

RAPPORT DEVOPS GITHUB ACTION - JENKINS - GITLAB CI

PRESENTE PAR : MOUHAMADOU LAMINE
NDIAYE M2-GL

Professeur : Mr. SECK

ISI - 2023 / 2024



GITHUB ACTION

Qu'est-ce que GitHub Actions?

GitHub Actions est un service d'automatisation intégré directement dans GitHub, permettant la création de workflows personnalisés pour l'intégration continue, le déploiement continu et d'autres processus automatisés.

Fonctionnalités clés

Les principales fonctionnalités comprennent la définition de workflows dans des fichiers YAML, les déclencheurs d'événements basés sur des actions dans le référentiel, et une bibliothèque d'actions prêtes à l'emploi. Ces fonctionnalités ont considérablement amélioré notre capacité à automatiser et à personnaliser nos flux de travail.

Mise en place de GitHub Actions

Configuration du Workflow

Nos workflows sont définis dans un fichier YAML situé à la racine du référentiel. Cela permet une gestion facile et une traçabilité transparente des modifications apportées aux workflows au fil du temps.

Environnements

Nous utilisons les environnements pour spécifier des configurations spécifiques pour nos jobs. Cela facilite le déploiement sur différents environnements, garantissant ainsi la cohérence dans les différents stades du processus de développement.

Intégration Continue avec GitHub Actions

Tests automatisés

Nos workflows incluent des jobs dédiés à l'exécution de tests automatisés à chaque push de code. Cela assure la détection rapide des erreurs potentielles, favorisant une approche robuste du développement.

Analyse statique du code

L'intégration d'outils d'analyse statique du code, tels que SonarQube, a permis d'améliorer la qualité globale du code en identifiant et en corrigeant les problèmes de code lors du processus d'intégration continue.

Déploiement Continu avec GitHub Actions

Configuration du déploiement

Nous avons configuré des workflows pour gérer le déploiement continu de nos applications sur des environnements de test et de production. L'utilisation de Docker et de GitHub Actions a simplifié ce processus, garantissant des déploiements rapides et cohérents.

Gestion des versions

GitHub Actions nous a aidés à gérer efficacement les versions de nos applications, intégrant le processus de gestion des versions dans nos workflows de déploiement continu.

Bonnes Pratiques

Sécurité

La gestion des secrets dans GitHub Actions a été une priorité. Nous avons mis en place des bonnes pratiques pour garantir la sécurité des informations sensibles utilisées dans nos workflows.

Efficacité

Nous avons optimisé nos workflows pour maximiser l'efficacité, en parallélisant les étapes lorsque cela est possible et en minimisant les temps d'attente pour les résultats des workflows.

CAS PRATIQUE AVEC UN PROJET SPRING BOOT

Avant de configurer le workflow, nous générons un jeton d'accès dans le registre Docker que nous utilisons et on lui attribue les droits de lecture, d'écriture et de suppression qui nous permettent aussi de gérer nos repos.

The screenshot shows the GitHub 'New Access Token' interface. On the left is a sidebar with navigation links: General, Security (highlighted), Default Privacy, Notifications, Convert Account, and Deactivate Account. The main content area is titled 'New Access Token' and includes a description of personal access tokens. A red box highlights the 'Access Token Description' field, which contains the text 'github-action-token'. Below this is a dropdown menu for 'Access permissions' set to 'Read, Write, Delete'. At the bottom of the main area are 'Cancel' and 'Generate' buttons, with the 'Generate' button highlighted by a red box. On the right side of the interface, another red box highlights the 'New Access Token' button in the sidebar.

New Access Token

A personal access token is similar to a password except you can have many tokens and revoke access to each one at any time. [Learn more](#)

Access Token Description *

github-action-token

Access permissions

Read, Write, Delete

Read, Write, Delete tokens allow you to manage your repositories.

Cancel Generate

New Access Token

maximum of 5 auto-generated tokens

created


Feb 23, 2024	19:49:00	⋮
Jan 25, 2024	17:37:23	⋮
Dec 20, 2023	22:41:45	⋮
Nov 09, 2023	12:48:27	⋮
Jan 09, 2024	20:17:40	⋮


Voici notre token créé manuellement :


	 github-action-token	MANUAL	Read, Write, Delete	Feb 23, 2024 20:17:46	Feb 23, 2024 19:49:00	
---	---	------------------------	---------------------	-----------------------	-----------------------	---


Dans le repo de notre projet github, sous le menu action puis variables and secrets, nous créons nos variables secrètes avec comme valeurs respectives le clé de notre token créé dans docker et notre username de docker.

Security

 Code security and analysis

 Deploy keys

 **Secrets and variables**






 Actions

 Codespaces

 Dependabot

Repository secrets

New repository secret

Name 	Last updated
 DOCKER_TOKEN	last week  
 DOCKER_USER	last week  

Voici notre fichier deploy.yml

name: Create a Docker image using Docker Compose and then push it to Docker hub

on:

push:

branches: [main]

paths-ignore:

- '**/README.md'

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Check out code

uses: actions/checkout@v3 LamineOzilJr

jobs: | LamineOzilJr, 2/23/2024 7:31 PM • initialization

build:

runs-on: ubuntu-latest

steps:

- name: Check out code

uses: actions/checkout@v3

- name: Build docker images

run: docker compose build

- name: log to docker hub

run: docker login -u \${ secrets.DOCKER_USER } -p \${ secrets.DOCKER_TOKEN }

- name: tag image

run: docker tag dockervolume-backend:1.0 lamineoziljr/dockervolume-backend:2.0

- name: push to docker hub tests

run: docker push lamineoziljr/dockervolume-backend:2.0

Ce fichier représente un workflow GitHub Actions qui se déclenche lorsqu'un push est effectué sur la branche principale (**main**). Le workflow réalise plusieurs étapes liées à la construction et au déploiement d'une image Docker à l'aide de Docker Compose.

