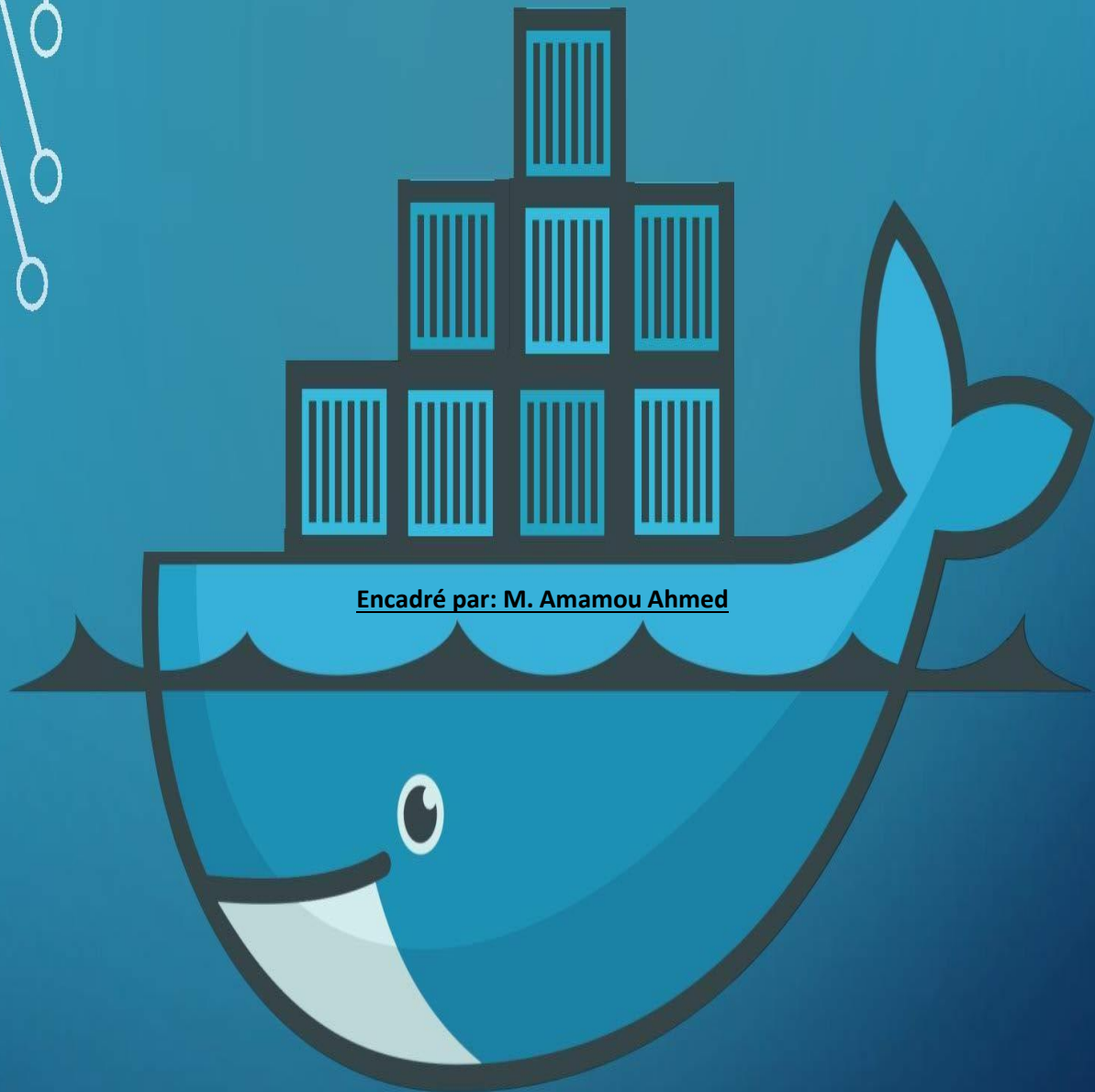


# Tp Docker



Encadré par: M. Amamou Ahmed

# docker

For the Absolute Beginner

# **SOMMAIRE**

**Partie 0 : Définitions**

**Partie 1 : Dockerfile**

**Partie 2: Docker-compose**

**Partie 3 : Docker-registry**

**Partie 4: Gestion du cycle de vie des images**

**Annexe**

## **Partie 0: Définitions**

0.1) Rappeler le concept d'un Dockerfile

Un Dockerfile est un fichier qui permet de construire une image Docker adaptée aux besoins,

0.2) Rappeler le concept d'un docker-compose

Docker Compose est un outil qui permet de décrire (dans un fichier YAML) et gérer (en ligne de commande) plusieurs conteneurs comme un ensemble de services interconnectés

0.3) C'est quoi le docker registry ?

Est un moyen ordinaire de stocker et de distribuer des images Docker. Le registre est un référentiel open source sous licence d'autorisation Apache. Le registre Docker permet également d'améliorer le contrôle d'accès et la sécurité des images Docker stockées dans son référentiel.

## **Partie 1: Dockerfile Télécharger le fichier zip suivant: <http://demo.amamou.com/app.zip> C'est une simple application permettant l'enregistrement par nom et email dans une base de donnée postgres**

1.1) Vérifier que l'application tourne bien sur votre machine pour cela vous devez ; - Installer postgresql sur votre machine linux ou sur la vm linux - Vous connecter sur postgres:

- su
- su postgres
- psql
- Une fois connecté sur postgres créer votre base de données
- create database test.

2

- Créer le user
- create user test with password 'test';
- grant ALL ON DATABASE test to test.
- Quitter la console postgres
- Spécifier les variables d'environnement suivantes dans le fichier entrypoint.sh fournis

Avec l'application:

- DBUSER
- DBPASS

- DBHOST
- DBNAME

Exemple: export DBUSER=test

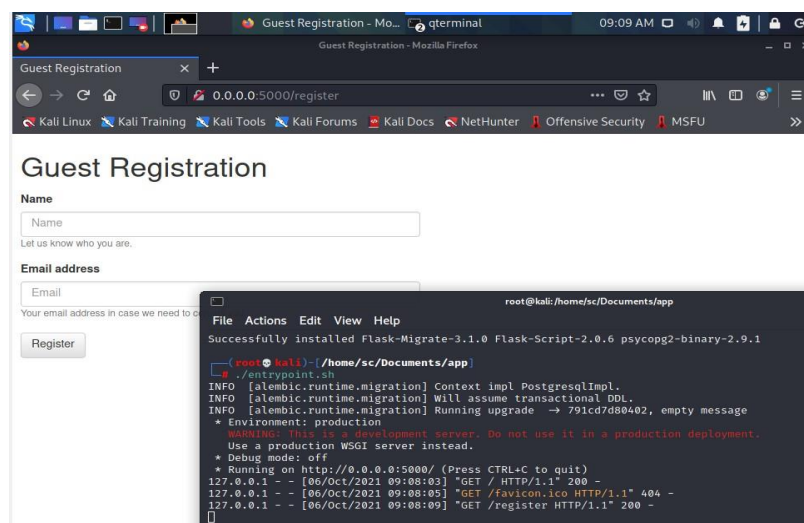
- Installer les requirements qui se trouvent dans le requirements.txt (pip3 install -r requirements.txt si pip3 n'est pas disponible vous pouvez l'installer via apt-get install python3-pip)
- Vous pouvez démarrer l'application en executant entrypoint.sh

On a bien créé la base de données avec PostgreSQL.

```
postgres@kali:/home/sc/Documents/app$ psql
psql (13.2 (Debian 13.2-1))
Type "help" for help.

postgres=# create database test;
CREATE DATABASE
postgres=# create user test with password 'test';
CREATE ROLE
postgres=# grant ALL ON DATABASE test to test;
GRANT
postgres=# ^Z
[1]+  Stopped                  psql
postgres@kali:/home/sc/Documents/app$ ./entrypoint.sh
Usage: flask [OPTIONS] COMMAND [ARGS] ...
Try 'flask --help' for help.

Error: No such command 'db'.
postgres@kali:/home/sc/Documents/app$ ^C
postgres@kali:/home/sc/Documents/app$ exit
exit
There are stopped jobs.
```



