



Módulo 2

Docker



Revolución tecnológica y contenedores



~\ (ツ) ~

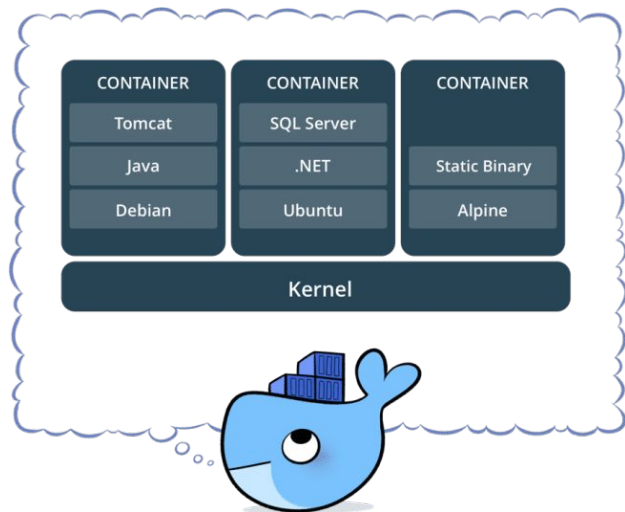
IT WORKS
on my machine

Un poco de historia

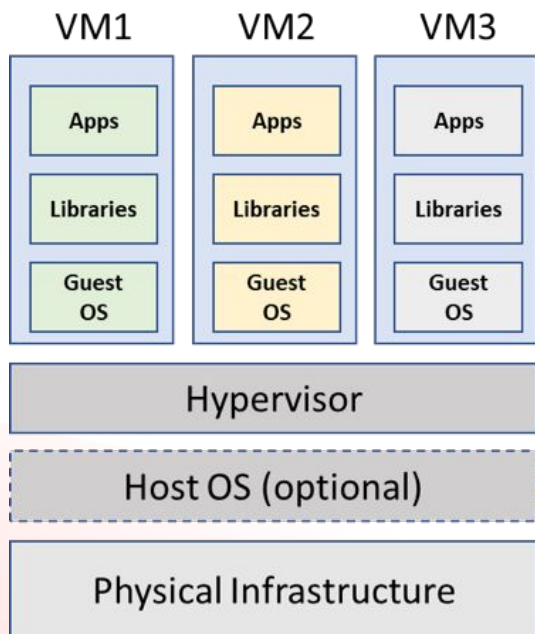


¿Qué son los contenedores?

- Un contenedor es una unidad estandarizada de software que contiene el código de una aplicación y sus dependencias

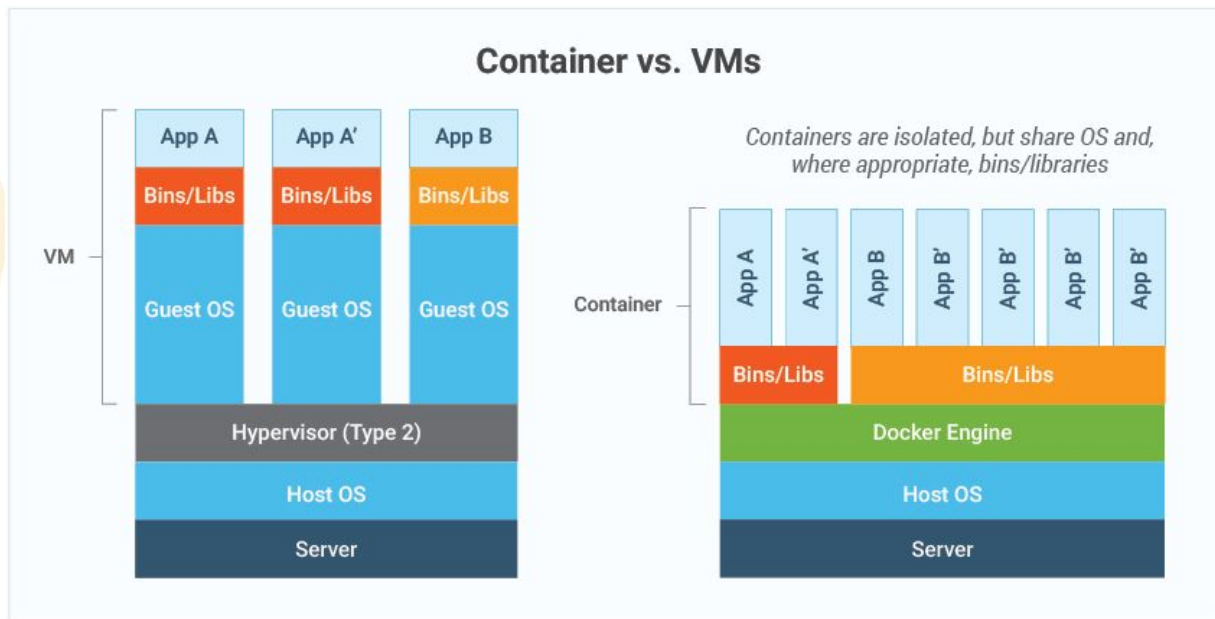


Virtualización



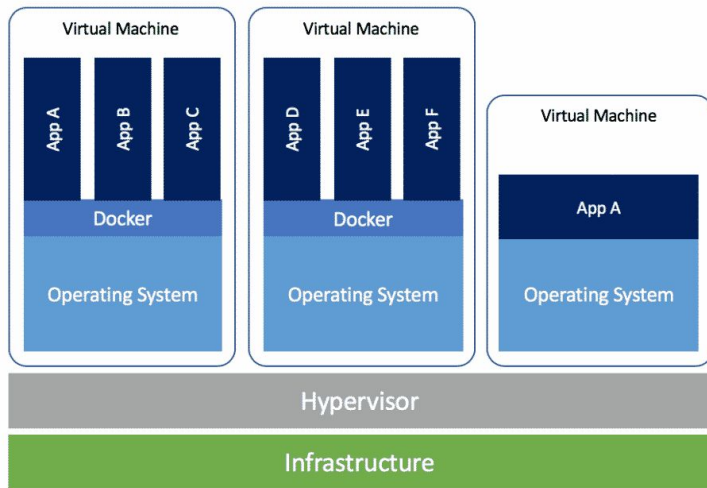
- ➔ **Hypervisor:** VMware, VirtualBox, etc
- ➔ **Sistema operativo:** Windows, Linux, etc
- ➔ **Hardware físico:** PC, servidores

Diferencia entre VMs y contenedores



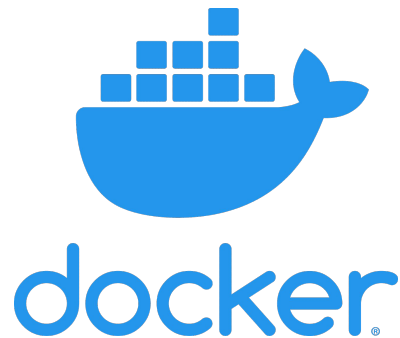
VMs y contenedores

En general, ambas tecnologías se utilizan en conjunto



¿Qué es Docker?

- Es un proyecto de código abierto, que permite construir, ejecutar y administrar **contenedores**
- Provee una solución para que las aplicaciones puedan ejecutarse en cualquier máquina con Docker instalado, independientemente de la arquitectura o sistema operativo



Registry

- Sistema de distribución y almacenamiento para **imágenes**
- Permite el versionado mediante el uso de **tags**
- Se organiza en **repositorios**, donde, en cada uno de ellos, se encuentran todas las versiones de una imagen específica



<https://hub.docker.com/>

Registry

docker **pull** redis



```
- > docker pull redis
Using default tag: latest
latest: Pulling from library/redis
42c077c10790: Already exists
a300d83d65f9: Pull complete
ebdc3afaab5c: Pull complete
31eec7f8651c: Pull complete
9c6a6b89d274: Pull complete
5c8099a4b45c: Pull complete
Digest: sha256:1b90dbfe6943c72a7469c134cad3f02eb810f016049a0e19ad78be07040cdb0c
Status: Downloaded newer image for redis:latest
docker.io/library/redis:latest
```

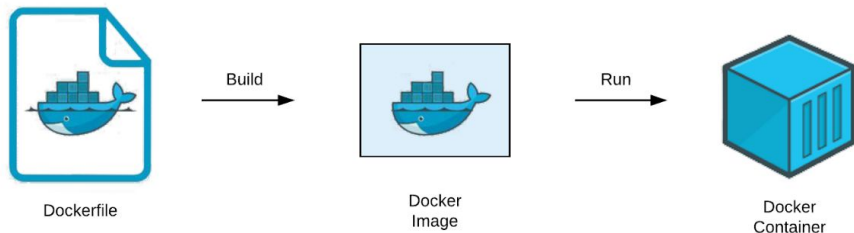
docker **run** redis



```
- > docker run redis
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
42c077c10790: Already exists
a300d83d65f9: Pull complete
ebdc3afaab5c: Pull complete
31eec7f8651c: Pull complete
9c6a6b89d274: Pull complete
5c8099a4b45c: Pull complete
Digest: sha256:1b90dbfe6943c72a7469c134cad3f02eb810f016049a0e19ad78be07040cdb0c
Status: Downloaded newer image for redis:latest
1:C 06 Jun 2022 00:38:35.939 # o000o000o000o Redis is starting o000o000o000o
1:C 06 Jun 2022 00:38:35.939 # Redis version=7.0.0, bits=64, commit=00000000, modified=0,
pid=1, just started
```

Contenedores vs. imágenes

- Una **imagen** es un archivo inmutable que contiene el código fuente, librerías, dependencias y otros archivos que se necesitan para correr una aplicación
- Un **contenedor** es una instancia de una imagen en ejecución



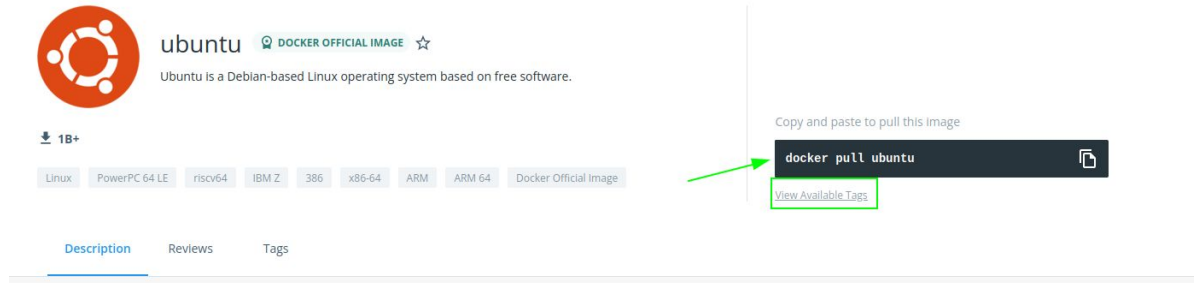
Docker Hub

- Docker Hub es un servicio proporcionado por Docker para buscar y compartir imágenes de contenedores.
- Es el repositorio de imágenes de contenedores más grande del mundo.
- Cuenta con una variedad de fuentes que incluyen desarrolladores de la de contenedores, proyectos open-source y proveedores de software independientes (ISV) que crean y distribuyen su código en contenedores.



HandsOn – Docker pull & docker run

1. Explorar DockerHub y buscar imágenes verificadas (Ubuntu, Alpine, Nginx, PostgreSQL, Node, Etc).
 - a. Leer documentación
 - b. Obtener y lanzar comando de **pull**
 - c. Utilización de tags



HandsOn – Docker pull & docker run

- Docker **images**
 - a. `docker images --help`
- Docker **run** parameters
 - a. `--help`
 - b. `--detach -d`
 - c. `--interactive -i`
 - d. `--tty -t`
 - e. `-n --network`
 - f. `--env --env-file`
- Docker **ps**
 - a. `docker ps --help`
- Docker **exec**
 - a. `docker exec --help`
 - b. `--it`

HandsOn – Docker pull & docker run

- Hacer pull a 2 imagenes de las exploradas en el punto anterior.
- Revisar documentación por configuraciones particulares.
- Lanzar 2 contenedores en una misma red:
 - a. `docker run - -help | grep network`
 - b. `docker network -help`
- Comunicar 2 contenedores:
 - a. Entrar en terminal interactiva a contenedor de Ubuntu/alpine
 - b. Instalar **ping**
 - c. Lanzar ping usando el nombre del contenedor

Dockerfile

Es un archivo de texto donde se definen los comandos que se necesitan realizar para construir una **imagen**

Seleccionar Sistema operativo	<code>FROM ubuntu</code>
Actualizar paquetes e instalar python	<code>RUN apt-get update && \</code> <code>apt-get -y install python</code>
Instalar dependencias de python con pip	<code>RUN pip install flask</code>
Copiar el código fuente al directorio /opt	<code>COPY . /opt/webapp</code>
Ejecutar el servidor web	<code>CMD ["python", "/app/webapp/app.py"]</code>

Dockerfile

```
FROM ubuntu
RUN apt-get update && \
    apt-get -y install python
RUN pip install flask
COPY . /opt/webapp
CMD ["python", "/app/webapp/app.py"]
```

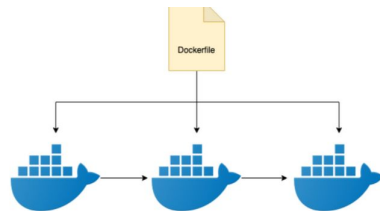
Instrucción + argumentos

- **Capa 1:** Imagen base
- **Capa 2:** Cambios en paquetes instalados
- **Capa 3:** Cambios en paquetes de pip
- **Capa 4:** Copiado del código fuente
- **Capa 5:** Ejecución del comando de python

HandsOn – Docker build

- Utilizando los recursos brindados en el repositorio de la clase 2, debe analizarse el contenido de los Dockerfile provistos y comprender cada etapa.
- Construir ambas imágenes utilizando los flags mencionados anteriormente.
- Lanzar ambos contenedores en la misma red.
- Analizar los logs y funcionamiento de los contenedores.

Multistage Building



- Uno de los desafíos más grandes sobre la creación de imágenes es mantener el tamaño de la imagen bajo.
- Muchas veces gran parte del tamaño de una imagen corresponde a librerías o paquetes que solo se utilizan en etapas de build o compilación.
- Para solventar este problema se utiliza el multistage building.
- Se utiliza una imagen base para la etapa de compilación y luego una imagen distinta para la etapa de ejecución.

ref: <https://docs.docker.com/develop/develop-images/multistage-build/>

DEMO – Multistage Building

- Utilizaremos el siguiente repositorio publico de GitHub:
 - a. <https://github.com/codefresh-contrib/golang-sample-app>
- Realizamos Build usando Dockerfile y analizamos la imagen
- Realizamos Build usando multistage building y analizamos la imagen



Buenas prácticas

- Uso de tags (versiones).
- Orden de las capas del Dockerfile
- Evitar imágenes grandes
- Seguir lineamientos productivos

¿Preguntas?



Tarea 1 – TodoList Flask

- Clonar el siguiente repositorio:
 - a. <https://github.com/craftech-academy/academy-devops-junio-2022-clases>
- Usar recursos ubicados en:
 - a. [academy-devops-junio-2022-clases/clase2/homework/0-todo-list-flask](#)
- Lograr lanzar un “docker build” a partir del Dockerfile brindado.
- Lograr lanzar un “docker run” a partir de la imagen creada.
- **Nota:** los archivos brindados cuentan con errores que deberán sortear para resolver el desafío.
- **Recomendación:** amigarse con “docker logs” :')

Tarea 1 - TodoList Flask

- Funcionamiento esperado:

```
mxchxdx@mxchxdx-craftech: ~/Documents/Craftech/Curso/clase2-docker/homework/0-todo-list-flask [master +] docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
bc1098550508   counter-app:0.0.1  "/bin/sh -c 'python3..."  4 minutes ago  Up 4 minutes  0.0.0.0:80->5000/tcp, :::80->5000/tcp  counter-app
```



To Do App

Task Title

Todo list funcionando! :)

Add



Task Title

Enter Todo...

Add

1 | Todo list funcionando! :)

Not Complete Finished Delete

Tarea 2 – API Django

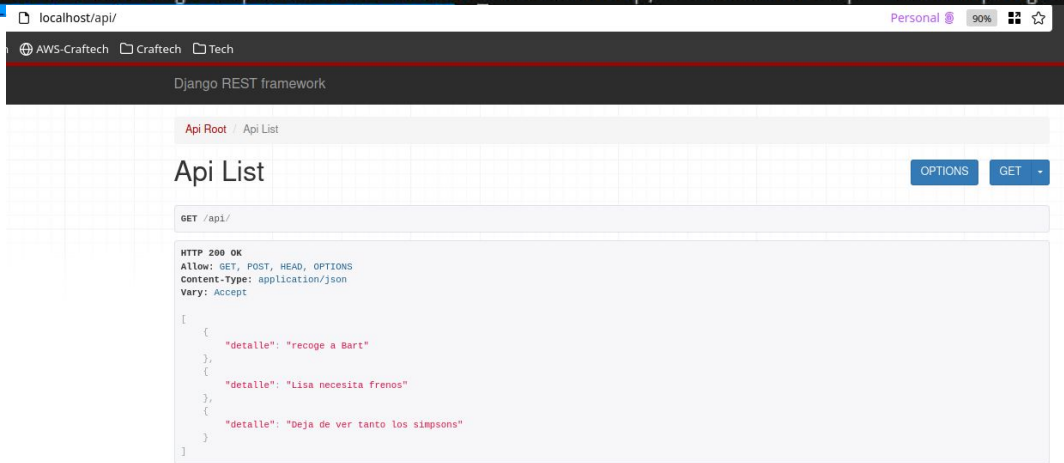
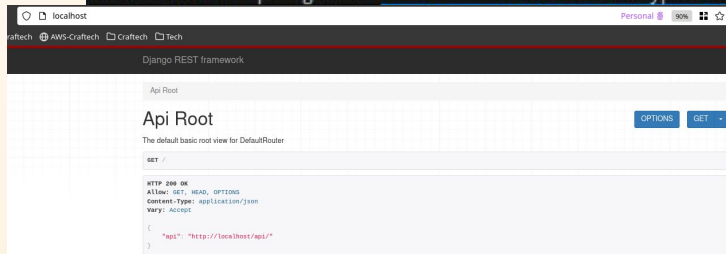
- Clonar el siguiente repositorio:
 - a. <https://github.com/craftech-academy/academy-devops-junio-2022-clases>
- Usar recursos ubicados en:
 - a. [academy-devops-junio-2022-clases/clase2/homework/1-api-django](#)
- Leer README.md y crear un Dockerfile que cumpla con todos los requerimientos de la api.
- Lanzar contenedor de PostgreSQL utilizando las variables de entorno ubicadas en:
 - a. [academy-devops-junio-2022-clases/clase2/homework/1-api-django/database/.env-postgres](#)
- Lanzar contenedor con imagen de API-Django y verificar su correcto funcionamiento (*localhost:<port>/api/*).
- **Nota:** los archivos brindados cuentan con errores que deberán sortear para resolver el desafío.

Tarea 2 – API Django

- Funcionamiento esperado:

```
mxchxdx@mxchxdx-craftech: ~ - ssh - /Documents/Craftech/Curso/clase2-docker/homework/1-api-django/backend$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
56952eed2c7a	python-backend:0.0.1	"./entrypoint.sh"	2 seconds ago	Up 1 second	0.0.0.0:80->8001/tcp, :::80->8001/tcp	backend-python
b988225ab19a	postgres	"docker-entrypoint.s..."	5 seconds ago	Up 4 seconds	0.0.0.0:5432->5432/tcp, :::5432->5432/tcp	backend-postgresql





CRAFTECH

Your DevOps team

¡Muchas gracias!