Homework 3 - Building Images

Exercise 1: Build and Containerize an API (Back-End)

Develop a simple API using a programming language of your choice (e.g., Node.js, Python, Go).

- The API must expose an endpoint (e.g., /info) that returns:
- The container's hostname
- The container's IP address
- Write a Dockerfile using multistage build to containerize the API.
- Build the image and run the container.
- Test the endpoint with curl to verify that it returns the correct information.
- Ensure the API is not exposed to the host.

1. *Dockerfile*

Dockerfile

```dockerfile
# --- Etapa de construcción ---
FROM python:3.13-slim as builder

WORKDIR /app
COPY requirements.txt .
RUN pip install --user --no-cache-dir -r requirements.txt

# --- Etapa final ---
FROM python:3.13-slim
WORKDIR /app

# Copia dependencias instaladas
COPY --from=builder /root/.local /root/.local
COPY . .

# Asegura que las dependencias estén en el PATH
ENV PATH=/root/.local/bin:$PATH
ENV PYTHONUNBUFFERED=1

EXPOSE 8000

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

2. *Construcción*

```
❯❯ backend
⊙ docker build -t fastapi-backend .
[+] Building 1.8s (12/12) FINISHED                                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                0.0s
 => => transferring dockerfile: 633B                                                                0.0s
 => WARN: FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 2)                      0.0s
 => [internal] load metadata for docker.io/library/python:3.13-slim                                 1.4s
 => [auth] library/python:pull token for registry-1.docker.io                                       0.0s
 => [internal] load .dockerignore                                                                   0.0s
 => => transferring context: 206B                                                                   0.0s
 => [internal] load build context                                                                   0.0s
 => => transferring context: 1.56kB                                                                 0.0s
 => [builder 1/4] FROM docker.io/library/python:3.13-slim@sha256:914bf5c12ea40a97a78b2bff97fbdb766cc36ec903bf  0.0s
 => => resolve docker.io/library/python:3.13-slim@sha256:914bf5c12ea40a97a78b2bff97fbdb766cc36ec903bfb4358faf  0.0s
 => CACHED [builder 2/4] WORKDIR /app                                                               0.0s
 => CACHED [builder 3/4] COPY requirements.txt .                                                    0.0s
 => CACHED [builder 4/4] RUN pip install --user --no-cache-dir -r requirements.txt                  0.0s
 => CACHED [stage-1 3/4] COPY --from=builder /root/.local /root/.local                              0.0s
 => CACHED [stage-1 4/4] COPY . .                                                                   0.0s
 => exporting to image                                                                              0.1s
 => => exporting layers                                                                             0.0s
 => => exporting manifest sha256:0fd51d5a8990dafc7f0fa607898b0e84a8de5be9f43918ac15c24a03150567a5    0.0s
 => => exporting config sha256:96a5970ec2010acae00b0718fc5a8d67ba48f5928681c1b17d5f67bf00dd21e8      0.0s
 => => exporting attestation manifest sha256:fe6f7fe08ce80b56614a6f695beada1f0c6918cd3032cd2b675dcc72565e2c65  0.0s
```

3. *Levantamos.*

En este caso, yo realice una aplicación de Tareas con el crud básico, y una conexión a MongoDB (Atlas), asi que tengo un archivo env, pero también tengo el endpoint de /info, pero hare el run mencionando el .env, porque sino fallara.

```
docker run -d --name api-container --env-file .env fastapi-backend
f66304871056797c77587c546b21cf8847bebb15ee81ccb629423657096c62cf
 backend
docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED         STATUS         PORTS      NAMES
f66304871056   fastapi-backend   "uvicorn main:app --…"   8 seconds ago   Up 8 seconds   8000/tcp   api-container
 backend
```

Como se puede ver, ya esta creado, y esta levantado sin ningún error, y el puerto esta visible solo dentro del contenedor.

4. *Prueba con el /info*

```
 backend
docker exec api-container curl -s http://localhost:8000/info
{"hostname":"f66304871056","ip":"172.17.0.2","message":"¡Endpoint de información del contenedor!"}
```

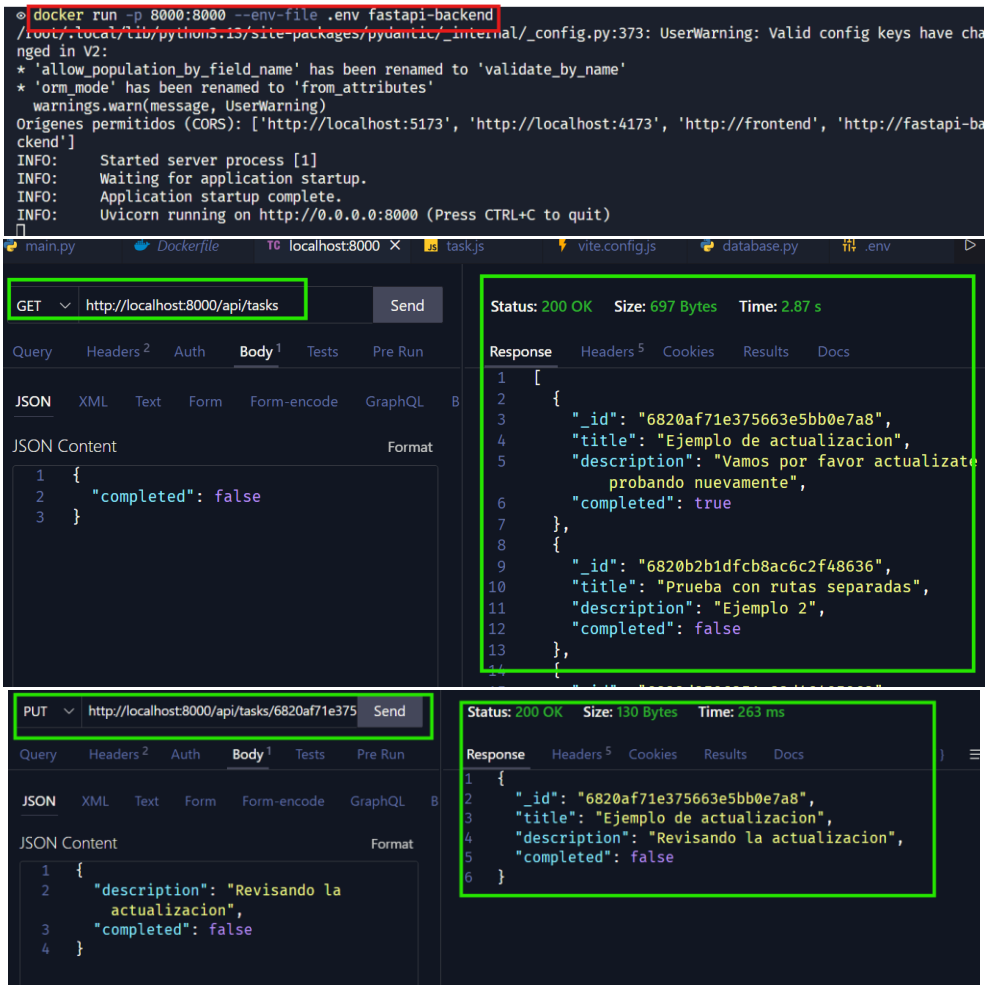
Si hacemos con el exec, para hacer un curl al endpoint info, podemos ver que funciona, y nos devuelve la información. Por otro lado, al no tener el puero expuesto, si hacemos, lo siguiente:

```
 backend
curl -v http://localhost:8000/info
* Host localhost:8000 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:8000...
*   Trying 127.0.0.1:8000...
* connect to ::1 port 8000 from :: port 61921 failed: Connection refused
* connect to 127.0.0.1 port 8000 from 0.0.0.0 port 61922 failed: Connection refused
* Failed to connect to localhost port 8000 after 2251 ms: Could not connect to server
* closing connection #0
curl: (7) Failed to connect to localhost port 8000 after 2251 ms: Could not connect to server
```

No da un error.

5. *Prueba extra*

Como mencione hice una aplicación de Tareas, así que tengo todo el crud, para eso, tendré que exponer el puerto para poder verlo desde el navegador u postman, pero es para demostrar que si funciona.

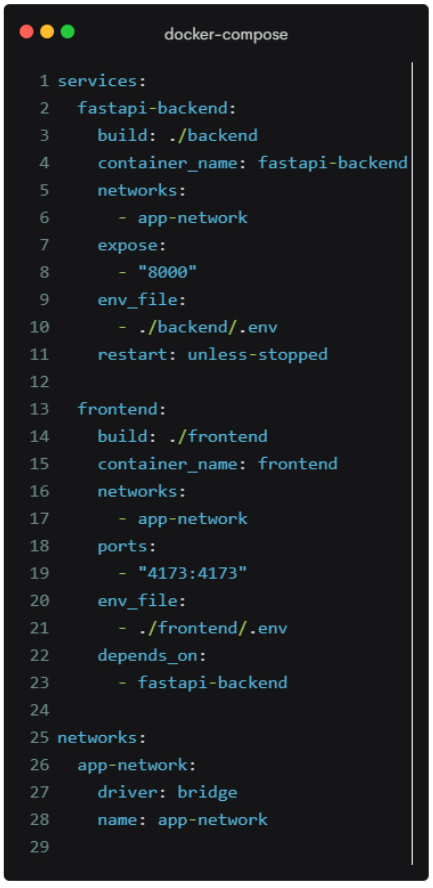


Exercise 2: Build and Containerize a Front-End Application

- Create a front-end application using HTML/JavaScript or a framework of your choice.
- The app must fetch the /info endpoint from the backend API and display the hostname and
- IP address.
- Write a Dockerfile using multistage build to containerize and minimize the final image.
- Create a user-defined Docker network and run both frontend and backend containers within
- it.
- Verify in the browser that the frontend correctly shows the container metadata served by the
- Back End.

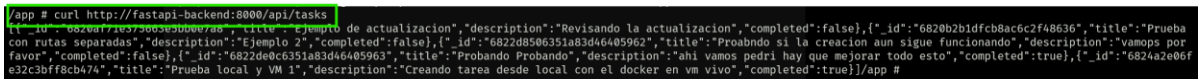
1. *Dockerfile*

2. *Dockercompose*



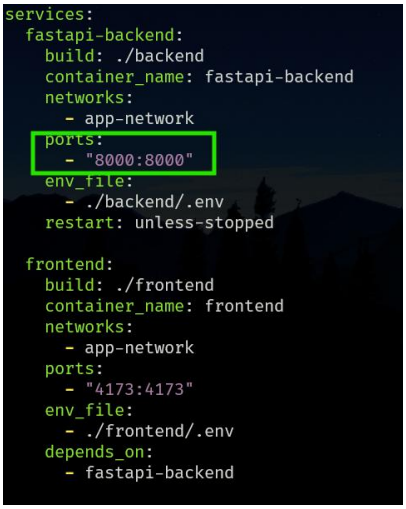
3. *Curls al backend desde el contendor del frontend /info y /api/tasks*

```
/app # curl http://fastapi-backend:8000/info
{"hostname":"d842d9b7a5eb","ip":"172.18.0.2","message":"¡Endpoint de información del contenedor!"}
/app # curl http://fastapi-backend:8000/api/tasks
```

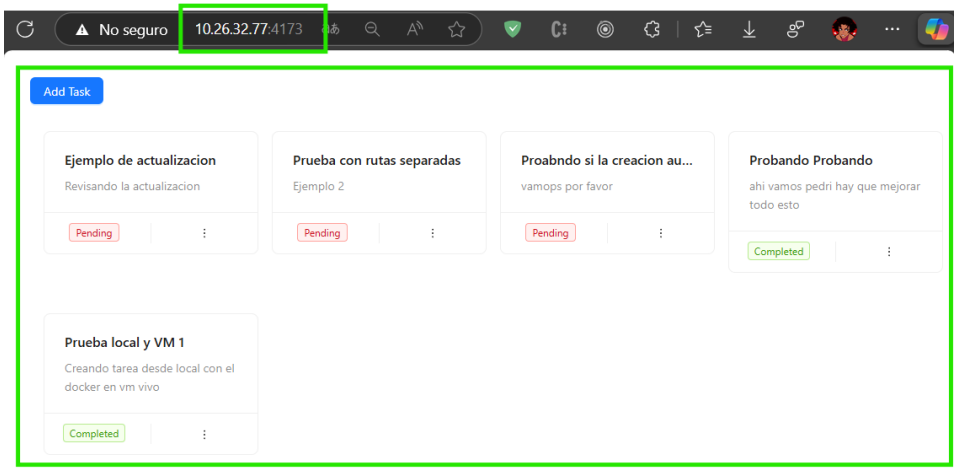


4. *Inspect del network*

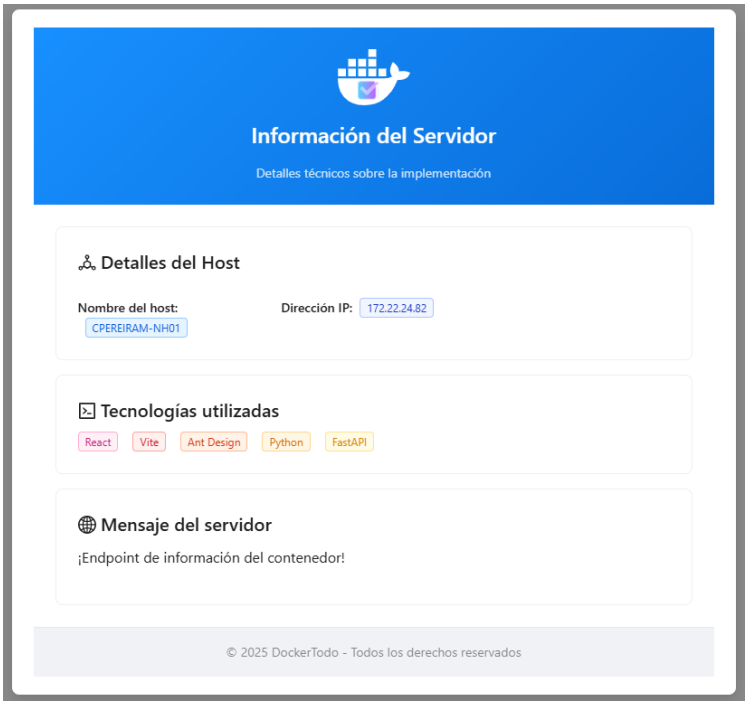
5. *Para la prueba en navegador, tuve que exponer el puerto, porque si no me salía que no encontraba la ruta fastapi:800, tengo un video explicando esto.*



6. *Imagen con la aplicación funcionando*
   Localhost:8000/api/tasks



Localhost:8000/info



Exercise 3: The .dockerignore File

- Create a .dockerignore file in both Back End and Front End repos to exclude all
- unnecessary files and directories when building

1. *Dockerignore para ambos proyectos*
   - Para el proyecto de bakend lo desarrolle en python con fastapy, me cree un ambiente viryual para no tener problemas de compatibilidad, por eso tengo el venv.
   - Para el rpoeycto de frontend lo hice en Vite, con reactRouter, y andt.

ckerignore-frontend

```
node_modules
dist
.DS_Store
.env.local
.env.development
.env.test
.git
.gitignore
.editorconfig
.eslint*
.vscode
.idea
*.log
```

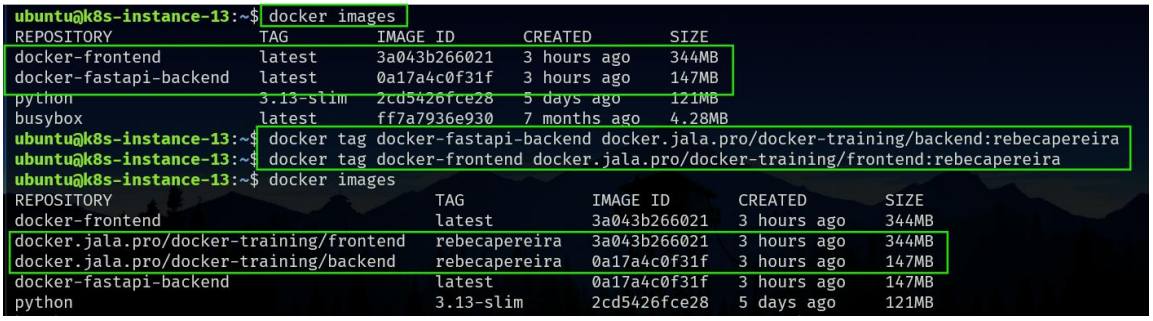
ckerignore-backend

```
.git
__pycache__/
*.pyc
*.pyo
*.pyd
.Python
env/
venv/
.env
*.sqlite3
.DS_Store
.idea/
.vscode/
*.log
*.swp
*.swo
__pycache__/
/routes/__pycache__/
```

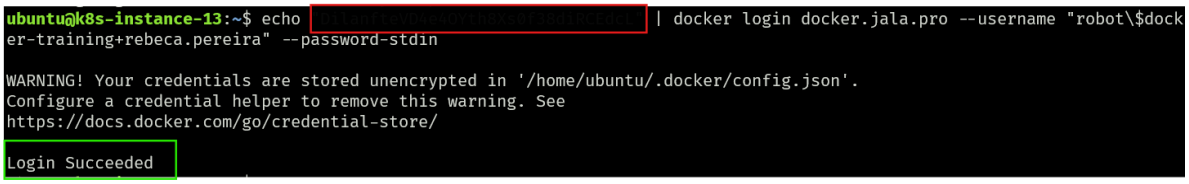
Exercise 4: Private Registry

- Push the previously built Back End and Front End images to the private registry at
- docker.jala.pro .
- Tag your images: docker.jala.pro/docker-training/[CONTAINER-NAME=BackEnd ||
- FrontEnd]:[TAG=FullName]
- For Instance: docker.jala.pro/docker-training/backend:calebespinoza

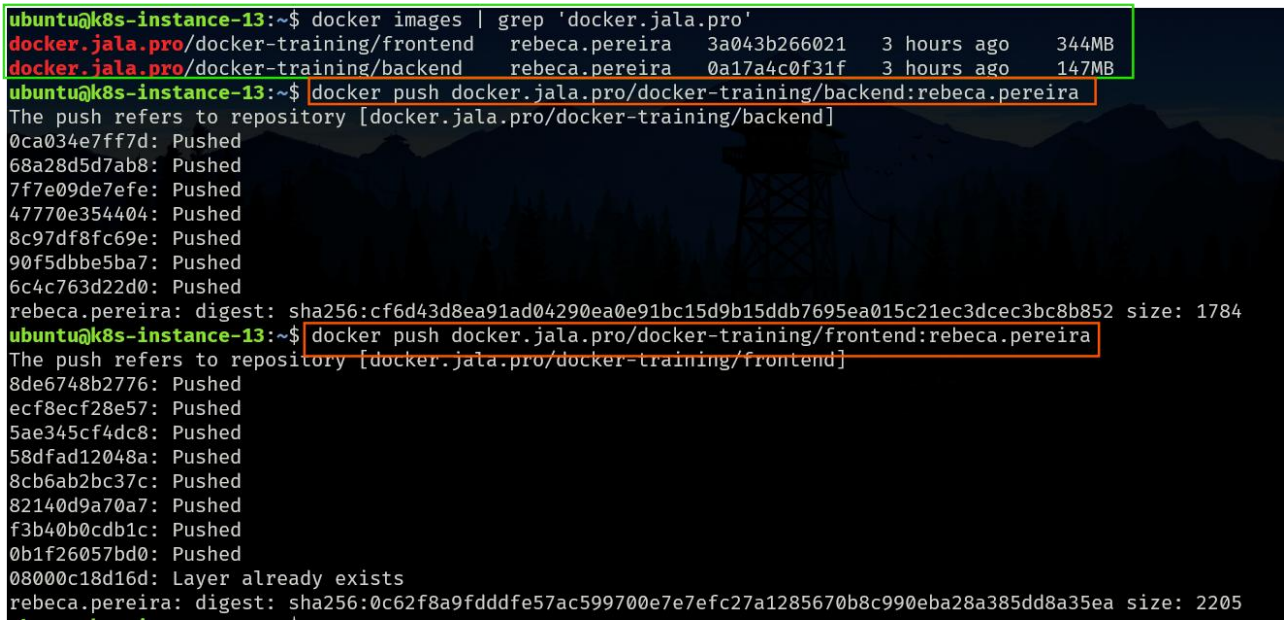
1. *Revisamos nuestras imágenes, para ver a las que les daremos un tag y les asignamos uno*



2. *Hacemos Login al Registry*



3. *Revisamos de nuevo las imágenes y hacemos push de ellas*



4. *Revisamos que se hayan subido ambas*