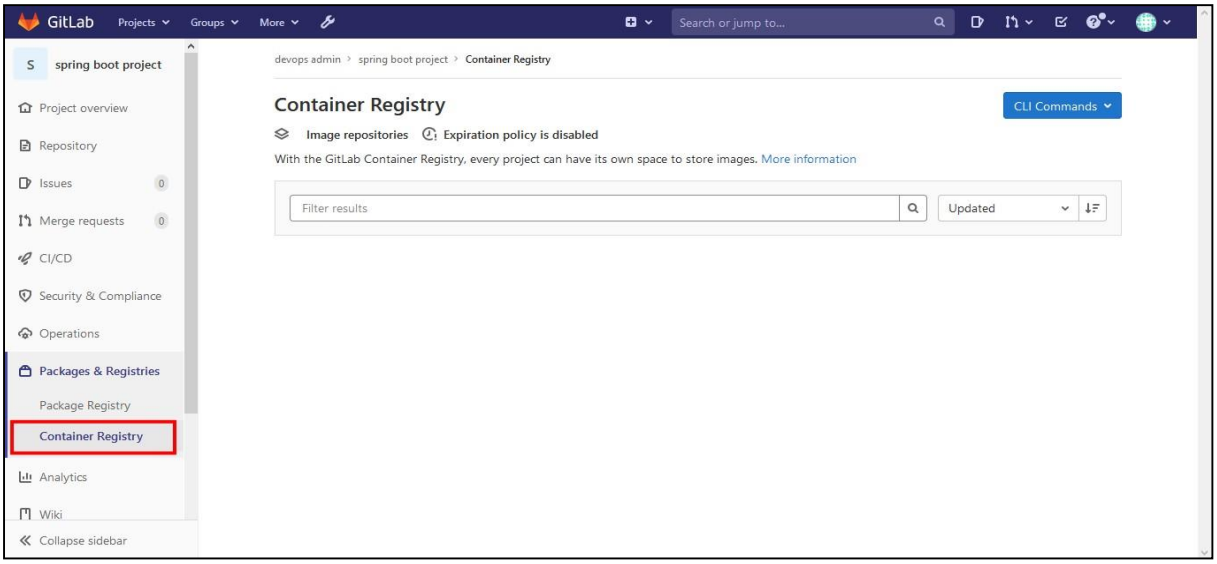


Figure III.1.4-3Container Registry interface

III.1.5 Setting up a Kubernetes with ansible

Subsequently, we added the list of target hosts in the file /etc/ansible/hosts.

We finally pinged all the servers we configured by typing the command: **ansible all-m ping**

Figure III.1.5-1 shows the results of testing relative inventory groups (hosts) to Ansible

This display testifies to the good connectivity and the execution of the installation task

```
root@worker01:/home/linuxadmin/kubernetes# ansible -i hosts all -m ping
master01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
worker01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

Figure III.1.5-2 Access test with the ping module to hosts managed by Ansible

- Creating a Kubernetes user with an ansible-playbook our first task in setting up the Kubernetes cluster is to create a new user on each node . This will be a non-root user , that has sudo privileges. It's a good idea not use the root account of daily operations, of course. We can use Ansible to set up the

account on all three nodes, quickly and easily. First, create a file in the working directory :

```
- hosts: 'workers, masters'
  become: yes

  tasks:
    - name: create the kube user account
      user: name=kube append=yes state=present createhome=yes shell=/bin/bash

    - name: allow 'kube' to use sudo without needing a password
      lineinfile:
        dest: /etc/sudoers
        line: 'kube ALL=(ALL) NOPASSWD: ALL'
        validate: 'visudo -cf %s'

    - name: set up authorized keys for the kube user
      authorized_key: user=kube key="{{item}}"
      with_file:
        - ~/.ssh/id_rsa.pub
```

Figure III.1.5-3: users.yml

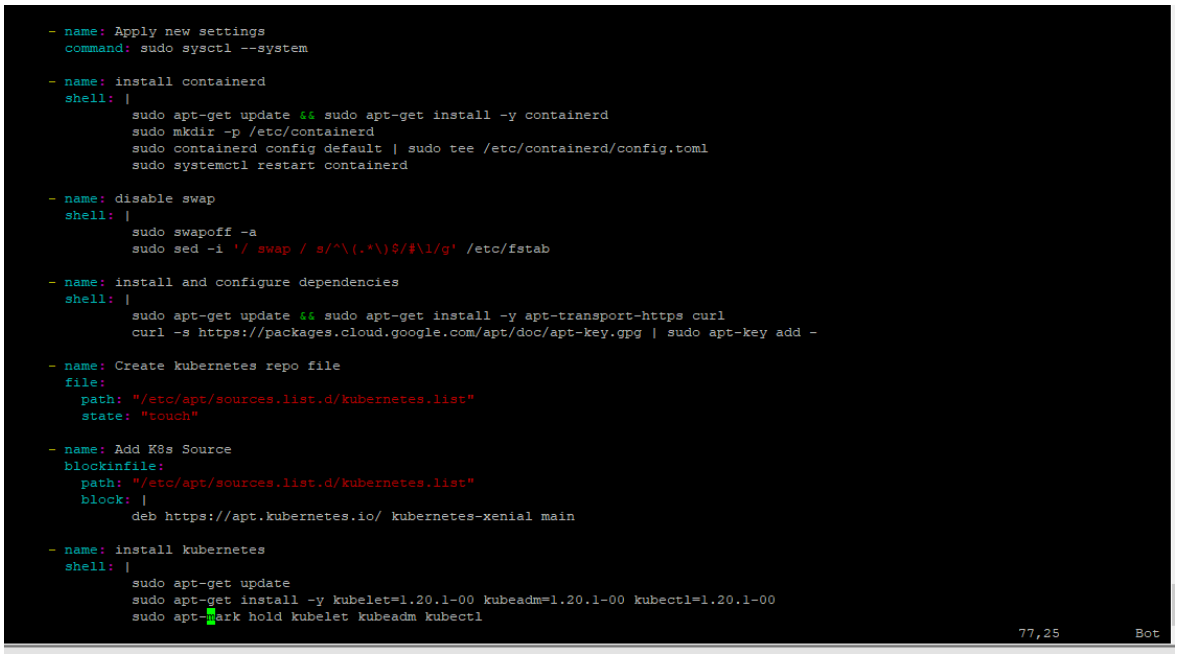
- We 're now ready to run our first playbook to do so :

$ ansible-playbook -i hosts users.yml

```
root@worker01:/home/linuxadmin/kubernetes# ansible-playbook -i hosts users.yml -K
BECOME password:

PLAY [workers, masters] ********************************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************************************
ok: [master01]
ok: [worker01]

TASK [create the kube user account] ********************************************************************************************************
[WARNING]: 'append' is set, but no 'groups' are specified. Use 'groups' for appending new groups.This will change to an error in Ansible
2.14.
changed: [worker01]
changed: [master01]

TASK [allow 'kube' to use sudo without needing a password] *********************************************************************************
changed: [master01]
changed: [worker01]

TASK [set up authorized keys for the kube user] ********************************************************************************************
changed: [master01] => (item=ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCuPfAWwYfTzgL0VvRxaJE33bGnTZW8qtA/eiJtC1ha5tZ2WO0oNas0EZvSlybba//mZtvMFN
/I6HJWp6MfJiC0Od8WO2n5lGXn5klXP4bgMR42yCSuNwuJ2NAFgiBm6RO85ZlagwSfuQ3nOUcapvKjGALO434QWQg2DOTorgMtbsmx0JyXSH5nrimFbJOrUoL+slXUMmqeFmZ62Skaa
n9kC+w4PUHBbMJi07eqbELuI2fKcMOcS7dMOYtjaNP4sRDspbDErU7gUVKUgidbcSO01A+3sdMj9Fzh5ondVdZfulpjsgfLWwLmND2+XK97e8PQ0hZUwXjla8salwtHEE6r4LjVAIPE
MzyjgKr/zf5E5xE8FJXlcLaYcyGZlV/MBwnKXhCArQGwL1RDRVpI2SHAmdqmtnO/DtBZ00TIZbXZiIEhPDzEV+nKvrls/B36eU9vjhn0sD2BHqPVaw5Ax3L4Z3JSnvEDE+qiyoO7DpW
fhil6dlfNJlHu67G2PyBSoy8= root@worker01)
changed: [worker01] => (item=ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCuPfAWwYfTzgL0VvRxaJE33bGnTZW8qtA/eiJtC1ha5tZ2WO0oNas0EZvSlybba//mZtvMFN
/I6HJWp6MfJiC0Od8WO2n5lGXn5klXP4bgMR42yCSuNwuJ2NAFgiBm6RO85ZlagwSfuQ3nOUcapvKjGALO434QWQg2DOTorgMtbsmx0JyXSH5nrimFbJOrUoL+slXUMmqeFmZ62Skaa
n9kC+w4PUHBbMJi07eqbELuI2fKcMOcS7dMOYtjaNP4sRDspbDErU7gUVKUgidbcSO01A+3sdMj9Fzh5ondVdZfulpjsgfLWwLmND2+XK97e8PQ0hZUwXjla8salwtHEE6r4LjVAIPE
MzyjgKr/zf5E5xE8FJXlcLaYcyGZlV/MBwnKXhCArQGwL1RDRVpI2SHAmdqmtnO/DtBZ00TIZbXZiIEhPDzEV+nKvrls/B36eU9vjhn0sD2BHqPVaw5Ax3L4Z3JSnvEDE+qiyoO7DpW
fhil6dlfNJlHu67G2PyBSoy8= root@worker01)

PLAY RECAP *********************************************************************************************************************************
master01                   : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
worker01                   : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

root@worker01:/home/linuxadmin/kubernetes#
```

Figure III.1.5-4: users prepared

- With our users now created we can move on to installing Kubernetes . Let's dive straight in and have a look at the playbook, which I have named install-k8s.yml



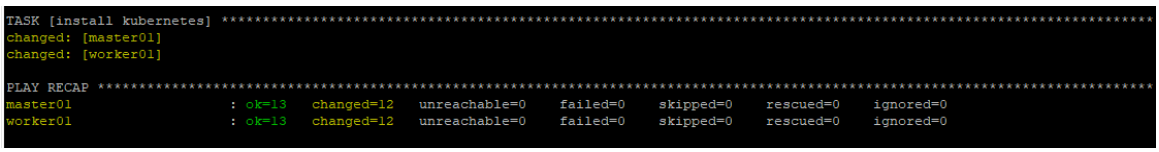
```
- name: Apply new settings
  command: sudo sysctl --system

- name: install containerd
  shell: |
         sudo apt-get update && sudo apt-get install -y containerd
         sudo mkdir -p /etc/containerd
         sudo containerd config default | sudo tee /etc/containerd/config.toml
         sudo systemctl restart containerd

- name: disable swap
  shell: |
         sudo swapoff -a
         sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab

- name: install and configure dependencies
  shell: |
         sudo apt-get update && sudo apt-get install -y apt-transport-https curl
         curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

