

Rapport du projet

Membres :

Birane Kebe

Mame Saye Kebe

Saliou Thioune

Encadreur : Mr Samb

Sommaire:

I. Introduction

II. Creation du projet

- Création du projet java standard avec Maven
- Ajout des dépendances de JUnit
- Création de la class **Calculator** dans le package **implementation**
- Création de la class **TestUnit** dans le package **testUnitaires**
- Génération du dossier target contenant le fichier projetateliergl-1.0-SNAPSHOT.jar
- Création du dockerfile

III. Installation de docker

IV. Publication du code sur github

- Ajout du projet local (projetateliergl) sur GitHub Desktop
- Pousser ce projet (repository) local vers gitHub
- Aperçu du projet sur GitHub
- Ajout des collaborateurs
- Création des branches

V. Mis en place de l'outil d'intégration continue et publication de notre application Java sous forme d'image sur le docker hub avec github actions

- Création du repository m1sir sur dockerhub
- Création d'un Access Token sur notre compte dockerhub
- Création de secrets sur github
- Creation du workflow CiDuProjet.yml

VI. Résultats

- Les tests unitaires
- Publication de l'image docker sur le repository `mamesayekebe/m1sir`

VII. Conclusion

I. Introduction

L'intégration continue est la pratique d'intégrer les changements des différents développeurs de l'équipe sur la branche principale le plus tôt possible. Dans le meilleur des cas, l'intégration se fait plusieurs fois par jour.

Cela permet de s'assurer que le code sur lequel un développeur travaille n'est pas trop différent de celui des autres.

Cette pratique devrait être accompagnée de tests automatisés.

L'intégration continue repose souvent sur la mise en place d'une brique logicielle permettant l'automatisation de tâches :

- Compilation,
- Tests unitaires et fonctionnels,
- Validation du produit,
- Tests de performances...

À chaque changement du code, cette brique logicielle va exécuter un ensemble de tâches et produire un ensemble de résultats, que le développeur peut par la suite consulter.

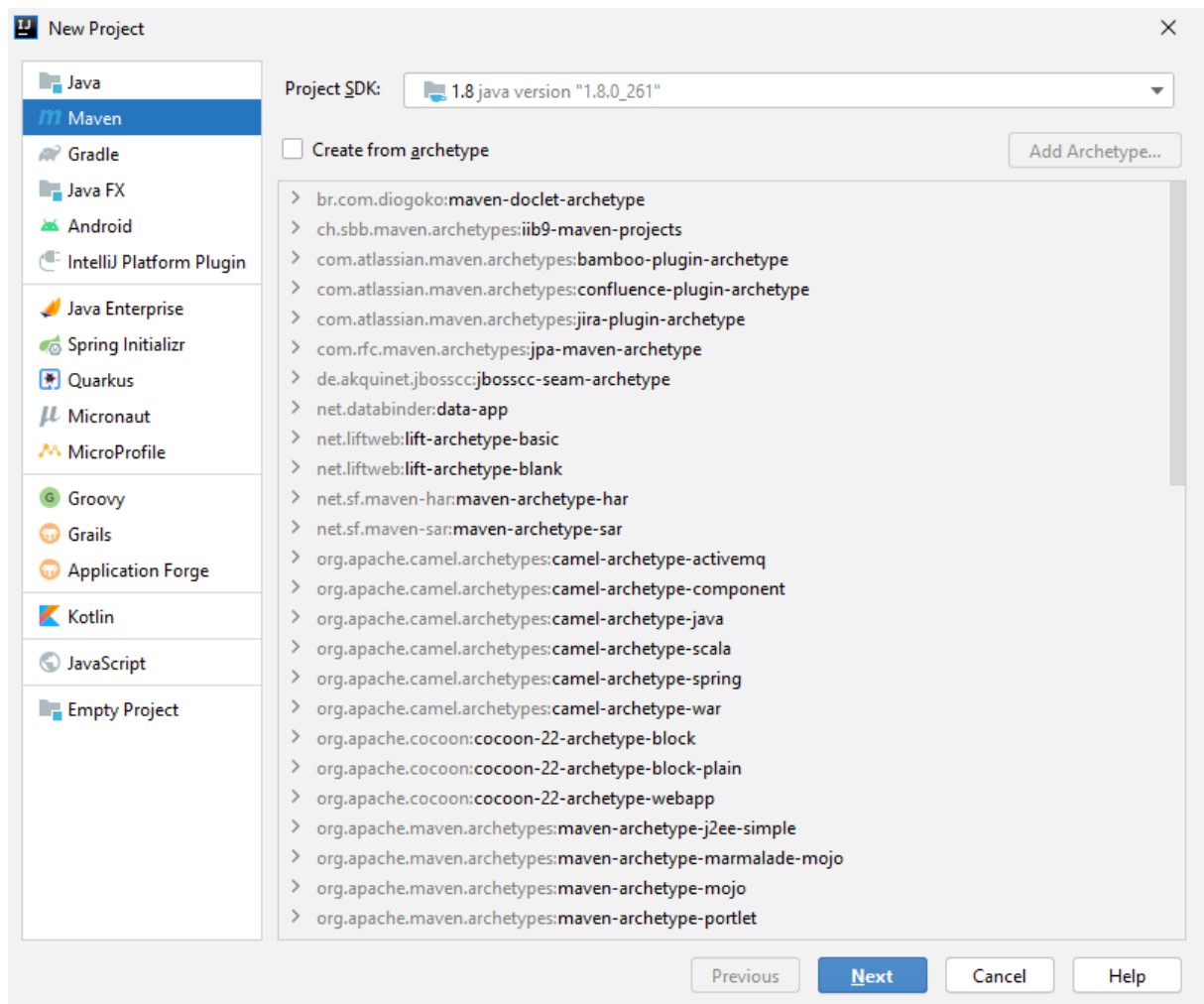
Cette intégration permet ainsi de ne pas oublier d'éléments lors de la mise en production et donc ainsi améliorer la qualité du produit.

Dans ce projet nous mettrons en place les **tests unitaires** ainsi que la **création** d'une **image docker de notre application** que nous **publierons sur dockerhub** grâce à **github Actions**.

