🚀 CHULETA COMPLETA PARA EXAMEN DE DOCKER

==

TODO LO QUE NECESITAS SABER PARA APROBAR

==

📋 ÍNDICE RÁPIDO

---

1. COMANDOS DOCKER ESENCIALES

📦 IMÁGENES

Listar imágenes

docker images

docker image ls

Descargar imagen

docker pull <imagen>:<tag>

```
docker pull nginx:latest
docker pull mysql:8.0

# Construir imagen desde Dockerfile
docker build -t <nombre>:<tag> .
docker build -t mi-app:v1 .
docker build -t mi-app:latest -f Dockerfile.prod .

# Eliminar imagen
docker rmi <imagen>
docker rmi nginx:latest
docker image rm <imagen>

# Ver historial de una imagen
docker history <imagen>

# Inspeccionar imagen
docker inspect <imagen>

# Limpiar imágenes no usadas
docker image prune
docker image prune -a  # Elimina TODAS las no usadas

# Etiquetar imagen
docker tag <imagen-origen> <imagen-destino>
docker tag mi-app:v1 mi-app:latest

# Guardar imagen en archivo tar
```

```
docker save -o imagen.tar mi-app:latest

# Cargar imagen desde archivo tar
docker load -i imagen.tar

### 🐳 CONTENEDORES

# Listar contenedores corriendo
docker ps
docker container ls

# Listar TODOS los contenedores (incluidos detenidos)
docker ps -a
docker container ls -a

# Crear y ejecutar contenedor
docker run <imagen>
docker run -d nginx              # Detached (segundo plano)
docker run -it ubuntu bash       # Interactivo con terminal
docker run -p 8080:80 nginx      # Mapeo de puertos
docker run -v /data:/var/data nginx  # Volumen
docker run --name mi-contenedor nginx # Nombre personalizado
docker run -e VAR=valor nginx    # Variable de entorno
docker run --network mi-red nginx    # Red específica
docker run --rm nginx            # Auto-eliminar al parar

# Parar contenedor
docker stop <contenedor>
```

```
docker stop mi-contenedor
docker stop $(docker ps -q)  # Parar todos

# Iniciar contenedor detenido
docker start <contenedor>

# Reiniciar contenedor
docker restart <contenedor>

# Pausar/despausar contenedor
docker pause <contenedor>
docker unpause <contenedor>

# Eliminar contenedor
docker rm <contenedor>
docker rm -f <contenedor>  # Forzar (aunque esté corriendo)
docker container prune     # Eliminar todos los detenidos

# Ver logs
docker logs <contenedor>
docker logs -f <contenedor>          # Seguir logs en tiempo real
docker logs --tail 100 <contenedor>   # Últimas 100 líneas
docker logs --since 5m <contenedor>   # Últimos 5 minutos

# Ejecutar comando en contenedor corriendo
docker exec <contenedor> <comando>
docker exec mi-contenedor ls -la
docker exec -it mi-contenedor bash    # Terminal interactiva
```

```
docker exec -it mi-contenedor sh     # Si no tiene bash

# Copiar archivos
docker cp <contenedor>:<origen> <destino>
docker cp mi-contenedor:/app/file.txt ./
docker cp ./file.txt mi-contenedor:/app/

# Ver estadísticas de recursos
docker stats
docker stats <contenedor>

# Ver procesos dentro del contenedor
docker top <contenedor>

# Inspeccionar contenedor
docker inspect <contenedor>
docker inspect <contenedor> | grep IPAddress

# Obtener IP de contenedor
docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
<contenedor>

# Ver cambios en el filesystem del contenedor
docker diff <contenedor>

# Crear imagen desde contenedor modificado
docker commit <contenedor> <nueva-imagen>:<tag>
```

🌐 REDES

Listar redes

docker network ls

Crear red

docker network create <nombre>

docker network create --driver bridge mi-red

docker network create --subnet=172.25.0.0/16 mi-red-custom

Inspeccionar red

docker network inspect <red>

Conectar contenedor a red

docker network connect <red> <contenedor>

Desconectar contenedor de red

docker network disconnect <red> <contenedor>

Eliminar red

docker network rm <red>

Limpiar redes no usadas

docker network prune

💾 VOLÚMENES

Listar volúmenes

docker volume ls

Crear volumen

docker volume create <nombre>

Inspeccionar volumen

docker volume inspect <nombre>

Eliminar volumen

docker volume rm <nombre>

Limpiar volúmenes no usados

docker volume prune

🧹 LIMPIEZA GENERAL

Limpiar TODO lo no usado (¡CUIDADO!)

docker system prune

Limpiar TODO incluyendo volúmenes

docker system prune -a --volumes

Ver uso de disco

docker system df

---

2. DOCKERFILE - ESTRUCTURA Y DIRECTIVAS

📝 ESTRUCTURA BÁSICA

```dockerfile
# Comentario
FROM <imagen-base>:<tag>
WORKDIR <directorio>
COPY <origen> <destino>
RUN <comando>
EXPOSE <puerto>
CMD ["ejecutable", "arg1", "arg2"]
```

