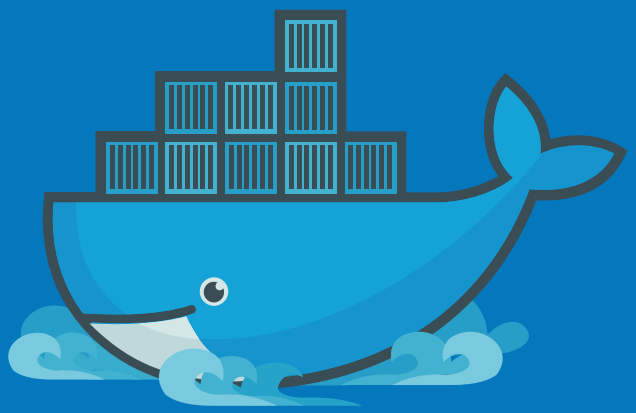
DE DESARROLLO A PRODUCCIÓN USANDO

DOCKER



AGENDA

- ¿Quiénes somos?
- Docker: introducción
- Consideraciones para trabajar con docker
- Volúmenes
- Docker Compose
- Docker en producción

¿QUIÉNES SOMOS?

LEANDRO DI TOMMASO

- Fundador de Mikroways.
- Docente en UNLP e Instructor CCNA.
- DevOp en CeSPI-UNLP.
- Miembro del NOC de UNLP entre 2007 y 2012.
- Miembro del NOC de INTA en 2012.

CHRISTIAN RODRIGUEZ

- Docente en UNLP
- Miembro del equipo de soporte CeSPI-UNLP hasta 2006
- Instructor CCNA/RedHat/Solaris/IRIX
- Coordinador del equipo de desarrollo de software interno (UNLP) perteneciente a CeSPI
- Aplicando DevOps desde 2012
- Coordino el área de IT para los desarrollos propios

¿QUÉ HACEMOS EN NUESTRO TRABAJO?

- Desde fines de 2013, nos consolidamos como equipo de IT para el área de desarrollo.
- Aplicando DevOps gestionamos:
  - 58 VMs virtualizando con Proxmox y VMWare.
  - Ambientes automatizados con chef.
    - 67 aplicaciones en ambiente de producción.
    - 55 aplicaciones en ambiente de pruebas.
  - Monitoreo y backups contemplados en la automatización.
  - Ambientes idénticos en desarrollo y producción: basados en vagrant y chef.

¿QUÉ HACEMOS EN NUESTRO TRABAJO?

- En 2016 formalizamos a Mikroways como una sociedad.
  - Trabajamos con DevOps (Chef y Docker).
  - Monitoreo inteligente (Estadísticas y logs).
  - Consultoría.
  - Capacitaciones.
  - Cloud computing.
  - IoT.
- Partners de Chef, Docker y Amazon.

DOCKER

INTRODUCCIÓN

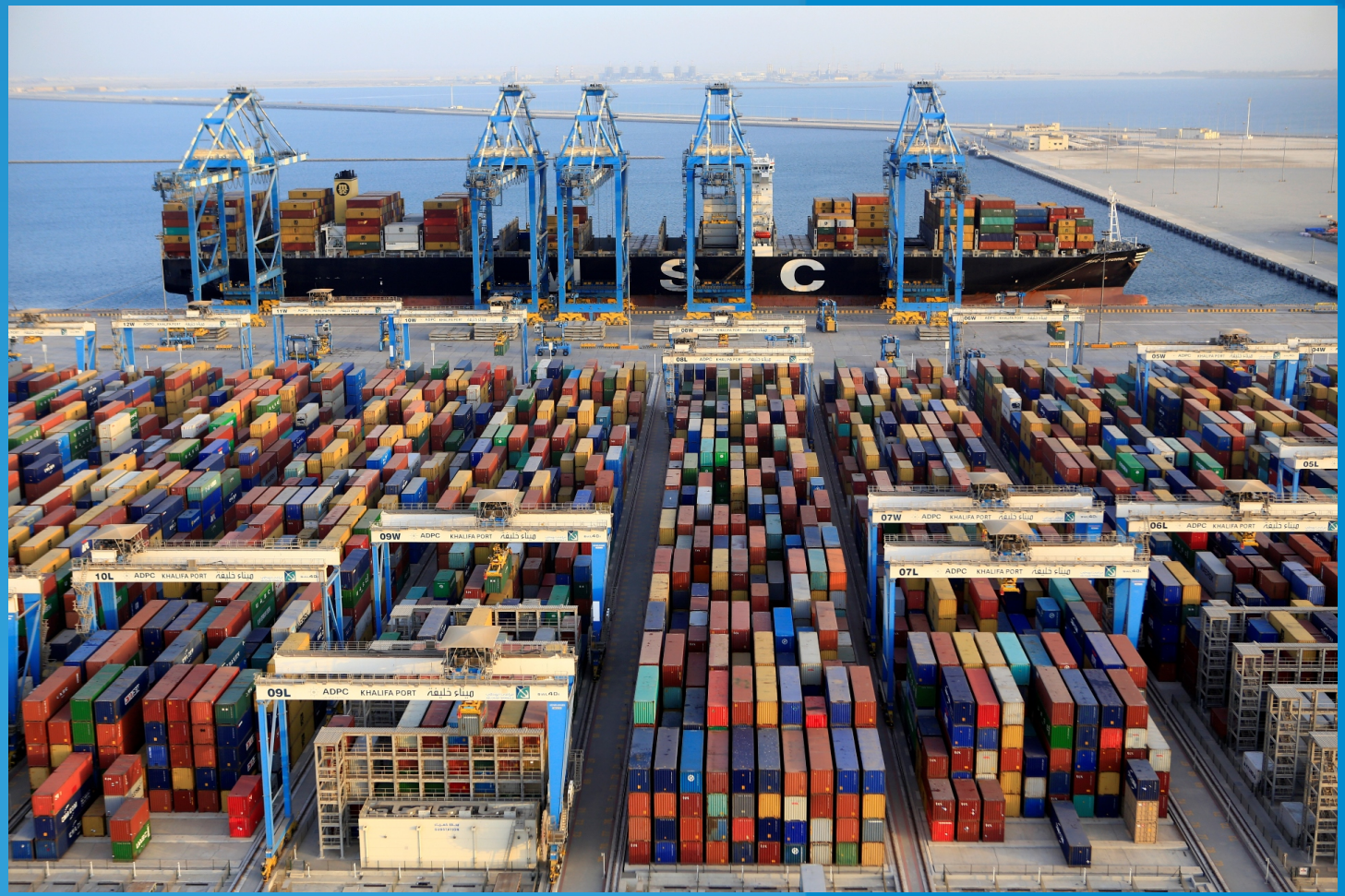
ANTECEDENTES

- Antiguamente, transportar bienes tenía muchos problemas.
  - Diferentes tamaños, formas, resistencias, etc.
  - Capacidad de transporte reducida.
  - Difícil realizar un seguimiento.
  - Pérdida parcial de mercadería.
  - Grandes costos.

