INTRODUCCIÓN A TECNOLOGÍAS DE CONTENEDORES

Distribuir aplicaciones más fácilmente

José Manuel Redondo López Proyecto "*L-62 Princesa de Asturias*" v1.2




Fuente: Bing Chat AI

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

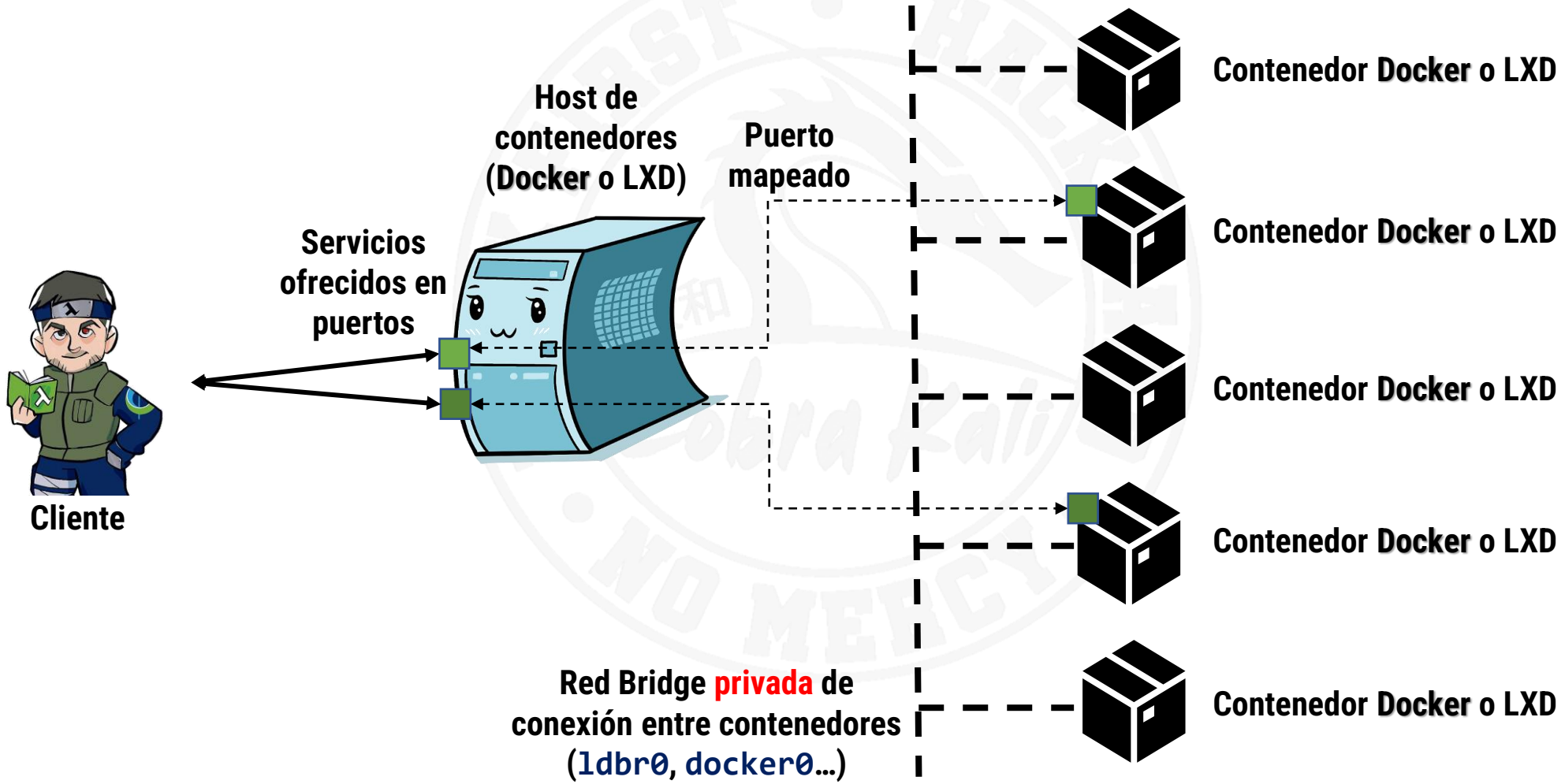
ÍNDICE

- **Contenedores de SO con LXD**
  - Componentes de LXD
  - Instancias de LXD y su configuración
  - Perfiles, imágenes, volúmenes y redes LXD
  - Gestión de contenedores LXD
- **Contenedores de aplicaciones con Docker**
  - Funcionamiento de Docker
  - Redes en Docker
  - Dockerfiles y optimización de imágenes
  - Volúmenes de Docker
- **Infraestructuras con Docker Compose**
- **Vagrant y Docker**
- **Últimas Consideraciones**

¿CÓMO DEBO VER A LOS CONTENEDORES?

José Manuel
Redondo López

- Ambas tecnologías de contenedores tienen en común la forma en la que se deben ver desde el exterior



< Ir al Índice

Contenedores de SO con LXD

Tecnología de contenedores de sistemas operativos

Fuente: Bing Chat AI

Componentes >

Instancias >

Elementos >

Gestión >

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- **Este apartado te explica todo lo necesario para que sepas usar la tecnología de contenedores de SO LXD, lo que incluye**
  - **Qué son** estos contenedores y sus características
  - Sus **componentes principales**
  - Que son las **instancias LXD** y su configuración
  - Los elementos principales de LXD: **Perfiles**, **imágenes**, **volúmenes** y **redes**
  - La **gestión de contenedores** de LXD

Linux Containers (LXC)

https://linuxcontainers.org/lxd/introduction/

José Manuel
Redondo López

- **Es una tecnología de contenedores de SO y forma de virtualización**
  - Permite ejecutar muchos sistemas **Linux** (contenedores) en un mismo host
  - Compartiendo **hardware** y kernel entre todos los contenedores
- **Ubuntu LXD es el hipervisor de Canonical (Ubuntu) construido encima de LXC**
  - Permite manejar contenedores de manera más sencilla y próxima a las MVs
- **El objetivo de estos contenedores es virtualizar un Linux completo**
  - De una forma muchos más económica en recursos que una máquina virtual tradicional
- **Dando la posibilidad de tener un ecosistema de contenedores repartido en miles de máquinas o en la nube**

Características de LXD

José Manuel
Redondo López

- **Seguridad**
  - Los contenedores se ejecutan en espacios de nombres aislados y solo pueden acceder a recursos predefinidos
- **Escalabilidad**
  - Se pueden gestionar contenedores individuales en un PC, o muchos repartidos en distintos entornos
- **API**
  - Tiene una API de programación y un cliente en línea de comandos
- **Solo Linux**
  - Funciona con todas las imágenes de Linux (**NO** Windows)
- **Control de recursos**
  - Admite límites de CPU, RAM, uso de la red, almacenamiento y recursos del kernel
- **Acceso al hardware**
  - Si la configuración lo permite, los contenedores pueden acceder a dispositivos USB, GPU almacenamiento
- **Gestión de red**
  - La configuración permite crear puentes y túneles de red
- **Gestión de almacenamiento**
  - LXD admite varios backends, grupos y volúmenes de almacenamiento

Componentes de LXD

Elementos que hacen posible esta tecnología

Componentes de LXD

José Manuel
Redondo López

- Las partes principales de LXD son **daemon**, API REST y herramienta la línea de comandos
- **Imágenes**: Son distribuciones de Linux a partir de las cuales se crean contenedores
  - LXD tiene estos **servidores de imágenes** predefinidos
    - `ubuntu`: imágenes estables de Ubuntu
    - `ubuntu-daily`: compilaciones diarias de imágenes de Ubuntu
    - `images`: imágenes de otras distribuciones de Linux y es administrado por la comunidad
  - Las imágenes descargadas por el daemon se guardan automáticamente en una caché
    - Si hay nuevas versiones, LXD **descargará las actualizaciones automáticamente** por defecto
  - Se pueden construir imágenes a partir de contenedores personalizados, como en Vagrant

ALIAS	FINGERPRINT	PUBLIC	DESCRIPTION	ARCHITECTURE	TYPE	SIZE
almalinux/8 (3 more)	7a0962afb63c	yes	Almalinux 8 amd64 (20210729_23:08)	x86_64	VIRTUAL-MACHINE	568.56MB
almalinux/8 (3 more)	fb8abab81937	yes	Almalinux 8 amd64 (20210729_23:08)	x86_64	CONTAINER	126.17MB
almalinux/8/arm64 (1 more)	ff89a5251253	yes	Almalinux 8 arm64 (20210729_23:08)	aarch64	CONTAINER	122.20MB
almalinux/8/cloud (1 more)	565b12f4ee1d	yes	Almalinux 8 amd64 (20210729_23:08)	x86_64	VIRTUAL-MACHINE	627.25MB
almalinux/8/cloud (1 more)	e22086903367	yes	Almalinux 8 amd64 (20210729_23:08)	x86_64	CONTAINER	145.23MB
almalinux/8/cloud/arm64	4d078f2fed95	yes	Almalinux 8 arm64 (20210729_23:08)	aarch64	CONTAINER	141.19MB
alpine/3.11 (3 more)	8c2adf7c5269	yes	Alpine 3.11 amd64 (20210730_13:00)	x86_64	CONTAINER	2.41MB
alpine/3.11 (3 more)	6357414d334a	yes	Alpine 3.11 amd64 (20210730_13:00)	x86_64	VIRTUAL-MACHINE	83.88MB
alpine/3.11/arm64 (1 more)	e46163ba235a	yes	Alpine 3.11 arm64 (20210730_13:00)	aarch64	CONTAINER	2.22MB
alpine/3.11/arm64 (1 more)	faddad6e2ba8	yes	Alpine 3.11 arm64 (20210730_13:00)	aarch64	VIRTUAL-MACHINE	74.00MB
alpine/3.11/armhf (1 more)	862fca4502b2	yes	Alpine 3.11 armhf (20210730_13:00)	armv7l	CONTAINER	2.14MB
alpine/3.11/i386 (1 more)	b223233546d7	yes	Alpine 3.11 i386 (20210730_13:00)	i686	CONTAINER	2.42MB
alpine/3.11/ppc64el (1 more)	e4f8acabfc87	yes	Alpine 3.11 ppc64el (20210730_13:00)	ppc64le	CONTAINER	2.31MB
alpine/3.11/s390x (1 more)	ad0454e0c3e3	yes	Alpine 3.11 s390x (20210730_13:00)	s390x	CONTAINER	2.10MB
alpine/3.12 (3 more)	6d85cab678ad	yes	Alpine 3.12 amd64 (20210730_13:00)	x86_64	CONTAINER	2.40MB
alpine/3.12 (3 more)	05535aa702bf	yes	Alpine 3.12 amd64 (20210730_13:00)	x86_64	VIRTUAL-MACHINE	96.38MB
alpine/3.12/arm64 (1 more)	60d3d68b03bb	yes	Alpine 3.12 arm64 (20210730_13:00)	aarch64	VIRTUAL-MACHINE	84.69MB
alpine/3.12/arm64 (1 more)	d28edd22f947	yes	Alpine 3.12 arm64 (20210730_13:00)	aarch64	CONTAINER	2.21MB
alpine/3.12/armhf (1 more)	babebdfcc092	yes	Alpine 3.12 armhf (20210730_13:00)	armv7l	CONTAINER	2.13MB
alpine/3.12/i386 (1 more)	5cd13feb9fcf	yes	Alpine 3.12 i386 (20210730_13:00)	i686	CONTAINER	2.41MB
alpine/3.12/ppc64el (1 more)	77645dd9b7bc	yes	Alpine 3.12 ppc64el (20210730_13:00)	ppc64le	CONTAINER	2.30MB
alpine/3.12/s390x (1 more)	6165c0624b09	yes	Alpine 3.12 s390x (20210730_13:00)	s390x	CONTAINER	2.08MB

Componentes de LXD

José Manuel
Redondo López

- **Contenedores**: principal abstracción de LXD que incluye
  - Un **sistema de archivos** de Linux
  - **Ajustes de configuración** como límites de recursos, variables de entorno, opciones de seguridad, etc.
  - Dispositivos de **red** y **almacenamiento**
  - **Perfil de configuración**, del cual el contenedor hereda sus ajustes
  - **Propiedades generales** como la arquitectura del contenedor, el nombre y si es de corta o larga duración
  - **Estado actual**: rendimiento de la red, uso de memoria, etc.
- **Snapshots**: Son el estado inmutable de un contenedor guardado
  - De manera que se pueda restaurar el contenedor al estado que contienen

```
operario@server:~$ lxc launch images:alpine/3.12/amd64 alpine-c1
Creating alpine-c1
Starting alpine-c1
```

```
operario@server:~$ lxc exec alpine-c1 sh
~ #
```

➢ Ejercicios **N1LXD_RUNCONT**, **N1LXD_INTERACT**, **N1LXD_STARTSTOP**

Componentes de LXD

José Manuel
Redondo López

- **Perfiles**: Agrupa opciones de configuración de contenedores
  - Se puede aplicar un perfil a varios contenedores (reutilizar)
  - Y varios perfiles sucesivamente a un mismo contenedor, sobrescribiendo valores ya definidos
    - Configuración por capas en ficheros especializados
  - LXD se distribuye con dos perfiles
    - `default` se aplica automáticamente a los contenedores si no se establece ninguno
      - Contiene ajustes de configuración básicos, como el dispositivo de red `eth0` del contenedor.
    - `docker` sirve para configurar un contenedor LXD que aloje un contenedor Docker
      - Hace que el contenedor LXD que cargue los módulos del kernel necesarios y lo configura adecuadamente
      - También activa el anidamiento de contenedores
- **Endpoints Remotos**: LXD es un daemon que funciona en red
  - El cliente en línea de comandos puede comunicarse con varios servidores LXD y de imágenes remotos
  - Se pueden configurar varios servidores como remotos y **mover o copiar contenedores existentes entre ellos**

Instancias de LXD y su configuración

Poniendo en marcha contenedores LXD

Instancias de LXD

José Manuel
Redondo López

- **LXD soporta tanto contenedores como máquinas virtuales**
  - En el segundo caso el proveedor de virtualización es QEMU
  - En ambos casos, al arrancar una instancia por defecto se le asigna vía DHCP una IP en la interfaz `eth0` del contenedor
  - Corresponde a una red **bridge** creada en el host al instalar LXD (`lxdbr0`)
    - Con ella podemos comunicarnos con ella desde el host y redirigir peticiones desde clientes externos
- **Los contenedores soportan las características de LXD vistas**
  - Las máquinas virtuales aún carecen de soporte para algunas de ellas:
    - https://linuxcontainers.org/lxd/docs/master/instances/#key-value-configuration
  - La idea es ir igualando ambas
- **Dado que las MVs ya las hemos cubierto con Vagrant, en esta parte manejaremos contenedores solamente**
  - Pero está bien saber que puedes unificar su manejo con el interfaz de LXD

CONFIGURACIÓN DE INSTANCIAS

José Manuel
Redondo López

- La configuración de instancias está dividida en categorías
  - **Propiedades de la instancia**: Datos acerca que pueden asignarse cuando se crean (nombre, arquitectura...)
    - **Listado completo**: https://linuxcontainers.org/lxd/docs/master/instances/#properties
  - **Opciones de la instancia**: Opciones de configuración que se aplican directamente
    - Opciones de arranque, configuración de seguridad, límites de hardware, módulos de kernel, instantáneas o teclas configuradas...
    - **Listado completo**: https://linuxcontainers.org/lxd/docs/master/instances/#keyvalue-configuration
  - **Dispositivos de instancia**: Dispositivos que se añaden a cada instancia
    - Como interfaces de red, puntos de montaje, dispositivos USB, GPUs...
    - **Listado completo**: https://linuxcontainers.org/lxd/docs/master/instances/#device-types
    - Cada dispositivo puede tener sus propias opciones dependiendo de su tipo
- Se pueden configurar en el arranque o durante su ejecución
- También se pueden crear **perfiles** con conjuntos de configuraciones
  - Para aplicarlos todos a la vez cuando una instancia se lance

Perfiles, imágenes, volúmenes y redes LXD

Los conceptos principales para trabajar con contenedores LXD

PERFILES

José Manuel
Redondo López

- Guardan opciones de configuración
- Pueden contener opciones de instancia y sus dispositivos (y opciones de estos)
- Se pueden aplicar muchos a una instancia
  - **Se aplican en orden**: el último perfil a aplicar tiene más prioridad en caso de coincidencia de opciones
  - La configuración específica de una instancia **siempre sobrescribe** a la que venga de un perfil
  - Si alguno de los contenidos no es aplicable para el tipo de instancia, se ignora y no da un error
- En caso de que no se especifique ninguno, se aplica el perfil `default`
  - Este perfil define una interfaz de red y un disco `root`, y no puede eliminarse ni renombrarse

```
config:
  environment.DISPLAY: :0
description: Aplicaciones X11 en el contenedor
devices:
  x11-socket:
    path: /tmp/.X11-unix/
    source: /tmp/.X11-unix/
    type: disk
name: aplicaciones_graficas_x11
```

IMÁGENES

José Manuel
Redondo López

- **Son la base de los contenedores**
  - Contienen un SO básico y otra información necesaria para que LXD funcione
  - Normalmente usaremos los **servidores de imágenes remotos preconstruidos** que ya vimos
  - Para listar imágenes disponibles en un servidor: `lxc image list images`
- **Si vemos una mención a "`cloud`" bajo la columna `ALIAS` significa que esas imágenes soportan `cloud-init`: https://linuxcontainers.org/lxd/advanced-guide/#cloud-init**
  - Significa que **las imágenes tienen un software de automatización** que permite personalizarlas
  - Instalando paquetes, aplicando y modificando configuraciones, añadiendo usuarios...
    - https://cloudinit.readthedocs.io/en/latest/
  - Las `cloud-init` **son más recomendables**: Modifican su tamaño dinámicamente y se testean más
- **Más información:**
  - **Tutorial de LXD**: https://www.cyberciti.biz/faq/install-lxd-pure-container-hypervisor-on-ubuntu-18-04-lts/
  - **Elementos más avanzados de LXD**: https://linuxcontainers.org/lxd/advanced-guide/

