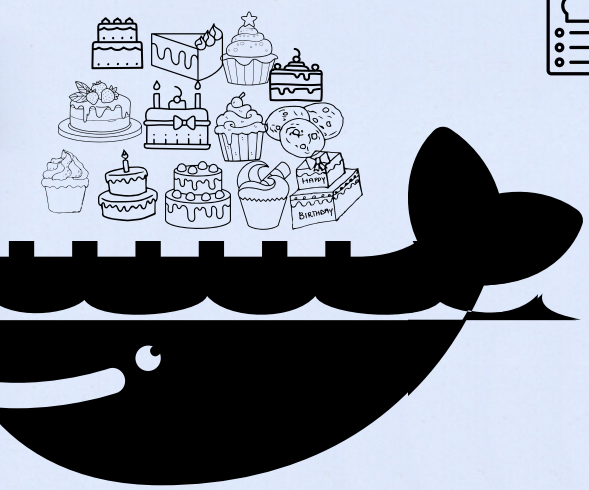
docker CHEATSHEET

Docker es una herramienta que permite a desarrolladores/administradores de sistemas/etc. implementar fácilmente sus aplicaciones en un entorno limitado (contenedores) para ejecutarlas en un S.O principal (host). Permite empaquetar una aplicación con todas sus dependencias en una unidad estandarizada para el desarrollo de software.

Los **contenedores** (pastel) son entornos ligeros y portátiles que contiene aplicaciones y lo necesario para hacerlas funcionar.



Las **imágenes** (receta) son paquetes que contienen todo lo necesario para que nuestra aplicación funcione.

El **Docker Daemon** (chef) es el servicio en 2º plano que se gestiona la creación, ejecución y distribución de los contenedores (pastel)

El **Docker Client** (camarero) es la herramienta de línea de comandos que permite al usuario interactuar con el daemon (chef).

El **Docker Hub** (mercado) es un registro de imágenes (recetas), como un directorio de todas las imágenes disponibles (propias o de otrxs) disponibles para extraerlas y usarlas.

Tipo de imágenes principales

- BASE: imagenes mínimas que contienen solo el SO y proporcionan un punto de partida limpio para construir las aplicaciones.
- SECUNDARIAS: se basan en las BASE y agregan funcionalidad adicional.
- OFICIALES: son imágenes mantenidas y respaldadas oficialmente por DOCKER
- USUARIO: son imágenes creadas y compartidas por usuarios basadas en una imagen BASE. Su formato suele ser user/image-name

Tipo de archivos principales

DOCKER-COMPOSE.yml

```yaml
version: '3'
services:
  web:
    image: nginx:latest
    ports:
      - "8080:80"
  db:
    image: postgres:latest
    environment:
      POSTGRES_PASSWORD: example
```

Es un archivo YAML que define la condiguración de varios servicios, redes y volúmenes. Permite definir y ejecutar aplicaciones multi-contenedor.

Facilita la gestión de múltiples contenedores y sus configuraciones, sobretodo es útil para aplicaciones complejas que requieren varios servicios que trabajan juntos.

DOCKERFILE

Archivo de texto simple que contiene una lista de comandos que el Docker client llama mientras crea una imagen. Se guarda sin extensión.

Se usa para describir cómo debe ser la configuración del entorno dentro del contenedor (aplicaciones a instalar, archivos a copiar, cómo ejecutarse,...)

```dockerfile
# Usa una imagen base
FROM ubuntu:18.04

# Instala paquetes
RUN apt-get update && apt-get install -y nginx

# Copia archivos al contenedor
COPY index.html /var/www/html/

# Define el comando a ejecutar al iniciar el contenedor
CMD ["nginx", "-g", "daemon off;"]
```

Cada contenedor debería de tener sus propios servicios ya que estos pueden tener necesidades distintas y diferentes ritmos. Por eso se crearon los **entornos multicontenedores**, una arquitectura de aplicación donde multiples contenedores docker trabajan juntos para proporcionar una aplicación completa o servicio. Algunas de sus caracteristicas clave son:

1. Microservicios
2. Independencia y escalabilidad
3. Desarrollo y mantenimiento modular
4. Fácil implementación y orquestación
5. Aislamiento y seguridad
6. Reusabilidad de contenedores
7. Gestión de dependencias
8. Escala horizontal

Para poderse comunicar entre contenedores existen las **redes Docker,** que permiten la comunicación tanto entre contenedores cómo contenedor y host o mundo exterior. Por defecto, cuando instalas Docker se crean 3 básicas: bridge, host i none. Algunas otras que admite son: overlay, macvlan y personalizadas.

HOST: conecta el contenedor a la red del host, eliminando el aislamiento de la red.

BRIDGE: permite la comunicación entre contenedores en el mismo host

NONE: no asigna interfaz de red al contenedor, dejándolo completamente aislado.

OVERLAY: facilita la comunicación entre contenedores en hosts diferentes

MACVLAN: asigna una dirección MAC única a cada contenedor, permitiendo la comunicación directa en la red física.

PERSONALIZADA: para aislar grupos de contenedores y controlar cómo se comunican.

COMANDOS PRINCIPALES

- Para instalar DOCKER

ANTES DE EMPEZAR

$ docker run hello-world
Comprobar que docker funciona tras su instalación

$ docker login
Loguearse en docker hub

$ docker help
Ver los comandos y opciones principales de docker

$ docker help COMMAND
Obtener más información sobre un comando concreto

CONTENEDORES

$ docker start/restart contenedor
Iniciar o reiniciar un contenedor

$ docker stop/pause/unpause contenedor
Detener, pausar o reanudar un contenedor en ejecución

$ docker port contenedor
Mostrar los mapeos de puertos del contenedor

$ docker container logs [opciones] contenedor
Listar los registros de un contenedor en ejecución

- -f o --follow: sigue los registros en tiempo real
- --tail N: Muestra las últimas N línieas de registros

$ docker ps
Listar los contenedores que están ejecutándose

$ docker ps -a
Listar los contenedores activos y inactivos

$ docker rm contenedor1 N / $(docker ps -a -q -f status=exited)
Eliminar los contenedores 1, N o que cumplan cierta condición

- -q: devuelve los ID numéricos
- -f: filtra la salida según las condiciones proporcionadas

$ docker container prune
Eliminar todos los contenedores parados **y** (confirmar) o **N** (cancelar)

IMÁGENES

$ docker images
Listar todas las imágenes de tu sistema
- REPOSITORY: nombre de la imagen
- TAG: instantánea particular de una imagen
- IMAGE ID: identificador único de una imagen
- CREATED: momento en que se creo dicha imagen

$ docker search registro
Listar las imágenes que hay dentro de un registro

$ docker rmi imagen
Eliminar una imagen

$ docker build -t image .
Construir una imagen a partir de un dockerfile y etiquetarla **(-t)**

$ docker push imagen
Publicar una imagen local en un registro
(Es importante tener el formato user/image_name para que el docker client sepa dónde publicar)

$ docker pull imagen:version
Recupera la imagen de un registro y la guarda en nuestro sistema
(Si no se indica, latest es la versión predeterminada)

COMANDOS CON RUN

$ docker run [opciones] imagen [comando] [argumentos]
Ejecutar un contenedor, especificando (o no) un comando que se ejecutará dentro del contenedor. Si la imagen no tiene un comando predeterminado, el contenedor se ejecutará y se cerrará al instante
- --rm: opción que elimina el contenedor una vez que se sale de él o termina el proceso de dentro (útil en contenedores temporales)
- Ctrl+C: Salir de la ejecución interactiva de un contenedor

$ docker run -it imagen sh
Ejecutar más de un comando en un mismo contenedor
- -it: asigna un pseudo-TTY(-t) e inicia sesión de Shell interactiva (-i)
- sh: inicia un intérprete de comandos tipo Shell dentro del contenedor
- /xxxx: comando a ejecutar dentro del contenedor (ej: exit o ctrl+D para cerrar sesión interactiva y finalizar el contenedor)

$ docker run -p [puerto_host]:[puerto_contenedor] imagen
Ejecutar una aplicación o servicio dentro del contenedor especificando los puertos de nuestro host que debe redirigir al contenedor
- localhost:[puerto_host]: para ver localmente nuestra aplicación o servicio

$ docker run -d -P --name [Nombre_contenedor] imagen
- -d: indica que el contenedor se debe ejecutar en 2º plano
- -P: mapea todos los puertos expuestos en el contenedor a aleatorios disponibles en el host
- --name: asigna un nombre al contenedor

$ docker run -it --rm --net red imagen bash
Iniciar un contenedor de la imagen, conectándolo a una red y abrir una sesión interactiva de bash dentro del mismo contendor
- --net red: conectar el contenedor a una red específica

$ docker run -d --name es --net red -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node" imagen
Crear un contenedor (es) en 2º plano, conectarlo a la red, mapeando los puertos 9200 y 9300 de es al host configurando la imagen para ejecutarse en modo de un único nodo (entornos de desarrollo o prueba)
- -e: establecer variables de entorno dentro del contenedor en el momento de su creación
- "discovery.type=single-node": es una variable (en este caso indica que la imagen se ejecute en modo de nodo único)

REDES

$ docker network ls
Listar todas las redes disponibles en el sistema

$ docker network inspect red
Inspeccionar una red en formato Json

$ docker network create red
Crear una nueva red personalizada

$ docker login
Loguearse en docker hub

COMANDOS CON EL ARCHIVO DOCKER-COMPOSE

$ pip install docker-compose
Instalar docker-compose

$ docker-compose --version
Comprobar que esta instalado docker

$ docker-compose ps
Mostrar estado de los servicios definidos en el archivo

$ docker-compose up (-d)
Iniciar i construir los servicios definidos en el archivo al directorio actual

$ docker-compose down -v
Detiene y elimina los contenedores y volúmenes definidos en el archivo

$ docker-compose run servicio bash
Ejecutar el servicio y abrir una terminal interactiva dentro de este.

COMANDOS ÚTILES MÁS ALLA DE DOCKER

wsl --update
Actualizar el subsistema de Windows para Linux

$ tree -L 2
Mostrar la estructura de directorios en forma de árbol **(tree)** hasta **(-L)** el nivel especificado de profundidad **(2)**

$ ./setup-docker.sh
Ejecutar un script llamado setup-docker.sh

$ curl 0.0.0.0:pppp
Realizar una solicitud HTTP a la dirección IP 0.0.0.0 en el puerto pppp para obtener información del servicio o aplicación que se está ejecutando en esa dirección y puerto (formato json)

$ curl -I 0.0.0.0:pppp/url
Realizar una solicitud HTTP, obteniendo información sobre la respuesta a la url (/url)

- -I: solicita solo la cabecera de la respuesta HTTP

$ pip show programa
Mostrar información sobre un programa instalado (ej: la ruta dónde se encuentra instalado dicho programa **(Location))**

$ export PATH=$PATH:Location program
Ampliar la variable PATH para incluir un directorio específico y poder ejecutar comandos de ese programa desde la terminal