- name: Create kubernetes repo file
  file:
    path: "/etc/apt/sources.list.d/kubernetes.list"
    state: "touch"

- name: Add K8s Source
  blockinfile:
    path: "/etc/apt/sources.list.d/kubernetes.list"
    block: |
          deb https://apt.kubernetes.io/ kubernetes-xenial main

- name: install kubernetes
  shell: |
         sudo apt-get update
         sudo apt-get install -y kubelet=1.20.1-00 kubeadm=1.20.1-00 kubectl=1.20.1-00
         sudo apt-mark hold kubelet kubeadm kubectl
                                                                                    77,25         Bot
```

Figure III.1.5-5install-k8s.yml

- This playbook will run against all three nodes and will install the containerd runtime (including some pre-requisite configuration, then go into install Kubernetes, which includes kubelet, kubeadm and kubectl , run the playbook using the following syntax :

```
$ ansible-playbook -i hosts  install-k8s.yml
```

- There's quite a lot going on here, so this one will take a little while to run whist the necessary packages are installed on each node. Once done you should see :



```
TASK [install kubernetes] ******************************************************************************************************************************
changed: [master01]
changed: [worker01]

PLAY RECAP *********************************************************************************************************************************************
master01                   : ok=13   changed=12   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
worker01                   : ok=13   changed=12   unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Figure III.1.5-6K8s installed

- Now we should have containerd and Kubernetes installed on all our nodes , the next step is to create the cluster on the master node.This is the master.yml file which will initialize the Kubernetes cluster on my master node and set up the pod network using calico :

```
    chdir: $HOME
    creates: cluster_initialized.txt

- name: create .kube directory
  become: yes
  become_user: kube
  file:
    path: $HOME/.kube
    state: directory
    mode: 0755

- name: copies admin.conf to user's kube config
  copy:
    src: /etc/kubernetes/admin.conf
    dest: /home/kube/.kube/config
    remote_src: yes
    owner: kube

- name: install Pod network
  become: yes
  become_user: kube
  shell: kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
  args:
    chdir: $HOME

- name: Get the token for joining the worker nodes
  become: yes
  become_user: kube
  shell: kubeadm token create  --print-join-command
  register: kubernetes_join_command

- debug:
  msg: "{{ kubernetes_join_command.stdout }}"

- name: Copy join command to local file.
  become: yes
  local_action: copy content="{{ kubernetes_join_command.stdout_lines[0] }}" dest="/tmp/kubernetes_join_command" mode=0777
```

Figure III.1.5-7 master.yml

- Note, towards the end of the playbook we generate the worker join command and save it to a local file on the Ansible host. We will use this file later to join the worker nodes to the cluster. Before then, execute the masters.yml playbook:

```
$ ansible-playbook -i hosts  master.yml
```

- There's quite a lot going on here, so this one will take a little while to run whist the necessary packages are installed on each node. Once done you should see :

```
root@worker01:/home/linuxadmin/kubernetes# vi n.yml
root@worker01:/home/linuxadmin/kubernetes# ansible-playbook -i hosts n.yml -K
BECOME password:

PLAY [masters] *****************************************************************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************************************************************
ok: [master01]

TASK [Start the cluster] *******************************************************************************************************************************************
ok: [master01]

TASK [create .kube directory] **************************************************************************************************************************************
ok: [master01]

TASK [copies admin.conf to user's kube config] *********************************************************************************************************************
ok: [master01]

TASK [install Pod network] *****************************************************************************************************************************************
ok: [master01]

TASK [Get the token for joining the worker nodes] *******************************************************************************************************************
changed: [master01]

TASK [Copy join command to local file.] ****************************************************************************************************************************
changed: [master01 -> localhost]

PLAY RECAP *********************************************************************************************************************************************************
master01                   : ok=7    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

root@worker01:/home/linuxadmin/kubernetes# vi k.yml
```

Figure III.1.5-8 Kubernetes cluster master done

- Now we have a Kubernetes cluster initialised, the final step is to join our worker nodes to the cluster. To do so, the final playbook – join-workers.yml – contains the following:

```
- hosts: 'workers, masters'
  become: yes

  tasks:
    - name: create the kube user account
      user: name=kube append=yes state=present createhome=yes shell=/bin/bash

    - name: allow 'kube' to use sudo without needing a password
      lineinfile:
        dest: /etc/sudoers
        line: 'kube ALL=(ALL) NOPASSWD: ALL'
        validate: 'visudo -cf %s'

    - name: set up authorized keys for the kube user
      authorized_key: user=kube key="{{item}}"
      with_file:
        - ~/.ssh/id_rsa.pub
```

Figure III.1.5-9 workers.yml

- This work by copying the file containing the worker join command saved locally earlier to the worker nodes, then it runs the command. Run the playbook with :

```
root@worker01:/home/linuxadmin/kubernetes# ansible-playbook -i hosts k.yml -K
BECOME password:

PLAY [workers] *****************************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************************
ok: [worker02]
ok: [worker01]

TASK [Copy join command from Ansiblehost to the worker nodes.] *****************************************************************
ok: [worker02]
ok: [worker01]

TASK [Join the Worker nodes to the cluster.] ***********************************************************************************
changed: [worker02]
changed: [worker01]

PLAY RECAP *********************************************************************************************************************
worker01                   : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
worker02                   : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Figure III.1.5-10 Kubernetes cluster workers

- Once the playbook has complete we can check the status of the cluster nodes by again running the following on the cluster master node:

```
root@worker01:/home/linuxadmin/kubernetes# kubectl get nodes
NAME          STATUS   ROLES                  AGE     VERSION
master01      Ready    <none>                 2m25s   v1.20.1
updocker02    Ready    <none>                 2m25s   v1.20.1
worker01      Ready    control-plane,master   18m     v1.20.1
root@worker01:/home/linuxadmin/kubernetes#
```

Figure III.1.5-11 nodes ready

III.1.6 Implementation of the CI/CD chain

We will start the functional realization of the CI/CD chain implementation as the configuration result following the pipeline launch , our application “ management system employees “ will take multiple stages to deploy to the servers.

The core of CI/CD using Gitlab is the GitLab-ci.yml file located at the root of the project, This file describes the steps that the runner follows to set up CI/CD chain, Each modification triggers the execution of the pipelines.

We will start by backend part this *Figure III.1.6-1* will show our CI/CD chain configuration which contains the build , quality , integration , package , deploy of a spring boot java application .

```yaml
image: docker:latest
services:
  - docker:dind
variables:
  MAVEN_OPTS: "-Dhttps.protocols=TLSv1.2 -Dmaven.repo.local=$CI_PROJECT_DIR/.m2/repository -Dorg.slf4j.simpleLogger.log.org.apache.maven.cli.transfer.Slf4jMaven
  MAVEN_CLI_OPTS: "--batch-mode --fail-at-end --show-version -Pproduction"
  DOCKER_DRIVER: overlay
  SPRING_PROFILES_ACTIVE: gitlab-ci
  USER_GITLAB: sanjaybsm
  APP_NAME: springbootgitlab
  REPO: springbootlabtest
  DOCKER_HOST: tcp://docker:2375/
  DOCKER_TLS_CERTDIR: ""
cache:
    key: "$CI_JOB_NAME"
    paths:
        - .m2/repository
stages:
  - build
  - quality
  - integration
  - package
  - deploy
```

Figure III.1.6-2 Gitlab-ci.yml file script for the backend

In the .gitlab-ci.yml file, we will run the build task on the docker docker :latest image. The runner host in this case has no need to have docker and java installed.

The next two lines are the "MAVEN_OPTS" and "MAVEN_CLI_OPTS" variables, Thesis variables contain Maven configuration options.

The "MAVEN_OPTS" environment variable contains the settings used when starting the JVM in which Maven will run. These settings are always applied when Maven is run.

The "MAVEN_CLI_OPTS" environment variable contains common, optional Maven command line parameters that can be included when running a build.

The cache keyword is used to speed up all our steps. Indeed, in the case of a Java compilation with Maven, this compilation recovers a lot of dependencies and external libraries. Thesis libraries are stored in the .m2 directory.

- Using the "cache" keyword and the GitLab predefined variable "$CI_JOB_NAME", allows to share a cache between branches, but have a unique cache for each job.

III.1.6.1 Build stage

The build part is divided into two parts , the script part and the artefact part. First part defines the script lines to run in order to execute the build stage. We launch the build by downloading the Maven build tool, all the application dependencies, and launching the project build. The “$MAVEN_CLI_OPTS” validate is another option that defines to speed up the processing time, The second part defines the job artifacts.

```
maven-build:
  image: maven:3-openjdk-11
  stage: build
  services:
  - docker:dind
  before_script:
    - echo "mvn clean"
    - mvn clean
  script:
    - echo "mvn compile"
    - mvn install compile
  artifacts:
    paths:
      - target/*.jar
  tags :
    - backend
    - docker
```

Figure III.1.6.1-1:backend Build stage

Implementation of SAST Tools in our pipeline by using job templates will added a existing CI/CD workflow , These are specific job provided by Gitlab we will ensure code note working and bad performance , Security issues

```

include:
  - template: Jobs/SAST.gitlab-ci.yml # Securities/xxx before version 14.
```

Figure III.1.6.1-2: include template SAST

The template had many tests like analyzing the source code to identify any vulnerabilities

SAST.gitlab-ci.yml 7.60 KiB — Open in Web IDE | Lock | Replace | Delete

```yaml
# Read more about this feature here: https://docs.gitlab.com/ee/user/application_security/sast/
#
# Configure SAST with CI/CD variables (https://docs.gitlab.com/ee/ci/variables/index.html).
# List of available variables: https://docs.gitlab.com/ee/user/application_security/sast/index.html#available-variables

variables:
  # Setting this variable will affect all Security templates
  # (SAST, Dependency Scanning, ...)
  TEMPLATE_REGISTRY_HOST: 'registry.gitlab.com'
  SECURE_ANALYZERS_PREFIX: "$TEMPLATE_REGISTRY_HOST/security-products"
  SAST_IMAGE_SUFFIX: ""

