Brahim HAMDI

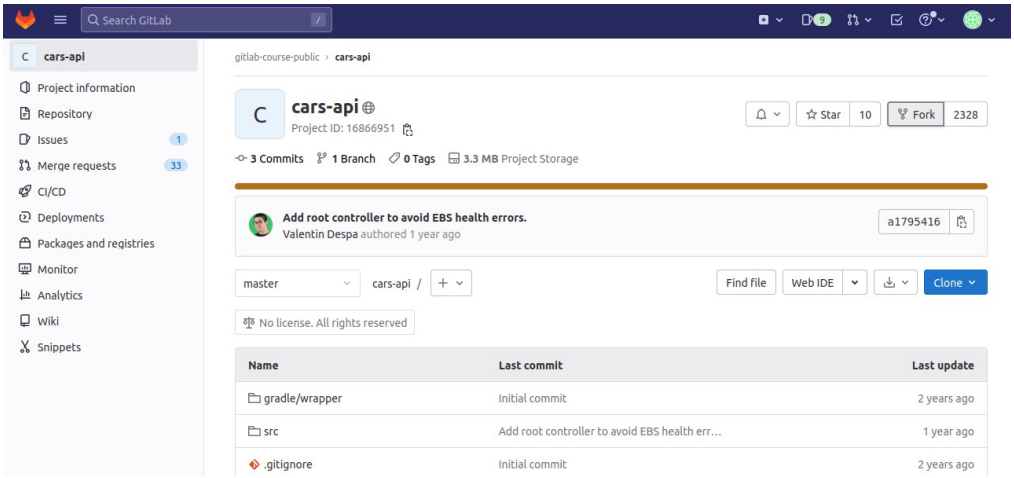
Lab 3 - Créer un projet CI/CD

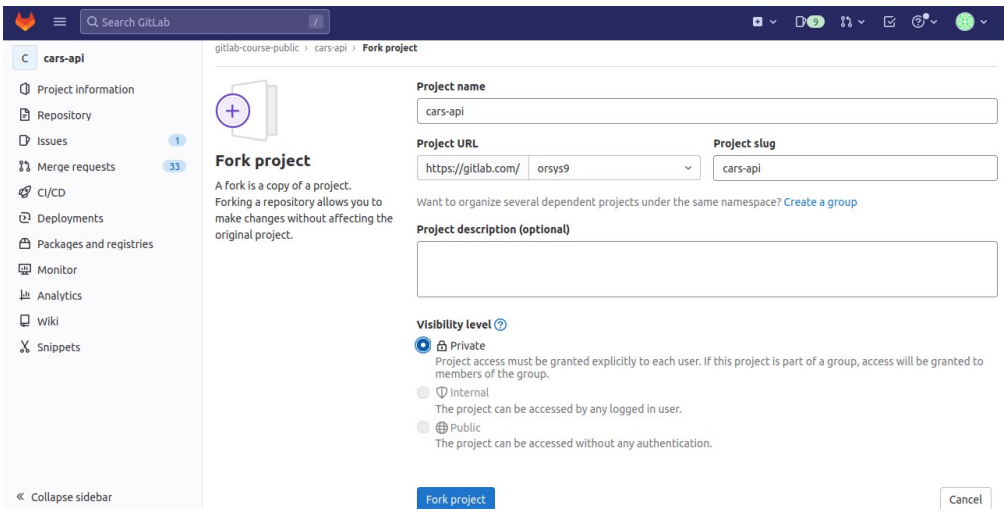
Objectifs :

- Mise en place d'un pipeline CI/CD

Création du projet et du runner spécifique :

1. Nous allons commencer par un fork de dépôt suivant :
   https://gitlab.com/gitlab-course-public/cars-api
   Un fork est une copie d'un dépôt distant qui nous permet d'effectuer des modifications sans affecter le projet original.

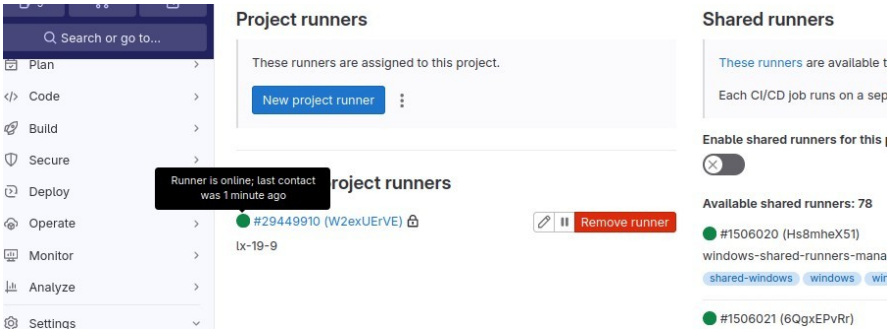




2. Dans ce TP on va utiliser des runners spécifiques. Assurez-vous que les runners partagés sont désactivés.

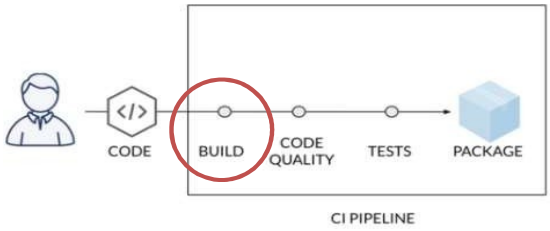
3. Enregistrez un nouveau runner docker avec la commande suivante, en utilisant le tocken que vous devez le copier d'abord depuis l'interface du projet :
   gitlab-runner register --non-interactive --url "https://gitlab.com/" --registration-token "COLLEZ-VOTRE-TOKEN" --executor "docker" --docker-image alpine:latest --run-untagged="true"

