Actumlogos

La palabra reflexionada puesta en acción

Docker: Fundamentos aplicados con LLMs

Taller • 10 horas • Zoom

Construye y despliega desde cero un stack completo (frontend, API, base vectorial y LLM) listo para producción. Pasarás de tener código que solo corre en tu laptop a empaquetar aplicaciones completas que funcionan igual en cualquier máquina o servidor y se levantan con un solo comando. Partirás desde tu primer contenedor y un LLM corriendo en local con Ollama, y avanzarás hasta las prácticas que usan los equipos profesionales: **Dockerfiles, volúmenes, redes, Docker Compose, manejo de secretos, límites de recursos y healthchecks**. Todo se hace a mano, en la terminal, sobre un proyecto real. Las habilidades sirven para cualquier stack (web, datos o IA); aquí las practicas construyendo una aplicación con LLMs. Entiendes qué hace cada pieza, cómo se conectan entre sí y qué revisar cuando algo no arranca.

Lo que harás:

- Levantar tu primer LLM en local con Ollama y consumirlo vía API REST, sin instalar librerías en tu sistema.
- Dominar el ciclo de vida de un contenedor: pull, run, ps, stop, rm y logs.
- Escribir un Dockerfile desde cero, empaquetar una API con FastAPI y publicar su imagen en Docker Hub.
- Resolver los fallos más comunes al construir imágenes: rutas relativas, dependencias faltantes y puertos ocupados.
- Blindar tus imágenes con .dockerignore, un usuario sin privilegios y escaneo de vulnerabilidades con docker scout.
- Persistir modelos y datos con volúmenes, y desarrollar con hot-reload mediante bind mounts.
- Crear una red privada donde la API, Ollama y la base vectorial Qdrant se comunican por nombre de servicio.
- Orquestar el stack completo (interfaz web, API, Qdrant y Ollama) con un solo docker-compose.yml.
- Manejar secretos sin exponerlos, limitar la memoria de tus servicios y vigilar su salud con healthchecks.

Herramientas a aprender a usar: Docker, Docker Compose, Docker Hub, Docker Scout, Ollama, FastAPI, Qdrant, cURL y la terminal.

Contenido

Módulo 1: Fundamentos de Docker y primer LLM en local

Empiezas por el problema que Docker resuelve: el "en mi máquina sí funciona" y los choques de versiones y librerías entre proyectos. Entiendes su arquitectura (Engine, daemon, imágenes, contenedores y registries), corres tu primer contenedor y dominas su ciclo de vida, incluido qué pasa con tus datos cuando lo destruyes. Cierras con tu primer servicio real: un LLM ligero corriendo en CPU con Ollama, al que consultas vía API REST.

1. Introducción y arquitectura:
    a. El problema de "en mi máquina sí funciona", conflictos de versiones y librerías del sistema.
    b. Conceptos base: Docker Engine, Daemon, Imágenes vs. Contenedores, Registries.
    c. Verificación del entorno y primer contenedor: `docker run hello-world`.
2. Ciclo de vida y modo interactivo:
    a. Comandos esenciales: `docker pull`, `docker run`, `docker ps`, `docker stop`, `docker rm`, `docker logs`.
    b. Contenedor interactivo con `python:3.11-slim`.
    c. La naturaleza efímera: comprobar qué pasa con los datos al destruir un contenedor.
3. Primer servicio funcional: LLM en CPU con Ollama:
    a. Por qué contenerizar motores de IA: cero configuración de librerías complejas en el host.
    b. Despliegue del contenedor de Ollama mapeando puertos (`11434:11434`).
    c. Descarga y prueba de un modelo ultraligero optimizado para CPU (ej. `llama3.2:1b` o `qwen2.5:1.5b`).
    d. Consumo del modelo vía API REST mediante cURL, Postman o script básico de Python.

Módulo 2: Creación de imágenes, seguridad y empaquetado de APIs

Dejas de depender de imágenes ajenas y construyes las tuyas. Aprendes la anatomía de un Dockerfile y empaquetas paso a paso una API con FastAPI que habla con el modelo, resolviendo los fallos típicos: rutas relativas, dependencias faltantes y puertos ocupados. Después la blindas con .dockerignore, un usuario sin privilegios y un escaneo con docker scout; conoces los multi-stage builds con una plantilla lista para usar y publicas tu imagen en Docker Hub.