  SAST_EXCLUDED_ANALYZERS: ""
  SAST_EXCLUDED_PATHS: "spec, test, tests, tmp"
  SCAN_KUBERNETES_MANIFESTS: "false"

sast:
  stage: test
  artifacts:
    reports:
      sast: gl-sast-report.json
  rules:
    - when: never
  variables:
    SEARCH_MAX_DEPTH: 4
  script:
    - echo "$CI_JOB_NAME is used for configuration only, and its script should not be executed"
    - exit 1
```

Figure III.1.6.1-3SAST Template

Once the Pipeline is executed by the runners we configured in the previous steps . We can find the pipeline runs and their history in our project's CI/CD

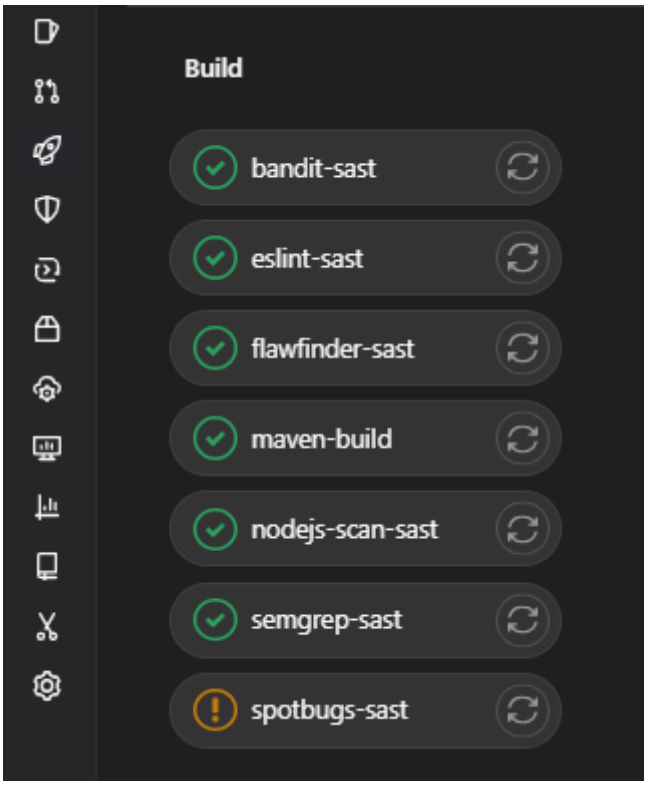


Figure III.1.6.1-4 Gitlab CI stage build status backend

We can follow each stage and its instructions in a console *Figure III.1.6.1-5* show the console out put in the build step

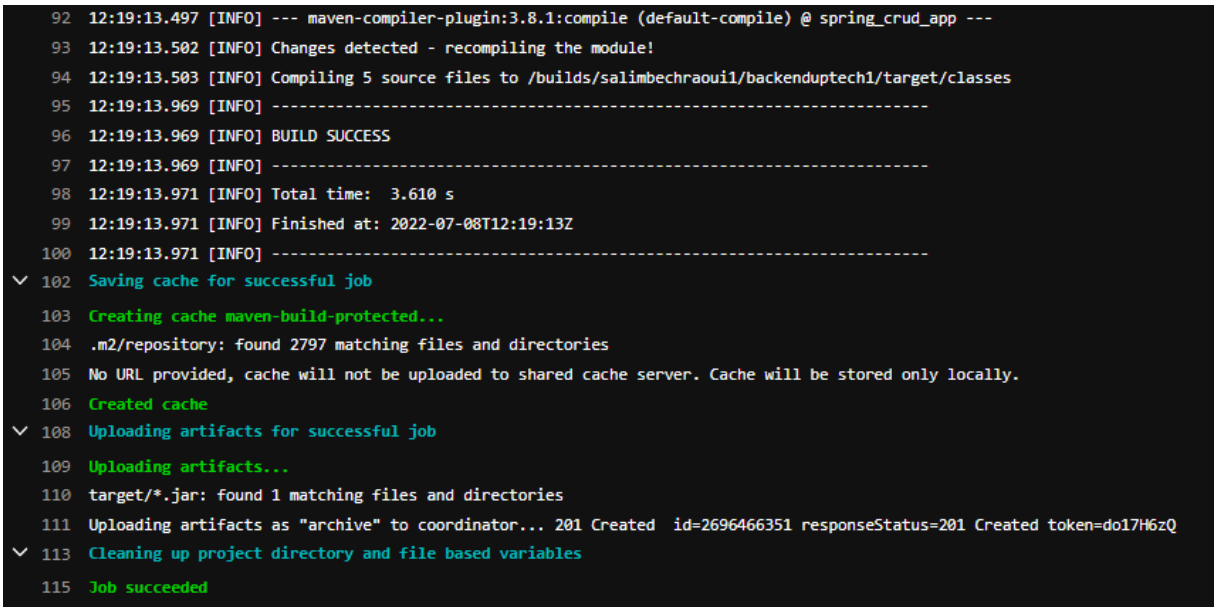


Figure III.1.6.1-6Console of build stage of the backend

In Front end part we used node images which contains by default everything we need for the build like NPM and NODE.

The " before script " part defines the display of the Node version and the NPM version.

```yaml
image: node:16.15.1-alpine
stages:
   - test
   - integration
   - package
   - deploy
cache:
 paths:
  - node_modules/
build:
  stage: test
  tags:
    - backend
    - docker
  before_script:
    - echo " Before script section "
    - node -v
    - npm i node@8.12.0
    - npm -v
  script:
    - cd frontend
    - npm install @angular/cli
    - npm install
    - npm run build
    - ls dist
  artifacts:
    name: release
    paths:
      - frontend/dist
    expire_in: 2 hours
```

Figure III.1.6.1-7 Frontend build stage

In this step we install the Angular client , then we do the build using the “ ng build “ command. Eventually the build result is in the /dist/front folder. We then declare an artifact that contains a version that is automatically saved in a specific folder created by GitLab and that can be used by another stage or pipeline.

The following console in *Figure III.1.6.1-8* shows the compilation instructions of the frontend part:

```
✓ Browser application bundle generation complete.
- Copying assets...
✓ Copying assets complete.
- Generating index html...
✓ Index html generation complete.
Initial Chunk Files               | Names         |      Size
main.304a947ee4f715529357.js      | main          | 249.24 kB
polyfills.e392a4210ba1c2c73b30.js | polyfills     |  36.23 kB
runtime.22fe1984e8ba22cee024.js   | runtime       |   1.06 kB
styles.31d6cfe0d16ae931b73c.css   | styles        |   0 bytes
| Initial Total | 286.53 kB
Build at: 2022-07-08T12:48:50.138Z - Hash: d9fbb17b73d4e6ff7ae9 - Time: 30506ms
$ ls dist
employee-management-system
Saving cache for successful job
Creating cache default-protected...
node_modules/: found 203 matching files and directories
No URL provided, cache will not be uploaded to shared cache server. Cache will be stored only locally.
Created cache
Uploading artifacts for successful job
Uploading artifacts...
frontend/dist: found 9 matching files and directories
Uploading artifacts as "archive" to coordinator... 201 Created  id=2696600202 responseStatus=201 Created token=EaGDuF6y
Cleaning up project directory and file based variables
Job succeeded
```

Figure III.1.6.1-9 frontend build console

These are the SAST Test tools used in the frontend part which are running successfully in the job.

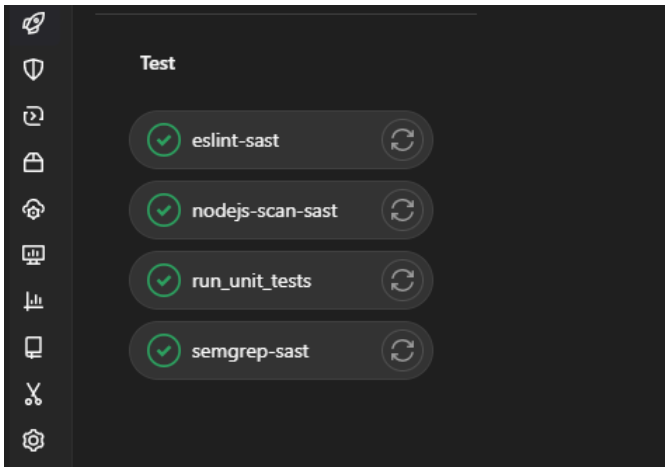


Figure III.1.6.1-10 Gitlab CI build stage frontend status

III.1.6.2 Test stage

In this step we add an additional code quality step, the “quality “ step is defined in the stages block. This means that the pipeline will add a quality task after compiling and testing.Then we added an option “ Allow failure “ , this line allows the quality step to fail. As this is not a critical step , we allow the failure and let the pipeline continue, then we used the word “ services” as keyword that allows lauching a docker daemon so that the execution of our code analysis program can take place

```
code_quality_job:
    stage: quality
    image: docker:stable
    allow_failure: true
    services:
        - docker:stable-dind
    script:
        - mkdir codequality-results
        - echo "Measure of quality started "
    tags:
        - backend
        - docker
    artifacts:
        paths:
            - codequality-results/
```

Figure III.1.6.2-1code quality measure

We can follow each stage and its instructions in a console *Figure III.1.6.2-2* shows the console output in the quality step :

```
Skipping Git submodules setup
Restoring cache                                                                                                    00:00
Checking cache for code_quality_job-protected...
No URL provided, cache will not be downloaded from shared cache server. Instead a local version of cache will be extracted.
Successfully extracted cache
Downloading artifacts                                                                                              00:14
Downloading artifacts for maven-build (2696466351)...
Downloading artifacts from coordinator... ok        id=2696466351 responseStatus=200 OK token=K99GTZiq
Executing "step_script" stage of the job script                                                                    00:00
Using docker image sha256:b0757c55a1fdbb59c378fd34dde3e12bd25f68094dd69546cf5ca00ddbaa7a33 for docker:stable with digest docker@sha256:fd4d028713fd05a1fb896412805daed82c4a0cc84331d8dad00cb596d7ce3e3a ...
$ mkdir codequality-results
$ echo "Measure of quality started "
Measure of quality started
Saving cache for successful job                                                                                    00:01
Creating cache code_quality_job-protected...
WARNING: .m2/repository: no matching files. Ensure that the artifact path is relative to the working directory
Archive is up to date!
Created cache
Uploading artifacts for successful job                                                                             00:02
Uploading artifacts...
codequality-results/: found 1 matching files and directories
Uploading artifacts as "archive" to coordinator... 201 Created  id=2696466353 responseStatus=201 Created token=K99GTZiq
Cleaning up project directory and file based variables                                                             00:00
Job succeeded
```

Figure III.1.6.2-3console of code quality

The next step is to integrate our projects with sonarqube after download sonarqube with docker we enter the interface we click on "add project" then "manually" in the window that opens. We must enter a key and a name for the project. Before finalizing, we have to address a name for the token so click on "Generate" then, we have to tap on next. At a tab that appears, we choose the type of project (in our case it's .NET) and we copy the script to use it in the pipeline.

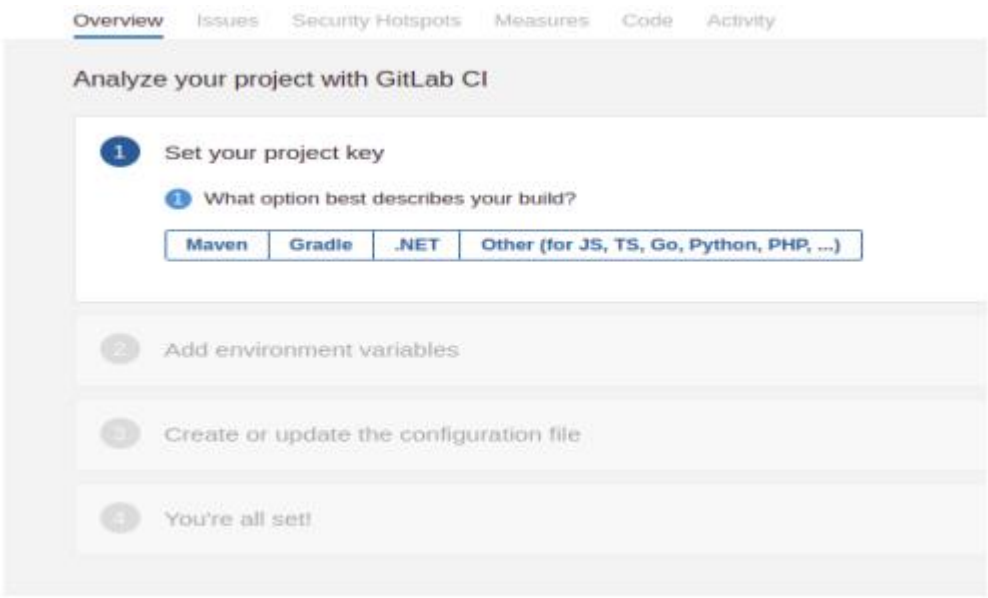


Figure III.1.6.2-4 Sonar startup project

The following figure shows the code of the integration sonarqube. This step of the pipeline allows us to check the quality of the code our application

```
sonarqube-check:
  stage: integration
  tags:
   - remote
   - sonarqube
  image:
   name: sonarsource/sonar-scanner-cli:latest
   entrypoint: [""]

  variables:
   SONAR_USER_HOME: "${CI_PROJECT_DIR}/.sonar"  # Defines the location of the analysis task cache
   GIT_DEPTH: "0"  # Tells git to fetch all the branches of the project, required by the analysis task
  cache:
   key: "${CI_JOB_NAME}"
   paths:
     - .sonar/cache
  script:
   - sonar-scanner
  allow_failure: true
```

Figure III.1.6.2-5integration stage

For the frontend part , we add some properties to make our integration correct

```
sonar.projectKey=frontenduptech1
sonar.qualitygate.wait=true
```

Figure III.1.6.2-6 Sonar-project properites frontend

But in the part of Backend part we add properites in pom.xml

```
        <name>spring_crud_app</name>
        <description>Employee Management with spring boot &amp; REST APIs</description>
        <properties>
                <java.version>11</java.version>
                  <sonar.projectKey>salimbechraoui1_backenduptech1_AYHdB9tpWzlrtYoUxR1i</sonar.projectKey>
  <sonar.qualitygate.wait>true</sonar.qualitygate.wait>
</properties>
```

Figure III.1.6.2-7 sonar-project.propeties backend

After launching the sonar scan job in the pipeline , we can find the result of analysis interface of sonarqube in Figure III.1.6.2-7 that we have 2 bugs that can lead to unexpected behavior at runtime.

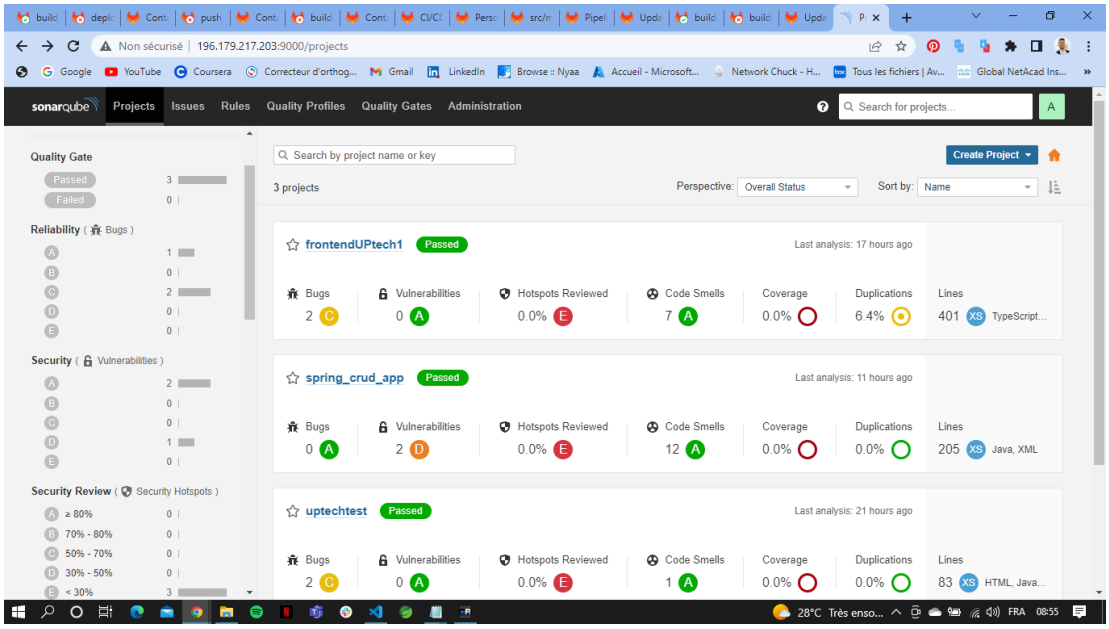


Figure III.1.6.2-8Sonarqube

We can follow each stage and its instructions in a console *Figure III.1.6.2-9* shows the console output in the test step

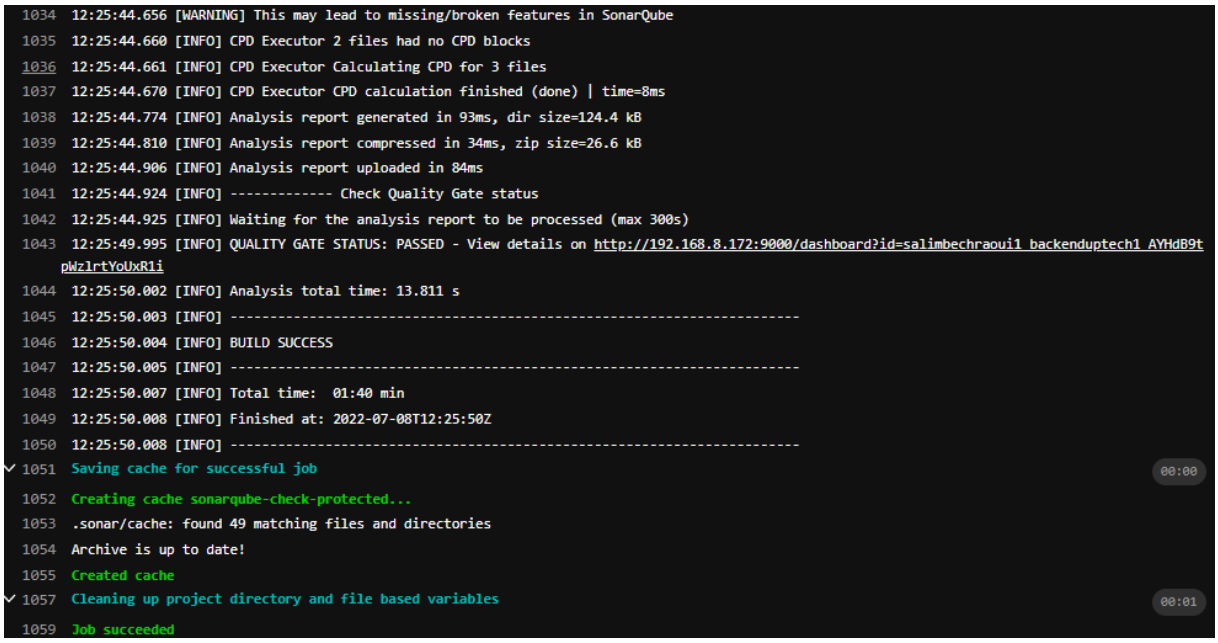


Figure III.1.6.2-10 console Test stage

The interesting thing about artifacts is that we can download them directly from the Gitlab user interface when our pipeline is complete

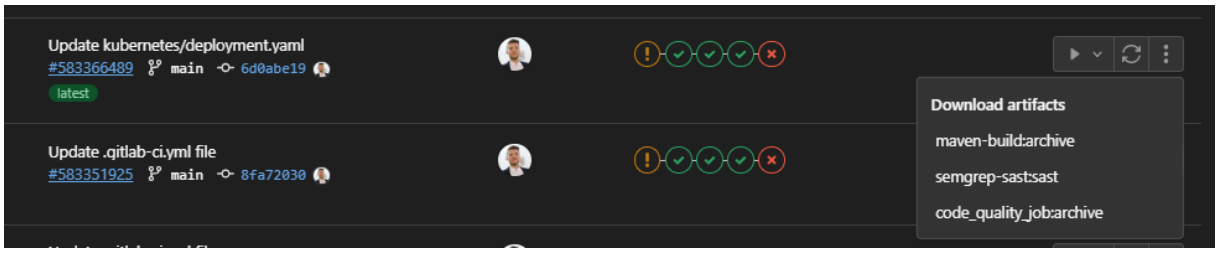


Figure III.1.6.2-11 artifacts backend

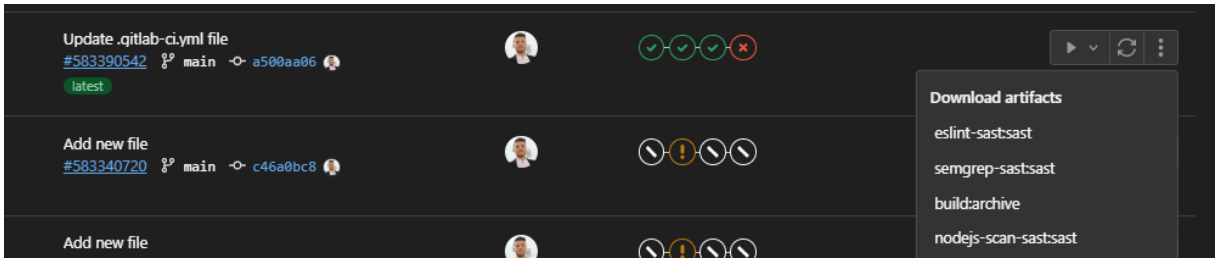


Figure III.1.6.2-12artifacts frontend

III.1.6.3 Package stage

The dockeriztation is done with a simple standard docker text file that includes the native docker commands used to describe our image in this section we will explain how to create our frontend image

- Installation of the necessary project packages
- The compilation that will build the project
- Copy the contents to Nginx hosting folder

Figure 3-42 shows the installation instructions for the necessary packages and commands that concern deployment with Nginx to build the image associated

```
FROM node:latest as builder

RUN mkdir -p /app

WORKDIR /app

COPY . .

