eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
GASTEIZKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE VITORIA-GASTEIZ

9 Aislamiento de Subsistemas y Contenedores.

Introducción a los Sistemas Operativos, 2019-2020

Pablo González Nalda

Depto. de Lenguajes y Sistemas Informáticos
EU de Ingeniería de Vitoria-Gasteiz,
UPV/EHU

25 de abril de 2020

Contenidos de la presentación

Contenidos

Automatización de la virtualización

Gestión automatizada de máquinas virtuales mediante `scripts` con `Vagrant`, que nos permite:

- Crear y destruir máquinas virtuales a través de un fichero de configuración `Vagrantfile`.
- Se usan máquinas genéricas preparadas como base.
- Al arrancar se ***provisionan*** con otro `script`, que instala, configura y ejecuta los programas adecuados para dar el servicio objetivo de la máquina.

```
1 vagrant up
  vagrant ssh
  vagrant destroy
4 uname -a
```

Se puede comprobar que son sistemas distintos con `uname` dentro y fuera de la máquina. La virtualización es a nivel hardware.

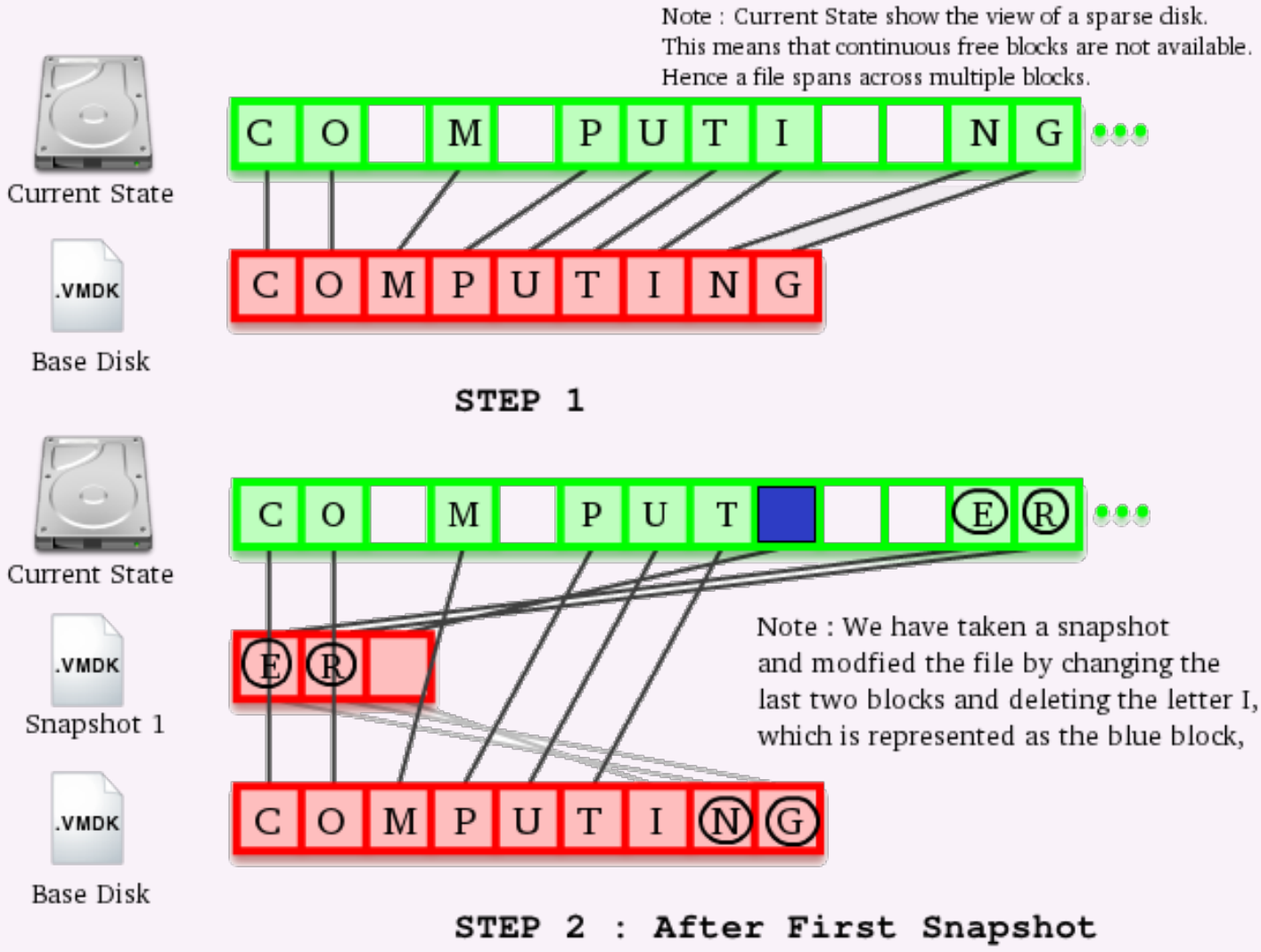
Control de una MV

Control de una MV:

