Fontaine Hugo

TP : DevOps

1-1 Document your database container essentials : commands and Dockerfile.

- Création des fichiers permettant la création des tables et l'insertion des données :
  -01-CreateScheme.sql

```sql
CREATE TABLE public.departments
(
 id      SERIAL      PRIMARY KEY,
 name    VARCHAR(20) NOT NULL
);

CREATE TABLE public.students
(
 id              SERIAL      PRIMARY KEY,
 department_id   INT         NOT NULL REFERENCES departments (id),
 first_name      VARCHAR(20) NOT NULL,
 last_name       VARCHAR(20) NOT NULL
);
```

02-InsertData.sql

```sql
INSERT INTO departments (name) VALUES ('IRC');
INSERT INTO departments (name) VALUES ('ETI');
INSERT INTO departments (name) VALUES ('CGP');


INSERT INTO students (department_id, first_name, last_name) VALUES (1, 'Eli', 'Copter');
INSERT INTO students (department_id, first_name, last_name) VALUES (2, 'Emma', 'Carena');
INSERT INTO students (department_id, first_name, last_name) VALUES (2, 'Jack', 'Uzzi');
INSERT INTO students (department_id, first_name, last_name) VALUES (3, 'Aude', 'Javel');
```

- Création d'un Dockerfile avec les commandes suivantes :

```dockerfile
FROM postgres:14.1-alpine

COPY /sql /docker-entrypoint-initdb.d

ENV POSTGRES_DB=db \
    POSTGRES_USER=usr \
    POSTGRES_PASSWORD=pwd
```

Fontaine Hugo

On utilise ces commandes pour créer une image pour la base de données postgre et définir donc les éléments de sécurité comme le nom d'utilisateur, mot de passe pour un utilisateur dans la bdd.

- Par la suite on crée un network permettant de gérer la connectivité entre les containers. Et on construit un container avec docker build contenant la base de données :

```
(base) Hugo@MacBook-Pro-de-PC TP % docker network create app-network
```

