

docker

Objetivos del curso.

- Visión y entendimiento de la plataforma de Docker
- Uso y buenas prácticas de las distintas herramientas
- Workflow aplicativo sobre la plataforma de Docker



Marcos Nils Lilljedahl (@marcosnils)
Head of R&D en Mantika (<http://mantika.ca>)

- Ing. en sistemas / MoIT
- OSS & Golang ❤
- Miembro del docker community speakers program
- Docker global hackday #3 (<https://blog.docker.com/2015/09/docker-global-hack-day-3-winners/>)
- Keynote cierre DockerCon EU 2015 (https://www.youtube.com/watch?v=ZBcMy-_xuYk)
- Crossfitter

Notas

- El hashtag oficial del curso es #PlatziDocker
- Utilicen las herramientas de la plataforma para enviar sus consultas
- El ícono de la terminal indica que el slide actual contiene ejemplos interactivos en consola
- En el margen inferior izquierdo se visualiza el tema actual
- Hagan el curso suyo, el objetivo es que puedan poner en práctica todo lo aprendido de manera **inmediata**



Mitos de docker

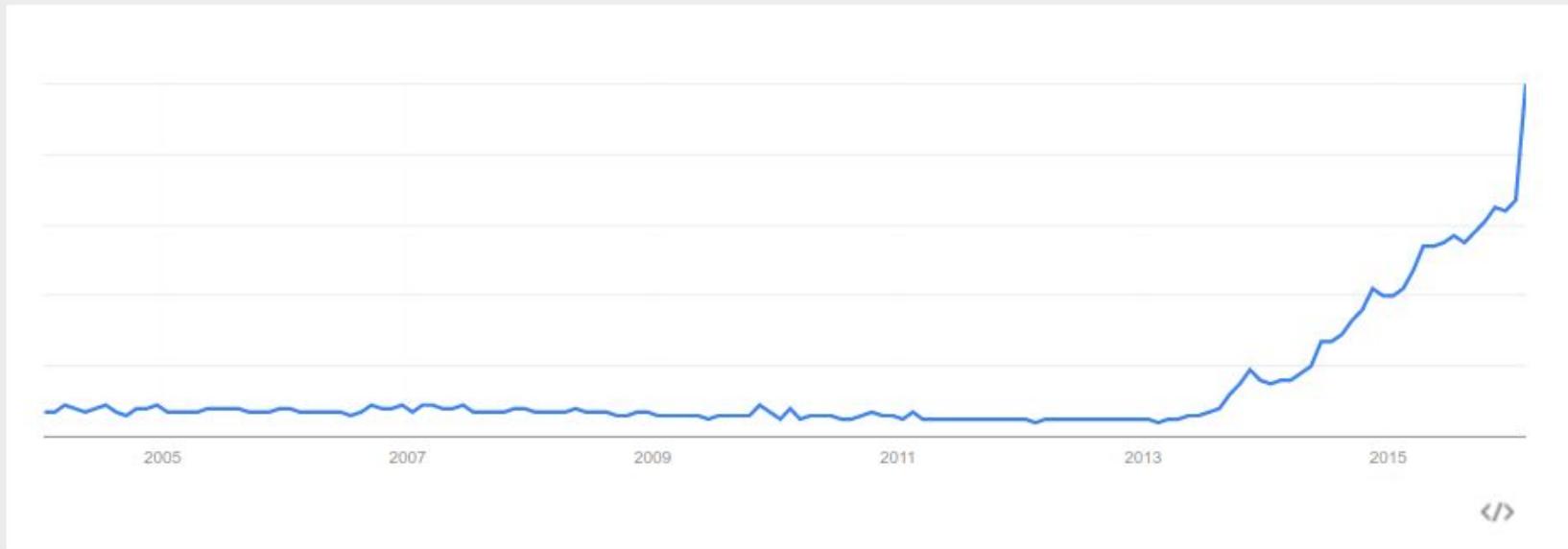
- Docker no aplica para mi organización o proyecto
- Docker no se encuentra lo suficientemente maduro como tecnología
- Docker funciona solamente en linux
- El equipo de infraestructura en mi compañía no utiliza docker
- Los contenedores no son aptos para cargas de trabajo críticas.
- Los contenedores no son seguros porque no proveen aislamiento de hardware

Por qué Docker?

Interés a lo largo del tiempo ?

Titulares de noticias ?

Previsión ?



Por qué Docker? (cont.)

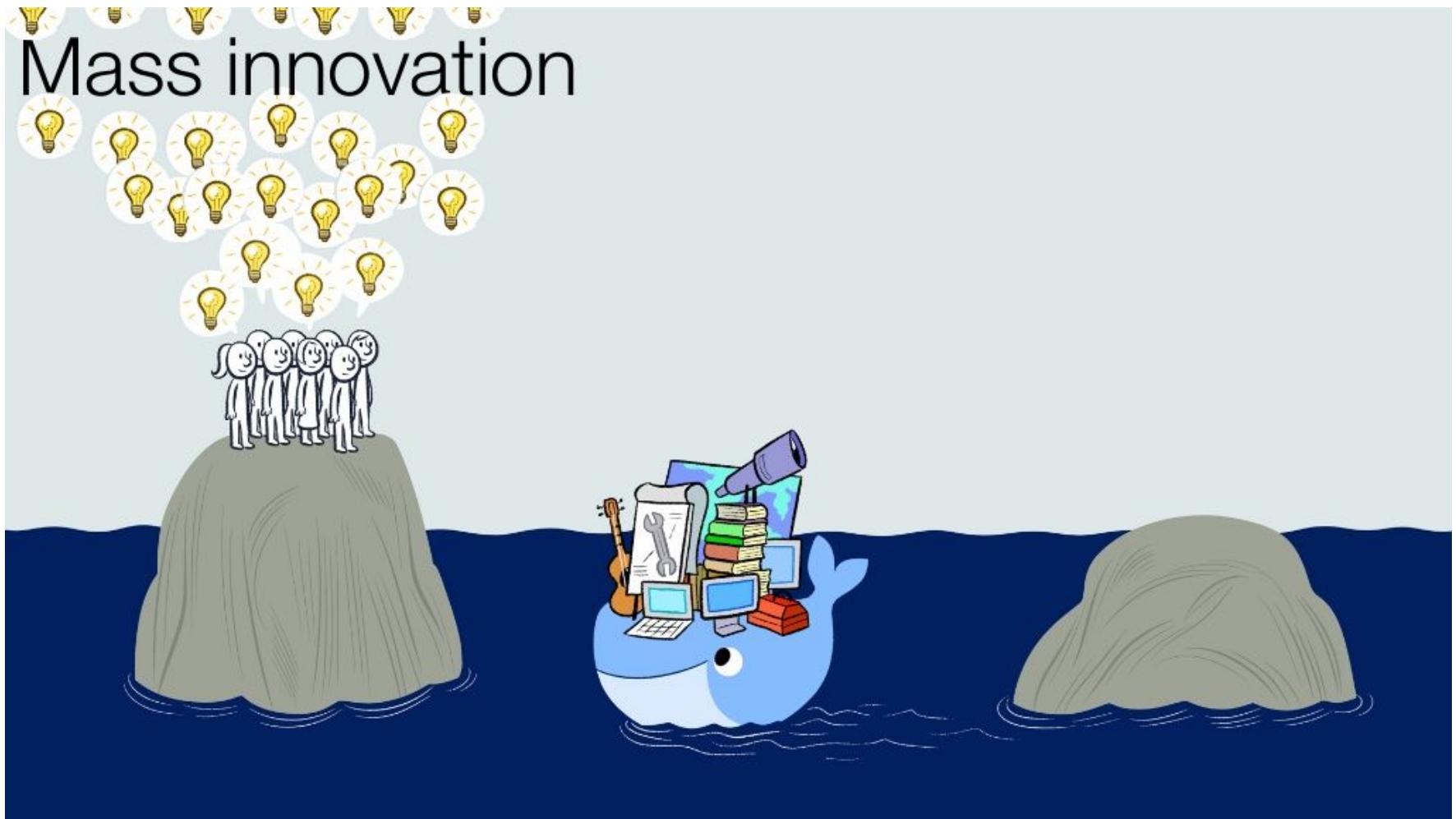
Billions of creative people



Incredible technology



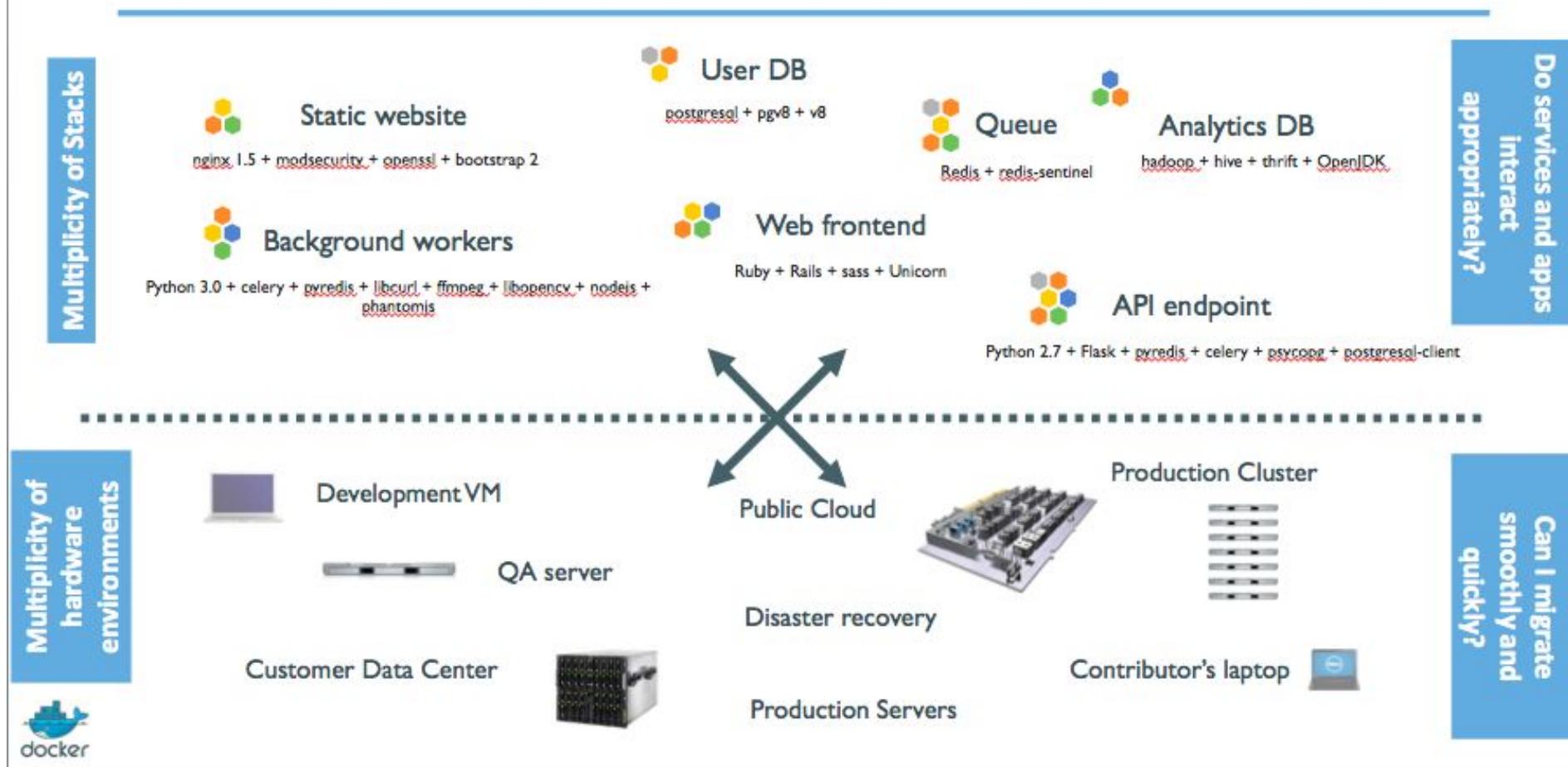
Por qué Docker? (cont.)



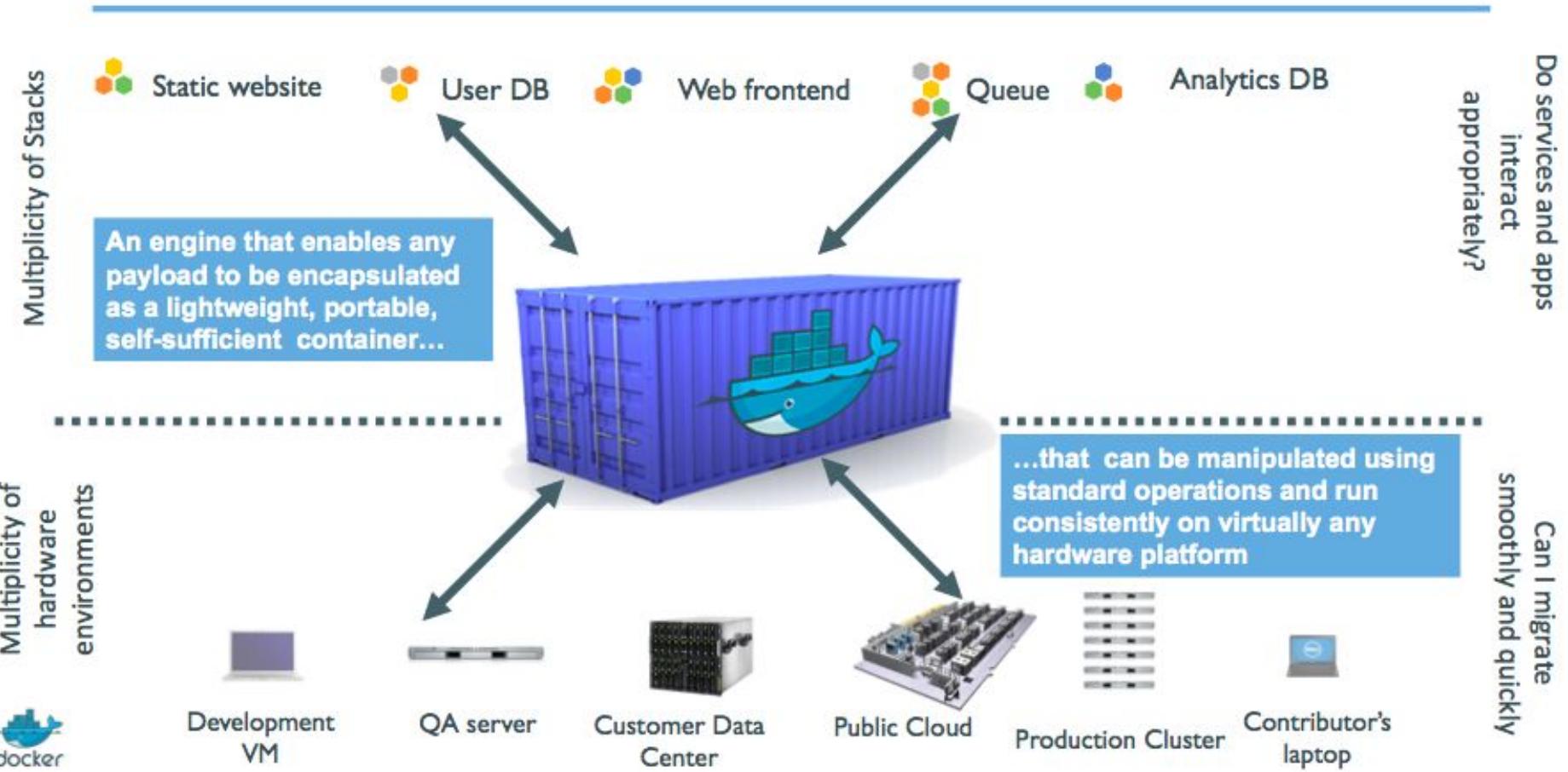
Por qué Docker? (cont.)