1. Anatomía de un Dockerfile:
    a. Estructura por capas e instrucciones principales: `FROM`, `WORKDIR`, `COPY`, `RUN`, `EXPOSE`, `CMD`.
    b. Selección de imágenes base seguras y ligeras (`python:3.11-slim`).
2. Empaquetado de una API (FastAPI) paso a paso:
    a. Presentación del código base del backend para interactuar con el modelo.
    b. Construcción del `Dockerfile` e instalación limpia de dependencias con `requirements.txt`.
    c. `docker build` y ejecución mapeando puertos.

   d. Resolución de fallos comunes: rutas relativas, dependencias faltantes y puertos ocupados.
3. Seguridad y optimización básica:
   a. Blindaje con `.dockerignore`: evitar fugas de .env, .git o entornos virtuales.
   b. Seguridad básica: creación y uso de usuario sin privilegios (`USER appuser`) para no correr como root.
   c. Escaneo rápido de vulnerabilidades en la imagen con `docker scout`.
   d. Multi-stage builds (enfoque conceptual): Diapositiva comparativa de reducción de tamaño y entrega de plantilla `Dockerfile.multistage` lista para descarga.
   e. Publicación básica en Docker Hub (`tag` y `push`).

Módulo 3: Persistencia de datos y redes entre contenedores

Resuelves los dos problemas que aparecen en cuanto tienes más de un contenedor: datos que se pierden al reiniciar y servicios que no se encuentran entre sí. Con volúmenes conservas los modelos de Ollama y los datos de tu base vectorial; con bind mounts desarrollas con hot-reload sin reconstruir la imagen. Luego creas tu propia red, despliegas Qdrant y ejecutas una consulta completa entre la API, Ollama y Qdrant dentro de la red privada.

1. Volúmenes y desarrollo en vivo:
   a. El problema de perder pesos de modelos y bases de datos al reiniciar contenedores.
   b. Docker Volumes: Persistencia de modelos de Ollama y datos de la base vectorial.
   c. Bind Mounts: Mapeo de carpetas locales para desarrollo con recarga automática (hot-reload) en FastAPI sin tener que reconstruir la imagen.
2. Redes personalizadas (Networking):
   a. Limitaciones de la red default bridge.
   b. Creación de red propia: `docker network create ia-net`.
   c. Despliegue de la base de datos vectorial (Qdrant).
   d. Comunicación entre contenedores por nombre de servicio (DNS interno) en lugar de IPs fijas.
3. Integración del flujo:
   a. Conexión entre el contenedor de la API, Ollama y Qdrant.
   b. Prueba de una consulta completa dentro de la red privada.

Módulo 4: Orquestación multi-contenedor con Docker Compose

Todo lo que levantaste a mano con docker run se vuelve un solo archivo declarativo. Aprendes la sintaxis de docker-compose.yml y ensamblas el stack completo: interfaz web, API, Qdrant y Ollama. Manejas secretos como en un equipo profesional (.env.example en el repositorio y .env local sin versionar), controlas el arranque con depends_on y operas todo con up, logs, down y exec. Cierras viendo por qué Docker sirve como entorno aislado cuando agentes de IA ejecutan herramientas externas.

1. Del comando manual al archivo declarativo:
   a. Por qué Compose resuelve la complejidad de múltiples comandos `docker run`.
   b. Sintaxis base de `docker-compose.yml`: servicios, redes, volúmenes y variables de entorno.
2. Ensamblado del Stack completo:

   a. Integración en un solo archivo: Base Vectorial (Qdrant) + LLM (Ollama en CPU) + API Backend + Interfaz web ligera.
   b. Gestión segura de secretos: uso de `.env.example` para el repositorio vs. `.env` local sin versionar.
   c. Control de arranque con `depends_on`.
3. Operación y buenas prácticas:
   a. Comandos de gestión: `docker compose up -d`, `docker compose logs -f`, `docker compose down`.
   b. Acceso e inspección con `docker compose exec`.
   c. Aislamiento y sandboxing: Breve mención conceptual sobre cómo Docker sirve de entorno aislado y seguro cuando agentes de IA ejecutan herramientas externas (como en protocolos tipo MCP).

Módulo 5: Recursos, estabilidad, GPU y cierre

Llevas tu stack de "funciona" a "funciona estable". Monitoreas CPU y RAM con docker stats, pones límites de memoria para que un modelo no tumbe tu sistema y configuras healthchecks para saber cuándo cada servicio está listo de verdad. Ves en demostración cómo se usa la GPU en producción y te llevas un docker-compose.gpu.yml para probarlo por tu cuenta, liberas espacio con docker system prune y cierras revisando tu repositorio final y las rutas para llevarlo a la nube.

