

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

1 # --- Etapa de construcción ---
2 FROM python:3.13-slim as builder
3
4 WORKDIR /app
5 COPY requirements.txt .
6 RUN pip install --user --no-cache-dir -r requirements.txt
7
8 # --- Etapa final ---
9 FROM python:3.13-slim
10 WORKDIR /app
11
12 # Copia dependencias instaladas
13 COPY --from=builder /root/.local /root/.local
14 COPY . .
15
16 # Asegura que las dependencias estén en el PATH
17 ENV PATH=/root/.local/bin:$PATH
18 ENV PYTHONUNBUFFERED=1
19
20 EXPOSE 8000
21
22 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@sha256:914bf5c12ea40a97a78b2bff97fbd766cc36ec903bf 0.0s
=> => resolve docker.io/library/python:3.13-slim@sha256:914bf5c12ea40a97a78b2bff97fbd766cc36ec903bf4358faf 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), así 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 puerto 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.

The screenshot shows a Docker terminal at the top and two Postman requests below it. The Docker terminal shows the command `docker run -p 8000:8000 --env-file .env fastapi-backend` and the output of the application startup, including a warning about config keys and CORS settings. The first Postman request is a GET to `http://localhost:8000/api/tasks` with a status of 200 OK, returning a list of two tasks. The second Postman request is a PUT to `http://localhost:8000/api/tasks/6820af71e375` with a status of 200 OK, returning the details of the updated task.

```

● docker run -p 8000:8000 --env-file .env fastapi-backend
/root/.local/lib/python3.13/site-packages/pydantic/_internal/_config.py:373: UserWarning: Valid config keys have changed in V2:
* 'allow_population_by_field_name' has been renamed to 'validate_by_name'
* 'orm_mode' has been renamed to 'from_attributes'
warnings.warn(message, UserWarning)
Origenes permitidos (CORS): ['http://localhost:5173', 'http://localhost:4173', 'http://frontend', 'http://fastapi-backend']
INFO: Started server process [1]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)

```

Postman Request 1: GET `http://localhost:8000/api/tasks`
Status: 200 OK, Size: 697 Bytes, Time: 2.87 s
Response: `[{"_id": "6820af71e375663e5bb0e7a8", "title": "Ejemplo de actualizacion", "description": "Vamos por favor actualizar probando nuevamente", "completed": true}, {"_id": "6820b2b1dfcb8ac6c2f48636", "title": "Prueba con rutas separadas", "description": "Ejemplo 2", "completed": false}]`

Postman Request 2: PUT `http://localhost:8000/api/tasks/6820af71e375`
Status: 200 OK, Size: 130 Bytes, Time: 263 ms
Response: `{"_id": "6820af71e375663e5bb0e7a8", "title": "Ejemplo de actualizacion", "description": "Revisando la actualizacion", "completed": false}`

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

```
Dockerfile
1 # --- Etapa de construcción ---
2 FROM node:18-alpine as builder
3
4 WORKDIR /app
5 COPY package.json package-lock.json ./
6 RUN npm ci
7 COPY . .
8 RUN npm run build
9
10 # --- Etapa de producción ---
11 FROM node:18-alpine
12 WORKDIR /app
13
14 # Copia solo los archivos necesarios
15 COPY --from=builder /app/dist ./dist
16 COPY --from=builder /app/node_modules ./node_modules
17 COPY package.json .
18
19 # Instala vite globalmente para el preview
20 RUN npm install vite -g
21
22 ENV HOST=0.0.0.0
23 EXPOSE 4173
24
25 CMD ["npm", "run", "preview", "--", "--host", "0.0.0.0"]
```

2. Dockercompose

```
docker-compose
1 services:
2   fastapi-backend:
3     build: ./backend
4     container_name: fastapi-backend
5     networks:
6       - app-network
7     expose:
8       - "8000"
9     env_file:
10      - ./backend/.env
11     restart: unless-stopped
12
13   frontend:
14     build: ./frontend
15     container_name: frontend
16     networks:
17       - app-network
18     ports:
19       - "4173:4173"
20     env_file:
21       - ./frontend/.env
22     depends_on:
23       - fastapi-backend
24
25 networks:
26   app-network:
27     driver: bridge
28     name: app-network
29
```

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

```
/app # curl http://fastapi-backend:8000/info
{"hostname":"d842d9b/a5eb","ip":"172.18.0.2","message":"¡Endpoint de información del contenedor!"},
/app # curl http://fastapi-backend:8000/api/tasks
{"id":"6822d8506351a83d46405962","title":"Ejemplo de actualización","description":"Revisando la actualización","completed":false}, {"id":"6820b2b1dfcb8ac6c2f48636","title":"Prueba con rutas separadas","description":"Ejemplo 2","completed":false}, {"id":"6822d8506351a83d46405962","title":"Proabndo si la creacion aun sigue funcionando","description":"vamos por favor","completed":false}, {"id":"6822de0c6351a83d46405963","title":"Probando Probando","description":"ahí vamos pedri hay que mejorar todo esto","completed":true}, {"id":"6824a2e0fe32c3bff8cb474","title":"Prueba local y VM 1","description":"Creando tarea desde local con el docker en vm vivo","completed":true}}/app #
```