   - Vérifiez que le runner spécifique est bien crée et a contacté gitlab.com :



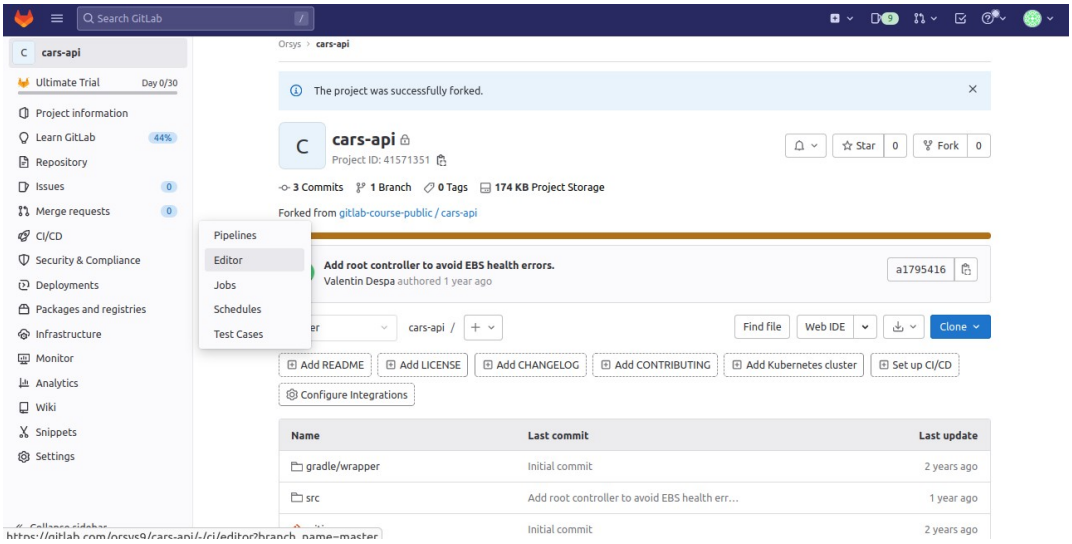
   - Lancez les runners spécifiques avec la commande suivante et sans sudo (puisqu'on les gère avec l'utilisateur courant) :
     gitlab-runner run

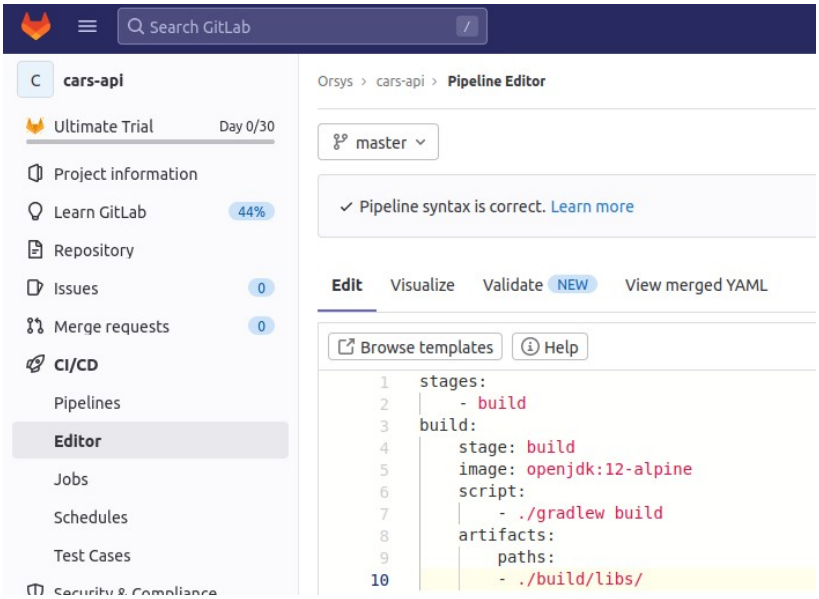
Stage de Build



4. Créez un nouveau fichier **.gitlab-ci.yml** sur la racine du projet, et y copiez le contenu suivant pour créer le job build sous le Stage de build.