Checkout du Code

- Utilisation de l'action **checkout** pour récupérer le code source depuis le référentiel.

2. Construction des Images Docker

- Exécution de la commande **docker compose build** pour construire les images Docker en utilisant Docker Compose.

3. Connexion à Docker Hub

- Utilisation des secrets GitHub (**DOCKER_USER** et **DOCKER_TOKEN**) pour se connecter à Docker Hub à l'aide de la commande **docker login**.

4. Étiquetage de l'Image Docker

- Étiquetage de l'image Docker construite avec la version **1.0** du tag **dockervolume-backend** pour la version **2.0** du tag **lamineoziljr/dockervolume-backend**.

5. Pousser l'Image Docker vers Docker Hub

- Utilisation de la commande **docker push** pour pousser l'image Docker étiquetée (**lamineoziljr/dockervolume-backend:2.0**) vers Docker Hub.

Voici mon docker compose

```
version: '3.9'

services:
  app-dockervolume-backend:
    image: dockervolume-backend:1.0
    container_name: container-dockervolume-backend
    ports:
      - 8080:8080
    restart: unless-stopped
    build:
      context: ./
      dockerfile: Dockerfile
    environment:
      directoryDatas: /app/data/
    volumes:
      - ./datas:/app/data
```

```
volumes:
  - ./datas:/app/data
```

```
volumes:
  datas:
```

Nous avons fait une configuration de service pour Docker Compose, un outil qui permet de définir et de gérer des applications. Nous avons d'abord spécifié la version de la syntaxe.

app-dockervolume-backend: C'est le nom du service.

- **image**: L'image Docker à utiliser pour ce service s'appelle **dockervolume-backend** avec la version **1.0**.
- **container_name**: Le nom du conteneur Docker qui sera créé pour ce service est nommé **container-dockervolume-backend**.
- **ports**: Mappage des ports entre le conteneur et l'hôte. Le service exposera le port **8080** sur l'hôte.
- **restart**: La politique de redémarrage du conteneur en cas d'arrêt inattendu. Dans ce cas, le conteneur sera redémarré sauf s'il est explicitement arrêté.
- **build**: Configuration pour la construction de l'image Docker à partir d'un fichier Dockerfile(fichier que nous allons détailler après).
 - **context**: Le chemin vers le répertoire contenant les fichiers nécessaires à la construction de l'image.
 - **dockerfile**: Le chemin vers le fichier Dockerfile à utiliser pour la construction.
- **environment**: Définition de variables d'environnement pour le conteneur. Ici, une variable **directoryDatas** est définie avec la valeur **/app/data/**.
- **volumes**: Montage d'un volume entre l'hôte et le conteneur. Le répertoire local **./datas** est monté dans le conteneur sous le chemin **/app/data**.

Voici notre dockerFile

```
FROM openjdk:17-jdk-slim

LABEL maintainer="Lamine NDIAYE lamzojr72@gmail.com"

EXPOSE 8080    LamineOzilJr, 2/23/2024 7:31 PM • initialization

RUN mkdir -p /app/data

ADD docker/dockervolume.jar dockervolume.jar

ENTRYPOINT ["java", "-jar", "dockervolume.jar"]
```

Notre Dockerfile est une instruction pour la construction d'une image Docker. Ce fichier construit une image basée sur OpenJDK 17, configure le mainteneur, expose le port 8080, crée un répertoire pour stocker des données, copie un fichier JAR dans l'image, et définit la commande d'entrée pour exécuter le fichier JAR lors du démarrage du conteneur. Cette image est conçue pour exécuter notre application Java encapsulée dans notre dockervolume.jar.

FROM openjdk:17-jdk-slim

- Cette instruction spécifie l'image de base à utiliser pour construire cette image Docker. Il s'agit de l'image officielle OpenJDK version 17 basée sur une image slim de Debian.

LABEL maintainer="Lamine NDIAYE lamzojr72@gmail.com"

- Cette instruction ajoute une étiquette (label) à l'image Docker, indiquant le mainteneur de l'image avec son adresse e-mail.

EXPOSE 8080

- Ceci informe Docker que le conteneur écoutera sur le port 8080.

RUN mkdir -p /app/data

- Creation d'un répertoire /app/data à l'intérieur du conteneur. Il s'agit du répertoire dans lequel les données seront stockées.

ADD docker/dockervolume.jar dockervolume.jar

- Cette instruction ajoute (copie) le fichier dockervolume.jar depuis le chemin local docker/dockervolume.jar dans le répertoire racine du conteneur.

ENTRYPOINT ["java", "-jar", "dockervolume.jar"]

- Cette instruction définit la commande qui sera exécutée lorsque le conteneur est démarré. Elle lance l'exécution du fichier JAR dockervolume.jar en utilisant Java.

Résultats

All workflows

Showing runs from all workflows

🔍 Filter workflow runs

5 workflow runs

Event ▾

Status ▾

Branch ▾

Actor ▾

✓ updated workflows to V2 Create a Docker image using Docker Compose and then push it to Docker hub #3: Commit a9faf7c pushed by LamineOzilJr	main	📅 last week 🕒 31s	...
✓ updated workflows to V2 build #3: Commit a9faf7c pushed by LamineOzilJr	main	📅 last week 🕒 34s	...
✓ Merge branch 'main' of https://github.com/LamineO... build #2: Commit 1a4861e pushed by LamineOzilJr	main	📅 last week 🕒 32s	...
✓ Merge branch 'main' of https://github.com/LamineO... Create a Docker image using Docker Compose and then push it to Docker hub #2: Commit 1a4861e pushed by LamineOzilJr	main	📅 last week 🕒 31s	...
✓ initialization	main	📅 last week	...