4. Inspect del network

```
Containers: {
  "43923ae0b4a19594aa864c8853317500379fffcba3374b52d7a83a7c38d90324": {
    "Name": "frontend",
    "EndpointID": "9507ef5bfc4f98666f0b285858552daec66ce6fda247cb1e6338268957475e88",
    "IPAddress": "172.18.0.3/16",
    "IPv6Address": ""
  },
  "d842d9b7a5eb26cd98433f7d0da1ddb061b020171ea3538e17089fc98041ae67": {
    "Name": "fastapi-backend",
    "EndpointID": "b44bc3811768bc0367fe4e44feb8dfee9fe8aef95dd226feb33687fa4d796b39",
    "IPAddress": "172.18.0.2/16",
    "IPv6Address": ""
  }
}
```

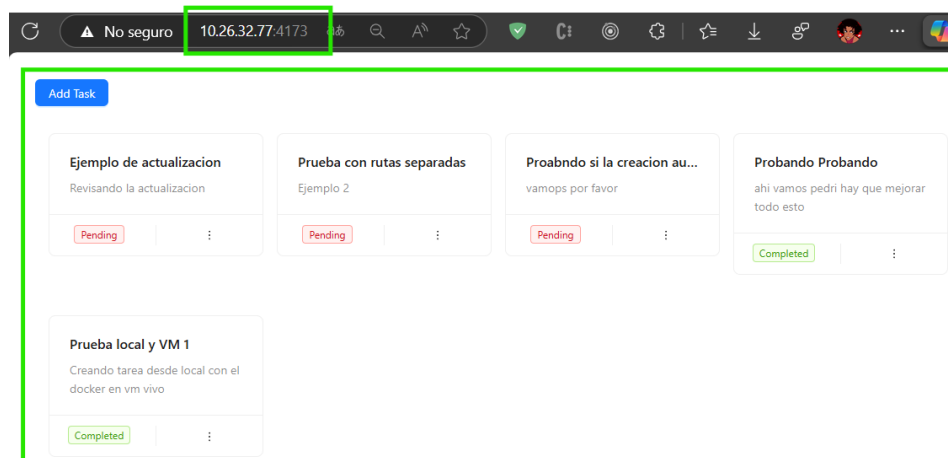
5. Para la prueba en navegador, tuve que exponer el puerto, porque si no me salía que no encontraba la ruta fastapi:800, tengo un video explicando esto.

```
services:
  fastapi-backend:
    build: ./backend
    container_name: fastapi-backend
    networks:
      - app-network
    ports:
      - "8000:8000"
    env_file:
      - ./backend/.env
    restart: unless-stopped

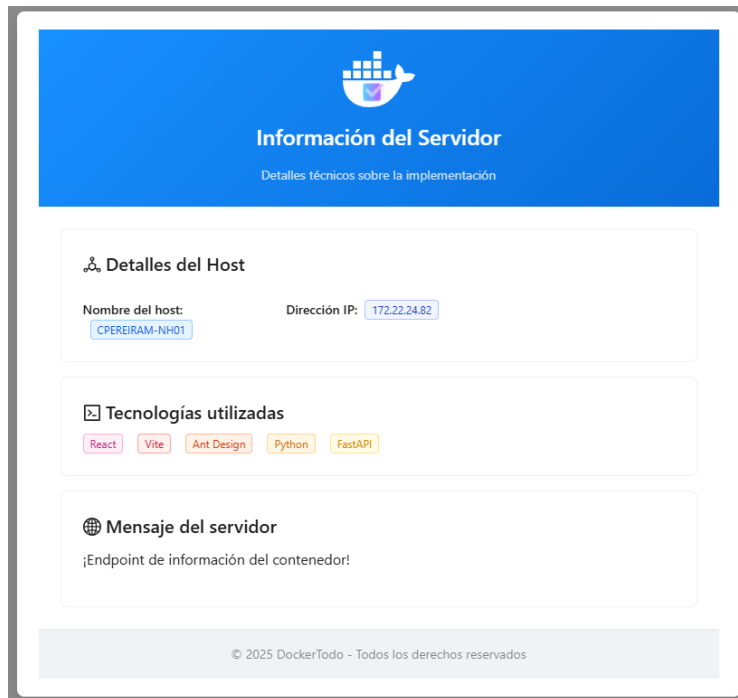
  frontend:
    build: ./frontend
    container_name: frontend
    networks:
      - app-network
    ports:
      - "4173:4173"
    env_file:
      - ./frontend/.env
    depends_on:
      - fastapi-backend
```

