DevOps Project

ICCN2

Realized by:

MIFOUAJ Mounssif

LAOUIJ Hamza

LAAFOU Anasse

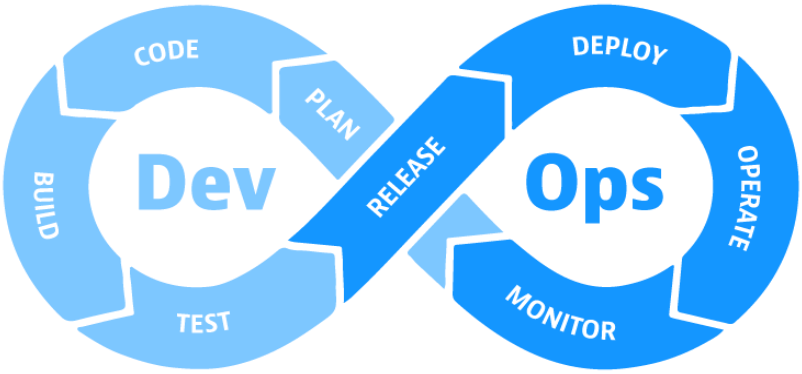
BENTAJA Othmane

ZRAIDI Meriem

AIT-IZANA Ismail

Supervised by:

Mr ALLAKI Driss

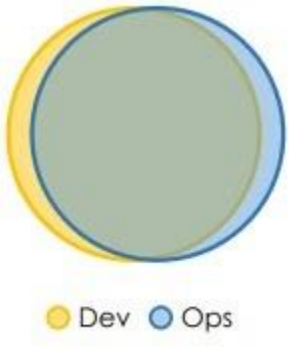


2021-2022

Team Topology

KATA is an organization with a single main web-based product, so we chose “Fully Shared Ops Responsibilities” topology where operations people have been integrated in product development teams.

There is so little separation between Dev and Ops that all people are highly focused on a shared purpose.



Scrum is an Agile software development framework that enables a team to communicate and self-organize.

The SCRUM team is composed of:

Product owner: responsible for the project's outcome and seeks to maximize a product's value by managing and optimizing the product backlog

The scrum master: helps the team enhance and streamline the processes by which they achieve their goals.

Team members: carry out all work required to build increments of value every sprint, and can include researchers, architects, designers …

However, the SCRUM workflow is composed of:

Sprint: is the basic unit of development in scrum, its length is agreed and fixed in advance for each sprint and is normally between one week and one month, with two weeks being the most common.

Sprint planning: The purpose of sprint planning is to define what can be delivered in the sprint and how that work will be achieved.

Sprint Review: one of the most important ceremonies in Scrum where the team gathers to review completed work and determine whether additional changes are needed.

Sprint Retrospective: a recurring meeting held at the end of a sprint used to discuss what went well during the previous sprint cycle and what can be improved for the next sprint.

To manage work in SCRUM, we use the following documents:

Product Backlog: a prioritized list of work for the development team that is derived from the roadmap and its requirements. The most important items are shown at the top of the product backlog so the team knows what to deliver first.

Sprint Backlog: a list of tasks identified by the Scrum team to be completed during the Scrum sprint. During the sprint planning meeting, the team selects some number of product backlog items, usually in the form of user stories, and identifies the tasks necessary to complete each user story.

User Story: an informal, general explanation of a software feature written from the perspective of the end user or customer. They don't go into detail. Requirements are added later, once agreed upon by the team.

Story Points: units of measure for expressing an estimate of the overall effort required to fully implement a product backlog item.

Definition of Done: represents the organization's formal definition of quality for all Product Backlog Items.

The Product Increment: the sum of all the Product Backlog items completed during a Sprint and the value of the increments of all previous Sprints.

To manage our agile project, we chose to use JIRA as our agile project management tool:
We divided our roadmap to 5 epics:

1- Web Development

2- Clientui Pipeline

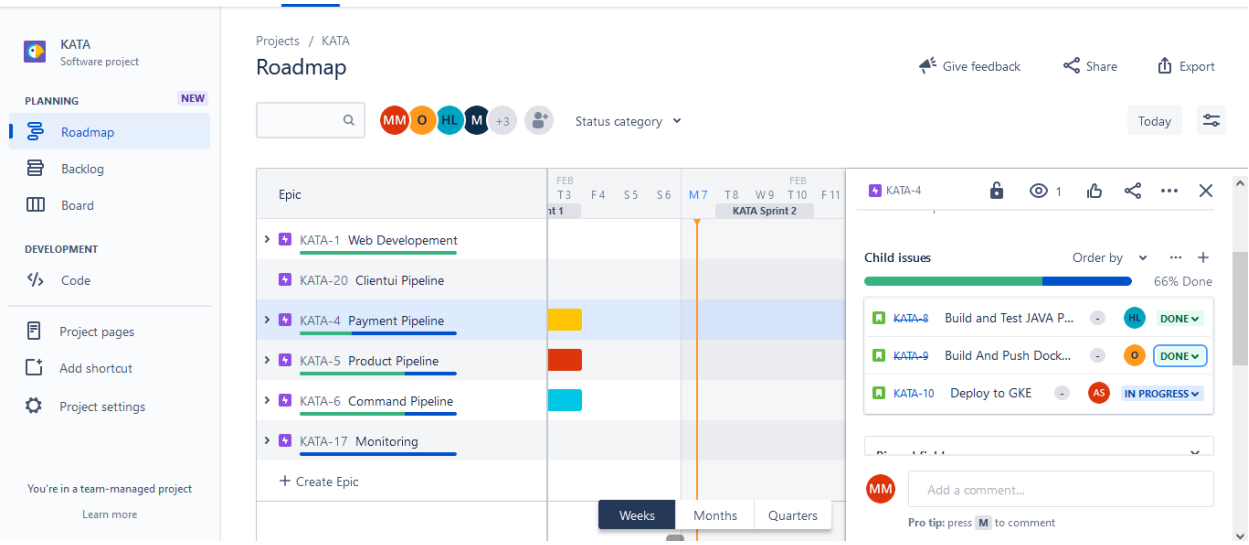
3- Command Pipeline

4- Product Pipeline

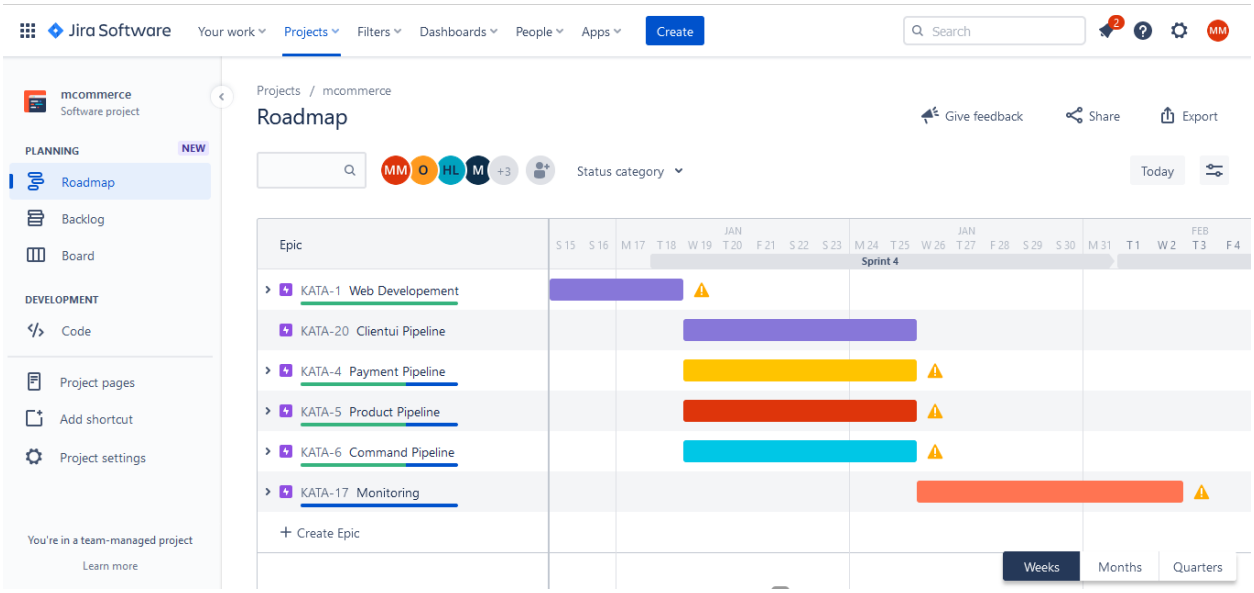
5- Payment Pipeline

6- Monitoring

In the image below, we created the jobs and the tasks in each jobs:

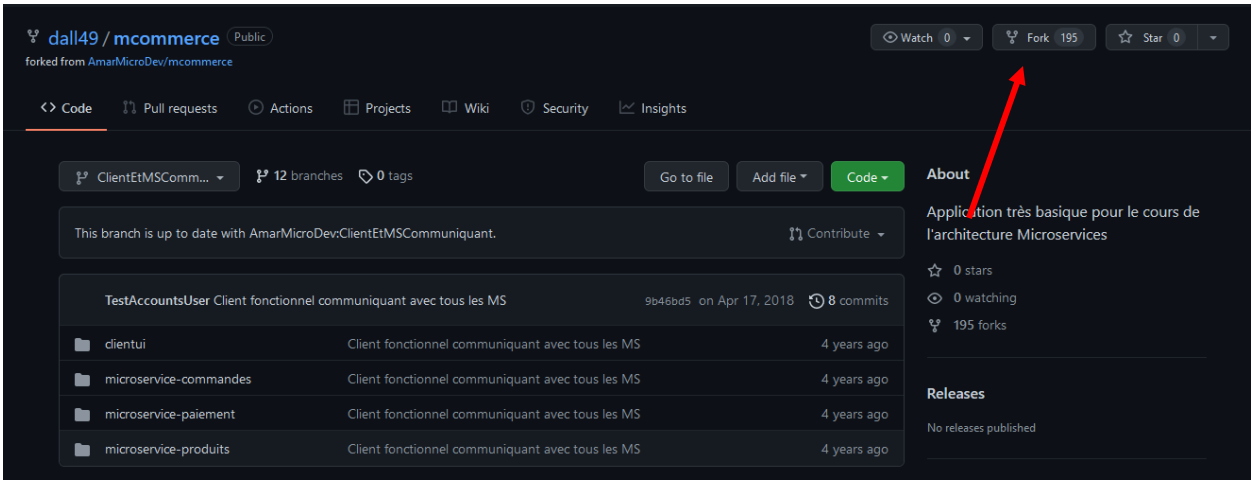


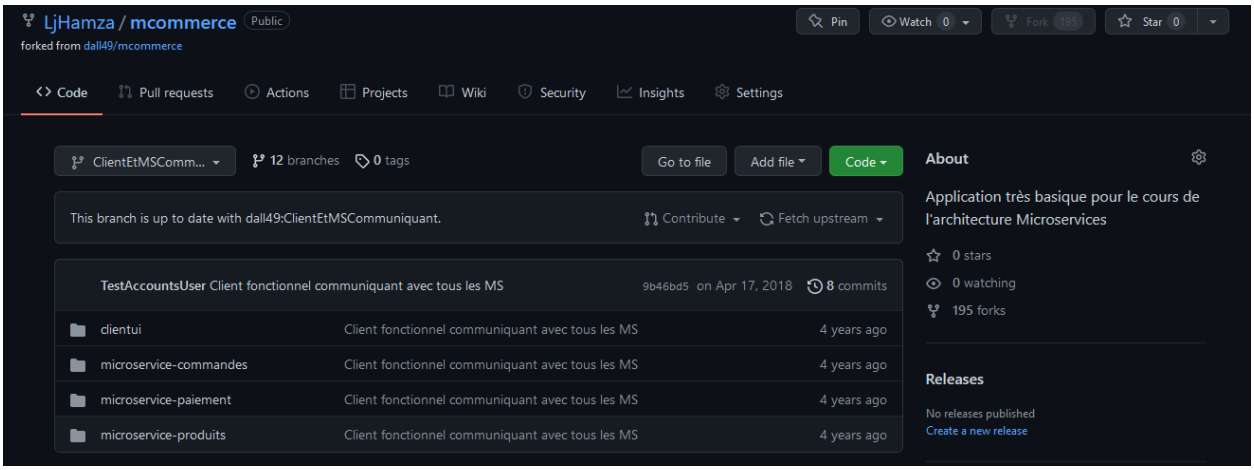
Finally, the image below shows an overview of the project management:


Jira Software
Your work
Projects
Filters
Dashboards
People
Apps
Create
Search
mcommerce
Software project
PLANNING
NEW
Roadmap
Backlog
Board
DEVELOPMENT
Code
Project pages
Add shortcut
Project settings
You're in a team-managed project
Learn more
Projects / mcommerce
Roadmap
Give feedback
Share
Export
Status category
Today
Epic
JAN
FEB
Sprint 4
KATA-1 Web Developement
KATA-20 Clientui Pipeline
KATA-4 Payment Pipeline
KATA-5 Product Pipeline
KATA-6 Command Pipeline
KATA-17 Monitoring
Create Epic
Weeks
Months
Quarters

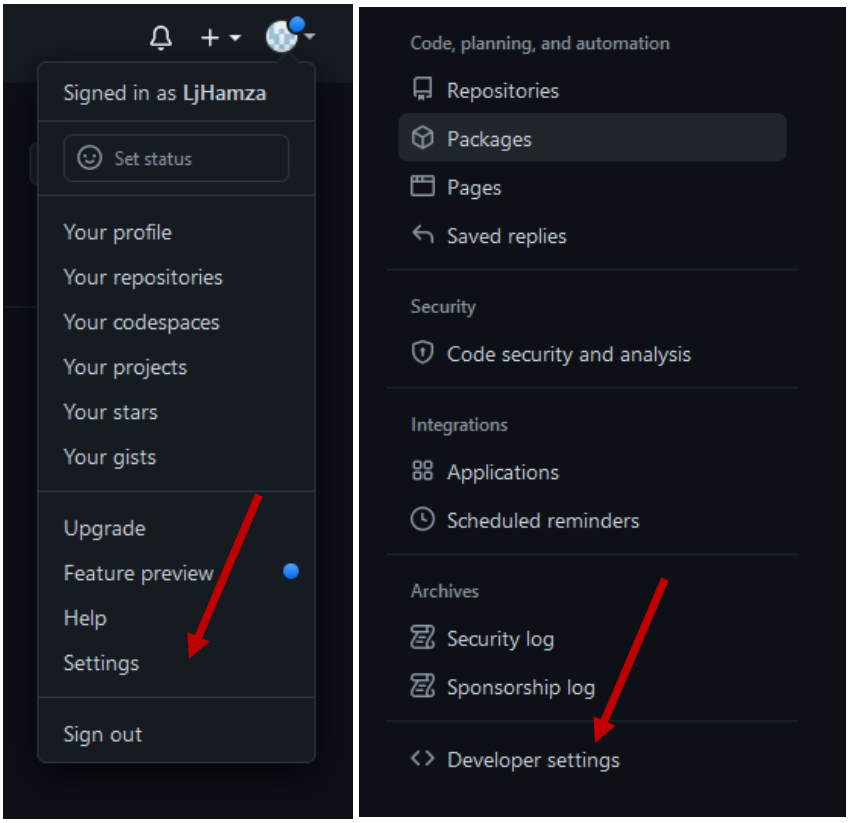
Fork Project from GitHub

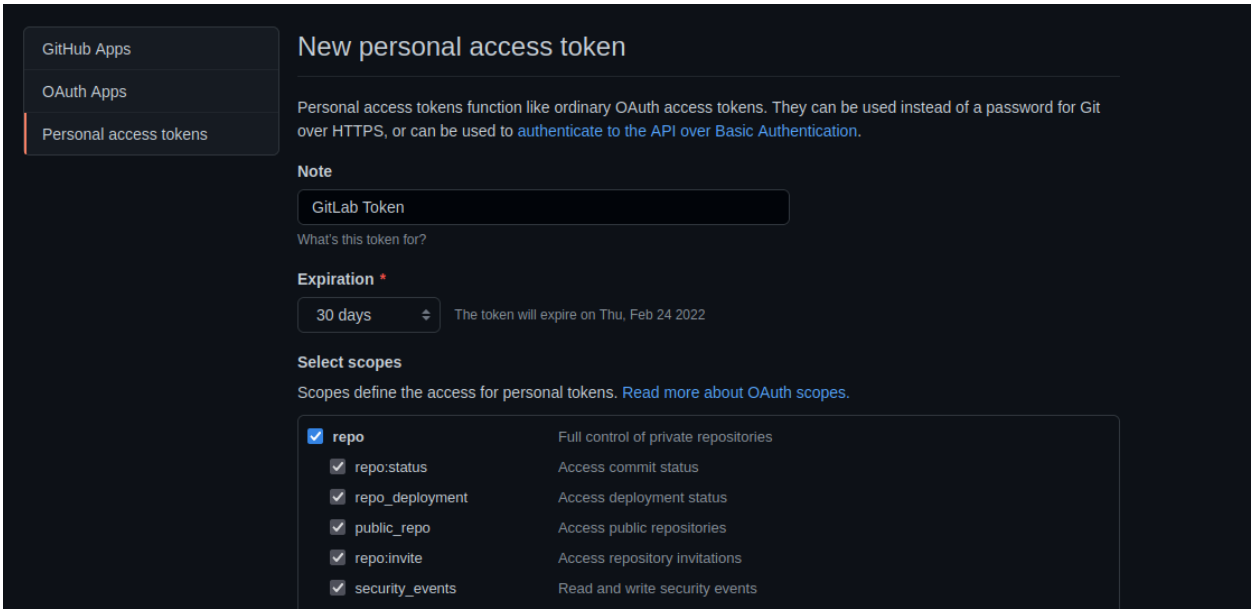
First, we will fork the project:

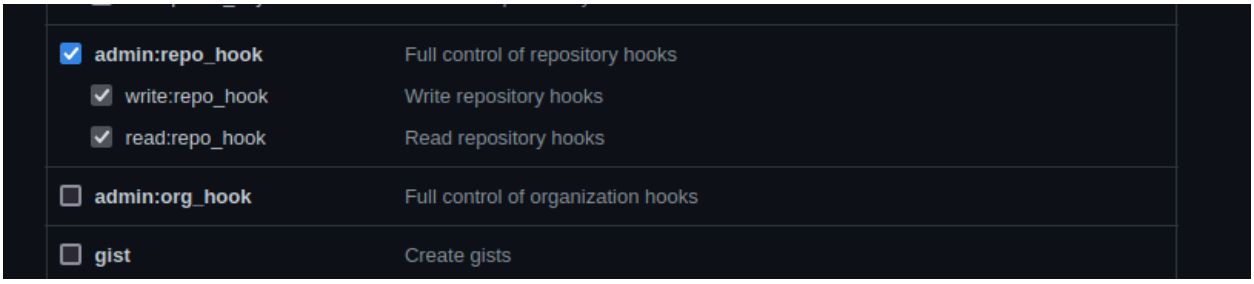




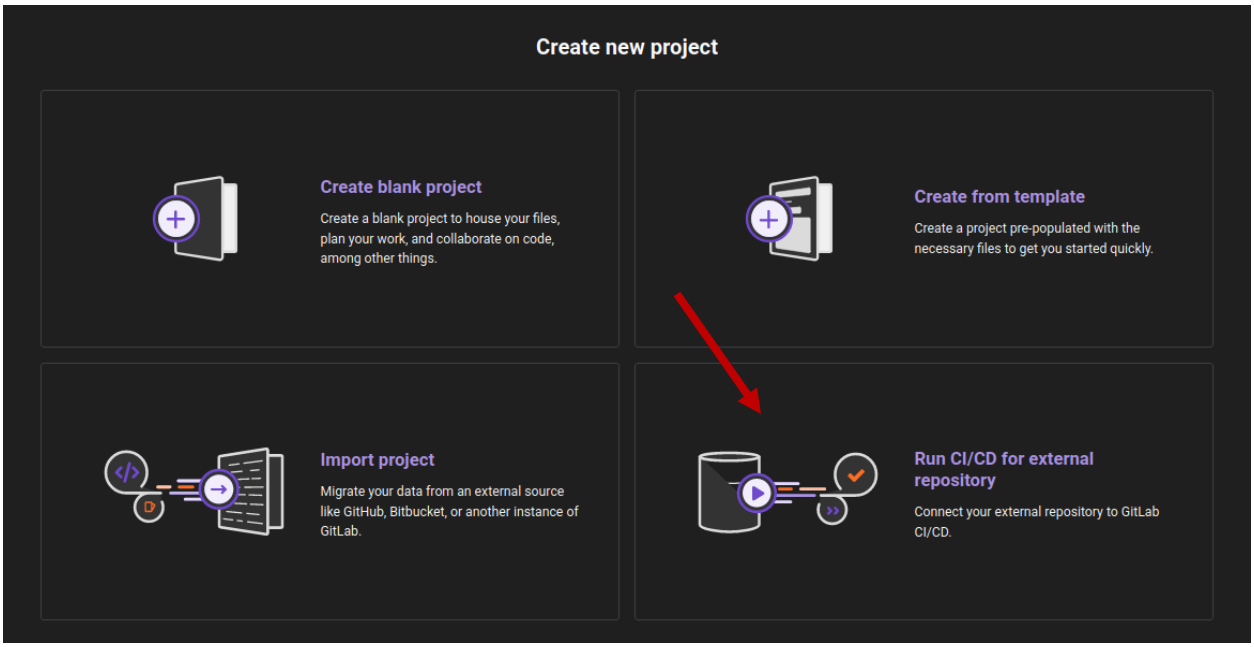
Then, we will create token to connect GitHub with Gitlab.

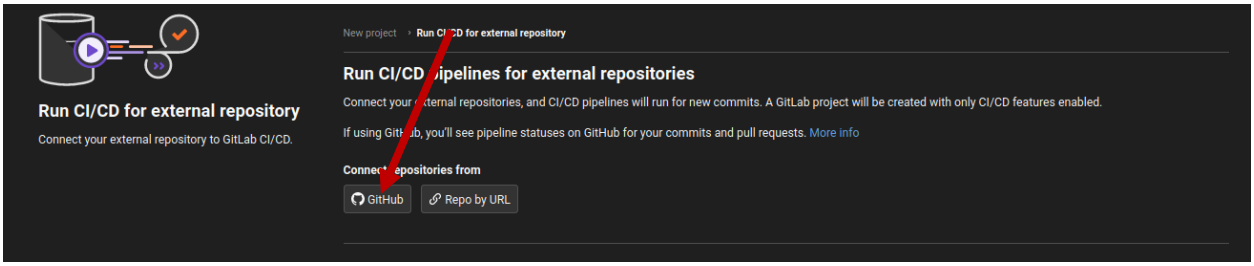

Signed in as LjHamza
Set status
Your profile
Your repositories
Your codespaces
Your projects
Your stars
Your gists
Upgrade
Feature preview
Help
Settings
Sign out
Code, planning, and automation
Repositories
Packages
Pages
Saved replies
Security
Code security and analysis
Integrations
Applications
Scheduled reminders
Archives
Security log
Sponsorship log
Developer settings


GitHub Apps
OAuth Apps
Personal access tokens
New personal access token
Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.
Note
GitLab Token
What's this token for?
Expiration *
30 days
The token will expire on Thu, Feb 24 2022
Select scopes
Scopes define the access for personal tokens. Read more about OAuth scopes.
repo
Full control of private repositories
repo:status
Access commit status
repo_deployment
Access deployment status
public_repo
Access public repositories
repo:invite
Access repository invitations
security_events
Read and write security events

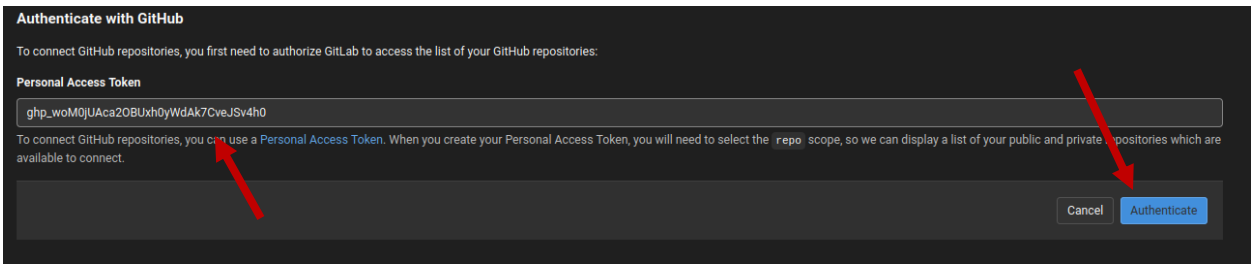

admin:repo_hook
Full control of repository hooks
write:repo_hook
Write repository hooks
read:repo_hook
Read repository hooks
admin:org_hook
Full control of organization hooks
gist
Create gists

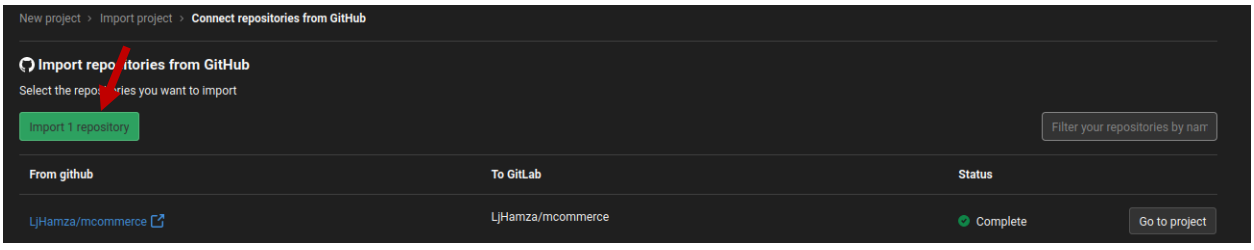
Create Gitlab Project





We paste the Github token:





Create Pipeline

Create pom.xml file in root directory

pom.xml 862 Bytes — Edit in Web IDE | Replace | Delete

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <packaging>pom</packaging>
    <version>2.0.0.RELEASE</version>

    <properties>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
                <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
                <java.version>1.8</java.version>
    </properties>


    <modules>
        <module>clientui</module>
        <module>microservice-commandes</module>
        <module>microservice-paiement</module>
        <module>microservice-produits</module>
    </modules>
</project>
```

Create .gitlab-ci.yml file for the pipeline configuration in root directory

Upload File | CI/CD configuration | Add README | Add LICENSE | Add CHANGELOG | Add CONTRIBUTING | Add Kubernetes cluster | Configure Integrations

Name	Last commit	Last update
microservice-commandes	Update pom.xml	1 week ago
microservice-paiement	Update .gitlab-ci.yml	1 week ago
microservice-produits	Update .gitlab-ci.yml	1 week ago
.gitlab-ci.yml	Update .gitlab-ci.yml file	1 week ago
pom.xml	Update pom.xml	1 week ago

.gitlab-ci.yml 3.26 KB — Edit in Web IDE | Pipeline Editor | Replace | Delete

```yaml
image: maven:3.8.4-jdk-8

services:
  - docker:dind

stages:
  - Build
  - Test
  - test
  - Build and Push Docker Image
  - Deploy
  - dast
```

Build Stage

To keep our code clean we created build template and a variable named MODULE which we will pass to it the name of the four services.

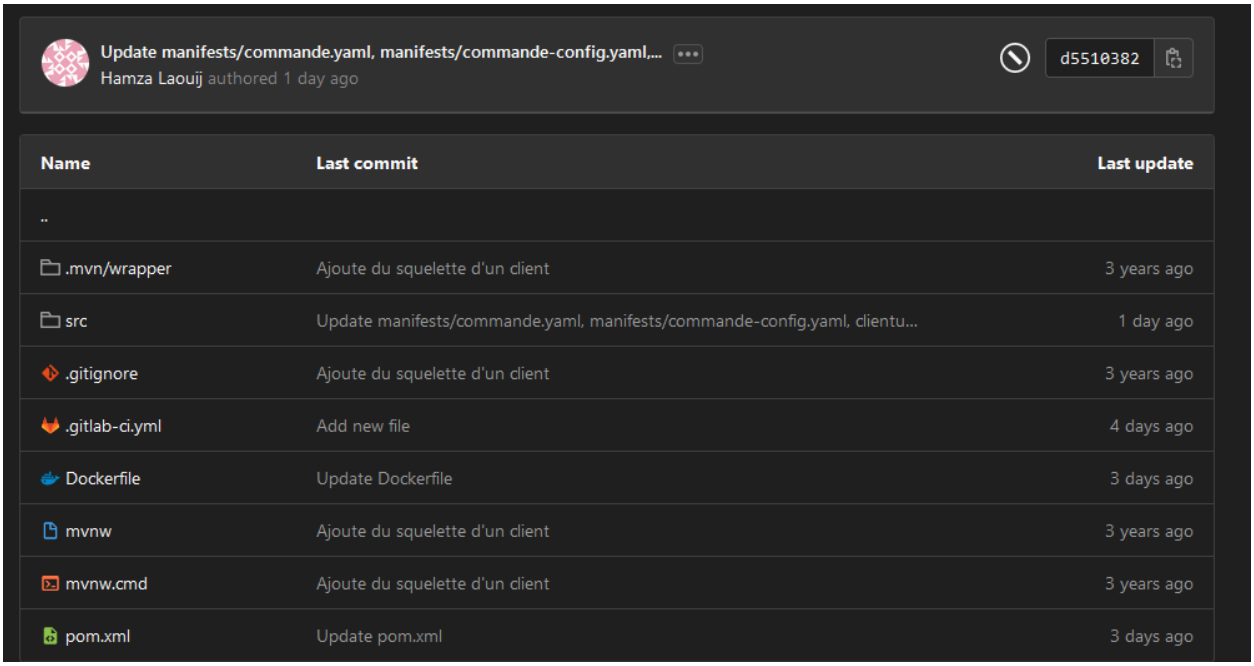
```yaml
variables:
  MAVEN_OPTS: "-Dmaven.repo.local=$CI_PROJECT_DIR/.m2/repository"
  MAVEN_CLI_OPTS: "--batch-mode --errors --fail-at-end --show-version"
  CACHE_KEY: $CI_COMMIT_REF_SLUG

cache:
  key: $CACHE_KEY
  paths:
   - .m2/repository

.build-module:
  stage: Build
  script:
  - echo "Building $MODULE"
  - mvn -pl $MODULE clean package --also-make
  artifacts:
    paths:
    - "*/target"
```

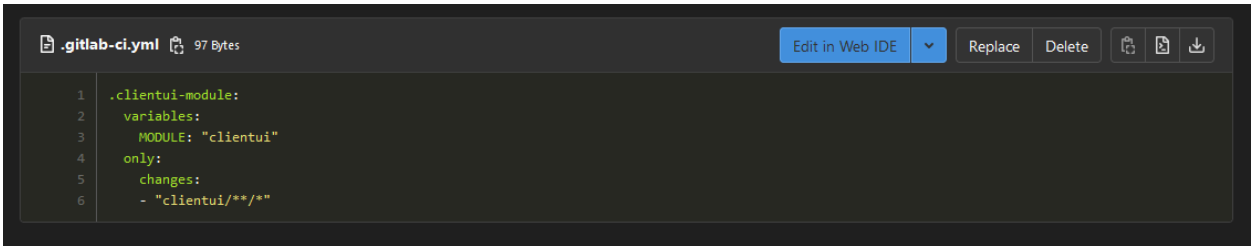
In the build job we pass the build template and the GitLab file that contains the variable content

```yaml
60  # BUILD JOBS
61  build-clientui-module:
62    extends:
63    - .clientui-module
64    - .build-module
65
66  build-commande-module:
67    extends:
68    - .commande-module
69    - .build-module
70
71  build-paiement-module:
72    extends:
73    - .paiement-module
74    - .build-module
75
76  build-produit-module:
77    extends:
78    - .produit-module
79    - .build-module
```

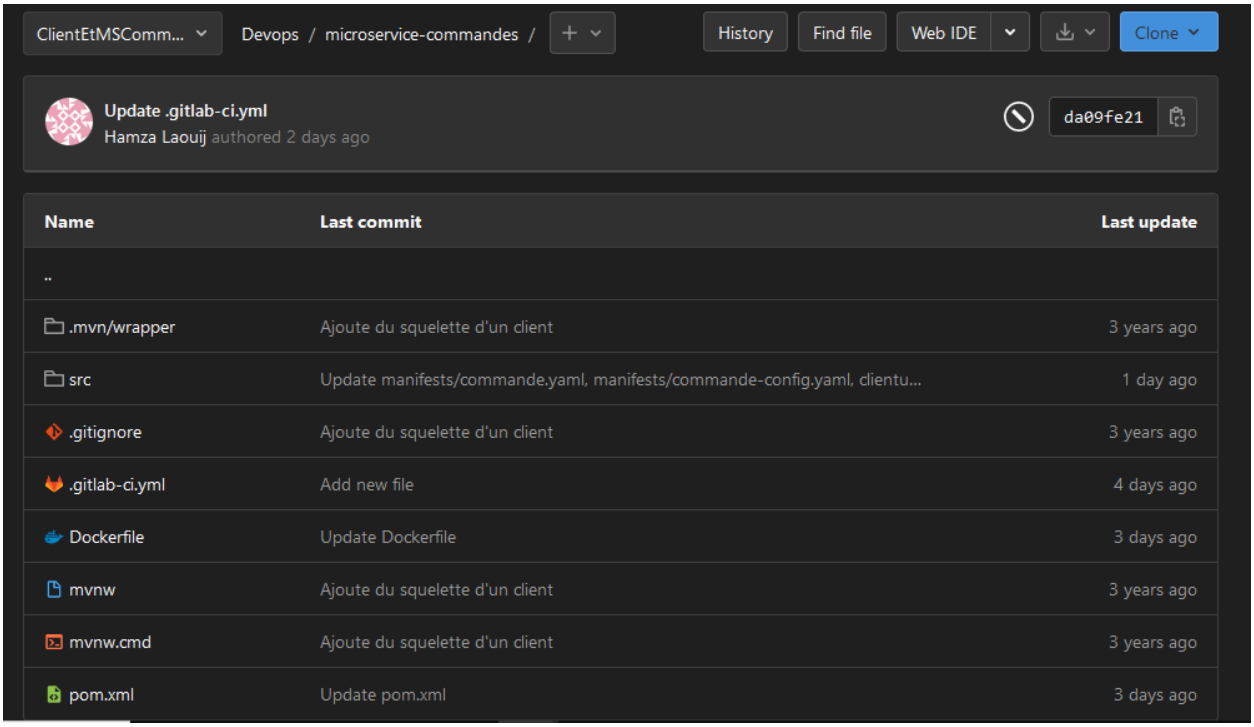
The GitLab files for the different services are:

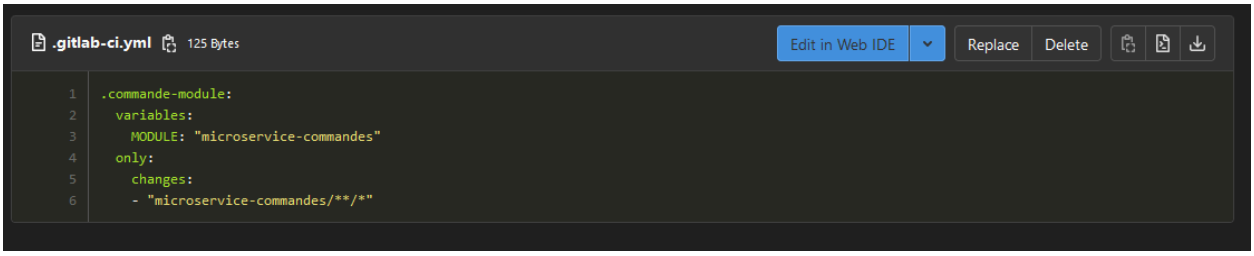
Clientui Service:



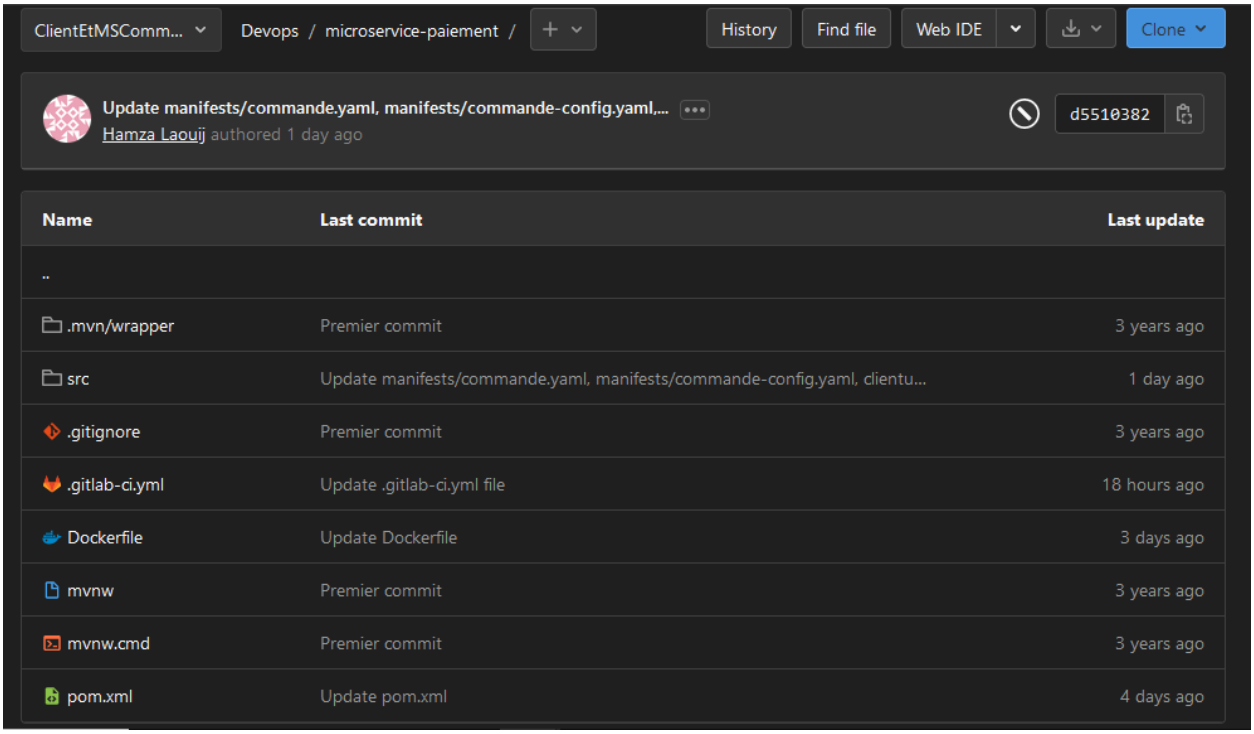


Commande Service:

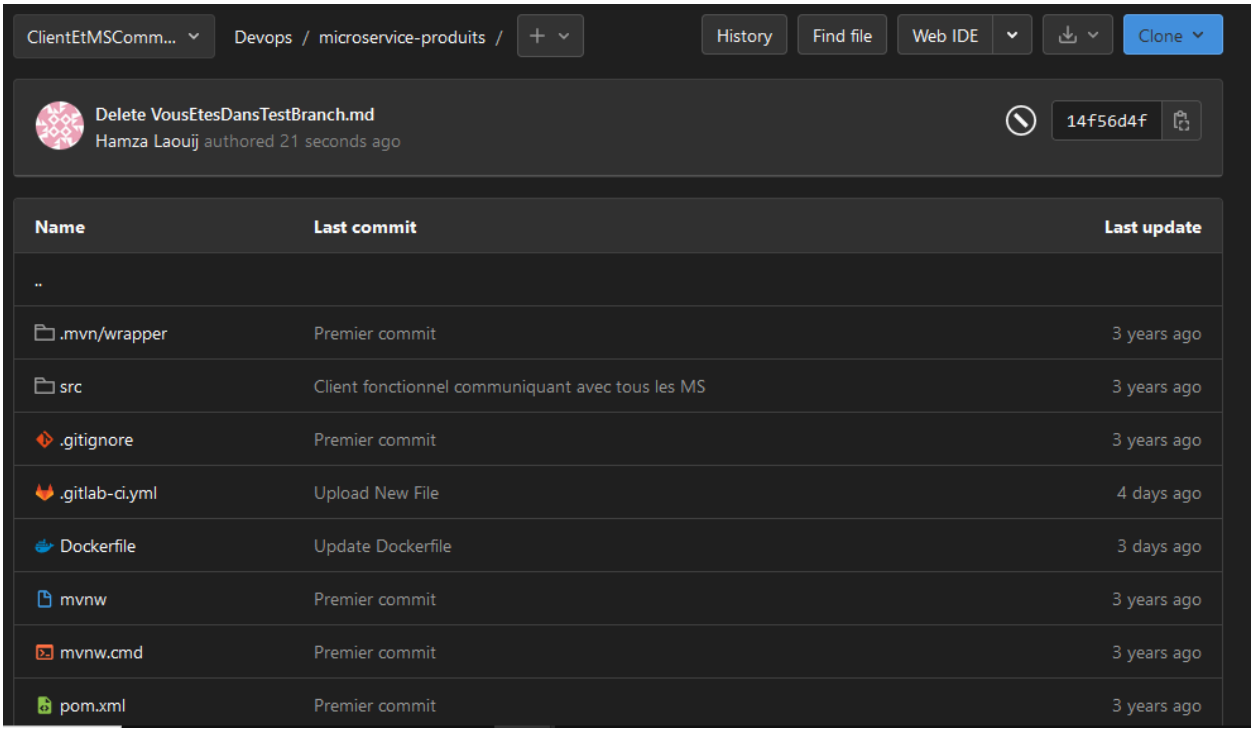


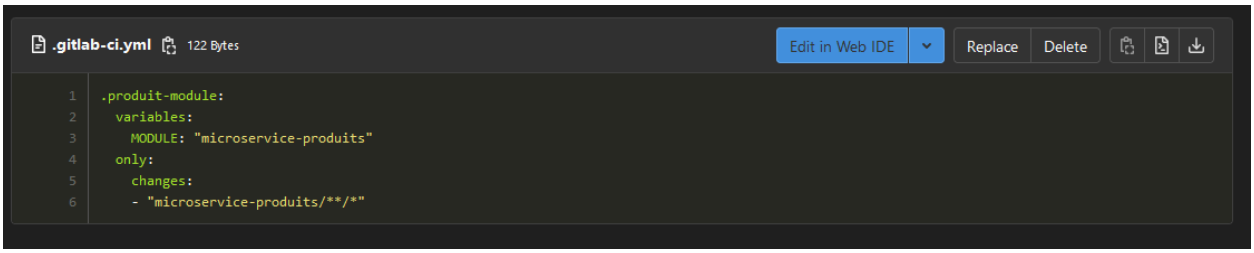


Paiment Service:



Produit Service:





Include files in the root pipeline:

```yaml
include:
  - local: clientui/.gitlab-ci.yml
  - local: microservice-commandes/.gitlab-ci.yml
  - local: microservice-paiement/.gitlab-ci.yml
  - local: microservice-produits/.gitlab-ci.yml
  - template: Security/SAST.gitlab-ci.yml
  - template: Security/Container-Scanning.gitlab-ci.yml
  - template: DAST.gitlab-ci.yml

```

Test Stage

Same as before, we will create test template and pass service name to it:

```
.test-module:
  stage: Test
  script:
    - echo "Testing $MODULE"
    - mvn $MAVEN_CLI_OPTS -pl $MODULE test --also-make

```

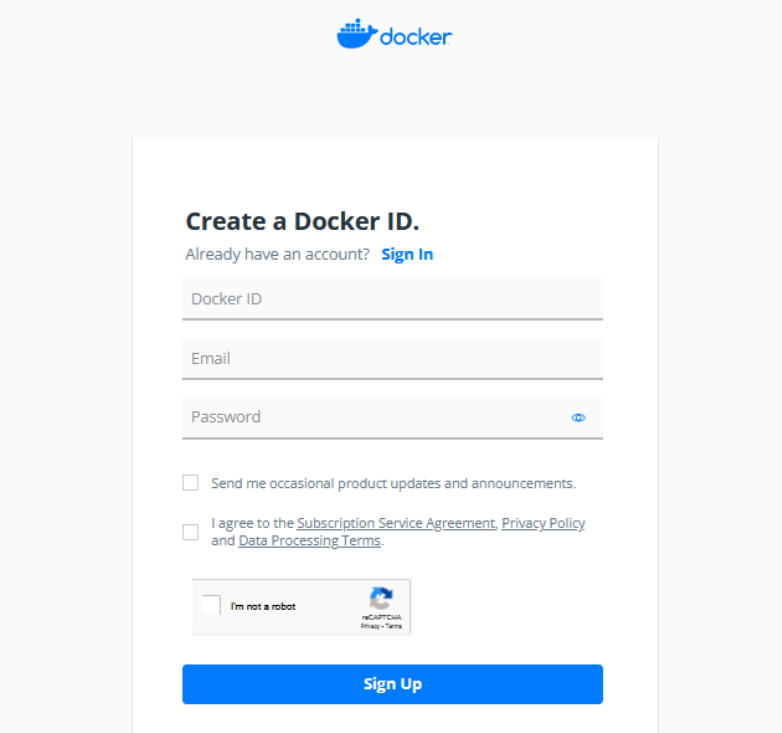
```
# TEST JOBS
test-clientui-module:
  extends:
  - .clientui-module
  - .test-module

test-commande-module:
  extends:
  - .commande-module
  - .test-module

test-paiement-module:
  extends:
    - .paiement-module
    - .test-module

test-produit-module:
  extends:
    - .produit-module
    - .test-module
```

Build And Push Docker Image

Prerequisites:

- Docker Hub Account

Create Docker Hub Account:

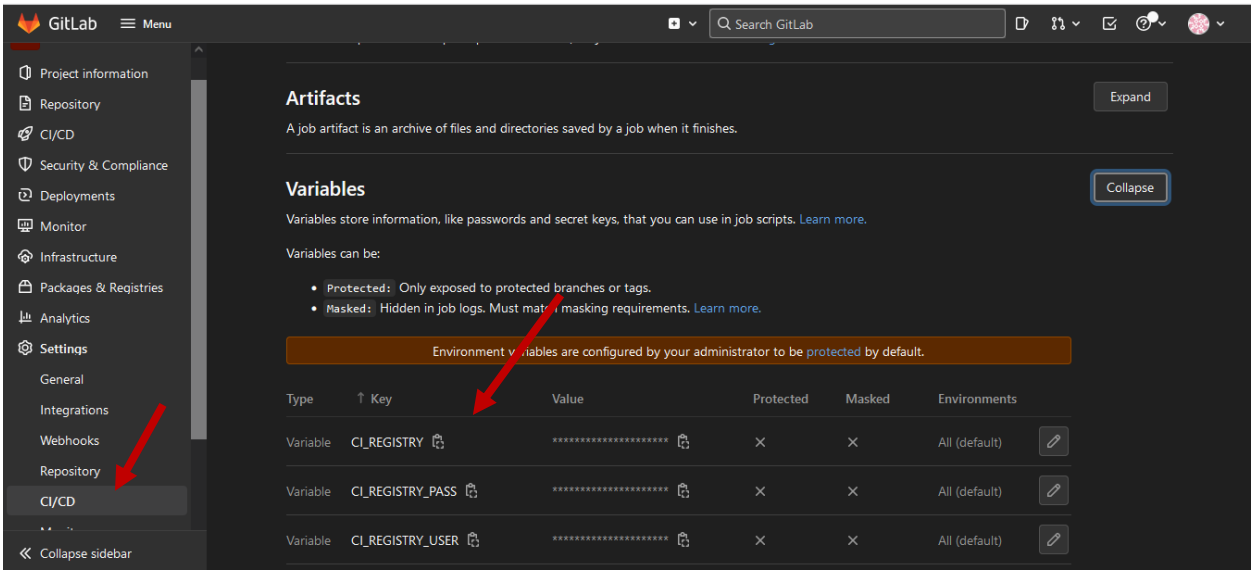


Add credentials of Docker Hub to our pipeline variables:

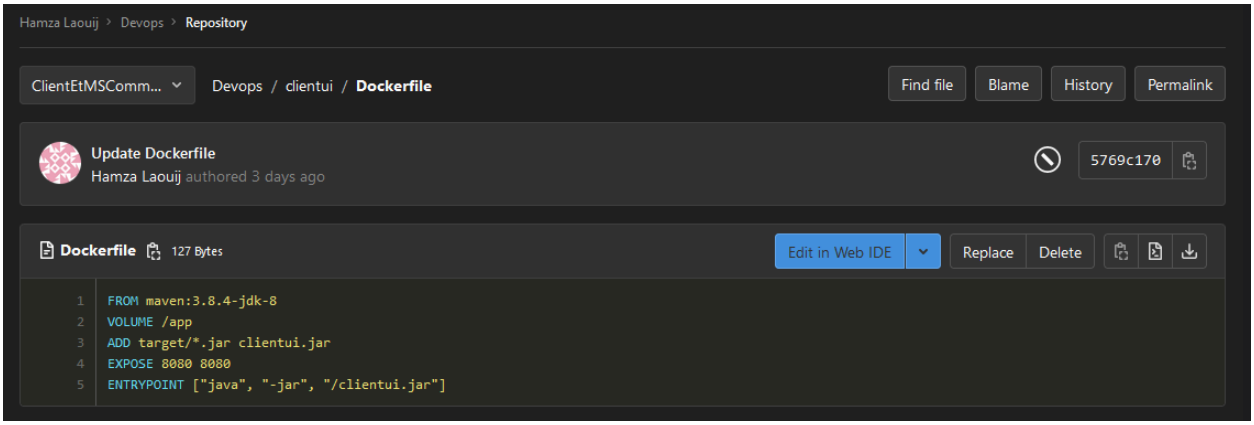
CI_REGISTRY: docker.io, the registry used to push docker images.

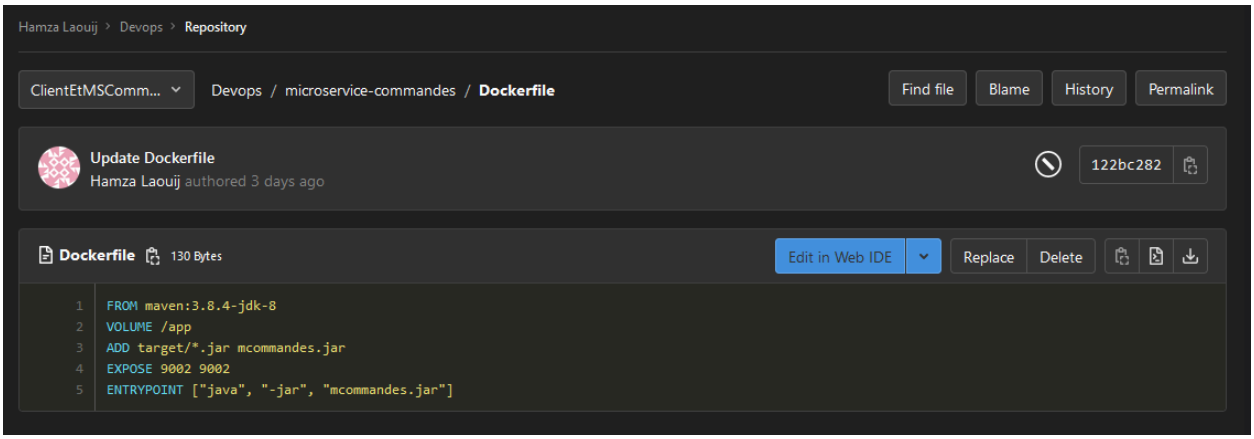
CI_REGISTRY_PASS: password of our Docker Hub account.

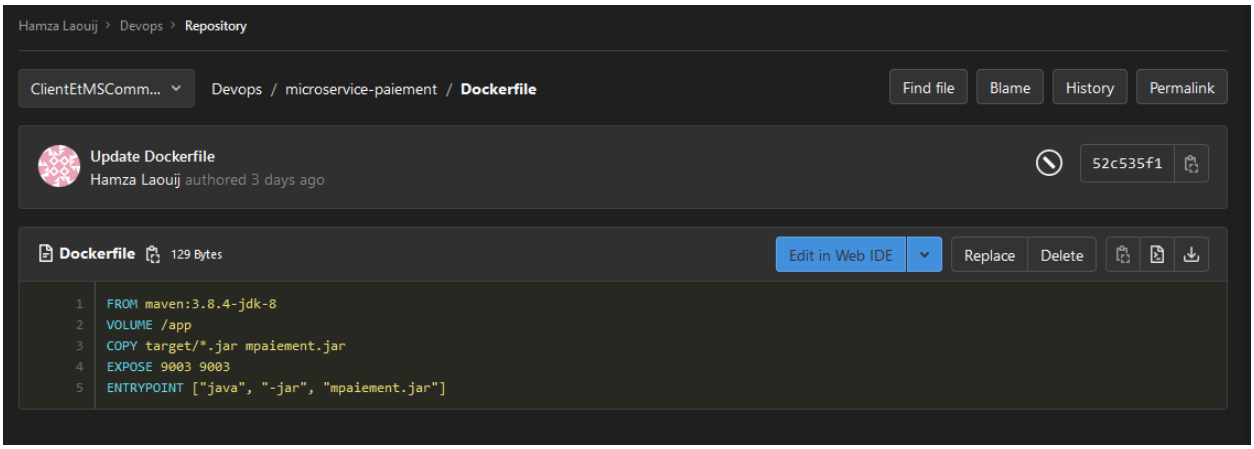
CI_REGISTRY_USER: username of our Docker Hub account.

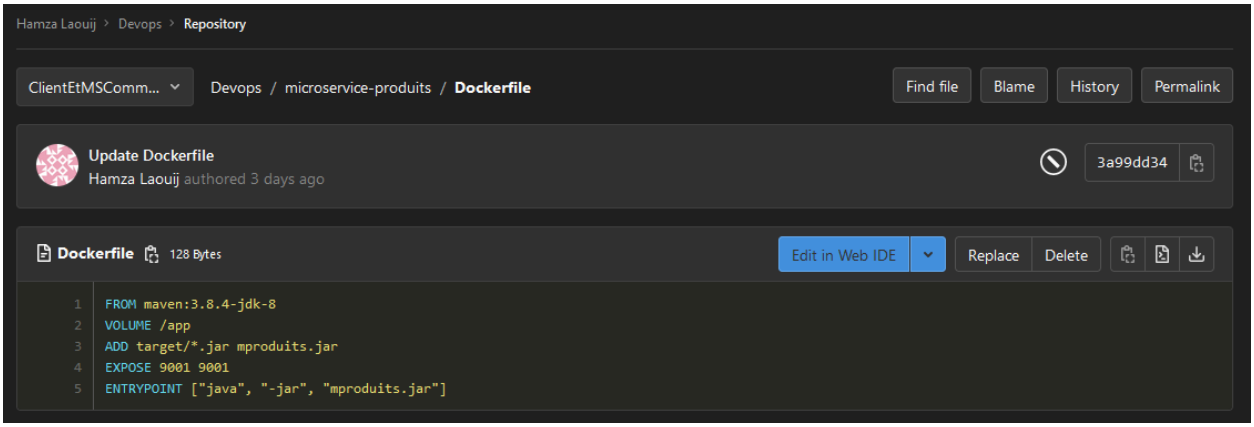


Create Dockerfile for all the services:







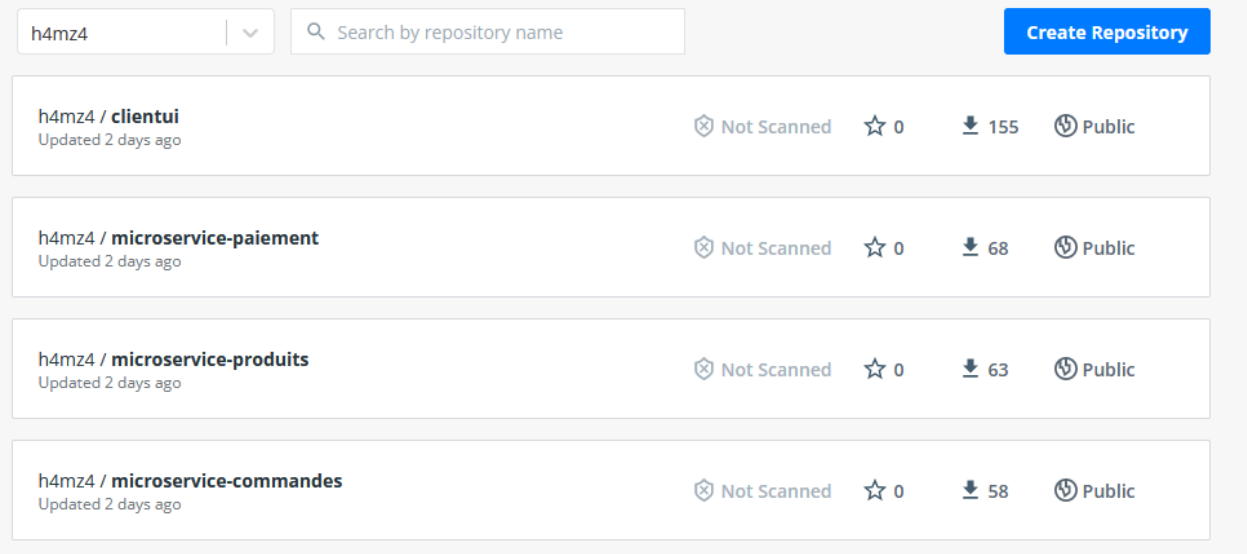


Create the job template:

```
48  .push-image:
49    image: docker:latest
50    stage: Build and Push Docker Image
51    before_script:
52      - echo $CI_REGISTRY_PASS | docker login -u $CI_REGISTRY_USER $CI_REGISTRY --password-stdin
53    script:
54      - echo "Building and Pushing Docker Images"
55      - docker build -t h4mz4/$MODULE ./$MODULE
56      - docker push h4mz4/$MODULE
57
```

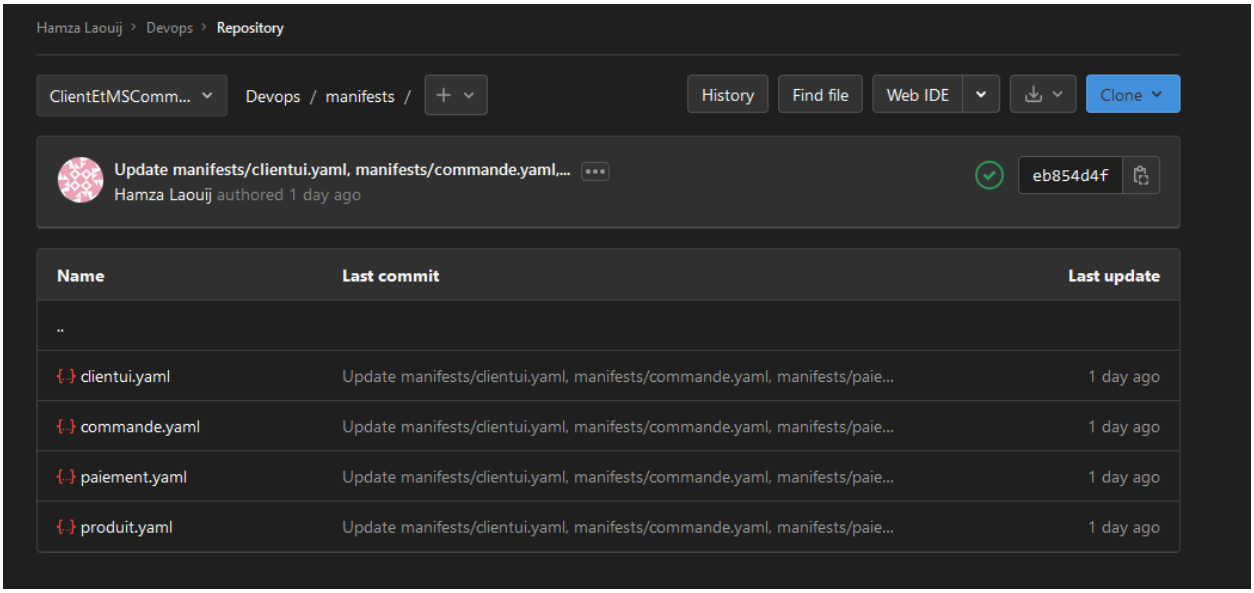
```yaml
# Push Docker Image
push-clientui-image:
  extends:
    - .clientui-module
    - .push-image

push-commande-image:
  extends:
    - .commande-module
    - .push-image

push-paiement-image:
  extends:
    - .paiement-module
    - .push-image
push-produit-image:
  extends:
    - .produit-module
    - .push-image

```

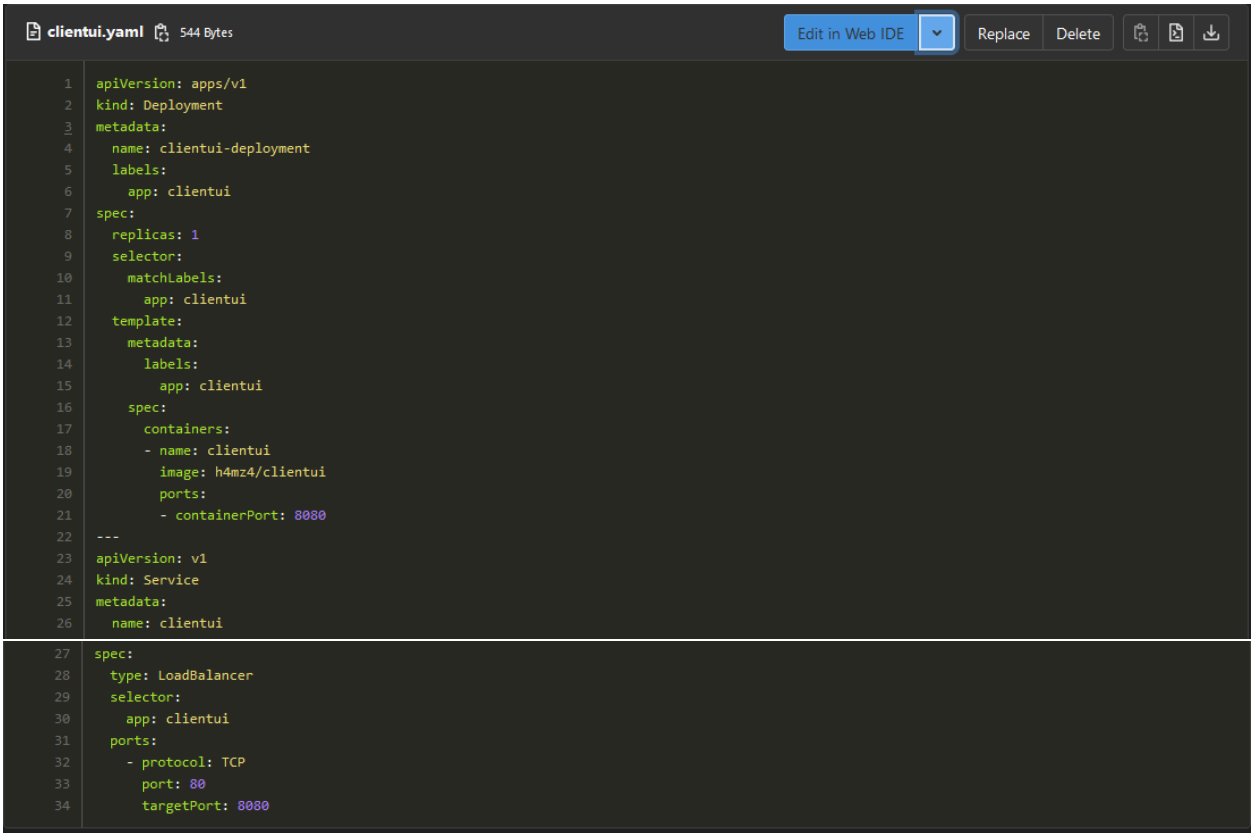
Final Result:



Deploy Stage:

Create Deployment files:





commande.yaml 558 Bytes

Edit in Web IDE | Replace | Delete

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: commande-deployment
  labels:
    app: commande
spec:
  replicas: 1
  selector:
    matchLabels:
      app: commande
  template:
    metadata:
      labels:
        app: commande
    spec:
      containers:
      - name: commande
        image: h4mz4/microservice-commandes
        ports:
        - containerPort: 9002
---
apiVersion: v1
kind: Service
metadata:
  name: commande
spec:
  type: LoadBalancer
  selector:
    app: commande
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9002
```

paiement.yaml 557 Bytes

Edit in Web IDE | Replace | Delete

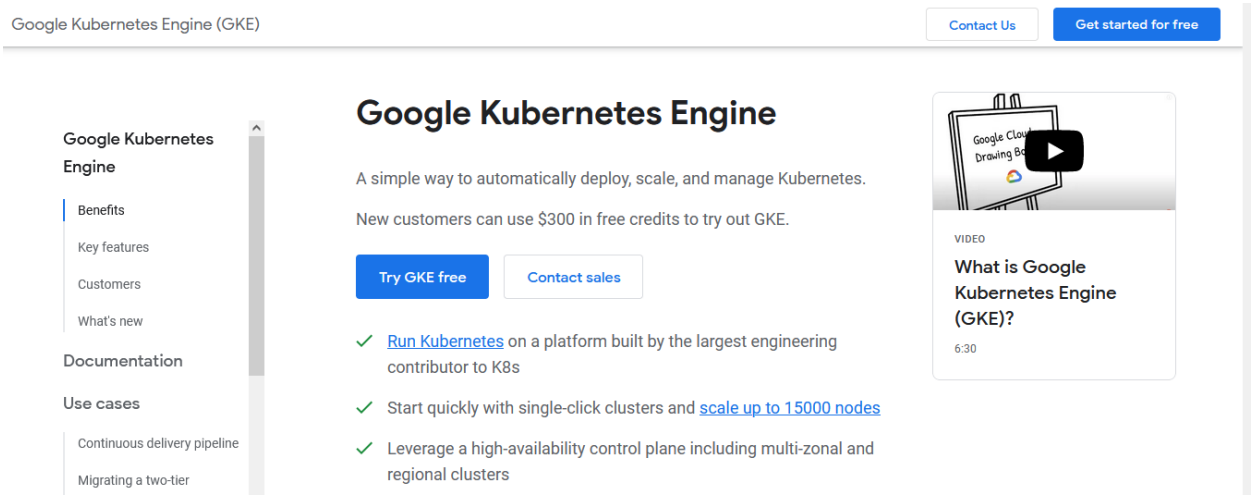
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: paiement-deployment
  labels:
    app: paiement
spec:
  replicas: 1
  selector:
    matchLabels:
      app: paiement
  template:
    metadata:
      labels:
        app: paiement
    spec:
      containers:
      - name: paiement
        image: h4mz4/microservice-paiement
        ports:
        - containerPort: 9003
---
apiVersion: v1
kind: Service
metadata:
  name: paiement
spec:
  type: LoadBalancer
  selector:
    app: paiement
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9003
```

produit.yaml 550 Bytes

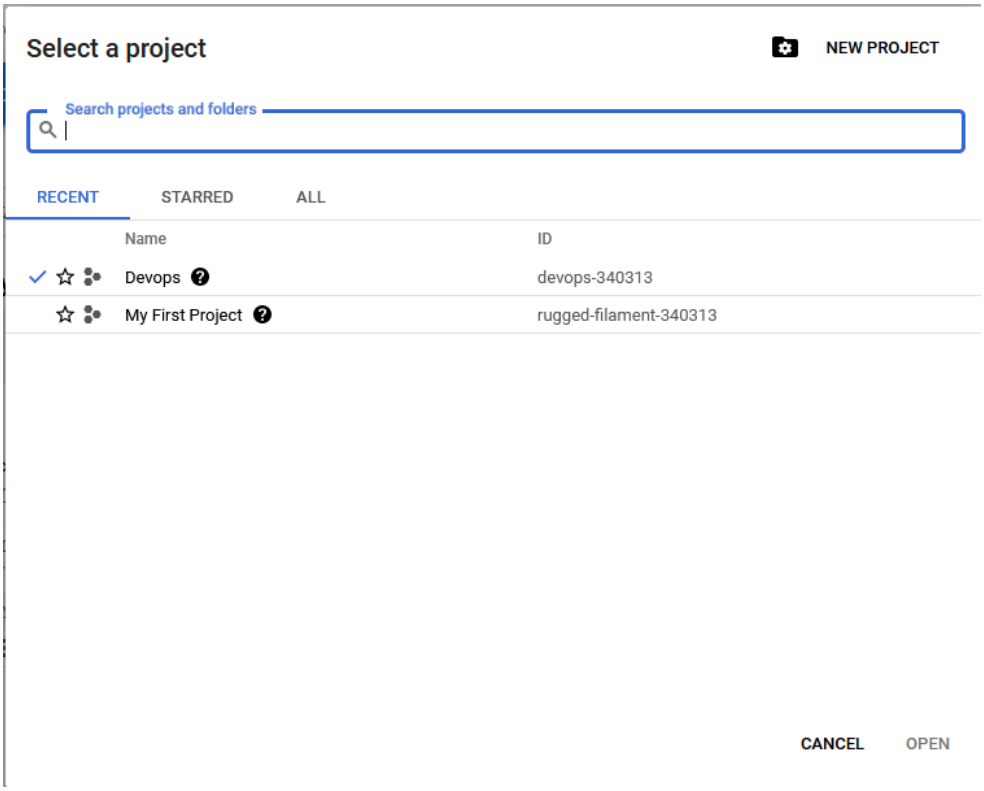
Edit in Web IDE | Replace | Delete

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: produit-deployment
  labels:
    app: produit
spec:
  replicas: 1
  selector:
    matchLabels:
      app: produit
  template:
    metadata:
      labels:
        app: produit
    spec:
      containers:
      - name: produit
        image: h4mz4/microservice-produits
        ports:
        - containerPort: 9001
---
apiVersion: v1
kind: Service
metadata:
  name: produit
spec:
  type: LoadBalancer
  selector:
    app: produit
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9001
```

Create Google Cloud Account:

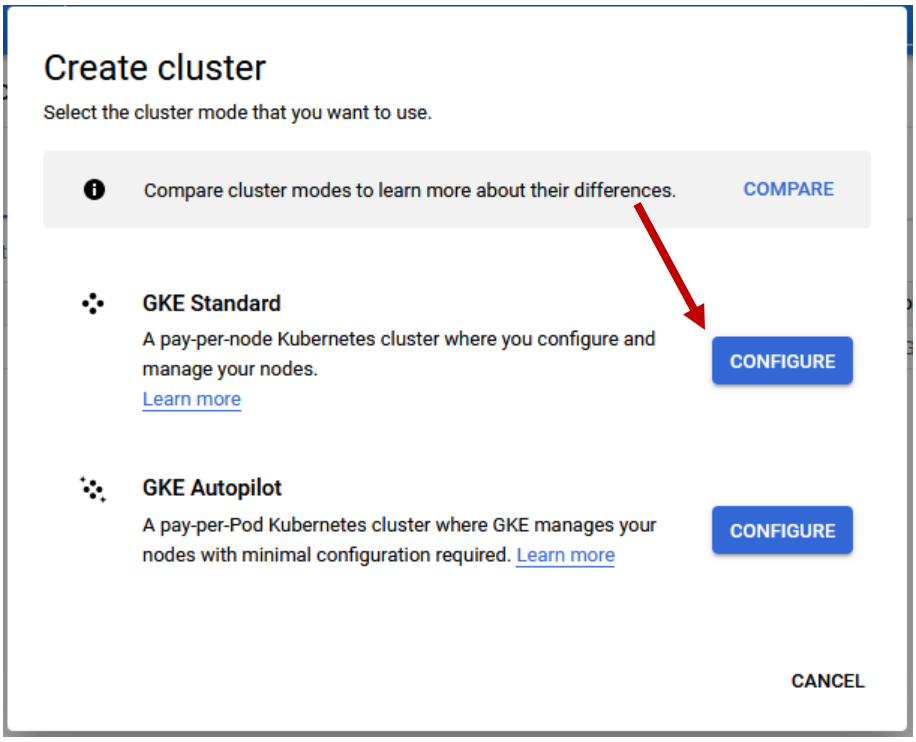


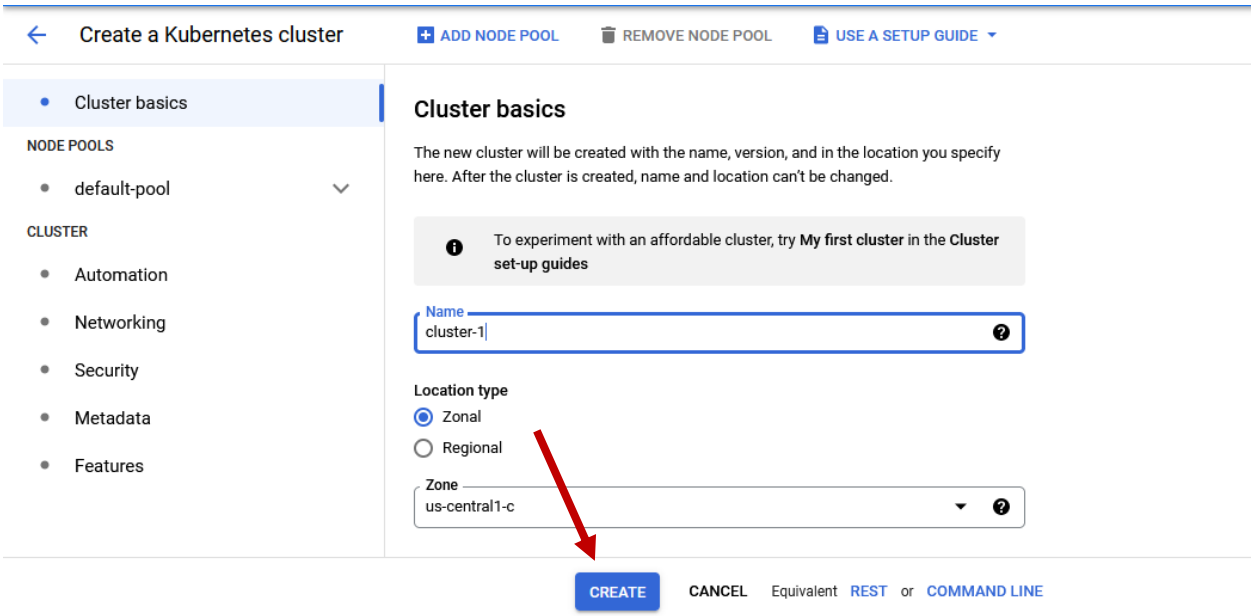
Create GKE Project:

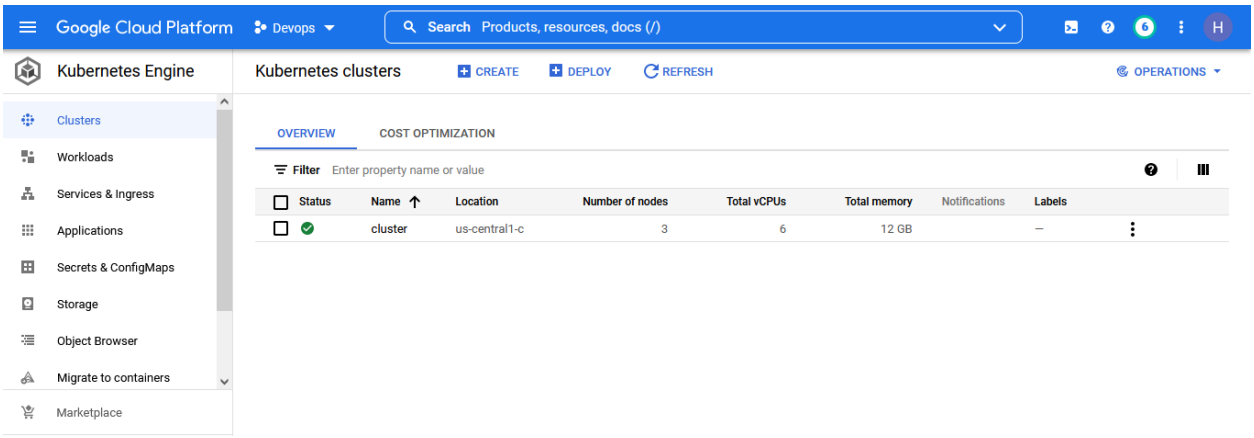


Create GKE Cluster:

We choose GKE Standard:

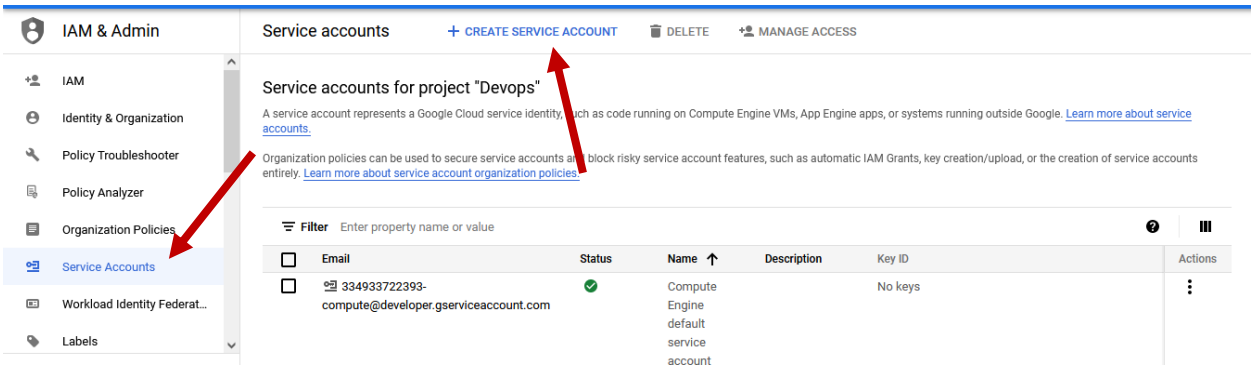


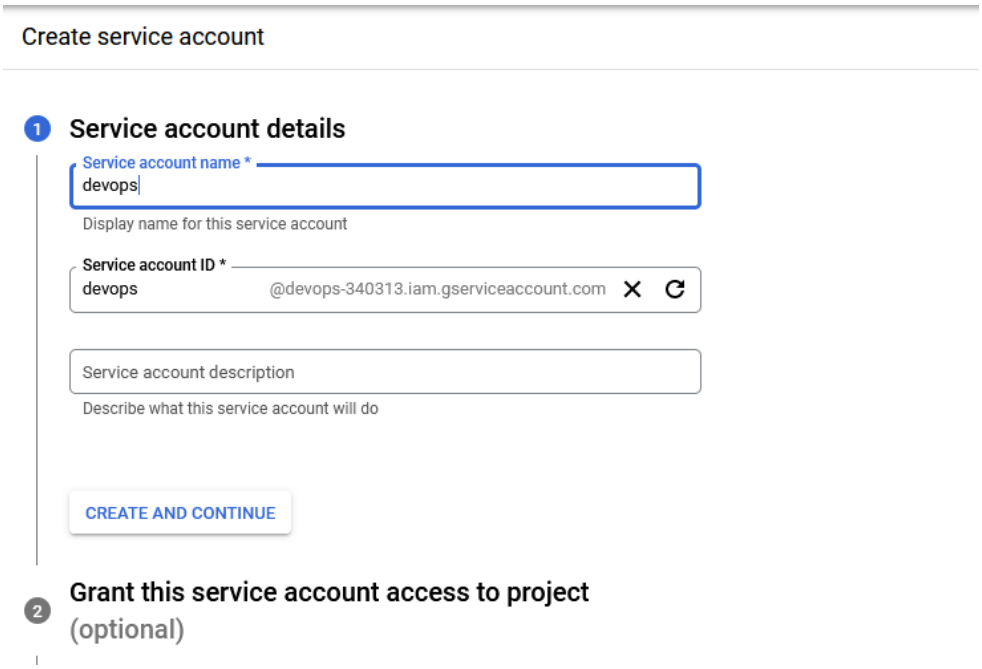




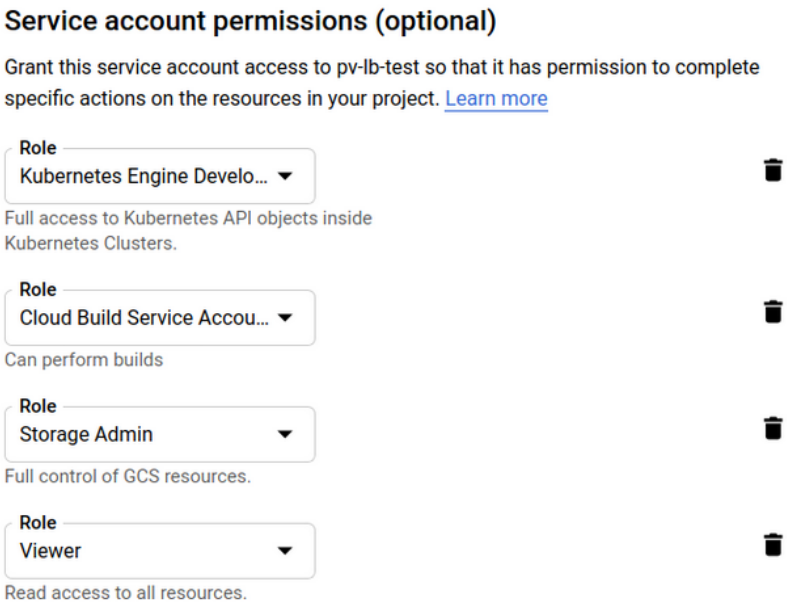
Create Service Account:

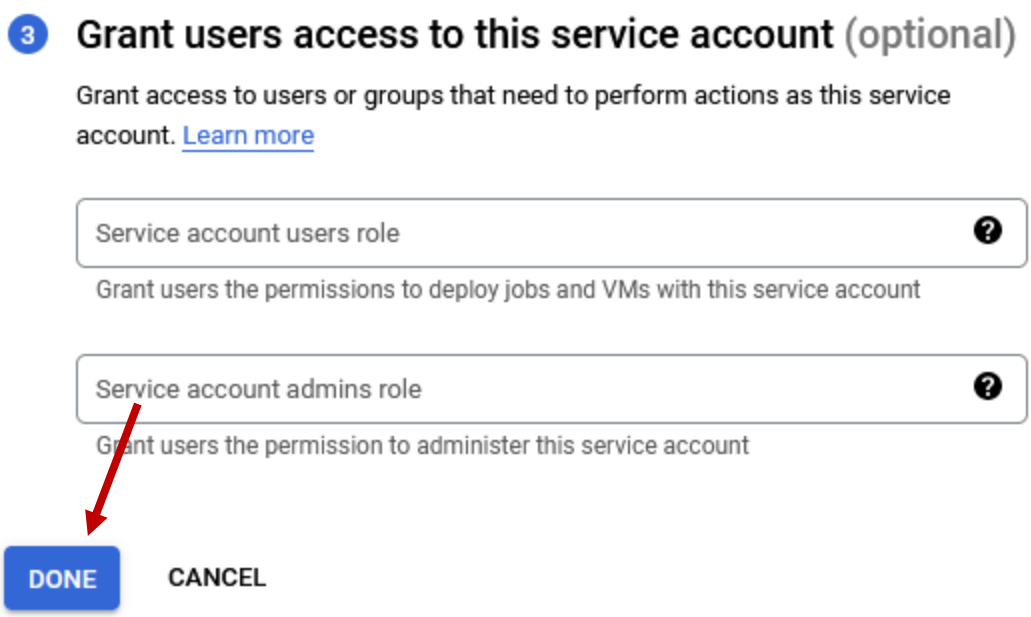
Service Account gives permissions to GitLab to deploy on GKE cluster



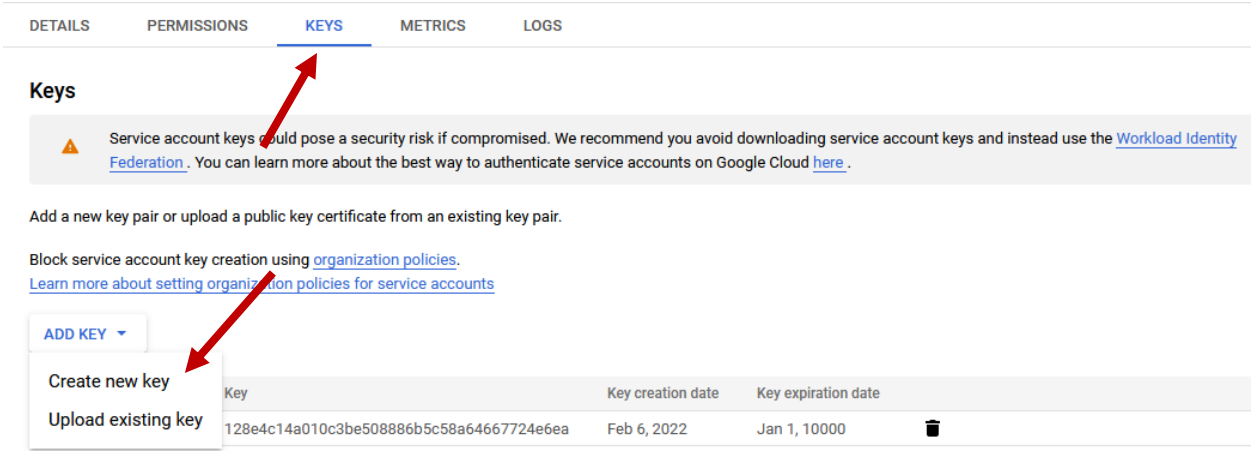


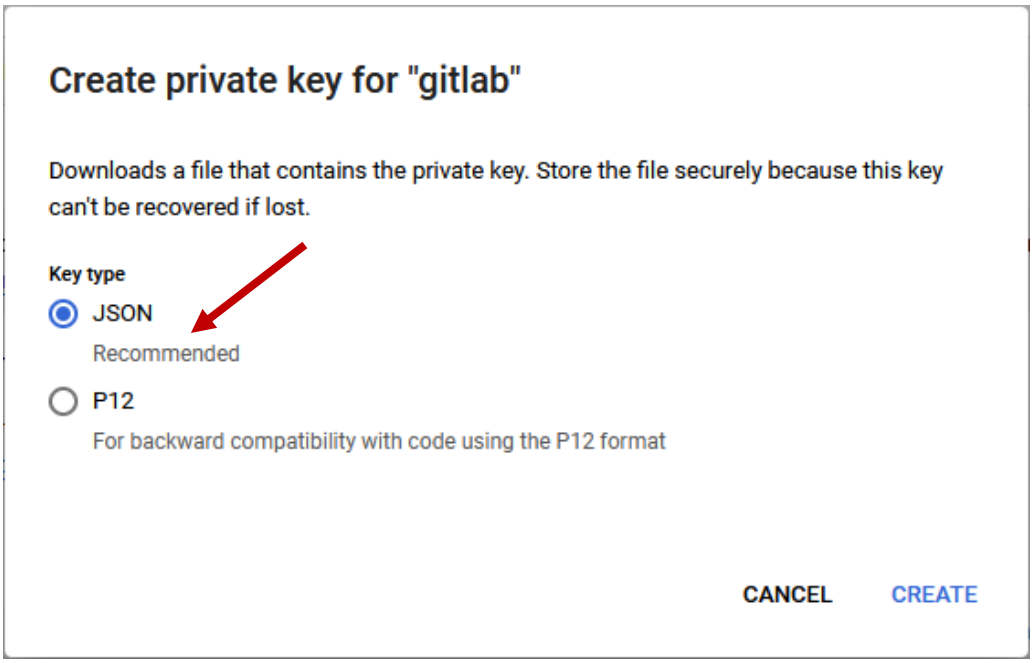
We give it the necessary roles:



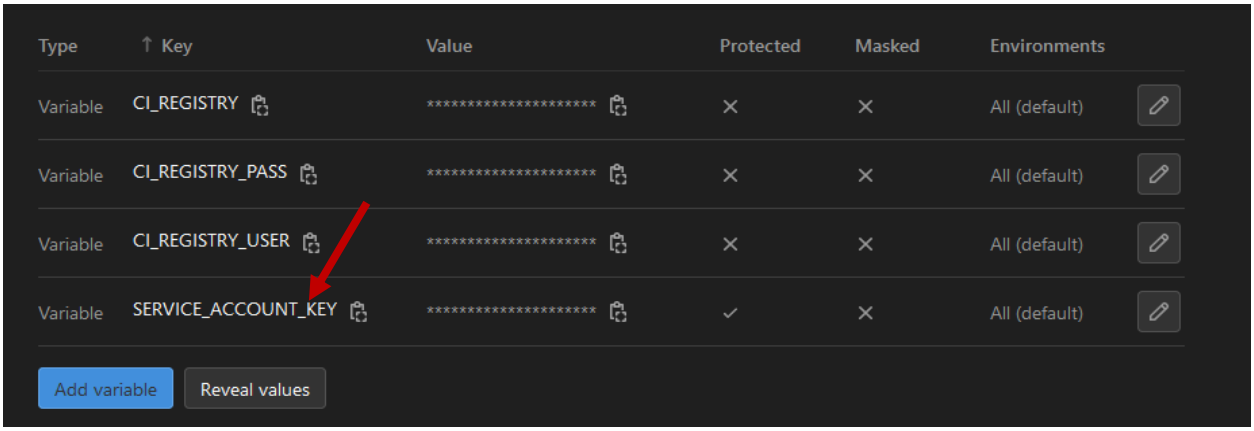


Create JSON Key:





Add JSON Key to Pipeline Variables:



Create Deploy Job:

```
# Deploy
deploy:
  stage: Deploy
  image: google/cloud-sdk
  script:
    - echo "$SERVICE_ACCOUNT_KEY" > key.json
    - gcloud auth activate-service-account --key-file=key.json
    - gcloud config set project devops-340313
    - gcloud config set container/cluster cluster
    - gcloud config set compute/zone us-central1-c
    - gcloud container clusters get-credentials cluster --zone us-central1-c --project devops-340313
    - kubectl apply -f manifests/produit.yaml
    - kubectl apply -f manifests/commande.yaml
    - kubectl apply -f manifests/paiement.yaml
    - kubectl apply -f manifests/clientui.yaml

```

Security Tests

Include security tests templates:

```yaml
include:
  - local: clientui/.gitlab-ci.yml
  - local: microservice-commandes/.gitlab-ci.yml
  - local: microservice-paiement/.gitlab-ci.yml
  - local: microservice-produits/.gitlab-ci.yml
  - template: Security/SAST.gitlab-ci.yml
  - template: Security/Container-Scanning.gitlab-ci.yml
  - template: DAST.gitlab-ci.yml

```

SAST:

SAST is automatically running by adding its template

Container Scanning:

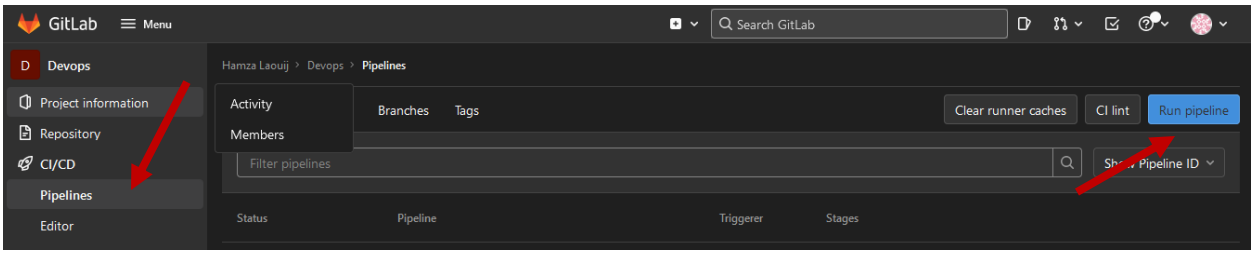
```yaml
# Container Scanning
container_scanning:
  stage: test
  variables:
    DOCKER_IMAGE: h4mz4/clientui

commande_scanning:
  extends:
    - container_scanning
  variables:
    DOCKER_IMAGE: h4mz4/microservice-commandes

paiement_scanning:
  extends:
    - container_scanning
  variables:
    DOCKER_IMAGE: h4mz4/microservice-paiement

produit_scanning:
  extends:
    - container_scanning
  variables:
    DOCKER_IMAGE: h4mz4/microservice-produits
```

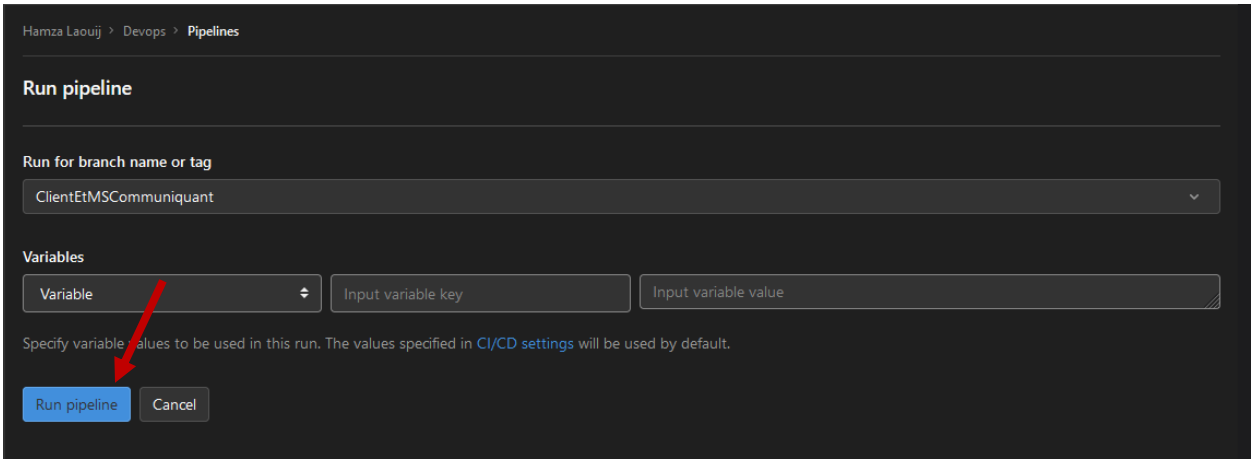
DAST:

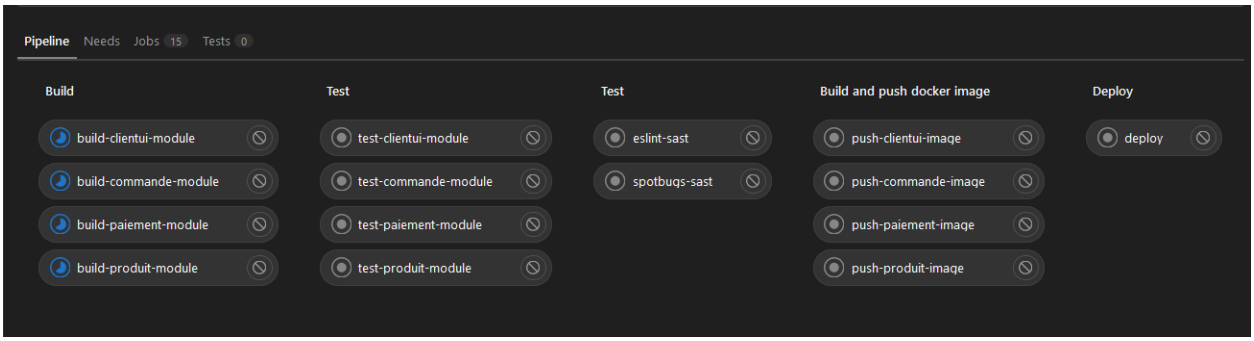
In DAST, we have to pass the website to the DAST_WEBSITE variable

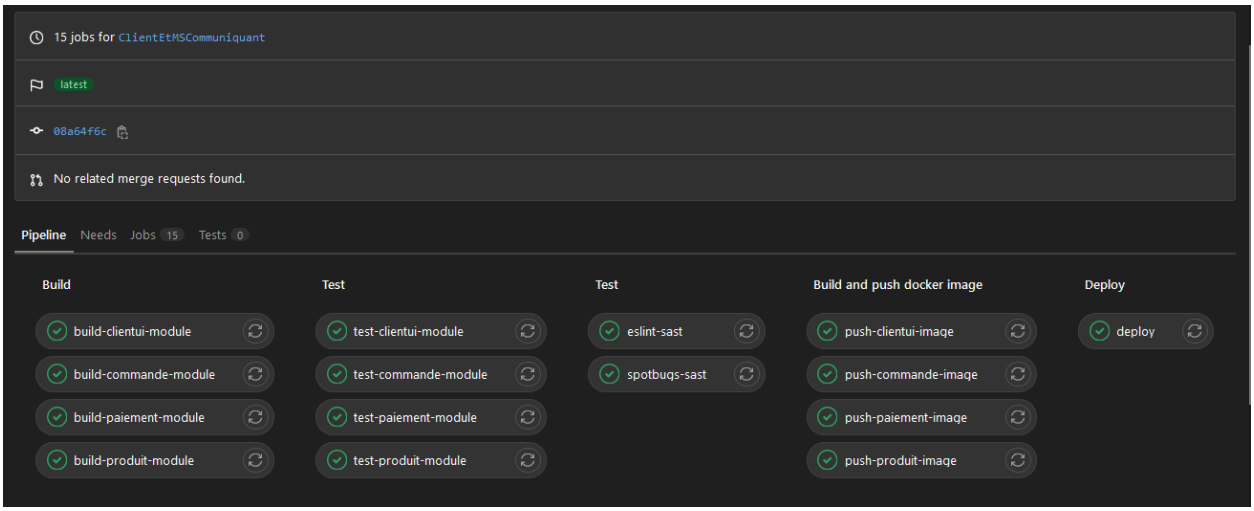
```
162  # DAST
163  dast:
164    variables:
165      DAST_WEBSITE: http://35.232.13.225/
166
```

Let’s Run The Pipeline !

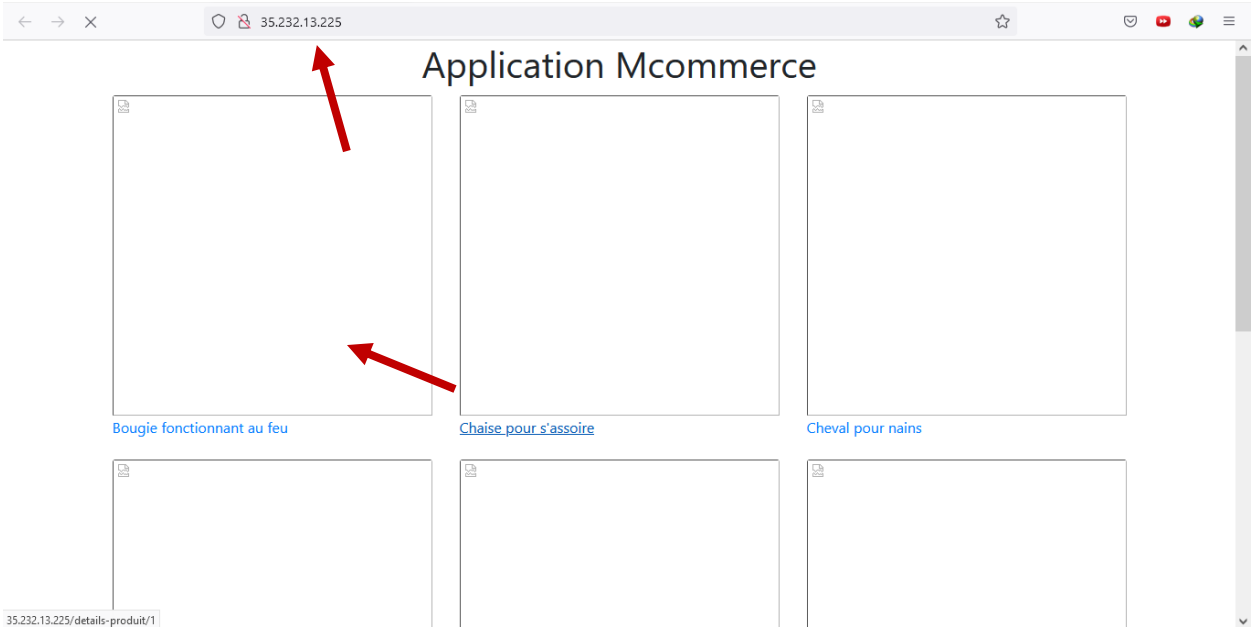








All jobs succeeded and the website is deployed on the url: http://35.232.13.225/



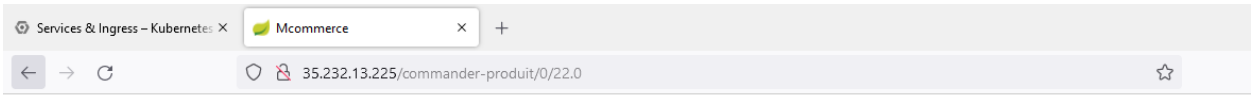
Let's try to click on "Bougie fonctionnant au feu"

Application Mcommerce

Bougie fonctionnant au feu

bougie qui fonctionne comme une ampoule mais sans électricité !

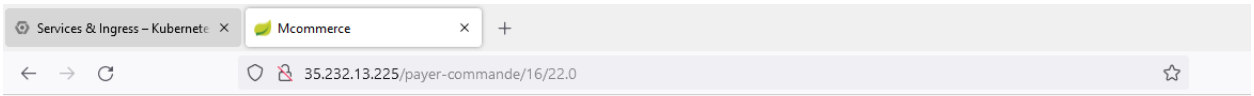
COMMANDER



Application Mcommerce

Ici l'utilisateur sélectionne en temps normal un moyen de paiement et entre les informations de sa carte bancaire.
Nous allons éviter d'ajouter les formulaires nécessaireS afin de garder l'application la plus basique et simple possible pour la suite.
Si vous vous sentez à l'aise, vous pouvez créer un formulaire pour accepter le numéro de la CB, que vous traiterez dans le contrôleur grâce à un *PostMapping*.

Payer Ma Commande



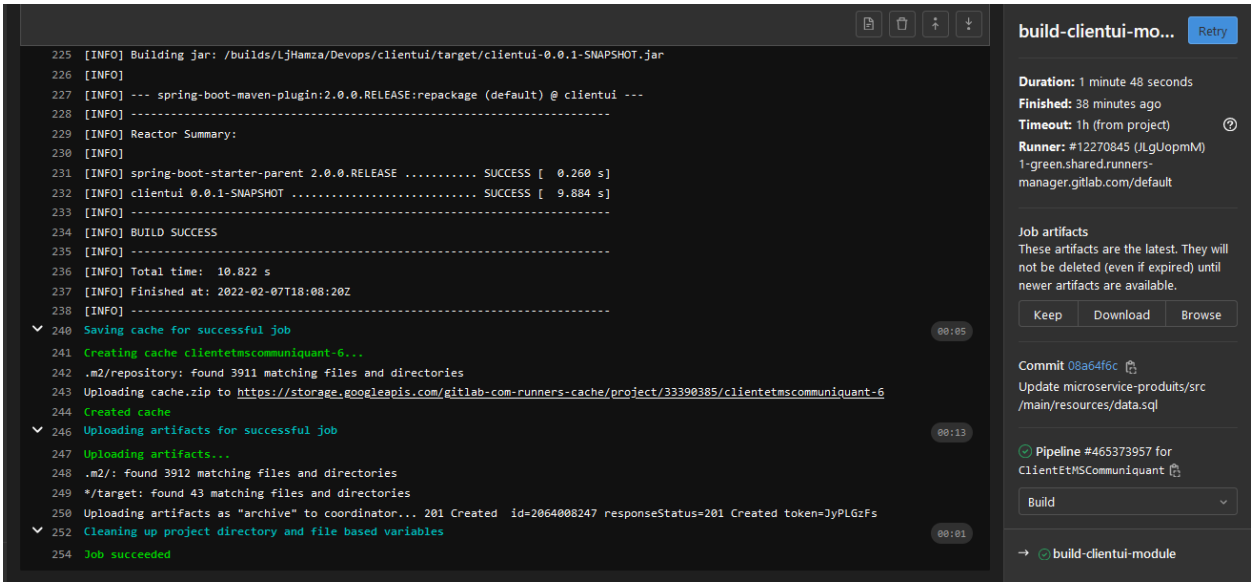
Application Mcommerce

Paiement Accepté

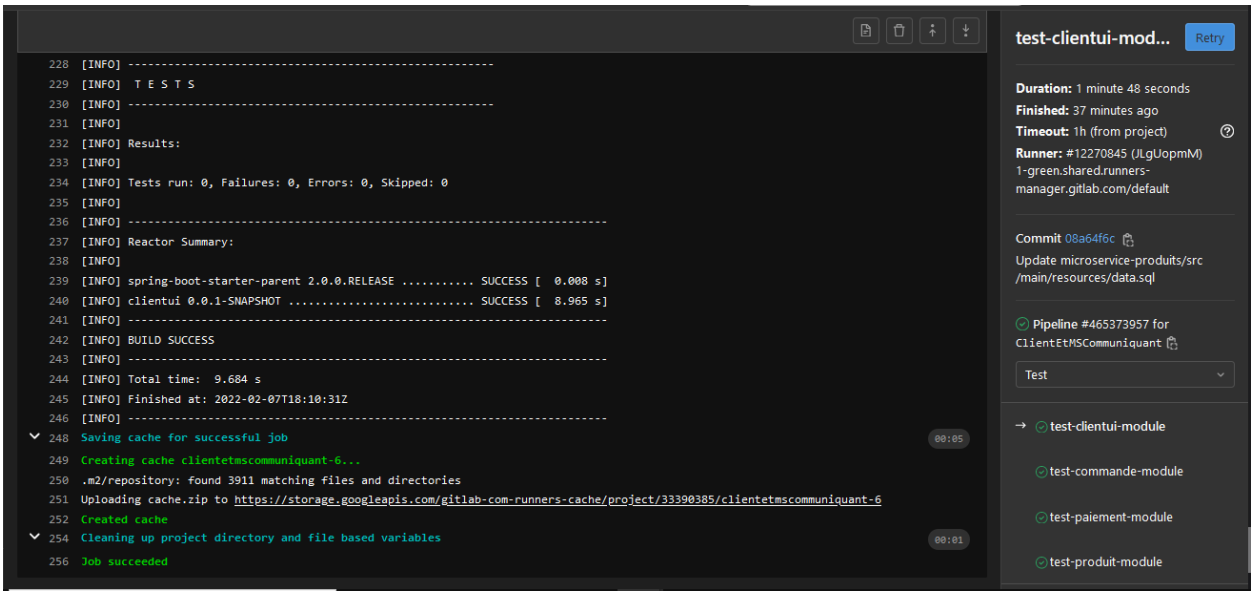
Everything works fine !

Let’s get a closer look at every job:

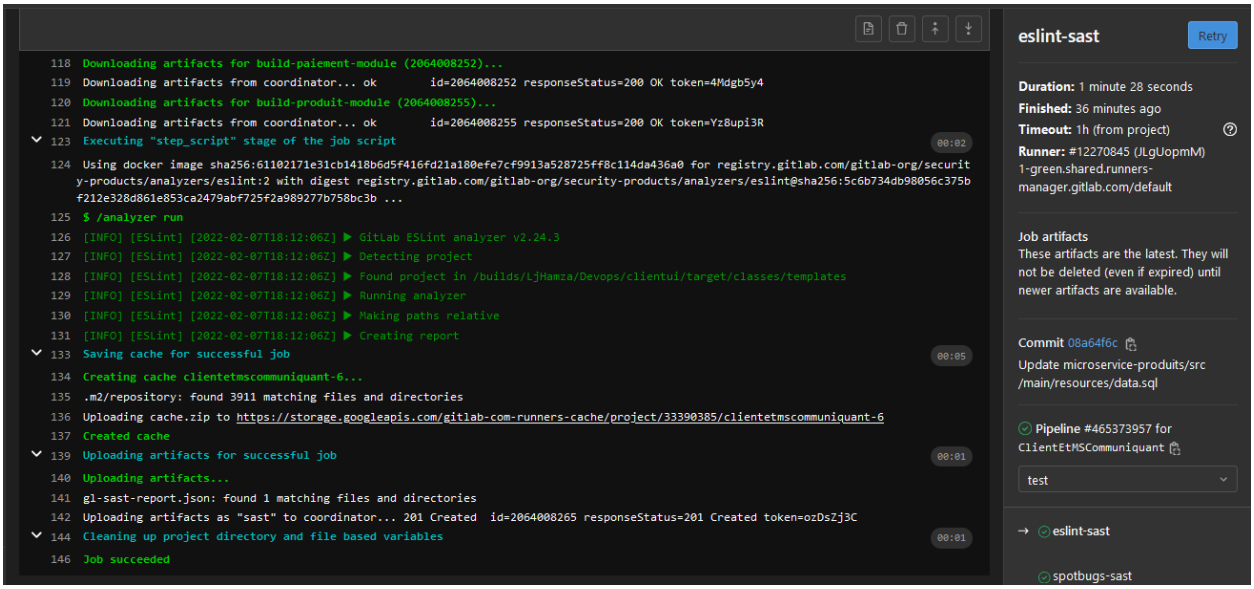
BUILD:



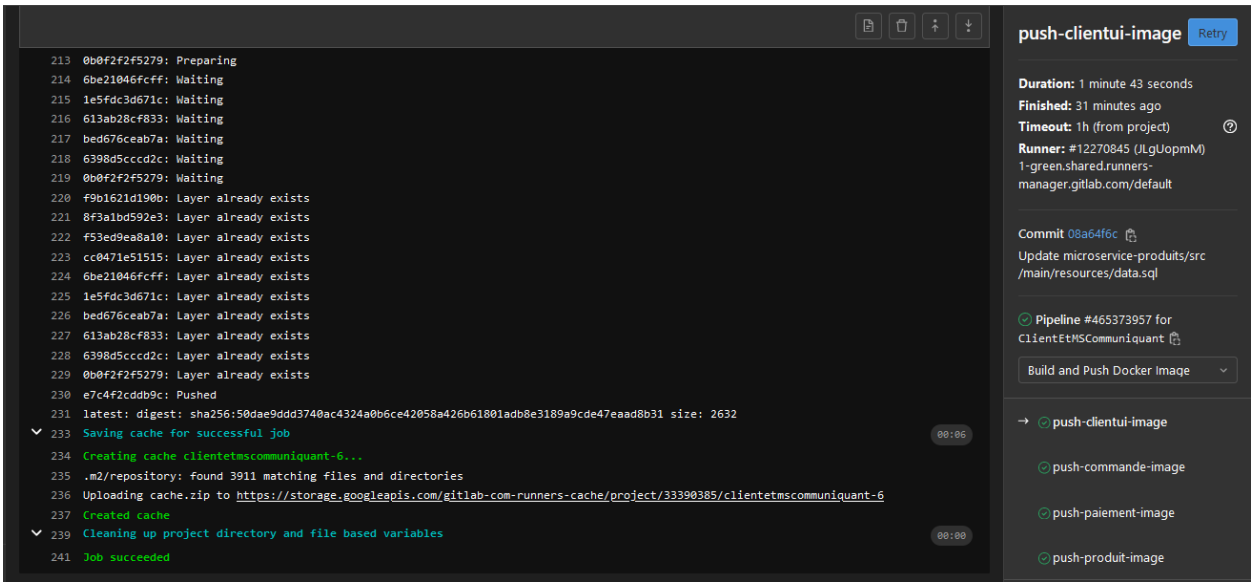
TEST:



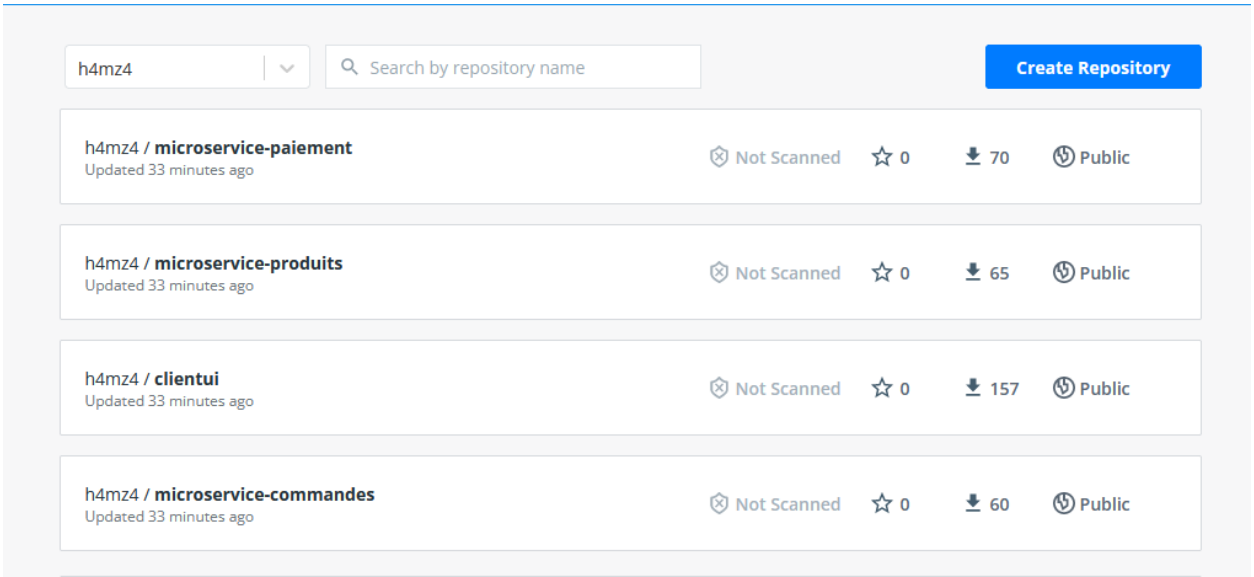
Security TEST:



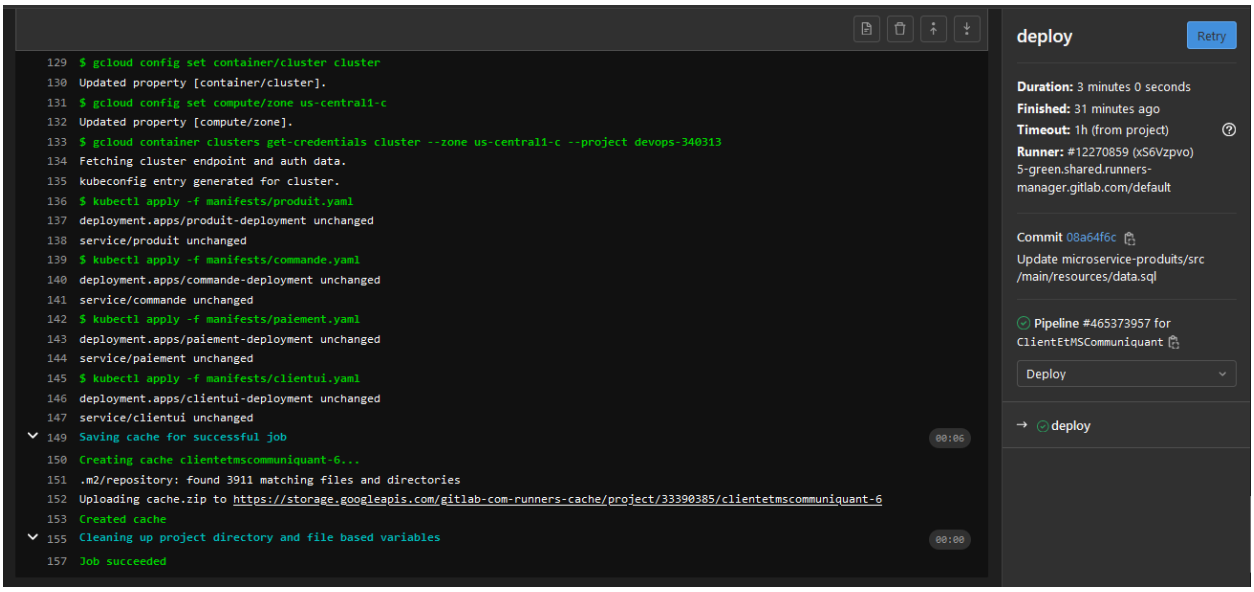
Build And Push Docker Images:



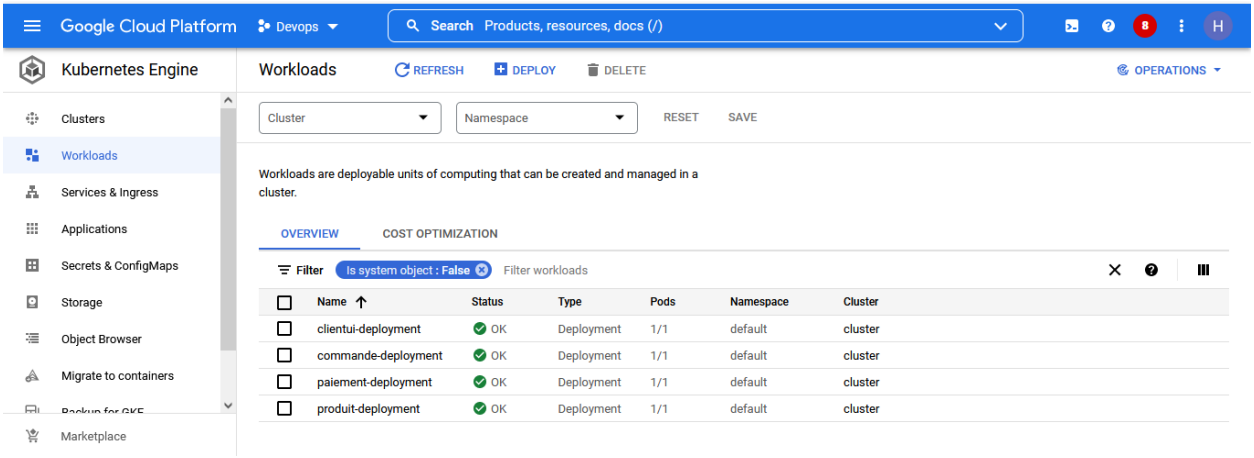
Let’s see Docker Hub:



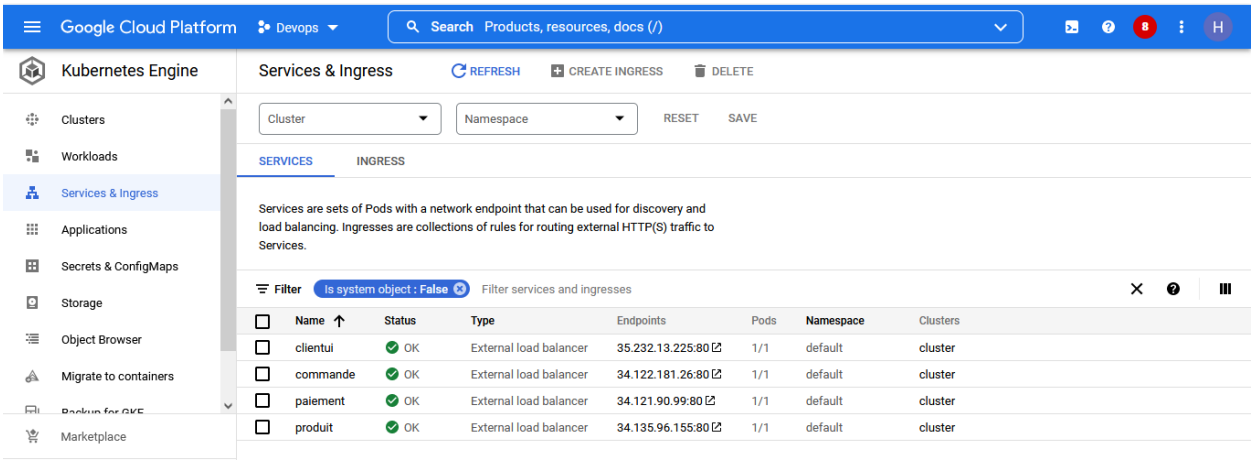
Deploy Job:



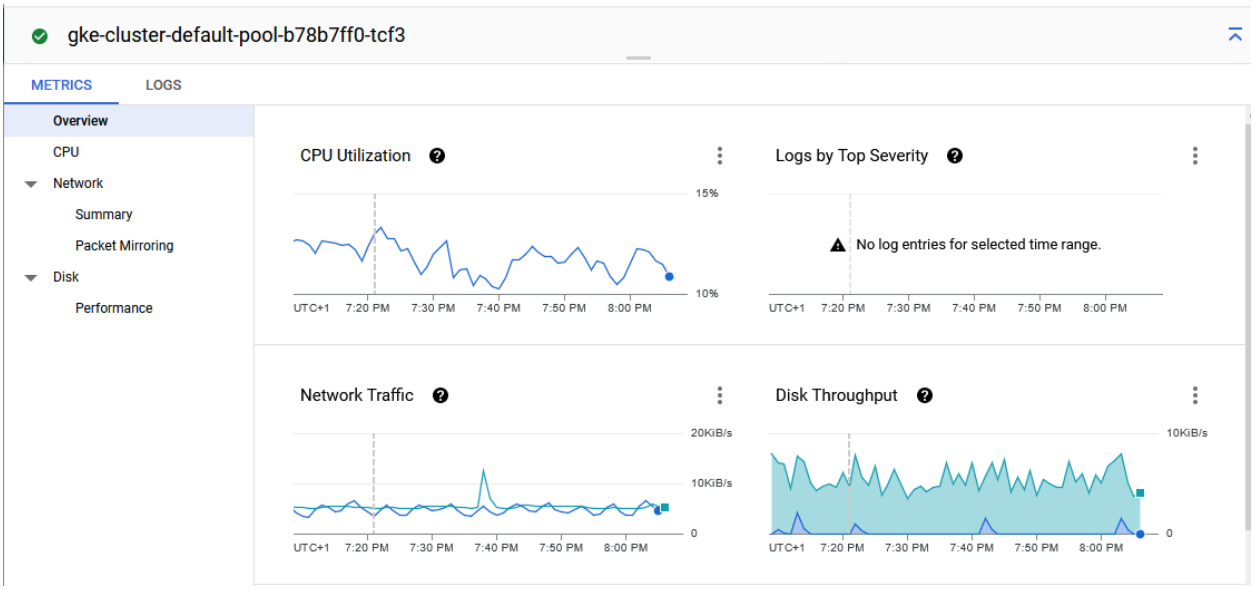
GKE Deployments:

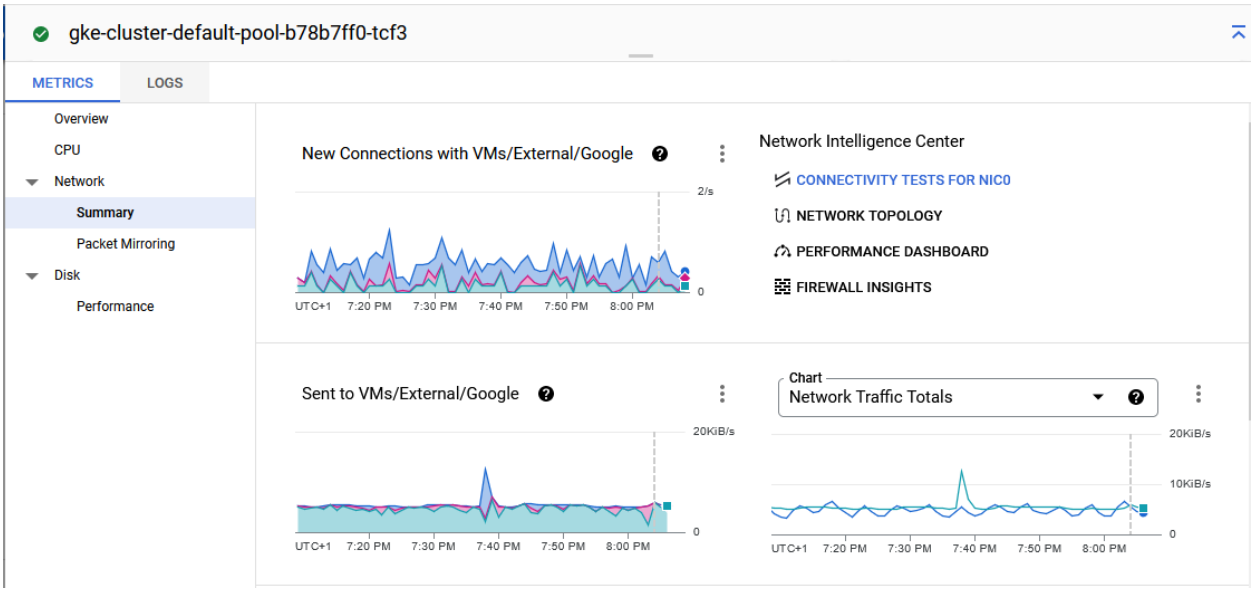


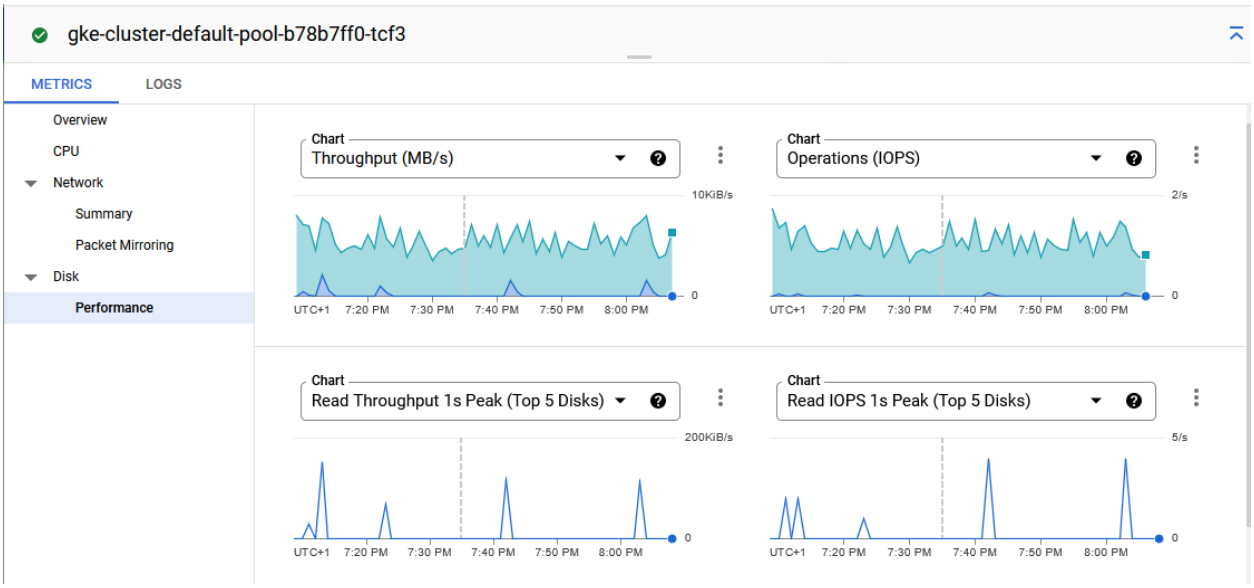
GKE Services:



Monitoring


gke-cluster-default-pool-b78b7ff0-tcf3
METRICS
LOGS
Overview
CPU
Network
Summary
Packet Mirroring
Disk
Performance
CPU Utilization
15%
10%
UTC+1 7:20 PM 7:30 PM 7:40 PM 7:50 PM 8:00 PM
Logs by Top Severity
No log entries for selected time range.
UTC+1 7:20 PM 7:30 PM 7:40 PM 7:50 PM 8:00 PM
Network Traffic
20KiB/s
10KiB/s
0
UTC+1 7:20 PM 7:30 PM 7:40 PM 7:50 PM 8:00 PM
Disk Throughput
10KiB/s
0
UTC+1 7:20 PM 7:30 PM 7:40 PM 7:50 PM 8:00 PM


gke-cluster-default-pool-b78b7ff0-tcf3
METRICS
LOGS
Overview
CPU
Network
Summary
Packet Mirroring
Disk
Performance
New Connections with VMs/External/Google
2/s
0
UTC+1 7:20 PM 7:30 PM 7:40 PM 7:50 PM 8:00 PM
Network Intelligence Center
CONNECTIVITY TESTS FOR NIC0
NETWORK TOPOLOGY
PERFORMANCE DASHBOARD
FIREWALL INSIGHTS
Sent to VMs/External/Google
20KiB/s
0
UTC+1 7:20 PM 7:30 PM 7:40 PM 7:50 PM 8:00 PM
Chart
Network Traffic Totals
20KiB/s
10KiB/s
0
UTC+1 7:20 PM 7:30 PM 7:40 PM 7:50 PM 8:00 PM


gke-cluster-default-pool-b78b7ff0-tcf3
METRICS
LOGS
Overview
CPU
Network
Summary
Packet Mirroring
Disk
Performance
Chart
Throughput (MB/s)
10KiB/s
0
UTC+1 7:20 PM 7:30 PM 7:40 PM 7:50 PM 8:00 PM
Chart
Operations (IOPS)
2/s
0
UTC+1 7:20 PM 7:30 PM 7:40 PM 7:50 PM 8:00 PM
Chart
Read Throughput 1s Peak (Top 5 Disks)
200KiB/s
0
UTC+1 7:20 PM 7:30 PM 7:40 PM 7:50 PM 8:00 PM
Chart
Read IOPS 1s Peak (Top 5 Disks)
5/s
0
UTC+1 7:20 PM 7:30 PM 7:40 PM 7:50 PM 8:00 PM

Project Resources

GitLab Project: https://gitlab.com/LjHamza/Devops/

Docker Hub: https://hub.docker.com/u/h4mz4

APP Website: http://35.232.13.225/