



## Commandos Básicos

Descargar una imagen

```
docker pull IMAGE_NAME
docker pull IMAGE_NAME:TAG
docker pull postgres
docker pull postgres:15.1
```

Correr un contenedor: en el puerto 80, en el background con la imagen "getting-started"

```
docker container run IMAGE_NAME
docker container run -d -p 80:80 docker/getting-started
```

**-d**: Corre la imagen desenlazada de la consola donde se ejecutó el comando.

**-p 80:80**: Mapea el puerto 80 de nuestra computadora con el puerto 80 del contenedor.

**docker/getting-started**: imagen a usar

### Pro Tip:

Puedes combinar banderas:

```
docker container run -dp 80:80 docker/getting-started
docker run -dp 80:80 docker/getting-started
```

Obtener ayuda de un comando

```
docker <comando> --help
```

Asignar un nombre al contenedor

```
docker container run --name myName docker/getting-started
```

Mostrar un listado de todos los contenedores corriendo en el equipo

```
docker container ls
docker ps
```

Mostrar todos los contenedores del equipo

```
docker container ls -a
docker ps -a
```

Detener un contenedor y eliminarlo

```
docker container stop <container-id>
docker container rm <container-id>
```

Iniciar un contenedor previamente creado

```
docker container start <container-id>
```

### Pro Tip:

Detener el contenedor (o varios) y removerlos de forma forzada.

```
docker container rm -f <container-id> o <ID1 ID2...>
```

Container-id: Puede ser los primeros 3 dígitos

Autenticarte en [docker.hub](https://docs.docker.com/docker-hub/)

```
docker login -u <TU USUARIO>
```

O bien, puedes [crear tokens de acceso](#) específicos.

## Imágenes

Construir y asignar un tag a la imagen: El objetivo del tag es que sea fácil de identificar y leer por humanos

```
docker build -t getting-started .
```

**-t**: Asigna el tag name

**.**: Indica a dónde buscar el archivo DockerFile en el directorio actual.

Renombrar una imagen local

```
docker image tag SOURCE[:TAG] TARGET_IMAGE[:TAG]
docker tag IMAGE NEW_IMAGE
docker tag <Tag Actual> <USUARIO>/<NUEVO NOMBRE>
```

```
docker tag getting-started YOUR-USERNAME/getting-started
```

Este comando es común para desplegar imágenes en diferentes registros.

Si se olvida el número de versión o lo quieres colocar

```
docker image tag IMAGEN IMAGEN:2.0.0
```

## Limpieza de imágenes

Listado de todas las imágenes

```
docker images
```

Eliminar una imagen específica

```
docker image rm <image-ID> o <ID1 ID2 ID3...>
docker rmi IMAGE
docker rmi getting-started
```

Eliminar imágenes colgadas

```
docker image prune
```

Borrar todas las imágenes no usadas

```
docker image prune -a
```

## Logs y examinar contenedores

Mostrar logs de un contenedor

```
docker container logs <container id>
docker container logs --follow CONTAINER
```

**--follow**: **Follows**: Seguir los nuevos logs mostrados  
Mostrar estadísticas y consumo de memoria

```
docker stats
```

Iniciar un comando **shell** dentro del contenedor.

```
docker exec -it CONTAINER EXECUTABLE
docker exec -it web bash
docker exec -it web /bin/sh
```

**-it**: Interactive Terminal



## Volumes

Hay 3 tipos de volúmenes, son usados para hacer persistente la data entre reinicios y levantamientos de imágenes.

### Named Volumes

Este es el volumen más usado.

#### Crear un nuevo volumen

```
docker volume create todo-db
```

#### Listar los volúmenes creados

```
docker volume ls
```

#### Inspeccionar el volumen específico

```
docker volume inspect todo-db
```

#### Remueve todos los volúmenes no usados

```
docker volume prune
```

#### Remueve uno o más volúmenes especificados

```
docker volume rm VOLUME_NAME
```

#### Usar un volumen al correr un contenedor

```
docker run -v todo-db:/etc/todos getting-started
```

### Bind volumes - Vincular volúmenes

Bind volumes trabaja con paths absolutos

#### Terminal

```
docker run -dp 3000:3000 \
-w /app -v "$(pwd):/app" \
node:18-alpine \
sh -c "yarn install && yarn run dev"
```

#### Powershell

```
docker run -dp 3000:3000 `
-w /app -v "$(pwd):/app" `
node:18-alpine `
sh -c "yarn install && yarn run dev"
```

**-w /app: Working directory:** donde el comando empezará a correr.

**-v "\$(pwd):/app": Volumen vinculado:** vinculamos el directorio del host con el directorio /app del contenedor

**node:18-alpine: Imagen a usar**

**sh -c "yarn install && yarn run dev": Comando Shell:** Iniciamos un shell y ejecutamos **yarn install** y luego correr el **yarn run dev**

### Anonymous Volumes

Volúmenes donde sólo se especifica el path del contenedor y Docker lo asigna automáticamente en el host

```
docker run -v /var/lib/mysql/data
```

## Container Networking

### Regla de oro:

Si dos o más contenedores están en la misma red, podrán hablar entre sí. Si no lo están, no podrán.

