Automating Build, Packaging, and Deployment for the Mcommerce Application Using DevSecOps

Authored by :

Mr. MOARRAF Yassine

Mr. ElOUAZZANI Soufian

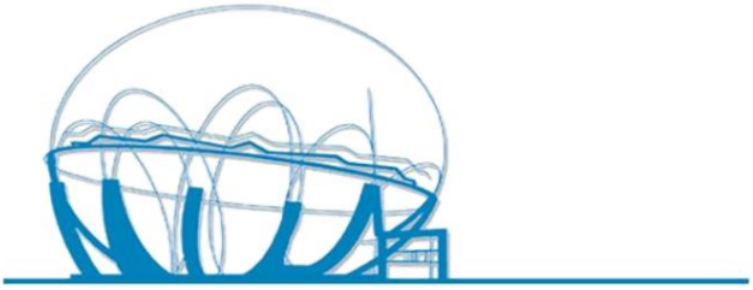
Mr. ELMOUMEN Yassine

Mr. TOURÉ Abdoulaye

Mr. TANAFAAT Marouane

Under the supervision of:

Prof. Driss ALLAKI



Agence National de Réglementation des Télécommunications

Institut National des Postes et Télécommunications

Class of : 2024/2025

Project repositories:

Main repository: https://gitlab.com/iccn7501450/cicd-demo

IaC repository: https://gitlab.com/adrian_v/product-microservice-kubernetes

List of Figures

List of Tables

Listings

Contents

Chapter 1

General Project Context

In today's fast-paced and highly competitive software industry, the ability to deliver high-quality solutions quickly and securely is a critical factor for success. Companies are increasingly adopting modern software development practices, such as DevOps and DevSecOps, to streamline their processes, enhance collaboration, and ensure the security and reliability of their applications. For KATA, a young and ambitious software development company, this shift represents both an opportunity and a challenge. While KATA is known for delivering high-quality software solutions, its current delivery process is slower compared to industry competitors, which limits its ability to seize new market opportunities effectively.

To address these challenges, KATA has decided to explore the DevSecOps approach, which integrates development, security, and operations into a seamless workflow. This approach aims to automate and secure the software delivery pipeline, enabling faster and more reliable deployments without compromising on quality. As part of this initiative, KATA has launched a project to develop a Proof of Concept (PoC) for automating the build, packaging, and deployment of a microservices-based application called Mcommerce. This application, composed of three core microservices (Product, Order, and Payment), a web client, and an API Gateway, serves as a testbed for evaluating the effectiveness of DevSecOps practices.

The primary goal of this project is to demonstrate how automation, containerization, and security integration can improve KATA's software delivery process. By implementing a CI/CD pipeline, containerizing the application using Docker, orchestrating deployments with Kubernetes, and integrating security tools into the workflow, the project aims to achieve faster delivery cycles, enhanced security, and greater operational efficiency. The success of this PoC will provide KATA with valuable insights into the potential of DevSecOps and guide its future strategy for adopting modern software development practices.

This report outlines the methodology, implementation, and results of the PoC, highlighting the key achievements, challenges, and lessons learned throughout the project.

Chapter 2

Creating a CI pipeline

2.1 Overview of GitLab CI/CD

CI/CD is a continuous method of software development, where you continuously build, test, deploy, and monitor iterative code changes. This iterative process helps reduce the chance that you develop new code based on failed or buggy previous versions. GitLab CI/CD can catch bugs early in the development cycle and help ensure that the code deployed to production complies with your established code standards. Gitlab provides what are called Gitlab runners processes that pick up and execute CI/CD jobs.

For this part, we will focus only on the CI part of the pipeline to build, test, and containerize the microservice.

2.1.1 jobs

Jobs are a list of commands to execute to accomplish tasks like building, testing, or deploying code. they are independent of other jobs and can be run in parallel. Jobs can have many status like: failed, warning, pending, running.

2.1.2 Pipelines

Pipelines are composed of:

- Global YAML keywords that control the overall behavior of the project's pipelines
- Jobs that execute commands to accomplish a task. For example, a job could compile, test, or deploy code. Jobs run independently from each other, and are executed by runners.

- Stages, which define how to group jobs together. Stages run in sequence, while the jobs in a stage run in parallel. For example, an early stage could have jobs that lint and compile code, while later stages could have jobs that test and deploy code. If all jobs in a stage succeed, the pipeline moves on to the next stage. If any job in a stage fails, the next stage is not (usually) executed and the pipeline ends early.

2.1.3 Runners

Runners are programs that their responsibility is to execute CI/CD jobs. In GitLab there are two types of runners, one's that run on physical machines and other in virtual instances. More concretely, in the ".gitlab-ci.yml" file that we will see next, you can specify a container image that you want to use when running the job. The runner loads the image, clones your project, and runs the job locally or in the container.

2.2 Creating a CI pipeline in GitLab

In this part we will discuss how to create a CI/CD pipeline in Gitlab.

2.2.1 The ".gitlab-ci.yml" file

The file ".gitlab-ci.yml" is a special yaml file that exists in the root directory of a repo. Its the main file that defines a pipline as it specifies the stages, jobs, and scripts to be executed during the CI/CD pipeline.

2.2.2 Syntax of ".gitlab-ci.yml" file

Generally the file follows the following format

```
stages:
    - build
    - test
    - deploy
variables:
    PROJECT_NAME: "my_project"
```

```
    BUILD_DIR: "build"

build:
    stage: build # Part of the build stage
    script: ...
test:
    stage: test # Part of the test stage
    script:
        - echo "Running tests..."
        - echo "Tests passed."
deploy:
    stage: deploy # Part of the deploy stage
    script: ...
```

First we define our stages and then for each stage we defines jobs we want to be executed part of that stage. A typical pipeline have 3 stages: build, test and deploy.

In our case we have a pipeline with two stages: Build/Test and Containerize, thus we will use the following format:

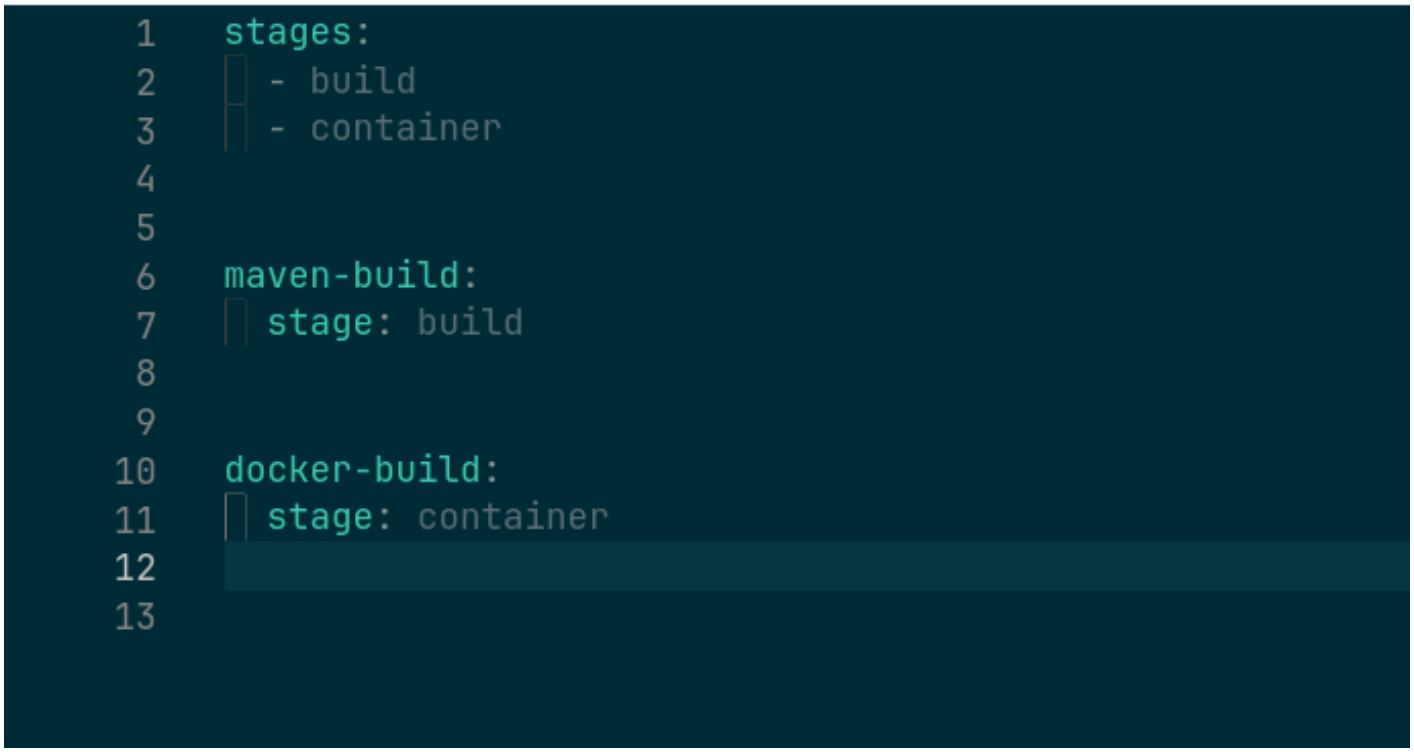


Figure 2.1: Initial .gitlab-ci.yml

2.2.3 Compilation and testing stage

Our microservice is written using Java 18 and uses Maven as a build system and for testing. These factors result in the following build job description.

```
1  stages:
2    - build
3    - container
4
5
6  maven-build:
7    stage: build
8    image: maven:3.8.7-openjdk-18-slim
9    script:
10     - chmod +x ./mvnw
11     - ./mvnw clean package
12   artifacts:
13     paths:
14       - ./target/*.jar
```

Figure 2.2: Compilation and testing job

Explication:

- Line 6: Specify our job name.
- Line 7: Specify that this job is part of build stage.
- Line 8: We want our job to be executed in a container that provides JDK version 18 with maven.
- Line 9: Here we define what our job does using shell commands.
- Line 10: This command make our maven binary file executable.
- Line 11: Start the test and build process
- Line 12-13-14: Specify that the content of the target folder is an artifact so it won't be deleted after the job and we can access it in other jobs.

2.2.4 Containerization stage

We want to user docker to containerize our app and Dockerhub as a container registery to share our docker image. for this we will use the following job.

```
16 docker-build:
17   stage: container
18   image: docker
19   services:
20     - docker:dind
21   before_script:
22     - docker login -u projdevopsiccn2 -p $DOCKER_TOKEN
23   script:
24     - docker version
25     - docker build -t projdevopsiccn2/mcommerce .
26     - docker push projdevopsiccn2/mcommerce
27
```

Figure 2.3: Compilation and testing job

Explanation:

- Line 16: Specify our job name.
- Line 17: Specify that this job is part of the container stage.
- Line 18: We want our job to be executed in a container that provides Docker (Nested container situation)
- Line 19: Services are additions to the image we want to use
- Line 20: dind (docker inside docker) specify that we want to have the ability to run nested docker containers.
- Line 21: These are command that get ran before the main script.
- Line 22: Because we want to upload our result image to Dockerhub we need first to login.
  The login here uses a private variable DOCKER_TOKEN that we defined in the settings under: Settings > CI/CD > Variables

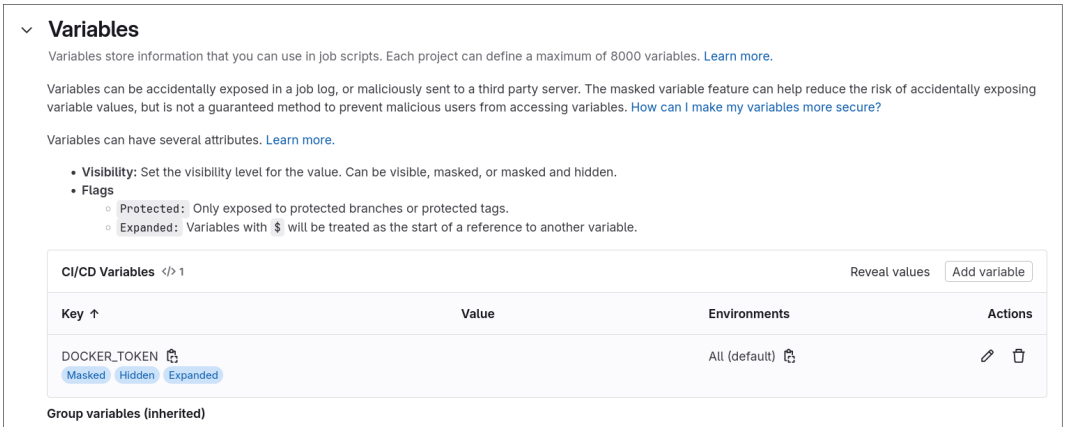


Figure 2.4: Private variable.

The token can be created from DockerHub under: Settings > Personal access tokens

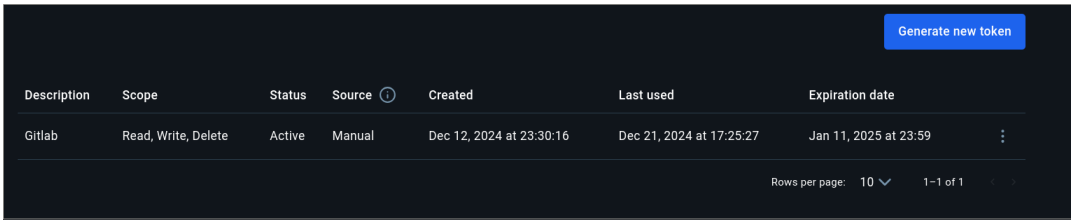


Figure 2.5: Private token in DockerHub

- Line 23: Here we define what our job does using shell commands
- Line 24: First we check if docker is working by showing its version
- Line 25: Then we build our image using Dockerfile in the repo
- Line 26: Then we push the resulting image to our Dockerhub repository.

2.3 Testing the result

The resulting ".gitlab-ci.yml" file is:

```
1  stages:
2    - build
3    - container
4
5
6  maven-build:
7    stage: build
8    image: maven:3.8.7-openjdk-18-slim
9    script:
10     - chmod +x ./mvnw
11     - ./mvnw clean package
12   artifacts:
13     paths:
14       - ./target/*.jar
15
16 docker-build:
17   stage: container
18   image: docker
19   services:
20     - docker:dind
21   before_script:
22     - docker login -u projdevopsiccn2 -p $DOCKER_TOKEN
23   script:
24     - docker version
25     - docker build -t projdevopsiccn2/mcommerce .
26     - docker push projdevopsiccn2/mcommerce
27
```

Figure 2.6: CI pipeline configuration inside .gitlab-ci.yml

Now We submit a dummy commit to test our CI pipeline. and the result is :

Figure 2.7 shows our pipeline running, and the commit that triggered it, also the different stages that exists in the pipeline. Figure 2.8 shows in more detail the status of each stage. Figures 2.9 and 2.10 shows log of the jobs "maven-build" and "docker-build" respectively, it also shows the duration, status and other information about each job.

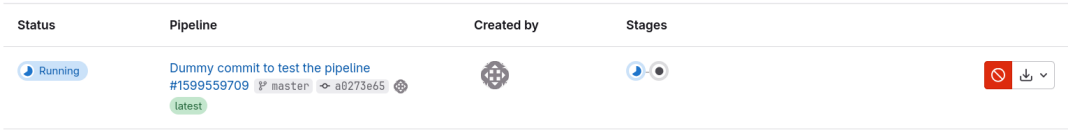


Figure 2.7: The CI pipeline running

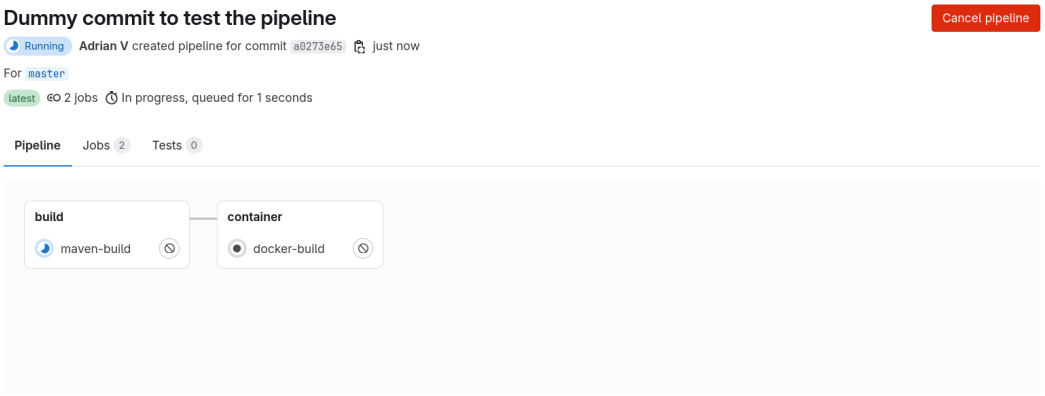


Figure 2.8: The CI pipeline running in more detail

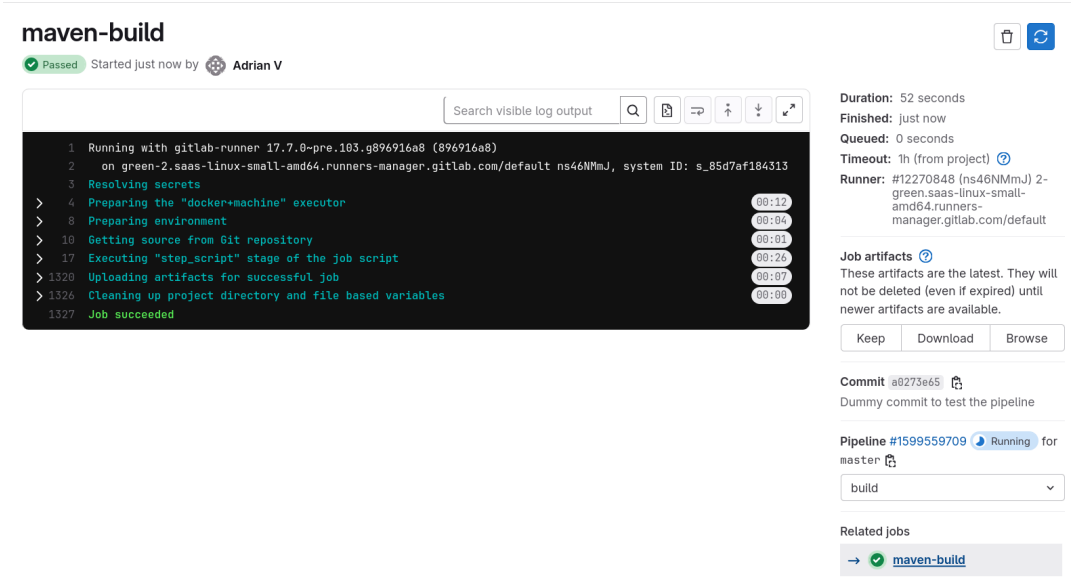


Figure 2.9: The maven-build job in more detail

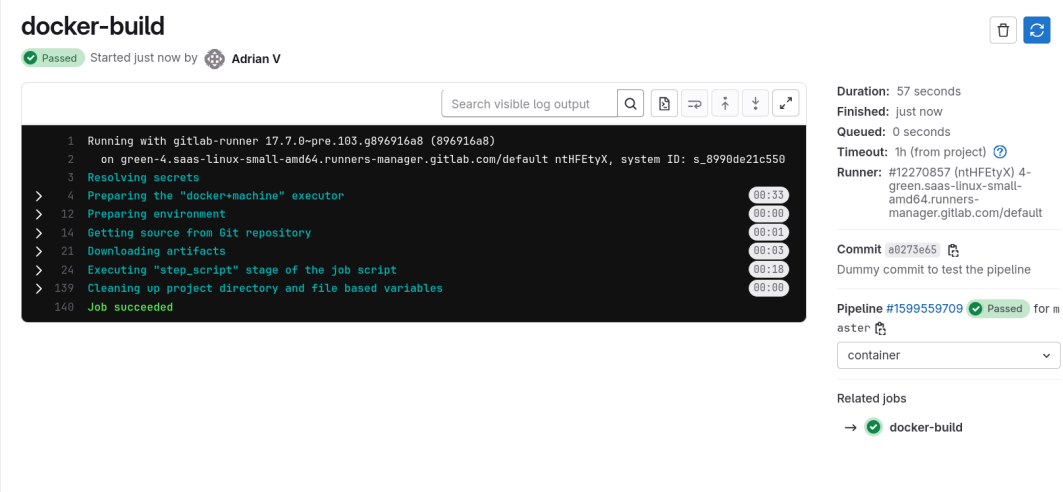


Figure 2.10: The docker-build job in more detail

Chapter 3

Local Deployment with Kubernetes

3.1 Overview of Technologies

In this chapter, we will explore the deployment of the **Product microservice** using container orchestration and automation tools. also we will use a VPS (Ubuntu machine) so we simulate a real word scenario and different members could connect and work on the same machine .The primary technologies utilized include **Kubernetes**, **Minikube**, and **kubectl**. Below is a brief description of these technologies:

- **Kubernetes**: A powerful open-source container orchestration platform designed to manage the deployment, scaling, and operation of containerized applications. Kubernetes simplifies the management of complex application architectures by automating deployment and ensuring high availability.
- **Minikube**: A lightweight Kubernetes implementation designed for local environments. It allows developers to create a single-node Kubernetes cluster on their machines for development and testing purposes.
- **kubectl**: A command-line tool used to interact with Kubernetes clusters. It provides commands to deploy applications, inspect resources, and troubleshoot issues within a Kubernetes environment.

By leveraging these tools, we demonstrate how to create a local Kubernetes cluster, configure a dedicated namespace, and deploy a microservice using declarative YAML configurations. This hands-on approach not only ensures efficient resource management but also prepares for scalable and production-ready deployments.

3.2 Installing Dependencies and Creating the Minikube Cluster

To begin, we will create a Kubernetes cluster on the VPS (Virtual Private Server) using Minikube. First, we connect to the VPS machine using SSH.

