**6) Despliegue y contenedorización (Docker)**

**1) Objetivo**

Contenerizar el sistema completo de inspección de anomalías (backend FastAPI + IA PatchCore + frontend web) para que pueda ejecutarse en cualquier equipo sin necesidad de configurar entornos locales o dependencias manualmente.  
El resultado es una imagen Docker reproducible, con el modelo precargado y el servidor accesible en http://127.0.0.1:8000.

**2) Imagen Docker – estructura y configuración**

**Base:**

* Imagen oficial python:3.10-slim (ligera y compatible).
* Instalación de dependencias del sistema: build-essential, git, libgl1, libglib2.0-0, etc., requeridas por PyTorch y OpenCV.
* Instalación de dependencias Python desde Backend/requirements.txt.
* Copia del código fuente completo (Backend/) y creación de la carpeta de resultados /app/Backend/static/overlays.

**Optimizaciones añadidas:**

* Variable PYTHONDONTWRITEBYTECODE=1 para evitar archivos .pyc.
* Variable PYTHONUNBUFFERED=1 para logs inmediatos.
* Uso de PIP\_NO\_CACHE\_DIR=1 para reducir peso de la imagen.
* Configuración de ENV TORCH\_HOME=/app/.cache/torch → fija la ruta donde se almacenan los pesos del modelo (evita descargas repetidas).
* Precarga del modelo ResNet18 durante el build (precalentado) o cacheado mediante volumen persistente.

**3) Precarga y persistencia del modelo**

El modelo base ResNet18 de PyTorch (≈44 MB) se descarga automáticamente la primera vez.  
Para evitarlo en ejecuciones posteriores se implementaron **dos estrategias**:

1. **Caché persistente (preferida en desarrollo):**
   * Se mapea una carpeta local .cache\torch en el contenedor.
   * Comando:

docker run --rm -p 8000:8000 ` -v ".\Backend\static\overlays:/app/Backend/static/overlays" -v ".\.cache\torch:/app/.cache/torch" ` inspector-patchcore:latest

* + La primera vez descarga el modelo; las siguientes, arranca directamente.

1. **Descarga en build (modo producción):**
   * El peso se “hornea” dentro de la imagen mediante:
   * RUN python - <<'PY'
   * import torch
   * from torchvision.models import resnet18, ResNet18\_Weights
   * torch.hub.set\_dir("/app/.cache/torch")
   * resnet18(weights=ResNet18\_Weights.IMAGENET1K\_V1)
   * print("ResNet18 weights baked into the image.")
   * PY
   * Resultado: la imagen final incluye el archivo  
     /app/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth.

**4) Construcción y ejecución**

**Build:**

cd \Inspector-anomalias-patchcore

docker build -t inspector-patchcore:latest .

**Ejecución estándar (modo prueba):**

docker run --rm -p 8000:8000 ` -v ".\Backend\static\overlays:/app/Backend/static/overlays" ` inspector-patchcore:latest

**Ejecución con caché persistente (recomendada):**

docker run --rm -p 8000:8000 ` -v ".\Backend\static\overlays:/app/Backend/static/overlays" ` -v ".\.cache\torch:/app/.cache/torch" ` inspector-patchcore:latest

**Modo background (sin bloquear la terminal):**

docker run -d --name inspector ` -p 8000:8000 ` -v ".\Backend\static\overlays:/app/Backend/static/overlays" ` inspector-patchcore:latest

**5) Resultado del despliegue**

Al ejecutar el contenedor:

* El servidor FastAPI se inicia automáticamente con Uvicorn:
* [startup] Device=cpu | IMG\_SIZE=256 | KNN\_K=5 | THRESHOLD=0.4 ...
* Uvicorn running on http://0.0.0.0:8000
* Endpoints disponibles:
  + GET / → interfaz web del frontend.
  + GET /health → estado de configuración activa.
  + POST /predict → inferencia de imagen, con resultado JSON y overlay visual.
* Las imágenes de salida se guardan en  
  Backend/static/overlays/ (en la carpeta del host).
* No se descargan pesos adicionales tras la primera ejecución (gracias al volumen persistente o al precacheado).

**6) Beneficios del enfoque con Docker**

• **Portabilidad total:** el sistema corre igual en Windows, Linux o GPU server.  
• **Entorno aislado:** dependencias y librerías reproducibles, sin conflictos locales.  
• **Inicio rápido:** modelo y dependencias ya listos, sin instalación manual.  
• **Escalabilidad:** fácil despliegue en entornos cloud (AWS, Azure, GCP) o integración en pipelines CI/CD.  
• **Uso dual CPU/GPU:** compatible con aceleración CUDA en entornos que dispongan de ella.

**7) Estado final del sistema**

* Docker Desktop operativo sobre WSL2.
* Imagen inspector-patchcore:latest creada y funcional.
* Backend accesible en http://127.0.0.1:8000.
* Frontend web integrado (HTML/CSS/JS sin dependencias externas).
* Carpeta de resultados (static/overlays) y cache (.cache/torch) mapeadas correctamente.
* Contenedor validado con peticiones POST /predict → resultados correctos y polígonos coherentes.

El entorno Dockerizado permite desplegar y ejecutar el sistema de inspección de anomalías PatchCore de manera estable, reproducible y portable, eliminando dependencias locales y garantizando que el backend, modelo y frontend funcionen de forma consistente en cualquier máquina.