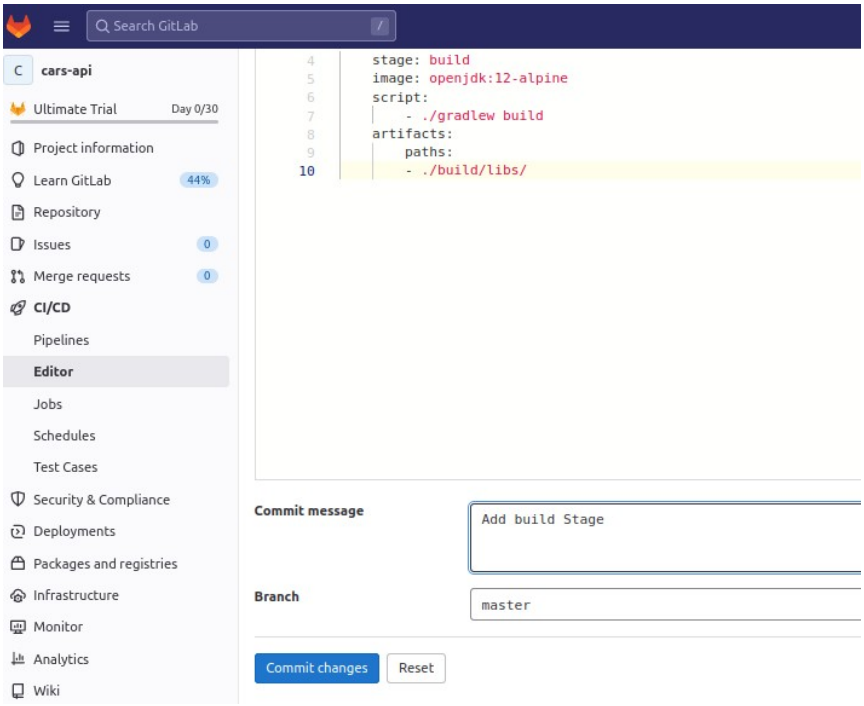
```
stages:
  - build

build:
  stage: build
  image: openjdk:12-alpine
  script:
    - ./gradlew build
  artifacts:
    paths:
      - ./build/libs/
```





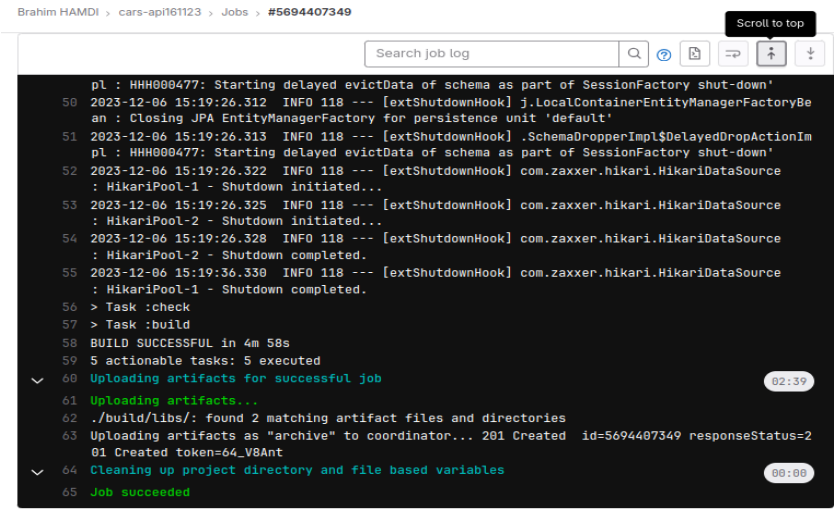
Gradle est un outil DevOps utilisé pour gérer le cycle de vie d'un projet java (compilation, tests, packaging, …). La commande ***./gradlew build*** génèrera le package ***cars-api.jar*** (sous le répertoire **./build/libs**). Ce fichier sera sauvegardé comme ***artifact***.

- Créez un nouveau commit sur le bouton « Commit changes ».

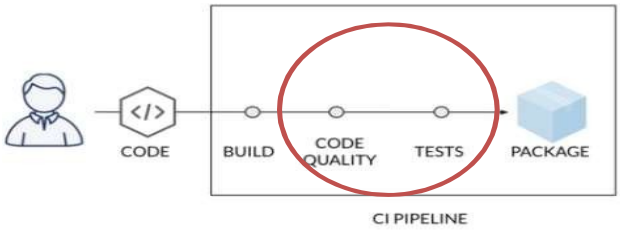


L'exécution du pipeline est lancé automatiquement à chaque commit.
Verrez la progression et la console de l'exécution du job dans **Build > Pipelines** de Gitlab.
L'exécution du job a-t-il passé ou échoué ?

- Cliquez sur le job build pour voir l'exécution du job.