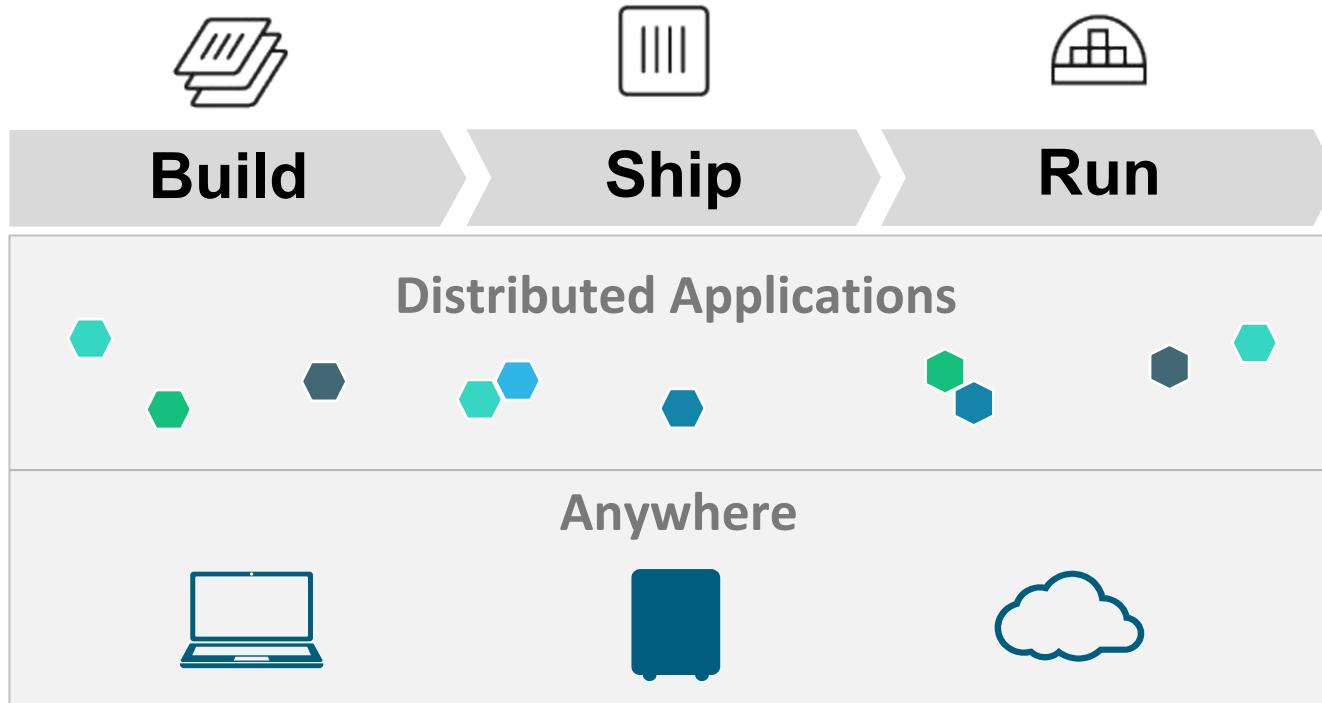
Por qué Docker? (cont.)



Por qué Docker? (cont.)



La misión de Docker



“Build, ship and run any app anywhere”

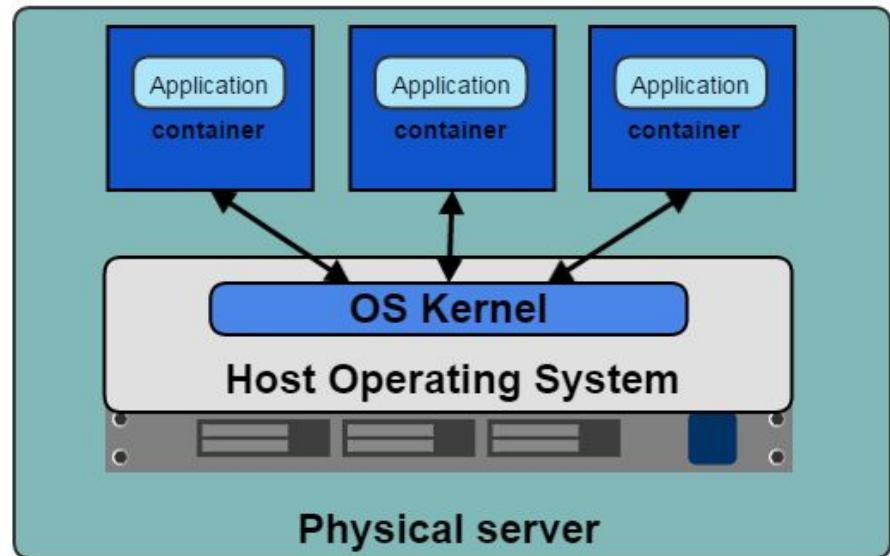
Contenedores



- Contenedor = namespaces + cgroups + chroot +
...
 - Namespaces: Vistas de los recursos del SO
 - Cgroups: Limitan y miden los recursos del SO
 - Chroot: Cambia el root directory de un proceso

Contenedor

- Los contenedores son más livianos que las VMs
- No es necesario instalar un OS por contenedor
- Menor utilización de recursos
- Mayor cantidad de contenedores por equipo físico
- Mejor portabilidad



Instalando docker.



<https://www.docker.com/products/docker-toolbox>

<https://docs.docker.com/engine/installation/linux/>

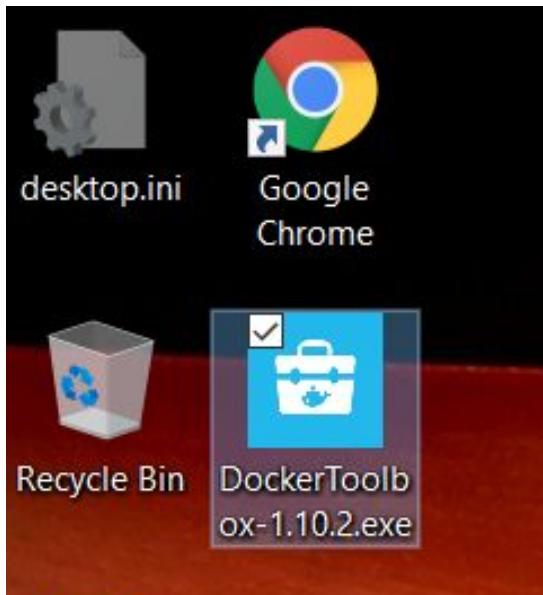


Instalando docker.

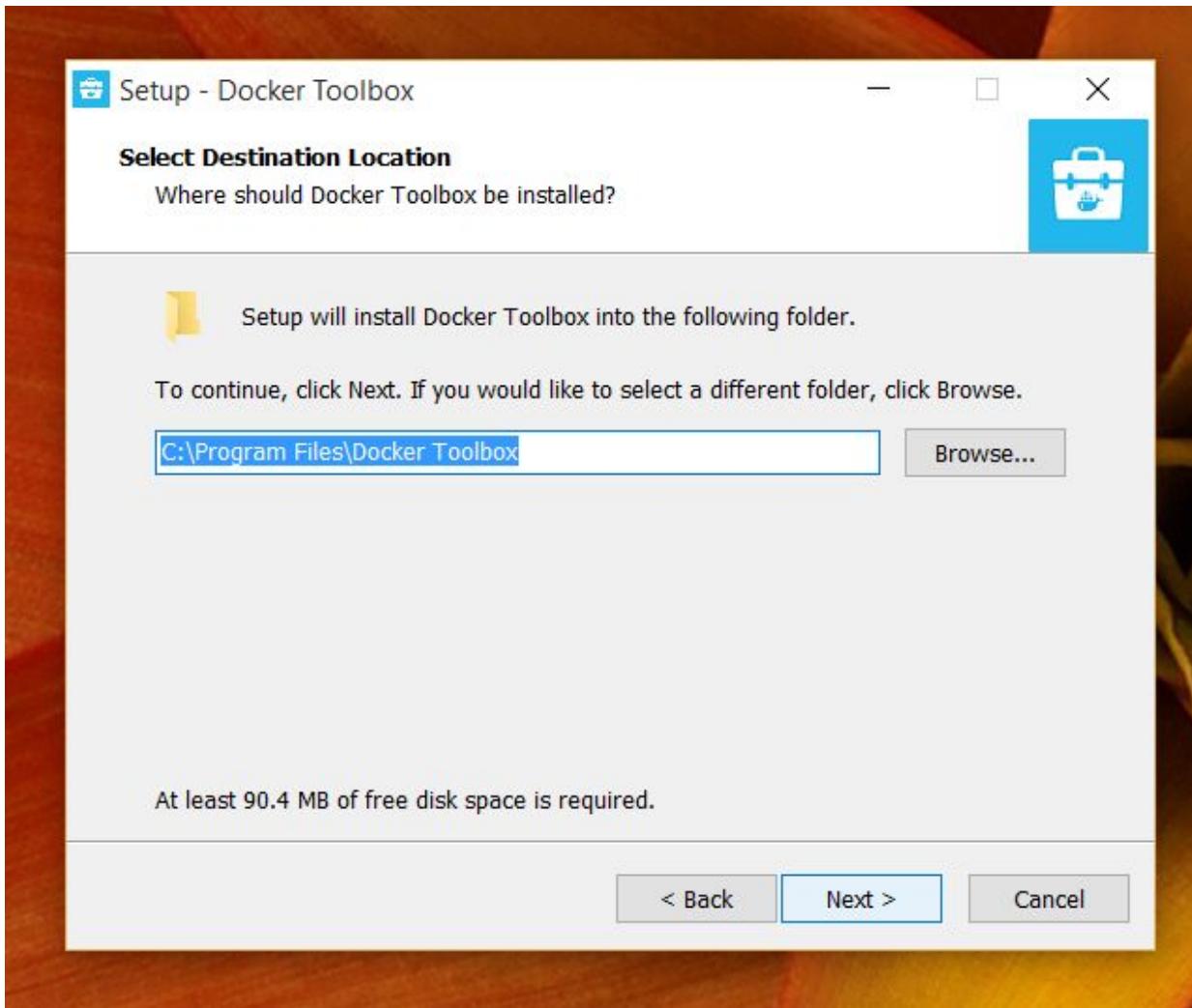
Phoenix TrustedCore(tm) Setup Utility	
Advanced	
Advanced Processor Configuration	Item Specific Help
CPU Mismatch Detection: [Enabled]	When enabled, a UMM (Virtual Machine Monitor) can utilize the additional hardware capabilities provided by Vanderpool Technology.
Core Multi-Processing: [Enabled]	
Processor Power Management: [Disabled]	
Intel(R) Virtualization Technology [Enabled]	
Execute Disable Bit: [Enabled]	
Adjacent Cache Line Prefetch: [Disabled]	
Hardware Prefetch: [Disabled]	
Direct Cache Access [Disabled]	
Set Max Ext CPUID = 3 [Disabled]	If this option is changed, a Power Off-On sequence will be applied on the next boot.

F1 Info ↑ Select Item -/+ Change Values F9 Setup Defaults
Esc Exit + Select Menu Enter Select ▶ Sub-Menu F10 Save and Exit

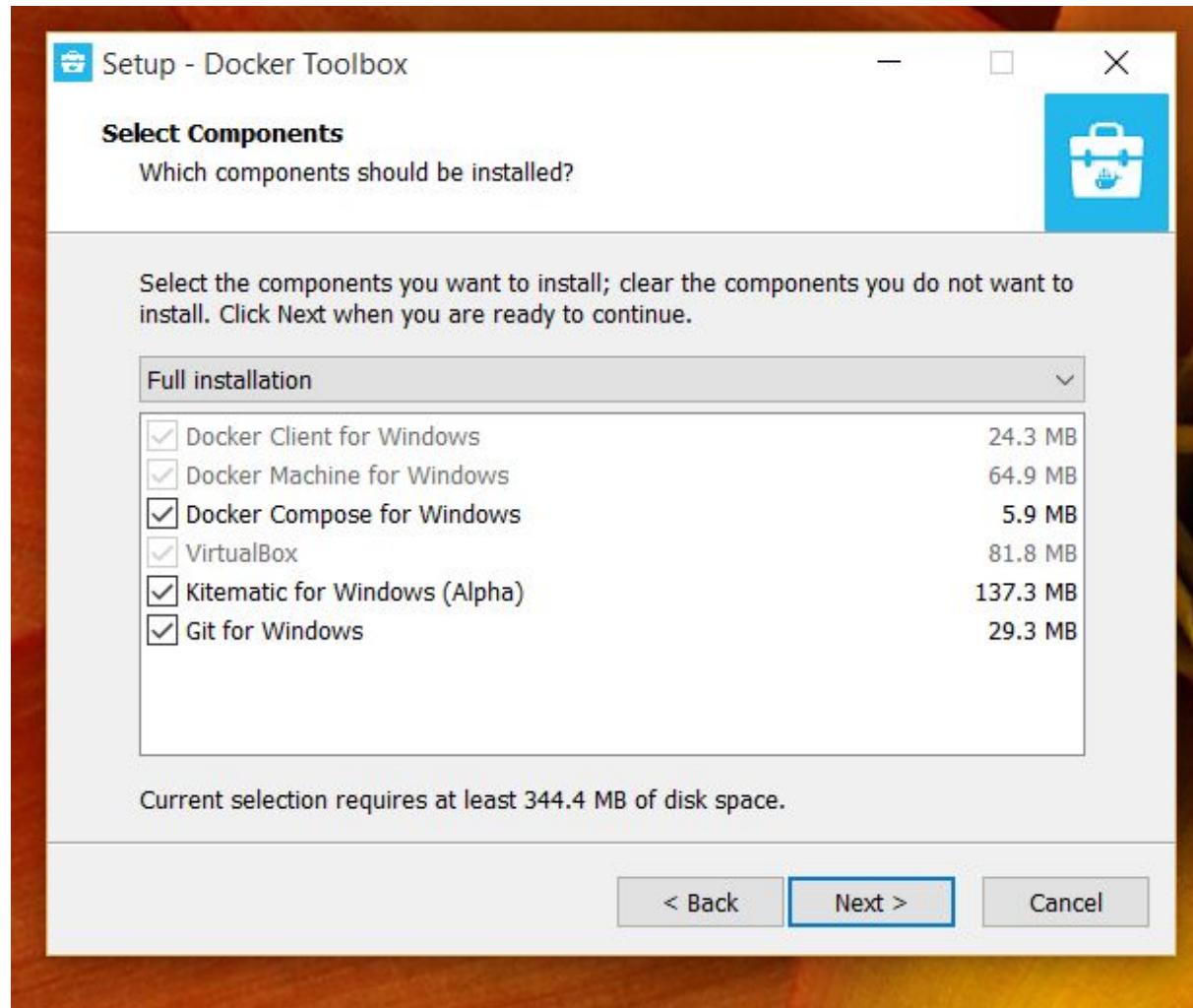
Instalando docker.



Instalando docker.



Instalando docker.