Résultats

← build

✓ updated workflows to V2 #3

Re-run all jobs



Summary

Jobs

✓ build

Run details

Usage

Workflow file

build

succeeded last week in 21s

Beta

Give feedback

Search logs



> ✓ Set up job	1s
> ✓ Run actions/checkout@v2	2s
> ✓ Set up JDK 17	3s
> ✓ Build with Maven	10s
> ✓ Post Set up JDK 17	0s
> ✓ Post Run actions/checkout@v2	0s
> ✓ Complete job	1s



JENKINS

Qu'est-ce que Jenkins?

Jenkins est un serveur d'intégration continue open-source permettant d'automatiser diverses étapes du processus de développement logiciel, y compris la compilation, les tests, et le déploiement.

Intégration avec Java et Docker

Jenkins offre une intégration fluide avec les projets Java, facilitant la gestion des dépendances, la compilation, et l'exécution de tests unitaires. De plus, les plugins Docker de Jenkins sont exploités pour la création, la gestion, et le déploiement d'images Docker.

Configuration de Jenkins

Installation et Configuration Initiale

La configuration initiale de Jenkins a été réalisée en utilisant Docker pour garantir une gestion efficace des dépendances et une isolation des environnements.

Intégration avec GitHub

Jenkins est connecté à notre référentiel GitHub, permettant ainsi la détection automatique des modifications de code et le déclenchement des workflows CI/CD à chaque push sur la branche principale.

Flux de Travail CI/CD avec Jenkins

Déclenchement Automatique des Builds

Les workflows CI sont configurés pour démarrer automatiquement lorsqu'un nouveau code est poussé sur GitHub. Cela inclut la compilation du code Java, l'exécution de tests unitaires, et la génération d'artefacts.

Tests Automatisés avec Jenkins

Jenkins est utilisé pour exécuter une suite complète de tests automatisés, garantissant la stabilité et la qualité du code avant tout déploiement.

Déploiement Continu avec Jenkins et Docker

Les workflows CD incluent le déploiement continu sur des environnements de test et de production. Jenkins orchestre le déploiement d'images Docker préalablement construites.

Gestion des Builds et des Versions

Builds Multi-branches

Jenkins est configuré pour gérer des builds multi-branches, permettant la création de builds distincts pour chaque branche, garantissant une isolation et une validation indépendante.

Gestion des Versions avec Jenkins

La gestion des versions est intégrée dans Jenkins, avec des tags automatiques générés lors des déploiements réussis, facilitant la traçabilité et le rollback si nécessaire.

Voici notre page d'accueil jenkins avec l'ensemble de nos projets déjà exécutés

Tableau de bord [Jenkins]

localhost:8080

rechercher (CTRL+K)

se déconnecter

Tableau de bord

+ Nouveau Item

Ajouter une description

Utilisateurs

Tous

Historique des constructions

Relations entre les builds

Vérifier les empreintes numériques

Administrer Jenkins

Mes vues

Utilisation du disque

File d'attente des constructions

S	M	Nom du projet ↓	Dernier succès	Dernier échec	Dernière durée	
✓	☀	springSecurity-Build-Jenkins	2 j 10 h #8	s. o.	1 mn 15 s	▶
✓	☀	springSecurity-Build-Jenkins-Pipeline	2 j 9 h #3	s. o.	1 mn 47 s	▶
✓	☀	springSecurity-Build-Jenkins-Pipeline-Docker	2 j 3 h #15	s. o.	9 mn 5 s	▶
✓	☀	springSecurity-Build-Jenkins-Pipeline-PushToDockerHub	2 j 2 h #2	s. o.	5 mn 38 s	▶

On crée un nouvel élément

+ Nouveau Item

On saisi le nom du projet et on sélectionne ce que l'on veut faire. Pour notre cas c'est un projet freestyle et d'autre projets pipeline. **NB : Nous ne pouvons choisir qu'un seul type par projet.**

Tableau de bord > Tous >

Saisissez un nom

springSecurity-Build-Jenkinss

» Champ obligatoire



Construire un projet free-style

Job legacy polyvalent qui récupère l'état depuis un outil de gestion de version au plus, exécute les étapes de build en série, suivi d'étapes post-construction telles que l'archivage d'artefacts et l'envoi de notifications par e-mail.



Pipeline

Organise des activités de longue durée qui peuvent s'étendre sur plusieurs agents de construction. Adapté pour la création des pipelines (anciennement connues comme workflows) et/ou pour organiser des activités complexes qui ne s'adaptent pas facilement à des tâches de type libre.

Cocher github et renseigner l'url de notre projet github



GitHub project

Project url ?

`https://github.com/LamineOzilJr/spring-security`

Définir(choisir) pipeline script from scm (Source Control Management) ou gestionnaire de contrôleur de source. Choisir Git et renseigner cette fois ci notre repository (.git)

Pipeline

Definition

Pipeline script from SCM



SCM ?

Git



Repositories ?

Repository URL ?



<https://github.com/LamineOzilJr/spring-security.git>

Sauvegarder

Appliquer

Définir la branche a builder, “main” pour notre cas et non “master” qui est la branche que jenkins défini par défaut

Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

Spécifier que nous allons utiliser un fichier jenkins. Ensuite appliquer et sauvegarder

Script Path ?

Jenkinsfile

☒ Lightweight checkout ?

[Pipeline Syntax](#)

Sauvegarder

Appliquer

WEBHOOK

C'est est un (end-Point) ou point de terminaison exposé par un serveur Java qui écoute les notifications déclenchées par des événements sur un référentiel GitHub. Cela permet d'automatiser des processus tels que le build, les tests, et le déploiement en réponse à des actions sur le référentiel.

Voici notre Webhook.

The screenshot shows a web browser window with the address bar displaying `github.com/LamineOziJr/spring-security/settings/hooks/463419939`. The page title is "Webhook · http://192.168.56.1:8080". The repository name "LamineOziJr / spring-security" is visible at the top. The navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings (which is highlighted). The left sidebar contains a list of repository settings: General, Access, Collaborators, Moderation options, Code and automation, Branches, Tags, Rules, Actions, Webhooks (selected), Environments, and Codespaces. The main content area is titled "Webhooks / Manage webhook" and has two tabs: "Settings" (active) and "Recent Deliveries". The "Settings" tab contains the following information:

- A descriptive text: "We'll send a `POST` request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, `x-www-form-urlencoded`, etc). More information can be found in [our developer documentation](#)." followed by a horizontal separator.
- Payload URL ***: A text input field containing `http://192.168.56.1:8080/github-webhook`.
- Content type**: A dropdown menu currently set to `application/json`.
- Secret**: A text input field that is currently empty.

Voici notre Webhook.

Secret

Which events would you like to trigger this webhook?

☒ Just the push event.

☐ Send me **everything**.

☐ Let me select individual events.

☒ **Active**

We will deliver event details when this hook is triggered.

Update webhook

Delete webhook

Voici notrejenkinsFile

```
pipeline {  
  agent any  
  stages {  
    stage('Build') {  
      steps {  
        bat "C:/apache-maven-3.9.5/bin/mvn clean package"  
      }  
    }  
    stage('Test') {  
      steps {  
        bat "C:/apache-maven-3.9.5/bin/mvn test"  
      }  
    }  
    stage('Deploy') {  
      steps {  
        bat 'C:/Docker/resources/bin/docker-compose up -d --build'  
      }  
    }  
  }  
}
```

LamineOzilJr, 2/26/2024 8:23 PM • first commit

Voici notrejenkinsFile

```
stage('Push to Docker Hub') {  
    steps {  
        withCredentials([usernamePassword(credentialsId: 'dockerhub_credent  
            bat "C:/Docker/resources/bin/docker login -u $DOCKER_HUB_USERNA  
            bat "C:/Docker/resources/bin/docker tag evalspringse:latest lan  
            bat "C:/Docker/resources/bin/docker push lamineoziljr/evalsprin  
        }  
    }  
}
```

Voici notrejenkinsFile

```
dockerhub_credentials', passwordVariable: 'DOCKER_HUB_PASSWORD', usernameVariable: 'DOCKER_HUB_USERNAME']])  
DOCKER_HUB_USERNAME -p $DOCKER_HUB_PASSWORD"  
ngse:latest lamineoziljr/evalspringse:v$BUILD_NUMBER"  
ziljr/evalspringse:v$BUILD_NUMBER"
```

Ce pipeline Jenkins orchestre les étapes de build, test, déploiement avec Docker Compose, et publication vers Docker Hub d'une application Java. Les identifiants sensibles sont gérés de manière sécurisée avec les credentials de Jenkins.

Déclaration du Pipeline:

- Le pipeline est configuré pour être exécuté sur n'importe quel agent disponible (**agent any**), ce qui signifie qu'il peut être exécuté sur n'importe quelle machine disponible dans l'environnement Jenkins.

Définition des Étapes (Stages):

- Les différentes étapes du pipeline sont définies dans la section stages. Chaque étape représente une phase spécifique du processus d'intégration et de déploiement.

Étape de Build:

- Cette étape est destinée à la construction du projet.
- La commande bat est utilisée pour exécuter des scripts batch sous Windows.
- La commande Maven spécifiée (**C:/apache-maven-3.9.5/bin/mvn clean package**) nettoie le projet et construit les artefacts de l'application.

Étape de Test:

- Cette étape est destinée à l'exécution des tests du projet.
- La commande Maven spécifiée (**C:/apache-maven-3.9.5/bin/mvn test**) exécute les tests unitaires définis dans le projet.

Étape de Déploiement:

- Cette étape est destinée au déploiement de l'application à l'aide de Docker Compose.
- La commande bat spécifiée (**C:/Docker/resources/bin/docker-compose up -d --build**) construit et lance les conteneurs Docker définis dans le fichier docker-compose.yml.

Étape de Push vers Docker Hub:

- Cette étape est destinée à la publication de l'image Docker sur Docker Hub.
- Les identifiants Docker Hub (nom d'utilisateur et mot de passe) sont récupérés de manière sécurisée depuis les credentials stockées dans Jenkins.
- La commande bat est utilisée pour exécuter des scripts Docker, notamment pour se connecter à Docker Hub (**docker login**), tagger l'image (**docker tag**), et pousser l'image taguée vers Docker Hub (**docker push**).
- Le tag de l'image est basé sur la version du build (**v\$BUILD_NUMBER**), assurant une traçabilité de l'image publiée.

Voici notre docker-compose

```
version: '3'

services:
  evalSpringMySQLSec:
    image: mysql:5.6
    container_name: evalspringmysqlsec
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=evalspringsecdb
      - MYSQL_USER=user
      - MYSQL_PASSWORD=user
    ports:
      - 3306:3306

  evalspringsec:
```

```
    image: evalspringse:latest
    container_name: container_evalspringsec
    ports:
      - 8087:8087
    restart: unless-stopped
    build:
      context: ./
      dockerfile: Dockerfile
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://evals
    depends_on:
      - evalSpringMySQLSec
```

Voici notre dockerFile

```
FROM openjdk:8
ADD target/evalsecurrity-0.0.1-SNAPSHOT.jar evalsecurrity.jar
EXPOSE 8087
ENTRYPOINT ["java", "-jar", "evalsecurrity.jar"]
```

Ceci est le résultat de notre dernier build qui est un pipeline qui push notre image dans docker hub

Tableau de bord > springSecurity-Build-Jenkins-Pipeline-PushToDockerHub >

springSecurity-Build-Jenkins-Pipeline-PushToDockerHub

Stage View

Average stage times:
(Average full run time: ~5min 38s)

	Declarative: Checkout SCM	Build	Test	Push to Docker Hub
#2 févr. 27 22:16 No Changes	3s	1min 19s	1min 9s	2min 57s

Liens permanents

- [Dernier build \(#2\), il y a 2 j 3 h](#)
- [Dernier build stable \(#2\), il y a 2 j 3 h](#)
- [Dernier build avec succès \(#2\), il y a 2 j 3 h](#)
- [Dernier build complété \(#2\), il y a 2 j 3 h](#)

Historique des builds tendance ▾

Filter...

#2 27 févr. 2024 22:16

[Atom feed des builds](#) [Atom feed des échecs](#)

Ceci est le résultat de notre dernier build qui est pipeline qui push notre image dans docker hub

springSecurity-Build-Jenkins-Pipeline-PushToDockerHub > #2



Sortie de la console

put

ain text

ormation

build "#2"

a

```
Started by user lamine
Obtained Jenkinsfile from git https://github.com/LamineOzilJr/spring-security.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\.jenkins\workspace\springSecurity-Build-Jenkins-Pipeline-PushToDockerHub
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
```

Ceci est le résultat de notre dernier build qui est pipeline qui push notre image dans docker hub

```
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] withEnv  
[Pipeline] {  
[Pipeline] stage  
[Pipeline] { (Build)  
[Pipeline] bat
```

```
C:\ProgramData\Jenkins\.jenkins\workspace\springSecurity-Build-Jenkins-Pipeline-PushToDockerHub>C:/apache-maven-  
3.9.5/bin/mvn clean package  
[INFO] Scanning for projects...  
[INFO]  
[INFO] -----< sn.minfinances:evalsecurrity >-----  
[INFO] Building evalsecurrity 0.0.1-SNAPSHOT
```

Ceci est le résultat de notre dernier build qui est pipeline qui push notre image dans docker hub

```
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 01:09 min  
[INFO] Finished at: 2024-02-27T22:17:47Z  
[INFO] -----  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] stage  
[Pipeline] { (Test)  
[Pipeline] bat  
  
C:\ProgramData\Jenkins\.jenkins\workspace\springSecurity-Build-Jenkins-Pipeline-PushToDockerHub>C:/apache-maven-3.9.5/bin/mvn test  
[INFO] Scanning for projects...  
[INFO]  
[INFO] -----< sn.minfinances:evalsecurity >-----  
[INFO] Building evalsecurity 0.0.1-SNAPSHOT  
[INFO]    from pom.xml
```

Ceci est le résultat de notre dernier build qui est pipeline qui push notre image dans docker hub


```
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 01:02 min  
[INFO] Finished at: 2024-02-27T22:18:56Z  
[INFO] -----  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] stage  
[Pipeline] { (Push to Docker Hub)  
[Pipeline] withCredentials  
Masking supported pattern matches of %DOCKER_HUB_PASSWORD%  
[Pipeline] {  
[Pipeline] bat  
Warning: A secret was passed to "bat" using Groovy String interpolation, which is insecure.  
Affected argument(s) used the following variable(s): [DOCKER_HUB_PASSWORD]  
See https://jenkins.io/redirect/groovy-string-interpolation for details.  
  
C:\ProgramData\Jenkins\.jenkins\workspace\springSecurity-Build-Jenkins-Pipeline-  
PushToDockerHub>C:/Docker/resources/bin/docker login -u lamineoziljr -p ****  
WARNING! Using --password via the CLI is insecure. Use --password-stdin.  
Login Succeeded
```

Ceci est le résultat de notre dernier build qui est pipeline qui push notre image dans docker hub

```
b626401ef603: Mounted from library/openjdk
293d5db30c9f: Mounted from library/openjdk
03127cdb479b: Mounted from library/openjdk
9c742cd6c7a5: Mounted from library/openjdk
3fb2af6711b4: Pushed
v2: digest: sha256:9f5cab10bfb1350bee79aaf64062c6dfea646981296c9e825761c708b8da0375 size: 2007
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Dans ce projet free Style nous avons intégré l'usage du disque

✓ springSecurity-Build-Jenkins

 Ajouter une description

Désactiver le projet



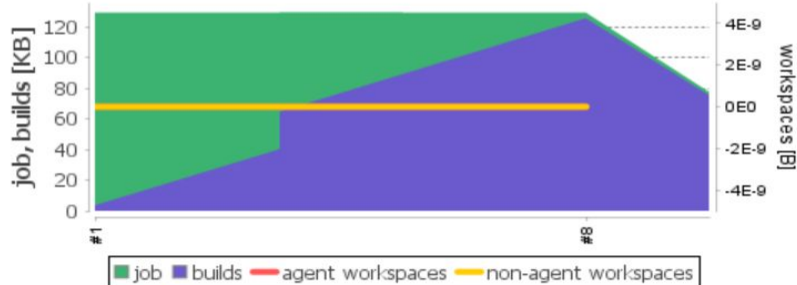
Utilisation du disque

Job	129 KB
All builds	129 KB
Locked builds	-
All workspaces	-
Agent workspaces	-
Non-agent workspaces	-

Liens permanents

- [Dernier build \(#8\), il y a 2 j 13 h](#)
- [Dernier build stable \(#8\), il y a 2 j 13 h](#)
- [Dernier build avec succès \(#8\), il y a 2 j 13 h](#)
- [Dernier build complété \(#8\), il y a 2 j 13 h](#)

Tendance d'utilisation du disque





GITLAB - CI

GitLab est une plateforme complète de gestion du cycle de vie des applications DevOps basée sur Git. GitLab offre des fonctionnalités telles que la gestion du code source, la CI/CD (Intégration Continue/Déploiement Continu), la gestion des versions, le suivi des problèmes, la planification des projets, la surveillance, et bien d'autres.

GitLab est souvent utilisé comme alternative à d'autres plates-formes de gestion de code source et de CI/CD telles que GitHub, Bitbucket, Jenkins, etc. L'une des principales caractéristiques de GitLab est qu'il fournit une suite complète d'outils intégrés dans un seul environnement, facilitant ainsi la collaboration et la gestion du cycle de vie du logiciel au sein d'une seule plateforme.

Voici notre projet gitlab créé dans le groupe CI-CD

The screenshot displays the GitLab interface for a project named 'springboot-gitlabci' under the 'CI-CD' group. The left sidebar contains navigation options like 'Project', 'Pinned', 'Issues', 'Merge requests', 'Manage', 'Plan', 'Code', 'Build', 'Secure', and 'Deploy'. The main content area shows the project name, a search bar, and a list of tabs: 'main', 'History', 'Find file', 'Edit', and 'Code'. Below these is a commit history table with columns 'Name', 'Last commit', and 'Last update'. The table lists files like '.idea', '.mvn/wrapper', 'src', and '.gitlab-ci.yml', all committed by 'initialization' 6 hours ago. To the right, 'Project information' shows 5 commits, 1 branch, 0 tags, and 542.1 MiB of storage. A progress bar is also visible.

CI-CD / springboot-gitlabci

S **springboot-gitlabci** 🔒

🔗 main ▾ History Find file Edit ▾ Code ▾

gitlab-cicd-springboot / + ▾

initialization
Lamine NDIAYE authored 6 hours ago ✓ 10414f9b

Name	Last commit	Last update
📁 .idea	initialization	6 hours ago
📁 .mvn/wrapper	initialization	8 hours ago
📁 src	initialization	6 hours ago
🔥 .gitlab-ci.yml	initialization	6 hours ago

Project information

🔗 5 Commits

🔗 1 Branch











🔗 0 Tags

💾 542.1 MiB Project Storage

🚀 CI/CD configuration

- + Add README
- + Add LICENSE
- + Add CHANGELOG
- + Add CONTRIBUTING
- + Add Kubernetes cluster

Nous avons créé nos deux variables

CI/CD Variables </> 2				Reveal values	Add variable
Key ↑	Value	Environments	Actions		
DOCKER_TOKEN  docker token Expanded	***** 	All (default) 	 		
DOCKER_USER  User Docker Expanded	***** 	All (default) 	 		

Ceci est notre fichier `.gitlab-ci.yml`

```
stages:
  - packaging
  - build_docker_image

default:
  image: maven:3.8.3-openjdk-17

variables:
  MAVEN_OPTS : "-Dmaven.repo.local=$CI_PROJECT_DIR/.m2/repository"

run unit test and package:
  inherit:
    default: true
    variables: true
  stage: packaging
  script:
    - mvn clean package -Dmaven.test.skip
  artifacts:
    paths:
```

```
stage: packaging
script:
  - mvn clean package -Dmaven.test.skip
artifacts:
  paths:
    - target/*.jar
cache:
  paths:
    - .m2/repository

# We'll build our docker image based on the dockerfile
build docker image:
  image: docker:latest
  stage: build_docker_image
  inherit:
    default: false
    variables: false
```

Ceci est notre fichier .gitlab-ci.yml

```
build docker image:
  image: docker:latest
  stage: build_docker_image
  inherit:
    default: false
    variables: false
  services:
    - docker:dind
  script:
    - docker build -t $DOCKER_USER/springboot-gitlabci:1.0 .
    - docker save $DOCKER_USER/springboot-gitlabci > springboot-gitlabci
    - docker login -u $DOCKER_USER -p $DOCKER_TOKEN
    - docker push $DOCKER_USER/springboot-gitlabci:1.0
  artifacts:
    paths: LamineOzilJr, Yesterday • initialization
    - springboot-gitlabci
  when: manual #must be run manually
```

Notre fichier décrit le pipeline d'intégration continue et de déploiement continu (CI/CD) pour notre projet Spring Boot, notamment les étapes de packaging, de construction d'une image Docker, et de déploiement manuel de cette image.

Stages :

- Définit deux étapes du pipeline : "**packaging**" et "**build_docker_image**".

Default Image :

- Spécifie l'image Docker par défaut utilisée dans le pipeline. Dans cet exemple, c'est une image Maven avec Java 17.

Variables :

- Définit une variable Maven pour le répertoire local des dépôts Maven.

Run Unit Test and Package :

- Configure l'étape de packaging qui exécute les tests unitaires et crée un package JAR de l'application Spring Boot.
- Les artefacts résultants (fichiers JAR) sont sauvegardés pour une utilisation ultérieure.

Build Docker Image :

- Configure l'étape de construction de l'image Docker.
- Utilise une image Docker pour exécuter le script de construction.
- Les artefacts résultants (l'image Docker) sont sauvegardés.
- Le déploiement de l'image Docker est configuré pour être déclenché manuellement, nécessitant une intervention humaine pour exécuter cette étape.

- L'étape "**build_docker_image**" utilise l'image **Docker:latest** et construit une image Docker étiquetée avec une version spécifiée.
- Cette étape utilise un service Docker-in-Docker (**docker:dind**) pour exécuter les commandes Docker nécessaires.
- Les artefacts résultants (l'image Docker) sont sauvegardés, et cette étape est configurée pour être déclenchée manuellement avec **when: manual**.

Juste après un **push** notre pipeline se déclenche et nos jobs automatiques s'exécutent. Par contre ceux manuels doivent être lancés manuellement.

CI-CD / springboot-githubcd / Pipelines / #1197835337

Initilization

✓ Passed Lamine NDIAYE created pipeline for commit 5d463e16 finished just now

For main

latest 2 jobs 0.59 35 seconds, queued for 0 seconds

Pipeline Needs Jobs 2 Tests 0

packaging

✓ run unit test and package

build_docker_image

⚙️ build docker image

CI-CD / springboot-githubcd / Pipelines / #1197835337

Initilization

✓ Passed Lamine NDIAYE created pipeline for commit 5d463e16 finished just now

For main

latest 2 jobs 1.95 1 minute 56 seconds, queued for 0 seconds

Pipeline Needs Jobs 2 Tests 0

packaging

✓ run unit test and package

build_docker_image

✓ build docker image

59 [INFO] Finished at: 2024-03-02T00:28:17Z

60 [INFO] -----

✓ 61 Saving cache for successful job

00:01

62 Creating cache default-1-non_protected...

63 .m2/repository: found 1921 matching artifact files and directories

64 Archive is up to date!

65 Created cache

✓ 66 Uploading artifacts for successful job

00:04

67 Uploading artifacts...

68 target/*.jar: found 1 matching artifact files and directories

69 WARNING: Upload request redirected location=https://gitlab.com/api/v4/jobs/6301886207/artifacts?artifact_format=zip&artifact_type=archive new-url=<https://gitlab.com>

70 WARNING: Retrying... context=artifacts-uploader error=request redirected

71 Uploading artifacts as "archive" to coordinator... 201 Created id=6301886207 responseStatus=201 Created token=glcvt-65

✓ 72 Cleaning up project directory and file based variables

00:00

73 Job succeeded

85 6be690267e47: Preparing

86 13a34b6fff78: Preparing

87 9c1b6dd6c1e6: Preparing

88 9c1b6dd6c1e6: Layer already exists

89 13a34b6fff78: Layer already exists

90 6be690267e47: Layer already exists

91 231884807cf9: Pushed

92 1.0: digest: sha256:dbb98b8c1570ecacd0e2e683755dc4a205268b9796863c70fd9a49976dc09c1b size: 1165

✓ 93 Uploading artifacts for successful job

00:22

94 Uploading artifacts...

95 springboot-gitlabciid: found 1 matching artifact files and directories

96 WARNING: Upload request redirected location=https://gitlab.com/api/v4/jobs/6301886208/artifacts?artifact_format=zip&artifact_type=archive new-url=<https://gitlab.com>

97 WARNING: Retrying... context=artifacts-uploader error=request redirected


98 Uploading artifacts as "archive" to coordinator... 201 Created id=6301886208 responseStatus=201 Created token=glcvt-65

✓ 99 Cleaning up project directory and file based variables


00:00

100 Job succeeded

Voici notre image pushé dans docker Hub



lamineoziljr/springboot-gitlabci 

Updated about 7 hours ago

This repository does not have a description 

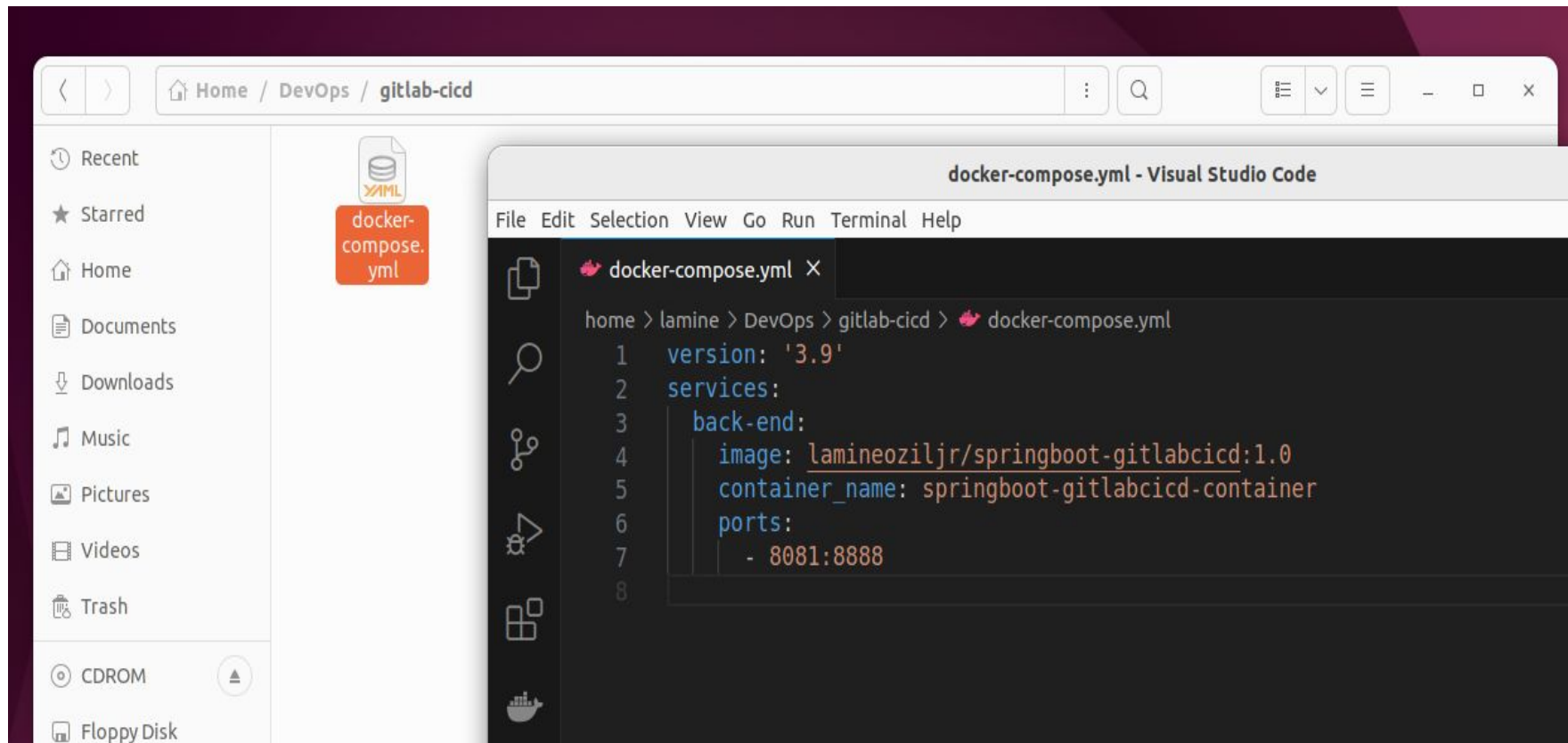
Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
 1.0		Image	an hour ago	7 hours ago

[See all](#)

Maintenant déployons notre projet en production. Après avoir installé docker sous ubuntu voici notre docker compose



On se place dans le répertoire de notre fichier et on lance la commande **docker compose up -d** NB: préfixer de sudo si on est pas en mode root

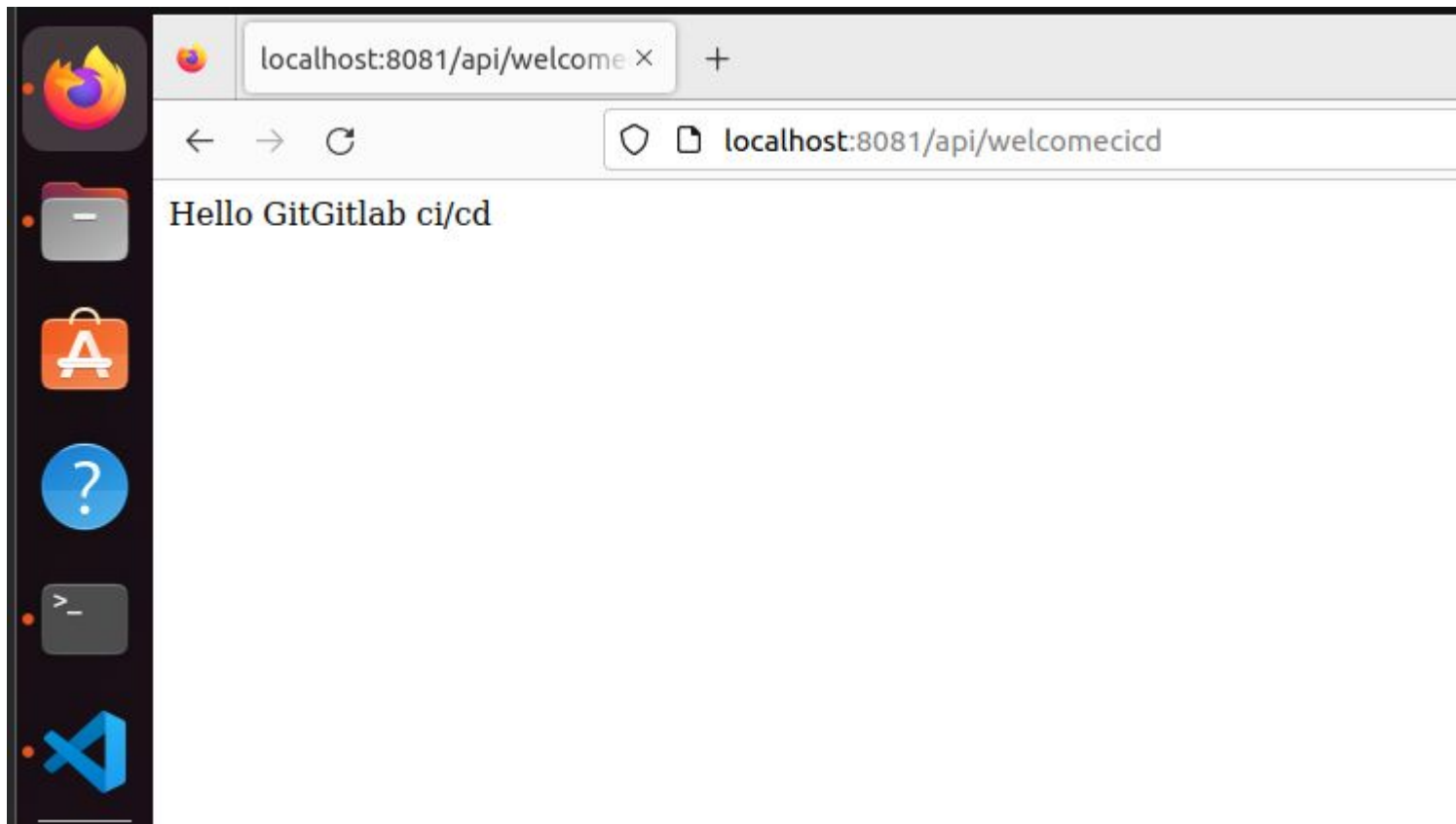
```
lamine@lamine-virtual-machine:~/DevOps/gitlab-cicd$ sudo docker-compose up -d
[sudo] password for lamine:
[+] Running 5/5
  ✓ back-end 4 layers [██████] 0B/0B Pulled
    ✓ 1fe172e4850f Pull complete
    ✓ 44d3aa8d0766 Pull complete
    ✓ 6ce99fdf16e8 Pull complete
    ✓ 5bc6a541da51 Pull complete
[+] Running 2/2
  ✓ Network gitlab-cicd_default Created
  ✓ Container springboot-gitlab-cicd-container Started
lamine@lamine-virtual-machine:~/DevOps/gitlab-cicd$
```

On voit que notre conteneur est bien démarré. On fait un **docker ps** pour checker

```
lamine@lamine-virtual-machine:~/DevOps/gitlab-cicd$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
c30827b6a5ba	lamineoziljr/springboot-gitlab-cicd:1.0	"java -jar springboo..."	53 seconds ago	Up 49 seconds	0.0.0.0:8081->8888/tcp, :::8081->8888/tcp

Enfin on accède a notre appli via : **<http://localhost:8081/api/welcomecid>**





FIN