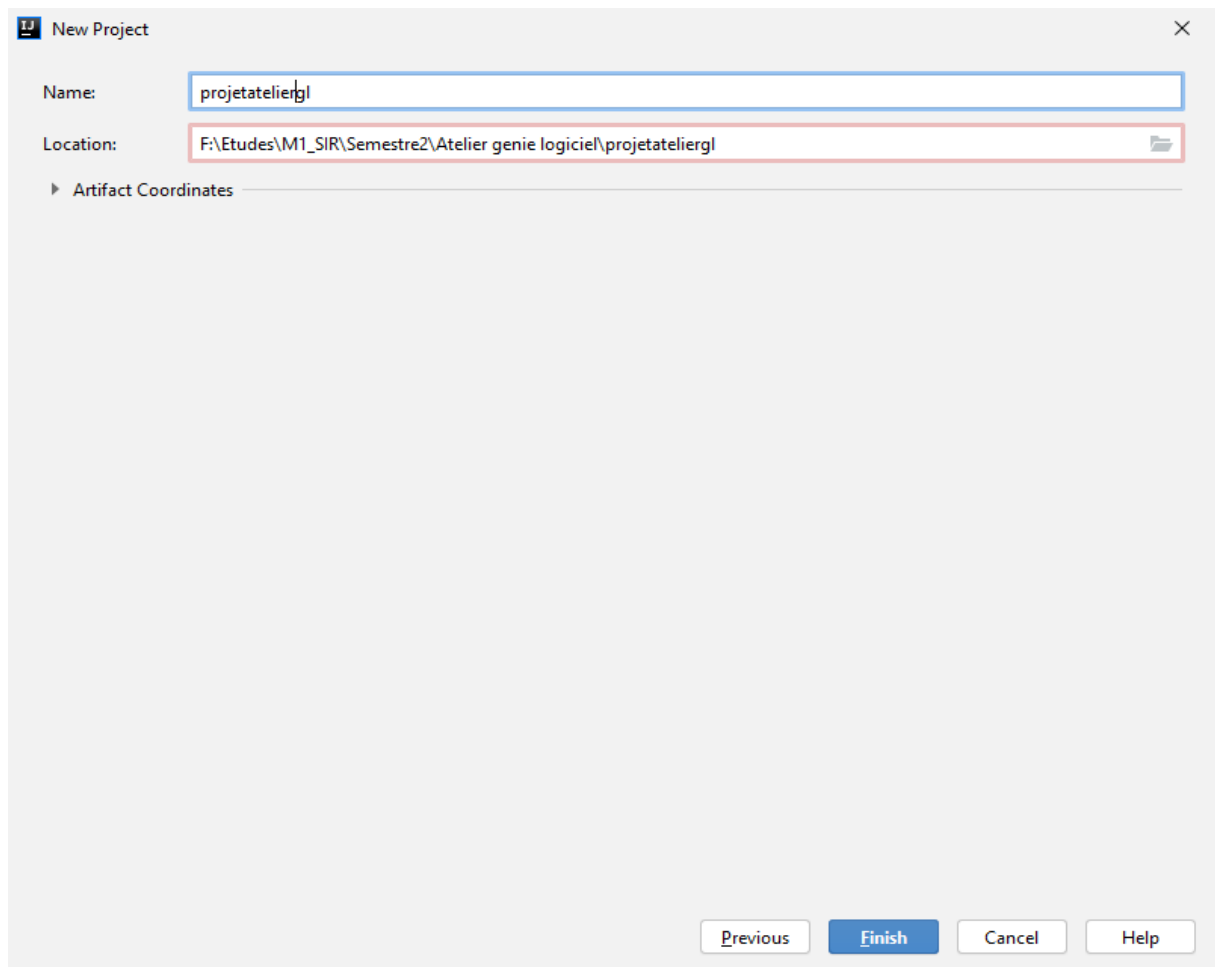
II. Création du projet

- **Création du projet java standard avec Maven**

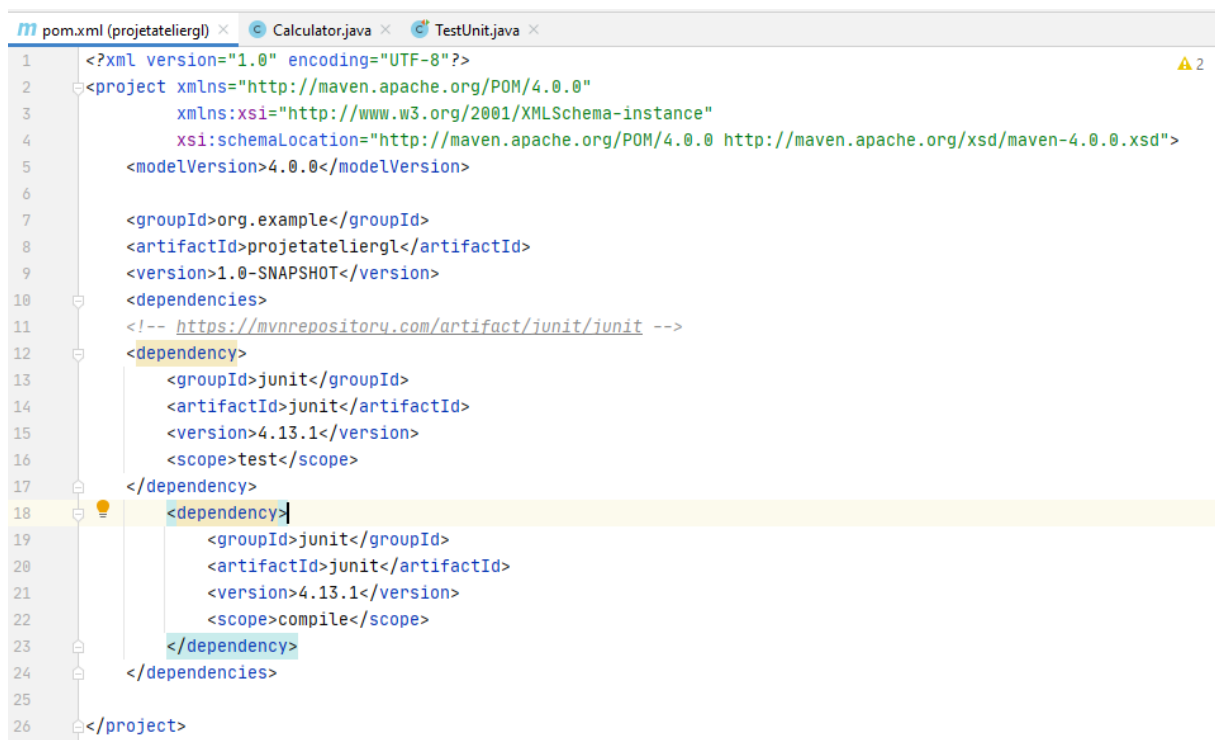
Sur IntelliJ sélectionnons new projet ->Maven



Notre projet s'appellera **projetatelierrgl**



- Ajoutons les dépendances de JUnit version 4.13.1 dans le fichier pom.xml



- Création de la class Calculator dans le package implementation

Créons dans le dossier `src/main/java` le **package implementation** pour contenir notre class **Calculator** dans laquelle se trouvent nos méthodes **sum**, **minus**, **divide**, **multiply**, **max**, **min**, **maxElement** et **minElement**

```
package implementation;

public class Calculator {
    public int sum(int a, int b) {
        return a+b;
    }
    public int minus(int a, int b) {
        return a-b;
    }

    public int divide(int a, int b) throws ArithmeticException{
        if(b==0) throw new ArithmeticException("La division par 0 est impossible");
        return a/b;
    }

    public int multiply(int a, int b) {
        return a*b;
    }

    public int min(int a, int b) {
        if(a<=b)
            return a;
        else
            return b;
    }
    public int max(int a, int b) {
        if(a>=b)
            return a;
        else
            return b;
    }

    public int minElement(int[] list) {
        int min = list[0];
        for (int i = 1; i < list.length; i++) {
            if (list[i] < min)
                min = list[i];
        }
        return min;
    }
    public int maxElement(int[] list) {
        int max=list[0];
        for(int i=1;i<list.length;i++) {
            if(list[i]>max)
                max=list[i];
        }
        return max;
    }
}
```