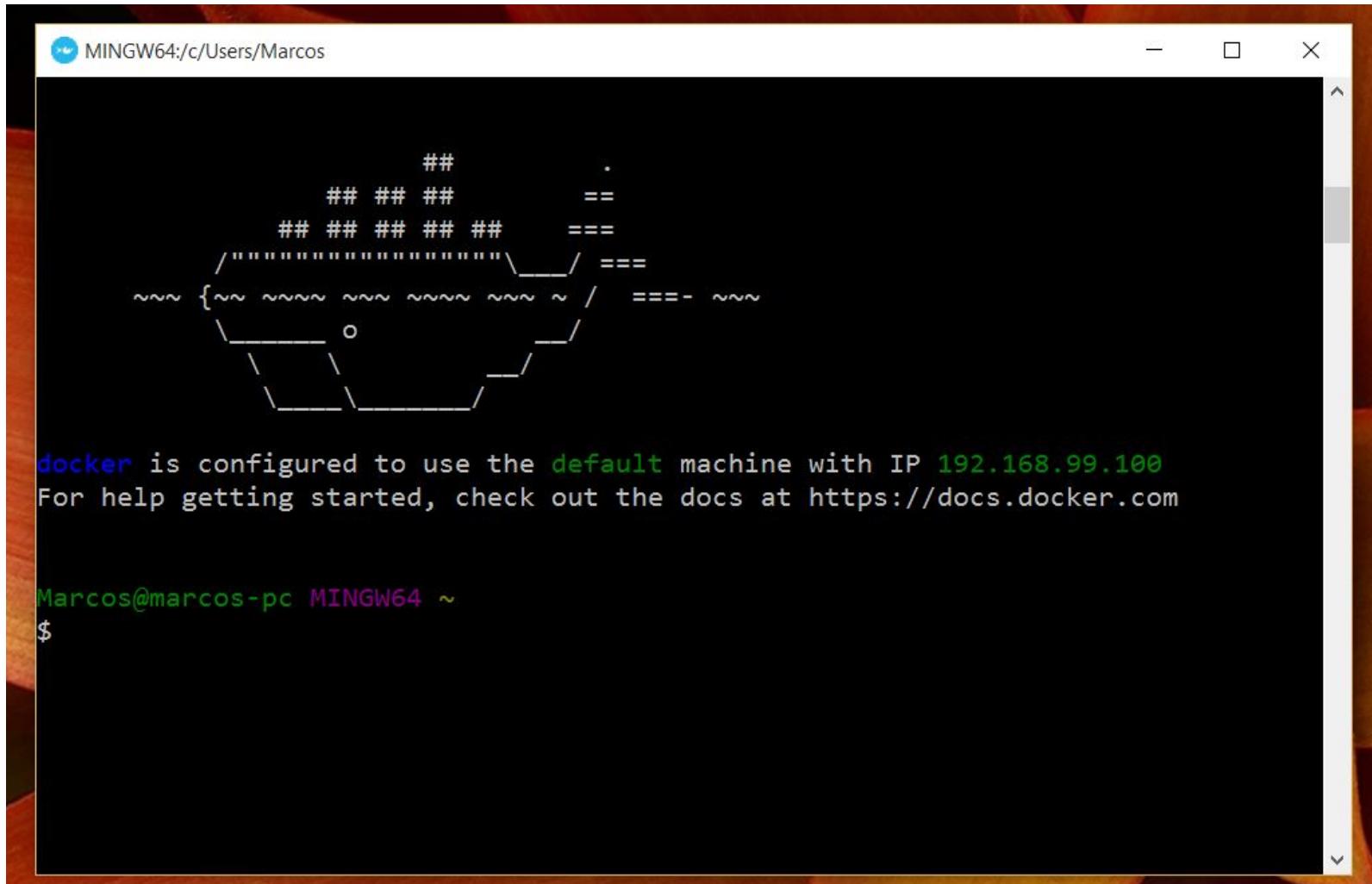
Instalando docker.



Docker Quickstart Terminal

```
Creating CA: C:\Users\Marcos\.docker\machine\certs\ca.pem
Creating client certificate: C:\Users\Marcos\.docker\machine\certs\cert.pem
Running pre-create checks...
Creating machine...
(default) Copying C:\Users\Marcos\.docker\machine\cache\boot2docker.iso to C:\Users\Marcos\.docker\machine\machines\default\boot2docker.iso...
(default) Creating VirtualBox VM...
(default) Creating SSH key...
(default) Starting the VM...
(default) Check network to re-create if needed...
(default) Windows might ask for the permission to create a network adapter. Sometimes, such confirmation window is minimized in the taskbar.
(default) Found a new host-only adapter: "VirtualBox Host-Only Ethernet Adapter #2"
(default) Windows might ask for the permission to configure a network adapter. Sometimes, such confirmation window is minimized in the taskbar.
(default) Windows might ask for the permission to configure a dhcp server. Sometimes, such confirmation window is minimized in the taskbar.
(default) Waiting for an IP...
```

Instalando docker.



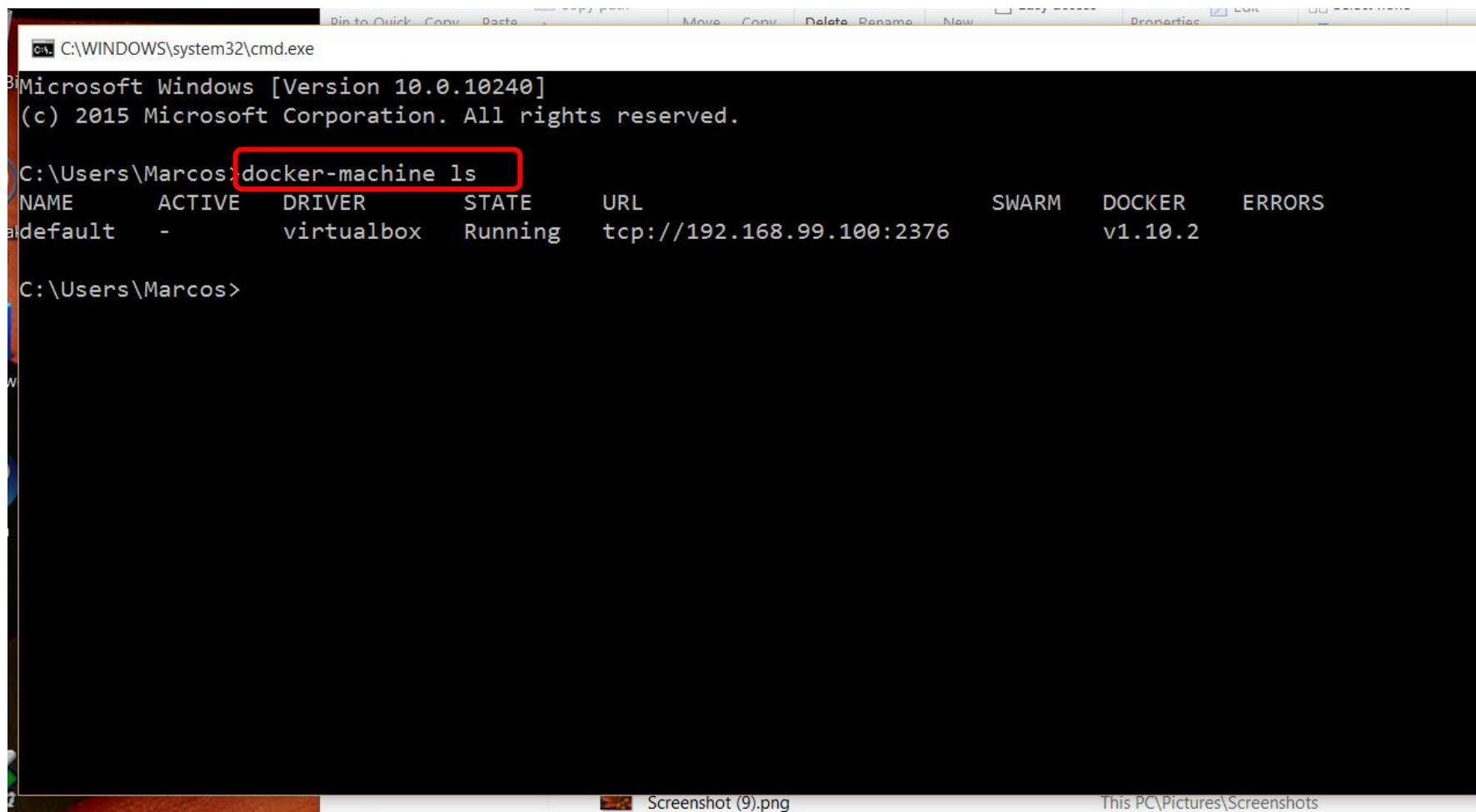
A screenshot of a Windows terminal window titled "MINGW64:/c/Users/Marcos". The window contains the following text:

```
##          ##
## ## ##      ==
## ## ## ## ## ===
/"""""""""""\_/_/ ===
~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
 \_/_/ o      _/_/
 \_/\_/\_/_/ _/_/
```

docker is configured to use the default machine with IP 192.168.99.100
For help getting started, check out the docs at <https://docs.docker.com>

Marcos@marcos-pc MINGW64 ~
\$

Instalando docker.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

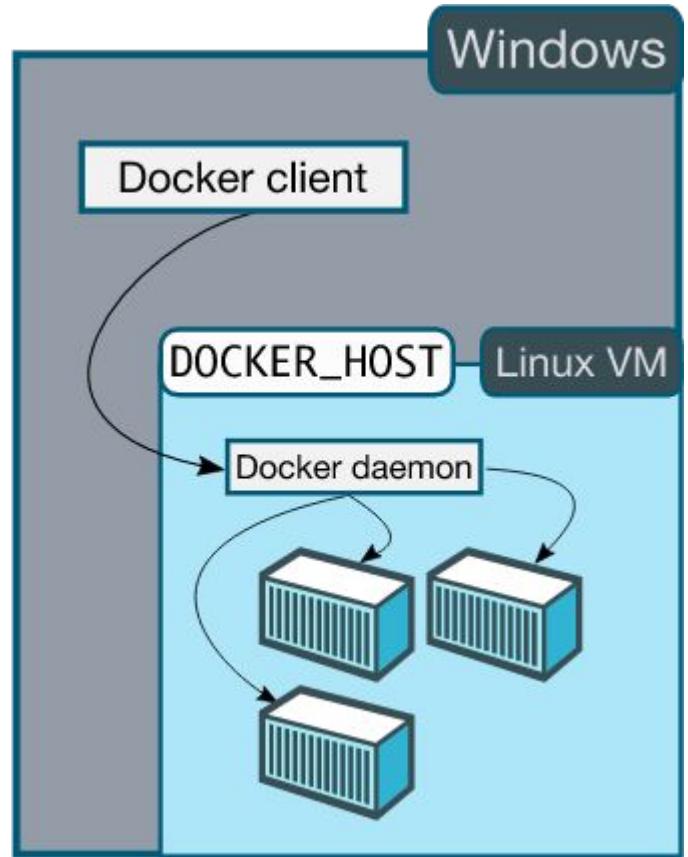
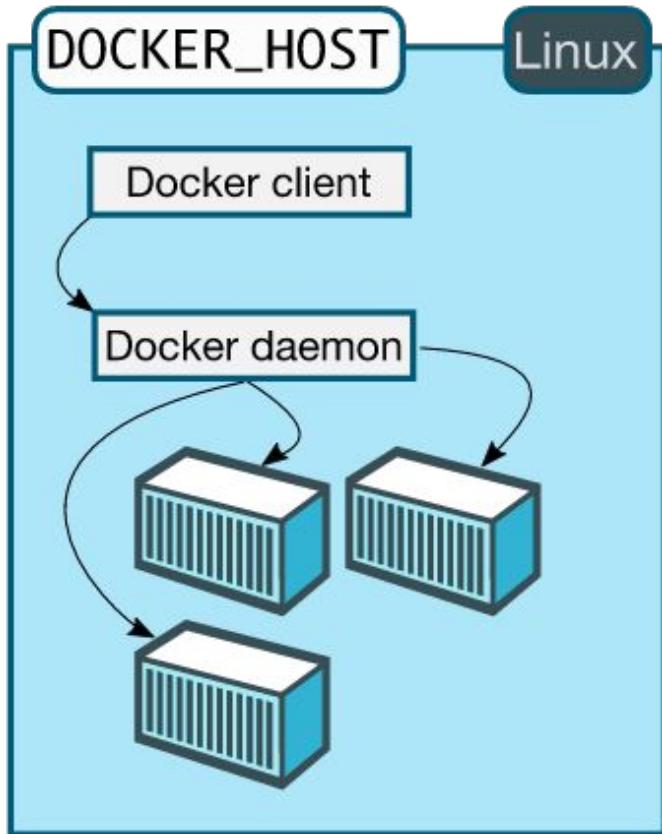
C:\Users\Marcos\docker-machine ls
NAME      ACTIVE   DRIVER      STATE      URL
default    -        virtualbox  Running    tcp://192.168.99.100:2376
SWARM     DOCKER    ERRORS
v1.10.2

C:\Users\Marcos>
```

Screenshot (9).png

This PC\Pictures\Screenshots

Docker engine.



Controlando las versiones.

'docker version'

```
marcos@XPS:~$ docker version
```

Client:

```
Version:      1.10.1
API version:  1.22
Go version:   go1.5.3
Git commit:   9e83765
Built:        Thu Feb 11 20:39:58 2016
OS/Arch:      linux/amd64
```

CLIENTE

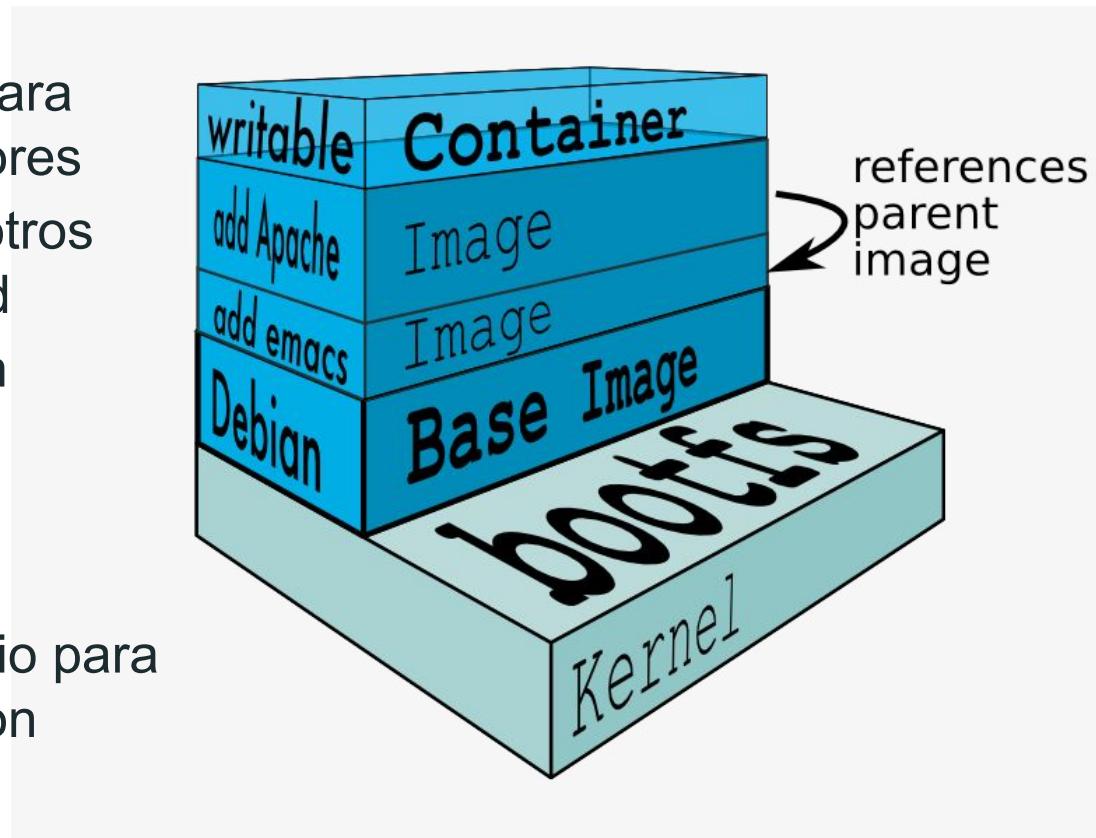
Server:

```
Version:      1.10.1
API version:  1.22
Go version:   go1.5.3
Git commit:   9e83765
Built:        Thu Feb 11 20:39:58 2016
OS/Arch:      linux/amd64
```

**SERVICIO /
DEMONIO**

Contenedores e imágenes

- Imágenes
 - Plantilla de sólo lectura para crear nuestros contenedores
 - Creadas por nosotros u otros usuarios de la comunidad
 - Se pueden guardar en un registro interno o público
- Contenedores
 - Aplicación aislada
 - Contiene todo lo necesario para ejecutar nuestra aplicación
 - Basados en una o más imágenes.



Imágenes

- Sitio de imágenes públicas (<https://hub.docker.com>)
- Utilización del docker CLI - `docker search`
- Imágenes locales



Imágenes - Formato

- El formato de las imágenes se compone de **repositorio:tag**
- Una misma imagen puede tener múltiples tags
- El tag por defecto de una imagen es “latest”

ubuntu ★
Last pushed: 5 days ago

Repo Info Tags

Tag Name	Size
utopic-20150528	194 MB
utopic-20150418	194 MB
utopic-20150319	194 MB
xenial-20160125	47 MB
trusty	66 MB



Descargando imágenes

- Para descargar una imagen del repositorio externo, se utiliza el comando `docker pull`
- Cuando se ejecuta un contenedor con el comando `docker run` las imágenes son descargadas automáticamente si no se encuentran en el repositorio local local copy is found



Ciclo de vida de los contenedores

- Ciclo de vida básico
 - Se crea el contenedor a partir de una imagen
 - Se ejecuta un proceso determinado en el contenedor
 - El proceso finaliza y el contenedor se detiene
 - Se destruye el contenedor
- Ciclo de vida avanzado
 - Se crea el contenedor a partir de una imagen
 - Se ejecuta un proceso determinado en el contenedor
 - Realizar acciones dentro del contenedor
 - Detener el contenedor
 - Lanzar el contenedor nuevamente

Creando nuestro primer contenedor

- Utilizando el comando docker run
- El comando docker run realiza 2 acciones
 - Crea el contenedor con la imagen especificada
 - Ejecuta el contenedor

- Sintaxis

```
docker run [opciones] [imagen] [comando]  
[args]
```

- El formato de la imagen es repository:[tag]



Listando los contenedores

- Utilizar **docker ps** para listar los contenedores
- La bandera **-a** lista todos los contenedores (inclusive aquellos que se encuentran detenidos)

```
marcos@XPS:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
3bdd1af899a2        ubuntu:14.04       "ps -ef"                8 seconds ago      Up 8 seconds
d8a0b13bdb08        ubuntu:14.04       "echo 'Hola Platzi\n'"  13 seconds ago     Up 13 seconds
3199e1bc692b        hello-world       "/hello"                22 seconds ago     Up 22 seconds
```



Contenedores interactivos

- Utilizar las banderas `-i` and `-t` en el comando docker run
- La bandera `-i` le indica a docker utilizar el STDIN del contenedor
- La bandera `-t` indica que se requiere de una pseudo terminal
- **Nota:** Es necesario ejecutar un proceso de terminal en el contenedor (ej: `sh /bash /zsh /etc`)



ID de contenedores

- Los contenedores pueden referenciarse utilizando su ID de contenedor o un nombre
- ID formato reducido y extendido
- La información de puede obtener del comando `docker ps`
- Utilizar la bandera `--no-trunc` en docker ps para obtener el formato extendido de ID.

```
docker ps -a --no-trunc
```



Listado con filtro de contenedores

- La bandera `--filter` agrega condiciones de filtrado
- Se puede filtrar basado en el código de salida y estado del contenedor
- El estado puede ser
 - Restarting
 - Running
 - Exited
 - Paused
- Para especificar múltiples condiciones utilizar la bandera `--filter` por cada condición
- Otros filtros: id, label, name, exited, status, ancestor, isolation



Ejecutando contenedores de fondo

- Correr de fondo (background) o como demonio
- Utilizar la bandera `-d`
- Para ver el output utilizar el comando `docker logs [id contenedor] / [nombre contenedor]`



Un caso práctico

- Ejecutar un servidor de aplicaciones (tomcat)
- La bandera `-P` expone los puertos utilizados por el contenedor



Vincularse a un contenedor

- Vincularse a un contenedor, traerá dicho contenedor al frente
- El output del proceso 1 será mostrado por pantalla
- Utilizar `docker attach` y especifica el nombre / ID del contenedor
- **Cuidado:** Si se presiona `CTRL + C` vinculado a un contenedor, el mismo se detendrá de manera inmediata



Docker exec

- docker exec habilita a ejecutar procesos adicionales dentro del contenedor
- Generalmente se utiliza para acceder a una terminal dentro de un contenedor en ejecución
- docker exec -i -t [ID contenedor] [comando]
- Salir de la terminal no finaliza el container



Logs del container

- El output del ID de proceso 1 del contenedor se puede ver utilizando el comando `docker logs`
- Muestra todo el log file desde que el contenedor fue iniciado
- Las banderas *-f* y *--tail* se utilizan para controlar la salida del log



Acciones sobre un contenedor

- docker stop y docker kill detienen un contenedor en ejecución
- docker start se utiliza para arrancar un contenedor en estado **STOPPED o KILLED**
- La bandera `'-a'` en docker start se utiliza para adjuntarse automáticamente al contenedor



Inspeccionando un contenedor

- El comando `docker inspect` se utiliza para acceder a información útil de un contenedor

```
marcos@XPS:~$ docker inspect 32b448edb5106c27b96ed9ed1f725262e47523
[
  {
    "Id": "32b448edb5106c27b96ed9ed1f725262e475230732723f1b4c66",
    "Created": "2016-02-08T17:04:20.502015908Z",
    "Path": "ping",
    "Args": [
      "-c",
      "20",
      "www.google.com"
    ],
    "State": {
      "Status": "exited",
      "Running": false,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 0,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2016-02-08T17:04:20.75396631Z",
      "FinishedAt": "2016-02-08T17:04:55.153787505Z"
    },
    "Config": {
      "Image": "ping:latest",
      "Cmd": [
        "ping"
      ],
      "ExposedPorts": {},
      "Labels": {}
    }
  }
]
```



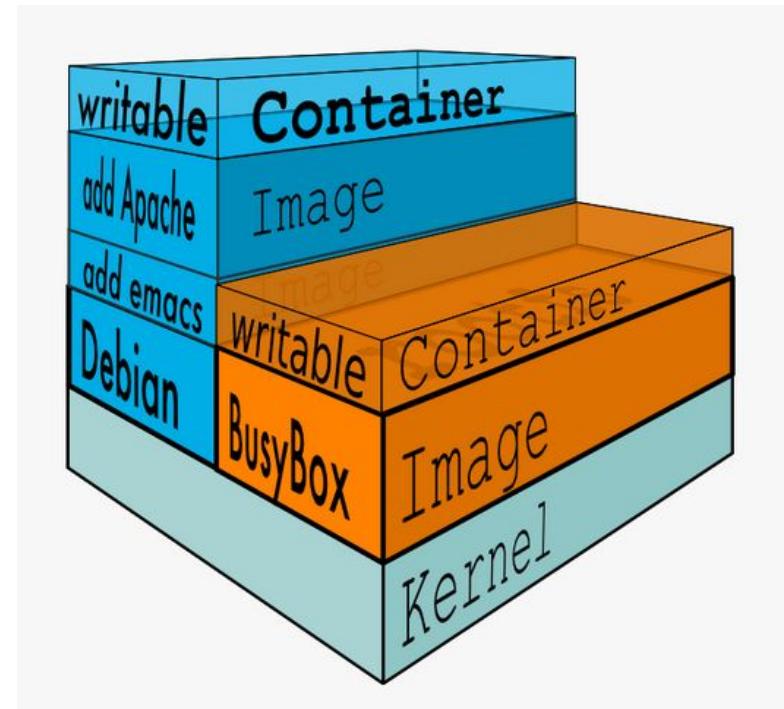
Eliminando contenedores

- Por defecto sólo pueden eliminarse contenedores detenidos
- Utilizar el comando `docker rm [ID contenedor] [nombre contenedor]`
- Para eliminar un contenedor que se encuentra en uso utilizar la bandera `-f`



Creación de imágenes

- Sistema de capas entre imágenes
- Docker crea una última capa de modo escritura para los contenedores
- Las imágenes padre son de sólo lectura
- Todos los cambios son realizados en la capa de escritura



Formas de crear imágenes

- Tres modos
 - Hacer `commit` de los contenidos de un contenedor
 - Construir una imagen basándonos en un Dockerfile
 - Importar un archivo Tar a docker con el contenido de una imagen

Forma interactiva

- Ingresar dentro del contenedor y realizar las modificaciones necesarias
- Utilizar `docker commit` para observar los cambios realizados
- Salvar los cambios realizados con el comando `docker commit` en una nueva imagen
- El formato de nombre de la imagen puede ser:
 - Oficial
 - ubuntu:14.04
 - nginx
 - Usuario / organización
 - johnnytu/myapp
 - mycompany/myapp
 - Registro privado
 - registry.mycompany.com:5000/my-image



Dockerfiles

- Provee una forma más efectiva de generar imágenes en vez de utilizar docker commit
- Se integra de manera automática en el flujo de desarrollo y de integración continua
- Las instrucciones más utilizadas son **FROM** y **RUN**
- Una vez construido el Dockerfile, utilizar `docker build` para generar la nueva imagen

```
#Ejemplo de un comentario  
FROM ubuntu:14.04  
RUN apt-get install vim  
RUN apt-get install curl
```



El build cache

- Docker guarda un snapshot de cada imagen luego de cada instrucción de build
- Antes de ejecutar un paso, docker chequea si la instrucción se encuentra en la cache teniendo en cuenta el orden original
- Si la condición anterior se cumple, docker utiliza la cache en vez de ejecutar el paso nuevamente
- Docker utiliza comparación de strings exactas para chequear con la cache
 - Simplemente cambiando el orden de las instrucciones la cache se invalida
- Para deshabilitar la cache manualmente se puede utilizar la bandera --no-cache

```
docker build --no-cache -t imagen .
```



Historial de una imagen

- El comando `docker history` muestra las capas de las cuales se encuentra creada una imagen.
- Se puede observar cada capa, cuándo fue creada, su tamaño y el comando el cual la generó.