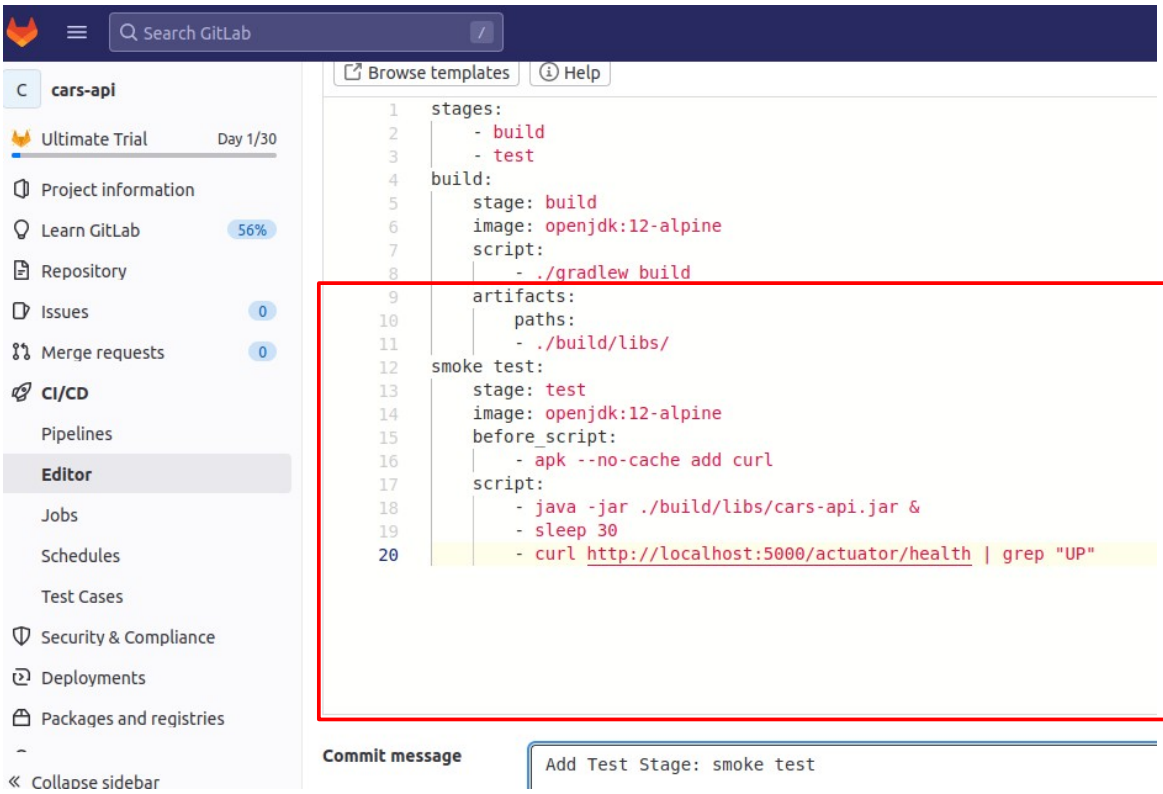
Stage de Test



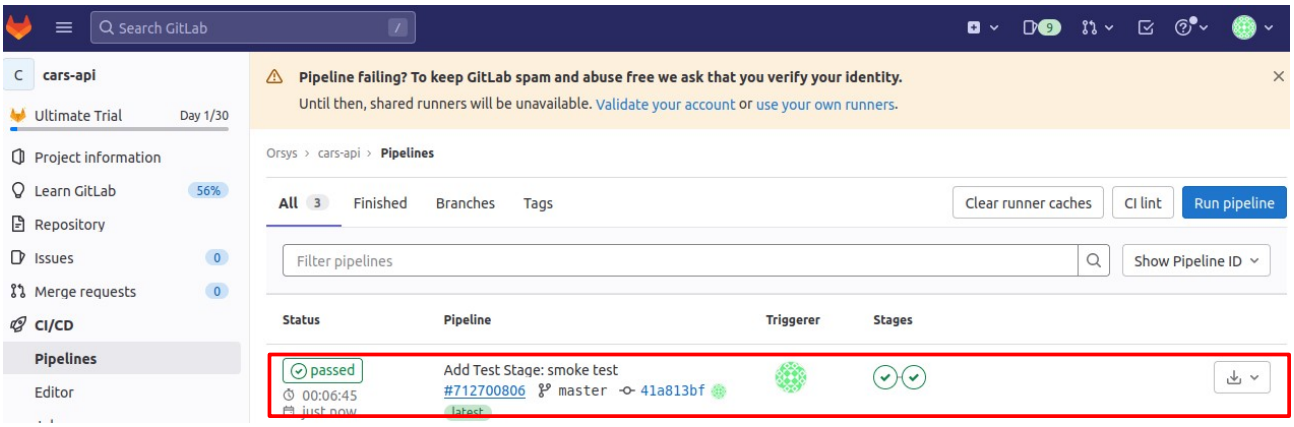
5. Dans cette étape (Stage), nous allons créer et lancer deux tests en parallèle :
   - **smoke test**: un test simple qui vérifie le bon fonctionnement de notre microservice en utilisant l'URL: *http://localhost:5000/actuator/health*
   - **unit test**: ce sont des tests chargés de tester des unités de code, généralement des classes. Pour les projets Java, **JUnit** est le framework le plus populaire pour l'écriture de tests unitaires.
   - Ajoutez au fichier **.gitlab-ci.yml** le job ***smoke test*** du premier test.

```
stages
  - build
  - test

build:
  stage: build
  image: openjdk:12-alpine
  script:
    - ./gradlew build
  artifacts:
    paths:
      - ./build/libs/
smoke test:
  stage: test
  image: openjdk:12-alpine
  before_script:
    - apk --no-cache add curl
  script:
    - java -jar ./build/libs/cars-api.jar &
    - sleep 30
    - curl http://localhost:5000/actuator/health | grep "UP"
```

- Créer un nouveau commit avec le message suivant : « Add Test Stage: smoke test »



- Vérifiez l'exécution de cette étape dans **Build > Pipelines**.