- Exportar MV a un fichero (copia de seguridad)
- Importar MV desde fichero:
  `vboxmanage import <file>.ova`
- Control sin entorno gráfico:
  `VBoxHeadless --startvm "testvm"`
- `Creación` de `instantáneas` (*snapshots*): los discos de una MV "se congelan" y las modificaciones se realizan en otro fichero. Cuando se accede a un punto de un disco duro virtualizado se comprueba el fichero incremental, y si no, en el base.

Instantáneas (*snapshots*)

Cortesía IBM



Contenidos

Ventajas de la gestión con máquinas virtuales

Gestionar un sistema con máquinas virtuales nos permite:

- diferenciar los sistemas y los recursos necesarios para proporcionar un servicio: cada sistema es una simulación de ordenador, un SO independiente.
- por lo que un problema en una MV sólo va a afectar a un servicio
- la creación es repetible y automatizable.

Contenidos

Mecanismos de aislamiento

En las siguientes subsecciones se van a presentar mecanismos del kernel que permiten aislar los diferentes recursos de un proceso del exterior.

Con estos mecanismos se evita que un proceso acceda o comparta información o recursos del exterior: tendrá acceso a sólo una parte del árbol de directorios, se renombrará la información del kernel y se limitará CPU. Su árbol de directorios será modular.

A partir de ellos se construyen los sistemas de *contenedores* como `Docker`.

Mecanismos de aislamiento: `chroot`

`chroot` es un mecanismo que ejecuta un programa cambiando la raíz de su sistema de ficheros a un directorio, por lo que ese programa puede ser el que arranque un sistema completo.

Sólo compartirá con el anterior el sub-árbol de directorios y el kernel.

Se pueden montar los subdirectorios especiales como `/proc`

Ejemplo de uso de `chroot`

Como `root`

```
mkdir jaula
cd jaula/
cp /bin/busybox .
./busybox ls
./busybox uname  -a
ldd busybox
mkdir proc dev run var
mount --bind /proc proc # directorio /proc disponible en la
      jaula
mount --bind /dev dev
chroot /home/pablo/jaula ./busybox ash
# En otra terminal
sudo ls -ld /proc/18582/root
lrwxrwxrwx 1 root root 0 abr 20 19:56
/proc/18582/root -> /home/pablo/jaula
```

Más ejemplos en `http://www.cyberciti.biz/faq/unix-linux-chroot-command-examples-usage-syntax/`

Mecanismos de aislamiento: *Namespaces*

`https://docs.docker.com/engine/understanding-docker/`

Docker usa diferentes mecanismos proporcionados por el kernel y construidos sobre él. Usa los `namespaces` para crear entornos aislados en el sistema, los contenedores. Cada elemento del contenedor se ejecuta en su propio *espacio de nombres* y no tiene acceso al exterior.

Algunos de los espacios de nombres que usa Docker son:

- **pid** namespace: Process isolation (PID: Process ID).
- **net** namespace: Managing NETwork interfaces.
- **ipc** namespace: Managing access to IPC resources (IPC: InterProcess Communication).
- **mnt** namespace: Managing MouNT-points .
- **uts** namespace: Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

