Tema 0: Planificación de la asignatura

Asignatura: Despliegue de aplicaciones Web

Planificación de la asignatura

- Impartida por Tacio Camba Espí
  - Contacto - tacio@ciclosmontecastelo.com
    - Tutorías - Concertar cita previa por mail
      - En remoto - A conveniencia de alumno y profesor
      - Presencial - En el centro, **jueves de 20:00 a 21:00**
    - Dudas, sugerencias y quejas

- 7 sesiones (5 ordinarias + 1 repaso + 1 examen)
  - Jueves 17:00 a 20:00 - Desde el 01/02/2024 al 21/03/2024
  - Sesión de repaso - Jueves 14/03/2024
  - Examen 1ra convocatoria - Jueves 22/03/2024
  - Examen 2da convocatoria - Jueves 06/06/2024

Evaluación

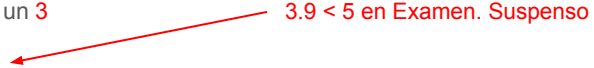
- 2 tareas obligatorias (1ra conv. - 30%)
  - Ejercicio teórico / prácticos sobre uno o varios temas impartidos en el momento de su publicación
  - Publicada como tarea en el *Classroom* de la asignatura
  - Entrega de la tarea
    - Fecha y hora límite - Ver planificación y enunciado (prioridad del enunciado)
    - Medio de entrega - Tarea de *Classroom* correspondiente
      - Fichero **apellidos-nombre.zip** (no se corregirá de otro modo)
    - Retrasos en la fecha de entrega
      - Menos de 24 h → Se resta 1 punto
      - Entre 24 y 48 h → Se restan 2 puntos
      - Entre 48 y 72 h → Se restan 3 puntos
      - Más de 72 h → Suspenso con un 0
  - Publicación de calificaciones y soluciones en *Classroom*
    - Fecha de entrega oficial + 10 días

Evaluación

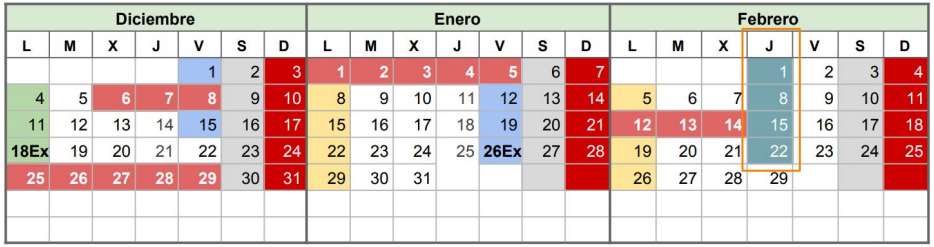
- Examen (1ra conv. - 70% / 2da conv. - 100%)
  - Prueba teórico-práctica sobre los contenidos de la asignatura al completo
  - 1ra convocatoria - Jueves 21/03/2024
  - 2da convocatoria - Jueves 06/06/2024
- Las calificación final de la asignatura se redondeará al entero más próximo
  - 8.4 ~ 8
  - 5.5 ~ 6
- La calificación del examen en 2da convocatoria será la media de la nota del examen oficial y el de recuperación. En caso de que la calificación sea superior a 4.5 pero inferior a 5, se calificará con un 5 (siempre y cuando el examen de recuperación esté aprobado con un 5 o más)
- La calificación media de las dos partes (tareas / examen) debe ser superior a 5 para aprobar la asignatura
- Plagio (examen y tareas)
  - Calificación de 0 para el que copia
  - Calificación de 0 para el que es copiado
- Revisión de calificaciones
  - Se solicitará por mail al profesor en los **7 días siguientes** a la comunicación de la nota
  - La solicitud de revisión implica recalificación (a favor o en contra del alumno) en caso de errata

Evaluación

- Ejemplos de evaluación
  - Roberto
    - Tareas
      - **T1** superada con un **5**
      - **T2** superada con un **8**
      - **T3** no entregada
    - **Examen 1ra** convocatoria → **7**
    - **Nota final** = 0.3 * (5 + 8 + 0) / 3 + 0.7* 7 = **6.2**
  - Alicia
    - Tareas
      - **T1** aprueba con **8**
      - **T2** aprueba con **8**
      - **T3** superada con **6.8**
    - **Examen 1ra** convocatoria suspendido con un 3
      - **Examen 2da** convocatoria → **7**
    - **Nota final** = (3 + 7) / 2 = **5**
  - Laura
    - Tareas
      - **T1** aprueba con **8**
      - **T2** aprueba con **8**
      - **T3** superada con **6.8**
    - **Examen 1ra** convocatoria suspendido con un 3
      - **Examen 2da** convocatoria → **4.9**
    - **Nota final** = (3 + 4.9) / 2 = 3,95



Planificación de sesiones, tareas y exámenes



Diciembre						
L	M	X	J	V	S	D
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18Ex	19	20	21	22	23	24
25	26	27	28	29	30	31

Enero						
L	M	X	J	V	S	D
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26Ex	27	28
29	30	31				

Febrero						
L	M	X	J	V	S	D
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29			

Marzo						
L	M	X	J	V	S	D
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21Ex	22	23	24
25	26	27	28	29	30	31

Abril						
L	M	X	J	V	S	D
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29Ex	30					

Mayo						
L	M	X	J	V	S	D
		1	2	3Ex	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Junio						
L	M	X	J	V	S	D
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27			

Tutorías			Exámenes	
Asignatura.	Prof	Día/Hora	1ª Conv.	2ª Conv
Diseño Interfaces	Tacio	Martes 18:00	26/01	31/05
Desarrollo Servidor	Martín	Miérc 12:00	15/1	03/06
Desarrollo Cliente	D. Sanchez	Jue 20:00	29/04	10/06
Despliegue Aplicacione	Tacio	Jueves 20:00	21/03	06/06
EIE	A Luna	Vie 15:00	03/05	14/05

Planificación de sesiones, tareas y exámenes

- Tarea 1:
  - Fecha de publicación: POR DETERMINAR
  - Fecha de entrega: POR DETERMINAR
  - Fecha final de prórroga: POR DETERMINAR
  - Contenido: Docker y Docker compose
  - Ponderación sobre la media: 50 %
- Tarea 2:
  - Fecha de publicación: POR DETERMINAR
  - Fecha de entrega: POR DETERMINAR
  - Fecha final de prórroga: POR DETERMINAR
  - Contenido: POR DETERMINAR
  - Ponderación sobre la media: 50 %

Materiales y recursos

- Apuntes, enunciados y videos
  - *Classroom* de la asignatura
    - Acceso por invitación enviada al correo del ciclo
    - Invitación no recibida → Mail al profesor
- Código
  - Repositorio Github de la asignatura
    - https://github.com/tcamba-ciclos-montecastelo/daw-22
- Todas las sesiones serán grabadas
  - *Classroom*

FORMACIÓN PROFESIONAL MONTECASTELO

Temario

Tema 1: Docker

Asignatura: Despliegue de aplicaciones Web

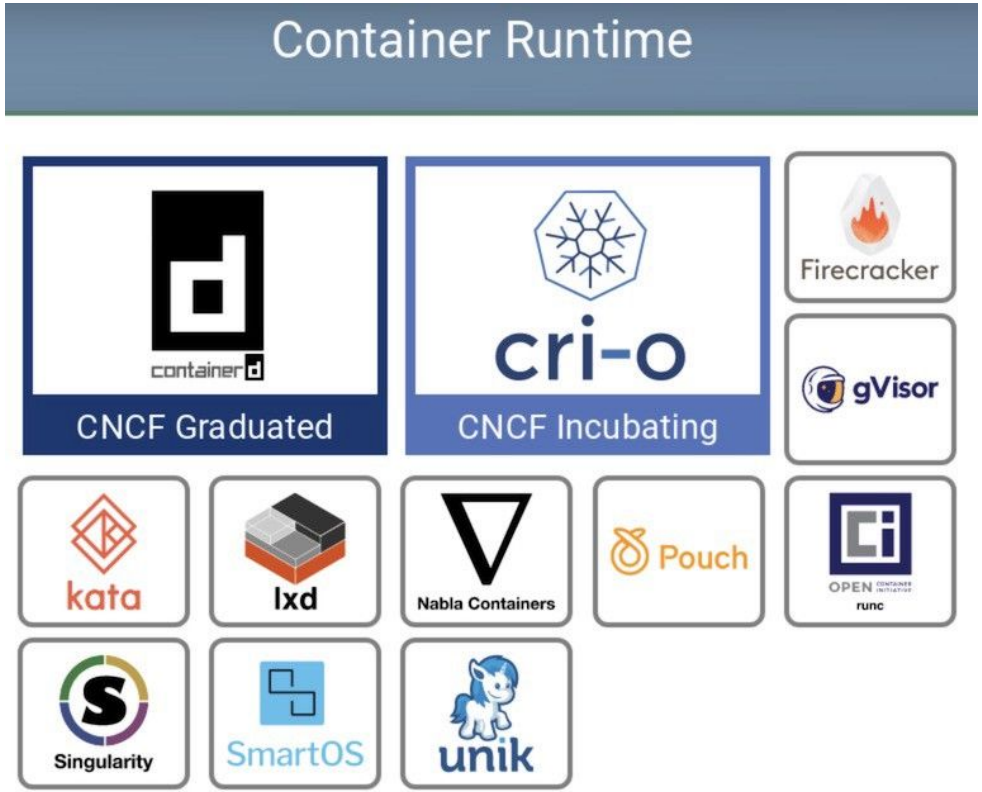
FORMACIÓN PROFESIONAL MONTECASTELO

Contenidos

Motivación

- Los **contenedores** son una forma de **aislar aplicaciones y servicios** en un sistema operativo compartido (Host)
- **Docker** es una plataforma de contenedores muy popular que permite empaquetar y distribuir aplicaciones de forma sencilla.
- Los **contenedores** se construyen a partir de **imágenes**, que pueden ser descargadas de un **registro de imágenes** o creadas manualmente.

Motivación


Container Runtime
containerd
CNCF Graduated
cri-o
CNCF Incubating
Firecracker
gVisor
kata
lxd
Nabla Containers
Pouch
OPEN CONTAINER INITIATIVE
runc
Singularity
SmartOS
unik

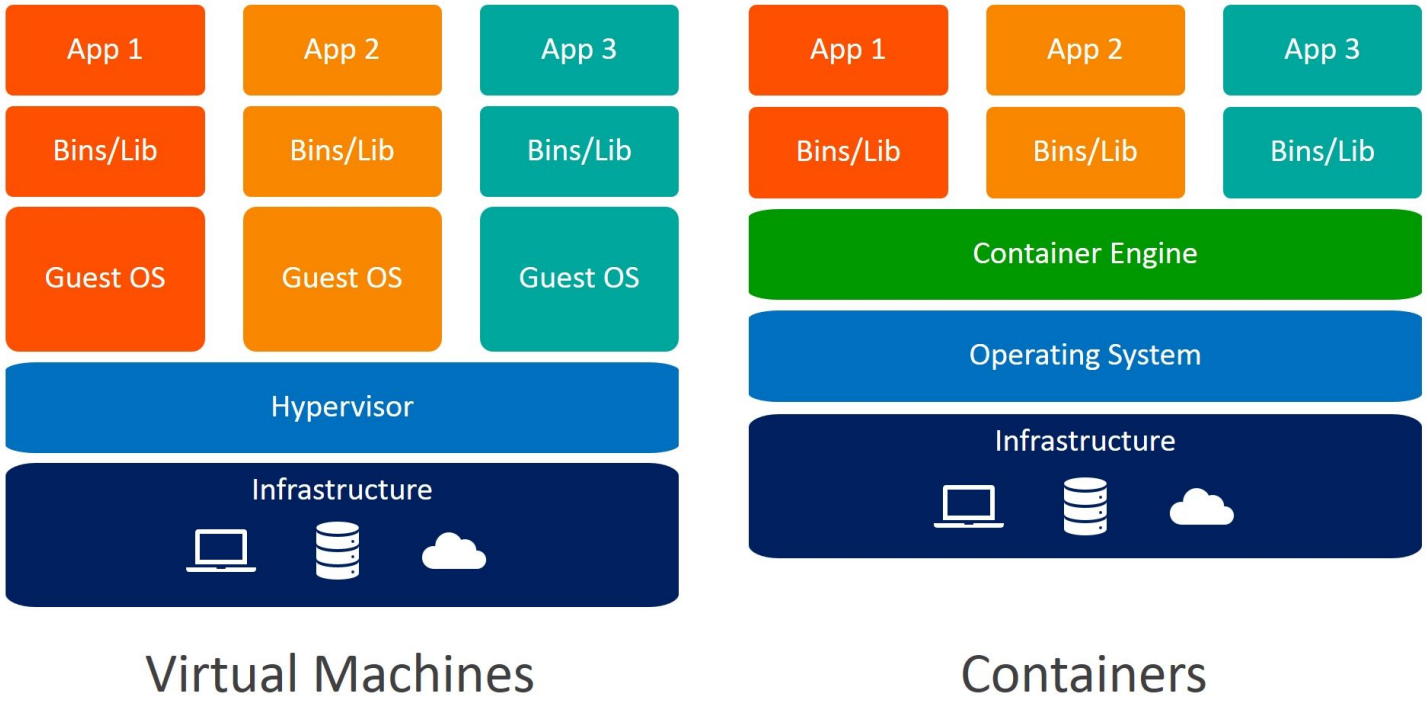
Motivación

- Containers vs VMs
  - Las máquinas virtuales (**VMs**) crean un **sistema operativo completo** y aislado dentro de otro sistema operativo anfitrión, mientras que los **contenedores comparten** el **kernel del sistema operativo anfitrión**.
  - Las VMs requieren una gran cantidad de recursos, como memoria y espacio en disco, mientras que los **contenedores** son más **ligeros** y requieren menos recursos.
  - Las **VMs** tienen un tiempo de **arranque** más **lento** que los contenedores debido a que deben cargar un sistema operativo completo, mientras que los **contenedores** se **inician rápidamente** al utilizar el kernel del sistema operativo anfitrión.

Motivación

- Containers vs VMs
  - Las VMs proporcionan un mayor aislamiento y seguridad, ya que cada VM tiene su propio sistema operativo y configuración, mientras que los contenedores comparten el kernel del sistema operativo anfitrión y pueden ser más vulnerables a ataques.
  - VMs son adecuadas para cargas de trabajo que requieren un gran aislamiento y recursos, mientras que los contenedores son adecuados para cargas de trabajo que requieren menos aislamiento y recursos.

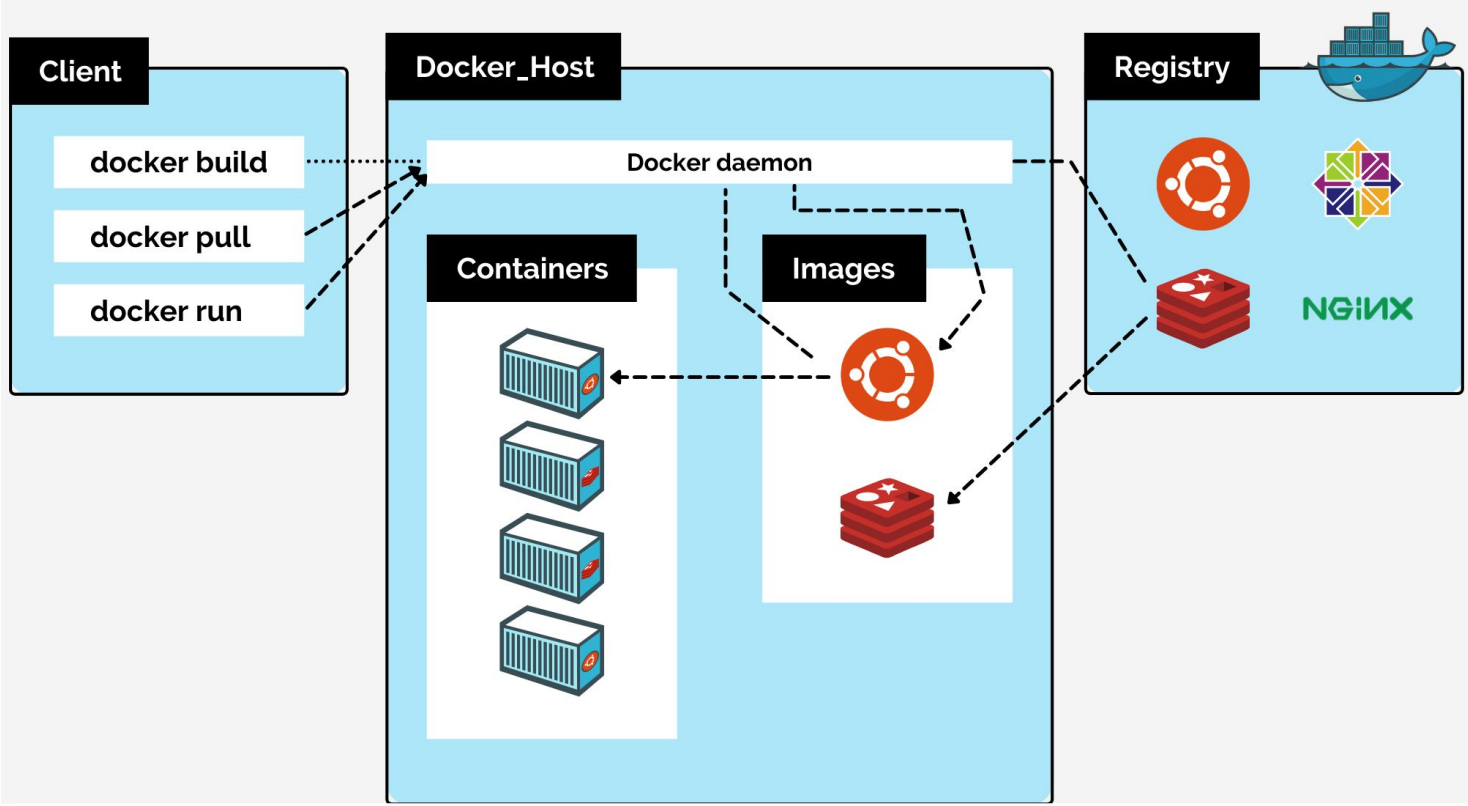
Motivación


App 1
App 2
App 3
Bins/Lib
Bins/Lib
Bins/Lib
Guest OS
Guest OS
Guest OS
Hypervisor
Infrastructure
Virtual Machines
App 1
App 2
App 3
Bins/Lib
Bins/Lib
Bins/Lib
Container Engine
Operating System
Infrastructure
Containers

Docker Engine y Docker CLI

- **Docker Engine** es el componente principal de la plataforma Docker que se encarga de **crear, ejecutar y administrar contenedores**.
- Permite **construir imágenes** a partir de un **archivo** de configuración llamado **Dockerfile**.
- Permite crear, iniciar, detener, eliminar y administrar contenedores.
- Proporciona una interfaz de línea de comandos (**CLI**) y una **API** de programación para **interactuar con los contenedores**.
- Se basa en el sistema de control de versiones de contenedores.
- Permite conectarse a un **registro de imágenes** para descargar e implementar contenedores.

Docker Engine y Docker CLI



Docker Engine y Docker CLI

- **Elementos o entidades de Docker**
  - Imágenes: son plantillas que contienen todo lo necesario para ejecutar una aplicación o servicio. Se construyen a partir de un archivo de configuración llamado Dockerfile.
  - Contenedores: son instancias de una imagen que se ejecutan como procesos independientes en el sistema operativo anfitrión. Contienen la aplicación o servicio y su configuración.
  - Redes: son un conjunto de interfaces virtuales que permiten comunicar los contenedores entre sí y con el sistema anfitrión.
  - Volúmenes: son un mecanismo de almacenamiento que permite a los contenedores acceder a datos persistentes, independientes del ciclo de vida del contenedor. Los volúmenes son útiles para mantener los datos a través de las actualizaciones o eliminaciones de un contenedor.