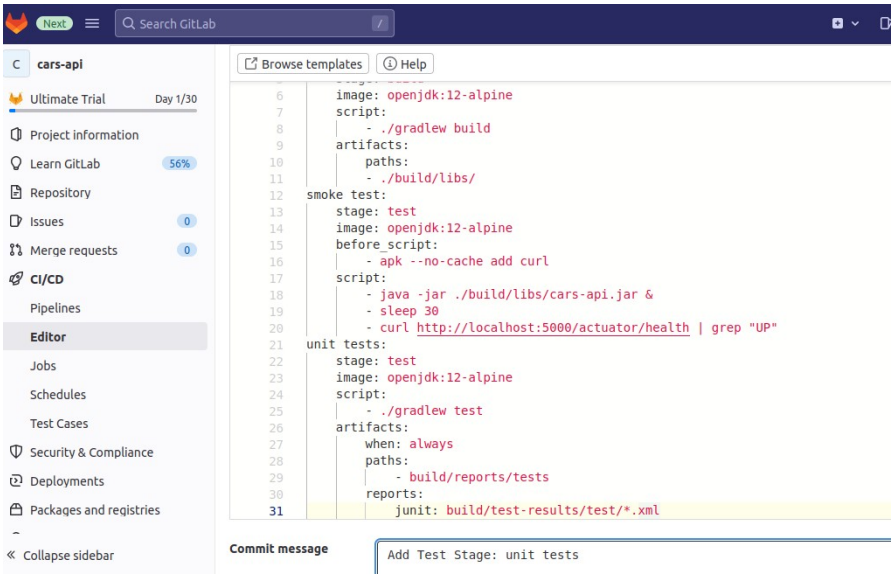


6. Ajoutez à la fin de votre fichier **.gitlab-ci.yml**, le script de deuxième test : **unit tests**

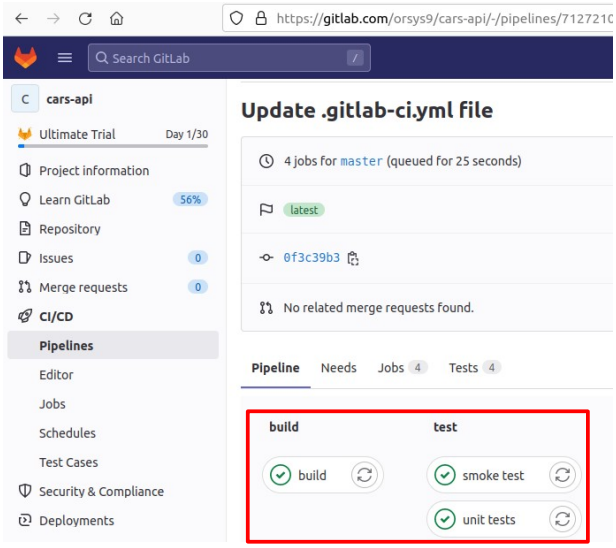
```
unit tests:
  stage: test
  image: openjdk:12-alpine
  script:
    - ./gradlew test
```

```
artifacts:
  when: always
  paths:
    - build/reports/tests
  reports:
    junit: build/test-results/test/*.xml
```

- Créez un nouveau commit avec le message suivant : « Add Test Stage: unit tests »

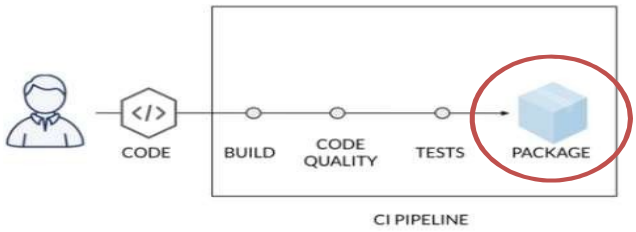


- Vérifiez l'exécution des deux tests en parallèle.



- Verrez le rapport de test unitaires dans votre *Artifact*. Quel est le résultat du rapport ?

Stage de Package



7. Ce processus nous permettra, dans la livraison continue, de pouvoir déployer facilement le même code sur différents environnements.
   Il sert aussi à figer le code compilé dans un package immutable.
   Un nouveau job supplémentaire compile le projet et l'encapsule dans un conteneur Docker. Il est ensuite poussé sur la *registry* de GitLab.
   - Tout d'abord, nous nous allons créer un nouveau fichier ***Dockerfile*** dans la racine de votre dépôt.
     Ajouter dans ce ficher le script permettant d'encapsuler dans un conteneur notre microservice à partir de l'image *openjdk:12-alpine* :

     FROM openjdk:12-alpine
     VOLUME /tmp
     ADD /build/libs/.jar cars-api.jar*
     EXPOSE 5000
     ENTRYPOINT exec java -jar cars-api.jar

8. Dans le job realease qu'on va ajouter, on aura besoin des commandes docker pour builder et pousser l'image docker de notre application, donc on peut pas utiliser l'image par défaut

openjdk:12-alpine. Bien qu'on peut utiliser l'image docker:latest pour exécuter ce job et les job suivant, ça sera plus simple d'enregistrer un nouveau runner qui utilise le shell comme exécuteur (au lieu de docker) puisqu'on docker-ce qu'est déjà installé sur la VM.

- Créez un nouveau runner avec les caractéristiques suivantes :
  - **executor** : shell
  - **tag** : shell

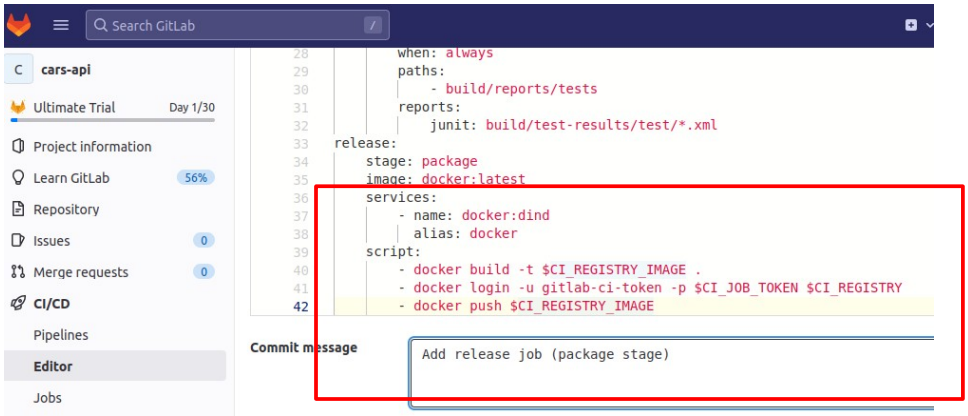
gitlab-runner register --non-interactive --url "https://gitlab.com/" --registration-token "COLLEZ-VOTRE-TOKEN" --executor "shell" --tag-list shell

Ce runner ***shell*** sera utiliser dans tous les jobs suivants.

