

TRAIL

TRUSTED AI LABS

TRAIL FACTORY

PRÉSENTATION DE LA TRAIL FACTORY — 28 AVRIL 2026

Raphaël Michel · raphael.michel@cetic.be
Guillaume Ginis · guillaume.ginis@cetic.be
Xavier Lessage · xavier.lessage@cetic.be



Plan

01 Concept & éléments clés

02 GitLab / OpenShift

03 Brique « LLMs White Hat Hacker »

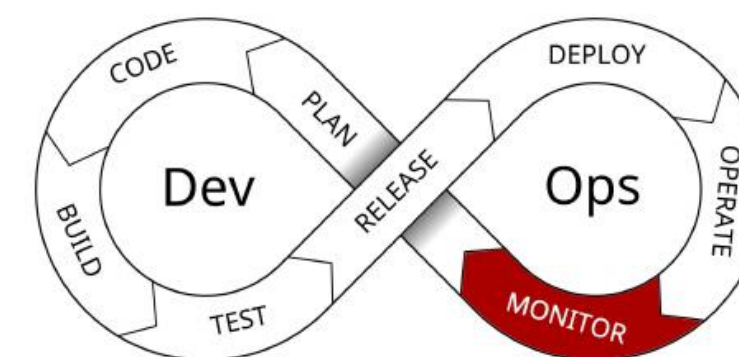
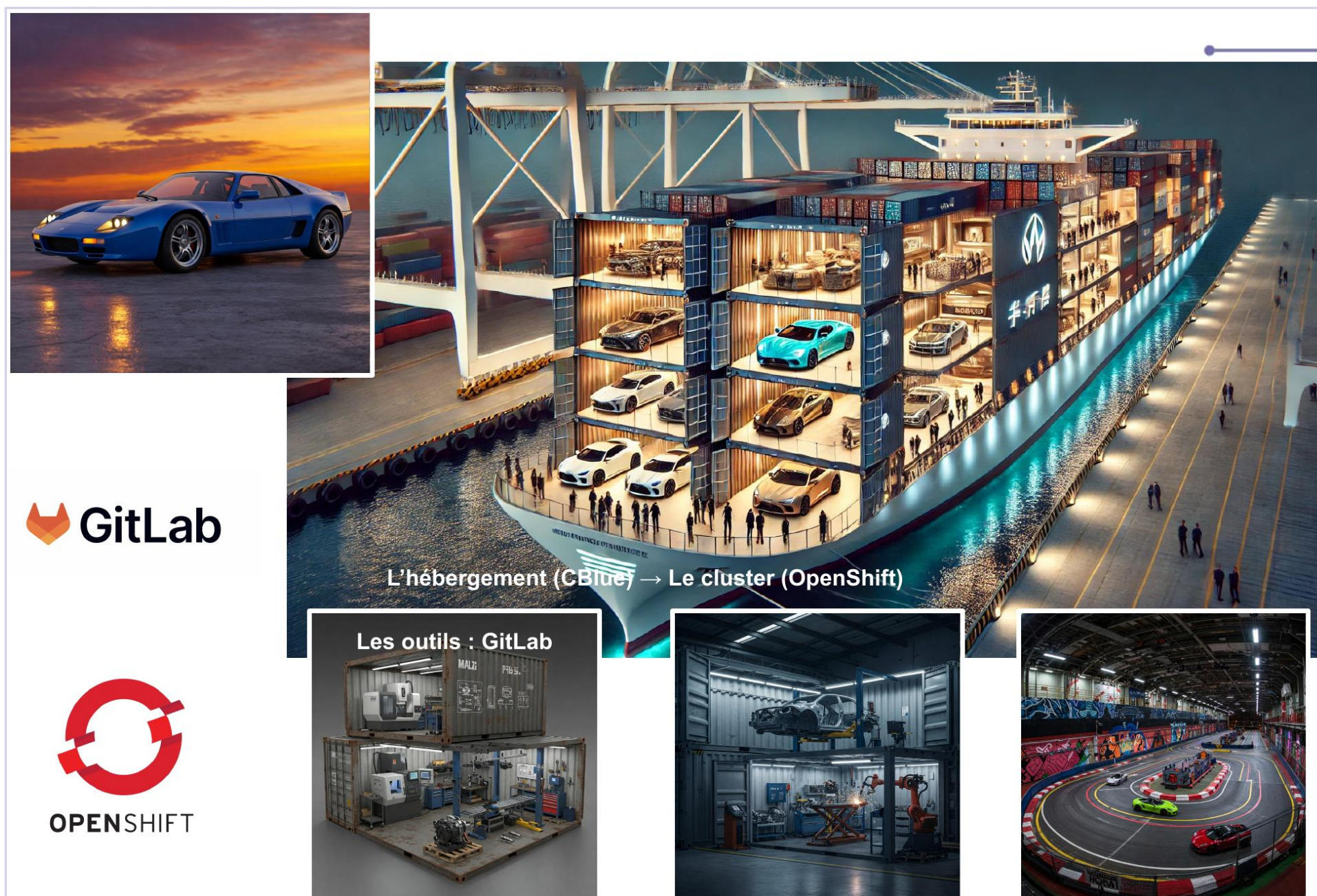
04 Q & R

05 Prise en main : installation d'une application web

- Validation sur le PC
- Intégration GitLab
- Création du projet dans OpenShift
- Activation de la CI/CD

01

Trail Factory | Introduction



L'hébergement

CBlue → le cluster OpenShift

Les outils

GitLab

TRAIL
TRUSTED AI LABS

| 02

GitLab | Points clés



Accès

<https://gitlab.apps.factory.trail.ac>

Demande d'accès : Laurent Gravez (laurent.gravez@cetic.be)

I 02

GitLab | Points clés

Coder, collaborer, tester et déployer au même endroit

Gestion du code source (Git)

- Versionnement complet du code (Git)
- Historique : qui a fait quoi, quand
- Retour possible à une version précédente
- Travail en branches (feature, bugfix, etc.)

Collaboration en équipe

- Plusieurs développeurs en parallèle
- Merge Requests (MR) pour proposer des changements
- Code review intégrée (commentaires, suggestions)
- Gestion des conflits simplifiée

CI/CD intégré

- Pipelines automatiques avec GitLab CI/CD
- Tests lancés automatiquement à chaque modification
- Déploiement automatique (site web, app, etc.)
- Gain de temps et moins d'erreurs humaines

I 02

GitLab | Points clés

Coder, collaborer, tester et déployer au même endroit

Gestion de projets

- Issues (tickets) pour suivre les tâches
- Tableaux de type Kanban
- Milestones pour organiser les versions
- Assignment des tâches aux membres

Sécurité

- Gestion des permissions (qui peut voir ou modifier quoi)
- Analyse automatique des failles de sécurité
- Gestion des clés SSH et accès sécurisés