On a bien démarré l'application avec la commande `./entrypoint.sh`

1.2) Créez une image docker a partir du Dockerfile fourni il manque des lignes au niveau du Dockerfile qu'il faudra remplir (dans cette étape l'image docker contiendra uniquement l'application et se connectera sur le postgres du host ) /!\ il faudra modifier les fichier de configuration de votre postgres au niveau du host pour autoriser les connexions depuis le réseau docker 172.17.0.0./16 et aussi il ne faut pas oublier de configurer votre postgres pour écouter sur toute les adresses 0.0.0.0.

```
(sc@kali)-[~/Documents/app]
$ docker build -t tp1
Sending build context to Docker daemon 33.79kB
Step 1/8 : FROM tiangolo/uwsgi-nginx-flask:python3.6
--> 79d81e4ae7e1
Step 2/8 : WORKDIR /docker-fs
--> Using cache
--> 0e63d451970e
Step 3/8 : COPY . .
--> d2fd3a1d5e80
Step 4/8 : RUN pip install -r requirements.txt
--> Running in e2ef76c499d3
Collecting alembic
  Downloading alembic-1.7.4-py3-none-any.whl (209 kB)
Collecting appdirs
  Downloading appdirs-1.4.4-py2.py3-none-any.whl (9.6 kB)
Requirement already satisfied: click in /usr/local/lib/python3.6/site-packages (from -r requirements.txt (line 3)) (8.0.1)
Requirement already satisfied: Flask in /usr/local/lib/python3.6/site-packages (from -r requirements.txt (line 4)) (2.0.1)
Collecting Flask-Migrate
  Downloading Flask_Migrate-3.1.0-py3-none-any.whl (20 kB)
Collecting Flask-Script
  Downloading Flask-Script-2.0.6.tar.gz (43 kB)
Collecting Flask-SQLAlchemy
  Downloading Flask_SQLAlchemy-2.5.1-py2.py3-none-any.whl (17 kB)
Requirement already satisfied: itsdangerous in /usr/local/lib/python3.6/site-packages (from -r requirements.txt (line 8)) (2.0.1)
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.6/site-packages (from -r requirements.txt (line 9)) (3.0.1)
Collecting Mako
  Downloading Mako-1.1.5-py2.py3-none-any.whl (75 kB)
```

```
python - standa... qterminal Desktop 09:13 AM
sc@kali: ~/Documents/app

File Actions Edit View Help

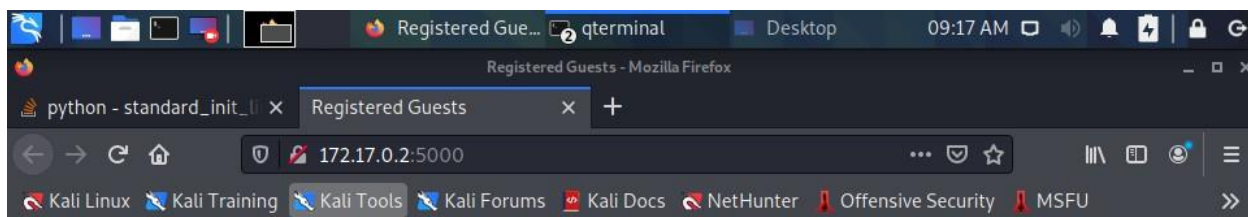
Building wheel for Flask-Script (setup.py): started
Building wheel for Flask-Script (setup.py): finished with status 'done'
Created wheel for Flask-Script: filename=Flask_Script-2.0.6-py3-none-any.whl size=14030 sha256=305fd683653618377fc01b6998e7bfb596057d72406d45b6cbc91cebe4a26329
Stored in directory: /root/.cache/pip/wheels/39/75/7b/11f0def239fcb1799611494670718eb7b59b633494cda10388
Successfully built Flask-Script
Installing collected packages: greenlet, SQLAlchemy, Mako, importlib-resources, pyparsing, Flask-SQLAlchemy, alembic, six, python-editor, pycpg2-binary, packaging, Flask-Script, Flask-Migrate, appdirs
Successfully installed Flask-Migrate-3.1.0 Flask-SQLAlchemy-2.5.1 Flask-Script-2.0.6 Mako-1.1.5 SQLAlchemy-1.4.25 alembic-1.7.4 appdirs-1.4.4 greenlet-1.1.2 importlib-resources-5.2.2 packaging-21.0 pycpg2-binary-2.9.1 pyparsing-2.4.7 python-editor-1.0.4 six-1.16.0
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
WARNING: You are using pip version 21.2.4; however, version 21.3 is available.
You should consider upgrading via the '/usr/local/bin/python -m pip install --upgrade pip' command.
Removing intermediate container e2ef76c499d3
--> 25f65805a4f6
Step 5/8 : EXPOSE 5000
--> Running in 0d3b05c7dbc7
Removing intermediate container 0d3b05c7dbc7
--> 020baf8b0863
Step 6/8 : ENV FLASK_APP=app.py
--> Running in dd8a4a371e6a
Removing intermediate container dd8a4a371e6a
--> 871bd959b87e
Step 7/8 : RUN chmod 750 /docker-fs/entrypoint.sh
--> Running in 8a63e0b0d84f
Removing intermediate container 8a63e0b0d84f
--> 71fbd10f0bdf
Step 8/8 : ENTRYPOINT ["/docker-fs/entrypoint.sh"]
--> Running in 8922c68d7a24
Removing intermediate container 8922c68d7a24
--> 32d5aefd648f
Successfully built 32d5aefd648f
Successfully tagged tp1:latest
```



Création de l'image avec la commande suivante : `docker build . -t tp1`

```
(sc@kali)-[~/Documents/app]
└─$ sudo docker run tp1
hello
INFO [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
* Serving Flask app 'app.py' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://172.17.0.2:5000/ (Press CTRL+C to quit)
```

Démarrer l'image docker avec la commande `sudo docker run tp1`.



## Registered Guests

Name Email

Register!

```
sc@kali: ~/Documents/app
File Actions Edit View Help
└─$ sudo docker run tp1
hello
INFO [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
* Serving Flask app 'app.py' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a producti
  Use a production WSGI server instead.
* Debug mode: off
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a producti
* Running on http://172.17.0.2:5000/ (Press CTRL+C to quit)
^C^C^C
^C^C^X^C172.17.0.1 - - [12/Oct/2021 13:16:10] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [12/Oct/2021 13:16:11] "GET /favicon.ico HTTP/1.1" 404
```

La connexion est bien établie avec la base de données.

1.3) Modifier votre dockerfile pour avoir votre serveur postgres au niveau de l'image générée

Voici les lignes entrées pour avoir le serveur postgres au niveau de l'image générée.

```
(sc@kali)~/Documents/app
$ cat Dockerfile
FROM tiangolo/uwsgi-nginx-flask:python3.6

RUN apt-get update && apt-get install -y postgresql-client postgresql-contrib

USER postgres
RUN /etc/init.d/postgresql start && \
    psql --command "CREATE USER test WITH PASSWORD 'test';" && \
    createdb -O test test
USER postgres

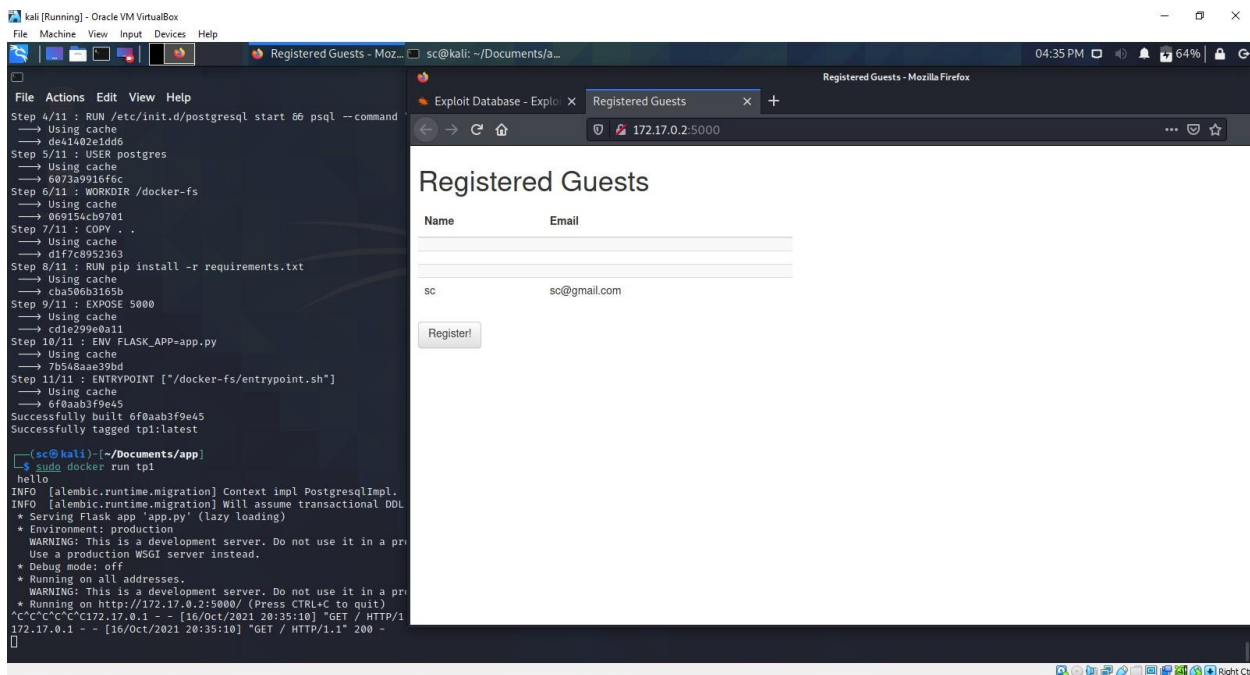
WORKDIR /docker-fs
COPY . .

RUN pip install -r requirements.txt

EXPOSE 5000
ENV FLASK_APP=app.py
#RUN chmod 750 /docker-fs/entrypoint.sh
ENTRYPOINT ["/docker-fs/entrypoint.sh"]

(sc@kali)~/Documents/app
```