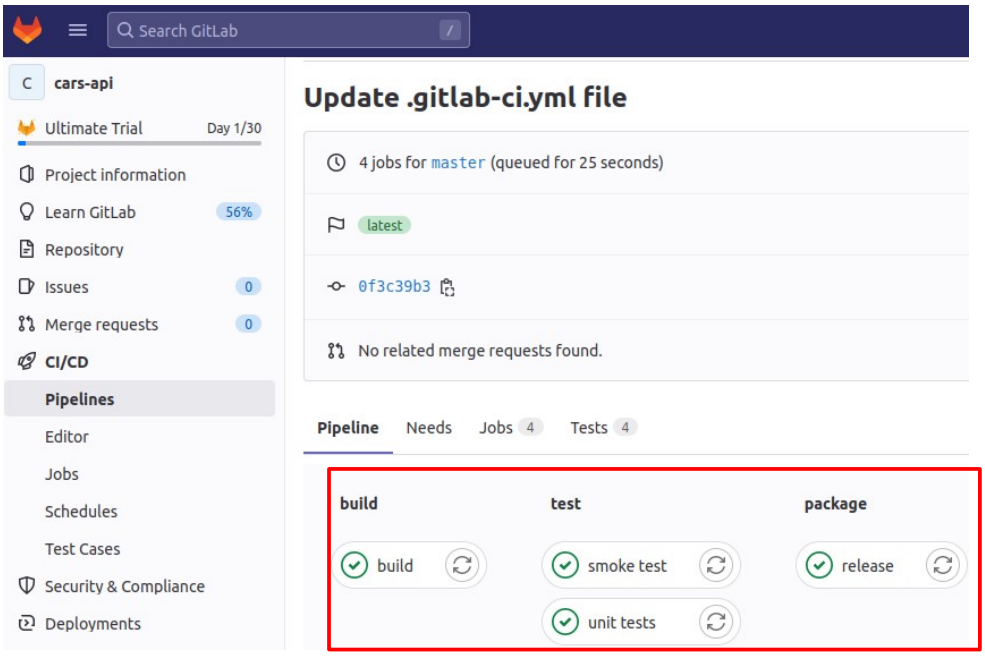
9. Ajoutez à la fin de votre fichier *.gitlab-ci.yml*, le job **release** de troisième stage **package**.

```
stages:
    - build
    - test
    - package

...
release:
    stage: package
    tags:
      - shell
    script:
      - docker build -t $CI_REGISTRY_IMAGE .
      - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $CI_REGISTRY
      - docker push $CI_REGISTRY_IMAGE
```

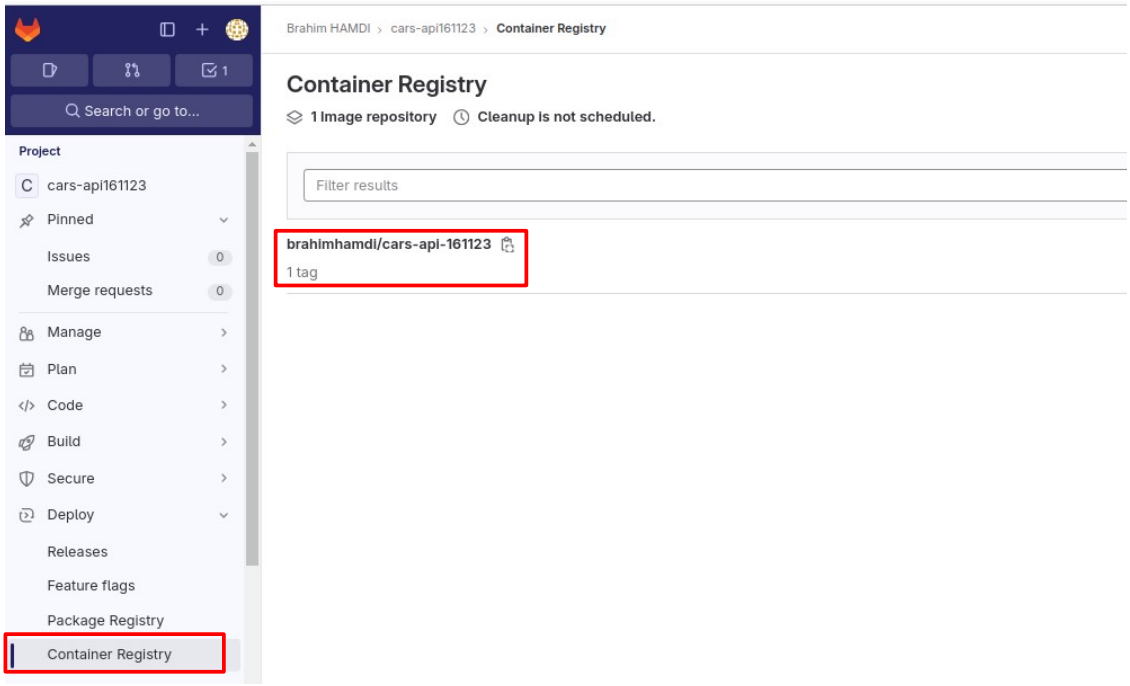
Nous connectons sur la *registry* interne de GitLab afin de pouvoir pousser (push) notre image Docker de façon sécurisée.



