Talleres

- Taller GIT
- UV: An extremely fast Python package and project manager, written in Rust.
- Taller Docker
- Taller Entornos de Ejecución

Taller GIT

□□ Integrantes:

- □□ Jorge Andrey Garcia
- □□ Miguel Pimiento

□□ Material de apoyo:

- □□ **Diapositivas:** Ver presentaciones

□□ Referencias

□□ □□ Documentación de Git

□□ □□ Libro Git pro

UV: An extremely fast Python package and project manager, written in Rust.

□□ Integrantes:

- □□ Fabián Pérez
- □□ Redactor: Juan Calderón

□□ Material de apoyo:

- □□ **Diapositivas:** Ver presentacion

Introducción: ¿Por qué deberías cambiar tu flujo de trabajo de Python?

Si trabajas con **Python**, sabes que los **entornos virtuales** son la columna vertebral de cualquier proyecto serio. Nos permiten aislar las dependencias y evitar el temido "infierno de dependencias" que surge al trabajar en múltiples proyectos con diferentes requisitos.

Tradicionalmente, hemos dependido de `venv` para crear el entorno y de `pip` para instalar los paquetes. Pero, ¿y si te dijera que existe una herramienta **hasta 10 veces más rápida** que puede manejar ambos procesos de forma integrada y moderna?

Conoce a **`uv`**, la nueva herramienta de gestión de paquetes y entornos que está revolucionando la comunidad Python. En este tutorial, aprenderás paso a paso a instalar `uv`, crear entornos virtuales y gestionar dependencias de forma relámpago.

⚙ I. Preparación: Instalando `uv`

Antes de empezar a volar con la gestión de entornos, necesitamos instalar la herramienta `uv`.

Requisitos Previos

Asegúrate de tener una versión reciente de **Python** instalada en tu sistema.

1. Opción Recomendada: Instalación del Binario (Ultrarrápida)

Esta es la forma más rápida y estable de instalar el binario de `uv` directamente en tu sistema:

Para Linux y macOS (usando `curl`):

```
$ curl -LsSf https://astral.sh/uv/install.sh | sh
```

Para Windows (usando PowerShell):

```
$ powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

2. Opción Alternativa: Instalación como Paquete Python

Si prefieres instalarlo a través de Python (por ejemplo, dentro de otro entorno virtual o usando `pipx`), también es posible:

```
$ pip install uv
```

Para verificar que la instalación fue exitosa, ejecuta:

```
$ uv --version
```

□□ II. Crear el Entorno Virtual: `uv venv` vs. `uv init`

`uv` ofrece dos comandos potentes para empezar tu proyecto. Ambos son mucho más rápidos que el método tradicional.

Opción A: Creación Pura del Entorno con `uv venv` (Reemplazo directo de `python -m venv`)

Este comando es ideal si solo quieres la carpeta `.venv/` en un directorio existente:

```
$ mkdir mi-proyecto-rapido
$ cd mi-proyecto-rapido
$ uv venv
```

Opción B: Inicialización Completa con `uv init` (Recomendado para Nuevos

Proyectos)

Este comando es más amplio y genera archivos clave para la configuración moderna de Python:

```
$ mkdir mi-proyecto-nuevo
$ cd mi-proyecto-nuevo
$ uv init
# Esto crea: .venv/, pyproject.toml, .gitignore, y más.
```

Activación del Entorno

Una vez creado (con cualquiera de los comandos anteriores), debes **activar** el entorno virtual:

Sistema Operativo	Comando de Activación
Linux/macOS	`$ source .venv/bin/activate`
Windows (CMD)	`$ .venv\Scripts\activate`
Windows (PowerShell)	`$ .venv\Scripts\Activate.ps1`

Verás el nombre del entorno (`.venv`) aparecer al inicio de tu línea de comandos, indicando que está activo.

□□ III. Instalación de Herramientas de Deep Learning con `uv add`

Aquí es donde `uv` brilla, permitiendo instalar librerías complejas como **PyTorch** de manera ultrarrápida.

1. Instalación de NumPy y PyTorch

Para ejecutar un proyecto de *Deep Learning*, instalaremos la base (**NumPy**) y el framework (**PyTorch**) junto con librerías de soporte (**torchvision** y **torchaudio**).

a. Instalación de NumPy:

```
$ uv add numpy
```

b. Instalación de PyTorch (con soporte CUDA/GPU para rendimiento):

```
# Reemplaza 'cu121' con la versión de CUDA instalada en tu sistema (ej. cu118, cu121, etc.)
$ uv add torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
```

> **Nota:** La URL del índice es crucial para obtener la versión correcta y optimizada para la GPU.

2. Eliminación de Paquetes

Para desinstalar un paquete (ej. si decides cambiar de PyTorch a TensorFlow):

```
$ uv remove torch
```

□□ IV. Ejecutando Herramientas Comunes con `uv`

`uv` no solo instala paquetes; también puede ejecutar herramientas directamente desde el entorno virtual sin instalarlas globalmente, usando `uv run` o `uvx`.

Herramienta	**Propósito**	**Comando `uv`**
Jupyter Notebook	Entorno interactivo para desarrollo de ciencia de datos.	`$ uv run jupyter notebook`
Ruff	Linter y formateador de código Python de alto rendimiento.	`$ uv run ruff check .`
pycowsay	Una librería de prueba divertida para verificar la ejecución.	`$ uv run pycowsay "hola soy Jorge"`

⌨ V. Más comandos de `uv`

Para una gestión completa del entorno virtual, aquí tienes comandos adicionales útiles:

Comando	Propósito
`$ uv tree`	Muestra el **árbol de dependencias** del proyecto de forma jerárquica.
`$ uv python list`	Lista todas las **instalaciones de Python** disponibles en tu sistema.
`$ uv add --dev ruff`	Agrega un paquete (ej. Ruff) específicamente como **dependencia de desarrollo** (no requerida en producción).
`$ uv run python`	Inicia el intérprete **REPL (shell) de Python** dentro del entorno activo.
`$ uv sync`	**Sincroniza** el entorno virtual con el archivo de bloqueo (lock file) del proyecto, asegurando que las versiones sean exactas.
`$ uv self update`	**Actualiza** el ejecutable de `uv` a su última versión.
`$ uv run --env-file .env app.py`	Ejecuta un script de Python (`app.py`) **cargando variables de entorno** desde un archivo `.env`.

□□ VI. Desactivación y Conclusión

Desactivación del Entorno

Cuando termines de trabajar, desactiva el entorno escribiendo `deactivate`:

```
$ deactivate
```

Conclusión

`uv` no es solo una herramienta más rápida; es un paso hacia la modernización de todo el ecosistema de dependencias de Python. Al integrar la creación del entorno (`venv`) con la gestión de paquetes (`pip`), ofrece un flujo de trabajo más limpio, consistente y eficiente.

Si valoras tu tiempo y buscas optimizar tus procesos de desarrollo en Python, **`uv` es la herramienta que necesitas implementar hoy mismo.**

□□ Referencias

- **Instalación** de `uv` e **instrucciones de uso**
- Comunidad de **discord** de `uv`
- Repositorio de **GitHub** de `uv`

Taller Docker

□□ Integrantes:

- □□ Daniel Sarmiento
- □□ Redactor: Jorge Garcia

□□ Material de apoyo:

- □□ **Diapositivas:** Ver presentaciones
- Repositorio

Primera imagen Segunda imagen

Taller docker

Uno de los problemas actuales es como compartir modelos, con sus configuraciones y dependencias, a otras personas. Una manera de hacer esto es con docker.

Docker es una plataforma de contenedorización que permite ejecutar aplicaciones dentro de entornos aislados denominados *containers*.
Cada contenedor incluye el código, las dependencias y configuraciones necesarias, asegurando que el comportamiento de la aplicación sea idéntico sin importar el sistema donde se ejecute.

Aunque a menudo se compara con una máquina virtual (VM), Docker no virtualiza hardware. En cambio, utiliza el *kernel* del sistema operativo anfitrión junto con tecnologías como *namespaces* y *cgroups* para aislar procesos, lo que lo hace mucho más liviano y rápido.

Comparativa de Container vs Virtual Machine

Comparativa de container vs virtual machine

En la ilustración se observa que las VMs requieren un sistema operativo completo dentro del host, mientras que los contenedores comparten el mismo kernel. Esto permite que un contenedor arranque en segundos y consuma una fracción de los recursos.

Instalación

Para la instalación, en linux se pueden utilizar sus paqueterías oficiales.

Linux

Arch Linux / Manjaro

```
sudo pacman -S docker
sudo systemctl enable --now docker
```

Ubuntu / Debian

```
sudo apt update
sudo apt install docker.io -y
sudo systemctl enable --now docker
```

Fedora

```
sudo dnf install docker -y
sudo systemctl enable --now docker
```

Una vez instalado, puedes verificar su funcionamiento con:

```
docker run hello-world
```

Windows y macOS

Docker ofrece una aplicación oficial llamada **Docker Desktop**, que integra todas las herramientas necesarias. Puede descargarse desde el sitio oficial: https://www.docker.com/products/docker-desktop

---

Comandos básicos

Comando	Descripción

`docker ps -a`	Muestra todos los contenedores (incluidos los detenidos)
`docker pull <imagen>`	Descarga una imagen desde un registro (por ejemplo, Docker Hub)
`docker run <imagen>`	Ejecuta una imagen en un nuevo contenedor
`docker images`	Lista las imágenes disponibles en el sistema
`docker rm <id>`	Elimina un contenedor
`docker rmi <id>`	Elimina una imagen
`docker exec -it <nombre> bash`	Abre una sesión interactiva dentro de un contenedor

Dockerfile

Para construir una imagen personalizada, se utiliza un archivo llamado `Dockerfile`. Este define los pasos necesarios para preparar el entorno de ejecución.

Ejemplo simple:

```
FROM docker.io/postgres:latest  # Imagen base
RUN apt update && apt upgrade -y
ENV POSTGRES_USER=user \
    POSTGRES_PASSWORD=password \
    POSTGRES_DB=exampledb
```

Cada instrucción genera una **capa (layer)**, lo que permite que las reconstrucciones sean más rápidas y eficientes.

Contenerizando un Modelo de IA con PyTorch

Imaginemos que queremos crear un contenedor para entrenar un modelo convolucional sencillo en el dataset CIFAR-10. El modelo en PyTorch podría ser el siguiente:

```
import torch.nn as nn
import torch.nn.functional as F
import torch
```

```
class Net(nn.Module):
    def __init__(self, dropout_rate=0.3):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
        self.dropout = nn.Dropout(dropout_rate)
        self._initialize_weights()

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, (nn.Conv2d, nn.Linear)):
                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
                if m.bias is not None:
                    nn.init.constant_(m.bias, 0)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = self.dropout(F.relu(self.fc2(x)))
        x = self.fc3(x)
        return x
```

Para contenerizarlo, creamos un `Dockerfile` como este:

```
# Imagen base ligera con Python
FROM python:3.12-slim

# Directorio de trabajo
WORKDIR /app

# Copiar el código fuente
COPY . /app
```

```
# Instalar dependencias (CPU-only)
RUN pip install torch torchvision --index-url https://download.pytorch.org/whl/cpu
RUN pip install matplotlib scikit-learn albumentations tqdm tensorboard

# Instalar dependencias adicionales (si existen)
RUN if [ -f requirements.txt ]; then pip install -r requirements.txt; fi

# Variables de entorno
ENV PYTHONUNBUFFERED=1
ENV PYTHONDONTWRITEBYTECODE=1

# Comando por defecto
CMD ["python", "train.py"]
```

> □□ Si cuentas con GPU y drivers CUDA, puedes usar una imagen base con soporte GPU, como:
>
> ```
> FROM pytorch/pytorch:2.2.0-cuda12.1-cudnn8-runtime
> ```
>
> e instalando las dependencias con la versión de cuda, es decir, la predeterminada
>
> ```
> RUN pip install torch torchvision
> ```

Orquestación y Docker Compose

En proyectos de IA es habitual tener múltiples tareas: entrenamiento, evaluación y monitoreo. Con **Docker Compose** podemos definir todos los servicios en un solo archivo `docker-compose.yml`:

```
services:
  train:
    build:
      context: .
      dockerfile: Dockerfile
```

```
    image: pytorch-uv-app:latest
    container_name: pytorch-train
    command: python train.py
    volumes:
      - ./data:/app/data
      - ./model:/app/model
      - ./runs:/app/runs
      - ./checkpoints:/app/checkpoints
    restart: "no"

  test:
    image: pytorch-uv-app:latest
    container_name: pytorch-test
    command: python test.py
    volumes:
      - ./data:/app/data
      - ./model:/app/model
      - ./runs:/app/runs
      - ./checkpoints:/app/checkpoints
    restart: "no"

  tensorboard:
    image: python:3.12-slim
    container_name: pytorch-tensorboard
    command: >
      sh -c "pip install --quiet tensorboard>=2.20.0 &&
             tensorboard --logdir=/app/runs --host=0.0.0.0 --port=6006"
    ports:
      - "6006:6006"
    volumes:
      - ./runs:/app/runs
    restart: unless-stopped
    profiles:
      - monitoring
```

Ejecución

Ejecutar el entrenamiento:

```
docker compose up train
```

Ejecutar las pruebas:

```
docker compose up test
```

Iniciar TensorBoard para monitoreo:

```
docker compose --profile monitoring up tensorboard
```

Luego, accede a http://localhost:6006 para visualizar las métricas.

— Jorge García

> **Recursos adicionales:**
>
> - Documentación oficial de Docker
> - Imágenes oficiales de PyTorch
> - Guía de Docker Compose

Taller Entornos de Ejecución

□□ Integrantes:

- □□ Guillermo Pinto
- □□ Brayan Yesid Quintero Santander

□□ Material de apoyo:

- □□ **Diapositivas - Do no just use Colab:** Ver presentación
- □□ **Notebooks - Hugging Face and Kaggle Datasets**
  - Colab version: Ver notebook
  - Kaggle version: Ver notebook

Entornos de ejecución

Los entornos de ejecución son las plataformas donde podemos ejecutar nuestros experimentos de manera gratuita. Generalmente, solo se conoce Google Colaboratory para este fin, sin embargo, actualmente existen diferentes alternativas, unas mejores y otras no tanto, que nos permiten sacar adelante alguna idea.

En este taller revisamos cada una de las alternativas disponibles en Colombia en el momento. Vimos cómo usarlas, las comparamos y definimos un *tier list* de cuáles, desde la perspectiva de Guille, son las mejores.

Al final revisamos cómo hacer uso de los *secretos*, los cuales son claves como *tokens* o *api keys* que nos permiten acceder a contenido privado de manera segura. Todo esto lo encuentras en las diapositivas ***Do not just use Colab***.

Datasets

Cuando estamos trabajando en un proyecto necesitamos datos. Pero más importante aún, necesitamos un lugar donde guardar estos datos de manera que podamos accederlos sin muchas complicaciones.

Entre las mejores alternativas encontramos *Hugging Face Datasets* y *Kaggle Datasets*. Dos plataformas donde podemos alojar nuestros datos de manera gratuita y luego accederlos, desde cualquier lugar, a través de los secretos (*api keys* o *tokens*).

¿Cómo subir un dataset?

Por último, se subió un **dataset de manera privada** siguiendo la estructura recomendada por *Hugging Face Datasets*, el conjunto de datos quedó organizado con la siguiente estructura:

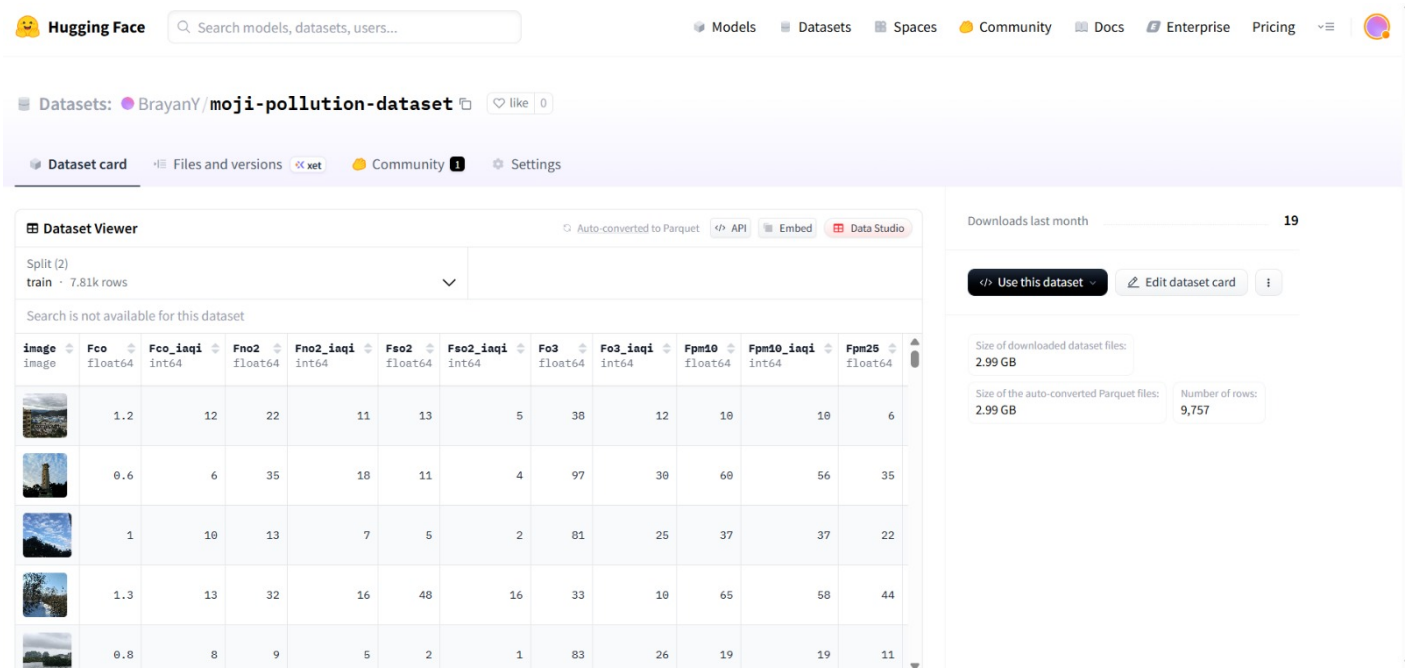

image image	Fco float64	Fco_iaqi int64	Fno2 float64	Fno2_iaqi int64	Fso2 float64	Fso2_iaqi int64	Fo3 float64	Fo3_iaqi int64	Fpm10 float64	Fpm10_iaqi int64	Fpm25 float64
	1.2	12	22	11	13	5	38	12	10	10	6
	0.6	6	35	18	11	4	97	30	60	56	35
	1	10	13	7	5	2	81	25	37	37	22
	1.3	13	32	16	48	16	33	10	65	58	44
	0.8	8	9	5	2	1	83	26	19	19	11

Esta estructura segura que cada subconjunto (train y test) tenga sus propias imágenes y metadatos, donde cada *metadata.csv* contenga únicamente la información correspondiente a cada subconjunto. Puedes descubrir de manera práctica cómo hacerlo en los dos notebooks que encuentras en el material de apoyo.

— Guillermo Pinto y Brayan Quintero

> **Recursos adicionales**
>
> - “Lightning AI | Turn Ideas Into AI, Lightning Fast.” Lightning AI, lightning.ai.
> - Kaggle: Your Machine Learning and Data Science Community. www.kaggle.com
> - SageMaker Studio Lab. studiolab.sagemaker.aws.

- “GitHub Codespaces.” GitHub, 2025, github.com/features/codespaces.
- “Modal: High-performance AI Infrastructure.” Modal, modal.com.
- AICodeKing. “These Are the Best Google Colab Alternatives! (Free Tiers With GPUs).” YouTube, 11 May 2024, www.youtube.com/watch?v=yvvNtkfJhGI.
- Datasets. huggingface.co/docs/datasets/en/index.
- Datasets Documentation. www.kaggle.com/docs/datasets.