Rendu examen final docker.

- Conteneurisation de l’application web.

Nous avons tout d'abord commencer par créer un fichier Dockerfile avec toutes les commandes utiles à notre projet :

```
FROM python:3.6-alpine
WORKDIR /opt
RUN pip3 install flask==1.1.2
ENV ODOO_URL="192.168.99.10:8069"
ENV PGADMIN_URL="192.168.99.10:5055"
COPY . /opt
#COPY ./templates/index.html ./templates/
EXPOSE 8080
ENTRYPOINT ["python3", "app.py"]
```

Ensuite, nous avons executé la commande docker build suivi du nom de l'image que nous voulions créer :

```
[vagrant@master docker-exams-1]$ docker build -t ic-webapp:1.0 .
Sending build context to Docker daemon  1.973MB
Step 1/8 : FROM python:3.6-alpine
 ---> 3a9e80fa4606
Step 2/8 : WORKDIR /opt
 ---> Using cache
 ---> e6e9ba448b5d
Step 3/8 : RUN pip3 install flask==1.1.2
 ---> Using cache
 ---> c6e3772cf88f
Step 4/8 : ENV ODOO_URL="192.168.99.10:8069"
 ---> Using cache
 ---> 04f4f2b7d06f
Step 5/8 : ENV PGADMIN_URL="192.168.99.10:5055"
 ---> Using cache
 ---> 19112840859e
Step 6/8 : COPY . /opt
 ---> fbc66a4bb783
Step 7/8 : EXPOSE 8080
 ---> Running in 3c01dc7eb2e7
Removing intermediate container 3c01dc7eb2e7
 ---> 56c6274f31f6
Step 8/8 : ENTRYPOINT ["python3", "app.py"]
 ---> Running in a28e70cd6a3e
Removing intermediate container a28e70cd6a3e
 ---> c8b7c1123871
Successfully built c8b7c1123871
Successfully tagged ic-webapp:1.0
[vagrant@master docker-exams-1]$
```

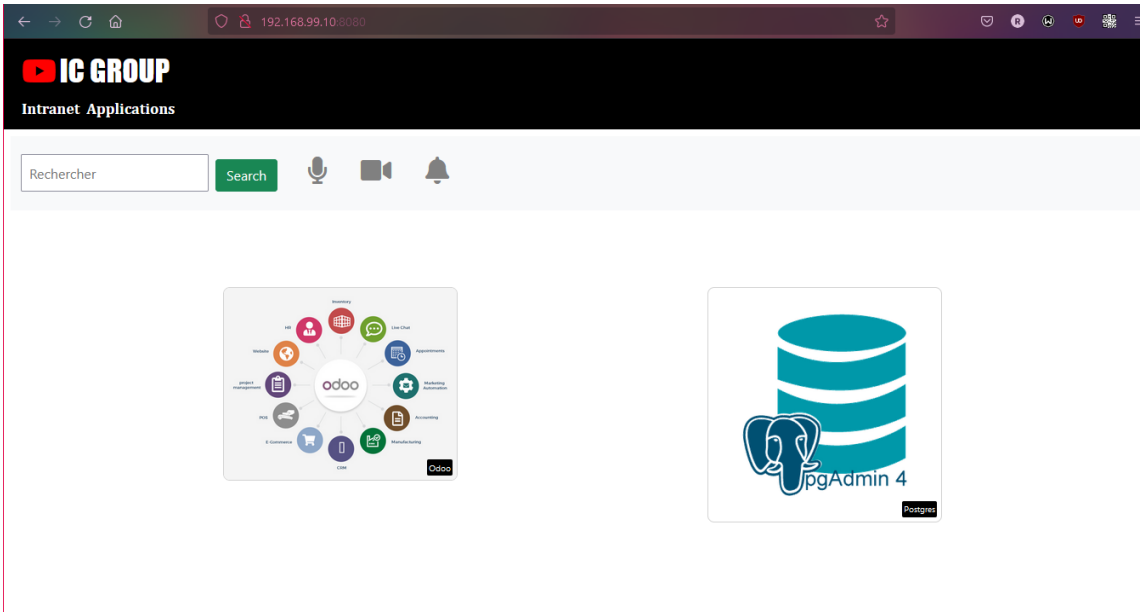
Avant de lancer le run de notre image, on verifie à supprimer tout les containers existants:

```
[vagrant@master docker-exams-1]$ docker rm -f ic-webapp odoo postgres
ic-webapp
odoo
postgres
```

Puis après avoir fait le build et le rm on fait un run en le mettant en arrière plan grace à "-d", on lui attribue un port libre avec “-p”, puis la machine locale “8080” et le conteneur “8080” :

```
[vagrant@master docker-exams-1]$ docker run -d -p 8080:8080 --name ic-webapp ic-webapp:1.0
940c562506592731c1f065f4d4cad37ef06e2a561cf3126d9a893665a2383335
```

Grace à celà, nous avons pu observer que nous avions bel et bien récupérer l'image du site web sur notre adresse ip au port indiqué: 192.168.99.10:8080 Voici donc le rendu que nous avons eu :



- Private registery

Nous avons créer un registre privé, pour ce faire nous avons executé ces commandes : D'abord, créons le registre:

```
[vagrant@master docker-exams-1]$ docker run -d -p 5001:5000  --name registry-ic-webapp registry:2
580eaa8b8ccdd9b7dd362433c9c423b25b21d39ce497a9d52a821c6e98bdaa5c
```

De plus, nous créons une interfaceweb pour visualiser notre registre privé :

```
[vagrant@master docker-exams-1]$ docker run -d -p 5002:80 -e REGISTRY_URL=http://registry-ic-webapp:5000 -e DELETE_IM
AGES=true -e REGISTRY_TITLE="IC WEBAPP REGISTERY" joxit/docker-registry-ui:static
Unable to find image 'joxit/docker-registry-ui:static' locally
static: Pulling from joxit/docker-registry-ui
540db60ca938: Pull complete
197dc8475a23: Pull complete
39ea657007e5: Pull complete
37afbf7d4c3d: Pull complete
0c01f42c3df7: Pull complete
d590d87c9181: Pull complete
3333c94ae44f: Pull complete
33d7cca6fc9f: Pull complete
076b2dd9bdd1: Pull complete
b70198f04ee7: Pull complete
1fb6c5acc953: Pull complete
Digest: sha256:b0657b6be748173583516e411bd71552e54cb7d5dda94964726297ce8774415c
Status: Downloaded newer image for joxit/docker-registry-ui:static
128ce7b919a19c835207e89afb4e27b34ef0fad23300ac63d9ae9ee77388815d
[vagrant@master docker-exams-1]$
```

Puis, nous ajoutons un tag à notre image pour pouvoir le push correctement sur notre registre privé :

```
[vagrant@master docker-exams-1]$ docker tag ic-webapp:1.0 localhost:5000/ic-webapp:1.0
```

Pour enfin faire un push final sur le registre:

```
[vagrant@master docker-exams-1]$ docker push localhost:5001/ic-webapp:1.0
The push refers to repository [localhost:5001/ic-webapp]
286c01e39050: Pushed
0f39ac867d0b: Pushed
3156423bd38f: Pushed
efa76becf38b: Pushed
671e3248113c: Pushed
1965cfbef2ab: Pushed
8d3ac3489996: Pushed
1.0: digest: sha256:12198922a155e705fcd350f39b05bb4925dc37cccd8f3f6db97f034b7a334985 size: 1790
```

- Docker-compose

Nous avons ensuite créé un fichier docker-compose.yml pour simplifier nos déploiements sur odoo et pgadmin. Voici la structure de notre fichier : Tout d'abord, nous devons créer l'environnement sur chaques services. Nous avons ic-webapp:

```yaml
version: '3.3'

services:
  ic-webapp:
    image: ic-webapp:1.0
    container_name: ic-webapp
    build:
      context: .
    environment:
      - ODOO_URL=https://www.odoo.com/fr_FR/web/login
      - PGADMIN_URL=http://192.168.99.10:5055/login?next=%2F
    depends_on:
      - postgres
    networks:
      - net-webapp
    ports:
      - 8080:8080
    # command: python3 app.py
    restart: unless-stopped
```

postgres:

```yaml
  postgres:
    image: postgres
    container_name: postgres
    environment:
      - POSTGRES_USER=admin
      - POSTGRES_PASSWORD=QWERTY!
    volumes:
      - v_postgres:/data/postgres
    ports:
      - 5432:5432
    networks:
      - net-webapp
    restart: unless-stopped

```

pgadmin :

```
35    pgadmin:
36      image: dpage/pgadmin4
37      container_name: pgadmin
38      ports:
39        - 5055:80
40      environment:
41        - PGADMIN_DEFAULT_EMAIL=abc@gmail.com
42        - PGADMIN_DEFAULT_PASSWORD=AZERY!
43      networks:
44        - net-webapp
45      depends_on:
46        - postgres
47      restart: unless-stopped
48
```

odoo :

```
49    odoo:
50      image: odoo
51      container_name: odoo
52      ports:
53        - 8069:8069
54      environment:
55        - POSTGRES_USER=admin
56        - POSTGRES_PASSWORD=QWERTY!
57      depends_on:
58        - postgres
59      networks:
60        - net-webapp
61      restart: unless-stopped
62
```

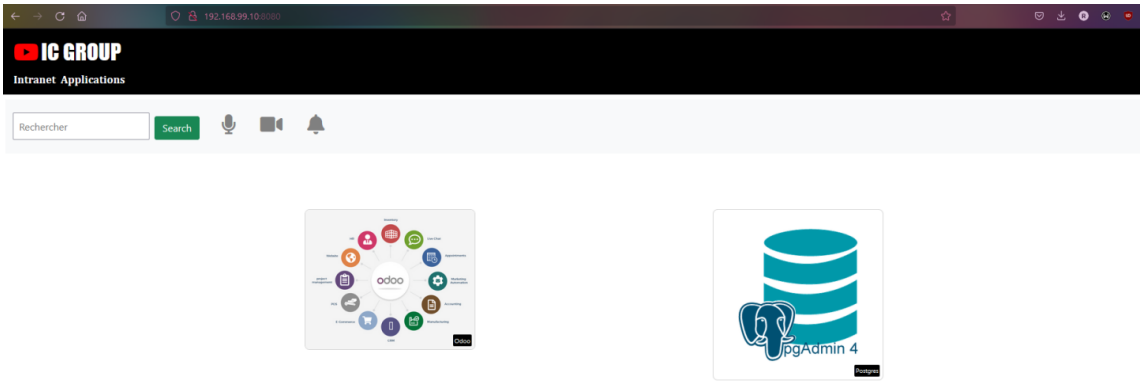
les networks et les volumes utilisés:

```
63  networks:
64    net-webapp:
65      driver: bridge
66
67  volumes:
68    v_postgres:
```

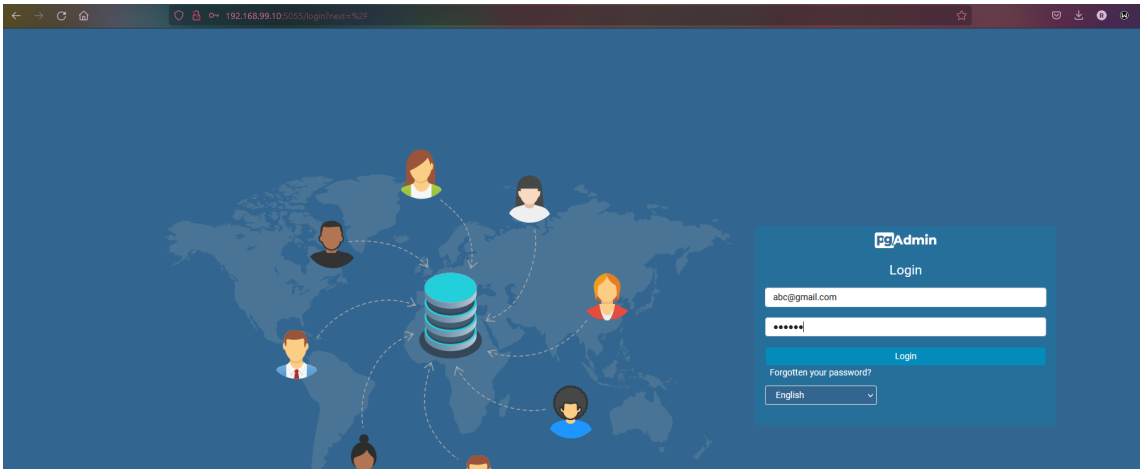
Après celà, on lance la commande docker-compose up --build

```
[vagrant@master docker-exams-1]$ docker-compose up --build
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on the current node

To deploy your application across the swarm, use `docker stack deploy`.

Building ic-webapp
Step 1/8 : FROM python:3.6-alpine
 ---> 3a9e80fa4606
Step 2/8 : WORKDIR /opt
 ---> Using cache
 ---> e6e9ba448b5d
Step 3/8 : RUN pip3 install flask==1.1.2
 ---> Using cache
 ---> c6e3772cf88f
Step 4/8 : ENV ODOO_URL="192.168.99.10:8069"
 ---> Using cache
 ---> 04f4f2b7d06f
Step 5/8 : ENV PGADMIN_URL="192.168.99.10:5055"
 ---> Using cache
 ---> 19112840859e
Step 6/8 : COPY . /opt
 ---> aa8c35d1e876
Step 7/8 : EXPOSE 8080
 ---> Running in d2165aecbab7
Removing intermediate container d2165aecbab7
 ---> 2f7b781e34cd
Step 8/8 : ENTRYPOINT ["python3", "app.py"]
```

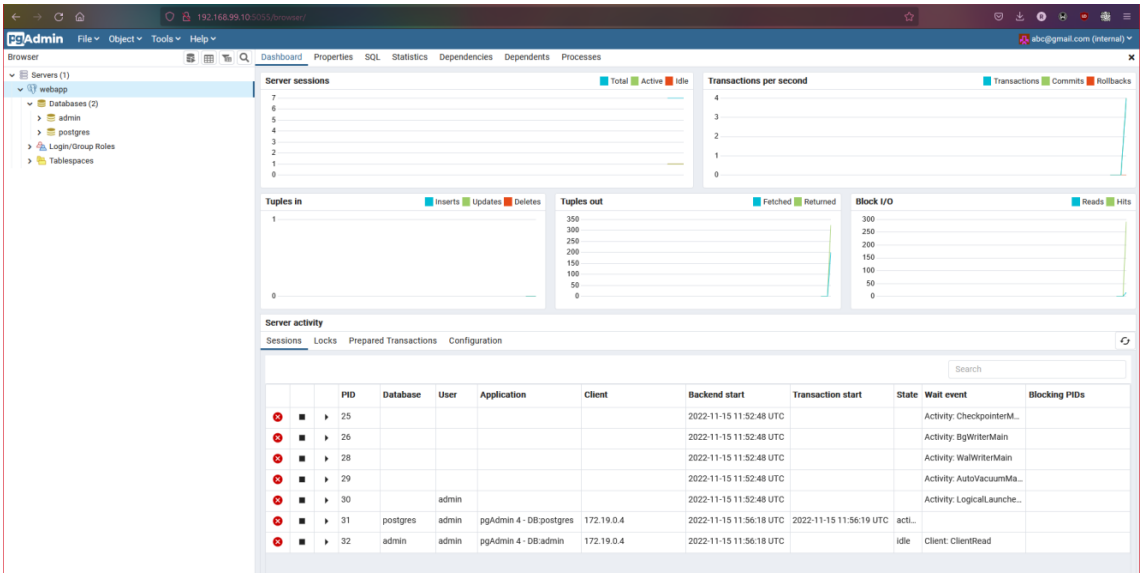
Nous pouvons donc nous rendre sur l'interface web:



Ensuite nous pouvons bien cliquer sur les liens présents : pgadmin: Le client peut se connecter avec les identifiants présent dans notre docker-compose:



Puis voici l'interface du client avec ses bases de données :



Liste des identifiants:

identifiant	password
abc@gmail.com	AZERY!
admin	QWERTY!

- Sources

Fichiers	Liens
Docker-compose	Exam_Docker/docker-compose.yml
Dockerfile	Exam_Docker/Dockerfile

Sites utiles :

```
https://hub.docker.com/r/dpage/pgadmin4/
https://hub.docker.com/_/odoo
https://openclassrooms.com/fr/courses/2035766-optimisez-votre-deploiement-en-creant-des-conteneurs-avec-docker/6211517-creez-votre-premier-dockerfile
https://openclassrooms.com/fr/courses/2035766-optimisez-votre-deploiement-en-creant-des-conteneurs-avec-docker/6211677-creez-un-fichier-docker-compose-pour-orchestrer-vos-conteneurs
https://devopssec.fr/article/deployer-manipuler-securiser-un-serveur-registry-docker-prive
```

- Licence

ParisYnovCampus

```
Les participants au projet:
    - Raphaël CARRILHO
```

```
    - Noah SUHARD
    - Ilyes BESBES
    - Keenan SZCZEPKOWSKI
```