#### Ver comandos de network

```
docker network
```

#### Crear una nueva red

```
docker network create todo-app
```

#### Listar todas las redes creadas

```
docker network ls
```

#### Inspeccionar una red

```
docker network inspect <NAME o ID>
```

#### Borrar todas las redes no usadas

```
docker network prune
```

#### Correr una imagen y unirla a la red

```
docker run -d \
--network todo-app --network-alias mysql \
-v todo-mysql-data:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=secret \
-e MYSQL_DATABASE=todos \
mysql:8.0
```

#### Powershell

```
docker run -d `
--network todo-app --network-alias mysql `
-v todo-mysql-data:/var/lib/mysql `
-e MYSQL_ROOT_PASSWORD=secret `
-e MYSQL_DATABASE=todos `
mysql:8.0
```

**-e: Variable de entorno:** MySQL necesita definir el root password y el nombre de la base de datos.

**-v todo-mysql-data:/var/lib/mysql: Volumen con nombre** hacia donde **MySQL** graba la base de datos.

**--network-alias: Crea un nombre** nuestra aplicación solo necesita conectarse a un host llamado mysql y se comunicará con la base de datos.

### Puertos y Paths

Cuando vean cosas como: **-p 6000:6379**

Recuerden que es

**HOST : CONTAINER**



## Dockerfile

Comandos comunes en este tipo de archivo. El orden de cada comando es importante, especialmente si se quiere manejar correctamente el caché de capas. (Lo que menos cambia arriba y lo que mas cambia abajo)

**Herencia:** Este paso basa nuestra imagen a crear, a partir de otra en particular.

```
FROM node:18.3.1
```

**Asignar alias (Multi-State):** Se asigna un alias a esta etapa llamada "builder", la cual permite realizar un multi-stage build

```
FROM node:18.3.1 AS builder
```



**Especificar la plataforma:** Útil para M1/M2 Macs

```
FROM --platform=linux/amd64 node:18-alpine
```

**Variables y uso:** se crea una variable llamada APP\_HOME con el valor de "/app"

```
ENV APP_HOME /app
RUN mkdir $APP_HOME
```

**Inicialización:** Se indica que se deben de descargar e instalar los módulos de node.

```
RUN npm install
RUN yarn install --frozen-lockfile
```

**Working directory:** Establece que partir de este punto, estamos en el directorio especificado, es como cambiarse de directorio via comando.

```
WORKDIR /app
```

**Punto de montaje:** Este punto de montaje se asignará a una ubicación en el host que se especifica cuando se crea el contenedor o, si no se especifica, se elige automáticamente desde un directorio creado en /var/lib/docker/volumes.

```
VOLUME ["/data"]
```

**Copiar archivos a una imagen:** Copia el archivo local file.xyz de mi equipo al working directory especificado seguido de /file.xyz

```
ADD file.xyz /file.xyz
```

**También se puede:** Copia los archivos package.json y yarn.lock al root del contenedor

```
COPY package.json yarn.lock ./
```

**Copia todos los archivos y directorios:** de mi proyecto hacia el working directory del contenedor, excluyendo lo especificado en el archivo .dockerignore

```
COPY . .
```

**Expose:** informa a Docker que el contenedor escucha en los puertos de red especificados en tiempo de ejecución

```
EXPOSE 3000
```

**Comando:** especifica la instrucción que se ejecutará cuando se inicie un contenedor Docker.

```
CMD [ "node", "dist/main" ]
```

Es buena practica reconstruir de vez en cuando toda la imagen.

```
docker build --no-cache -t myImage:myTag
```

**BuildX:** Información relacionada a la creación de multiples arquitecturas con un solo comando.

## Docker Compose

Docker Compose es una herramienta que se desarrolló para ayudar a definir y compartir aplicaciones de varios contenedores. Ejemplo: de docker-compose.yml

```
# Versión a usar en el docker-compose
version: '3'

services:
  # Nombre del servicio
  anylistapp:

    # Depende de otro servicio
    depends_on:
      - db

    # Construcción de imagen
    build:
      # Path del dockerfile (./path/)
      context: .
      dockerfile: Dockerfile

    # Nombre de la imagen a usar
    image: node:18-alpine

    # Ejecutar un comando shell dentro
    command: sh -c "yarn install && yarn run dev"

    # Directorio a de trabajo dentro del contenedor
    working_dir: /app

    # Nombre del contenedor
    container_name: AnylistApp

    # Reiniciar el contenedor si se detiene
    restart: always
    # Puertos: MI_EQUIPO : CONTENEDOR
    ports:
      - 8080:3000
```



```
# Variables de entorno
environment:
  # variable de entorno en duro
  STATE: prod
  # ${DB_PASSWORD} proviene de un archivo
  # de variables de entorno .env
  DB_PASSWORD: ${DB_PASSWORD}
  DB_NAME: ${DB_NAME}
  DB_HOST: ${DB_HOST}
  DB_PORT: ${DB_PORT}
  DB_USERNAME: ${DB_USERNAME}
  JWT_SECRET: ${JWT_SECRET}
  PORT: ${PORT}

# Paths relativos aquí
volumes:
  - ./:/app
```

