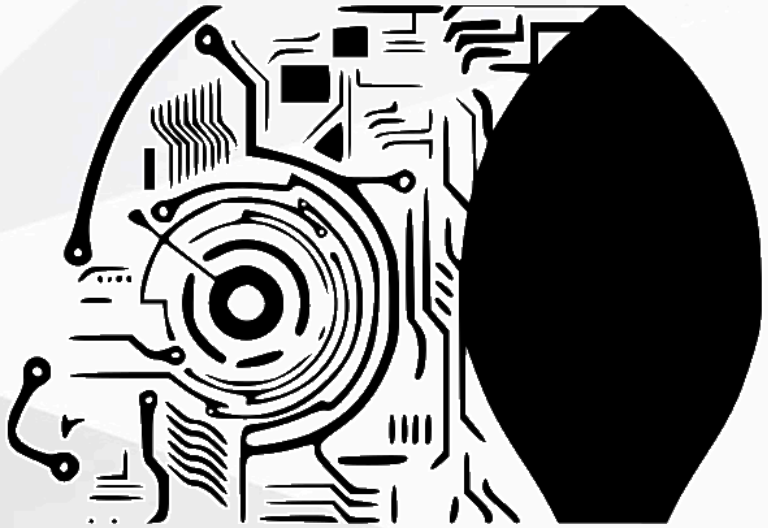


MODELOS DE I.A.

https://martinezpenya.es/ModelosIA/

UD00: Presentación y curso rápido de Docker

Modelos de Inteligencia Artificial

version: 2026-08-24

IES EDUARDO PRIMO MARQUÉS

¿Qué veremos?

1. El curso, el módulo y cómo se evalúa
2. El calendario del curso
3. Por qué los contenedores
4. Docker: imágenes, contenedores y volúmenes
5. Dockerfile y Compose
6. El entorno de prácticas del curso

El módulo 5071

Código	**5071** · Modelos de Inteligencia Artificial
Duración	**90 h** · 3 h/semana · 4 ECTS
Curso	Especialización en **IA y Big Data**
Resultados de aprendizaje	**6** propios (RA1-RA6) + el **proyecto intermodular** (RA7)

> Del 1 de octubre al 28 de mayo. Clases de **lunes a jueves**: los viernes son para tareas y talleres.

d.martinezpena@edu.gva.es

Esta unidad no desarrolla un RA propio

El currículo desarrolla **6 RA** para el módulo (RA1-RA6). La UD00 es la unidad de **presentación y puesta en marcha** que prepara el terreno: sin ella, los RA1-RA6 perderían horas configurando entornos.

Lo que sí trabaja se asocia al **RA7-i** (ética y cuidado del entorno de trabajo) y a los **hábitos de trabajo** del curso.

Al terminar la unidad serás capaz de...

- **Describir** el curso de especialización y el papel del módulo 5071 en él.
- **Explicar** cómo se evalúa el módulo: criterios, calificación y recuperación.
- **Identificar** el entorno del curso: Aules, la web, Python, Jupyter y Docker.
- **Diferenciar** contenedor, imagen, volumen y red, y qué aporta Docker frente a una máquina virtual.
- **Crear, ejecutar, detener y eliminar** contenedores con `docker run`.
- **Escribir** un `Dockerfile` sencillo y **orquestar** varios servicios con Compose.
- **Levantar** el contenedor de prácticas con Jupyter y las bibliotecas del curso.

Cómo se evalúa

- Cada **RA** se califica de **1 a 10, sin decimales**
- Nota de cada RA = **40 %** tareas, talleres y ejercicios + **60 %** prueba escrita
- **Una prueba por RA** en Moodle, al cerrar cada unidad
- Hace falta **5 o más en CADA RA** para aprobar el módulo

> Puedes aprobar las dos evaluaciones y suspender el módulo por **un solo RA**.

Obligatoria y no calificable: los tres entregables

Tres entregables, marcados **hecho / no hecho**:

1. **Taller 1** · verificación del entorno → memoria en PDF
2. **Taller 2** · contenedor de prácticas → informe breve
3. **Notebook** de la unidad → con las respuestas de la actividad

> Son la prueba de que tu entorno funciona. Sin ellos no se pueden hacer las prácticas de la UD02.

El calendario, en un vistazo

Cuándo	Qué
1 – 8 oct	UD00 · presentación y Docker
12 oct – 3 dic	UD01 y UD02
7 dic – 14 ene	UD05 · sistemas expertos
18 ene – 11 mar	UD03 (PLN) y UD04 (robótica)
15 mar – 23 abr	UD06 · ética y legalidad
26 abr – 28 may	**Proyecto intermodular (RA7)**

Las interrupciones del curso

Cuándo	Qué
22 dic – 6 ene	Vacaciones de Navidad
17 y 18 mar	Fallas
25 mar – 5 abr	Vacaciones de Pascua

> **Marzo es el mes más roto del curso**: tenlo en cuenta antes de dejar una entrega para el final.

El problema: «en mi máquina funciona»

- Cada equipo tiene un Python distinto, con versiones distintas de cada biblioteca
- Un notebook que va en clase **falla en casa**, y al revés
- Reproducir un resultado ajeno se vuelve imposible
- En IA esto es más grave: los resultados **no son comparables**

> La IA se hace con **entornos reproducibles**.

Contenedor frente a máquina virtual

	Máquina virtual	Contenedor
Qué virtualiza	Hardware completo	Solo el proceso
Sistema operativo	Uno **propio** por máquina	Comparte el **kernel** del anfitrión
Arranque	Minutos	**Segundos**
Tamaño	Gigabytes	Megabytes

Lo mismo, en un dibujo

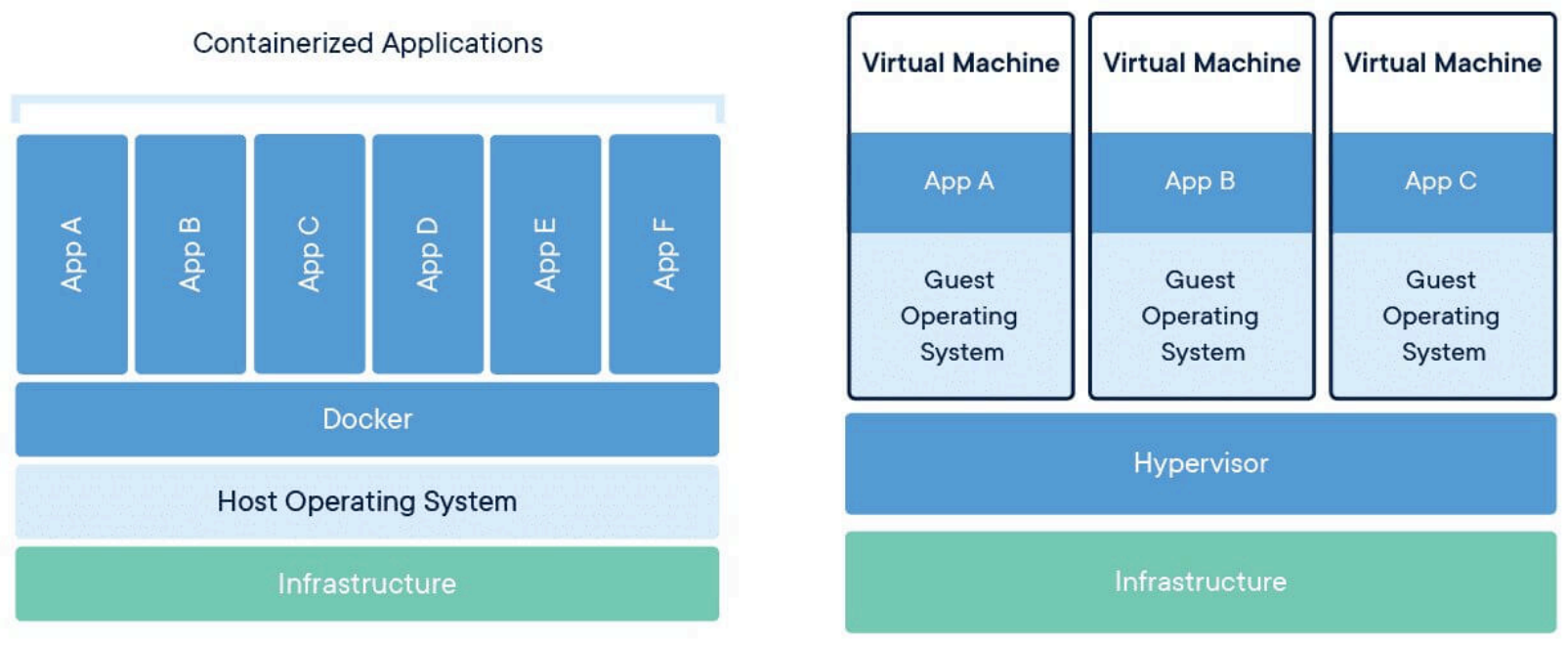


Imagen y contenedor

- **Imagen**: la plantilla, inmutable, hecha de **capas**
- **Contenedor**: una **instancia en ejecución** de esa imagen
- **Registro** (Docker Hub): donde viven las imágenes

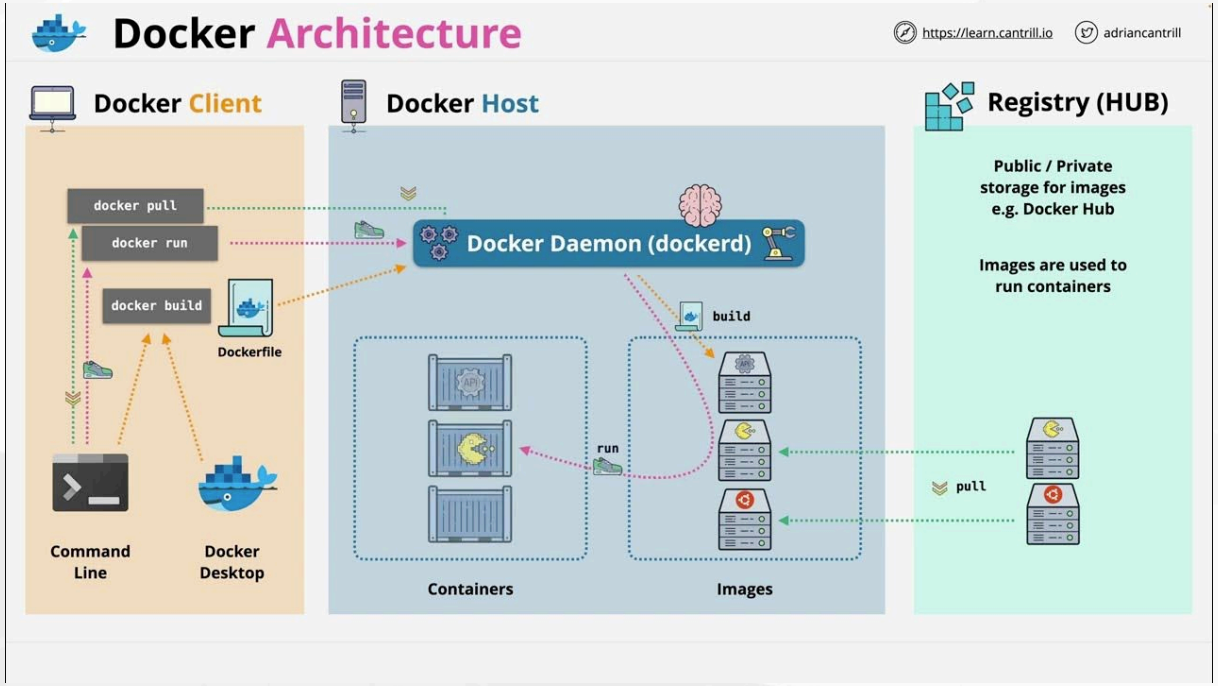
> Una imagen, muchos contenedores. Como una clase y sus objetos.

Las tres piezas de Docker

- **Cliente** (`docker`): el comando que escribes tú. Solo manda órdenes
- **Daemon** (`dockerd`): el servicio que las ejecuta — descarga imágenes, crea contenedores
- **Registro** (Docker Hub): el almacén público de imágenes

> Tú nunca hablas con el contenedor: hablas con el **daemon**. Sin permiso sobre su *socket*, nada funciona.

El recorrido completo



Instalar Docker en Linux

```
# 1. Clave GPG y repositorio oficial de Docker
# 2. Instalar el motor
sudo apt install docker-ce docker-ce-cli containerd.io docker-compose-plugin
# 3. Poder usarlo sin sudo
sudo usermod -aG docker $USER
```

> El paso 3 no surte efecto **hasta cerrar y volver a abrir la sesión**.
> Es el error del primer día.

Docker Desktop

Para **Windows y macOS** es la vía normal; en Linux es opcional.

- Incluye el motor, la interfaz gráfica y Compose
- En Windows se apoya en **WSL 2**
- En Linux se instala con el `.deb` oficial

> Comprueba siempre la instalación con `docker run hello-world` antes de seguir.

Los comandos que usarás el 90 % del tiempo

```
docker run -it --name mi-python python:3.12 bash   # crear y entrar
docker ps -a                                       # qué contenedores tengo
docker images                                      # qué imágenes tengo
docker start / stop / rm                           # ciclo de vida
docker run -d -p 8080:80 nginx                     # servicio en segundo plano
```

> `-it` interactivo · `-d` en segundo plano · `-p` publicar puerto · `--rm` borrar al salir

Lo que se acumula sin darte cuenta

```
docker images                 # ¿cuánto ocupan?
docker rmi <imagen>           # borrar una concreta
docker image prune            # borrar las huérfanas, sin etiqueta
docker image prune -a         # TODAS las que no use ningún contenedor
```

> `prune -a` decide por ti: se lleva también las imágenes que construiste y no has publicado. Mira antes con `docker images`.

Volúmenes: que los datos sobrevivan

Un contenedor es **desechable**: lo que escribes dentro desaparece al borrarlo.

```
docker run --rm -v "$PWD":/app -w /app python:3.12 python app.py
```

- **bind mount**: una carpeta tuya se ve dentro del contenedor
- **volumen nombrado**: Docker gestiona el almacenamiento

> Tus notebooks viven **en tu equipo**, no dentro del contenedor.

Copias de seguridad

Qué copias	Comandos	Conserva
Una **imagen**	`docker save` / `docker load`	Capas, etiquetas y metadatos
Un **contenedor**	`docker export` / `docker import`	Solo los ficheros, aplanados

> Con `save` puedes seguir trabajando sobre la imagen; con `export` obtienes una foto plana.

Dockerfile: la receta

```
FROM python:3.12-slim
WORKDIR /home/mia
COPY requirements-ia.txt .
RUN pip install -r requirements-ia.txt
EXPOSE 8888
CMD ["jupyter", "notebook", "--ip=0.0.0.0"]
```

> Cada instrucción es una **capa** y las capas se **cachean**: por eso las dependencias van antes que el código, para no reinstalarlas en cada cambio.

Compose: describir en vez de recordar

```yaml
services:
  mia:
    build: .
    ports:
      - "8888:8888"
    volumes:
      - ../notebooks:/home/mia
```

```bash
docker compose up -d      # levantar
docker compose down       # parar y limpiar
```

El mismo código, dos entornos

`experta` es el motor de reglas de la UD02 y la UD05. Última versión: **2019**.

```
PRUEBA="pip install -q experta && python -c 'import experta'"

docker run --rm python:3.9-slim  bash -c "$PRUEBA"    # funciona
docker run --rm python:3.12-slim bash -c "$PRUEBA"    # falla
```

- `collections.Mapping` **desapareció en Python 3.10**
- Se arregla con **tres líneas** antes del import

Tres lecciones

1. **El entorno importa**: el mismo código y dos resultados distintos
2. **Las dependencias se abandonan**: elegir una biblioteca es apostar por quien la mantiene
3. **Se puede convivir con ello**: un parche documentado y sigues adelante

> Lo que no vale es descubrirlo el día de la entrega.

Nuestro entorno de prácticas

- Un `Dockerfile` con **Python 3.12** y las bibliotecas del curso
- Un `docker-compose.yml` que publica **Jupyter** en el puerto 8888
- Un volumen para que **tus notebooks** queden en tu equipo

```
docker compose up -d      # y Jupyter en http://localhost:8888
```

Puntos clave

- El curso de especialización dura **600 h / 36 ECTS**; el módulo 5071 son **90 h** en la Comunitat Valenciana.
- El módulo desarrolla **6 RA** más el **RA7** de proyecto intermodular, con nota consensuada.
- **La normativa exige alcanzar todos los RA**; el centro lo concreta en **≥ 5 en cada uno**. Cada RA: **40 % tareas + 60 % prueba**. Se pierde la evaluación continua con más del **15 % de inasistencia**.
- Un **contenedor** es una instancia ejecutable de una **imagen**: aislamiento por procesos, ligereza y reproducibilidad.
- Los contenedores son **efímeros**: los datos se conservan con **volúmenes** o *bind mounts*.
- Un `Dockerfile` define la imagen **por capas**, y el orden de las instrucciones aprovecha la caché.

Las dos sesiones

6 h en dos semanas (1-8 de octubre), a 3 h por semana.

Semana	Contenido	Evidencia
1	Presentación del curso, del módulo y de la evaluación · instalar Docker · imagen, contenedor, registro	Docker funcionando (`docker run hello-world`)
2	`docker run` y volúmenes · Dockerfile y Compose · contenedor de prácticas de IA	**Taller 1** y el entorno levantado

> Si el horario es 2 h + 1 h, el bloque de **2 h** es el único que admite trabajo con contenedores.

¿Y ahora?

1. Instala Docker y haz el **Taller 1**
2. Levanta el entorno con el **Taller 2**
3. Ejecuta el **notebook** de la unidad y entrégalo

> En la **UD01**: sistemas de IA, dónde se aplican y qué técnicas usan.

d.martinezpena@edu.gva.es