- Création de la class **TestUnit** dans le package **testUnitaires**

Créons dans le dossier **src/test/java** le **package testUnitaires** pour contenir notre class **TestUnit** dans laquelle se trouvent nos **tests unitaires**

```
package testUnitaires;

import implementation.Calculator;
import org.junit.Assert;
import org.junit.Test;

public class TestUnit {
    Calculator cal=new Calculator();

    @Test
    public void testSumResAttenduEgal5(){
        int sum=cal.sum( a: 5, b: 0);
        Assert.assertTrue( condition: sum==5);
    }

    @Test
    public void testMinusResAttenduEgal2(){
        int dif=cal.minus( a: 3, b: 1);
        Assert.assertFalse( condition: dif!=2);
    }

    @Test
    public void testDivideResAttenduEgal3(){
        int q=cal.divide( a: 9, b: 3);
        Assert.assertEquals(q, actual: 3);
    }

    @Test(expected=ArithmeticException.class)
    public void testDivideArithmeticExceptionAttenduEgal() throws ArithmeticException{
        int q=cal.divide( a: 9, b: 0); //Cette ligne doit declancher une exception de type ArithmeticException
    }
}
```

```

@Test
public void testMultiplyResAttenduEgal10(){
    int mul=cal.multiply( a: 5, b: 2);
    Assert.assertEquals(mul, actual: 10);
}

@Test
public void testMinResAttenduEgal2(){
    int min=cal.min( a: 5, b: 2);
    Assert.assertTrue( condition: min==2);
}

@Test
public void testMaxResAttenduEgal5(){
    int max=cal.max( a: 5, b: 2);
    Assert.assertFalse( condition: max!=5);
}

int [] list= {-1,-5,2,5,8,15};
@Test
public void testMinElementResAttenduEgal/*-5*/(){
    int minElmt=cal.minElement(list);
    Assert.assertFalse( condition: minElmt!=-5);
}

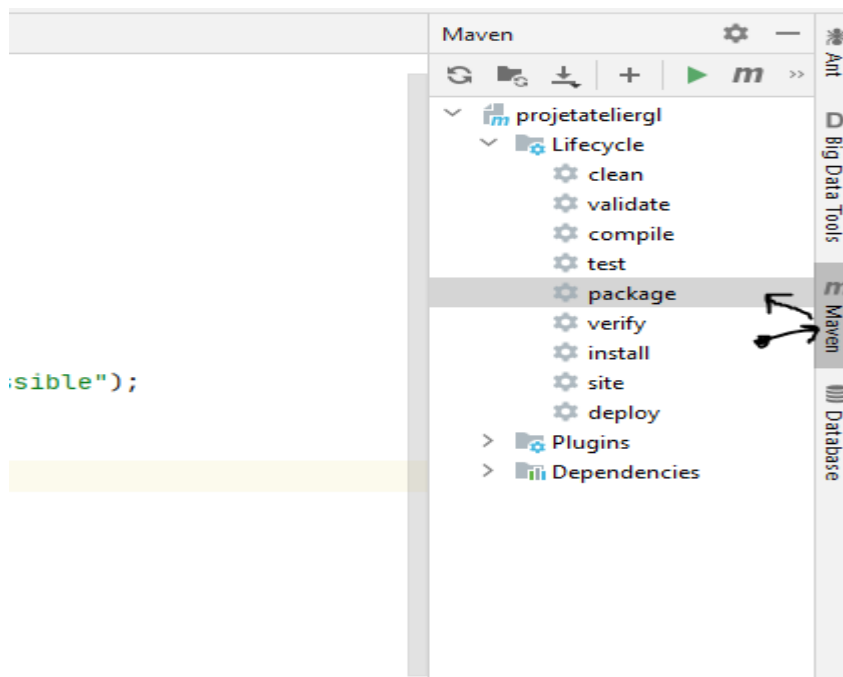
@Test
public void testMaxElementResAttenduEgal15(){
    int maxElmt=cal.maxElement(list);
    Assert.assertTrue( condition: maxElmt==15);
}
}

```

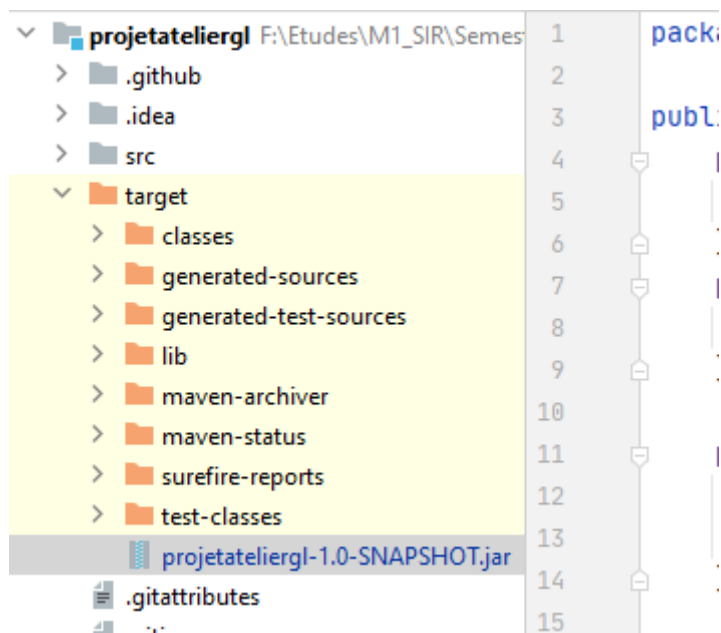
- Génération du dossier target contenant le fichier projetatelierngl-1.0-SNAPSHOT.jar

Générons le dossier target contenant le fichier projetatelierngl-1.0-SNAPSHOT.jar qui nous sera plus tard utile pour la mise en place d'un outil d'intégration continue.

Sur IntelliJ cliquons sur maven puis package



Résultat :



- Création du dockerfile

```

    }

    @Test
    public void test() {
        int [] list= {-1,-5,2,5,8,15};
    }
}

```

Voici le contenu du dockerfile

```

FROM openjdk:8-jdk-alpine

ARG JAR_FILE=target/projetatelierngl-1.0-SNAPSHOT.jar
ARG JAR_LIB_FILE=target/lib/

# cd / usr / local / runme
WORKDIR /usr/local/runme

##copier target /find-links.jar/usr/local/runme/app.jar
COPY ${JAR_FILE} app.jar

# copier les dépendances du projet
# cp -rf cible / lib / / usr / local / runme / lib
##AJOUTER
ADD ${JAR_LIB_FILE} lib/

#java -jar /usr/local/runme/app.jar
ENTRYPOINT [ "java" , "-jar" , "app.jar" ]

```

III. Installation de docker sur windows

Configuration

- ☒ Enable Hyper-V Windows Features
- ☒ Install required Windows components for WSL 2
- ☒ Add shortcut to desktop

Ok

Docker Desktop 2.4.0.0

Unpacking files...