Pueden otro volumen al final o dentro de cada servicio.

```
version: '3'

services:
  app:
  database:
  bucket:

volumes:
  todo-mysql-data:
```

Levantar y ejecutar el comando

```
docker compose up -d
```

**-d**: Corre desenlazado de la consola donde se ejecutó el comando.

Revisar logs de los contenedores levantados con el compose

```
docker compose logs -f
```

**-f**: Follows: Seguir los nuevos logs mostrados

Nomenclatura de los contenedores usados en el docker compose

```
<project-name>_<service-name>_<replica-number>
```

**Limpiar todo**, los contenedores se detendrán y la red se removerá

```
docker compose down
```

## Best Practices

### Escaneo de imagen

Después de construir una imagen, es buena práctica realizar un scan en ella para buscar vulnerabilidades.

```
docker scan getting-started
docker scan getting-started:1.0.0
```

### Capas de la imagen

Cada imagen se construye basado en capas, cada capa permite un nivel de abstracción independiente. Se puede rastrear así

```
docker image history getting-started
```

### Caché de capas

Una vez que cambia una capa, todas las capas posteriores también deben volver a crearse, esto atrasa los tiempos de construcción.

### Multi-Stage builds

Permiten separar dependencias necesarias para construir la app y las necesarias para correr la aplicación en producción, y reduce el tamaño de la imagen final. Ejemplo:

```
FROM node:18 AS build
WORKDIR /app
COPY package* yarn.lock ./
RUN yarn install
COPY public ./public
COPY src ./src
RUN yarn run build

FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
```

Aquí, estamos usando una imagen de Node 18 para realizar la compilación (maximizando el almacenamiento en caché de capas) y luego copiando la salida en un contenedor nginx.

### Crear contenedores efímeros

Esto quiere decir que el contenedor se puede detener y destruir, luego reconstruir y reemplazar con una instalación y configuración mínimas absolutas.

### Desacoplar aplicaciones

Cada contenedor debe tener una sola preocupación. El desacoplamiento de aplicaciones en varios contenedores facilita el escalado horizontal y la reutilización de contenedores.

Continuar la lectura de [mejores prácticas](#) aquí



## Glosario de Términos

### Docker

Docker es una herramienta diseñada para facilitar la creación, implementación y ejecución de aplicaciones mediante el uso de contenedores.

### Container - Contenedor

Es una instancia de una imagen ejecutándose en un ambiente aislado.

### Image - Imagen de contenedor

Es un archivo construido por capas, que contiene todas las dependencias para ejecutarse, tal como: las dependencias, configuraciones, scripts, archivos binarios, etc.

### Dockerizar una aplicación

Proceso de tomar un código fuente y generar una imagen lista para montar y correrla en un contenedor.

### Dockerfile

Un archivo de texto con instrucciones necesarias para crear una imagen. Se puede ver como un blueprint o plano para su construcción.

[Más info aquí.](#)

### .dockerignore (archivo)

Similar al .gitignore, el .dockerignore especifica todo lo que hay que ignorar en un proceso de construcción (**build**)

### docker-compose.yml

Archivo para definir los servicios y con un solo comando en lugar de definir todo directamente en la consola.

### Volumes - Volúmenes

Proporcionan la capacidad de conectar rutas específicas del sistema de archivos del contenedor a la máquina host.

Si se monta un directorio en el contenedor, los cambios en ese directorio también se ven en la máquina host.

## Alpine - Linux

Alpine Linux es una distribución de Linux ligera y orientada a la seguridad basada en musl libc y busybox.

## Nginx

Es un servidor web que también se puede utilizar como proxy inverso, balanceador de carga, proxy de correo y caché HTTP. El software fue creado por Igor Sysoev y lanzado al público en 2004. Nginx es un software gratuito y de código abierto.

## Container Orchestration

La orquestación de contenedores es la automatización de gran parte del esfuerzo operativo requerido para ejecutar cargas de trabajo y servicios en contenedores. Ejemplos de herramientas de orquestación son Kubernetes, Swarm, Nomad, and ECS.

## Docker Layers - Capas

Las capas son el resultado de la forma en que se construyen las imágenes de Docker. Cada paso en un Dockerfile crea una nueva "**capa**" que es esencialmente una diferencia de los cambios en el sistema de archivos desde el último paso.

## Snyk

Es una plataforma de seguridad para desarrolladores para proteger el código, las dependencias, los contenedores y la infraestructura como código.

## Registry - Registro

Es una aplicación del lado del servidor altamente escalable y sin estado que almacena y le permite distribuir imágenes de Docker.

## Docker Daemon

Es el servicio en segundo plano que se ejecuta en el host que administra la creación, ejecución y distribución de contenedores Docker.