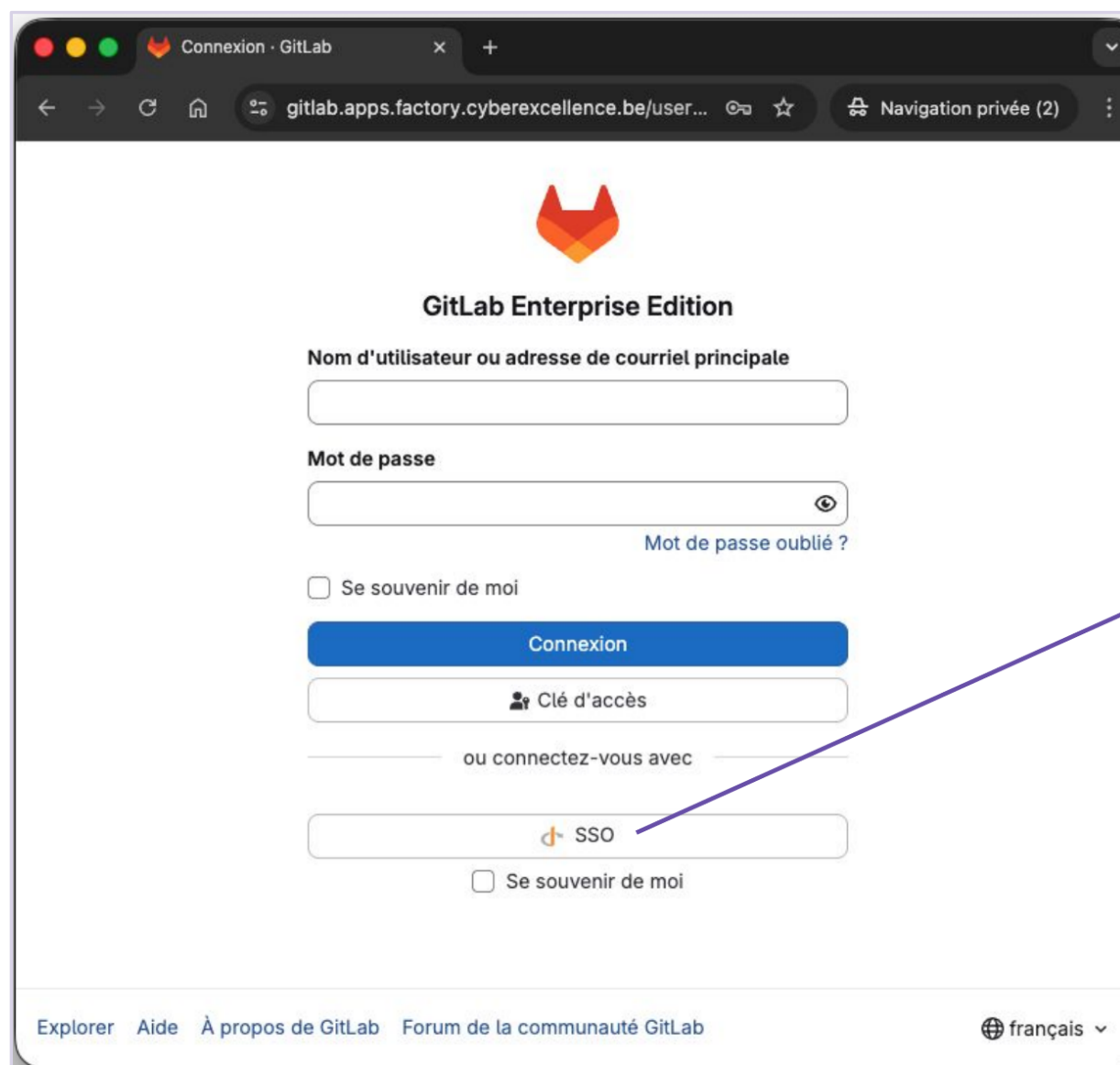
Centralisation

- Code, documentation, pipelines et issues au même endroit
- Évite de multiplier les outils

02

GitLab | Connexion

<https://gitlab.apps.factory.trail.ac>



GitLab Enterprise Edition

Nom d'utilisateur ou adresse de courriel principale

Mot de passe

☐ Se souvenir de moi

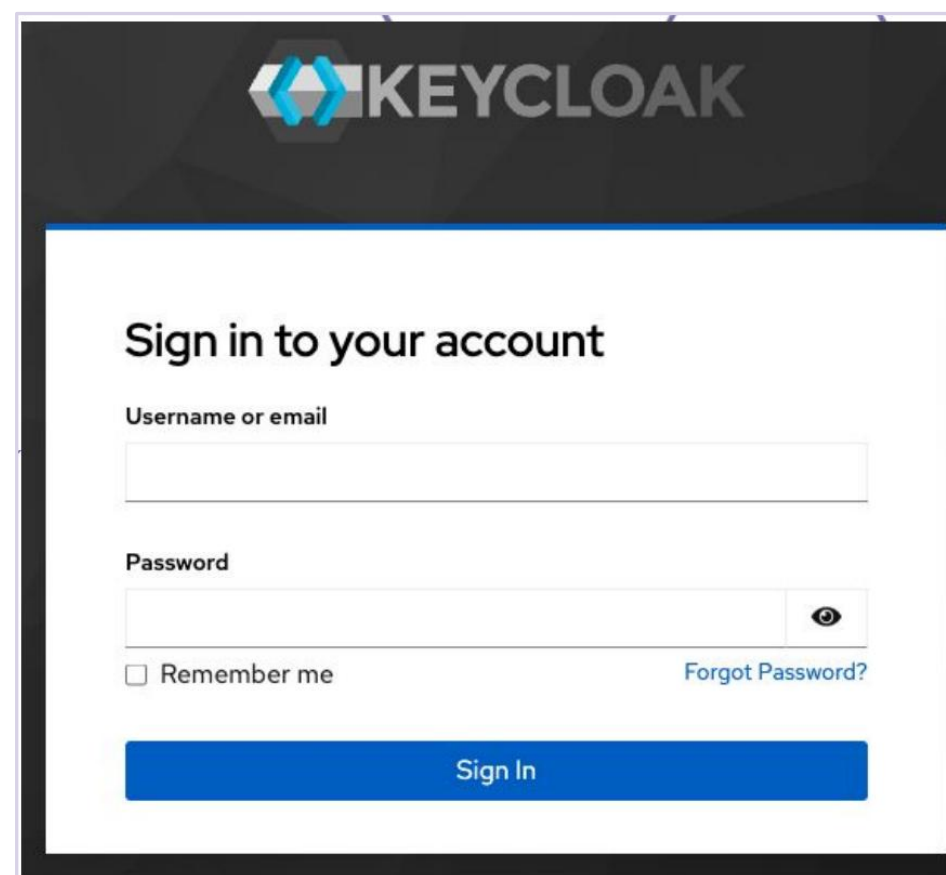
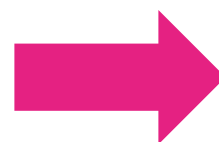
Connexion

Clé d'accès

ou connectez-vous avec

SSO

☐ Se souvenir de moi



KEYCLOAK

Sign in to your account

Username or email

Password

☐ Remember me

Forgot Password?

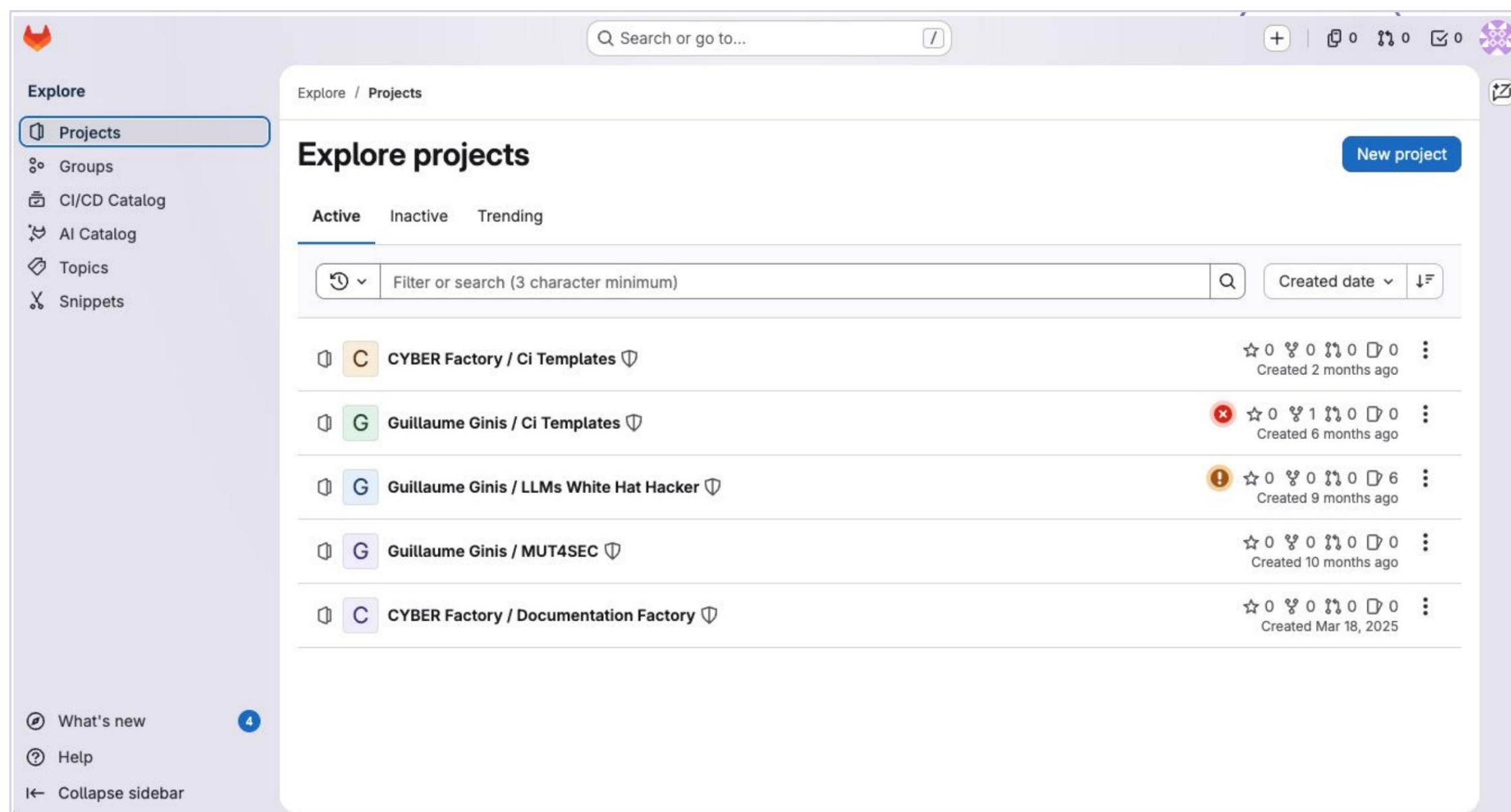
Sign In

Single Sign-On : gestion centralisée des utilisateurs pour tous les outils de la factory

02

GitLab | Projets

<https://gitlab.apps.factory.trail.ac>



02

GitLab | Templates

<https://gitlab.apps.factory.trail.ac>

The screenshot shows the GitLab web interface for the 'CYBER Factory / CI Templates' project. The left sidebar contains a 'Project' section with 'Ci Templates' selected, and a 'Manage' section with options like Plan, Code, Build, Deploy, Operate, Monitor, and Analyze. The main content area is divided into three panes. The left pane shows a 'Files' view with a search bar and a list of folders: 'docker-jobs', 'nexus-jobs', 'other-jobs', 'python-jobs', 'quality-image', 'ruby-jobs', and 'web-jobs'. Below the folders are two files: '.gitlab-ci.yml' (highlighted) and 'README.md'. The right pane shows the content of the '.gitlab-ci.yml' file, which is a YAML configuration for a CI pipeline. The pipeline includes a 'workflow' section with a 'rules' block that defines a conditional rule for Ruby-based images. It also includes a 'update_owasp_database' job that runs on a scheduled trigger and a 'dockerfile_lint' job that runs on a 'lint' stage.

```
20 - build
21 - test
22 - push
23
24 variables:
25   IMAGE_NAME: quality-assurance-image-python
26   DOCKERFILE_PATH: ./quality-image/Dockerfile.python
27
28 workflow:
29   rules:
30     - if: '$RUBY == "1"' # Conditional rule to make a Ruby based image
31       variables:
32         IMAGE_NAME: "quality-assurance-image-ruby"
33         DOCKERFILE_PATH: ./quality-image/Dockerfile.ruby
34     - when: always
35
36 # To save time when running the tests, the OWASP dependency-check tool must be installed on the runner's machine.
37 # on the machine means that the tool database needs to be updated regularly.
38 # The owasp_update.tpl.yml job called every day by the pipeline takes care of this on the runner's machine (s
39
40 update_owasp_database: # Scheduled job to update the security database on the runner's machine
41   stage: scheduled
42   extends: .owasp_update
43   only:
44     - schedules # Automatic trigger every day at 1 p.m
45
46 dockerfile_lint: # Use Hadolint job to lint the Dockerfile
47   stage: lint
48   extends: .hadolint
49   rules:
```

| 02

GitLab | CI/CD

Automatiser tout le cycle de vie du code : du test jusqu'au déploiement

CI — Continuous Integration

- Intégration continue
- Tester automatiquement le code à chaque modification

CD — Continuous Delivery / Deployment

- Livraison / déploiement continu
- Déployer automatiquement l'application

`.gitlab-ci.yml` décrit les étapes automatiques à exécuter. Exemple simple :

```
stages:
  - build
  - test
  - deploy

build_job:
  stage: build
  script:
    - echo "Build en cours"
test_job:
  stage: test
  script:
    - echo "Tests en cours"
deploy_job:
  stage: deploy
  script:
    - echo "Déploiement en cours"
```

| 02

GitLab | CI/CD

Automatiser tout le cycle de vie du code : du test jusqu'au déploiement

Pipelines (suite d'étapes automatiques)

- Se lance à chaque push et à chaque merge request
- Contient plusieurs jobs
- Exemple : Push → Build → Test → Deploy

Runners (exécutent les jobs)

- Machine locale, serveur ou cloud
- Peuvent utiliser Docker
- Exécutent les scripts définis dans le .gitlab-ci.yml

Variables et sécurité

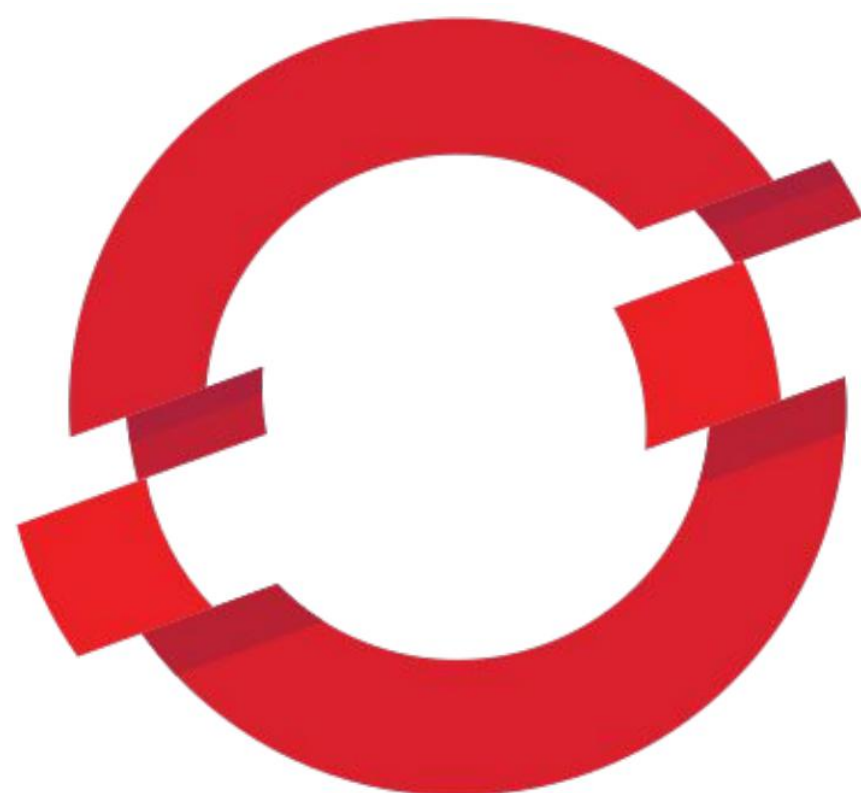
- Variables d'environnement (tokens, mots de passe)
- Stockées de façon sécurisée
- Utilisées dans les pipelines sans être visibles

Tests automatisés

- Tests unitaires
- Tests d'intégration
- Analyse de qualité du code

| 02

OpenShift



OPENSHIFT

Accès

<https://console-openshift-console.apps.factory.trail.ac/quickstart>

Demande d'accès : Laurent Gravez (laurent.gravez@cetic.be)

I 02

OpenShift | Concept

Une PaaS (Platform as a Service) de Red Hat pour déployer des projets dans des conteneurs

Idée centrale

- Transformer une application en conteneur et la gérer sur un cluster
- OpenShift s'occupe de :
 - build, déploiement, exécution
 - scaling et monitoring

Architecture

- Control Plane (masters) : décide et orchestre
- Nodes (workers) : exécutent les applications
- Pods : contiennent les conteneurs
- Routes : exposent les applications à l'extérieur

Basé sur Kubernetes

- Pods (conteneurs), Services (réseau interne)
- Deployments (gestion des versions), scaling automatique
- Ajoute une couche plus complète et prête pour l'entreprise

I 02

OpenShift vs Kubernetes

Par rapport à Kubernetes, OpenShift apporte :

- Des pipelines CI/CD standardisés pour une intégration et une mise en opération plus aisées
- Des outils d'automatisation du cycle de vie des conteneurs, inclus par défaut
- Une interface en ligne de commande en plus de celle de Kubernetes, et une interface graphique
- Une intégration et une migration facilitées sur la plupart des plateformes cloud
- Une sécurité renforcée, conforme aux standards
- Un accès des utilisateurs basé sur un contrôleur d'accès et une notion de routes
- Des opérateurs : la méthode préférée pour emballer, déployer et gérer des services dans le control plane

Kubernetes (K8s)

Système open source qui automatise le déploiement, la montée en charge et la mise en œuvre de conteneurs d'application sur des clusters de serveurs.

| 02

OpenShift | GPU ?

Pas de GPU sur l'infrastructure de la factory

Mais il est possible d'exploiter des GPU de providers externes (cloud ou infrastructure privée) directement depuis la factory.

02

OpenShift | Console

<https://console-openshift-console.apps.factory.trail.ac/quickstart>

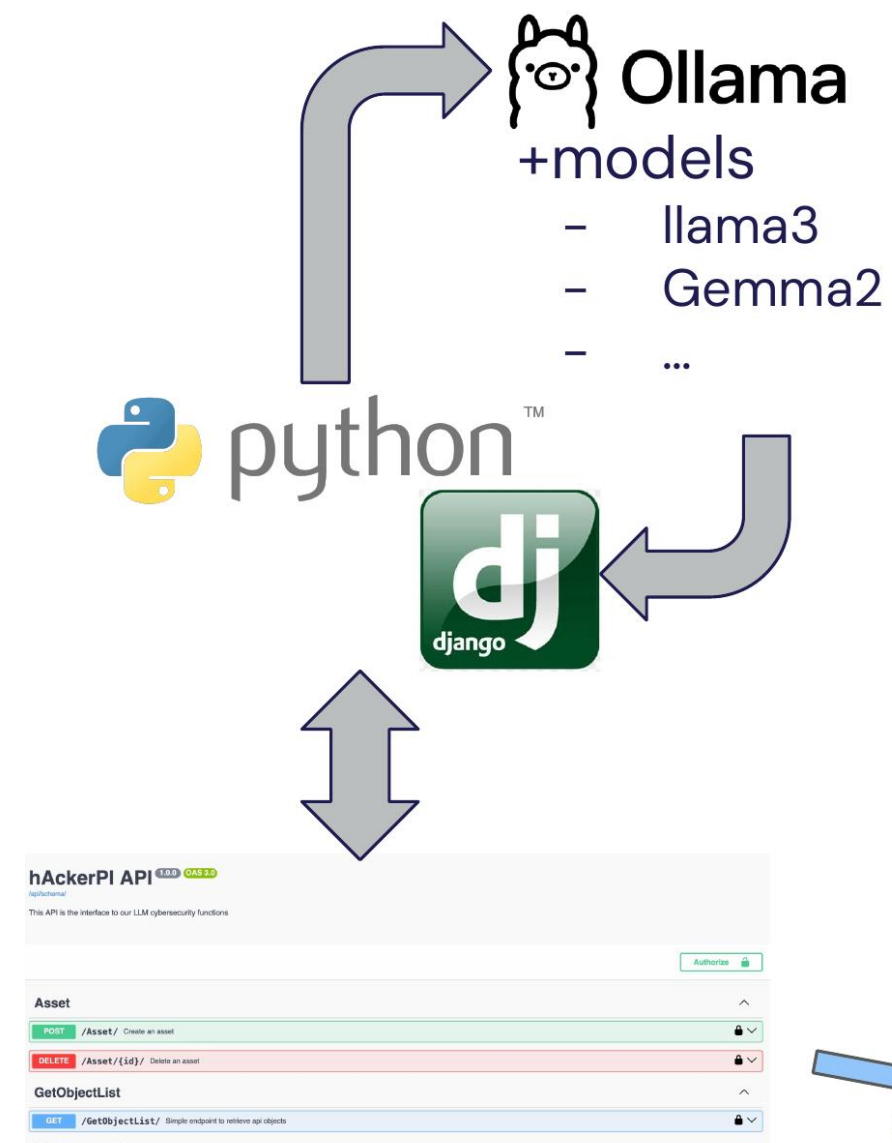
The screenshot displays the OpenShift console interface. The top navigation bar includes the 'okd' logo, a user profile 'Xavier Lessage', and a 'Créer Projet' button. The main content area is titled 'Projets' and features a table of projects. The table has columns for 'Nom', 'Nom complet', 'Statut', 'Demandeur', and 'Créé'. A single project is listed: 'demo1-xle' in the 'demo1' namespace, with a status of 'Active' and creator 'xlessage'. The left sidebar provides navigation options for the 'Plateforme de base', including 'Accueil', 'Projets', 'Recherche', 'Explorateur d'API', 'Événements', 'Favoris', 'Écosystème', 'Charges de travail', 'Mise en réseau', 'Stockage', 'Compilations', 'Observer', 'Gestion des utilisateurs', and 'Administration'.

Nom	Nom complet	Statut	Demandeur	Créé
PR demo1-xle	Aucun nom complet	Active	xlessage	16 avr. 2026, 14:52

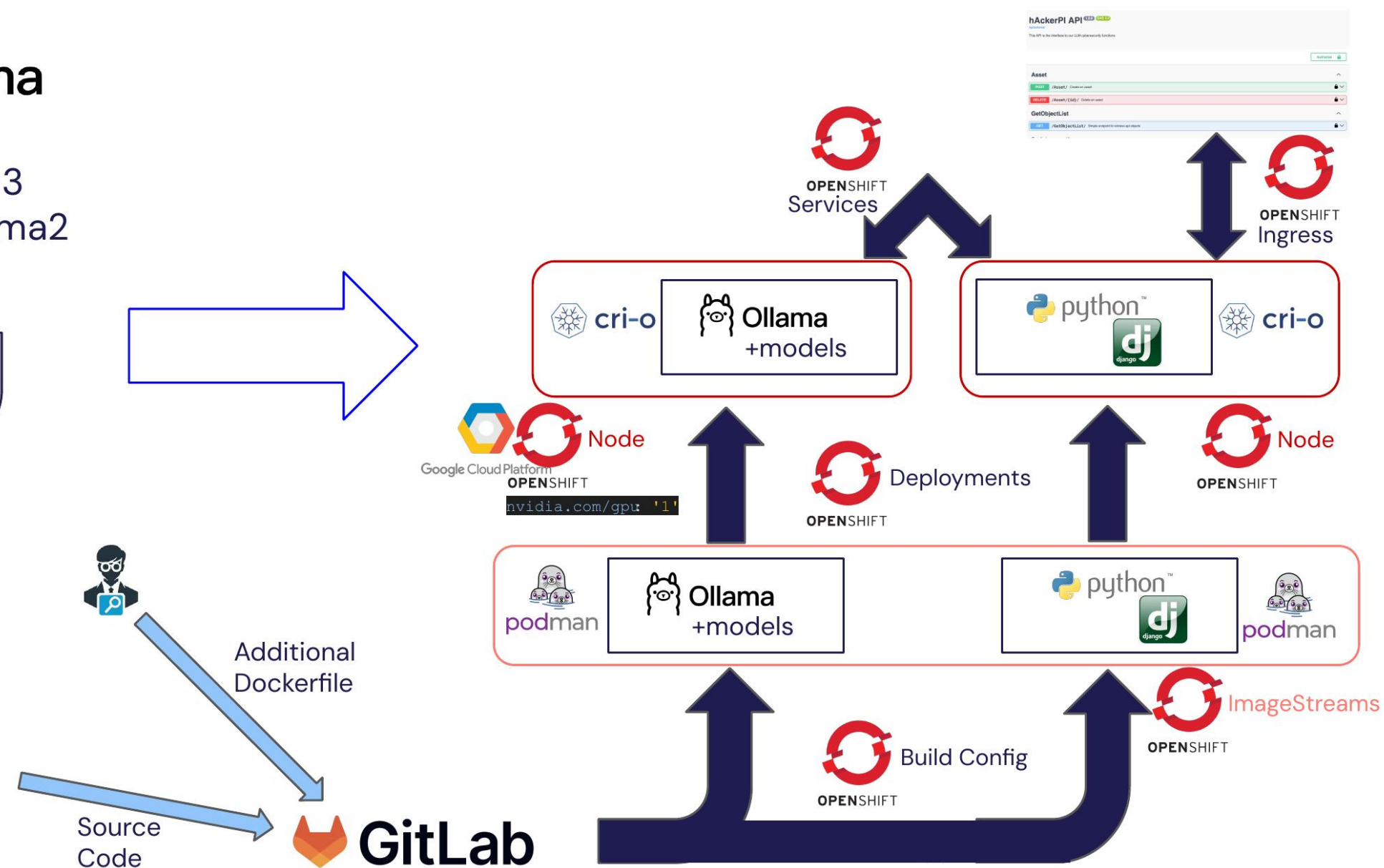
03

Brique « LLMs White Hat Hacker »

ON MY MACHINE



ON THE FACTORY

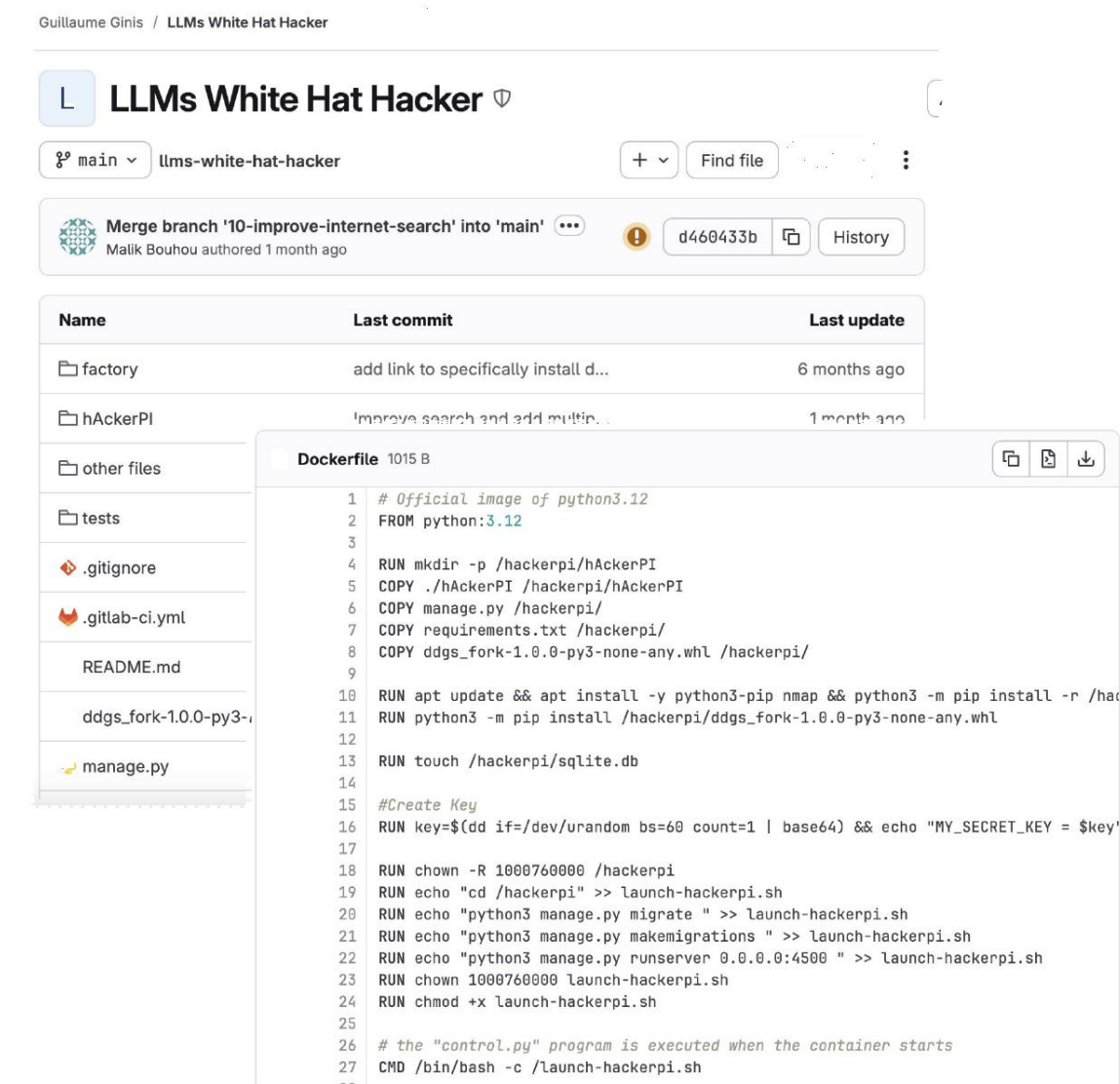


03

Brique « LLMs White Hat Hacker »