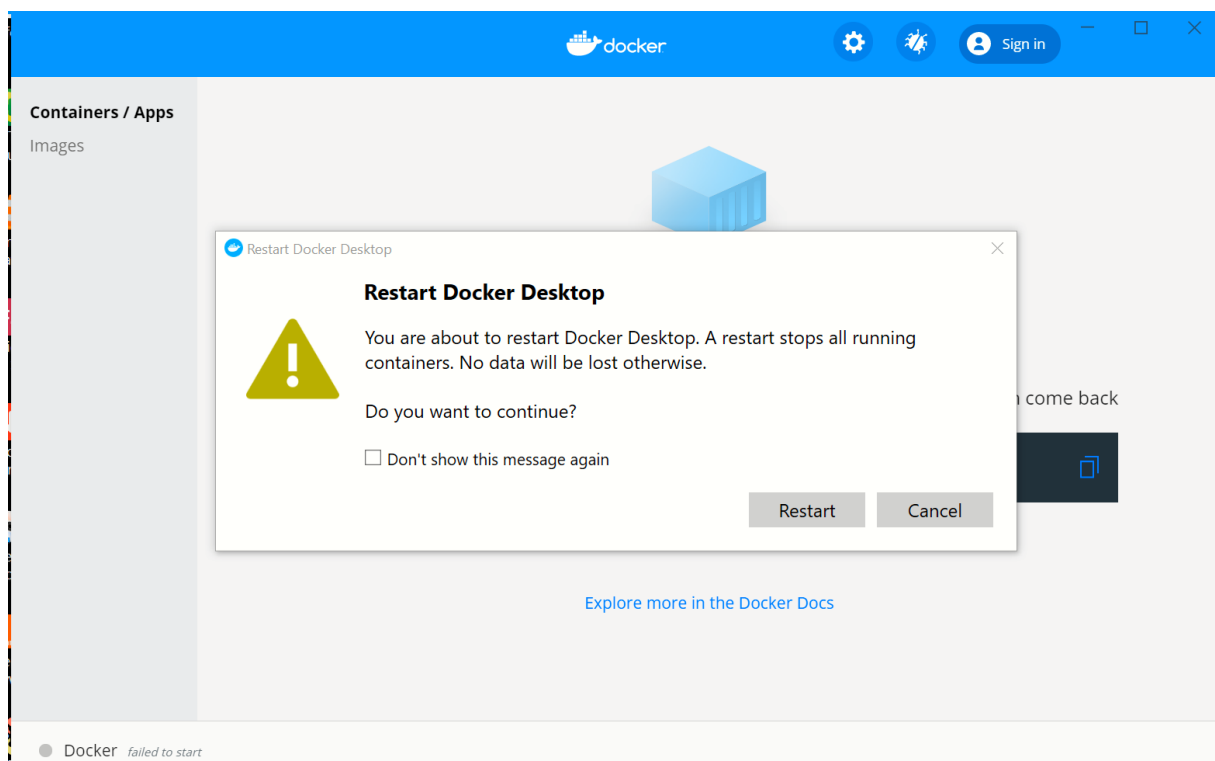
Unpacking file: resources/docker-desktop.iso
Unpacking file: resources/docker
Unpacking file: resources/ddvp.ico
Unpacking file: resources/config-options.json
Unpacking file: resources/componentsVersion.json
Unpacking file: resources/CHANGELOG.md
Unpacking file: resources/bin/docker-compose
Unpacking file: resources/.gitignore
Unpacking file: InstallerCli.pdb
Unpacking file: InstallerCli.exe.config
Unpacking file: frontend/vk_swiftshader_icd.json
Unpacking file: frontend/v8_context_snapshot.bin
Unpacking file: frontend/snapshot_blob.bin
Unpacking file: frontend/resources/app.asar.unpacked/node_modules/ssh2/util/pagent.c

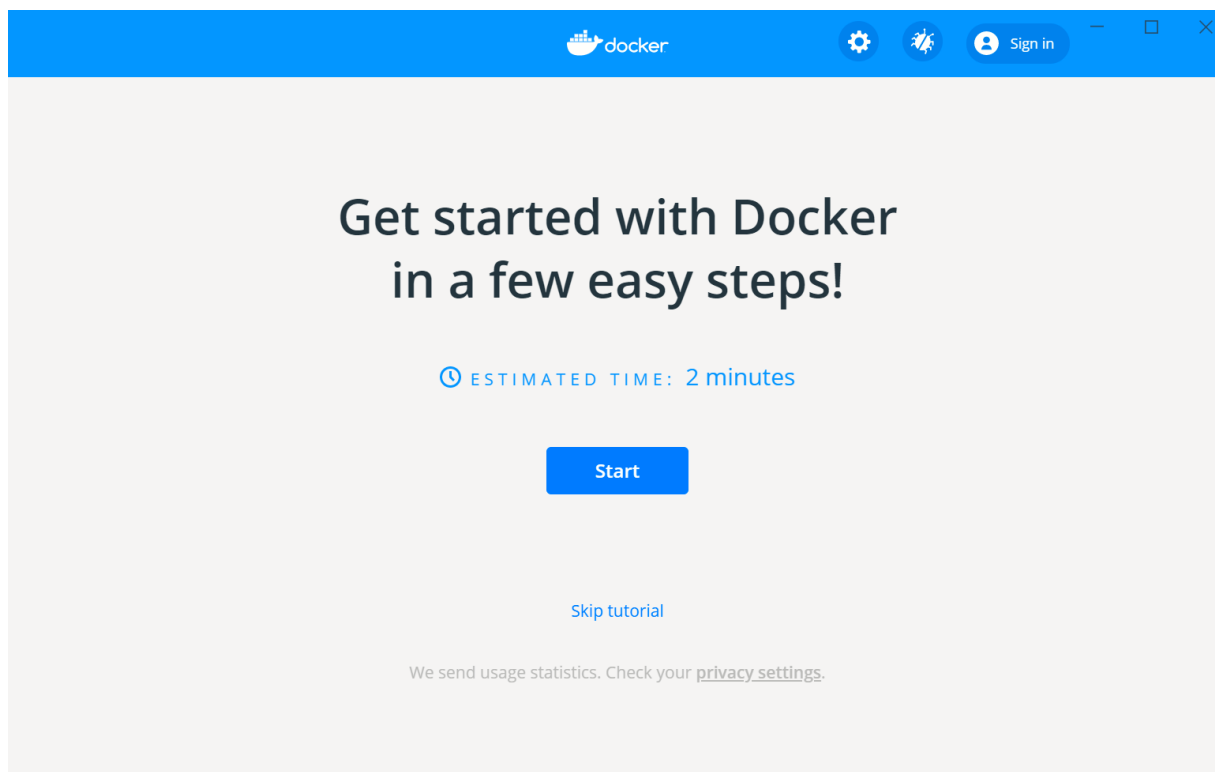
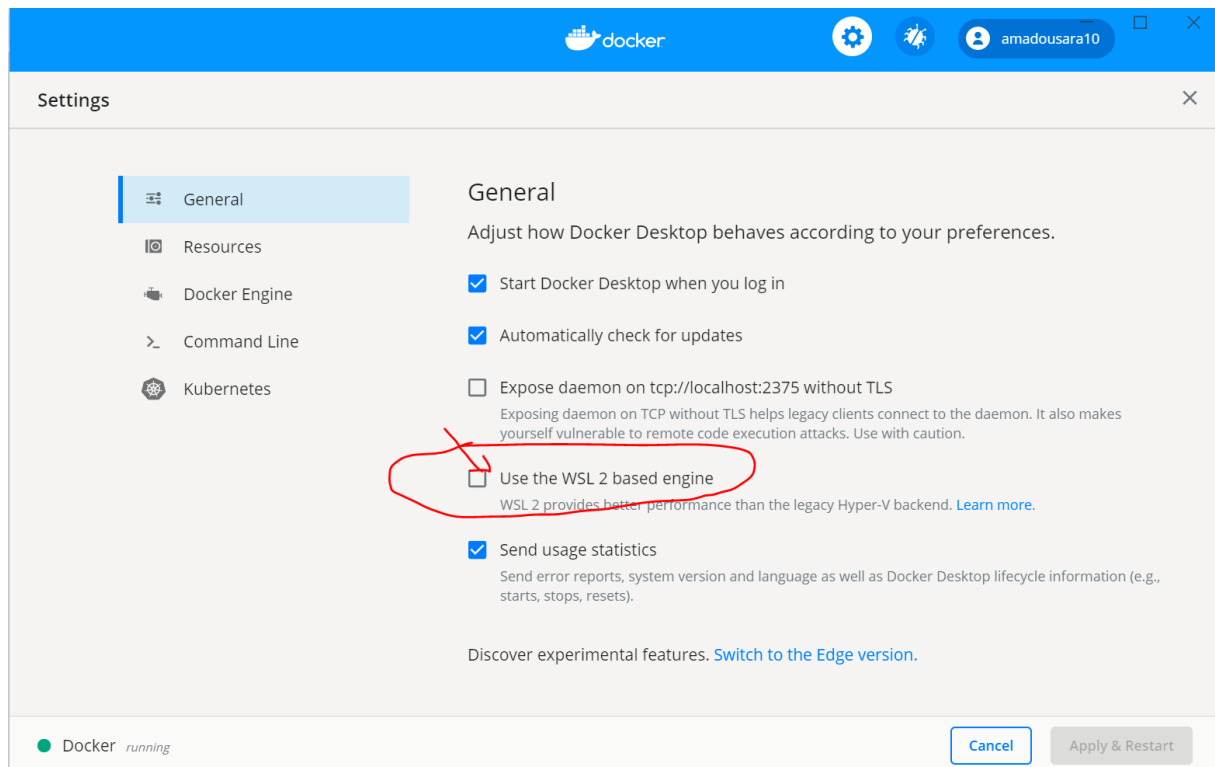
Docker Desktop 2.4.0.0

Installation succeeded

You must restart Windows to complete installation.

