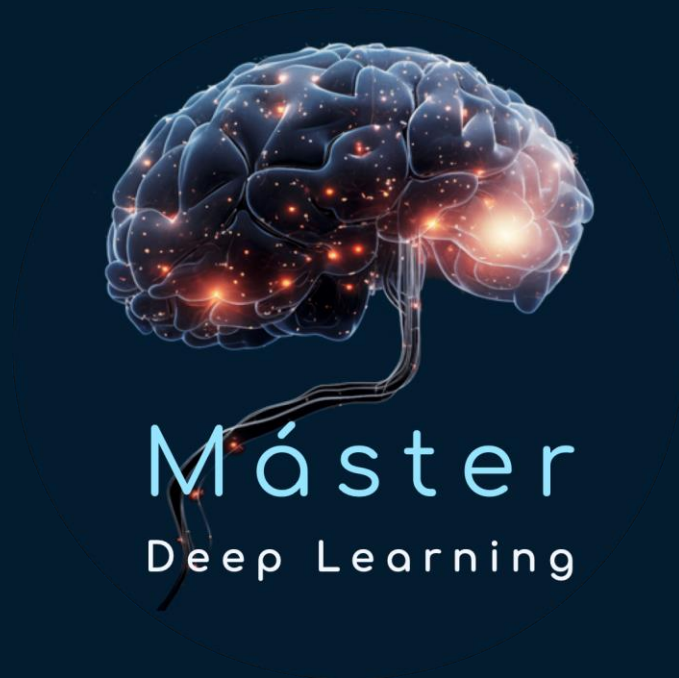


MLOps

Tema 3.

Empaquetamiento y Contenedorización



POLITÉCNICA

UNIVERSIDAD
POLITÉCNICA
DE MADRID



Máster
Deep Learning

¿Cómo recrear un entorno o instancia de aplicación?

Desde el desarrollo a la producción

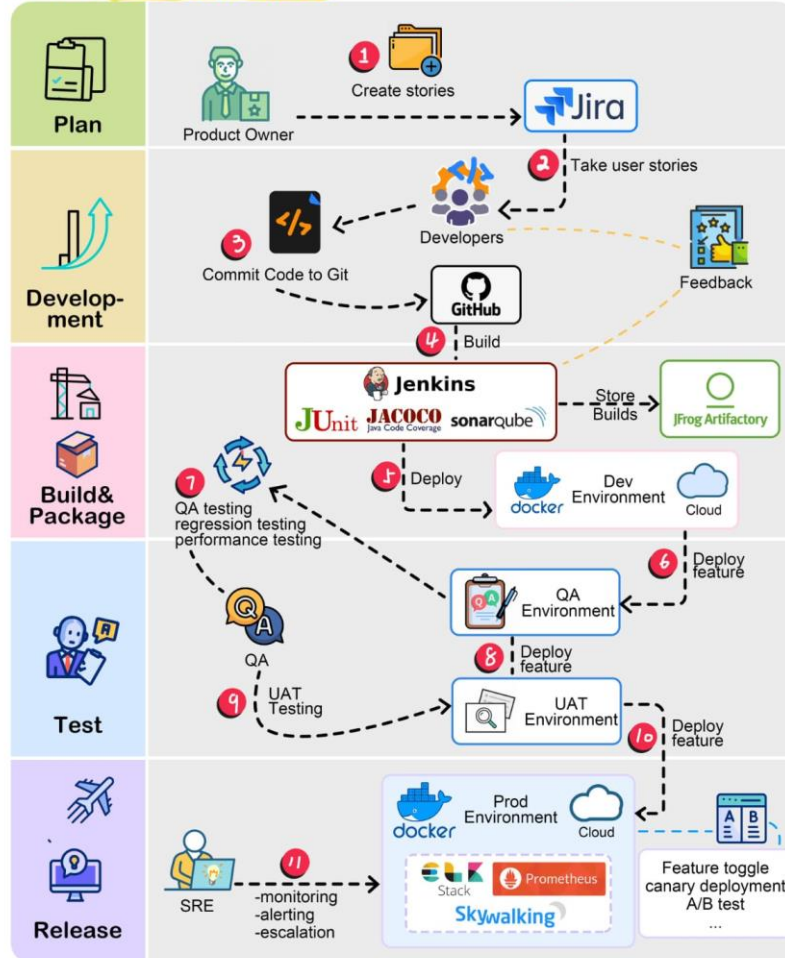
How Companies Ship Code

To Production

blog.bytebytego.com



Máster
Deep Learning



Código y Git
(Tema 2)

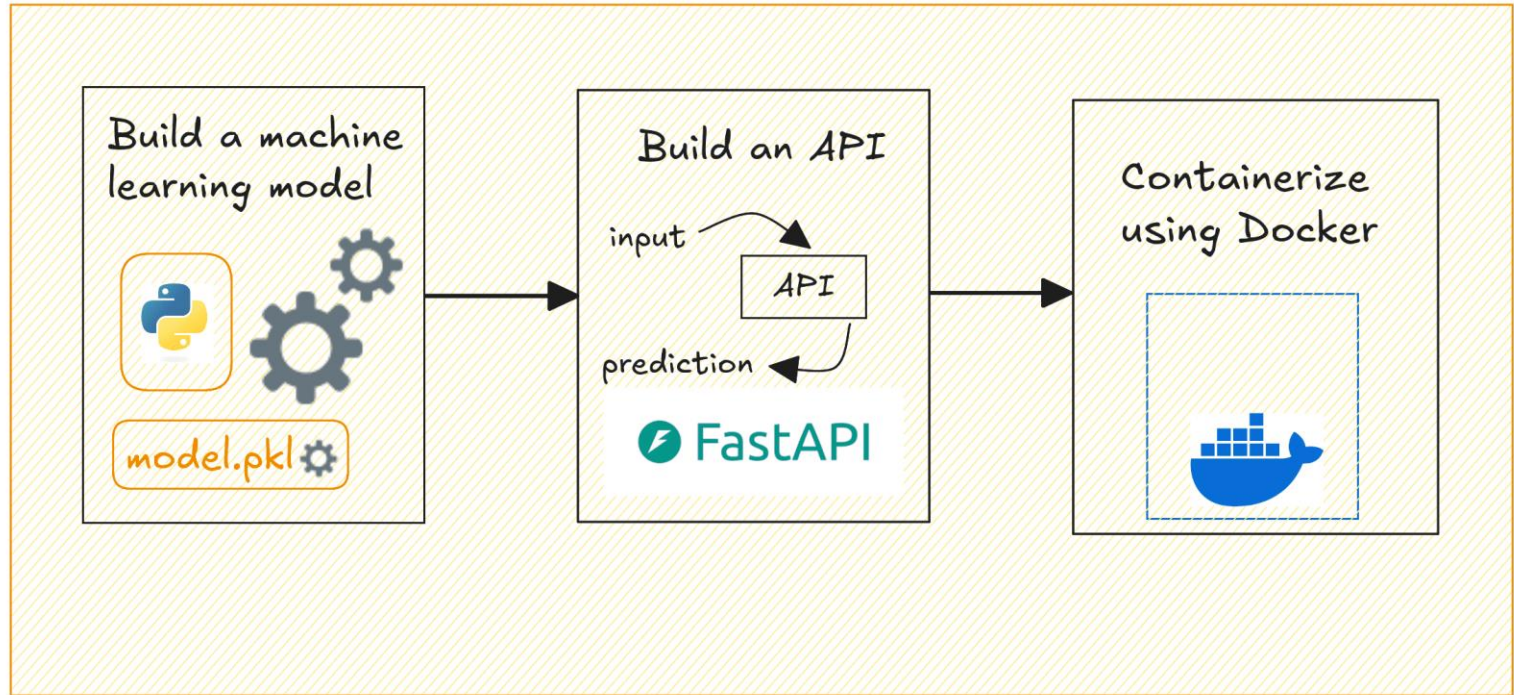
Empaquetamiento y
Contenedorización
(Tema 3)

CI/CD
(Tema 4)

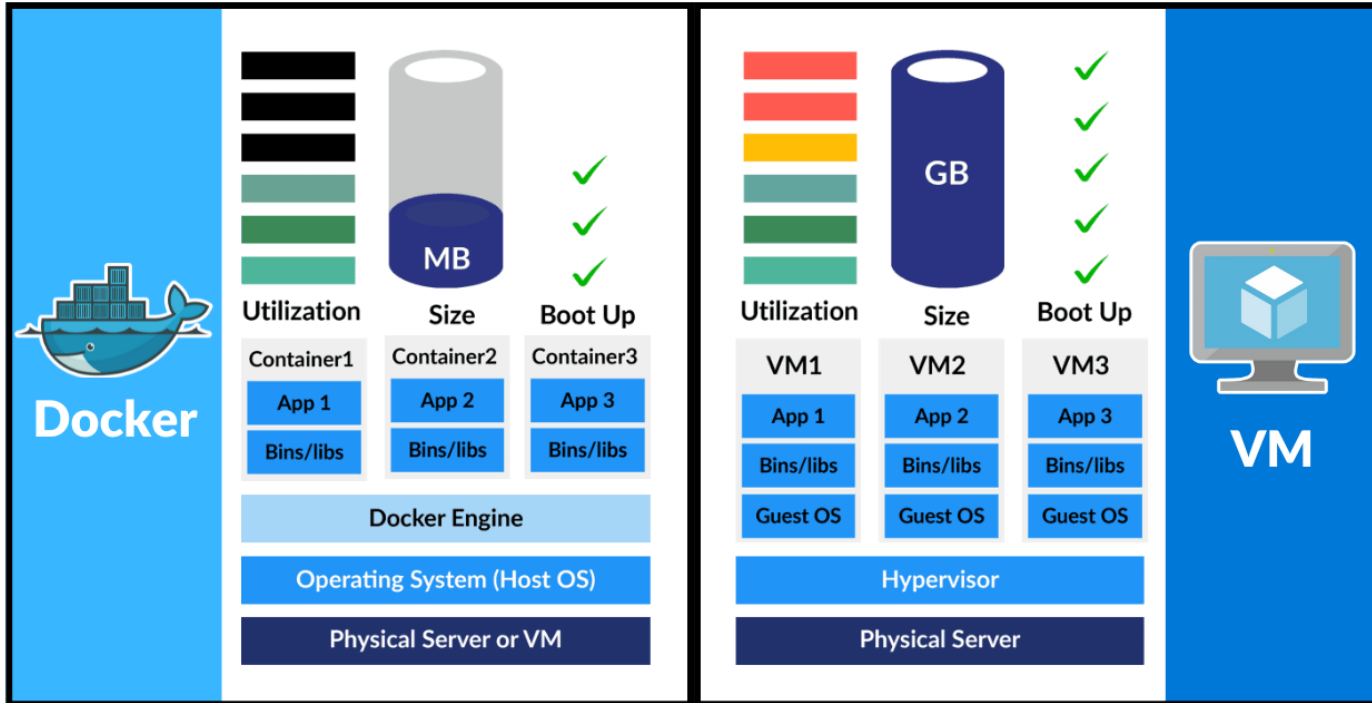
MLOps (Tema 1)

- ▶ ML suele depender de versiones (TensorFlow, scikit-learn, CUDA drivers) o requisitos software específicos
- ▶ Enfoque tradicional: documentación + requirements.txt
 - ▶ Esto puede seguir fallando en entornos diferentes
- ▶ Los **contenedores** resuelven esto empaquetando todas las dependencias de manera reproducible.

Empaquetamiento



¿Por qué contenedores?



Contenedores

- ▶ La contenedorización empaqueta código, dependencias, herramientas e instrucciones de ejecución en una **imagen**.
- ▶ **Docker** es la plataforma de contenedores más popular
 - ▶ Un **Dockerfile** es la receta con acciones para recrear un entorno.
 - ◆ El responsable es el desarrollador del contenedor
 - ▶ Un **Docker Image** es el molde final resultado de compilar la receta.
 - ◆ Puede ser reutilizado por terceros
 - ▶ Un **Docker Container** es una instancia en ejecución de la imagen.



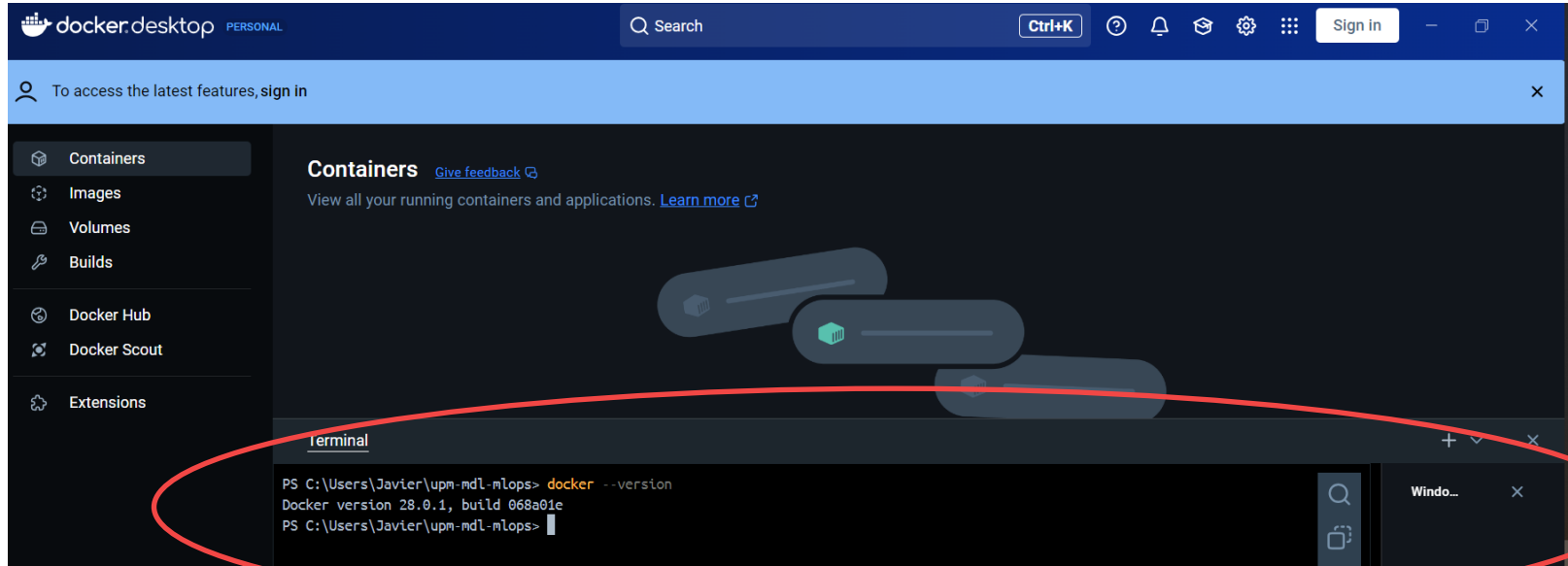
Contenedorización

Descarga de Docker

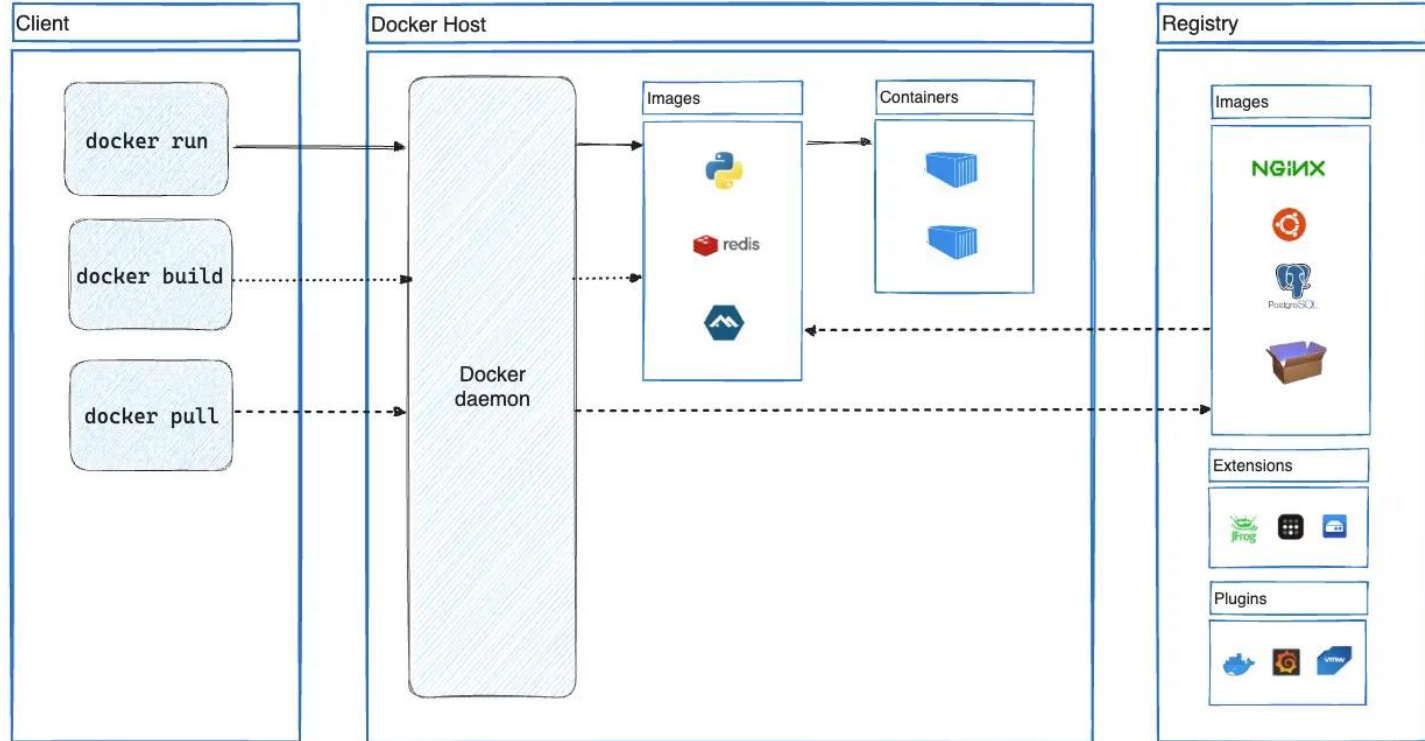


- ▶ Docker Engine en **Linux**:
 - ▶ <https://docs.docker.com/engine/install/>
- ▶ Docker Engine con Docker Desktop para **Windows, macOS o Linux**:
 - ▶ **Windows**
 - ◆ <https://docs.docker.com/desktop/setup/install/windows-install/>
 - ▶ **macOS**
 - ◆ <https://docs.docker.com/desktop/setup/install/mac-install/>
 - ▶ **Linux**
 - ◆ <https://docs.docker.com/desktop/setup/install/linux/>