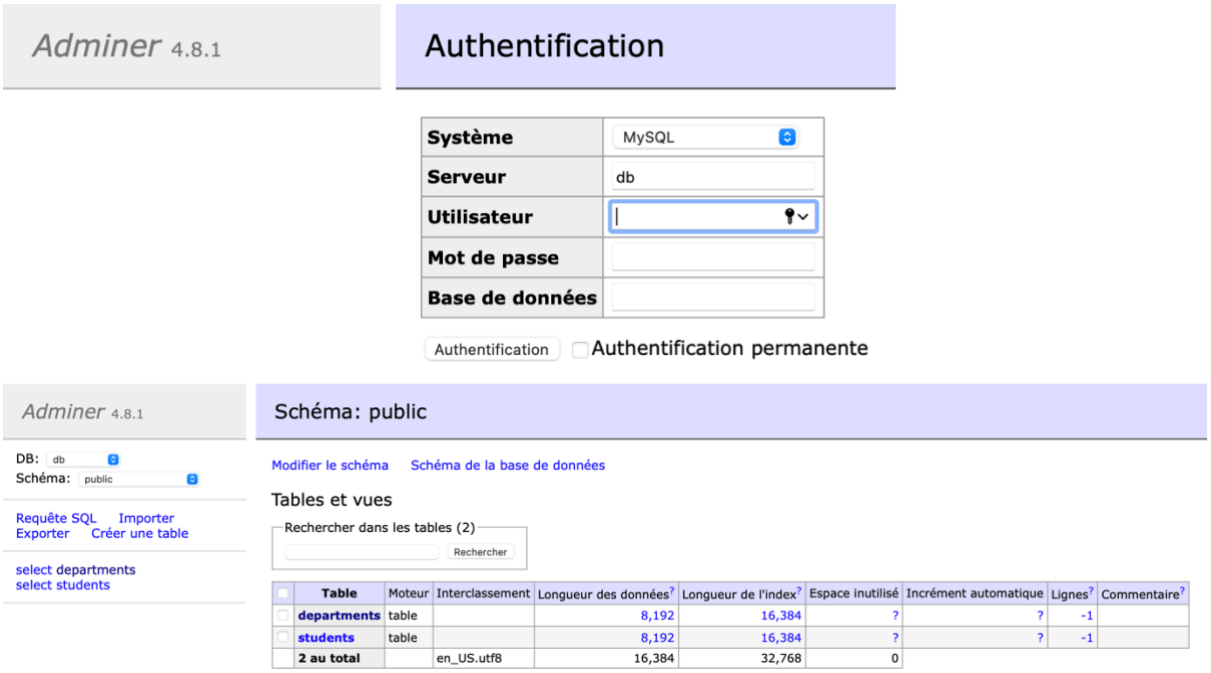
```
(base) Hugo@MacBook-Pro-de-PC TP % docker build -t hugo/post_gre_tp .
[+] Building 12.3s (6/6) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 133B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/postgres:14.1-alpine
 => [auth] library/postgres:pull token for registry-1.docker.io
 => [1/1] FROM docker.io/library/postgres:14.1-alpine@sha256:578ca5c8452c08a4e0f5e65b55dce5e1812fe63c8fee40ea837641031598e51e
 => => resolve docker.io/library/postgres:14.1-alpine@sha256:578ca5c8452c08a4e0f5e65b55dce5e1812fe63c8fee40ea837641031598e51e
 => => sha256:578ca5c8452c08a4e0f5e65b55dce5e1812fe63c8fee40ea837641031598e51e 1.65kB / 1.65kB
 => => sha256:884c142deb4a141f67489c807534ec6139f13b9a6432d2f87a4de283aaec0b5c 1.99kB / 1.99kB
 => => sha256:a0646b0f1eadaf0cd3fdb4c4490a69c4c7aed9b7ae10b24eb9005c59aa0b6e57 148B / 148B
 => => sha256:1149d285a5f5c4340cefa2211869c3a6b1128ac78974545e0b4fe62d3d0e66a8 7.84kB / 7.84kB
 => => sha256:59bf1c3509f33515622619af21ed55bbe26d24913cedbca106468a5fb37a50c3 2.82MB / 2.82MB
 => => sha256:c50e01d57241cf7ef93a91060f5eb0b895a4b443f20dc1ce5e77d441184a6dc2 1.28kB / 1.28kB
 => => sha256:7433e5151e0cee31a0d5b90433751e809eb3b860b250f73f0720f7d05788dc34 78.90MB / 78.90MB
 => => sha256:8854018388d9035028f41a2c0494acf2868e2189888840026df2227cc4728a8d 9.20kB / 9.20kB
 => => extracting sha256:59bf1c3509f33515622619af21ed55bbe26d24913cedbca106468a5fb37a50c3
 => => sha256:8de463f7fd190f136a6f5cf50d5d8add707fd78d079d6391d952a02dcd40e412 162B / 162B
 => => sha256:b39ee18abab98838412adb823af56cad2e732c25ac47da3ed4af92e41dbd2a7f 194B / 194B
 => => sha256:11d7473a0ff973b03640db89b7278d2694eb298a52a84ad71b4e14a1ce3de45f 4.72kB / 4.72kB
 => => extracting sha256:c50e01d57241cf7ef93a91060f5eb0b895a4b443f20dc1ce5e77d441184a6dc2
 => => extracting sha256:a0646b0f1eadaf0cd3fdb4c4490a69c4c7aed9b7ae10b24eb9005c59aa0b6e57
 => => extracting sha256:7433e5151e0cee31a0d5b90433751e809eb3b860b250f73f0720f7d05788dc34
 => => extracting sha256:8854018388d9035028f41a2c0494acf2868e2189888840026df2227cc4728a8d
 => => extracting sha256:8de463f7fd190f136a6f5cf50d5d8add707fd78d079d6391d952a02dcd40e412
 => => extracting sha256:b39ee18abab98838412adb823af56cad2e732c25ac47da3ed4af92e41dbd2a7f
 => => extracting sha256:11d7473a0ff973b03640db89b7278d2694eb298a52a84ad71b4e14a1ce3de45f
 => exporting to image
 => => exporting layers
 => => writing image sha256:6685d0c4833878dbaf36a3d7be0ba0cd1b87bd37dd22ad8d74ef42391e614a48
 => => naming to docker.io/hugo/post_gre_tp
```

- Par la suite on lance un container sur le port 8090 avec l'image Adminer, un outil pour gérer la base de données qu'on a construit plus tôt. L'image va être pull directement juste après.

```
(base) Hugo@MacBook-Pro-de-PC TP %     docker run \
    -p "8090:8080" \
    --net=app-network \
    --name=adminer \
    -d \
    adminer
```

- On lance enfin la base de données avec la commande suivante, on peut donc se connecter sur Adminer pour voir les différentes tables :

```
(base) Hugo@MacBook-Pro-de-PC TP % docker run -p 8889:5000 --name post_gre_TP --network app-network hugo/post_gre_tp
```



1-2 Why do we need a multistage build? And explain each step of this dockerfile

On a besoin d'un multistage build pour pouvoir construire des images afin de créer des containers réduits en taille, plus efficaces et plus sécurisés.
On peut avec cette méthode créer du code, compiler en plusieurs étapes.

```
1   # Build
2   FROM maven:3.8.6-amazoncorretto-17 AS myapp-build
3   ENV MYAPP_HOME /opt/myapp
4   WORKDIR $MYAPP_HOME
5   COPY pom.xml .
6   COPY src ./src
7   RUN mvn package -DskipTests
8
9   # Run
10  FROM amazoncorretto:17
11  ENV MYAPP_HOME /opt/myapp
12  WORKDIR $MYAPP_HOME
13  COPY --from=myapp-build $MYAPP_HOME/target/*.jar $MYAPP_HOME/myapp.jar
14
15  ENTRYPOINT java -jar myapp.jar
```

Dans ce Dockerfile, on importe une image de base de Maven, on crée une variable d'environnement « MYAPP_HOME » permettant de spécifier le répertoire de travail de l'application avec une valeur « opt/myapp », on exécute les commandes d'avant avant avec WORKDIR, on copie le fichier pom.xml (fichier de configuration de

maven) depuis le répertoire locale (même répertoire que le fichier Dockerfile) vers le répertoire de travail (/opt/myapp) et on fait pareil avec le répertoire .src qui contient le code source de l'application.

1-3 Document docker-compose most important commands. 1-4 Document your docker-compose file.

- **docker-compose up.**
- **docker-compose ps :** Cette commande affiche les journaux (logs) de tous les services ou d'un service spécifique.
- **docker-compose build :** Cette commande est utilisée pour reconstruire les images des services définis dans le fichier docker-compose.yml. Elle est utile lorsque vous avez apporté des modifications à un service et que vous souhaitez mettre à jour son image.

Docker compose up permet de lancer tous les containers.

```
version: '3.7'

services:
  backend:
    build:
      context: ./simple-api-student-main
      dockerfile: Dockerfile
    container_name: project_simpleapi
    networks:
    - app-network
    depends_on:
    - database
    ports:
      - "8080:8080"

  database:
    build:
      context: ./Database
      dockerfile: Dockerfile
    container_name : post_gre_TP
    networks:
    - app-network

  httpd:
    build:
      context: ./devops-front-main
      dockerfile: Dockerfile
    container_name: front_final_simpleapi
    ports:
      - "8081:80"
    networks:
    - app-network
    depends_on:
    - backend

networks:
  app-network:
```

On a dans ce docker-compose file, toutes les commandes permettant de trouver les dockerfile, et de lancer les containers.

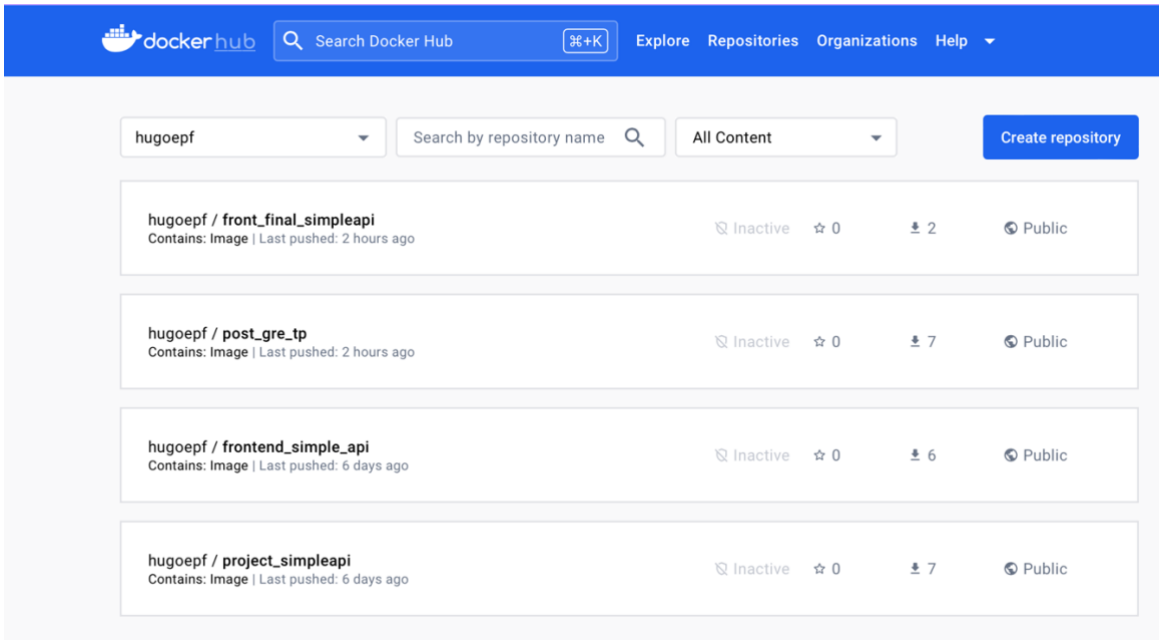
1-5 Document your publication commands and published images in dockerhub.

```
(base) Hugo@MacBook-Pro-de-PC simple-api-student-main % docker tag hugo/post_gre_tp hugoepf/post_gre_tp:1.0
(base) Hugo@MacBook-Pro-de-PC simple-api-student-main % docker push hugoepf/post_gre_tp:1.0
```

```
(base) Hugo@MacBook-Pro-de-PC simple-api-student-main % docker tag hugo/project_simpleapi hugoepf/project_simpleapi:1.0
(base) Hugo@MacBook-Pro-de-PC simple-api-student-main % docker push hugoepf/project_simpleapi:1.0
```

```
(base) Hugo@MacBook-Pro-de-PC simple-api-student-main % docker tag hugo/frontend_simple_api hugoepf/frontend_simple_api:1.0
(base) Hugo@MacBook-Pro-de-PC simple-api-student-main % docker push hugoepf/frontend_simple_api:1.0
```

Nous avons toutes les commandes pour push sur docker hub, voici donc l'interface par la suite.



2-1 What are testcontainers?

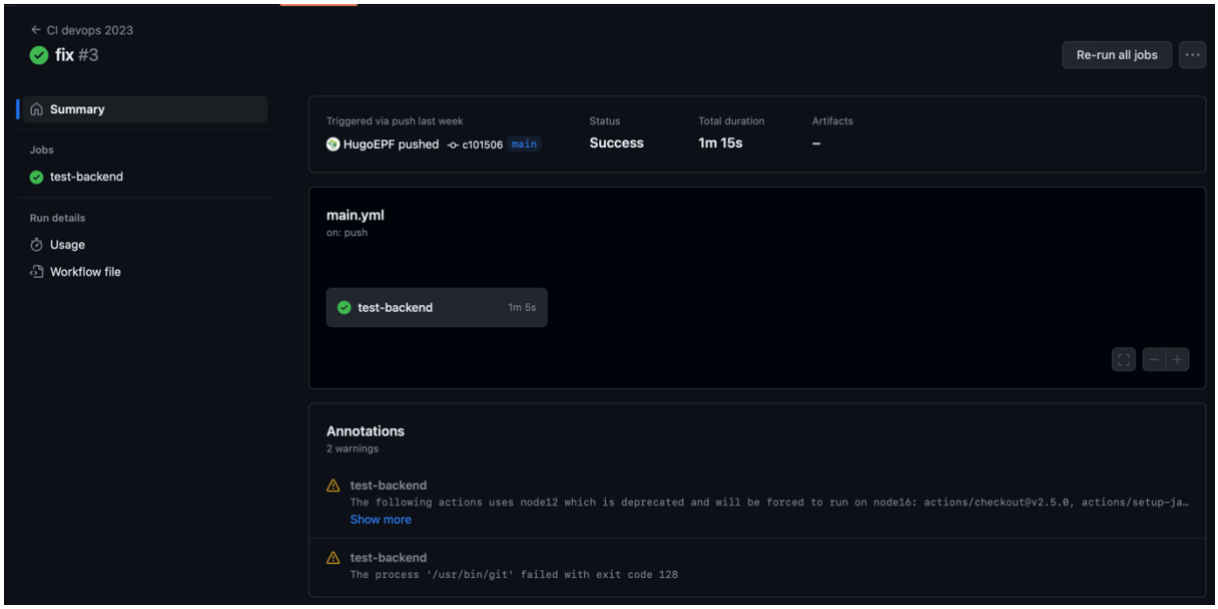
Testcontainers est une bibliothèque Java open source qui permet aux développeurs de créer, gérer et détruire des conteneurs Docker dans le but de réaliser des tests d'intégration et des tests unitaires. Cette bibliothèque simplifie la création d'environnements de test isolés en utilisant des conteneurs Docker pour exécuter des services tiers, des bases de données, des serveurs web, ou d'autres composants nécessaires pour tester une application.

2-2 Document your Github Actions configurations.

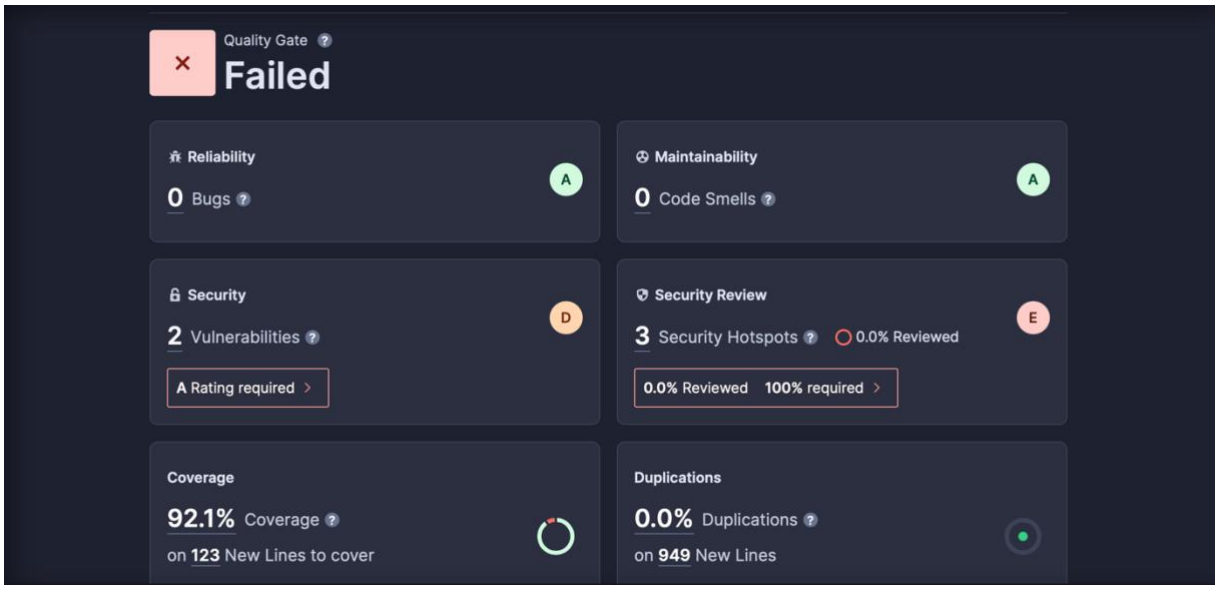
Le code suivant est un workflow qui est composé de plusieurs « jobs » :
- Construire l'application et faire passer des tests de développement à l'aide de Maven sur un environnement ubuntu.
-Demander une analyse à SonarCloud du code.
- Push sur Docker Hub les images du projet.
Ce workflow démarre lors du git push du projet.

```
name: CI devops 2023
on:
  #to begin you want to launch this job in main and developp
  push:
    branches:
      - main
  pull_request:

jobs:
  test-backend:
    runs-on: ubuntu-22.04
    steps:
      #checkout your github code using actions/checkout@v2.5.0
      - name: Checkout Repository
        uses: actions/checkout@v2.5.0

      #do the same with another action (actions/setup-java@v3) that enable
to setup jdk 17
      - name: Set up JDK 17
        uses: actions/setup-java@v2  # Assurez-vous d'utiliser la version
correcte de l'action
        with:
          java-version: 17
          distribution: 'adopt'

      #finally build your app with the latest command
      - name: Build and test with Maven
        working-directory: simple-api-student-main
        run: mvn -B verify sonar:sonar -Dsonar.projectKey=devops-project-
hugo_hugoepf -Dsonar.organization=devops-project-hugo -
Dsonar.host.url=https://sonarcloud.io -Dsonar.login=${{ secrets.SONAR_TOKEN
}}  --file ./pom.xml


  # define job to build and publish docker image
  build-and-push-docker-image:
    needs: test-backend
    # run only when code is compiling and tests are passing
    runs-on: ubuntu-22.04

    # steps to perform in job
    steps:
      - name: Checkout code
        uses: actions/checkout@v2.5.0

      - name: Login to DockerHub
        run: docker login -u ${{ secrets.DOCKERHUB_USERNAME }} -p ${{
secrets.DOCKERHUB_PASSWORD }}

      - name: Build image and push backend
```

```
        uses: docker/build-push-action@v3
        with:
          # relative path to the place where source code with Dockerfile is
located
          context: ./simple-api-student-main
          # Note: tags has to be all lower-case
          tags:  ${{secrets.DOCKERHUB_USERNAME}}/project_simpleapi
          push: ${{ github.ref == 'refs/heads/main' }}

      - name: Build image and push database
        uses: docker/build-push-action@v3
        with:
          context: ./Database # Remplacez par le chemin relatif vers le
code source de la base de données
          tags: ${{secrets.DOCKERHUB_USERNAME}}/post_gre_tp
          push: ${{ github.ref == 'refs/heads/main' }}
        # DO the same for database

      - name: Build image and push httpd
        uses: docker/build-push-action@v3
        with:
          context: ./FrontEnd # Remplacez par le chemin relatif vers le
code source d'Apache HTTP Server
          tags: ${{secrets.DOCKERHUB_USERNAME}}/frontend_simple_api
          push: ${{ github.ref == 'refs/heads/main' }}
        # DO the same for httpd
```



2-3 Document your quality gate configuration.



3-1 Document your inventory and base commands

```
all:
  vars:
    ansible_user: centos
    ansible_ssh_private_key_file: /Users/Hugo/Documents/5A/id_rsa
  children:
    prod:
      hosts: hugo.fontaine.takima.cloud
```

Fontaine Hugo

3-2 Document your playbook

```yaml
- hosts: all
  gather_facts: false
  become: yes
  roles:
    - docker
    - network
    - database
    - app
    - proxy
```

3-3 Document your docker_container tasks configuration.

Rôle Docker :

```
# Install Docker

- name: Clean packages
  command:
    cmd: yum clean -y packages


- name: Install device-mapper-persistent-data
  yum:
    name: device-mapper-persistent-data
    state: latest

- name: Install lvm2
  yum:
    name: lvm2
    state: latest

- name: add repo docker
  command:
    cmd: sudo yum-config-manager --add-
repo=https://download.docker.com/linux/centos/docker-ce.repo

- name: Install Docker
  yum:
    name: docker-ce
    state: present

- name: Make sure Docker is running
  service: name=docker state=started
  tags: docker
```

Fontaine Hugo

Rôle Network:

```
# Create Docker Network

- name: network
  community.docker.docker_network:
    name: app-network
```

Rôle Database:

```
# Launch database
- name: Launch PostgreSQL Container
  docker_container:
    name: post_gre_TP
    image: hugoepf/post_gre_tp:latest
    pull: true
    state: started
    ports:
      - "8090:8080"
    env:
      POSTGRES_DB: db
      POSTGRES_USER: usr
      POSTGRES_PASSWORD: pwd

    networks:
      - name: app-network
```

Rôle Back (app):

```
- name: Launch Application Container
  docker_container:
    name: project_simpleapi
    image: hugoepf/project_simpleapi
    pull: true
    ports:
      - "8080:8080"
    networks:
      - name: app-network
```

Rôle Front (proxy):

```
- name: Launch Proxy Container
  docker_container:
    name: front_final_simpleapi
    image: hugoepf/front_final_simpleapi:1.0
    pull: true
    state: started
    ports:
      - "80:80"
    networks:
      - name: app-network

  register: proxy_result


- name: Debug Proxy Container Result
  debug:
    var: proxy_result
```