🔧 DIRECTIVAS PRINCIPALES

FROM - Imagen base (OBLIGATORIA, debe ser la primera)

```dockerfile
FROM ubuntu:22.04
FROM node:18-alpine
FROM python:3.11-slim
FROM scratch  # Imagen vacía
```

WORKDIR - Establece directorio de trabajo

```dockerfile
WORKDIR /app
WORKDIR /var/www/html
```

Todos los comandos siguientes se ejecutan en este directorio

```

**COPY** - Copia archivos del host al contenedor

```dockerfile
COPY . .                # Copia todo

COPY package.json ./        # Copia archivo específico

COPY src/ /app/src/        # Copia directorio

COPY --chown=user:group file.txt /app/  # Con permisos
```

ADD - Similar a COPY pero puede descomprimir y descargar

```dockerfile
ADD archivo.tar.gz /app/    # Descomprime automáticamente

ADD https://url.com/file /app/  # Descarga desde URL

# ⚠️ Preferir COPY cuando sea posible
```

RUN - Ejecuta comandos durante la construcción

```dockerfile
RUN apt-get update && apt-get install -y nginx

RUN npm install

RUN pip install -r requirements.txt

# Formato exec (preferido)

RUN ["npm", "install"]

# Múltiples comandos (mejor práctica)
```
```

```
RUN apt-get update \
  && apt-get install -y \
     git \
     curl \
     vim \
  && apt-get clean \
  && rm -rf /var/lib/apt/lists/*
```

CMD - Comando por defecto al iniciar contenedor

```dockerfile
CMD ["npm", "start"]
CMD ["python", "app.py"]
CMD ["nginx", "-g", "daemon off;"]
# Solo puede haber UN CMD (el último gana)
```

ENTRYPOINT - Punto de entrada principal

```dockerfile
ENTRYPOINT ["docker-entrypoint.sh"]
ENTRYPOINT ["python"]
CMD ["app.py"]  # Se pasa como argumento a ENTRYPOINT

# Resultado: python app.py
```

EXPOSE - Documenta puertos (NO los publica)

```dockerfile
```

EXPOSE 80

EXPOSE 3000

EXPOSE 8080/tcp

EXPOSE 53/udp

```

**ENV** - Variables de entorno

```dockerfile

ENV NODE_ENV=production

ENV PATH="/app/bin:${PATH}"

ENV DB_HOST=localhost \

  DB_PORT=3306

```

ARG - Variables de construcción

```dockerfile

ARG VERSION=latest

ARG BUILD_DATE

FROM node:${VERSION}

RUN echo "Build date: ${BUILD_DATE}"

# Usar: docker build --build-arg VERSION=18 .

```

LABEL - Metadatos

```dockerfile

LABEL maintainer="tu@email.com"

LABEL version="1.0"

LABEL description="Mi aplicación"

```

USER - Usuario para ejecutar comandos

```dockerfile
USER node

USER 1001

USER myuser:mygroup
```

VOLUME - Punto de montaje

```dockerfile
VOLUME /data

VOLUME /var/log

VOLUME ["/data", "/logs"]
```

HEALTHCHECK - Verificación de salud

```dockerfile
HEALTHCHECK --interval=30s --timeout=3s \

  CMD curl -f http://localhost/ || exit 1
```

📋 EJEMPLOS COMPLETOS

Node.js:

```dockerfile
FROM node:18-alpine
```
```

WORKDIR /app

Copiar solo package.json primero (cache)

COPY package*.json ./

RUN npm install

Copiar el resto del código

COPY . .

EXPOSE 3000

CMD ["npm", "start"]

```

**Python:**

```dockerfile

FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["python", "app.py"]

```

Java:

```dockerfile
FROM openjdk:17-jdk-alpine

WORKDIR /app

COPY target/*.jar app.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "app.jar"]
```

Multi-stage build (optimizado):

```dockerfile
# Stage 1: Build
FROM node:18-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# Stage 2: Production
FROM node:18-alpine
```
```

```
WORKDIR /app

COPY --from=builder /app/dist ./dist

COPY package*.json ./

RUN npm install --production

EXPOSE 3000

CMD ["node", "dist/server.js"]
```

⚠️ MEJORES PRÁCTICAS

✅ Usar imágenes base oficiales y versiones específicas

✅ Copiar package.json primero para aprovechar cache

✅ Minimizar el número de capas (combinar RUN)

✅ Limpiar caches después de instalar paquetes

✅ Usar .dockerignore para excluir archivos

✅ No usar root user (USER node)

✅ Multi-stage builds para imágenes pequeñas

❌ No usar latest en producción

❌ No instalar paquetes innecesarios

❌ No copiar archivos sensibles (.env)

❌ No dejar contraseñas hardcodeadas

📄 .dockerignore

```
node_modules
```

```
npm-debug.log
.git
.gitignore
.env
*.md
.vscode
.idea
```

---

3. DOCKER COMPOSE - SINTAXIS COMPLETA

𝄞 ESTRUCTURA BÁSICA

```yaml
version: '3.8'  # Opcional en versiones nuevas

services:
  nombre-servicio:
    # Configuración del servicio

volumes:
  # Definición de volúmenes

networks:
  # Definición de redes
```

```

### 🔧 OPCIONES DE SERVICIOS

**image** - Usar imagen existente

```yaml
services:
 db:
   image: mysql:8.0
   image: postgres:15-alpine
```

build - Construir desde Dockerfile

```yaml
services:
 app:
   build: .                # Dockerfile en directorio actual
   build: ./backend        # Dockerfile en ./backend
   build:
     context: ./backend
     dockerfile: Dockerfile.prod
     args:
      VERSION: "1.0"
```

container_name - Nombre del contenedor

```yaml
services:
```
```

```
 db:
   container_name: mi-mysql
```

restart - Política de reinicio

```yaml
services:
 app:
   restart: no             # Nunca reiniciar
   restart: always         # Siempre reiniciar
   restart: on-failure     # Solo si falla
   restart: unless-stopped    # Siempre excepto si lo paras manualmente
```

ports - Mapeo de puertos

```yaml
services:
 web:
   ports:
     - "80:80"           # host:container
     - "8080:80"
     - "127.0.0.1:3000:3000" # Solo localhost
     - "3000-3005:3000-3005" # Rango
```

expose - Exponer puertos (solo interno)

```yaml
services:
```

```
 backend:
   expose:
     - "3000"
     - "8080"
```

environment - Variables de entorno

```yaml
services:
 db:
   environment:
     MYSQL_ROOT_PASSWORD: root
     MYSQL_DATABASE: mydb
     NODE_ENV: production

   # O desde archivo
   env_file:
     - .env
     - .env.prod
```

volumes - Volúmenes y bind mounts

```yaml
services:
 app:
   volumes:
     # Volumen nombrado
     - mysql_data:/var/lib/mysql
```

```
    # Bind mount (desarrollo)
    - ./src:/app/src

    # Volumen anónimo
    - /app/node_modules

    # Solo lectura
    - ./config:/app/config:ro

volumes:
  mysql_data:
    driver: local
```

networks - Redes
```yaml
services:
  app:
    networks:
      - frontend
      - backend
    # Con configuración avanzada
    networks:
      frontend:
        ipv4_address: 172.16.0.10
        aliases:
          - web
```

```
      - app

networks:
 frontend:
   driver: bridge
 backend:
   driver: bridge
```

depends_on - Dependencias

```yaml
services:
 backend:
   depends_on:
     - db

 # Con condición de salud
 backend:
   depends_on:
     db:
       condition: service_healthy
```

healthcheck - Verificación de salud

```yaml
services:
 db:
   healthcheck:
```

```
    test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
    interval: 10s
    timeout: 5s
    retries: 5
    start_period: 30s
```

command - Sobrescribir CMD

```yaml
services:
 app:
  command: npm run dev
  command: ["npm", "run", "dev"]
```

entrypoint - Sobrescribir ENTRYPOINT

```yaml
services:
 app:
  entrypoint: /app/entrypoint.sh
  entrypoint: ["python", "-u"]
```

working_dir - Directorio de trabajo

```yaml
services:
 app:
  working_dir: /app/src
```

```

**user** - Usuario para ejecutar

```yaml
services:
  app:
    user: "1000:1000"
    user: node
```

stdin_open y **tty** - Terminal interactiva

```yaml
services:
  app:
    stdin_open: true
    tty: true
```

labels - Metadatos

```yaml
services:
  app:
    labels:
      - "com.example.description=Web app"
      - "com.example.version=1.0"
```

logging - Configuración de logs
```

```yaml
services:
 app:
  logging:
   driver: json-file
   options:
    max-size: "10m"
    max-file: "3"
```

deploy - Configuración de despliegue (Swarm)
```yaml
services:
 app:
  deploy:
   replicas: 3
   resources:
    limits:
     cpus: '0.50'
     memory: 512M
    reservations:
     cpus: '0.25'
     memory: 256M
```

📋 TEMPLATE GENÉRICO COMPLETO

```yaml
```

```
version: '3.8'

services:
 # =======================================
 # BASE DE DATOS
 # =======================================
 database:
  image: mysql:8.0              # O postgres:15-alpine
  container_name: app-database
  restart: unless-stopped
  environment:
   MYSQL_ROOT_PASSWORD: rootpass
   MYSQL_DATABASE: appdb
   MYSQL_USER: appuser
   MYSQL_PASSWORD: apppass
  ports:
   - "3306:3306"
  volumes:
   - db_data:/var/lib/mysql
   - ./init.sql:/docker-entrypoint-initdb.d/init.sql
  networks:
   - backend-network
  healthcheck:
   test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
   interval: 10s
   timeout: 5s
   retries: 5
   start_period: 30s
```

```
# ======================================
# BACKEND
# ======================================
backend:
 build:
  context: ./backend
  dockerfile: Dockerfile
 container_name: app-backend
 restart: unless-stopped
 environment:
  DB_HOST: database
  DB_PORT: 3306
  DB_USER: appuser
  DB_PASSWORD: apppass
  DB_NAME: appdb
  NODE_ENV: production
 ports:
  - "3000:3000"
 volumes:
  - ./backend:/app
  - /app/node_modules
 depends_on:
  database:
   condition: service_healthy
 networks:
  - backend-network
  - frontend-network
```

```
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:3000/health"]
    interval: 30s
    timeout: 10s
    retries: 3

# ======================================
# FRONTEND
# ======================================
frontend:
  build:
    context: ./frontend
    dockerfile: Dockerfile
  container_name: app-frontend
  restart: unless-stopped
  ports:
    - "80:80"
  depends_on:
    - backend
  networks:
    - frontend-network
  stdin_open: true
  tty: true

# ======================================
# REDIS (CACHE) - OPCIONAL
# ======================================
cache:
```

```
    image: redis:7-alpine
    container_name: app-cache
    restart: unless-stopped
    ports:
      - "6379:6379"
    networks:
      - backend-network
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s
      timeout: 3s
      retries: 5

# ========================================
# VOLÚMENES
# ========================================
volumes:
  db_data:
    driver: local

# ========================================
# REDES
# ========================================
networks:
  frontend-network:
    driver: bridge
  backend-network:
    driver: bridge
```

```

### 🎯 COMANDOS DOCKER COMPOSE

```bash
# Iniciar servicios
docker-compose up
docker-compose up -d                # Detached
docker-compose up --build           # Reconstruir imágenes
docker-compose up -d --force-recreate   # Forzar recreación

# Detener servicios
docker-compose stop
docker-compose down                 # Detener y eliminar
docker-compose down -v              # Incluir volúmenes

# Ver logs
docker-compose logs
docker-compose logs -f              # Seguir logs
docker-compose logs -f backend      # Solo un servicio
docker-compose logs --tail=100      # Últimas 100 líneas

# Ver estado
docker-compose ps
docker-compose ps -a

# Ejecutar comando
docker-compose exec backend bash
```
```

```
docker-compose exec database mysql -u root -p

# Escalar servicios
docker-compose up -d --scale backend=3

# Validar archivo
docker-compose config
docker-compose config --quiet          # Solo validar

# Reconstruir
docker-compose build
docker-compose build --no-cache

# Reiniciar
docker-compose restart
docker-compose restart backend

# Pausar/Despausar
docker-compose pause
docker-compose unpause

# Ver imágenes
docker-compose images

# Ver procesos
docker-compose top
```

═══

4. REDES Y SUBREDES

🌐 TIPOS DE REDES

bridge (Por defecto)

- Red privada en el host

- Contenedores en la misma red se ven

- DNS automático

host

- Contenedor usa red del host directamente

- Sin aislamiento

- Mejor performance

none

- Sin red

- Aislamiento total

overlay

- Para Docker Swarm

- Múltiples hosts

macvlan

- Asigna MAC address al contenedor

- Aparece como dispositivo físico

📊 SUBREDES Y CIDR

Notación CIDR:

```
172.20.0.0/16
│       │
│       └─ Bits para red (16)
└─ Dirección base
```

Ejemplos:

CIDR	Máscara	IPs disponibles	Rango
/8	255.0.0.0	16,777,214	10.0.0.0 - 10.255.255.255
/16	255.255.0.0	65,534	172.16.0.0 - 172.16.255.255
/24	255.255.255.0	254	192.168.1.0 - 192.168.1.255

Subredes privadas (RFC 1918):

- 10.0.0.0/8 (10.0.0.0 - 10.255.255.255)
- 172.16.0.0/12 (172.16.0.0 - 172.31.255.255)
- 192.168.0.0/16 (192.168.0.0 - 192.168.255.255)

🔧 CONFIGURACIÓN DE REDES

Red simple:

```yaml
```

```
networks:
  app-network:
    driver: bridge
```

Red con subred personalizada:

```yaml
networks:
  app-network:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.25.0.0/16
          gateway: 172.25.0.1
```

IP fija para contenedor:

```yaml
services:
  db:
    networks:
      app-network:
        ipv4_address: 172.25.0.10
```

Múltiples redes:

```yaml
```

```
services:
 backend:
  networks:
   - frontend-net
   - backend-net

networks:
 frontend-net:
  driver: bridge
 backend-net:
  driver: bridge
```

Red externa (ya existe):

```yaml
networks:
 existing-network:
  external: true
  name: mi-red-externa
```

🔍 DNS INTERNO

Docker crea automáticamente DNS para cada red:

```
Servidor DNS: 127.0.0.11 (dentro del contenedor)
```

Resolución automática:

- Nombre del servicio → IP

- Nombre del contenedor → IP

- Aliases → IP

```

**Ejemplo:**

```javascript

// ✅ CORRECTO - Usa nombre del servicio

const host = 'mysql';

// ❌ INCORRECTO - Usa localhost

const host = 'localhost';

```

---

5. VOLÚMENES Y PERSISTENCIA

💾 TIPOS DE VOLÚMENES

Volúmenes nombrados (Recomendado)

```yaml

volumes:

  - db_data:/var/lib/mysql

volumes:

```
  db_data:
    driver: local
```

Bind mounts (Desarrollo)

```yaml
volumes:
  - ./src:/app/src
  - ./config.json:/app/config.json
```

Volúmenes anónimos

```yaml
volumes:
  - /app/node_modules
```

Volúmenes de solo lectura

```yaml
volumes:
  - ./config:/app/config:ro
```

🔧 CONFIGURACIÓN AVANZADA

Driver específico:

```yaml
volumes:
```

```
 db_data:
   driver: local
   driver_opts:
    type: none
    o: bind
    device: /path/on/host
```

Volumen externo:

```yaml
volumes:
 external_volume:
   external: true
   name: my-existing-volume
```

📊 UBICACIÓN FÍSICA

Linux: `/var/lib/docker/volumes/`

Windows: `C:\ProgramData\Docker\volumes\`

Mac: `~/Library/Containers/com.docker.docker/Data/vms/0/`

---

6. ERRORES COMUNES Y SOLUCIONES

❌ ERROR 1: "Port is already allocated"

Problema:

```
ERROR: ... Bind for 0.0.0.0:3306 failed: port is already allocated
```

Solución:

```bash
# Ver qué usa el puerto
netstat -ano | findstr :3306  # Windows
lsof -i :3306              # Linux/Mac

# Cambiar puerto en docker-compose.yml
ports:
  - "3307:3306"  # Usa otro puerto en el host
```

❌ ERROR 2: "services must be a mapping"

Problema:

```yaml
# ❌ MAL
version: '3.8'
mysql:
  image: mysql
```

Solución:

```yaml
# ✅ BIEN
version: '3.8'

services:  # ← Faltaba esto
  mysql:
    image: mysql
```

❌ ERROR 3: "Cannot connect to database"

Problema:
Backend no puede conectar a MySQL

Causas y soluciones:

1. **Usar localhost en lugar del nombre del servicio**

```javascript
// ❌ MAL
host: 'localhost'

// ✅ BIEN
host: 'mysql'  // Nombre del servicio
```

2. **MySQL aún no está listo**

```yaml
# Añadir health check y depends_on
backend:
 depends_on:
   mysql:
    condition: service_healthy
```

3. **Credenciales incorrectas**

```yaml
# Verificar que coincidan
mysql:
 environment:
   MYSQL_USER: myuser
   MYSQL_PASSWORD: mypass

backend:
 environment:
   DB_USER: myuser     # ← Deben coincidir
   DB_PASSWORD: mypass
```

❌ ERROR 4: "Network not found"

Problema:

```
ERROR: Network todolist_default not found
```

Solución:

```bash
# Crear la red manualmente
docker network create todolist_default

# O reconstruir todo
docker-compose down
docker-compose up -d --build
```

❌ ERROR 5: "No space left on device"

Problema:
Docker se quedó sin espacio

Solución:

```bash
# Ver uso de disco
docker system df

# Limpiar todo lo no usado
```

docker system prune -a

Limpiar volúmenes

docker volume prune

Limpiar imágenes

docker image prune -a

```

---

### ❌ ERROR 6: "volume is in use"

**Problema:**

No puedes eliminar un volumen

**Solución:**

```bash

# Ver qué contenedor lo usa

docker ps -a --filter volume=<nombre-volumen>

# Detener y eliminar contenedores

docker-compose down

# Ahora eliminar volumen

docker volume rm <nombre-volumen>

```
```

❌ ERROR 7: "build context too large"

Problema:

Dockerfile intenta copiar muchos archivos

Solución:

Crear `.dockerignore`:

```
node_modules
.git
*.log
.env
dist
build
```

❌ ERROR 8: "healthcheck failed"

Problema:

Contenedor no pasa el health check

Solución:

```bash
# Ver logs del contenedor
```

docker logs <contenedor>

Ejecutar el health check manualmente

docker exec <contenedor> <comando-del-healthcheck>

Ejemplo para MySQL

docker exec mysql mysqladmin ping -h localhost

```

---

### ❌ ERROR 9: "Cannot resolve hostname"

**Problema:**

DNS no resuelve nombres de servicio

**Solución:**

```yaml

# Asegurarte de que estén en la misma red

services:

 backend:

  networks:

   - app-network

 mysql:

  networks:

   - app-network  # ← Deben estar en la misma red

networks:

```
  app-network:
    driver: bridge
```

❌ ERROR 10: "permission denied"

Problema:
Permisos insuficientes en volúmenes

Solución:
```yaml
# Usar usuario específico
services:
  app:
    user: "1000:1000"  # UID:GID

# O dar permisos en el host
chmod -R 777 ./data
```

═══

7. TEMPLATES LISTOS PARA USAR

🎯 TEMPLATE 1: LAMP Stack (Linux, Apache, MySQL, PHP)

```yaml
version: '3.8'

services:
  mysql:
    image: mysql:8.0
    container_name: lamp-mysql
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: myapp
      MYSQL_USER: user
      MYSQL_PASSWORD: pass
    ports:
      - "3306:3306"
    volumes:
      - mysql_data:/var/lib/mysql
    networks:
      - lamp-network

  php:
    image: php:8.2-apache
    container_name: lamp-php
    restart: unless-stopped
    ports:
      - "80:80"
    volumes:
```

```
    - ./www:/var/www/html
  depends_on:
    - mysql
  networks:
    - lamp-network

volumes:
 mysql_data:

networks:
 lamp-network:
  driver: bridge
```

🎯 TEMPLATE 2: MERN Stack (MongoDB, Express, React, Node)

```yaml
version: '3.8'

services:
 mongodb:
  image: mongo:7
  container_name: mern-mongo
  restart: unless-stopped
  environment:
   MONGO_INITDB_ROOT_USERNAME: admin
```

```
    MONGO_INITDB_ROOT_PASSWORD: admin123
   ports:
    - "27017:27017"
   volumes:
    - mongo_data:/data/db
   networks:
    - mern-network

  backend:
   build: ./backend
   container_name: mern-backend
   restart: unless-stopped
   environment:
    MONGO_URI:
mongodb://admin:admin123@mongodb:27017/mydb?authSource=admin
    PORT: 5000
   ports:
    - "5000:5000"
   depends_on:
    - mongodb
   networks:
    - mern-network

  frontend:
   build: ./frontend
   container_name: mern-frontend
   restart: unless-stopped
   ports:
```

```
    - "3000:3000"
  depends_on:
    - backend
  networks:
    - mern-network
  stdin_open: true
  tty: true

volumes:
 mongo_data:

networks:
 mern-network:
  driver: bridge
```

🎯 TEMPLATE 3: PostgreSQL + Node + React

```yaml
version: '3.8'

services:
 postgres:
  image: postgres:15-alpine
  container_name: app-postgres
  restart: unless-stopped
```

```
  environment:
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: postgres
    POSTGRES_DB: appdb
  ports:
    - "5432:5432"
  volumes:
    - pg_data:/var/lib/postgresql/data
  networks:
    - app-network
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -U postgres"]
    interval: 10s
    timeout: 5s
    retries: 5

backend:
  build: ./backend
  container_name: app-backend
  restart: unless-stopped
  environment:
    DATABASE_URL: postgresql://postgres:postgres@postgres:5432/appdb
    PORT: 4000
  ports:
    - "4000:4000"
  depends_on:
    postgres:
      condition: service_healthy
```

```
    networks:
      - app-network

  frontend:
    build: ./frontend
    container_name: app-frontend
    restart: unless-stopped
    ports:
      - "3000:3000"
    depends_on:
      - backend
    networks:
      - app-network

volumes:
  pg_data:

networks:
  app-network:
    driver: bridge
```

🎯 TEMPLATE 4: WordPress + MySQL

```yaml
version: '3.8'
```

```
services:
 mysql:
  image: mysql:8.0
  container_name: wordpress-mysql
  restart: unless-stopped
  environment:
   MYSQL_ROOT_PASSWORD: rootpass
   MYSQL_DATABASE: wordpress
   MYSQL_USER: wpuser
   MYSQL_PASSWORD: wppass
  volumes:
   - mysql_data:/var/lib/mysql
  networks:
   - wordpress-network

 wordpress:
  image: wordpress:latest
  container_name: wordpress-app
  restart: unless-stopped
  ports:
   - "8080:80"
  environment:
   WORDPRESS_DB_HOST: mysql
   WORDPRESS_DB_USER: wpuser
   WORDPRESS_DB_PASSWORD: wppass
   WORDPRESS_DB_NAME: wordpress
  volumes:
```

```
    - wordpress_data:/var/www/html
   depends_on:
    - mysql
   networks:
    - wordpress-network

volumes:
 mysql_data:
 wordpress_data:

networks:
 wordpress-network:
   driver: bridge
```

🎯 TEMPLATE 5: Django + PostgreSQL + Redis

```yaml
version: '3.8'

services:
 postgres:
   image: postgres:15-alpine
   container_name: django-postgres
   restart: unless-stopped
   environment:
```

```
    POSTGRES_DB: djangodb
    POSTGRES_USER: django
    POSTGRES_PASSWORD: django123
  volumes:
    - pg_data:/var/lib/postgresql/data
  networks:
    - django-network

redis:
  image: redis:7-alpine
  container_name: django-redis
  restart: unless-stopped
  networks:
    - django-network

django:
  build: ./backend
  container_name: django-app
  restart: unless-stopped
  command: python manage.py runserver 0.0.0.0:8000
  environment:
    DATABASE_URL: postgresql://django:django123@postgres:5432/djangodb
    REDIS_URL: redis://redis:6379/0
  ports:
    - "8000:8000"
  volumes:
    - ./backend:/app
  depends_on:
```

```
    - postgres
    - redis
  networks:
    - django-network

volumes:
 pg_data:

networks:
 django-network:
  driver: bridge
```

---

8. TRUCOS Y TIPS PARA EL EXAMEN

🎓 CONCEPTOS CLAVE QUE DEBES SABER

1. Diferencia entre imagen y contenedor

- Imagen = Plantilla (inmutable)

- Contenedor = Instancia ejecutándose (mutable)

2. Diferencia entre COPY y ADD

- COPY: Solo copia archivos

- ADD: Copia + descomprime + descarga URLs

3. Diferencia entre CMD y ENTRYPOINT

- CMD: Comando por defecto (se puede sobrescribir)

- ENTRYPOINT: Punto de entrada fijo

- Se pueden combinar

4. Diferencia entre RUN y CMD

- RUN: Se ejecuta durante BUILD

- CMD: Se ejecuta al START del contenedor

5. Volúmenes vs Bind Mounts

- Volúmenes: Gestionados por Docker, persistentes

- Bind Mounts: Carpeta del host montada

6. Redes bridge vs host

- Bridge: Red aislada con DNS

- Host: Usa red del host directamente

7. depends_on vs healthcheck

- depends_on: Orden de inicio (no espera a que esté listo)

- healthcheck: Verifica que el servicio esté funcionando

🔥 COMANDOS MÁS IMPORTANTES

```bash

# Los 10 comandos que DEBES saber

docker ps -a

docker images

docker build -t nombre .
```

```
docker run -d -p 8080:80 nginx
docker exec -it contenedor bash
docker logs -f contenedor
docker-compose up -d
docker-compose down
docker network inspect red
docker volume ls
```

⚡ SHORTCUTS Y ATAJOS

```bash
# Parar todos los contenedores
docker stop $(docker ps -q)

# Eliminar todos los contenedores
docker rm $(docker ps -aq)

# Eliminar todas las imágenes
docker rmi $(docker images -q)

# Ver solo IDs
docker ps -q

# Formato personalizado
docker ps --format "table {{.ID}}\t{{.Names}}\t{{.Status}}"

# Ejecutar y eliminar automáticamente
```

docker run --rm -it ubuntu bash

Ver logs de los últimos 100 líneas

docker logs --tail 100 contenedor

Copiar archivo

docker cp contenedor:/path/file.txt ./

```

### 📝 CHECKLIST PRE-EXAMEN

```

✅ Sé crear un Dockerfile básico

✅ Sé construir una imagen

✅ Sé ejecutar un contenedor con puertos mapeados

✅ Sé crear un docker-compose.yml

✅ Entiendo la diferencia entre services, volumes y networks

✅ Sé configurar variables de entorno

✅ Sé crear health checks

✅ Entiendo depends_on

✅ Sé crear redes personalizadas

✅ Sé asignar IPs fijas

✅ Sé crear volúmenes para persistencia

✅ Sé ver logs y depurar

✅ Sé limpiar recursos de Docker

✅ Conozco los errores comunes y sus soluciones

```
```

🎯 ESTRUCTURA TÍPICA DE EXAMEN

Parte 1: Dockerfile

- Te darán una aplicación y debes crear el Dockerfile

- Recuerda: FROM, WORKDIR, COPY, RUN, EXPOSE, CMD

Parte 2: Docker Compose

- Te pedirán orquestar múltiples servicios

- Recuerda: services, volumes, networks, depends_on, healthcheck

Parte 3: Redes

- Configurar comunicación entre contenedores

- Recuerda: mismo network, usar nombre de servicio, no localhost

Parte 4: Troubleshooting

- Te darán un error y debes solucionarlo

- Recuerda: revisar logs, verificar redes, puertos, variables

💡 ÚLTIMO CONSEJO

En el examen:

1. Lee TODO el enunciado antes de empezar

2. Dibuja la arquitectura en papel

3. Identifica: frontend, backend, database

4. Define qué servicios necesitas

5. Piensa en las dependencias (depends_on)

6. Configura las redes

7. No olvides los volúmenes para persistencia

8. Prueba con docker-compose config antes de up

9. Si algo falla, mira los logs

Comandos de verificación:

```bash
docker-compose config          # Validar sintaxis
docker-compose up -d           # Ejecutar
docker-compose ps              # Ver estado
docker-compose logs -f         # Ver logs
docker network inspect <red>   # Ver red
curl http://localhost:puerto   # Probar endpoint
```

🎁 BONUS: DOCKER-COMPOSE.YML GENÉRICO UNIVERSAL

```yaml
version: '3.8'

#
================================================================
==========
# SERVICIOS
#
================================================================
==========
services:
```

```
 #
================================================================
========
 # BASE DE DATOS (Elige una)
 #
================================================================
========


 # MySQL
 mysql:
  image: mysql:8.0
  container_name: app-mysql
  restart: unless-stopped
  environment:
   MYSQL_ROOT_PASSWORD: rootpass
   MYSQL_DATABASE: appdb
   MYSQL_USER: appuser
   MYSQL_PASSWORD: apppass
  ports:
   - "3306:3306"
  volumes:
   - mysql_data:/var/lib/mysql
   - ./init.sql:/docker-entrypoint-initdb.d/init.sql
  networks:
   - database-network
  healthcheck:
   test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
   interval: 10s
```

```
    timeout: 5s
    retries: 5
    start_period: 30s

# PostgreSQL (alternativa)
# postgres:
#   image: postgres:15-alpine
#   container_name: app-postgres
#   restart: unless-stopped
#   environment:
#     POSTGRES_USER: postgres
#     POSTGRES_PASSWORD: postgres
#     POSTGRES_DB: appdb
#   ports:
#     - "5432:5432"
#   volumes:
#     - pg_data:/var/lib/postgresql/data
#   networks:
#     - database-network
#   healthcheck:
#     test: ["CMD-SHELL", "pg_isready -U postgres"]
#     interval: 10s
#     timeout: 5s
#     retries: 5

# MongoDB (alternativa)
# mongodb:
#   image: mongo:7
```

```
  #   container_name: app-mongo
  #   restart: unless-stopped
  #   environment:
  #     MONGO_INITDB_ROOT_USERNAME: admin
  #     MONGO_INITDB_ROOT_PASSWORD: admin123
  #   ports:
  #     - "27017:27017"
  #   volumes:
  #     - mongo_data:/data/db
  #   networks:
  #     - database-network


  #
================================================================
========
  # BACKEND
  #
================================================================
========
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: app-backend
    restart: unless-stopped
    environment:
      # MySQL
      DB_HOST: mysql
      DB_PORT: 3306
```

```
    DB_USER: appuser
    DB_PASSWORD: apppass
    DB_NAME: appdb

    # PostgreSQL (si usas postgres)
    # DATABASE_URL: postgresql://postgres:postgres@postgres:5432/appdb

    # MongoDB (si usas mongo)
    # MONGO_URI:
mongodb://admin:admin123@mongodb:27017/appdb?authSource=admin

    NODE_ENV: production
    PORT: 3000
  ports:
    - "3000:3000"
  volumes:
    - ./backend:/app
    - /app/node_modules
  depends_on:
    mysql:
      condition: service_healthy
    # postgres:
    #   condition: service_healthy
  networks:
    - backend-network
    - database-network
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:3000/health"]
```

```
    interval: 30s
    timeout: 10s
    retries: 3
    start_period: 40s

 #
==========================================================================
========
 # FRONTEND
 #
==========================================================================
========
 frontend:
   build:
     context: ./frontend
     dockerfile: Dockerfile
   container_name: app-frontend
   restart: unless-stopped
   ports:
     - "80:80"       # Producción (nginx)
     # - "4200:4200"  # Desarrollo (Angular)
     # - "3000:3000"  # Desarrollo (React)
   volumes:
     - ./frontend:/app
     - /app/node_modules
   depends_on:
     - backend
   networks:
     - backend-network
```

```
    - frontend-network
  stdin_open: true
  tty: true

 #
==========================================================================
========
 # CACHE (OPCIONAL)
 #
==========================================================================
========
 redis:
  image: redis:7-alpine
  container_name: app-redis
  restart: unless-stopped
  ports:
   - "6379:6379"
  networks:
   - backend-network
  healthcheck:
   test: ["CMD", "redis-cli", "ping"]
   interval: 10s
   timeout: 3s
   retries: 5

 #
==========================================================================
========
 # PROXY REVERSO (OPCIONAL)
```

```
 #
================================================================
========
 nginx:
   image: nginx:alpine
   container_name: app-nginx
   restart: unless-stopped
   ports:
    - "80:80"
    - "443:443"
   volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf:ro
    - ./ssl:/etc/nginx/ssl:ro
   depends_on:
    - frontend
    - backend
   networks:
    - frontend-network


#
================================================================
==========
# VOLÚMENES
#
================================================================
==========
volumes:
  mysql_data:
   driver: local
  # pg_data:
```

```
  #   driver: local
  # mongo_data:
  #   driver: local


#
==========================================================================
==========
# REDES
#
==========================================================================
==========
networks:
  frontend-network:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.25.0.0/24
          gateway: 172.25.0.1

  backend-network:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.26.0.0/24
          gateway: 172.26.0.1

  database-network:
```

```
  driver: bridge
  ipam:
    driver: default
    config:
      - subnet: 172.27.0.0/24
        gateway: 172.27.0.1
```

✅ FIN DE LA CHULETA

¡MUCHA SUERTE EN TU EXAMEN! 🍀

Recuerda:

- Lee bien el enunciado
- Dibuja la arquitectura
- Valida con docker-compose config
- Revisa los logs si algo falla
- No olvides los volúmenes para persistencia
- Usa nombres de servicio, no localhost
- Verifica que todo esté en la misma red

¡TÚ PUEDES! 💪