La connexion à la base de données est bien établie d'après la capture ci-dessous.



Les requêtes sont bien visibles depuis le terminal

## Partie 2: Docker-compose

2.1) créer un fichier docker-compose permettant de mettre en place deux containers avec un pour l'application et un deuxième pour la base de données.

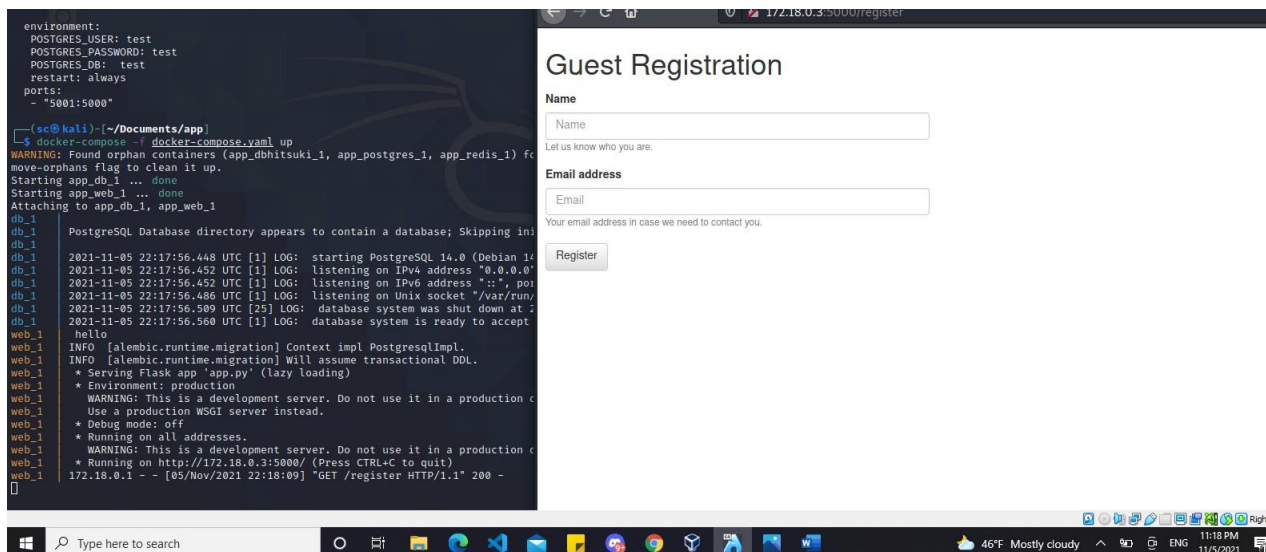
Contenue du docker-compose.yaml

```
(sc@kali)-[~/Documents/app]
$ cat docker-compose.yaml
version: "3.9"
services:
  web:
    build: .
    ports:
      - "5000:5000"
    restart: always
    environment:
      DBUSER: test
      DBPASS: test
      DBHOST: localhost
      DBNAME: test
      FLASK_APP: app.py
    depends_on:
      - db
  db:
    image: "postgres"

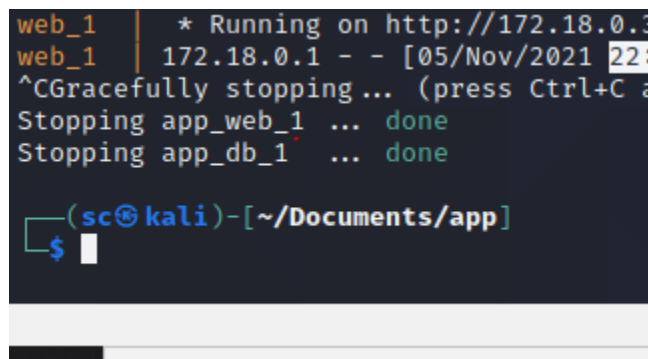
    environment:
      POSTGRES_USER: test
      POSTGRES_PASSWORD: test
      POSTGRES_DB: test
    restart: always
    ports:
      - "5001:5000"

(sc@kali)-[~/Documents/app]
$
```





On a bien la db et le web fonctionnels on le lançant depuis un docker compose qui permet de lancer plusieurs conteneurs



Arrêt des services (db et web)

2.2) créer un deuxième fichier docker compose permettant cette fois ci de générer deux containers pour l'application avec deux accès différents au niveau de la db l'application A1 aura le droit d'écriture sur la DB l'application A2 copie de l'application A1 n'aura que les droits de lecture sur la DB

On a créé un script `init.sh`, dans laquelle on crée deux user, dans laquelle on leur définit des droits de lecture et d'écriture.

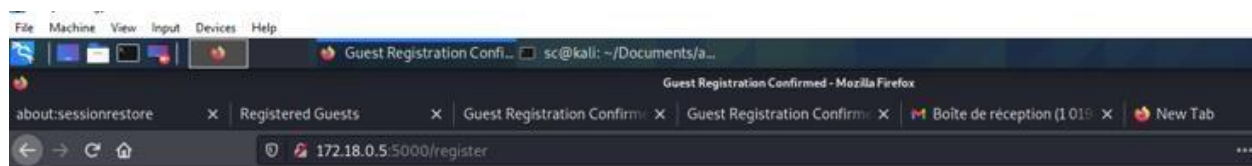
```
Workspace Trust  $ init.sh  X
C: > Users > Admin > Documents > docker tp > fichier > $ init.sh
1  #!/bin/bash
2
3  set -e
4
5
6
7  psql -v ON_ERROR_STOP=1 --username "$DBUSER" --dbname "$DBNAME" <<-EOSQL
8  CREATE USER test WITH PASSWORD 'test';
9  CREATE USER test2 WITH PASSWORD 'test2';
10
11
12  CREATE DATABASE test;
13  GRANT ALL ON DATABASE test TO test;
14
15  ALTER DEFAULT PRIVILEGES FOR ROLE test IN SCHEMA public GRANT SELECT ON TABLES TO test2;
16
17
18
19 EOSQL
20
```

```
File Actions Edit View Help
GNU nano 5.4
version: "3.9"
services:
  app1:
    build: .
    restart: always
    depends_on:
      - db
    ports:
      - "5005:5000"
    environment:
      DBUSER: test
      DBPASS: test
      DBHOST: db
      DBNAME: test
      FLASK_ENV: development

  app2:
    build: .
    restart: always
    depends_on:
      - db
    ports:
      - "5006:5000"
    environment:
      DBUSER: test2
      DBPASS: test2
      DBHOST: db
      DBNAME: test
      FLASK_ENV: development

  db:
    image: "postgres"
    volumes:
      - ./init.sh:/docker-entrypoint-initdb.b/init.sh
    environment:
      POSTGRES_USER: test
```

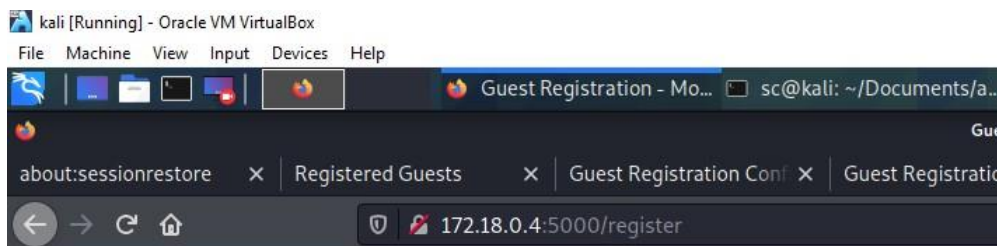
```
db:
  image: "postgres"
  volumes:
    - ./init.sh:/docker-entrypoint-initdb.b/init.sh
  environment:
    POSTGRES_USER: testmail.com
    POSTGRES_PASSWORD: testw
    POSTGRES_DB: test
  restart: always
```



## sqlalchemy.exc.ProgrammingError

sqlalchemy.exc.ProgrammingError: (psycopg2.errors.InsufficientPrivilege) permission denied for table guests:  
[SQL: INSERT INTO guests (name, email) VALUES (%(name)s, %(email)s) RETURNING guests.id]

On a une erreur quand on veut se connecter avec le deuxième user avec l'adresse 172.18.0.5 car il a  
Accès que a la lecture



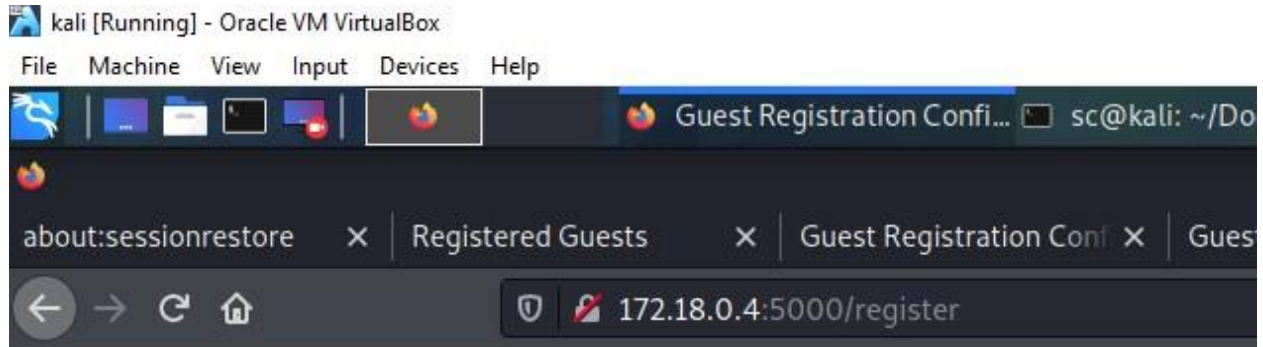
## Guest Registration

Name

Let us know who you are.

Email address

Your email address in case we need to contact you.



# You are confirmed!

Name: srikanth

Email: srikanth@gmail.com

[View all attendees](#)

Mais par contre on c'est bien connecter avec le premier user avec l'adresse suivante :172.18.0.4

2.3) modifier le code de A2 puisse démarrer car elle ne peut plus être utilisée vu que l'accès à la DB se fait uniquement en lecture

On modifie le fichier app.py

```
@app.route('/')
def view_registered_guests():
    from models import Guest
    guests = Guest.query.all()
    return render_template('guest_list.html', guests=guests)

@app.route('/register', methods=['GET'])
def view_registration_form():
    if os.version['DBUSER'] == "test2":
        return ("ERREUR")
    else :
        return render_template('guest_registration.html')

@app.route('/register', methods=['POST'])
def register_guest():
    from models import Guest
    name = request.form.get('name')
    email = request.form.get('email')

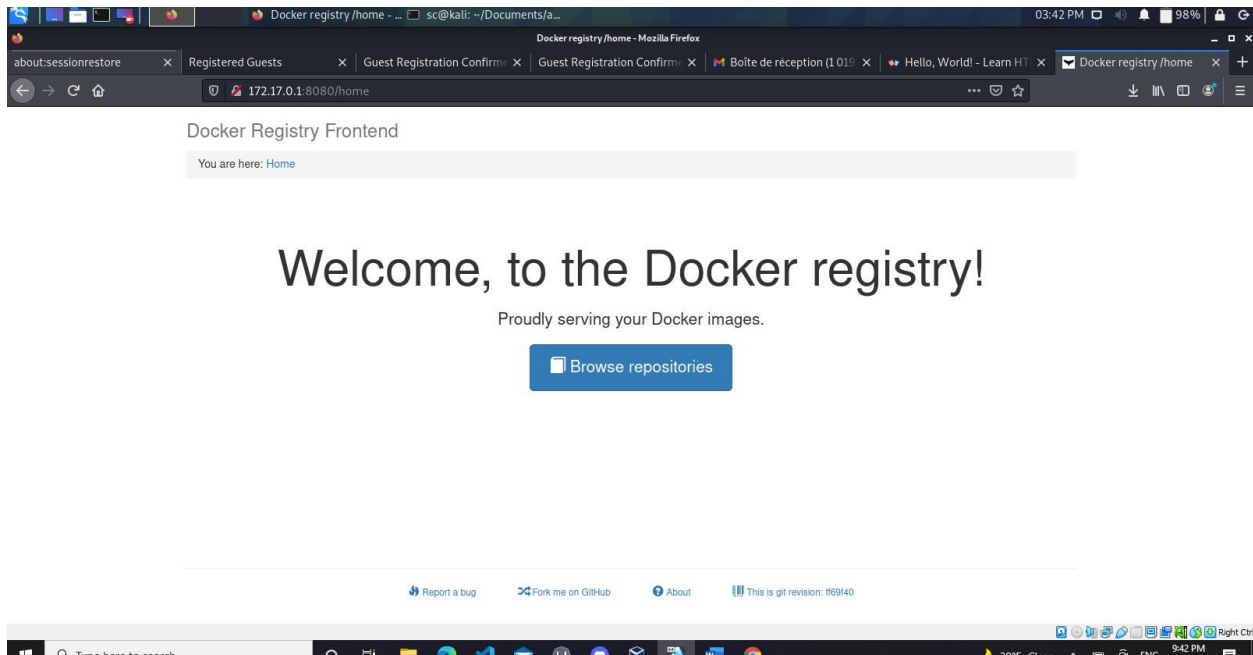
    guest = Guest(name, email)
    db.session.add(guest)
    db.session.commit()
```

Après avoir modifier le fichier app.py on obtient bien une erreur.

## Partie 3 : Docker-registry

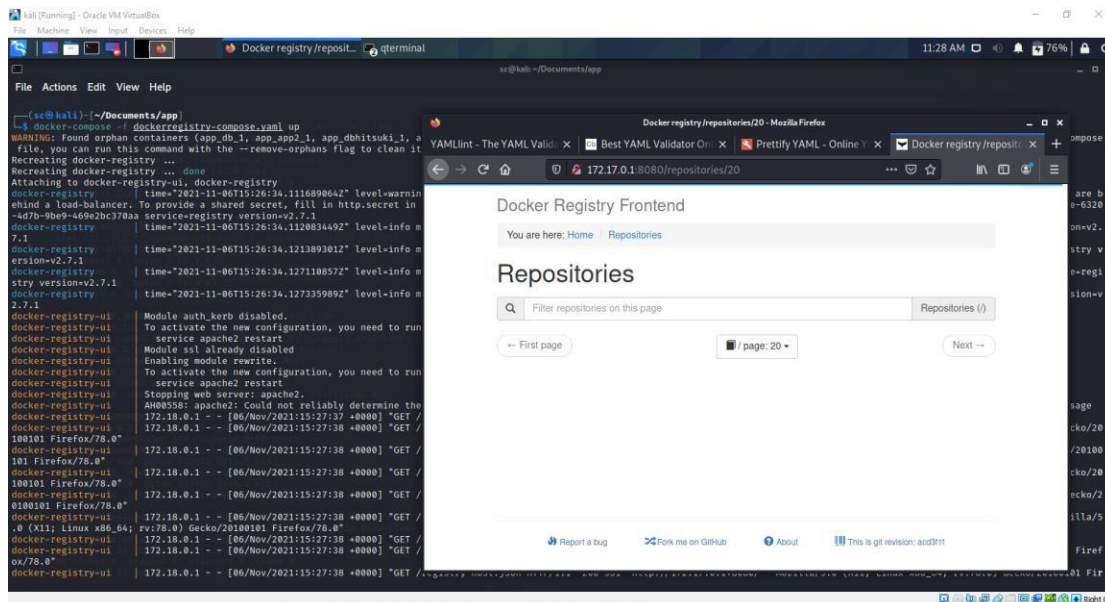
### 3.1) Mettrez en place localement un docker registry

#### Mise en place du docker-registry (local)



Comme on peut le voir ici on a mis en place le docker registry en utilisant le docker-registry frontend a l'adresse suivante : <https://hub.docker.com/r/konradkleine/docker-registry-frontend/>

Pour le mettre en place il fallait faire un pull : `docker pull konradkleine/docker-registry-frontend`



Voici le code pour mettre en place le docker registry

```
File Actions Edit View Help
GNU nano 5.4
version: '3.9'
services:
  docker-registry:
    image: 'registry:2'
    container_name: docker-registry1
    ports:
      - '5000:5000'
    restart: always
    volumes:
      - './volume:/var/lib/registry'
    environment:
      REGISTRY_STORAGE_DELETE_ENABLED: 'true'
  docker-registry-ui:
    image: 'konradkleine/docker-registry-frontend:v2'
    container_name: docker-registry-ui1
    ports:
      - '8080:80'
    environment:
      ENV_DOCKER_REGISTRY_HOST: docker-registry
      ENV_DOCKER_REGISTRY_PORT: 5000
```

3.2) tager les images générées précédemment et envoyer les au niveau de votre registry local

Dans le fichier hosts on configure l'adresse IP avec le nom domaine, ici avec mes initial et local.com

```
File Machine View Input Devices Help
Docker registry /reposit... qterminal
root@kali: /home/sc/Documents/app
File Actions Edit View Help
GNU nano 5.4 /etc/hosts
127.0.0.1 localhost
127.0.1.1 kali
172.16.174.128 www.parisestmagique.com
172.17.0.1 sc.local.com
# The following lines are desirable for IPv6 capable hosts
::1 localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```



```
(root@kali)-[/home/sc/Documents/app]
# cat /etc/docker/daemon.json
{
  "insecure-registries": ["sc.local.com:5000"]
}

(root@kali)-[/home/sc/Documents/app]
#
```

On a créé un fichier daemon.json dans le but de se connecter sans certificat.

```
(sc@kali)-[~/Documents/app]
$ ping sc.local.com
PING sc.local.com (172.17.0.1) 56(84) bytes of data:
64 bytes from sc.local.com (172.17.0.1): icmp_seq=1 ttl=64 time=0.058 ms
64 bytes from sc.local.com (172.17.0.1): icmp_seq=2 ttl=64 time=0.065 ms
64 bytes from sc.local.com (172.17.0.1): icmp_seq=3 ttl=64 time=0.066 ms
64 bytes from sc.local.com (172.17.0.1): icmp_seq=4 ttl=64 time=0.064 ms
```

La commande « docker images » affiche les images qui se trouve sur notre poste

```
(sc@kali)-[~/Documents/app]
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
tp1	latest	a09099b46402	3 days ago	1.67GB
app_app1	latest	dfd7ffc0f568	4 days ago	1.22GB
app_app2	latest	dfd7ffc0f568	4 days ago	1.22GB
sc.local.com:5006/sc/app_app1-sc	v1	dfd7ffc0f568	4 days ago	1.22GB
app_web	latest	10d6660d600f	3 weeks ago	1.22GB
postgres	9.6	beb690634356	3 weeks ago	200MB

Voici la commande pour tagger les images générées.

```
(sc@kali)-[~/Documents/app]
$ docker tag app_app1:latest sc.local.com:5000/sc/app1-sc:tpdocker

(sc@kali)-[~/Documents/app]
$ docker images
```

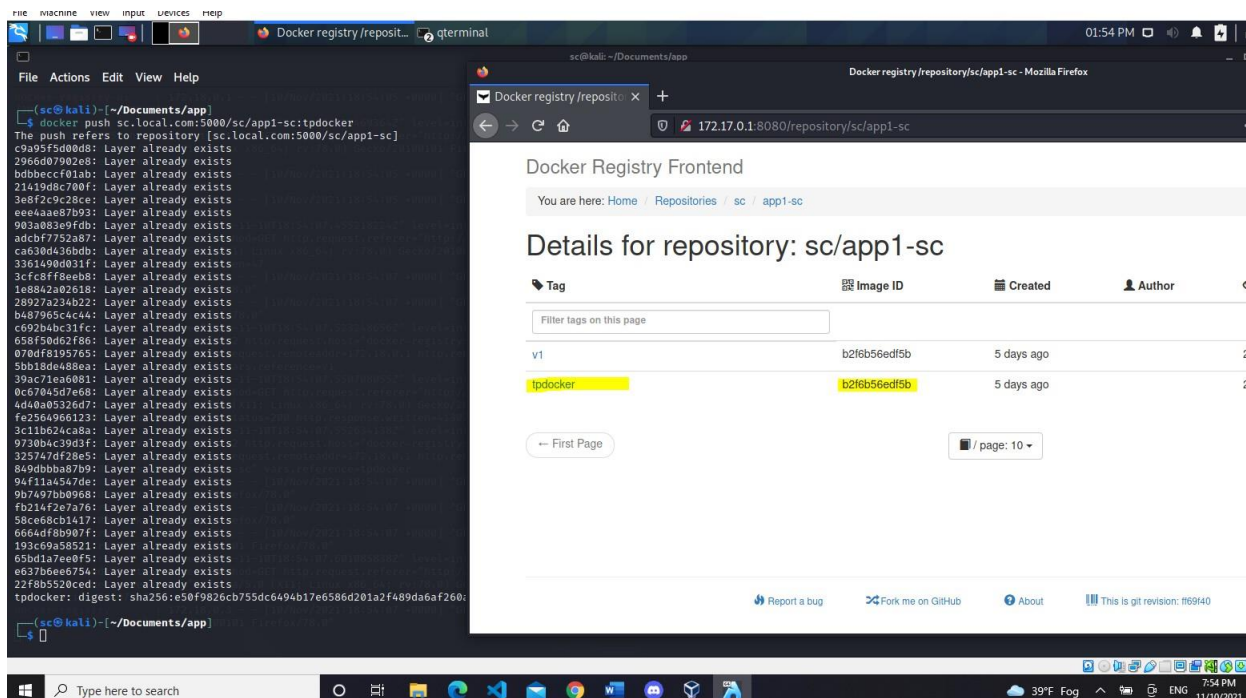
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
tp1	latest	a09099b46402	3 days ago	1.67GB
app_app1	latest	dfd7ffc0f568	4 days ago	1.22GB
app_app2	latest	dfd7ffc0f568	4 days ago	1.22GB
sc.local.com:5000/sc/app1-sc	tpdocker	dfd7ffc0f568	4 days ago	1.22GB
sc.local.com:5006/sc/app_app1-sc	v1	dfd7ffc0f568	4 days ago	1.22GB
app_web	latest	10d6660d600f	3 weeks ago	1.22GB

Voici la commande pour push l'image dans le registry (docker push sc.local.com :5000/sc/app1-sc :tpdocker

```
(sc@kali)-[~/Documents/app]
$ docker push sc.local.com:5000/sc/app1-sc:tpdocker
The push refers to repository [sc.local.com:5000/sc/app1-sc]
c9a95f5d00d8: Layer already exists
2966d07902e8: Layer already exists
bdbbeccf01ab: Layer already exists
21419d8c700f: Layer already exists
3e8f2c9c28ce: Layer already exists
eee4aae87b93: Layer already exists
903a083e9fdb: Layer already exists
adcbf7752a87: Layer already exists
ca630d436bdb: Layer already exists
3361490d031f: Layer already exists
3cfc8ff8eeb8: Layer already exists
1e8842a02618: Layer already exists
28927a234b22: Layer already exists
b487965c4c44: Layer already exists
c692b4bc31fc: Layer already exists
658f50d62f86: Layer already exists
070df8195765: Layer already exists
5bb18de488ea: Layer already exists
39ac71ea6081: Layer already exists
0c67045d7e68: Layer already exists
4d40a05326d7: Layer already exists
fe2564966123: Layer already exists
3c11b624ca8a: Layer already exists
9730b4c39d3f: Layer already exists
325747df28e5: Layer already exists
849dbbba87b9: Layer already exists
94f11a4547de: Layer already exists
9b7497bb0968: Layer already exists
fb214f2e7a76: Layer already exists
58ce80cb1417: Layer already exists
6664df8b907f: Layer already exists
193c69a58521: Layer already exists
65bd1a7ee0f5: Layer already exists
e637b6ee6754: Layer already exists
22f8b5520ced: Layer already exists
tpdocker: digest: sha256:e50f9826cb755dc6494b17e6586d201a2f489da6af260ace65df4d5c47854d63 size: 7635

(sc@kali)-[~/Documents/app]
```

On a bien push l'image dans le registry , et on a bien le hash générées.





3.3) reconstruire les images après une légère modification du code et taguer les avec un nouveau tag permettant de les différencier de la première version

Création d'un nouveau tag à partir de la première version

```
(sc@kali)-[~/Documents/app]
$ docker tag sc.local.com:5000/sc/app1-sc:tpdocker sc.local.com:5000/sc/app1-sc:tpdocker2

(sc@kali)-[~/Documents/app]
$
```

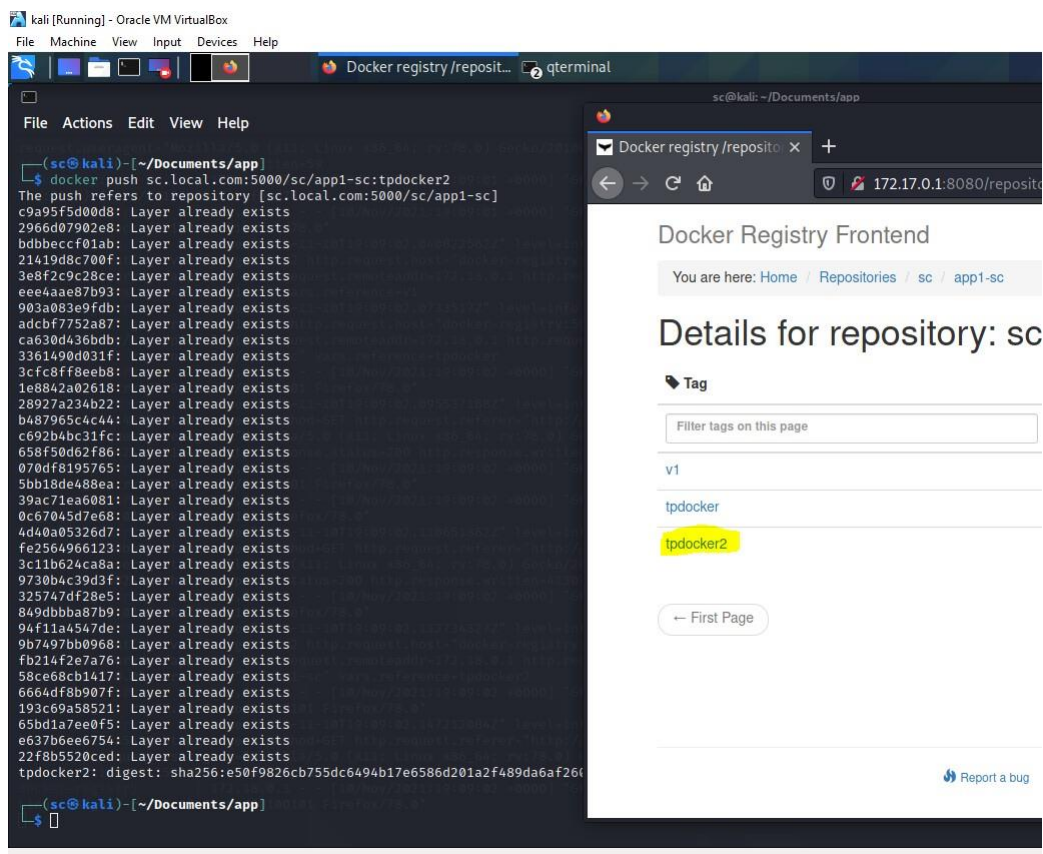
On a bien deux tags qui permettent de les différencier.

```
(sc@kali)-[~/Documents/app]
$ docker tag sc.local.com:5000/sc/app1-sc:tpdocker sc.local.com:5000/sc/app1-sc:tpdocker2

(sc@kali)-[~/Documents/app]
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
tp1	latest	a09099b46402	3 days ago	1.67GB
app_app2	latest	dfd7ffc0f568	4 days ago	1.22GB
sc.local.com:5000/sc/app1-sc	tpdocker	dfd7ffc0f568	4 days ago	1.22GB
sc.local.com:5000/sc/app1-sc	tpdocker2	dfd7ffc0f568	4 days ago	1.22GB

3.4) push les nouvelles images sur votre registry local



On a push la deuxième image générer à partir de la première image avec le tag suivant : tpdocker2

3.5) supprimer de votre registry local les premières image tagués

Voici la configuration du fichier config.yml , on a modifier le fichier config.yml qui se trouve dans le container. Pour ajouter les lignes suivante delete :enabled :true, dans le but d'autoriser la suppression.

```
(sc@kali)-[/etc/docker/registry]
$ sudo docker cp config.yml a7d321e9156e:/etc/docker/registry

(sc@kali)-[/etc/docker/registry]
$ docker exec registry cat /etc/docker/registry/config.yml
version: 0.1
log:
  fields:
    service: registry
storage:
  cache:
    blobdescriptor: inmemory
  filesystem:
    rootdirectory: /var/lib/registry
delete:
  enabled: true
http:
  addr: :5000
  headers:
    X-Content-Type-Options: [nosniff]
health:
  storagedriver:
    enabled: true
    interval: 10s
    threshold: 3

(sc@kali)-[/etc/docker/registry]
$
```

```
(sc@kali)-[/etc/docker/registry]
$ sudo curl -X GET sc.local.com:5000/v2/_catalog
{"repositories":["sc/app1-sc"]}

(sc@kali)-[/etc/docker/registry]
$
```

Ici, on affiche les noms des répertoires.

```
(sc@kali)-[/etc/docker/registry]
$ sudo curl -X GET sc.local.com:5000/v2/sc/app1-sc/tags/list
{"name": "sc/app1-sc", "tags": ["tpdocker2"]}

(sc@kali)-[/etc/docker/registry]
$
```

Ici, on affiche à liste des tags,

J'étais obligé de changer d'image après un problème technique, donc j'ai choisie l'image hello-world.

```
(sc@sc)-[/etc/docker/registry]
$ sudo curl -i -X DELETE localhost:5000/v2/hello-world/manifests/sha256:f54a58bc1aac5ea1a25d796ae155dc228b3f0e11d046ae276b39c4bf2f13d8c4
HTTP/1.1 202 Accepted
Docker-Distribution-API-Version: registry/2.0
X-Content-Type-Options: nosniff
Date: Fri, 12 Nov 2021 19:19:12 GMT
Content-Length: 0

(sc@sc)-[/etc/docker/registry]
$
```

Et on effectue la commande suivante pour supprimer les tags du registre, et on voit que la commande est acceptée, l'hash choisi et le hash au moment de la création de l'image.

```
(sc@sc)-[/etc/docker/registry]
$ sudo curl -i -X DELETE localhost:5000/v2/hello-world/manifests/sha256:f54a58bc1aac5ea1a25d796ae155dc228b3f0e11d046ae276b39c4bf2f13d8c4
HTTP/1.1 404 Not Found
Content-Type: application/json; charset=utf-8
Docker-Distribution-API-Version: registry/2.0
X-Content-Type-Options: nosniff
Date: Fri, 12 Nov 2021 19:22:58 GMT
Content-Length: 70

{"errors":[{"code":"MANIFEST_UNKNOWN","message":"manifest unknown"}]}

(sc@sc)-[/etc/docker/registry]
$
```

D'après la capture au-dessus on vérifie que l'image a bien été supprimée et c'est le cas car il nous renvoie une erreur.



3.6) déclencher le garbage collector au niveau de votre registry pour nettoyer les blob non référencés (suite à la suppression du tag)

On a déclencher le Garbage collector pour différentes images et on voit bien les blobs accompagnés de leur hash

```
(sc@kali)-[/etc/docker/registry]
$ docker exec -it registry bin/registry garbage-collect --dry-run /etc/docker/registry/config.yml
sc/app1-sc: marking manifest sha256:e50f9826cb755dc6494b17e6586d201a2f489da6af260ace65df4d5c47854d63
sc/app1-sc: marking blob sha256:dfd7ffc0f5689ad6bd26d0f9bce978e43622d333fd72c60c210bb942da059d56
sc/app1-sc: marking blob sha256:5e7b6b7bd506c12399d65977c0ba8dd02824dc5d0e65fc55d7382da889bdac7d
sc/app1-sc: marking blob sha256:fd67d668d6911bf21ad4701522e1ed3af416837433fdbba3f88cfff06a23e23861
sc/app1-sc: marking blob sha256:1ae016bc26876abbd5e952133b02b04d4c1dae1bc75a3d9386250e4797ccd87a
sc/app1-sc: marking blob sha256:0b0af05a4d868593f859eaa5815fc1c3596d77318a4ed756f3865a5fa3f290c6
sc/app1-sc: marking blob sha256:ca4689f0588c1ba01108f5c6d33943d3d542ae339fd526390d07c7a00ded6473
sc/app1-sc: marking blob sha256:981ebf72add42058f70231650b421de73b04336f9ad09fa02dbfe634a9761950
sc/app1-sc: marking blob sha256:94ed2ee963370e769d5b3400af5c458e9162d80749450bd59c23304e08a297a2
sc/app1-sc: marking blob sha256:e49e07f12c652ae50a250eb96f984976bd253dfb659d819fa6a61187abced450
sc/app1-sc: marking blob sha256:0f08bac7320eccc65f5b4ba2a758fd1d62e584617d333d524a6cf4eed8ec216a
sc/app1-sc: marking blob sha256:81a0d7f51d3df4bea6012a0859ae575846eedb9671109c2a4accc7686c56bd67
sc/app1-sc: marking blob sha256:45b3ee0e1290263a5c01a2cd54f2d09a3bd66311209694bf252f537d4e47e3b9
sc/app1-sc: marking blob sha256:ce9e707137700f5617677fd510c4bb89e7960d174160892cfc501384459c2c9c
sc/app1-sc: marking blob sha256:e882f9f0a345dc1f210d03b8bb63ffa6ccf18e6af7b5b904fda10e22fe345abb
sc/app1-sc: marking blob sha256:cd67f5e189949f304f6519a6c667a1f24d3b2a1e3a4ad7211393782ae6cbc359
sc/app1-sc: marking blob sha256:fac5e97674c4db9bb52feb9af78a9e69f36300ac25618f789850edbcb8b1bf6a
sc/app1-sc: marking blob sha256:8e8b6cae656caf446650460c6cfab42b151d059cf61cc49a65ed2b577715c5eb
sc/app1-sc: marking blob sha256:d6aedd13aa3a09442bc9a1bcde977adca82b4444c6a68017e4682bc339575a2
sc/app1-sc: marking blob sha256:8af53450c1135986ecd083fe520ce6a5f1ab8976617bde095d6bf31786590201
sc/app1-sc: marking blob sha256:f4085b389d61c7f50b508048a01d75b0c6225277b7ed73d2a24ae71a35ceab73
sc/app1-sc: marking blob sha256:bd0175ca87a37b4da9b05f0e51a655e04861b7e0e6c062aa3b9e211ede9d12d6
sc/app1-sc: marking blob sha256:8409f1b445a628521a0deb2b3dfa5424dc794e432a62403ee17a96af71cb6db8
sc/app1-sc: marking blob sha256:3e06db044205e56991cc9d9ae0a5ff926d2da8bc7af253d832544e67acc7ff31
sc/app1-sc: marking blob sha256:bc4fdf3baacd4f77f1dc5ca7925071cb802cd2836144d6a065df4d52bb17e79c
sc/app1-sc: marking blob sha256:a34149461b9fdfdade5d121b508a170e4525fad352fe7c812103112ef797411
sc/app1-sc: marking blob sha256:52b8d7e7503940fe06cc8935ad186527bf107920f89bcd8a975d69c4c75d0f62
sc/app1-sc: marking blob sha256:829e42bb45c79956c2694bb7cd36d3ef9e3c09c7833aaa7bf086df37a4104f7e
sc/app1-sc: marking blob sha256:b6d7f7b867d36494bc287b69e89d6da25db1bcf88c2c68f7ef0d3d01074b525
sc/app1-sc: marking blob sha256:3a20ff7974682494ffc50caccba0832a5664b3653357a245515a11b40183e790
sc/app1-sc: marking blob sha256:aab9399febebe00dd6bb8dd2318204633e025df20bec527b4d967a5eddb8fc7c
sc/app1-sc: marking blob sha256:ad39310f24a72f1ad87b07619607452e1bd45884d57d74403a0086c9548a3928
sc/app1-sc: marking blob sha256:3304f23ff92a27cab6e64ad6911e4f2097ff10b7d648a5c627682276a19f8604
sc/app1-sc: marking blob sha256:d88aa20c47cd47758af75861d4b67e62d9d745a9cfff41b08fa74489343a50
sc/app1-sc: marking blob sha256:6553ec8be8fbd0175d148a57bc193d53efba8c9ad5b42e7ff8b1d3b87d7790c6
sc/app1-sc: marking blob sha256:06087a32c2acbb0c15a05a26edd4e6c91e6d43b02b9ae1e70db67cf510b367cd
sc/app1-sc: marking blob sha256:b56fb0791af077b4a38fca077cfa512ca0de0031e7117508227fdb24c708156

37 blobs marked, 1 blobs and 0 manifests eligible for deletion
blob eligible for deletion: sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
```

```
(sc@sc)-[/etc/docker/registry]
$ docker exec -it registry bin/registry garbage-collect --dry-run /etc/docker/registry/config.yml
hello-world

0 blobs marked, 4 blobs and 0 manifests eligible for deletion
blob eligible for deletion: sha256:2db29710123e3e53a794f2694094b9b4338aa9ee5c40b930cb8063a1be392c54
blob eligible for deletion: sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
blob eligible for deletion: sha256:f54a58bc1aac5ea1a25d796ae155dc228b3f0e11d046ae276b39c4bf2f13d8c4
blob eligible for deletion: sha256:feb5d9fea6a5e9606aa995e879d862b825965ba48de054caab5ef356dc6b3412

(sc@sc)-[/etc/docker/registry]
$
```



Le Garbage collector ou ramasse-miette est le processus de suppression des blobs du système de fichiers lorsqu'ils ne sont plus référencés par un manifeste. Les blobs peuvent inclure à la fois des couches et des manifestes.

## Partie 4: Gestion du cycle de vie des images:

4.1) Construire un script qui détecte la dernière version de l'image hébergée sur le registry et tag automatiquement la nouvelle image avec la version suivante. Exemple : dernier image tag=1.2 image suivante tag > 1.2 vous pouvez choisir librement les règles de tag

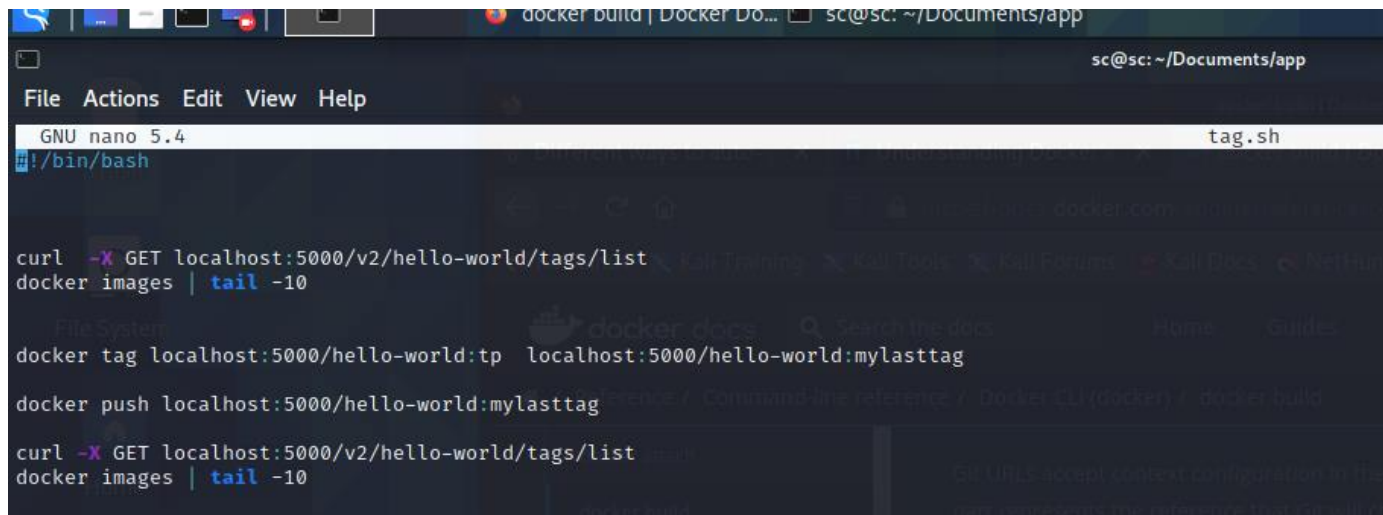
4.2) modifier le script de 4-1 pour automatiquement tester le bon fonctionnement de votre application, tag et push l'image.

