

1

Instalación del entorno de trabajo: Jupyter Notebook (en Docker) y Visual Studio Code

En este capítulo se explica cómo instalar y configurar el entorno de desarrollo que usaremos a lo largo del libro. La estrategia consiste en **desplegar Python y Jupyter Notebook dentro de un contenedor Docker**, de modo que el lector trabaje sobre un entorno reproducible, idéntico al del autor e independiente del sistema operativo de su máquina (Windows, macOS o Linux). Adicionalmente, se describe cómo emplear Visual Studio Code como editor frente al servidor Jupyter en ejecución dentro del contenedor, lo que aporta una experiencia de desarrollo más rica que la del navegador para proyectos con varios ficheros.

1.1. ¿Por qué un entorno aislado en Docker?

Aprender a programar implica equivocarse: instalar bibliotecas que rompen otras versiones, mezclar entornos, terminar con una instalación de Python desordenada en el sistema. Para evitarlo, este libro adopta un enfoque de **entorno reproducible**: empaquetar Python, Jupyter y todas las dependencias dentro de un contenedor Docker. Las ventajas son tres:

- *Aislamiento*: lo que se instale dentro del contenedor no afecta al resto del sistema. Si algo falla, basta con borrar el contenedor y crearlo de nuevo.
- *Reproducibilidad*: el mismo «Dockerfile» produce el mismo entorno en cualquier máquina, hoy y dentro de tres años. Los ejemplos del libro funcionan exactamente

1. Instalación del entorno de trabajo: Jupyter Notebook (en Docker) y Visual Studio Code

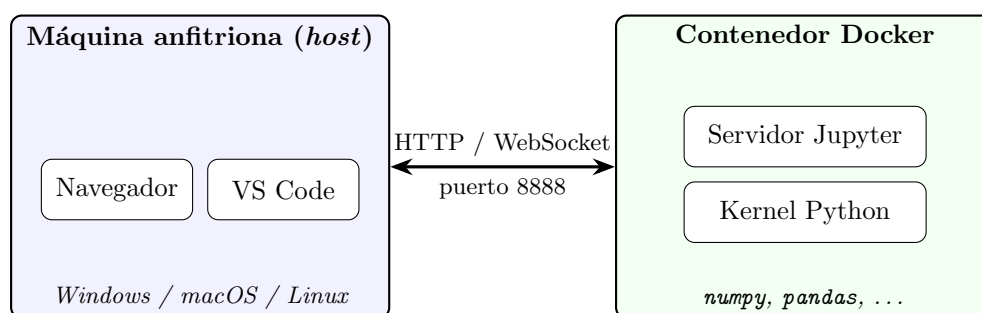


Figura 1: Arquitectura del entorno de trabajo. El navegador o Visual Studio Code, instalados en la máquina anfitriona, se comunican con el servidor Jupyter que corre dentro del contenedor Docker mediante el puerto 8888.

igual en tu equipo que en el del autor.

- *Portabilidad:* el contenedor se ejecuta de la misma forma en Windows, macOS y Linux.

La Figura 1 resume la arquitectura del entorno: la máquina anfitriona (*host*) ejecuta Docker, dentro del cual corre el servidor Jupyter; el lector accede al servidor desde un navegador o desde Visual Studio Code mediante el puerto 8888.

1.2. Instalación de Docker

Antes de comenzar, asegúrate de tener Docker instalado en tu máquina. Si aún no lo tienes, descárgalo desde el sitio oficial: <https://www.docker.com/get-started/>. En este libro recomendamos instalar Docker Desktop (<https://docs.docker.com/desktop/>), ya que ofrece una interfaz sencilla para gestionar contenedores y simplifica la configuración del entorno tanto en Windows como en macOS o Linux.

Tras instalarlo, ejecuta la aplicación. Aparecerá una ventana similar a la Figura 2. No es necesario crear ninguna cuenta: puedes saltar este paso (opción `Skip` en la esquina superior derecha).

1.3. Configuración del entorno: Dockerfile y dependencias

La forma más cómoda de definir un entorno Docker es escribir un fichero llamado «Dockerfile» que describe, paso a paso, cómo construirlo: imagen base, dependencias a instalar, puerto a exponer y comando que se ejecuta al arrancar. A continuación se muestra el «Dockerfile» que usaremos a lo largo del libro (Código 1.1). Crea este fichero, con ese nombre exacto, en el directorio que vayas a usar como espacio de trabajo.

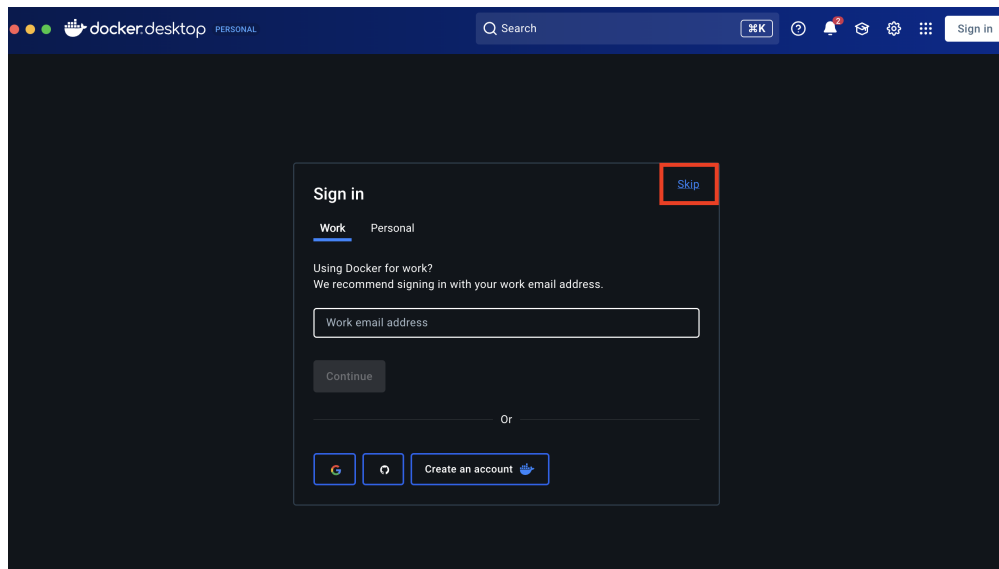


Figura 2: Ventana de inicio de Docker Desktop.

Código 1.1: Fichero «Dockerfile» para crear el entorno de trabajo del libro.

```
1  # Imagen oficial de Python como base
2  FROM python:3.12
3
4  # Variables de entorno recomendadas para Python en contenedor
5  ENV PYTHONDONTWRITEBYTECODE=1
6  ENV PYTHONUNBUFFERED=1
7
8  # Directorio de trabajo dentro del contenedor
9  WORKDIR /workspace
10
11 # Copia el listado de dependencias al contenedor
12 COPY requirements.txt .
13
14 # Instala JupyterLab y las bibliotecas del libro
15 RUN pip install --no-cache-dir jupyterlab && \
16     pip install --no-cache-dir -r requirements.txt
17
18 # Expone el puerto 8888 para Jupyter
19 EXPOSE 8888
20
```

```
21 # Arranca Jupyter al iniciar el contenedor
22 ENTRYPOINT ["jupyter", "lab", "--ip=0.0.0.0", "--port=8888",
↪ \
23     "--no-browser", "--allow-root"]
```

Cada bloque cumple una función concreta:

- `FROM python:3.12` parte de una imagen oficial de Python en su versión 3.12, que ya incluye el intérprete y `pip` preinstalados.
- Las dos variables `PYTHONDONTWRITEBYTECODE` y `PYTHONUNBUFFERED` son recomendaciones habituales para Python dentro de un contenedor: evitan generar ficheros `.pyc` y aseguran que las salidas (`print`, errores) aparezcan al instante en el log.
- `WORKDIR /workspace` fija `«/workspace»` como directorio de trabajo dentro del contenedor.
- `COPY requirements.txt .` introduce dentro del contenedor el listado de dependencias (que veremos enseguida).
- `RUN pip install ...` instala JupyterLab (que incluye Jupyter Notebook clásico) y las bibliotecas adicionales declaradas en `«requirements.txt»`.
- `EXPOSE 8888` declara que el contenedor publicará el servicio en el puerto 8888 (el puerto por defecto de Jupyter).
- `ENTRYPOINT [...]` indica el comando que se ejecuta al iniciar el contenedor: arrancar JupyterLab escuchando en todas las interfaces, sin abrir un navegador interno y permitiendo el acceso como `root` dentro del contenedor.

1.3.1. Dependencias adicionales

Para los ejemplos y ejercicios del libro será necesario instalar algunas bibliotecas adicionales en el entorno de trabajo. Se trata de bibliotecas para el manejo eficiente de datos numéricos y estructurados, así como para su visualización:

- **Manipulación de datos:**
 - `numpy`: cálculo numérico vectorizado y manipulación de vectores y matrices multidimensionales.
 - `pandas`: análisis y manipulación de datos tabulares (lectura de CSV, agrupaciones, filtrados, etc.).
- **Visualización de datos:**
 - `matplotlib`: representación gráfica de datos (líneas, barras, dispersión, etc.).
 - `seaborn`: gráficos estadísticos de alto nivel contruidos sobre `matplotlib`.