Volúmenes LXD

José Manuel
Redondo López

- Cuando se crea una instancia de LXD se le asigna un volumen de almacenamiento automáticamente (disco raíz)
- Pero se pueden añadir volúmenes adicionales a cualquier instancia
  - Son además independientes, por lo que pueden moverse entre instancias, compartirse (si es de tipo `filesystem`, tipo por defecto) y no se borran si la instancia que los tiene se elimina
- Para crear un volumen hay que usar el siguiente comando
  - `lxc storage volume create <pool_name> <volume_name> [configuration_options...]`
  - Las opciones de configuración dependen del driver usado para crear el volumen
- Una vez creado, se puede añadir a una instancia con
  - `lxc storage volume attach <pool_name> <filesystem_volume_name> <instance_name> <location>`
- También se pueden introducir o sacar ficheros en un contenedor
  - Y usarlos para **provisionarlos** (shell scripts)
- Más información: https://linuxcontainers.org/lxd/docs/master/howto/storage_add_volume/

➢ Ejercicios **N3LXD_FICHEROSEXT**, **N3LXD_CREARVOL**, **N3LXD_PROVISION**

Redes LXD

José Manuel
Redondo López

- **Cuando se arranca una instancia de una imagen LXD, el perfil por defecto asigna una red de tipo bridge (normalmente `LXDBR0`)**
  - **Permite que tenga conexión a Internet**
  - No obstante, es posible crear otros tipos de red, como los especificados aquí:
    - https://linuxcontainers.org/lxd/docs/master/howto/network_create/
- **Para crear una red tenemos que usar este comando**
  - `lxc network create <name> --type=<network_type> [configuration_options...]`
  - Existen varios tipos de red que se pueden consultar en el enlace anterior
  - Normalmente usaremos uno tipo `bridge` o `macvlan`
- **Podemos configurar otras características de la red con `lxc network set`**
  - Las características configurables dependerán del tipo de red creada
- **Tras crear un interfaz de red se le puede añadir a una instancia con**
  - `lxc config device add <instance_name> <device_name> nic nictype=<nic_type> ...`
  - O bien añadirse gracias a un perfil

Redes LXD

José Manuel
Redondo López

- La creación de una **red local privada** entre contenedores con LXD requiere lo siguiente
  - **Asignar una interfaz de red en el host** que haga de interfaz para la red entre los contenedores
    - Es fácil de hacer con un proveedor de virtualización como VirtualBox
    - Al fin y al cabo, una máquina virtual puede tener hasta 4, y podemos añadir una solo para eso
  - **Crear una interfaz de red tipo** `macvlan` vinculada a dicha interfaz de red del host
  - **Fijar el tipo de direccionamiento** (DHCP, estático) y si se usa IPv4 o no
  - Repetir esto en todos los contenedores que se quieran vincular en esa red local
  - Si tenemos que unir contenedores en varios grupos de manera aislada entre ellos, tendríamos que crear más interfaces en el host para hacer grupos de los mismos
    - Varias redes entre contenedores aisladas entre ellas son una forma de conseguir la SNI del primer tema ☺
- Podemos obtener información avanzada sobre la creación de redes en
  - https://linuxcontainers.org/lxd/docs/master/howto/network_create/

Redes LXD y perfiles

José Manuel
Redondo López

- Los perfiles de LXD se pueden usar para configurar distintos aspectos de las redes de LXD

```
config: {}
description: Este perfil añade un interfaz ethernet al contenedor de tipo macvlan
devices:
  <nombre del interfaz en el contenedor. Ej.: eth1>:
    nictype: macvlan
    parent: <Nombre del interfaz en el host de LXD. Ej.: enp3s8>
    type: nic
name: macvlan
```

```
config:
  environment.http_proxy: <dirección del proxy HTTP>
  environment.https_proxy: <dirección del proxy HTTPS>
  environment.ftp_proxy: <dirección del proxy FTP>
description: Variables de entorno para establecer un proxy en el contenedor
devices: {}
name: proxy
```

Gestión de contenedores LXD

Uso operativo de contenedores LXD

Gestión de contenedores LXD

José Manuel
Redondo López

- **Todos los contenedores LXD que creemos en una misma máquina comparten los recursos de la misma**
  - CPU, RAM, el disco creado para guardar su almacenamiento...
  - Si uno de ellos usa demasiados recursos, podría afectar a los demás o al rendimiento general del sistema
  - Es por eso que LXD permite **establecer límites de uso de recursos** que previenen estos casos
    - https://www.maketecheasier.com/limit-lxd-containers-resources/
- **También podemos copiar contenedores existentes y publicar nuevos a partir de otros que hayamos personalizado previamente**
  - https://www.oreilly.com/library/view/ubuntu-server-cookbook/9781785883064/ch08s05.html
  - https://discuss.linuxcontainers.org/t/lxd-publish-documentation/5176

Gestión de contenedores LXD: Proyectos

José Manuel
Redondo López

- **En despliegues grandes puede llegar un momento en el que el nº de contenedores que tenemos puede ser enorme**
- **LXD permite una mayor organización creando proyectos**
  - Permiten agrupar los contenedores en entidades independientes (los proyectos)
  - Podemos cambiar entre ellos a voluntad y sólo veremos los contenedores asociados a dichos proyectos
  - Así podremos agrupar contenedores por funcionalidad, capa de la aplicación, etc.
- **Son muy sencillos de crear y manejar**
  - https://ubuntu.com/tutorials/introduction-to-lxd-projects#1-overview

Minimizando imágenes LXD

José Manuel
Redondo López

- Nuevamente es importante hacer que las imágenes de LXD sean lo más pequeñas posibles por temas de seguridad, portabilidad y eficiencia
- Tenemos dos opciones
  - Usar una imagen **`ubuntu-minimal`**, que está disponible si añadimos el servidor de imágenes indicado en estas instrucciones: https://wiki.ubuntu.com/Minimal
  - Usar una imagen de Alpine Linux, que está listada junto con las imágenes oficiales disponibles
- La siguiente imagen muestra la diferencia entre ambas

ALIAS	FINGERPRINT	PUBLIC	DESCRIPTION	ARCH	SIZE	UPLOAD DATE
pruebasclon	4e988b9c89ef	no		x86_64	334.61MB	Jul 20, 2022 at 1:43pm (UTC)
	5f499a82282b	no	Ubuntu bionic amd64 (20220801_07:42)	x86_64	106.49MB	Aug 4, 2022 at 6:48am (UTC)
	64cdedcf7f37	no	Alpine 3.16 amd64 (20220801_13:00)	x86_64	2.49MB	Aug 4, 2022 at 6:54am (UTC)
	86d36162a1f3	no		x86_64	334.54MB	Jul 20, 2022 at 1:40pm (UTC)
	b46db39f29fa	no	ubuntu 18.04 LTS amd64 (release) (20220712)	x86_64	208.48MB	Aug 4, 2022 at 6:48am (UTC)
	cf3cf9b49582	no	ubuntu 18.04 LTS amd64 (minimal release) (20220712.1)	x86_64	117.21MB	Aug 4, 2022 at 6:52am (UTC)

¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

?

- *¿Entiendes qué son los contenedores LXD y sus principales características?*
- **Componentes**
  - *¿Sabes qué es una imagen LXD?*
  - *¿Entiendes el concepto de contenedor LXD y su relación con las imágenes?*
  - *¿Comprendes para que sirve un snapshot?*
  - *¿Entiendes el concepto de perfil LXD y para qué sirve?*
- **Instancias y configuración**
  - *¿Entiendes que con LXD puedes manejar de la misma forma contenedores LXD y MV con QEMU?*
  - *¿Entiendes que cualquier contenedor LXD por defecto tiene una red que le permite comunicarse con internet con IP asignada por DHCP?*
  - *¿Comprendes que una instancia LXD tiene propiedades, opciones y se le pueden añadir dispositivos?*

¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

?

- **Perfiles, imágenes, volúmenes y redes LXD**
  - *¿Entiendes que son y para qué sirven los perfiles?*
  - *¿Comprendes que existe un perfil por defecto y que puedes aplicar todos los perfiles que quieras a una instancia?*
  - *¿Eres consciente de todo lo que contiene una imagen y lo que es una imagen cloud-init?*
  - *¿Sabes para qué sirve un volumen LXD?*
  - *¿Entiendes el modo de funcionamiento de las redes LXD, como se crean y cómo se añaden a una instancia?*
- **Gestión de contenedores LXD**
  - *¿Comprendes que todos los contenedores LXD comparten los recursos de la máquina y que por ello tiene sentido limitar el uso de dichos recursos?*
  - *¿Puedes explicar para que sirve un proyecto LXD?*
  - *¿Sabes elegir imágenes LXD que tengan un tamaño mínimo?*

< Ir al Índice

Fuente: Bing Chat AI

Funcionamiento >

Redes >

Dockerfiles >

Volúmenes >

Contenedores de Aplicaciones con Docker

Una idea general del sistema de referencia para contenedores de aplicaciones

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- **Este apartado te explica todo lo necesario para que sepas usar la tecnología de contenedores de aplicaciones Docker, lo que incluye**
  - Entender su **arquitectura y orígenes**
  - Aprender sus **conceptos principales** de funcionamiento: **Imágenes**, **registros**, **contenedores**
  - Saber **cómo se ejecutan** y los modos de ejecución que tienen
  - Entender **cómo funcionan las redes** de Docker y las distintas opciones de visibilidad de los contenedores
  - Saber **qué es una Dockerfile** y como crearlas para construir imágenes que se adapten a nuestras necesidades
  - Entender el concepto y utilidad de los **volúmenes de Docker**

Open Container Initiative (OCI Containers)

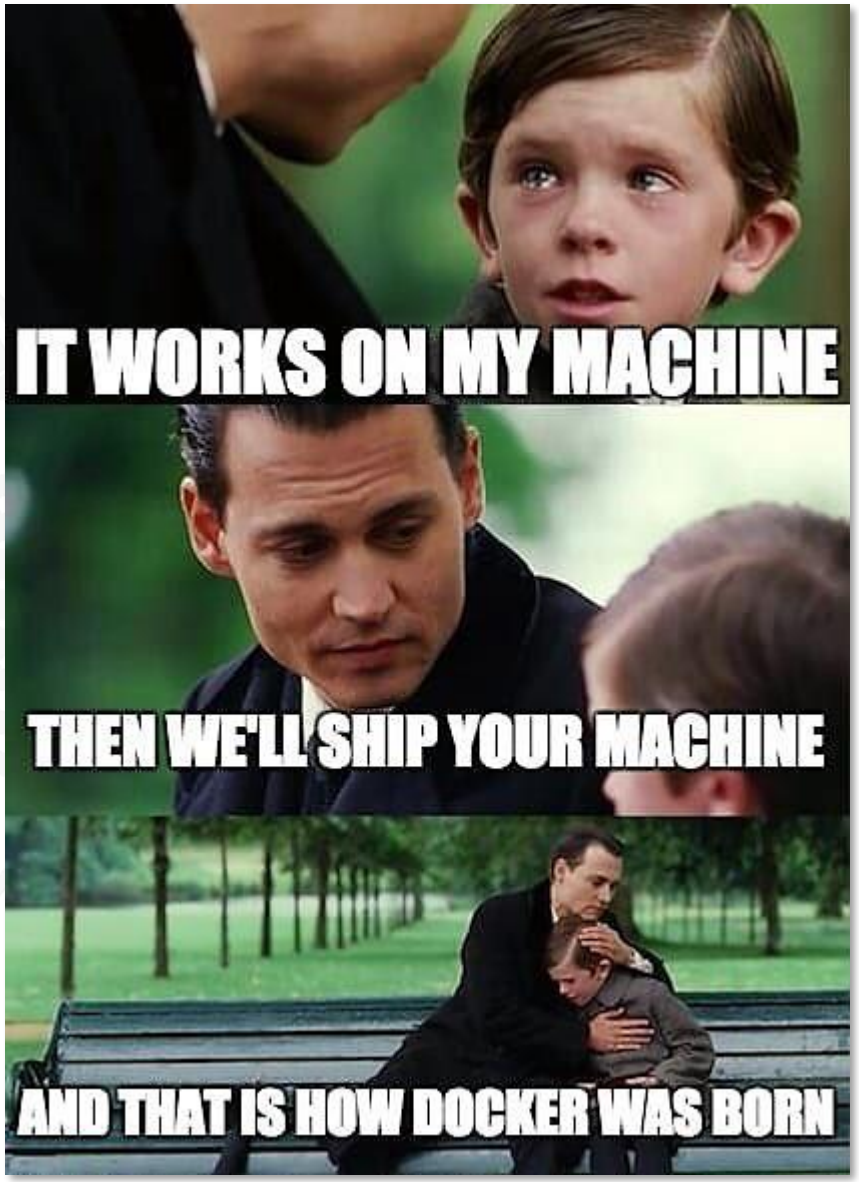
José Manuel
Redondo López

- Organización que se encarga de crear los estándares de contenedores, relativos a su formato y su entorno de ejecución
  - https://www.opencontainers.org/
- **Docker** es miembro fundador, pero existen más miembros
- Y, por tanto, existen más implementaciones de **OCI Containers** que podríamos usar como alternativa a **Docker**
- **Podman** (https://podman.io/) es una de ellas
  - Motor de contenedores que no usa **daemons** para crear y gestionar contenedores OCI en **Linux**
  - Los contenedores pueden ejecutarse como `root` o como usuario estándar
  - Reemplazar **Docker** con **Podman es posible**
    - https://medium.com/@ganeshmani009/replacing-docker-with-podman-power-of-podman-cloudnweb-23cfb7541538
- Por tanto, existen otras alternativas a **Docker**, pero nos centraremos en él como ejemplo de las capacidades de estas tecnologías
  - https://containerjournal.com/2019/01/22/5-container-alternatives-to-docker/

Introducción

José Manuel
Redondo López

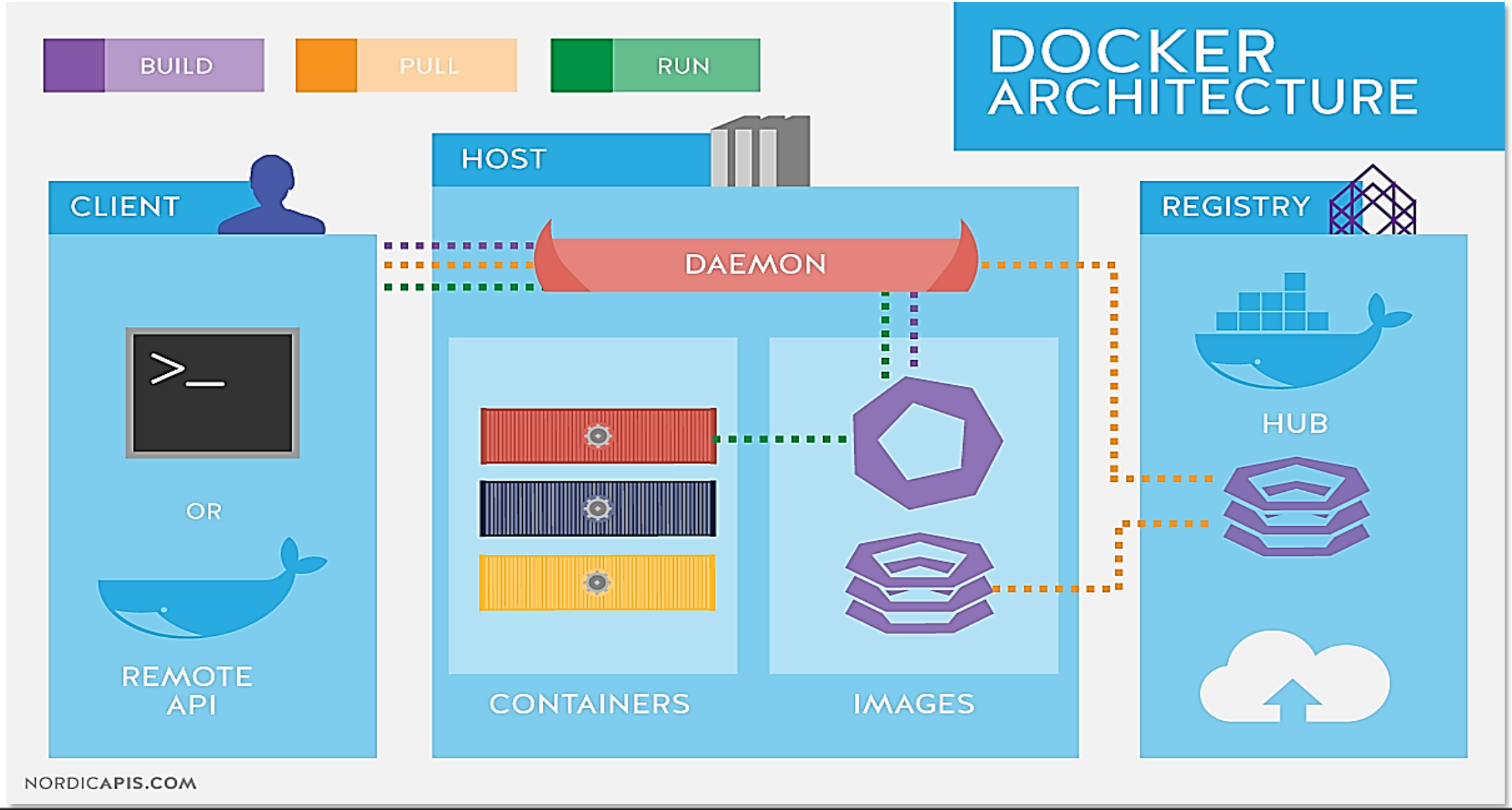
- **Docker es un creador de contenedores de aplicaciones**
  - Que ya vimos que realmente son procesos, no máquinas virtuales
- **Cada contenedor tiene un sistema de ficheros**
  - Este sistema de ficheros se crea introduciendo todo lo que unas determinadas aplicaciones necesitan para funcionar
  - Código, librerías, herramientas,…
- **El objetivo es que las aplicaciones de un contenedor siempre se ejecuten de una misma forma**
  - Sin importar el entorno en el que lo hagan
  - La aplicación no se mueve de un sistema a otro, se mueve su contenedor
  - De aquí viene el lema de la aplicación: **“Build, ship, run”**