RUN npm install
RUN npm run build --prod
EXPOSE 8080
ENTRYPOINT  ["npm", "start"]

FROM nginx:1.17.1-alpine
COPY src/nginx/default.conf /etc/nginx/conf/default.conf

COPY /dist/employee-management-system  /usr/share/nginx/html
```

Figure III.1.6.3-1: frontend Dockerfile

We will use the gitlab container registry which allows us to store our images within Gitlab to package the docker image we add a new this stage in the pipeline CI/CD therefore the new step in our pipeline is described as follows in *Figure III.1.6.3-2*

```
build_image:
  stage: package
  tags:
   - front
   - shell
  before_script:
    - echo "Linux user is $USER"
    - echo "Docker registry user is $CI_REGISTRY_USER"
    - echo "Docker registry name is $CI_REGISTRY"
    - echo "Docker registry image is $CI_REGISTRY_IMAGE"
    - pwd
    - cd frontend

  script:
    - docker build -t registry.gitlab.com/salimbechraoui1/frontenduptech1/microservice/employee:1.5 .
```

Figure III.1.6.3-3: frontend package build step

To avoid hard-coding values, we used environmment variables this table show these environmental variables and their descriptions:

Table III.1.6.3-I:Environnement variables

Variables	Description
$CI_REGISTRY_USER	User name to be used to push containers to the " Gitlab Container Registry“ for our current project.
$CI_REGISTRY_PASSWORD	The password to use for pushing

	containers to the “ Gitlab Container Registry “
$CI_REGISTRY	Is the address of the GitLab container registry. This variable includes a port value if it is specified in the registry configuration.
$CI_REGISTRY_IMAGE	Represents the URL of the container registry related to the specific project. this URL depends on the Gitlab instance

After we build our image the instruction should in the console look like in the *Figure III.1.6.3-4*

```
78  #8 sha256:e8c613e07b0b7ff33893b694f7759a10d42e180f2b4dc349fb57dc6b71dcab00
79  #8 exporting layers done
80  #8 writing image sha256:7ed549910b5579bb497608120cc436418b76652a165ac9df76cc8eb077326043 0.2s done
81  #8 naming to registry.gitlab.com/salimbechraoui1/frontenduptech1/microservice/employee:1.5 done
82  #8 DONE 0.2s
83  Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
84  Path
85  ----
86  C:\gitlab-runner\builds\Emuy3cEF\0\salimbechraoui1\frontenduptech1
87  $ cd frontend
88  $ docker build -t registry.gitlab.com/salimbechraoui1/frontenduptech1/microservice/employee:1.5 .
90  Saving cache for successful job
91  Version:      15.1.0
92  Git revision: 76984217
93  Git branch:   15-1-stable
94  GO version:   go1.17.7
95  Built:        2022-06-20T10:10:56+0000
96  OS/Arch:      windows/amd64
97  Creating cache default-protected...
98  Runtime platform                                    arch=amd64 os=windows pid=22464 revision=76984217 version=15.1.0
99  WARNING: node_modules/: no matching files. Ensure that the artifact path is relative to the working directory
100 Archive is up to date!
101 Created cache
103 Cleaning up project directory and file based variables
105 Job succeeded
```

Figure III.1.6.3-5 frontend build image console

Then after we build our image and login to the docker repository we gonna push the image in the gitlab container registry we will add satge called in our pipeline therefore the new step in our pipeline is described as follows *Figure III.1.6.3-6*

```
74  push_image:
75    stage: package
76    needs:
77      - build_image
78    tags:
79      - front
80      - shell
81    before_script:
82      - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
83    script:
84      - docker push registry.gitlab.com/salimbechraoui1/frontenduptech1/microservice/employee:1.5
```

Figure III.1.6.3-7 frontend package push step

To verife that our images are pushed we can check the console this *Figure III.1.6.3-8* show the detail of push stage

```
33  $ docker push registry.gitlab.com/salimbechraoui1/frontenduptech1/microservice/employee:1.5
34  The push refers to repository [registry.gitlab.com/salimbechraoui1/frontenduptech1/microservice/employee]
35  4d1799bb6778: Preparing
36  d32da9556234: Preparing
37  fbe0fc9bcf95: Preparing
38  f1b5933fe4b5: Preparing
39  4d1799bb6778: Layer already exists
40  f1b5933fe4b5: Layer already exists
41  fbe0fc9bcf95: Layer already exists
42  d32da9556234: Layer already exists
43  1.5: digest: sha256:7acdde2d6a184e22278c13e914a3f5b54203721fe536fc2e157f71aa09e7859d size: 1155
45  Saving cache for successful job
46  Version:      15.1.0
47  Git revision: 76984217
48  Git branch:   15-1-stable
49  GO version:   go1.17.7
50  Built:        2022-06-20T10:10:56+0000
51  OS/Arch:      windows/amd64
52  Creating cache default-protected...
53  Runtime platform                                    arch=amd64 os=windows pid=17636 revision=76984217 version=15.1.0
54  WARNING: node_modules/: no matching files. Ensure that the artifact path is relative to the working directory
55  Archive is up to date!
56  Created cache
58  Cleaning up project directory and file based variables
60  Job succeeded
```

Figure III.1.6.3-9 frontend push image console

GitLab provides a private container registry. We can explore the container registry by accessing Packages & Registries → Container Registry in our GitLab project. *Figure III.1.6.3-10* shows that our back-end docker images are successfully created and stored in the GitLab docker container registry.

Figure III.1.6.3-11Placement of images in the gitlab repository

We create this Dockerfile in three lines contains only the build jar to run it , it will compile the application and create a docker image that contains the backend applciation. *Figure III.1.6.3-12* shows our configurations

```
FROM openjdk:11
COPY ./target/spring_crud_app-0.0.1-SNAPSHOT.jar spring_crud_app-0.0.1-SNAPSHOT.jar
CMD ["java","-jar","spring_crud_app-0.0.1-SNAPSHOT.jar"]
```

Figure III.1.6.3-13Backend Dockerfile

In this case we make build and push stage combined in our " gitlab-ci.yml " that allows us to package our backend application, *Figure III.1.6.3-14* shows the instructions that we explained earlier:

```
build_image:
  stage: package
  services:
  - docker:dind

  tags:
   - backend
   - docker

  before_script:
    - echo "Linux user is $USER"
    - echo "Docker registry user is $CI_REGISTRY_USER"
    - echo "Docker registry name is $CI_REGISTRY"
    - echo "Docker registry image is $CI_REGISTRY_IMAGE"

  script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
    - docker build -t registry.gitlab.com/salimbechraoui1/backenduptech1:1.3 .
    - docker push registry.gitlab.com/salimbechraoui1/backenduptech1:1.3
```

Figure III.1.6.3-15Packaging image script backend

After we build our image the instruction should in the console look like in the *Figure III.1.6.3-16*

```
99  7e145627a1fb: Preparing
100 6812ae0d9f23: Preparing
101 369123a7ed65: Preparing
102 5afd661c6106: Preparing
103 66183893ba24: Preparing
104 6840c8ff46bd: Preparing
105 97d5fec864d8: Preparing
106 66183893ba24: Waiting
107 97d5fec864d8: Waiting
108 7e145627a1fb: Layer already exists
109 5afd661c6106: Layer already exists
110 6812ae0d9f23: Layer already exists
111 369123a7ed65: Layer already exists
112 66183893ba24: Layer already exists
113 97d5fec864d8: Layer already exists
114 6840c8ff46bd: Layer already exists
115 e678a19efec5: Pushed
116 1.3: digest: sha256:78444b536e276ca7c10ebffa23ff43b3eb7baf75d5d6733a6d51baea90aba6ef size: 2007
118 Saving cache for successful job
119 Creating cache build_image-protected...
120 WARNING: .m2/repository: no matching files. Ensure that the artifact path is relative to the working directory
121 Archive is up to date!
122 Created cache
124 Cleaning up project directory and file based variables
126 Job succeeded
```

Figure III.1.6.3-17 Backend package stage

After we have packaged our Backend application, we can see in a *Figure III.1.6.3-18* that our image is successfully stored in the gitlab container registry

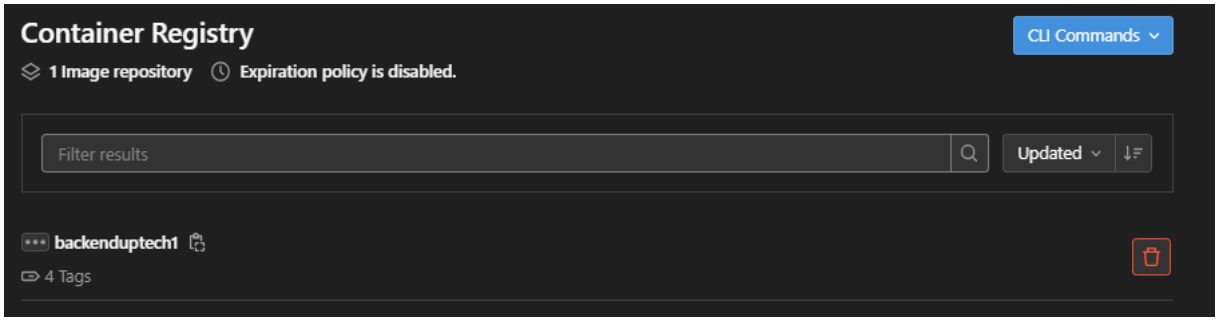


Figure III.1.6.3-19Placement of image in the GitLab repository

III.1.6.4 Deploy stage

In this section we are going to focus on how to deliver our code continuously into the server quickly and reliably, Continuous delivery is a discipline where the application is constructed in a way that it can be put into production at any time, the frontend and backend part will have same execution in the deploy stage

III.1.6.4.1 Setting deployment containers

We used docker-compose to be able to start and stop multiple containers simultaneously while defining the links between them. Therefore, to start the application at once with docker-compose, we need to follow these steps:

- Creating a " Docker-compose.yml " file to define the services that compose the application and add some of the properties defined in their respective docker files.
- For each service we need to give a link to the folder where its dockerfile is located , or the URL of the git repository where it is stored.
- Launching the application with the "docker-compose up" command

Figure III.1.6.4.1-1 shows our " docker-compose.yml " file script which contains the 3 services (frontend, backend,and database) to link and communicate the containers together:

```yaml
version: '3'

services:
  employee-jdbc-container1:
    image: backend
    container_name: employee-jdbc-container1
    ports:
      - "8888:8888"
    networks:
      - backend
      - frontend
    build:
      context: ./
      dockerfile: Dockerfile
    depends_on:
      - mysqldb
     #environment:
      #SPRING_DATASOURCE_URL: jdbc:mysql://mysql/springboot?allowPublicKeyRetrieval=true&useSSL=false
      #SPRING_DATASOURCE_USERNAME: root
      #SPRING_DATASOURCE_PASSWORD: salimbackend
  mysqldb:
    image: mysql:8
    networks:
      - backend
    restart: always
    container_name: mysqldb
    environment:
      - MYSQL_ROOT_PASSWORD=salimbackend
      - MYSQL_DATABASE=springboot
   #   - MYSQL_USER_PASSWORD=salimbackend
    #  - MYSQL_PASSWORD=salimbackend

  angular-service: # name of the first service
    image: frontend
    container_name: angular-container2
    build: ./ # specify the directory of the Dockerfile
    volumes: # Volume binding
      - "./:/usr/src"
    ports:
      - "4200:80" # specify port forwarding
    networks:
      -  frontend
    depends_on:
      - employee-jdbc-container1


networks:
  frontend:
  backend:
```

Figure III.1.6.4.1-2 Docker-compose script

III.1.6.4.2 Delivery of the application

In this step , we will deploy our compiled images to a docker environment, using the continuous delivery pipeline .

III.1.6.4.2.1Continuous Deployment

We added another step in the gitlab-ci.yml file, in order to start our deploying images to the stage test environment, The following *Figure III.1.6.4.2.1-1* shows our configuration of the " deploy " step

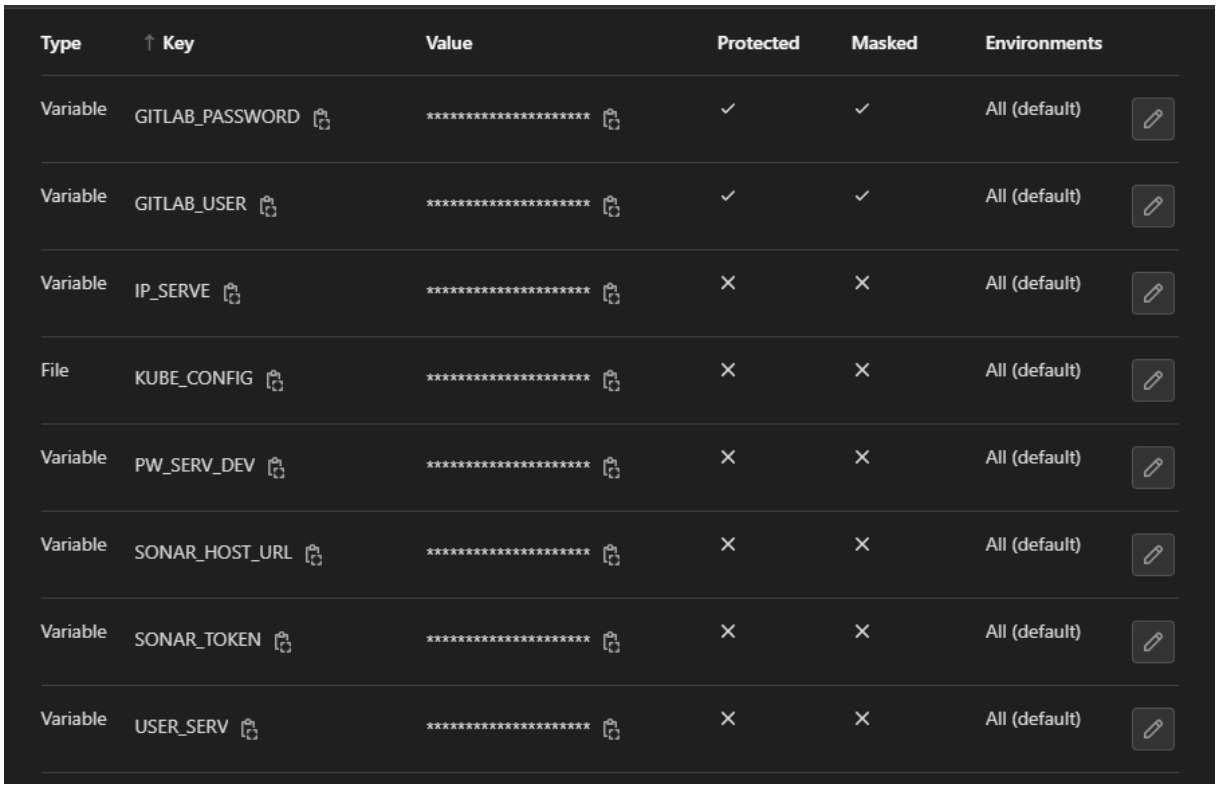
```
deploy_to_dev:
   stage: deploy
   tags:
    - backend
    - docker
   before_script:
     - apk update
     - apk add --no-cache openssh sshpass
     - echo "ssh pass installer"
     - export SSHPASS="$PW_SERV_DEV"
     - sshpass
     - mkdir ~/.ssh
     - sshpass -v
     - echo "StrictHostKeyChecking no " >> ~/.ssh/config
   script:
    - sshpass -p "$PW_SERV_DEV" ssh -o StrictHostKeyChecking=no $USER_SERV@$IP_SERVE "
        docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY &&
        docker run -d -p 3000:3000 registry.gitlab.com/salimbechraoui1/frontenduptech1/microservice/employee:1.5
        docker-compose -f  docker-compose.yaml down &&
        docker-compose -f  docker-compose.yaml up "
```

Figure III.1.6.4.2.1-2Stage test environment gitlab-ci.yml Script

SSHPASS is a command line tool that allows us to provide a password (non-interactive password authentication) at the command prompt proper, such that the automated shell scripts can be run to take backups. It defines how to secure the password by the declaration in an environment variable called SSHPASS .

For making SSHPASS work we need to add and configure environment variables. So , we went to Settings → CI/CD → Variables and then created a new variable named PW_SERV_DEV whose value contains the server password and USER_SERV contains the username.

We also added another variable named IP _SERVE having as value the IP address of the server on which we deployed the application. *Figure III.1.6.4.2.1-3* below shows the configuration of these variables .

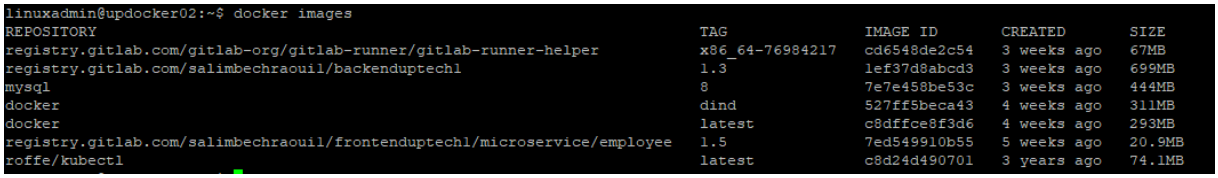


Type	↑ Key	Value	Protected	Masked	Environments
Variable	GITLAB_PASSWORD	**********	✓	✓	All (default)
Variable	GITLAB_USER	**********	✓	✓	All (default)
Variable	IP_SERVE	**********	✗	✗	All (default)
File	KUBE_CONFIG	**********	✗	✗	All (default)
Variable	PW_SERV_DEV	**********	✗	✗	All (default)
Variable	SONAR_HOST_URL	**********	✗	✗	All (default)
Variable	SONAR_TOKEN	**********	✗	✗	All (default)
Variable	USER_SERV	**********	✗	✗	All (default)

Figure III.1.6.4.2.1-4Environnement variables configuration

To deploy our architecture, a single command run by docker-compose should set up our continuous integration platform The “docker-compose up” command creates a new network for the cluster of containers configured in the “ gitlab.yml “ file and then it launches the set of containers .

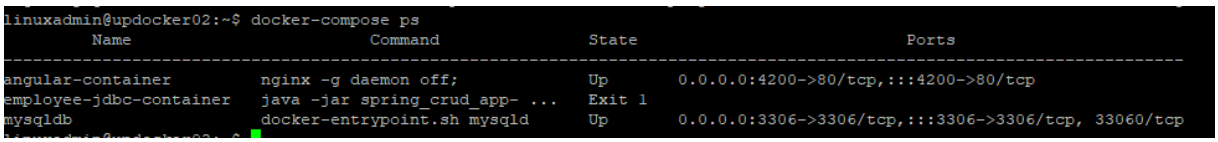
After running docker can read the configured images and download them from the Gitlab container registry once it downloads we can list docker images as shown in *Figure III.1.6.4.2.1-5*



```
linuxadmin@updocker02:~$ docker images
REPOSITORY                                                          TAG                 IMAGE ID       CREATED         SIZE
registry.gitlab.com/gitlab-org/gitlab-runner/gitlab-runner-helper   x86_64-76984217     cd6548de2c54   3 weeks ago     67MB
registry.gitlab.com/salimbechraouil/backenduptech1                  1.3                 1ef37d8abcd3   3 weeks ago     699MB
mysql                                                               8                   7e7e458be53c   3 weeks ago     444MB
docker                                                              dind                527ff5beca43   4 weeks ago     311MB
docker                                                              latest              c8dffce8f3d6   4 weeks ago     293MB
registry.gitlab.com/salimbechraouil/frontenduptech1/microservice/employee   1.5        7ed549910b55   5 weeks ago     20.9MB
roffe/kubectl                                                       latest              c8d24d490701   3 years ago     74.1MB
```

Figure III.1.6.4.2.1-6 Docker images

In the following figure we notice that our services are running successfully *Figure III.1.6.4.2.1-7*



```
linuxadmin@updocker02:~$ docker-compose ps
         Name                       Command               State                          Ports
-------------------------------------------------------------------------------------------------------------------
angular-container          nginx -g daemon off;             Up       0.0.0.0:4200->80/tcp,:::4200->80/tcp
employee-jdbc-container    java -jar spring_crud_app- ...   Exit 1
mysqldb                    docker-entrypoint.sh mysqld      Up       0.0.0.0:3306->3306/tcp,:::3306->3306/tcp, 33060/tcp
```

Figure III.1.6.4.2.1-8 Docker-compose containers

Then we configured our services and launching our docker-compose and deployed our images the enter link " http://192.179.217.202 " shows *Figure III.1.6.4.2.1-9* the application interface

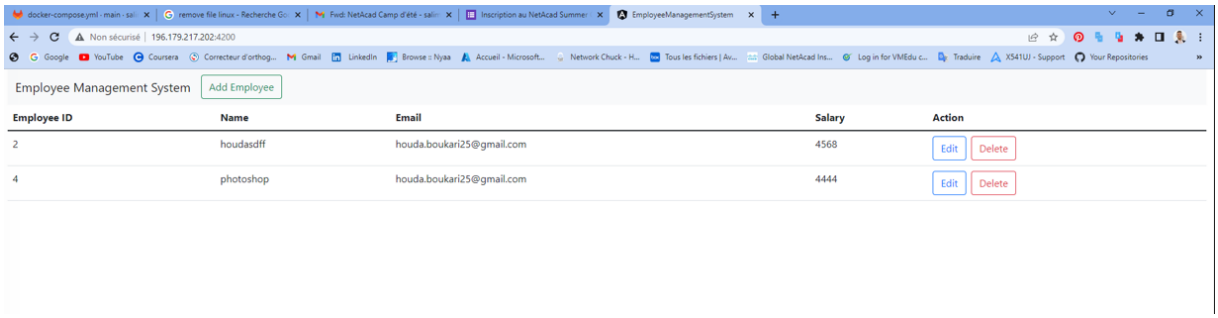


Figure III.1.6.4.2.1-10Application Deployed

III.1.6.4.2.2 Continuous Delivery

Once the stage environment test has been deployed and tested all that remains is to deploy the application to the Kubernetes cluster. This deployment task is executed from the deployment.yml file *Figure III.1.6.4.2.2-1* and the service in *Figure III.1.6.4.2.2-2*

deployment.yaml 470 bytes

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backenduptech1
  namespace: my-micro-service
spec:
  selector:
    matchLabels:
      app: backenduptech1
  replicas: 2
  template:
    metadata:
      labels:
        app: backenduptech1
    spec:
      imagePullSecrets:
      - name: my-registry-keyy
      containers:
      - name: backenduptech1
        image: registry.gitlab.com/salimbechraoui1/backenduptech1:1.3
        ports:
        - containerPort: 8080
```

Figure III.1.6.4.2.2-3Configuration deployment.yml

service.yaml 203 bytes

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backenduptech1
  namespace: my-micro-service
spec:
  selector:
    app: backenduptech1
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080

```

Figure III.1.6.4.2.2-4Service.yml

After we put the YAML files we will execute our pipeline CICD by adding deploy stage manual and will create to us service and deployment and will deploy our container to kubernetes clusters therefore the new step in our pipeline is described as follows *Figure III.1.6.4.2.2-5*

```yaml
deploy_prod:
    stage: deploy
    image: roffe/kubectl
    tags:
    - deploy
    - kube
    - remote

    before_script:
      - export IMAGE_NAME=$CI_REGISTRY_IMAGE
      - export IMAGE_TAG=1.3
      - export MICRO_SERVICE=backenduptech1
      - export SERVICE_PORT=8080
      - export REPLICAS=2
      - export KUBECONFIG=$KUBE_CONFIG

    when: manual
    script:
        - kubectl create secret docker-registry my-registry-keyy --docker-server=$CI_REGISTRY --docker-username=$GITLAB_USER --docker-password=$GITLAB_PASSWORD -n my-micro-service --dry-run=1  -o yaml | kubectl apply -f -
        - kubectl apply -f kubernetes/deployment.yaml
        - kubectl apply -f kubernetes/service.yaml
```

Figure III.1.6.4.2.2-6Deploy Kubernetes cluster

we have added the keyword "when: manual" in order to deploy in production only with human intervention. The execution validation is necessary to know if there are any errors during the production deployment.

On our continuous delivery pipeline, manual deployment is symbolized by the icon ▸.

The following figure shows that the pre-production deployment is manual

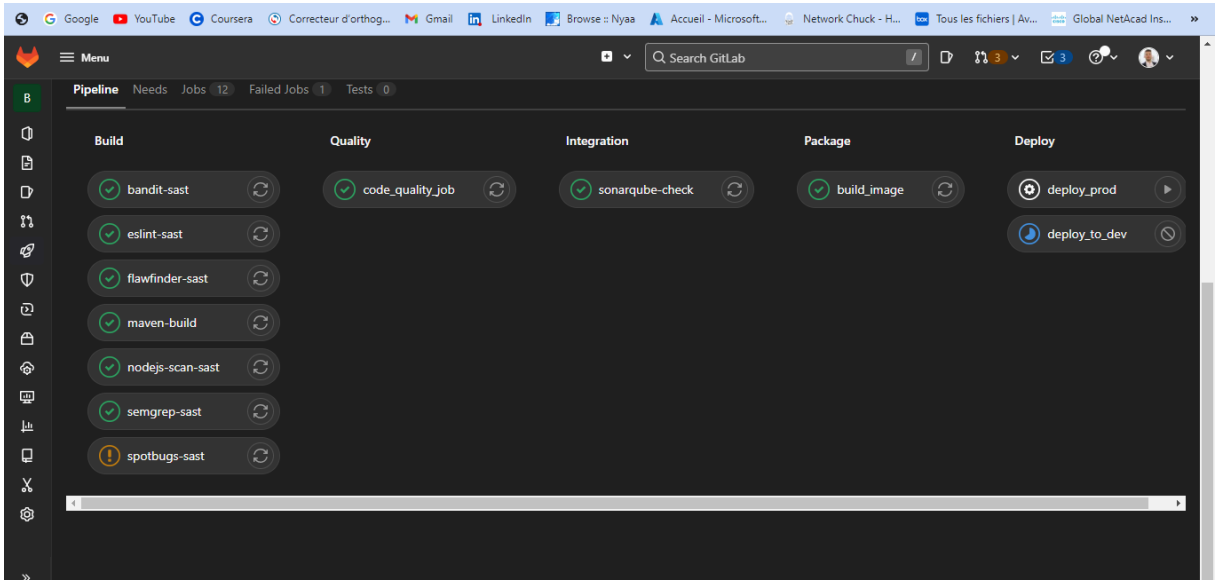


Figure III.1.6.4.2.2-7 Manual Deployment

This deployment task is executed from the deploy.yml *Figure III.1.6.4.2.2-8*

```
49 Using docker image sha256:c8d24d490701efec4c8d544978b3e4ecc4855475a221b002a8f9e5e473398805 for roffe/kubectl with digest r
   offe/kubectl@sha256:ba13f8ffc55c83a7ca98a6e1337689fad8a5df418cb160fa1a741c80f42979bf ...
50 $ export IMAGE_NAME=$CI_REGISTRY_IMAGE
51 $ export IMAGE_TAG=1.3
52 $ export MICRO_SERVICE=backenduptech1
53 $ export SERVICE_PORT=8080
54 $ export REPLICAS=2
55 $ export KUBECONFIG=$KUBE_CONFIG
56 $ kubectl create secret docker-registry my-registry-keyy --docker-server=$CI_REGISTRY --docker-username=$GITLAB_USER --doc
   ker-password=$GITLAB_PASSWORD -n my-micro-service --dry-run=1  -o yaml | kubectl apply -f -
57 secret/my-registry-keyy configured
58 $ kubectl apply -f kubernetes/deployment.yaml
59 deployment.apps/backenduptech1 created
60 $ kubectl apply -f kubernetes/service.yaml
61 service/backenduptech1 created
63 Saving cache for successful job                                                                    00:00
64 Creating cache deploy_prod-protected...
65 WARNING: .m2/repository: no matching files. Ensure that the artifact path is relative to the working directory
66 No URL provided, cache will not be uploaded to shared cache server. Cache will be stored only locally.
67 Created cache
69 Cleaning up project directory and file based variables                                            00:01
71 Job succeeded
```

Figure III.1.6.4.2.2-9 log console display

Finally this Figure III.1.6.4.2.2-10 shows that the container of the application is ready and running in the server .

```
root@worker01:/home/linuxadmin# kubectl get  pod  -n my-micro-service
NAME                               READY   STATUS             RESTARTS   AGE
backenduptech1-6b99cbc59b-mc98x    0/1     ImagePullBackOff   0          48m
backenduptech1-6b99cbc59b-wxc62    0/1     ImagePullBackOff   0          48m
frontend2-789fbdfd6-n76gt          0/1     ImagePullBackOff   0          2d22h
frontend3-7f9d658d64-mw66k         0/1     ImagePullBackOff   0          2d22h
frontend3-7f9d658d64-vg4kt         0/1     ImagePullBackOff   0          2d22h
frontenduptech1-8dcff4f5c-q8jvf    0/1     ErrImagePull       0          112s
frontenduptech1-8dcff4f5c-t8z56    0/1     ErrImagePull       0          112s
uptechtest-684b5bcb56-f4tm8        0/1     ImagePullBackOff   0          47h
uptechtest-75d575555f-7dzv4        0/1     ImagePullBackOff   0          26h
uptechtest-7c5dbfdbfd-h9dtg        0/1     ImagePullBackOff   0          2d22h
root@worker01:/home/linuxadmin#
```

Figure III.1.6.4.2.2-11 Containers deployed

III.2 Conlusion

During this last chapter, we have implemented our solution which consists of setting up a CI/CD platform, which was carried out in two stages: a stage technique for configuring the environment, and a functional step for configuring components.

General Conclusion

These days, everything is changing faster and faster. Existing applications must constantly be adapted, at an ever-increasing pace. A wall of confusion often appears between developers and operations teams. The DevOps approach overcomes these problems.

As part of this graduation project, and based on the DevOps concept, we had to implement a continuous delivery pipeline (chain) using Docker and Kubernetes technologies.

After a comparison of DevOps tools, we chose the tools best suited to our needs such as Gitlab-ci , Sonarqube. The conceptual analysis of our architecture allowed us to identify the functional and non-functional needs, as well as the different cases overall usage of the platform. This analysis allowed us to have a functional and physical architecture for the two parts, CI (Continuous Integration) and CD (Continuous Deployment).

Finally, we finished by installing the various components of our continuous delivery architecture, in particular Gitlab-ci , which is the heart of the CI/CD chain. The latter has allowed us to automate the entire production chain which will allow the acceleration of the deployment of applications and the obtaining of deliverables on time.

As a perspective to this work, we prove to integrate the automation of application monitoring processes in order to supervise the state of the company's infrastructure.

On the other hand, we can implement our entire production chain on the Kubernetes infrastructure to ensure the scalability of our servers and guarantee high availability.

Bibliography

1. **[En ligne] https://www.uptech.com.tn/.**

2. [En ligne] https://www.redhat.com/en/topics/devops.

3. [En ligne] https://www.atlassian.com/agile/devops.

4. [En ligne] https://en.wikipedia.org/wiki/Apache_Maven.

5. [En ligne] https://en.wikipedia.org/wiki/Gradle.

6. [En ligne]

7. [En ligne] https://marketplace.visualstudio.com/items?itemName=SonarSource.sonarlint-vscode.

8. [En ligne] http://www.cloudbees.com/jenkins/what-is-jenkins.

9. [En ligne] https://docs.gitlab.com/ee/ci/.

10. [En ligne] https://jfrog.com/knowledge-base/docker-hub-and-docker-registries-a-beginners-guide/.

11. [En ligne] https://about.gitlab.com/blog/2016/05/23/gitlab-container-registry/#tight-integration.

12. [En ligne] https://docs.docker.com/engine/swarm/.

13. [En ligne] https://www.dynatrace.com/news/blog/kubernetes-vs-docker/.

14. [En ligne] https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59#:~:text=A%20monolithic%20application%20is%20simply,deployed%2C%20scaled%2C%20and%20monitored..

15. [En ligne] https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59#:~:text=A%20monolithic%20application%20is%20simply,deployed%2C%20scaled%2C%20and%20monitored..

16. [En ligne] https://github.com/joelparkerhenderson/monorepo-vs-polyrepo.

17. [En ligne] https://en.wikipedia.org/wiki/Static_application_security_testing.

18. [En ligne] https://loft.sh/blog/docker-compose-to-kubernetes-step-by-step-migration/.

19. [En ligne] https://firefox-source-docs.mozilla.org/devtools-user/web_console/.

20. [En ligne] https://www.digitalocean.com/community/tutorials/how-to-set-up-let-s-encrypt-with-nginx-server-blocks-on-ubuntu-16-04.

Annexe

- Input sequence diagram

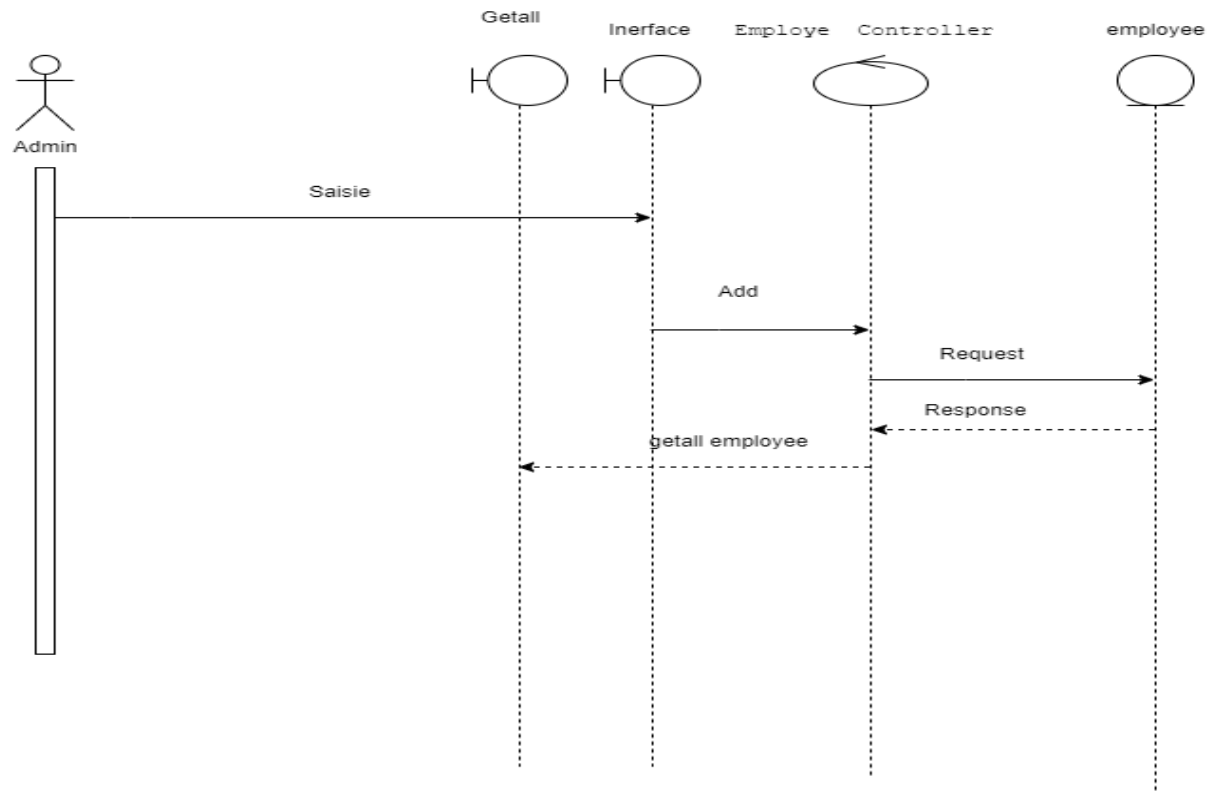


Figure 0-1-Imput sequence diagram

- Edit sequence diagram

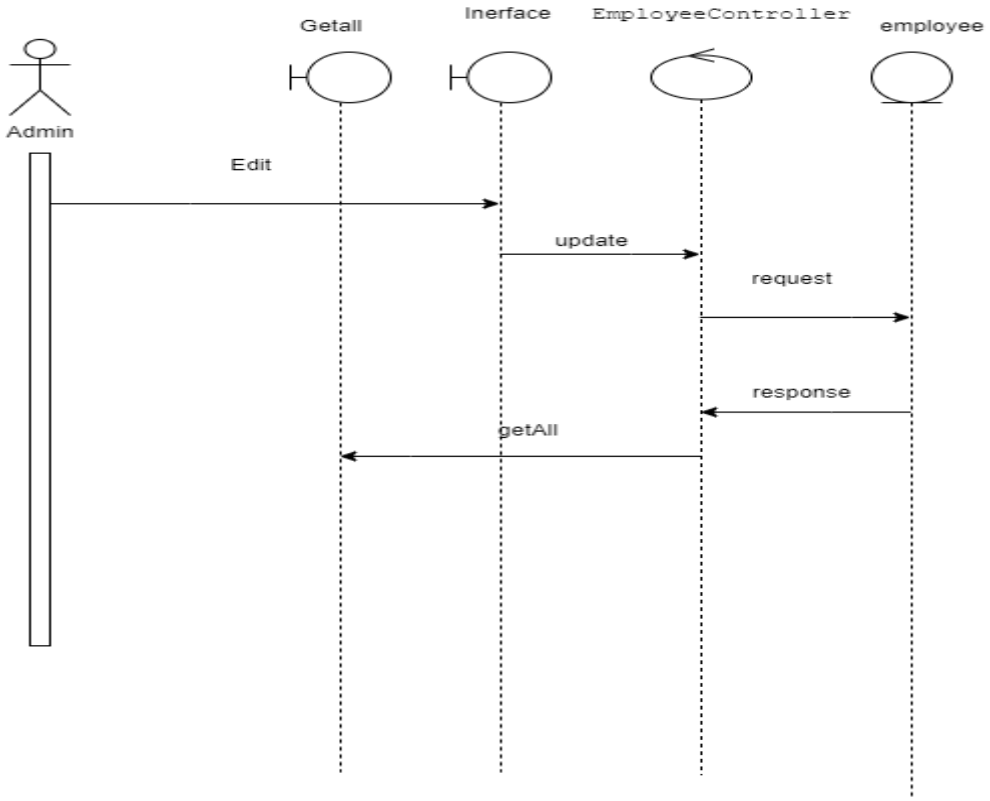


Figure 0-2-Edit sequence diagram

- Delete sequence diagram

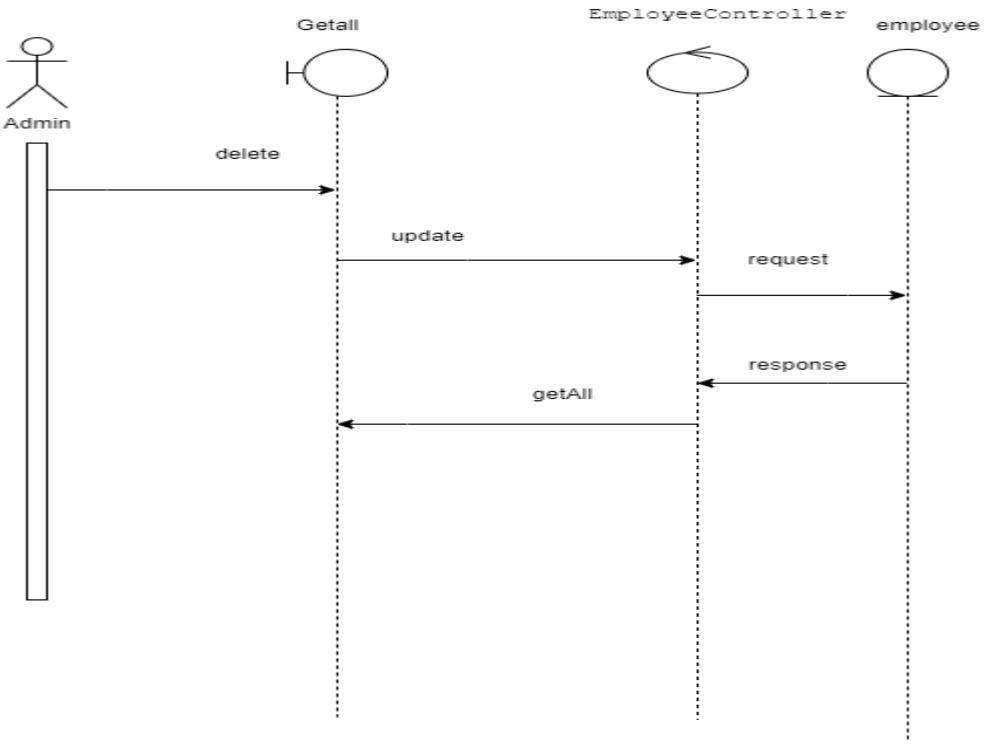


Figure 0-3-Delete sequence diagram

- Install Ansible

```
$ sudo apt update
$ sudo apt updgrade-y
$ sudo reboot
$ sudo apt install -y software-properties-common
$ sudo add-apt-repository –yes –update ppa:ansible/ansible
$ sudo apt update
$ sudo apt install -y ansible
$ ansible –version
```

- Create ansible cfg and inventory file

```
$  mkdir  demo
$  cp/etc/ansible.cfg ~ /demo/
$ vi ~/demo/ansible.cfg
```

- Under the default sections

```
inventory =/home/sysops/demo/inventory
remote_user = sysops
host_key_checking = False
```

- Under the privilege_escalation section

```
Become=true
Become_method=sudo
Become_user=root
Become_ask_pass=false
```

- Save and Close the file. Now let's create the inventory file that we have defined in ~/demo/ansible.cfg file .

```
$ vi ~/demo/inventory

[master ]

192.168.8.174
[worker ]

192.168.8.173
```

- Now finally instruct ansible to use demo project's ansible file by declaring ANSIBLE_CONFIG variable, run the following commands

```
$  export ANSIBLE_CONFIG=/home/sysops/demo/ansible.cfg
$ echo " export ANSIBLE_CONFIG=/home/sysops/demo/ansible.cfg " >> ~/.profile
```

- Run ansible-version command from demo folder to verify ansible configuration

```
$ cd demo/
$ ansible –version
```

RESUME

Ce travail s'inscrit dans le cadre du projet de fin d'études à l'école nationale d'ingénieurs de Gabès "ENIG" pour l'acquisition du Diplôme National d'Ingénieur en Communications et Réseaux. Dans ce contexte, le projet "Pipeline CI/CD" a été réalisé au profit de UPtech et vise à construire un pipeline standard et sécurisé pour réaliser des applications prêtes à la production de manière rapide et efficace. Dans un premier temps, nous avons étudié les technologies qui correspondent à nos besoins. Dans un deuxième temps, nous avons implémenté les outils choisis dans des environnements de développement et de production et nous avons développé l'architecture ainsi que le modèle conceptuel.

Mots-clés : pipeline, CI, CD, Automation, K8s, Sonarqube Docker, DevOps ...

ABSTRACT

This work is part of the graduation project at the national school of engineering of Gabes “ENIG” for the acquisition of the National Diploma of Engineering in Communications and Networks. In this context, "Pipeline CI/CD" project was carried out for the benefit of “UPtech“ and aims to build a standard and secure pipeline to achieve fast and efficient production-ready applications. As a first step, we studied the technologies that match our needs. As a second step, we implemented the chosen tools in staging and production environments and we developed the architecture along with the conceptual pattern.

Keywords: pipeline, CI, CD, Automation, K8s, Sonarqube, Docker, DevOps

ملخص

هذا العمل هو جزء من مشروع التخرج في المدرسة الوطنية للهندسة بقابس “ENIG” للحصول على الدبلوم الوطني للهندسةفي الاتصالات والشبكات.في هذا السياق ، تم تنفيذ مشروع "Pipeline CI / CD" لصالح "UPtech" ويهدف إلى بناء

طريقة معيارية وآمنة لتحقيق تطبيقات جاهزة للإنتاج بسرعة وكفاءة. كخطوة أولى ، درسنا التقنيات التي تتوافق مع احتياجاتنا. كخطوة ثانية ، قمنا بتنفيذ الأدوات المختارة في بيئات ما قبل الإنتاج والإنتاج وقمنا بتطوير البنية جنبًا إلى جنب مع النمط

المفاهيمي.

الكلمات الدالة : pipeline, CI, CD, Automation, K8s, Sonarqube, Docker, DevOps