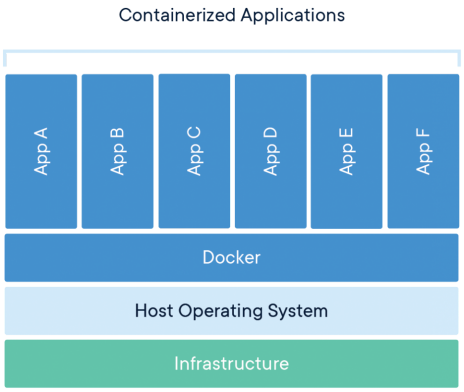
Tutorial 2: Automatizar el despliegue de aplicaciones con Docker y Docker-compose

1. ¿Qué es Docker y el despliegue por contenedores?

- Docker es una herramienta de código abierto que sirve para empaquetar, transportar y ejecutar softwares de distintos niveles de complejidad.

- Docker crea una capa de abstracción con el sistema operativo.



- La idea principal de Docker es crear contenedores portables para que una aplicación se pueda ejecutar en cualquier máquina que tenga Docker instalado (sin importar el sistema operativo que la máquina tenga instalado)
- Docker automatiza el despliegue mejorando mucho la **portabilidad** de nuestras aplicaciones

Requisitos no funcionales - atributos de calidad

Algunas ventajas:

1. Se puede entregar una arquitectura de software en una sola pieza. Docker ofrece un formato de imagen unificado para distribuir sus aplicaciones.
2. Las imagenes Docker son compatibles con los servicios de "Cloud Computing". Se puede despliegar en cualquier servidor.

3. Reduce a casi cero la probabilidad de error causado por diferentes versiones de sistemas operativos, dependencias del sistema, etc.
4. La plataforma nativa de Docker es Linux, ya que se basa en las características proporcionadas por el núcleo de Linux. Sin embargo, se puede ejecutarlo en macOS y Windows.

2. Terminología útil:

- **Imagen** Una descripción estática de la arquitectura de software que se quiere despliegar.
- **Contenedor** Una instancia en ejecución que encapsula la arquitectura de software funcionando.

  Los contenedores siempre se crean a partir de imágenes.

  Un contenedor puede exponer puertos y volúmenes para interactuar con otros contenedores o con el mundo exterior.

  Los contenedores pueden ser fácilmente eliminados / removidos y recreados de nuevo en muy poco tiempo.
- **Port (Puerto)** un puerto TCP/UDP corresponde a una dirección que sirve de punto de integración entre varios softwares. Los puertos pueden estar expuestos al mundo exterior (accesible desde el sistema operativo del host) o conectados a otros contenedores, es decir, accesibles sólo desde esos contenedores e invisibles para el mundo exterior.
- **Volumen** puede ser descrito como una carpeta compartida dónde se almacenan los datos útiles. Los volúmenes se inicializan cuando se crea un contenedor. Los volúmenes están diseñados para datos persistentes, independientemente del ciclo de vida del contenedor.
- **Docker Hub** un servidor con interfaz web proporcionada por Docker Inc. Almacena muchas imágenes Docker con diferentes programas. Docker Hub es una fuente de imágenes "oficiales" de Docker realizadas por el equipo Docker o en cooperación con el fabricante original del software (no significa necesariamente que estas imágenes "originales" procedan de fabricantes oficiales de software).

  Ejemplo: Imagen de MariaDB

  https://hub.docker.com/search?q=mariadb&type=image

2. Instalación de Docker

- Puede consultar las instrucciones de instalación de Docker aquí. Si está ejecutando Docker en Linux, necesita ejecutar todos los siguientes comandos como root o añadir su usuario al grupo de dockers y volver a iniciar sesión:

```
sudo usermod -aG docker username
```

💡 **Nota Bene**: reemplazar username por su nombre de usuario

```
docker -v
```

El último comando permite ver qué versión de docker está instalada.

3. ¿Cómo despliegar un primer contenedor Docker básico?

Ejemplo 1: Hello World

Ejecutemos nuestro primer contenedor Docker:

```
docker run ubuntu /bin/echo 'Hello world'
```

- **docker run** es una orden para levantar un contenedor.
- **ubuntu** es la imagen a partir de la cual se construye el contenedor. Por ejemplo, la imagen del sistema operativo Ubuntu. Al especificar una imagen, Docker busca primero la imagen en el host del Docker. Si la imagen no existe localmente, entonces la imagen se extrae del registro de imagen pública Docker Hub.
- **/bin/echo 'Hello world'** es el comando que se ejecutará dentro de un nuevo contenedor. Este contenedor simplemente imprime "Hello world" y detiene la ejecución.

Ejemplo 2: Contenedor interactivo

Creamos un shell interactivo dentro de un contenedor Docker:

```
docker run -i -t --rm ubuntu
```

- -**t** crea un terminal dentro del contenedor
- -**i** le permite interactuar directamente con el terminal del contenedor
- El parámetro **rm** elimina automáticamente el contenedor cuando el proceso termina.

> 💡 Por defecto, los contenedores no se eliminan. Este contenedor existe hasta que mantenemos la sesión shell y termina cuando salimos de la sesión (como una sesión SSH con un servidor remoto).

Veamos qué contenedores tenemos en este momento:

```
docker ps -a
```

Salida de la consola:

```
CONTAINER ID  IMAGE   COMMAND                  CREATED             STATUS                         PORTS  NAMES
c006f1a02edf  ubuntu  "/bin/echo 'Hello ..."  About a minute ago  Exited (0) About a minute ago         gifted_nobel
```

- **docker ps** es un comando para listar contenedores.
- **a** muestra todos los contenedores (sin -a la bandera ps sólo mostrará los contenedores en ejecución).

Para eliminar todos los contenedores, podemos usar el siguiente comando:

```
docker rm -f $(docker ps -aq)
```

- **docker rm** es el comando para remover el contenedor.
- **f** (para rm) detiene el contenedor si está en marcha (es decir, borrar por la fuerza).
- **q** (para ps) es imprimir sólo los IDs del contenedor.

Ejemplo 3: Variables de entorno y volumenes

- Actualizar el repositorio https://github.com/matthieuvernier/INFO229_2021 y posicionarse en la carpeta "tutorial_2_Docker".

Crearemos contenedores a partir de softwares existentes, por ejemplo NGinx. NGinx es un servidor web de código abierto que permite escuchar consultas HTTP.

- Entrar en la carpeta "nginx":

```
docker run -d --name "test-nginx" -p 8080:80 -v $(pwd):/usr/share/nginx/html:ro nginx:latest
```

💡 **Advertencia**: Este comando parece bastante pesado, pero es sólo un ejemplo para explicar los conceptos de volúmenes y variables de entorno (env). En el 99% de los casos de la vida real, no iniciará los contenedores Docker manualmente, sino que utilizará servicios de orquestación (ilustraremos el uso de Docker-Compose a continuación).

Salida de la consola:

```
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
683abbb4ea60: Pull complete
a470862432e2: Pull complete
977375e58a31: Pull complete
Digest: sha256:a65beb8c90a08b22a9ff6a219c2f363e16c477b6d610da28fe9cba37c2c3a2ac
Status: Downloaded newer image for nginx:latest
afa095a8b81960241ee92ecb9aa689f78d201cff2469895674cec2c2acdcc61c
```

- **p** es un mapeo
- de los puertos HOST PORT:CONTAINER PORT.

- **v** es un mapeo de los volumenes HOST DIRECTORY:CONTAINER DIRECTORY.

💡 **Importante**: el comando sólo acepta rutas absolutas. En nuestro ejemplo, hemos usado $(pwd) para establecer la ruta absoluta del directorio actual.

- Conectarse a 127.0.0.1:8080 en su navegador web.
- Podemos intentar cambiar nginx/index.html (que está montado como un volumen en el directorio /usr/share/nginx/html dentro del contenedor) y actualizar la página.
- Obtenemos la información sobre el contenedor **test-nginx*:

```
docker inspect test-nginx
```

Este comando muestra información de todo el sistema sobre la instalación del Docker. Esta información incluye la versión del núcleo, el número de contenedores e imágenes, los puertos expuestos, los volúmenes montados, etc.

3. ¿Cómo crear sus propias imágenes con el archivo *Dockerfile?*

3.1 ¿Qué es el archivo *Dockerfile*?

- Dockerfile es un archivo de textos que sirve para crear sus propias imágenes, y así configurar el funcionamiento de sus propios softwares.

3.2 Ejemplo: Escribir un Dockerfile

- Para crear una imagen Docker, debe crear un archivo **Dockerfile**. Es un archivo de texto plano con instrucciones y argumentos.

Aquí está la descripción de las instrucciones que vamos a usar en nuestro próximo ejemplo:

> **FROM** - establece la imagen base
> **RUN** - ejecuta ciertos comandos al momento de construir la imágen
> **ENV** - define variables de entorno
> **WORKDIR** - define directorio de trabajo
> **VOLUME** - crea un punto de montaje para un volumen
> **CMD** - ejecuta ciertos comandos al momento de despliegar un contenedor

💡 Puede consultar la referencia de Dockerfile para obtener más detalles.