```
marcos@XPS:~/Projects/platzidocker/dockerfiles$ docker history b3429bc09b4f
IMAGE           CREATED          CREATED BY               SIZE
b3429bc09b4f   17 seconds ago   /bin/sh -c apt-get install -y curl   11.37 MB
c51d9d5a351d   36 seconds ago   /bin/sh -c apt-get install -y vim   43.13 MB
06ab2de020f4   8 weeks ago      /bin/sh -c #(nop) CMD ["/bin/bash"] 0 B
```



La instrucción CMD

- CMD define el comando por defecto a ejecutar cuando se crea el contenedor
- Se puede usar tanto el formato Shell como Exec
- Sólo puede especificarse una vez en el dockerfile
 - Si se especifica múltiples veces, sólo la última será válida
- **Puede ser anulado manualmente via el docker CLI**



La instrucción ENTRYPOINT

- Define el punto de entrada con el comando que el container correrá cuando se crea
- Los argumentos de runtime son enviados como parámetros a la instrucción ENTRTPOINT
- También posee forma de EXEC y Shell
- El contenedor funciona a modo de ejecutable



Copiando archivos

- Cuando creamos imágenes generalmente necesitamos hacer más cosas que instalar paquetes
- Ejemplos:
 - Compilar nuestro código y ejecutar nuestra aplicación
 - Copiar archivos de configuración
 - Copiar otro contenido
- Para eso utilizamos la instrucción `COPY <SRC> ... <SRC> <DST>`



Dockerizando nuestra aplicación

- Utilizar Dockerfiles resulta esencial para lograr que nuestra aplicación se ejecute en contenedores
- Tomemos nuestra por ejemplo. Para ejecutarla necesitamos lo siguiente
 - Python
 - Librerías accesorias
 - Archivos adicionales de configuración
- Qué sucede cuando queremos llevar nuestra aplicación a otro entorno?
- Podemos hacer un docker container y utilizarlo de igual manera en todos nuestros ambientes



Especificando un directorio de trabajo

- La instrucción `WORKDIR` permite setear el directorio de trabajo para el cual las instrucciones `RUN`, `CMD`, `ENTRYPOINT` y `COPY` puedan utilizar
- **Sintaxis**

`WORKDIR /ruta/a/carpeta`

- La ruta puede ser tanto absoluta como relativa dentro del contenedor
- La instrucción puede utilizarse más de una vez



La instrucción ADD

- Posee el mismo formato que la instrucción *COPY* y ambos operan de manera muy similar
- ADD tiene la habilidad de descomprimir archivos automáticamente
- ADD puede obtener archivos de una URL (no descomprime en este caso)
- Ambas instrucciones realizan un checksum de los archivos añadidos para calcular la cache.



Otras instrucciones

- *MAINTAINER* agrega metadata al dockerfile sobre el dueño de la imagen
- *ENV* permite añadir variables de entorno al contenedor
- *LABEL* permite agregar etiquetas al contenedor



Consejos y buenas prácticas para Dockerfiles

- **IMPORTANTE:** Cada línea del dockerfile crea una imagen nueva si modifica el estado de la imagen. Es importante buscar el balance entre la cantidad de capas creadas y la legibilidad del Dockerfile
- No instalar paquetes innecesarios
- Utilizar un ENTRYPOINT por dockerfile
- Combinar comandos similares utilizando “&&” y “\”
- Utilizar el sistema de cache de manera inteligente

<https://docs.docker.com/engine/reference/builder/>



Compartir nuestras imágenes

- Para compartir nuestras imágenes tenemos las siguientes opciones
 - Utilizar docker hub
 - Utilizar nuestro propio servidor de registro interno
 - Los comandos docker export y docker import
- Las imágenes en Docker hub pueden ser públicas o privadas

<https://hub.docker.com>



Etiquetando las imágenes

- Utilizado para renombrar imágenes locales antes de compartirlas en un repositorio
- Sintaxis:
`docker tag [ID imagen] [repo:tag]`
`docker tag [local repo:tag] [repo:tag]`
- Una imagen puede tener múltiples etiquetas



Borrando imágenes

- Utilizar el comando `docker rmi`
- `docker rmi [ID imagen]`
o
`docker rmi [repo:tag]`



Ejemplo

HAGAMOS UNA IMAGEN DE NUESTRA APLICACIÓN DE
EJEMPLO

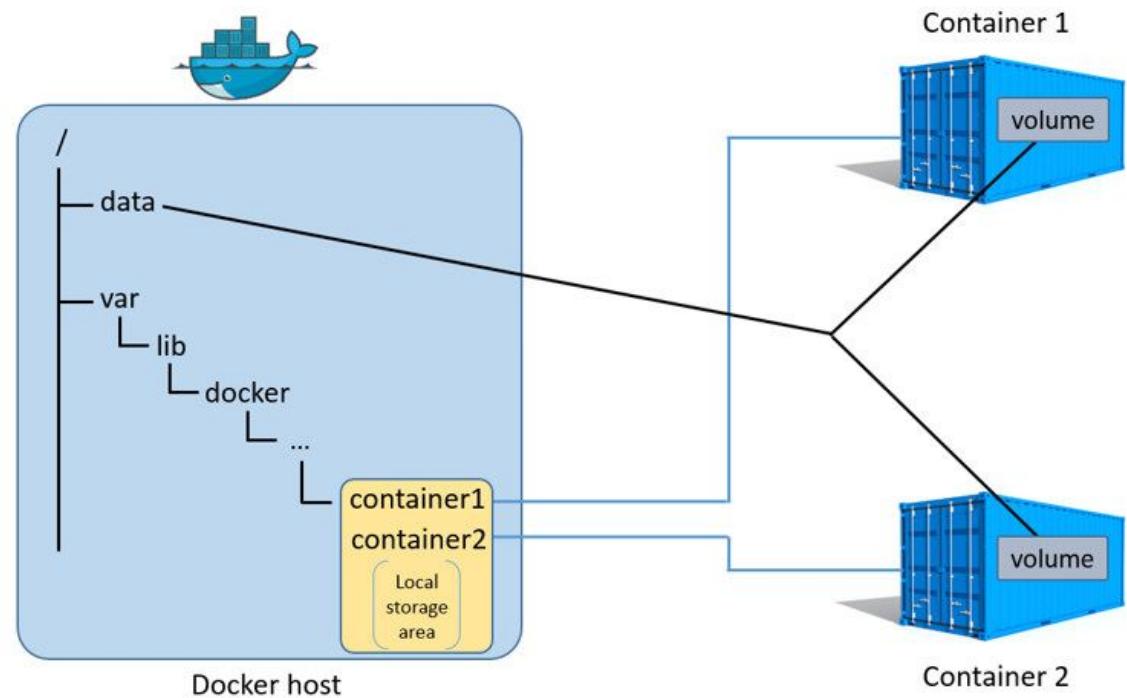


Volúmenes

- Un directorio designado en el contenedor en el cual se persiste información independientemente del ciclo de vida del contenedor
- Los cambios en un volumen son excluidos cuando se guarda una imagen
- La información se persiste aunque se elimine el contenedor
- Pueden estar mapeados a un directorio del host.
- Pueden compartirse entre contenedores
- Las instrucciones `RUN` del Dockerfile no modifican la información de los volúmenes

Volúmenes

- Separa información del contenedor vs persistente
- Sirven para compartir datos entre contenedores
- Tienen mejor performance ya que no utilizan COW



Utilizando volúmenes

- El comando `docker volume` contiene sub comando para gestionar los volúmenes en docker
- Los comandos son:
 - `docker volume create`
 - `docker volume ls`
 - `docker volume inspect`
 - `docker volume rm`
- Para montar un volúmen se utiliza la bandera `-v` en el comando `docker run` (puede utilizarse más de una vez)
- Se pueden montar directorios del host con el formato `-v [host path] : [container path] : [ro | rw]`

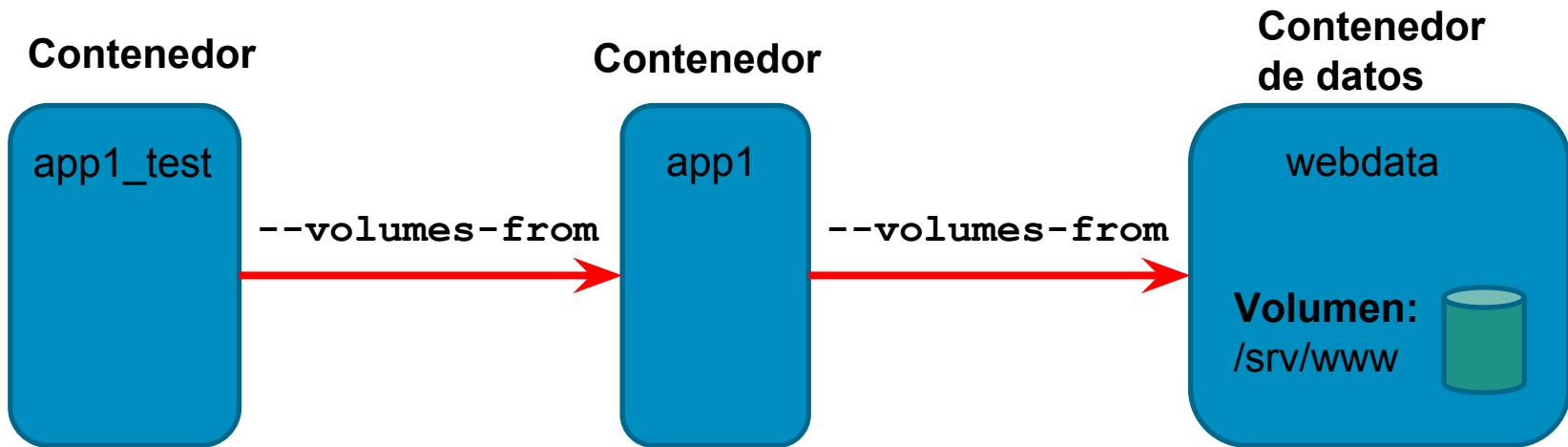


Volúmenes en el Dockerfile

- La instrucción VOLUME crea un punto de montaje de volúmenes
- Se pueden especificar los argumentos como JSON o string
- Con este método no se pueden mapear archivos del host (debido a que el dockerfile puede ejecutarse en cualquier equipo)
- Los volúmenes son inicializados cuando el contenedor inicia con la data existente en el directorio
- Los volúmenes se crean con nombre aleatorio.



Contenedores de datos



- El contenedor `app1` va a montar `/srv/www` de `appdata`
- El contenedor `app1_test` va a montar todos los volúmenes de `app1`, que es el volumen original `/srv/www` de `appdata`



Ejemplo

UTILICEMOS VOLÚMENES EN NUESTRA APLICACIÓN DE
EJEMPLO



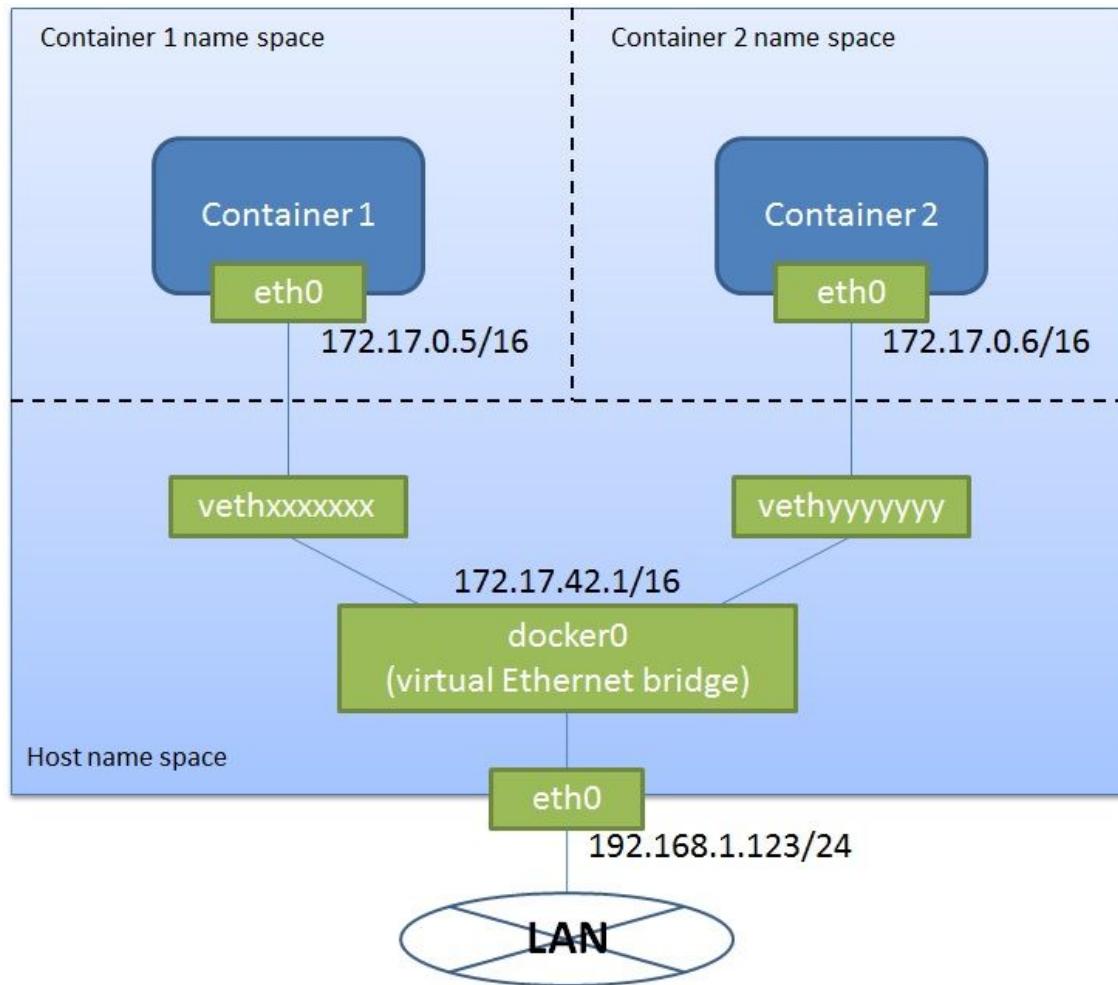
Modelo de Red de Docker

- Cuando docker inicia, crea una interfaz virtual `docker0` en el equipo de host
- `docker0` es asignado un rango de IP de la subnet privada definida por la RFC 1918

```
marcos@XPS:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether e8:b1:fc:00:88:27 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.110/24 brd 192.168.0.255 scope global dynamic wlan0
        valid_lft 4585sec preferred_lft 4585sec
    inet6 fe80::eab1:fcff:fe00:8827/64 scope link
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:c7:24:3e:a2 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
        valid_lft forever preferred_lft forever
```



Modelo de Red de Docker



El comando docker network

- El comando `docker network` nos permite interactuar con las redes en docker y los contenedores dentro de ellas
- Los sub commandos son:
 - `docker network create`
 - `docker network connect`
 - `docker network ls`
 - `docker network rm`
 - `docker network disconnect`
 - `docker network inspect`
- Utilizar la bandera `--net` para especificar una red cuando se crea un contenedor
- Utilizando la bandera `--link` se puede acceder a los contenedores por nombre.

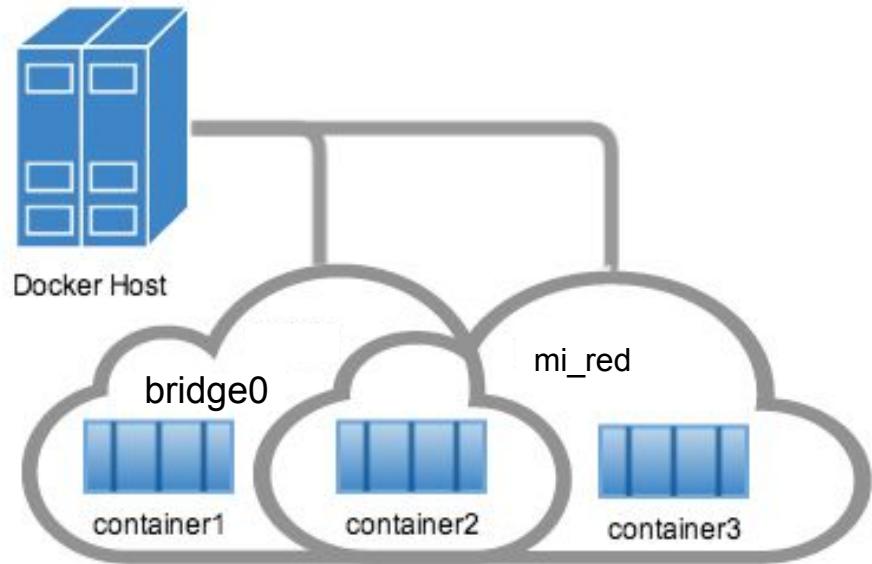
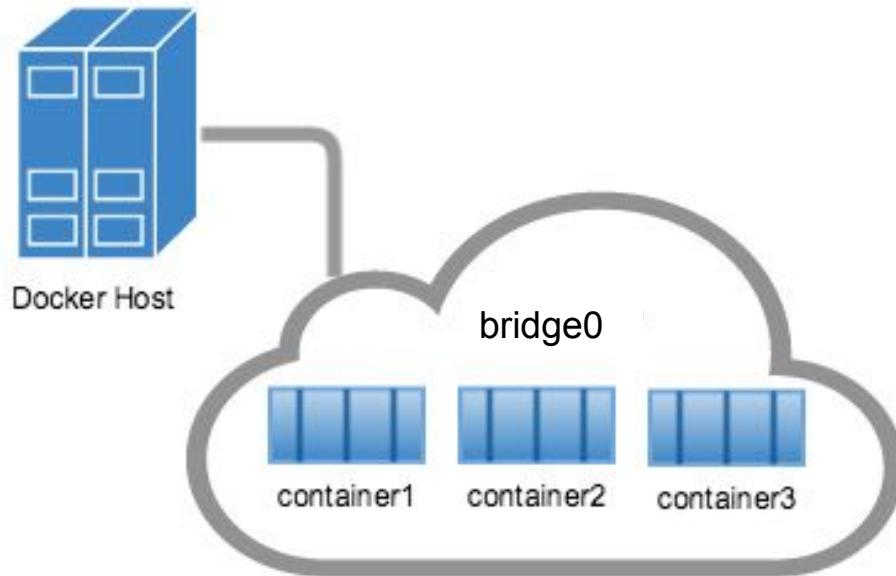


Creando nuestra red

- Utilizar el comando `docker network create`
- Hay 2 tipos de redes que pueden crearse
 - Bridge
 - Overlay
- Una red bridge es igual a la red `docker0` (la red utilizada por defecto en docker)
- Una red Overlay permite desplegar contenedores en diferentes hosts físicos y que los mismos se comuniquen de manera directa.
- Los contenedores se pueden conectar a más de una red mediante el comando `docker network connect`

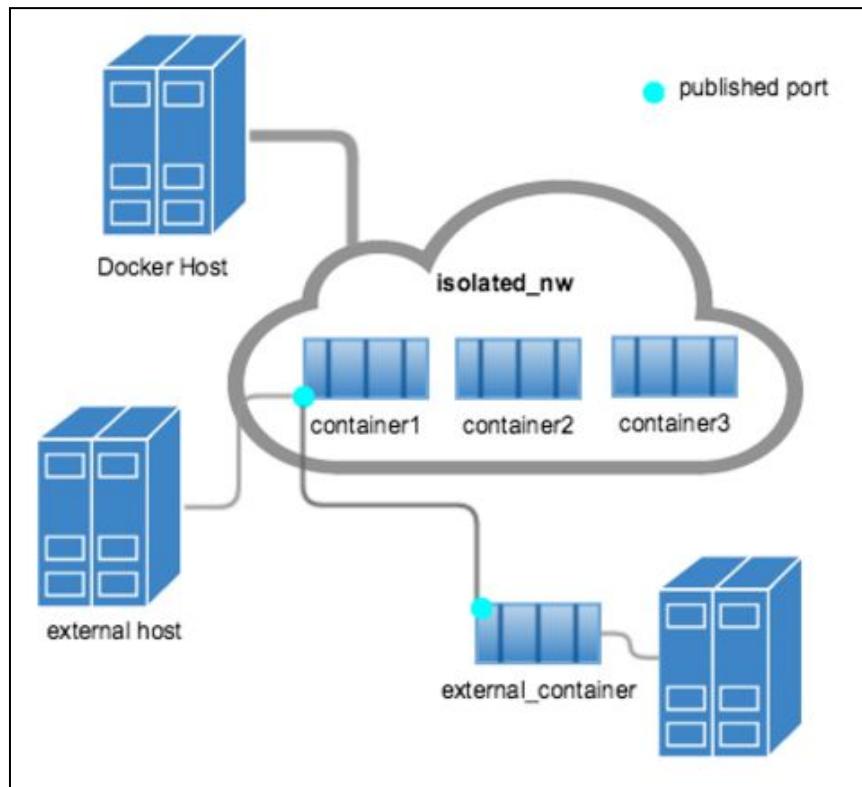


Visualizando nuestra red



Exponiendo contenedores a una red externa

- Los contenedores en una red bridge sólo pueden ser accedidos por otros contenedores en la misma red
- Para hacer que un contenedor pueda ser accedido desde el exterior, es necesario mapear sus puertos mediante el host
- El contenedor puede ser accedido mediante el puerto mapeado en el host



Ejemplo

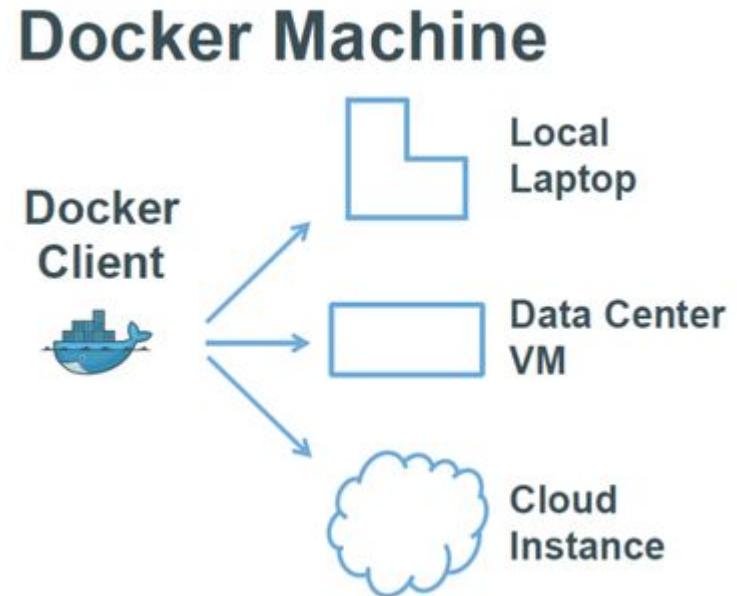
AGREGUEMOS RED A NUESTRA APLICACIÓN DE
EJEMPLO



Docker Machine

“Docker machine es una herramienta para provisionar hosts de Docker y configurar el Engine en los mismos

- Crea hosts de docker adicionales en el entorno local
- Crea hosts de docker en clouds providers(e.j. Amazon AWS, DigitalOcean etc...)
- Machine crea el servidor, instala y configura Docker.



<https://github.com/docker/machine/releases/>

Docker Machine

- Usando el comando **docker-machine create** y especificar el driver a usar
- El driver permite a **docker-machine** utilizar el entorno deseado
- Sintaxis

