DOCKER

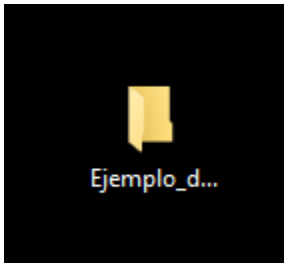
Emanuel Alejandro Gutierrez Romero

30/10/23

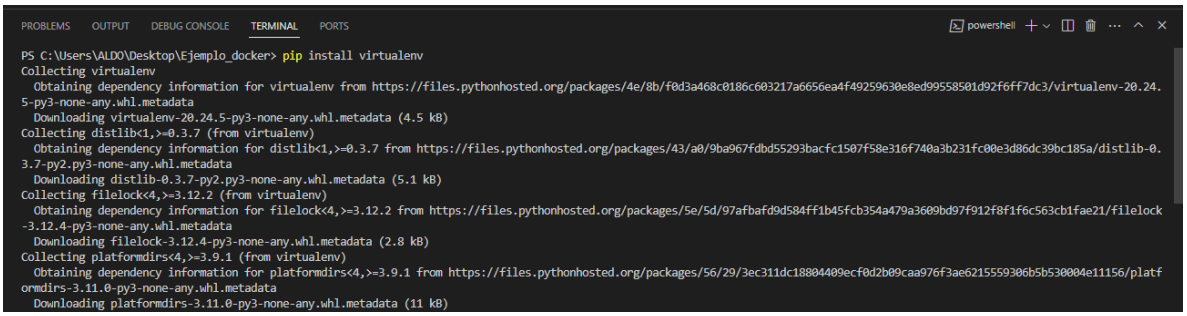
COMPUTACION TOLERANTE A FALLAS

Prof. Michel Emanuel López Franco

Lo primero que vamos a realizar es crear un proyecto dentro de nuestro entorno de desarrollo, en este caso será visual studio code, el nombre de nuestro proyecto puede ser el que nosotros queramos.

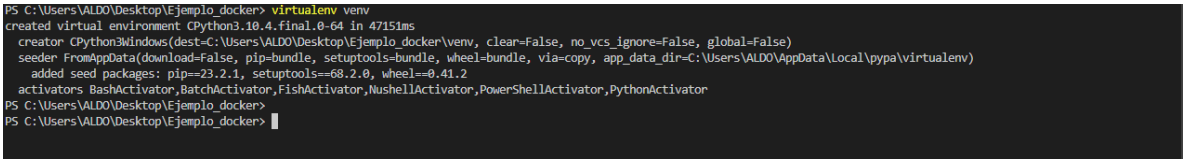


Lo nombramos “Ejemplo_docker” y lo llevamos a nuestro entorno.

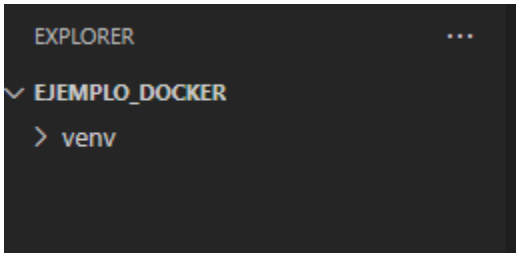


Dentro de nuestro proyecto crearemos una nueva terminal y descargaremos virtualenv para poder trabajar.

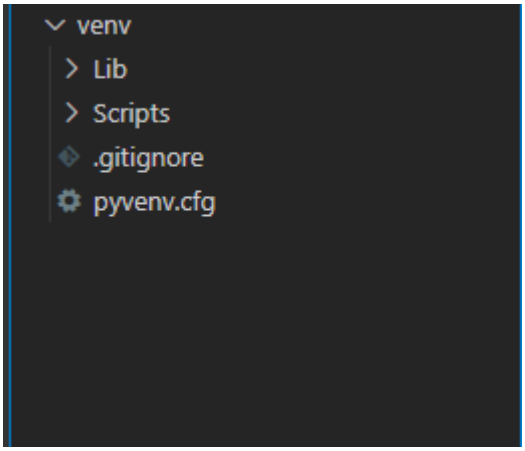
Ahora creamos una nueva carpeta que es en donde vamos a estar trabajando, usando el comando “virtualenv venv”.