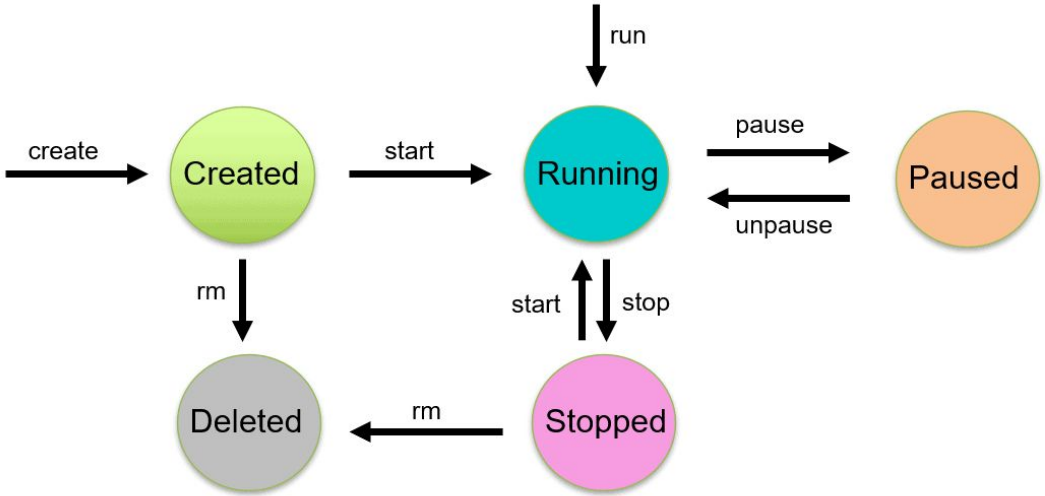
Docker Engine y Docker CLI

- **Ciclo de vida en Docker**
  - Construido: La imagen del contenedor se ha construido a partir de un archivo de configuración llamado Dockerfile utilizando el comando docker build
  - Creado: El contenedor se ha creado a partir de una imagen utilizando el comando docker create
  - Iniciado: El contenedor se ha iniciado utilizando el comando docker start, la aplicación o servicio dentro del contenedor se está ejecutando
  - Detenido: El contenedor se ha detenido utilizando el comando docker stop, la aplicación o servicio dentro del contenedor deja de ejecutarse
  - Eliminado: El contenedor se ha eliminado utilizando el comando docker rm, el contenedor y su configuración se han eliminado del sistema

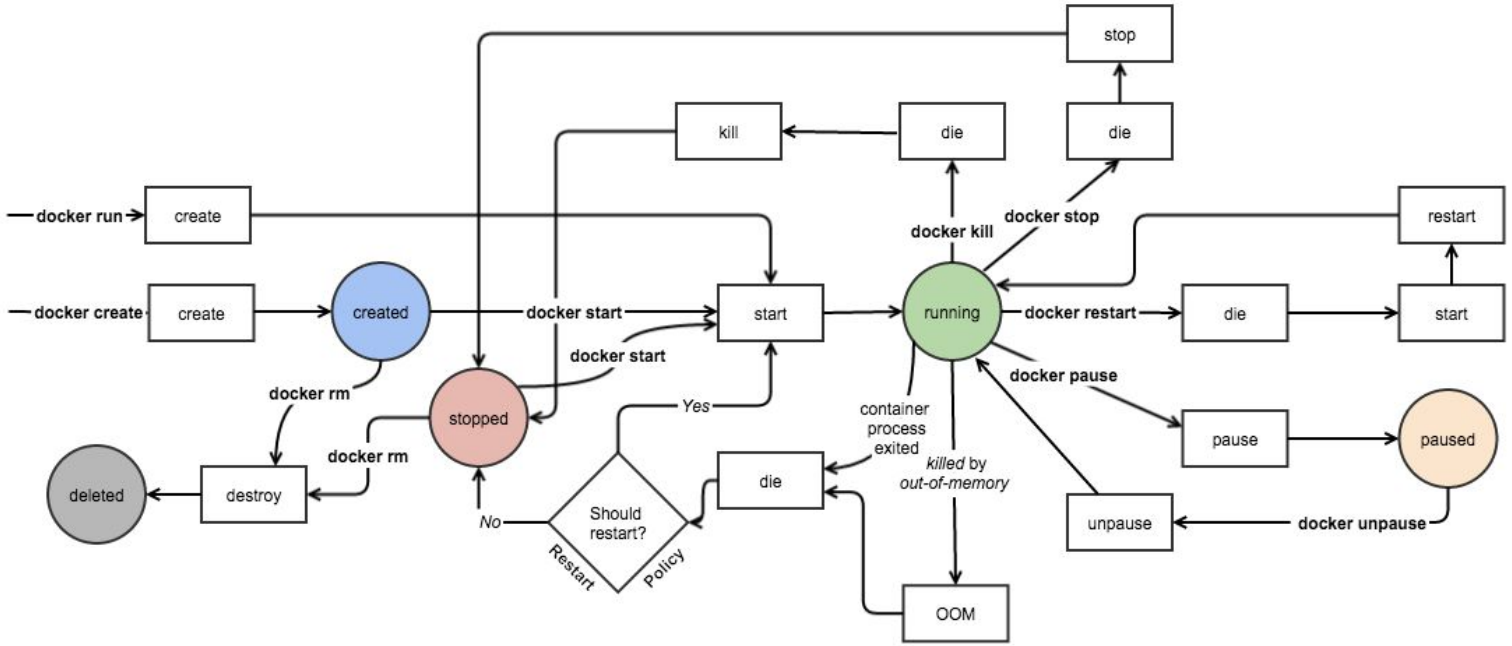
Es importante tener en cuenta que un contenedor sólo puede encontrarse en un estado a la vez, es decir, no puede estar ejecutándose y detenido al mismo tiempo

Docker Engine y Docker CLI

- **Ciclo de vida en Docker**



Docker Engine y Docker CLI



Docker Engine y Docker CLI

- Instalación de Docker (A0 - 10m)
  - Windows (Docker Desktop)
    - https://docs.docker.com/desktop/install/windows-install/
    - Backends:
      - **WSL2** (Windows Subsystem for Linux) - https://learn.microsoft.com/es-es/windows/wsl/install
      - Hyper-V
  - Linux
    - https://docs.docker.com/desktop/install/linux-install/

Docker Engine y Docker CLI

- **Docker CLI**
  - Docker **CLI** es la **interfaz de línea de comandos** de Docker que permite **interactuar** con el **Docker Engine**
  - Con el Docker CLI se pueden realizar diferentes tareas como **crear, ejecutar, detener y eliminar**… entidades de Docker

```
docker [OPCIONES] COMANDOS [ARG...]
docker [ --help | -v | --version ]

Opciones:
        --help              Muestra la ayuda por
pantalla
  -v,   --version           Muestra la información de
la versión
```

Docker Engine y Docker CLI

- **Docker CLI**

```
Comandos:

    attach    Conectarse a un contenedor en ejecución
    run       Crear y ejecutar un nuevo contenedor a partir de una imagen
    ps        Ver los contenedores en ejecución
    stop      Detener un contenedor en ejecución
    rm        Eliminar un contenedor
    images    Ver las imágenes disponibles en el sistema
    pull        Descargar una imagen desde un registro de imágenes
    push      Subir una imagen a un registro de imágenes
    build     Construir una imagen a partir de un archivo de configuración (Dockerfile)
    exec      Ejecutar un comando dentro de un contenedor en ejecución
    network   Crear, conectar y administrar redes de contenedores
    volume    Crear y administrar volúmenes
    inspect   Ver información detallada sobre un contenedor, imagen o red
    system    Ver información detallada del sistema, limpieza, actualizaciones, etc
```

docker run -d --name whoami -p 8080:80

Imágenes y registros

- ¿Qué son las imágenes de Docker?
  - Las imágenes de Docker son plantillas que contienen todo lo necesario para ejecutar una aplicación o servicio.
  - Se construyen a partir de un archivo de configuración llamado Dockerfile.
- ¿Cómo se construyen las imágenes de Docker?
  - Se utiliza el comando docker build para construir una imagen a partir de un archivo de configuración (Dockerfile)
  - El archivo Dockerfile especifica los comandos necesarios para construir la imagen, como instalar paquetes, copiar archivos, etc

Imágenes y registros

- ¿Cómo se utilizan las imágenes de Docker?
  - Se utiliza el comando docker create para crear un nuevo contenedor a partir de una imagen
  - El contenedor contiene la aplicación o servicio y su configuración
- ¿Dónde se almacenan las imágenes de Docker?
  - Las imágenes de Docker se pueden almacenar localmente en el sistema o en un registro de imágenes remoto
  - El registro de imágenes por defecto es https://hub.docker.com/
  - El comando docker images permite ver las imágenes disponibles en el sistema.
  - El comando docker pull para descargar una imagen desde un registro de imágenes remoto
  - Con el comando docker push subimos una imagen a un registro de imágenes remoto

Imágenes y registros

- ¿Cómo se comparten las imágenes de Docker?
  - Se pueden compartir las imágenes de Docker subiéndolas a un registro de imágenes remoto.
  - Los usuarios pueden entonces descargar y utilizar las imágenes compartidas.
- ¿Qué es una imagen oficial de Docker?
  - Una imagen oficial de Docker es una imagen mantenida por el equipo de Docker o por una comunidad de desarrolladores de confianza.
  - Las imágenes oficiales de Docker están disponibles en el registro de imágenes oficial de Docker.

Imágenes y registros

- ¿Qué es una imagen no oficial de Docker?
  - Una imagen no oficial de Docker es una imagen construida y mantenida por un usuario o una comunidad que no está afiliada o respaldada oficialmente por Docker
  - Estas imágenes pueden ser encontradas en registros de imágenes no oficiales o en repositorios de código abierto.
  - Aunque no están respaldadas oficialmente, estas imágenes pueden ser útiles y confiables, pero es importante evaluar cuidadosamente la confiabilidad y seguridad antes de utilizarlas.

Imágenes y registros

- Usando imágenes oficiales (A1 - 15m)
  - Buscar una imagen de WhoAmI utilizando el CLI
  - Inspeccionar la documentación de la imagen en DockerHub
  - Descargar la imagen desde el registro
  - Inspeccionarla con el CLI

Imágenes y registros

- Archivo Dockerfile
  - Un **Dockerfile** es un **archivo de configuración** que contiene una serie de **instrucciones** para construir una imagen de Docker
  - Estas **instrucciones** especifican los comandos necesarios para **configurar la imagen**, como instalar paquetes, copiar archivos y definir variables de entorno
  - Cada **instrucción** del **Dockerfile** crea una **capa** en la imagen final.
  - Una vez construida la imagen, se puede utilizar para crear contenedores que ejecutan la aplicación o servicio especificado en el Dockerfile

Imágenes y registros

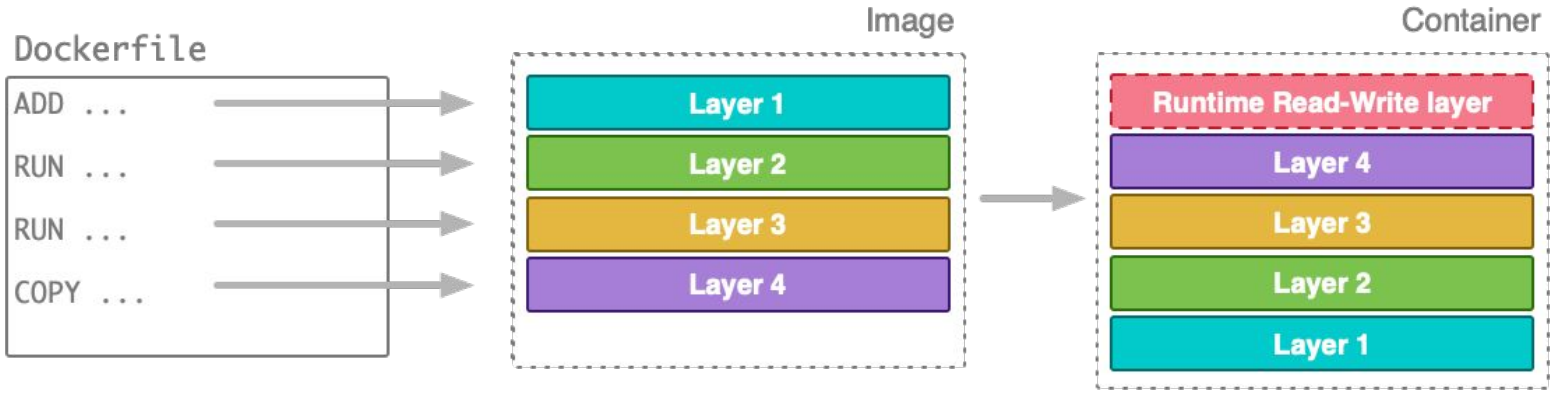
```
FROM python:3.9

# create the app user
RUN addgroup --system app && adduser --system --group app

WORKDIR /app/

# https://docs.python.org/3/using/cmdline.html#envvar-PYTHONDONTWRITEBYTECODE
# Prevents Python from writing .pyc files to disk
ENV PYTHONDONTWRITEBYTECODE 1

# ensures that the python output is sent straight to terminal (e.g. your container log)
# without being first buffered and that you can see the output of your application (e.g. django logs)
# in real time. Equivalent to python -u: https://docs.python.org/3/using/cmdline.html#cmdoption-u
ENV PYTHONUNBUFFERED 1
ENV ENVIRONMENT prod
ENV TESTING 0

# Install Poetry
RUN curl -sSL https://raw.githubusercontent.com/python-poetry/poetry/master/get-poetry.py | POETRY_HOME=/opt/poetry python && \
    cd /usr/local/bin && \
    ln -s /opt/poetry/bin/poetry && \
    poetry config virtualenvs.create false

# Copy poetry.lock* in case it doesn't exist in the repo
COPY ./app/pyproject.toml ./app/poetry.lock* /app/

# Allow installing dev dependencies to run tests
ARG INSTALL_DEV=false

RUN bash -c "if [ $INSTALL_DEV == 'true' ] ; then poetry install --no-root ; else poetry install --no-root --no-dev ; fi"

COPY ./app /app
RUN chmod +x run.sh

ENV PYTHONPATH=/app

# chown all the files to the app user
RUN chown -R app:app $HOME

# change to the app user
# Switch to a non-root user, which is recommended by Heroku.
USER app

# Run the run script, it will check for an /app/prestart.sh script (e.g. for migrations)
# And then will start Uvicorn
CMD ["./run.sh"]
```

Imágenes y registros

- Archivo Dockerfile - Concepto de capas
  - Cada **instrucción** en un Dockerfile crea **una capa** en la imagen final.
  - Estas **capas** son archivos independientes que **contienen los cambios** realizados en la imagen en una instrucción específica
  - Al utilizar varias capas, se permite a los desarrolladores crear imágenes más eficientes y ligeras, ya que **solo se almacenan los cambios realizados en cada capa** y no toda la imagen completa

Imágenes y registros

- Archivo Dockerfile - Concepto de capas



Imágenes y registros

- Archivo Dockerfile - Comandos
  - FROM: Especifica la imagen base a partir de la cual se construirá la nueva imagen. Por ejemplo, FROM ubuntu:18.04 utiliza la imagen de Ubuntu 18.04 como base.
  - RUN: Comando o una serie de comandos que se deben ejecutar para configurar la imagen. Por ejemplo, RUN apt-get update && apt-get install -y apache2 instalaría Apache en la imagen.
  - COPY: Declara un archivo o un directorio que se debe copiar a la imagen. Por ejemplo, COPY index.html /var/www/html copia el archivo index.html a la carpeta del servidor web en la imagen.
  - ENV: Especifica una variable de entorno que se debe establecer en la imagen. Por ejemplo, ENV APACHE_RUN_USER www-data establece la variable de entorno APACHE_RUN_USER en la imagen.
  - EXPOSE: Indica los puertos que se deben exponer desde la imagen. Por ejemplo, EXPOSE 80 expone el puerto 80 en la imagen. Meramente informativo.
  - CMD: Establece el comando que se debe ejecutar cuando se inicia un contenedor a partir de la imagen. Por ejemplo, CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"] ejecuta Apache en el foreground cuando se inicia el contenedor.
  - ENTRYPOINT: Define el comando que se debe ejecutar cuando se inicia un contenedor a partir de la imagen. Se escribe como CMD.
  - LABEL: Etiquetas que se deben agregar a la imagen. Por ejemplo, LABEL maintainer="John Doe <johndoe@example.com>" agrega una etiqueta de mantenedor a la imagen.
  - USER: Usuario con el que se ejecutarán los comandos en la imagen. Por ejemplo, USER root establece el usuario root como el usuario con el que se ejecutan los comandos en la imagen.

Imágenes y registros

- Construyendo un servidor web sencillo (A2 - 20m)
  - Inspecciona la documentación disponible en DockerHub de la imagen oficial de nginx
  - En una carpeta cualquiera de tu equipo prepara los estáticos de una página web de tu elección (html, css y js)
  - Escribe un archivo Dockerfile basado en la imagen de nginx que copie la carpeta con los estáticos de tu equipo de forma que el index.html quede en la ruta indicada en la documentación
  - Construye la imagen

Contenedores

- ¿Qué son los contenedores de Docker?
  - Los contenedores de Docker son una forma de aislar aplicaciones o servicios de un sistema operativo compartido
  - Utilizan un sistema de archivos aislado y recursos limitados para asegurar que la aplicación o servicio no afecte al sistema operativo host
- ¿Cómo se crean los contenedores de Docker?
  - docker create para crear un nuevo contenedor a partir de una imagen
  - Los contenedores se crean con un sistema de archivos aislado y recursos limitados
- ¿Cómo se utilizan los contenedores de Docker?
  - Se utiliza el comando docker start para iniciar un contenedor
  - docker stop para detener un contenedor
  - docker exec para ejecutar un comando dentro de un contenedor

Contenedores

- ¿Cómo se gestionan los contenedores de Docker?
  - docker ps para ver los contenedores en ejecución.
  - docker logs para ver los registros
  - docker inspect para ver información detallada sobre un contenedor
  - docker rm para eliminar un contenedor

Contenedores

- ¿Qué es un contenedor en ejecución?
  - Un contenedor en ejecución es un contenedor que ha sido iniciado y se está ejecutando
  - Puede ser accedido y controlado a través de comandos
- ¿Qué es un contenedor detenido?
  - Un contenedor detenido es un contenedor que ha sido detenido pero aún existe en el sistema
  - Puede ser iniciado nuevamente y **su estado y configuración serán mantenidos**
- ¿Qué es un contenedor eliminado?
  - Un contenedor eliminado es un contenedor que ha sido eliminado y **ya no existe en el sistema**
  - Toda **su información y configuración se pierde**

Contenedores

- Construyendo un contenedor nginx (A3 - 10m)
  - Crea un contenedor nginx a partir de la imagen personalizada de la actividad A2 usando los comandos build y create del CLI
  - Inspecciona con el CLI la información del contenedor y verifica que está ejecutándose
  - Inspecciona los logs del contenedor
  - Detén el contenedor y eliminalo
  - Constrúyelo de nuevo usando el comando docker run
  - ¿Cómo puedes acceder al sitio web servido por el nginx que acabas de desplegar?

Redes

- ¿Qué son las redes de Docker?
  - Las redes de Docker son una forma de **conectar contenedores entre sí**.
  - Permiten la comunicación entre contenedores y el acceso a servicios en la red.
- ¿Cómo se crean las redes de Docker?
  - Se utiliza el comando docker network create para crear una nueva red.
  - Se pueden especificar opciones como el tipo de red y las subredes.
- ¿Cómo se conectan los contenedores a las redes de Docker?
  - Se utiliza el comando docker network connect para conectar un contenedor a una red existente.
  - También se puede especificar la conexión a una red al crear un contenedor con el comando docker run (argumento –network)

Redes

- ¿Qué son las redes por defecto de Docker?
  - Docker tiene tres redes por defecto: **bridge, host y none**
  - **Bridge** es la red por defecto cuando se crea un contenedor
  - **Host** utiliza la red del host
  - **None** no tiene conectividad de red
- ¿Cómo se configuran las redes de Docker?
  - docker network inspect para ver la configuración de una red
- ¿Cómo se eliminan las redes de Docker?
  - Se utiliza el comando docker network rm para eliminar una red
  - Es necesario desconectar todos los contenedores de la red antes de eliminarla

Redes

- Mapeo de puertos
  - El mapeo de puertos en Docker permite exponer puertos específicos en un contenedor para que puedan ser accedidos desde el host.
  - Esto es especialmente útil para exponer servicios y aplicaciones en un contenedor.
  - Para mapear puertos en Docker, se utiliza la opción -p o --publish en el comando docker run

```
docker run -p 80:80 nginx
```

  - o varios puertos a la vez:

```
docker run -p 80:80 -p 443:443 nginx
```

Redes

- Mapeo de puertos
  - Es posible especificar una dirección IP específica para el host utilizando la opción -p IP:host_port:container_port en lugar de solo -p host_port:container_port
  - Por ejemplo, para exponer el puerto 80 en un contenedor con una imagen de nginx en la dirección IP 192.168.1.100 se utilizaría el siguiente comando:

```
docker run -p 192.168.1.100:80:80 nginx
```

Redes

- Mapeo de puertos
  - Es posible declarar una dirección IP específica para el contenedor utilizando la opción --ip o --network-alias al crear el contenedor
  - Es importante tener en cuenta que al utilizar una dirección IP específica para el host o el contenedor, se debe asegurar que esta dirección IP esté disponible y configurada correctamente en el sistema

Redes

- Haciendo accesible nuestro contenedor nginx (A4 - 5m)
  - Haz que el servidor nginx de la actividad anterior sea accesible a través del puerto 8088 de tu máquina

Volúmenes

- Los volúmenes en Docker son una forma de almacenar datos fuera del ámbito de un contenedor individual.
- Permiten compartir datos entre contenedores o preservarlos después de que el contenedor se detenga o se elimine.
- Existen dos tipos de volúmenes en Docker:
  - **Volúmenes administrados:** creados y administrados automáticamente por Docker.
    - Recomendado para almacenar datos que cambian con frecuencia o específicos de un contenedor.
  - **Volúmenes de host:** volúmenes existentes en el sistema de archivos del host que se montan en un contenedor.
    - Recomendado para almacenar datos que deben ser compartidos entre varios contenedores o preservados después de que el contenedor se detenga o se elimine.

Volúmenes

- Creación de un volúmen administrado

```
docker volume create my_volume
```

- Creación de un contenedor alpine que mapea el directorio **/app** del contenedor al volumen administrado **my_volume**

```
docker run -it -v my_volume:/app alpine sh
```

Volúmenes

- Creación de un contenedor ubuntu que mapea el directorio **/path/on/host** del host al directorio **/path/in/container** del contenedor

```
docker run -it -v /path/on/host:/path/in/container alpine sh
```

- Todo lo que se guarde o modifique en ese directorio en el contenedor se reflejará en el host y viceversa.

Volúmenes

- Persistencia en nuestro contenedor nginx (A5 - 10m)
  - Accede al contenedor nginx de la actividad anterior usando el comando docker exec
  - Busca el fichero index.html en el contenedor y modifica su contenido. Sal del contexto del contenedor
  - ¿Puedes visualizar los cambios realizados en el punto anterior?
  - En caso negativo, ¿cómo resolverías el problema?

Referencias

- Documentación oficial de Docker
  - https://docs.docker.com/
- Cheatsheet de Docker
  - https://dockerlabs.collabnix.com/docker/cheatsheet/
- Docker Networking explicado
  - https://techblog.geekyants.com/understanding-docker-networking-1
- Breve historia de la tecnología de contenedores
  - https://www.section.io/engineering-education/history-of-container-technology/

Tema 2: Docker compose

Asignatura: Despliegue de aplicaciones Web

Contenidos

Motivación

- **Docker Compose** es una herramienta que se encarga de automatizar la creación, configuración y ejecución de aplicaciones multi-contenedor
- Esto significa que, en lugar de tener que escribir comandos complejos y largos para crear y configurar contenedores, volúmenes y redes, se pueden especificar todas las configuraciones en un **único archivo de configuración**
- Al tener todas las configuraciones en un solo archivo, es más fácil de entender y mantener, ya que se puede ver toda la configuración de un vistazo.
- **Compose** también facilita el despliegue y escalabilidad de aplicaciones, ya que permite crear varios contenedores de una sola vez y conectarlos entre sí automáticamente.

Fichero de especificación de Compose:

- El fichero de especificación de **Compose** se llama "**docker-compose.yml**" y define los contenedores, volúmenes, redes y configuraciones necesarias para ejecutar una aplicación
- El archivo está escrito en formato YAML, lo que facilita su lectura y escritura
- Es posible tener varios archivos de configuración para diferentes entornos, como desarrollo, pruebas y producción

Fichero de especificación de Compose:

- El archivo de configuración, se divide en tres secciones principales:
  - **Services**: Se especifican los contenedores, su configuración y las dependencias entre ellos
  - **Volumes**: Sección donde se especifican los volúmenes que se usan en la aplicación
  - **Networks:** Se especifican las redes que se usan en la aplicación. En esta sección se pueden especificar configuraciones como el tipo de red (p. ej. bridge, host, none), así como otras opciones avanzadas, como la asignación de IPs fijas o la creación de redes internas.

Fichero de especificación de Compose:

En este ejemplo se especifican dos servicios:

- "web": usa la imagen **"nginx"** y expone el puerto 80 del contenedor en el puerto 80 del **host**. También monta un volumen con un archivo de configuración de nginx y depende del servicio "db" para funcionar.
- "db": usa la imagen **"postgres"** y monta un volumen llamado "pgdata". También se especifican variables de entorno para configurar el usuario y la contraseña de la base de datos.
- Los volúmenes se especifican en la sección **"volumes"** y la red se especifica en la sección **"networks"**.

```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "80:80"
    volumes:
      - "./nginx.conf:/etc/nginx/nginx.conf"
    depends_on:
      - db
  db:
    image: postgres
    volumes:
      - "pgdata:/var/lib/postgresql/data"
    environment:
      - POSTGRES_USER=myuser
      - POSTGRES_PASSWORD=mypassword
volumes:
  pgdata:
    driver: local
networks:
  default:
    driver: bridge
```

Docker CLI - compose

Docker compose cuenta con un CLI muy similar a Docker. Los comandos principales son:

- docker-compose up: Arranca los servicios especificados en el archivo de configuración
- docker-compose down: Detiene y elimina los servicios y redes
- docker-compose ps: Muestra el estado de los servicios
- docker-compose logs: Muestra los logs de los servicios
- docker-compose exec: Ejecuta un comando dentro de un contenedor específico
- docker-compose build: Construye las imágenes especificadas en el archivo de configuración

Docker CLI - compose

- Construye y arranca un contenedor con una BBDD postgresql usando docker compose (A1 - 5m)
  - Red custom: app
  - Puerto del host: 33066
- Verifica que puedes conectarte con un cliente cualquiera
- Crea una BD y una tabla cualquier
- Ejecuta el comando docker compose down
- Vuelve a levantar el contenedor ¿Se conserva la BD y la tabla que has creado?

Docker CLI - compose

- Añadiendo un cliente web a nuestro stack (A2 - 10m)
  - Añade al fichero de especificación de la actividad anterior el cliente web de BD pgadmin
  - Accesible desde el puerto 80 del host
  - No olvides hacer persistente el almacenamiento de la BD postgresql usando un volumen nombrado

Referencias

- Cheatsheet de docker-compose
  - https://dockerlabs.collabnix.com/docker/cheatsheet/
- Docker compose *best practices*
  - https://release.com/blog/6-docker-compose-best-practices-for-dev-and-prod
- Tutorial completo Docker (incluye Swarm)
  - https://www.simplilearn.com/tutorials/docker-tutorial/docker-interview-questions
- Canal de youtube para aprender DevOps
  - https://www.youtube.com/@TechWorldwithNana

Tema 3: Servidores/Servicios de automatización - GitHub Actions

Asignatura: Despliegue de aplicaciones Web

Contenidos

Motivación

La integración y entrega continua es un enfoque clave para el desarrollo de software que se ha vuelto cada vez más popular en los últimos años. Se trata de un proceso automatizado que integra y entrega el código a producción de manera continua y constante, en lugar de realizar entregas esporádicas.

Esta forma de trabajar tiene varios beneficios importantes, entre ellos:

- Aumenta la velocidad y la eficiencia del proceso de desarrollo: Al automatizar muchas de las tareas manuales y tediosas, los desarrolladores pueden enfocarse en escribir código de alta calidad y en solucionar problemas importantes.
- Mejora la calidad del software: La automatización de las pruebas y la entrega continua permite detectar y corregir errores de manera temprana y constante, lo que reduce el riesgo de errores graves y mejora la calidad general del software.
- Facilita la colaboración y el trabajo en equipo: La integración y entrega continua permite a los desarrolladores trabajar juntos de manera más eficiente y colaborar en tiempo real, lo que a su vez aumenta la velocidad y la eficiencia del proceso de desarrollo.

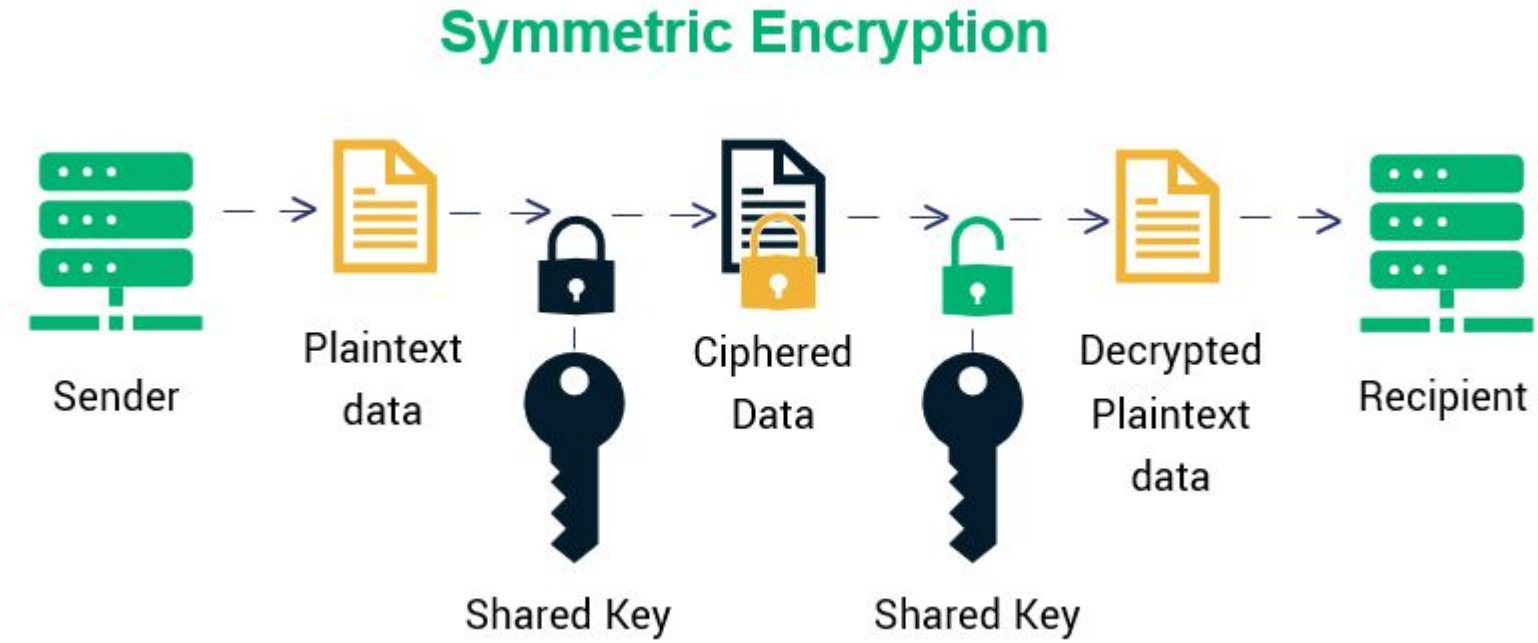
Acceso remoto - SSH

- SSH es un protocolo de red seguro que permite conectarse y controlar dispositivos a distancia.
- Es popular en sistemas operativos Unix y Linux para realizar tareas remotas de manera segura.
- El proceso de acceso remoto con SSH implica la creación de una conexión cifrada entre el dispositivo local y el servidor remoto.
- Una vez establecida la conexión, el usuario puede iniciar sesión en el servidor y ejecutar comandos en la línea de comandos.
- SSH ofrece seguridad, así como la capacidad de transferir archivos de manera segura y la opción de redirigir puertos.
- Es una herramienta esencial para los administradores de sistemas y desarrolladores que necesitan trabajar con servidores remotos de manera segura y eficiente.

Acceso remoto - SSH

- Encriptación simétrica:
  - El cifrado simétrico es un método de cifrado de datos en el que se utiliza una única clave para cifrar y descifrar la información.
  - La clave es compartida entre el remitente y el receptor, y se utiliza para cifrar el mensaje antes de ser transmitido y para descifrarlo después de ser recibido.
  - Este tipo de cifrado es rápido y eficiente en términos de recursos computacionales.
  - Sin embargo, la clave debe ser compartida de manera segura, lo que puede ser un problema de seguridad si la clave es interceptada o se pierde.
  - Todos los usuarios comparten la misma clave, por lo que si una persona malintencionada logra acceder a ella, toda la información cifrada será vulnerable.
  - Por esta razón, el cifrado simétrico se utiliza a menudo en conjunto con otros métodos de cifrado, como el cifrado asimétrico, para aumentar la seguridad de la información transmitida.

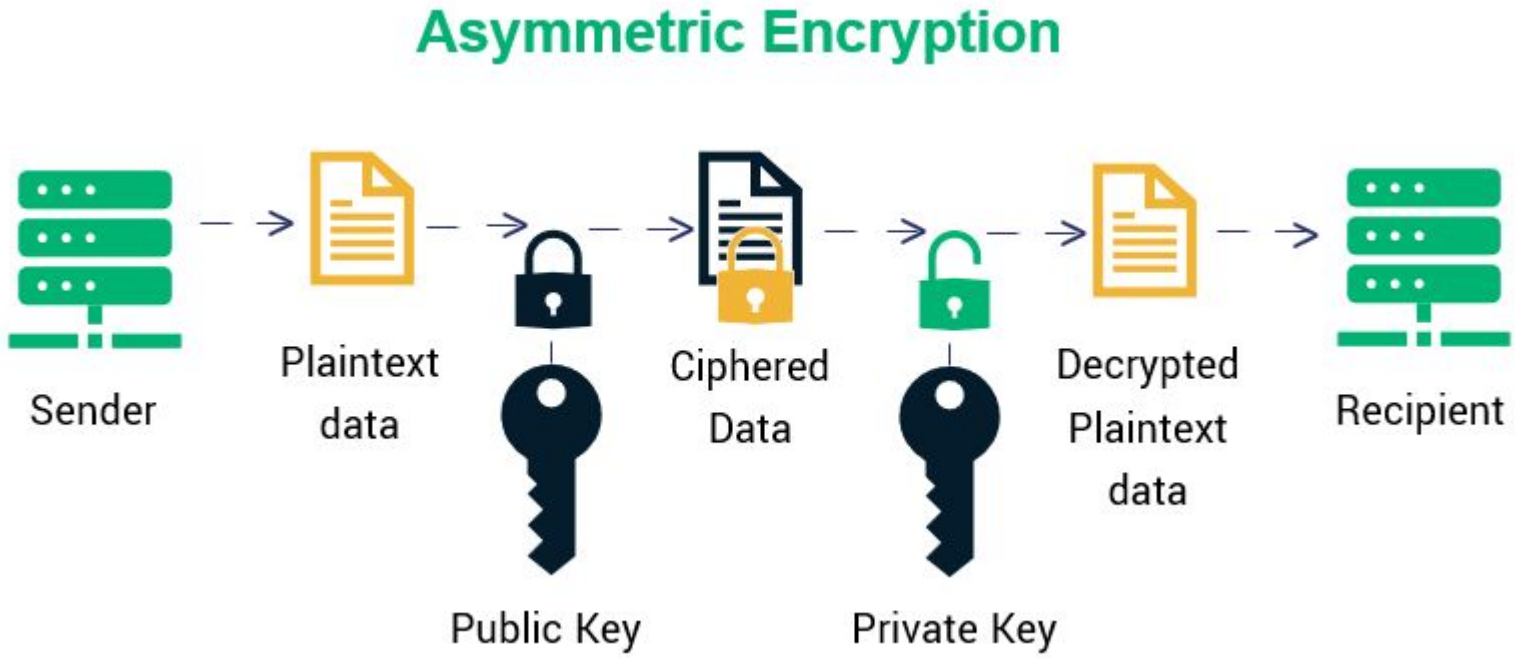
Acceso remoto - SSH



Acceso remoto - SSH

- Encriptación asimétrica:
  - El cifrado asimétrico, también conocido como cifrado de clave pública, es un método de cifrado en el que se utilizan dos claves diferentes, una clave pública y una clave privada.
  - La clave pública se puede compartir con cualquier persona, mientras que la clave privada debe ser guardada en secreto por su propietario.
  - La información cifrada con la clave pública sólo puede ser descifrada con la clave privada correspondiente, y viceversa.
  - Este método de cifrado es más seguro que el cifrado simétrico, ya que la clave pública puede ser compartida sin preocupación, mientras que la clave privada está protegida.
  - Además, el cifrado asimétrico es utilizado a menudo para establecer un canal de comunicación seguro entre dos partes, y para autenticar la identidad de un remitente a través de la verificación digital de una firma digital.

Acceso remoto - SSH



Acceso remoto - SSH

- Creación de un par criptográfico SSH (A1 - 5m)
  - Utilizando la herramienta **ssh-keygen** genera un par criptográfico **ed25519**
    - Nombre de los ficheros de claves: **ssh-github**
    - Passphrase: **testpassword**
  - Valida la clave privada con **ssh-keygen**
  - Inicia el servicio **ssh-agent** y añade la clave generada anteriormente
  - Asegurate de que la clave se ha añadido correctamente al agente

Sistemas de control de versiones (Git)

- **¿Qué son los sistemas de control de versiones?** - Son herramientas que permiten gestionar los cambios realizados en el código fuente y otros archivos de un proyecto.
- **Importancia de los sistemas de control de versiones** - Permiten a los desarrolladores trabajar de forma colaborativa y mantener un registro de todas las modificaciones realizadas en el proyecto.
- **Tipos de sistemas de control de versiones** - Existen sistemas centralizados y sistemas distribuidos. Los sistemas centralizados tienen un repositorio central que almacena toda la información del proyecto, mientras que los sistemas distribuidos permiten que cada desarrollador tenga una copia completa del repositorio en su computadora
- **Ventajas de los sistemas de control de versiones** - Permiten trabajar de forma colaborativa, mantener un historial de cambios, revertir cambios no deseados, experimentar con diferentes ideas sin afectar la versión principal del proyecto, entre otras
- **Git -** Es uno de los sistemas de control de versiones más populares y utilizado por muchos desarrolladores. Es un sistema distribuido y ofrece una gran cantidad de funcionalidades

Sistemas de control de versiones (Git)

- **Conceptos principales**
  - **Repositorio -** Es el lugar donde se almacenan los archivos y la información del proyecto
  - **Clonar un repositorio -** Permite descargar una copia del repositorio en tu computadora
  - **Rama (Branch) -** Es una línea de desarrollo separada que permite trabajar en nuevas funcionalidades sin afectar la rama principal
  - **Commit -** Permite guardar los cambios realizados en los archivos del repositorio
  - **Merge -** Combina los cambios realizados en diferentes ramas del repositorio
  - **Pull -** Permite descargar los cambios realizados en el repositorio remoto al repositorio local
  - **Push -** Permite subir los cambios realizados en el repositorio local al repositorio remoto
  - **Etiqueta (Tag) -** Permite marcar un commit específico en la historia del repositorio
  - **Conflictos -** Pueden ocurrir cuando se realizan cambios en un archivo que también fue modificado por otro colaborador
  - **Resolución de conflictos -** Permite solucionar los conflictos en un archivo para combinar los cambios realizados por diferentes colaboradores

Sistemas de control de versiones (Git)

- **El 90% del uso de Git se resume en**
  - Inicializar un repositorio local
    - Crea un directorio
    - Ábrelo
    - **git init**
  - Clonar un repositorio
    - **git clone /ruta/a/repositorio/local**
    - **git clone usuario@host:/ruta/a/repositorio**
  - Proponer cambios
    - **git add nombre_de_fichero**
    - **git add *** (para proponer todos los ficheros creados o modificados)
  - Confirmar cambios
    - **git commit -m “mensaje de confirmación”**

Sistemas de control de versiones (Git)

- **El 90% del uso de Git se resume en**
  - **Subir los cambios confirmados al repositorio remoto**
    - **git push origin <rama>**
  - **Enlazar tu repositorio local a uno remoto**
    - **git remote add origin <servidor>**
  - **Crear rama**
    - **git checkout -b <nombre-de-la-rama>**
  - **Borrar rama**
    - **git branch -d <rama>**
  - **Subir rama local a repositorio remoto**
    - **git push origin <rama>**

Sistemas de control de versiones (Git)

- El 90% del uso de Git se resume en
  - Descargar cambios del repositorio remoto (rama activa)
    - git pull
  - Fusionar rama en rama activa
    - git merge <rama>
  - Visualizar diferencias entre ramas
    - git diff <rama-origen> <rama-destino>
  - Poner un tag
    - git tag <tag> <id-del-commit-a-etiquetar>

Sistemas de control de versiones (Git)

- **El 90% del uso de Git se resume en**
  - **Info sobre el repositorio**
    - **git log**
  - **Filtrar commits**
    - **git log --author=bob** (filtrar por autor del commit)
    - **git log --pretty=oneline** (formatea la info en una línea por commit)
    - **git log --graph --oneline --decorate --all** (formato complejo)
    - **git log --name-status** (ver solo archivos modificados)

Sistemas de control de versiones (Git)

- Personal Access Token en GitHub (A3 - 5m)
  - Crea un token de acceso personal (PAT) con permisos de sólo lectura
  - Establece una expiración de 30 días para el token
  - Copia el PAT en cualquier sitio. Sólo será visible en el momento de su creación
  - Clona un repositorio cualquiera de tu cuenta utilizando el PAT en lugar del usuario/password
  - Crea un fichero testfile.txt en el repositorio local, propón los cambios, confírmalos y súbelos al repositorio remoto

Sistemas de control de versiones (Git)

- Añadir una clave SSH a GitHub (A2 - 10m)
  - Añade la clave pública generada en la actividad 1 a tu cuenta de Github
  - Valida que la clave se ha añadido correctamente
  - Crea un fichero config en el directorio correspondiente con la siguiente entrada
    - Host ssh-github
      - HostName github.com
      - User git
      - IdentityFile ~/.ssh/ssh-github
  - Modifica la URL para SSH de un repositorio cualquiera de tu cuenta sustituyendo el prefijo git@github.com por ssh-github
  - Clona el repositorio, crea un fichero **testfile.txt**, propón el cambio, confírmalo y súbelo al repositorio remoto

Github Actions

- ¿Qué es Github Actions?
  - Github Actions es una herramienta de integración y entrega continua (CI/CD) que permite automatizar tareas de desarrollo, como la compilación, pruebas, empaquetado, despliegue, entre otros.
- Beneficios de utilizar Github Actions
  - Automatización de procesos repetitivos.
  - Mejora en la calidad del código.
  - Reducción de errores.
  - Aumento de la eficiencia en el desarrollo.
  - Integración con otros servicios de Github.

Github Actions

- ¿Qué es un runner?

Un runner es un componente de Github Actions que ejecuta tareas en un entorno específico.

- Tipos de runners

Los tipos de runners que se pueden utilizar en Github Actions son:

- Hosted runners: son runners preconfigurados y mantenidos por Github, que están disponibles para su uso de forma gratuita.
- Self-hosted runners: son runners configurados y mantenidos por el usuario, que se pueden utilizar en su propio hardware o en la nube. Se ejecutan en una máquina virtual, en un contenedor de Docker, en un dispositivo físico, entre otros.

Github Actions

- Ventajas de un runner hospedado:
  - No es necesario preocuparse por la configuración o mantenimiento del runner.
  - No se necesita tener un servidor o máquina disponible para ejecutar el runner.
  - El runner hospedado está en un entorno controlado y aislado de la red, lo que aumenta la seguridad.
- Ventajas de un runner auto hospedado:
  - Permite tener un mayor control sobre la configuración del runner y su entorno de ejecución.
  - Permite ejecutar tareas que requieren recursos específicos que no están disponibles en un runner hospedado.
  - Permite ejecutar tareas en una red interna o privada que no es accesible desde un runner hospedado.

Github Actions

- Instalación de un runner auto-hospedado de Github Actions (A3 - 5m)
  - Instala un runner auto-hospedado en tu equipo siguiendo las instrucciones de la documentación de GitHub
    - https://docs.github.com/en/actions/hosting-your-own-runners/adding-self-hosted-runners

Github Actions

- ¿Qué es un **workflow**?
  - Un **workflow** es una secuencia de tareas que se ejecutan automáticamente en respuesta a un evento específico, como una solicitud de extracción o una confirmación de envío de código.
- Componentes de un **workflow**
  - **Nombre del workflow**: Este elemento se encuentra al inicio del archivo y se utiliza para nombrar el workflow. Se recomienda utilizar un nombre descriptivo para facilitar su identificación.
  - **Eventos**: Los eventos definen las acciones que desencadenan el inicio del workflow. Por ejemplo, un evento puede ser la creación de una nueva rama o el envío de un pull request. Los eventos se definen en la parte superior del archivo, antes de la sección de trabajos (jobs).

Github Actions

- **Trabajos (jobs)**: Los trabajos son la sección principal del workflow y se utilizan para definir las tareas que se deben realizar. Cada trabajo tiene un nombre y consiste en una o más tareas. Los trabajos se definen después de los eventos.
- **Tareas (steps)**: Las tareas son las acciones específicas que se deben realizar dentro de un trabajo. Cada tarea se define como una secuencia de comandos (scripts) que se ejecutan en un ambiente específico. Las tareas se definen dentro de cada trabajo.
- **Ambientes (environments)**: Los ambientes se utilizan para definir los entornos de ejecución para los trabajos y las tareas. Por ejemplo, se puede definir un ambiente de pruebas o de producción. Los ambientes se definen antes de la sección de trabajos.
- **Matriz de ejecución (matrix)**: La matriz de ejecución se utiliza para definir una lista de variables que se utilizan para ejecutar el mismo conjunto de tareas en diferentes entornos. Por ejemplo, se puede definir una matriz para ejecutar las pruebas en diferentes versiones de un sistema operativo. La matriz de ejecución se define dentro de cada trabajo.
- **Acciones (actions)**: Las acciones son paquetes de código reutilizable que se pueden utilizar dentro de un trabajo. Las acciones se definen en la parte superior del archivo, antes de la sección de eventos.
- **Ejecución condicional**: Se pueden utilizar expresiones condicionales para definir la ejecución de tareas o trabajos en función de una o varias variables. Por ejemplo, se puede definir que una tarea se ejecute solo si una variable de entorno cumple una condición determinada.

Github Actions

```yaml
name: Build and deploy

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Install dependencies
        run: npm install

      - name: Build application
        run: npm run build

      - name: Deploy to production
        uses: my-actions/deploy-to-production@v1.0.0
        with:
          environment: production
          secret_key: ${{ secrets.SECRET_KEY }}

env:
  PRODUCTION_URL: https://my-production-app.com
```

En este ejemplo, se define un workflow llamado "Build and deploy" que se activa cuando se realiza un push en la rama main. El workflow consta de un trabajo llamado "build", que se ejecuta en un ambiente de Ubuntu Latest y contiene cuatro tareas.

- La primera tarea es "Checkout code", que utiliza la acción "actions/checkout@v2" para descargar el código del repositorio.
- La segunda tarea es "Install dependencies", que instala las dependencias del proyecto utilizando el comando "npm install".
- La tercera tarea es "Build application", que construye la aplicación utilizando el comando "npm run build".
- La cuarta tarea es "Deploy to production", que utiliza una acción personalizada "my-actions/deploy-to-production@v1.0.0" para desplegar la aplicación en un ambiente de producción. La acción toma dos parámetros, "environment" y "secret_key", que se definen mediante la sintaxis "${{ secrets.SECRET_KEY }}" para mantenerlos seguros.
- Por último, se define una variable de entorno llamada "PRODUCTION_URL" que contiene la URL de producción del proyecto.

En este ejemplo, la acción "my-actions/deploy-to-production@v1.0.0" se utiliza para desplegar la aplicación en un ambiente de producción. Las acciones son paquetes de código reutilizable que se pueden utilizar dentro de un trabajo. Estas acciones pueden ser personalizadas o predefinidas y están disponibles en el Marketplace de Github Actions.

Github Actions

- Build & Push de una imagen de docker al Docker Hub (A4 - 15m)
  - Crea un repositorio en GitHub que contenga los estáticos de un sitio web cualquiera
  - Crea un repositorio en Docker Hub llamado “webaction-sample”
  - Crea un Dockerfile a partir de la última versión de nginx que sirva los estáticos de la web
  - Crea un fichero build-and-push.yml en la ruta .github/workflows
  - Completa el workflow anterior para que construya la imagen a partir del Dockerfile anterior y la “*pushee”* al repositorio “webaction-sample” creado anteriormente
    - Actions de terceros a emplear:
      - https://github.com/marketplace/actions/checkout
      - https://github.com/marketplace/actions/docker-metadata-action
      - https://github.com/marketplace/actions/build-and-push-docker-images

Referencias

- Casi todo lo que hay que saber de Git
  - https://rogerdudler.github.io/git-guide/
- Documentación GitHub
  - https://docs.github.com/es
- Cheatsheet SSH
  - https://www.marcobehler.com/guides/ssh-commands
- ¿Cómo funciona la criptografía de clave pública?
  - https://www.tutorialspoint.com/cryptography/public_key_encryption.htm
- Una guía algo más seria sobre criptografía
  - https://www.ibm.com/docs/en/ztpf/1.1.0.15?topic=concepts-public-key-cryptography
- GH Actions Cheatsheet
  - https://github.github.io/actions-cheat-sheet/actions-cheat-sheet.pdf

Tema 3: Taller de despliegues

Asignatura: Despliegue de aplicaciones Web

Contenidos

1. CRUD con FASTAPI y MySQL
2. CI/CD con GitHub Actions
3. *Frontal con Bootstrap

CRUD con FASTAPI y MySQL 17:38

- ¿Qué es el protocolo **HTTP**?
  - **HTTP** es un protocolo de aplicación utilizado para la transferencia de datos en la World Wide Web (WWW).
  - **HTTP** se ejecuta sobre el protocolo de transporte TCP (Transmission Control Protocol) o, en algunos casos, sobre UDP (User Datagram Protocol).
  - **HTTP** utiliza el modelo cliente-servidor, donde un cliente envía una solicitud a un servidor, y el servidor envía una respuesta al cliente.
  - Las solicitudes y respuestas de **HTTP** están formateadas en mensajes que contienen un encabezado y un cuerpo de mensaje. El encabezado describe la solicitud o respuesta, mientras que el cuerpo contiene los datos que se están transfiriendo.
  - Los métodos de solicitud comunes de **HTTP** incluyen GET, POST, PUT, DELETE y HEAD. Cada método se utiliza para realizar una acción diferente en el recurso solicitado.
  - Los códigos de respuesta de **HTTP** indican el resultado de una solicitud. Por ejemplo, el código de respuesta 200 indica que la solicitud fue exitosa, mientras que el código de respuesta 404 indica que el recurso solicitado no se pudo encontrar.
  - **HTTP** es un protocolo sin estado, lo que significa que cada solicitud y respuesta son independientes entre sí. Esto permite que las solicitudes y respuestas se procesen de manera más eficiente y rápida.

CRUD con FASTAPI y MySQL

- ¿Qué es el protocolo **HTTP**?
  - GET: Se utiliza para solicitar un recurso específico del servidor. La solicitud se envía en la URL y el servidor devuelve el recurso solicitado en la respuesta. Este verbo no tiene cuerpo de mensaje en la solicitud y se utiliza comúnmente para recuperar páginas web, imágenes y otros recursos estáticos.
  - POST: Se utiliza para enviar datos al servidor, generalmente para crear o actualizar recursos. El cuerpo de mensaje de la solicitud contiene los datos que se envían al servidor. A menudo, se utiliza para enviar formularios, enviar datos de usuario, y en general para enviar cualquier tipo de información que necesite ser procesada en el servidor.
  - PUT: Se utiliza para actualizar un recurso existente en el servidor. El cuerpo de mensaje de la solicitud contiene el nuevo estado del recurso. Si el recurso no existe, se crea uno nuevo.
  - DELETE: Se utiliza para eliminar un recurso específico del servidor. La solicitud no contiene cuerpo de mensaje y el servidor simplemente elimina el recurso si existe.
  - HEAD: Es similar a GET, pero en lugar de devolver el recurso solicitado, devuelve solo los encabezados de respuesta. Esto es útil cuando solo se necesita conocer la información del recurso, como su tamaño o la fecha de última modificación.

CRUD con FASTAPI y MySQL

- ¿Qué es una **API REST**?
  - Una **API REST** es una interfaz de programación de aplicaciones que utiliza el protocolo **HTTP** para permitir que diferentes sistemas se comuniquen entre sí.
  - **REST** significa **Representational State Transfer**, y se basa en una serie de principios que guían el diseño y la implementación de la API.
  - Las características principales de una **API REST** incluyen:
    - La comunicación se realiza a través de solicitudes **HTTP**, utilizando los verbos **HTTP** para indicar la acción que se desea realizar (**GET**, **POST**, **PUT**, **DELETE**, etc.).
    - La **API** es stateless, lo que significa que cada solicitud se procesa de forma independiente, sin tener en cuenta ninguna solicitud anterior.
    - Los recursos son identificados por **URIs** (**Uniform Resource Identifiers**), y las operaciones sobre estos recursos se realizan mediante la manipulación de su representación, que puede ser en diferentes formatos (JSON, XML, etc.).
    - La **API** debe seguir los principios de **HATEOAS** (Hypermedia as the Engine of Application State), lo que significa que la respuesta a una solicitud debe incluir enlaces a otros recursos relevantes, permitiendo la navegación a través de la API de manera más intuitiva.
  - Las **API REST** se utilizan comúnmente para la creación de aplicaciones web, la integración de sistemas y la comunicación entre diferentes servicios en la nube.
  - Para implementar una **API REST**, es necesario definir los recursos que se expondrán, definir las operaciones que se pueden realizar sobre ellos y especificar los formatos de los datos que se utilizarán. Además, se deben definir las políticas de seguridad y autenticación para proteger la **API** de accesos no autorizados.

CRUD con FASTAPI y MySQL

- ¿Qué es un CRUD?
  - Un CRUD es un acrónimo que representa las operaciones básicas de cualquier sistema de gestión de bases de datos o aplicaciones web que involucren interacción con una base de datos
    - Create (Crear): esta operación se refiere a la creación de un nuevo registro en la base de datos. Por ejemplo, si estamos trabajando en una aplicación de gestión de clientes, podríamos usar la operación "Create" para crear un nuevo registro con información como el nombre del cliente, dirección, número de teléfono, etc.
    - Read (Leer): esta operación se refiere a la lectura o consulta de información existente en la base de datos. Por ejemplo, si estamos trabajando en una aplicación de gestión de clientes, podríamos usar la operación "Read" para buscar información sobre un cliente específico, como su nombre, dirección, número de teléfono, etc.
    - Update (Actualizar): esta operación se refiere a la actualización de información existente en la base de datos. Por ejemplo, si estamos trabajando en una aplicación de gestión de clientes, podríamos usar la operación "Update" para cambiar la dirección de un cliente o actualizar su número de teléfono.
    - Delete (Borrar): esta operación se refiere a la eliminación de información existente en la base de datos. Por ejemplo, si estamos trabajando en una aplicación de gestión de clientes, podríamos usar la operación "Delete" para eliminar la información de un cliente que ya no es relevante o ha dejado de hacer negocios con nosotros.

CRUD con FASTAPI y MySQL

- Provisión de servidor MySQL con Docker Compose (A1 - 15m)
  - Crea un repositorio en GitHub: worldstats-app
  - Declara en un fichero de Compose lo siguiente:
    - Network tipo bridge: worldstats-net
    - Volúmen administrado: worldstats-data
    - Servicio: db
      - imagen: mysql:5.7.41
      - nombre: worldstats-db
  - Descargar la BD de ejemplo
    - https://dev.mysql.com/doc/world-setup/en/world-setup-installation.html
    - A partir del fichero SQL descargado inicializar de forma automática la BD

CRUD con FASTAPI y MySQL

-

Referencias

- Documentación de FastApi
  - https://fastapi.tiangolo.com/
- Documentación de Traefik (Proxy inverso y LB)
  - https://doc.traefik.io/traefik/
- Guía para desplegar una API securizada con FastAPI, Traefik y Docker
  - https://towardsdatascience.com/how-to-deploy-a-secure-api-with-fastapi-docker-and-traefik-b1ca065b100f

Índice de grabaciones

- **Sesión 1** - Viernes 20 de enero de 2023 (17:00 a 20:00)
  - Inicio Sesión 1. Criterios de evaluación y planificación - 0h0m0s
  - Motivación Docker - 0h26m20s
  - Docker CLI - 0h41m0s
  - Instalación Docker - 0h50m0s
  - Descanso - 1h18m0s
  - Cont. Docker CLI - 1h32m0s
  - Imágenes y Registros - 1h59m0s
  - Actividad 1 - 2h0m0s
- **Sesión 2** - Viernes 27 de enero de 2023 (17:00 a 20:00)
  - Inicio Sesión 2 - 0h0m0s
  - Actividad 2 - 0h35m34s
  - Problemas + Descanso - 1h14m30s
  - Cont. Actividad 2 - 1h33m0s
  - Volúmenes - 2h04m0s

Índice de grabaciones

- **Sesión 3** - Viernes 10 de febrero de 2023 (17:00 a 20:00)
  - Inicio Sesión 3 - 0h0m0s
  - Docker Compose - 0h21m40s
  - Spec docker-compose.yaml - 0h45m00s
  - Descanso - 1h19m0s
  - Actividad 1 - 1h36m0s
  - Actividad 2 - 2h15m0s
- **Sesión 4** - Viernes 17 de febrero de 2023 (17:00 a 20:00)
  - Inicio Sesión 4. Repaso Docker Compose - 0h0m0s
  - Inicio Tema 3. - 0h29m14s
  - Acceso remoto SSH - 0h52m07s
  - Descanso - 1h15m0s
  - Actividades SSH - 1h35m0s
  - GIT - 1h59m0s

Índice de grabaciones

- **Sesión 5** - Viernes 3 de marzo de 2023 (17:00 a 20:00)
  - Repaso sesión 4 - 0h0m0s
  - Github Actions - 0h18m52s
  - Actividad 3 - 0h45m00s
  - GH Workflows - 0h59m33s
  - Descanso - 1h31m12s
  - Actividad 4 - 1h45m44s