Si más adelante quieres trabajar con técnicas concretas de aprendizaje automático o de análisis de datos, podrás añadir bibliotecas adicionales: `scipy` (operaciones matemáticas avanzadas), `statsmodels` (modelos estadísticos), `plotly` (visualización interactiva), `requests` (acceso a servicios web) o `Pillow` (procesamiento de imágenes), entre muchas otras. Sin embargo, este libro sólo necesita las cinco listadas anteriormente.

Crea, en el mismo directorio donde guardaste el «`Dockerfile`», un fichero llamado «`requirements.txt`» con el contenido del Código 1.2. Las versiones se fijan a un mínimo razonable para garantizar la reproducibilidad de los ejemplos del libro a lo largo del tiempo.

Código 1.2: Fichero «`requirements.txt`» con las dependencias usadas en el libro.

```
1 numpy>=2.0,<3
2 pandas>=2.2,<3
3 matplotlib>=3.9,<4
4 seaborn>=0.13,<1
5 scikit-learn>=1.4,<2
```

1.4. Construcción de la imagen

A partir del «`Dockerfile`» y del «`requirements.txt`» es necesario construir la imagen de Docker, es decir, materializar la receta en una plantilla ejecutable. Para ello, abre una terminal en el directorio donde se encuentran ambos ficheros y ejecuta:

```
docker build -t fundamentos-python .
```

Este comando crea una imagen llamada `fundamentos-python` con todas las dependencias del libro. El proceso puede tardar unos minutos la primera vez, ya que descargará la imagen base de Python 3.12, JupyterLab y las bibliotecas del «`requirements.txt`». Una vez finalizado, la imagen quedará lista para ejecutar el contenedor.

1.5. Ejecución del contenedor

Una vez construida la imagen, hay dos formas de levantar el contenedor: a través de la interfaz gráfica de Docker Desktop o desde la línea de comandos.

1.5.1. Desde Docker Desktop

Abre Docker Desktop y dirígete a la sección `Images`. Como se muestra en la Figura 3, la imagen recién creada aparece en la lista con el nombre `fundamentos-python` y la etiqueta `latest`.

Para iniciar el contenedor desde la propia aplicación, pulsa el botón ► (parte derecha de la Figura 3). Se abrirá una ventana de configuración opcional (Figura 4) donde conviene rellenar:

1. Instalación del entorno de trabajo: Jupyter Notebook (en Docker) y Visual Studio Code

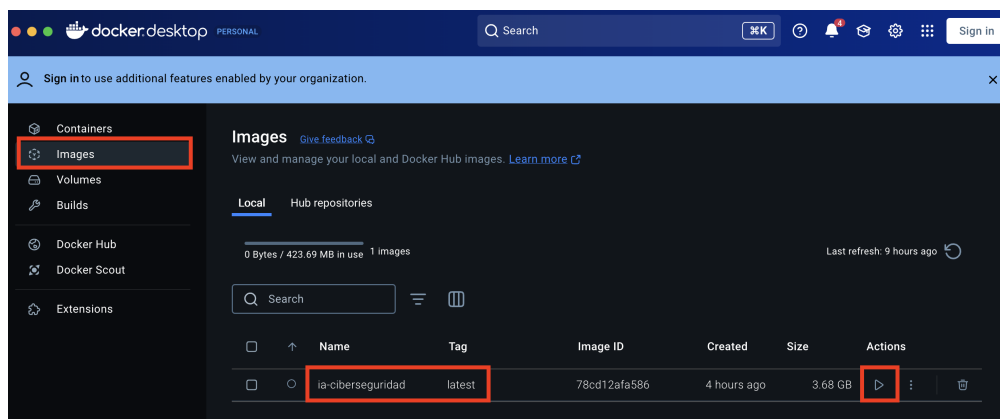


Figura 3: Docker Desktop mostrando la imagen `fundamentos-python` creada anteriormente.

- *Nombre del contenedor*: puedes dejar `fundamentos-python` o elegir otro nombre.
- *Puertos*: asegúrate de que el campo *Host port* sea 8888. Si ese puerto está ocupado por otro servicio en tu máquina, puedes usar otro distinto. Esta configuración mapea el puerto interno del contenedor (donde escucha Jupyter) al puerto correspondiente de la máquina *host*, permitiendo el acceso desde el navegador o desde VS Code.

Al pulsar `Run` se inicia el contenedor y aparece la pantalla de log (Figura 5). Anota la URL que aparece en el log: la necesitarás para conectarte al servidor Jupyter desde el navegador.

1.5.2. Desde la terminal

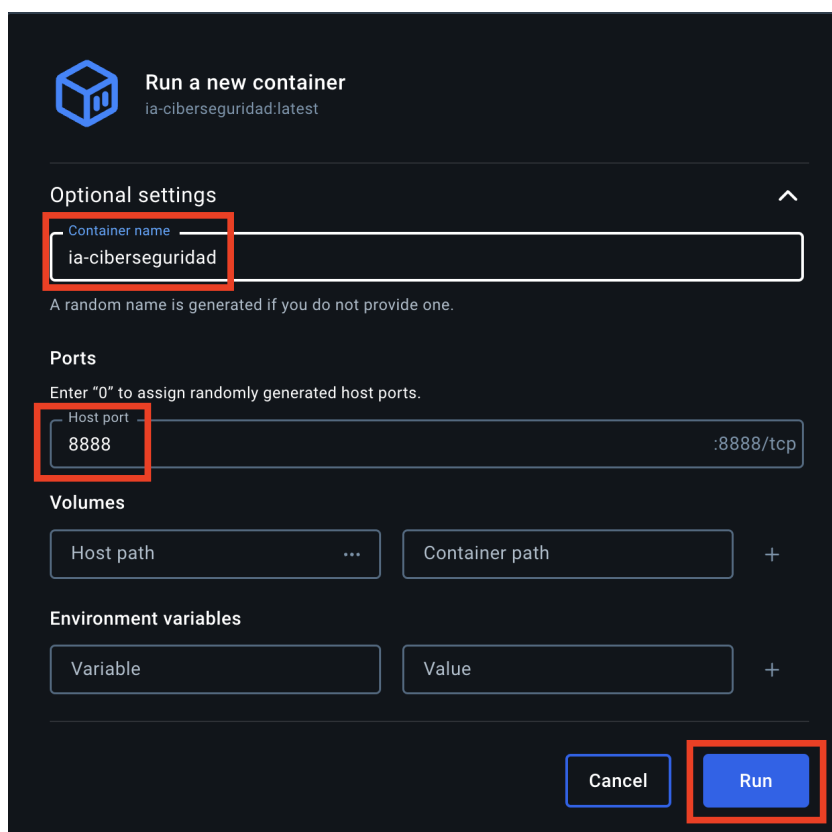
Como alternativa, en macOS o Linux abre una terminal y ejecuta:

```
docker run --rm -p 8888:8888 --name fundamentos-python \
-v "$(pwd)":/workspace fundamentos-python
```

En Windows, si usas *PowerShell*, sustituye `$(pwd)` por `${PWD}`; si usas la consola `cmd`, por `%cd%`.

El comando `docker run` usa varios parámetros:

- `--rm`: borra el contenedor automáticamente al pararse, evitando dejar contenedores huérfanos.
- `-p 8888:8888`: mapea el puerto 8888 de la máquina anfitriona al puerto 8888 del contenedor (formato `HOST:CONTENEDOR`).
- `--name fundamentos-python`: asigna un nombre identificativo al contenedor.



The screenshot shows the 'Run a new container' window in Docker Desktop. At the top, it says 'Run a new container' and 'ia-ciberseguridad:latest'. Below this is an 'Optional settings' section. The 'Container name' field is highlighted with a red box and contains the text 'ia-ciberseguridad'. Below this, it says 'A random name is generated if you do not provide one.' The 'Ports' section is next, with the instruction 'Enter "0" to assign randomly generated host ports.' The 'Host port' field is highlighted with a red box and contains the text '8888'. To the right of this field, it says ':8888/tcp'. Below the ports section is the 'Volumes' section, which has two input fields: 'Host path' and 'Container path', with a plus sign to the right. The 'Environment variables' section is at the bottom, with two input fields: 'Variable' and 'Value', with a plus sign to the right. At the bottom right of the window, there are two buttons: 'Cancel' and 'Run'. The 'Run' button is highlighted with a red box.

Figura 4: Configuración del contenedor de la imagen `fundamentos-python` en Docker Desktop.

- `-v "$(pwd)":/workspace`: monta el directorio de trabajo actual de la máquina anfitriona dentro de `/workspace` en el contenedor, de modo que los notebooks que crees se guarden en tu disco y no se pierdan al borrar el contenedor.
- El último `fundamentos-python` es el nombre de la imagen a ejecutar.

Tras el arranque, la terminal muestra el log del servidor Jupyter, incluida la URL completa con el *token* de autenticación.

1.6. Acceso a Jupyter desde el navegador

Jupyter es una herramienta interactiva que permite ejecutar código en celdas, ver resultados de forma inmediata y combinar texto, gráficos y código en un mismo documento. Su unidad básica de trabajo es el *notebook*: un fichero con extensión `.ipynb` que contiene una secuencia de celdas en las que se puede escribir y ejecutar código Python, añadir explicaciones en texto enriquecido (formato Markdown) o insertar imágenes, tablas y gráficos.

1. Instalación del entorno de trabajo: Jupyter Notebook (en Docker) y Visual Studio Code

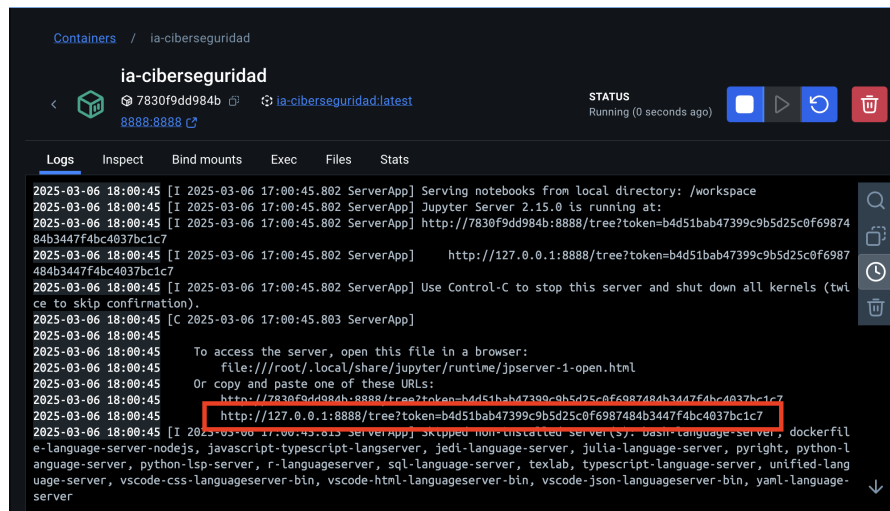


Figura 5: Pantalla de *log* en Docker Desktop tras arrancar el contenedor: muestra la URL con el *token* para acceder a Jupyter.

1.6.1. Conectar al servidor

Con el contenedor en ejecución, abre un navegador web y accede a la URL que aparece en el log. Tendrá la forma:

`http://127.0.0.1:8888/lab?token=...`

Si cambiaste el puerto del *host* en el paso anterior, actualízalo en la URL. Es imprescindible incluir el `?token=...` que ofrece el log: es el mecanismo de autenticación de Jupyter.

Al acceder, se abre la interfaz principal de JupyterLab, donde se ven los ficheros del directorio de trabajo (entre ellos, el `requirements.txt` si montaste el directorio con `-v`); véase la Figura 6.

1.6.2. Crear un notebook y elegir kernel

Para crear un *notebook*, dirígete a `File` `New` `Notebook` (Figura 7). Aparecerá una ventana de selección de *kernel* (Figura 8).

El *kernel* es el motor que ejecuta el código del *notebook*. Jupyter soporta múltiples lenguajes, aunque el más habitual es Python con el *kernel* `ipykernel`. En la ventana de selección, elige `Python 3 (ipykernel)` y pulsa `Select`. Si en algún momento el *kernel* deja de responder, puedes reiniciarlo desde `Kernel` `Restart`.

Un *notebook* se compone de celdas de varios tipos:

- *Celdas de código*: contienen código Python que se ejecuta al pulsar `⇧` + `⇩` o el botón `▶`; el resultado aparece debajo de la celda.

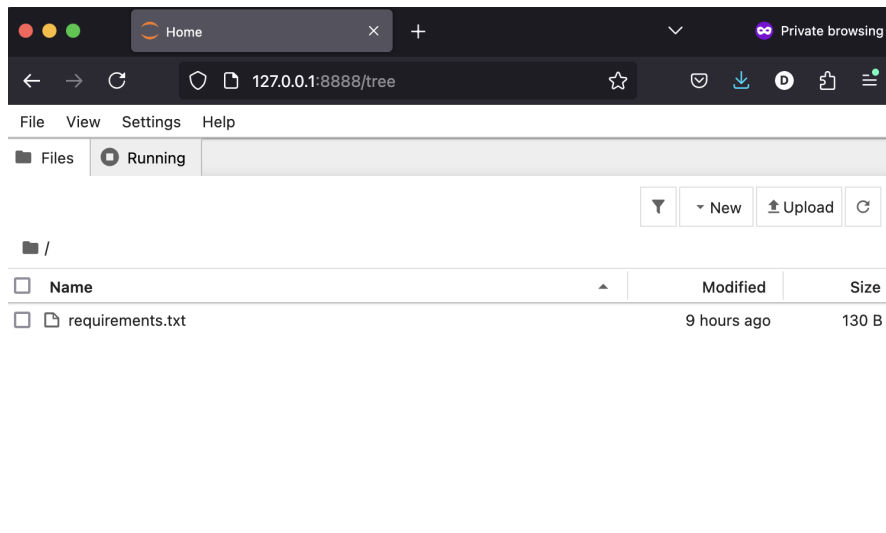


Figura 6: Interfaz principal de Jupyter al acceder desde la máquina anfitriona.

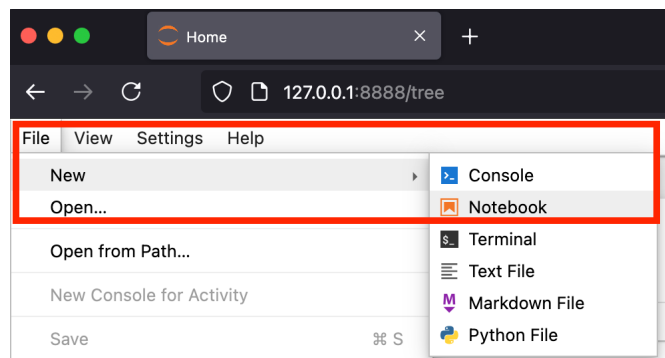


Figura 7: Creación de un nuevo *notebook* en Jupyter.

- *Celdas Markdown*: permiten escribir texto formateado con Markdown, incluyendo títulos, listas, imágenes y ecuaciones matemáticas en $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$.
- *Celdas de salida*: se generan automáticamente bajo cada celda de código y muestran su salida (texto, gráficos, datos tabulares).

1.6.3. Probar el entorno

En la primera celda del *notebook* recién creado, escribe el siguiente código de prueba:

```
1 | print("Hola, mundo desde Python")
```

Al ejecutar la celda con `⇧ + ↵`, el mensaje `Hola, mundo desde Python` aparece justo debajo (Figura 9), lo que **confirma que el entorno está funcionando correctamente**. A partir de aquí, ya puedes empezar a programar.

1. Instalación del entorno de trabajo: Jupyter Notebook (en Docker) y Visual Studio Code

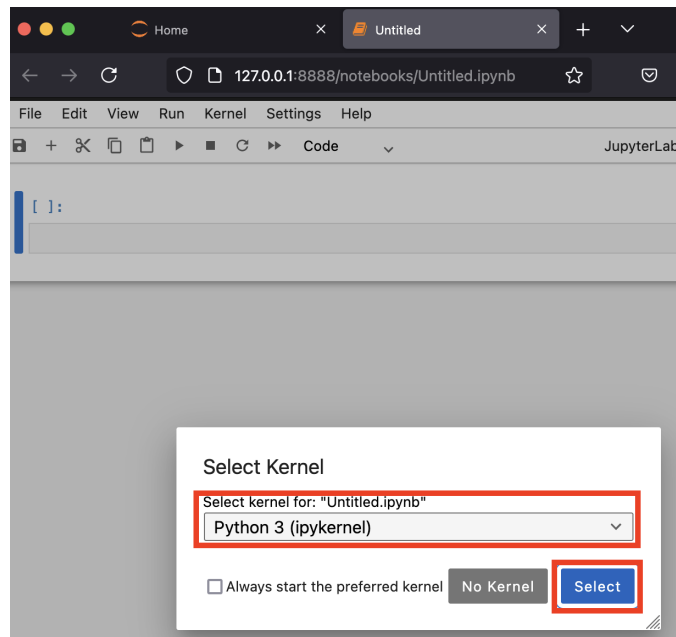


Figura 8: Selección de *kernel* en Jupyter.

1.7. Visual Studio Code como entorno alternativo

Visual Studio Code (VS Code) es un *entorno de desarrollo integrado* (en inglés, *Integrated Development Environment*, o IDE) ligero, gratuito y altamente configurable, que permite programar en múltiples lenguajes (incluido Python) y que puede usarse para todo el contenido de este libro. A diferencia de Jupyter, que se centra en la ejecución interactiva por celdas y el análisis exploratorio, VS Code ofrece una experiencia de desarrollo más estructurada con depurador, control de versiones integrado con Git, resaltado de sintaxis, autocompletado inteligente y soporte para entornos virtuales.

Mientras Jupyter resulta ideal para pruebas rápidas y experimentación con datos, VS Code permite organizar proyectos con varios ficheros y trabajar con ellos de manera eficiente. Además, gracias a la extensión de Jupyter, es posible combinar lo mejor de ambos mundos: ejecutar *notebooks* dentro de VS Code, sin necesidad de abrir el navegador.

1.7.1. Instalación

Descarga VS Code desde su página oficial e instálalo: <https://code.visualstudio.com/download>.

Una vez instalado, abre la aplicación. Para poder ejecutar *notebooks* dentro de VS Code es necesario instalar la extensión de Jupyter. En la barra lateral izquierda, haz clic en el icono de **Extensions** (Figura 10), busca **Jupyter** e instala la extensión oficial de Microsoft (Figura 11).

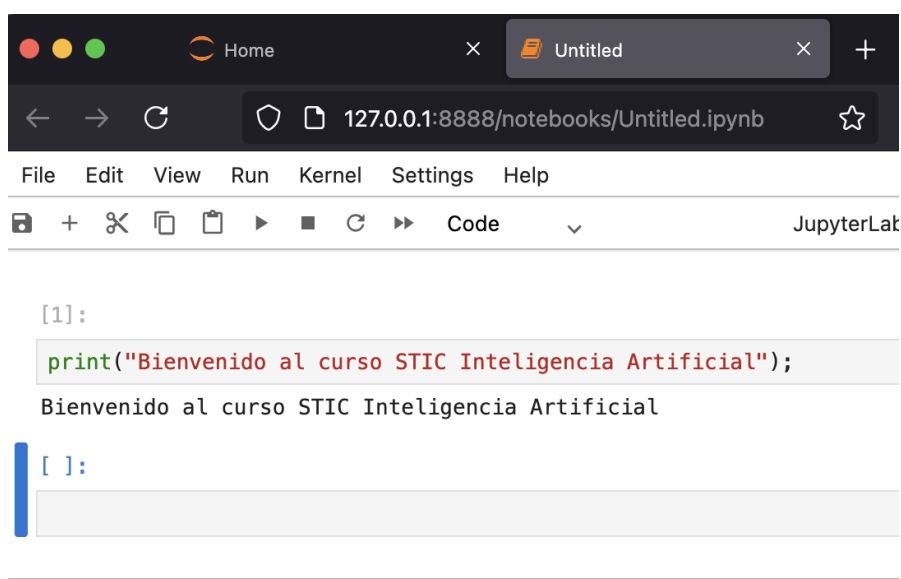


Figura 9: Ejecución del primer programa Python en Jupyter desde el navegador.

1.7.2. Conectar VS Code al servidor Jupyter del contenedor

Una vez instalada la extensión, crea un nuevo fichero llamado «prueba.ipynb» (la extensión `.ipynb` corresponde a los *notebooks* de Jupyter). Para ejecutarlo, antes hay que seleccionar un *kernel*: la opción se encuentra en la parte derecha de la barra superior del fichero recién creado (Figura 12).

Si VS Code detecta que no tienes la extensión de Python instalada (Figura 13), instálala: la necesita para gestionar los *kernels*.

Como Jupyter está corriendo dentro del contenedor Docker, en lugar de usar un *kernel* local nos conectamos al servidor remoto. Selecciona la opción `Existing Jupyter Server` (Figura 14) e introduce la URL del servidor que se está ejecutando en el contenedor (por ejemplo, `http://127.0.0.1:8888/lab?token=tu_token`). Pulsa `↵` para confirmar.

Tras conectar (VS Code preguntará por un nombre identificativo del servidor; puedes poner el que prefieras), selecciona el *kernel* Python 3 (`ipykernel`) que se ejecutará dentro del contenedor (Figura 15).

A partir de aquí ya puedes programar y ejecutar código Python desde VS Code conectado al contenedor. En la primera celda escribe el código de prueba:

```
1 | print("Hola, mundo desde Python")
```

Pulsa `⇧+↵` (ejecuta y crea una nueva celda debajo) o el botón de ejecutar celda (sólo ejecuta la actual). El otro botón de la barra superior del fichero ejecuta **todas** las celdas del *notebook* de una vez.

Si todo está correctamente configurado, el mensaje `Hola, mundo desde Python` se imprime justo debajo de la celda (Figura 16). Llegado este punto, dispones de VS Code

1. Instalación del entorno de trabajo: Jupyter Notebook (en Docker) y Visual Studio Code

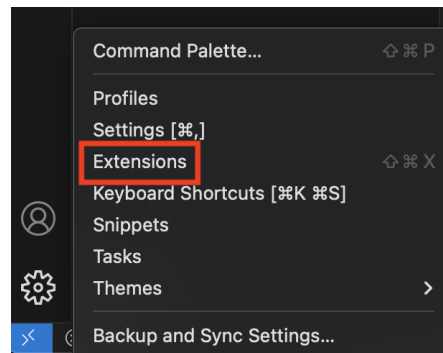


Figura 10: Pestaña `Extensions` de VS Code en la barra lateral izquierda.

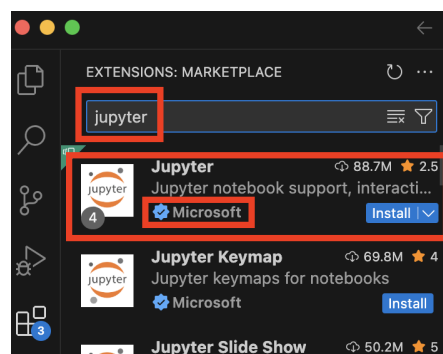


Figura 11: Instalación de la extensión `Jupyter` en VS Code.

conectado a un servidor Jupyter en Docker: puedes aprovechar el editor y el depurador de VS Code mientras trabajas con *notebooks* dentro de un contenedor aislado.

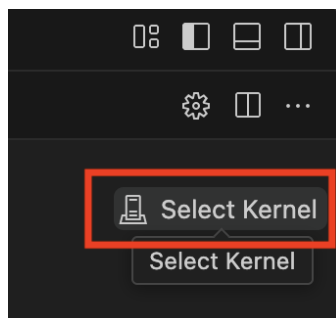


Figura 12: Selector de *kernel* en la parte derecha de la barra superior del fichero «.ipynb» en VS Code.

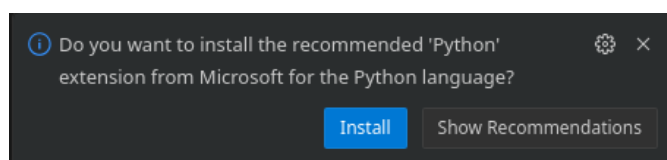


Figura 13: Sugerencia de VS Code para instalar la extensión de Python.

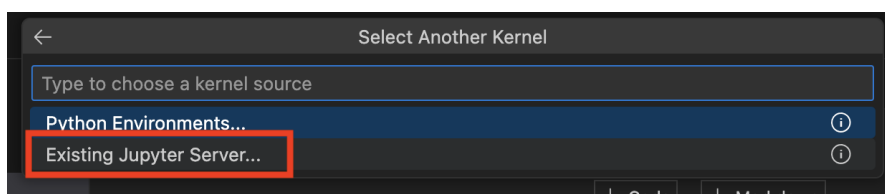


Figura 14: Conexión a un servidor Jupyter existente desde VS Code.

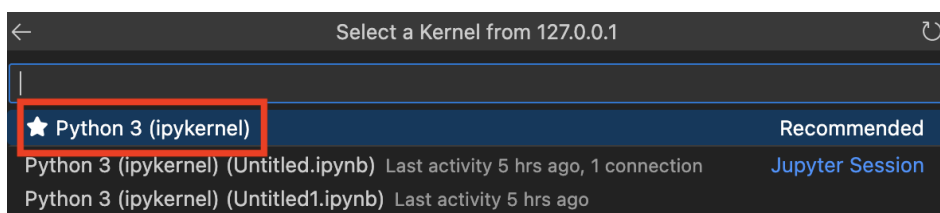


Figura 15: Selección del *kernel* en el servidor Jupyter remoto.

1. Instalación del entorno de trabajo: Jupyter Notebook (en Docker) y Visual Studio Code

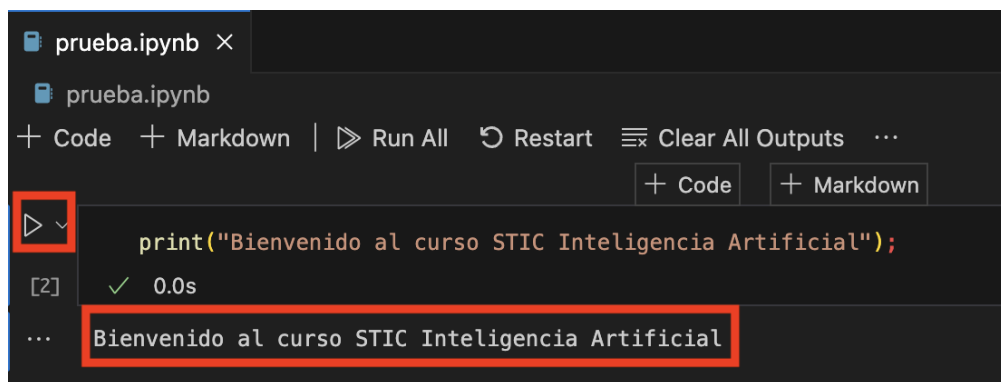


Figura 16: Ejecución del primer programa Python desde un *notebook* en VS Code conectado al contenedor.