- Vamos a crear una imagen que consultará el contenido del sitio web con el comando **curl** y lo almacenará en un archivo de texto. Necesitamos pasar la URL del sitio web a través de la variable de entorno SITE_URL. El archivo generado se guardará en un directorio, montado como un volumen.

Coloque el nombre de archivo Dockerfile en el directorio de `curl` con el siguiente contenido:

```
FROM ubuntu:latest
RUN apt-get update
    && apt-get install --no-install-recommends --no-install-suggests -y curl
    && rm -rf /var/lib/apt/lists/*
ENV SITE_URL http://example.com/
WORKDIR /data
VOLUME /data
CMD sh -c "curl -Lk $SITE_URL > /data/results"
```

El archivo Dockerfile está listo. Es hora de construir la imagen real.

En el directorio curl y ejecute el siguiente comando para construir una imagen:

```
docker build . -t test-curl
```

Salida del terminal:

```
Sending build context to Docker daemon  3.584kB
Step 1/6 : FROM ubuntu:latest
 ---> 113a43faa138
Step 2/6 : RUN apt-get update     && apt-get install --no-install-recommends --no-install-suggests -y curl     && rm -rf /var/lib/apt/lists
 ---> Running in ccc047efe3c7
Get:1 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
Get:2 http://security.ubuntu.com/ubuntu bionic-security InRelease [83.2 kB]
...
Removing intermediate container ccc047efe3c7
 ---> 8d10d8dd4e2d
Step 3/6 : ENV SITE_URL http://example.com/
 ---> Running in 7688364ef33f
Removing intermediate container 7688364ef33f
 ---> c71f04bdf39d
Step 4/6 : WORKDIR /data
Removing intermediate container 96b1b6817779
 ---> 1ee38cca19a5
Step 5/6 : VOLUME /data
 ---> Running in ce2c3f68dbbb
Removing intermediate container ce2c3f68dbbb
 ---> f499e78756be
Step 6/6 : CMD sh -c "curl -Lk $SITE_URL > /data/results"
 ---> Running in 834589c1ac03
Removing intermediate container 834589c1ac03
 ---> 4b79e12b5c1d
Successfully built 4b79e12b5c1d
Successfully tagged test-curl:latest
```

- **docker build** construye una nueva imagen localmente.

- **t** establece una etiqueta para nombrar la imagen.

Ahora tenemos la nueva imagen, y podemos verla en la lista de imágenes existentes:

```
docker images
```

Salida del terminal:

```
REPOSITORY  TAG     IMAGE ID      CREATED         SIZE
test-curl   latest  5ebb2a65d771  37 minutes ago  180 MB
nginx       latest  6b914bbcb89e  7 days ago      182 MB
ubuntu      latest  0ef2e08ed3fa  8 days ago      130 MB
```

Podemos crear y ejecutar el contenedor desde la imagen. Vamos a hacerlo con los parámetros por defecto:

```
docker run --rm -v $(pwd)/vol:/data/:rw test-curl
```

Para ver los resultados en el archivo:

`cat ./vol/results`

Probemos definiendo la variable de entorno, tomando un sitio real como ejemplo (por ejemplo Facebook.com):

```
docker run --rm -e SITE_URL=https://facebook.com/ -v $(pwd)/vol:/data/:rw test-curl
```

Ver los resultados:

```
cat ./vol/results
```

Nota bene:

- Para suprimir una imagen específica:

`docker rmi -f [nombre o ID de la imágen]`

- Para suprimir todas las imágenes:

`docker rmi $(docker images -qa)`

4. ¿Cómo combinar varias imágenes para construir software complejos con *Docker-compose?*

- **Docker compose** es un software CLI utilizada para conectar contenedores entre sí.

Puede instalar docker-compose vía pip:

`pip install docker-compose`

4.1 Ejemplo 1: Aplicación Python y Redis (Sistema de gestión base de datos)

En este ejemplo, vamos a crear una aplicación web en Python (utilizando la librería Flask) y el SGBD Redis.

La aplicación Python simplemente va a contar cuántas veces un usuario se conecta a una página web y actualizará un contador en una base de datos Redis.

Etapa 1: crear la aplicación Python

Crear una carpeta llamada *compose* y posicionarse a dentro. En *compose*, crear una carpeta *app*. Esta carpeta almacenará los archivos útiles para nuestra aplicación.

En un archivo app.py, crear la aplicación Flask:

```
import os

from flask import Flask
from redis import Redis


app = Flask(__name__)
redis = Redis(host=os.environ['REDIS_HOST'], port=os.environ['REDIS_PORT'])
bind_port = int(os.environ['BIND_PORT'])


@app.route('/')
def hello():
    redis.incr('hits')
    total_hits = redis.get('hits').decode()
    return f'Hello from Redis! I have been seen {total_hits} times.'


if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True, port=bind_port)
```

Nota bene: Nuestra aplicación tiene dos dependencias. Depende de las librerias Flask y Redis.

- Creamos un archivo txt listando las dependencias de nuestra applicación.

En *app/requirements.txt*:

```
flask==1.0.2
redis==2.10.6
```

Etapa 2: Crear una imágen de nuestra aplicación Python

En la carpeta *app*, crear un archivo Dockerfile.

```
FROM python:3.6.3

ENV BIND_PORT 5000
ENV REDIS_HOST localhost
ENV REDIS_PORT 6379

COPY ./requirements.txt /requirements.txt

RUN pip install -r /requirements.txt

COPY ./app.py /app.py

EXPOSE $BIND_PORT

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

Etapa 3: Describir la arquitectura de software con Docker-compose

En la carpeta *compose* , creamos una archivo de configuración llamado **docker-compose.yml**. Tiene el contenido siguiente:

```
version: '3.6'
services:
  app:
    build:
      context: ./app
    depends_on:
      - redis
    environment:
      - REDIS_HOST=redis
    ports:
      - "3001:5000"
  redis:
    image: redis:3.2-alpine
    volumes:
      - redis_data:/data
```

```
volumes:
  redis_data:
```

En la carpeta `compose` ejecutar el siguiente comando:

```
docker-compose up
```

- El ejemplo de aplicación incrementará la vista del contador en Redis.
- Abrir la dirección 127.0.0.1:3001 en su navegador web y compruébelo.

> 💡 Siempre deben dar nombres explícitos a sus volúmenes en docker-compose.yml (si la imagen tiene volúmenes). Esta sencilla regla les ahorrará un problema en el futuro cuando inspeccionen sus volúmenes.

En este caso, redis_data será el nombre dentro del archivo docker-compose.yml.

Para ver los volúmenes, ejecuten:

```
docker volume ls
```

Salida del terminal:

```
DRIVER              VOLUME NAME
local               apptest_redis_data
```

Sobre Redis...

Redes es un SGBD no relacional ("*No SQL*") que permite almacenar datos en un formato "key --> value".

`docker ps` : para ver el ID del contenedor Redis
`docker exec -it [ID] redis-cli`: para ejecutar el cliente redis-cli en el contenedor de Redis

una vez conectado a Redis:

`keys *`: para listar todas las claves almacenedas
`GET Hits` : para leer el valor de la clave "Hits"
`SET Hits 500` : para escribir un nuevo valor en la clave "Hits"

5. Conclusión del tutorial

- Docker se ha convertido en una herramienta importante de desarrollo de software
- Se puede utilizar en todo tipo de proyectos independientemente de su tamaño y complejidad
- **La encapsulación de su código en contenedores Docker le ayudará a crear procesos de integración y despliegue más rápidos y eficientes**
- No hay persistencia de datos en los contenedores
- Eviten configuraciones manuales dentro de los contenedores. Deberian añadir comandos en el archivo de configuración Dockerfile.
- Pueden utilizar el comando `docker exec` para ejecutar comandos puntualment en un contenedor activo.

6. Preguntas

Algunas preguntas para autoevaluar su comprensión del tutorial:

1. ¿Cuál es la diferencia entre una Imágen Docker y un Contenedor Docker?

Un contenedor docker es una instancia que encapsula la arquitectura de software funcionando, ésta se crea a partir de imagenes docker, que son descripciones estáticas de las arquitecturas que se quieren desplegar

1. ¿Cuál es la diferencia entre el archivo Dockerfile y el archivo Docker-compose.yml?

Dockerfile es un archivo de textos plano que contiene las instrucciones para crear una imagen que define el entorno de una aplicación. Docker-compose.ym sirve para conectar contenedores entre si, para que puedan ejecutarse juntos.

1. ¿Si utilizo un contenedor que contiene una Base de Datos (Redis, Mysql u otro), cómo y dónde se guardan los datos?

Se almacenan en imagenes, mediante volumenes.

1. ¿Qué es un *port*? ¿Por qué algunas imágenes requieren hacer un *bind* entre distintos puertos?

Port, define un puerto para la dirección del navegador.

Enlazar los puertos, permite exponerlos y evitar fallar por puertos ocupados.

7. Ejercicio

A partir de la aplicación Python desarrollada en tutorial 1 ("API web con FastAPI" —> my-api)

1. Dockerizar la aplicación para poder despliegar fácilmente en cualquier máquina con el comando **docker run.** (Necesitarán utilizar el comando **docker-compose up** para ejecutar su aplicación)

No la puedo realizar, ya que no pude concluir de manera correcta el ejercicio en tutorial 1