Dentro de la consola se muestra que ya etá listo y dentro de nuestro proyecto también.



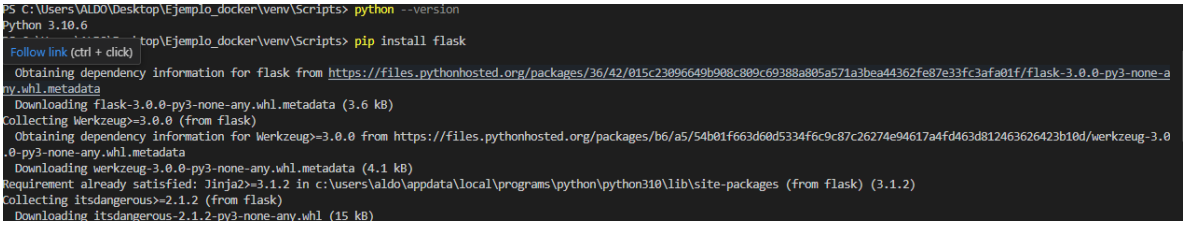
Vemos que tenemos varios componentes dentro de nuestra carpeta que recién creamos.



Ahora, para trabajar en código necesitaremos crear una nueva carpeta dentro de nuestro proyecto con el nombre que queramos (en este caso lo nombré ‘src’) y ahí crearemos un nuevo archivo .py de nombre app.

Ahora para activar nuestro entorno y poder trabajar necesitaremos desde la consola de nuestro programa ingresar a las carpetas de nuestro proyecto y ejecutar el actívate.bat que es el que va a activar el entorno.

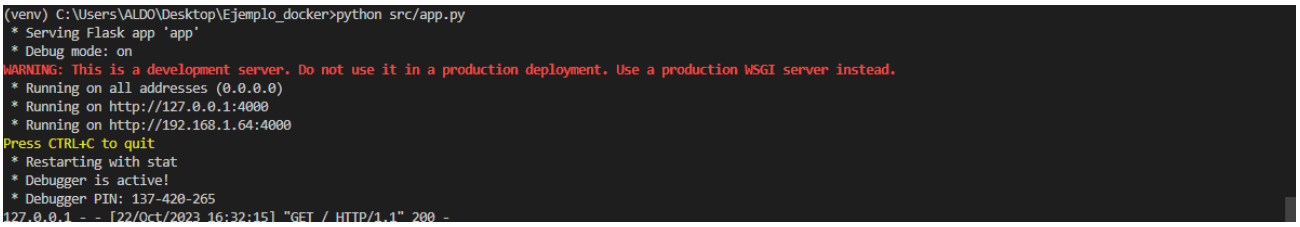
Introduciendo los siguientes pasos veamos que pasará. Descargaremos nuevos paquetes.



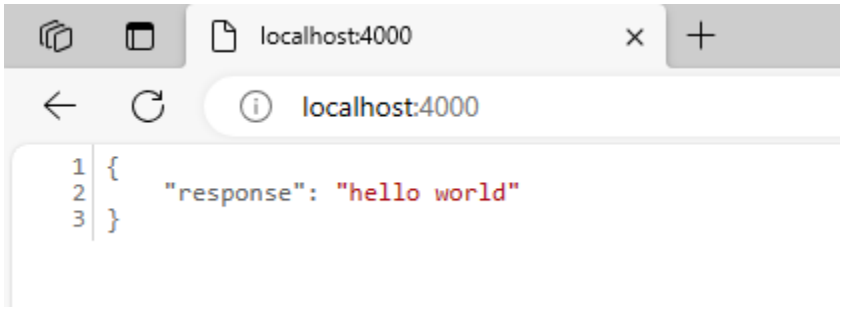
Ahora para verificar que todo está funcionando correctamente creamos una aplicación sencilla para ver si se montó de manera correcta el servidor.

```python
from flask import Flask, jsonify

#Aplicación del servidor
app = Flask(__name__)

@app.route('/', methods=['GET'])
def ping():
    return jsonify({"response": "hello world"})

if __name__ == '__main__':
    #Establecemos los parametros para que se pueda acceder desde donde sea y
el puerto en el que vamos a usar el contenedor.
    app.run(host="0.0.0.0", port=4000, debug = True)
```