INTRODUCCIÓN

José Manuel
Redondo López

Fuente: http://apachebooster.com/kb/what-is-a-docker-container-for-beginners/



Arquitectura de Docker: Cliente-servidor

José Manuel
Redondo López

- **Docker usa una arquitectura cliente-servidor**
  - El cliente se comunica con el daemon de Docker a través de sockets o de un API REST
  - Es este daemon el que crea, ejecuta y distribuye los contenedores
  - Este daemon se considera **una posible vulnerabilidad** si no se configura correctamente
    - Por lo que soluciones compatibles sin daemons como `podman` están ganando terreno
- **El cliente y el servidor pueden estar en la misma máquina**
  - También es posible conectar un cliente a un daemon remoto
- **El cliente de Docker es el único que se comunica con el daemon, que acepta las órdenes del usuario**

Docker CE vs Docker EE

José Manuel
Redondo López

- **Docker EE (Enterprise Edition) es la versión de pago para empresas, con certificación y soporte técnico**
  - Tiene versiones básica, estándar y avanzada
  - Las versiones más potentes poseen software de gestión de contenedores (Docker Datacenter) y aspectos de seguridad (Docker Security Scanner)
- **Docker CE (Docker Community Edition) es la versión soportada por la comunidad y gratuita**
  - Tiene dos versiones: Edge y Stable
  - La versión Edge se actualiza cada mes con las características más novedosas de la plataforma, recibiendo parches y soluciones a vulnerabilidades más frecuentemente
  - La versión Stable se actualiza con las mismas actualizaciones, pero cada 4 meses
    - Es la versión recomendada para evitar errores de funcionamiento en entornos de producción
- **Ambas se crean a partir del mismo código fuente**
  - Pero son para diferentes públicos

Imágenes (images)

José Manuel
Redondo López

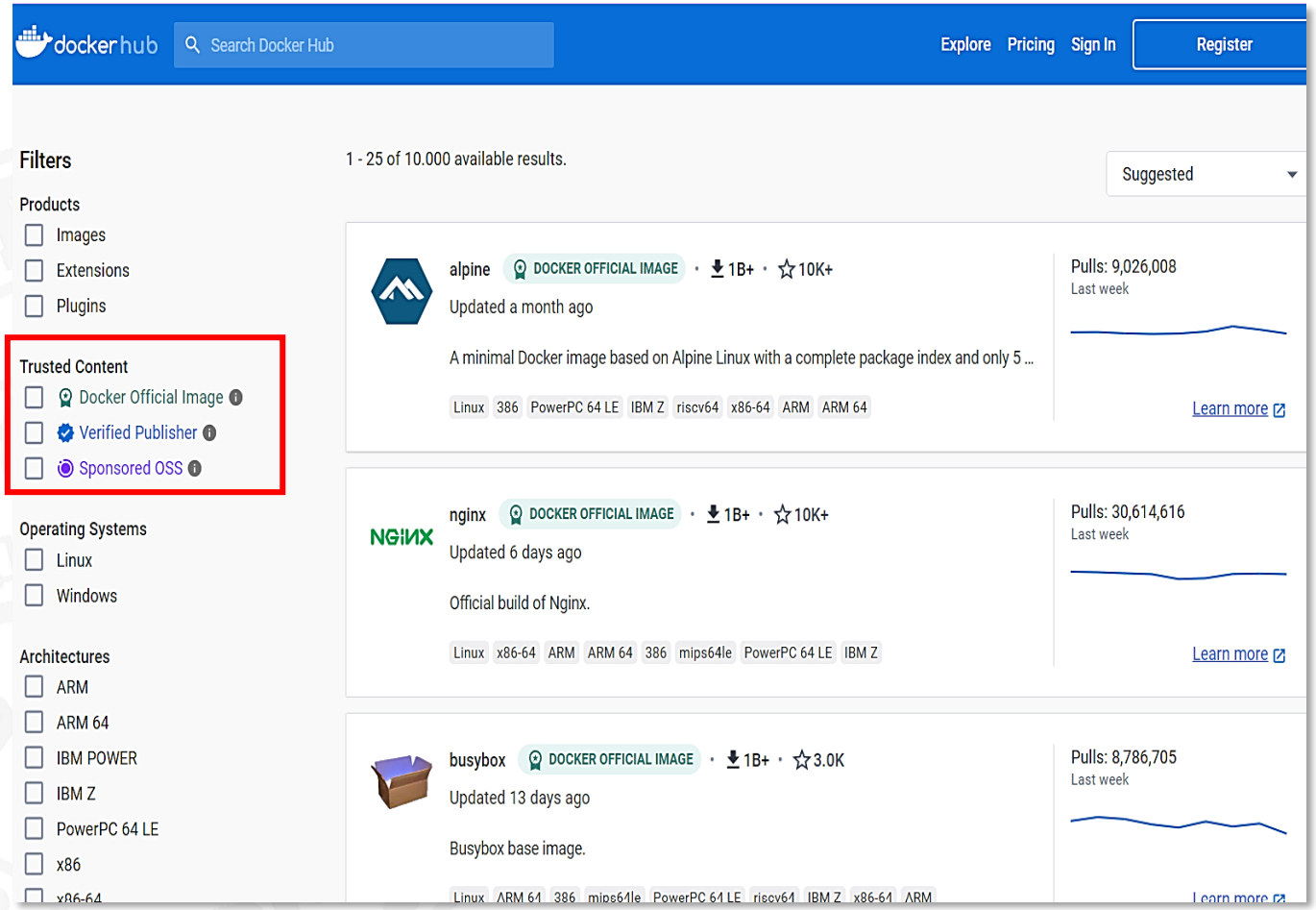
- **Son "plantillas" de solo lectura**
- **Contienen un SO y programas instalados y configurados**
  - Por ejemplo, Ubuntu Server con Apache y una aplicación web concreta instalada
- **Elementos a partir de los cuales se crean los contenedores Docker**
- **Docker permite crear nuevas imágenes o modificar existentes**
  - Se pueden usar imágenes creadas por otras personas
  - Descargándolas de algún punto de distribución
- **Son un concepto similar a las clases en la POO**
  - Modelos a partir de los cuales crear sus instancias (contenedores)

Ficha de una imagen oficial en el Docker Hub.
https://hub.docker.com/_/alpine

REGISTROS (REGISTRIES)

José Manuel
Redondo López

- Son almacenes de imágenes
- Existen púbicos y privados
- Permiten subir o descargar imágenes de otras personas
- Existe un registro **Docker** público (**Docker Hub**)
  - https://hub.docker.com/
  - Contiene una colección enorme de imágenes existentes para usar / modificar
- Para empresas, que requieren imágenes seguras y verificadas, existe **Docker Bussiness**
  - https://www.docker.com/products/business/



El registro público de imágenes Docker Hub. Por temas de seguridad, por favor usa solo imágenes de la categoría "Trusted Content"
https://hub.docker.com/search?q=

Contenedores (containers)

José Manuel
Redondo López

- Contienen todo lo necesario para que una determinada aplicación funcione
- Se crean a partir de imágenes
  - Son los “objetos” de estas “clases”
- Se pueden arrancar, parar, mover y eliminar
- Están aislados de otros contenedores Docker, aislando por tanto las aplicaciones que contienen entre ellas
- Son las **unidades de funcionamiento** de Docker

Funcionamiento de Docker

Un repaso rápido a cómo funciona el sistema

IMÁGENES Y CAPAS

José Manuel
Redondo López

- **Las imágenes se construyen a partir de una imagen base (ej.: Ubuntu, Ubuntu + Apache,…), personalizándolas**
  - Las imágenes **son inmutables** y se usan de base para construir otras configuradas para aplicaciones concretas (también inmutables una vez construidas)
- **Esta personalización consiste en añadir nuevas capas a la obtenida copiando la imagen base, través de una serie de pasos llamados instrucciones**
  - Cada instrucción incluye acciones como
    - Ejecutar un comando
    - Añadir un fichero o directorio
    - Añadir una variable de entorno
    - Especificar qué procesos ejecutar cuando se cree un contenedor a partir de esta imagen
- **Todas las instrucciones se pueden guardar en un fichero llamado Dockerfile**
  - Es un script que contiene las instrucciones y comandos para construir nuestra imagen personalizada a partir de la base
  - **Instrucciones de las capas a añadir** a la obtenida de clonar la imagen base para obtener el contenedor final

REGISTRO

José Manuel
Redondo López

- Una vez construida una imagen de **Docker** puede guardarse localmente o subirse a un registro público o privado (nuestro)
- El cliente de **Docker** puede buscar imágenes ya creadas y descargarlas al **host** donde se ejecuta para crear contenedores con ellas
- **Docker Hub** tiene un **espacio público** para guardar imágenes
  - Son visibles en cualquier búsqueda que se haga en el registro (y cualquiera podrá descargárselas)
  - También cualquiera puede subirlas, por lo que no es raro detectar contenedores con malware (troyanos, criptomineros...)
    - https://unaaldia.hispasec.com/2022/11/borrador-automatico.html/amp
    - **Usa solo imágenes oficiales en los registros** (están marcadas como tales, como vimos antes)
- También tiene un **espacio privado** para imágenes
  - Las búsquedas no las muestran
  - Solo los usuarios autorizados pueden descargárselas
  - Es necesario registrarse y establecer un **storage plan** de pago (https://www.docker.com/pricing)

CONTENEDORES

José Manuel
Redondo López

- **La imagen a partir de la que se crea un contenedor le dice a Docker**
  - Qué contiene (SO, software…)
  - Qué procesos ejecutar cuando se arranca un contenedor a partir de ella
  - Qué configuración tiene esta imagen
- **Un contenedor está compuesto además de**
  - Procesos en funcionamiento (partes del SO, software instalado…)
  - Ficheros añadidos por el usuario
  - Metadatos
- **Un contenedor al arrancar añade una nueva capa de lectura / escritura a la imagen de base (que es de solo lectura)**
  - De esta forma las aplicaciones pueden ejecutarse sin problemas
  - **Ningún contenedor puede modificar su imagen base** (como decíamos, son inmutables...)
  - **Si un contenedor muere, sus cambios se pierden**
    - Salvo que se guarden en un volumen, que veremos después

EJECUCIÓN

José Manuel
Redondo López

- **La siguiente orden:** `docker run -i -t ubuntu /bin/bash`
  - Indica a Docker que debe construir un nuevo contenedor a partir de la imagen de base llamada "`ubuntu`"
  - Docker se encarga de obtener la imagen automáticamente y construir el contenedor a partir de ella
  - Una vez arrancado, debe ejecutarse la orden "`/bin/bash`" sobre dicho contenedor

```
operario@ubuntuserver:~$ docker run -i -t ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
43db9dbdcb30: Pull complete
2dc64e8f8d4f: Pull complete
670a583e1b50: Pull complete
183b0bfcd10e: Pull complete
Digest: sha256:c6674c44c6439673bf56536c1a15916639c47ea04c3d6296c5df938add67b54b
Status: Downloaded newer image for ubuntu:latest
root@f674de7ed8dd:/# _
```

EJECUCIÓN

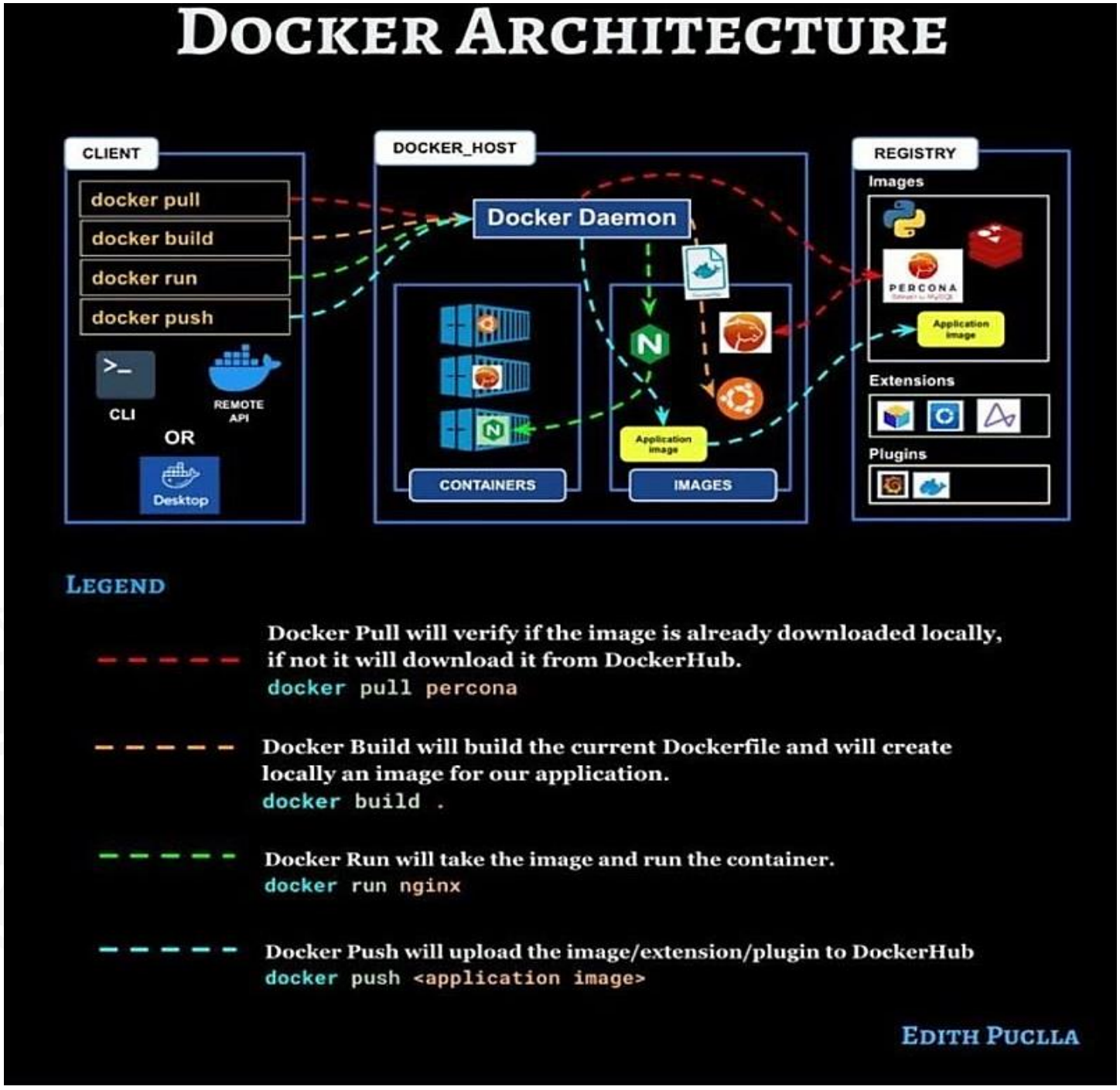
José Manuel
Redondo López

- **Para que todo esto sea posible, Docker**
  1. Adquiere la imagen llamada "`ubuntu`"
     A. Primero busca si esta imagen ya existe localmente
     B. En caso contrario, la descarga de Docker Hub
  2. Crea un nuevo contenedor a partir de la imagen descargada
  3. Crea un nuevo sistema de ficheros para el contenedor y le añade la capa de lectura/escritura
  4. Crea una interfaz de red tipo **bridge** (permite al contenedor comunicarse con el host)
  5. Asigna una IP a dicha interfaz, de una red que Docker crea por defecto para estos fines llamada normalmente `docker0`
  6. Ejecuta el proceso indicado (`/bin/bash`)
  7. Captura y muestra al usuario la salida del contenedor, conectando (**attach**) automáticamente también la entrada y las salidas de error

Ejecución vista en la arquitectura

José Manuel
Redondo López

- Los pasos vistos tienen un reflejo claro en la arquitectura de Docker
- Cada orden que damos a Docker interactúa con su daemon
  - Y actúa sobre muchos de sus elementos



Modo foreground

José Manuel
Redondo López

- **Antes arrancamos un contenedor así**
  - `docker run –i –t miw/apache_ubuntu:v1 /bin/bash`
  - `-t`: Crear una terminal
  - `-i`: Mantener la terminal creada abierta
  - `/bin/bash`: Proceso a ejecutar al crear en contenedor
- **Esta combinación de opciones es la típica para tener una terminal interactiva sobre la que ejecutar comandos en el contenedor**
  - A esto se le denomina **modo foreground**
  - Se suele abreviar con `–ti`
- **Efectivamente, el contenedor ha ejecutado 1 sola aplicación, el `bash`**
  - Es válido, pero un poco “trampa”, no es la típica aplicación que uno piensa en arrancar…
- **Más información: https://docs.docker.com/engine/reference/run/**