```
docker-machine create --driver <driver> <hostname>
```



Docker Machine - AWS

- Para crear un host en AWS es necesario
 - AWS access key
 - AWS secret key
 - El VPC ID para la instancia donde correrá docker
- La imagen utilizada por defecto es Ubuntu 14.04 LTS

```
docker-machine create
  --driver amazonec2 \
  --amazonec2-access-key <AWS access key> \
  --amazonec2-secret-key <AWS secret key> \
  --amazonec2-vpc-id <VPC ID> \
testhost
```

- Otros providers
 - DigitalOcean, GCE, Azure, SoftLayer, Rackspace, VMware, Openstack



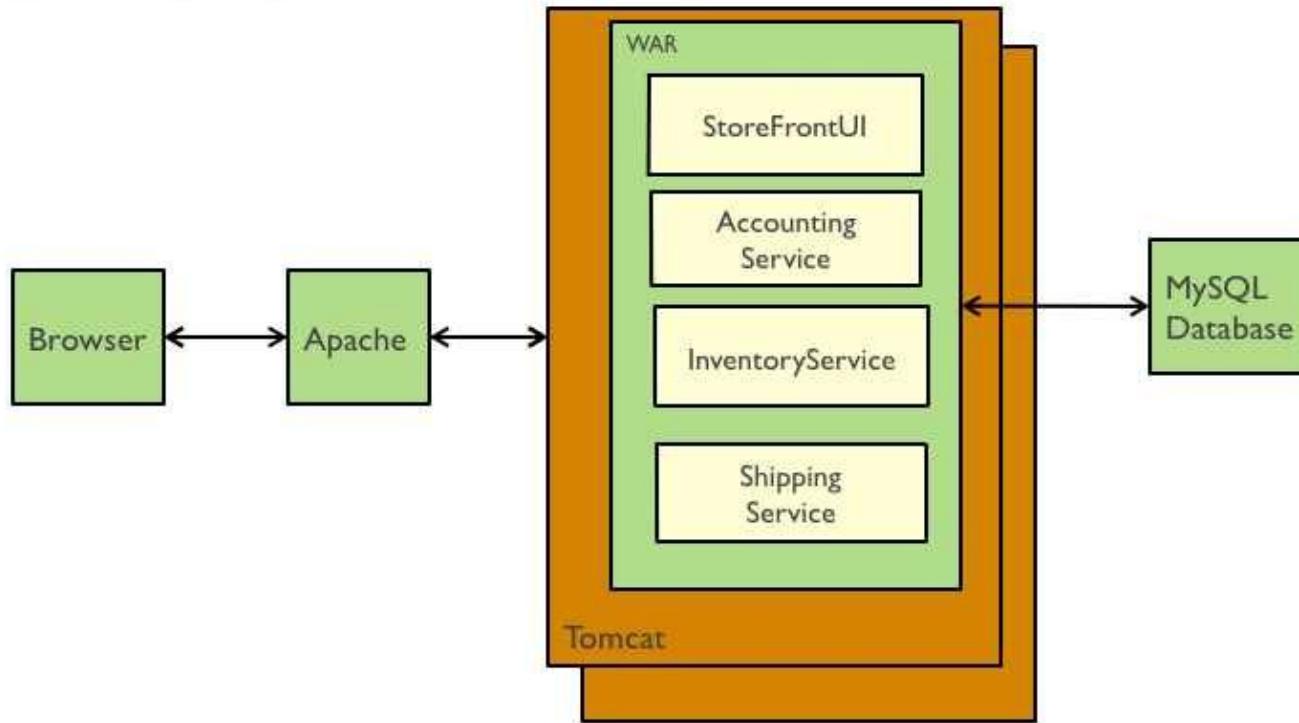
Docker machine - extras

- Existen 2 métodos para conectarse a un host con docker-machine
 - Utilizar docker-machine ssh
 - Setear las variables de entorno para apuntar al host de docker
 - **Hack:** Usar la clave ssh manualmente (no recomendado)
- El comando `env` retorna las variables necesarias para que nuestro cliente de Docker se conecte al host creado
- Los hosts pueden ser detenidos o iniciados con el comando `docker-machine stop/start/restart`
- Utilizar `docker-machine inspect <name>` para obtener más información del host provisionado
- `docker-machine rm` elimina nuestro host creado.



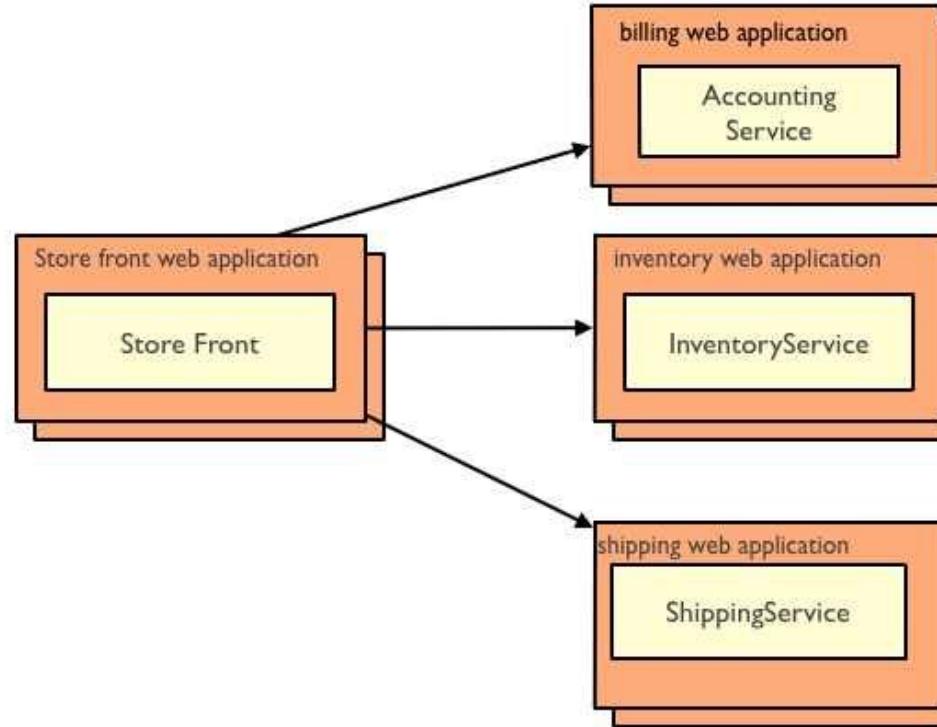
Microservicios con Docker

Traditional web application architecture

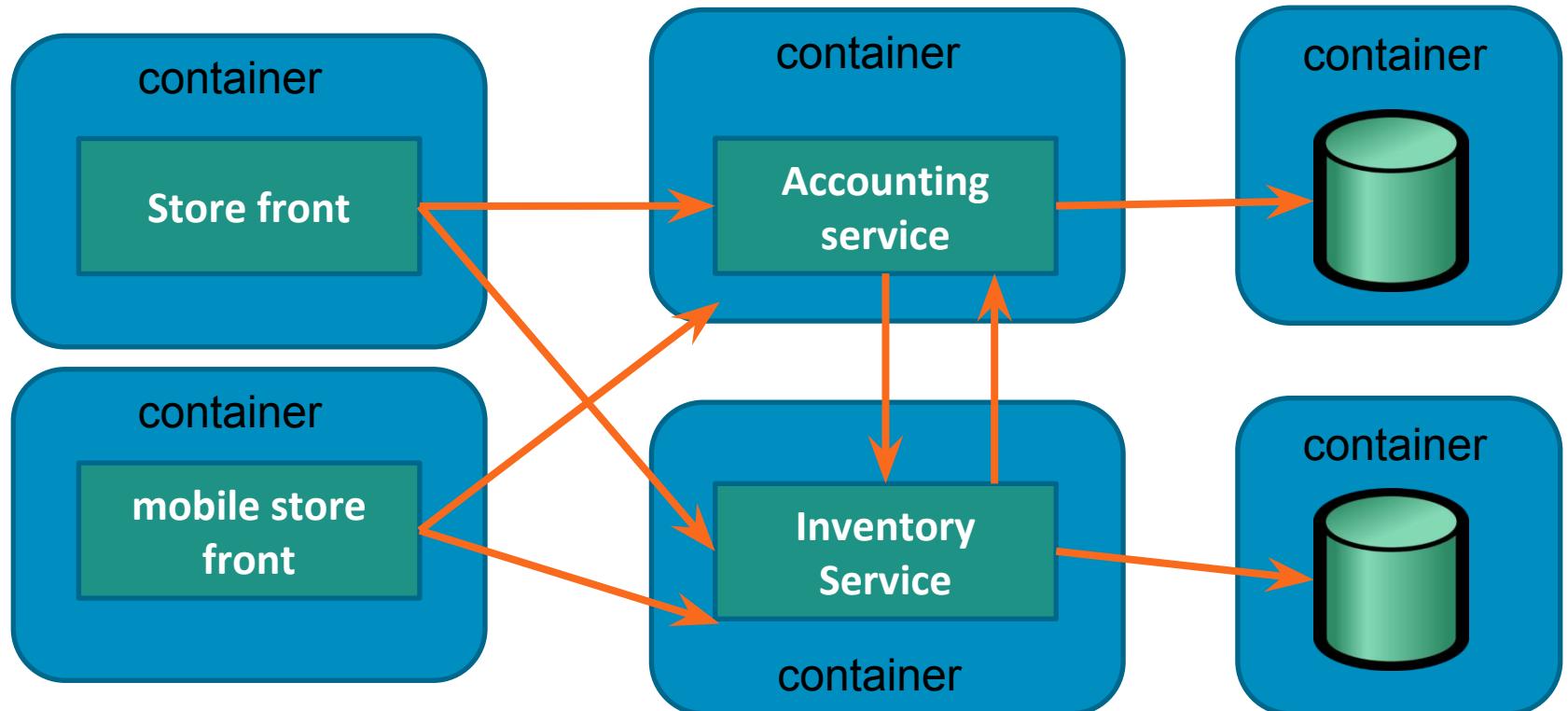


Microservicios con Docker

Y axis scaling - application level



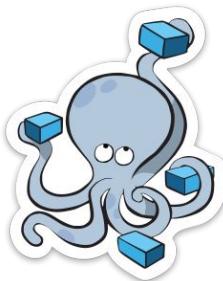
Docker y microservicios



Docker Compose

“Compose es una herramienta para crear y administrar aplicaciones multi-contenedor”

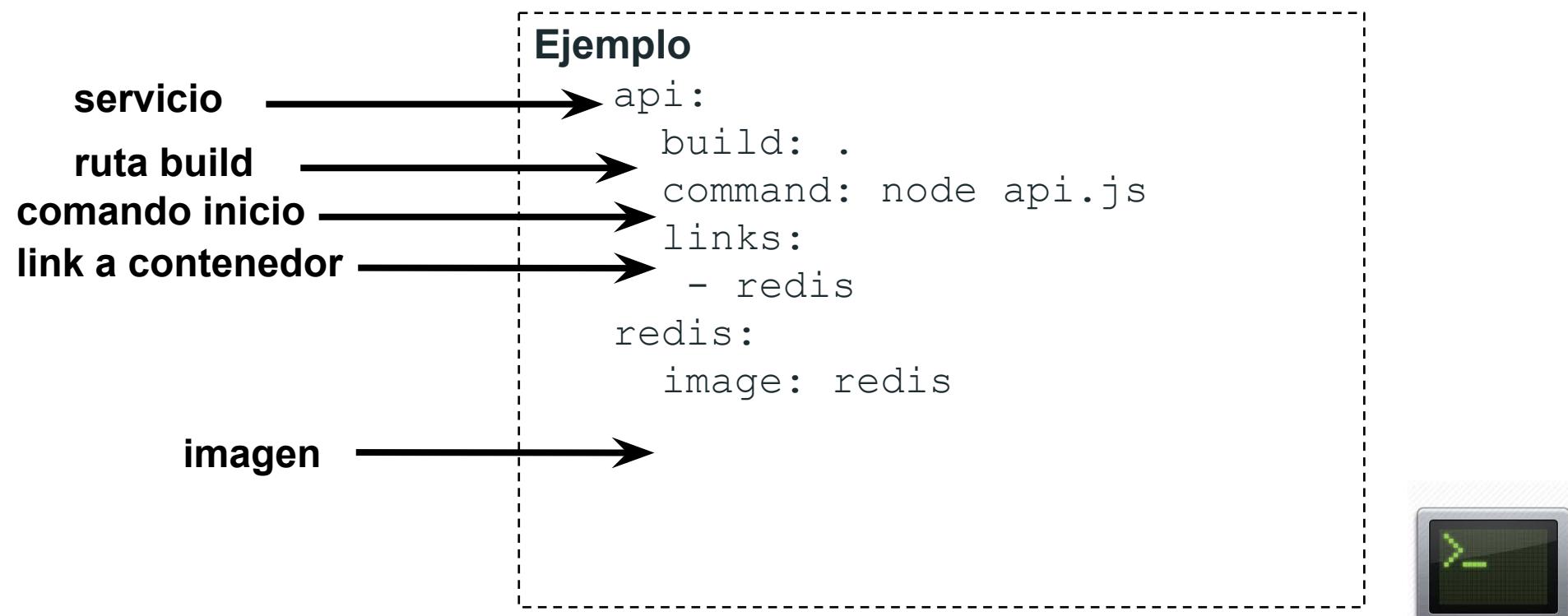
- En una arquitectura de microservicios existen demasiados servicios, por ende múltiples componentes para correr
- Resulta tedioso ejecutar `docker run` por cada componente y luego manualmente orquestar todo
- **Docker Compose** al rescate



<https://github.com/docker/compose/releases>

Docker Compose

- **docker-compose.yml** define los servicios que componen la aplicación
- Cada servicio contiene las instrucciones para construir y ejecutar el contenedor



Docker Compose

- Utilizar `docker-compose up` para iniciar nuestra aplicación
- El comando up:
 - Construye la imagen para cada servicio
 - Crea e inicializa los contenedores
- Compose automáticamente se da cuenta cuales contenedores iniciar primero
- Los contenedores pueden correr en background (bandera -d)
- Los nombres de los contenedores tendrán el formato
`<proyecto>_<servicio>_<numero>`
- Otras acciones: logs/start/stop/restart/pull/rm



Docker Compose - extras

- Muchos de los parámetros en el archivo `docker-compose.yml` tienen su equivalente como instrucción utilizando el Docker engine
- Las opciones especificadas en el Dockerfile son respetadas por compose y no es necesario redefinirlas
- Ejemplo
 - Exponemos el puerto 8080 en el Dockerfile utilizado por tomcat
 - Necesitamos correr tomcat mediante un servicio via Compose
 - No es necesario especificar el parámetro `ports` siendo que se encuentra definido en el Dockerfile
- `docker-compose build` solamente generará las imágenes de aquellos Dockerfiles q hayan sido modificados
- Pueden utilizarse variables de entorno en compose mediante la sintaxis `$(MI_VARIABLE)`

<https://docs.docker.com/compose/compose-file/>



Docker Compose - escalando servicios

- En una arquitectura de microservicios, generalmente tenemos la ventaja de poder escalar un servicio específico para soportar la carga
- **Ejemplo:** Si el servicio de Orders recibe demasiado tráfico, necesitamos escalarlo para satisfacer así la demanda.
- Docker compose posee el comando `scale` convenientemente para tal fin.
- **Sintaxis:**
`docker-compose scale <servicio>=<número_instancias>`
- Permite escalar los servicios de manera incremental y decremental
- Los comandos siguientes serán ejecutados contra todos las instancias del servicio especificado por defecto.



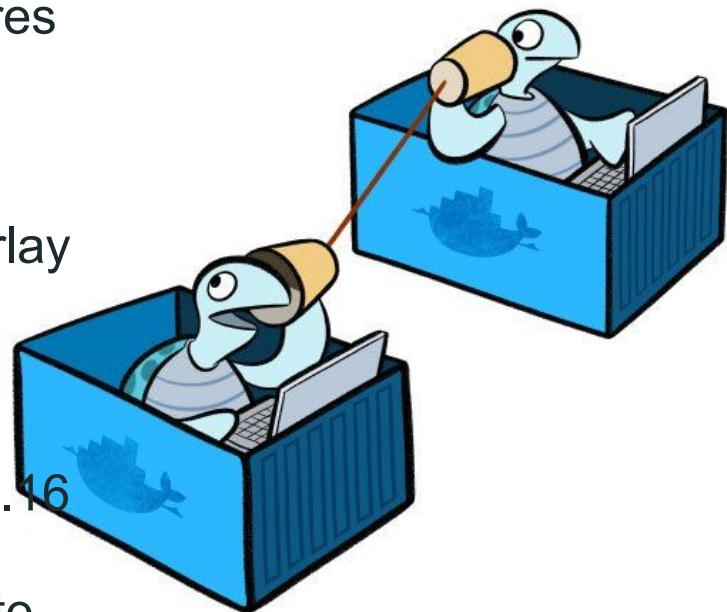
Ejemplo

UTILICEMOS COMPOSE EN NUESTRA APP DE EJEMPLO

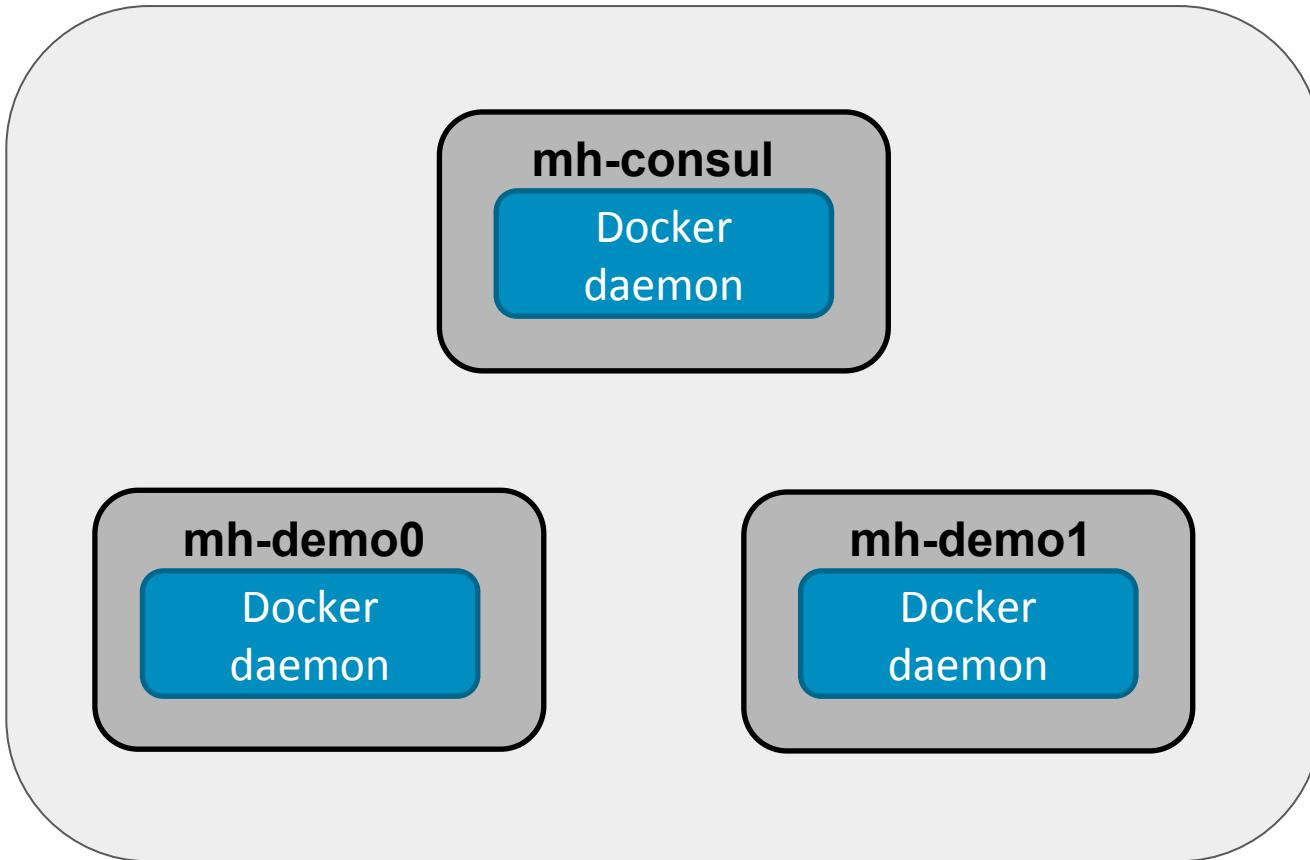


Multi-host networking

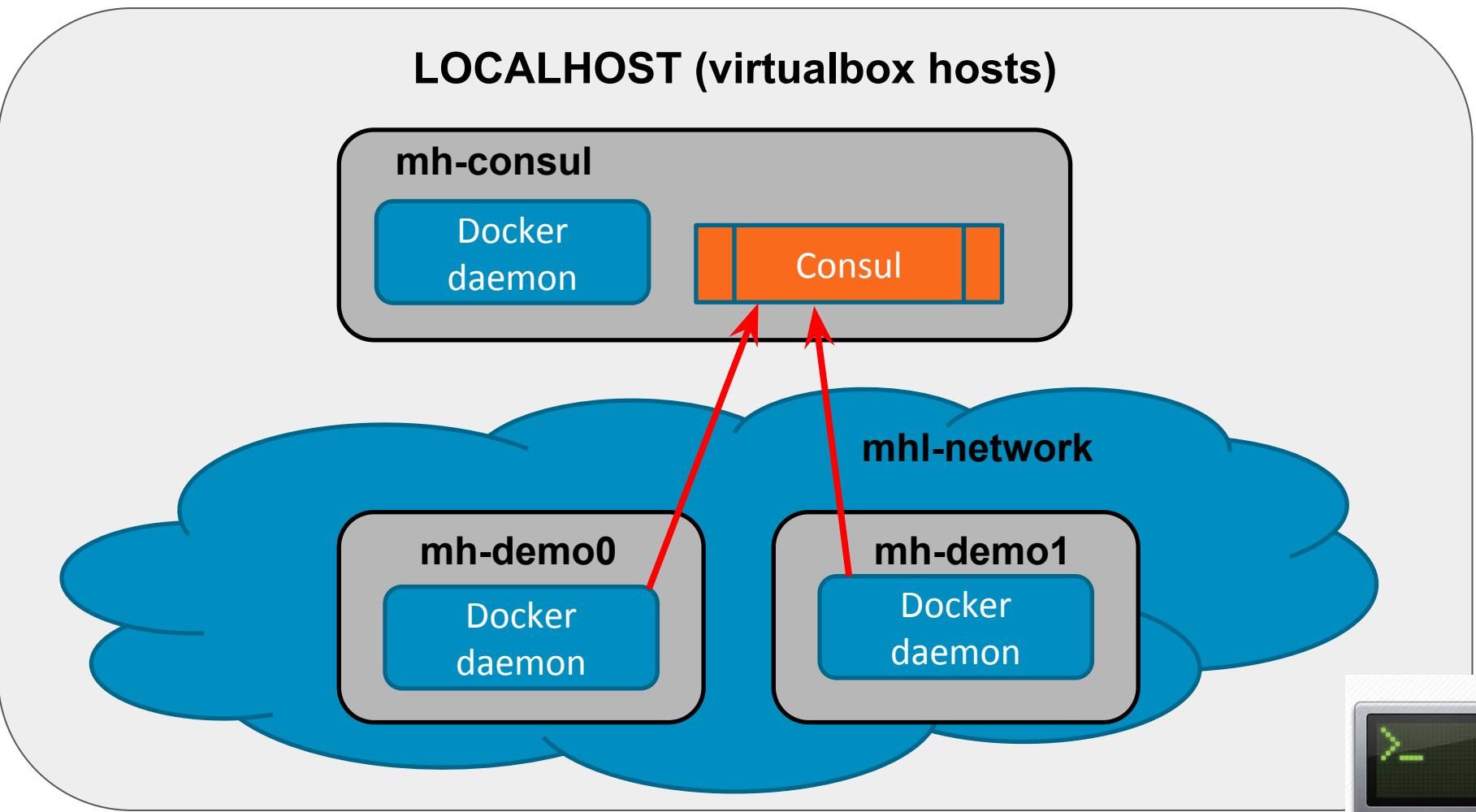
- Los contenedores que corren en diferentes hosts no pueden comunicarse entre sí a menos que expongan sus puertos mediante el host
- Multi-host networking habilita a que contenedores en diferentes hosts puedan comunicarse sin necesidad de exponer sus puertos
- El Docker engine permite realizar multi-host networking nativamente mediante el driver overlay de red
- Los requerimientos para una red overlay son:
 - Acceso a un key-value store compartido
 - Todos los hosts deben soportar un kernel 3.16 o superior
 - El Docker Engine configurado correctamente



Multi-host networking (ejemplo)



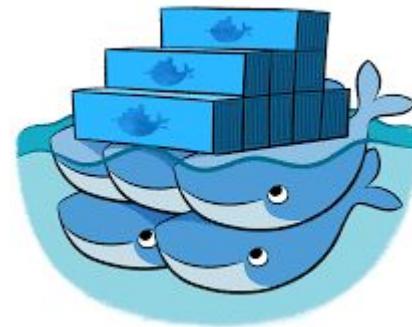
Multi-host networking (red)



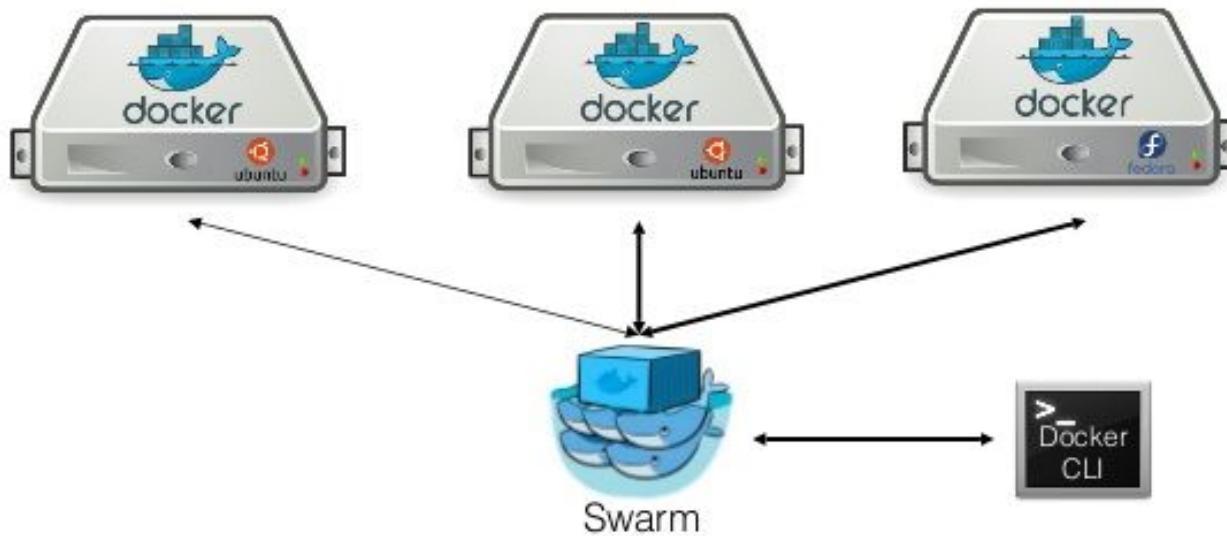
Docker swarm

“Docker swarm permite formar clusters de Docker hosts y decide dónde lanzar los contenedores (scheduler)”

- Presenta varios hosts de Docker como si fueran uno solo
- Permite distribuir contenedores por varios equipos dentro del cluster
- Utiliza la API standard de Docker
- Incluye una política simple de programación de contenedores y descubrimiento



Docker swarm

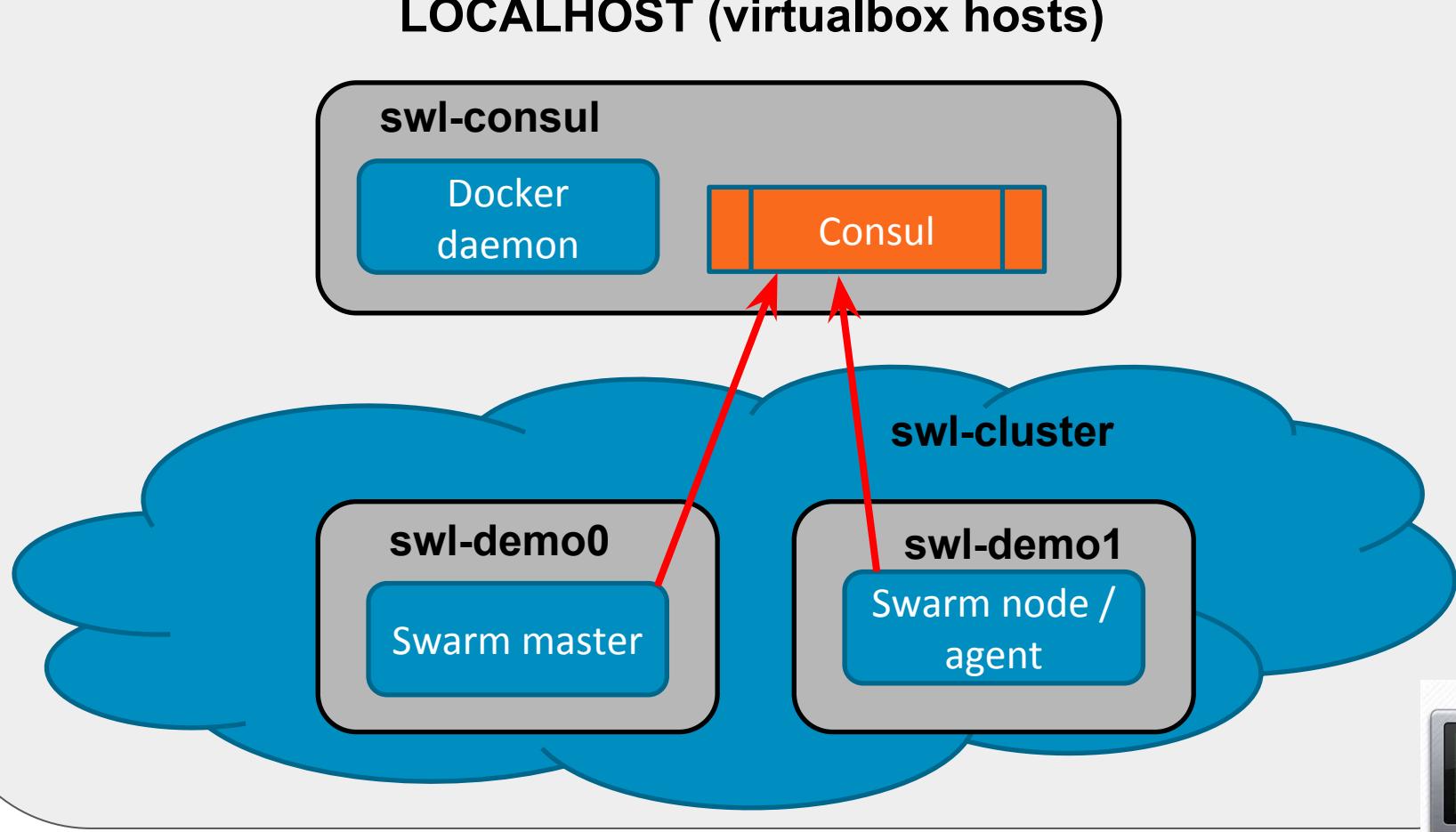


- Soporta múltiples formas de discovery(hosted / archivos estáticos y dockerhub)
<https://docs.docker.com/swarm/discovery/>

Docker swarm

- Tiene 2 métodos de instalación
 - Instalar el binario de swarm
 - Utilizar la imagen oficial de swarm del hub
- En caso de usar el binario es necesario instalarlo en todos los miembros que forman el cluster (<https://github.com/docker/swarm>)
- Utilizar la imagen de swarm es conveniente ya que los hosts ya tienen Docker instalado y configurado (<https://registry.hub.docker.com/u/library/swarm/>)
- Utilizar la imagen también nos ayuda a clear nuestro cluster de swarm **utilizando hosted discovery** y configurar el manager y los agentes correspondientes.

Docker swarm



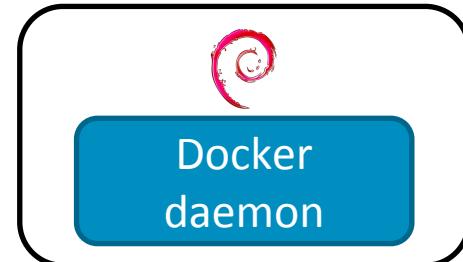
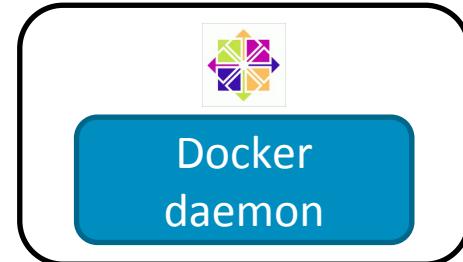
Docker swarm - Estrategias

- Docker swarm rankea los nodos en función de diferentes estrategias / algoritmos
- Cuando se ejecuta un contenedor, Swarm lo correrá en el nodo con mayor ranking
- El ranking se calcula en función de la estrategia seleccionada
- Las estrategias son:
 - Spread (estrategia por defecto) - Rankea según **la cantidad** de contenedores
 - Binpack - Intentar llenar el nodo **con menos recursos disponibles** antes de utilizar el siguiente (CPU/RAM)
 - Random

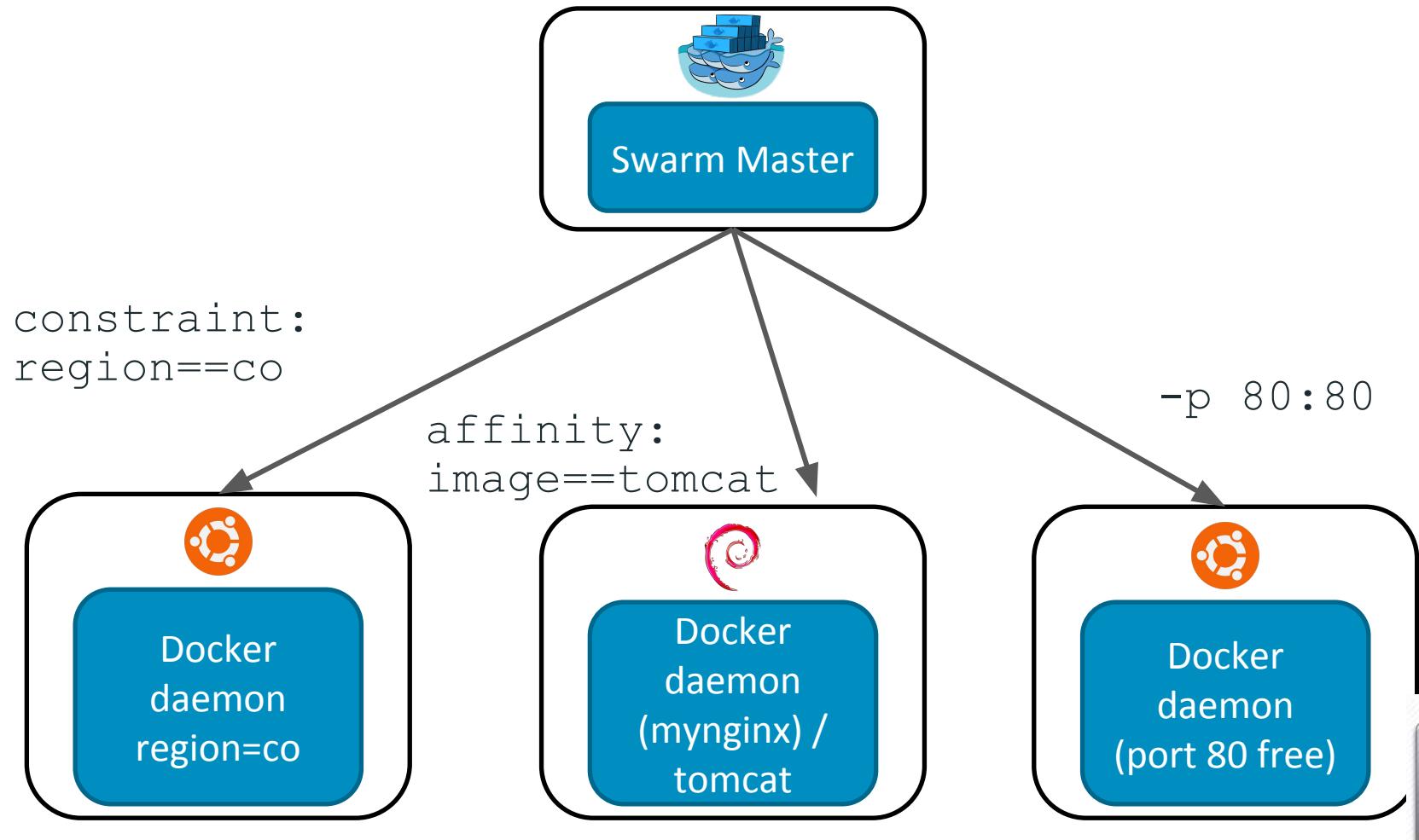


Docker swarm - Filtros

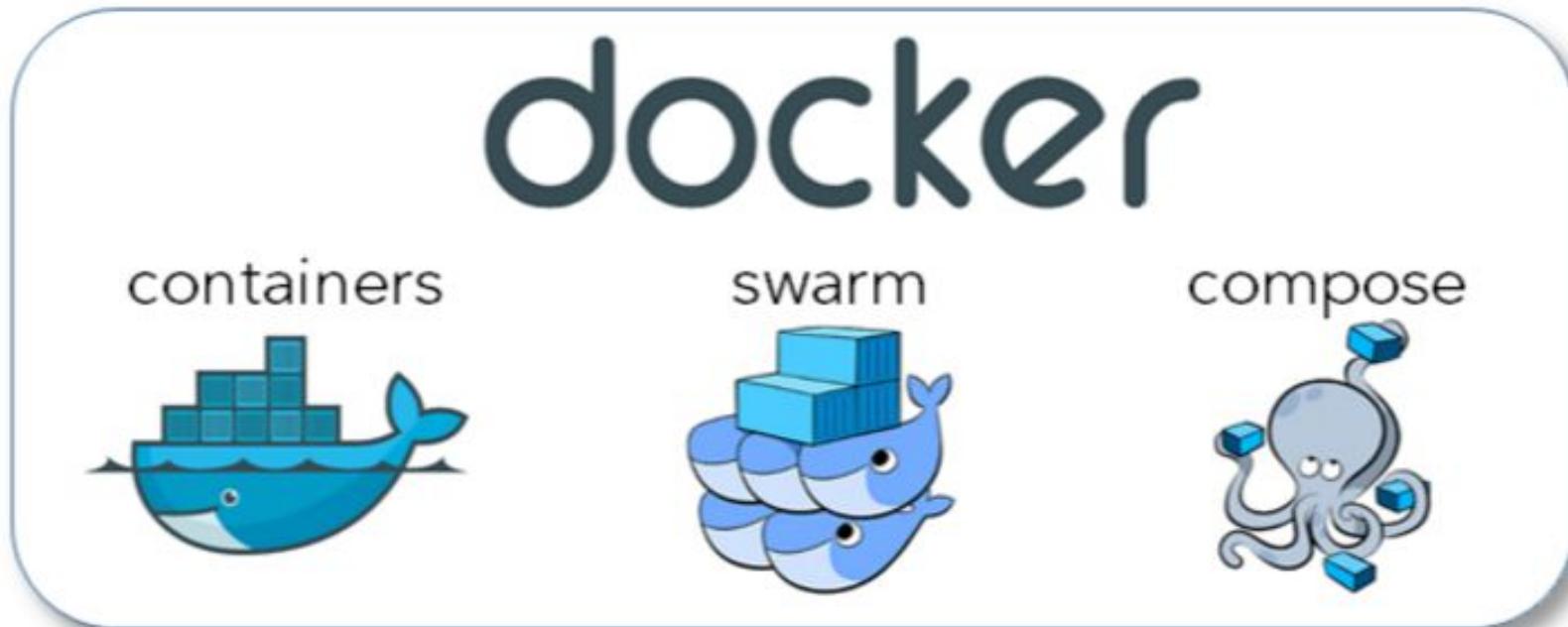
- Los filtros se utilizan para indicar a Swarm qué hosts utilizar para lanzar los contenedores
- Swarm aplica el filtro **antes** de aplicar la estrategia para lanzar los contenedores
- El filtro elimina aquellos hosts que no cumple con el mismo. Luego, se corre la estrategia definida en los nodos restantes
- Existen 3 tipos de filtros, restricciones (constraint), afinidad (affinity) y puerto (port)



Docker swarm - Filtros



Docker swarm - Composing



- Utilizando Compose podemos desplegar nuestro stack aplicativo en un entorno distribuido



Ejemplo

DESPLEGUEMOS NUESTRA APLICACIÓN EN UN CLUSTER
DE DOCKER

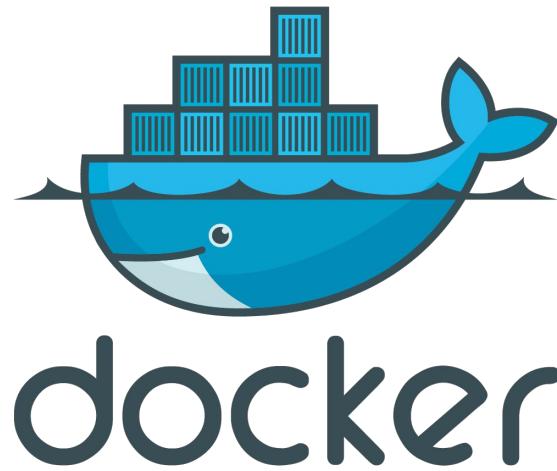


Docker swarm - Notas

- Todas las redes creadas dentro de swarm, son multi-host por defecto
- Cada backend de discovery debe configurarse apropiadamente (<https://docs.docker.com/swarm/discovery/>)
- En el caso de crear clusters de swarm manualmente es necesario utilizar los comandos swarm manage y swarm join



Muchas gracias!



[@marcosnils](https://twitter.com/marcosnils)