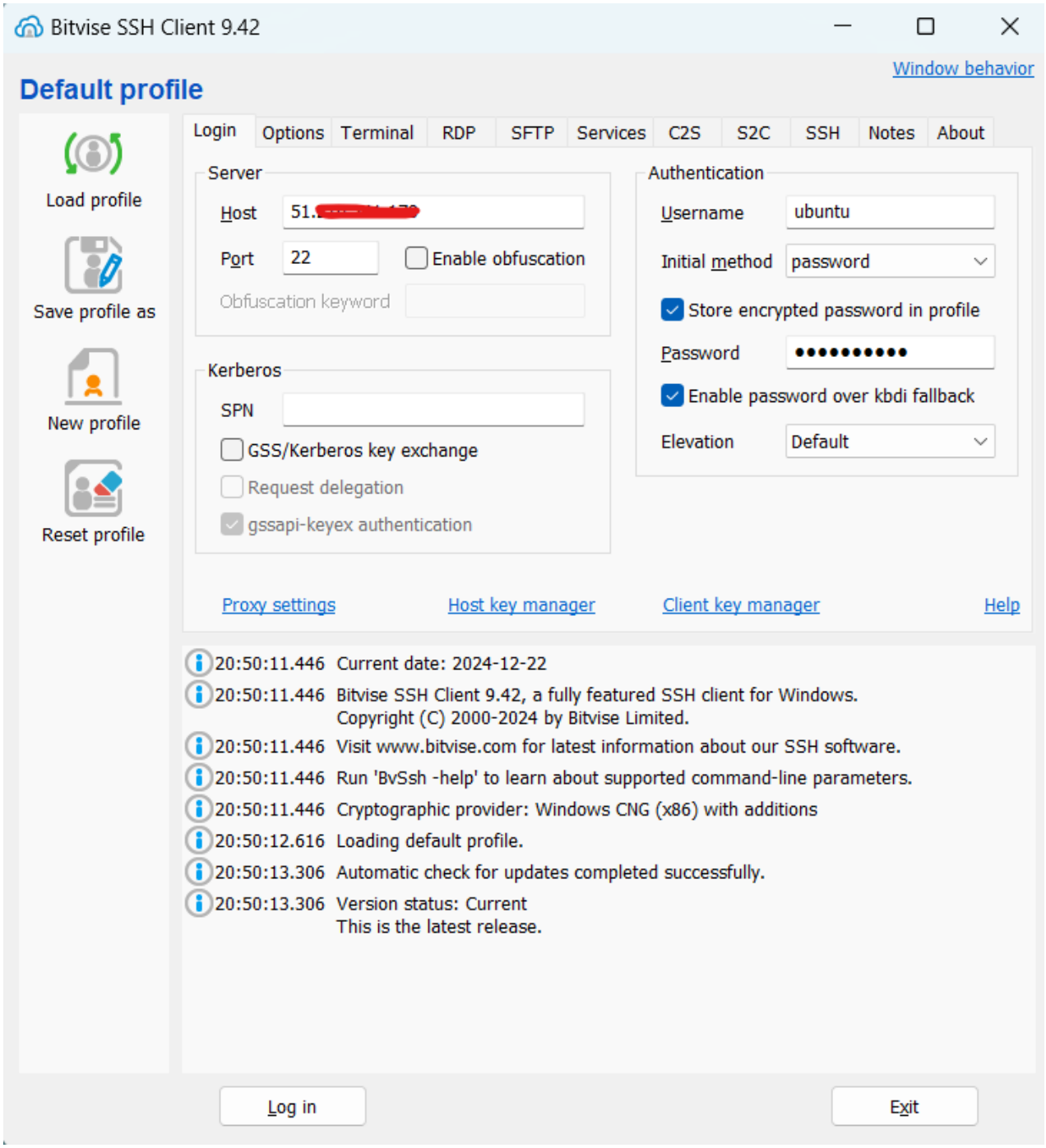


Figure 3.1: SSH Connection

After connecting to the VPS, and before installing Docker, it's a good idea to update the local packages to ensure we have the latest available versions.

```
ubuntu@vps-4f4e3327:~$ sudo apt update && sudo apt upgrade -y
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Get:2 http://nova.clouds.archive.ubuntu.com/ubuntu jammy InRelease [270 kB]
Get:3 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1997 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security/main Translation-en [317 kB]
Get:5 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [2701 kB]
Get:6 http://security.ubuntu.com/ubuntu jammy-security/restricted Translation-en [470 kB]
Get:7 http://nova.clouds.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:8 http://nova.clouds.archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:9 http://nova.clouds.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [2241 kB]
Get:10 http://nova.clouds.archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [378 kB]
Get:11 http://nova.clouds.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [2817 kB
]
Get:12 http://nova.clouds.archive.ubuntu.com/ubuntu jammy-updates/restricted Translation-en [490 kB]
Get:13 http://nova.clouds.archive.ubuntu.com/ubuntu jammy-backports/main amd64 Packages [91.5 kB]
Get:14 http://nova.clouds.archive.ubuntu.com/ubuntu jammy-backports/universe amd64 Packages [31.4 kB
```

Figure 3.2: Upgrades Packages

Next, we will install the required prerequisite packages.

```
ubuntu@vps-4f4e3327:~$ sudo apt-get install -y apt-transport-https ca-certificates curl gpg
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20240203~22.04.1).
ca-certificates set to manually installed.
curl is already the newest version (7.81.0-1ubuntu1.20).
curl set to manually installed.
gpg is already the newest version (2.2.27-3ubuntu2.1).
gpg set to manually installed.
The following packages were automatically installed and are no longer required:
  linux-headers-5.15.0-127 linux-headers-5.15.0-127-generic linux-image-5.15.0-127-generic
  linux-modules-5.15.0-127-generic
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  apt-transport-https
```

Figure 3.3: Adding certificates

Now, let's add Docker's GPG key and the Docker repository to our system.

```
ubuntu@vps-4f4e3327:~$ sudo install -m 0755 -d /etc/apt/keyrings
ubuntu@vps-4f4e3327:~$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings
/docker.asc
ubuntu@vps-4f4e3327:~$ sudo chmod a+r /etc/apt/keyrings/docker.asc
ubuntu@vps-4f4e3327:~$ echo \
>   "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.doc
ker.com/linux/ubuntu \
>   $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
>   sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
ubuntu@vps-4f4e3327:~$ sudo apt-get update
Hit:1 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:2 http://nova.clouds.archive.ubuntu.com/ubuntu jammy InRelease
Get:3 http://nova.clouds.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:4 https://download.docker.com/linux/ubuntu jammy InRelease [48.8 kB]
Get:5 http://nova.clouds.archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:6 https://download.docker.com/linux/ubuntu jammy/stable amd64 Packages [42.5 kB]
Fetched 347 kB in 1s (475 kB/s)
Reading package lists... Done
```

Figure 3.4: Adding GPG Docker Key

```
ubuntu@vps-4f4e3327:~$  sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  docker-ce-rootless-extras libltdl7 libslirp0 pigz slirp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli docker-ce-rootless-extras
  docker-compose-plugin libltdl7 libslirp0 pigz slirp4netns
0 upgraded, 10 newly installed, 0 to remove and 1 not upgraded.
Need to get 124 MB of archives.
After this operation, 446 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 https://download.docker.com/linux/ubuntu jammy/stable amd64 containerd.io amd64 1.7.24-1 [29.5 MB
Get:2 http://nova.clouds.archive.ubuntu.com/ubuntu jammy/universe amd64 pigz amd64 2.6-1 [63.6 kB]
```

Figure 3.5: Installing Docker

We then verify that Docker is successfully installed:

```
ubuntu@vps-4f4e3327:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu@vps-4f4e3327:~$ sudo docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
89ad04b7fd33   hello-world   "/hello"   47 seconds ago   Exited (0) 46 seconds ago             heuristic
 faraday
```

Figure 3.6: Docker Ps Command

Now, let's download the latest Minikube binary:

```
zenlnex@vps-4f4e3327:/home/zenlnex$ curl -LO https://storage.googleapis.com/minikube/releases/latest
/minikube-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 99.0M  100 99.0M    0     0  26.2M      0  0:00:03  0:00:03 --:--:-- 26.2M
```

Figure 3.7: Downloading Minikube binary

Once the binary is downloaded, we can install Minikube:

```
zenlnex@vps-4f4e3327:/home/zenlnex$ sudo install minikube-linux-amd64 /usr/local/bin/minikube
zenlnex@vps-4f4e3327:/home/zenlnex$ minikube version
minikube version: v1.34.0
commit: 210b148df93a80eb872ecbeb7e35281b3c582c61
```

Figure 3.8: Installing Minikube binary

Next, let's create the Kubernetes cluster locally using Minikube:

```
zenlnex@vps-4f4e3327:/home/zenlnex$ sudo minikube start --driver=docker --force
* minikube v1.34.0 on Ubuntu 22.04 (kvm/amd64)
! minikube skips various validations when --force is supplied; this may lead to unexpected behavior
* Using the docker driver based on user configuration
* The "docker" driver should not be used with root privileges. If you wish to continue as root, use --force.
* If you are running minikube within a VM, consider using --driver=none:
*   https://minikube.sigs.k8s.io/docs/reference/drivers/none/
* Using Docker driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.45 ...
* Downloading Kubernetes v1.31.0 preload ...
    > preloaded-images-k8s-v18-v1...:  326.69 MiB / 326.69 MiB  100.00% 24.57 M^[[B
    > gcr.io/k8s-minikube/kicbase...:  486.80 MiB / 487.90 MiB  99.77% 32.81 Mi
* Creating docker container (CPUs=2, Memory=2200MB) ...
* Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Figure 3.9: Starting Minikube

After installing the latest version of kubectl using Snap:

```
zenlnex@vps-4f4e3327:/home/zenlnex$ sudo snap install kubectl --classic
kubectl 1.31.4 from Canonical✓ installed
zenlnex@vps-4f4e3327:/home/zenlnex$
```

Figure 3.10: Installing kubectl

Finally, let's check that kubectl is installed successfully:

```
zenlnex@vps-4f4e3327:/home/zenlnex$ sudo kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   11m   v1.31.0
```

Figure 3.11: Listing nodes by kubectl

3.3 Configuring the Namespace

Let configure a namespace named product-local-use to logically separate the resources related to the Product microservice. using the command:

```
zenlnex@vps-4f4e3327:/home/zenlnex$ sudo kubectl create namespace product-local-use
namespace/product-local-use created
```

Figure 3.12: Creating Namespace

The namespace provides a dedicated space to manage all Kubernetes objects for the microservice.

3.4 Adding YAML Files for Deployment and Service

The deployment and service YAML files manage the microservice's lifecycle and expose it to external access:

- **Deployment YAML**: Configures the number of replicas, container image, and resource specifications for the Product microservice.
- **Service YAML**: Defines the network access point for the microservice using a ClusterIP.

Now, let's clone our `cicd-demo` repository from GitLab in order to add the YAML files for the deployment.

```
ubuntu@vps-4f4e3327:/mcommerce$ sudo git clone https://gitlab.com/iccn7501450/cicd-demo.git
Cloning into 'cicd-demo'...
Username for 'https://gitlab.com': g1d3on_wh1t3
Password for 'https://g1d3on_wh1t3@gitlab.com':
remote: Enumerating objects: 130, done.
remote: Counting objects: 100% (60/60), done.
remote: Compressing objects: 100% (58/58), done.
remote: Total 130 (delta 31), reused 0 (delta 0), pack-reused 70 (from 1)
Receiving objects: 100% (130/130), 23.37 KiB | 1.80 MiB/s, done.
Resolving deltas: 100% (49/49), done.
ubuntu@vps-4f4e3327:/mcommerce$ ls
cicd-demo
```

Figure 3.13: Cloning Microservice repository

After that, let's create a `.Kubernetes` folder that will contain the `product-deployment` and `product-service` YAML files for deployment and service.

3.4.1 Deployment YAML File

The following YAML file describes the deployment configuration for the **mcommerce** microservice in a Kubernetes cluster:

```
zenlnex@vps-4f4e3327:/mcommerce/cicd-demo/.kubernetes$ cat product-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mcommerce
  labels:
    app: mcommerce
spec:
  replicas: 2
  selector:
    matchLabels:
      app: mcommerce
  template:
    metadata:
      labels:
        app: mcommerce
    spec:
      containers:
      - name: mcommerce
        image: projdevopsiccn2/mcommerce:latest
        ports:
        - containerPort: 9001
```

Figure 3.14: Product-deployment YAML file

Now, let's describe this file:

- **apiVersion** and **kind: Deployment**: Specifies that this YAML file defines a Deployment, which ensures the desired number of application instances are running.
- **metadata**: Contains identifying information, including the deployment name `mcommerce` and a label.
- **spec.replicas: 2**: Ensures that two pod replicas of the `mcommerce` microservice are running for redundancy and load balancing.
- **spec.selector.matchLabels**: Matches pods labeled with `app: mcommerce`, ensuring that the deployment manages the intended pods.
- **spec.template**: Defines the pod template, including:

- **metadata.labels**: Assigns the label `app: mcommerce` to the pods created by this deployment.
- **spec.containers**: Configures the container, specifying:
  - **name: mcommerce**: The container name.
  - **image: projdevopsiccn2/mcommerce:latest**: The container image to use.
  - **ports.containerPort: 9001**: Exposes port 9001 inside the container.

This configuration ensures that the `mcommerce` microservice is deployed with two replicas and is accessible on port 9001.

3.4.2 Service YAML File

The following YAML file describes the service configuration for exposing the **mcommerce** microservice in a Kubernetes cluster:

```
zenlnex@vps-4f4e3327:/mcommerce/cicd-demo/.kubernetes$ cat product-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: product-service
spec:
  selector:
    app: mcommerce
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9001
```

Figure 3.15: Product-service YAML file

Now, let's describe this file:

- **apiVersion: v1** and **kind: Service**: Specifies that this YAML defines a Service, which provides a stable network interface to access the microservice.
- **metadata**: Contains the service name (`product-service`), which identifies this resource.
- **spec.selector**: Matches pods labeled with `app: mcommerce`, ensuring that the service routes traffic to the appropriate pods managed by the deployment.
- **spec.ports**: Configures the ports for communication:
  - **protocol: TCP**: Specifies TCP as the communication protocol.

- **port: 80**: Defines the external port for accessing the service.
- **targetPort: 9001**: Maps traffic from the external port 80 to the container's internal port 9001.

3.4.3 Applying Deployment and Service to the Cluster

Let's apply our YAML files to the cluster using the following command:

```
zenlnex@vps-4f4e3327:/mcommerce/cicd-demo/.kubernetes$ kubectl apply -f product-deployment.yaml -n
zenlnex@vps-4f4e3327:/mcommerce/cicd-demo/.kubernetes$ sudo kubectl apply -f product-deployment.yaml -n product-local-use
deployment.apps/mcommerce created
zenlnex@vps-4f4e3327:/mcommerce/cicd-demo/.kubernetes$ sudo kubectl apply -f product-service.yaml -n product-local-use
service/product-service unchanged
```

Figure 3.16: Applying the product-service and product-deployment files

3.5 Verifying Kubernetes Resources

To confirm the successful creation of resources, we use the following kubectl commands:

3.5.1 Check Pods:

```
zenlnex@vps-4f4e3327:/mcommerce/cicd-demo/.kubernetes$ sudo kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
mcommerce-74d49b66fc-cbplx    1/1     Running   0          2m8s
mcommerce-74d49b66fc-g6bmb    1/1     Running   0          2m8s
```

Figure 3.17: Checking Pods

The screenshot demonstrates the pods running as expected.

3.5.2 Check Service:

```
zenlnex@vps-4f4e3327:/mcommerce/cicd-demo/.kubernetes$ sudo kubectl get svc
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes        ClusterIP   10.96.0.1       <none>        443/TCP   77m
product-service   ClusterIP   10.98.190.212   <none>        80/TCP    3m58s
zenlnex@vps-4f4e3327:/mcommerce/cicd-demo/.kubernetes$
```

Figure 3.18: Checking Services

The screenshot highlights the service exposing the microservice.

3.5.3 Check Deployments:

```
zenlnex@vps-4f4e3327:/mcommerce/cicd-demo/.kubernetes$ sudo kubectl get deployments

NAME        READY   UP-TO-DATE   AVAILABLE   AGE
mcommerce   2/2     2            2           8m31s
zenlnex@vps-4f4e3327:/mcommerce/cicd-demo/.kubernetes$ sudo kubectl describe deployments mcommerce
Name:                   mcommerce
Namespace:              default
CreationTimestamp:      Wed, 18 Dec 2024 15:53:31 +0000
Labels:                 app=mcommerce
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=mcommerce
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=mcommerce
  Containers:
   mcommerce:
    Image:         projdevopsiccn2/mcommerce:latest
    Port:          9001/TCP
    Host Port:     0/TCP
    Environment:   <none>
    Mounts:        <none>
  Volumes:         <none>
  Node-Selectors:  <none>
  Tolerations:     <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   mcommerce-74d49b66fc (2/2 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  9m1s  deployment-controller  Scaled up replica set mcommerce-74d49b66fc to 2
```

Figure 3.19: Checking Deployments

The screenshot shows the deployment status, indicating successful pod creation.

3.5.4 Check ReplicaSet:

```
zenlnex@vps-4f4e3327:/mcommerce/cicd-demo/.kubernetes$ sudo kubectl get replicasets
NAME                   DESIRED   CURRENT   READY   AGE
mcommerce-74d49b66fc   2         2         2       11m
zenlnex@vps-4f4e3327:/mcommerce/cicd-demo/.kubernetes$ sudo kubectl describe replicasets mcommerce-74d49b66fc
Name:           mcommerce-74d49b66fc
Namespace:      default
Selector:       app=mcommerce,pod-template-hash=74d49b66fc
Labels:         app=mcommerce
                pod-template-hash=74d49b66fc
Annotations:    deployment.kubernetes.io/desired-replicas: 2
                deployment.kubernetes.io/max-replicas: 3
                deployment.kubernetes.io/revision: 1
Controlled By:  Deployment/mcommerce
Replicas:       2 current / 2 desired
Pods Status:    2 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=mcommerce
           pod-template-hash=74d49b66fc
  Containers:
   mcommerce:
    Image:         projdevopsiccn2/mcommerce:latest
    Port:          9001/TCP
    Host Port:     0/TCP
    Environment:   <none>
    Mounts:        <none>
  Volumes:         <none>
  Node-Selectors:  <none>
  Tolerations:     <none>
Events:
  Type    Reason            Age   From                   Message
  ----    ------            ----  ----                   -------
  Normal  SuccessfulCreate  11m   replicaset-controller  Created pod: mcommerce-74d49b66fc-cbplx
  Normal  SuccessfulCreate  11m   replicaset-controller  Created pod: mcommerce-74d49b66fc-g6bmb
```

Figure 3.20: Checking ReplicaSet

The screenshot verifies the ReplicaSet managing the pods.

3.5.5 Check All Components:

```
zenlnex@vps-4f4e3327:/mcommerce/cicd-demo/.kubernetes$ sudo kubectl get all -o wide -n=product-local-use
NAME                                READY   STATUS    RESTARTS   AGE     IP           NODE       NOMINATED NODE   READINESS GATES
pod/mcommerce-74d49b66fc-f87n7   1/1     Running   0          4m53s   10.244.0.6   minikube   <none>           <none>
pod/mcommerce-74d49b66fc-z8fc7   1/1     Running   0          4m53s   10.244.0.5   minikube   <none>           <none>

NAME                      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE   SELECTOR
service/product-service   ClusterIP   10.101.21.168   <none>        80/TCP    13m   app=mcommerce

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE     CONTAINERS   IMAGES                           SELECTOR
deployment.apps/mcommerce   2/2     2            2           4m53s   mcommerce    projdevopsiccn2/mcommerce:latest   app=mcommerce

NAME                                      DESIRED   CURRENT   READY   AGE     CONTAINERS   IMAGES                           SELECTOR
replicaset.apps/mcommerce-74d49b66fc   2         2         2       4m53s   mcommerce    projdevopsiccn2/mcommerce:latest   app=mcommerce,pod-template-hash=74d49b66fc
```

Figure 3.21: Displaying Resource Details

This screenshot displays detailed information about the resources in the Kubernetes cluster, such as deployments, pods, ReplicaSets, and services. It helps verify that all components are running correctly

and provides useful information like pod IPs and node assignments.

```
zenlnex@vps-4f4e3327:/mcommerce/cicd-demo/.kubernetes$ curl http://localhost:8001/api/v1/namespaces/product-local-use/services/product-service:80/proxy/ | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   388    0   388    0     0   3629      0 --:--:-- --:--:-- --:--:--  3695
[
  {
    "id": 1,
    "name": "Morocco Football Home Kit",
    "description": "Morocco 2022 World Cup Home Kit with Red and Green Colors",
    "price": 200
  },
  {
    "id": 2,
    "name": "Morocco Football Away Kit",
    "description": "Morocco 2022 World Cup Away Kit with White Color",
    "price": 200
  },
  {
    "id": 3,
    "name": "Morocco Football Pullover",
    "description": "Morocco 2022 World Cup Pullover with Red and Green Colors",
    "price": 500
  }
]
```

Figure 3.22: Accessing Data Hosted by the Product Microservice

This screenshot demonstrates that the data within the `mcommerce` microservice is accessible. It shows the successful connection to the service and retrieval of the data, confirming that the microservice is functioning as expected.

Chapter 4

Extension of the CI pipeline with GitOps

4.0.1 What is GitOps

Infrastructure as code give us the ability to describe our deployment environment using machine readable code. This help us to automate and manage infrastructure efficiently. GitOps leverages version control systems, like Git, to enable versioned and auditable workflows for managing code that's used for infrastructure and application deployments.

4.0.2 Using ArgoCD

Argo CD is a declarative, GitOps continuous delivery tool for Kubernetes. It enables users to manage and automate the deployment of applications in Kubernetes clusters by leveraging Git repositories as the source of truth for application states.

4.0.3 Preparing the IaC repository

So the first thing we did is creating a repository to hold our IaC.

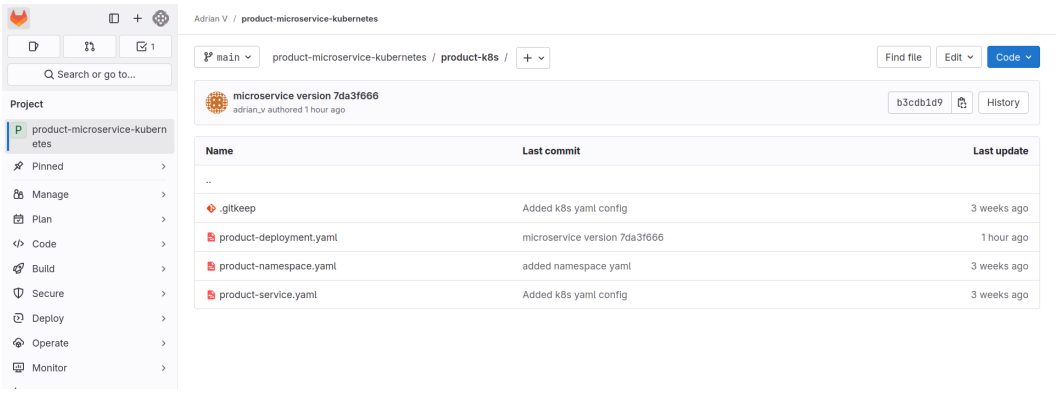


Figure 4.1

As we can see these are the same yaml file used in previous chapter.

The next important thing is to add a new job in our project CI pipeline to update the IaC repository to deploy the newly pushed docker image pushed by the CI pipeline.

```
gitops-k8s-deploy:
  rules:
    - if: $CI_COMMIT_BRANCH == "master"
      changes:
        paths:
          - Dockerfile
          - src/**/*
  image: bitnami/git:latest
  stage: deploy-k8s
  before_script:
    - git config --global user.email "62484408+terriblemoment0836x@users.noreply.github.com"
    - git config --global user.name "adrian_v"
  script:
    - git clone --depth 1 "https://gitlab-ci:$K8S_REPO_TOKEN@gitlab.com/adrian_v/product-microservice-kubernetes.git"
    - cd product-microservice-kubernetes/product-k8s
    - sed -i "s/projdevopsiccn2.*$/projdevopsiccn2\/mcommerce:$TAG/g" product-deployment.yaml
    - cat product-deployment.yaml
    - git add product-deployment.yaml
    - git commit -m "microservice version $TAG"
    - git push -uf origin main
```

Figure 4.2

4.0.4 Argo CD configuration

Now we need to install Argo CD in our kubernetes cluster and point it to the IaC repository.

```
D77R@vps-4f4e3327:~$ kubectl create namespace argocd
namespace/argocd created
D77R@vps-4f4e3327:~$ kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
customresourcedefinition.apiextensions.k8s.io/applications.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/applicationsets.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/appprojects.argoproj.io created
serviceaccount/argocd-application-controller created
serviceaccount/argocd-applicationset-controller created
serviceaccount/argocd-dex-server created
serviceaccount/argocd-notifications-controller created
serviceaccount/argocd-redis created
serviceaccount/argocd-repo-server created
serviceaccount/argocd-server created
role.rbac.authorization.k8s.io/argocd-application-controller created
role.rbac.authorization.k8s.io/argocd-applicationset-controller created
role.rbac.authorization.k8s.io/argocd-dex-server created
role.rbac.authorization.k8s.io/argocd-notifications-controller created
```

Figure 4.3: Installing Argo CD

To access Argo CD web UI we use the command 'kubectl port-forward'

```
D77R@vps-4f4e3327:~$ kubectl port-forward svc/argocd-server -n argocd 8080:443
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

Figure 4.4: Port forwarding to access Argo CD

Then using ssh port forwarding we can access the UI locally.

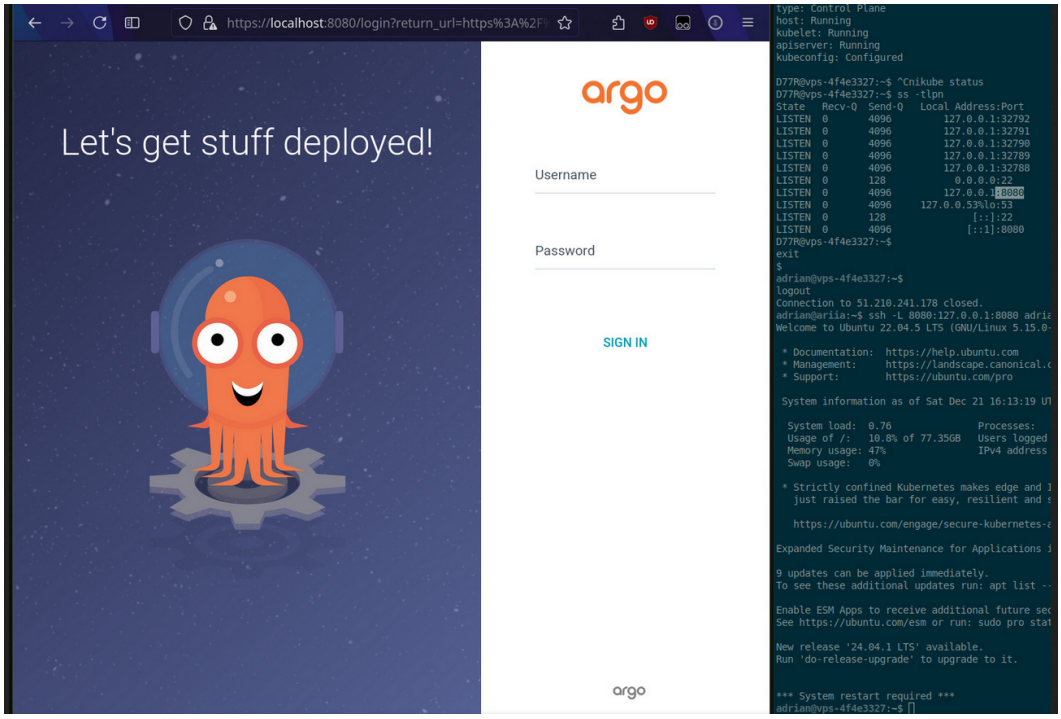


Figure 4.5: Ssh tunneling to access the UI locally

Then we add the IaC repository to Argo CD

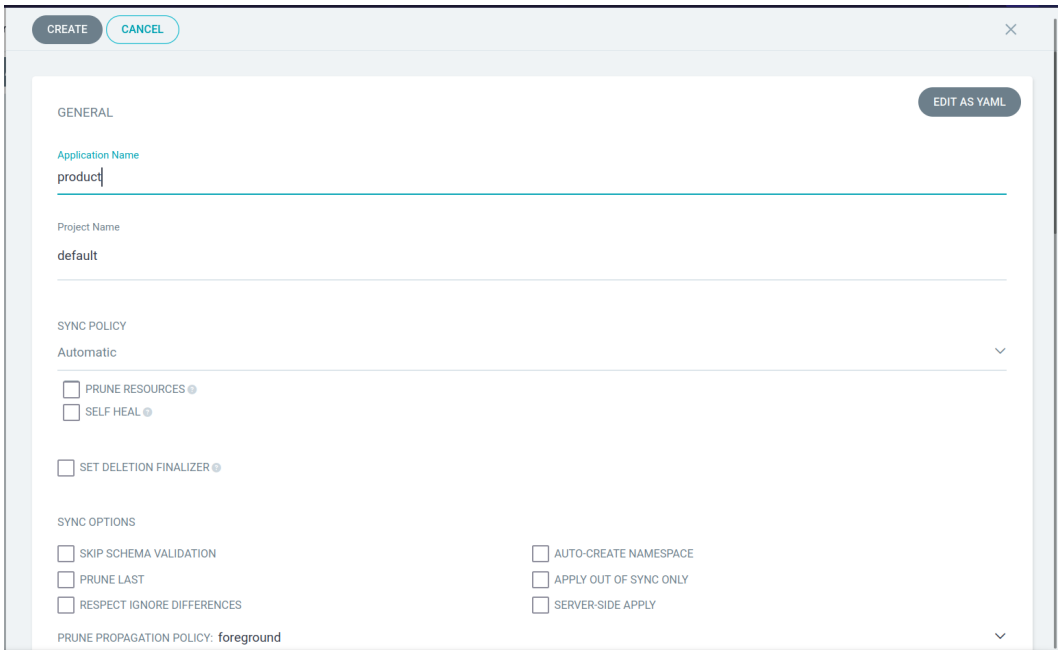


Figure 4.6

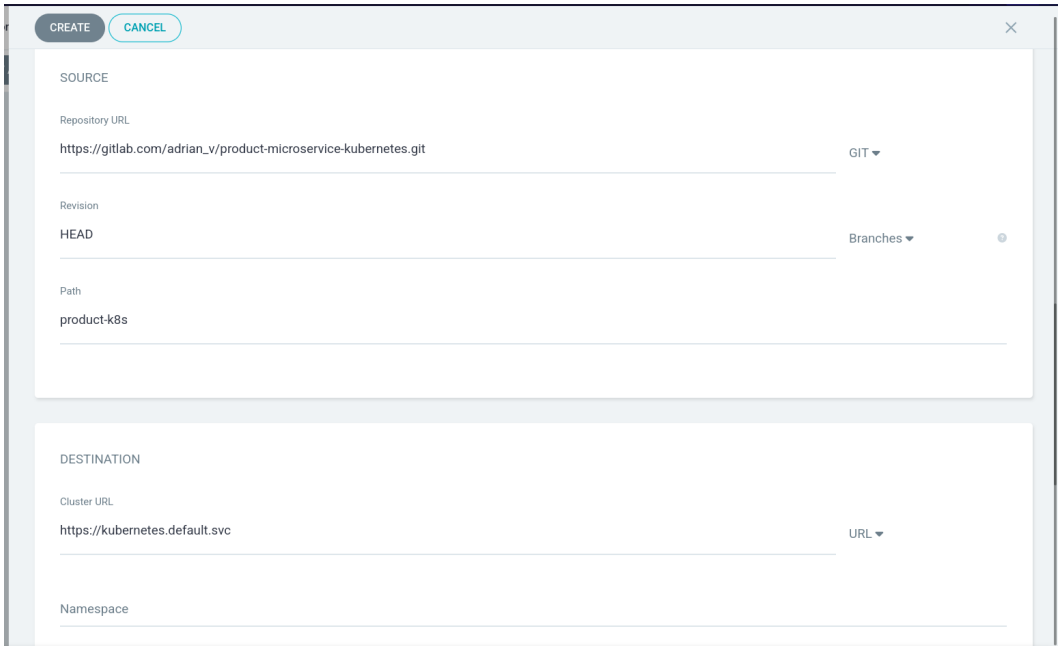


Figure 4.7

And as a result, we can see that our repository got synced, and the deployment is done.

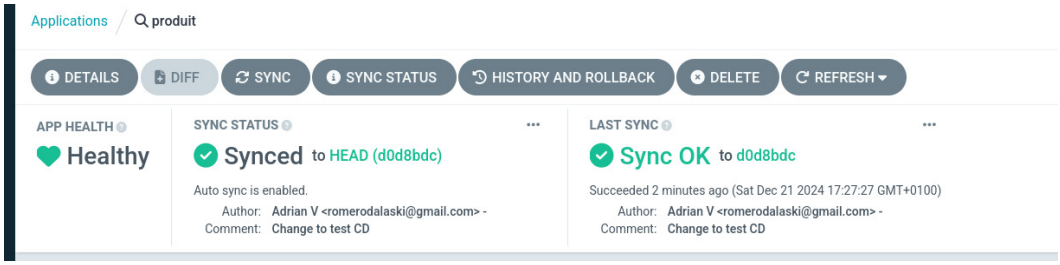


Figure 4.8

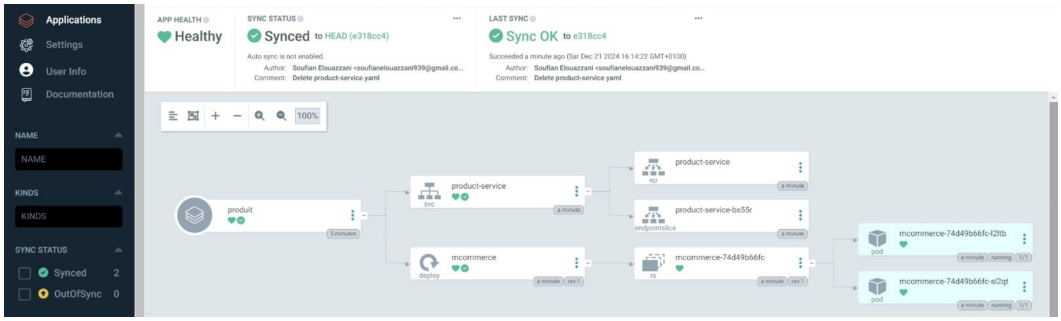


Figure 4.9

Chapter 5

Optimisation et Sécurisation du VCS et du CI Pipeline

5.1 Optimizing the CI pipeline

In this part we will implement some practices to help optimize our CI pipeline execution in terms of: Speed of execution, downloaded Data and configuration organization.
The main practices we will discuss are:

- Running CI inside a local GitLab Runner.
- Job rules.
- Caching.
- Generic jobs.

Next, we will discuss each practice, its benefits, and implementation.

5.1.1 CI in Local GitLab Runner

A GitLab Runner is an open-source application that is used to run jobs in a GitLab CI/CD (Continuous Integration/Continuous Deployment) pipeline. It executes scripts defined in a .gitlab-ci.yml file, allowing for automated testing, building, and deployment of code. By default GitLab provides its own GitLab runners that run jobs inside a docker container. One of the drawbacks of using the GitLab provided ones is:

- **Resource Limitations:** GitLab's shared runners may have limited resources (CPU, memory, etc.), leading to slower job execution times, especially during peak usage periods.
- **Queue Times:** Shared runners can experience longer queue times due to high demand, delaying the execution of CI/CD pipelines and impacting development cycles.
- **Customization:** Running your own GitLab Runner allows for greater customization of the environment, including specific software versions, dependencies, and configurations required for your projects.
- **Security and Compliance:** Using shared runners may pose security risks, as jobs run in a shared environment. Custom runners ensure sensitive data and proprietary code are handled securely.

Thus it's beneficial for us to run the CI pipeline in our own GitLab runner. Now to achieve that we will run a GitLab Runner in a Docker container (docker executor). So the first thing is to download the gitlab-runner image:

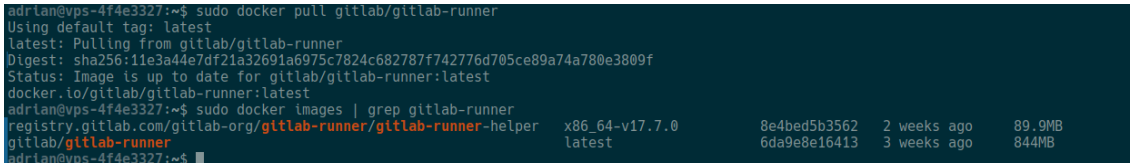


Figure 5.1

Then we create a volume to persist our runner configuration.

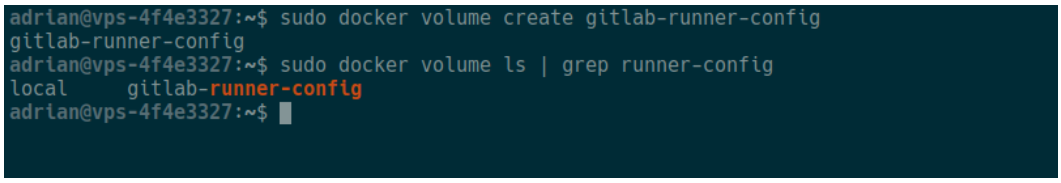


Figure 5.2

Now we need to configure the runner by editing the file 'config.toml' inside the volume we created.

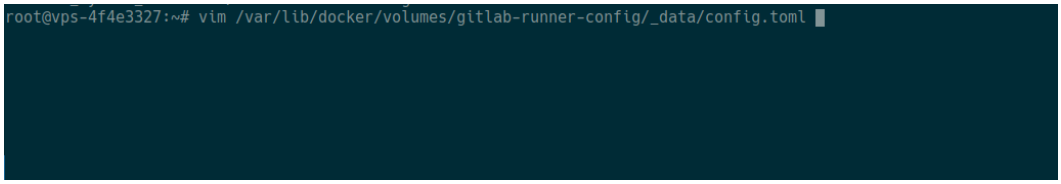


Figure 5.3

```
concurrent = 4
check_interval = 0
shutdown_timeout = 0

[session_server]
#  listen_address = "[::]:8093"
  session_timeout = 1800

[[runners]]
  name = "main-runner"
  url = "https://gitlab.com"
  executor = "docker"
  token = "xxxxxxxxxxxxxxxxxx"
  [runners.docker]
    image = "docker:dind"
    privileged = true
    disable_cache = false
    cache_dir = ""
    volumes = ["/certs/client", "/cache"]
~
~
```

Figure 5.4

The provided configuration file sets up a GitLab Runner with the following key features:

- **Concurrency:** The runner can execute up to 4 jobs simultaneously.
- **Shutdown Behavior:** There is no timeout for shutting down the runner (shutdown timeout set to 0).
- **Session Management:** A session server is configured with a 30-minute session timeout.
- **Runner Details:**
  - The runner is named "main-runner."
  - It connects to the GitLab instance at `https://gitlab.com`.
  - It uses the Docker executor to run jobs.
  - A registration token is specified.
- **Docker Configuration:**
  - The runner uses the `docker:dind` image for Docker-in-Docker functionality.
  - It runs in privileged mode to allow Docker commands within the container.
  - Caching is enabled to speed up builds, and specific volumes are mounted for sharing data between jobs (artifacts).

Now to obtain the GitLab runner token we need to create a new GitLab runner in our GitLab Project.

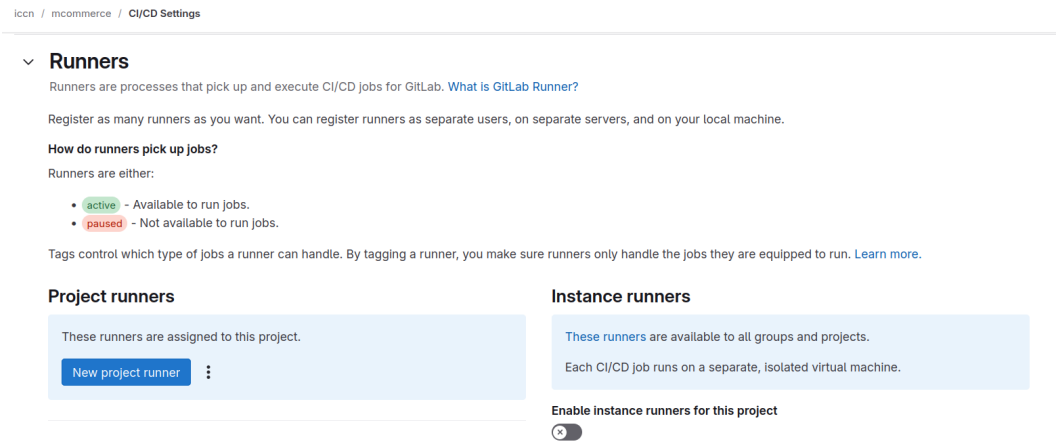


Figure 5.5

First we disable Giltab provided instance runners and we create a new project runner

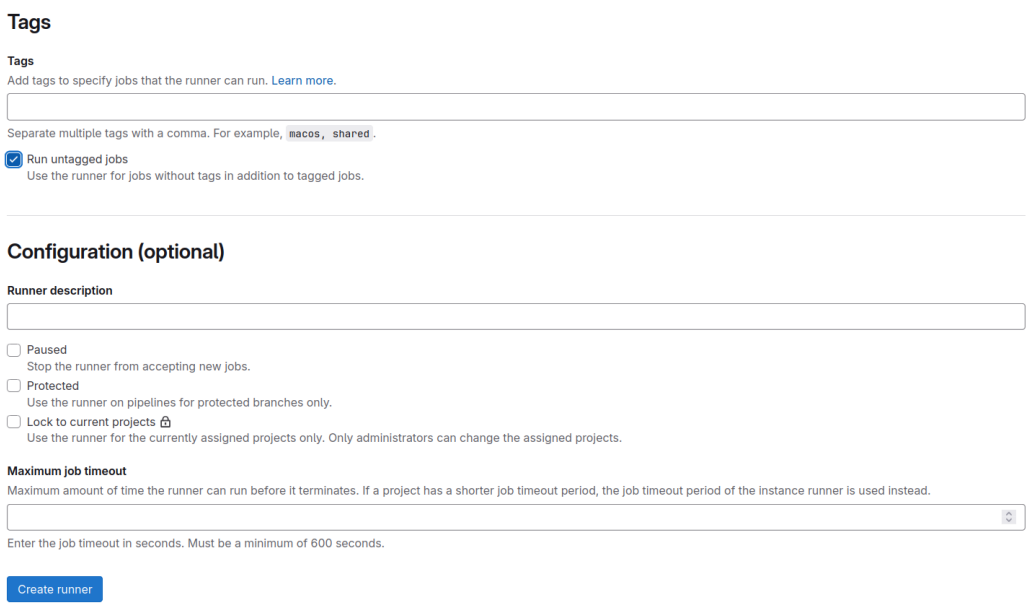


Figure 5.6

After we click 'Create runner" a token appears that we will need to put in the config.toml file.

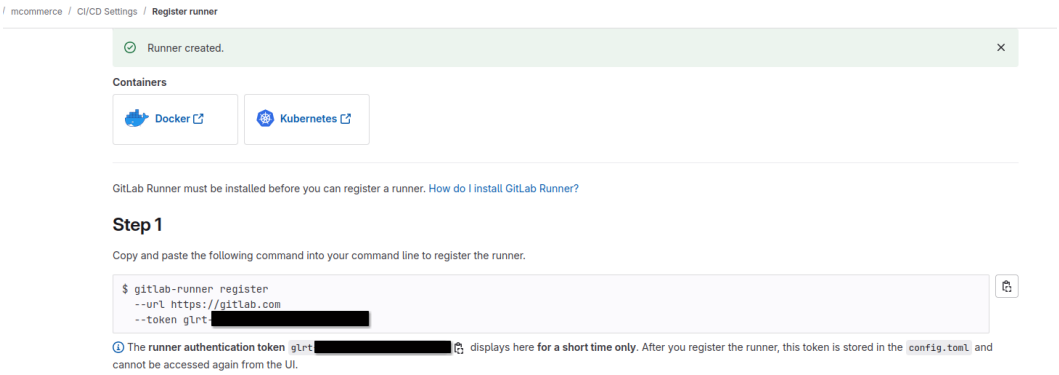


Figure 5.7

After this, we start the gitlab runner container and as we can see the runner is working.

Figure 5.8: gitlab-runner container is running

Figure 5.9

Figure 5.10: The local gitlab-runner is connected to GitLab

5.1.2 Job rules

Job rules are utilized to control the execution of jobs based on predefined conditions. These rules allow for the specification of criteria that determine when a job should be executed or skipped, thereby facilitating a more efficient workflow.

By leveraging job rules, the behavior of CI/CD processes can be tailored to respond dynamically to various contextual factors, such as the specific branch being built or the presence of certain environment variables. This capability enhances the adaptability of the pipeline and optimizes resource utilization by ensuring that jobs are executed only when relevant.

In GitLab CI/CD, YAML job rules are defined using the 'rules' keyword within a job's configuration in the .gitlab-ci.yml file. The syntax allows for specifying conditions under which a job should run or be skipped. Here is a simple example:

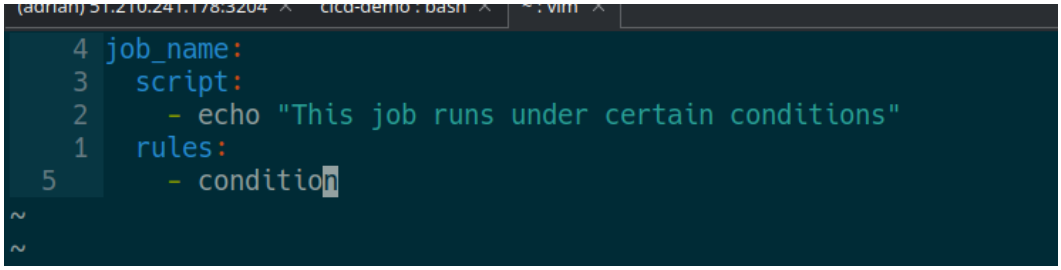


Figure 5.11

In our case, we defined several rules for several jobs across the pipeline:

```
maven-build:
  rules:
    - if: $CI_COMMIT_BRANCH == "master"
      changes:
        - src/**/*
  stage: build
  image: maven:3.8.7-openjdk-18-slim
  cache:
    key: "maven-dep"
    paths:
      - .m2/repository
  script:
    - chmod +x ./mvnw
    - ./mvnw clean package -Dmaven.repo.local=.m2/repository
  artifacts:
    paths:
      - ./target/*.jar
```

Figure 5.12

```
.docker-build:
  rules:
    - if: $CI_COMMIT_BRANCH == "master"
      changes:
        paths:
          - Dockerfile
          - src/**/*
  stage: container
  image: docker
  dependencies:
    - maven-build
  services:
    - docker:dind
  before_script:
    - docker login -u $DOCKER_LOGIN -p $DOCKER_TOKEN
  script:
    - docker version
    - docker build -t $DOCKER_LOGIN/$IMAGE_NAME:$TAG .
    - docker push $DOCKER_LOGIN/$IMAGE_NAME:$TAG
```

Figure 5.13

```
.gitops-k8s-deploy:
  variables:
    GIT_EMAIL: ""
    GIT_USERNAME: ""
    GITLAB_TOKEN_NAME: ""
    GITLAB_TOKEN: ""
    GITLAB_REPO_URL: ""
    MICROSERVICE_NAME: ""

  rules:
    - if: $CI_COMMIT_BRANCH == "master"
      changes:
        paths:
          - Dockerfile
          - src/**/*
  image: bitnami/git:latest
  stage: deploy-k8s
  before_script:
    - git config --global user.email $GIT_EMAIL
    - git config --global user.name $GIT_USERNAME
    - git clone --depth 1 "https://$GITLAB_TOKEN_NAME:$GITLAB_TOKEN@$GITLAB_REPO_URL"
  script:
    - cd $MICROSERVICE_NAME-microservice-kubernetes/$MICROSERVICE_NAME-k8s
    - sed -i "s/projdevopsiccn2.*$/projdevopsiccn2\/mcommerce:$TAG/g" $MICROSERVICE_NAME-deployment.yaml
    - cat $MICROSERVICE_NAME-deployment.yaml
    - git add $MICROSERVICE_NAME-deployment.yaml
    - git commit -m "microservice version $TAG"
    - git push -uf origin main
```

Figure 5.14

- **In figure 5.12**: The rule here signifies that we want to build the project only if changes happened to the master branch and these changes affect only the application source code inside the src directory. thus, we avoid unnecessary builds when other files than changes.
- **in figure 5.13**: The rule here signifies that we want to containerize the application only if changes happened to the 'Dockerfile' or the application source code in the master branch.
- **In figure 5.14**: The rule here signifies that we want to update our IaC repository only if there are changes to the way the application is containerized or there are changes to the source code of the application in the master branch.

5.1.3 Caching

Caching is defined as a mechanism by which frequently accessed data is stored temporarily to improve the performance of subsequent operations. By utilizing caching, the retrieval of data is expedited, thereby reducing the time and resources required for tasks such as job execution in CI/CD pipelines. Caching is implemented to minimize redundant data processing and to enhance overall efficiency within the development workflow.

In our case, caching makes sense only for the application dependencies. Thus, we modified the maven-build job to cache downloaded dependencies to be used for subsequent builds.

Figure 5.15

The modification includes adding a new entry 'cache' that declares that we want to cache the content of the directory '.m2/repository' that contains the downloaded dependencies. Also, we gave this cache a key name as a way to identify it. Note that we have changed the maven command (-Dmaven.repo.local) to specify to it to use the '.m2/repository' because it's the cached directory.

5.1.4 Generic jobs

In GitLab, the generic job include feature is defined as a mechanism that allows the inclusion of external YAML files into a CI/CD pipeline configuration. By utilizing this feature, shared configurations can be centralized, promoting consistency across multiple projects. The benefits of using job includes include improved maintainability, as updates can be made in one location and automatically reflected in all pipelines, and enhanced collaboration, as teams can share common job definitions easily. Additionally, the overall complexity of pipeline configurations is reduced, leading to clearer and more manageable

CI/CD setups.

In our case, we created a new directory '.gitlab' each file represents a generic job, That we will include in the main 'gitlab-ci.yml' file and apply changes specific to the current project, as shown in figure 5.18.

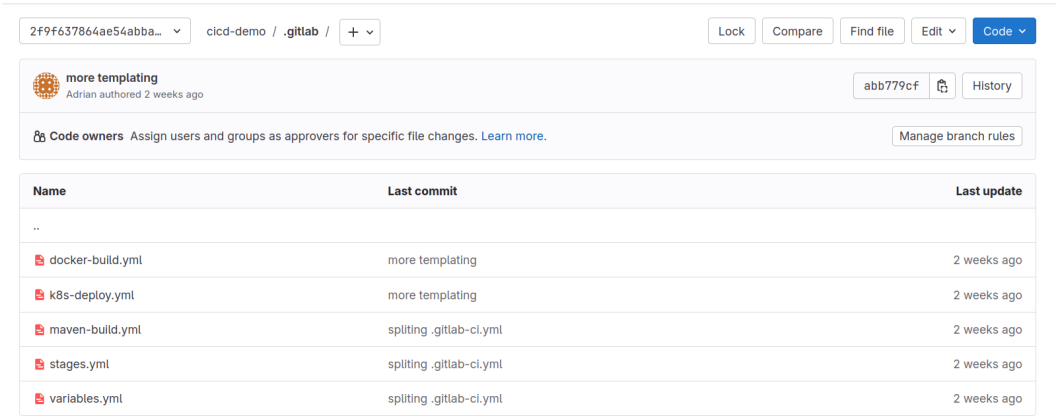


Figure 5.16: Different generic jobs

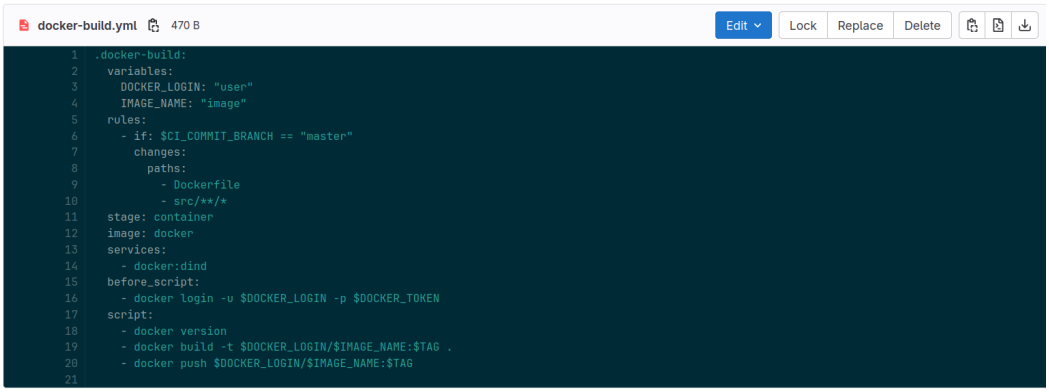


Figure 5.17: Example of a generic job with default variable that will be overridden

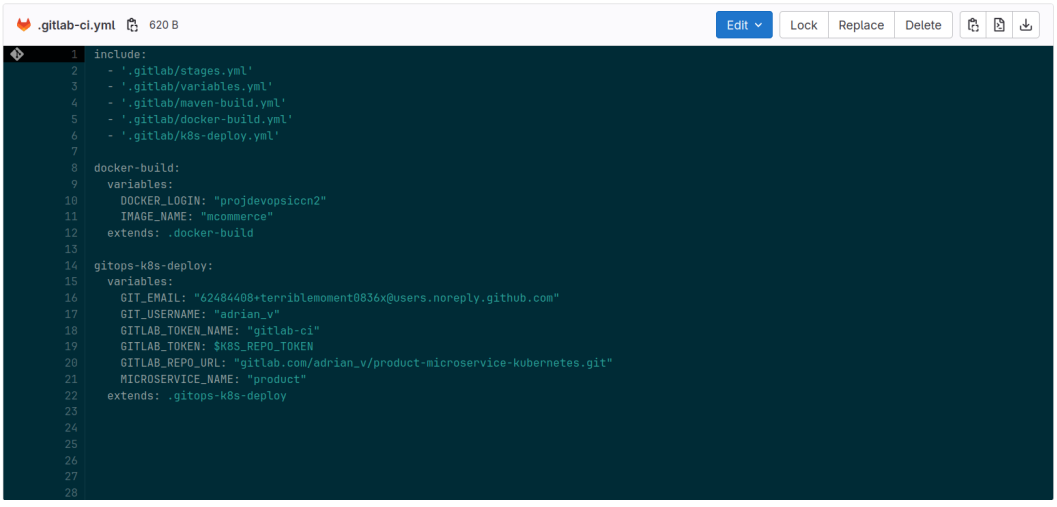


Figure 5.18: Main CI configuration where the included generic jobs are customized

5.2 gestion des groupes et permissions

5.2.1 Définir des rôles

The role definition section establishes two primary access levels within the project: maintainer and developer roles.

The maintainer role is granted elevated privileges necessary for project administration and security management, including code deployment, environment configuration, and security policy enforcement.

The developer role is configured with permissions appropriate for daily development tasks while maintaining security boundaries.

```yaml
# Role definitions
roles:
  maintainer_role:
    permissions:
      - push_code
      - merge_code
      - manage_project_settings
      - manage_members
      - create_protected_tags
      - delete_protected_tags
      - manage_pipelines
      - manage_environments
      - manage_deployments
      - approve_merge_requests
      - override_project_settings
      - manage_security_policies

  developer_role:
    permissions:
      - push_code
      - create_merge_requests
      - create_branches
      - run_pipelines
      - create_issues
      - manage_issues
      - approve_merge_requests
      - deploy_to_staging

```

Figure 5.19: Defining Roles


```yaml
# Role definitions
roles:
  maintainer_role:
    permissions:
      - push_code
      - merge_code
      - manage_project_settings
```

```
      - manage_members
      - create_protected_tags
      - delete_protected_tags
      - manage_pipelines
      - manage_environments
      - manage_deployments
      - approve_merge_requests
      - override_project_settings
      - manage_security_policies

  developer_role:
    permissions:
      - push_code
      - create_merge_requests
      - create_branches
      - run_pipelines
      - create_issues
      - manage_issues
      - approve_merge_requests
      - deploy_to_staging
```

5.2.2 Group definitions with roles

Group definitions implement the practical application of roles through structured team organization. This configuration creates two distinct groups: project-maintainers and project-developers. Each group is assigned specific visibility settings and access levels, ensuring proper segregation of duties and access control. The private visibility setting ensures that group membership and activities remain restricted to authorized personnel.

Figure 5.20: Groups

```
groups:
  - name: project-maintainers
    path: project-maintainers
    description: "Equipe de maintenance avec acces complet au projet"
    visibility: private
    access_level: maintainer
    role: maintainer_role
    members:
      - username: maintainer1
        access_level: maintainer

  - name: project-developers
    path: project-developers
    description: "Equipe de developpement"
    visibility: private
    access_level: developer
    role: developer_role
    members:
      - username: developer1
        access_level: developer
```

5.2.3 Project access configuration

5.2.4 Protection des branches

Branch protection mechanisms safeguard critical code bases by implementing hierarchical access controls. The main branch receives the highest level of protection, requiring maintainer approval for all modifications.

The develop branch allows developer access while maintaining maintainer oversight for critical operations.

```
43  # Project access configuration
44  project:
45    # Protection des branches
46    protected_branches:
47      - name: main
48        push_access_level: maintainer
49        merge_access_level: maintainer
50        unprotect_access_level: maintainer
51
52      - name: develop
53        push_access_level: developer
54        merge_access_level: developer
55        unprotect_access_level: maintainer
```

Figure 5.21: protection des branches

```
project:
  # Protection des branches
  protected_branches:
    - name: main
      push_access_level: maintainer
      merge_access_level: maintainer
      unprotect_access_level: maintainer

    - name: develop
      push_access_level: developer
      merge_access_level: developer
      unprotect_access_level: maintainer
```

5.2.5 Configuration des merge requests

The merge request configuration implements a code review process that ensures quality control while maintaining development efficiency. A single approval is required from either maintainers or developers, promoting collaborative code review while preventing deployment bottlenecks.

```
44  project:
57    # Configuration des merge requests
58    merge_requests:
59      approval_rules:
60        - name: "Default"
61          approvals_required: 1
62          user_groups:
63            - project-maintainers
64            - project-developers
65
```

Figure 5.22: merge request

```
# Configuration des merge requests
merge_requests:
  approval_rules:
    - name: "Default"
      approvals_required: 1
      user_groups:
        - project-maintainers
        - project-developers
```

5.2.6 CI/CD

The CI/CD configuration establishes a secure deployment pipeline by implementing environment-specific access controls. Production deployments are restricted to maintainers only, ensuring maximum security for critical infrastructure. Staging environment access is granted to developers, facilitating testing and validation while maintaining the security boundary between development and production environments.

Figure 5.23: CI/CD parameter

Paramètres CI/CD

```
ci_cd:
  protected_environments:
    - name: production
      deploy_access_levels:
        - group_id: project-maintainers
          access_level: maintainer


    - name: staging
```

```
deploy_access_levels:
  - group_id: project-developers
    access_level: developer
```

5.2.7 Configuration des membres

Member configuration implements the practical assignment of users to their respective groups and access levels. This section defines specific user-role associations, ensuring that each team member has appropriate access permissions for their responsibilities.

```
44  project:
78
79  # Configuration des membres
80  members:
81    - group: project-maintainers
82      users:
83        - username: soufian-elouazzani1
84          access_level: maintainer
85
86    - group: project-developers
87      users:
88        - username: adrian_v
89          access_level: developers
```

Figure 5.24: members

```
# Configuration des membres
members:
  - group: project-maintainers
    users:
      - username: soufian-elouazzani1
        access_level: maintainer

  - group: project-developers
    users:
      - username: adrian_v
        access_level: developer
```

finally

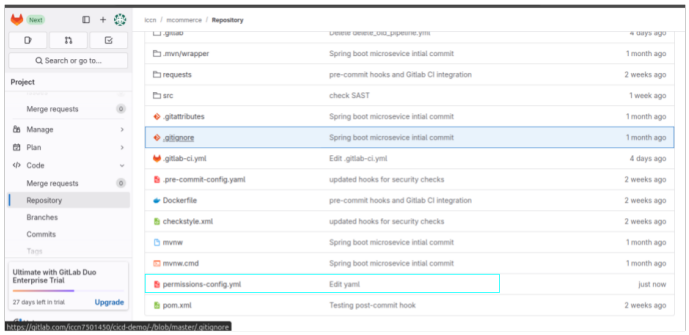


Figure 5.25: push the file to the root folder of the project

5.3 Protected Branches

In GitLab, permissions are fundamentally defined around the idea of having read or write permission to the repository and branches. To impose further restrictions on certain branches, they can be protected.

5.3.1 Protected Branch Controls

A protected branch controls:

- Which users can merge into the branch.
- Which users can push to the branch.
- If users can force push to the branch.

5.3.2 Who Can Modify a Protected Branch

When a branch is protected, the default behavior enforces these restrictions on the branch.

Action	Who can do it
Protect a branch	At least the Maintainer role.
Push to the branch	Anyone with **Allowed** permission. (1)
Force push to the branch	No one.
Delete the branch	No one. (2)

5.3.3 Steps to Protect a Branch for All Projects in a Group

a) On the left sidebar, select **Search** or go to and find your group.

b) Select **Settings > Repository**.

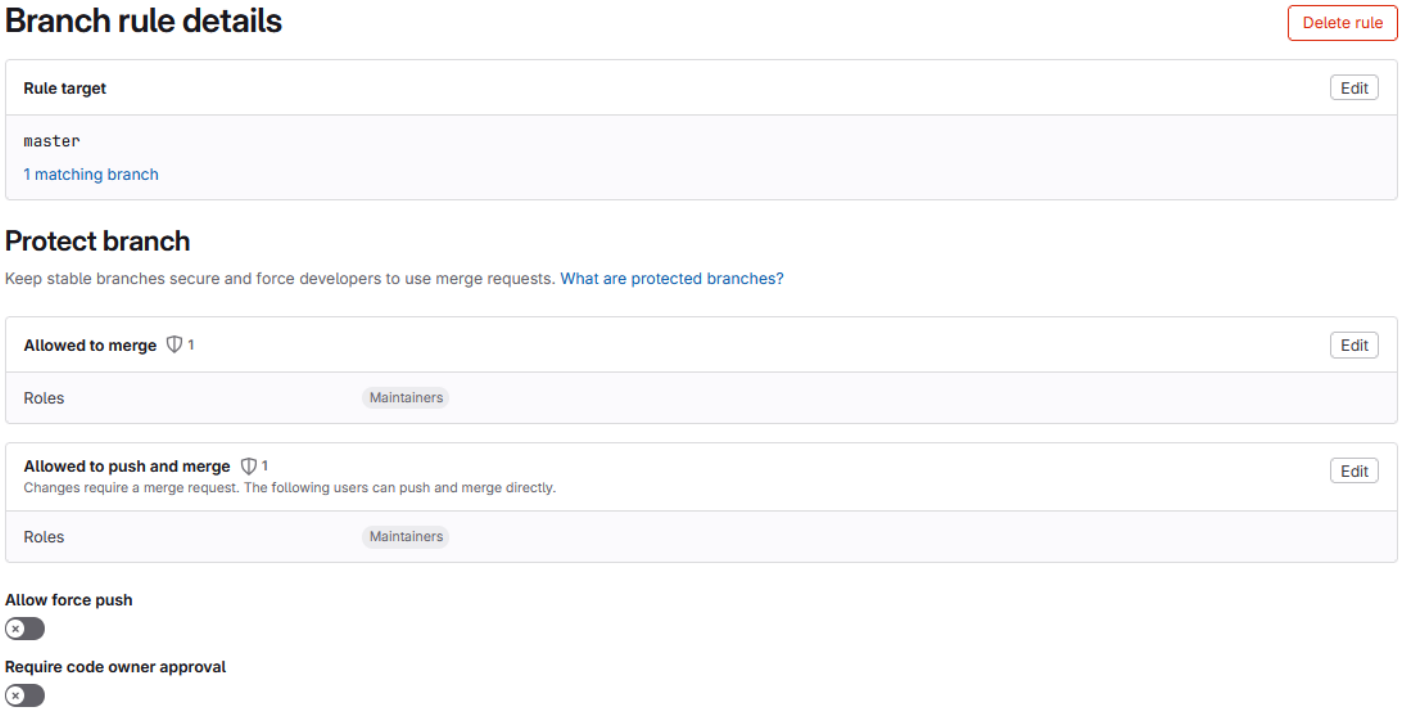
c) Expand **Protected branches**.

d) Select **Add protected branch**.

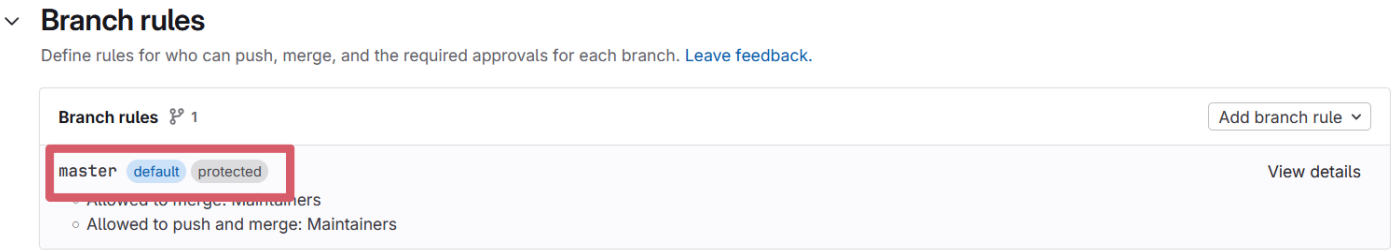
e) From the **Allowed to merge** list, select a role that can merge into this branch.

f) From the **Allowed to push and merge** list, select a role that can push to this branch.

g) Select **Protect**.



The tag *„protected"* is put on all protected branches.



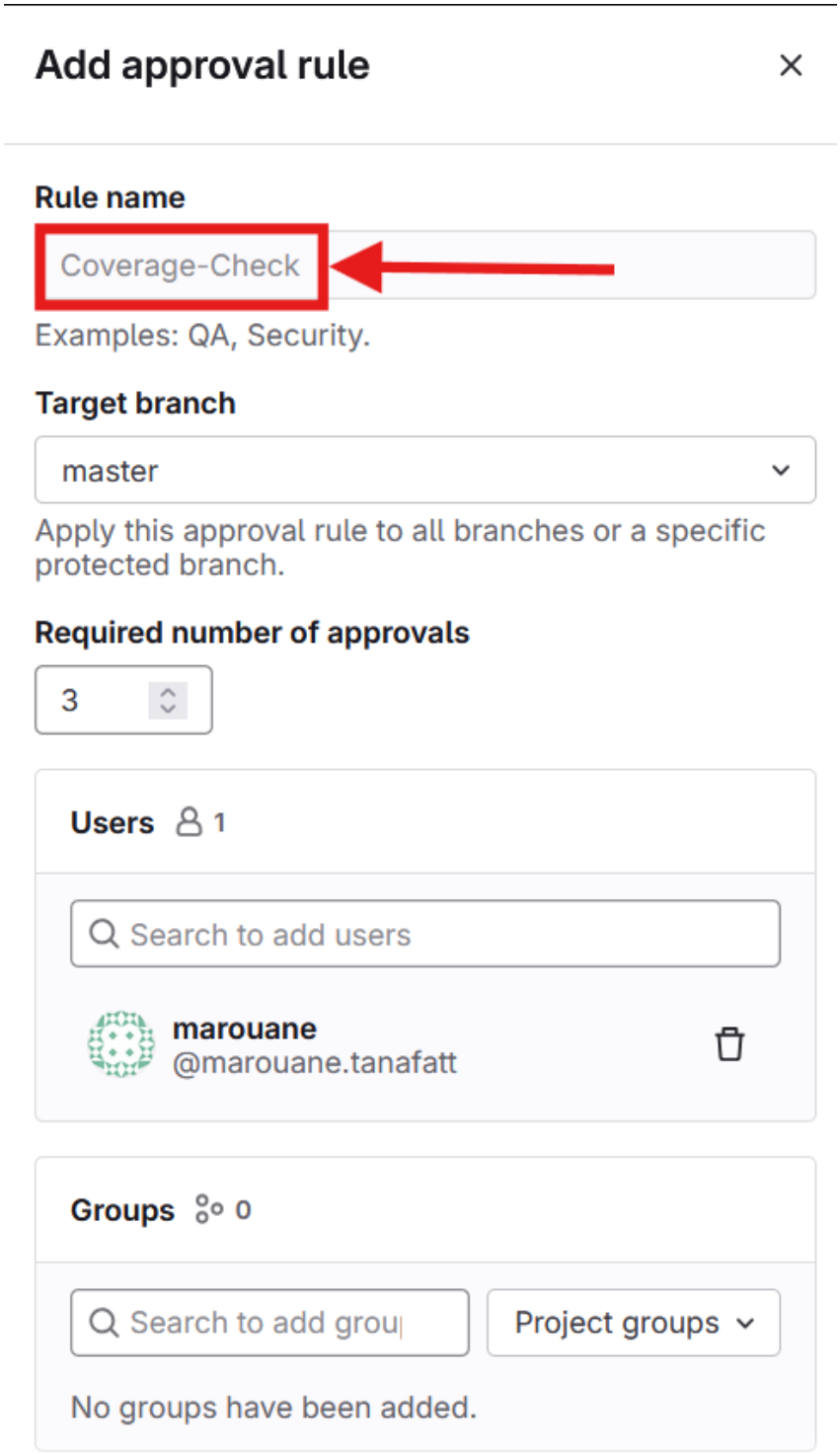
5.4 Add a Coverage Check Approval Rule

5.4.1 Steps to Add a Coverage-Check Approval Rule

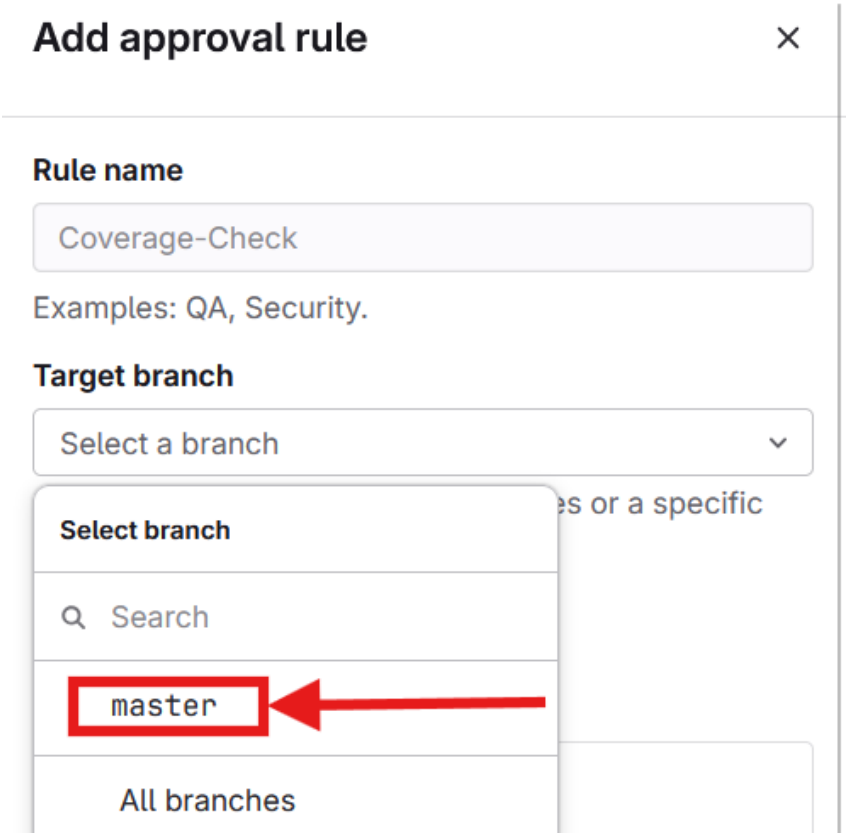
a) Go to your project and select **Settings** > **Merge requests**.

b) Under **Merge request approvals**, do one of the following:

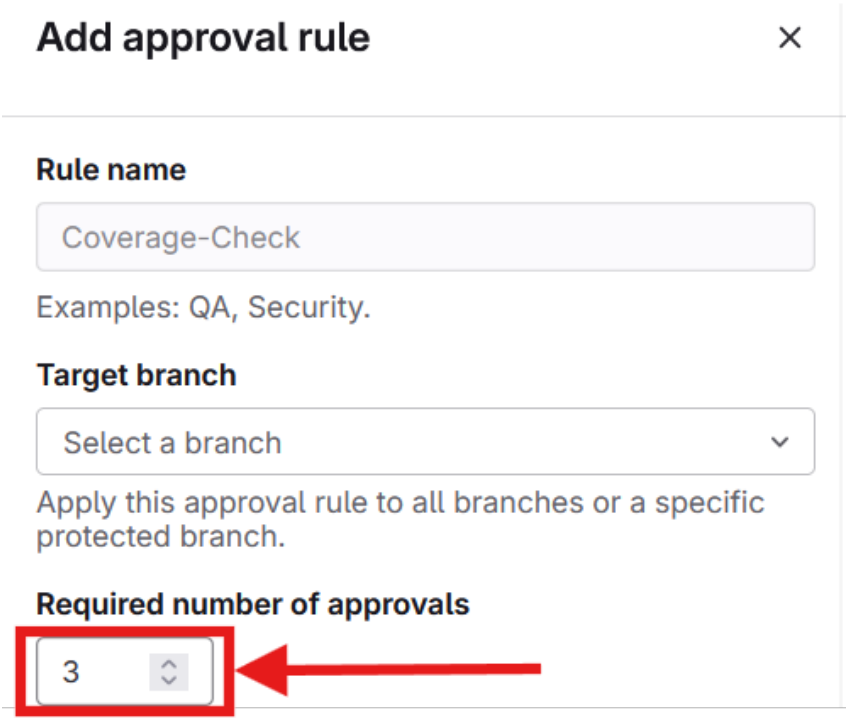
- Next to the **Coverage-Check approval rule**, select **Enable**.
- For manual setup, select **Add approval rule**, then enter the **Rule name**. For example: *Coverage Check*.



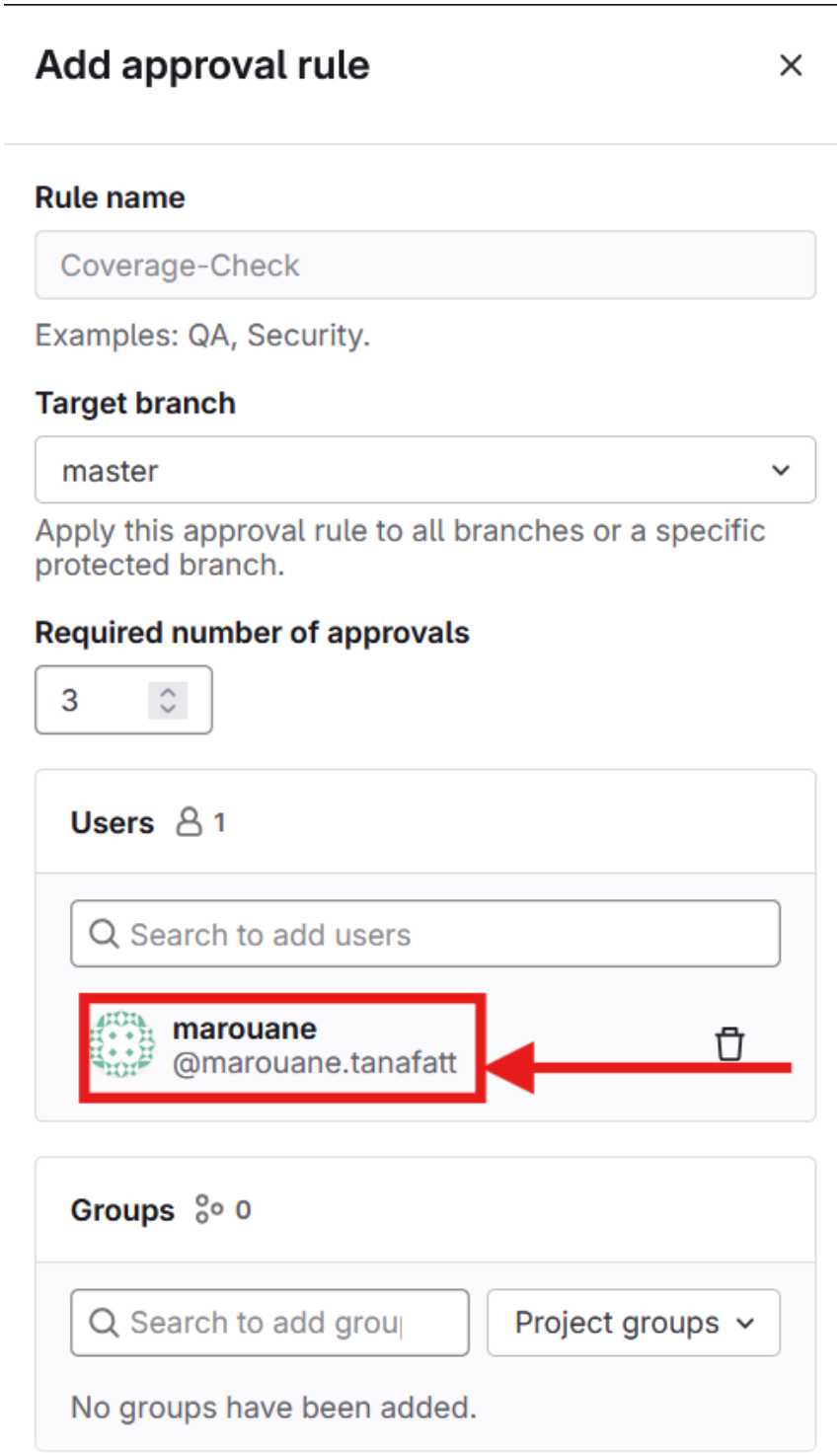
c) Select a **Target branch**.



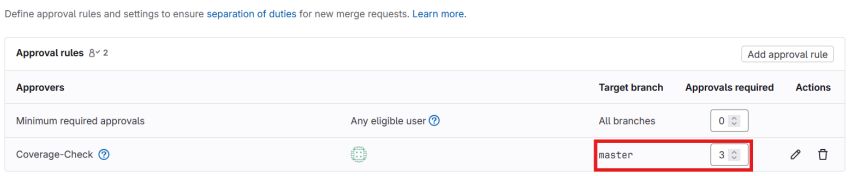
d) Set the **Required number of approvals**.



e) Select the **Users or Groups** to provide approval.



f) Select **Save changes**.



5.5 Pre-Commit Hooks

5.5.1 Integration of Pre-Commit Hooks in GitLab

In this section, we address the issue raised in question 5-a):

> *« In the case where a developer disables pre-commit hooks locally, integrate a step in the pipeline to handle local commits without active pre-commit hooks. »*

To address this, we set up two files:

- `.gitlab/pre-commit.yml` (or `pre-commit.yml`)
- `.pre-commit-config.yaml`

Main Objective: To enforce quality checks (such as Checkstyle, Spotless, dependency analysis, etc.) in the GitLab pipeline, even if a developer has disabled their local *pre-commit* hook.

5.5.2 Structure of the `pre-commit.yml` File in GitLab

The `pre-commit.yml` file (included later in `.gitlab-ci.yml`) defines a specific job that uses the `pre-commit` tool within the pipeline. Here is an example configuration snippet:

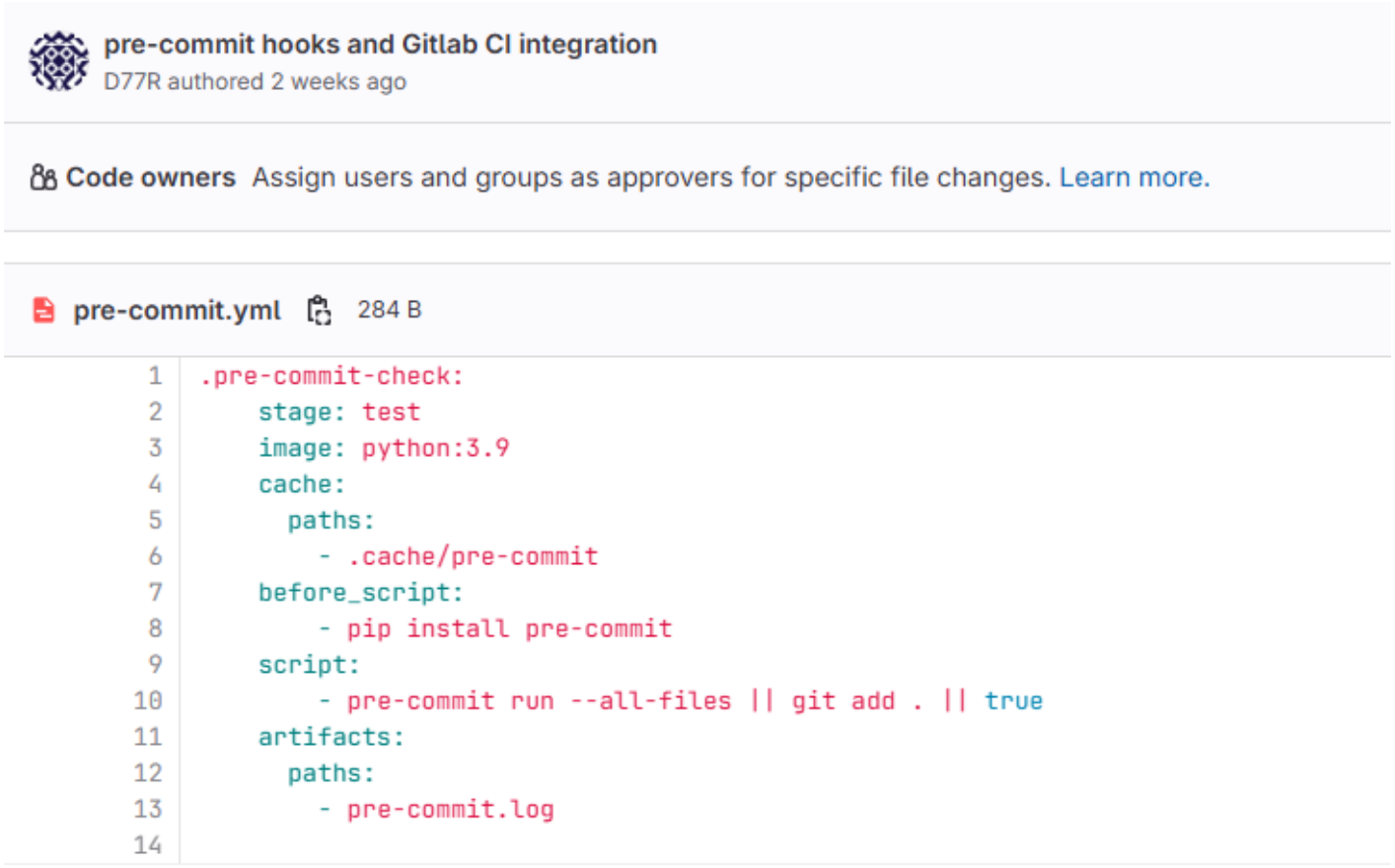


Figure 5.26: Excerpt from the .gitlab/pre-commit.yml file

Explanation:

- `stage: test` : The job is associated with the testing phase of the pipeline.
- `image: python:3.9` : We use a Docker image containing Python to install and run `pre-commit`.
- `before_script` : Installation of `pre-commit` via `pip`.
- `script` : Execution of `pre-commit` hooks on all files (`-all-files`). *Even if a developer disables the hook locally, this job ensures the same checks are run in GitLab.*
- `artifacts` : The `pre-commit.log` file is retained for later review.

5.5.3 Configuration of `.pre-commit-config.yaml`

The `.pre-commit-config.yaml` file specifies the list of hooks to execute. Below is an example:

```yaml
repos:
    - repo: https://github.com/pre-commit/pre-commit-hooks
      rev: v4.4.0
      hooks:
        - id: trailing-whitespace
        - id: end-of-file-fixer
        - id: check-yaml
        - id: detect-private-key #ICCN o kda o hadi o mandiruch security 7chuma

    - repo: local
      hooks:
        - id: checkstyle
          name: Run Checkstyle
          entry: mvn checkstyle:check
          language: system
          always_run: true
        - id: spotless
          name: Format Java Code with Spotless
          entry: mvn spotless:apply
          language: system
          always_run: true
        - id: dependency-check
          name: Run OWASP Dependency Check
          entry: mvn org.owasp:dependency-check-maven:check
          language: system
          always_run: tru
~
```

Figure 5.27: Excerpt from the .pre-commit-config.yaml file

Notes:

- **The official repository** (`https://github.com/pre-commit/pre-commit-hooks`), which provides common hooks such as:
  - `trailing-whitespace` (removal of unnecessary spaces),
  - `end-of-file-fixer` (ensuring newline at the end of files),
  - `check-yaml` (validation of YAML syntax),
  - `detect-private-key` (detection of potentially exposed private keys).
- **The local repository** (`repo: local`), which contains custom hooks based on Maven commands (`language: system`), including:
  - `checkstyle` (Java code analysis via `mvn checkstyle:check`),
  - `spotless` (automatic formatting with `mvn spotless:apply`),
  - `dependency-check` (vulnerability detection with `mvn org.owasp:dependency-check-maven:check`).

For each hook, the `always_run: true` option forces systematic execution on every commit, ensuring a consistent level of quality and security, even if the relevant files have not been modified. Overall, this

configuration guarantees code style compliance, adherence to security best practices, and continuous dependency control to prevent vulnerabilities.

5.5.4 Inclusion in `.gitlab-ci.yml` and Resolution of the Issue

To ensure this job runs *even if the developer has disabled their local pre-commit* hook, we include `pre-commit.yml` in `.gitlab-ci.yml`:

.gitlab-ci.yml 1.10 KiB

```yaml
image : docker:latest
services :
  - docker:dind

include:
  - '.gitlab/pre-commit.yml'
  - '.gitlab/stages.yml'
  - template: Jobs/SAST.gitlab-ci.yml
  - template: Jobs/Dependency-Scanning.gitlab-ci.yml
  - '.gitlab/variables.yml'
  - '.gitlab/maven-build.yml'
  - '.gitlab/docker-build.yml'
  - '.gitlab/k8s-deploy.yml'
  - '.gitlab/delete_old_pipeline.yml'


variables:
  GITLAB_ADVANCED_SAST_ENABLED: 'true'


pre-commit-check:
  extends: .pre-commit-check

```

.gitlab/stages.yml

```
stages:
+   - test
    - build
    - container
    - deploy-k8s
```

Figure 5.28: Excerpt from the .gitlab-ci.yml file

- `include: '- '.gitlab/pre-commit.yml'` : Imports the dedicated configuration for `pre-commit` hooks, defined in a separate file.
- `pre-commit-check: extends: .pre-commit-check` : Creates a `pre-commit-check` job in the pipeline, inheriting the instructions contained in `.pre-commit-check`. This ensures the execution

of `pre-commit` checks on GitLab's side, even if a developer has disabled their hooks locally.

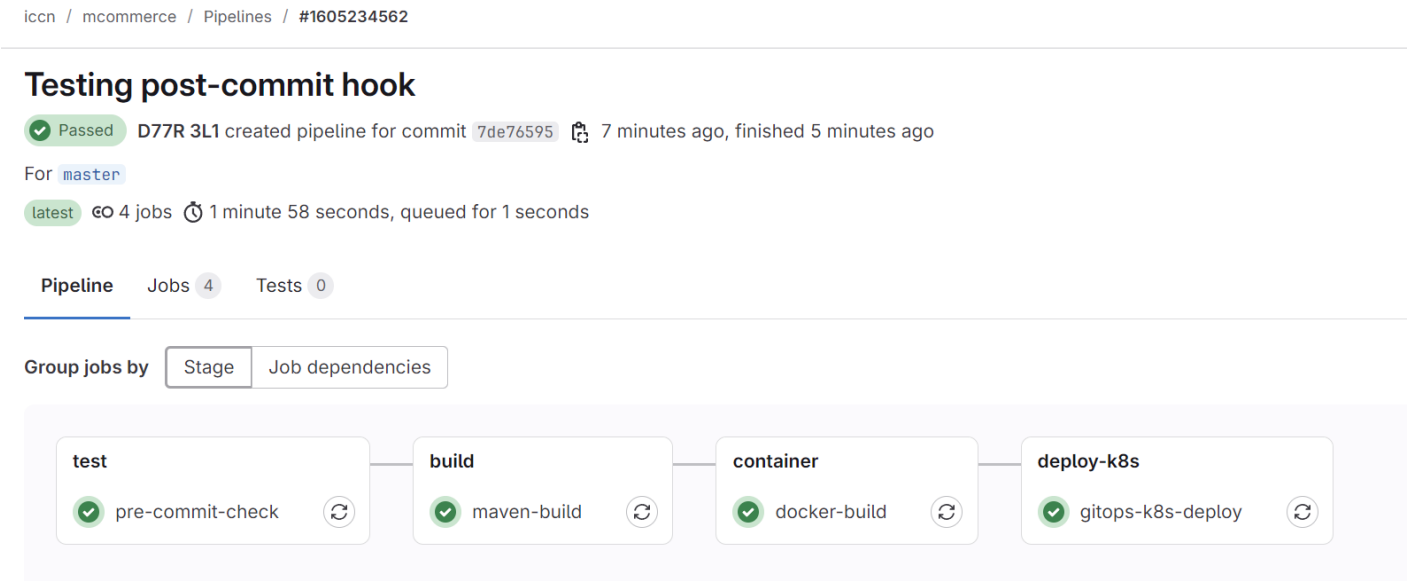


Figure 5.29: GitLab pipeline validated after execution of `pre-commit-check`

Although developers can disable `pre-commit` hooks locally, the `pre-commit-check` job (figure 5.29, *test* phase) enforces the same checks on GitLab. Thus, even if the hook is disabled locally, the code must pass style and security checks for the pipeline to succeed. If issues are detected (trailing spaces, vulnerabilities, etc.), the pipeline fails and blocks merging or deployment. In the example above, all four jobs, including `pre-commit-check`, completed successfully, proving that the code passed these mandatory checks.

How Does This Solve the Problem?

1. **Developer disables the hook locally**: If, for any reason, the developer removes or disables `pre-commit` on their machine, they can push their code without the checks being performed locally.

2. **Mandatory execution in CI**: *However*, when creating a `merge request` or pushing, GitLab triggers the pipeline where the `pre-commit-check` job is defined.

3. **Validation or blocking**: If the code violates the rules defined in `.pre-commit-config.yaml`, the job fails, and the pipeline is blocked. The developer is then required to fix these issues (or justify any potential workaround) before merging their code.

Thus, the setup of the two files `pre-commit.yml` and `.pre-commit-config.yaml`, coupled with their execution in the GitLab pipeline (`.gitlab-ci.yml`), ensures compliance with the same quality rules, even if local hooks are disabled.

5.5.5 Evaluation of Using Frameworks like `pre-commit` (Tests for *pre-push*, *post-commit*)

After configuring *pre-commit* hooks, it is often relevant to extend `pre-commit` to include other stages of the Git lifecycle, such as *pre-push* and *post-commit*.

- ***pre-push***: Triggered before pushing code. Allows rerunning essential checks (unit tests, security analysis) to block non-compliant `push`es.
- ***post-commit***: Executed immediately after creating a commit. Ideal for running internal scripts (e.g., updating documentation) without blocking the commit phase.

```
D77R@vps-4f4e3327:~/cicd-demo/.git/hooks$ ls
applypatch-msg.sample      pre-commit              pre-receive.sample
commit-msg.sample          pre-commit.sample       prepare-commit-msg.sample
fsmonitor-watchman.sample  pre-merge-commit.sample push-to-checkout.sample
post-commit                pre-push                update.sample
post-update.sample         pre-push.sample
pre-applypatch.sample      pre-rebase.sample
D77R@vps-4f4e3327:~/cicd-demo$ pre-commit run --all-files
trim trailing whitespace.....................................................Passed
fix end of files.............................................................Passed
check yaml...................................................................Passed
detect private key...........................................................Passed
Run Checkstyle...............................................................Passed
Format Java Code with Spotless...............................................Passed
Run OWASP Dependency Check...................................................Passed
```

Figure 5.30: `pre-commit run -all-files` executing various checks (formatting, checkstyle, etc.)

Benefits of `pre-commit` for *pre-push* or *post-commit*:

- **Standardize checks**: Reuse the same logic for checks (linting, tests, etc.) during `push` or immediately after `commit`.
- **Reduce risks of broken commits**: Prevent untested code from being pushed to the remote branch.

- **Improve feedback**: If *pre-push* hooks detect an issue, the developer is notified before polluting the repository.

```
D77R@vps-4f4e3327:~/cicd-demo$ vim .gitlab-ci.yml
D77R@vps-4f4e3327:~/cicd-demo$ vim .pre-commit-config.yaml
D77R@vps-4f4e3327:~/cicd-demo$ git add .pre-* .gitlab-ci.yaml
fatal: pathspec '.gitlab-ci.yaml' did not match any files
D77R@vps-4f4e3327:~/cicd-demo$ git add .pre-* .gitlab-ci.yml
D77R@vps-4f4e3327:~/cicd-demo$ git commit -m "pre-commit hooks and Gitlab CI integration"
[WARNING] Unstaged files detected.
[INFO] Stashing unstaged files to /home/D77R/.cache/pre-commit/patch1735407789-3517086.
trim trailing whitespace.....................................................Passed
fix end of files.............................................................Passed
check yaml...................................................................Passed
black.....................................................(no files to check)Skipped
[INFO] Restored changes from /home/D77R/.cache/pre-commit/patch1735407789-3517086.
[master eeff7d8] pre-commit hooks and Gitlab CI integration
 2 files changed, 28 insertions(+), 4 deletions(-)
 create mode 100644 .pre-commit-config.yaml
D77R@vps-4f4e3327:~/cicd-demo$ git push origin master
Username for 'https://gitlab.com': D73L1
Password for 'https://D73L1@gitlab.com':
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 721 bytes | 721.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
To https://gitlab.com/iccn7501450/cicd-demo.git
   2f9f637..eeff7d8  master -> master
D77R@vps-4f4e3327:~/cicd-demo$
```

Figure 5.31: `git commit` command triggering `pre-commit` hooks, followed by `git push`

5.5.5.1 Installing a `pre-push` Hook via `pre-commit`

Listing 5.1: Installing a `pre-push` hook with pre-commit

From the project root

```
pre-commit install --hook-type pre-push
```

This command creates (or updates) the `.git/hooks/pre-push` file to call the same checks defined in your `pre-commit` configuration.

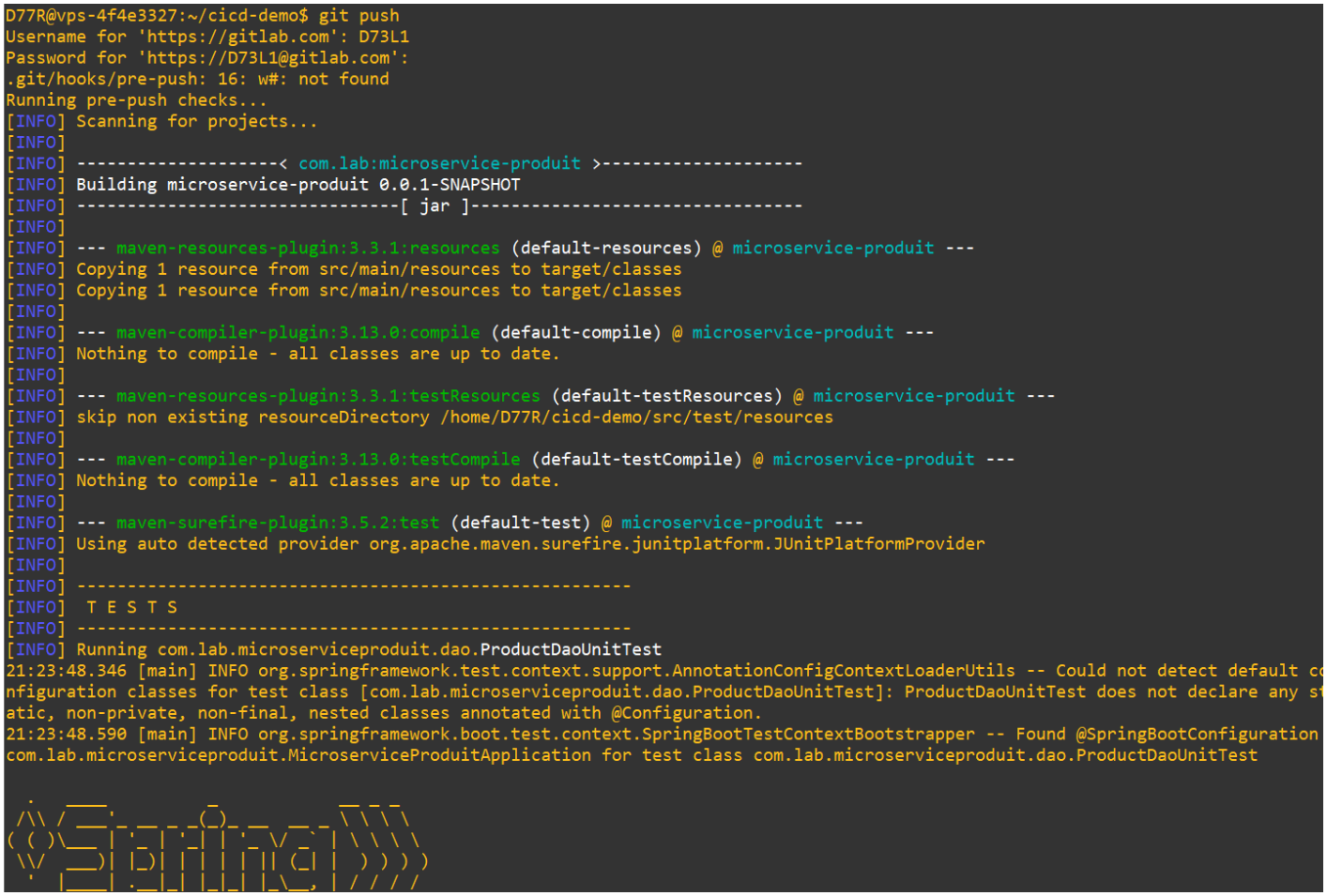


Figure 5.32: Example of Maven tests executed before `push`

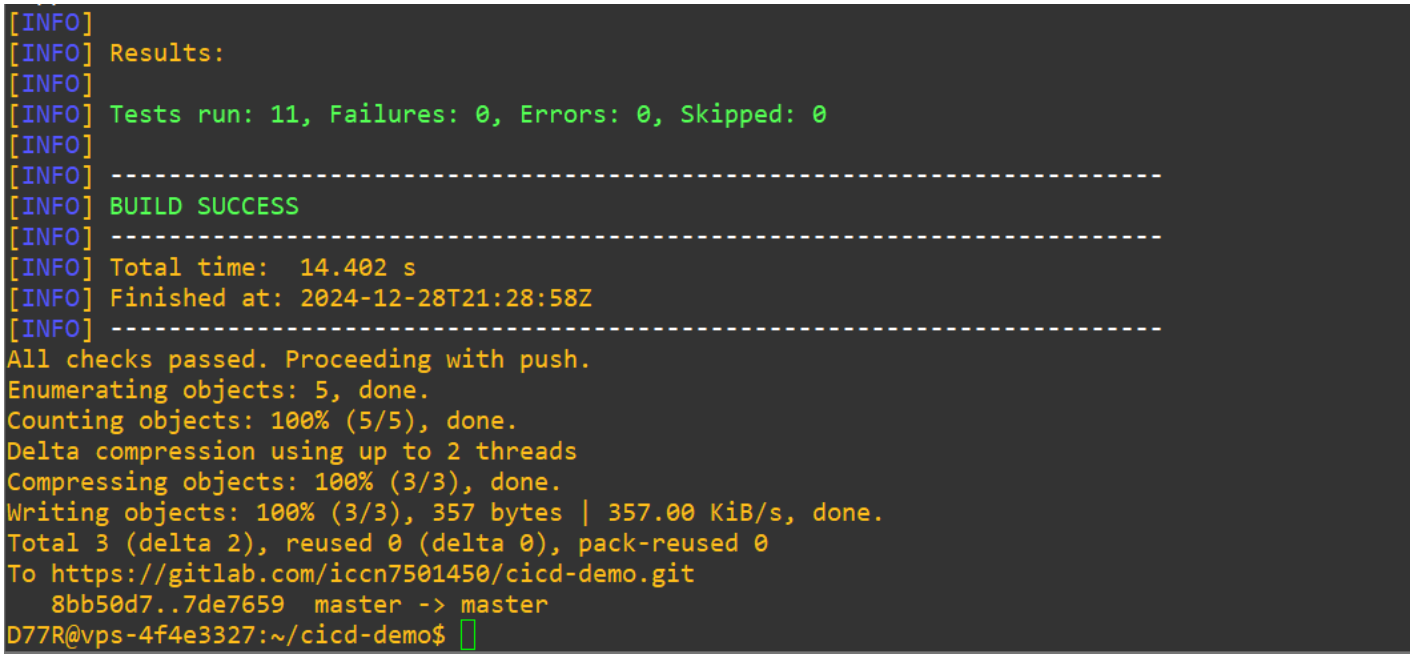


Figure 5.33: Output of the `pre-push` hook: 11 tests executed, no failures

5.5.6 Example Scripts for `post-commit` and `pre-push`

```
#!/bin/sh

LATEST_COMMIT=$(git log -1 --pretty=format:'%h - %s (%an)')
BRANCH_NAME=$(git rev-parse --abbrev-ref HEAD)

echo "Sending notification about the commit..."
curl -X POST -H "Content-Type: application/json" \
     -d '{"text":"New commit on branch *'"$BRANCH_NAME"'*: `'"$LATEST_COMMIT"'`"
}' \
     https://hooks.slack.com/services/your/slack/webhook/url

 TAG="tag-$(date +'%Y%m%d%H%M%S')"
 git tag $TAG
 echo "Tag $TAG created for commit $LATEST_COMMIT"

exit 0
```

Figure 5.34: Example `post-commit` script (notification and tagging)

This `post-commit` script notifies a Slack channel with information about the latest commit (`BRANCH_NAME`, `LATEST_COMMIT`) and automatically creates a `tag`.

```
# the standard input in the form:
remote="$1"
url="$2"

zero=$(git hash-object --stdin </dev/null | tr '[0-9a-f]' '0')

while read local_ref local_oid remote_ref remote_oid
do
    if test "$local_oid" = "$zero"
    then
        # Handle delete
        :
    else
        if test "$remote_oid" = "$zero"
        then
            # New branch, examine all commits
            range="$local_oid"
        else
            # Update to existing branch, examine new commits
            range="$remote_oid..$local_oid"
        fi

        # Example: Run tests using pytest before allowing the push
        echo "Running pre-push checks..."
        mvn test
        if [ $? -ne 0 ]; then
            echo "Tests failed. Push aborted."
            exit 1
        fi

        # Add any additional custom checks here
        # For example, check for specific patterns in commit messages
        commit=$(git rev-list -n 1 --grep 'FORBIDDEN_PATTERN' "$range")
        if test -n "$commit"
        then
            echo >&2 "Commit with forbidden pattern found in $local_ref. Not pus
hing."
            exit 1
        fi
    fi
done

echo "All checks passed. Proceeding with push."
exit 0
```

Figure 5.35: Example `pre-push` script (tests and checks)

This `pre-push` script runs `mvn test` before the `push` and can block the push if tests fail or if undesirable patterns (e.g., `FORBIDDEN_PATTERN`) are detected. If no issues are found, the `push` is allowed.

5.5.7 Strategies to Minimize False Positives

To mitigate false positives generated by automated hooks like `pre-commit`, it is advisable to adopt both technical and methodological approaches:

- **Refinement of validation rules**: Adapt the configurations of tools (Checkstyle, Spotless, OWASP Dependency-Check, etc.) to exclude irrelevant cases. For example, refining analysis rules to ignore certain directories, formats, or patterns known to trigger false alerts. Copy
- **Fine-tuning sensitivity thresholds**: Adjust detection thresholds in static and dynamic analysis tools so that only critical issues are flagged. This technical operation, based on code and behavior analysis, reduces unnecessary interruptions and focuses attention on what matters.
- **Advanced contextual filtering**: Integrate scripts or extensions capable of analyzing the context of changes (branch, author, nature of changes) to distinguish relevant alerts from background noise. This filtering helps target checks without generating false alerts for specific cases.
- **Feedback loop and continuous improvement**: Establish regular feedback with developers to evaluate the relevance of alerts. Feedback allows adjusting tool configurations and refining rules, leading to a gradual reduction in false positives and better alignment with project specifics.

By combining these strategies, the reliability of automated tools is optimized, ensuring they focus on genuine anomalies and enhance code quality and security without hindering the development cycle.

We demonstrate with a small example how to identify and resolve a false positive in Checkstyle by adjusting rules to fit a project's specific conventions.

5.5.8 Step 1: Creating a Java Class with a False Positive

```
public class test {
    private String _name;
    // Espace inutile
}
```

Figure 5.36: Java class `test.java` generating a false positive

In the `test` class, we intentionally introduce:

- A variable `_name`, which violates Checkstyle's default naming rules.
- An unnecessary space after a comment, likely to trigger a warning about unnecessary spaces.

5.5.9 Step 2: Configuring Checkstyle Rules

```
<?xml version="1.0"?>
<!DOCTYPE module PUBLIC
    "-//Checkstyle//DTD Checkstyle Configuration 1.3//EN"
    "https://checkstyle.org/dtds/configuration_1_3.dtd">

<module name="Checker">
    <!-- LineLength module -->
    <module name="LineLength">
        <property name="max" value="120"/>
        <property name="severity" value="warning"/>
    </module>

    <!-- FileTabCharacter module -->
    <module name="FileTabCharacter">
        <property name="severity" value="warning"/>
    </module>

    <module name="LocalVariableName">
        <property name="format" value="^[a-z][a-zA-Z0-9]*$"/>
    </module>
</module>
```

Figure 5.37: `checkstyle.xml` file with rule configuration

The `checkstyle.xml` file contains the following rules:

- **LineLength**: Limits line length to 120 characters.
- **FileTabCharacter**: Detects the use of tabs.
- **LocalVariableName**: Requires local variable names to follow the format `[a-z][a-zA-Z0-9]*`.

5.5.10 Step 3: Running Checkstyle on the Class

```
D77R@vps-4f4e3327:~/cicd-demo$ vi checkstyle.xml
D77R@vps-4f4e3327:~/cicd-demo$ checkstyle -c checkstyle.xml test.java
com.puppycrawl.tools.checkstyle.api.CheckstyleException: LocalVariableName is not allowed as a child in Checker
        at com.puppycrawl.tools.checkstyle.Checker.setupChild(Checker.java:502)
        at com.puppycrawl.tools.checkstyle.api.AutomaticBean.configure(AutomaticBean.java:201)
        at com.puppycrawl.tools.checkstyle.Main.runCheckstyle(Main.java:404)
        at com.puppycrawl.tools.checkstyle.Main.runCli(Main.java:331)
        at com.puppycrawl.tools.checkstyle.Main.execute(Main.java:190)
        at com.puppycrawl.tools.checkstyle.Main.main(Main.java:125)
Checkstyle ends with 1 errors.
```

Figure 5.38: Running Checkstyle and the error generated

The following command is used to check the class:

```
checkstyle -c checkstyle.xml test.java
```

Checkstyle generates an error related to the `LocalVariableName` rule because the variable `_name` violates the format defined in the configuration.

5.5.11 Step 4: Resolving the False Positive

To resolve the issue, we adjust the rule in `checkstyle.xml` to allow variable names starting with an underscore:

```
<module name="LocalVariableName">
<property name="format" value="^[_a-z][a-zA-Z0-9]*$"/>
</module>
```

5.5.12 Step 5: Validating the Changes

After adjusting the configuration, we rerun Checkstyle:

```
checkstyle -c checkstyle.xml test.java
```

The false positive related to `_name` is resolved, and no errors are reported.

```
D77R@vps-4f4e3327:~/cicd-demo$ vi test.java
D77R@vps-4f4e3327:~/cicd-demo$ checkstyle -c checkstyle.xml test.java
Starting audit...
Audit done.
```

Figure 5.39: Resolution of the False Positive

As a conclusion, suppose a project has:

- A strict Checkstyle rule prohibiting variables starting with an underscore (_).
- Multiple teams allowing this convention as an accepted practice.

Problem: At every commit, the `pre-commit` hook generates unnecessary errors, disrupting developers' workflow.

Implemented Solution:

1. **Adjusting the Checkstyle rule**: Modifying the configuration to include `_name` as a valid format in the `LocalVariableName` rule.
2. **Adding targeted exclusions**: Excluding specific modules, such as test folders or prototypes, to avoid irrelevant alerts.
3. **Conditional validation**: Enabling strict validation only during *pre-push* hooks or on main branches (`main`, `develop`), reducing the impact on developers' working branches.

5.6 Vulnerability scanning

Vulnerability scanning in GitLab is a security feature that is integrated into the CI/CD pipeline to identify and assess potential security vulnerabilities in the codebase and its dependencies. By utilizing automated scanning tools, GitLab can analyze the project for known vulnerabilities in libraries, containers, and configurations during the build process. GitLab provides many types of vulnerability scanning, notably:

- **Static Application Security Testing (SAST):** This scanning method analyzes source code for security vulnerabilities without executing the application. It helps identify issues such as code injection and insecure coding practices early in the development process.
- **Software Composition Analysis (SCA):** SCA examines the project's dependencies and libraries for known vulnerabilities. It ensures that third-party components do not introduce security risks into the application.
- **Dynamic Application Security Testing (DAST):** DAST scans the running application to identify vulnerabilities that can be exploited during runtime, such as cross-site scripting (XSS) and SQL injection.

- **Container Scanning:** This type of scanning assesses container images for known vulnerabilities before deployment. It checks both the base images and any installed packages within the container for security issues.

5.6.1 Static Application Security Testing

Lucky for SAST, GitLab comes with pre-made templates that can be easily included in our CI/CD pipeline as shown in figure 5.47. Note this template contains a job that's part of a stage named test, thus we also added a stage test as the first stage in the file 'stages.yml'

Figure 5.40

The scan results are saved in an artifact that we can download.

5.6.2 Software Composition Analysis

Similarly to enabling SAST, GitLab provides a template that contains a job that does software composition analysis. And we only need to include it.

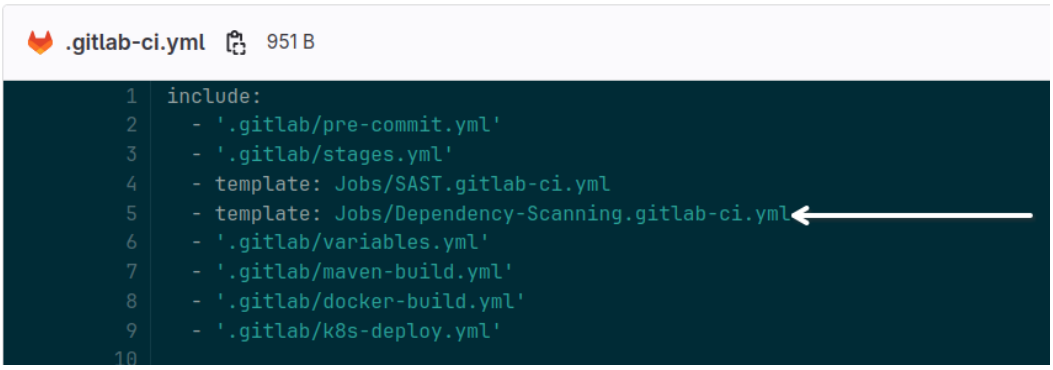


Figure 5.41

The scan results are saved in an artifact that we can download (figure 5.42). Or in what is coming next we can visualize the discovered vulnerabilities in GitLab Security Dashboard.

Figure 5.42

5.6.3 Container Security

The objective for Container Security is to scan the Docker image for vulnerabilities using **Docker Scout** and ensure no critical vulnerabilities are present before deployment.

Implementation Steps

1. **Define the `docker-scout` Job in `.gitlab-ci.yml`**: The `docker-scout` job is added to the CI/CD pipeline to scan the Docker image after it is built. The job performs the following steps:

- Install Docker Scout using `curl`.
- Log in to Docker Hub using the provided credentials.
- Scan the Docker image for vulnerabilities using Docker Scout.

```
docker-scout:
  stage: security_scan
  image: docker:latest
  services:
    - docker:dind
  variables:
    DOCKER_IMAGE: "${DOCKER_LOGIN}/${IMAGE_NAME}"
    DOCKER_HUB_USERNAME: "$DOCKER_LOGIN"
    DOCKER_HUB_TOKEN: "$DOCKER_TOKEN"
  script:
    - echo "Installing Docker Scout..."
    - apk add --no-cache curl tar file
    - echo "Logging in to Docker Hub..."
```

```
- echo "$DOCKER_HUB_TOKEN" | docker login -u "$DOCKER_HUB_USERNAME" --password-stdin
- echo "Scanning Docker image for vulnerabilities..."
- curl -fsSL https://github.com/docker/scout-cli/releases/download/v1.16.1/docker-scou
- tar -xzf docker-scout.tar.gz -C /usr/local/bin docker-scout
- chmod +x /usr/local/bin/docker-scout
- /usr/local/bin/docker-scout cves $DOCKER_IMAGE
```

```yaml
# container security
docker-scout:
  stage: container_scan
  image: docker:latest
  services:
    - docker:dind
  variables:
    DOCKER_IMAGE: "${DOCKER_LOGIN}/${IMAGE_NAME}"
    DOCKER_HUB_USERNAME: "$DOCKER_LOGIN"
    DOCKER_HUB_TOKEN: "$DOCKER_TOKEN"
  script:
    - echo "Installing Docker Scout..."
    - apk add --no-cache curl tar file  # Install curl, tar, and file
    - echo "Checking system architecture..."
    - uname -m
    - echo "Checking Docker version..."
    - docker --version
    - echo "Logging in to Docker Hub..."
    - echo "$DOCKER_HUB_TOKEN" | docker login -u "$DOCKER_HUB_USERNAME" --password-stdin
    - echo "Verifying Docker login..."
    - docker info
    - echo "Downloading Docker Scout version 1.16.1 for AMD64..."
    - curl -fsSL https://github.com/docker/scout-cli/releases/download/v1.16.1/docker-scout_1.16.1_linux_amd64.tar.gz -o docker-scout.tar.gz
    - echo "Verifying the downloaded file..."
    - file docker-scout.tar.gz
    - echo "Listing contents of the archive..."
    - tar -tf docker-scout.tar.gz
    - echo "Extracting the binary..."
    - tar -xzf docker-scout.tar.gz -C /usr/local/bin docker-scout
    - chmod +x /usr/local/bin/docker-scout
    - echo "Checking the binary..."
    - ls -l /usr/local/bin/docker-scout
    - echo "Testing Docker Scout binary..."
    - /usr/local/bin/docker-scout --help
    - echo "Testing Docker Scout with a simple command..."
    - /usr/local/bin/docker-scout cves hello-world
    - echo "Scanning Docker image for vulnerabilities..."
    - /usr/local/bin/docker-scout cves $DOCKER_IMAGE
  allow_failure: false
```

Figure 5.43: configuration for Docker Scout inside a container

2. **Results**: The Docker image is successfully scanned, and no critical vulnerabilities are detected. The pipeline continues without interruption, confirming the image meets security standards.

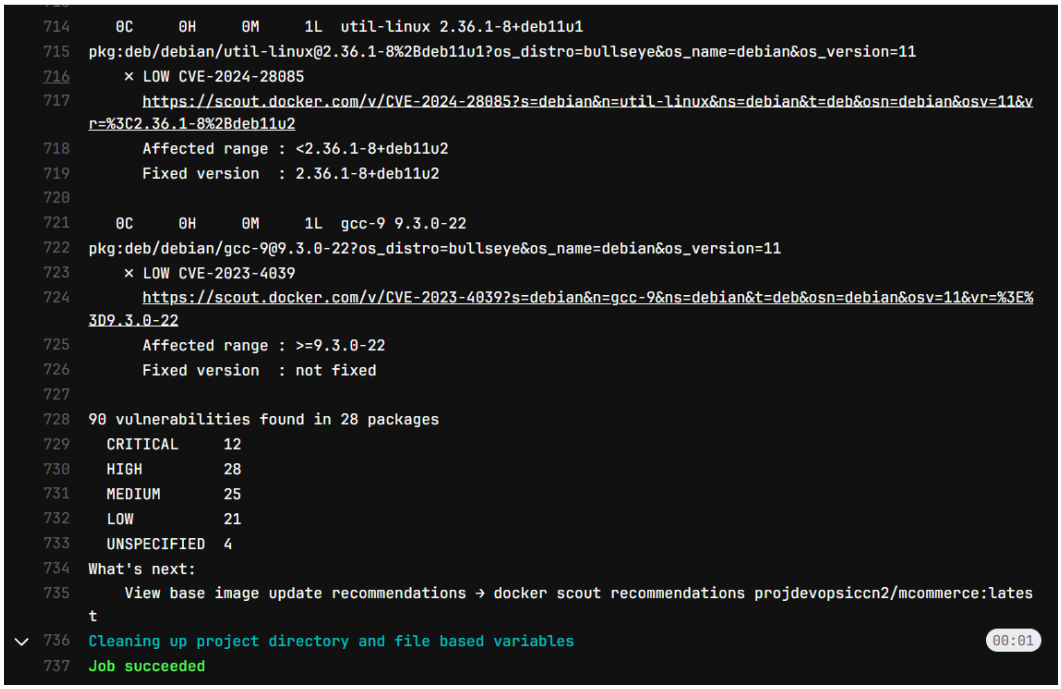


Figure 5.44: docker scout job results

5.6.4 Dynamic Application Security Testing

1. **Define the `dast` Job in `.gitlab-ci.yml`**: The `dast` job is added to the pipeline to scan the running application for vulnerabilities. The job performs the following steps:

- Deploy the application using Docker.
- Run the DAST scan on the application's endpoints.
- Save the scan results as artifacts for further analysis.

```
dast:
  stage: security_scan
  services:
    - name: ${DOCKER_LOGIN}/${IMAGE_NAME}
      alias: app
  variables:
    DAST_WEBSITE: "http://app:8080"
  script:
    - echo "Starting DAST scan..."
    - /analyze
```

```
artifacts:
  paths:
    - dast_artifacts/
```

```yaml
image: docker:latest
services:
  - docker:dind

include:
  - '.gitlab/pre-commit.yml'
  - '.gitlab/stages.yml'
  - template: Jobs/SAST.gitlab-ci.yml
  - template: Security/DAST.gitlab-ci.yml  # Correct DAST template
  - template: Jobs/Dependency-Scanning.gitlab-ci.yml
  - '.gitlab/variables.yml'
  - '.gitlab/maven-build.yml'
  - '.gitlab/docker-build.yml'
  - '.gitlab/k8s-deploy.yml'
  - '.gitlab/delete_old_pipeline.yml'

variables:
  GITLAB_ADVANCED_SAST_ENABLED: 'true'
  DOCKER_LOGIN: "projdevopsiccn2"
  IMAGE_NAME: "mcommerce"
  DAST_WEBSITE: "http://app:8080"  # URL for DAST scan

```

Figure 5.45: dast test configuration -1-

```yaml
# Override the DAST job stage
dast:
  stage: dast
  services:
    - name: ${DOCKER_LOGIN}/${IMAGE_NAME}  # Run the application as a service
      alias: app
  variables:
    DAST_WEBSITE: "http://app:8080"  # URL for DAST scan
  script:
    - echo "Starting DAST scan..."
    - /analyze
  artifacts:
    paths:
      - dast_artifacts/
  allow_failure: false
```

Figure 5.46: dast test configuration -2-

2. **Results**: The DAST scan is executed successfully, and no critical vulnerabilities are detected in the application. The pipeline continues without interruption, confirming the application meets security requirements.

Conclusion

All tests, including **SAST**, **DAST**, **Container Security**, and **Pre-Commit Hooks**, passed successfully. The Docker image is free from critical vulnerabilities, the application is secure against runtime threats, and the code meets quality standards. The Product Microservice is now ready for deployment.

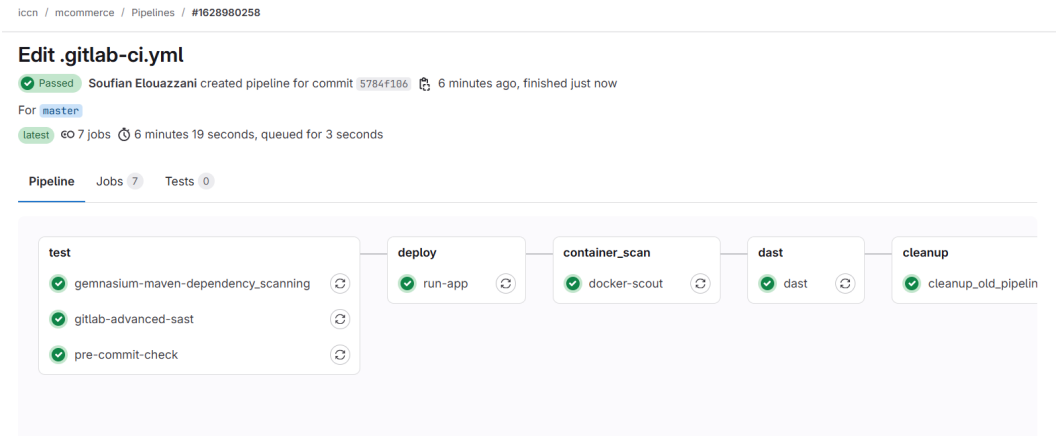


Figure 5.47: pipline for security testing gone Well

5.7 Vulnerability management and notification

To visualize the detected vulnerabilities in the different scans in our CI pipeline we need to use a vulnerability management platform. For our case we chose GitLab Security dashboard because it's already part of the tooling we have been using in this project and and easy to work with.

5.7.1 Vulnerability management

After enabling scanning such as DAST or SCA, scan result are automatically shown in the vulnerability report.

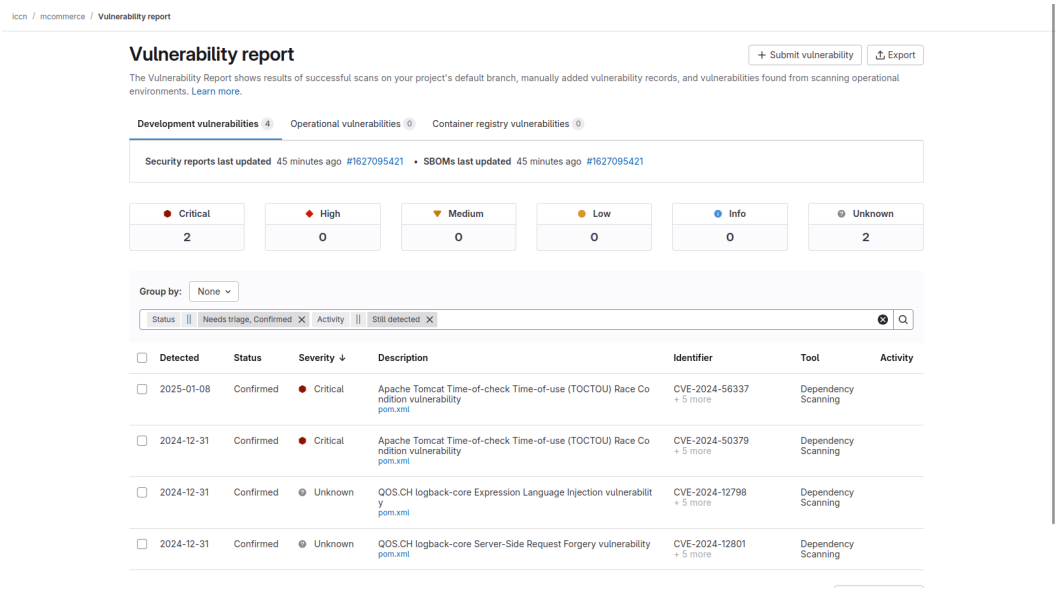


Figure 5.48

5.7.2 Configuring notification using Slack

GitLab support sending event notifications such as finding a new vulnerability to Slack. To configure this. First we will create a Slack Workspace and a channel for security notifications.

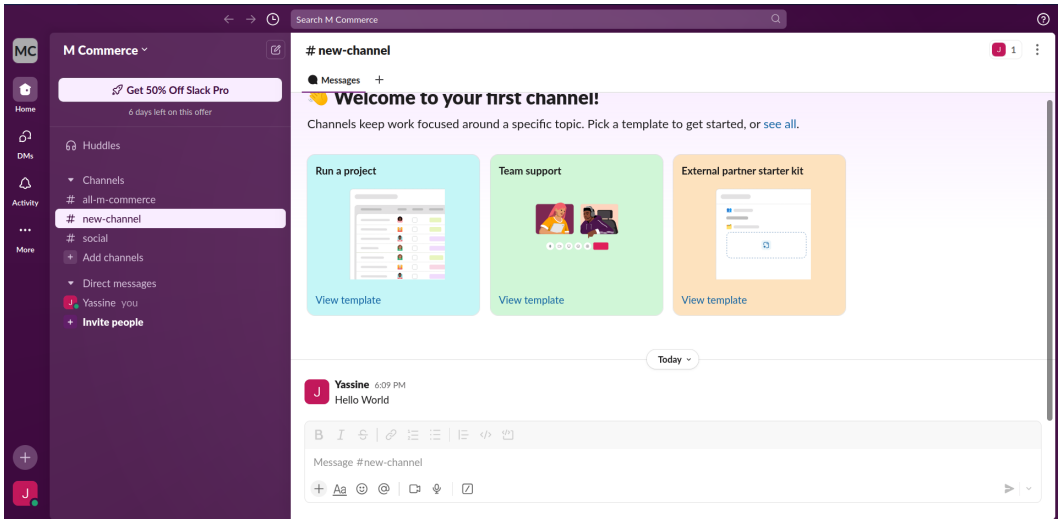


Figure 5.49: Creating a Slack workplace

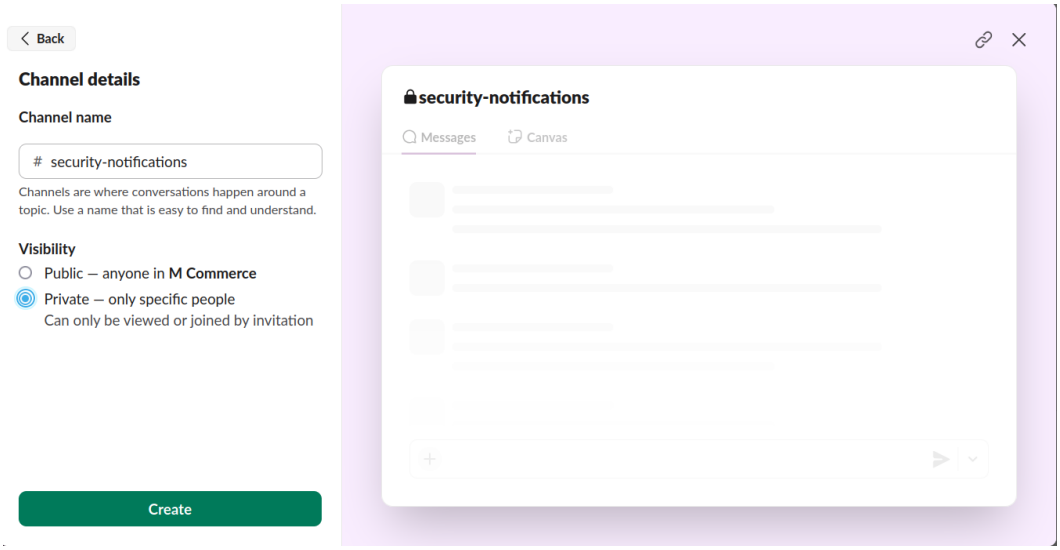


Figure 5.50: Creating a channel for vulnerabilities notifications

Then we connect Slack with our GitLab repository.

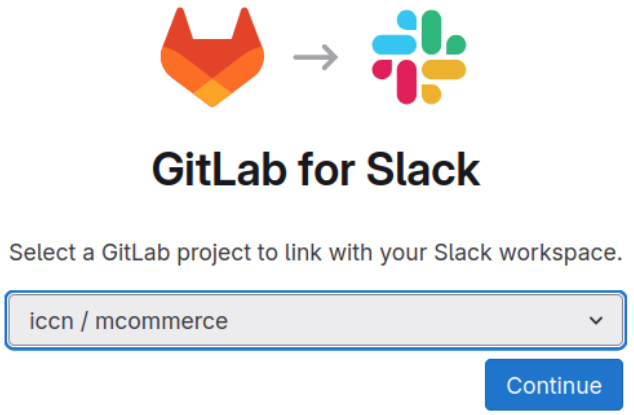


Figure 5.51

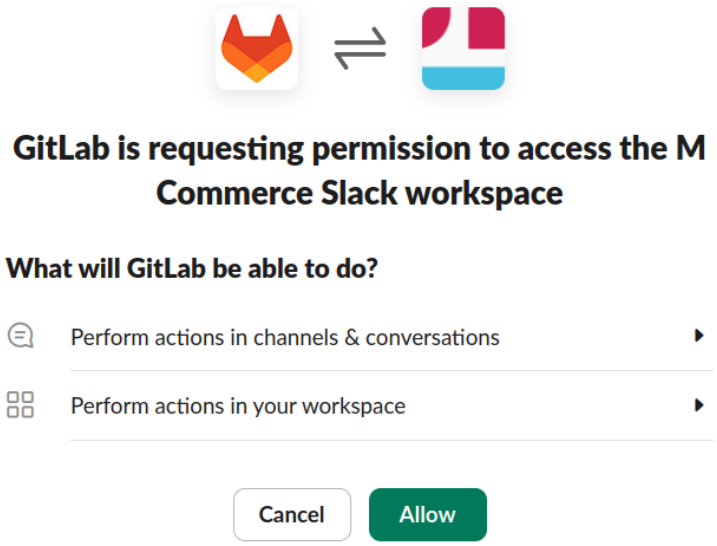


Figure 5.52

We then add GitLab to our security-notifications channel.

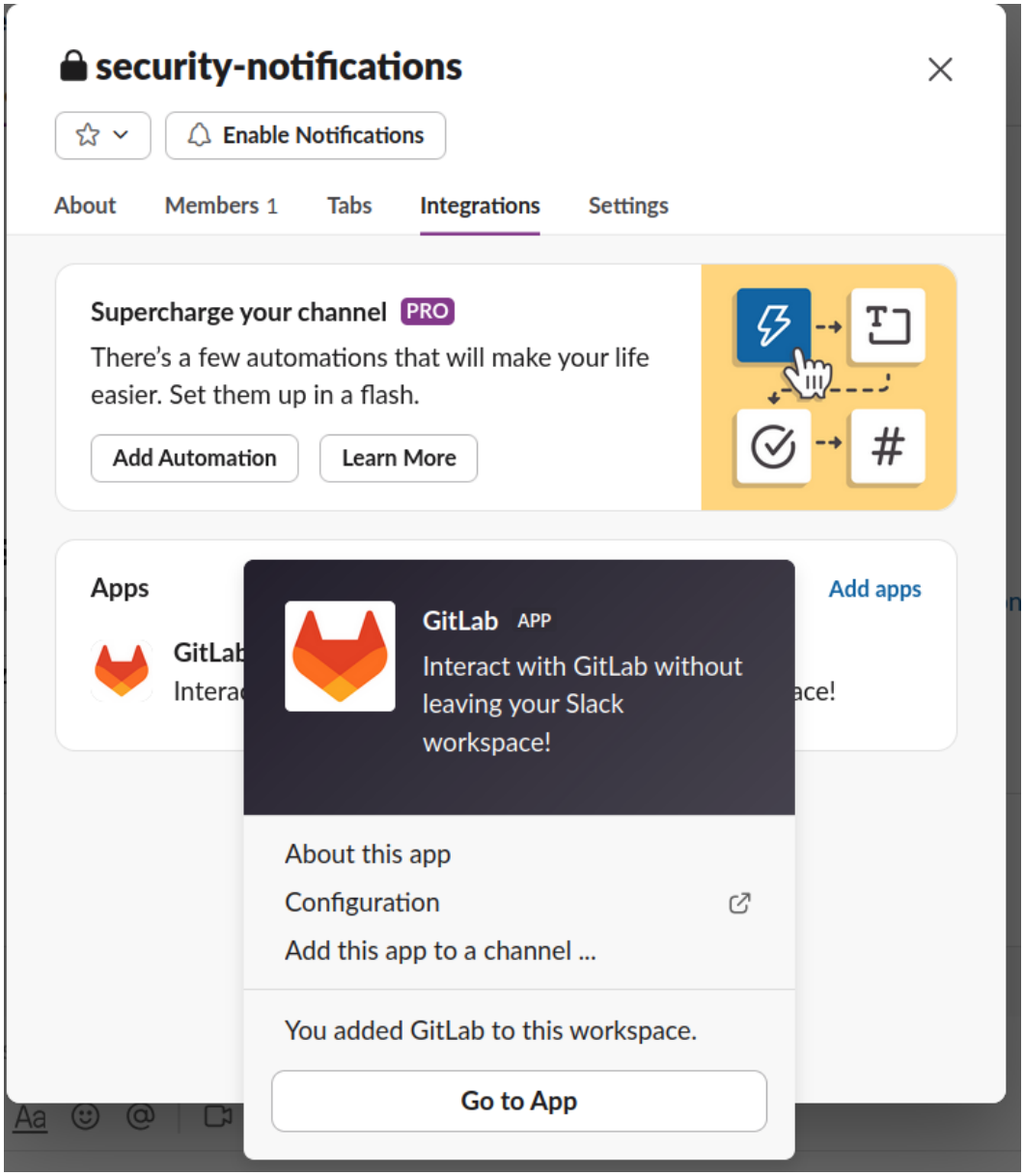


Figure 5.53

And finally we configure GitLab to send new vulnerabilities notifications to the channel security-notifications.

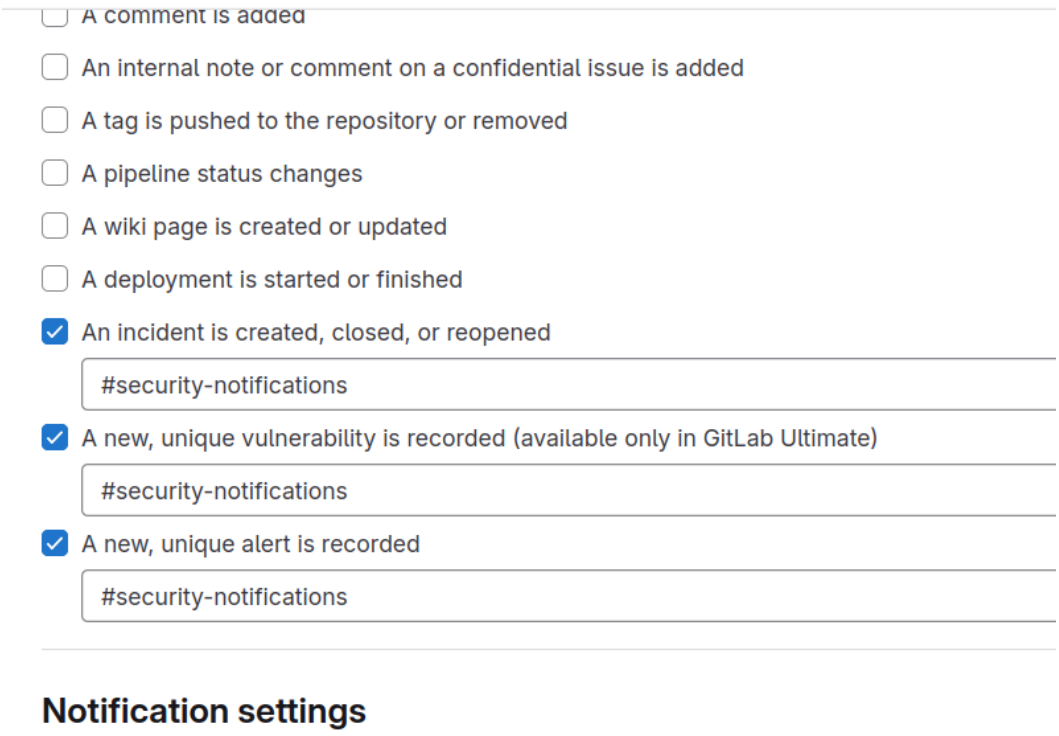


Figure 5.54

5.8 Automated Pipeline Cleaning

First, we added the `cleanup` stage to the list of stages in `stages.yml` as shown below:

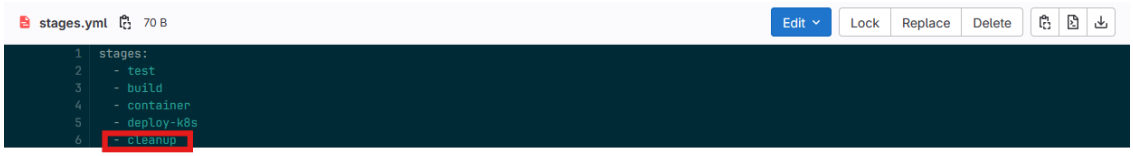


Figure 5.55: Stages.yml updated with the cleanup stage.

Then, we created the `delete old pipeline.yaml` file in which we included the instructions for running our stage:

To ensure the pipeline supports it, we included the stage in the `.gitlab-ci.yml` pipeline file and made an extension of the job file:

Finally, we checked the pipeline to verify that our job is working well. We can observe that the pipeline is executing correctly:

delete_old_pipeline.yml 1.66 KiB Edit Lock Replace Delete

```yaml
.cleanup_old_pipelines:
  stage: cleanup
  image: debian:latest  # Utiliser une image avec les outils nécessaires
  before_script:
    - apt-get update && apt-get install -y  curl httpie jq  # Installer httpie et jq
  script:
    - echo "Démarrage du nettoyage du pipeline..."

    # Suppression des artefacts utilisés depuis plus de 3 mois
    - echo "Suppression des artefacts anciens..."
    - find artifacts/ -type f -mtime +90 -exec rm {} \; || echo "Aucun artefact ancien à supprimer."

    # Suppression des anciens pipelines
    - echo "Suppression des pipelines anciens de plus de 3 mois..."
    - |
      # Récupérer tous les pipelines
      pipelines=$(curl "https://gitlab-ci-token:$MCOMMERCE_TOKEN@gitlab.com//api/v4/projects/<id_projet>/pipelines?per_page=100")

      # Obtenir la date actuelle
      current_date=$(date +%s)

      # Extraire et supprimer les pipelines âgés de plus de 90 jours
      echo "$pipelines" | jq -c '.[]' | while read -r pipeline; do
        created_at=$(echo "$pipeline" | jq -r '.created_at')
        pipeline_id=$(echo "$pipeline" | jq -r '.id')

        # Convertir la date de création en timestamp
        created_at_timestamp=$(date -d "$created_at" +%s)

        # Vérifier si le pipeline a plus de 90 jours
        if [ $(( (current_date - created_at_timestamp) / 86400 )) -gt 90 ]; then
          echo "Suppression du pipeline ID: $pipeline_id, créé le: $created_at"
          http --ignore-stdin --request DELETE --header "PRIVATE-TOKEN: $MCOMMERCE_TOKEN" "https://gitlab.com/api/v4/projects/<id_projet>/pipeline
        fi
      done
```

Figure 5.56: Instructions for the cleanup stage.

.gitlab-ci.yml 1.10 KiB Edit Lock Replace Delete

```yaml
image : docker:latest
services :
  - docker:dind

include:
  - '.gitlab/pre-commit.yml'
  - '.gitlab/stages.yml'
  - template: Jobs/SAST.gitlab-ci.yml
  - template: Jobs/Dependency-Scanning.gitlab-ci.yml
  - '.gitlab/variables.yml'
  - '.gitlab/maven-build.yml'
  - '.gitlab/docker-build.yml'
  - '.gitlab/k8s-deploy.yml'
  - '.gitlab/delete_old_pipeline.yml'


variables:
  GITLAB_ADVANCED_SAST_ENABLED: 'true'


pre-commit-check:
  extends: .pre-commit-check

docker-build:
  needs: [pre-commit-check,maven-build]
  variables:
    DOCKER_BUILDKIT: 1
    DOCKER_CACHE: "true"
    DOCKER_LOGIN: "projdevopsiccn2"
    IMAGE_NAME: "mcommerce"
  extends: .docker-build

gitops-k8s-deploy:
  needs: [docker-build]
  variables:
    GIT_EMAIL: "62484488+terriblemoment0836x@users.noreply.github.com"
    GIT_USERNAME: "adrian_v"
    GITLAB_TOKEN_NAME: "gitlab-ci"
    GITLAB_TOKEN: $K8S_REPO_TOKEN
    GITLAB_REPO_URL: "gitlab.com/adrian_v/product-microservice-kubernetes.git"
    MICROSERVICE_NAME: "product"
  extends: .gitops-k8s-deploy


cleanup_old_pipelines:
#  needs: [gitops-k8s-deploy]
  extends: .cleanup_old_pipelines
```

Figure 5.57: Extension of the pipeline job file.

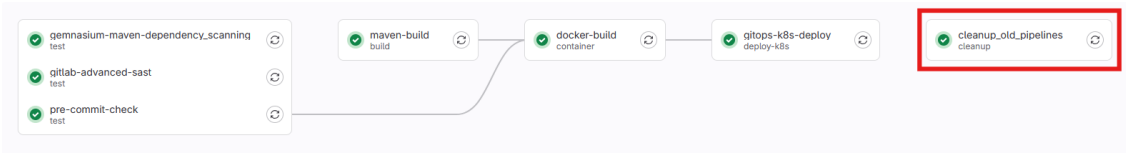


Figure 5.58: Pipeline execution verification.

Chapter 6

Observability and Monitoring

6.1 infrastructure Monitoring

6.1.1 Installing Prometheus on Kubernetes

```
D77R@vps-4f4e3327:/home/marouane$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
"prometheus-community" has been added to your repositories
D77R@vps-4f4e3327:/home/marouane$ helm repo add stable https://charts.helm.sh/stable
"stable" has been added to your repositories
D77R@vps-4f4e3327:/home/marouane$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "stable" chart repository
Update Complete. ⎈Happy Helming!⎈
D77R@vps-4f4e3327:/home/marouane$
```

Figure 6.1: Add prometheus chart to Helm repo

```
D77R@vps-4f4e3327:/home/marouane$ helm install prometheus prometheus-community/kube-prometheus-stack --timeout 10m
NAME: prometheus
LAST DEPLOYED: Mon Jan 13 15:09:16 2025
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace default get pods -l "release=prometheus"

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmana
ger and Prometheus instances using the Operator.
D77R@vps-4f4e3327:/home/marouane$
```

Figure 6.2: Installing Prometheus on Kubernetes

```
D77R@vps-4f4e3327:/home/marouane$ kubectl get all
NAME                                                     READY   STATUS             RESTARTS      AGE
pod/prometheus-grafana-578946f5d5-qtnb7                  3/3     Running            3 (79s ago)   8m5s
pod/prometheus-kube-prometheus-operator-64cc856b7f-mxlzl 0/1     CrashLoopBackOff   1 (20s ago)   8m5s
pod/prometheus-kube-state-metrics-6f5574c548-jwltt       0/1     Running            3 (32s ago)   8m5s
pod/prometheus-prometheus-node-exporter-vd7wm            1/1     Running            2 (88s ago)   8m5s

NAME                                              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)             AGE
service/kubernetes                                ClusterIP   10.96.0.1        <none>        443/TCP             23d
service/prometheus-grafana                        ClusterIP   10.99.250.104    <none>        80/TCP              8m5s
service/prometheus-kube-prometheus-alertmanager   ClusterIP   10.110.98.196    <none>        9093/TCP,8080/TCP   8m5s
service/prometheus-kube-prometheus-operator       ClusterIP   10.99.25.234     <none>        443/TCP             8m5s
service/prometheus-kube-prometheus-prometheus     ClusterIP   10.107.102.208   <none>        9090/TCP,8080/TCP   8m5s
service/prometheus-kube-state-metrics             ClusterIP   10.108.187.46    <none>        8080/TCP            8m5s
service/prometheus-prometheus-node-exporter       ClusterIP   10.104.22.197    <none>        9100/TCP            8m5s

NAME                                                 DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
daemonset.apps/prometheus-prometheus-node-exporter   1         1         1       1            1           kubernetes.io/os=linux   8m5s

NAME                                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/prometheus-grafana                   1/1     1            1           8m5s
deployment.apps/prometheus-kube-prometheus-operator  0/1     1            0           8m5s
deployment.apps/prometheus-kube-state-metrics        0/1     1            0           8m5s

NAME                                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/prometheus-grafana-578946f5d5                  1         1         1       8m5s
replicaset.apps/prometheus-kube-prometheus-operator-64cc856b7f 1         1         0       8m5s
replicaset.apps/prometheus-kube-state-metrics-6f5574c548       1         1         0       8m5s
D77R@vps-4f4e3327:/home/marouane$
```

Figure 6.3: Creation of prometheus components : pods ,service and daemon-set ...

```
D77R@vps-4f4e3327:/home/marouane$ kubectl get configmap
NAME                                                  DATA   AGE
kube-root-ca.crt                                      1      23d
prometheus-grafana                                    1      9m16s
prometheus-grafana-config-dashboards                  1      9m16s
prometheus-kube-prometheus-alertmanager-overview      1      9m16s
prometheus-kube-prometheus-apiserver                  1      9m16s
prometheus-kube-prometheus-cluster-total              1      9m16s
prometheus-kube-prometheus-controller-manager         1      9m16s
prometheus-kube-prometheus-etcd                       1      9m16s
prometheus-kube-prometheus-grafana-datasource         1      9m16s
prometheus-kube-prometheus-grafana-overview           1      9m16s
prometheus-kube-prometheus-k8s-coredns                1      9m16s
prometheus-kube-prometheus-k8s-resources-cluster      1      9m16s
prometheus-kube-prometheus-k8s-resources-multicluster 1      9m16s
prometheus-kube-prometheus-k8s-resources-namespace    1      9m16s
prometheus-kube-prometheus-k8s-resources-node         1      9m16s
prometheus-kube-prometheus-k8s-resources-pod          1      9m16s
prometheus-kube-prometheus-k8s-resources-workload     1      9m16s
prometheus-kube-prometheus-k8s-resources-workloads-namespace 1 9m16s
prometheus-kube-prometheus-kubelet                    1      9m16s
prometheus-kube-prometheus-namespace-by-pod           1      9m16s
prometheus-kube-prometheus-namespace-by-workload      1      9m16s
prometheus-kube-prometheus-node-cluster-rsrc-use      1      9m16s
prometheus-kube-prometheus-node-rsrc-use              1      9m16s
prometheus-kube-prometheus-nodes                      1      9m16s
prometheus-kube-prometheus-nodes-aix                  1      9m16s
prometheus-kube-prometheus-nodes-darwin               1      9m16s
prometheus-kube-prometheus-persistentvolumesusage     1      9m16s
prometheus-kube-prometheus-pod-total                  1      9m16s
prometheus-kube-prometheus-prometheus                 1      9m16s
prometheus-kube-prometheus-proxy                      1      9m16s
prometheus-kube-prometheus-scheduler                  1      9m16s
prometheus-kube-prometheus-workload-total             1      9m16s
D77R@vps-4f4e3327:/home/marouane$
```

Figure 6.4: Prometheus ConfigMap

```
D77R@vps-4f4e3327:/home/marouane$ kubectl get secret
NAME                                                TYPE                 DATA   AGE
alertmanager-prometheus-kube-prometheus-alertmanager Opaque              1      10m
prometheus-grafana                                  Opaque               3      10m
prometheus-kube-prometheus-admission                Opaque               3      10m
sh.helm.release.v1.prometheus.v1                    helm.sh/release.v1   1      14m
D77R@vps-4f4e3327:/home/marouane$
```

Figure 6.5: Secrets

```
D77R@vps-4f4e3327:/home/marouane$ kubectl get crd
NAME                                        CREATED AT
alertmanagerconfigs.monitoring.coreos.com   2025-01-13T15:06:12Z
alertmanagers.monitoring.coreos.com         2025-01-13T15:06:12Z
applications.argoproj.io                    2024-12-21T14:26:22Z
applicationsets.argoproj.io                 2024-12-21T14:26:23Z
appprojects.argoproj.io                     2024-12-21T14:26:23Z
podmonitors.monitoring.coreos.com           2025-01-13T15:06:13Z
probes.monitoring.coreos.com                2025-01-13T15:06:13Z
prometheusagents.monitoring.coreos.com      2025-01-13T15:06:13Z
prometheuses.monitoring.coreos.com          2025-01-13T15:06:14Z
prometheusrules.monitoring.coreos.com       2025-01-13T15:06:15Z
scrapeconfigs.monitoring.coreos.com         2025-01-13T15:06:15Z
servicemonitors.monitoring.coreos.com       2025-01-13T15:06:16Z
thanosrulers.monitoring.coreos.com          2025-01-13T15:06:17Z
D77R@vps-4f4e3327:/home/marouane$
```

Figure 6.6: Custom Resource Definition CRD

6.1.2 Access Prometheus UI

```
D77R@vps-4f4e3327:/home/marouane$ kubectl get pod
NAME                                                READY   STATUS             RESTARTS        AGE
prometheus-grafana-578946f5d5-qtnb7                 3/3     Running            3 (5m11s ago)   11m
prometheus-kube-prometheus-operator-64cc856b7f-mxlzl 0/1    CrashLoopBackOff   5 (83s ago)     11m
prometheus-kube-state-metrics-6f5574c548-jwltt      0/1     CrashLoopBackOff   4 (44s ago)     11m
prometheus-prometheus-node-exporter-vd7wm           1/1     Running            6               11m
D77R@vps-4f4e3327:/home/marouane$
```

Figure 6.7: Pods

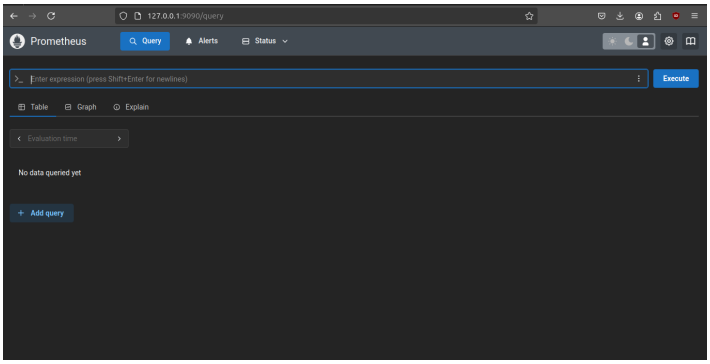


Figure 6.8: Prometheus WEB UI

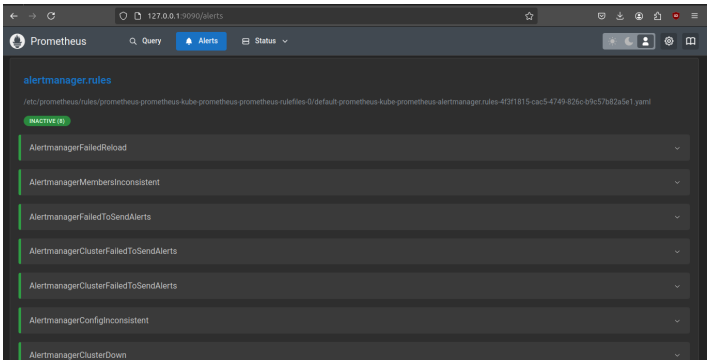


Figure 6.9: Prometheus Alerts

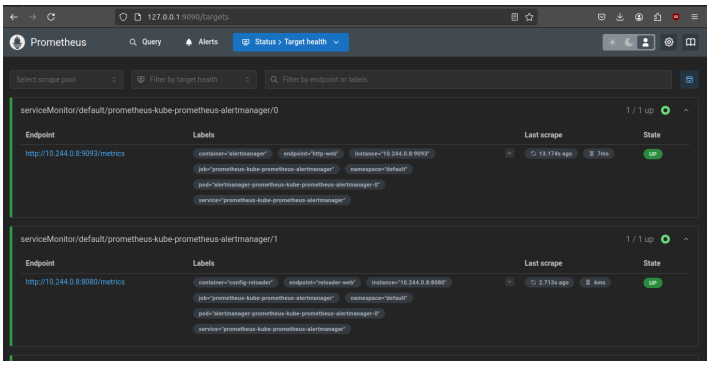


Figure 6.10: Prometheus Taragets

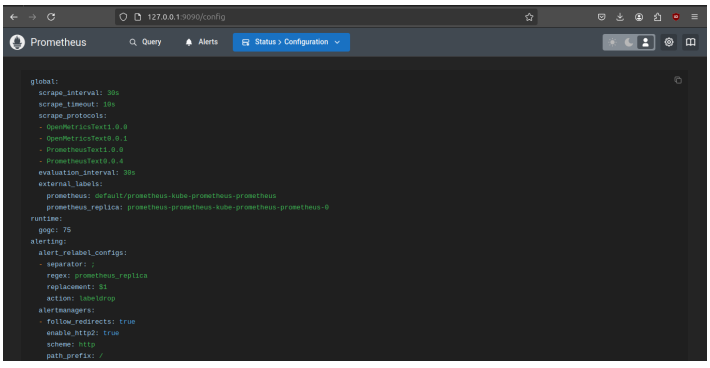


Figure 6.11: Prometheus yaml file for configurations

6.1.3 Graphana

```
D77R@vps-4f4e3327:~$ kubectl get pod
NAME                                                   READY   STATUS    RESTARTS   AGE
alertmanager-prometheus-kube-prometheus-alertmanager-0 2/2     Running   0          38m
grafana-69d855495d-lhm6k                               0/1     Running   0          18s
prometheus-grafana-578946f5d5-jm8pw                    3/3     Running   0          38m
prometheus-kube-prometheus-operator-6cb5954785-jw7m6   1/1     Running   0          38m
prometheus-kube-state-metrics-6f5574c548-kvrq9         1/1     Running   0          38m
prometheus-prometheus-kube-prometheus-prometheus-0     2/2     Running   0          38m
prometheus-prometheus-node-exporter-nkfv2              1/1     Running   0          38m
D77R@vps-4f4e3327:~$
```

Figure 6.12: Graphana Pod

```
D77R@vps-4f4e3327:/home/marouane$ kubectl port-forward prometheus-grafana-578946f5d5-jm8pw 3001
Forwarding from 127.0.0.1:3001 -> 3001
Forwarding from [::1]:3001 -> 3001
```

Figure 6.13: Access the Grafana application in a cluster using Port Forwarding

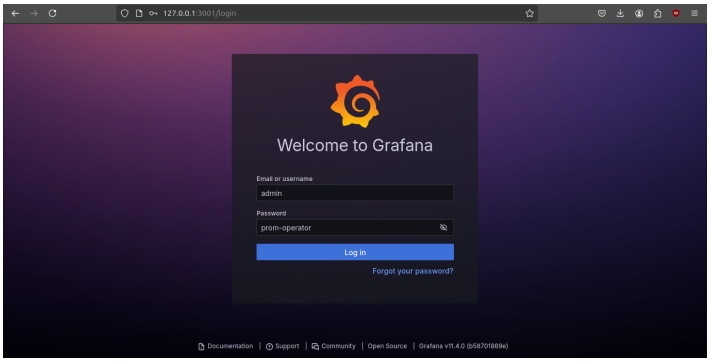


Figure 6.14: Graphana login

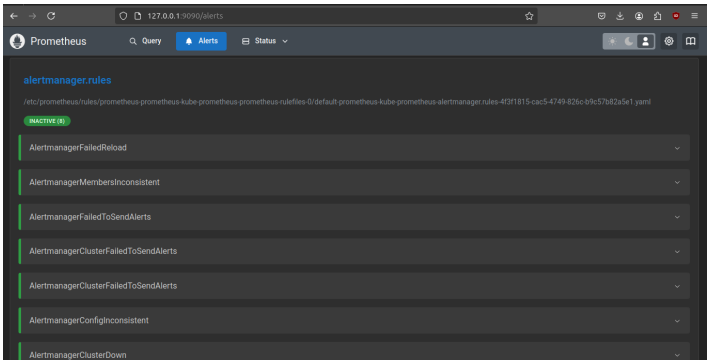


Figure 6.15: alerts

Graphana also offers a user-freindly interface for mesuring system metrics suj as : RAM , CPU , network ...

Figure 6.16: CPU

Figure 6.17: Memory

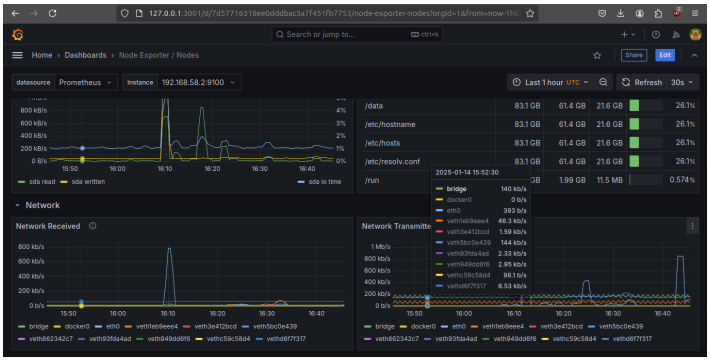


Figure 6.18: Network

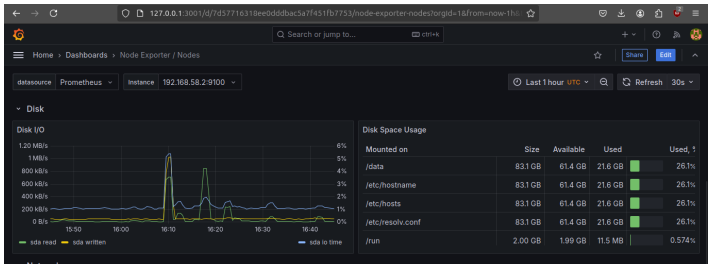


Figure 6.19: Disk usage

6.2 Implementation of Application Monitoring for the Produit Microservice

To ensure the reliability and performance of the **Produit** microservice, we integrated **Prometheus** and **Grafana** into our GitLab CI/CD pipeline. This setup enables real-time monitoring and visualization of the microservice's behavior.

6.2.1 Why Prometheus and Grafana?

- **Prometheus**: An open-source monitoring tool that collects metrics from applications and provides a powerful query language (`PromQL`) for analysis.
- **Grafana**: A widely-used visualization platform that integrates with Prometheus to provide detailed insights through custom dashboards and alerts.

6.2.2 Implementation Steps

6.2.2.1 Prometheus Setup

1. Added Prometheus to the GitLab pipeline as a service using the `prom/prometheus:latest` image.
2. Created a `prometheus.yml` file to scrape metrics from the microservice's endpoint (`/metrics`).
3. Integrated Prometheus into the Maven project by exposing metrics using the Prometheus Java Client library.

6.2.2.2 Grafana Setup

1. Added Grafana to the GitLab pipeline as a service using the `grafana/grafana:latest` image.

2. Configured Grafana to use Prometheus as a data source.

6.2.3 Screenshots and Observations

Below are screenshots capturing key aspects of the implementation:

- Prometheus scraping metrics from the **Produit** microservice (Figure 6.20).
- Grafana dashboard visualizing the metrics (Figure 6.21).

```
111  prometheus:
112    stage: monitoring
113    image: prom/prometheus:latest
114    script:
115      - echo "Prometheus service is running."
116    artifacts:
117      paths:
118        - prometheus.yml
119    variables:
120      PROMETHEUS_CONFIG_PATH: prometheus.yml
121    allow_failure: false
```

Figure 6.20: Prometheus configuration and metrics scraping.

```
123 grafana:
124   stage: monitoring
125   image: grafana/grafana:latest
126   script:
127     - echo "Grafana service is running."
128     - echo "Grafana is accessible at http://localhost:3000 with default credentials (admin/admin)."
129   allow_failure: false
```

Figure 6.21: Grafana dashboard visualizing metrics.

6.2.4 Conclusion

The integration of Prometheus and Grafana into the GitLab CI/CD pipeline provided real-time visibility into the **Produit** microservice's performance. This enabled efficient monitoring and proactive resolution of potential issues.

6.3 Adding Log Monitoring to Analyze Microservice Pod Logs (Using an EFK stack)

6.3.1 Preparation of Minikube and Addition of the Stable Helm Repository

In this step, we prepared the Minikube environment and added the stable Helm repository to enable the installation of necessary Helm charts.

```
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$ mi
nikube addons enable default-storageclass
💡  default-storageclass is an addon maintained by Kubernetes. For any concer
ns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernet
es/minikube/blob/master/OWNERS
🌟  The 'default-storageclass' addon is enabled
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$ mi
nikube addons enable storage-provisioner
💡  storage-provisioner is an addon maintained by minikube. For any concerns
contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernet
es/minikube/blob/master/OWNERS
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  The 'storage-provisioner' addon is enabled
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$ he
lm repo add stable https://charts.helm.sh/stable
"stable" has been added to your repositories
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$ he
lm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "fluent" chart repository
...Successfully got an update from the "elastic" chart repository
...Successfully got an update from the "stable" chart repository
Update Complete. ⎈Happy Helming!⎈
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$
```

Figure 6.22: Minikube Preparation and Addition of the Stable Helm Repository

Main Objective: Ensure that the Minikube environment is ready to deploy applications with Helm.

6.3.2 Deployment of Elasticsearch with Helm

We installed Elasticsearch using a Helm chart to create an Elasticsearch cluster in the Kubernetes environment.

6.3.2.1 Installation and Verification of Elasticsearch Pods

```
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$ he
lm install elasticsearch stable/elasticsearch
WARNING: This chart is deprecated
NAME: elasticsearch
LAST DEPLOYED: Fri Jan 17 07:23:06 2025
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
This Helm chart is deprecated. Please use https://github.com/elastic/helm-cha
rts/tree/master/elasticsearch instead.

---

The elasticsearch cluster has been installed.

Elasticsearch can be accessed:

  * Within your cluster, at the following DNS name at port 9200:

    elasticsearch-client.default.svc

  * From outside the cluster, run these commands in the same shell:

    export POD_NAME=$(kubectl get pods --namespace default -l "app=elasticsea
rch,component=client,release=elasticsearch" -o jsonpath="{.items[0].metadata.
name}")
    echo "Visit http://127.0.0.1:9200 to use Elasticsearch"
    kubectl port-forward --namespace default $POD_NAME 9200:9200
```

Figure 6.23: Installation of Elasticsearch with Helm

Once the Elasticsearch pods were deployed, we checked their status to confirm they were running correctly.

```
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$ kubectl get pods
NAME                                    READY   STATUS            RESTARTS   AGE
elasticsearch-client-8755879d9-8wgqq    0/1     PodInitializing   0          10s
elasticsearch-client-8755879d9-hsvtz    0/1     Init:0/1          0          10s
elasticsearch-data-0                    0/1     Init:0/2          0          10s
elasticsearch-master-0                  0/1     Init:0/2          0          10s
```

Figure 6.24: Initial Status of Elasticsearch Pods

6.3.2.2 Verification of Elasticsearch Services

We then verified the services exposed by Elasticsearch to confirm that the necessary ports were open.

```
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$ kubectl get svc
NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
elasticsearch-client    ClusterIP   10.98.172.134   <none>        9200/TCP   62s
elasticsearch-discovery ClusterIP   None            <none>        9300/TCP   62s
kubernetes              ClusterIP   10.96.0.1       <none>        443/TCP    5m43s
```

Figure 6.25: Verification of Elasticsearch Services

6.3.2.3 Confirmation of Stability of Elasticsearch Pods

After a few minutes, we checked the status of the pods again to ensure they were stable and fully operational.

```
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
elasticsearch-client-8755879d9-7jft2   1/1     Running   0          8m9s
elasticsearch-client-8755879d9-8s8n6   1/1     Running   0          8m9s
elasticsearch-data-0                1/1     Running   0          8m9s
elasticsearch-data-1                1/1     Running   0          3m12s
elasticsearch-master-0              1/1     Running   0          8m9s
elasticsearch-master-1              1/1     Running   0          3m10s
elasticsearch-master-2              1/1     Running   0          2m52s
```

Figure 6.26: Status of Elasticsearch Pods After Stabilization

6.3.3 Configuration and Deployment of Fluentd as a DaemonSet

Fluentd was configured as a DaemonSet to collect logs from pods and send them to Elasticsearch.

6.3.3.1 Creation of the Fluentd YAML File

A YAML file was created to define the Fluentd DaemonSet configuration.

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
    version: v1
spec:
  selector:
    matchLabels:
      k8s-app: fluentd-logging
      version: v1
  template:
    metadata:
      labels:
        k8s-app: fluentd-logging
        version: v1
    spec:
      tolerations:
      - key: node-role.kubernetes.io/control-plane
        effect: NoSchedule
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd
        image: fluent/fluentd-kubernetes-daemonset:v1-debian-elasticsearch
        env:
          - name: K8S_NODE_NAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
          - name:  FLUENT_ELASTICSEARCH_HOST
            value: "elasticsearch-logging"
          - name:  FLUENT_ELASTICSEARCH_PORT
            value: "9200"
          - name: FLUENT_ELASTICSEARCH_SCHEME
            value: "http"
```

Figure 6.27: Fluentd YAML Configuration

6.3.3.2 Verification of Fluentd Pods

We deployed Fluentd and verified its pod in the "kube-system" namespace.

```
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$ ku
bectl apply -f fluentd-daemonset-elasticsearch.yaml
daemonset.apps/fluentd created
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$ ku
bectl get pods -n kube-system
NAME                               READY   STATUS              RESTARTS   AGE
coredns-668d6bf9bc-k5jwg           1/1     Running             0          20m
etcd-minikube                      1/1     Running             0          20m
fluentd-4m5vm                      0/1     ContainerCreating   0          28s
```

Figure 6.28: Verification of Fluentd Pod

6.3.4 Deployment of Kibana with a Custom Configuration File

Finally, we deployed Kibana to visualize the logs collected by Fluentd.

6.3.4.1 Configuration of Kibana

The file `kibana-value.yaml` was created to configure Kibana settings.

```
files:
  kibana.yml:
    server.name: kibana
    server.host: "0"
    elasticsearch.hosts: http://elasticsearch-client:9200
service:
  type: ClusterIP
~
~
```

Figure 6.29: Custom Configuration for Kibana

6.3.4.2 Deployment and Verification of Kibana Pods

We deployed Kibana with Helm and confirmed that the pods were running.

```
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$ vi
 kibana-calue.yaml
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$ he
lm install kibana stable/kibana -f kibana-calue.yaml
WARNING: This chart is deprecated
NAME: kibana
LAST DEPLOYED: Fri Jan 17 08:54:11 2025
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
THE CHART HAS BEEN DEPRECATED!

Find the new  official version @ https://github.com/elastic/helm-charts/tree/
master/kibana

To verify that kibana has started, run:

  kubectl --namespace=default get pods -l "app=kibana"

Kibana can be accessed:

  * From outside the cluster, run these commands in the same shell:

    export POD_NAME=$(kubectl get pods --namespace default -l "app=kibana,rel
ease=kibana" -o jsonpath="{.items[0].metadata.name}")
    echo "Visit http://127.0.0.1:5601 to use Kibana"
    kubectl port-forward --namespace default $POD_NAME 5601:5601
```

Figure 6.30: Deployment of Kibana with Helm

```
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$ ku
bectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
elasticsearch-client-8755879d9-7jft2    1/1     Running   0          26m
elasticsearch-client-8755879d9-8s8n6    1/1     Running   0          26m
elasticsearch-data-0                    1/1     Running   0          26m
elasticsearch-data-1                    1/1     Running   0          21m
elasticsearch-master-0                  1/1     Running   0          26m
elasticsearch-master-1                  1/1     Running   0          21m
elasticsearch-master-2                  1/1     Running   0          21m
kibana-5d5d49f79-xxs85                  1/1     Running   0          6m26s
```

Figure 6.31: Verification of Kibana Pods

6.3.4.3 Verification of Service and Access to Kibana

After verifying the service exposed by Kibana, we configured a port-forward to access Kibana from a web browser.

```
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$ ku
bectl get svc
NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
   AGE
elasticsearch-client    ClusterIP   10.109.22.182    <none>        9200/TCP
   27m
elasticsearch-discovery ClusterIP   None             <none>        9300/TCP
   27m
kibana                  ClusterIP   10.106.246.243   <none>        443/TCP
   6m45s
kubernetes              ClusterIP   10.96.0.1        <none>        443/TCP
   34m
```

Figure 6.32: Verification of Kibana Service

```
yassine@yassine-Ubuntu-Data:~/product-microservice-kubernetes/product-k8s$ ku
bectl port-forward service/kibana 5601:443
Forwarding from 127.0.0.1:5601 -> 5601
Forwarding from [::1]:5601 -> 5601
Handling connection for 5601
```

Figure 6.33: Configuration of Port-Forward to Access Kibana

6.3.4.4 Access to Kibana Dashboard

By accessing the local URL via port-forward, we confirmed that Kibana was functioning correctly.

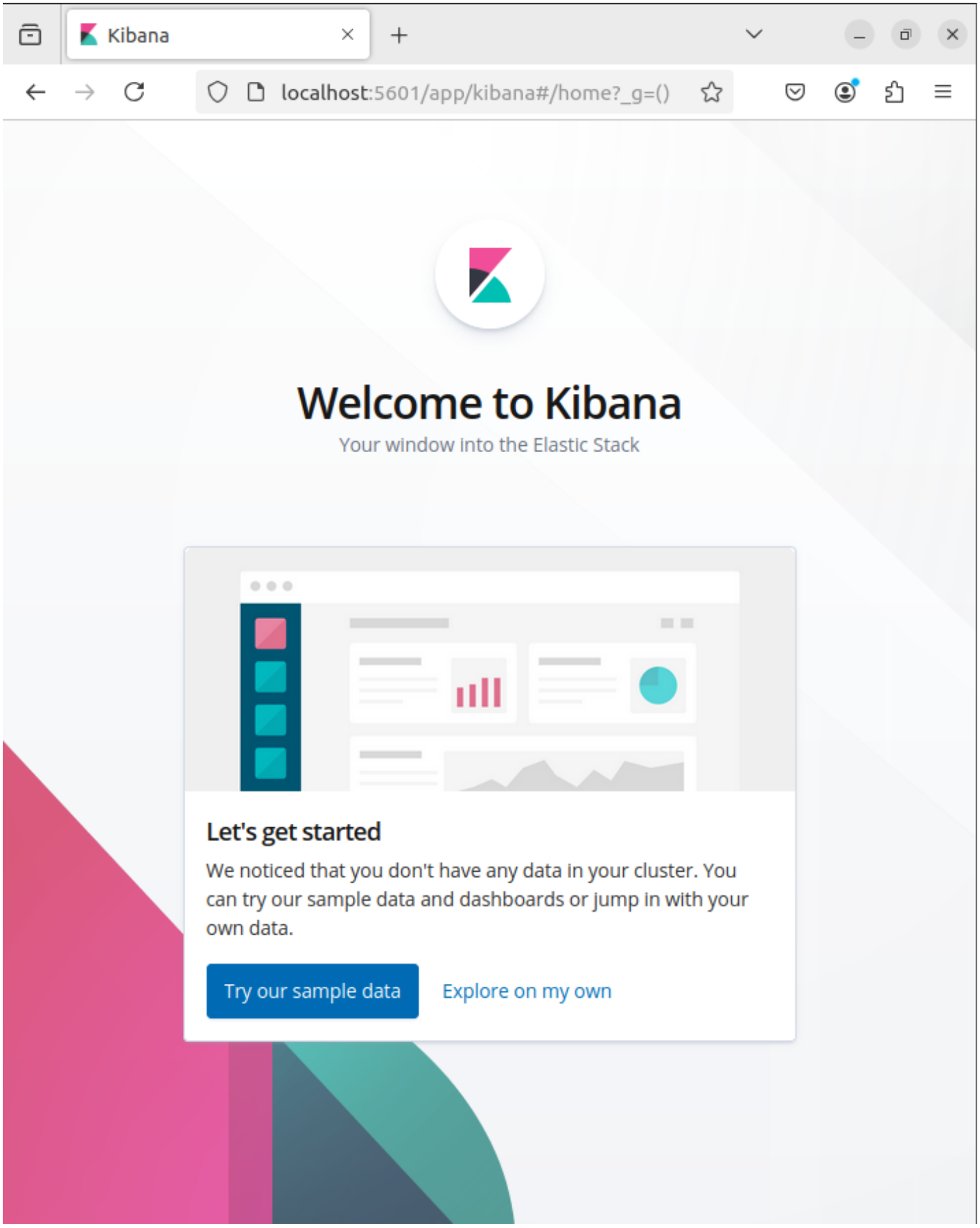


Figure 6.34: Kibana Home Page

6.3.4.5 Log Visualization

Once Kibana was configured, we visualized the logs of the microservice pods to analyze events.

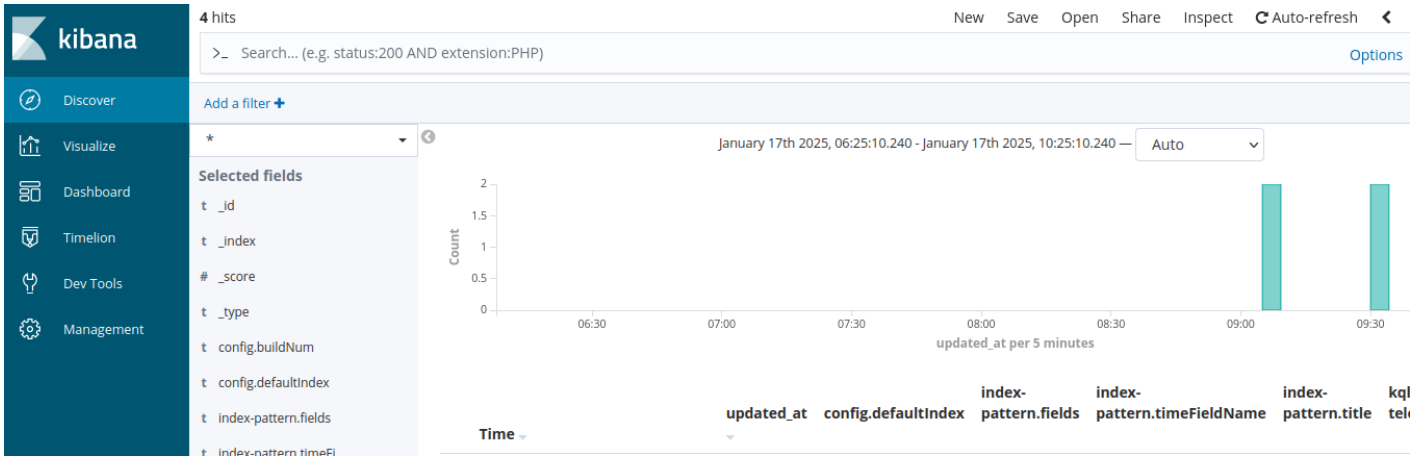


Figure 6.35: Log Visualization in Kibana

6.3.5 Conclusion

With the installation of Elasticsearch, Fluentd, and Kibana, we have set up a complete log monitoring solution to analyze the logs of microservice pods. This architecture ensures better visibility of events and facilitates problem diagnosis in the Kubernetes environment.

General Conclusion and Perspectives

The Proof of Concept (PoC) for automating the build, packaging, and deployment of the Mcommerce application using a DevSecOps approach has demonstrated significant potential for improving KATA's software delivery process. By integrating CI/CD pipelines, containerization, orchestration, and security practices, the project successfully addressed key challenges such as slow delivery cycles, security vulnerabilities, and operational inefficiencies. The results of this PoC highlight the transformative impact of DevSecOps on KATA's ability to deliver high-quality software solutions faster and more securely. While the PoC achieved its objectives, it also revealed areas for improvement, such as the need for more comprehensive testing strategies, further optimization of the CI/CD pipeline. These insights will be invaluable as KATA scales its DevSecOps initiatives across future projects.