

Informe de Actividad: Contenerización de una API de Machine Learning con Docker

Asignatura: Módulo 10 - MLOps

Estudiante: Daniel Araya

Fecha: 10 de Septiembre de 2025

1. Introducción

El presente informe detalla el proceso y los resultados obtenidos en la actividad de contenerización de una API de Machine Learning. El objetivo fue entrenar un modelo de clasificación, exponerlo a través de una API REST desarrollada con Flask y, finalmente, empaquetar toda la aplicación y sus dependencias en una imagen de Docker para asegurar un despliegue portable y reproducible. A continuación, se presentan las evidencias fotográficas que documentan cada etapa del proceso.

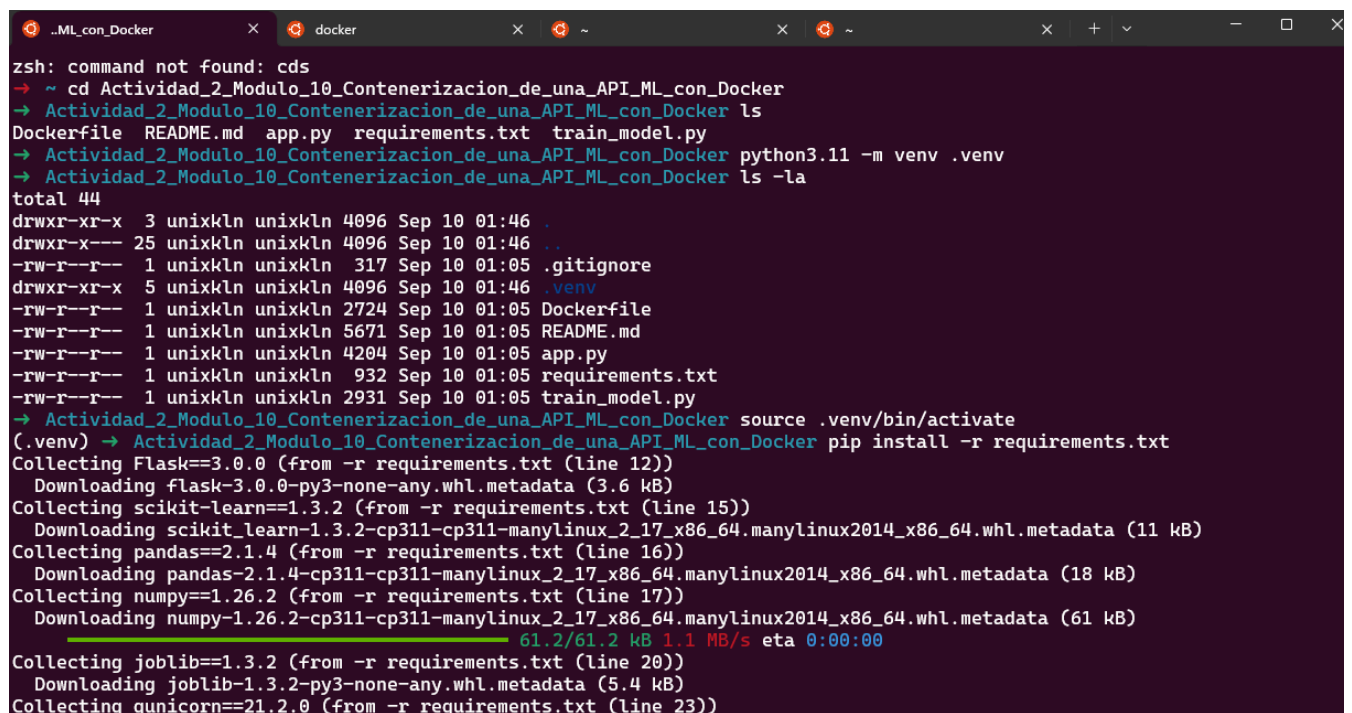
2. Evidencias del Proceso

Evidencia 1: Configuración del Entorno y Dependencias

En esta primera etapa, se preparó el entorno de desarrollo local. Los pasos realizados fueron:

1. **Navegación al Directorio:** Se accedió a la carpeta del proyecto.
2. **Creación del Entorno Virtual:** Se utilizó el comando `python3.11 -m venv .venv` para crear un entorno virtual aislado llamado `.venv`. Esto es una buena práctica para gestionar las dependencias del proyecto sin afectar la configuración global del sistema.
3. **Activación del Entorno:** Se activó el entorno con `source .venv/bin/activate`, lo que se evidencia por el prefijo `(.venv)` en la línea de comandos.
4. **Instalación de Dependencias:** Finalmente, se ejecutó `pip install -r requirements.txt` para instalar todas las librerías necesarias para el proyecto, como `Flask`, `scikit-learn` y `gunicorn`, dentro del entorno aislado.

Esta imagen demuestra que el entorno de trabajo fue configurado correctamente, asegurando una base limpia y reproducible para los siguientes pasos.



```
zsh: command not found: cds
→ ~ cd Actividad_2_Modulo_10_Contenerizacion_de_una_API_ML_con_Docker
→ Actividad_2_Modulo_10_Contenerizacion_de_una_API_ML_con_Docker ls
Dockerfile README.md app.py requirements.txt train_model.py
→ Actividad_2_Modulo_10_Contenerizacion_de_una_API_ML_con_Docker python3.11 -m venv .venv
→ Actividad_2_Modulo_10_Contenerizacion_de_una_API_ML_con_Docker ls -la
total 44
drwxr-xr-x  3 unixkln unixkln 4096 Sep 10 01:46 .
drwxr-x--- 25 unixkln unixkln 4096 Sep 10 01:46 ..
-rw-r--r--  1 unixkln unixkln  317 Sep 10 01:05 .gitignore
drwxr-xr-x  5 unixkln unixkln 4096 Sep 10 01:46 .venv
-rw-r--r--  1 unixkln unixkln 2724 Sep 10 01:05 Dockerfile
-rw-r--r--  1 unixkln unixkln 5671 Sep 10 01:05 README.md
-rw-r--r--  1 unixkln unixkln 4204 Sep 10 01:05 app.py
-rw-r--r--  1 unixkln unixkln  932 Sep 10 01:05 requirements.txt
-rw-r--r--  1 unixkln unixkln 2931 Sep 10 01:05 train_model.py
→ Actividad_2_Modulo_10_Contenerizacion_de_una_API_ML_con_Docker source .venv/bin/activate
(.venv) → Actividad_2_Modulo_10_Contenerizacion_de_una_API_ML_con_Docker pip install -r requirements.txt
Collecting Flask==3.0.0 (from -r requirements.txt (line 12))
  Downloading flask-3.0.0-py3-none-any.whl.metadata (3.6 kB)
Collecting scikit-learn==1.3.2 (from -r requirements.txt (line 15))
  Downloading scikit_learn-1.3.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (11 kB)
Collecting pandas==2.1.4 (from -r requirements.txt (line 16))
  Downloading pandas-2.1.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (18 kB)
Collecting numpy==1.26.2 (from -r requirements.txt (line 17))
  Downloading numpy-1.26.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (61 kB)
  61.2/61.2 kB 1.1 MB/s eta 0:00:00
Collecting joblib==1.3.2 (from -r requirements.txt (line 20))
  Downloading joblib-1.3.2-py3-none-any.whl.metadata (5.4 kB)
Collecting gunicorn==21.2.0 (from -r requirements.txt (line 23))
```

Evidencia 2: Entrenamiento del Modelo de Machine Learning

Una vez configurado el entorno, el siguiente paso fue entrenar el modelo de clasificación.

1. **Ejecución del Script de Entrenamiento:** Se ejecutó el comando `python train_model.py`. Este script carga el dataset "Wine" de scikit-learn, lo divide, inicializa un modelo `RandomForestClassifier` y lo entrena con los datos.
2. **Generación del Modelo:** El script finaliza guardando el modelo entrenado en un archivo binario llamado `modelo.pkl`.
3. **Verificación:** El comando `ls` posterior confirma la creación exitosa del archivo `modelo.pkl`, que es esencial para que la API pueda realizar predicciones. El mensaje de salida también confirma que el entrenamiento fue exitoso y reporta una precisión del 100% en el conjunto de prueba.

```
..ML_con_Docker  X  docker  X  ~  X  ~  X  +  v  -  □  X
35.4/35.4 MB 52.8 MB/s eta 0:00:00
Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Downloading tzdata-2025.2-py2.py3-none-any.whl (347 kB)
347.8/347.8 kB 28.0 MB/s eta 0:00:00
Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB)
224.5/224.5 kB 20.6 MB/s eta 0:00:00
Downloading packaging-25.0-py3-none-any.whl (66 kB)
66.5/66.5 kB 6.8 MB/s eta 0:00:00
Downloading MarkupSafe-3.0.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (23 kB)
Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: pytz, tzdata, threadpoolctl, six, packaging, numpy, MarkupSafe, joblib, itsdangerous, click, blinker, Werkzeug, scipy, python-dateutil, Jinja2, gunicorn, scikit-learn, pandas, Flask
Successfully installed Flask-3.0.0 Jinja2-3.1.6 MarkupSafe-3.0.2 Werkzeug-3.1.3 blinker-1.9.0 click-8.2.1 gunicorn-21.2.0 itsdangerous-2.2.0 joblib-1.3.2 numpy-1.26.2 packaging-25.0 pandas-2.1.4 python-dateutil-2.9.0.post0 pytz-2025.2 scikit-learn-1.3.2 scipy-1.16.1 six-1.17.0 threadpoolctl-3.6.0 tzdata-2025.2

[notice] A new release of pip is available: 24.0 -> 25.2
[notice] To update, run: pip install --upgrade pip
(.venv) → Actividad_2_Modulo_10_Contenerizacion_de_una_API_ML_con_Docker python train_model.py
Cargando el dataset de vinos...
Dividiendo los datos en conjuntos de entrenamiento y prueba...
Iniciando el modelo RandomForestClassifier...
Entrenando el modelo...
Guardando el modelo entrenado en 'modelo.pkl'...

¡Entrenamiento completado y modelo guardado exitosamente!
El modelo tiene una precisión de 1.00 en el conjunto de prueba.
(.venv) → Actividad_2_Modulo_10_Contenerizacion_de_una_API_ML_con_Docker ls
Dockerfile README.md app.py modelo.pkl requirements.txt train_model.py
(.venv) → Actividad_2_Modulo_10_Contenerizacion_de_una_API_ML_con_Docker
```

Evidencia 3: Construcción de la Imagen Docker

Con el modelo ya entrenado y todos los archivos de la aplicación listos, se procedió a construir la imagen de Docker.

1. **Comando de Construcción:** Se utilizó el comando `docker build -t ml-wine-api .`. Este comando le indica a Docker que lea las instrucciones del archivo `Dockerfile` en el directorio actual (`.`) y construya una imagen con el nombre `(-t) ml-wine-api`.
2. **Proceso de Construcción:** La salida de la terminal muestra cada paso de la construcción: se parte de una imagen base de Python, se establece el directorio de trabajo, se copian los archivos (`requirements.txt`, `app.py`, `modelo.pkl`, etc.) y se instalan las dependencias. El estado `FINISHED` confirma que la imagen se creó sin errores.

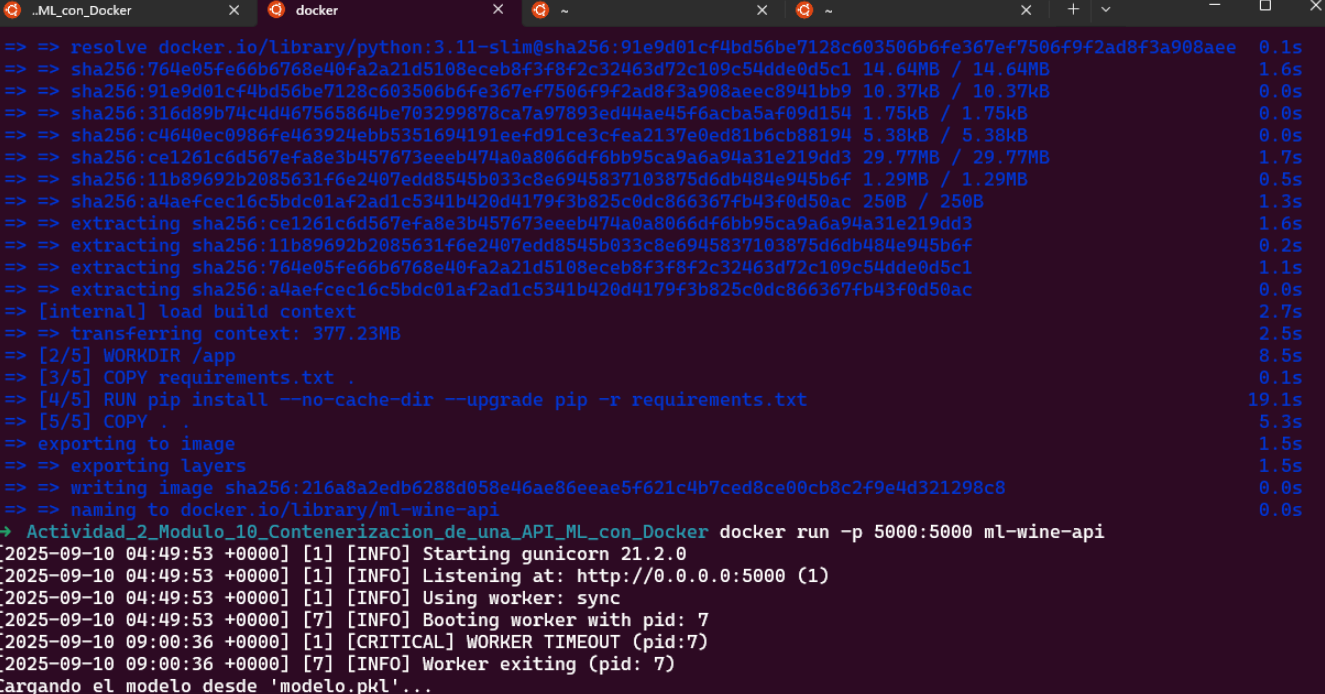
Esta imagen es la evidencia de que la aplicación fue empaquetada exitosamente en una imagen de Docker, lista para ser ejecutada.

```
→ ~ cd Actividad_2_Modulo_10_Contenerizacion_de_una_API_ML_con_Docker
→ Actividad_2_Modulo_10_Contenerizacion_de_una_API_ML_con_Docker docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS        NAMES
→ Actividad_2_Modulo_10_Contenerizacion_de_una_API_ML_con_Docker docker build -t ml-wine-api .
[+] Building 42.2s (11/11) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 2.76kB                             0.0s
=> [internal] load metadata for docker.io/library/python:3.11-slim 2.2s
=> [auth] library/python:pull token for registry-1.docker.io      0.0s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                       0.0s
=> [1/5] FROM docker.io/library/python:3.11-slim@sha256:91e9d01cf4bd56be7128c603506b6fe367ef7506f9f2ad8f3a908aee 5.2s
=> => resolve docker.io/library/python:3.11-slim@sha256:91e9d01cf4bd56be7128c603506b6fe367ef7506f9f2ad8f3a908aee 0.1s
=> => sha256:764e05fe66b6768e40fa2a21d5108eceb8f3f8f2c32463d72c109c54dde0d5c1 14.64MB / 14.64MB 1.6s
=> => sha256:91e9d01cf4bd56be7128c603506b6fe367ef7506f9f2ad8f3a908aee8941bb9 10.37kB / 10.37kB 0.0s
=> => sha256:316d89b74c4d467565864be703299878ca7a97893ed44ae45f6acba5af09d154 1.75kB / 1.75kB 0.0s
=> => sha256:c4640ec0986fe463924ebb5351694191ee9d91ce3cfea2137e0ed81b6cb88194 5.38kB / 5.38kB 0.0s
=> => sha256:ce1261c6d567efa8e3b457673eeeb474a0a8066df6bb95ca9a6a94a31e219dd3 29.77MB / 29.77MB 1.7s
=> => sha256:11b89692b2085631f6e2407edd8545b033c8e6945837103875d6db484e945b6f 1.29MB / 1.29MB 0.5s
=> => sha256:a4aefcec16c5bdc01af2ad1c5341b420d4179f3b825c0dc866367fb43f0d50ac 250B / 250B 1.3s
=> => extracting sha256:ce1261c6d567efa8e3b457673eeeb474a0a8066df6bb95ca9a6a94a31e219dd3 1.6s
=> => extracting sha256:11b89692b2085631f6e2407edd8545b033c8e6945837103875d6db484e945b6f 0.2s
=> => extracting sha256:764e05fe66b6768e40fa2a21d5108eceb8f3f8f2c32463d72c109c54dde0d5c1 1.1s
=> => extracting sha256:a4aefcec16c5bdc01af2ad1c5341b420d4179f3b825c0dc866367fb43f0d50ac 0.0s
=> [internal] load build context                                  2.7s
=> => transferring context: 377.23MB                                2.5s
=> [2/5] WORKDIR /app                                           8.5s
=> [3/5] COPY requirements.txt .                                0.1s
=> [4/5] RUN pip install --no-cache-dir --upgrade pip -r requirements.txt 19.1s
=> [5/5] COPY . .                                              5.3s
```

Evidencia 4: Ejecución del Contenedor Docker

Una vez construida la imagen, el siguiente paso fue crear y ejecutar un contenedor a partir de ella.

1. **Comando de Ejecución:** Se lanzó el comando `docker run -p 5000:5000 ml-wine-api`. Este comando inicia un contenedor desde la imagen `ml-wine-api` y mapea (-p) el puerto 5000 del contenedor al puerto 5000 de la máquina local, permitiendo el acceso a la API desde el exterior.
2. **Logs del Servidor:** La terminal muestra los logs del servidor Gunicorn. Los mensajes `[INFO] Listening at: http://0.0.0.0:5000` y `[INFO] Booting worker with pid: 7` confirman que el servidor web dentro del contenedor se ha iniciado correctamente y está escuchando peticiones. También se observa el mensaje "Cargando el modelo desde 'modelo.pkl'...", indicando que la aplicación Flask se inicializó y cargó el modelo sin problemas.



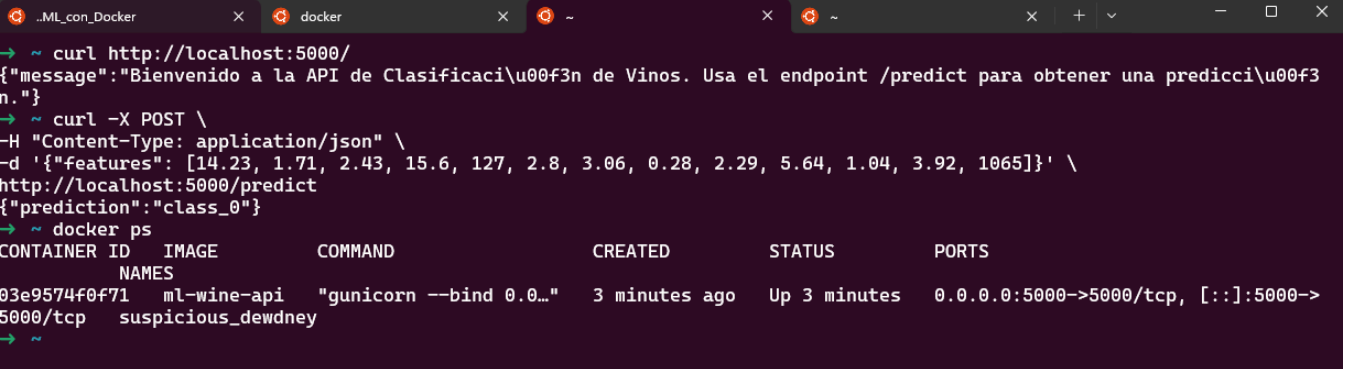
```
..ML_con_Docker x docker x ~ x + v - □ x
=> => resolve docker.io/library/python:3.11-slim@sha256:91e9d01cf4bd56be7128c603506b6fe367ef7506f9f2ad8f3a908aee 0.1s
=> => sha256:764e05fe66b6768e40fa2a21d5108eceb8f3f8f2c32463d72c109c54dde0d5c1 14.64MB / 14.64MB 1.6s
=> => sha256:91e9d01cf4bd56be7128c603506b6fe367ef7506f9f2ad8f3a908aee8941bb9 10.37kB / 10.37kB 0.0s
=> => sha256:316d89b74c4d467565864be703299878ca7a97893ed44ae45f6acba5af09d154 1.75kB / 1.75kB 0.0s
=> => sha256:c4640ec0986fe463924ebb5351694191eeef91ce3cfea2137e0ed81b6cb88194 5.38kB / 5.38kB 0.0s
=> => sha256:ce1261c6d567efa8e3b457673eeeb474a0a8066df6bb95ca9a6a94a31e219dd3 29.77MB / 29.77MB 1.7s
=> => sha256:11b89692b2085631f6e2407edd8545b033c8e6945837103875d6db484e945b6f 1.29MB / 1.29MB 0.5s
=> => sha256:a4aefcec16c5bdc01af2ad1c5341b420d4179f3b825c0dc866367fb43f0d50ac 250B / 250B 1.3s
=> => extracting sha256:ce1261c6d567efa8e3b457673eeeb474a0a8066df6bb95ca9a6a94a31e219dd3 1.6s
=> => extracting sha256:11b89692b2085631f6e2407edd8545b033c8e6945837103875d6db484e945b6f 0.2s
=> => extracting sha256:764e05fe66b6768e40fa2a21d5108eceb8f3f8f2c32463d72c109c54dde0d5c1 1.1s
=> => extracting sha256:a4aefcec16c5bdc01af2ad1c5341b420d4179f3b825c0dc866367fb43f0d50ac 0.0s
=> [internal] load build context
=> => transferring context: 377.23MB 2.7s
=> [2/5] WORKDIR /app 2.5s
=> [3/5] COPY requirements.txt . 8.5s
=> [4/5] RUN pip install --no-cache-dir --upgrade pip -r requirements.txt 0.1s
=> [5/5] COPY . . 19.1s
=> exporting to image 5.3s
=> => exporting layers 1.5s
=> => writing image sha256:216a8a2edb6288d058e46ae86eeae5f621c4b7ced8ce00cb8c2f9e4d321298c8 1.5s
=> => naming to docker.io/library/ml-wine-api 0.0s
→ Actividad_2_Modulo_10_Contenerizacion_de_una_API_ML_con_Docker docker run -p 5000:5000 ml-wine-api 0.0s
[2025-09-10 04:49:53 +0000] [1] [INFO] Starting gunicorn 21.2.0
[2025-09-10 04:49:53 +0000] [1] [INFO] Listening at: http://0.0.0.0:5000 (1)
[2025-09-10 04:49:53 +0000] [1] [INFO] Using worker: sync
[2025-09-10 04:49:53 +0000] [7] [INFO] Booting worker with pid: 7
[2025-09-10 09:00:36 +0000] [1] [CRITICAL] WORKER TIMEOUT (pid:7)
[2025-09-10 09:00:36 +0000] [7] [INFO] Worker exiting (pid: 7)
Cargando el modelo desde 'modelo.pkl'...
```

Evidencia 5: Validación y Pruebas de la API

La etapa final consistió en validar que la API dentro del contenedor estuviera completamente funcional.

1. **Prueba del Endpoint Raíz:** Se utilizó `curl http://localhost:5000/` para enviar una petición GET al endpoint de bienvenida. La API respondió correctamente con el mensaje JSON esperado.
2. **Prueba del Endpoint de Predicción:** Se envió una petición POST al endpoint `/predict` usando `curl -X POST`, incluyendo los datos de un vino en formato JSON. La API procesó los datos, utilizó el modelo cargado y devolvió la predicción `{"prediction": "class_0"}`, lo cual es el comportamiento esperado.
3. **Verificación del Contenedor:** Finalmente, el comando `docker ps` muestra que el contenedor basado en la imagen `ml-wine-api` se encuentra en estado `Up` (en ejecución), confirmando que el servicio está activo.

Esta evidencia demuestra de manera concluyente que la API contenerizada funciona correctamente y es accesible desde el exterior.



```
→ ~ curl http://localhost:5000/
{"message": "Bienvenido a la API de Clasificaci\u00f3n de Vinos. Usa el endpoint /predict para obtener una predicción."}
→ ~ curl -X POST \
-H "Content-Type: application/json" \
-d '{"features": [14.23, 1.71, 2.43, 15.6, 127, 2.8, 3.06, 0.28, 2.29, 5.64, 1.04, 3.92, 1065]}' \
http://localhost:5000/predict
{"prediction": "class_0"}
→ ~ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
03e9574f0f71   ml-wine-api    "unicorn --bind 0.0..." 3 minutes ago  Up 3 minutes  0.0.0.0:5000->5000/tcp, [::]:5000->5000/tcp
suspicious_dewdney
```

3. Conclusión

La actividad se completó con éxito, cubriendo todo el ciclo de vida básico de un proyecto de MLOps: desde la configuración del entorno y el entrenamiento del modelo hasta su encapsulamiento en un contenedor Docker y la validación de su funcionalidad. El resultado es un servicio de Machine Learning robusto, portable y listo para un despliegue en cualquier entorno que soporte Docker.