CONTENEDORES

- Los contenedores solucionaron muchos problemas:
  - Un vendedor pone todos sus productos en un contenedor y sólo debe preocuparse por ese contenedor.
  - Los productos nunca se manipulan individualmente.
  - Tamaños y formas estandarizadas, simplifica toda la cadena de transporte: el transporte sólo debe llevar contenedores.

CONTENEDORES



¿QUÉ ES DOCKER?

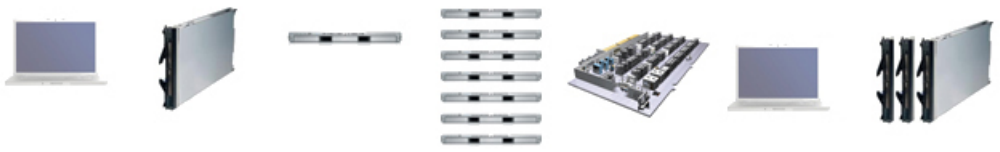
- Contenedores de software.
  - Empaqueta aplicaciones en una unidad estándar de intercambio.
- Única pieza de software en un filesystem completo que contiene **todo lo necesario** para ejecutar una aplicación: código, librerías, herramientas, etc.
- Garantiza que el software **siempre correrá de igual forma** sin importar su ambiente.

¿POR QUÉ DOCKER?

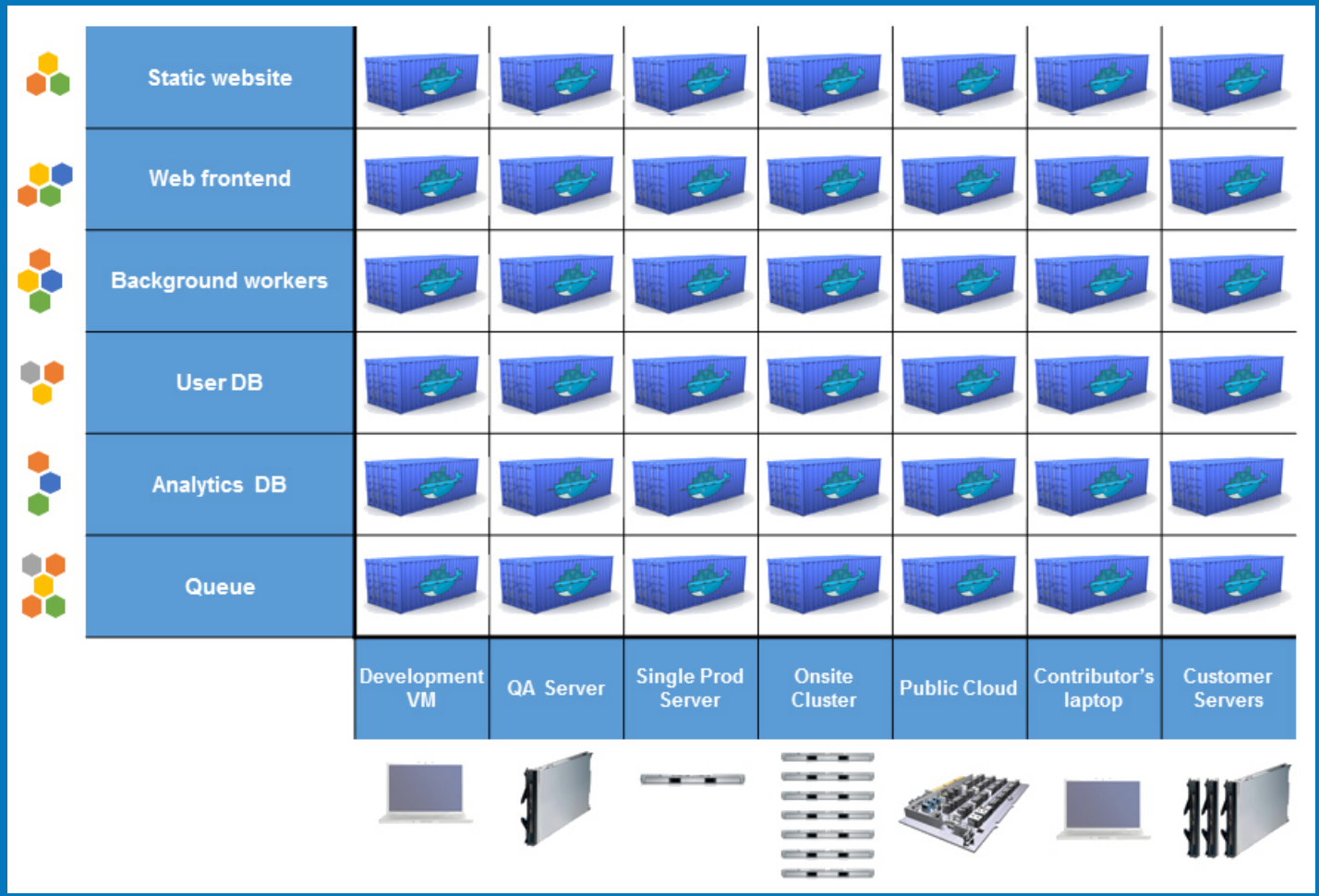
- Diferencias entre el ambiente de desarrollo, testing y producción.
- Instalación de una aplicación en diferentes plataformas.
- Deploy de aplicaciones complejas.
- Ejecución de código antiguo.
- Simplicidad para escalar horizontalmente.