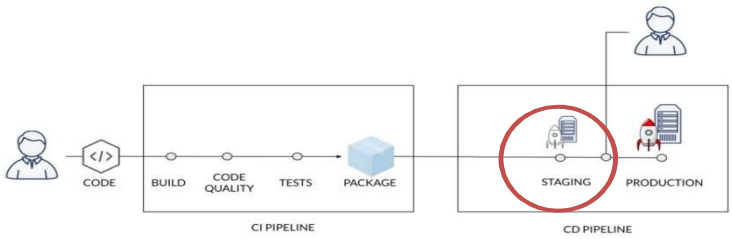
- Observez l'exécution de votre pipeline CI.



- Si l'exécution passe avec succès, vérifiez la création de votre image Docker dans **Deploy > Container Registry** de Gitlab



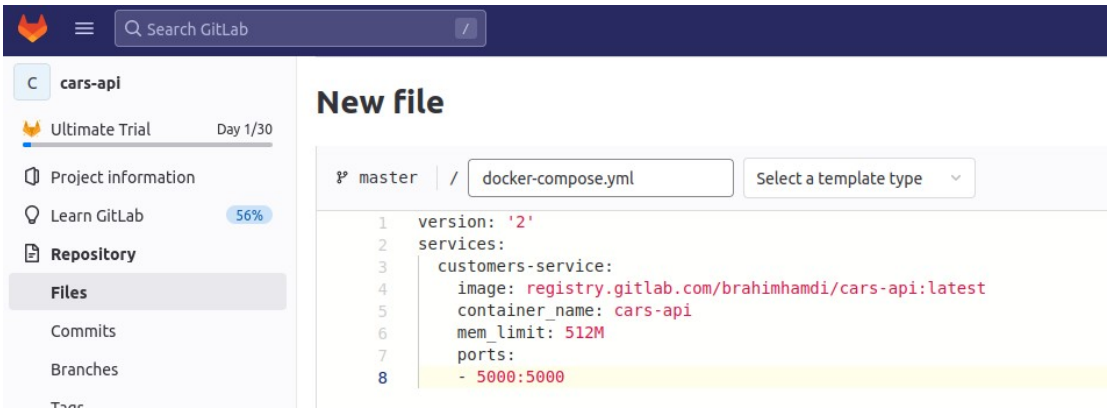
Déploiement sur l'environnement de Staging



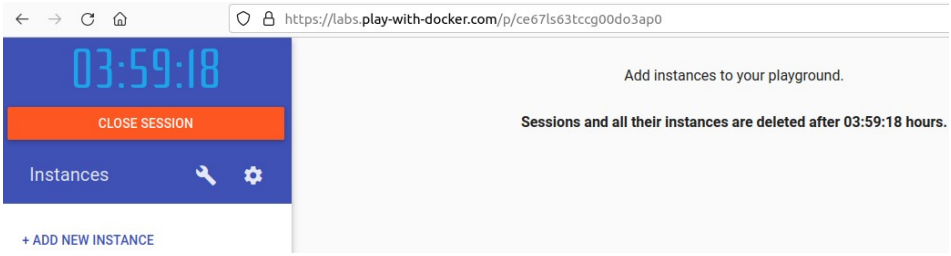
10. Avant de passer à la production, nous allons créer une étape de pré-production afin de tester le bon fonctionnement de notre microservice dans un **environnement** de test.
    Dans ce stage on va utiliser l'outil ***Docker Compose*** (qu'on doit installer) pour déployer l'application sur le serveur de staging.
    - Installez Docker Compose sur la VM vagrant en utilisant la commande suivante :
      sudo apt install docker-compose

    - Docker compose utilise le fichier ***docker-compose.yml*** pour déployer l'application, il faut l'ajouter au projet.
      Créez un nouveau fichier ***docker-compose.yml*** dans la racine de votre dépôt dont le contenu est le suivant (remplacez brahimhamdi par votre login gitlab) :

```
version: '2'
services:
    customers-service:
        image: registry.gitlab.com/brahimhamdi/cars-api:latest
        container_name: cars-api
        mem_limit: 512M
        ports:
        - 5000:5000
```

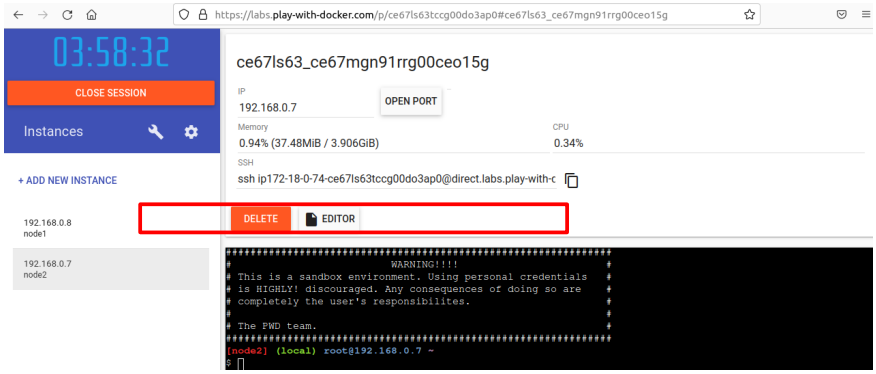


11. Pour créer une infrastructure Docker rapidement, rendez-vous sur le site **https://labs.play-with-docker.com/** et connectez-vous avec vos identifiants Docker Hub (créer un compte sur https://hub.docker.com/ si vous n'avez pas).
    - Une fois connecté, une session de 4 heures est créée afin de vous permettre de déployer vos images.



    - Sur la page d'accueil, cliquez deux fois sur l'icône **+ ADD NEW IINSTAN** pour créer deux instances pour le déploiement en STAGING et en PRODUCTION.

    - Une fois créés, récupérez l'URL de l'environnement. Il suffit de copier l'URL de la case SSH.



Nous utiliserons cette URL pour configurer l'environnement de déploiement dans le fichier *.gitlab- ci.yml*.

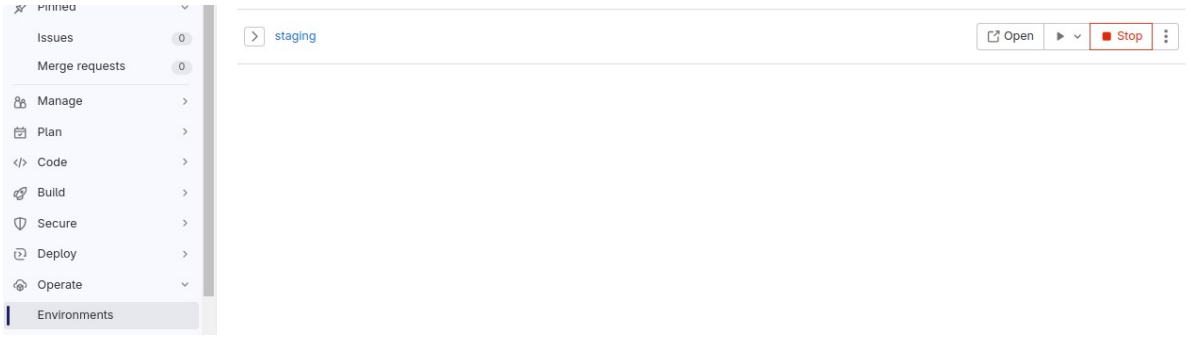
12. Ajoutez par la suite le job ***staging*** dans l'étape de **deploy *staging***.

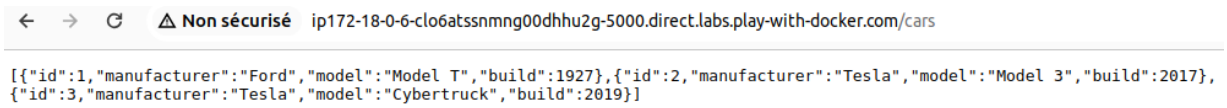
```
stages:
    - build
    - test
    - package
    - deploy staging
…
staging:
    stage: deploy staging
    tags :
    - shell
    variables:
        PLAYWD: ip172-18-0-74-ce67ls63tccg00do3ap0
    script:
    - export DOCKER_HOST=tcp://$PLAYWD.direct.labs.play-with-docker.com:2375
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $CI_REGISTRY
    - docker-compose down
    - docker-compose up -d
    environment:
        name: staging
        url: http://$PLAYWD-5000.direct.labs.play-with-docker.com
```

La nouvelle variable PLAYWD contient l'URL copiée précédemment

- Vérifiez l'exécution de votre pipeline CI/CD.

13. Si tout s'est bien passé, vous devriez voir apparaître dans vos environnements **Deploy > Environments**, le nouvel environnement ***Staging***.
    - Cliquez sur **Operate > Environments**, puis le lien "**Open live environment**", pour voir l'interface de l'application déployée (ajoutez **/cars** à la fin de votre URL).





Déploiement sur Production

14. Ajoutez par la suite dans votre manifeste le code source de l'étape de **production**.

```
stages:
    - build
    - test
    - package
    - deploy staging
    - deploy production
...
production:
  stage: deploy production
  tags :
  - shell
  variables:
      PLAYWD: ip172-18-0-103-c2gksclmrepg00arjblg
  script:
  - apk add --no-cache --quiet py-pip
  - pip install --quiet docker-compose~=1.23.0
  - export DOCKER_HOST=tcp://$PLAYWD.direct.labs.play-with-docker.com:2375
  - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $CI_REGISTRY
```

```
- docker-compose down
- docker-compose up -d
when: manual
allow_failure: false
environment:
        name: master
        url: http://$PLAYWD-5000.direct.labs.play-with-docker.com
```

- Vous devez utiliser l'URL de la deuxième instance déjà créée dans **https://labs.play-with-docker.com/**
- Gitlab offre la possibilité de déclencher manuellement des jobs avec la déclaration **when:manual**.
- S'il y a des étapes supplémentaires après ce job, elles seront toujours exécutées.
- Pour bloquer ces jobs, nous devons ajouter la déclaration **allow_failure:false**

15. Sur votre pipeline de livraison continue, le déploiement manuel est symbolisé par l'icône à côté de l'étape deploy production.

    - Lancer le déploiement manuel en cliquant sur l'icône
    - Vous devriez voir apparaître dans vos environnements (*Operations* > *Environnements*), le nouvel environnement production.