MODO DETACHED

José Manuel
Redondo López

- El **modo detached** (opción `-d`) es la otra forma de arranque
  - Es excluyente de las opciones `-i` y `-t`
- **No es interactivo**: ejecuta el proceso que se indique y finaliza automáticamente una vez termine
- No sirve para arrancar determinados servicios pensados como daemons o procesos en segundo plano ejecutándose permanentemente
  - `docker run -d miw/servidor_nginx service nginx start` (**¡NO!**)
  - Arranca el contenedor, ejecuta el servidor web `nginx` y…¡finaliza inmediatamente! (mata el proceso)
- Para mantener el contenedor en funcionamiento con el proceso elegido, debemos asegurarnos de que se ejecuta en primer plano
  - Lo cual muchas veces depende del proceso a ejecutar
    - `docker run -d miw/servidor_nginx nginx -g 'daemon off;'` (**¡SI!**)
  - En el caso de Apache puede lograrse con
    - `docker run -d miw/apache_ubuntu:v1 /usr/sbin/apache2ctl -D FOREGROUND`

INTERACTUAR CON UN CONTENEDOR ARRANCADO

José Manuel
Redondo López

- **En un contenedor `foreground`:**
  - Si tecleamos `exit` o `Ctrl+D`, lo finalizaremos (apagado definitivo)
  - Si hacemos `Ctrl + p + Ctrl + q` saldremos del contenedor, pero este no finalizará (como salir de sesión)
  - Podemos "reconectarnos" a dicho contenedor con `docker attach <ID o nombre del contenedor>`
  - Para obtener el id, podemos usar la salida de `docker ps`
- **En un contenedor `detached`:**
  - Para obtener una terminal que nos permita interactuar con él: `docker exec -it <ID o nombre del contenedor> bash`
  - Para salir de este terminal creado con `exec`: `Ctrl + p + Ctrl + q`

Otros comandos de Docker

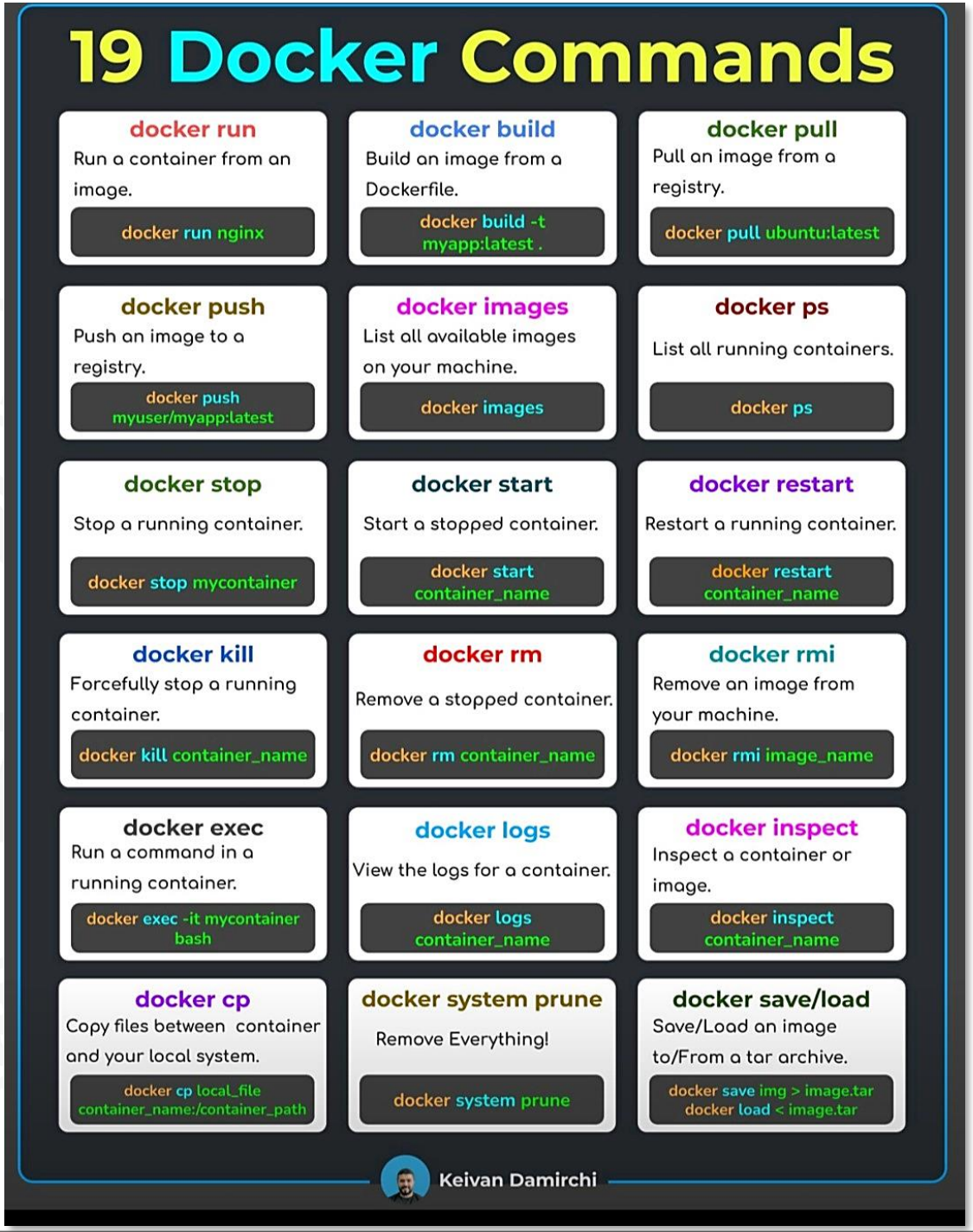
José Manuel
Redondo López

- **Estos comandos muestran el estado de Docker**
  - `docker ps`: Lista los contenedores arrancados
  - `docker images`: Lista las imágenes disponibles localmente
  - `docker inspect <id o nombre>`: Muestra la configuración de un contenedor, incluyendo la IP
    - Que se puede usar para conectarse a él desde el host (ver imagen)
- **Estos comandos necesitan un ID/nombre de contenedor y ejecutan diversas operaciones**
  - `docker kill`: Envía la señal `SIGKILL` a un contenedor (o la especificada como parámetro) para finalizarlo
  - `docker restart`: Reinicia un contenedor, con temporización opcional
  - `docker stop`: Para un contenedor mandándole las señales `SIGTERM` y posteriormente `SIGKILL`
  - `docker rm`: Elimina contenedores
    - `docker rm $(docker ps -a -q)` elimina **todos** los contenedores
  - `docker rmi`: Elimina imágenes
    - `docker rmi $(docker images -q)` elimina **todas** las imágenes

Comandos de Docker

José Manuel
Redondo López

- La aplicación en línea de comandos de Docker tiene muchos comandos
  - Al principio puede resultar intimidante
- Pero basta con usar unos pocos para la mayoría de los casos de uso
- En esta imagen están los más comunes
- En la página siguiente hay una referencia completa



https://www.fosstechnix.com/docker-command-cheat-sheet/

Docker Lifecycle Commands

Command	Use
1. docker create	Create a new container
2. docker run	Creates a docker container from docker image
3. docker pause	To pause a running container
4. docker unpause	To unpause a running container
5. docker stop	To stop a docker container
6. docker start	To start a docker container
7. docker restart	To restart docker container
8. docker attach	Attach Terminal to Running container
9. docker wait	Block until one or more containers stop, then print their exit codes
10. docker rm	To remove the Docker Container, stop it first and then remove it
11. docker kill	To stop and remove Docker containers

Docker Image Commands

Command	Use
1. docker build	To build Docker Image from Dockerfile
2. docker pull	To pull Docker Image from Docker Hub Registry
3. docker tag	To add Tag to Docker Image
4. docker images	To list Docker Images
5. docker push	To push Docker Images to
6. docker history	To show history of Docker Image
7. docker inspect	To show complete information in JSON format
8. docker save	To save an existing Docker Image
9. docker import	Create Docker Image from Tarball
10. docker export	To export existing Docker
11. docker load	To load Docker Image from file or archives
12. docker rmi	To remove docker images

Docker Compose Commands

Command	Use
1. docker-compose build	To build docker compose file
2. docker-compose up	To run docker compose file
3. docker-compose ls	To list docker images declared inside docker compose file
4. docker-compose start	To start containers which are already created using docker compose file
5. docker-compose run	To run one one of application inside docker-compose.yml
6. docker-compose rm	To remove docker containers from docker compose
7. docker-compose ps	To check docker container status from docker compose

Docker Basic Commands

Command	Use
1. docker	To check all available Docker Commands
2. docker version	To show Docker version
3. docker info	Displays system wide information
4. docker pull	To pull the docker Images from Docker Hub Repository
5. docker build	To Docker Image from Dockerfile
6. docker run	Run a container from a docker
7. docker commit	To commit a changes in container file OR create new Docker Image
8. docker ps	List all the running containers. Add the -a flag to list all the containers.
9. docker start	To start a docker container
10. docker stop	To stop a docker container
11. docker logs	To view Logs for a Docker Container
12. docker rename	To rename Docker Container
13. docker rm	To remove the Docker Container, stop it first

Docker Container Commands

Command	Use
1. docker start	To start a Docker container
2. docker stop	To stop a running docker container
3. docker restart	To restart docker container
4. docker pause	To pause a running container
5. docker unpause	To unpause a running container
6. docker run	Creates a docker container from docker image
7. docker ps	To list Docker containers
8. docker exec	To Access the shell of Docker Container
9. docker logs	To view Logs for a Docker Container
10. docker rename	To rename Docker Container
11. docker rm	To remove Docker container
12. docker inspect	Docker container info command
13. docker attach	Attach Terminal to Running container
14. docker kill	To stop and remove Docker containers
15. docker cp	To copy files or folders between a container and from local filesystem.

Docker Networking Commands

Command	Use
1. docker network create	To create docker network
2. docker network ls	To list docker networls
3. docker network inspect	To view network configuration details

Docker Prune Commands

Command	Use
1. docker system prune	To clean all resources which are dangling or not associated with any docker containers
2. docker image prune	To remove Dangling Docker images
3. docker container prune	To remove all unused docker containers
4. docker volume prune	To remove all unused docker volumes
5. docker network prune	To remove all unused docker network

Docker Volume Commands

Command	Use
1. docker volume create	To create docker volume
2. docker volume inspect	To inspect docker volume
3. docker volume rm	To remove docker volume

Docker Logs and Monitoring Commands

Command	Use
1. docker ps -a	To show running and stopped containers
2. docker logs	To show Docker container logs
3. docker events	To get all events of docker container
4. docker top	To show running process in docker container
5. docker stats	To check cpu, memory and network I/O usage
6. docker port	To show docker containers public ports

Docker Hub Commands

Command	Use
1. docker search	To search docker image
2. docker pull	To pull image from docker hub
3. docker login	To Log in to a Docker registry
4. docker push	Push an image or a repository to a registry
5. docker tag	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
6. docker logout	To logout from Docker Hub Registry

Contenedores con nombre

José Manuel
Redondo López

- **Si no queremos usar el ID de un contenedor (es aleatorio), podemos asignar un nombre a los contenedores**
  - Especificando `--name <nombre sin espacios>` al hacer `docker run`
- **Por defecto Docker asigna uno aleatorio compuesto de dos palabras separadas por _**
  - Ej.: `focused_shockley`
- **Podemos usar el nombre de una máquina en cualquier lugar donde hayamos dicho que debe usarse su ID**
  - Ambos son intercambiables para todos los comandos vistos

PROBLEMAS TÍPICOS DE ARRANQUE

José Manuel
Redondo López

- Tenemos un contenedor **Ubuntu** actualizado, con una serie de paquetes instalados, independiente y aislado del host
- ¿Qué problemas tiene?
  1. **Apache no se arranca automáticamente al arrancar el contenedor de esta forma**
     - Tendríamos que hacer `service apache2 start` una vez dentro de él
     - O copiar y ejecutar un **script** al contenedor que arranque lo necesario
     - Lo mismo ocurre con otros servicios que queramos arrancar complementariamente
  2. **No hay forma de comunicarse con él desde fuera del host**
     - Esto lo veremos en la siguiente sección
  3. **A lo mejor no queremos arrancarlo inmediatamente**: En ese caso usamos `docker create` para crear el contenedor, pero **NO arrancarlo inmediatamente**
     - Luego podemos hacerlo con `docker start`
     - https://docs.docker.com/engine/reference/commandline/create/

Redes en Docker

Comunicación entre contenedores y con el exterior

Configuración de red por defecto en Docker

José Manuel
Redondo López

- **Docker** modifica la configuración de red del host
  - Se añade una interfaz adicional tipo **bridge** llamada `docker0`
  - Comunica **bidireccionalmente host y contenedores (y los contenedores entre sí)**
  - Sí, **el host aparecerá por defecto como una IP más ante los contenedores**
  - Hace de **gateway** y permite a los contenedores acceder a Internet / otras redes del host
- **Se crea con un rango de IPs que no entre en conflicto con otro existente en el host**

```
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        inet6 fe80::42:13ff:fe76:8b1  prefixlen 64  scopeid 0x20<link>
        ether 02:42:13:76:08:b1  txqueuelen 0  (Ethernet)
        RX packets 26744  bytes 1114960 (1.1 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 27281  bytes 46133942 (46.1 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe38:e887  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:38:e8:87  txqueuelen 1000  (Ethernet)
        RX packets 326784  bytes 356455031 (356.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 87763  bytes 5368109 (5.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.105  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::a00:27ff:fe04:c635  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:04:c6:35  txqueuelen 1000  (Ethernet)
        RX packets 279  bytes 57273 (57.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 82  bytes 23968 (23.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 152  bytes 16150 (16.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 152  bytes 16150 (16.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Configuración de red por defecto en Docker

José Manuel
Redondo López

- **Dentro de cada contenedor Docker la configuración de red por defecto es**
  - Se crea un interfaz `eth0` con una IP en la red que Docker usa por defecto
  - La red interna de Docker tendrá un CIDR como `172.17.0.0/16`
  - Por tanto, cada contenedor tiene una IP `172.17.0.x` con mascara `255.255.0.0`
  - El host será siempre la `172.17.0.1`
- **En función de lo que ya hayamos creado, la red puede ser también `172.18.0.x`, `172.19.0.x`, etc.**