Point de départ = base de code existante

- Utilisation d'un gestionnaire de dépôt
 - GitHub / GitLab
 - En ligne ou local (mais accessible depuis la factory)
 - Par défaut : le GitLab de la factory
- Ajout d'un Dockerfile
 - Le service à l'intérieur du conteneur ne peut pas s'exécuter en root !
 - Création d'un répertoire de travail avec les bons droits et autorisations, et modification des droits sur les fichiers



The screenshot shows a GitLab repository named 'LLMs White Hat Hacker' by Guillaume Ginis. The 'main' branch is selected. A merge request is visible: 'Merge branch '10-improve-internet-search' into 'main'' by Malik Bouhou, authored 1 month ago, with commit hash d460433b. The file explorer on the left shows the repository structure: 'factory', 'hAckerPI', 'other files', 'tests', '.gitignore', '.gitlab-ci.yml', 'README.md', 'ddgs_fork-1.0.0-py3-', and 'manage.py'. The 'Dockerfile' (1015 B) is open, showing the following content:

```
1 # Official image of python3.12
2 FROM python:3.12
3
4 RUN mkdir -p /hackerpi/hAckerPI
5 COPY ./hAckerPI /hackerpi/hAckerPI
6 COPY manage.py /hackerpi/
7 COPY requirements.txt /hackerpi/
8 COPY ddgs_fork-1.0.0-py3-none-any.whl /hackerpi/
9
10 RUN apt update && apt install -y python3-pip nmap && python3 -m pip install -r /hac
11 RUN python3 -m pip install /hackerpi/ddgs_fork-1.0.0-py3-none-any.whl
12
13 RUN touch /hackerpi/sqlite.db
14
15 #Create Key
16 RUN key=$(dd if=/dev/urandom bs=60 count=1 | base64) && echo "MY_SECRET_KEY = $key"
17
18 RUN chown -R 1000760000 /hackerpi
19 RUN echo "cd /hackerpi" >> launch-hackerpi.sh
20 RUN echo "python3 manage.py migrate " >> launch-hackerpi.sh
21 RUN echo "python3 manage.py makemigrations " >> launch-hackerpi.sh
22 RUN echo "python3 manage.py runserver 0.0.0.0:4500 " >> launch-hackerpi.sh
23 RUN chown 1000760000 launch-hackerpi.sh
24 RUN chmod +x launch-hackerpi.sh
25
26 # the "control.py" program is executed when the container starts
27 CMD /bin/bash -c /launch-hackerpi.sh
28
```

03

OpenShift | Build Config

The screenshot shows the OpenShift console interface for editing a Build Config named 'hackerpi-build'. The 'YAML' tab is selected, displaying the following configuration:

```
42 spec:
51   strategy:
53     dockerStrategy:
55       postCommit: {}
56     source:
57       type: Git
58       git:
59         uri: 'https://gitlab.apps.factory.cyberexcellence.be/gginis/llms-white-hat-hacker.git'
60         ref: main
61       contextDir: /
62       sourceSecret:
63         name: hackerpi-token
64       runPolicy: Serial
65 status:
```

Annotations with arrows point to specific fields:

- Type : Git** points to `type: Git`.
- URL du Repo** points to the `uri` field.
- Branch Repository** points to the `ref: main` field.
- Access Token** points to the `name: hackerpi-token` field.
- Strategy (Dockerfile)** points to the `strategy` section in a separate code block below.
- Output ImageStream Tag** points to the `name: 'hackerpi-st:latest'` field in another code block below.

Below the main editor, two additional code snippets are shown:

```
strategy:
  type: Docker
  dockerStrategy:
    dockerfilePath: factory/API/Dockerfile
```

```
output:
  to:
    kind: ImageStreamTag
    name: 'hackerpi-st:latest'
```

03

OpenShift | Build Config

Project: hackerpi ▾

Configure via: ☒ Form view ☐ YAML view

Note: Some fields may not be represented in this form view. Please select "YAML view" for full control. ×

Name *

Source

Source type *

Please select your source type ▾

Source could be a git repository or Dockerfile

Images

Build from *

Please select ▾

Create

Cancel

Aussi disponible en formulaire

Certaines configurations peuvent toutefois ne pas être accessibles : la vue YAML reste nécessaire pour un contrôle complet.

03

OpenShift | Deployments

Project: hackerpi ▾

Create Deployment

Configure via: ☒ Form view ☐ YAML view

Name *

Deployment strategy

Strategy type

Rolling Update ▾

The rolling strategy will wait for pods to pass the

Maximum number of unavailable Pods

25%

The maximum number of pods that can be unavailable (10%) or a whole number (1).

Maximum number of surge Pods

Images

Container:  container

☒ Deploy image from an image stream tag

Image stream tag

Project * Image Stream * Tag *

hackerpi / Select Image Stream : Select tag

☐ Auto deploy when new Image is available

> Show advanced image options

Environment Variables

Container:  container

Aussi disponible en YAML pour les configurations avancées

- Paramétrage de la stratégie (nombre de pods, etc.)
- Paramétrage de la source (ImageStreamTag)
- Si besoin de GPU :

```
resources:
  limits:
    nvidia.com/gpu: '1'
```

03

OpenShift | Services & Ingress

```
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Services

Configuration réseau
(interne à OpenShift)

Project: keycloak

Create Route

Routing is a way to make your application publicly visible

Configure via: ☒ Form view ☐ YAML view

Name *

route-bronze-harrier

A unique name for the Route within the project

Hostname

Public hostname for the Route. If not specified, a hostname is generated.

Path

Path that the router watches to route traffic to the service.

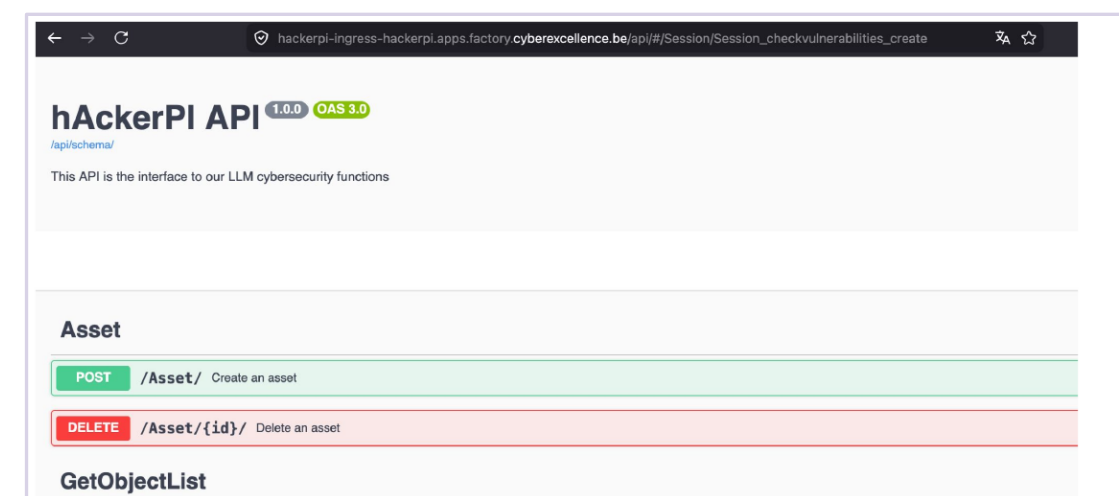
Service *

Select a Service

Service to route to.

Ingress (route)

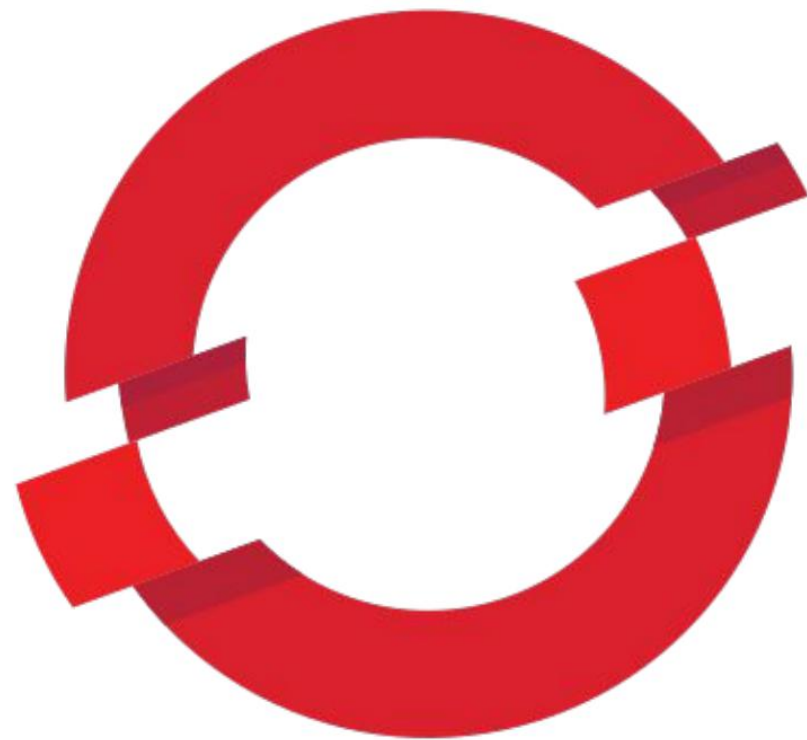
URL où l'application sera accessible



L'API hAckerPI exposée sur la factory

| 04

Q & R



OPENSIFT



GitLab

TRAIL
TRUSTED AI LABS

I 05

Hands on | Déploiement sur la factory

Trail Factory

- GitLab : <https://gitlab.apps.factory.trail.ac>
- OpenShift : <https://console-openshift-console.apps.factory.trail.ac>
- Client OpenShift (oc), prérequis — documentation : <https://trail-tutorials-trail-tutorials.apps.factory.trail.ac/>

Installing and Using the OpenShift CLI (oc)

The OpenShift CLI (`oc`) is the main tool for interacting with an OpenShift cluster from your terminal. This guide will walk you through installing `oc` , verifying the installation, and logging in to your OpenShift cluster.

Step 1: Download the OpenShift CLI

The `oc` CLI is available for Linux, macOS, and Windows. You can download it from the official OpenShift website.

Linux macOS Windows

Download and extract oc cli

```
curl -LO https://mirror.openshift.com/pub/openshift-v4/clients/oc/latest/linux/oc.  
mkdir -p ~/.local/bin  
sudo tar -xvzf oc.tar.gz -C ~/.local/bin  
export PATH=$HOME/.local/bin:$PATH
```

```
curl -LO https://mirror.openshift.com/pub/openshift-v4/clients/oc/latest/linux/oc.tar.gz  
mkdir -p ~/.local/bin  
sudo tar -xvzf oc.tar.gz -C ~/.local/bin  
export PATH=$HOME/.local/bin:$PATH
```

I 05

Hands on | Étapes clés

1

**Validation de
l'application sur le
PC**

2

**Création du projet
dans GitLab**

- Code de l'application
- Dockerfile et requirements.txt
- .gitlab-ci.yml

3

**Création du projet
dans OpenShift et
exécution**

4

**Activation de la
CI/CD et exécution**

I 05

Hands on | Application « Hello World! »

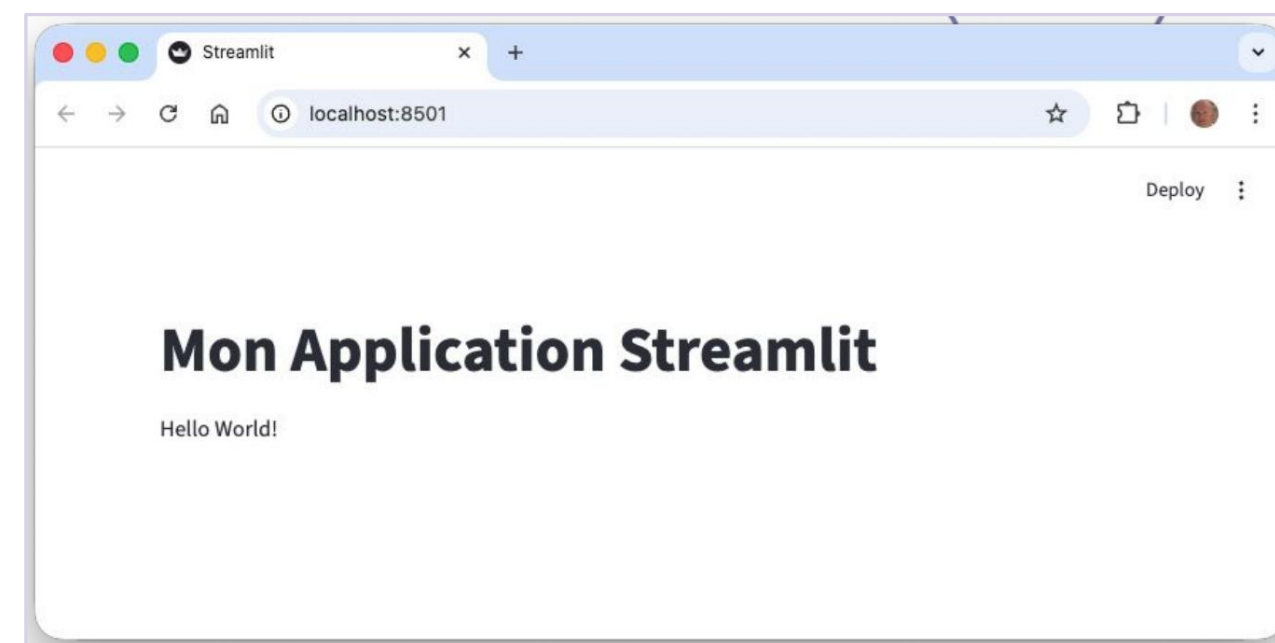
Création de l'application web « demo » avec Visual Studio Code

```
# app.py
import streamlit as st
st.title("Mon Application Streamlit")
st.write("Hello World!")

# Environnement virtuel
python3 -m venv .venv
source .venv/bin/activate

# Module
pip install streamlit

# Exécution
streamlit run demo.py
```



05

Hands on | Dockerfile et requirements.txt

Dockerfile

```
# Start from the official Streamlit image
FROM python:3.9-slim

# Set up working directory
WORKDIR /app

# Install dependencies
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

# Copy app files
COPY . ./

# Expose the Streamlit port
EXPOSE 8501

# Run the Streamlit app
CMD ["streamlit", "run", "app.py", \
    "--server.port=8501", "--server.address=0.0.0.0"]
```

requirements.txt

```
streamlit
```

05

Hands on | Création du projet dans GitLab

<https://gitlab.apps.factory.trail.ac>

Your work / Projects / New project / Create blank project

Create blank project

Create a blank project to store your files, plan your work, and collaborate on code, among other things.

Project name

Must start with a lowercase or uppercase letter, digit, emoji, or underscore. Can also contain dots, pluses, dashes, or spaces.

Project URL

Project slug

Visibility Level ?

☒ Private
Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.

☐ Internal
The project can be accessed by any logged in user except external users.

☐ Public
The project can be accessed without any authentication.

Project Configuration

☒ Initialize repository with a README
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

☐ Enable Static Application Security Testing (SAST)
Analyze your source code for known security vulnerabilities. [Learn more.](#)

☐ Enable Secret Detection
Scan your code for secrets and credentials to prevent unauthorized access. [Learn more.](#)

Connexion - GitLab

gitlab.apps.factory.cyberexcellence.be/user...

Navigation privée (2)

GitLab Enterprise Edition

Nom d'utilisateur ou adresse de courriel principale

Mot de passe

[Mot de passe oublié ?](#)

☐ Se souvenir de moi

ou connectez-vous avec

☐ Se souvenir de moi

[Explorer](#) [Aide](#) [À propos de GitLab](#) [Forum de la communauté GitLab](#) [français](#)

- Connexion via SSO (Keycloak) : <https://sso.apps.factory.trail.ac>
- Create blank project : nom du projet et niveau de visibilité

| 05

Hands on | Configuration du dépôt

Dans Visual Studio Code

```
git init --initial-branch=main --object-format=sha1
git remote add origin git@gitlab.apps.factory.trail.ac:xmessage/demo.git
git add .
git commit -m "Initial commit"
git push --set-upstream origin main
```

05

Hands on | Ajout du projet dans OpenShift

```
oc login --web --server=https://api.factory.trail.ac:6443

oc new-project demo-xavier
oc new-build --binary --name demo -l app=demo

oc start-build demo --from-dir=. --follow # à relancer si modification du code

oc new-app demo:latest
oc expose service/demo
```

Application accessible sur : <http://demo-demo-xavier.apps.factory.trail.ac/>

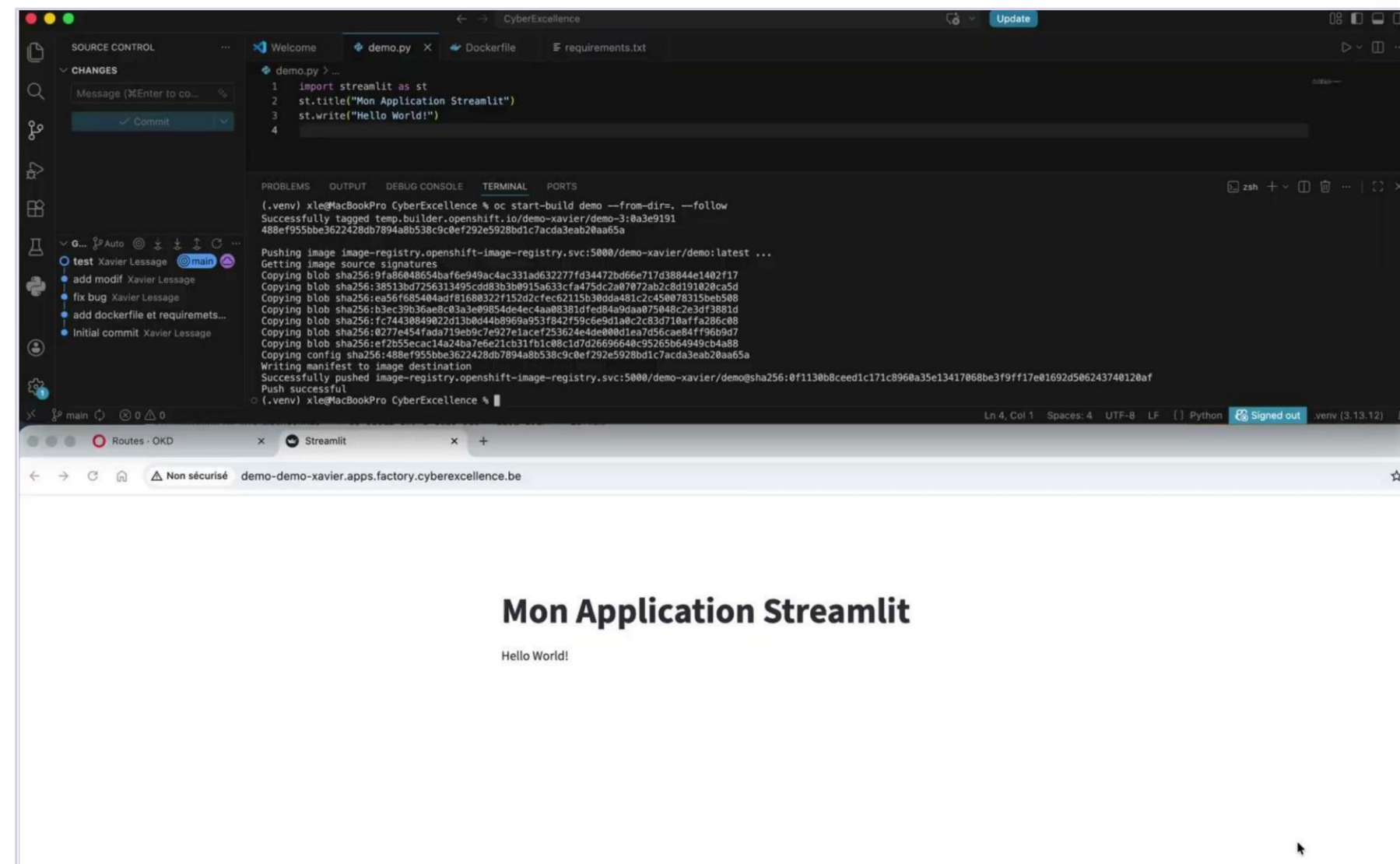
05

Hands on | CI/CD GitLab et OpenShift

Manuellement

- Mise à jour du code dans GitLab
- Relancer le build à chaque modification du code :

```
oc start-build demo \
  --from-dir=. --follow
```



05

Hands on | CI/CD GitLab et OpenShift

```
.gitlab-ci.yml
1  stages:
2    - build
3    - deploy
4
5  variables:
6    IMAGE_NAME: "registry.apps.factory.cyberexcellence.be/demo1-xle/demo-jdc"
7    IMAGE_TAG: "latest"
8    STORAGE_DRIVER: vfs
9
10 build-streamlit:
11   image: quay.io/buildah/stable:latest
12   stage: build
13   variables:
14     STORAGE_DRIVER: vfs
15     BUILDAH_FORMAT: docker
16   rules:
17     - if: '$CI_COMMIT_BRANCH == "main"'
18   before_script:
19     - export HOME=$CI_PROJECT_DIR
20     - mkdir -p $HOME/.config/containers
21
22     # (optionnel mais recommandé)
23     - export XDG_RUNTIME_DIR=$CI_PROJECT_DIR/run
24     - mkdir -p $XDG_RUNTIME_DIR
25
26     # Auth registry
27     - export REGISTRY_AUTH_FILE=$CI_PROJECT_DIR/auth.json
28     - echo "{\"auths\":{\"registry.apps.factory.cyberexcellence.be\":{\"username\":\"gitlab-ci-token\",\"password\":\"$CI_JOB_TOKEN\"}}}" > $
29
30   script:
31     # - buildah bud --storage-driver=$STORAGE_DRIVER -t $IMAGE_NAME:$IMAGE_TAG .
32     - buildah bud --isolation=chroot --storage-driver=vfs -t $IMAGE_NAME:$IMAGE_TAG .
33     - buildah push --storage-driver=$STORAGE_DRIVER $IMAGE_NAME:$IMAGE_TAG
34
```

Automatiquement

Avec un template .gitlab-ci.yml :

- Stages build puis deploy
- Image construite avec Buildah
- Push vers le registry de la factory
- Déclenché sur la branche main

TRAIL

TRUSTED AI LABS

MERCI POUR VOTRE ATTENTION !

RAPHAEL.MICHEL@CETIC.BE
GUILLAUME.GINIS@CETIC.BE
XAVIER.LESSAGE@CETIC.BE



UNIVERSITÉ
LIBRE
DE BRUXELLES