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 backend 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

1 node_modules
2 dist
3 .DS_Store
4 .env.local
5 .env.development
6 .env.test
7 .git
8 .gitignore
9 .editorconfig
10 .eslint*
11 .vscode
12 .idea
13 *.log
```

```
ckerignore-backend

1 .git
2 __pycache__/_
3 *.pyc
4 *.pyo
5 *.pyd
6 .Python
7 env/_
8 venv/_
9 .env
10 *.sqlite3
11 .DS_Store
12 .idea/_
13 .vscode/_
14 *.log
15 *.swp
16 *.swo
17 __pycache__/_
18 /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

```
ubuntu@k8s-instance-13:~$ docker images
REPOSITORY          TAG          IMAGE ID       CREATED        SIZE
docker-frontend     latest      3a043b266021   3 hours ago    344MB
docker-fastapi-backend latest      0a17a4c0f31f   3 hours ago    147MB
python              3.13-slim   2cd5426fce28   5 days ago     121MB
busybox             latest      ff7a7936e930   7 months ago   4.28MB
ubuntu@k8s-instance-13:~$ docker tag docker-fastapi-backend docker.jala.pro/docker-training/backend:rebecapereira
ubuntu@k8s-instance-13:~$ docker tag docker-frontend docker.jala.pro/docker-training/frontend:rebecapereira
ubuntu@k8s-instance-13:~$ docker images
REPOSITORY          TAG          IMAGE ID       CREATED        SIZE
docker-frontend     latest      3a043b266021   3 hours ago    344MB
docker.jala.pro/docker-training/frontend rebecapereira 3a043b266021   3 hours ago    344MB
docker.jala.pro/docker-training/backend rebecapereira 0a17a4c0f31f   3 hours ago    147MB
docker-fastapi-backend latest      0a17a4c0f31f   3 hours ago    147MB
python              3.13-slim   2cd5426fce28   5 days ago     121MB
```

2. Hacemos Login al Registry

```
ubuntu@k8s-instance-13:~$ echo [REDACTED] | docker login docker.jala.pro --username "robot\docker-training+rebeca.pereira" --password-stdin
WARNING! Your credentials are stored unencrypted in '/home/ubuntu/.docker/config.json'.
Configure a credential helper to remove this warning. See
https://docs.docker.com/go/credential-store/

Login Succeeded
```

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

```
ubuntu@k8s-instance-13:~$ docker images | grep 'docker.jala.pro'
docker.jala.pro/docker-training/frontend rebeca.pereira 3a043b266021 3 hours ago 344MB
docker.jala.pro/docker-training/backend rebeca.pereira 0a17a4c0f31f 3 hours ago 147MB
ubuntu@k8s-instance-13:~$ docker push docker.jala.pro/docker-training/backend:rebeca.pereira
The push refers to repository [docker.jala.pro/docker-training/backend]
0ca034e7ff7d: Pushed
68a28d5d7ab8: Pushed
7f7e09de7efe: Pushed
47770e354404: Pushed
8c97df8fc69e: Pushed
90f5dbbe5ba7: Pushed
6c4c763d22d0: Pushed
rebeca.pereira: digest: sha256:cf6d43d8ea91ad04290ea0e91bc15d9b15ddb7695ea015c21ec3dcec3bc8b852 size: 1784
ubuntu@k8s-instance-13:~$ docker push docker.jala.pro/docker-training/frontend:rebeca.pereira
The push refers to repository [docker.jala.pro/docker-training/frontend]
8de6748b2776: Pushed
ecf8ecf28e57: Pushed
5ae345cf4dc8: Pushed
58dfad12048a: Pushed
8cb6ab2bc37c: Pushed
82140d9a70a7: Pushed
f3b40b0cdb1c: Pushed
0b1f26057bd0: Pushed
08000c18d16d: Layer already exists
rebeca.pereira: digest: sha256:0c62f8a9fdddf57ac599700e7e7efc27a1285670b8c990eba28a385dd8a35ea size: 2205
```

4. Revisamos que se hayan subido ambas

```
ubuntu@k8s-instance-13:~$ curl -u "robot\docker-training+rebeca.pereira:[REDACTED]" \
https://docker.jala.pro/v2/docker-training/backend/tags/list
{"name": "docker-training/backend", "tags": ["calebespinoza", "gabriela", "rebeca.pereira", "rolandovillca", "watsonluis"]}
ubuntu@k8s-instance-13:~$ curl -u "robot\docker-training+rebeca.pereira:[REDACTED]" https://docker.jala.pro/v2/docker-training/frontend/tags/list
{"name": "docker-training/frontend", "tags": ["calebespinoza", "gabriela", "rebeca.pereira", "rolandovillca", "watsonluis"]}
ubuntu@k8s-instance-13:~$
```