MATRIZ DEL INFIERNO

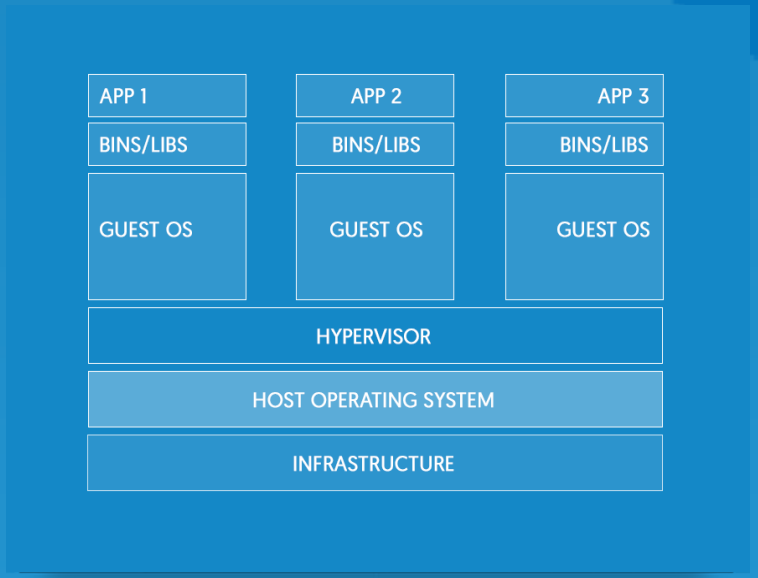
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

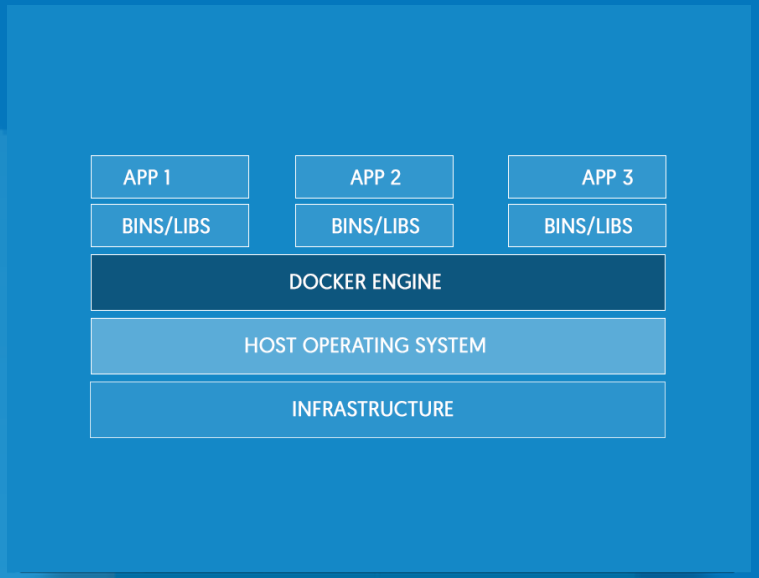


MATRIZ DEL INFIERNO



COMPARACIÓN CON MÁQUINAS VIRTUALES





HISTORIA

- Emerge como proyecto de SL en 2013.
- Virtualización a nivel de sistema operativo.
- Se basa en el uso de:
  - Cgroups para restringir recursos como cpu, memoria, IO, red, etc.
  - Kernel namespaces permite aislar y virtualizar recursos de una colección de procesos como por ejemplo: PID, hostname, UID, acceso a la red, comunicación entre procesos, filesystem, etc.
  - Filesystem de unión como es el caso de AUFS, OverlayFS, Btrfs, Device Mapper, ZFS, etc.

HISTORIA

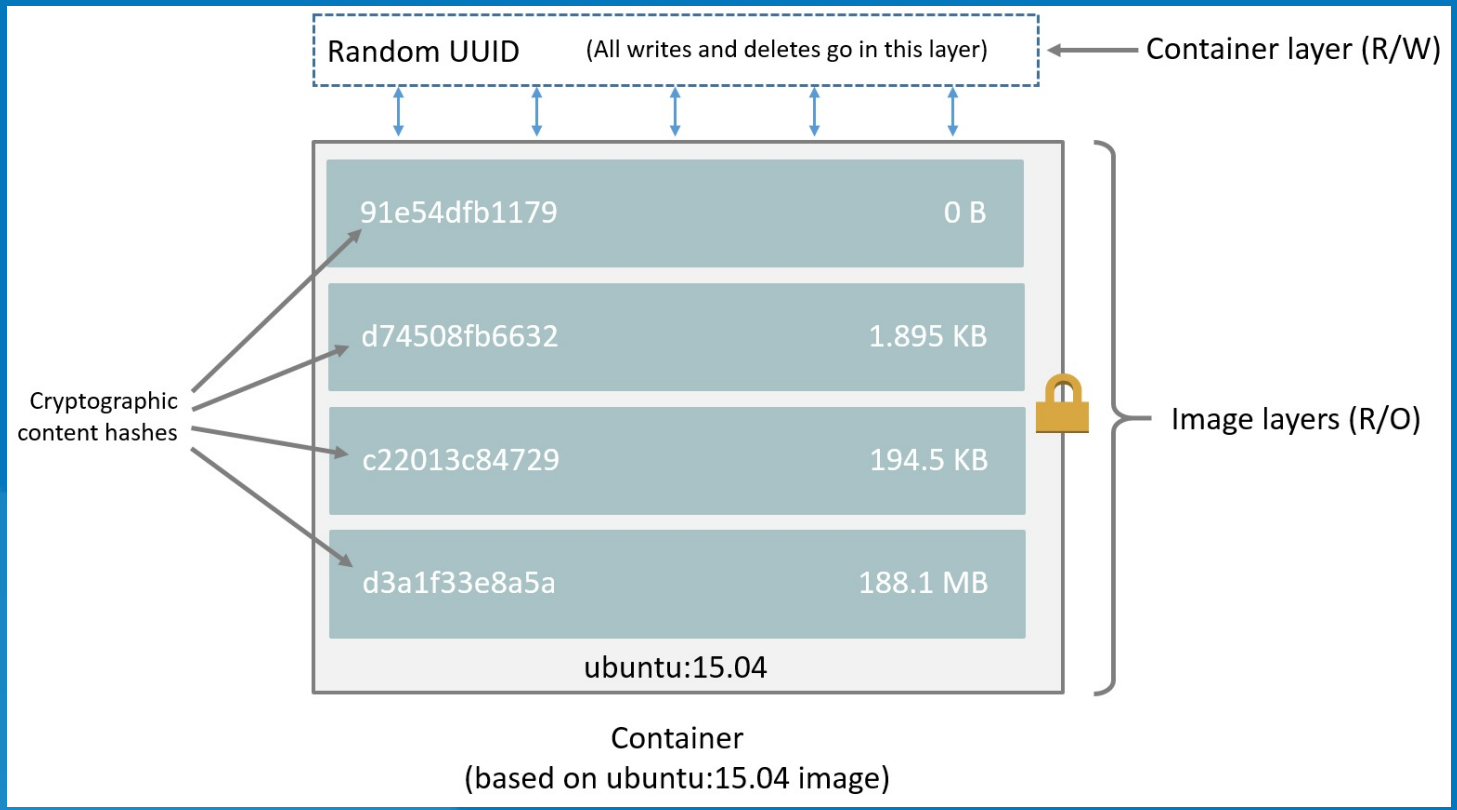
- Con las características antes mencionadas se obtienen contenedores independientes en una instancia Linux que evita el overhead de manipular VMs.
- Antes de la versión 0.9, Docker usaba LXC como base. A partir de la 0.9 incorporaron libcontainer, eliminando la dependencia de LXC dado que accede directamente al kernel para manipular cgroups, namespaces, apparmor, interfaces de red, etc.

IMÁGENES Y CONTENEDORES

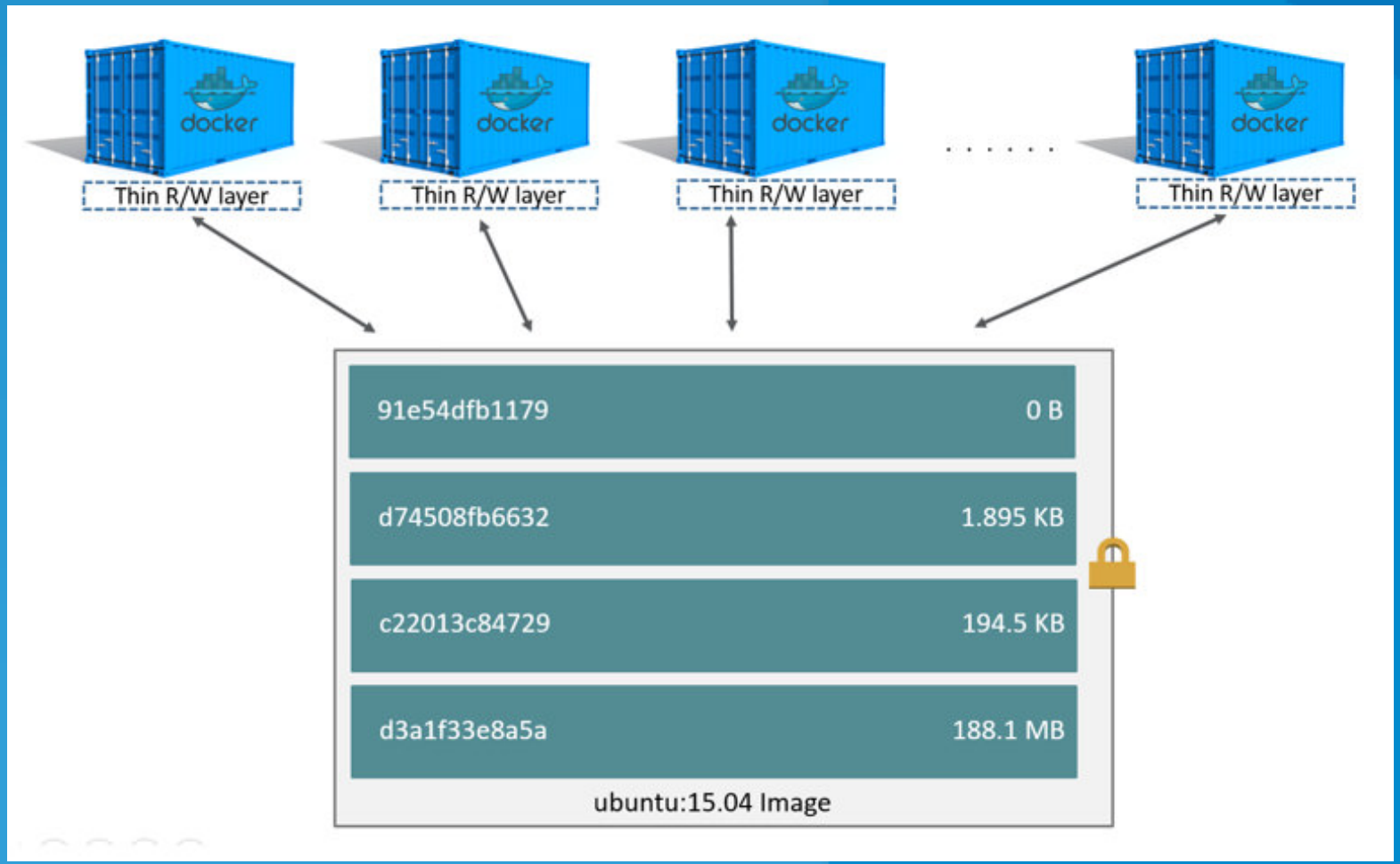
- Imagen:
  - Filesystem y parámetros para utilizarla.
  - No cambia nunca y no tiene estados.
- Contenedor:
  - Instancia de una imagen (resultado de ejecutarla).
  - Tiene una capa de RW volátil.

IMÁGENES Y CONTENEDORES

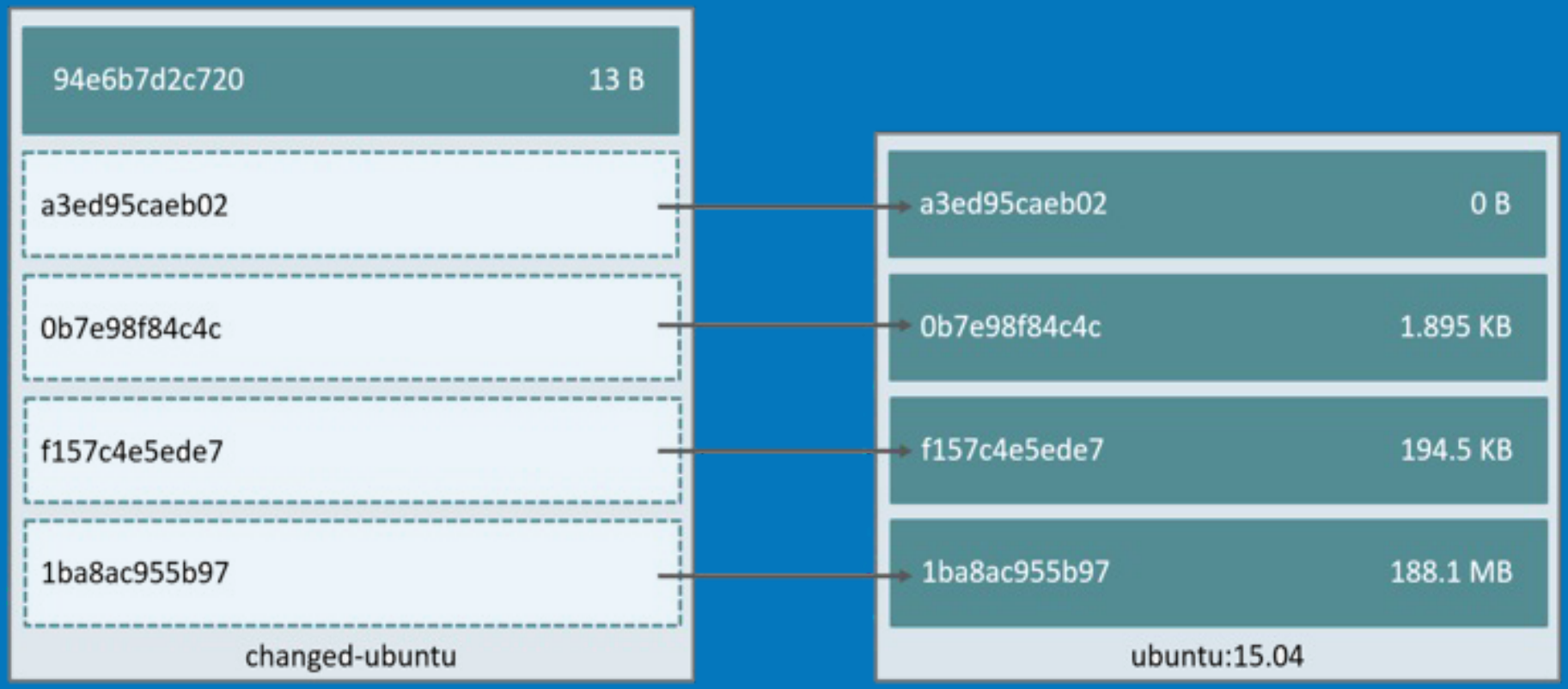
IMÁGENES Y CONTENEDORES



IMÁGENES Y CONTENEDORES



IMÁGENES Y CONTENEDORES



INSTALACIÓN DE DOCKER

- Docker puede instalarse en:
  - Linux.
  - MacOS.
  - Windows.

INSTALACIÓN DE DOCKER EN LINUX

- Requisitos:
  - Sistema de 64 bits.
  - Kernel 3.10 o superior.
- Existen binarios para la mayoría de las distribuciones.

INSTALACIÓN DE DOCKER EN WINDOWS/MACOS

- Usando Docker Toolbox.
  - Utiliza Docker Machine (no nativo).
  - Windows 7/MacOS 10.8 o superior
- Docker for (Windows/Mac):
  - Corre una aplicación nativa usando (Hyper-V/xhyve para virtualizar la Docker Engine).
  - Windows 10/MacOS 10.10.3 o superior.

COMANDOS BÁSICOS

```
# Más usados
docker run
docker ps
docker build
docker images
docker logs
docker inspect
docker volume

# Otros comandos comunes
docker commit
docker pull
docker push
docker tag
```

NUESTRO PRIMER CONTENEDOR

```
$ docker run -it ubuntu:16.04 /bin/bash
  Unable to find image 'ubuntu:16.04' locally
  16.04: Pulling from library/ubuntu

  6bbedd9b76a4: Pull complete
  fc19d60a83f1: Pull complete
  de413bb911fd: Pull complete
  2879a7ad3144: Pull complete
  668604fde02e: Pull complete
  Digest: sha256:2d44ae143feeb36f4c898d32ed2ab2dffeb3a573d2d8928646dfc9c
  Status: Downloaded newer image for ubuntu:16.04

  root@99a3403db59a:/# cat /etc/issue
  Ubuntu 16.04.1 LTS \n \l
```

DOCKERFILE

- Archivo de texto plano para crear imágenes de Docker.
- Permite escribir instrucciones a ejecutar.
- Automatiza el proceso de la creación de imágenes.
- Permite repetir y modificar fácilmente una imagen.
- Generar de forma simple imágenes derivadas.

DOCKERFILE

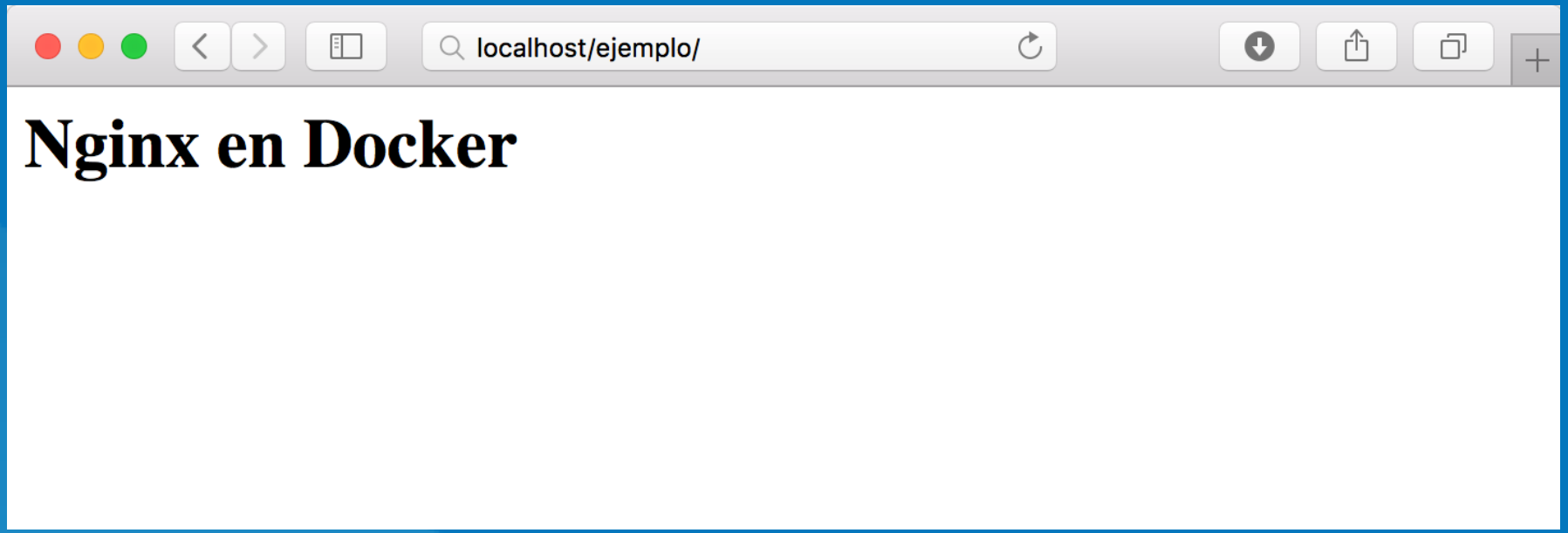
```
FROM ubuntu:16.04
MAINTAINER Leandro Di Tommaso

# Instalar Nginx y configurar una página personalizada
RUN apt-get update && apt-get install -y nginx
RUN mkdir /var/www/html/ejemplo
RUN echo "<html><h1>Nginx en Docker</h1></html>" > /var/www/html/ejemplo

EXPOSE 80
CMD    ["nginx", "-g", "daemon off;"]
```

NUESTRA PRIMER IMAGEN

```
$ docker build -t leoditommaso/nginx:1.0.0 .
  Sending build context to Docker daemon  2.048 kB
  Step 1 : FROM ubuntu:16.04
  16.04: Pulling from library/ubuntu
  6bbedd9b76a4: Already exists
  fc19d60a83f1: Already exists
  de413bb911fd: Already exists
  2879a7ad3144: Already exists
  668604fde02e: Already exists
  Digest: sha256:2d44ae143feeb36f4c898d32ed2ab2dffeb3a573d2d8928646dfc9c
  Status: Downloaded newer image for ubuntu:16.04
   ---> f753707788c5
  Step 2 : MAINTAINER Leandro Di Tommaso
   ---> Running in f93e6923c21e
   ---> a1144bb80b28
  Removing intermediate container f93e6923c21e
  Step 3 : RUN apt-get update && apt-get install -y nginx
   ---> Running in 489697f5e5d5
```

NUESTRA PRIMER IMAGEN

```
$ docker run -d -p 80:80 leoditommaso/nginx:1.0.0
```



LA REGISTRY

- Servicio para almacenar y distribuir imágenes de Docker.
- Open source (Licencia Apache).
- Instalación privada
  - Acceso local para mayor velocidad de descarga.
  - Imágenes en un ambiente controlado y gestionado por la organización.
- Servicio en la nube (Docker Hub).
  - Libre de mantenimiento.

DOCKER HUB

- Gratis para imágenes públicas.
- Soporta builds automáticos (desde Github/Bitbucket).
- Cuentas para organizaciones.
- Plan pago para imágenes privadas.
- https://hub.docker.com

CONSIDERACIONES PARA TRABAJAR CON DOCKER

INTRODUCCIÓN

- Ya sabemos que:
  - Las imágenes Docker son inmutables.
  - Los contenedores crean una capa con las diferencias correspondientes respecto de la imagen original.
- Entonces los contenedores deberían minimizar los cambios respecto de la imagen original.
  - Optimizando el uso de espacio y evitando impactos de performance.
  - Promoviendo la reusabilidad.

INMUTABILIDAD EN LA INFRAESTRUCTURA

- Desplegar una actualización de una aplicación, consiste en crear nuevas intancias y destruir las anteriores, en vez de actualizarlas sobre la instancia productiva.
- Una vez que una aplicación está corriendo, **¡evitamos tocarla!** promoviendo así:
  - Repetibilidad.
  - Reducir costos de mantenimiento.
  - Simplificar rollbacks.

INMUTABILIDAD EN LA INFRAESTRUCTURA

- Para lograr este tipo de inmutabilidad deben cumplirse los siguientes requerimientos:
  - La aplicación debe ser stateless. Su estado debe almacenarse en un servicio por fuera del alcance de la *infraestructura inmutable*.
  - Existe un template y/o conjunto de instrucciones que permiten desplegar una instancia de la aplicación desde cero.
- El segundo punto lo resuelve fácilmente docker

¿QUÉ ES DINÁMICO ENTONCES?

- La creación de las imágenes debe conocer bien el dominio para identificar las partes que son dinámicas:
  - Archivos que se generan por la aplicación.
  - Uploads desde la aplicación.
  - Logs.
  - Spool.

¿CÓMO VERIFICAR SI MIS CONTENEDORES CRECEN?

Un mal diseño de las imágenes impactará en la performance de los contenedores que generarán grandes capas con datos dinámicos.

Ante la actualización del contenedor, estos datos se perderán.

EL SIGUIENTE COMANDO PERMITE VERIFICAR ESTO

```
$ docker ps -s

CONTAINER ID        IMAGE      .....                SIZE
0d5c12033ee3        nginx      .....     2 B (virtual 182.8 MB)
```

El tamaño es lo que crece el contenedor respecto de la imagen. El tamaño virtual es lo que ocupa el contenedor sumado al tamaño de la imagen.

BUENAS PRÁCTICAS

- **Los contenedores deben ser efímeros:** pararlos, destruirlos y volverlos a iniciar con una mínima configuración.
- **Evitar paquetes innecesarios:** las imágenes no deben incluir paquetes que no se utilicen.
- **Un proceso por contenedor:** en la mayoría de los casos, se debe correr un proceso por contenedor. Desacoplar aplicaciones en múltiples contenedores hace mucho más simple el escalamiento horizontal y reuso de contenedores.
- **La (in)necesidad de ssh**: acceder a un contenedor es algo que debemos evitar. En términos de infraestructura inmutable, el servicio no debería considerar SSH.

VOLÚMENES

¿CÓMO GUARDO LA INFORMACIÓN?

- Los contenedores son volátiles e inmutables.
- Debemos preservar la información importante.
- ¿Dónde?
  - En volúmenes de datos.

CARACTERÍSTICAS DE LOS VOLÚMENES

- No utilizan un sistema de archivos de unión (UFS).
- Pueden compartirse y reusarse entre contenedores.
- Los cambios se hacen directamente en el volumen.
- La información del volumen **no se incluye** en la imagen.
- Persisten aún cuando se eliminen todos los contenedores que los usan.
  - Pueden quedar volúmenes sin referenciar.

TIPOS DE VOLÚMENES

- Volúmenes anónimos.
- Volúmenes nombrados.
- Volúmenes desde el SO host.

TIPOS DE VOLÚMENES

- Al crear un volúmen anónimo o nombrado, la información que exista en el punto de montaje se copia al volumen.
- Con volúmenes desde el SO host o desde otro contenedor, se oculta la información que exista en el punto de montaje.
  - Correspondencia con el comando mount.

ANALIZANDO LOS VOLÚMENES

Al iniciar un contenedor, la opción -v permite indicar qué volumen utilizar. El siguiente ejemplo define tres volúmenes: uno anónimo, uno nombrado y uno desde el SO host:

```
$ docker run -it \
      -v /usr/local                    # anonimo
      -v test-volume:/test-volume      # nombrado
      -v /tmp:/tmp                     # SO host
      ubuntu bash
```

Inspeccionando los volúmenes vemos:

```
$ docker volume ls
  DRIVER    VOLUME NAME
  local     e9c7022b8c7bec55891ca44b8c40de1e5f41cf0fe9505a334bca06a484a5
  local     test-volume
```

DOCKER COMPOSE

¿QUÉ ES DOCKER COMPOSE?

- Herramienta que permite levantar aplicaciones compuestas por múltiples contenedores.
- La arquitectura se define y configura en un archivo de texto (YAML).
  - Simple e intuitivo.
- Se vale de un comando para:
  - Iniciar, detener y reconstruir servicios.
  - Ver el estado de los servicios, los logs, etc.