Ahora para ver que funcione necesitamos ejecutar nuestro programa desde nuestra terminal usando el comando pyhton src/app.py y nos mostrará en teoría los puertos que está utilizando y si se estableció de manera correcta.



Como vemos se ejecuta de manera correcta, ahora en nuestro navegador web usaremos o bien la dirección que nos indica en la consola o tecleamos localhost:4000 y veamos que ocurre.



Como vemos ahí está el bloque de instrucciones que le indicamos, ahora bien sigamos con el ejemplo.

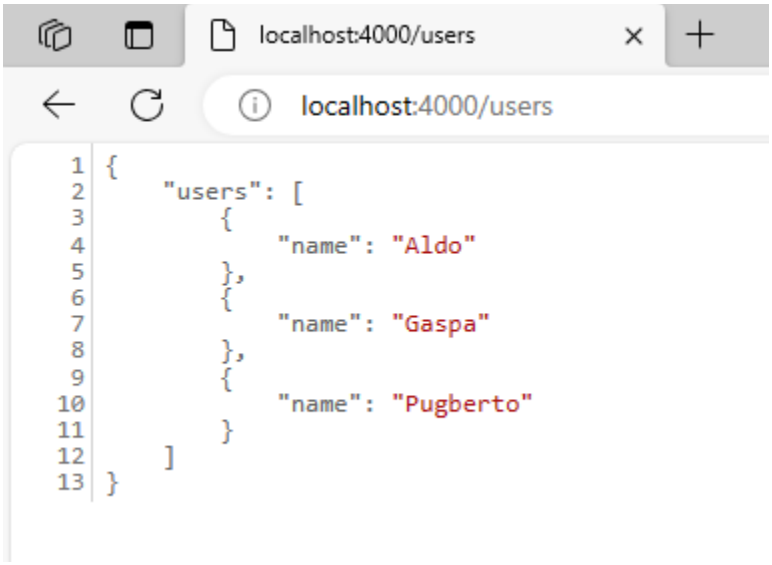
Creamos una nueva ruta para ver un nuevo ejemplo, creamos un nuevo archivo de nombre users donde a una variable le asignamos el nombre de user y será una lista de nombres.

```
users = [
    {"name" : "Aldo"},
    {"name" : "Gaspa"},
    {"name" : "Pugberto"}
]
```

Ahora creamos la nueva ruta y veamos que pasa.

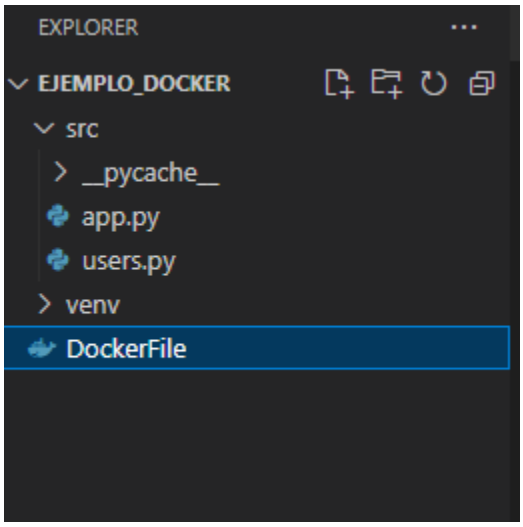
```
#Creamos una nueva ruta
@app.route('/users')
def usersHandlers():
    return jsonify({"users": users})
```

Ingresamos a la misma página, pero ahora usamos el localhost:4000/users.

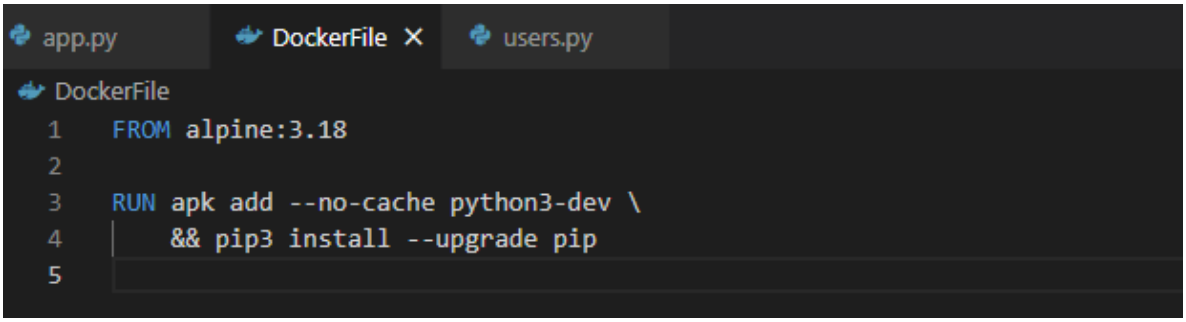


Como vemos el ejemplo sigue funcionando.

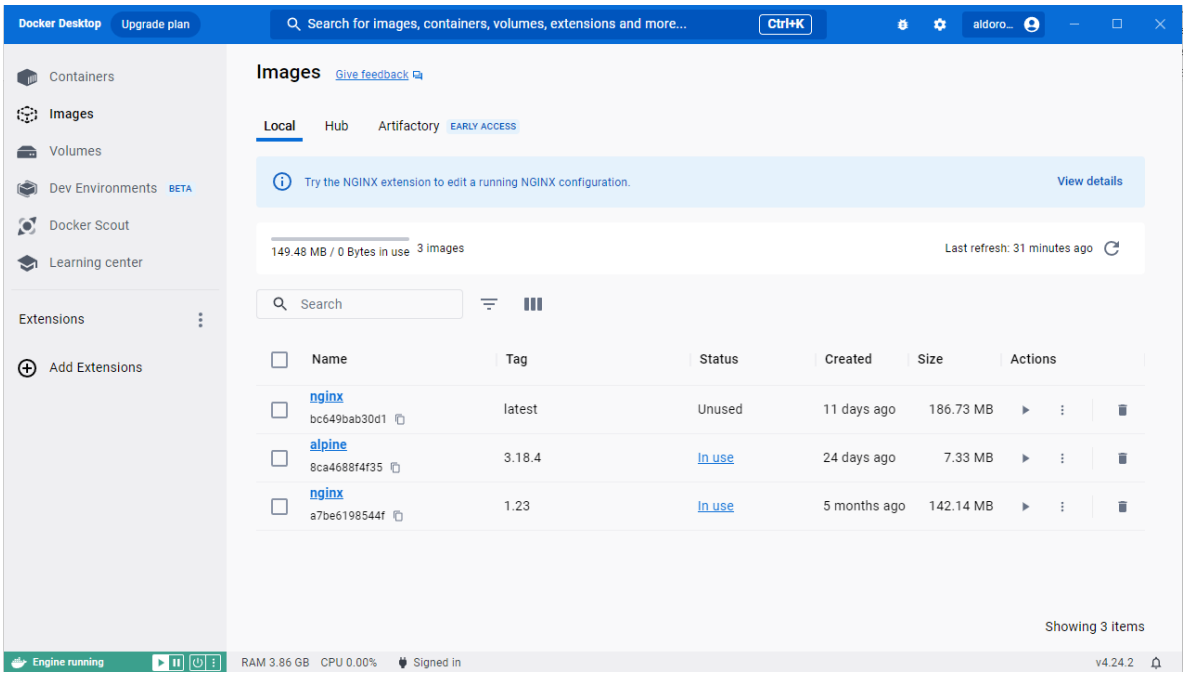
Ahora creamos un nuevo archivo de tipo dockerfile.



Creamos las especificaciones que queremos que tenga el docker, como la versión que vamos a utilizar y también la versión de pyhton que va a utilizar y todas las herramientas.



Antes tendremos que instalar Docker en nuestra computadora.



Una vez instalado ahora desde la terminal indicamos que cree una imagen usando nuestro archivo docker.

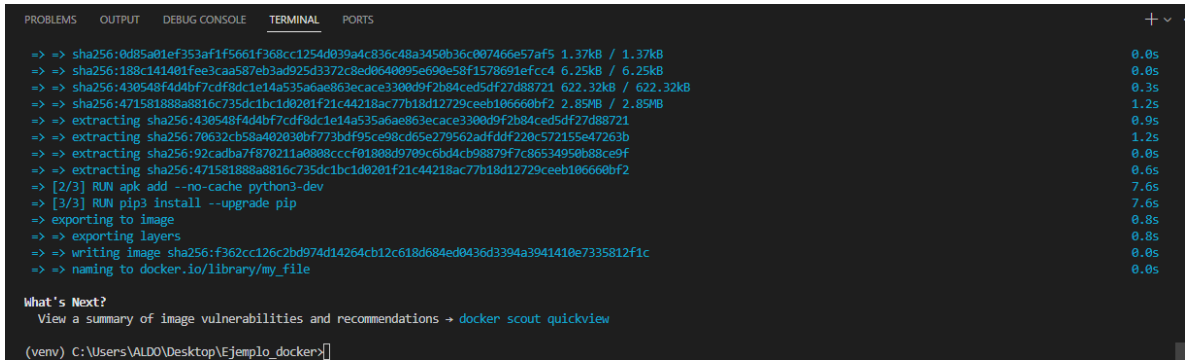
Ahora veamos si podemos crear la imagen.

Marca error. Veamos y cambiemos el código.

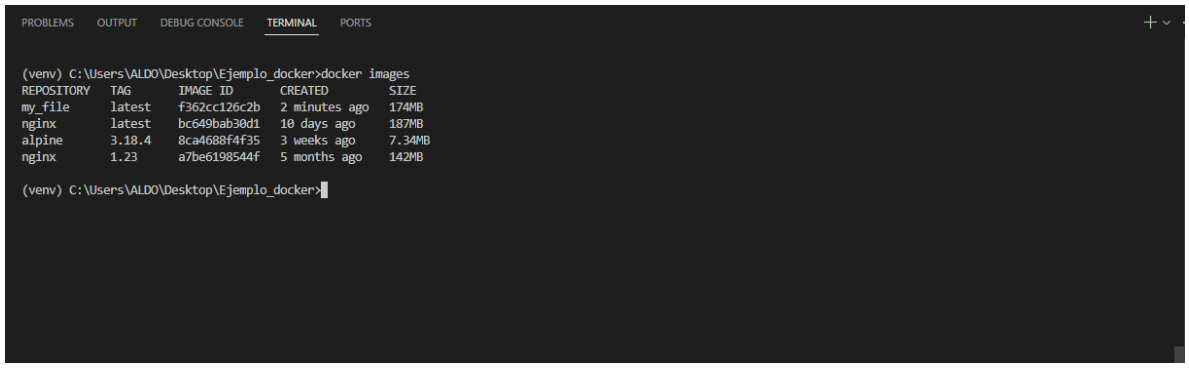
```
# Establecer la imagen base de Python Alpine
FROM python:3.9-alpine

# Instalar paquetes adicionales si es necesario
RUN apk add --no-cache python3-dev
```

```
# Actualizar pip
RUN pip3 install --upgrade pip
```

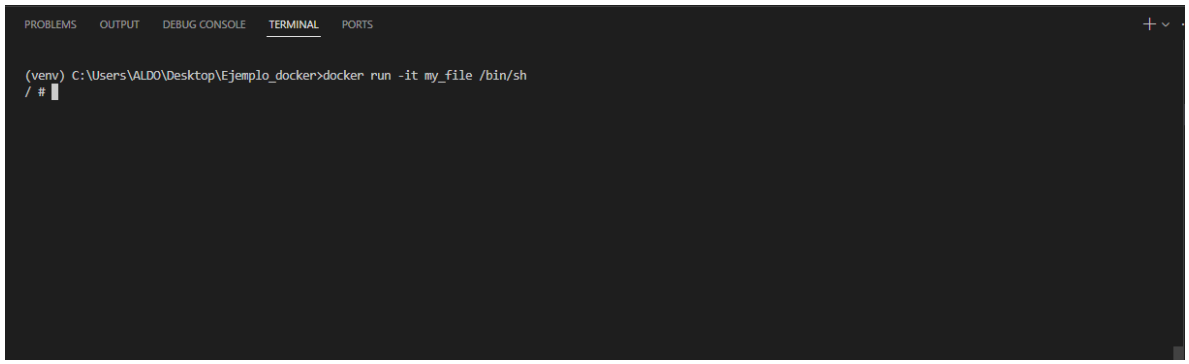
Una vez cambiado el código volvamos a ejecutar el mismo comando.



En este caso ya funcionó y en teoría ya creó la imagen, ejecutemos un comando para verificarlo.



Después de hacer cambios por 2 horas y casi acabar mi salud mental, ya se creó la imagen correctamente. Ahora quiero que le ejecute de manera iterativa por lo que pasamos el siguiente comando:



Ahora veamos que más hacemos interactuando con el sistema operativo.

Vemos las carpetas usando el comando ls

Veamos que versión de Python tenemos dentro de nuestro sistema.

Ahora veamos la versión de pip.

Como vemos ahora tenemos pip para Python versión 3.9, esto quiere decir que a cualquier vez que se instale este contenedor instalará estas versiones de Python en cualquier otra aplicación, en orden instala alpine y después Python. Ahora vamos a crear una carpeta y dentro mandaremos todos los archivos de nuestro trabajo.

Y para mostrar todo lo que queremos necesitaremos un .txt de los requisitos para que el programa funcione, veamos cuales son ejecutando un comando.

```
(venv) C:\Users\ALDO\Desktop\Ejemplo_docker>pip freeze
blinker==1.6.3
certifi==2023.7.22
charset-normalizer==3.3.1
click==8.1.7
colorama==0.4.6
distlib==0.3.7
filelock==3.12.4
flack==1.3.7
Flask==3.0.0
idna==3.4
itsdangerous==2.1.2
Jinja2==3.1.2
MarkupSafe==2.1.3
platformdirs==3.11.0
```

Aquí están algunos de los requisitos. Ahora hay que ponerlos dentro de un .txt

```
click==8.1.7
colorama==0.4.6
distlib==0.3.7
filelock==3.12.4
flack==1.3.7
Flask==3.0.0
idna==3.4
itsdangerous==2.1.2
Jinja2==3.1.2
MarkupSafe==2.1.3
platformdirs==3.11.0
requests==2.31.0
urllib3==2.0.7
virtualenv==20.24.5
Werkzeug==3.0.0

(venv) C:\Users\ALDO\Desktop\Ejemplo_docker>pip freeze > requirements.txt

(venv) C:\Users\ALDO\Desktop\Ejemplo_docker>
```

Ejecutamos este comando y nos crea nuestro .txt.

```dockerfile
# Establecer la imagen base de Python Alpine
FROM python:3.9-alpine

# Instalar paquetes adicionales si es necesario
RUN apk add --no-cache python3-dev

# Actualizar pip
RUN pip3 install --upgrade pip

WORKDIR /app

COPY . /app

RUN pip3 --no-cache-dir install -r requirements.txt

CMD ["python3","src/app.py"]
```

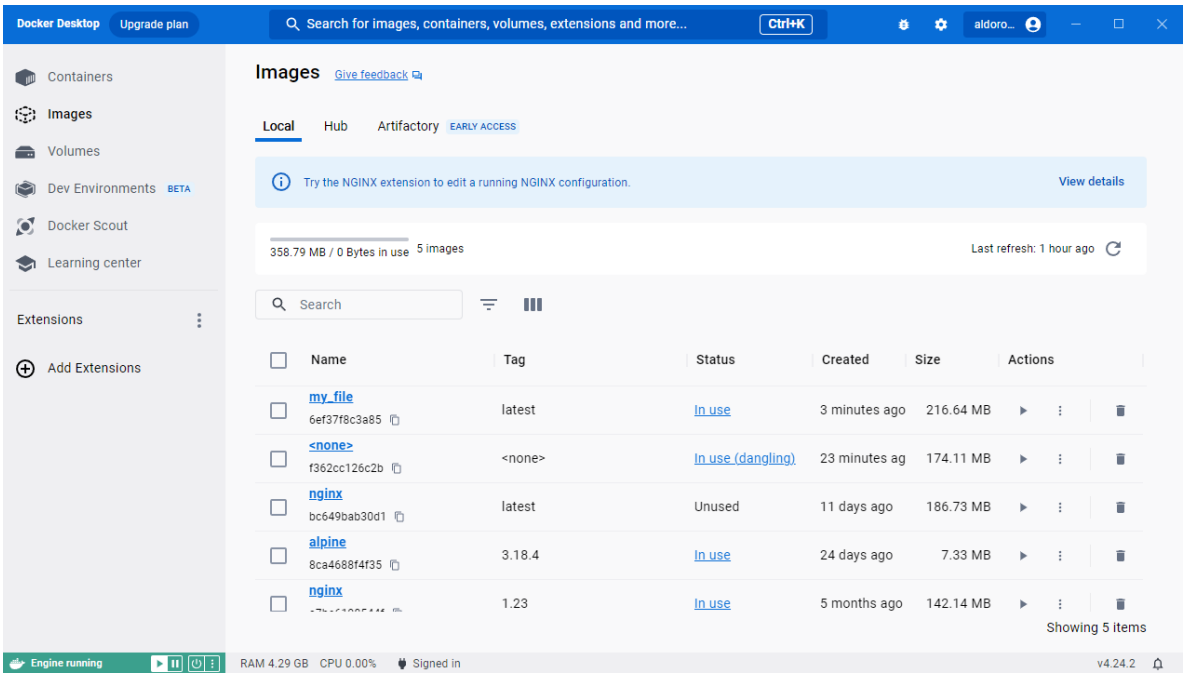
Volvemos a generar nuestra imagen ya con nuestros datos modificados.

```
 => [internal] load metadata for docker.io/library/python:3.9-alpine                                                1.8s
 => [auth] library/python:pull token for registry-1.docker.io                                                       0.0s
 => [1/6] FROM docker.io/library/python:3.9-alpine@sha256:37310032694208418392f8ea663ffe7aabd342c4ba950aad44c50926445a7cb9  0.0s
 => [internal] load build context                                                                                   1.8s
 => => transferring context: 29.02MB                                                                                1.7s
 => CACHED [2/6] RUN apk add --no-cache python3-dev                                                                 0.0s
 => CACHED [3/6] RUN pip3 install --upgrade pip                                                                     0.0s
 => [4/6] WORKDIR /app                                                                                              0.1s
 => [5/6] COPY . /app                                                                                               1.1s
 => [6/6] RUN pip3 --no-cache-dir install -r requirements.txt                                                      10.1s
 => exporting to image                                                                                              0.5s
 => => exporting layers                                                                                             0.5s
 => => writing image sha256:6ef37f8c3a8524ecd58af895521f479b30fa0ed54a3513467c058dca16809db8                        0.0s
 => => naming to docker.io/library/my_file                                                                          0.0s

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview

(venv) C:\Users\ALDO\Desktop\Ejemplo_docker>
```

Nuevamente se ejecuta de manera correcta.

```
(venv) C:\Users\ALDO\Desktop\Ejemplo_docker>docker run -it --publish 7000:4000  my_file
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:4000
 * Running on http://172.17.0.4:4000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 368-350-126
```

Ahora se está ejecutando desde el contenedor.



Aquí vemos la imagen.