Close and restart





1 Clone

First, clone a repository

The *Getting Started* project is a simple GitHub repository which contains everything you need to build an image and run it as a container.

Clone the repository by running Git in a container.

```
docker run --name repo alpine/git clone \
https://github.com/docker/getting-started.git
docker cp repo:/git/getting-started/ .
```

You can also type the command directly in a command line interface.

[Next Step](#)

[Skip tutorial](#)

Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.
Testez le nouveau système multiplateforme PowerShell
11 https://aka.ms/pscore6
PS C:\Users\pc> docker run --name repo alpine/git
clone https://github.com/docker/getting-started.git
t
Unable to find image 'alpine/git:latest' locally
[]

Containers / Apps

Images

No containers running

Try running a container: Copy and paste this command into your terminal and then come back

```
docker run -d -p 80:80 docker/getting-started
```

[Explore more in the Docker Docs](#)

● Docker *running*

Installation fait!

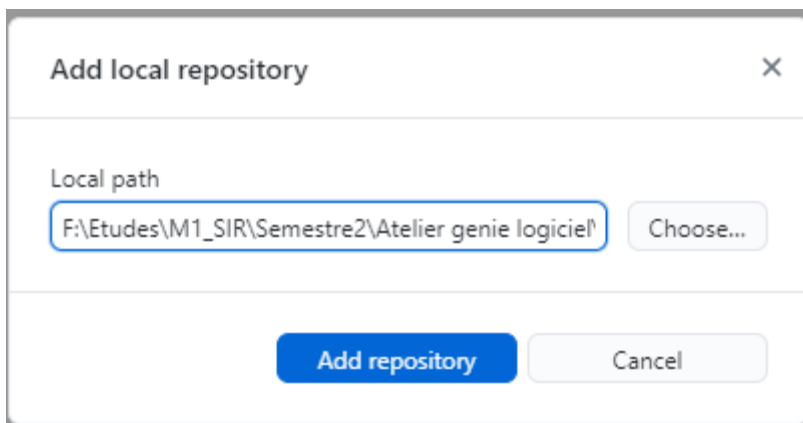
Cliquons sur **Sign in** et **loggons** nous avec notre **login** (**mamesayekebe**) et notre **mot de passe** pour acceder a notre compte dockerhub.

IV. Publication du code sur github

Nous utiliserons GitHub Desktop pour les interactions entre notre repository local et celui qui se trouve github.

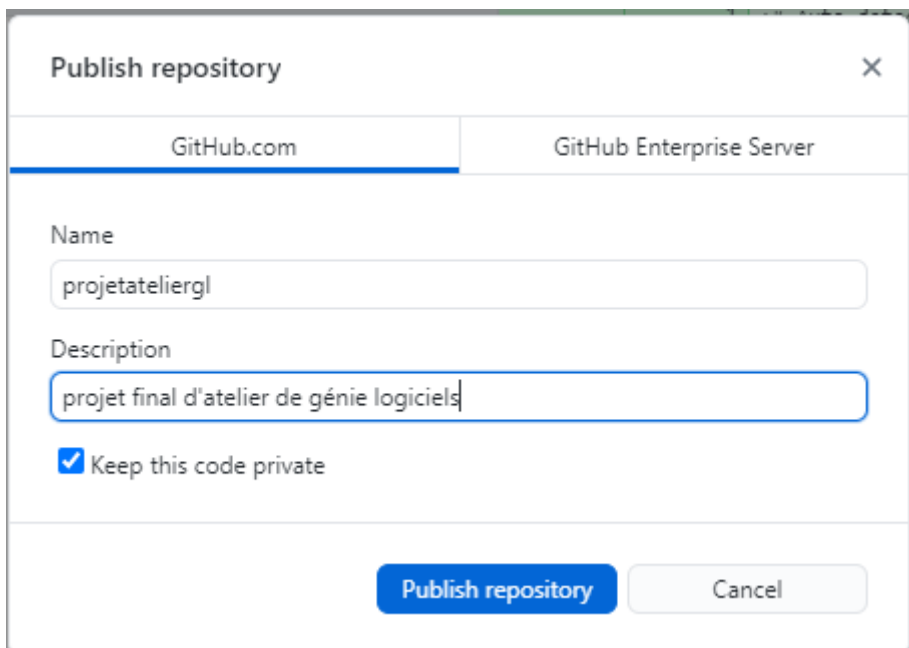
- Ajout du repository local (projetateliergl) sur GitHub Desktop

Sur GitHub Desktop nous allons au sous menu file->Add local repository. Nous accédons à cette fenetre pour ajouter notre nouveau repository local (projetateliergl).

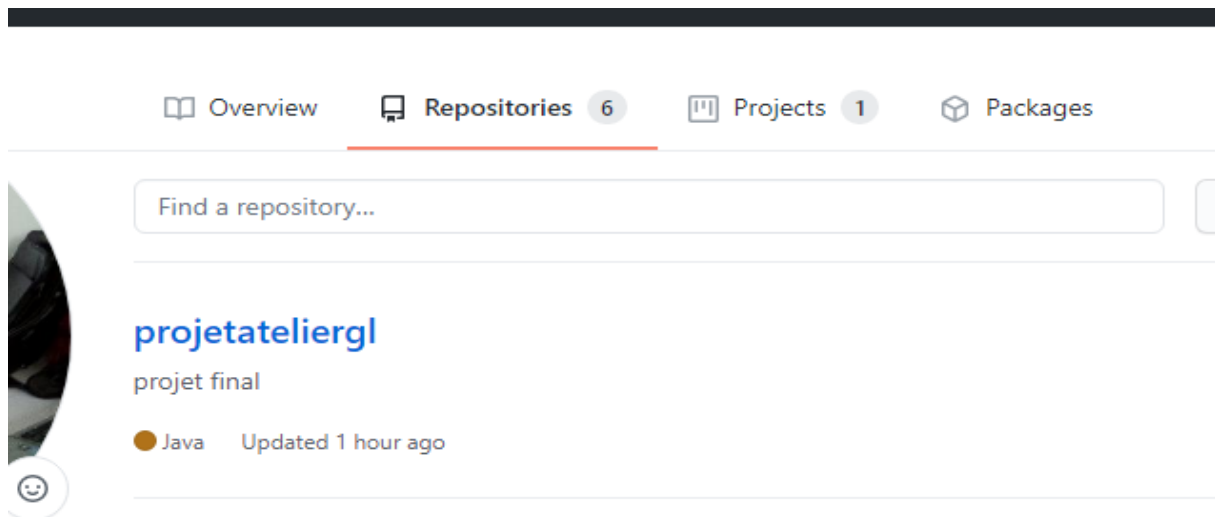


- Pousser ce repository local vers gitHub

Pour cela allons au sous menu Repository->Push



Nous allons nous rendre sur <https://github.com/birane012?tab=repositories> pour nous assurer que notre projet (projetateliergl) y est bien poussé.



- Aperçu du projet sur GitHub

File	Commit	Time
.idea	commit	yesterday
src	update calculator class	yesterday
target	lib folder do not have to be empty	19 hours ago
.gitattributes	Initial commit	4 days ago
.gitignore	pom security error rectification	3 days ago
Dockerfile	mame commit	3 days ago
README.md	Update README.md	4 days ago
pom.xml	pom security error rectification	3 days ago
projetateliergl.iml	Initial commit	4 days ago

README.md

projetateliergl

projet final d'atelier de génie logiciels

Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

Contributors 3

- birane012 Birane KEBE
- seynaboumame Mame Saye Kebe
- saliouthioune

Languages

Java 84.8% Dockerfile 15.2%

- Ajout des collaborateurs


Ajoutons les collaborateurs qui sont Saliou Thioune et Mame Saye Kebe

Pour ce fait sur <https://github.com/birane012/projetateliergl> allons sur

Settings-> Manage access->invite a collaborator.


☐ Select all

☐



saliouthioune
Collaborator

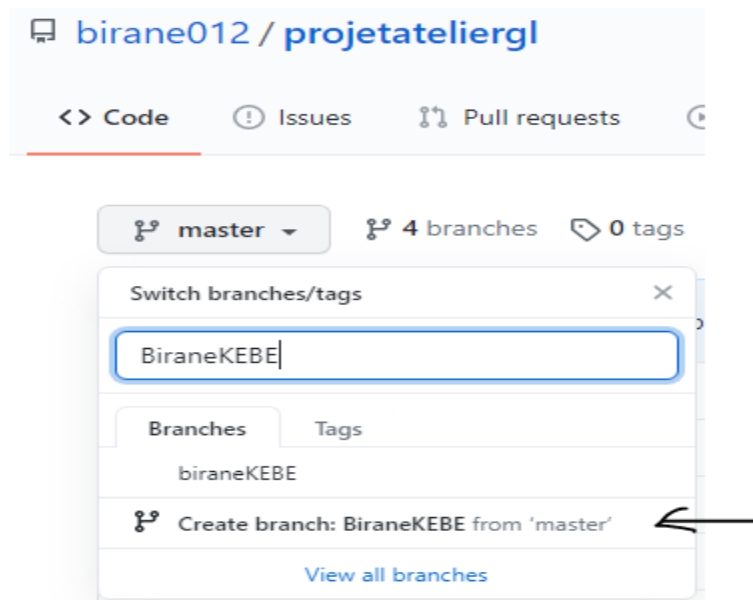
☐



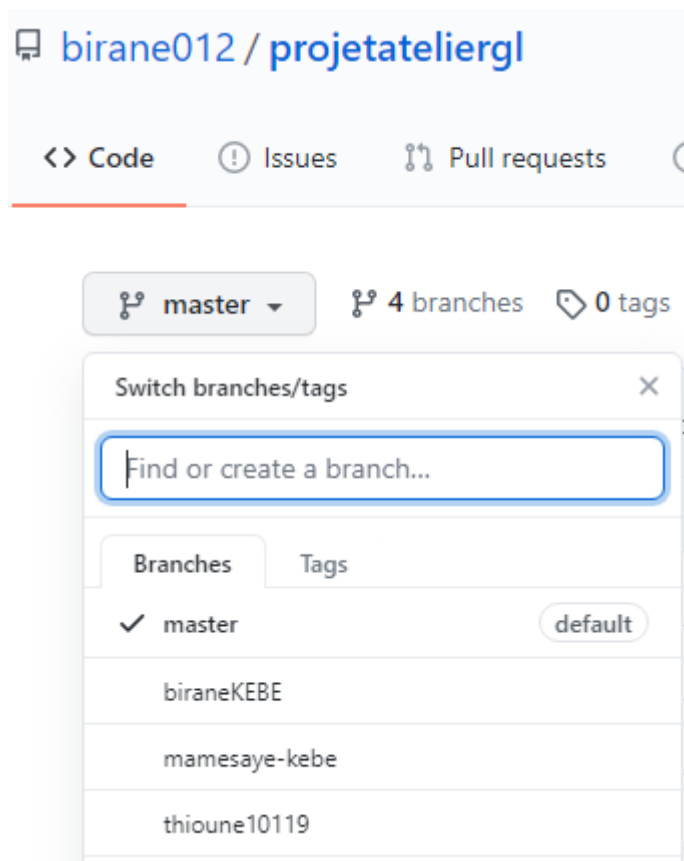
Mame Saye Kebe
seynaboumame • Collaborator

- Création des branches

A présent chaque collaborateur à créer sa propre branche de la manière suivante :



Liste des branches des différents collaborateurs.

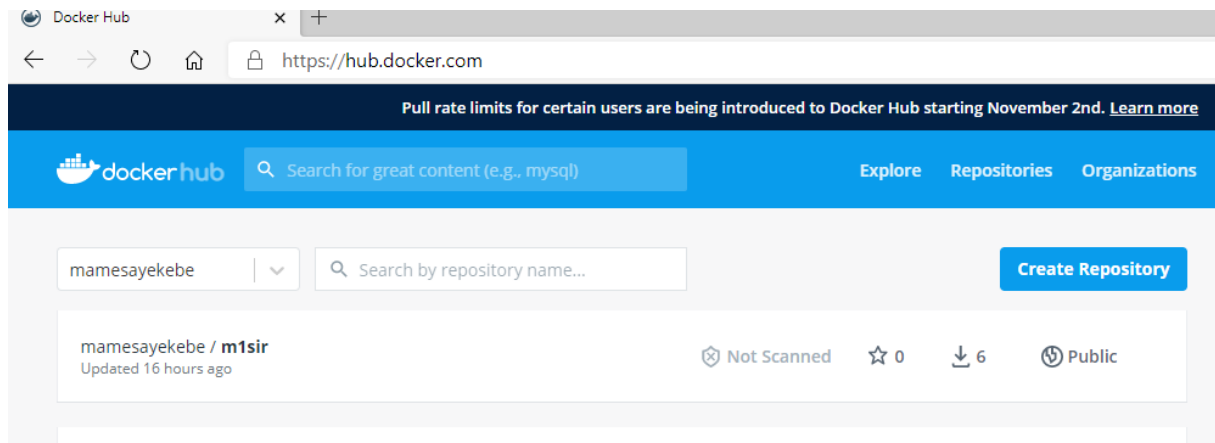


V. Mis en place de l'outil d'intégration continue et publication de notre application Java sous forme d'image sur le docker hub avec github actions

- Création du repository m1sir sur dockerhub

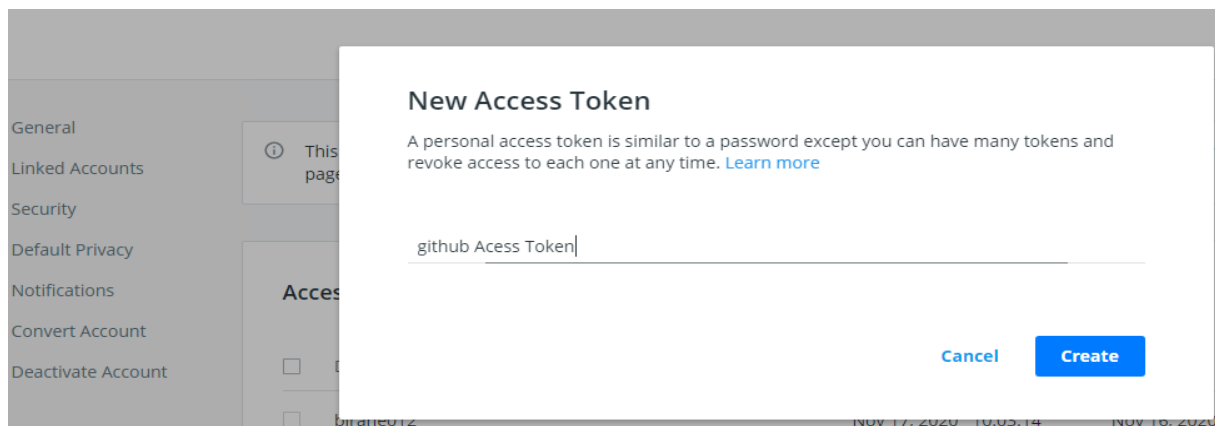
Rendons nous sur <https://hub.docker.com/u/mamesayekebe>

et y créer le repository **m1sir** pour contenir l'image docker de notre application que nous allons créer depuis github actions.



- Creation d'un Access Token sur notre compte dockerhub

Ensuite nous allons **générer** une **Access Token** sur docker pour permettre à notre **compte gitHub** de publier l'**image** sur le **repository m1sir** que nous venons de créer sur **docker**.



Copy Access Token

When logging in from your Docker CLI client, use this token as a password. [Learn more](#)

ACCESS TOKEN DESCRIPTION

github Access Token

To use the access token from your Docker CLI client:

- 1.Run `docker login --username mamesayekebe`
2. At the password prompt, enter the personal access token.

`alc7fd28.3aad.4753.899b.339b4535989e`

WARNING: This access token will only be displayed once. It will not be stored and cannot be retrieved. Please be sure to save it now.

Copy and Close

- Création de secrets sur github

Nous allons à présent nous rendre sur notre repository github

<https://github.com/birane012/projetateliergl/settings/secrets/actions> et créer deux secrets l'un pour contenir notre nom d'utilisateur dockerhub et l'autre pour contenir le token ci-dessus.

The screenshot shows the 'Secrets' section of a GitHub repository's settings. On the left is a sidebar with navigation links: Options, Manage access, Security & analysis, Branches, Webhooks, Notifications, Integrations, Deploy keys, Actions, Secrets (highlighted), and Moderation settings. The main area is titled 'Secrets' and includes a 'New repository secret' button. Below this, it explains that secrets are encrypted environment variables. A list of secrets is shown with two entries: 'DOCKER_PASSWORD' and 'DOCKER_USERNAME', both updated yesterday, with 'Update' and 'Remove' buttons for each.

- Creation du workflow CiDuProjet.yml :

Pour ce faire nous allons nous rendre sur

<https://github.com/birane012/projetateliergl/actions>

The screenshot shows the 'Actions' tab of a GitHub repository. The 'Actions' tab is circled in the navigation bar. Below the navigation bar, there's a section titled 'Choose a workflow template' with a brief description and a link to 'Skip this and set up a workflow yourself'. Underneath, 'Workflows made for your Java repository' are listed, with a 'Suggested' tag. Two workflow templates are visible: 'Scala' and 'Java with Maven'. The 'Java with Maven' template is highlighted with a black arrow. Both templates show a 'Set up this workflow' button and a preview of the workflow steps.

projetateliargl / .github / workflows / CiDuProjet.yml Cancel

```
<> Edit new file Preview Spaces 2 No wrap
1 name: Java CI with Maven
2
3 on:
4   push:
5     branches: [ master ]
6   pull_request:
7     branches: [ master ]
8
9 jobs:
10  build:
11
12    runs-on: ubuntu-latest
13    steps:
14      - uses: actions/checkout@v2
15      - name: Set up JDK 1.8
16        uses: actions/setup-java@v1
17        with:
18          java-version: 1.8
19      - name: test
20        run: mvn test
21
22      - name: Build with Maven
23        run: mvn -B package --file pom.xml
24      - name: Build and Push Docker Image
25        uses: mr-smithers-excellent/docker-build-push@v4
26        with:
27          image: mamesayekebe/mlsir
28          registry: docker.io
29          username: ${ secrets.DOCKER_USERNAME }}
30          password: ${ secrets.DOCKER_PASSWORD }}
```

Faisons un comite pour terminer la création de ce fichier

Start commit

Commit new file

Create CiDuProjet.yml

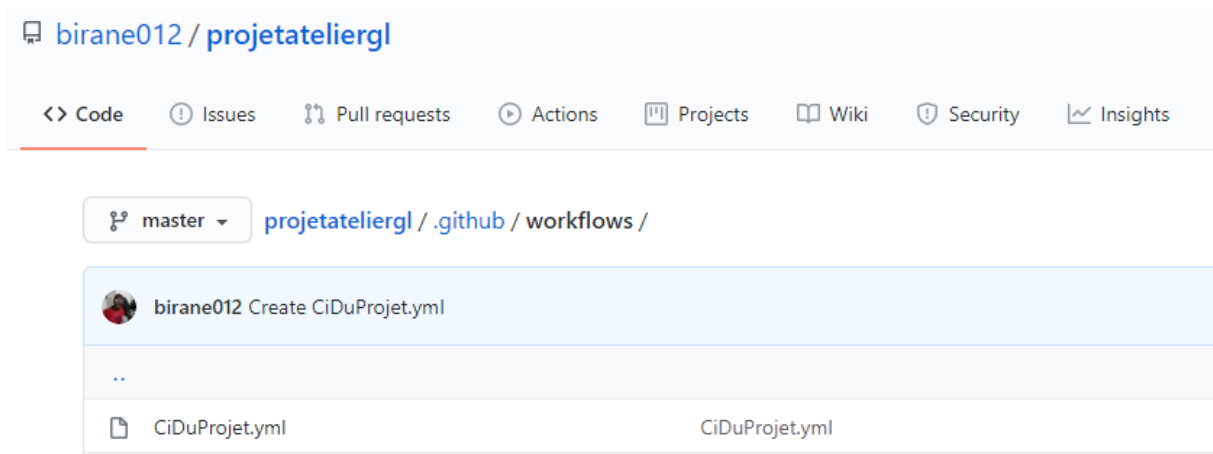
Add an optional extended description...

☒ Commit directly to the master branch.

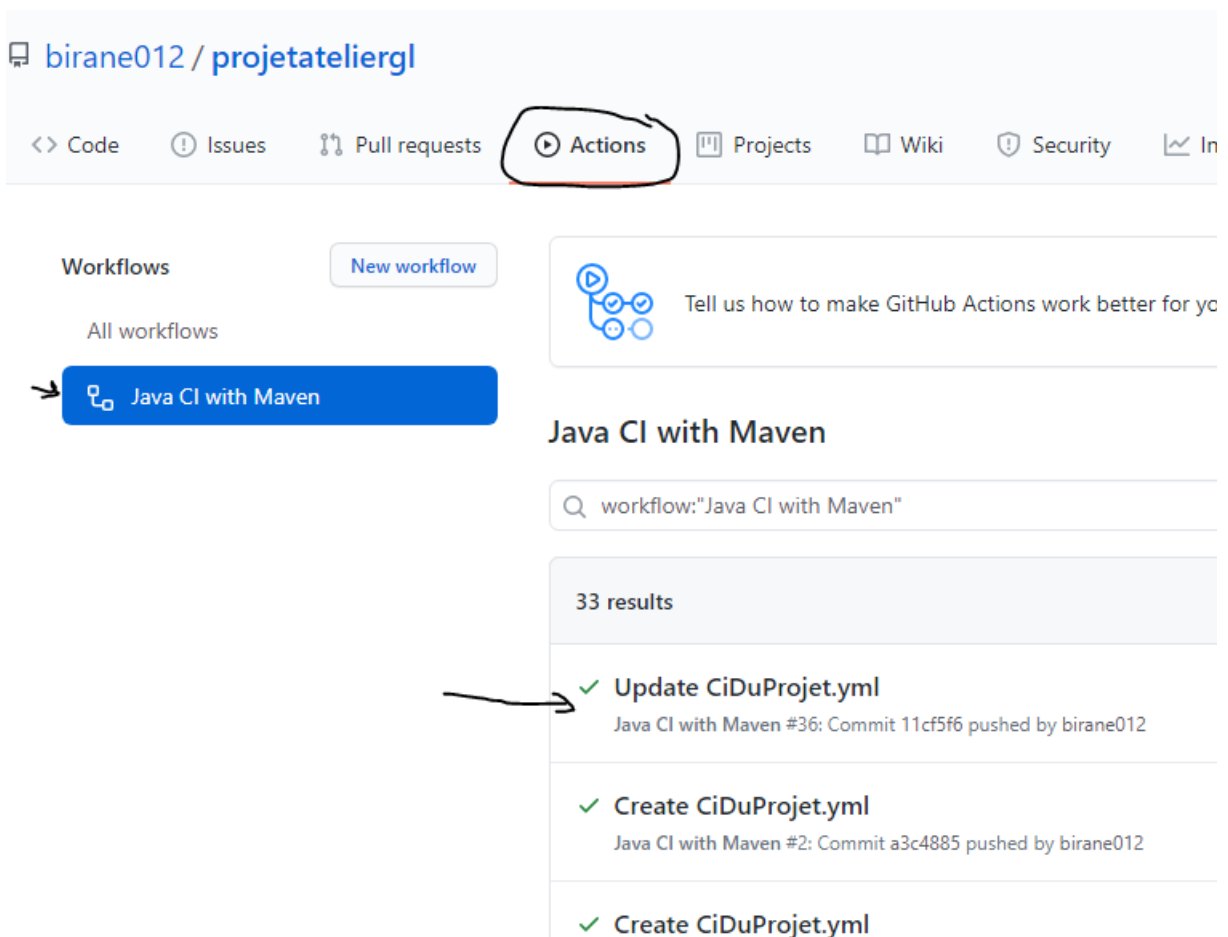
☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit new file

Le fichier CiDuProjet.yml est bien créer :



Rendons-nous sur Actions pour voir si les tests ainsi que la publication de l'image de notre application sur dockerhub ont bien été effectuer.



VI. Résultats :

- Les tests unitaires

birane012 / projetateliergl

Unwatch 1

<> Code ! Issues 🔄 Pull requests ▶ Actions 📁 Projects 📖 Wiki 🛡 Security 📊 Insights ⚙ Settings

✓ Update CiDuProjet.yml
master 686279d

Java CI with Maven
on: push

1

✓ build

build

succeeded 4 minutes ago in 51s

Search logs

✓ Les tests unitaires

3354 [INFO] Running testUnitaires.TestUnit
3355 [INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.072 s - in testUnitaires.TestUnit
3356 [INFO]
3357 [INFO] Results:
3358 [INFO]
3359 [INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0
3360 [INFO]
3361 [INFO] -----
3362 [INFO] BUILD SUCCESS
3363 [INFO] -----
3364 [INFO] Total time: 13.740 s
3365 [INFO] Finished at: 2020-11-29T18:03:36Z
3366 [INFO] -----

- Publication de l'image docker sur le repository mamesayekebe/m1sir

Update CiDuProjet.yml
master 686279d

Java CI with Maven
on: push

1

✓ build

build

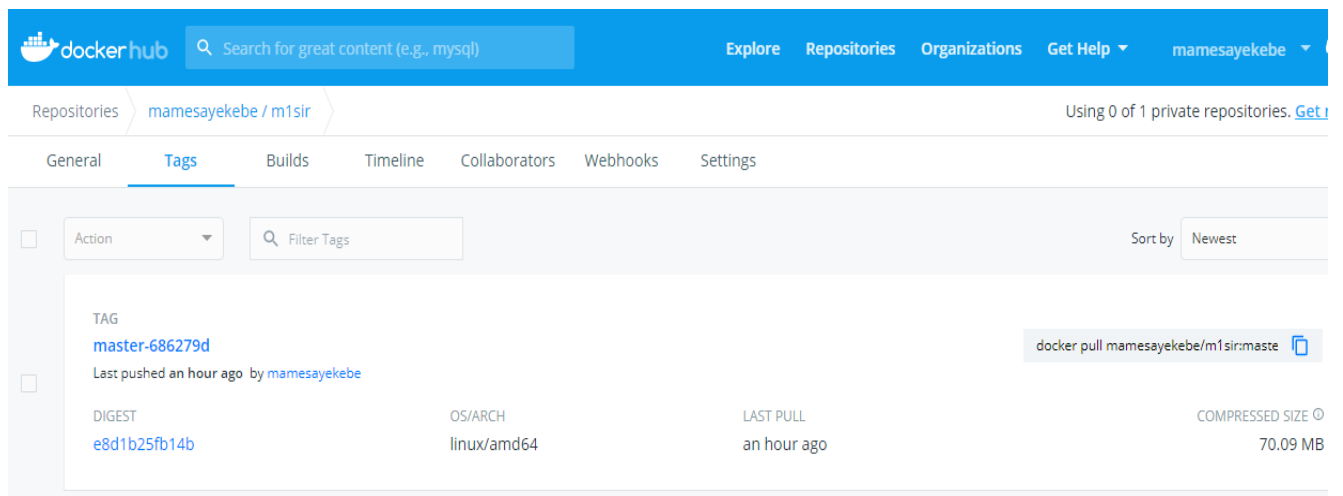
succeeded 8 minutes ago in 51s

Search logs

✓ Build and Push Docker Image in the repository ***/m1sir

59 ----> 0ad3083ebf8c
60 Successfully built 0ad3083ebf8c
61 Successfully tagged ***/m1sir:master-686279d
62 Pushing Docker image docker.io/***/m1sir:master-686279d
63 The push refers to repository [docker.io/***/m1sir]
64 84abaf51e6f5: Preparing
65 76dac8077d91: Preparing
66 642693f0ba05: Preparing
67 ceaf9e1ebef5: Preparing
68 9b9b7f3d56a0: Preparing
69 f1b5933fe4b5: Preparing
70 f1b5933fe4b5: Waiting
71 9b9b7f3d56a0: Layer already exists
72 ceaf9e1ebef5: Layer already exists
73 f1b5933fe4b5: Layer already exists
74 642693f0ba05: Pushed
75 84abaf51e6f5: Pushed
76 76dac8077d91: Pushed
77 master-686279d: digest: sha256:e8d1b25fb14bec04ed21a14a804c8920bdb11f4eef64b5d1fd521342b3a6ef53 size: 1569

Le tag de notre image docker est: **mater-686279d**



The screenshot shows the Docker Hub interface for the repository 'mamesayekebe / m1sir'. The 'Tags' tab is selected, displaying a list of tags. The tag 'master-686279d' is highlighted, showing its digest 'e8d1b25fb14b', OS/ARCH 'linux/amd64', and last pull time 'an hour ago'. The compressed size is 70.09 MB. A button 'docker pull mamesayekebe/m1sir:mater-686279d' is visible.

TAG	DIGEST	OS/ARCH	LAST PULL	COMPRESSED SIZE
master-686279d	e8d1b25fb14b	linux/amd64	an hour ago	70.09 MB

VII. Conclusion

Pour **conclure**, ce qu'il faut retenir de l'**intégration continue** c'est le test permanent du code tout au long du développement du projet. On retiendra que plus les bugs seront découverts tôt dans la phase de projet, moins le coût des correctifs sera important et plus grande sera la productivité.