Docker_

Guía de atajos · contenedores y Compose

```
user@dajomar:~$ docker --help
```

Conceptos clave

Imagen	Plantilla de solo lectura: el sistema base, tu app y sus dependencias.
Contenedor	Una imagen en ejecución, aislada y desechable.
Dockerfile	Receta paso a paso para construir una imagen.
Volumen	Almacenamiento que sobrevive al contenedor.
Red	Permite que los contenedores se comuniquen entre sí.
Registry	Repositorio de imágenes (Docker Hub, GHCR, ECR).

Una imagen es como una clase y el contenedor es una instancia: de una imagen puedes levantar muchos contenedores.

Información y ayuda

```
docker version
docker info  # contenedores, imágenes, storage
docker <comando> --help
docker system df  # espacio usado por Docker
```

Imágenes

Descargar y listar

```
docker pull nginx  # por defecto toma :latest
docker pull postgres:16-alpine
docker images
docker image ls --filter dangling=true
```

Inspeccionar

```
docker image inspect nginx
docker history nginx  # capas y su tamaño
```

Etiquetar, subir y borrar

```
docker tag miapp:latest dajomar/miapp:1.0.0
docker login
docker push dajomar/miapp:1.0.0
docker rmi nginx
docker image prune     # borra imágenes colgadas
docker image prune -a  # borra las que no se usan
```

Formato: `USUARIO/REPOSITORIO:TAG`. Si no pones tag, Docker asume `latest`.

Correr contenedores

```
docker run -d --name web -p 8080:80 nginx
docker run -it --rm ubuntu bash
docker run -d --name db \
  -e POSTGRES_PASSWORD=secreto \
  -v pgdata:/var/lib/postgresql/data \
  --restart unless-stopped \
  postgres:16-alpine
```

-d	Segundo plano (detached)
-it	Terminal interactiva
--rm	Se borra al detenerse
--name	Nombre fijo para el contenedor
-p 8080:80	Puerto **HOST : CONTENEDOR**
-e VAR=valor	Variable de entorno
--env-file .env	Variables desde un archivo
-v origen:destino	Monta un volumen o carpeta
--network	Conecta a una red
--restart	no · always · unless-stopped · on-failure

Pro Tip:
Las banderas cortas se pueden combinar: `-d -p 80:80` es lo mismo que `-dp 80:80`. Y para referirte a un contenedor bastan los primeros caracteres de su ID.

Gestionar contenedores

```
docker ps          # los que están corriendo
docker ps -a       # todos, incluso detenidos
docker stop web ; docker start web
docker restart web
docker rm web
docker rm -f web  # detiene y borra
docker rename web web-prod
```

Logs y depuración

```
docker logs web
docker logs -f --tail 100 web  # seguir en vivo
docker exec -it web sh  # entrar al contenedor
docker exec web cat /etc/nginx/nginx.conf
docker inspect web  # config completa en JSON
docker inspect web \
  -f '{{.NetworkSettings.IPAddress}}'
docker stats  # CPU, RAM y red en vivo
docker top web                 # procesos dentro
docker cp web:/var/log/app.log ./
```

Si la imagen no trae `bash` (como Alpine), usa `sh`.

Volúmenes: persistir datos

Todo lo que se escribe dentro de un contenedor se pierde al borrarlo. Los volúmenes lo evitan.

Tipo	Uso
`Named volume`	Docker lo gestiona. Ideal para bases de datos.
`Bind mount`	Una carpeta de tu equipo. Ideal en desarrollo.
`tmpfs`	Solo en RAM. Datos temporales o sensibles.

```
docker volume create pgdata
docker volume ls
docker volume inspect pgdata
docker volume rm pgdata
docker volume prune  # borra los que no se usan
```

Montarlos al correr

```
# named volume
docker run -v pgdata:/var/lib/postgresql/data \
  postgres
# bind mount (ruta absoluta del host)
docker run -v "$(pwd)":/app -w /app \
  node:20 npm test
# solo lectura
docker run \
  -v "$(pwd)"/conf:/etc/nginx/conf.d:ro nginx
```

En PowerShell usa `${PWD}` y la comilla invertida `` ` `` para cortar líneas en lugar de `\`.

Redes

Regla de oro: contenedores en la misma red definida por el usuario se encuentran **por nombre**, gracias al DNS interno de Docker.

`bridge`	Por defecto. Red privada dentro del host.
`host`	Usa directamente la red del host (sin `-p`).
`none`	Sin red: aislamiento total.

```
docker network create app-net
docker network ls
docker network inspect app-net
docker run -d --name db --network app-net postgres
docker run -d --name api --network app-net miapi
# api se conecta a la base usando el host "db"
docker network connect app-net web
docker network prune
```

Dockerfile

Instrucción	Para qué sirve
`FROM`	Imagen base
`WORKDIR`	Carpeta de trabajo
`COPY`	Copia archivos al build
`RUN`	Ejecuta comandos al construir
`ENV / ARG`	Variables de entorno / de build
`EXPOSE`	Documenta el puerto de la app
`USER`	Usuario que corre el proceso
`CMD`	Comando por defecto (se puede reemplazar)
`ENTRYPOINT`	Ejecutable fijo del contenedor

Ejemplo: app Node.js en dos etapas

```
# Etapa 1: construir
FROM node:20-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

# Etapa 2: imagen final liviana
FROM node:20-alpine
WORKDIR /app
COPY --from=build /app/dist ./dist
COPY --from=build /app/node_modules ./node_modules
USER node
EXPOSE 3000
CMD ["node", "dist/index.js"]
```

Construir

```
docker build -t miapp:1.0 .
docker build -f Dockerfile.prod -t miapp:prod .
docker build --no-cache -t miapp .
```

`-t` nombre y tag · `.` contexto de build (carpeta actual) · `-f` otro Dockerfile

Pro Tip:
Copia primero `package.json` e instala dependencias, y después el resto del código. Así Docker reutiliza la caché y el build tarda segundos en vez de minutos.

Docker Compose

Define varios servicios en un solo archivo `compose.yaml`

```yaml
services:
  api:
    build: .
    ports:
      - "3000:3000"
    env_file: .env
    depends_on:
      - db
  db:
    image: postgres:16-alpine
    environment:
      POSTGRES_PASSWORD: secreto
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

Compose crea una red para el proyecto: `api` llega a la base con el host `db`.

Comandos de Compose

```
docker compose up -d          # levanta todo
docker compose up -d --build  # reconstruye antes
docker compose ps
docker compose logs -f api
docker compose exec db psql -U postgres
docker compose restart api
docker compose down           # detiene y borra
docker compose down -v  # ⚠ también los volúmenes
docker compose config         # valida el archivo
```

Limpieza

```
docker container prune  # contenedores detenidos
docker image prune -a
docker volume prune
docker network prune
docker builder prune  # caché de builds
docker system prune -a           # ⚠ casi todo
docker system prune -a --volumes  # ⚠ todo
```

Seguridad y buenas prácticas

Tags fijos	Usa `postgres:16-alpine`, no `latest`.
No root	Agrega `USER` en el Dockerfile.
.dockerignore	Excluye `.git`, `.env`, `node_modules`.
Secretos	Nunca en la imagen: usa `.env` o secrets.
Imágenes chicas	Alpine, slim o distroless + multi-stage.
Puertos	Publica solo lo necesario: `-p 127.0.0.1:5432:5432`
Escaneo	Revisa vulnerabilidades antes de desplegar.

```
docker scout quickview miapp:1.0
docker scout cves miapp:1.0
docker run --read-only --cap-drop ALL miapp
```

Pro Tip:
Si un contenedor se cae al iniciar, revisa en orden: `docker ps -a` (¿con qué código salió?) → `docker logs app` → `docker inspect app` → prueba entrar con `docker run -it --entrypoint sh imagen`.