

# Contenedores con Docker

## Explicación de práctica 4 - segunda parte

Sistemas Operativos

Facultad de Informática  
Universidad Nacional de La Plata

2018



## 1 Docker



## 1 Docker



¿Qué es Docker?



- Docker permite empaquetar y ejecutar una aplicación en containers aislados.
- Docker Engine está dividido en 3 componentes: el demonio **dockerd**, una **API REST** y la CLI **docker**.
- Usos posibles:
  - En desarrollo/Testing.
  - Escalado y despliegue (deployment).
  - Más servicios en un equipo sin VMs.



- Docker permite empaquetar y ejecutar una aplicación en containers aislados.
- Docker Engine está dividido en 3 componentes: el demonio **dockerd**, una **API REST** y la CLI **docker**.
- Usos posibles:
  - En desarrollo/Testing.
  - Escalado y despliegue (deployment).
  - Más servicios en un equipo sin VMs.



- Docker permite empaquetar y ejecutar una aplicación en containers aislados.
- Docker Engine está dividido en 3 componentes: el demonio **dockerd**, una **API REST** y la CLI **docker**.
- Usos posibles:
  - En desarrollo/Testing.
  - Escalado y despliegue (deployment).
  - Más servicios en un equipo sin VMs.



- Docker permite empaquetar y ejecutar una aplicación en containers aislados.
- Docker Engine está dividido en 3 componentes: el demonio **dockerd**, una **API REST** y la CLI **docker**.
- Usos posibles:
  - En desarrollo/Testing.
  - Escalado y despliegue (deployment).
  - Más servicios en un equipo sin VMs.





- Docker permite empaquetar y ejecutar una aplicación en containers aislados.
- Docker Engine está dividido en 3 componentes: el demonio **dockerd**, una **API REST** y la CLI **docker**.
- Usos posibles:
  - En desarrollo/Testing.
  - Escalado y despliegue (deployment).
  - Más servicios en un equipo sin VMs.



- Docker permite empaquetar y ejecutar una aplicación en containers aislados.
- Docker Engine está dividido en 3 componentes: el demonio **dockerd**, una **API REST** y la CLI **docker**.
- Usos posibles:
  - En desarrollo/Testing.
  - Escalado y despliegue (deployment).
  - Más servicios en un equipo sin VMs.



Docker es una herramienta que utiliza una serie de características del kernel para proveer containers:

**Namespaces:** Docker lo utiliza para proveer el espacio de trabajo aislado que denominamos container. Por cada container Docker crea un conjunto de espacios de nombres (entre ellos **pid**, **net**, **ipc** y **mnt**).

**Control groups:** Para, opcionalmente, limitar los recursos asignados a un contenedor.

**Union file systems:** Se utilizan como filesystem de los containers. Docker puede utilizar **overlay2**, **AUFS**, **btrfs**, **vfs** y **DeviceMapper**.



Docker es una herramienta que utiliza una serie de características del kernel para proveer containers:

**Namespaces:** Docker lo utiliza para proveer el espacio de trabajo aislado que denominamos container. Por cada container Docker crea un conjunto de espacios de nombres (entre ellos **pid**, **net**, **ipc** y **mnt**).

**Control groups:** Para, opcionalmente, limitar los recursos asignados a un contenedor.

**Union file systems:** Se utilizan como filesystem de los containers. Docker puede utilizar **overlay2**, **AUFS**, **btrfs**, **vfs** y **DeviceMapper**.



Docker es una herramienta que utiliza una serie de características del kernel para proveer containers:

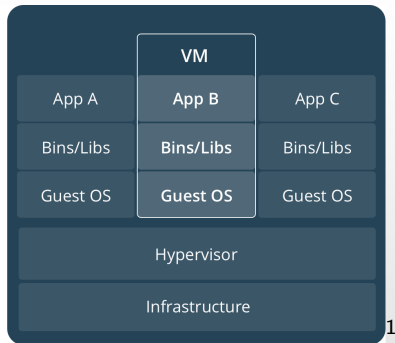
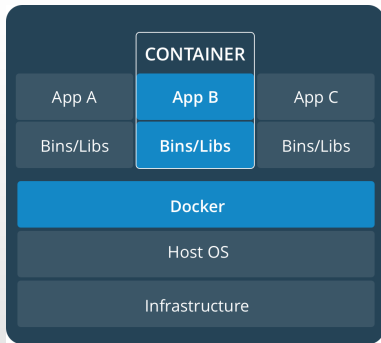
**Namespaces:** Docker lo utiliza para proveer el espacio de trabajo aislado que denominamos container. Por cada container Docker crea un conjunto de espacios de nombres (entre ellos **pid**, **net**, **ipc** y **mnt**).

**Control groups:** Para, opcionalmente, limitar los recursos asignados a un contenedor.

**Union file systems:** Se utilizan como filesystem de los containers. Docker puede utilizar **overlay2**, **AUFS**, **btrfs**, **vfs** y **DeviceMapper**.



# Containers VS VMs



<sup>1</sup><https://docs.docker.com/get-started/#containers-and-virtual-machines>



**imagen:** Paquete (sólo lectura) que contiene todo lo necesario para ejecutar una aplicación (librerías, configuraciones, etc...).

**registry:** Es un almacén de imágenes de Docker, por defecto docker utiliza *Docker Hub*.

**container:** Es una instancia de una imagen en ejecución.

**Dockerfile:** Archivo que define como construir una imagen.

Una imagen puede basarse en otras, por ejemplo:

`httpd → debian:jessie-backports → debian:jessie → scratch`  
donde *scratch* es un nombre especial que significa que se inicia desde una imagen vacía.



**imagen:** Paquete (sólo lectura) que contiene todo lo necesario para ejecutar una aplicación (librerías, configuraciones, etc...).

**registry:** Es un almacén de imágenes de Docker, por defecto docker utiliza *Docker Hub*.

**container:** Es una instancia de una imagen en ejecución.

**Dockerfile:** Archivo que define como construir una imagen.

Una imagen puede basarse en otras, por ejemplo:

`httpd → debian:jessie-backports → debian:jessie → scratch`

donde *scratch* es un nombre especial que significa que se inicia desde una imagen vacía.





**imagen:** Paquete (sólo lectura) que contiene todo lo necesario para ejecutar una aplicación (librerías, configuraciones, etc...).

**registry:** Es un almacén de imágenes de Docker, por defecto docker utiliza *Docker Hub*.

**container:** Es una instancia de una imagen en ejecución.

**Dockerfile:** Archivo que define como construir una imagen.

Una imagen puede basarse en otras, por ejemplo:

`httpd → debian:jessie-backports → debian:jessie → scratch`  
donde *scratch* es un nombre especial que significa que se inicia desde una imagen vacía.



**imagen:** Paquete (sólo lectura) que contiene todo lo necesario para ejecutar una aplicación (librerías, configuraciones, etc...).

**registry:** Es un almacén de imágenes de Docker, por defecto docker utiliza *Docker Hub*.

**container:** Es una instancia de una imagen en ejecución.

**Dockerfile:** Archivo que define como construir una imagen.

Una imagen puede basarse en otras, por ejemplo:

`httpd → debian:jessie-backports → debian:jessie → scratch`  
donde *scratch* es un nombre especial que significa que se inicia desde una imagen vacía.



**imagen:** Paquete (sólo lectura) que contiene todo lo necesario para ejecutar una aplicación (librerías, configuraciones, etc...).

**registry:** Es un almacén de imágenes de Docker, por defecto docker utiliza *Docker Hub*.

**container:** Es una instancia de una imagen en ejecución.

**Dockerfile:** Archivo que define como construir una imagen.

Una imagen puede basarse en otras, por ejemplo:

`httpd → debian:jessie-backports → debian:jessie → scratch`

donde *scratch* es un nombre especial que significa que se inicia desde una imagen vacía.



**imagen:** Paquete (sólo lectura) que contiene todo lo necesario para ejecutar una aplicación (librerías, configuraciones, etc...).

**registry:** Es un almacén de imágenes de Docker, por defecto docker utiliza *Docker Hub*.

**container:** Es una instancia de una imagen en ejecución.

**Dockerfile:** Archivo que define como construir una imagen.

Una imagen puede basarse en otras, por ejemplo:

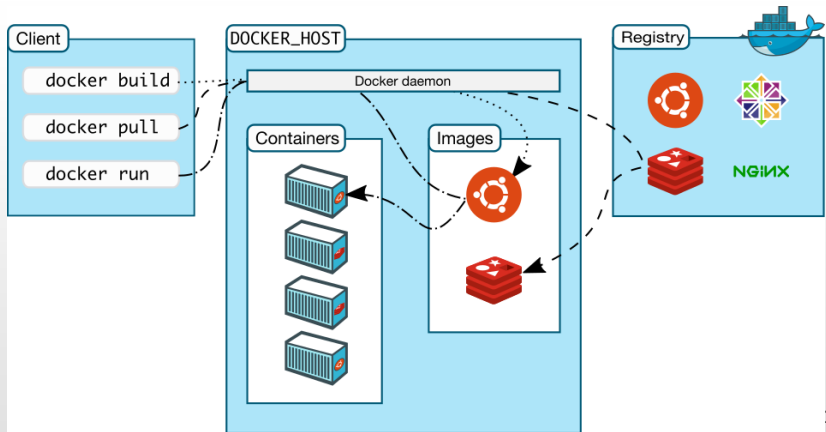
**httpd → debian:jessie-backports → debian:jessie → scratch**

donde *scratch* es un nombre especial que significa que se inicia desde una imagen vacía.



```
# Descargar imagen de Apache de DockerHUB
docker pull httpd
# Ejecutar imagen
docker run httpd
# Crear una imagen a partir de un Dockerfile
docker image build -t NOMBRE_TAG .
# Ejecutar la imagen creada
docker run NOMBRE_TAG
# Subir la imagen a DockerHUB
# antes hay que ejecutar `docker login`
docker push NOMBRE_TAG\
            USUARIO DOCKERHUB/REPOSITORIO
```





2

<sup>2</sup><https://docs.docker.com/engine/docker-overview/>  
#docker-architecture



```
# Información general y configuración
docker info

# Containers en ejecución
docker ps

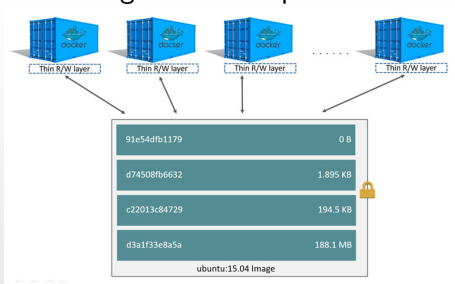
# Imágenes y containers
docker image ls
docker container ls -a

# Ejecutar el container de Ubuntu en modo
  interactivo (bash)
docker pull ubuntu &&\
    docker run -v ./dir_comp:/mnt -it ubuntu

# Crear una nueva imagen con los cambios del
  container
docker commit CONTAINER REPOSITORY:TAG
```



- Cada imagen está compuesta de una serie de capas.



- Las capas se montan una sobre otra.
- Solo la última es R/W (la capa del container)

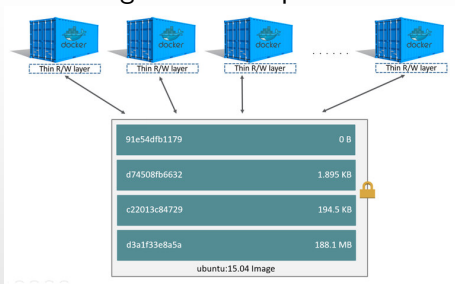
3

<sup>3</sup><https://docs.docker.com/storage/storagedriver/#container-and-layers>





- Cada imagen está compuesta de una serie de capas.



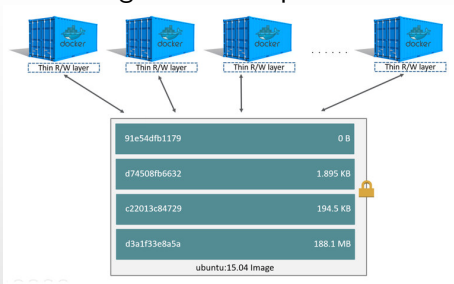
- Las capas se montan una sobre otra.
- Solo la última es R/W (la capa del container)

3

<sup>3</sup><https://docs.docker.com/storage/storagedriver/#container-and-layers>



- Cada imagen está compuesta de una serie de capas.



- Las capas se montan una sobre otra.
- Solo la última es R/W (la capa del container)

3

<sup>3</sup><https://docs.docker.com/storage/storagedriver/#container-and-layers>



- <https://docs.docker.com/engine/docker-overview/#docker-objects>
- <https://docs.docker.com/engine/reference/commandline/>
- <https://medium.com/@nagarwal/understanding-the-docker-internals-7ccb052ce9fe>
- <http://docker-saigon.github.io/post/Docker-Internals/>
- <https://www.safaribooksonline.com/library/view/using-docker/9781491915752/>
- <https://washraf.gitbooks.io/the-docker-ecosystem>



¿Preguntas?