```
(sc@sc)-[~/Documents/app]
$ ./tag.sh
{"name":"hello-world","tags":["v1","last","lasttag"]}
hello-world      last      feb5d9fea6a5  7 weeks ago  13.3kB
hello-world      lasttag   feb5d9fea6a5  7 weeks ago  13.3kB
hello-world      latest    feb5d9fea6a5  7 weeks ago  13.3kB
hello-world      v1        feb5d9fea6a5  7 weeks ago  13.3kB
localhost:5000/hello-world last      feb5d9fea6a5  7 weeks ago  13.3kB
localhost:5000/hello-world lasttag   feb5d9fea6a5  7 weeks ago  13.3kB
localhost:5000/hello-world tp        feb5d9fea6a5  7 weeks ago  13.3kB
localhost:5000/hello-world v1        feb5d9fea6a5  7 weeks ago  13.3kB
registry          2         b2cb11db9d3d  2 months ago  26.2MB
registry          latest     b2cb11db9d3d  2 months ago  26.2MB
The push refers to repository [localhost:5000/hello-world]
e07ee1baac5f: Layer already exists
mylasttag: digest: sha256:f54a58bc1aac5ea1a25d796ae155dc228b3f0e11d046ae276b39c4bf2f13d8c4 size: 525 is all the
{"name":"hello-world","tags":["mylasttag","v1","last","lasttag"]}
hello-world      lasttag   feb5d9fea6a5  7 weeks ago  13.3kB
hello-world      latest    feb5d9fea6a5  7 weeks ago  13.3kB
hello-world      v1        feb5d9fea6a5  7 weeks ago  13.3kB
localhost:5000/hello-world last      feb5d9fea6a5  7 weeks ago  13.3kB
localhost:5000/hello-world lasttag   feb5d9fea6a5  7 weeks ago  13.3kB
localhost:5000/hello-world mylasttag feb5d9fea6a5  7 weeks ago  13.3kB
localhost:5000/hello-world tp        feb5d9fea6a5  7 weeks ago  13.3kB
localhost:5000/hello-world v1        feb5d9fea6a5  7 weeks ago  13.3kB
registry          2         b2cb11db9d3d  2 months ago  26.2MB
registry          latest     b2cb11db9d3d  2 months ago  26.2MB

(sc@sc)-[~/Documents/app]
$
```

Le script détecte bien les derniers tag, ici le tag initial était latest et on l'a modifier avec le tag suivant « mylasttag » et on a bien push l'image comme on le voit dans la capture , et il est aussi dans le docker registry.

Voici le script :



```
GNU nano 5.4 tag.sh
#!/bin/bash

curl -X GET localhost:5000/v2/hello-world/tags/list
docker images | tail -10

docker tag localhost:5000/hello-world:tp localhost:5000/hello-world:mylasttag

docker push localhost:5000/hello-world:mylasttag

curl -X GET localhost:5000/v2/hello-world/tags/list
docker images | tail -10
```

# ANNEXE

Les fichiers utilisés pour ce TP sont les suivantes :

- docker-compose.yaml
- Dockerfile
- dockerregistry-compose.yaml
- entrypoint.sh
- seconddocker-compose.yaml
- config.yml
- tag.sh
- app.py
- nginx.conf
- daemon.json
- hosts

Pour la partie 4 on devait ajouter ces lignes suivantes :

```
(sc@sc)-[/etc/nginx]
$ cat nginx.conf
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections 768;
    # multi_accept on;
}

location / {

    if ($http_user_agent ~ "^(docker\/1\.(3|4|5(?:!\. [0-9]-dev)) | Go). *$"){
        return 404;
    }

    proxy_pass http://localhost:5000;
    proxy_set_header Host $http_host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_read_timeout 900;
}
```

La commande qui permettait d'éditer un fichier depuis un conteneur est la suivante :

### Depuis **DockerContainer** au **LocalMachine**

```
docker cp containerId:/sourceFilePath/someFile.txt C:/localMachineDestinationFolder
```

On envoie le fichier sur notre machine local et on l'édite avec nano, vim etc.. ,et par la suite on renvoie le fichier dans le conteneur avec la commande en-dessous.

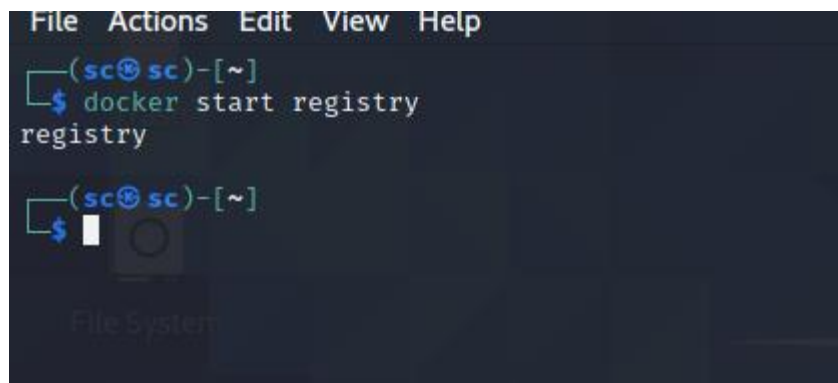
### Depuis **LocalMachine** au **DockerContainer**

```
docker cp C:/localMachineSourceFolder/someFile.txt containerId:/containerDestinationFolder
```

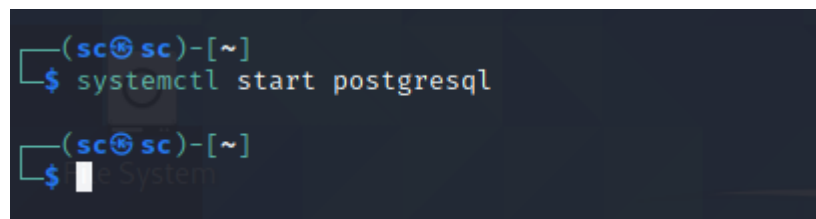
Ou sinon on aurait pu le faire avec les volumes depuis un docker-compose.

Également ne pas oublier de démarrer le registre avec la commande suivante :

```
docker start registry
```

A terminal window with a dark background and light-colored text. The window has a menu bar at the top with 'File', 'Actions', 'Edit', 'View', and 'Help'. The prompt is '(sc@sc)-[~]'. The command '\$ docker start registry' is entered, and the output 'registry' is shown on the next line. The prompt is then '(sc@sc)-[~]' again, followed by '\$' and a cursor. A faint 'File System' watermark is visible in the background.

Pour activer la base de données (PostgreSQL)

A terminal window with a dark background and light-colored text. The window has a menu bar at the top with 'File', 'Actions', 'Edit', 'View', and 'Help'. The prompt is '(sc@sc)-[~]'. The command '\$ systemctl start postgresql' is entered. The prompt is then '(sc@sc)-[~]' again, followed by '\$' and a cursor. A faint 'File System' watermark is visible in the background.