Mecanismos de aislamiento: *Control groups*

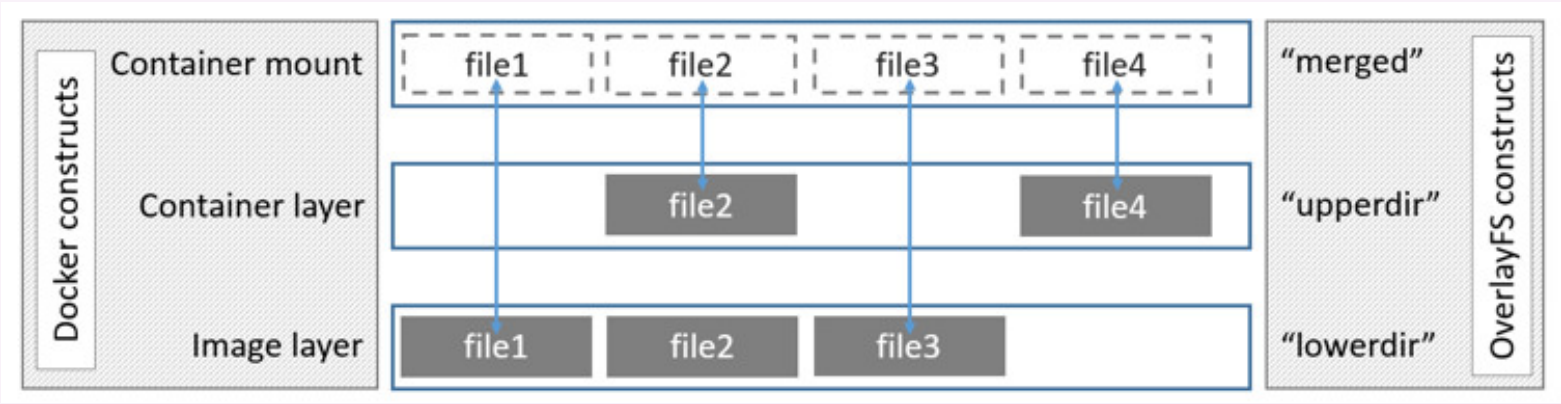
Los `cgroups` o *grupos de control* controlan la cantidad de recursos que consume el contenedor. De esta forma los contenedores comparten los recursos de hardware y tienen un límite, por ejemplo de memoria disponible. Controlan:

- Limitación de recursos.
- Control de prioridades.
- Contabilidad.
- Control de grupos de procesos (congelar y rearrancar).

Mecanismos de aislamiento *Union file systems*

Un sistema de ficheros *Union* es el que compone, une la información de varios sistemas de ficheros (las capas) para obtener uno. Esas capas son las fases de creación del sistema de ficheros del contenedor. Las capas pueden ser compartidas por varios contenedores (como el sistema operativo base).

```
1 ls -l /var/lib/docker/overlay/
```



https://docs.docker.com/engine/userguide/

Contenidos

¿Qué es Docker?

Docker es un mecanismo para aislar, confinar, gestionar y modularizar la ejecución de un servicio, sus procesos y recursos necesarios, como almacenamiento, redes y otros dispositivos.

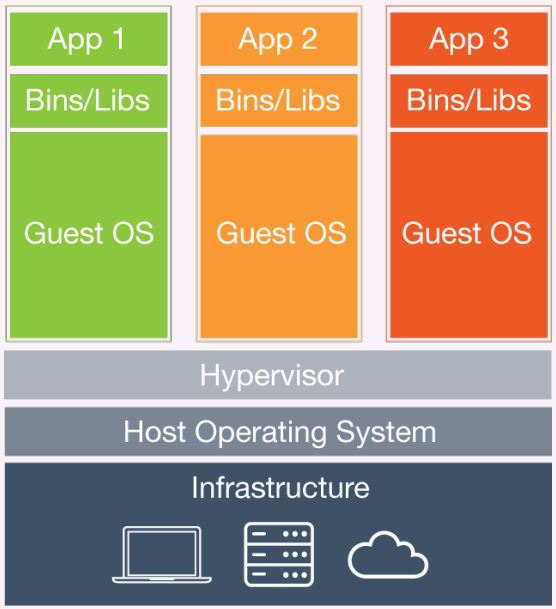


Figura: Virtualización a nivel de hardware

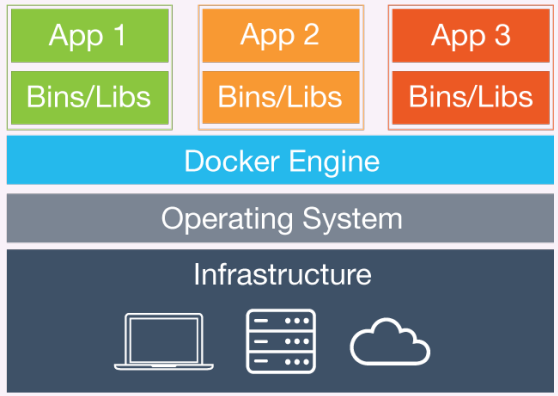
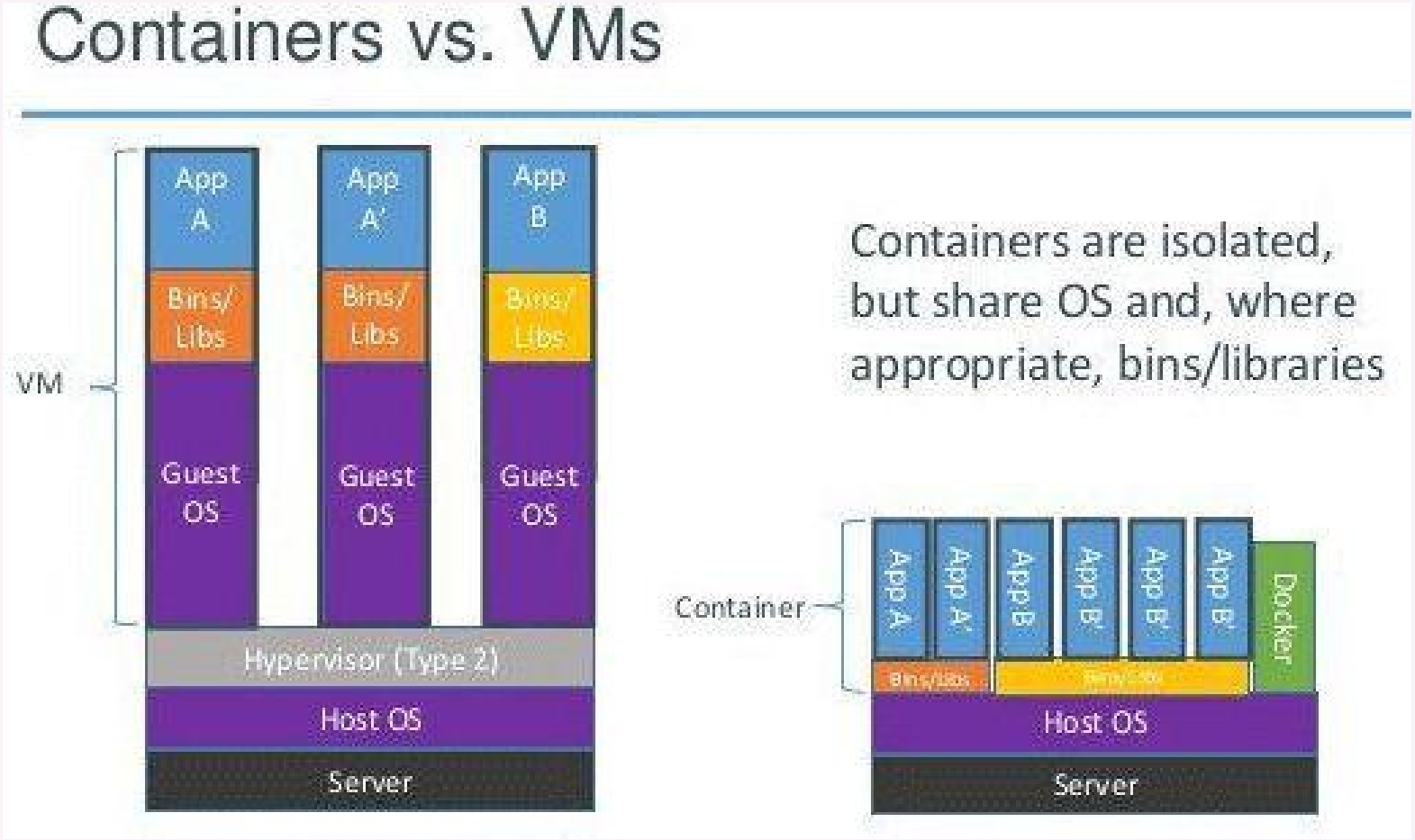


Figura: Contenedores

¿Qué es Docker? B

Gráfico alternativo.



¿Qué es Docker?

Por tanto, la funcionalidad de Docker no es diferente de la virtualización. Contenedores y máquinas virtuales modularizan la ejecución de servicios (un servidor web, por ejemplo).

La diferencia reside en el planteamiento y técnicas que virtualizan los recursos del servicio: cómo se aisla y controla el uso de CPU y de RAM, el Sistema de Ficheros y los interfaces de Red (NIC convertidos en SDN (Software Defined Network)).

Mientras en un contenedor sólo se ejecutan aislados unos cuantos procesos, en una máquina virtual se ejecuta un kérnel sobre hardware simulado.

Estructura de un contenedor

Un contenedor comparte el kernel con el resto de procesos del sistema, puesto que el contenedor aísla servicios formados por varios procesos que colaboran. Se puede distinguir

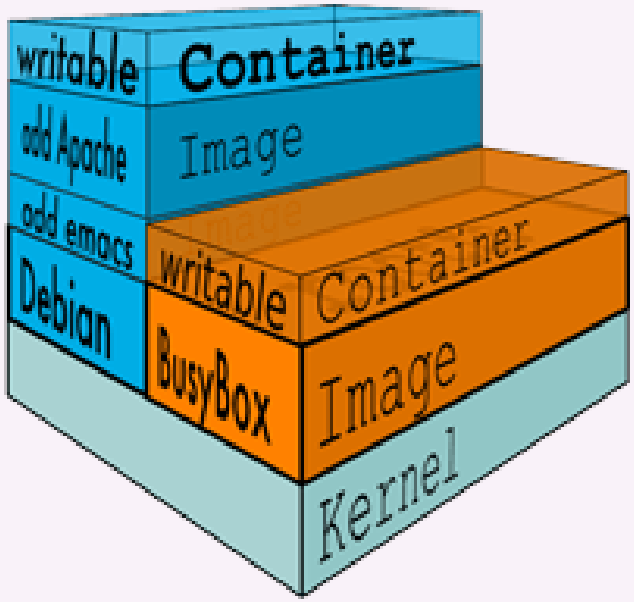
imagen el conjunto de capas de sólo lectura que forman el sistema de ficheros

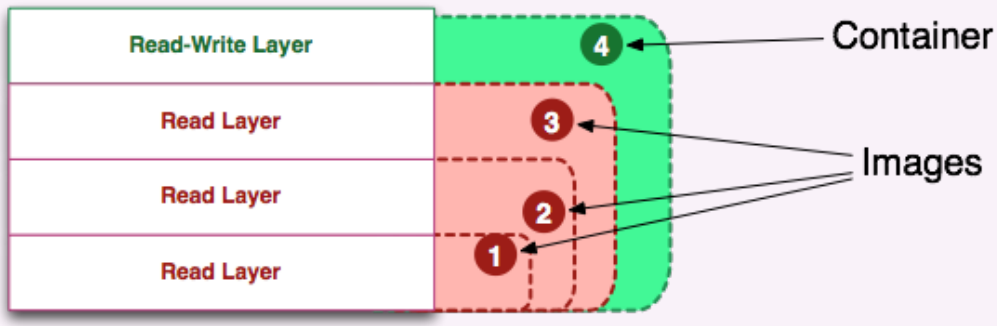
contenedor una imagen con una capa de lectura y escritura

Cuando hablamos de SO en el sistema de ficheros del contenedor, pensamos en el conjunto de ficheros y mecanismos básicos para arrancar en el contenedor los servicios necesarios para un programa estándar de ese sistema. Sin embargo, el contenedor no arranca ningún proceso.

Estructura de un contenedor

Cada capa (*image-layer*) es un sistema de ficheros de sólo lectura, y es el resultado de una fase de instalación de software. Las capas se unen con un sistema de ficheros de `UnionFS`. Cuando arranca el contenedor se añade una capa vacía de escritura. Las imágenes se componen de varias capas, se pueden comprender como una plantilla, y el contenedor es la instancia.





`http://merrigrove.blogspot.com.es/2015/10/visualizing-docker-containers-and.html`

Entornos de trabajo con Docker

EN un entorno de trabajo con Docker virtualizamos los recursos necesarios para ejecutar diferentes procesos de la aplicación, el *microservicio*.

Red tenemos una *red definida por software* (SDN),

FS un sistema de ficheros local accesible desde el contenedor (creado para la imagen y con una capa añadida para hacerlo R/W) y

RAM y CPU tanta memoria y CPU como tenga el sistema (si no se restringe).

En ese entorno se ejecuta una sola aplicación, un *microservicio*. Puede estar compuesta de varios ejecutables, cada uno forma un proceso.

Entornos de trabajo con Docker

Aunque tenga disponibles todos los ficheros de una distribución GNU/Linux como la imagen de Ubuntu, no se ejecuta ningún software de gestión del Sistema Operativo (demonios, entorno gráfico…) aparte del que señalemos al arrancar el contenedor.

Dockerfile mínimo

Dockerfile mínimo para crear una imagen:

```
# Crear la imagen con
# docker build --tag ejemplo .
#
# y arrancar para hacer un ls -l dentro del contenedor con
# docker run --rm ejemplo ./busybox ls -l

# y para crear una línea de comandos interna (salir con
    exit)
# docker run -it --rm --name "EjemploDocker" -h "ED"
    ejemplo ./busybox ash
# y dentro por ejemplo: uname -a

FROM scratch
COPY busybox /
```

El fichero `busybox` se copia en una capa de la imagen, que se crean en `/var/lib/docker`

docker pull ubuntu

Cuando ya tenemos instaldo Docker, podemos bajar una imagen de Ubuntu. Observa el tamaño don `docker images`.

```
$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
5bed26d33875: Pull complete
f11b29a9c730: Pull complete
930bda195c84: Pull complete
78bf9a5ad49e: Pull complete
Digest: sha256:
    bec5a2727be7fff3d308193cfde3491f8fba1a2ba392b7546b43a05185

Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

docker run

Para arrancar un microservicio (en este caso sólo es un comando de ubuntu para comprobar que se usa el mismo kérnel y borra con `--rm` el contenedor):

```
docker pull ubuntu
docker images # imágenes, tenemos el ubuntu
docker run --rm ubuntu uname -a # ejecuta el comando
docker run --rm -it ubuntu bash # interactivo, línea de
    comandos
docker ps -a # contenedores en ejecución y parados con el -
    a, se han borrado
docker run ubuntu uname -a # no borra el contenedor
docker ps -a # contenedores en ejecución y parados con el -
    a
docker rm practical_lehmann # borra el contenedor usando el
     nombre de NAMES
docker images # sigue estando la imagen
docker rmi ubuntu # borra la imagen
```

`docker inspect ubuntu`

Información sobre el contenedor en formato JSON con `docker inspect ubuntu`

Ubicación de las capas en el SO anfitrión, línea de comandos por defecto, PATH, versión y nombre (*tag*) de la imagen…

Red en Docker

Las redes en Docker se crean y destruyen por software. Los contenedores se pueden conectar a redes al crearlos y posteriormente.

```
$ docker network ls
NETWORK ID          NAME                DRIVER
7fca4eb8c647        bridge              bridge
9f904ee27bf5        none                null
cf03ee007fb4        host                host

$ docker network create --driver bridge red
$ docker run -it -network=red ubuntu:latest /bin/bash
```

Para más información, consulta la documentación de Docker.

Puntos de montaje y volúmenes en Docker

Los volúmenes son etiquetas que denominan directorios accesibles desde los contenedores:

```
# Punto de montaje de mi directorio en un directorio del contenedor
$ docker run -it -v /home/pablo/catkin_ws/:/catkin_ws ros

$ docker volume ls
DRIVER              VOLUME NAME
$ docker volume create my-vol
$ docker volume inspect my-vol
[
    {
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/my-vol/_data",
        "Name": "my-vol",
        "Options": {},
        "Scope": "local"
    }
]
$ docker run -d \
    --name devtest \
    -v myvol:/app \
    nginx:latest
```

Contenidos

Ventajas de Docker frente a máquinas virtuales

Gestionar un sistema con Docker tiene ciertas ventajas frente a usar máquinas virtuales:

- Usa menos recursos, no necesita todos los servicios de un SO estándar por lo que arranca mucho más rápidamente y con menos memoria y espacio en disco.
- No emula hardware virtual sino que usa el mismo kérnel que el sistema "anfitrión", así que es mucho más rápido en ejecución.
- Los recursos están virtualizados (como las redes definidas por software) pero se pueden compartir fácilmente entre contenedor y SO base.

Sin embargo, los sistemas deben repensarse para reimplementarse como un sistema de contenedores.

Ejemplos de `Dockerfiles`

Ejemplos de `Dockerfiles`:

`https://github.com/PabloGN/Docker-raspbian-ros-indigo`

Contenedores para Raspberry Pi y forma de uso:

`https://hub.docker.com/r/pablogn/`

Contenidos

¿Más preguntas?

¿Más preguntas?

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
GASTEIZKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE VITORIA-GASTEIZ

9 Aislamiento de Subsistemas y Contenedores.

Introducción a los Sistemas Operativos, 2019-2020

Pablo González Nalda

Depto. de Lenguajes y Sistemas Informáticos
EU de Ingeniería de Vitoria-Gasteiz,
UPV/EHU

25 de abril de 2020