VERSIONES DE DOCKER COMPOSE

- Hay dos versiones diferentes, la 1 y la 2.
- No son compatibles entre sí.
- Pequeños cambios en el archivo de texto.
- Veremos la sintaxis de la versión 2.

DOCKER COMPOSE: EJEMPLO

- Instalación de Wordpress.
  - Vamos a crear un archivo llamado docker-compose.yml.
  - Definiremos allí la arquitectura de la aplicación.
  - Nos valdremos del comando docker-compose para levantar Wordpress e interactuar con los contenedores generados.

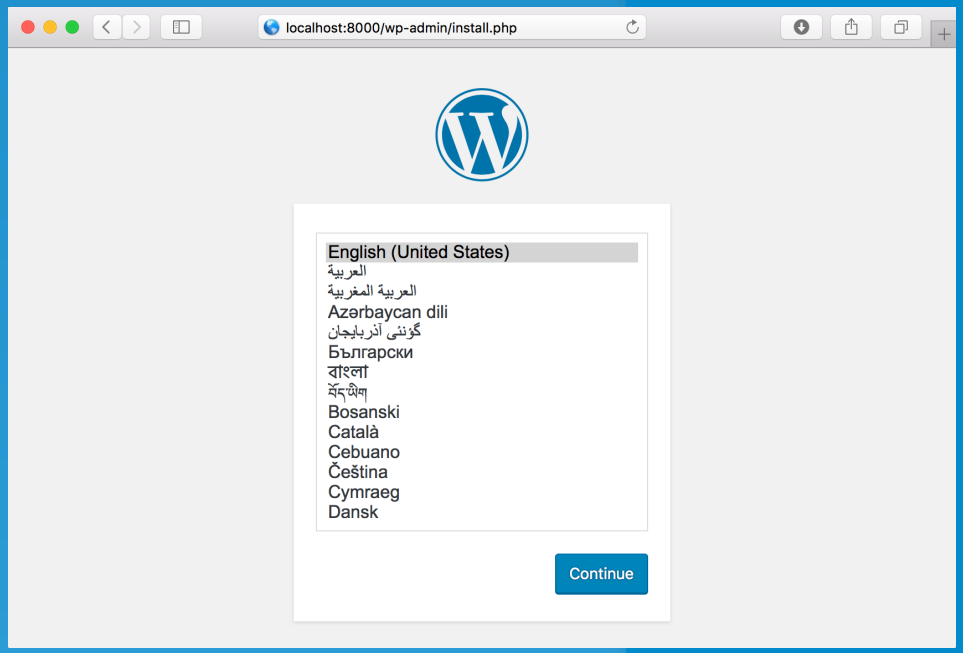
LEVANTANDO UN WORDPRESS

```
version: '2'

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

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
```

LEVANTANDO UN WORDPRESS

```
$ docker-compose up -d
  Creating network "wordpress_default" with the default driver
  Creating volume "wordpress_dbdata" with default driver
  Creating wordpress_db_1
  Creating wordpress_wordpress_1

$ docker-compose ps
        Name                      Command               State          Po
-------------------------------------------------------------------------
wordpress_db_1          docker-entrypoint.sh mysqld      Up      3306/tcp
wordpress_wordpress_1   /entrypoint.sh apache2-for ...   Up      0.0.0.0:
```

LEVANTANDO UN WORDPRESS




```
$ docker-compose logs -f
  wordpress_1  | 127.0.0.1 - - [16/Nov/2016:17:56:39 +0000] "GET / HTTP/
  wordpress_1  | 127.0.0.1 - - [16/Nov/2016:17:56:39 +0000] "GET /wp-adm
  wordpress_1  | 127.0.0.1 - - [16/Nov/2016:17:56:41 +0000] "GET /favicc
```

DOCKER EN PRODUCCIÓN

LOS DISTINTOS ESQUEMAS

- Usando Docker para iniciar servicios de forma aislada.
- Usando un cluster de docker.

DOCKER STANDALONE

- Cada servidor Linux corre el servicio de Docker en forma aislada.
- Los contenedores pueden iniciarse automáticamente durante el booteo usando:
  - Manejadores de procesos como upstart, systemd o supervisor.
  - A través de políticas de reinicio (Docker >= 1.2).

A TRAVÉS DE MANEJADORES DE PROCESOS

Dado que Docker no setea políticas de reinicio por defecto, cuando un servicio iniciado con Docker termina, no se toma ninguna acción.

Las políticas de reinicio podrían conflictuar con los manejadores de procesos.

INTEGRACIÓN CON LOS MANEJADORES DE PROCESOS

- Cuando un contenedor ya corre como esperamos, entonces podemos attacharlo a un manejador de procesos para que él lo maneje.
- Corriendo `docker start -a` Docker attachará al contenedor corriendo (o iniciará si no está corriendo) reenviando las señales al manejador de procesos.

EJEMPLOS

Para entender los siguientes ejemplos veremos qué hace:

`docker start -a`

```
# Iniciamos un contenedor nginx daemonizado y nombrado:
docker run -d  --name=nginx_docker -p 9090:80 nginx

# El contenedor ya atiende en el puerto 9090:
curl http://localhost:9090

# Usando docker start para attachar al contenedor nombrado
docker start -a nginx_docker
Ctrl+C # envía la señal SIGTERM al proceso. Muere el contenedor
       # el comando curl ya no es exitoso

# Usando nuevamente docker start
docker start -a nginx_docker # reinicia el servicio
```

EJEMPLO UPSTART

Un contenedor que inicia Redis.

```
description "Redis container"
author "Me"
start on filesystem and started docker
stop on runlevel [!2345]
respawn
script
  /usr/bin/docker start -a redis_server
end script
```

EJEMPLO SYSTEMD

```
[Unit]
Description=Redis container
Requires=docker.service
After=docker.service

[Service]
Restart=always
ExecStart=/usr/bin/docker start -a redis_server
ExecStop=/usr/bin/docker stop -t 2 redis_server

[Install]
WantedBy=default.target
```

`docker stop -t TIME` envía la señal `SIGTERM` y luego del tiempo especificado envía `SIGKILL`

POLÍTICAS DE REINICIO

Si no queremos utilizar manejadores de procesos, entonces podemos emplear las políticas de reinicio.

Estas políticas permiten especificar cómo un contenedor debería o no ser reiniciado cuando termina.

POLÍTICAS DE REINICIO

- **no**: no iniciar el contenedor cuando termina. *Valor por defecto*.
- **on-failure:[max]**: reiniciar solo si el contenedor termina con exit status diferente a cero. Limitar opcionalmente los reintentos de reinicio.
- **always**: siempre reiniciar el contenedor. Además el contenedor se iniciará cuando inicia el daemon Docker.
- **unless-stopped**: idem anterior, salvo que en un reinicio del servicio Docker considera si previamente fue detenido.

EJEMPLO DE POLÍTICA DE REINICIO

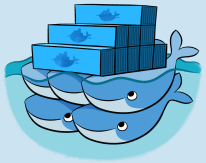
```
# Iniciamos ninx con restart policy always
docker run -d --restart=always --name=nginx_docker -p  9090:80 nginx

# Verificamos la cantidad de reinicios:
docker inspect  -f "{{ .RestartCount }}" nginx_docker

# Matamos abruptamente el contenedor
docker exec nginx_docker kill -QUIT 1

# Verificamos la cantidad de reinicios:
docker inspect  -f "{{ .RestartCount }}" nginx_docker
```

nginx recibe la señal QUIT para finalizar el proceso
https://www.nginx.com/resources/wiki/start/topics/tutorials/commandline/

CLUSTERS DOCKER

- La idea detrás de los clusters Docker es la de disponer de nodos Linux con el Docker Engine de tal forma de poder utilizarlos para correr contenedores.
  - Estos Linux deben ser muy pequeños dado que su única razón de ser es la de proveer un kernel, no utilidades.
- Serían como equipos físicos pertenecientes a un pool de hardware disponible en un virtualizador como XEN o VMWare.

LOS CLUSTERS MÁS CONOCIDOS



Swarm

Rancher

Kubernetes

Apache Mesos

CARACTERÍSTICAS DE TODOS LOS CLUSTERS

- Diseño descentralizado.
- Servicios, pods o stacks en vez de contenedores.
- Posibilidad de escalar.
- Conciliación para alcanzar el estado deseado.
- Service discovery.
- Load balancing.
- Actualizaciones en caliente.

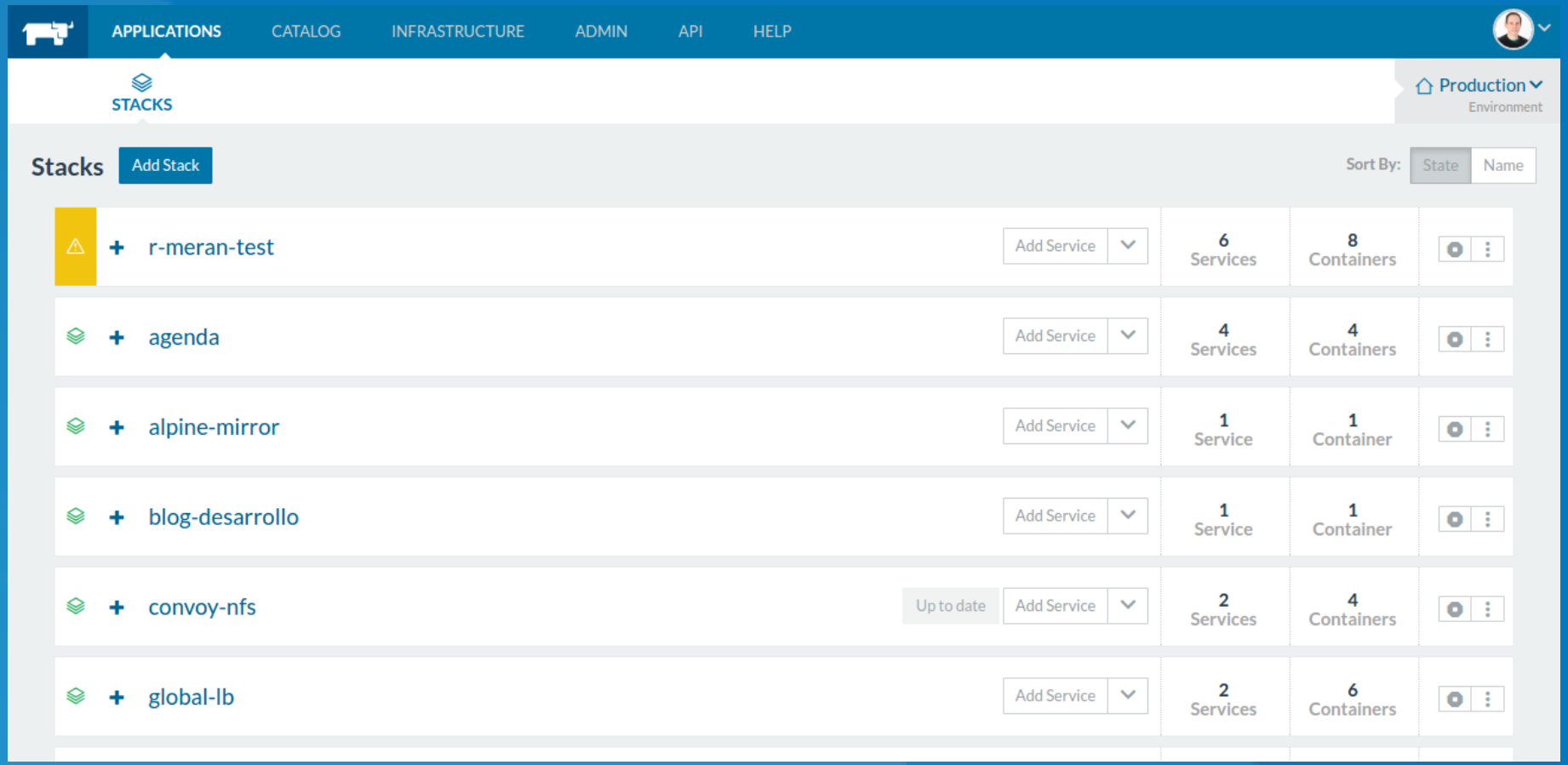
CONSIDERACIONES

- El scheduler es el encargado de determinar donde se inicia cada contenedor.
- Asociado al scheduler trabajan los health checks que garantizan la conciliación de un estado deseado: que hayan N contenedores para el servicio X.
- La distribución mágica del scheduler complica el manejo de volúmenes.
  - Los volúmenes pertenecen a un nodo.
  - Si el nodo cambia, se pierden los datos.

VOLÚMENES DISTRIBUIDOS

- Necesidad de compartir datos entre los nodos del cluster.
- Aparecen diferentes implementaciones de volúmenes compartidos. Las más populares son:
  - Convoy
  - Flocker

EJEMPLO RANCHER



¿PREGUNTAS?

Y SI LAS PREGUNTAS SURGEN MÁS TARDE...

- Leandro Di Tommaso
  - leandro.ditommaso@mikroways.net
  - https://github.com/leoditommaso
- Christian Rodriguez
  - christian.rodriguez@mikroways.net
  - https://github.com/chrodriguez

¡GRACIAS!