Docker Desktop (UI o Terminal)



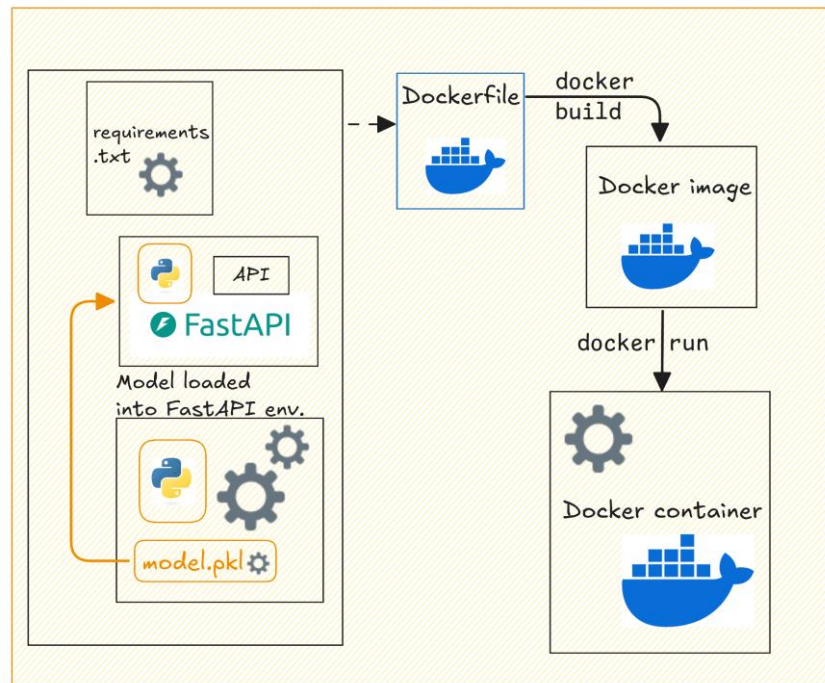
Arquitectura de Docker



Pasos de contenedorización

En un proyecto ML:

1. Crear Dockerfile para cada servicio
 - ▶ Instrucciones para recrear
 - ▶ servicio dentro de un contenedor.
2. Construir y guardar imagen
 - ▶ A partir del Dockerfile
 - ▶ Se guardar localmente o registry
 - ▶ App “empaquetada”
3. Ejecutar contenedor
 - ▶ Se ejecuta a partir de una imagen
 - ▶ Se pueden ejecutar varios contenedores a partir de una imagen

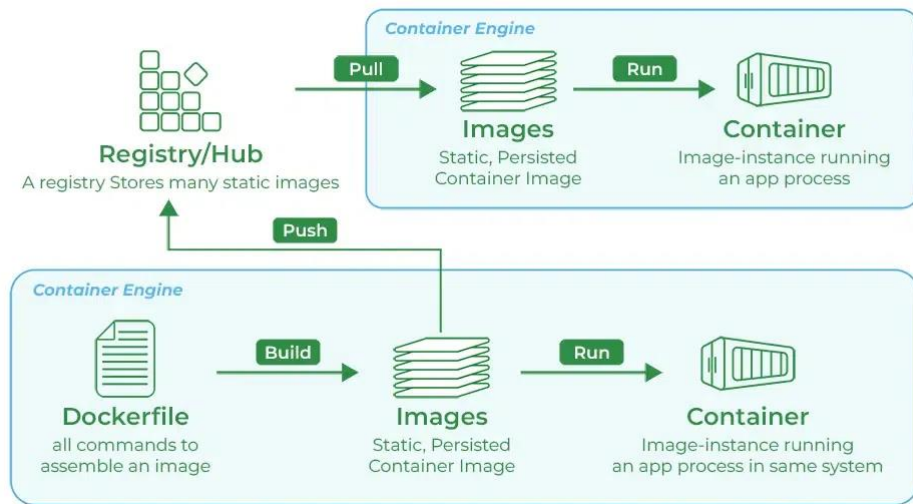


Pasos de contenedorización

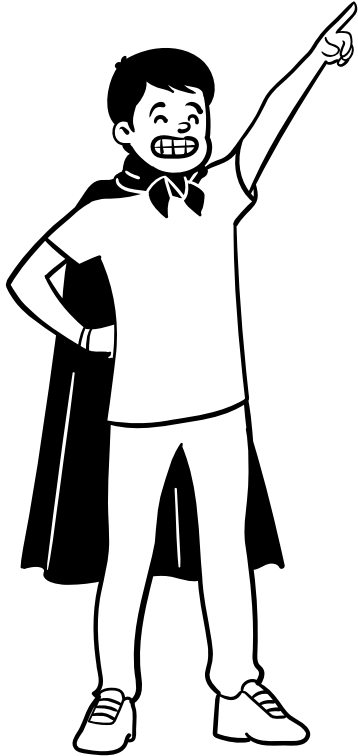


4. Compartir la imagen (opcional)

- ▶ **Registry:** repositorio remoto de imágenes que pueden ser reutilizadas
 - ◆ *Docker Hub, GitHub Container Registry, Google Container Registry, Amazon Elastic Container Registry, Azure Container Registry...*



Actualización del repositorio



Puedes descargarlo automáticamente con Git

```
$ git clone https://github.com/javier-pg/upm-dl-mlops.git
```

1. Crear Dockerfile



Fichero de instrucciones para que Docker construya la imagen

*Creamos Dockerfile
para cada servicio*

```
# Dockerfile Se suele empezar a partir de otra imagen
FROM python:3.9-slim

WORKDIR /app Establece el directorio de trabajo en /app

# Copy requirements and install them
COPY requirements.txt .
RUN pip install -r requirements.txt Instala las dependencias

# Copy the entire api folder to the container
COPY . . Copia todo el proyecto dentro

# Env variables for the API
ENV API_PORT=8000 Se definen variables de entorno

# By default, let's run the training script
CMD ["python", "inference_api.py"] Se establece comando por defecto
```

- Comandos existentes en <https://docs.docker.com/reference/dockerfile/>



1. Crear Dockerfile

```
...  
# Copy the entire project  
COPY . .  
...
```

- Evitar que algunos ficheros no se copien dentro de la imagen

***Podemos crear
.dockerignore***

```
__pycache__/  
*.pyc  
*.pyo  
*.pyd  
env/  
venv/  
.build/  
.idea/  
.DS_Store
```


2. Construir imagen

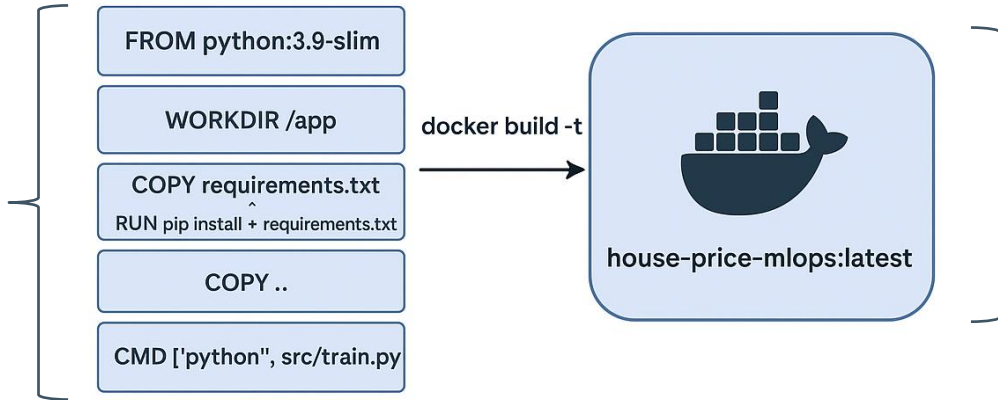
Construye imagen a partir de Dockerfile (.) y etiqueta la imagen resultante

```
$ docker build -t house-price-mlops:latest .
```

TAG *nombre imagen* *version* *dir con Dockerfile*

Dockerfile

Una imagen se crea por capas (layers), una encima de la otra



Y se guarda como objeto comprimido (capas+metadatos) en /var/lib/docker



3. Ejecutar contenedor (train)



Entrenamiento del modelo

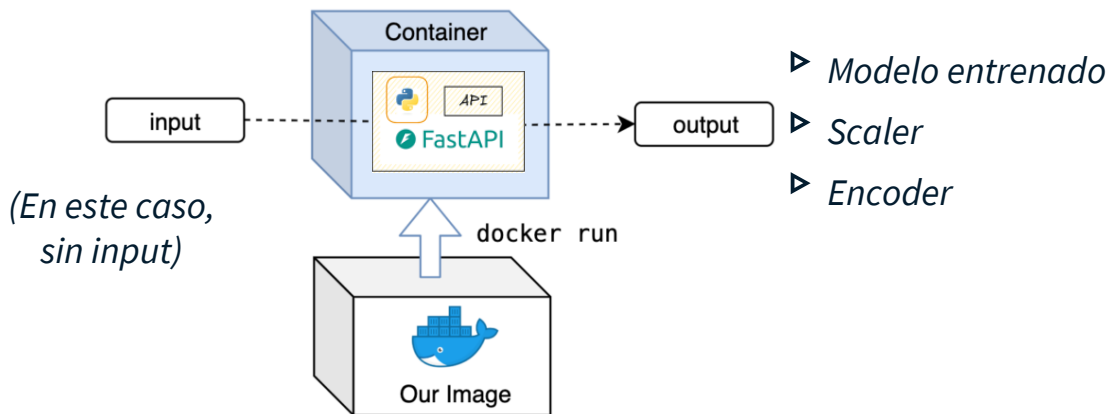
```
$ docker run -it --rm house-price-mlops:latest python3 train.py
```

*Modo interactivo
con terminal*

*Eliminar tras
ejecución*

*Versión de la
imagen de base*

*Comando que
sobreescribe el
CMD por defecto*



3. Ejecutar contenedor (train)



Entrenamiento del modelo

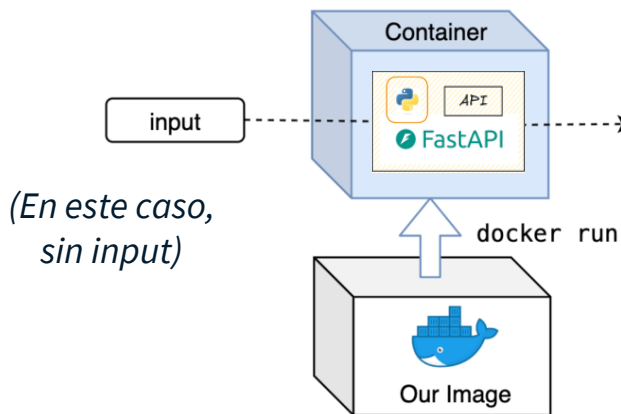
```
$ docker run -it --rm house-price-mlops:latest python3 train.py
```

Modo interactivo
con terminal

Eliminar tras
ejecución

Versión de la
imagen de base

Comando que
sobreescribe el
CMD por defecto x6



- ▶ Modelo entrenado
- ▶ Scaler
- ▶ Encoder



**¿Dónde se
guardan los
artifacts?**

Dentro del contenedor
(en /app/models)

3. Ejecutar contenedor



- ▶ El contenedor se **elimina** al terminar la ejecución
 - ▶ Por lo tanto, una vez ejecutado el entrenamiento, se elimina el contenedor y con ello los artefactos generados.
 - ▶ No podremos inferir sin tener y cargar el modelo.
- ▶ Necesitamos leer o persistir artefactos en nuestro equipo host
 - ▶ En entorno de **desarrollo**, a través de **volúmenes**:
 - ◆ *Train*: Conectar la carpeta deseada para que el contenedor guarde artefactos.
 - ◆ *Inference*: Conectar la carpeta con los artefactos para que el contenedor tenga acceso.
 - ▶ Para **CI/CD** y entornos de producción, a través de **registries**:
 - ◆ *Train*: Guarda los artefactos de manera externa (W&B, S3, Model Registries, ...)
 - ◆ *Inference*: Descarga los artefactos de manera remota (en lugar de localmente)

3. Ejecutar contenedor (train)



- ▶ Ejecutar entrenamiento, **guardando artefactos en la máquina host**

```
docker run -it --rm -v .\artifacts\:/app/artifacts house-price-mlops:latest python train.py
```

Montar volumen

Directorio
local
a montar

Ubicación en
el contenedor
donde se monta

Comando

```
# Guardar modelo, encoders y scaler en la carpeta artifacts
joblib.dump(model, "artifacts/model.pkl")
joblib.dump(encoders, "artifacts/encoders.pkl")
joblib.dump(scaler, "artifacts/scaler.pkl")
```

Importante la consistencia con
el código/app/servicio!
train.py

Terminal

```
PS C:\Users\Javier\upm-mdl-mlops> docker run -it --rm -v .\artifacts\:/app/artifacts house-price-mlops:latest python train.py
Métricas de evaluación:
MAE: 915955.8864463973
MSE: 1552389233075.0469
RMSE: 1245949.1294090007
R²: 0.6631280462200981
Artefactos guardados en la carpeta artifacts.
PS C:\Users\Javier\upm-mdl-mlops>
```



3. Ejecutar contenedor (inferencia)



- ▶ Ejecutar servidor de inferencia, **cargando los artefactos del entrenamiento**

```
$ docker run -it -rm -v .\artifacts\:/app/artifacts house-price-mlops:latest
```

Montar volumen *Directorio local a montar* *Ubicación en el contenedor donde se monta*

Por defecto, ejecuta lo que haya en la instrucción CMD del Dockerfile

Terminal

```
PS C:\Users\Javier\upm-mdl-mlops\project\src\api> docker run -it --rm -v .\artifacts\:/app/artifacts house-price-mlops:latest
INFO:      Started server process [1]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
INFO:      Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

```
(venv) javier@DESKTOP-DHCCEGH: /mnt/c/Users/Javier/upm-mdl-mlops/project$ python test/test_inference_api.py
Verificando que el servidor esté levantado...
Traceback (most recent call last):
  File "/mnt/c/Users/Javier/upm-mdl-mlops/project/venv/lib/python3.10/site-packages/urllib3/connection.py", line 198, in _new_conn
    sock = connection.create_connection(
  File "/mnt/c/Users/Javier/upm-mdl-mlops/project/venv/lib/python3.10/site-packages/urllib3/util/connection.py", line 85, in create_connection
    raise err
  File "/mnt/c/Users/Javier/upm-mdl-mlops/project/venv/lib/python3.10/site-packages/urllib3/util/connection.py", line 73, in create_connection
    sock.connect(sa)
ConnectionRefusedError: [Errno 111] Connection refused
```



EJECUTAR CON PERSISTENCIA

3. Ejecutar contenedor (inferencia)



El servidor está en el contenedor, no en nuestra máquina host ❌

```
$ docker run -it --rm -v .\artifacts:/app/artifacts \
  -p 8000:8000 Se publica el puerto 8000 del contenedor en el puerto 8000 de la máquina host
  house-price-mlops:latest
```

- ▶ Lanza servidor **uvicorn** con el modelo previamente entrenado (artifacts/model.pkl) mediante **fastapi**

- ▶ `http://localhost:8000/predict`



-p 8000:8000



```
PS C:\Users\Javier\upm-mdl-mlops\project\src> python3 .\test\test_inference_api.py
Verificando que el servidor esté levantado...
✓ El servidor está disponible.
✓ Servidor activo. Ejecutando tests...

Test predict_success:
Payload enviado: {'area': 5000.0, 'bedrooms': 4, 'bathrooms': 3, 'stories': 2, 'mainrshingstatus': 'semi-furnished'}
Respuesta de la API: {'predicted_price': 7980709.402862446}
✓ Test predict_success PASSED.

Test predict_validation_error:
Payload enviado (error): {'area': 'invalid', 'bedrooms': 4, 'bathrooms': 3, 'stories': no, 'furnishingstatus': 'semi-furnished'}
Respuesta de la API: {'detail': [{'type': 'float_parsing', 'loc': ['body', 'area'], 'm
```

Puerto 8000

Puerto 8000

Host

Contenedor



EJECUTAR INFERENCIA

4. Compartir imagen [opcional]



- Subir la imagen a un registry (e.g., Docker Hub)

```
# 1) Tag the image
$ docker tag house-price-mlops:latest yourdockerhubuser/house-price-mlops:latest

# 2) Log in
$ docker login --username yourdockerhubuser

# 3) Push
$ docker push yourdockerhubuser/house-price-mlops:latest
```

- Cualquiera con acceso puede descargar la imagen y ejecutar contenedor

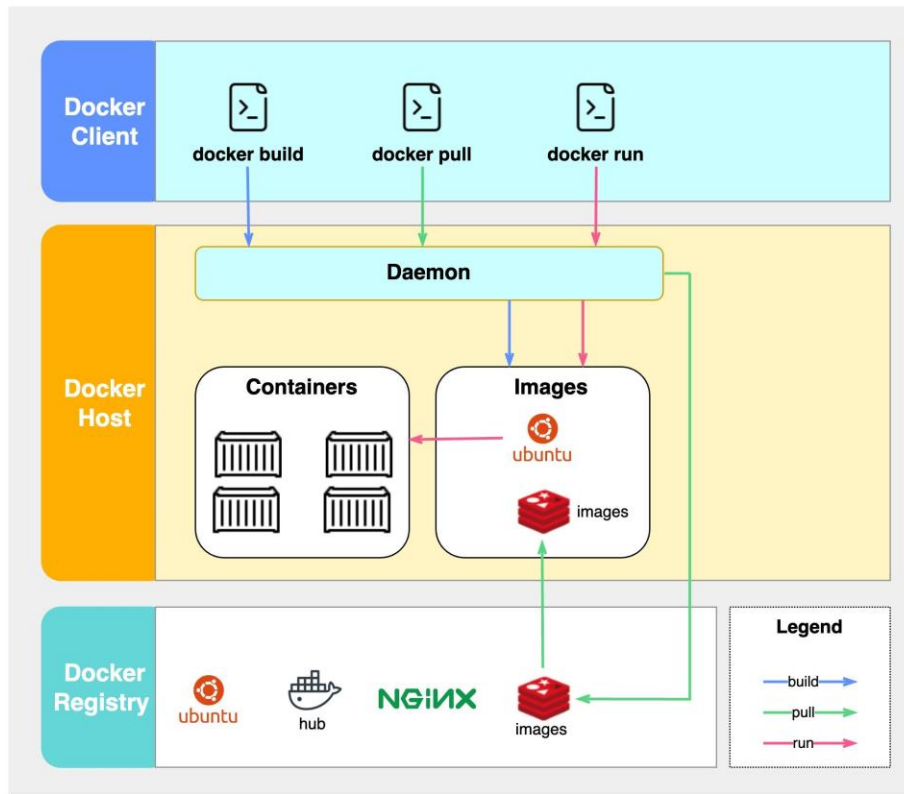
```
$ docker pull yourdockerhubuser/house-price-mlops:latest
```

PROBLEMA

*La imagen (house-price-mlops) actual no tiene los artefactos dentro
¿Cómo se solucionaría?*

How does Docker Work?

 blog.bytebytego.com



Comandos básicos de Docker

Comandos para Imágenes



- Listar imágenes disponibles

```
$ docker images
```

- Descargar imagen

```
$ docker pull <image>
```

- Construir imagen

```
$ docker build [options] <path to Dockerfile>
```

- Nombrar una imagen

```
$ docker tag <image> <new tag>
```

- Subir imagen

```
$ docker push <image>
```

- Eliminar imagen

```
$ docker rmi <image>
```

Comandos para Contenedores



- ▶ Ejecutar contenedor desde imagen

```
$ docker run [options] <image>
```

- ▶ Variables de entorno (pasar parámetros a ejecución)

```
$ docker run -e WANDB_API_KEY=<YOUR_KEY> -it --rm house-price-mlops:latest
```

- ▶ Listar contenedores en ejecución

```
$ docker ps
```

- ▶ Parar, reanudar, reiniciar o eliminar contenedor

```
$ docker stop/start/restart/rm <container>
```

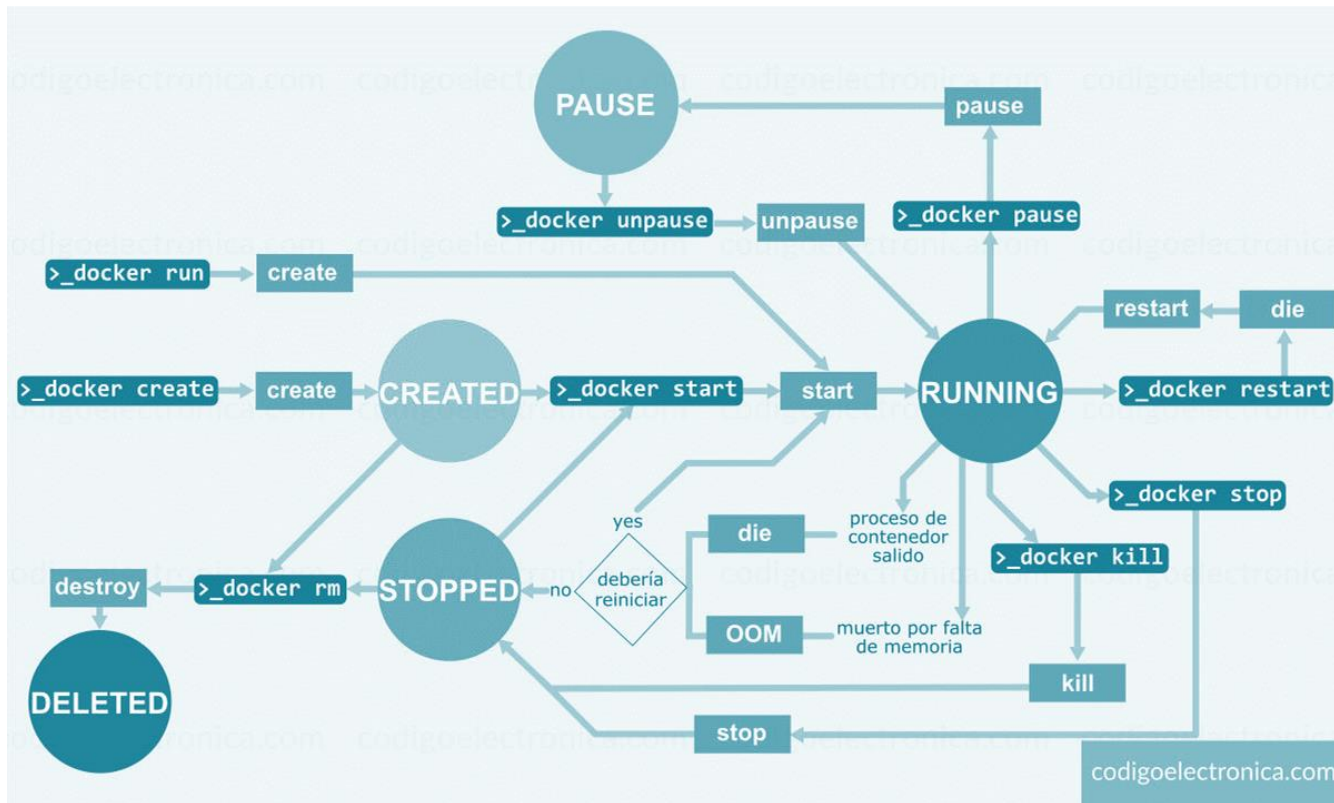
- ▶ Mostrar logs de un contenedor

```
$ docker logs [options] <container>
```

- ▶ Ejecutar comando dentro de un contenedor

```
$ docker exec [options] <container> <command>
```


Ciclo de vida de un contenedor





Interfaces gráficas o plugins



- Se puede usar con interfaz gráfica o a través de pluggins en IDE


 **Docker Desktop for Mac**
A native application using the macOS sandbox security model that delivers all Docker tools to your Mac.

 **Docker Desktop for Windows**
A native Windows application that delivers all Docker tools to your Windows computer.




 **Docker Desktop for Linux**
A native Linux application that delivers all Docker tools to your Linux computer.

Visual Studio | Marketplace

Visual Studio Code > Programming Languages > Docker



Docker

Microsoft  microsoft.com |  43,754,704 installs |  (100)

Makes it easy to create, manage, and debug containerized applications.

[Install](#) [Trouble Installing?](#)

[Overview](#) [Version History](#) [Q & A](#) [Rating & Review](#)

Docker for Visual Studio Code version **v1.29.4** installs **44M**

Orquestación de contenedores

docker-compose

Orquestar múltiples servicios

- ▶ ¿Cómo podemos ejecutar y comunicar varios contenedores que forman una aplicación?
- ▶ En la vida real, los servicios suelen exponer un front-end
 - ▶ Web UI + ML API + BBDD + Caché ...
- ▶ Docker Compose simplifica la ejecución y la comunicación entre los contenedores

Primero creamos otro servicio



- ▶ En **src/frontend**
 - ▶ Servidor en Flask (*web_app.py*)
 - ▶ Formulario web (*templates/index.html*)
 - ▶ Dockerfile
- ▶ Ejecución

```
$ docker build -t web-house-price-mlops .
```

```
$ docker run -it --rm -p 8080:8080 web-house-price-mlops
```



Primero creamos otro servicio



<http://localhost:8080/>

Ingresa los datos de la casa

Área (en m²):

Número de dormitorios:

Número de baños:

Número de plantas:

¿Tiene acceso a vía principal?

¿Tiene habitación de invitados?

¿Tiene sótano?

¿Tiene calefacción con agua caliente?

¿Tiene aire acondicionado?

Número de espacios de estacionamiento:

¿Está ubicada en una zona preferencial?

Estado de la amueblación:

ConnectionError

`requests.exceptions.ConnectionError: HTTPConnectionPool(host='api', port=8080):`

Traceback (most recent call last)

```
File "/usr/local/lib/python3.9/site-packages/urllib3/connection.py", line 198, in _new_conn
    sock = connection.create_connection(
File "/usr/local/lib/python3.9/site-packages/urllib3/util/connection.py", line 60, in create_connection
```



No sabe llegar al modelo
¿Por qué?

Docker Compose



- ▶ Herramienta para definir y ejecutar varios contenedores Docker
- ▶ Se basa en una especificación de contenedores: **docker-compose.yml**
 - ▶ Construye imágenes
 - ▶ Expone puertos
 - ▶ Define la red de comunicación
 - ▶ Establece dependencias de orden
- ▶ Workflow:
 1. Escribir el **docker-compose.yml**
 - ◆ <https://docs.docker.com/reference/compose-file/>
 2. Ejecutar **docker-compose up**

docker-compose.yml



Hostname visibles entre los contenedores!

Docker-compose crea una red privada que conecta todos los servicios.

```
version: "3.8"
services:
  api:
    image: house-price-mlops:latest # Si ya tienes una imagen, puedes usarla directamente
    build: # o se puede construir desde el Dockerfile
      context: ./api
      dockerfile: Dockerfile
    ports:
      - "8000:8000" # exposición en el host
    volumes:
      - ./api/artifacts:/app/artifacts
    environment:
      - API_PORT=8000
  web:
    image: web-house-price-mlops
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    depends_on:
      - api
    environment:
      - WEB_APP_PORT=8080
      - API_HOST=api
      - API_PORT=8000
```

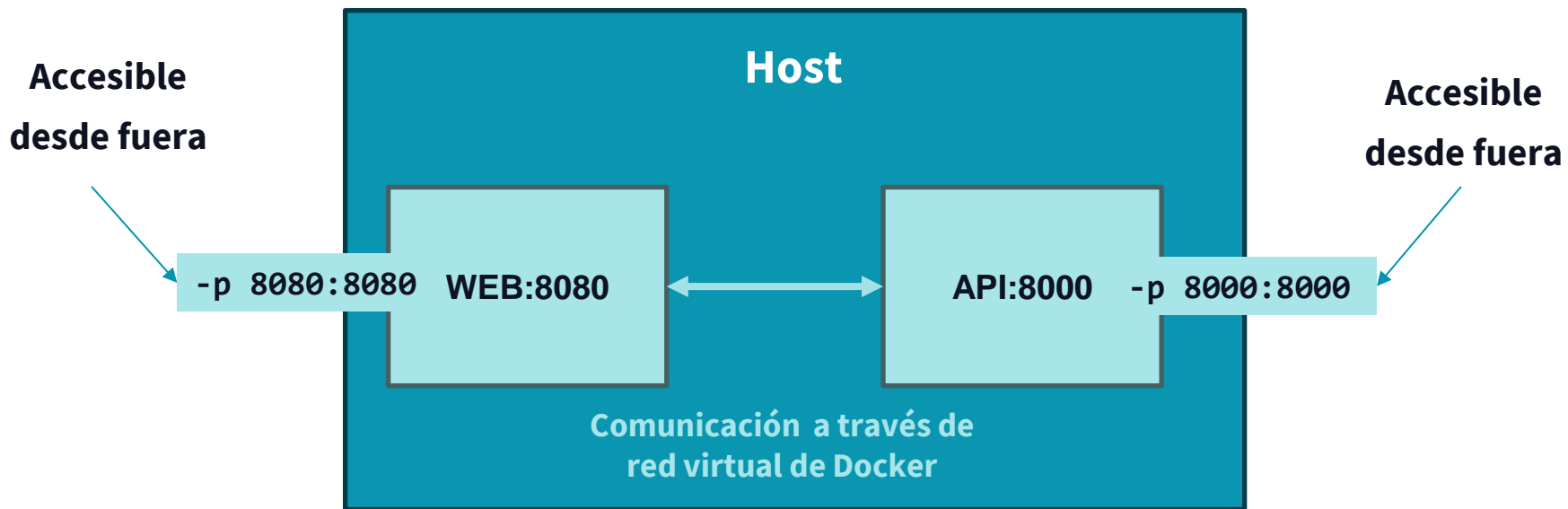
Lanzar los servicios



```
$ docker-compose up
```

Si se quieren reconstruir imágenes

```
$ docker-compose build
```



Lanzar los servicios



Containers

[Give feedback](#)

View all your running containers and applications. [Learn more](#)

Container CPU usage ⓘ

0.31% / 1200% (12 CPUs available)

Container memory usage ⓘ

192.95MB / 7.45GB



Only show running containers

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Last started
<input type="checkbox"/>	src	-	-	-	0.31%	13 seconds ago
<input type="checkbox"/>	web_service	d4039027b74f	web-house-price-mlops	8080:8080	0.11%	13 seconds ago
<input type="checkbox"/>	api_service	d1ee0e545070	house-price-mlops:late	8000:8000	0.2%	13 seconds ago

Terminal

```
web_service | * Restarting with stat
web_service | * Debugger is active!
web_service | * Debugger PIN: 102-655-477
api_service | INFO: Started server process [1]
api_service | INFO: Waiting for application startup.
api_service | INFO: Application startup complete.
api_service | INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```



Lanzar los servicios

localhost:8080

Ingresa los datos de la casa

Área (en m²):

Número de dormitorios:

Número de baños:

Número de plantas:

¿ Tiene acceso a vía principal?

¿ Tiene habitación de invitados?

localhost:8080/predict_price

El precio estimado de la casa es: 8898359.80

Comandos básicos de docker-compose

Comandos básicos



- ▶ Los comandos de contenedores e imágenes siguen funcionando
 - ▶ Nombre contenedores: `<nombre-directorio>-<service-name>-<num_instancia>`

```
$ docker ps
```

```
$ docker images
```

...

- ▶ Lanzar en segundo plano

```
$ docker-compose up -d
```

- ▶ Limpiar los contenedores (porque no se eliminan automáticamente)

```
$ docker-compose down
```

- ▶ Construir imágenes

```
$ docker-compose build
```

- ▶ Ejecutar un servicio puntual (parecido a *docker run*)

```
$ docker-compose run [opciones] <servicio> <comando>
```

Comandos básicos



- ▶ Ejecutar comando dentro del contenedor

```
$ docker-compose exec [opciones] <servicio> <comando>
```

- ▶ Listar contenedores definidos en el compose

```
$ docker-compose ps
```

- ▶ Ver logs

```
$ docker-compose logs [opciones] <servicio>
```

- ▶ Escalar un servicio

```
$ docker-compose up --scale <servicio>=N
```

Escalabilidad en contenedores

Se pueden incluir más servicios...



```
version: "3.8"
services:
  api:
    ...
  test:
    image: test-house-price-mlops
    build:
      context: ./test
      dockerfile: Dockerfile
    depends_on:
      - api
    environment:
      - API_HOST=api
      - API_PORT=8000

  web:
    ...
```

**Contenedor para lanzar
el testeo**

Se pueden incluir más servicios...



```
version: "3.8"
services:
  api:
    ...
  test:
    ...
  web:
    ...
# almacenamiento MINIO y Cache
minio:
  image: minio/minio
  ports:
    - "9000:9000"
  environment:
    MINIO_ROOT_USER: minioadmin
    MINIO_ROOT_PASSWORD: minioadmin
  command: server /data
  volumes:
    - minio_data:/data
redis:
  image: redis:latest
  ports:
    - "6379:6379"
  volumes:
    - redis_data:/data
```

**Almacenamiento con
caché**

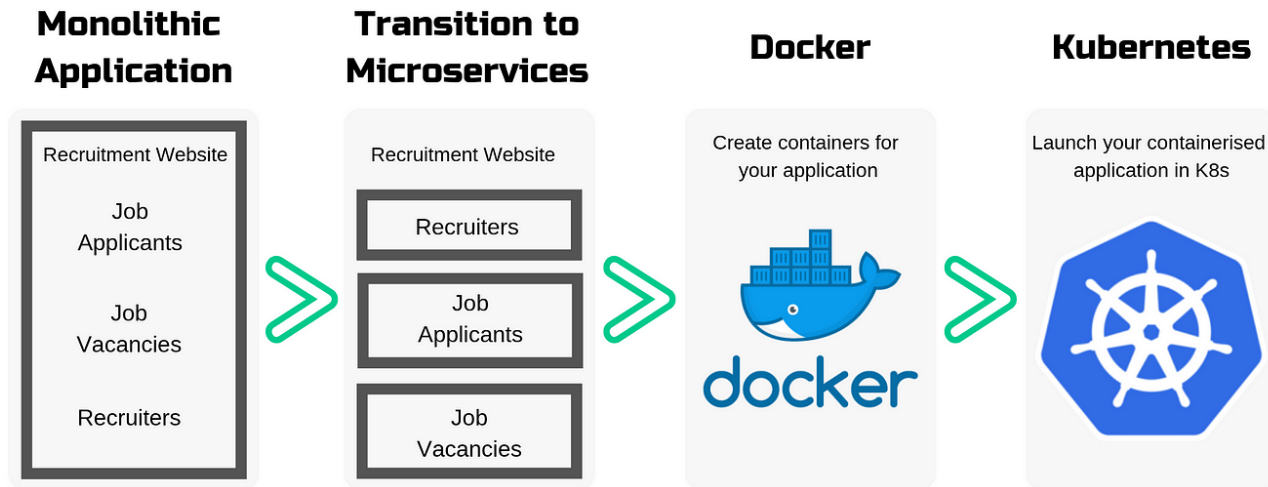
Se pueden incluir más servicios...

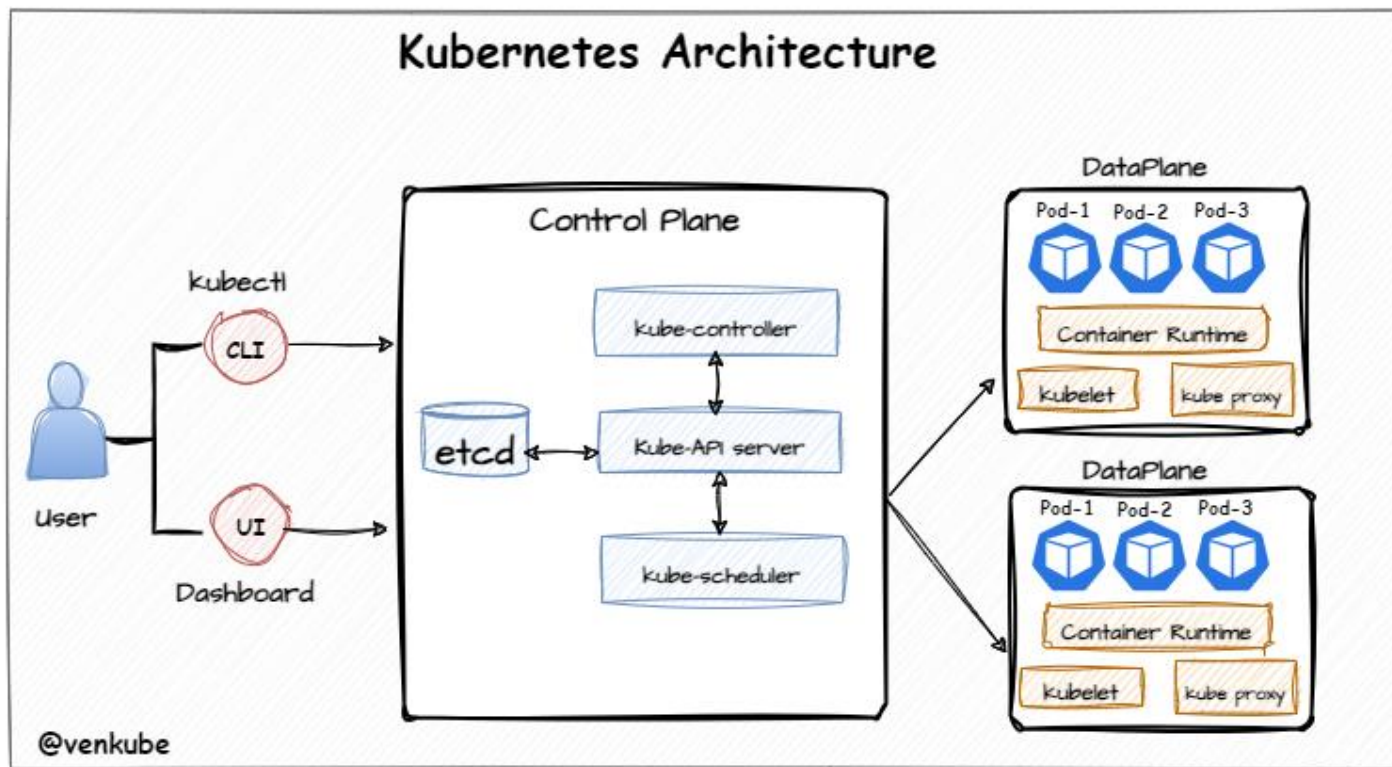


```
version: "3.8"
services:
  web:
    build: .
    scale: 3
...
# balanceo de carga y proxy inverso
nginx:
  image: nginx:latest
  ports:
    - "80:80"
  volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf
  depends_on:
    - api
    - web
```

**Balanceo de carga para
servicios escalados**

Cuando app escala mucho

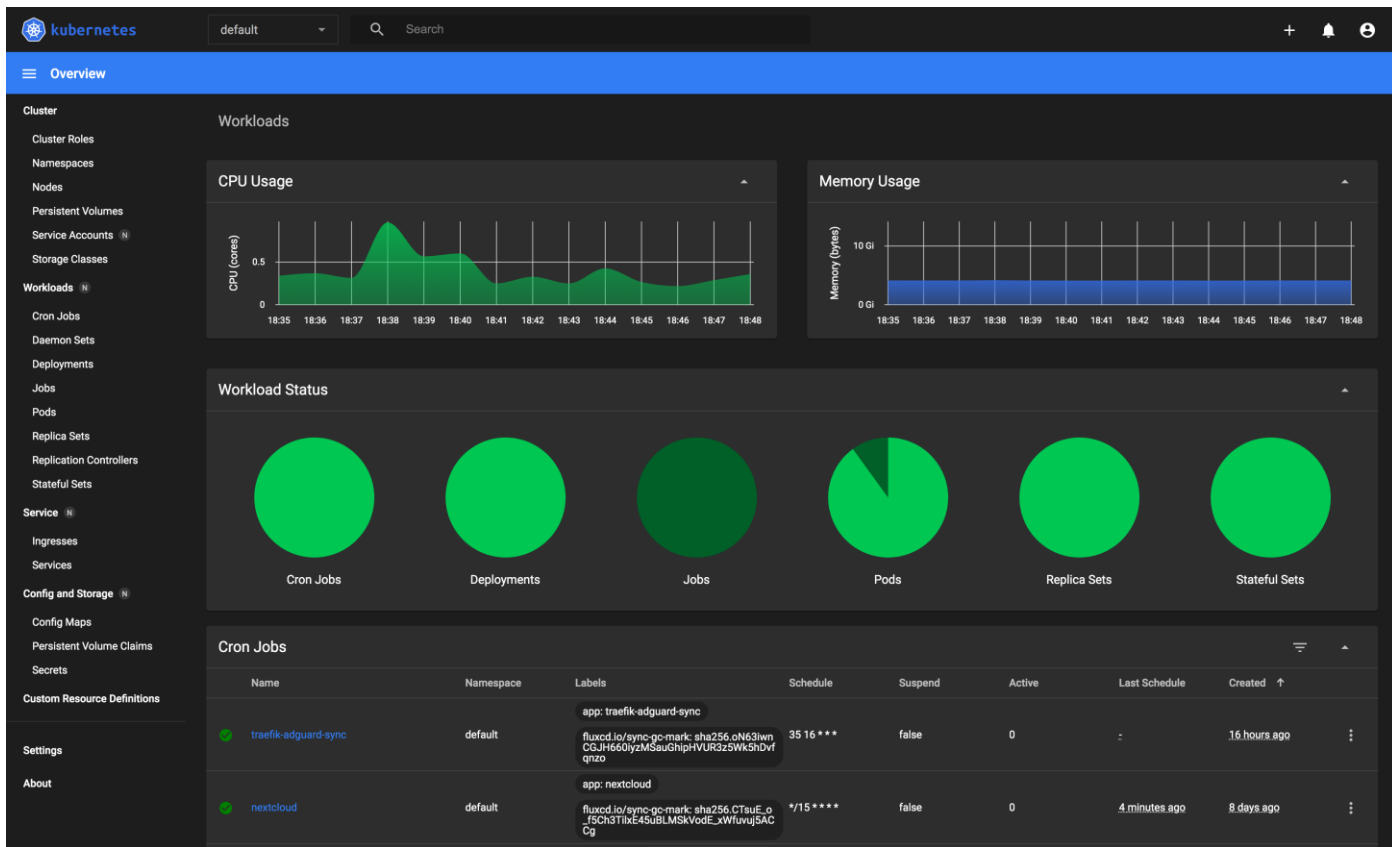




Kubernetes UI



Master
Deep Learning



Contenedores en producción

Microsoft Azure


Azure Container Registry


Microsoft Azure


Search resources, services, and docs (G+J)


Copilot


Azure services


Create a resource


Container registries


Quickstart Center


Azure AI services


More services

Home > Container registries >

Create container registry

Basics Networking Encryption Tags Review + create

Azure Container Registry allows you to build, store, and manage container images and artifacts in a private registry for all types of container deployments. Use Azure container registries with your existing container development and deployment pipelines. Use Azure Container Registry Tasks to build container images in Azure on-demand, or automate builds triggered by source code updates, updates to a container's base image, or timers. [Learn more](#)

Project details

Subscription *

Azure for Students

Resource group *

(New) upm-mlops

[Create new](#)

Instance details

Registry name *

javierpg


.azurecr.io

Location *

Spain Central

Use availability zones ⓘ

☐

 Availability zones are activated on premium registries and in regions that support availability zones. [Learn more](#)

Pricing plan * ⓘ


Basic



CREA TU CONTAINER REGISTRY

AzureCR- Access Keys

Home > javierpg


 **javierpg** | Access keys ☆ ...
Container registry

Search



- Overview
- Activity log
- Access control (IAM)
- Tags
- Quick start
- Resource visualizer
- Events
- Settings
- Access keys**
- Encryption
- Identity
- Networking
- Microsoft Defender for Cloud

Registry name: javierpg

Login server: javierpg.azurecr.io

Admin user  ☒

Username: javierpg

Name	Password	Regenerate
password	Show 
password2	Show 



GUARDA TU USERNAME Y PASSWORD

AzureCR- Push image



```
$ docker tag nombre-imagen usuario.azurecr.io/nombre-imagen:latest
```

```
$ docker login usuario.azurecr.io
```

te pedirá el usuario/contraseña que copiaste antes

```
$ docker push usuario.azurecr.io/nombre-imagen:latest
```

```
PS C:\Users\Javier\upm-mdl-mlops\project\src> docker login javierpg.azurecr.io
Username: javierpg
```

Info → A Personal Access Token (PAT) can be used instead.
To create a PAT, visit <https://app.docker.com/settings>

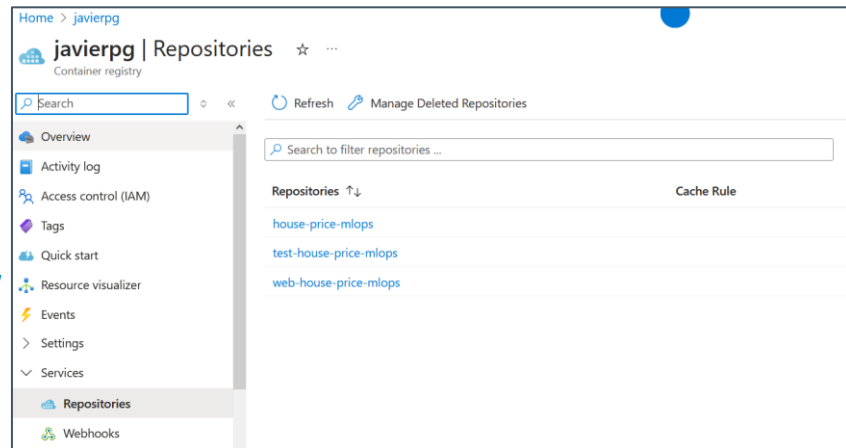
Password:

Login Succeeded

```
PS C:\Users\Javier\upm-mdl-mlops\project\src> 
```

```
PS C:\Users\Javier\upm-mdl-mlops\project\src> docker push javierpg.azurecr.io/house-price-mlops:latest
The push refers to repository [javierpg.azurecr.io/house-price-mlops]
```

```
da1cbe0d584f: Pushed
7640937c9928: Pushed
b99ab043338f: Pushing [=====] 320.9MB/450.7MB
9a95d1744747: Pushed
cec49b84de9d: Pushed
6b8b4e49f6db: Pushed
6e909acdb790: Pushed
3a411c8bf18a: Pushed
5324932f124c: Pushed
```



Azure Web App – Contenedor principal

Home > App Services >

Create Web App

Basics Database Container Networking Monitor + secure Tags Review + create

App Service Web Apps lets you quickly build, deploy, and scale enterprise-grade web, mobile, and API apps running on any platform. Meet rigorous performance, scalability, security and compliance requirements while using a fully managed platform to perform infrastructure maintenance. [Learn more](#)

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *

Resource Group *
[Create new](#)

Instance Details

Name .azurewebsites.net

☐ Try a secure unique default hostname. [More about this update](#)

Publish * ☐ Code ☒ Container

Operating System * ☒ Linux ☐ Windows

Region *
Not finding your App Service Plan? Try a different region or select your App Service Environment.

Pricing plans

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app. [Learn more](#)

Linux Plan (Spain Central) *
[Create new](#)

Pricing plan
[Explore pricing plans](#)

Home > App Services >

Create Web App

Basics Database **Container** Networking Monitor + secure Tags Review + create

Select your preferred source for container images. You can change these settings and other dependencies after creating the app. [Learn more](#)

Sidecar support ☒ Enhanced configuration with sidecar support on [Learn More](#)

Image Source * ☐ Quickstart ☒ Azure Container Registry ☐ Other container registries

Name *

Azure container registry options

Registry *

Authentication * ☐ Managed identity ☒ Admin credentials

Image *

Tag *

Port *

Startup Command

SPOILER: PACIENCIA CON LOS DESPLIEGUES



CREAR WEB APP

Azure Web App – App Service Plan

- ▶ Define **cómo** y **dónde** se ejecuta tu aplicación en Azure App Service
- ▶ Toda aplicación usa un Service plan. Puede haber varias apps en un plan.

Home > App Service plans > house-prices-plan >

Select App Service Pricing Plan

☒ Hardware view ☐ Feature view

Showing 19 App Service pricing plans

Name	ACU/vCPU	vCPU	Memory (GB)	Remote Storage (GB)	Scale (instance)	SLA	Cost per hour (instance)	Cost per month (instance)
Dev/Test (For less demanding workloads)								
Free F1	60 minutes/day co	N/A	1	1	N/A	N/A	Free	Free
<input checked="" type="checkbox"/> Basic B1	100	1	1.75	10	3	99.95%	0.018 USD	13.14 USD
Basic B2	100	2	3.5	10	3	99.95%	0.036 USD	26.28 USD
Basic B3	100	4	7	10	3	99.95%	0.071 USD	51.83 USD
Production (For most production workloads)								
Premium v3 P0V3	195*	1	4	250	30	99.95%	0.089 USD	64.97 USD
Premium v3 P1V3	195	2	8	250	30	99.95%	0.178 USD	129.94 USD
Premium v3 P1mv3	195*	2	16	250	30	99.95%	0.214 USD	155.928 USD
Premium v3 P2V3	195	4	16	250	30	99.95%	0.356 USD	259.88 USD
Premium v3 P3V3	195	8	32	250	30	99.95%	0.712 USD	519.76 USD
Premium v3 P2mv3	195*	4	32	250	30	99.95%	0.427 USD	311.856 USD
Premium v3 P3mv3	195*	8	64	250	30	99.95%	0.854 USD	623.712 USD
Premium v3 P4mv3	195*	16	128	250	30	99.95%	1.709 USD	1247.424 USD
Premium v3 P5mv3	195*	32	256	250	30	99.95%	3.418 USD	2494.848 USD
Legacy								
Standard S1	100	1	1.75	50	10	99.95%	0.095 USD	69.35 USD
Standard S2	100	2	3.5	50	10	99.95%	0.19 USD	138.70 USD
Standard S3	100	4	7	50	10	99.95%	0.38 USD	277.40 USD
Premium v2 P1V2	210	1	3.5	250	30	99.95%	0.115 USD	83.95 USD
Premium v2 P2V2	210	2	7	250	30	99.95%	0.231 USD	168.63 USD
Premium v2 P3V2	210	4	14	250	30	99.95%	0.462 USD	337.26 USD

Azure Web App – Página web



Home > Microsoft.Web-WebApp-Portal-650f9707-8c49 | Overview

Deployment

Search

Overview

Inputs

Outputs

Template

Your deployment is complete

Deployment name: Microsoft.Web-WebApp-Portal-650f9707-8c49
Subscription: Azure for Students
Resource group: upm-mlops

Deployment details

Resource	Type
house-prices/scm	Microsoft.Web
house-prices/web	Microsoft.Web
house-prices	Microsoft.Web
house-prices-plan	Microsoft.Web

Next steps

Manage deployments for your app. Recommended

Protect your app with authentication. Recommended

Go to resource

Give feedback

Tell us about your experience with deployment

Formulario - Precio de Casas

https://house-prices.azurewebsites.net

Ingresa los datos de la casa

Área (en m²):

Número de dormitorios:

Número de baños:

Número de plantas:

¿Tiene acceso a vía principal?

Sí

¿Tiene habitación de invitados?

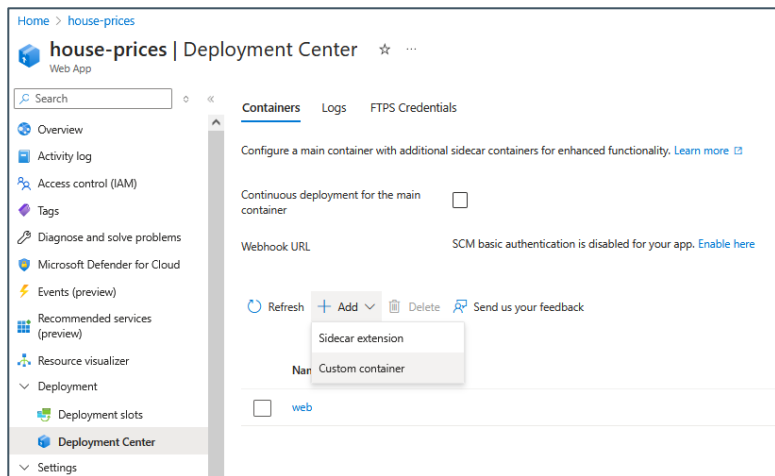
Sí

<https://house-prices.azurewebsites.net/>



ACCEDE A LA WEB DESPLEGADA

Azure Web App - Sidecar



**Una vez desplegado en estado “Running”,
ya se predicen precios de casas**

Add container

Pull container images from Azure Container Registry, Docker Hub, or a private Docker repository. App Service will deploy the containerized app with your preferred dependencies to production in seconds. You can change these settings after creating the app.

Name *

Type

Image source * ☒ Azure Container Registry ☐ Other container registries

Subscription

Registry *

Authentication * ☐ Managed Identity ☒ Admin Credentials

Image *

Image tag *

Port

Startup command



Referencias

- ▶ Brownlee, J. (2024). A Practical Guide to Deploying Machine Learning Models.
- ▶ Docker Docs. <https://docs.docker.com>
- ▶ Azure App Service. Team Blog (2025). Using the Redis Sidecar Extension with Azure App Service for Linux.
- ▶ Tuulos, V. (2022). *Effective Data Science Infrastructure. How to make data scientists productive.*
- ▶ Yasser, M (2022). Kaggle Datasets. *Housing Prices Prediction - Regression Problem.*
- ▶ Setu, S (2024). Kaggle Code. *House Pricing – Regression.*