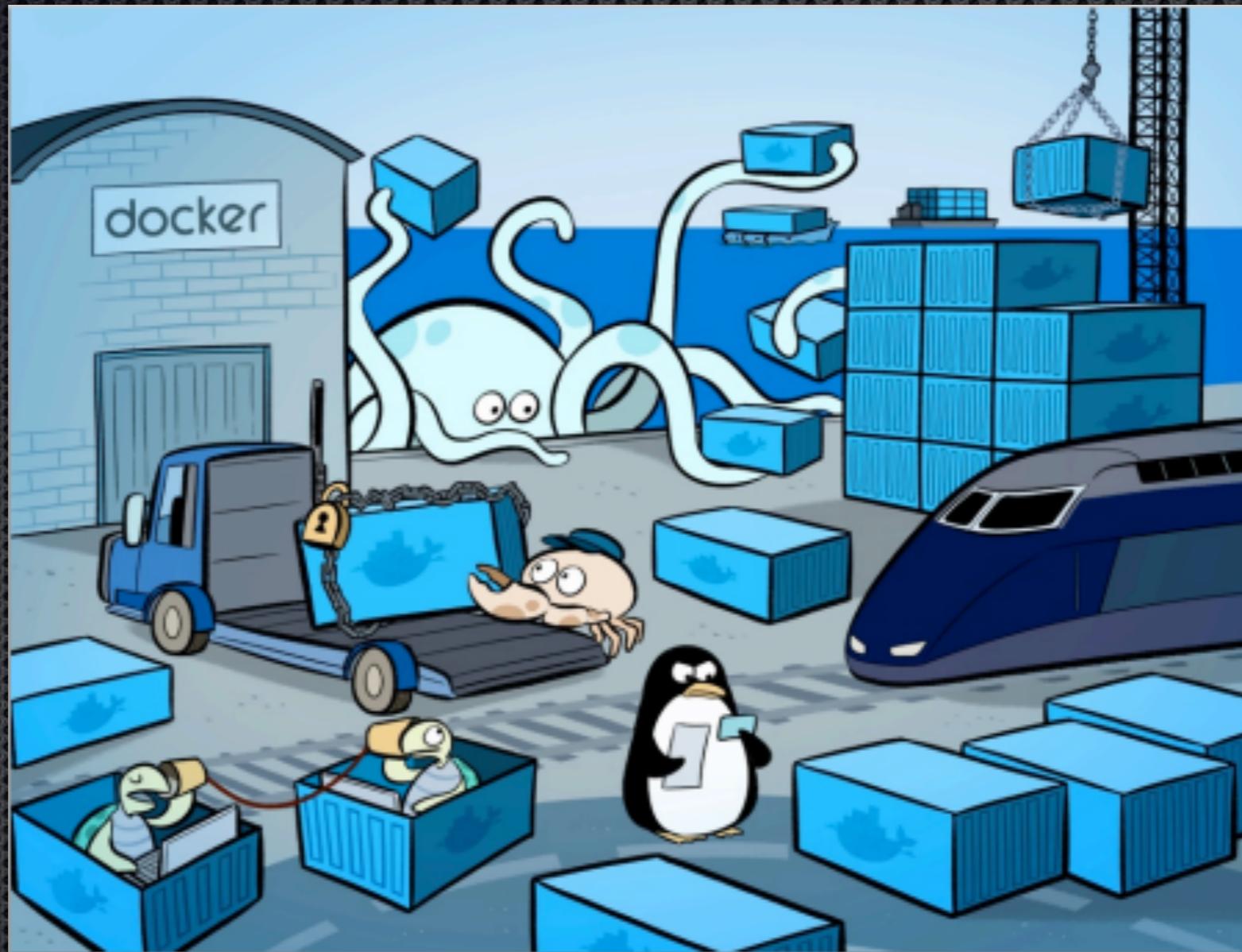


# GLab Docker



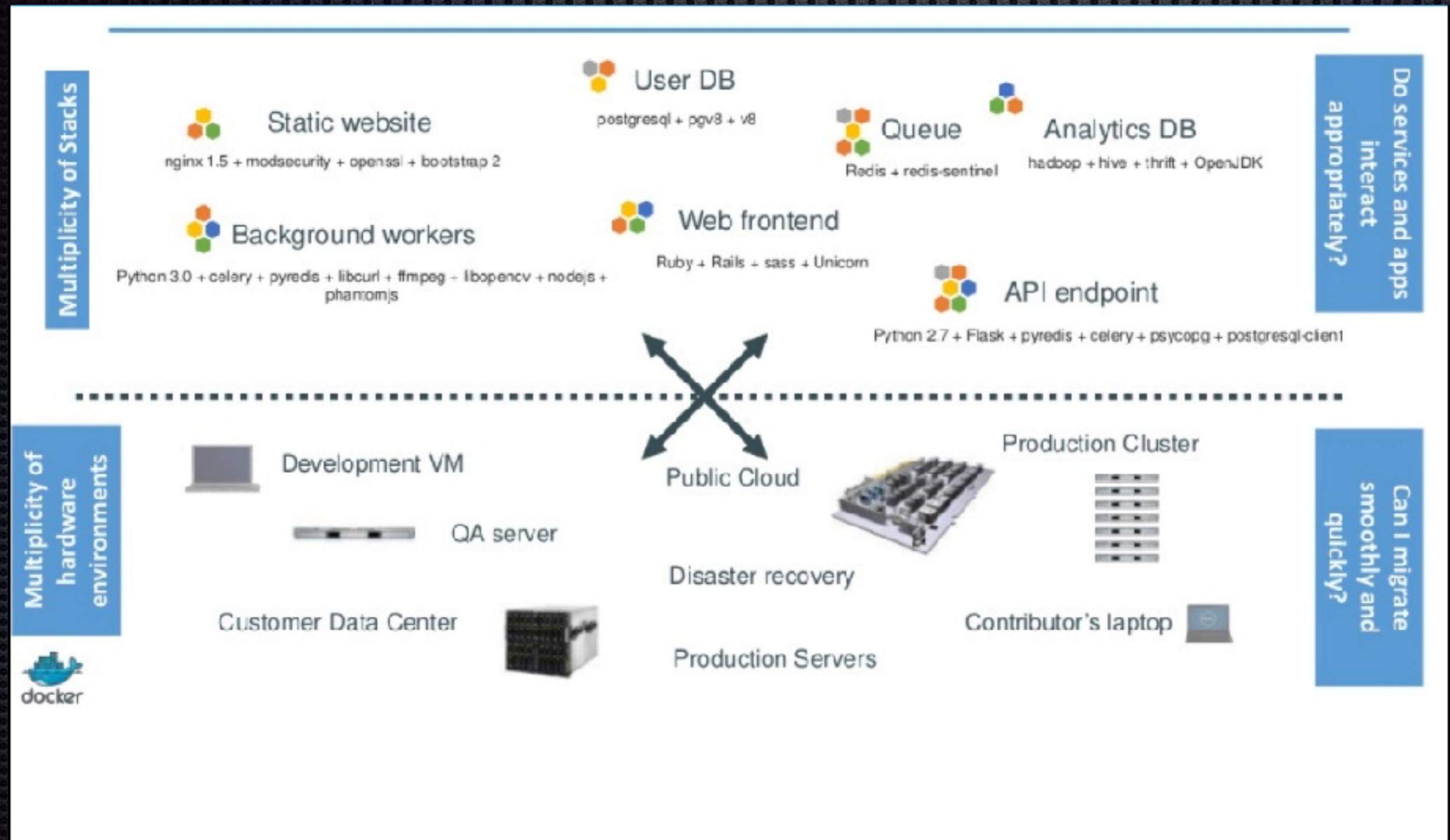
by @mario21ic

# Agenda

- Intro a Docker
- Images
- Containers
- Port forwarding
- Volumes
- Linking
- Extender un image con Dockerfile
- Docker compose

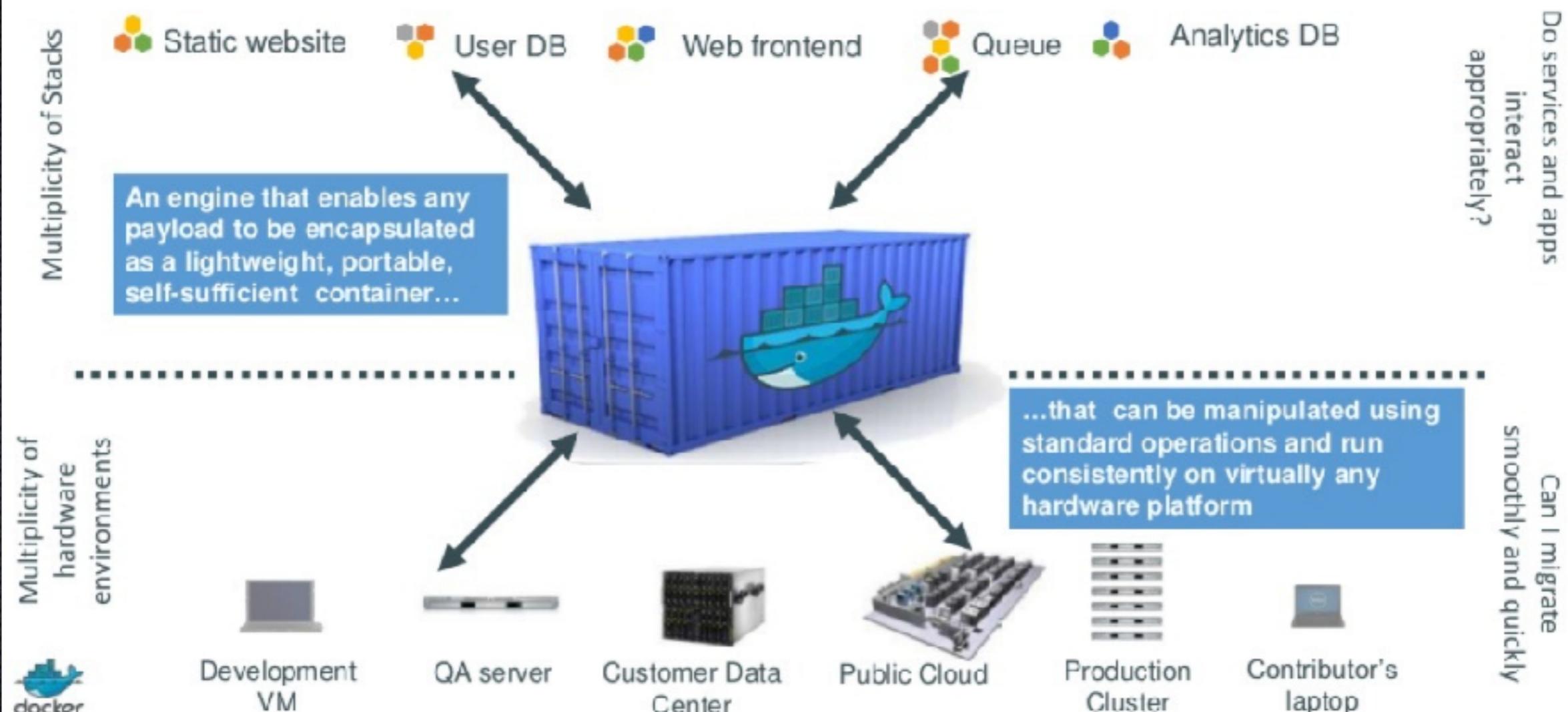
hashtag:  
[#GLabDocker](#)  
[#DockerLima](#)

# El problema

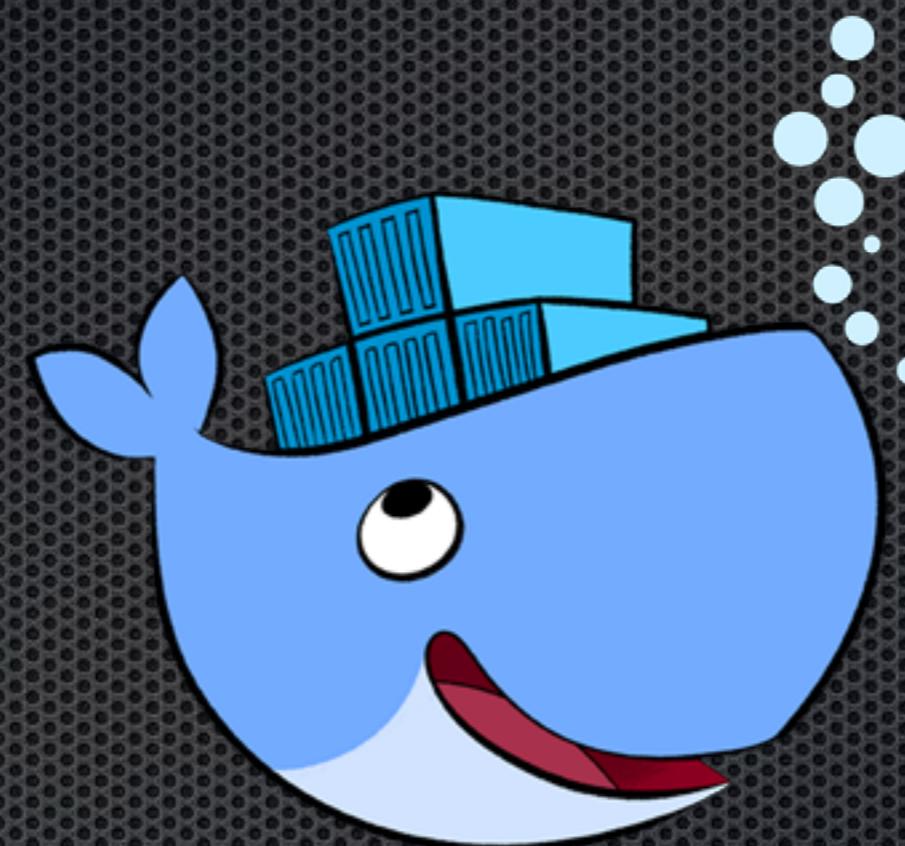


# Solución

## Docker is a shipping container system for code



# Docker

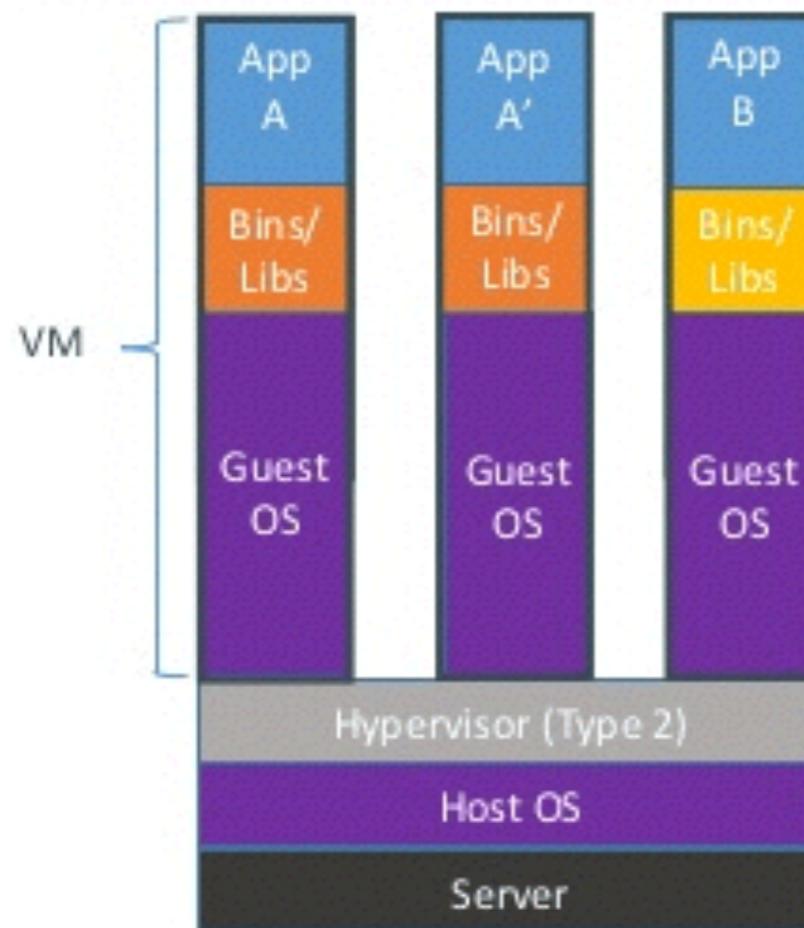


# Docker

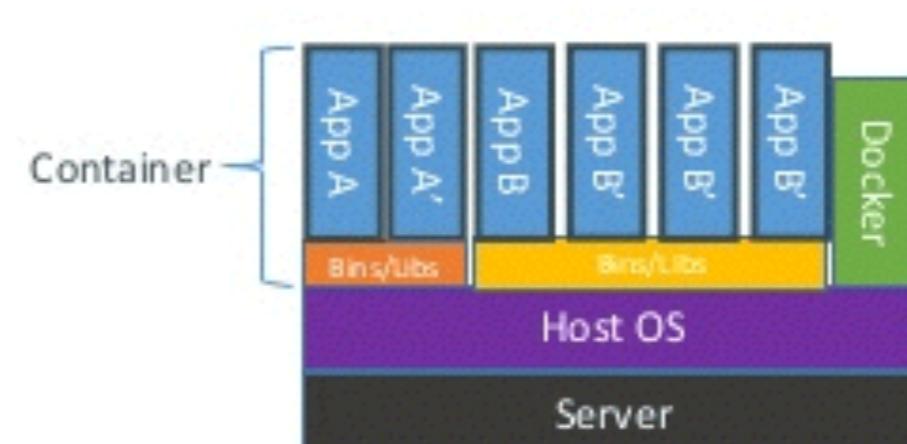
- Una plataforma abierta para aplicaciones distribuidas para developers y sysadmins
- Desarrollado inicialmente en Python y migrado a Go
- Servidor - Cliente
- Soporte para generar images a partir de Dockerfiles
- Open source

# Docker

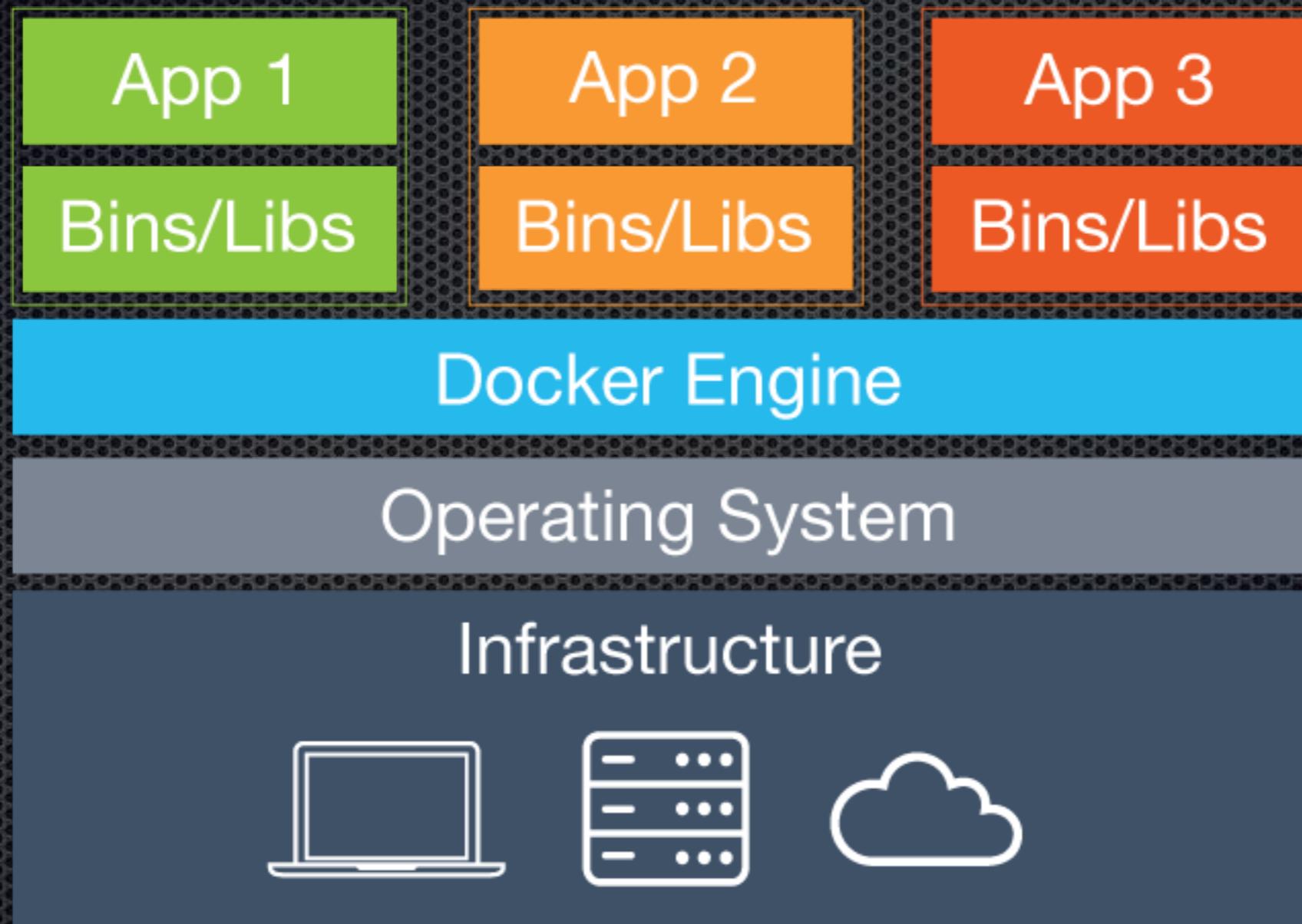
## Containers vs. VMs



Containers are isolated,  
but share OS and, where  
appropriate, bins/libraries

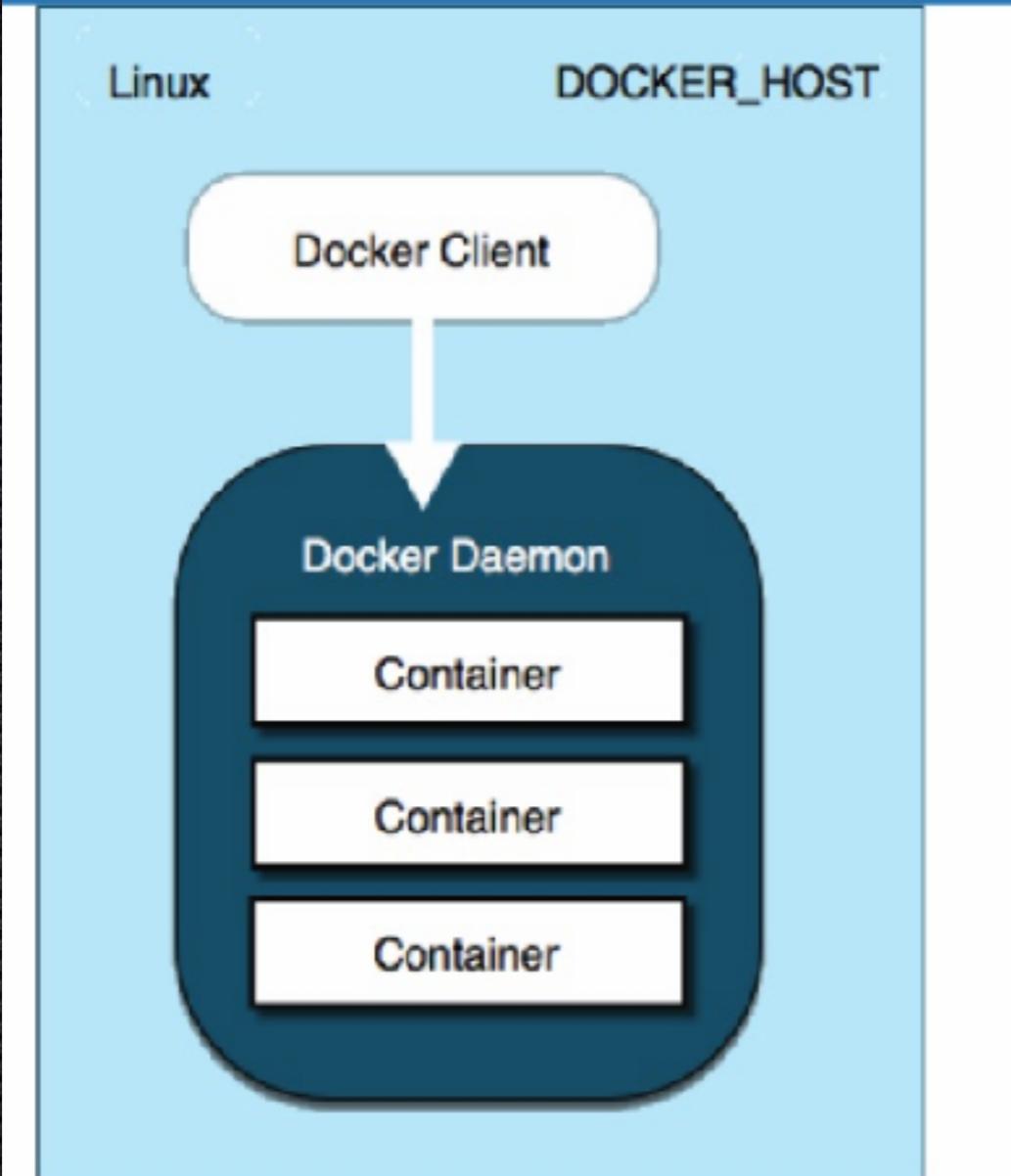


# Docker

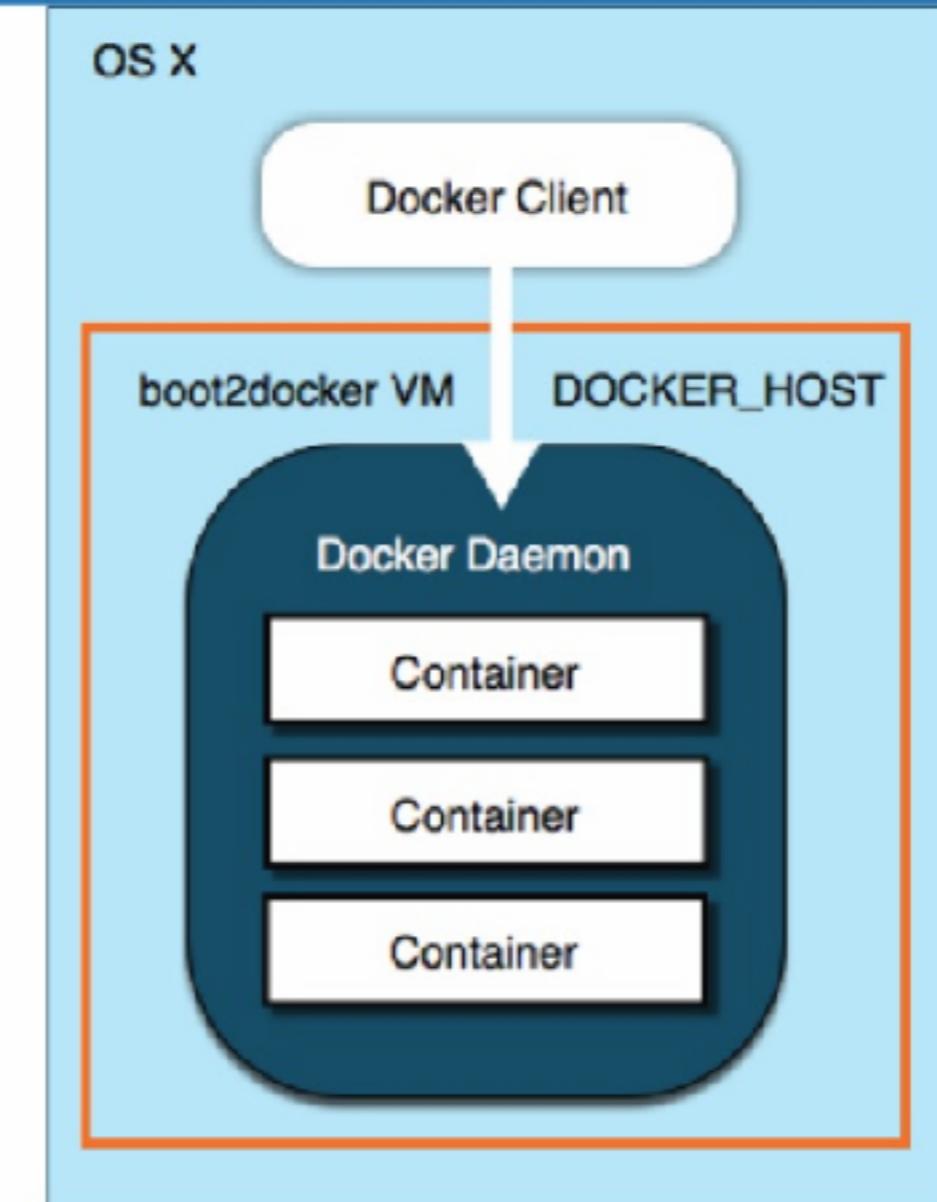


# Docker Server - Cliente

Linux



OS X/Windows



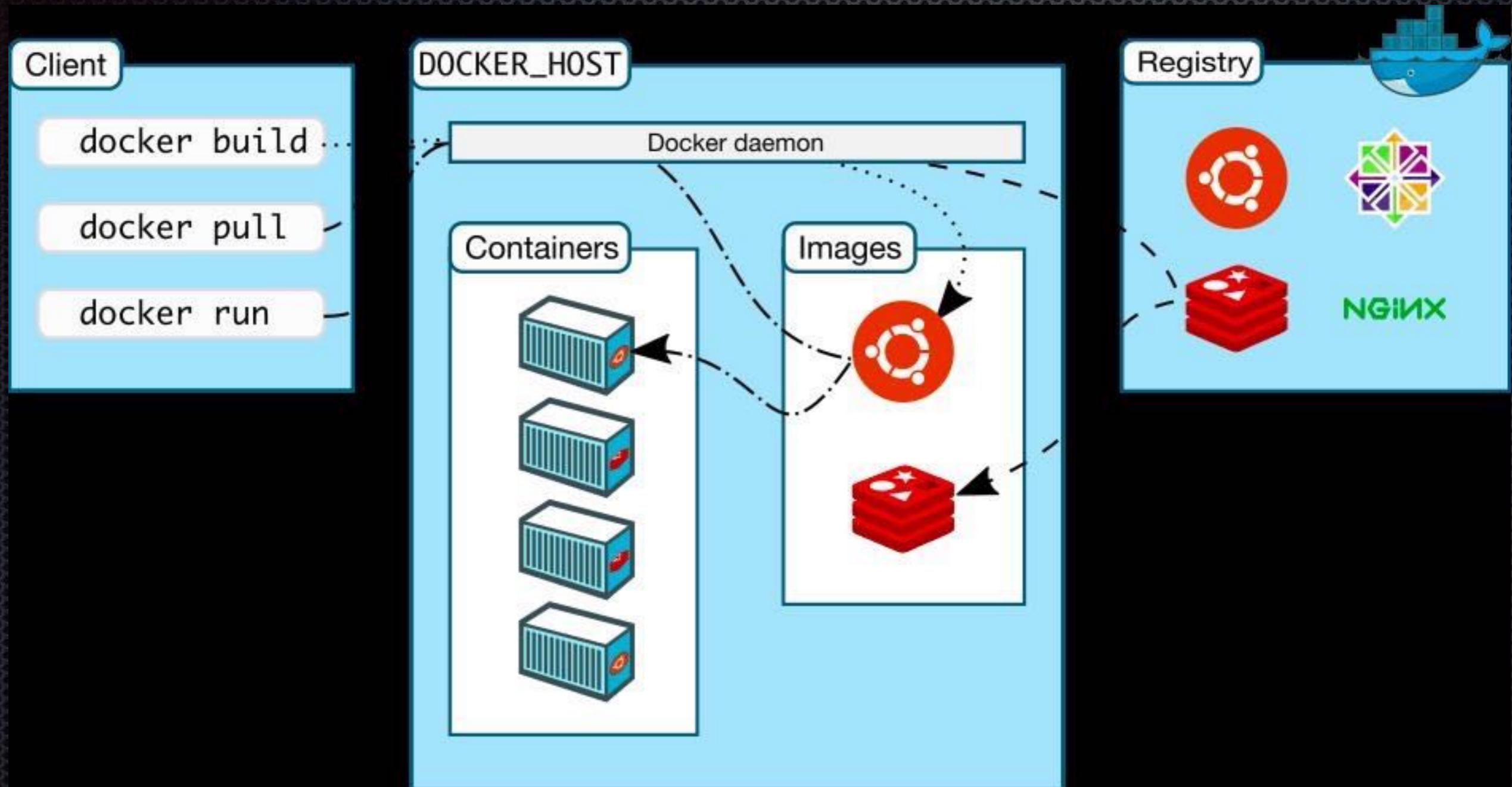
# Docker CLI - Syntax

- *docker run --name myalpine alpine:latest echo "Hello from container"*
  - cliente: ***docker***
  - comando: ***run***
  - key param: ***--name***
  - value param: ***myalpine***
  - image: ***alpine***
  - tag: ***latest***
  - comando container: ***echo***
  - parametros container: ***"Hello from container"***

# Vocabulario básico

- **Image**: capa de solo lectura, similar a un template
- **Container**: capa de lectura-escritura, instancia aislada generada a partir de un image
- **Repository**: Docker Hub donde se almacenan images
- **Volume**: para montar directorios entre host y container
- **Link**: acción de linkear un container con otro

# Docker



**VAMO A**



**DOCKERIZARNO**

Imagen creada en GeneradorMemes.com

# Requerimientos:

- Tener docker instalado correctamente, verificar mediante comando:

```
$ docker ps
```

- Clonar el repositorio del Laboratorio:

```
git clone https://github.com/GDGLima/glab\_docker.git
```

# Image: search & pull

- Buscar image:

*\$ docker search alpine*

*\$ docker search alpine --limit 5*

- Descargar un image:

*\$ docker pull alpine*

*\$ docker pull alpine:3.4*

# Image: list, tag & delete

- Listar imágenes locales:

```
$ docker images
```

- Tagar:

```
$ docker tag alpine:3.4 myalpine:mytag
```

- Eliminar:

```
$ docker rmi alpine:3.4
```

# Containers: run

- En un terminal, ejecutar:

```
$ docker run -ti alpine /bin/sh
```

```
# echo "Hello GLab" > /tmp/hello.txt
```

- En otro terminal:

```
$ docker run --name mysh -ti alpine:3.4 /bin/sh
```

```
# echo "Bye GLab" > /tmp/bye.txt
```

# Containers: ps & exec

- En un 3er terminal, ejecutar:

```
$ docker ps
```

- Ejecutar en un container:

```
$ docker exec mysh ps
```

```
$ docker exec -ti mysh sh
```

```
# ls -lha /tmp
```

# Containers: run, ps & start

- En un 4to terminal, ejecutar:

```
$ docker run --name myecho alpine:3.3 echo "Hello GLab"
```

- Listar todos los containers:

```
$ docker ps -a
```

- Iniciar un container:

```
$ docker start myecho
```

# Containers: create, attach, kill

- Create:

```
$ docker create --name echoxD debian:latest echo "xD"
```

- Iniciar:

```
$ docker start --attach echoxD
```

- En otro terminal, eliminar un container:

```
$ docker kill mysh
```

# Containers: start & delete

- Iniciar mysh:

```
$ docker start mysh
```

- Eliminar un container:

```
$ docker rm mysh
```

```
$ docker rm -f mysh
```

- Eliminar un container:

```
$ docker ps -a
```

# Port forwarding

- Del puerto 80 del container al localhost 8080:

```
$ docker run --name mynginx -p 8080:80 -d nginx
```

- Abrir browser:

<http://localhost:8080/>

# Demo Container: Volume

- En cli ejecutar:

```
$ docker run --name mynginx -v $(pwd)/html:/usr/share/nginx/html -p 8080:80 -d nginx
```

- Abrir browser:

<http://localhost:8080/>

# Demo Container: Volume

- En cli ejecutar:

```
$ docker run --name mynginx -v $(pwd)/html:/usr/share/nginx/html -v $(pwd)/subdir:/usr/share/nginx/html/subdir -p 8080:80 -d nginx
```

- Abrir browser:

<http://localhost:8080/subdir/>

# Container save as Image

- En cli ejecutar:

```
$ docker run --name container_debian -ti debian bash
```

```
# echo "hello from container debian"
```

- En otro terminal:

```
$ docker commit container_debian debian_hola
```

- Verificar:

```
$ docker images
```

# Dockerfile

Y como extiendo un Docker  
Image?

# Dockerfile

- Generar archivo Dockerfile con contenido:

```
FROM nginx:latest
MAINTAINER Mario Inga <mario21ic@gmail.com>
```

```
RUN echo 'Building from Dockerfile'
COPY html/index.html /usr/share/nginx/html/index.html
```

# Dockerfile

- Ejecutar:

*\$ docker build -t image\_from\_dockerfile:latest .*

- Verificar: *\$ docker images*

- Usar:

*\$ docker run --name newcontainer -p 8088:80 -d image\_from\_dockerfile*

# Docker Link

- Ejecutar:

```
$ docker run --name some-mysql -e  
MYSQL_ROOT_PASSWORD=myclavesecreta -d  
mysql:latest
```

```
$ docker run --name some-wordpress --link some-  
mysql:mysql -p 8080:80 -d wordpress
```

- Browser: <http://localhost:8080/>

# Docker Compose



# Problema

- Containers aislados
- Configurar la ip cada vez que reinicia
- Compartir configs
- Construir cada Image desde Dockerfile
- Escalar un container

# Docker Compose

- Permite una fácil orchestration entre containers
- Desarrollado en Python
- Instalacion: pip install -U docker-compose
- Configuración en un “docker-compose.yml” como template y “docker-compose.override.yml” para dev
- Define cada services como componente.

# docker-compose.yml

```
version: '2'

services:
  nginx:
    image: nginx:1.11
```

# docker-compose.override.yml

```
version: '2'

services:
  nginx:
    volumes:
      - ./html:/usr/share/nginx/html
    ports:
      - 8080:80
```

- Ejecutar:  
  \$ *docker-compose up*

# docker-compose.prod.yml

```
version: '2'  
services:  
  nginx:  
    build: .  
    ports:  
      - 80:80
```

- Ejecutar:  
*\$ docker-compose -f docker-compose.prod.yml up*

# docker-compose.yml

```
web:
  image: odoo:8
  volumes:
    - ./odoo_vps:/mnt/extra-addons/odoo_vps
  links:
    - db
  ports:
    - "8069:8069"
db:
  image: postgres:latest
  environment:
    POSTGRES_USER: odoo
    POSTGRES_PASSWORD: odoo
```

## CLIENT



Docker  
CLI



Docker  
Compose

## SERVER



Local



Docker  
Machine



Docker  
Machine



Docker  
Machine



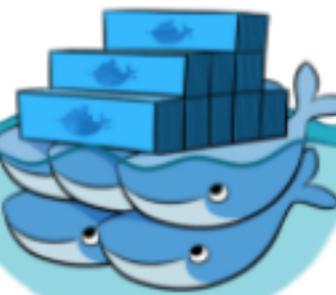
Docker  
Machine



## REPOSITORY



Docker  
Registry



Docker Swarm

# Recomendaciones

- Usa base image oficiales
- Los containers son desechables
- Ejecutar solo un proceso en un container
- Linkear solo lo necesario
- No abusar del depends\_on
- Separar tus docker-compose.yml
- Dividir mediante networking

Preguntas?