```
redondo@miw:~$ sudo docker run -d -p 192.168.56.106:80:80 miw/apache_ubuntu:v1 /usr/sbin/apache2ctl
-D FOREGROUND
2212cef85af2fceda7f048468b7c27749da0aa1f41d97f5e0004954677a0bcbb
redondo@miw:~$ sudo docker ps
CONTAINER ID        IMAGE                 COMMAND                  CREATED             STATUS
       PORTS                         NAMES
2212cef85af2        miw/apache_ubuntu:v1   "/usr/sbin/apache2ct…"   5 seconds ago       Up 4 seconds
       192.168.56.106:80->80/tcp   eager_fermi
62b5436301b4        miw/apache_ubuntu:v1   "/usr/sbin/apache2ct…"   14 minutes ago      Up 14 minute
s                                  zen_lehmann
631b37dfd3b5        miw/apache_ubuntu:v1   "/bin/bash"              18 minutes ago      Up 18 minute
s                                  objective_ptolemy
redondo@miw:~$ sudo docker exec -it eager_fermi bash
root@2212cef85af2:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
       inet 172.17.0.4  netmask 255.255.0.0  broadcast 172.17.255.255
       ether 02:42:ac:11:00:04  txqueuelen 0  (Ethernet)
       RX packets 9  bytes 726 (726.0 B)
       RX errors 0  dropped 0  overruns 0  frame 0
       TX packets 0  bytes 0 (0.0 B)
       TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
       inet 127.0.0.1  netmask 255.0.0.0
       loop  txqueuelen 1000  (Local Loopback)
       RX packets 0  bytes 0 (0.0 B)
       RX errors 0  dropped 0  overruns 0  frame 0
       TX packets 0  bytes 0 (0.0 B)
       TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@2212cef85af2:/# _
```

VISIBILIDAD DE UN CONTENEDOR

José Manuel
Redondo López

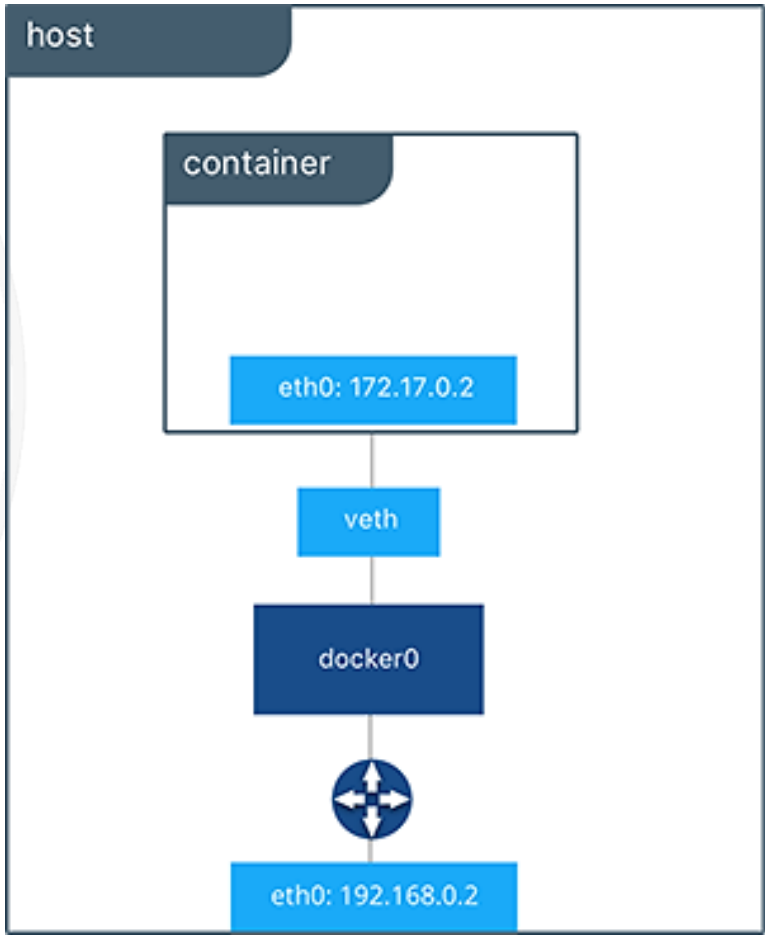
- **¿Qué podemos hacer para que los contenedores creados (los servicios que ejecutan) puedan ser usados externamente?**
- **Una opción sencilla es la siguiente**
  - En el arranque de los contenedores, especificar **qué puertos de cada contenedor se mapearán** sobre su IPs (y puertos de las mismas)
    - Ejemplo: `docker run -d -p 192.168.56.106:80:80 miw/apache_ubuntu:v1 /usr/sbin/apache2ctl -D FOREGROUND`
  - Cualquier acceso a esas combinaciones **IP:puerto** del host se hará en realidad sobre un servicio en ejecución en el contenedor sobre el que se haya mapeado
  - También se pueden asignar al host **múltiples IPs accesibles desde el exterior**
  - Si el host es una MV de VirtualBox, conviene hacerlo sobre una tarjeta de red secundaria habilitada para ello
    - No la que se usa para acceder habitualmente a Internet vía NAT por DHCP

Creación de redes entre contenedores

José Manuel
Redondo López

- Cualquier contenedor puede estar conectado a una o más redes que se creen para ellos
- Docker tiene de serie dos tipos de redes
  - **Bridge**: redes que solo pueden estar dentro de una misma máquina donde está instalado Docker
  - **Overlay**: redes que pueden incluir varios hosts, si bien es algo que supera los objetivos de esta asignatura
- Recién instalado Docker tiene 3 redes creadas: `none`, `host` y `bridge`
  - Se pueden ver con `docker network ls`
  - Esta última es la **red bridge por defecto** (`default`), a la que se conectan todos los contenedores que no especifican una al crearse
  - **También da salida a Internet vía NAT** por la conexión del host
- `docker inspect` saca un listado de todas las redes a las que un host está conectado

https://docs.docker.com/engine/tutorials/networkingcontainers/

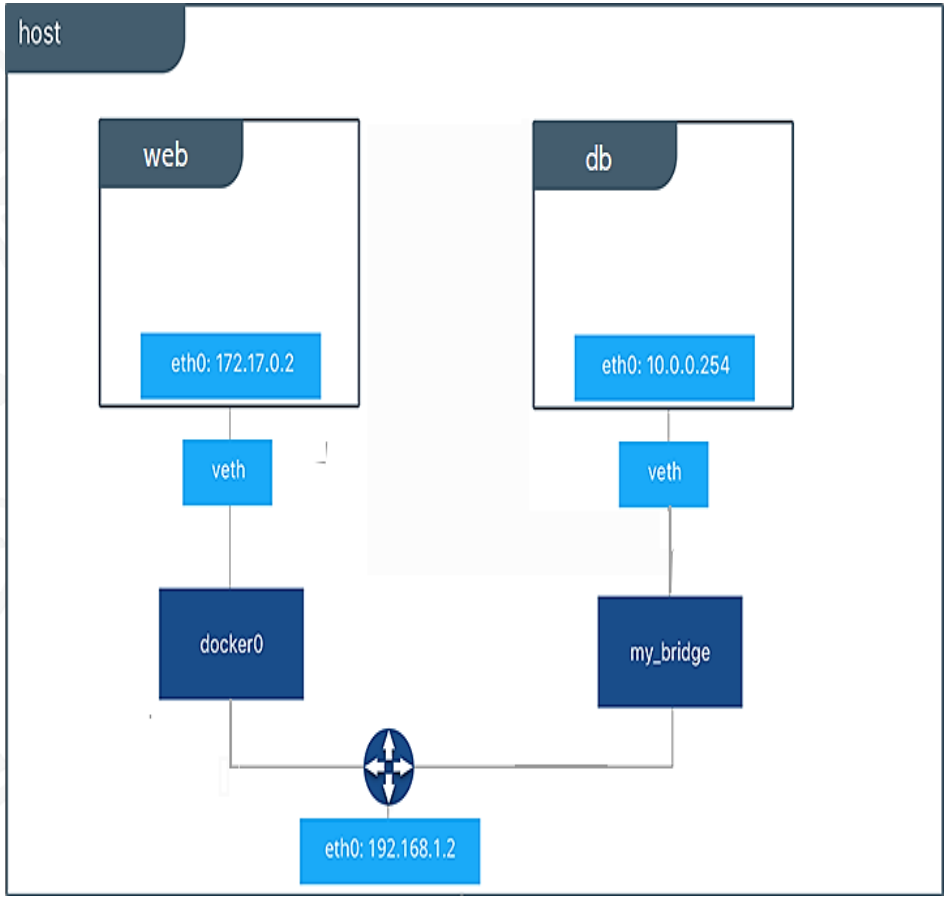


CREACIÓN DE REDES ENTRE CONTENEDORES

José Manuel
Redondo López

- **Se pueden crear nuevas redes con** `docker network create`
  - Con un rango de IPs elegido automáticamente
  - Ej.: `docker network create -d bridge nueva_red`
  - Podemos también especificar un rango de Ips para las nuevas redes
    - `docker network create –d bridge nueva_red --subnet=10.0.0.0/24`
- **Esto hace que cualquier contenedor pueda usar una nueva red tipo bridge llamada** `nueva_red`
  - Ej.: `docker run -d --net=nueva_red --name db mysql`
- **Si arrancamos** `docker run -d --name web apache` **sin especificar una red, se unirá a la red por defecto**
  - Quedando el esquema del dibujo

https://docs.docker.com/engine/tutorials/networkingcontainers/

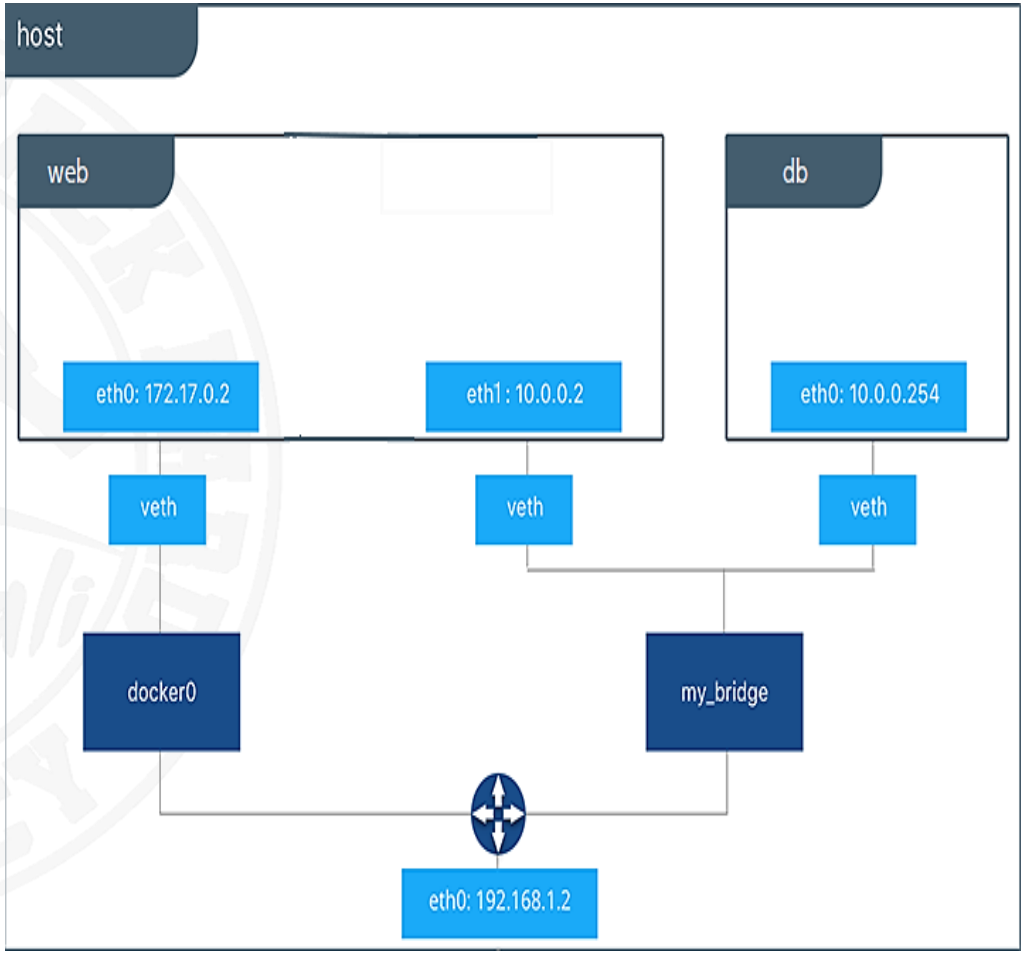


CREACIÓN DE REDES ENTRE CONTENEDORES

José Manuel
Redondo López

- **Pero con este esquema `web` no puede ver a `db` porque están aislados en redes distintas**
  - Podemos solucionarlo con `docker network connect nueva_red web`, quedando el esquema del dibujo
  - Ahora si habría conectividad entre ambas máquinas
- **Cualquier contenedor se puede desconectar de una o más redes en cualquier momento**
  - Ej.: `docker network disconnect nueva_red web`
- **Usar redes bridge creadas manualmente tiene una serie de ventajas**
  - https://docs.docker.com/engine/userguide/networking/work-with-networks/

https://docs.docker.com/engine/tutorials/networkingcontainers/



PROHIBIR LA COMUNICACIÓN

José Manuel
Redondo López

- **Podemos impedir que los contenedores se comuniquen entre ellos por defecto**
  - **Por seguridad:** un contenedor comprometido por malware no podrá localizar otros
- **Para ello modificamos el fichero** `/lib/systemd/system/docker.service`
  - Con la opción `--icc=false` en el arranque de Docker (al final de la línea)
  - Posteriormente, reiniciamos los daemons con `sudo systemctl daemon-reload`
  - Luego reiniciamos el servicio (`sudo service docker restart`)
  - **¡Hacer esto implica que los contenedores arrancados se pararán!**

```
 GNU nano 2.5.3          Archivo: /lib/systemd/system/docker.service

[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
After=network.target docker.socket
Requires=docker.socket

[Service]
EnvironmentFile=-/etc/default/docker
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --icc=false
ExecReload=/bin/kill -s HUP $MAINPID
# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity
# Uncomment TasksMax if your systemd version supports it.
# Only systemd 226 and above support this version.
TasksMax=infinity
TimeoutStartSec=0
# set delegate yes so that systemd does not reset the cgroups of docker containers
Delegate=yes
# kill only the docker process, not all processes in the cgroup
KillMode=process

[Install]
```

Enlazado de contenedores

José Manuel
Redondo López

- **Es posible enlazar contenedores (docker container linking) entre ellos usando los nombres que les hemos asignado**
  - Manualmente con `--name` o los que se crean por defecto
  - Dentro de los contenedores enlazados **se crean variables de entorno** que contienen las IPs de los contenedores a los que se enlaza
- **Se hace por dos motivos**
  - Facilitar localizar las IPs en la red de Docker de contenedores que colaboran
    - Las IPs se asignan por DHCP y dependen del orden de arranque, dificultando hacerlas fijas
  - Si hemos deshabilitado la comunicación entre contenedores, **los contenedores enlazados SÍ pueden comunicarse entre ellos**
  - En cualquiera de los casos, podemos comunicar unos contenedores con otros con las variables de entorno que se crean, cuyo valor podría cambiar en cada arranque
    - Las aplicaciones que usen las variables de entorno les da igual que cambien las IPs ☺

HOST NETWORKING

José Manuel
Redondo López

- Si se pasa `--net=host` en el arranque de un contenedor, éste podrá usar directamente la configuración de red del **host**
- Esto quiere decir que si desde el contenedor se hace `ifconfig` la configuración de red y los interfaces que veremos serán los mismos que tiene el **host**
- En este escenario exponer puertos (opción `-p`) no tiene sentido
  - Ya que los servicios que se estén ejecutando en el contenedor serán directamente visibles en el host
- **Pero cuidado**: ¡Todo lo que ejecutemos queda expuesto a los clientes del **host** automáticamente!

Dockerfiles y optimización de imágenes

Configurando contenedores óptimos desde ficheros de texto

Ficheros Docker (Dockerfiles)

José Manuel
Redondo López

- **Son los ficheros donde los usuarios programan una serie de comandos que construyen una imagen (el DSL de Docker)**
- **Podemos poner cualquier secuencia de comandos Docker en forma de programa, haciéndolo plenamente automatizable**
  - Esto nos permite distribuir y reproducir cómo construir nuestras imágenes muy fácilmente (simplemente pasamos **ficheros de texto** de pocos Kb)
  - En lugar de trabajar con las imágenes en sí (que en realidad son **ficheros** `.tar` más grandes)
- **Simplifica el mantenimiento**
  - Múltiples Dockerfiles con variantes de una imagen base personalizadas para múltiples propósitos
  - Asi podemos “desplegar” una arquitectura completa simplemente procesando las Dockerfiles de cada componente

DOCKERFILES

José Manuel
Redondo López

- **Para probarlo, creamos un directorio y dentro un fichero llamado `Dockerfile`**
- **Fíjate en las instrucciones usadas**
  - `FROM`: La imagen base a partir de la cual trabajar
  - `RUN`: Acciones a ejecutar en la imagen
  - `EXPOSE`: Indica el puerto a partir del cual va a estar disponible un servicio
- **Ninguna instrucción debe hacer preguntas al usuario**
  - En caso contrario, el proceso se interrumpirá
  - De ahí que se haya usado la opción `-y` y la variable de entorno `DEBIAN_FRONTEND`

```
FROM ubuntu
ENV DEBIAN_FRONTEND=noninteractive
RUN apt update
RUN apt -y dist-upgrade
RUN apt -y install apache2
RUN apt -y install net-tools
RUN apt -y install nano
RUN apt -y install curl
RUN apt -y install nmap
RUN echo 'Imagen construida con una Dockerfile' \
        > /var/www/html/index.html
EXPOSE 80
```

SampleUbuntuDockerfile

DOCKERFILES

José Manuel
Redondo López

- Los **Dockerfiles** se procesan con `docker build`
- Debe especificarse con `-t` cómo se llamará la imagen a construir
- Por defecto busca un fichero `Dockerfile` en la ruta actual
  - Pero puede especificarse otro fichero
- Si la construcción termina correctamente veremos una imagen nueva en el listado de imágenes de **Docker**
  - Que podemos usar igual que las que hemos creado anteriormente

```
redondo@miw:~$ sudo docker build -t="miw/dockerfile_apache:v1" .
Sending build context to Docker daemon  62.58MB
Step 1/10 : FROM ubuntu
 ---> adafef2e596e
Step 2/10 : RUN apt update
 ---> Running in 06df6051fb8e

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [107 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [111 kB]
Get:4 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages
 [33.9 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [
44.5 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [164
kB]
Get:7 http://archive.ubuntu.com/ubuntu focal-backports InRelease [98.3 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages
 [1077 B]
Get:9 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
Get:10 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB
]
```

```
redondo@miw:~$ sudo docker images
REPOSITORY              TAG                 IMAGE ID            CREATED
    SIZE
miw/dockerfile_apache   v1                  2a23962551ae        7 seconds ago
    242MB
<none>                  <none>              cbf51963182e        5 minutes ago
    96.1MB
miw/apache_ubuntu       v1                  0c1ef5e144d3        11 hours ago
    237MB
ubuntu                  latest              adafef2e596e        7 days ago
    73.9MB
hello-world             latest              bf756fb1ae65        6 months ago
    13.3kB
```

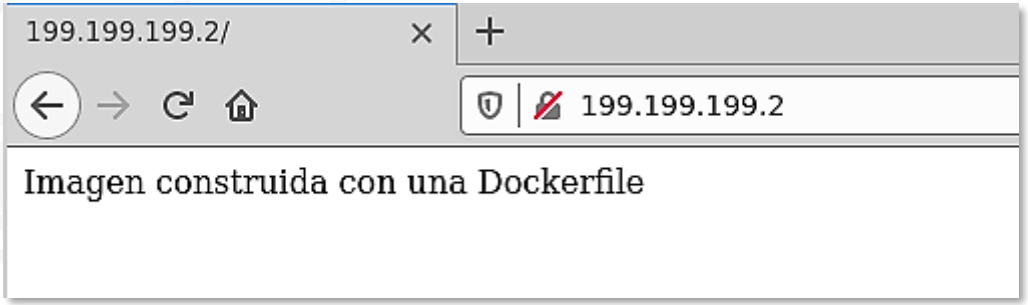
DOCKERFILES

José Manuel
Redondo López

- La imagen queda construida y funciona exactamente igual que las que vimos anteriormente
- Que, por supuesto, permite al **host** acceder a su contenido
  - Una página construida con la orden `ECHO` de la `Dockerfile`
- Aprender a crear **Dockerfiles** es muy importante
  - Su composición, sintaxis, parámetros y referencia completa de comandos son información pública
  - https://docs.docker.com/engine/reference/builder/

```
redondo@miw:~$ sudo docker run -d miw/dockerfile_apache:v1 /usr/sbin/apache2ctl
-D FOREGROUND
abde03a142fddad9ddaf92903d92faeb16fd95b90eefaf1ac667eea731fca456
redondo@miw:~$ sudo docker ps
CONTAINER ID        IMAGE                        COMMAND                  CREATED
             STATUS              PORTS               NAMES
abde03a142fd        miw/dockerfile_apache:v1     "/usr/sbin/apache2ct…"   15 secon
ds ago      Up 14 seconds       80/tcp              happy_meninsky
redondo@miw:~$ sudo docker exec -it happy_meninsky bash
root@abde03a142fd:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 199.199.199.2  netmask 255.255.255.0  broadcast 199.199.199.255
        ether 02:42:c7:c7:c7:02  txqueuelen 0  (Ethernet)
        RX packets 11  bytes 906 (906.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```



USO DE UNA IMAGEN BASE PEQUEÑA

José Manuel
Redondo López

- **Disminuir el tamaño** de las imágenes que usemos es prioritario si vamos a usar una gran cantidad de ellas
- Para ello hay una serie de trucos que veremos a continuación
- Si bien lo enfocaremos a contenedores interactivos, se puede aplicar sin problema a cualquier otro tipo de contenedor
- Pero no debemos olvidar la norma de **instalar solo lo mínimo necesario**
  - Y no cosas que no vamos a usar o tenemos disponibles en el host (Ej.: el `man`)

Uso de una imagen base pequeña

José Manuel
Redondo López

- Para crear una imagen de **Docker** necesitamos partir de una imagen base a la que añadir lo que necesitemos
- Cuanto más pequeña sea dicha imagen menos costará su descarga y menos ocupará todo lo que derivamos de ella
- La imagen **Linux** más pequeña posible se llama **Alpine** (como en el caso de LXD)
  - Es la que tendríamos que intentar usar siempre si nos es posible
  - Ocupa solo 5Mb
- No obstante, puede no ser suficiente para lo que pretendemos
  - En esos casos, es típico también usar imágenes base oficiales como `ubuntu:focal` (alrededor de 70Mb)

USO DE UNA IMAGEN BASE PEQUEÑA

José Manuel
Redondo López

- La instalación de paquetes es clave a la hora de controlar el tamaño de la imagen
- **Cada instrucción RUN añade una capa al contenedor y aumenta su tamaño**
  - En lugar de hacer varios RUN, es mejor instalar todos los paquetes necesarios con uno
  - Separando las ordenes con &&
- **Los paquetes se pueden poner todos seguidos**
  - O en líneas distintas, separándolos con \

```
# Latest public official Ubuntu image release
FROM ubuntu:focal
# Avoid errors from asking for user input
ENV DEBIAN_FRONTEND=noninteractive

# Color terminal for better user experience
ENV TERM=xterm-color

# Do not exclude man pages & other documentation
# RUN rm /etc/dpkg/dpkg.cfg.d/excludes
# COPY excludes /etc/dpkg/dpkg.cfg.d/

# Prepare the base packages with some useful tools in a way image size is reduced.
RUN apt-get update && apt-get -y full-upgrade \
    && apt-get install -y --no-install-recommends \
        apt-utils \
        curl \
        iputils-ping \
        nano \
        net-tools \
        wget \
        less \
        tmux \
        gpm \
        mc \
    && rm -rf /var/lib/apt/lists/*

COPY tmux.conf /etc/tmux.conf
COPY nanorc /root/.nanorc

LABEL   maintainer="Jose Manuel Redondo Lopez" \
    version="1.0" \
    description="Base Ubuntu image for SSI laboratories (2020-2021)"
```

MinimumSizeUbuntuDockerfile

USO DE UNA IMAGEN BASE PEQUEÑA

José Manuel
Redondo López

- Cuando se instalan paquetes se instalan también sus dependencias necesarias
- Pero a veces se instalan también otras recomendadas
  - No son estrictamente necesarias
- Esto aumenta mucho el tamaño de una imagen
- Para evitarlo, se debe usar la opción `--no-install-recommends`

```
# Latest public official Ubuntu image release
FROM ubuntu:focal
# Avoid errors from asking for user input
ENV DEBIAN_FRONTEND=noninteractive

# Color terminal for better user experience
ENV TERM=xterm-color

# Do not exclude man pages & other documentation
# RUN rm /etc/dpkg/dpkg.cfg.d/excludes
# COPY excludes /etc/dpkg/dpkg.cfg.d/

# Prepare the base packages with some useful tools in a way image size is reduced.
RUN apt-get update && apt-get -y full-upgrade \
    && apt-get install -y --no-install-recommends \
        apt-utils \
        curl \
        iputils-ping \
        nano \
        net-tools \
        wget \
        less \
        tmux \
        gpm \
        mc \
    && rm -rf /var/lib/apt/lists/*

COPY tmux.conf /etc/tmux.conf
COPY nanorc /root/.nanorc

LABEL   maintainer="Jose Manuel Redondo Lopez" \
    version="1.0" \
    description="Base Ubuntu image for SSI laboratories (2020-2021)"
```

Uso de una imagen base pequeña

José Manuel
Redondo López

- **Actualizar los paquetes de la imagen mejora la seguridad**
  - Pero una vez desplegado debe hacerse con cuidado: queremos un entorno estable
- **Si solo actualizamos la base, hay riesgo de que otros que dependen de ella tengan problemas al instalar software**
  - Las referencias a los paquetes pueden haber cambiado de IP
  - Al menos tendrían que hacer un `apt-get update` o equivalente
- **Una vez actualizado se puede ahorrar mucho espacio borrando la caché de `apt` con `&& rm -rf /var/lib/apt/lists/*`**

```
# Latest public official Ubuntu image release
FROM ubuntu:focal
# Avoid errors from asking for user input
ENV DEBIAN_FRONTEND=noninteractive

# Color terminal for better user experience
ENV TERM=xterm-color

# Do not exclude man pages & other documentation
# RUN rm /etc/dpkg/dpkg.cfg.d/excludes
# COPY excludes /etc/dpkg/dpkg.cfg.d/

# Prepare the base packages with some useful tools in a way image size is reduced.
RUN apt-get update && apt-get -y full-upgrade \
    && apt-get install -y --no-install-recommends \
        apt-utils \
        curl \
        iputils-ping \
        nano \
        net-tools \
        wget \
        less \
        tmux \
        gpm \
        mc \
    && rm -rf /var/lib/apt/lists/*

COPY tmux.conf /etc/tmux.conf
COPY nanorc /root/.nanorc

LABEL   maintainer="Jose Manuel Redondo Lopez" \
    version="1.0" \
    description="Base Ubuntu image for SSI laboratories (2020-2021)"
```

USO DE UNA IMAGEN BASE PEQUEÑA

José Manuel
Redondo López

Otros trucos

- **Recuerda**: para evitar que la instalación de un paquete pregunte algo al usuario (rompe la instalación) debe usarse al principio de la Dockerfile: `ENV DEBIAN_FRONTEND=noninteractive`
- Si vamos a usar la consola de forma interactiva, puede que nos interese usar colores, y para ello debemos añadir esto al principio de la Dockerfile: `ENV TERM=xterm-color`
- Podemos configurar editores (Ej.: `nano`) y uso de varias terminales simultaneas (`tmux`)
- Para ello lo mejor es copiar nuestra configuración al contenedor siempre usando `COPY` en lugar de `ADD` (recomendación de Docker)
  - `COPY tmux.conf /etc/tmux.conf`
  - `COPY nanorc /root/.nanorc`
  - Si no queremos copiar todo el contenido de una carpeta se puede usar un fichero `.dockerignore`
- Vemos también `LABEL`, que permite etiquetar nuestras imágenes **para documentarlas**
- Se podría usar un fichero `excludes` que evitaría que en el contenedor se instalasen documentaciones (`man`) y otros ficheros de lo que instalemos en él (en el ejemplo está comentado)
- ¡A menos ficheros en el contenedor, menos tamaño!

excludes

```
# Drop all man pages
# path-exclude=/usr/share/man/*

# Drop all translations
path-exclude=/usr/share/locale/*/LC_MESSAGES/*.mo

# Drop all documentation ...
path-exclude=/usr/share/doc/*

# Drop all copyright files ...
path-exclude=/usr/share/doc/*/copyright

# Drop all Debian changelogs
path-exclude=/usr/share/doc/*/changelog.Debian.*
```

Uso de una imagen base pequeña

José Manuel
Redondo López

- Con nuestra imagen base creada ya podemos empezar a crear otras imágenes con funcionalidades más especializadas que dependan de ella
  - Así tendremos una base común para nuestros servidores web, FTP, MySQL, etc.
  - Y **maximizaremos el uso de la caché de Docker**
- Una vez construida la imagen base, todas las demás la reutilizarán tal cual fue construida, ahorrando mucho tiempo de construcción
  - https://phoenixnap.com/kb/docker-image-size
- Otra opción para otros contextos es usar **multi-stage builds** (vistas después)

IMÁGENES ENLAZADAS

José Manuel
Redondo López

- **Vemos como esta nueva imagen se basa en la anterior**
  - No tenemos que repetir sus operaciones, y solo instalar lo nuevo
- **Además, expone el puerto HTTP para Apache2**
- **Arranca los servicios instalados copiando y ejecutando un script**
  - Este arranca Apache2 y usa un truco para parar la ejecución indefinidamente (`tail`)
  - Y que el contenedor no muera

```
#!/bin/bash
service apache2 start
tail -f /dev/null
```

```
# Start from our base image to have a common toolset
FROM ssi/ubuntu_base

# Avoid errors from asking for user input
ENV DEBIAN_FRONTEND=noninteractive

# Prepare the base packages with some useful tools
# in a way image size is reduced.
RUN     apt-get -y update && \
    apt-get install -y --no-install-recommends \
        apache2 \
    && rm -rf /var/lib/apt/lists/*

# Copy and run an initialization script to
# start the appropriate servers in the containers
COPY container_init.sh container_init.sh
RUN chmod +x container_init.sh
CMD ["./container_init.sh"]

# Expose ports
EXPOSE 80

LABEL   maintainer="Jose Manuel Redondo Lopez" \
    version="1.0" \
    description="Base Ubuntu Apache HTTP web server"
```

ApacheDockerfile

Volúmenes de Docker

Almacenamiento externo para contenedores

Volúmenes en Docker

José Manuel
Redondo López

- **Por la naturaleza de los contenedores, cualquier dato que se escriba en su sistema de ficheros será eliminado al destruirse los mismos**
  - Un contenedor **añade una capa de lectura/escritura** a la imagen que usa de base
  - Pero que **la imagen en sí no puede ser alterada** por ninguno de sus contenedores
- **Para aquellos casos en los que necesitemos almacenamiento persistente en sistema de ficheros existen los volúmenes de datos Docker**
  - La instrucción `VOLUME` de las Dockerfiles

Volúmenes en Docker

José Manuel
Redondo López

- **Un volumen de datos es un directorio especial del host que puede ser compartido entre varios contenedores**
- **Aporta las siguientes ventajas**
  - **Se inicializan en la creación del contenedor**: Si la imagen base tiene información en el punto de montaje, esos datos se copian en el volumen creado durante la inicialización
  - **Pueden compartirse y reutilizarse** entre contenedores
  - Los cambios en los datos del volumen **se hacen directamente**
  - Si se actualiza una imagen no se actualizan los datos del volumen
  - Y más importante, **si una imagen se borra sus volúmenes no se borran**
    - Los volúmenes están pensados pues para persistir datos independientemente del ciclo de vida de los contenedores que los usen

¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

- *¿Entiendes que Docker es una tecnología de entre varias que pertenecen a la iniciativa OCI?*
- *¿Comprendes bien qué puedes encontrar dentro de un contenedor Docker y cómo es la arquitectura que los pone en marcha?*
- **Funcionamiento de Docker**
  - *¿Entiendes qué es una imagen, que funciona con capas y que las imágenes son inmutables?*
  - *¿Entiendes que el registro es un repositorio de imágenes y que el Docker Hub no es más que un registro accesible globalmente?*
  - *¿Comprendes que si no tienes cuidado puedes descargarte malware del Docker Hub?*
  - *¿Entiendes la relación que hay entre un contenedor y su imagen base y lo que se hace para construir uno a partir de ella?*
  - *¿Recuerdas cómo arrancar un contenedor en modo foreground y en modo detached y la diferencia entre ellos?*
  - *¿Recuerdas algunos comandos básicos para interactuar con el daemon de Docker?*

¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

?

Redes de Docker

- *¿Entiendes que por defecto todos los contenedores tienen una tarjeta de red que les comunica con Internet y el rango de IPs por defecto de esa red?*
- *¿Sabes cómo exponer un puerto de un contenedor al exterior?*
- *¿Comprendes que puedes crear redes nuevas entre contenedores y añadirlas a ellos según te convenga?*
- *¿Sabes que por defecto todos los contenedores que crees se pueden ver entre ellos, pero que puedes prohibir este comportamiento?*
- *¿Entiendes qué es y para qué sirve enlazar contenedores?*

Dockerfiles

- *¿Entiendes el concepto de Dockerfile y para qué sirve?*
- *¿Comprendes como es el proceso básico de construcción de una Dockerfile?*
- *¿Recuerdas cómo construir una imagen a partir de una Dockerfile que especifique su contenido?*
- *¿Recuerdas cuáles son los trucos principales para minimizar el tamaño de las imágenes que uses?*
- *¿Entiendes el proceso de construcción enlazada de imágenes?*

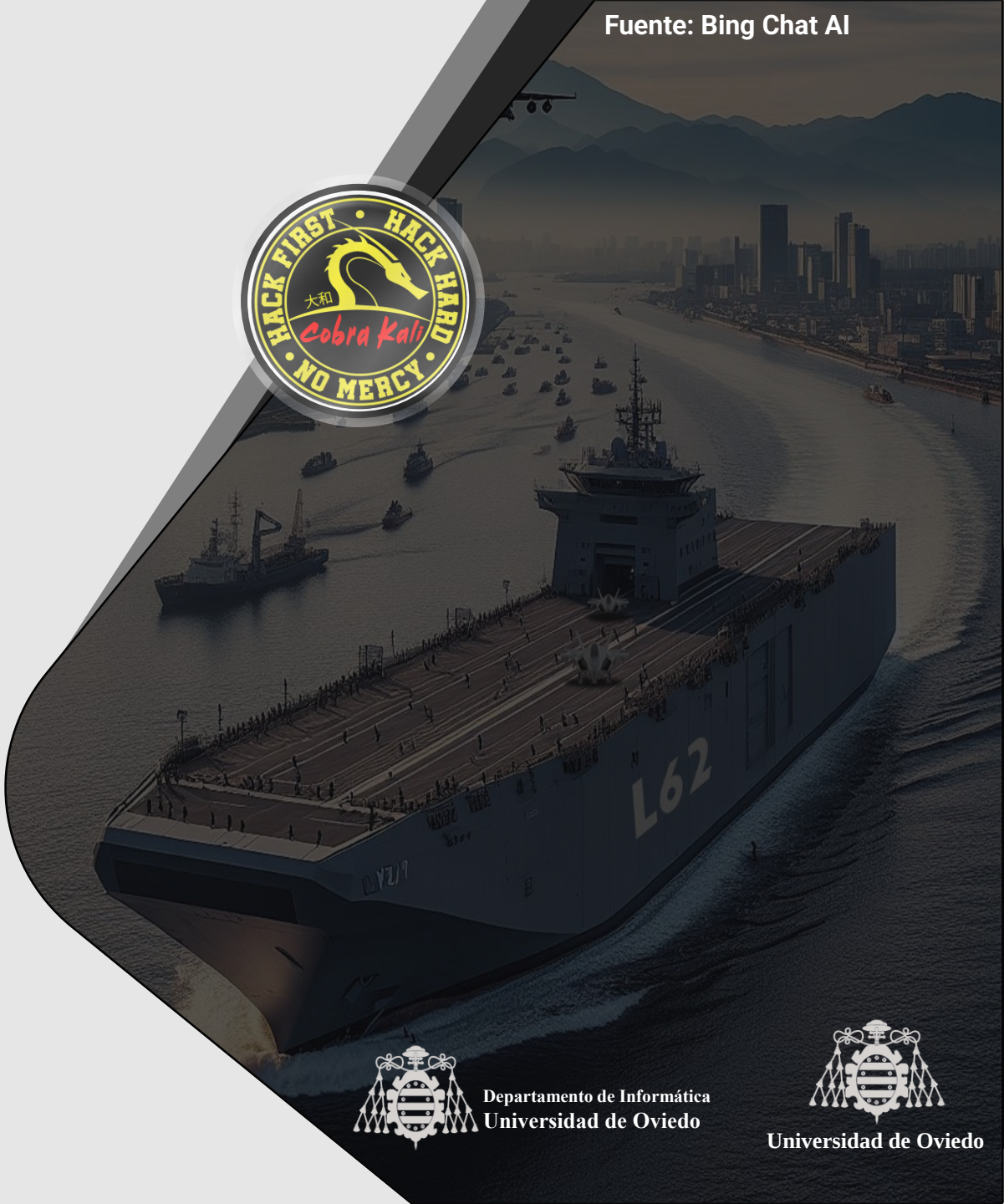
Volúmenes

- *¿Sabes para qué sirven los volúmenes y cómo pueden solventar que los contenedores sean efímeros?*
- *¿Entiendes que los volúmenes pueden añadirse y quitarse de los contenedores a voluntad?*

< Ir al Índice

Infraestructuras con Docker Compose

Una forma de crear infraestructuras de contenedores



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿Qué vamos a ver en este apartado?

José Manuel
Redondo López

- **Este apartado te mostrará como construir infraestructuras con contenedores Docker usando Docker Compose, lo que implica**
  - Saber qué es y **cómo permite construir** infraestructuras de contenedores
  - Poder diferenciar entre **contenedores** usados y **servicios** ofertados
  - Conocer **ejemplos operativos** de cómo construir infraestructuras de contenedores típicas

Docker Compose

José Manuel
Redondo López

- **Es una herramienta para definir y ejecutar aplicaciones cuyos elementos residen en varios contenedores**
- **Usa un fichero YAML (`docker-compose.yml`) para configurar los servicios de una aplicación**
  - Luego con un solo comando permite arrancar todos esos servicios al mismo tiempo y poner la aplicación en marcha
  - Esto incluye todos sus contenedores con funciones especializadas (servidor web, BBDD) y las comunicaciones entre ellos (redes)
- **Usable en cualquier etapa de un ciclo de integración continua, desde desarrollo a producción**
- **Compatible con contenedores Docker y también con Podman (no la función Swarm que mencionaremos después)**
  - https://www.redhat.com/sysadmin/podman-docker-compose

Docker Compose

José Manuel
Redondo López

- Usar **Docker Compose** requiere dar tres pasos
  - **Definir los contenedores** que van a formar parte de la aplicación en sus **Dockerfiles** correspondientes y construirlos previamente
    - Podemos también usar la opción `build: <ruta de una Dockerfile>` para que se construyan "al vuelo"
    - Cuando **Docker Compose** procese el fichero `.yml` con la infraestructura
  - **Definir los servicios** que van a formar parte de la aplicación a partir de dichos contenedores
  - Todo en un mismo fichero `docker-compose.yml`
    - Esto hace que todos estos contenedores se ejecuten de forma conjunta en un entorno aislado
  - Ejecutar `docker-compose up` para arrancar toda la infraestructura de la aplicación
    - Ejemplo: `sudo docker-compose up --build`
    - El arranque de la infraestructura tardará un tiempo y **al final la pantalla quedará parada**
    - Si abortamos ese proceso, **destruiremos todos los contenedores** de la infraestructura creada
    - Para hacer limpieza correctamente de los recursos usados por **Docker Compose** es mejor completar la instrucción anterior así
      - `sudo docker-compose up --build && docker-compose rm -fsv`

Docker Compose

José Manuel
Redondo López

- **Para la construcción de infraestructuras de contenedores con Docker Compose, necesitamos que todas las imágenes de sus contenedores estén construidas**
- **Si usamos una aproximación como la anterior (imagen base, imágenes derivadas) implica que 1º debemos construir también todas las imágenes base que usemos**
  - En ese caso poner `build: <ruta de una Dockerfile>` en el fichero `.yml` puede ser insuficiente
  - Pero si no las tenemos **construidas de antemano**, podemos automatizarlo fácilmente con un **script** que llame a todos los `docker build` necesarios en el orden correcto
  - Si una imagen ya se ha construido previamente, **no volverá a construirse** salvo que hagamos cambios en su **Dockerfile**
- **Si hemos prohibido la comunicación entre contenedores (`--icc=false`), tendremos que enlazar los contenedores creados adecuadamente**
  - Esto puede lograrse añadiendo a un servicio `links: <nombre o lista de nombres de contenedores de los servicios enlazados>`
  - **Más información**: https://docs.docker.com/compose/networking/

Ejemplos de Docker Compose

José Manuel
Redondo López

- **Crea contenedores de imágenes Docker ya construidas** (`image`)
  - Con el nombre que indiquemos (`container_name`)
- **Reinicia los contenedores en caso de fallo** (`restart: on-failure`)
- **En el caso del Kali, admite consola interactiva** (`stdin_open` y `tty`)
- **Crea un volumen en cada contenedor**
  - Mapea una carpeta del host (subdirectorio `/volume_data/<nombre del SO>` de la carpeta actual) a una carpeta del contenedor
  - Todo lo que escribamos desde el host se verá automáticamente en cada contenedor y viceversa
  - Supone una forma muy sencilla de introducir o sacar contenidos de los contenedores

```yaml
version: "2"
services:
    ubuntu_crypto:
        image: ssi/ubuntu_ssh_crypto
        container_name: ubuntu_crypto
        restart: on-failure
        volumes:
            - ./volume_data/ubuntu:/home/ssiuser/shared
    kali_crypto:
        image: ssi/kali_for_crypto
        container_name: kali_crypto
        restart: on-failure
        stdin_open: true # docker run -i
        tty: true        # docker run -t
        volumes:
            - ./volume_data/kali:/root/shared
```

Fichero `docker-compose1.yml`

Ejemplos de Docker Compose

José Manuel
Redondo López

- **Esta variante del ejemplo anterior conecta dos contenedores distintos de una misma imagen con un tercero**
  - De nuevo usando una red local interna de Docker
  - Los dos 1º tienen opción de ser interactivos
- **Se mapean dos directorios distintos en cada contenedor**
  - ¡No hay limite a la hora de mapear directorios!

```yaml
version: "2"
services:
    network_attack_1:
        image: ssi/ubuntu_ssh_network_attack
        container_name: lab07_network_attack_1
        hostname: lab07_network_attack_1
        stdin_open: true # docker run -i
        tty: true        # docker run -t
        restart: on-failure
        volumes:
            - ./volume_data/network_attack_1:/shared
            - ./volume_data/network_attack_1/sysctl.d/:/etc/sysctl.d

    network_attack_2:
        image: ssi/ubuntu_ssh_network_attack
        container_name: lab07_network_attack_2
        hostname: lab07_network_attack_2
        stdin_open: true # docker run -i
        tty: true        # docker run -t
        restart: on-failure
        volumes:
            - ./volume_data/network_attack_2:/shared
            - ./volume_data/network_attack_2/sysctl.d/:/etc/sysctl.d

    vpn_client:
        image: ssi/ubuntu_ssh_vpn_client
        container_name: lab07_vpn_client
        hostname: lab07_vpn_client
        stdin_open: true # docker run -i
        tty: true        # docker run -t
        restart: on-failure
        privileged: true
        volumes:
            - ./volume_data/vpn_client:/shared
            - ./volume_data/vpn_client/wireguard:/etc/wireguard
```

Fichero `docker-compose2.yml`

Ejemplos de Docker Compose

José Manuel
Redondo López

- Contenedores unidos por una red definida por el usuario (`networks`)
- Se define una red `ssi_net` que conecta dos servidores web (`eii` y `epi`) con un proxy
- Además, otra red (`ssi_front_net`) conecta un **Kali** con un proxy
  - Kali y Ubuntus **no se ven entre sí**: los separa el proxy
- Cada red tiene un rango de direcciones IP distinto
- **Kali** tiene además la red `default` para permitir conectarse a Internet
  - Ej.: para actualizaciones

```yaml
version: "2"
services:
    eii:
        image: ssi/ubuntu_apache2_ssh
        container_name: lab10_web_eii
        hostname: lab10_web_eii
        restart: on-failure
        volumes:
            - ../../webs/web_eii:/var/www/html
        networks:
            lab10_back_net:
                ipv4_address: 172.10.0.3
    epi:
        image: ssi/ubuntu_apache2_ssh
        container_name: lab10_web_epi
        hostname: lab10_web_epi
        restart: on-failure
        volumes:
            - ../../webs/web_epi:/var/www/html
        networks:
            lab10_back_net:
                ipv4_address: 172.10.0.4
    proxy:
        image: ssi/ubuntu_apache2_ssh_proxy
        container_name: lab10_reverse_proxy
        hostname: lab10_reverse_proxy
        restart: on-failure
        volumes:
            - ./volume_data/ssh:/shared
            - ./volume_data/rules:/rules/
        networks:
            lab10_back_net:
                ipv4_address: 172.10.0.2
            lab10_front_net:
                ipv4_address: 198.102.0.3
```

```yaml
    kali:
        image: ssi/kali_nids
        container_name: lab10_kali
        hostname: lab10_kali
        restart: on-failure
        stdin_open: true # docker run -i
        tty: true        # docker run -t
        volumes:
            - ./volume_data/kali:/shared
        networks:
            default:
            lab10_front_net:
                ipv4_address: 198.102.0.2

networks:
    lab10_back_net:
        driver: bridge
        ipam:
            config:
                - subnet: 172.10.0.0/16
    lab10_front_net:
        driver: bridge
        ipam:
            config:
                - subnet: 198.102.0.0/16
```

Fichero `docker-compose3.yml`

Ejemplos de Docker Compose

José Manuel
Redondo López

- **Este último ejemplo conecta cuatro contenedores servidor web con un DNS, todo en una red definida manualmente**
  - Solo el contenedor Kali tiene salida a internet (red `default`)
  - Cada servidor web:
    - Apunta su DNS a nuestro contenedor `dns`
    - Son contenedores de la misma imagen a los que se le ha mapeado una web diferente en su `/var/www/html`
    - Esto ahorra muchos recursos respecto a crear un contenedor distinto para cada web

Ejemplos de Docker Compose

José Manuel
Redondo López

```yaml
version: "2"
services:
    dns:
        image: ssi/ubuntu_dns
        container_name: lab2_dns_server
        hostname: lab2_dns_server
        restart: on-failure
        volumes:
            - ./dns_conf:/dns_conf
        networks:
            lab02_net:
                ipv4_address: 172.2.0.2
    eii:
        image: ssi/ubuntu_apache2_ssh
        container_name: lab2_web_eii
        hostname: lab2_web_eii
        restart: on-failure
        dns: 172.2.0.2
        volumes:
            - ../../webs/web_eii:/var/www/html
        networks:
            lab02_net:
                ipv4_address: 172.2.0.3
    epi:
        image: ssi/ubuntu_apache2_ssh
        container_name: lab2_web_epi
        hostname: lab2_web_epi
        restart: on-failure
        dns: 172.2.0.2
        volumes:
            - ../../webs/web_epi:/var/www/html
        networks:
            lab02_net:
                ipv4_address: 172.2.0.4
    uniovi:
        image: ssi/ubuntu_apache2_ssh
        container_name: lab2_web_uniovi
        hostname: lab2_web_uniovi
        restart: on-failure
        dns: 172.2.0.2
        volumes:
            - ../../webs/web_uniovi:/var/www/html
        networks:
            lab02_net:
                ipv4_address: 172.2.0.5
    asturias:
        image: ssi/ubuntu_apache2_ssh
        container_name: lab2_web_asturias
        hostname: lab2_web_asturias
        restart: on-failure
        dns: 172.2.0.2
        volumes:
            - ../../webs/web_asturias_es:/var/www/html
        networks:
            lab02_net:
                ipv4_address: 172.2.0.6
    kali:
        image: ssi/kali_for_enumeration
        container_name: lab2_kali
        hostname: lab2_kali
        restart: on-failure
        dns: 172.2.0.2
        stdin_open: true # docker run -i
        tty: true        # docker run -t
        environment:
            - USER_NAME
        volumes:
            - ./volume_data/kali:/shared
        networks:
            default:
            lab02_net:
                ipv4_address: 172.2.0.7
networks:
    lab02_net:
        driver: bridge
        ipam:
            config:
                - subnet: 172.2.0.0/16
```

Fichero `docker-compose4.yml`

ARRANQUE DE CONTENEDORES ORDENADO

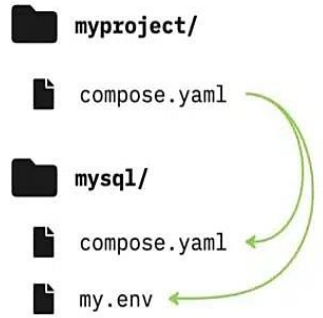
José Manuel
Redondo López

- Cuando se tienen muchos contenedores, a veces hay que **controlar cómo se arrancan**
- **Docker Compose** por defecto no espera a que un contenedor esté listo, sólo a que arranque
- Esto puede presentar problemas de sincronización
  - Ej.: La web arranca antes de la base de datos
- Pero se pueden controlar las dependencias con `depends_on`
  - Podemos hacer esperar un contenedor a que otro tenga su servicio arrancado, responda a un health check, etc.
  - https://docs.docker.com/compose/startup-order/

```yaml
include:
  - path: ../mysql/compose.yaml
    env_file: ../mysql/my.env

services:
  backend:
    build:
      context: backend
      target: builder
    secrets:
      - db-password
    depends_on:
      db:
        condition: service_healthy
```

from mysql/compose.yaml



El backend no arranca hasta que el contenedor de la BD arranque y de señales de vida (Docker hace health check de los contenedores periódico para saber su estado).
Fuente: https://www.docker.com/blog/docker-desktop-4-22/

¿CREES QUE LO HAS ENTENDIDO TODO? ¡AUTOEVALÚATE!

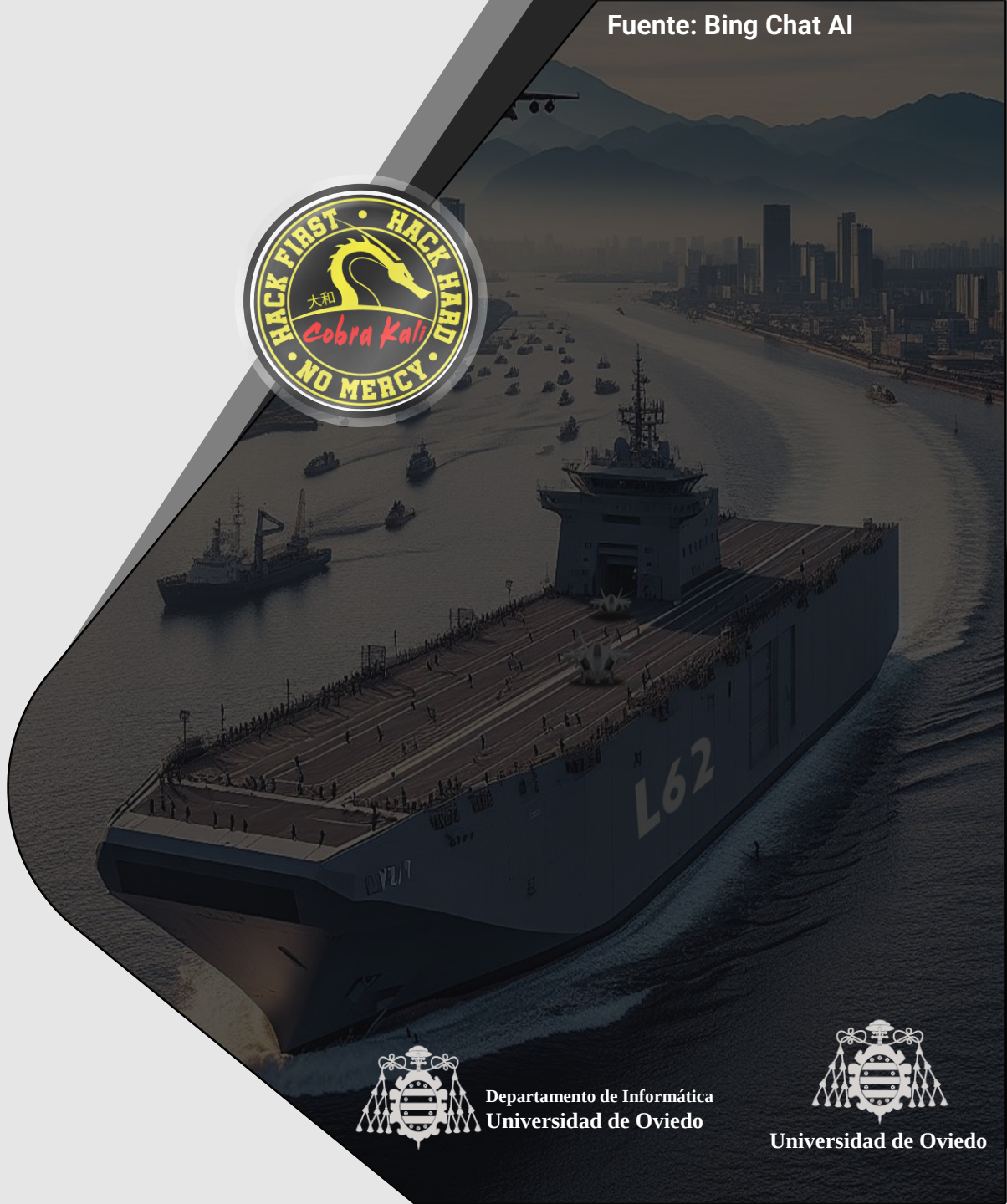
José Manuel
Redondo López

- *¿Entiendes para qué sirve **Docker Compose** y que las infraestructuras se construyen a través de ficheros YAML?*
- *¿Entiendes que para que una infraestructura **Docker Compose** funcione los contenedores que forman parte de ella deben estar ya construidos y como puedes intentar construirlos en la creación?*
- *¿Comprendes que **Docker Compose** se basa en la definición de servicios a partir de estos contenedores?*
- *¿Sabes hacer que un servicio se reinicie automáticamente en caso de fallo?*
- *¿Sabes cómo habilitar la consola interactiva en un contenedor de la infraestructura?*
- *¿Sabes cómo añadir un volumen a un contenedor de la infraestructura?*
- *¿Sabes cómo hacer infraestructuras que conecten contenedores entre ellos con redes independientes?*

< Ir al Índice

Vagrant y Docker

Uniendo ambas tecnologías



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Usando Vagrant con Docker

José Manuel
Redondo López

- **Si queremos hacer funcionar Vagrant con Docker, uniendo así los dos temas anteriores, necesitamos varios elementos:**
  - Una `Vagrantfile` con una serie de opciones particulares para configurar Docker
    - https://www.vagrantup.com/docs/docker/configuration.html
  - Y una `Dockerfile` con una serie de comandos Docker que configuren el contenedor a construir
    - Como los ya vistos
  - En este caso la `Vagrantfile` principalmente dice **dónde está la** `Dockerfile` **a usar**
  - Tened en cuenta que un contenedor Docker muere salvo que haya un proceso en primer plano que se siga ejecutando
    - Por tanto, seguimos teniendo que especificar **algo que ejecutar** que cumpla estas características
  - En caso contrario, Vagrant dará un error

Usando Vagrant con Docker

José Manuel
Redondo López

- En la `Vagrantfile` especificamos que la `Dockerfile` a usar está en el mismo directorio (“`.`”)
- Además, avisamos a **Vagrant** de que la máquina seguirá ejecutándose tras arrancarla
  - `remains_running = true`
- La `Dockerfile` instala varios paquetes
  - Y pone **Apache** a funcionar en 1er plano para que el contenedor no muera
- Ahora ya podemos hacer peticiones web a la IP asignada por **Docker**
  - En la `Dockerfile` podemos abrir puertos, crear ficheros, etc. según necesitemos

```
 GNU nano 2.5.3                    Archivo: Vagrantfile

Vagrant.configure("2") do |config|
        config.vm.provider "docker" do |d|
                d.build_dir = "."
                d.remains_running = true
        end
end
```

```
 GNU nano 2.5.3                    Archivo: Dockerfile


FROM ubuntu:14.04
RUN apt-get update
RUN apt-get install -y apache2
RUN service apache2 start
RUN apt-get install -y openssh-server

RUN /usr/sbin/apache2ctl -D FOREGROUND
```

Usando Vagrant con Docker

José Manuel
Redondo López

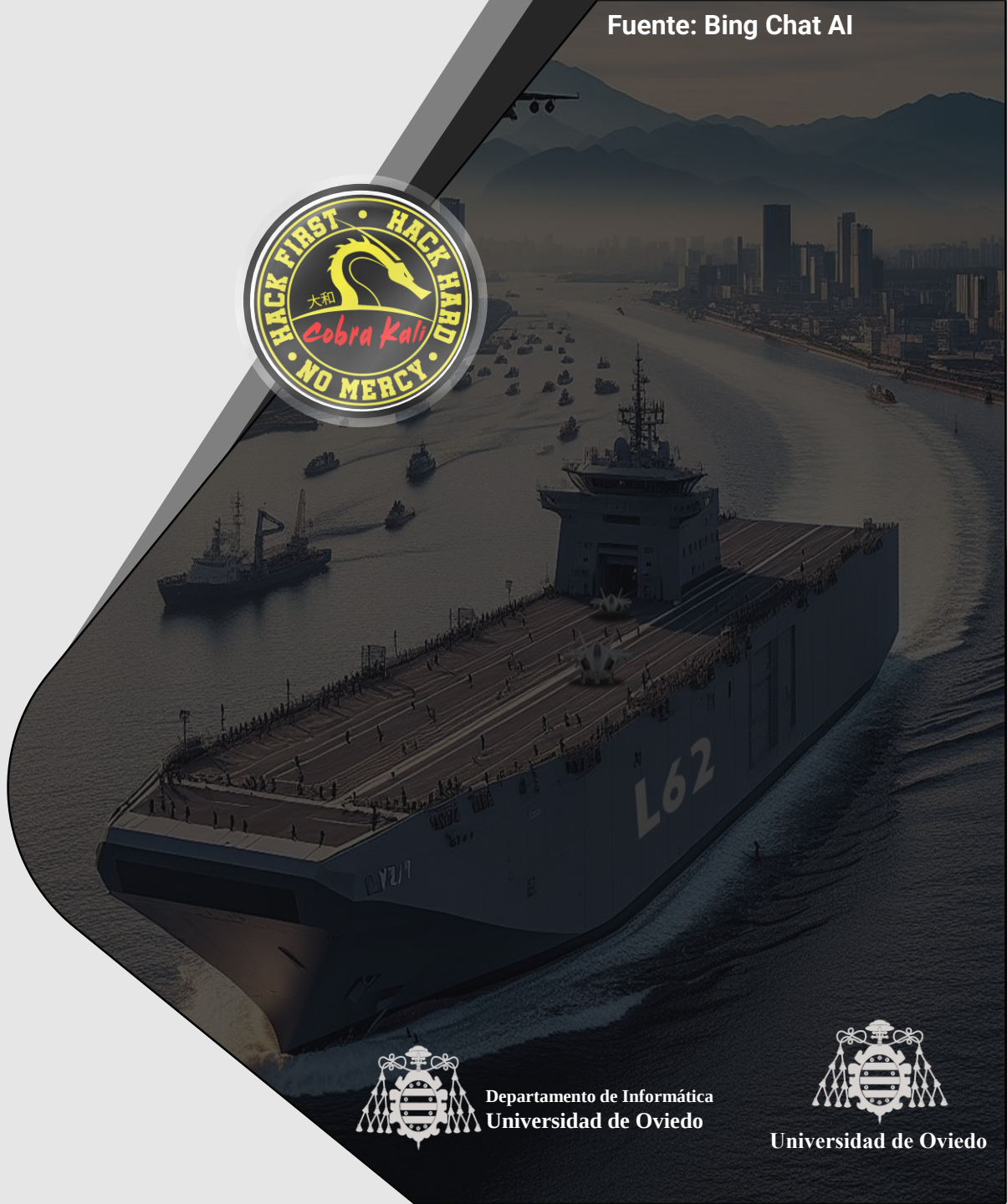
- El problema es que, aunque funciona, **Vagrant** no cambia el estado a "`running`" (se queda en "`preparing`")
  - Por tanto, el comando `vagrant ssh` no funcionará
  - Pero si podemos usar un **shell** con `docker exec`, como vimos
- Si al **daemon** `docker` no se le ha configurado en la instalación un usuario autorizado para usarlo, **debemos usar** `sudo` **con Vagrant**

```
operario@ubuntuserver:~/test3$ sudo vagrant up --provider=docker
Bringing machine 'default' up with 'docker' provider...
==> default: Building the container from a Dockerfile...
    default: Sending build context to Docker daemon 8.192 kB
    default: Step 1 : FROM ubuntu:14.04
    default:  ---> 4a725d3b3b1c
    default: Step 2 : RUN apt-get update
    default:  ---> Using cache
    default:  ---> 67b6136fcea5
    default: Step 3 : RUN apt-get install -y apache2
    default:  ---> Using cache
    default:  ---> 9349a08e6ba7
    default: Step 4 : RUN /usr/sbin/apache2ctl -D FOREGROUND
    default:  ---> Running in 91c20c721eb8
    default: AH00558: apache2: Could not reliably determine the server's fully qualified domain name
, using 172.17.0.2. Set the 'ServerName' directive globally to suppress this message
    default:
```

< Ir al Índice

ÚLTIMAS CONSIDERACIONES

Otros conceptos y utilidades interesantes para trabajar con Docker



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Cifrado de comunicaciones en Docker

José Manuel
Redondo López

- **La conexión entre el cliente y el daemon en los ejemplos que hemos visto ocurre dentro del mismo host**
  - Es un socket Unix, cuyo tráfico no se emite por red
- **Sin embargo, si el cliente y el daemon están en máquinas separadas, se comunican por HTTP**
  - En este caso, por temas de seguridad podemos necesitar que el tráfico entre ambos elementos **esté cifrado**
  - Sobre todo, si el tráfico va a enviarse por una red pública
  - Se puede **implementar comunicación cifrada**
    - https://docs.docker.com/engine/security/https/
    - El enlace usa una autoridad certificadora propia con certificados auto-firmados
    - No es lo ideal para un entorno de producción, pero si para pruebas

Imágenes Distroless

José Manuel
Redondo López

- **Nuevamente vamos a hablar de formas de crear contenedores de tamaño mínimo por seguridad y eficiencia**
  - De nuevo una forma de hacerlo es crear contenedores de tipo Alpine Linux para instalar encima lo que necesitemos
    - Estos contenedores **usan el shell** /bin/sh y no el bash
  - Pero en este caso tenemos una segunda opción: un contenedor a partir de una **imagen distroless** (https://github.com/distroless)
    - Son imágenes que **solo tienen una determinada aplicación y sus dependencias**, y nada más
    - Como se ve en la imagen, el tamño de un Nginx distroless es notablemente menor que el de la imagen Nginx oficial normal

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
distroless.dev/nginx	latest	17b961163b7d	7 hours ago	31.1MB
nginx	latest	b692a91e4e15	2 days ago	142MB
alpine	latest	d7d3d98c851f	2 weeks ago	5.53MB
<none>	<none>	bb7295985d77	2 weeks ago	221MB
portainer/portainer-ce	latest	f8f097584d69	3 weeks ago	278MB
ubuntu	latest	27941809078c	8 weeks ago	77.8MB

Multi-stage builds

José Manuel
Redondo López

- **Otra forma de minimizar el tamaño de una imagen es usar una multi-stage build**
  - En ellas dividimos la construcción de una imagen **en fases**
  - En las primeras, **construimos la aplicación**, compilándola y enlazándola e incluyendo en la imagen todo lo necesario para hacerlo
  - Cuando se termina, **el producto de la compilación** (ejecutable) se copia a otra imagen en la siguiente fase, que será la encargada de **desplegarla**
  - Esta segunda imagen ya no necesita (ni tiene) **ningún software necesario para compilar la imagen**,
    - ¡Por lo que tiene un tamaño mucho menor que si contuviera ambas cosas!
  - En otras palabras, **se separa el entorno de compilación y construcción del de ejecución**
    - En contenedores diferentes
    - Cuya ejecución se enlaza en una `Dockerfile`
    - Y así solo se ejecutaría el contenedor que tiene lo necesario para la ejecución, mucho más pequeño
- **Es un tema avanzado y muy situacional para desarrolladores**
  - Aquí pueden verse ejemplos fáciles de entender
  - https://docs.docker.com/develop/develop-images/multistage-build/

Docker Swarm

José Manuel
Redondo López

- **Es una extensión del API de Docker para tratar clúster de contenedores de la misma forma que un sólo contenedor**
  - Se ha demostrado que sus funcionalidades son insuficientes para cubrir determinados casos de uso (y habría que usar otras soluciones como Kubernetes)
- **Se basa en convertir un grupo de Docker engines en un solo Docker engine virtual**
- **El interfaz de comandos de Docker se puede usar para ejecutar contenedores, localizarlos, listar los disponibles, quitarlos, etc.**
  - Para ello Docker se configura en modo Swarm
  - Excede los objetivos de esta asignatura, pero lo veréis en servicios web
- **Tutorial: https://docs.docker.com/engine/swarm/swarm-tutorial/**

¿Y en Windows?

José Manuel
Redondo López

- **El soporte para contenedores de aplicaciones containerd puede instalarse en Windows**
  - El manejo de contenedores en línea de comandos es idéntico al visto
- **Microsoft también proporciona contenedores Windows con instalaciones mínimas para usarlos como hemos visto**
  - Windows Nano Server, sin GUI
  - Y desprovisto de la mayoría de los servicios de sus "hermanos mayores"
- **Es necesario un host Windows al que hay que preparar adecuadamente antes**
  - https://docs.microsoft.com/es-es/virtualization/windowscontainers/quick-start/set-up-environment
- **Hecho esto, poner en marcha un contenedor Windows es muy sencillo**
  - https://docs.microsoft.com/es-es/virtualization/windowscontainers/quick-start/run-your-first-container

Ejemplos de contenedores ya creados

José Manuel
Redondo López

- Dada su popularidad, hay una gran cantidad de ejemplos de contenedores ya hechos que podemos usar
- En este enlace hay gran cantidad de ejemplos de cómo ejecutar varias aplicaciones con **Docker**
  - Con aislamiento y control de consumo de recursos
  - https://blog.jessfraz.com/post/docker-containers-on-the-desktop/
- Ejemplos de **Dockerfiles** de uso común
  - https://github.com/kstaken/dockerfile-examples
- Repositorio **GitHub** con gran cantidad de **Dockerfiles** ya creadas para trabajar con múltiples aplicaciones
  - https://github.com/jessfraz/dockerfiles

Ejemplos de contenedores ya creados relativos a la asignatura

José Manuel
Redondo López

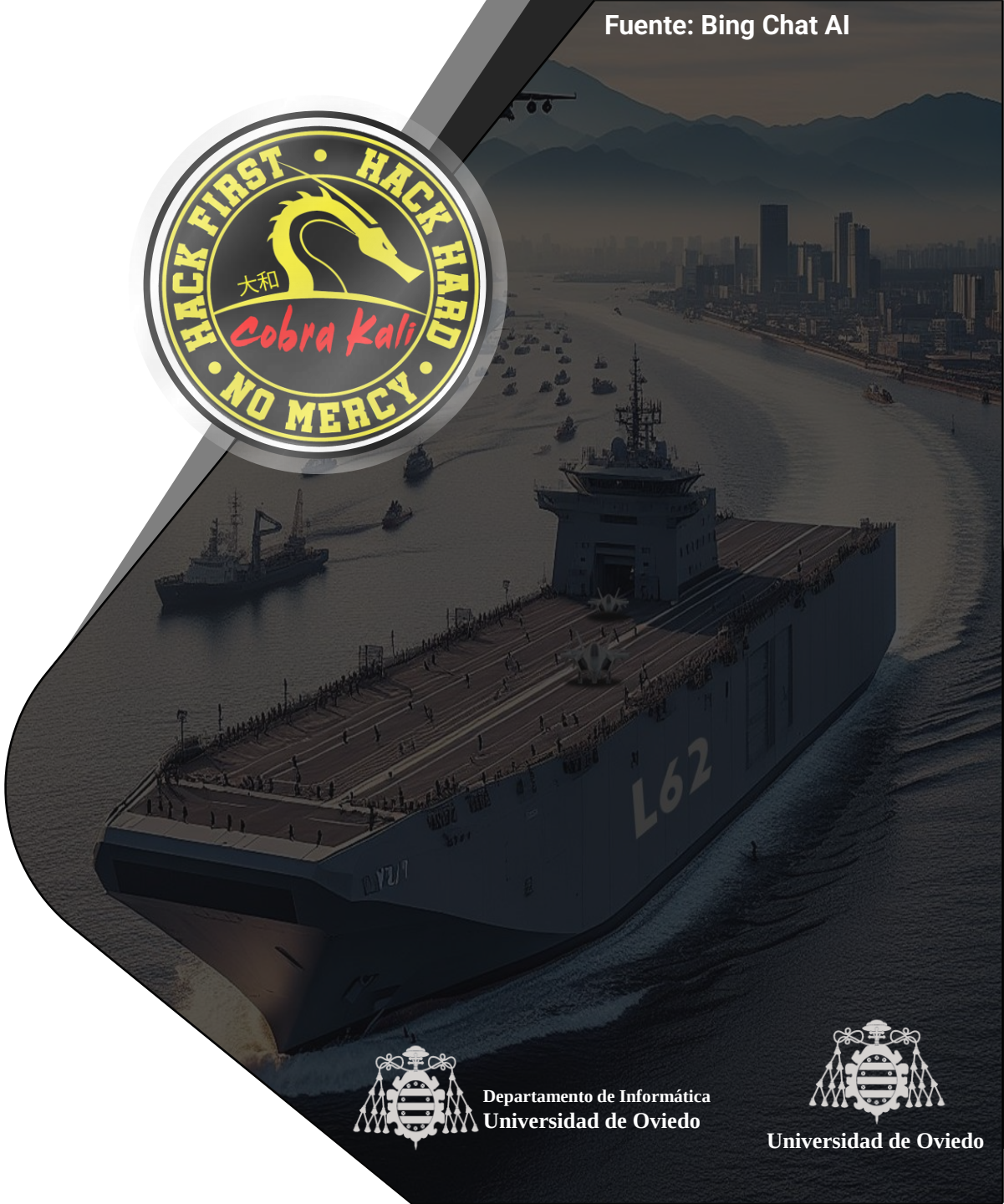
- **Todas las tecnologías que vimos en el primer tema se pueden encontrar en contenedores de sus proveedores oficiales en el Docker Hub**
  - ¡El instalar se va a acabar! ☺
  - Os lo cuento por si vuestras futuras empresas no están actualizadas 😉
- **Ejemplos**
  - **Clústeres Tomcat**: https://hub.docker.com/_/tomcat
    - Ejemplos de implementación (Ej.: “Arquitecturas y Diseño de Sitios Web”)
      - https://westzq1.github.io/docker/2019/06/13/tomcat-cluster-in-docker.html
      - https://mpolinowski.github.io/docs/DevOps/Tomcat/2020-12-25-tomcat10-docker-cluster/2020-12-25/
  - **Memcached**: https://hub.docker.com/_/memcached
  - **RabbitMQ**: https://hub.docker.com/_/rabbitmq
  - **Apache Kafka**: https://hub.docker.com/r/bitnami/kafka

Bibliografía y Referencias

José Manuel
Redondo López

- **Información general**
  - https://www.adictosaltrabajo.com/tutoriales/docker-for-dummies/
  - https://medium.com/better-programming/an-overview-to-docker-architecture-15407c482c52
  - https://docs.docker.com/v1.8/introduction/understanding-docker/
  - https://itnext.io/docker-from-the-beginning-part-i-ae809b84f89f
- **Información sobre el script pipework, que facilita el trabajo con redes en Docker**
  - https://opsbot.com/advanced-docker-networking-pipework/
  - https://github.com/jpetazzo/pipework
- **Información ampliada sobre redes en Docker**
  - https://docs.docker.com/engine/userguide/networking/
  - https://docs.docker.com/engine/userguide/networking/default_network/container-communication/
  - https://docs.docker.com/v1.5/articles/networking/
- **Ejecución de contenedores en Windows**
  - https://blog.sixeyed.com/how-to-dockerize-windows-applications/

INTRODUCCIÓN A TECNOLOGÍAS DE CONTENEDORES



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo