Centro Universitario de Ciencias Exactas e Ingeniería

Computación Tolerante a Fallas

Ejemplo utilizando Docker.

MICHEL EMANUEL LOPEZ FRANCO

Juan Pablo Hernández Orozco

219294285

Introducción:

Docker se ha convertido en una herramienta fundamental para el desarrollo y despliegue de aplicaciones modernas, permitiendo empaquetar entornos completos de forma reproducible y aislada. En el presente ensayo, se analiza en detalle un ejemplo de implementación de Docker para una aplicación híbrida .NET y Python, orientada a cómputo tolerante a fallas. Se profundiza en cada etapa del Dockerfile, describiendo su propósito, los beneficios asociados y las buenas prácticas adoptadas. Además, se discuten aspectos de rendimiento, seguridad y mantenimiento relevantes para un escenario universitario de nivel avanzado.

Objetivo:

El objetivo de este ejemplo es demostrar cómo utilizar multistage builds en Docker para separar el proceso de compilación de la aplicación .NET de su entorno de ejecución (runtime), integrando asimismo un entorno Python virtualizado. Esto permite:

- Minimizar el tamaño de la imagen final.
- Asegurar la coherencia de dependencias.
- Facilitar el mantenimiento y la escalabilidad.
- Incorporar componentes heterogéneos (.NET y Python) de forma robusta.

Desarrollo:

Diseño y Arquitectura del Contenedor Docker

El *Dockerfile* está dividido en dos etapas bien definidas:

1. **Build stage**: utiliza la imagen oficial mcr.microsoft.com/dotnet/sdk:8.0 para restaurar y compilar el proyecto .NET.
2. **Runtime stage**: parte de mcr.microsoft.com/dotnet/aspnet:8.0, incorpora Python 3 en un entorno virtual, instala dependencias científicas y copia la aplicación compilada desde la etapa anterior .

Esta separación permite reducir sustancialmente la superficie de la imagen final, ya que no se incluyen herramientas de compilación innecesarias en producción.

Etapa de compilación con .NET SDK

```
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build-env

WORKDIR /app


# Copiar archivos del proyecto y restaurar dependencias
```

```
COPY *.csproj ./

RUN dotnet restore


# Copiar el resto de archivos y compilar la aplicación

COPY . ./

RUN dotnet publish -c Release -o out
```

1. **Imagen base**: dotnet/sdk:8.0 provee el SDK completo de .NET 8, necesario para restaurar y compilar.
2. **WORKDIR**: define /app como directorio de trabajo, un patrón que mejora la legibilidad y estructura del contenedor.
3. **COPY + dotnet restore**: al copiar sólo el archivo .csproj inicialmente, se aprovecha el sistema de *caching* de Docker, de modo que si no cambian las dependencias, esta etapa no se repite en reconstrucciones posteriores.
4. **dotnet publish**: compila en modo Release y coloca el resultado en la carpeta out, lista para producción.

Este enfoque reduce tiempos de construcción y tamaños intermedios de las capas.

Etapa final (Runtime) con .NET y Python

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS runtime-env

WORKDIR /app


# Instalar dependencias de sistema y Python

RUN apt-get update && apt-get install -y --no-install-recommends \
    libexpat1 python3 python3-pip python3-venv && \
    python3 -m venv /opt/venv && \
    /opt/venv/bin/pip install --no-cache-dir scipy scikit-fuzzy \
    requests networkx numpy opencv-python-headless Pillow packaging && \
    apt-get clean && rm -rf /var/lib/apt/lists/*


# Copiar la aplicación compilada

COPY --from=build-env /app/out .
```

```
# Copiar scripts de Python

COPY --from=build-env /app/visionartificial.py .

COPY --from=build-env /app/controlador_difuso.py .

COPY --from=build-env /app/PerceptronNotificaciones.py .


# Configurar entorno virtual en PATH

ENV PATH="/opt/venv/bin:$PATH"


# Definir punto de entrada

ENTRYPOINT ["dotnet", "AquaponiaApi.dll"]
```

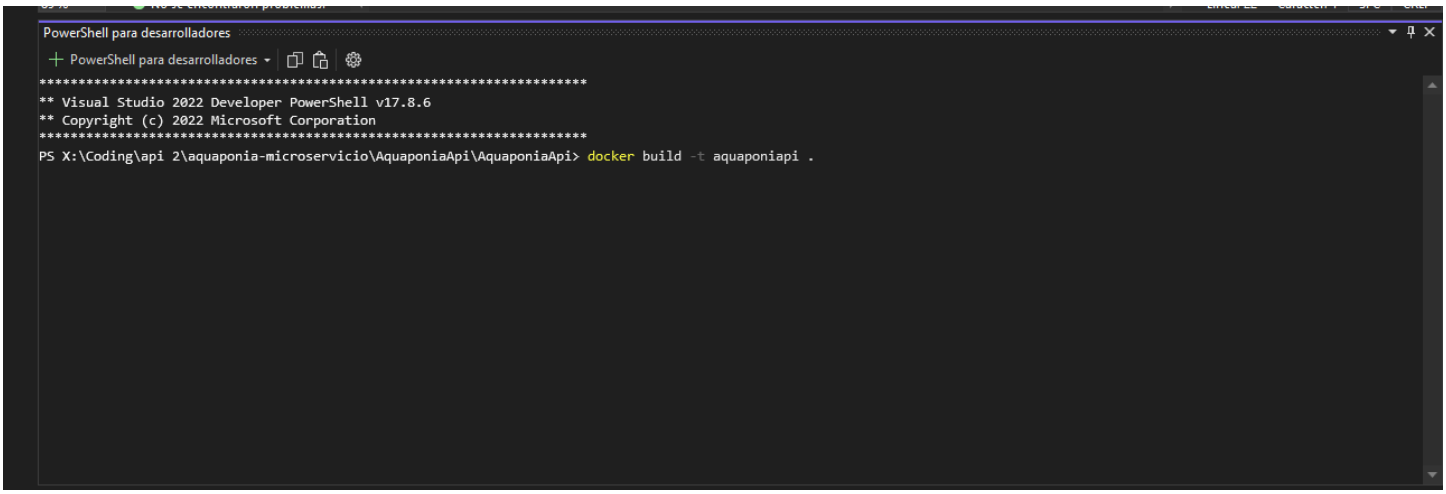
1. **Base runtime**: aspnet:8.0 contiene sólo el entorno necesario para ejecutar aplicaciones ASP.NET, reduciendo el tamaño.
2. **Dependencias de sistema**: libexpat1 (XML), python3-venv y otras utilidades.
3. **Entorno virtual Python**: crea /opt/venv, aislando las librerías científicas instaladas con pip. La opción --no-cache-dir evita archivos temporales, y al limpiar apt se reduce la imagen final.
4. **Copia de artefactos**: mediante COPY --from=build-env, se transfiere únicamente lo publicado, dejando fuera todo el SDK y archivos intermedios.
5. **Scripts Python**: se incluyen tres módulos (visionartificial.py, controlador_difuso.py, PerceptronNotificaciones.py) que aportan lógica de procesamiento de imagen y notificaciones basadas en inteligencia artificial .
6. **ENTRYPOINT**: fija el arranque de la aplicación .NET, garantizando consistencia en despliegues.

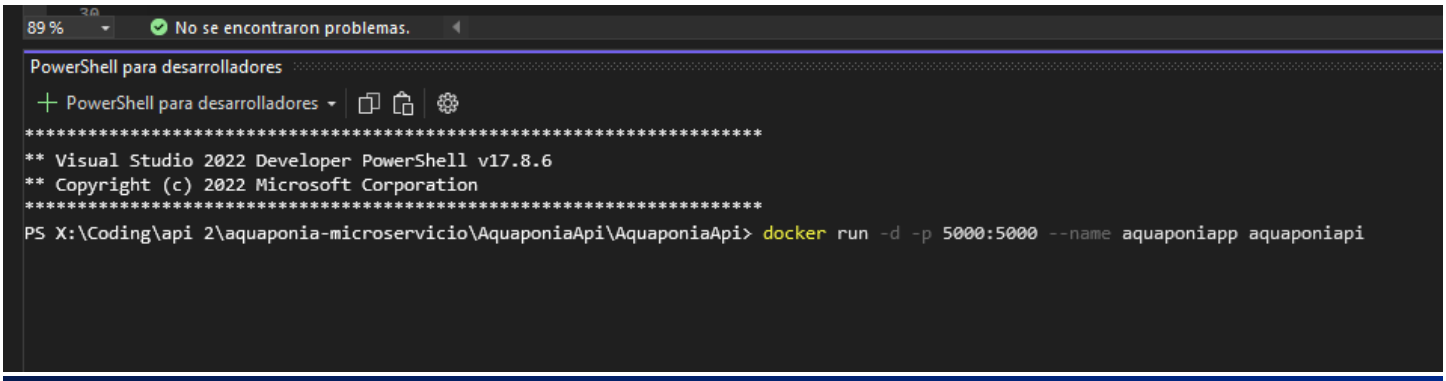
Integración de componentes heterogéneos

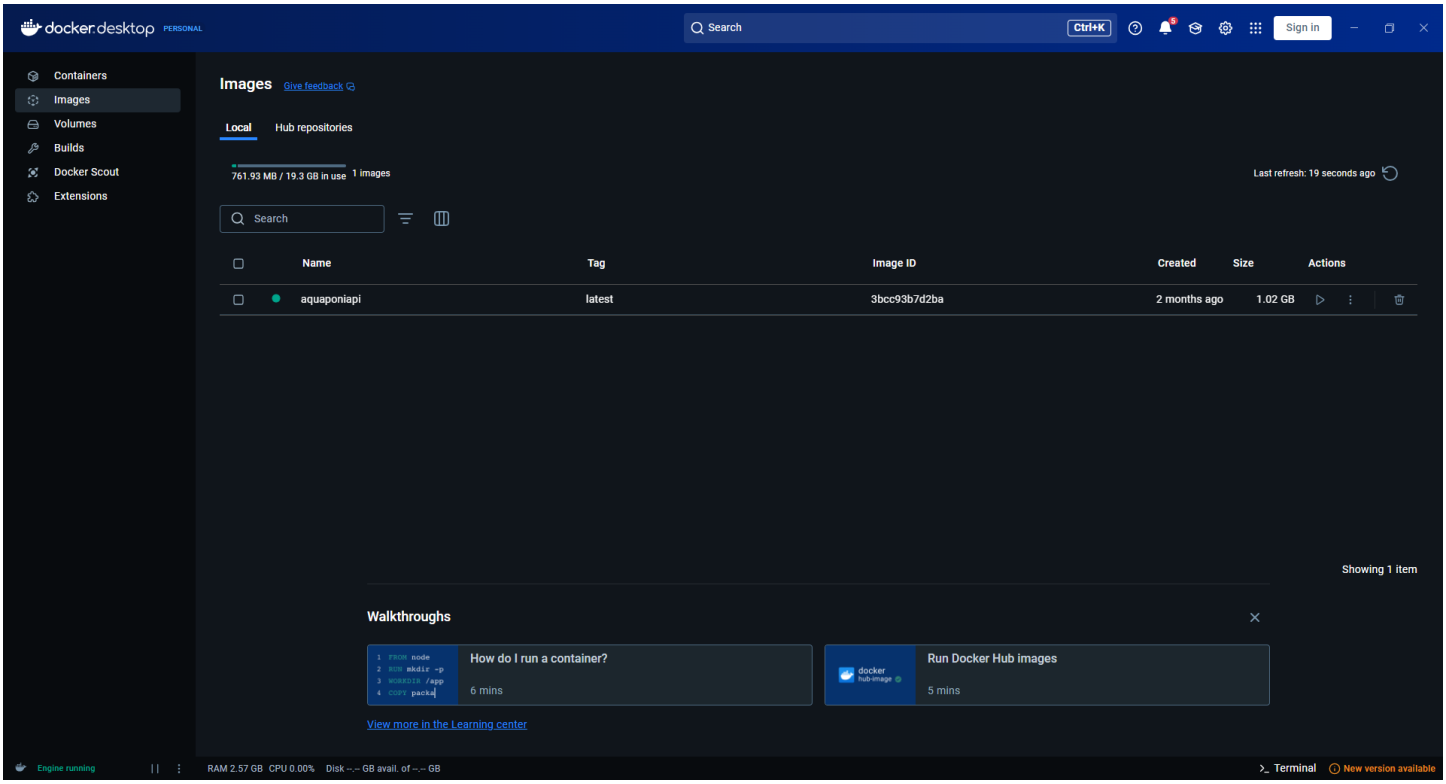
La combinación de .NET y Python en un mismo contenedor brinda flexibilidad para:

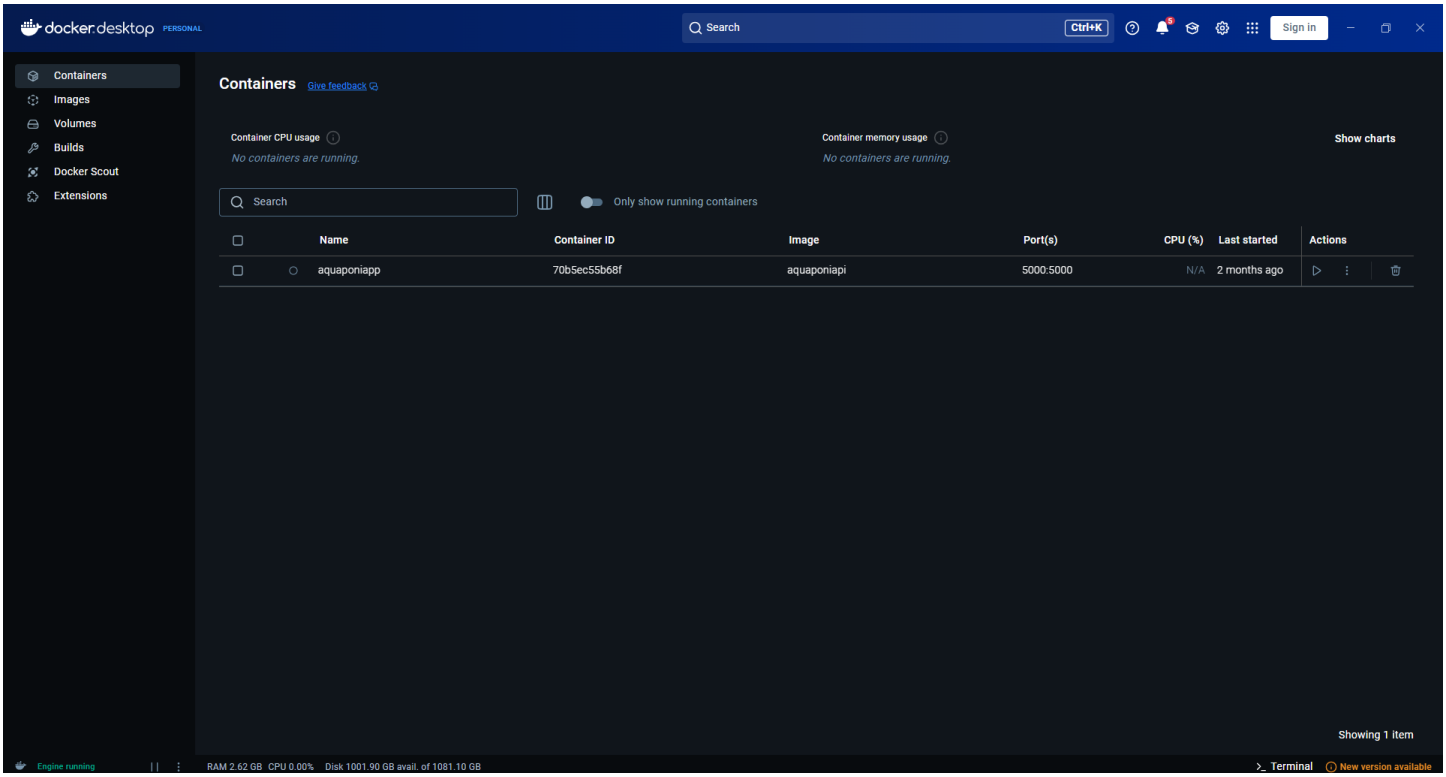
- Reutilizar código en diferentes lenguajes.
- Aprovechar bibliotecas especializadas (por ejemplo, scipy, opencv-python-headless).
- Mantener un único artefacto Docker para todo el stack.

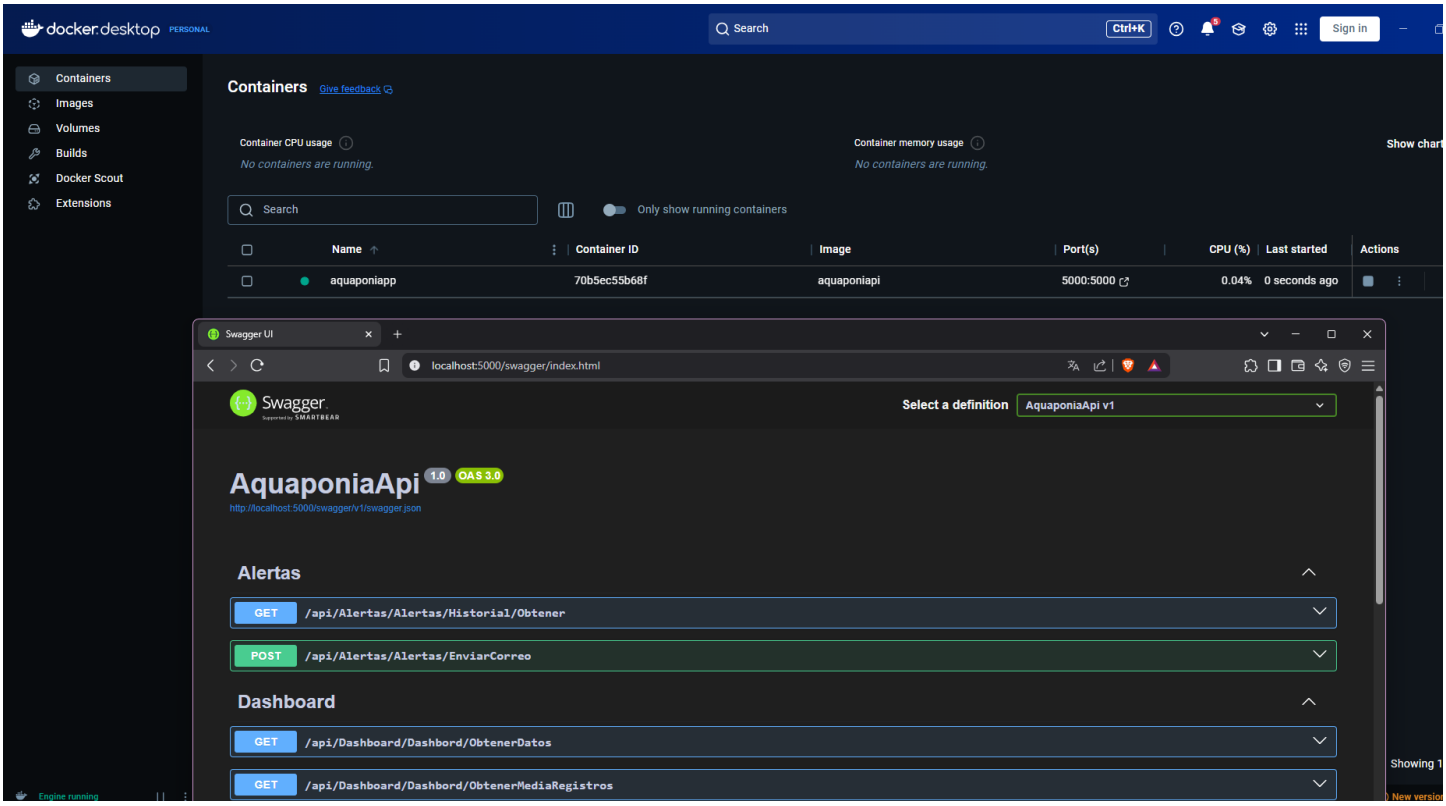
Gracias al uso de un entorno virtual, se evitan conflictos con versiones de Python del sistema y se facilita la actualización de paquetes sin reconstruir la imagen base.


PowerShell para desarrolladores
PowerShell para desarrolladores
** Visual Studio 2022 Developer PowerShell v17.8.6
** Copyright (c) 2022 Microsoft Corporation
PS X:\Coding\api 2\aquaponia-microservicio\AquaponiaApi\AquaponiaApi> docker build -t aquaponiapi .


89 %
No se encontraron problemas.
PowerShell para desarrolladores
PowerShell para desarrolladores
** Visual Studio 2022 Developer PowerShell v17.8.6
** Copyright (c) 2022 Microsoft Corporation
PS X:\Coding\api 2\aquaponia-microservicio\AquaponiaApi\AquaponiaApi> docker run -d -p 5000:5000 --name aquaponiapp aquaponiapi


docker desktop
Search
Ctrl+K
Sign in
Containers
Images
Volumes
Builds
Docker Scout
Extensions
Images
Give feedback
Local
Hub repositories
761.93 MB / 19.3 GB in use
1 images
Last refresh: 19 seconds ago
Search
Name
Tag
Image ID
Created
Size
Actions
aquaponiapi
latest
3bcc93b7d2ba
2 months ago
1.02 GB
Showing 1 item
Walkthroughs
How do I run a container?
6 mins
Run Docker Hub images
5 mins
View more in the Learning center
Engine running
RAM 2.57 GB CPU 0.00% Disk --.-- GB avail. of --.-- GB
Terminal
New version available

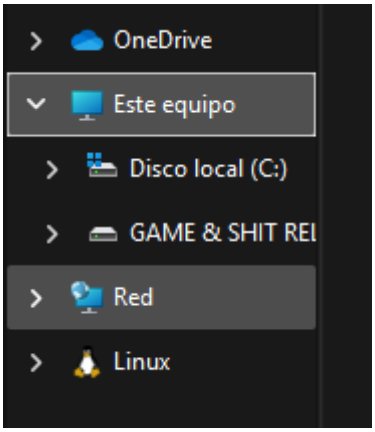




Etapa de compilaci□n con .NET SDK

```
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build-env

# Establecer directorio de trabajo
WORKDIR /app

# Copiar archivos del proyecto y restaurar dependencias
COPY *.csproj ./
RUN dotnet restore

# Copiar el resto de archivos y compilar la aplicaci□n
COPY . ./
RUN dotnet publish -c Release -o out

# ==============================
# Etapa final (Runtime) con .NET y Python
# ==============================
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS runtime-env
WORKDIR /app

# Instalar dependencias necesarias en una sola capa
RUN apt-get update && apt-get install -y --no-install-recommends \
    libexpat1 python3 python3-pip python3-venv && \
    python3 -m venv /opt/venv && \
    /opt/venv/bin/pip install --no-cache-dir scipy scikit-fuzzy requests networkx numpy opencv-python-headless Pillow packaging && \
    apt-get clean && rm -rf /var/lib/apt/lists/*

# Copiar la aplicaci□n publicada desde la imagen de compilaci□n
COPY --from=build-env /app/out .
```

```
# Copiar el script de Python
COPY --from=build-env /app/visionartificial.py .
COPY --from=build-env /app/controlador_difuso.py .
COPY --from=build-env /app/PerceptronNotificaciones.py .

# Configurar Python para que use el entorno virtual
ENV PATH="/opt/venv/bin:$PATH"

# Definir punto de entrada
ENTRYPOINT ["dotnet", "AquaponiaApi.dll"]
```



Conclusión:

El ejemplo desarrollado demuestra cómo los *multistage builds* de Docker optimizan el flujo de desarrollo y despliegue de aplicaciones complejas, combinando .NET y Python en un único contenedor eficiente. Al separar compilación y runtime, se logra un menor tamaño de imagen, mayor seguridad y tiempos de despliegue más rápidos. Además, la integración de scripts de IA en Python dota a la aplicación de capacidades avanzadas de procesamiento y notificación, alineadas con los requerimientos de cómputo tolerante a fallas. Esta estrategia constituye una base sólida para proyectos universitarios y profesionales que demanden flexibilidad, rendimiento y escalabilidad.

Repositorio:

https://gitlab.com/cucei5601406/aquaponia-microservicio

https://oyster-app-g3sar.ondigitalocean.app/swagger/index.html