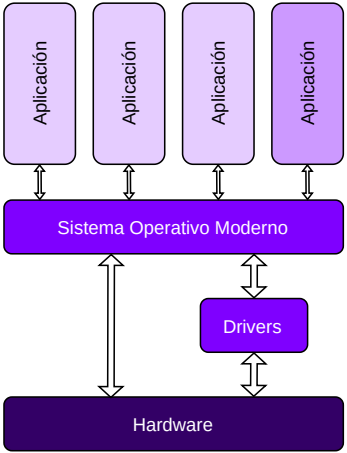
Introducción a Docker y `docker-compose`

Talleres *i*Lab / DEI-USM

Israel Figueroa

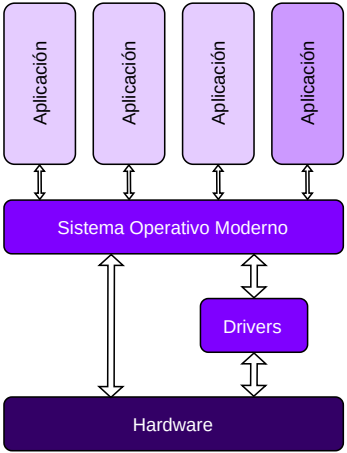
USM - Sede Concepción

israel.figueroa@usm.cl



Visión tradicional

¿Posibles inconvenientes al momento de integrar una aplicación nueva al sistema?

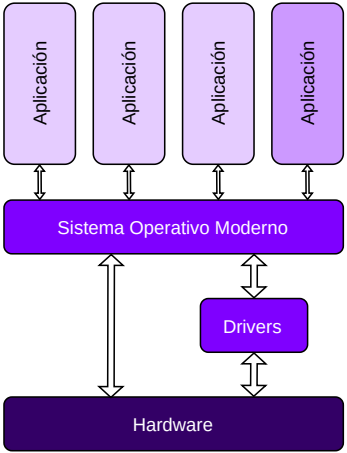


Visión tradicional

¿Posibles inconvenientes al momento de integrar una aplicación nueva al sistema?

- Aplicación diseñada para otro Sistema Operativo

Hardware y SO

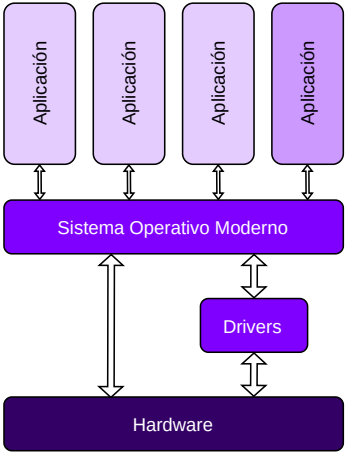


Visión tradicional

¿Posibles inconvenientes al momento de integrar una aplicación nueva al sistema?

- Aplicación diseñada para otro Sistema Operativo
- Aplicación interfiere con el funcionamiento de las demás aplicaciones

Hardware y SO

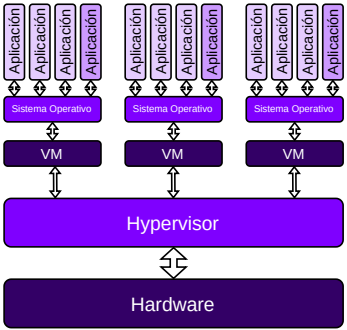


Visión tradicional

¿Posibles inconvenientes al momento de integrar una aplicación nueva al sistema?

- Aplicación diseñada para otro Sistema Operativo
- Aplicación interfiere con el funcionamiento de las demás aplicaciones
- El síndrome "En mi PC si funciona".

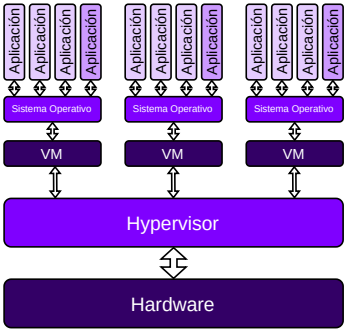
Virtualización: El músculo detras de la nube



Caso Virtualización

La virtualización actualmente permite usar la misma infraestructura (Hardware), para crear máquinas virtuales **totalmente** independientes. ¿Será suficiente?

Virtualización: El músculo detras de la nube

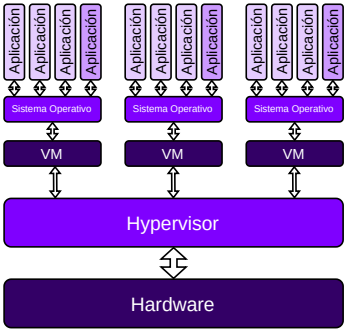


Caso Virtualización

La virtualización actualmente permite usar la misma infraestructura (Hardware), para crear máquinas virtuales **totalmente** independientes. ¿Será suficiente?

- Cada Sistema Operativo requiere una maquina virtual.

Virtualización: El músculo detras de la nube

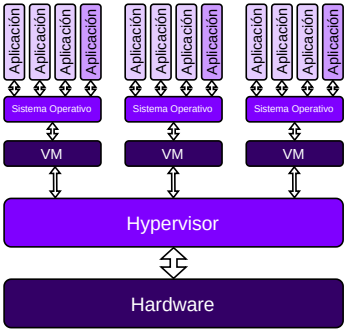


Caso Virtualización

La virtualización actualmente permite usar la misma infraestructura (Hardware), para crear máquinas virtuales **totalmente** independientes. ¿Será suficiente?

- Cada Sistema Operativo requiere una maquina virtual.
- Cada Sistema Operativo debe ser instalado manualmente.

Virtualización: El músculo detras de la nube

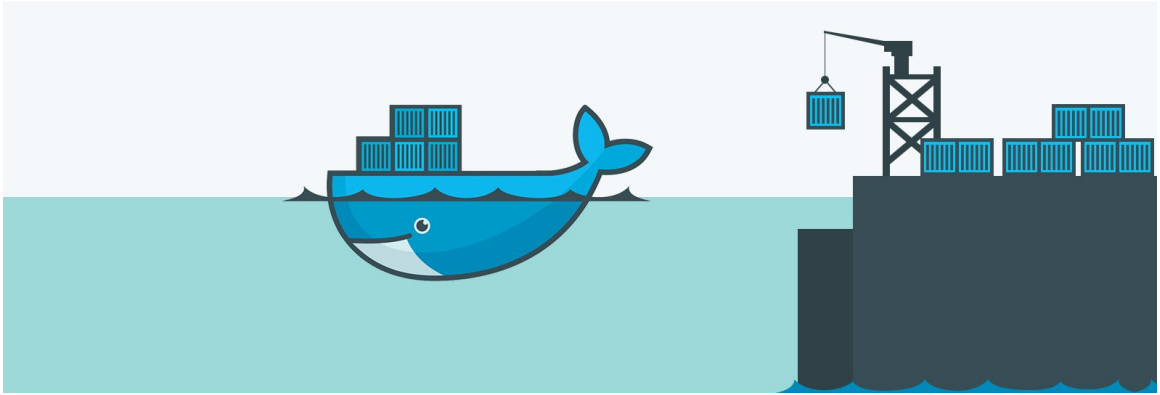


Caso Virtualización

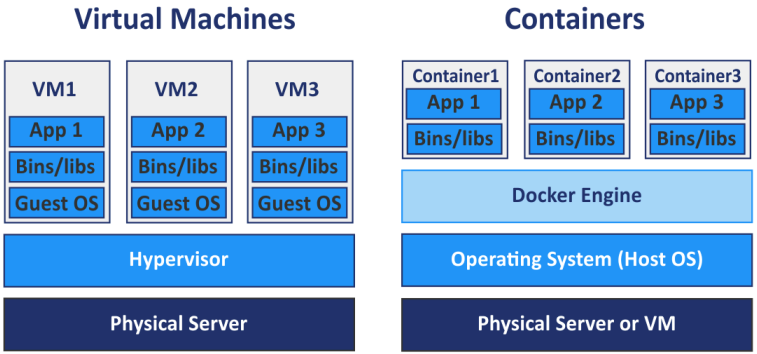
La virtualización actualmente permite usar la misma infraestructura (Hardware), para crear máquinas virtuales **totalmente** independientes. ¿Será suficiente?

- Cada Sistema Operativo requiere una maquina virtual.
- Cada Sistema Operativo debe ser instalado manualmente.
- Provee una total encapsulación, pero a un costo (Worth it?).

Hasta que llegó Docker!

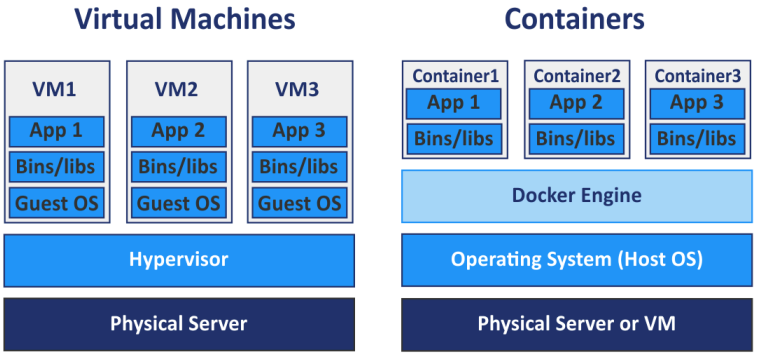


Modelo Docker: Infraestructura



¿Que es docker?

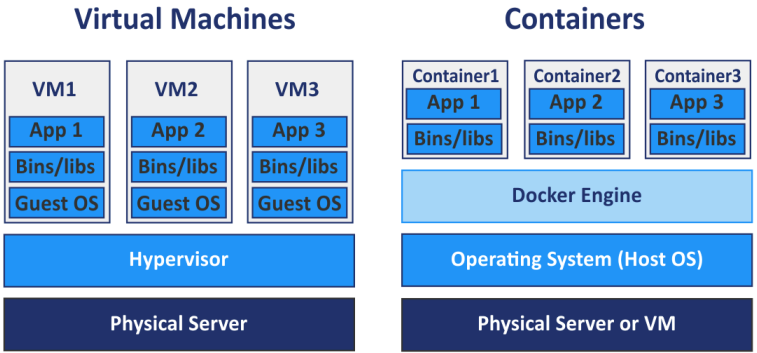
Modelo Docker: Infraestructura



¿Que es docker?

- Docker **no es** una máquina virtual.

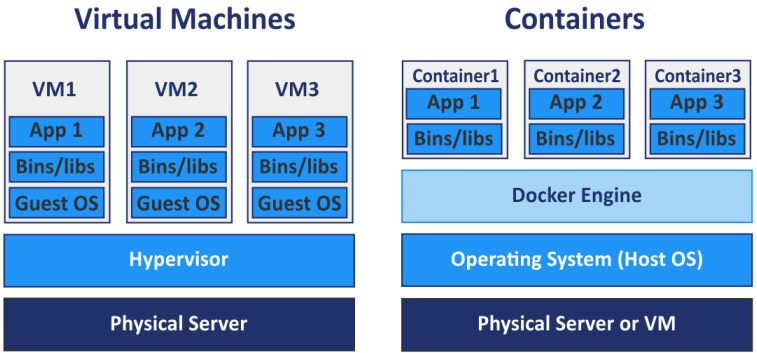
Modelo Docker: Infraestructura



¿Que es docker?

- Docker **no es** una máquina virtual.
- Docker **no es** un sistema operativo.

Modelo Docker: Infraestructura



¿Que es docker?

- Docker **no es** una máquina virtual.
- Docker **no es** un sistema operativo.
- Docker es un gestor de aplicaciones que permite la creación, descarga y ejecución de programas dentro de *contenedores de software*.

docker: Hello world

Nota: En linux debe agregar `sudo` antes de `docker`

Ejercicio 1: hello world

```
docker run hello-world
```

docker: Hello world

Nota: En linux debe agregar `sudo` antes de `docker`

Ejercicio 1: hello world

```
docker run hello-world
```

Ejercicio 2: getting started

```
docker run -d -p 12345:80 docker/getting-started
```

¿Que ocurre al ejecutar el comando?
Abra en el navegador la dirección: `http://localhost:12345`

docker: Hello world

Nota: En linux debe agregar `sudo` antes de `docker`

Ejercicio 1: hello world

```
docker run hello-world
```

Ejercicio 2: getting started

```
docker run -d -p 12345:80 docker/getting-started
```

¿Que ocurre al ejecutar el comando?
Abra en el navegador la dirección: `http://localhost:12345`

Ejercicio 3: getting started x2

```
docker run -d -p 9999:80 docker/getting-started
```

Abra en el navegador la dirección: `http://localhost:9999`

Break: Repo del Taller

Los ejemplos y ejercicios vistos a continuación están disponibles en un repositorio de `git`. Descárguelos y utilice el editor para modificar codigos y ver su contenido.

Cree un directorio en el computador y ejecute el siguiente comando de consola dentro del directorio.

```
git clone http://dev.ilab.cl/ilab/taller-docker.git
```

Repositorio de Contenedores

Docker no es solo un software, también es un *app store* de contenedores, donde **cualquiera** puede publicar sus contenedores de manera pública para sus desarollos.

Cuando intentas ejecutar o utilizar un contenedor(imagen) que `docker` no tiene localmente en su computador, el programa lo buscará y descargará desde:

`http://hub.docker.com/`

Ejemplo: Directorio 1.ciclos

Actividad

En el directorio ejecute:

```
docker build -t ciclo .
```

```
1  FROM python:3
2
3  COPY myapp.py .
4
5  ENTRYPOINT ["python"]
6
7  CMD ["myapp.py"]
8
```

Ejemplo: Directorio 1.ciclos

Actividad

En el directorio ejecute:

```
docker build -t ciclo .
```

```
1  FROM python:3
2
3  COPY myapp.py .
4
5  ENTRYPOINT ["python"]
6
7  CMD ["myapp.py"]
8
```

Como funciona: Paso a paso

- La primera línea especifica la imagen/capa desde la cual se hará el contenedor (`ubuntu:20.04`, `debian:10`, `python:2`, etc)

Ejemplo: Directorio 1.ciclos

Actividad

En el directorio ejecute:

```
docker build -t ciclo .
```

```
1  FROM python:3
2
3  COPY myapp.py .
4
5  ENTRYPOINT ["python"]
6
7  CMD ["myapp.py"]
8
```

Como funciona: Paso a paso

- La primera línea especifica la imagen/capa desde la cual se hará el contenedor (`ubuntu:20.04`, `debian:10`, `python:2`, etc)
- La línea 3 copia el archivo `myapp.py` al contenedor.

Ejemplo: Directorio 1.ciclos

Actividad

En el directorio ejecute:

```
docker build -t ciclo .
```

```
1  FROM python:3
2
3  COPY myapp.py .
4
5  ENTRYPOINT ["python"]
6
7  CMD ["myapp.py"]
8
```

Como funciona: Paso a paso

- La primera línea especifica la imagen/capa desde la cual se hará el contenedor (`ubuntu:20.04`, `debian:10`, `python:2`, etc)
- La línea 3 copia el archivo `myapp.py` al contenedor.
- Las últimas dos instrucciones indican que al ejecutar el contenedor el comando a ejecutar es '`python myapp.py`'.

Ejemplo: Directorio 1.ciclos

Actividad

En el directorio ejecute:

```
docker build -t ciclo .
```

```
1  FROM python:3
2
3  COPY myapp.py .
4
5  ENTRYPOINT ["python"]
6
7  CMD ["myapp.py"]
8
```

Como funciona: Paso a paso

- La primera línea especifica la imagen/capa desde la cual se hará el contenedor (`ubuntu:20.04`, `debian:10`, `python:2`, etc)
- La línea 3 copia el archivo `myapp.py` al contenedor.
- Las últimas dos instrucciones indican que al ejecutar el contenedor el comando a ejecutar es '`python myapp.py`'.
- Una vez creado el contenedor puede ejecutarlo mediante: '`docker run -t ciclo`'

Algunos modificadores importantes de conocer son:

Modificadores

Algunos modificadores importantes de conocer son:

Modificadores

- **-t TAG:version**: Indica un TAG para identificar el contenedor. Normalmente tienen un modificador para indicar la versión, si no se indica la versión será: *latest*

Algunos modificadores importantes de conocer son:

Modificadores

- **-t TAG:version**: Indica un TAG para identificar el contenedor. Normalmente tienen un modificador para indicar la versión, si no se indica la versión será: *latest*
- **-p mimaquina:contenedor**: Realiza un mapeo de puertos desde el contenedor a tu maquina.

Algunos modificadores importantes de conocer son:

Modificadores

- **-t TAG:version**: Indica un TAG para identificar el contenedor. Normalmente tienen un modificador para indicar la versión, si no se indica la versión será: *latest*
- **-p mimaquina:contenedor**: Realiza un mapeo de puertos desde el contenedor a tu maquina.
- **-d**: Daemonize, indica a `docker` que quieres que el proceso se ejecute en segundo plano. Como un servicio.

Algunos modificadores importantes de conocer son:

Modificadores

- **-t TAG:version**: Indica un TAG para identificar el contenedor. Normalmente tienen un modificador para indicar la versión, si no se indica la versión será: *latest*
- **-p mimaquina:contenedor**: Realiza un mapeo de puertos desde el contenedor a tu maquina.
- **-d**: Daemonize, indica a `docker` que quieres que el proceso se ejecute en segundo plano. Como un servicio.
- **-e**: Pasa una configuración al contenedor como variable.

Cheatlist: docker

Procesos

- **docker start etiqueta/id**: Inicia una instancia del contenedor.
- **docker ps**: Indica todos las instancias en ejecución.
- **docker stop uuid**: Cierra la instancia indicada por `uuid`.
- **docker kill uuid**: Detiene la instancia `uuid`.
- **docker logs uuid**: Muestra los mensajes de la instancia `uuid`.

Imagenes

- **docker image ls**: Te muestra todas las imágenes en tu sistema.
- **docker pull contenedor**: Descarga un contenedor desde hub.docker.com.
- **docker push contenedor**: Respalda el contenedor en hub.docker.com.
- **docker build ruta**: Crea un contenedor a partir de `Dockerfile`.
- **docker image prune**: Elimina todas las imágenes en desuso en tu sistema.

Ejemplo: Directorio 2.flask

Actividad: Cree un contenedor a partir del `Dockerfile` en el directorio "2.flask".

Ejemplo: Directorio 2.flask

Actividad: Cree un contenedor a partir del `Dockerfile` en el directorio "2.flask".

```
1  FROM ubuntu:20.04
2  #FROM debian:buster-slim
3  #FROM python:3
4
5  RUN apt-get update -y
6  RUN apt-get install -y python3-pip python3-dev build-essential git
7
8
9  # Crea una cuenta de usuario y grupo para ejecutar el servidor web
10 RUN groupadd app && useradd -g app app
11
12 # Descarga/clona el la última versión del codigo del sitio desde el
   repositorio al directorio de servicio
13 RUN git clone https://dev.ilab.cl/public/pythonweb.git /srv
14
15
16 #  Inicializa el codifo (descarga las bibliotecas)
17 WORKDIR /srv/web-interface
18 RUN pip3 install setuptools gunicorn
19 RUN pip3 install --no-cache-dir -r requirements.txt
20
21 # Cambia los permisos del directorio de servicio y abre el puerto del
   firewall
22 RUN chown -R app:app /srv
23 USER app
24 EXPOSE 8000
25
26 # Indica el programa y los parametros que deben ejecutarse al ejecutar
   "ejecutar" el contenedor
27 ENTRYPOINT ["gunicorn"]
28 CMD ["-b", "0.0.0.0:8000", "wsgi:app"]
```

Servidor web Flask

En el directorio, se encuentra un Dockerfile mas complejo. Este crea un contenedor a partir de un repositorio `git` en una imagen de `ubuntu:20.04`.

Este proceso incluye pasos como la actualización del sistema e instalación de software(líneas 5 y 6), además de la instalación de bibliotecas de python necesarias para el funcionamiento del sitio en flask (líneas 18 y 19)

Ejemplo: Directorio 2.flask

Actividad: Cree un contenedor a partir del `Dockerfile` en el directorio "2.flask".

Preguntas:

- ¿En que dirección(url) queda el sitio web?
- ¿Como se inicia o se detiene el contenedor?
- ¿Que problemas tengo si quiero iniciar dos instancias el contenedor?

Ejemplo: Directorio 2.flask

Actividad: Cree un contenedor a partir del `Dockerfile` en el directorio "2.flask".

Preguntas:

- ¿En que dirección(url) queda el sitio web?
- ¿Como se inicia o se detiene el contenedor?
- ¿Que problemas tengo si quiero iniciar dos instancias el contenedor?
- ¿Como puedo hacer que el contenedor se inicie con el sistema?

Diseño de contenedores

La tendencia actual indica que los contenedores deben:

Diseño de contenedores

La tendencia actual indica que los contenedores deben:

- Ser lo mas simples posibles.

Diseño de contenedores

La tendencia actual indica que los contenedores deben:

- Ser lo mas simples posibles.
- No contener información como claves o variables de configuración.

Diseño de contenedores

La tendencia actual indica que los contenedores deben:

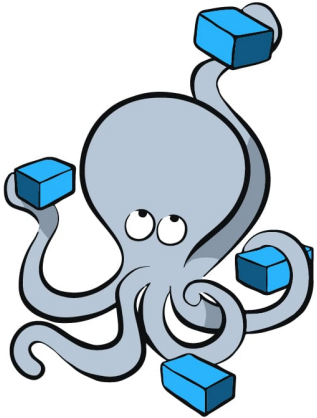
- Ser lo mas simples posibles.
- No contener información como claves o variables de configuración.
- No almacenar información del sistema (sin estado).

Diseño de contenedores

La tendencia actual indica que los contenedores deben:

- Ser lo mas simples posibles.
- No contener información como claves o variables de configuración.
- No almacenar información del sistema (sin estado).
- Ser lo más eficientes en la tarea que fueron ideados.

Docker Composer al rescate!



docker

Compose

Sistemas en un archivo: `docker-compose`

Orquestación

Al hacer sistemas complejos que requieren múltiples contenedores, se requiere que varios contenedores trabajen en conjunto. Como por ejemplo un sitio web, con una base de datos. De ésta manera, en el mundo real los contenedores normalmente no son iniciados manualmente usando `docker`, debido a que la tarea rápidamente se vuelve tediosa y compleja.

docker-compose es una herramienta básica de orquestación. Al igual que `docker` lee un archivo que tiene las instrucciones sobre que imagenes subir y cuales son los parametros de inicialización.

Ejemplo: 3.Wordpress

Actividad: Ingrese al directorio y ejecute '`docker-compose up`'. `http://localhost:8000/`

Ejemplo: 3.Wordpress

Actividad: Ingrese al directorio y ejecute '`docker-compose up`'. `http://localhost:8000/`

```yaml
version: "2"

services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
```

Wordpress

El archivo indica que se deben crear dos servicios `db` y `wordpress`. También indica que contenedor utilizar para cada uno e información de configuración.

De esta manera se crea una aplicación, usando solo un archivo de texto que contiene las instrucciones para subir el servicio y dejarlo operativo.

Orquestación

Las funcionalidades de `docker` son mucho más extensas y complejas que las vistas en el taller. Esto es sólo una demostración introductoria de lo que se puede realizar con ésta herramienta.

Al hacer sistemas complejos que requieren múltiples contenedores, se requiere que varios contenedores trabajen en conjunto. Como por ejemplo un sitio web, con una base de datos. De ésta manera, en el mundo real los contenedores normalmente no son iniciados manualmente usando `docker`, debido a que la tarea rápidamente se vuelve tediosa y compleja.

docker-compose es una herramienta básica de orquestación. Al igual que `docker` lee un archivo que tiene las instrucciones sobre que imagenes subir y cuales son los parametros de inicialización.

Ejercicios

4.chat

Indique: ¿Cuantos servicios tiene este software? ¿En que puerto queda el servicio una vez iniciado?
Fuente: `https://docs.rocket.chat/installation/docker-containers/docker-compose`

5.plex

Indique: ¿Cuantos servicios tiene este software? ¿En que puerto queda el servicio una vez iniciado?
Fuente: `https://docs.linuxserver.io/images/docker-plex#docker-compose-recommended`