1. Control de recursos y aceleración:
   a. Monitoreo en tiempo real con `docker stats` (consumo de CPU y memoria RAM).
   b. Establecer límites de memoria (`mem_limit`) para proteger el sistema operativo contra caídas por falta de RAM.
   c. GPU en producción (explicación en diapositiva y demostración):
      i. Cómo funciona el NVIDIA Container Toolkit en servidores Linux y entornos cloud.
      ii. Limitaciones en macOS (sin passthrough de GPU) y ventajas del esquema universal en CPU.
      iii. Entrega de archivo `docker-compose.gpu.yml` para alumnos que cuenten con GPU dedicada y deseen probarlo por su cuenta.
2. Estabilidad y mantenimiento del entorno:
   a. Configuración básica de `healthcheck` para verificar servicios listos.
   b. Higiene del sistema y liberación de espacio en disco: `docker system prune` y gestión de volúmenes huérfanos.
3. Proyecto final y siguientes pasos:
   a. Revisión del repositorio estructurado y listo para portafolio.
   b. Visión general de despliegue a servicios en la nube (AWS ECS, GCP Cloud Run, Azure Container Apps).
   c. Cierre y resolución de dudas finales.

Actumlogos
La palabra reflexionada puesta en acción

Ya viste lo que vas a aprender.

Asegura tu lugar

Inscríbete ahora · $1,000 MXN →

Constancia con registro STPS · Grabaciones por 1 año · Acceso a Zoom al instante

¿Dudas antes de inscribirte? Escríbenos por WhatsApp

Al terminar tendrás:

- Un stack completo de IA (interfaz web, API, base vectorial y LLM) que levantas con un solo comando.
- El criterio para decidir qué va en un volumen, qué va en un bind mount y qué nunca debe entrar en una imagen.
- La habilidad de diagnosticar cuando algo falla (puertos ocupados, dependencias faltantes, servicios que no arrancan o se quedan sin memoria) en lugar de reinstalar todo.
- Imágenes propias, construidas con buenas prácticas de seguridad y publicadas en Docker Hub.
- Plantillas listas para reutilizar: un Dockerfile multi-stage y un docker-compose.gpu.yml para cuando tengas GPU.
- Un repositorio estructurado y listo para tu portafolio.
- Constancia de participación verificable en línea.

¿Para quién es este taller?

- Personas que trabajan en ML, IA o ciencia de datos y saben construir modelos o scripts, pero se atoran al empaquetarlos, compartirlos o llevarlos a un servidor.
- Desarrolladores y estudiantes que nunca han usado Docker, o lo usan copiando comandos de tutoriales y quieren entender de verdad lo que hacen.
- Quienes quieren integrar LLMs en aplicaciones reales y necesitan que todo el stack (API, base vectorial y modelo) corra igual en cualquier máquina.

Este taller NO es para ti si:

- Ya usas Docker y Docker Compose con soltura a diario: Dockerfiles, volúmenes, redes y stacks multi-servicio.
- Buscas un curso de IA o Machine Learning. Aquí no entrenas modelos ni estudias cómo funcionan por dentro; aprendes a empaquetar, conectar y desplegar aplicaciones que los usan.

- Esperas Kubernetes, orquestación de clústeres o un despliegue completo en la nube. Aquí llegas hasta un stack multi-servicio con Docker Compose y una visión general de los servicios cloud, no a la infraestructura a gran escala.

Clases

5 sesiones de 2 horas (10 hrs en total), martes 20, jueves 22, martes 27 y jueves 29 de octubre, y martes 3 de noviembre de 2026, de 20:00 a 22:00 (hora de la Ciudad de México). Cada sesión es práctica: levantas contenedores, escribes Dockerfiles y armas el stack en vivo en tu propia máquina, con la teoría justa para entender qué está pasando. Todo se imparte por Zoom y queda grabado; tendrás acceso a las grabaciones durante 1 año para repasar a tu ritmo.

Requisitos

Computadora con Windows 10/11 (WSL2 activo), macOS (Intel o Apple Silicon) o Linux; 16 GB de RAM recomendados (mínimo 8 GB) y al menos 20 GB libres en disco. No necesitas GPU: todo el stack corre en CPU. Tampoco necesitas experiencia previa con Docker, IA ni Machine Learning.

Instructor

Dr. Uriel Escalona — LinkedIn

Testimonios

Hemos capacitado a más de 2,000 personas en IA — visita actumlogos.com

Medios de pago

PayPal, transferencia SPEI, Tarjetas de crédito/débito, Link, Apple Pay, Google Pay.
Ver checkout

Lista de Espera

Constancia de participación con folio único, verificable en línea

Al terminar recibirás una constancia de participación firmada por el instructor y el director de Actumlogos — agente capacitador registrado ante la Secretaría del Trabajo y Previsión Social. Cada constancia incluye un folio único con el que cualquier persona puede verificar su autenticidad en línea, y podrás compartirla directamente en LinkedIn.

Registro STPS: ZAGE-810930-FW2-0005

Este es un ejemplo para ilustrar qué esperar: