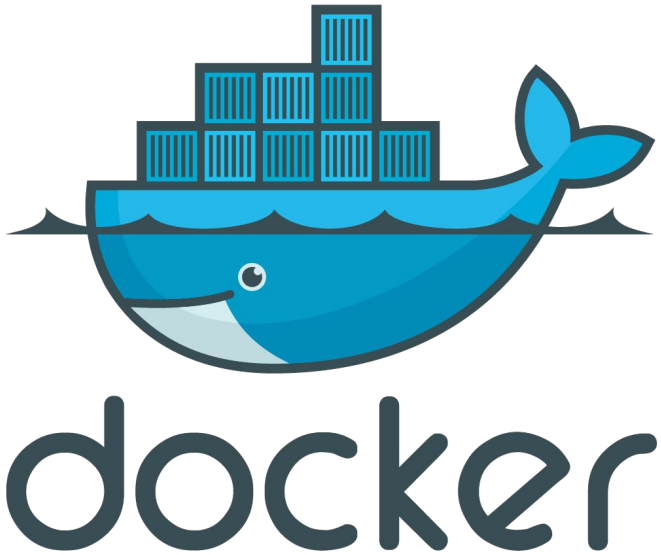
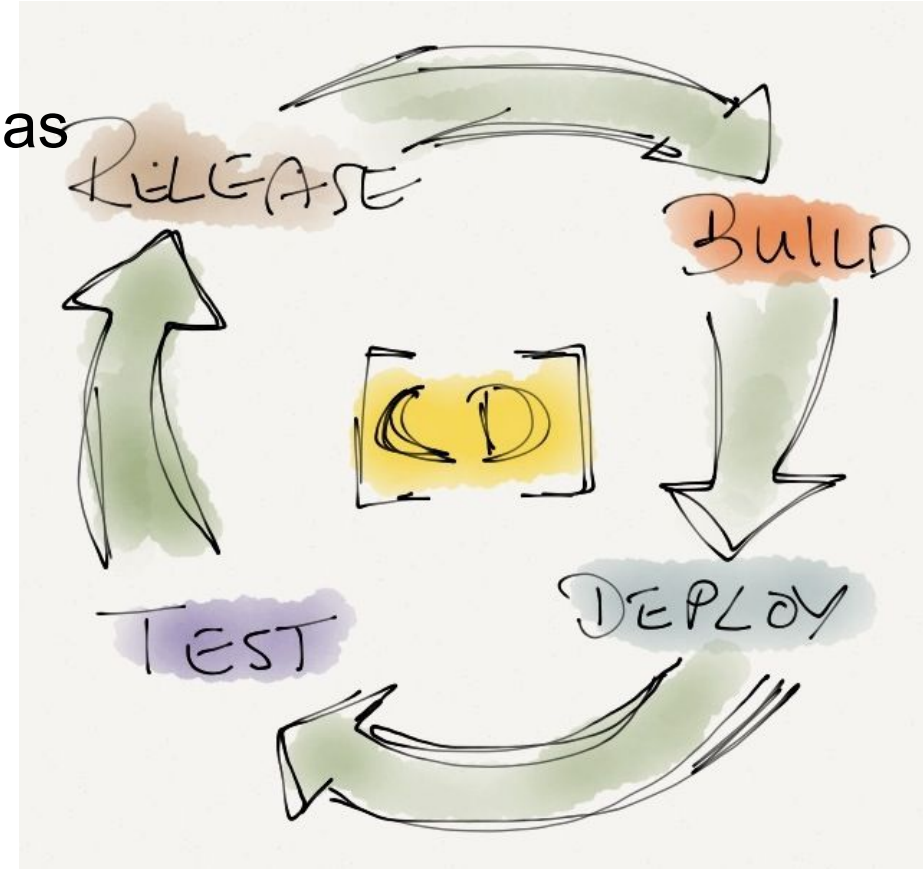

docker

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

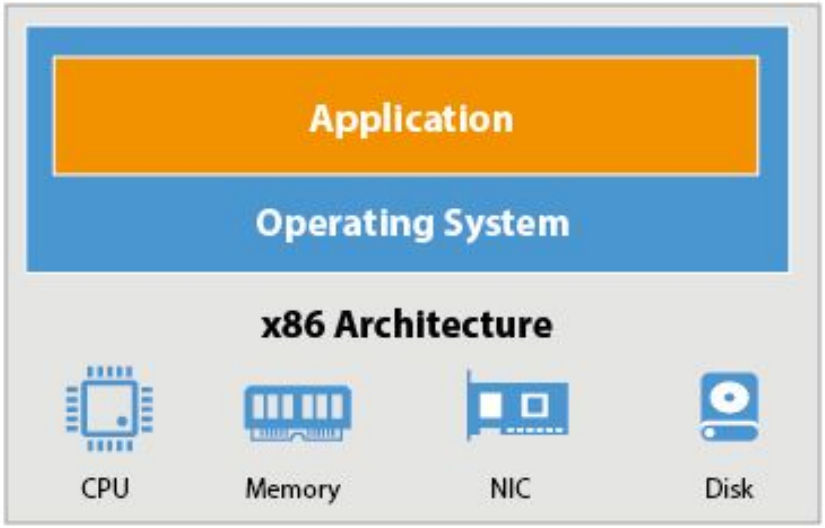
- Ing. en sistemas / MoIT
- OSS & Golang ❤
- Miembro del docker community speakers program
- Docker global hackday #3 (https://blog.docker.com/2015/09/docker-global-hack-day-3-winners/)
- Keynote cierre DockerCon EU 2015 (https://www.youtube.com/watch?v=ZBcMy-_xuYk)
- Crossfitter

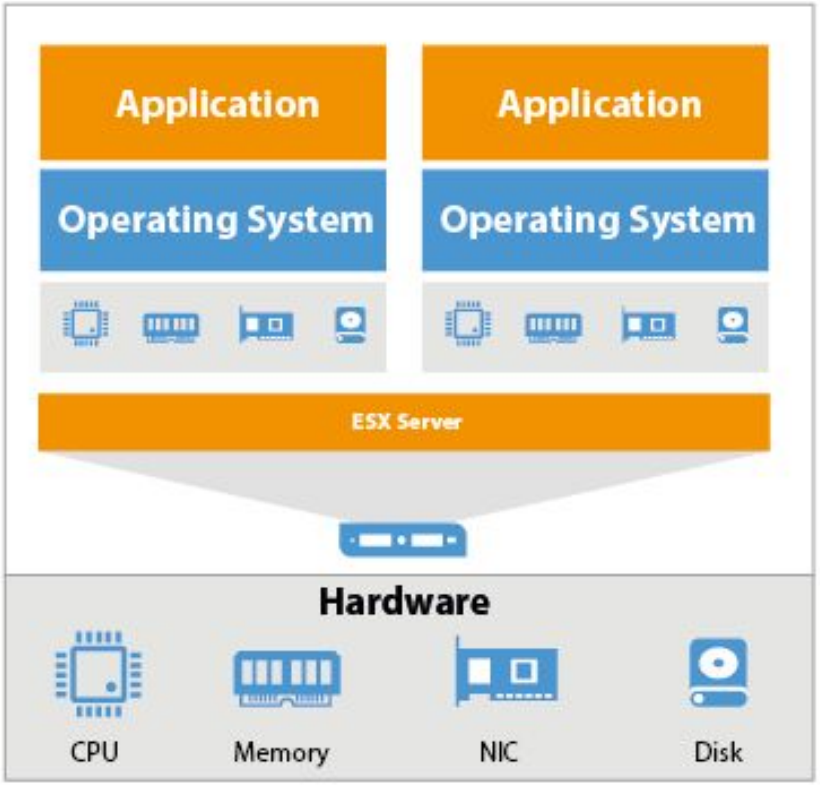
#PlatziDocker

- Configuración de entornos
- Ejecución y control de pruebas automatizadas
- Controles de calidad y métricas tecnológicas
- Controles de cambios y configuraciones
- Puesta en producción y seguimiento



Un poco de historia





MANTIKA

Las VMs son buenas pero….

- Alocación de recursos dedicada
- Un sistema operativo independiente por cada VM (uso de recursos + performance)
- Entornos de desarrollo limitados
- Velocidad de elasticidad limitada (tiempo de inicio del OS)

Qué es Docker?

“Docker es una plataforma para desarrollo, entrega y ejecución de aplicaciones mediante la tecnología de contenedores”

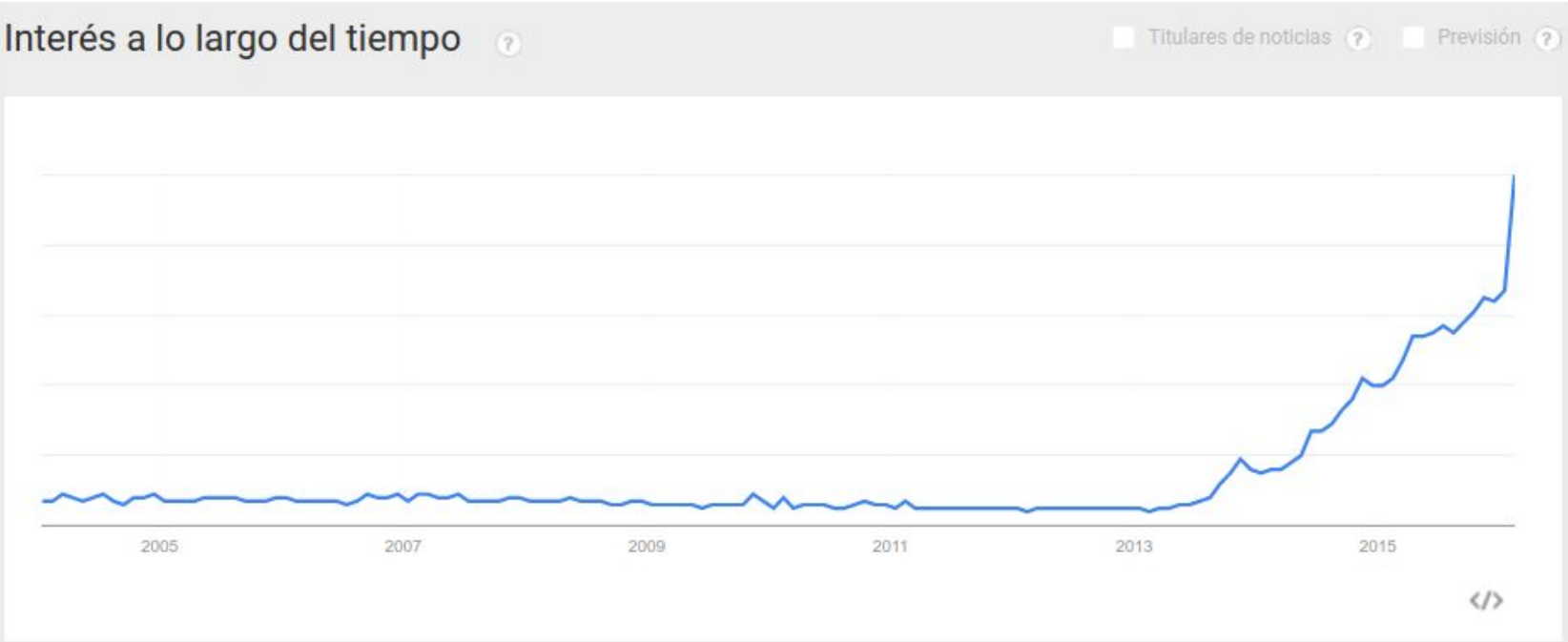
- La plataforma de Docker se compone de múltiples herramientas
  - Docker Engine
  - Docker Hub
  - Docker Trusted Registry
  - Docker Machine
  - Docker Swarm
  - Docker Compose
  - Kitematic

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

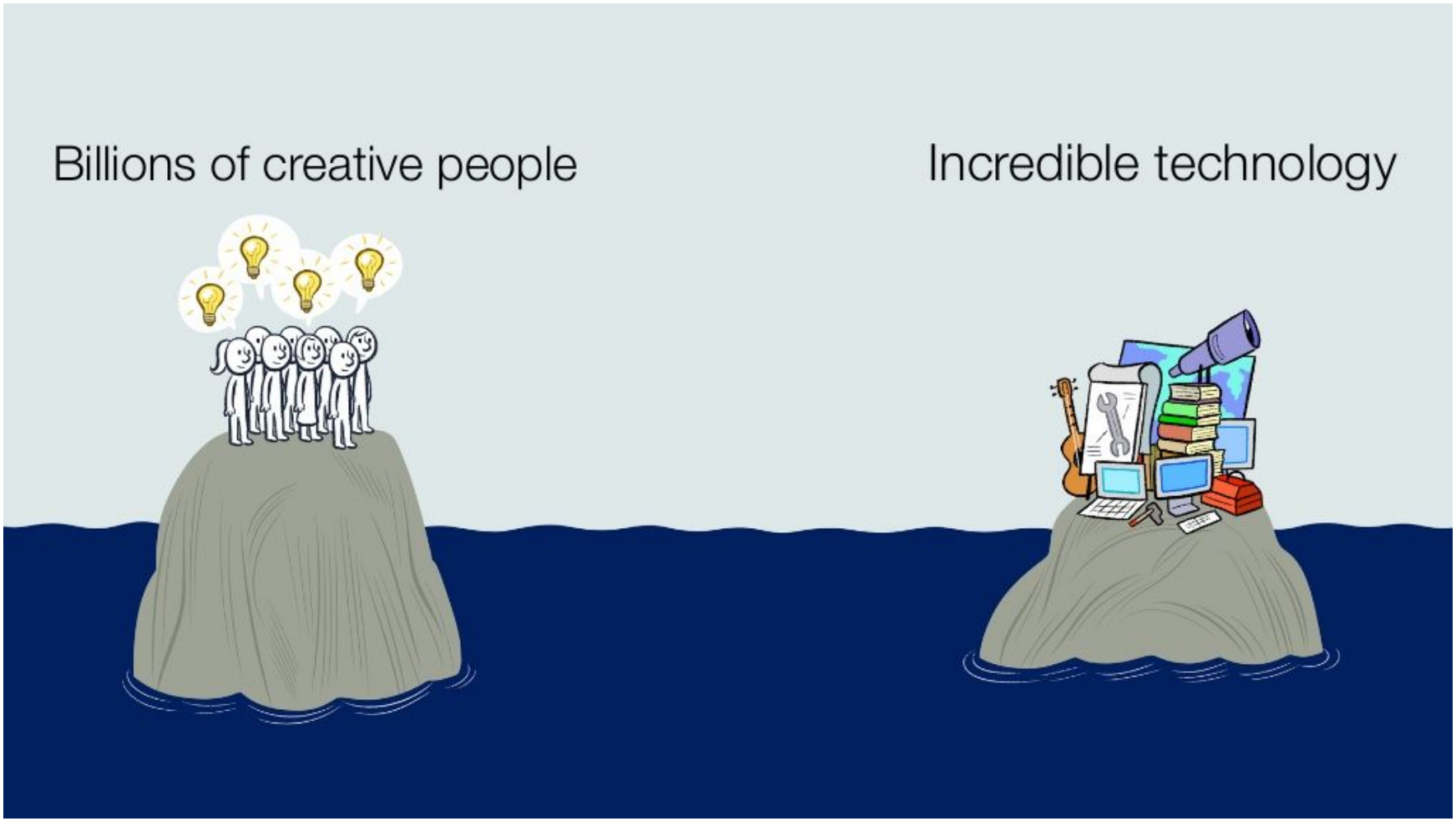
El ecosistema de Docker

Por qué Docker?

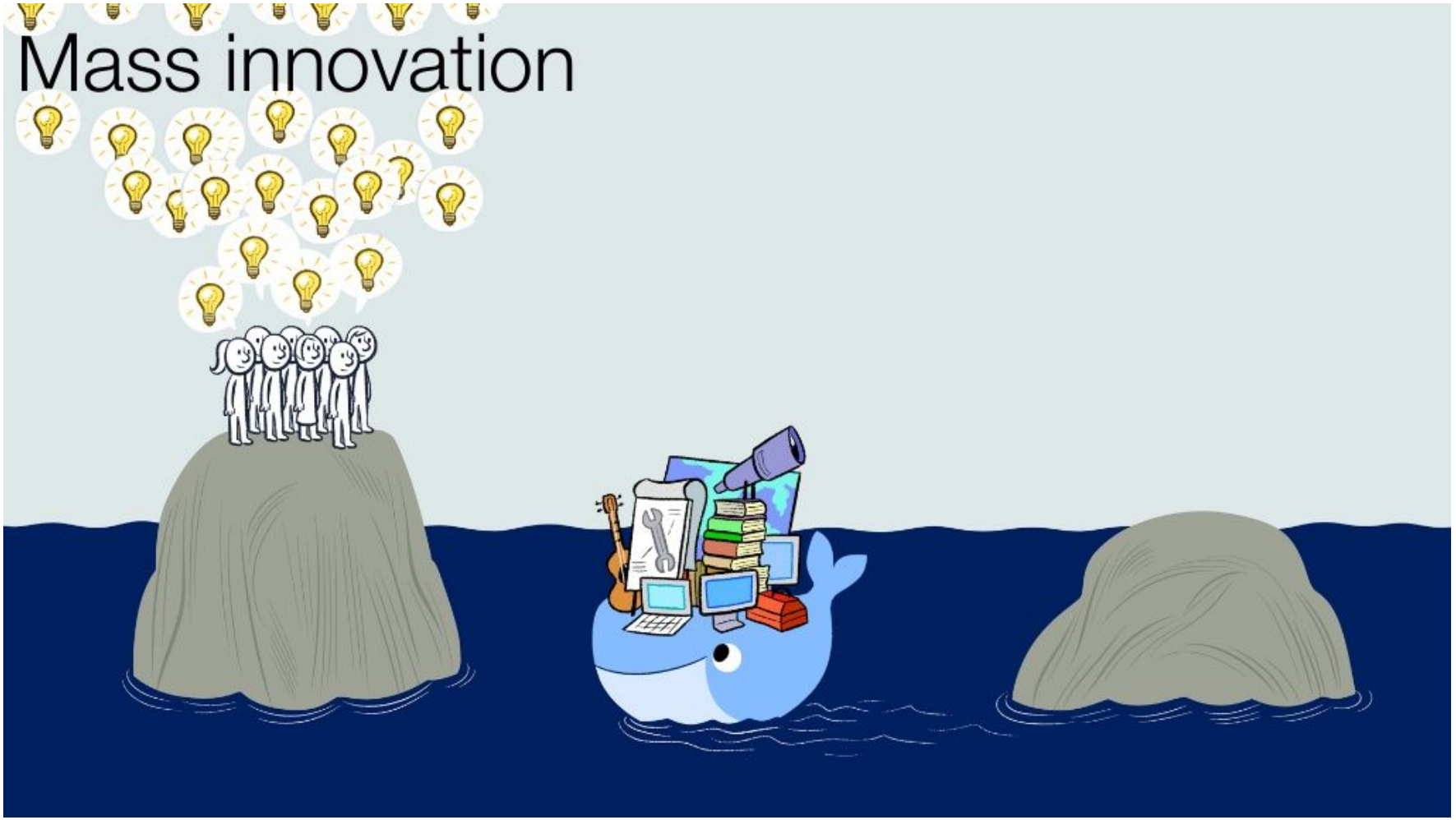


MANTIKA

Por qué Docker? (cont.)

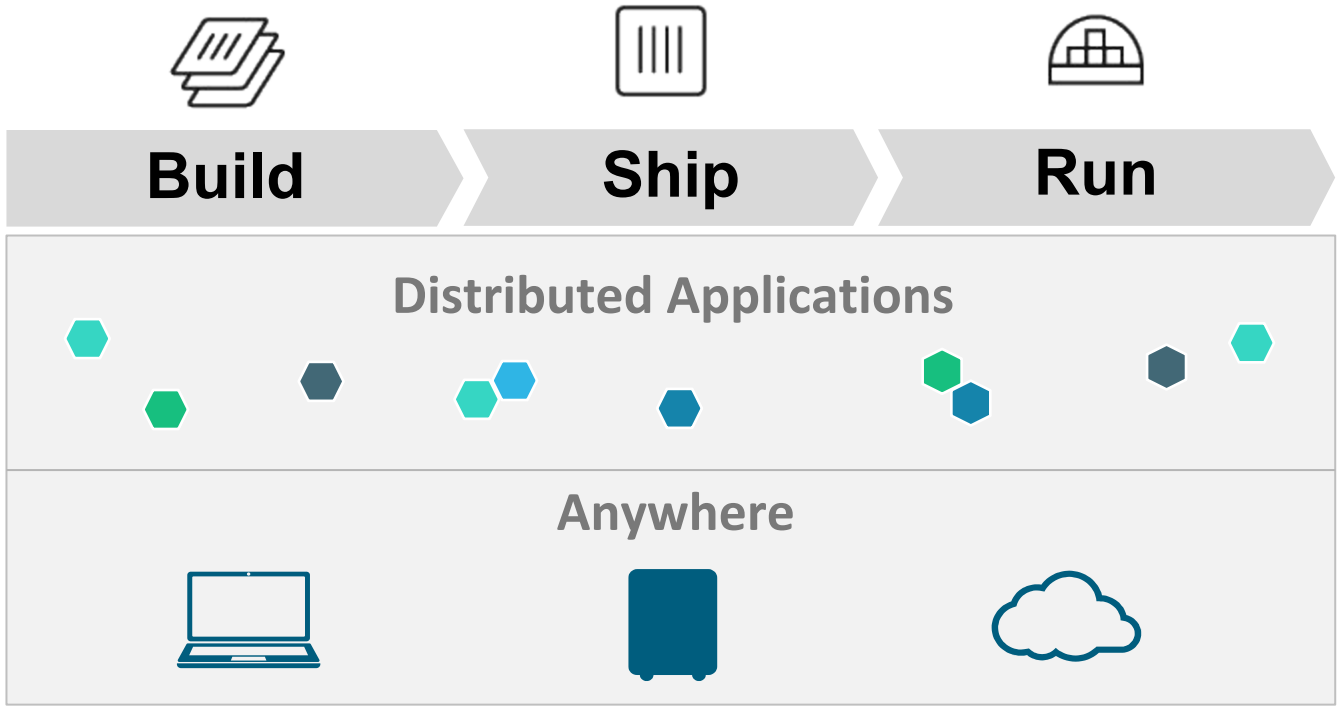


Por qué Docker? (cont.)



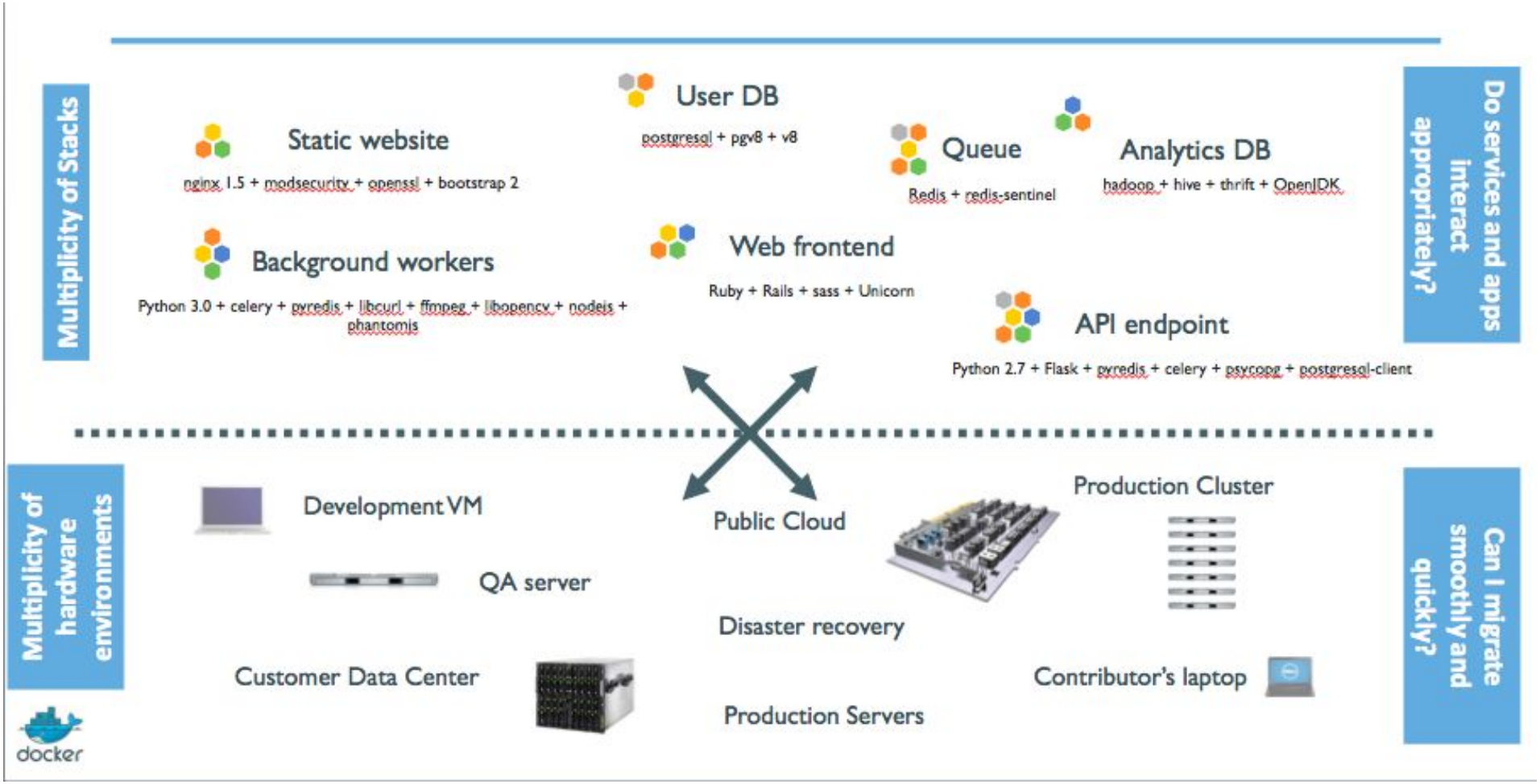
Por qué Docker? (cont.)

La misión de Docker



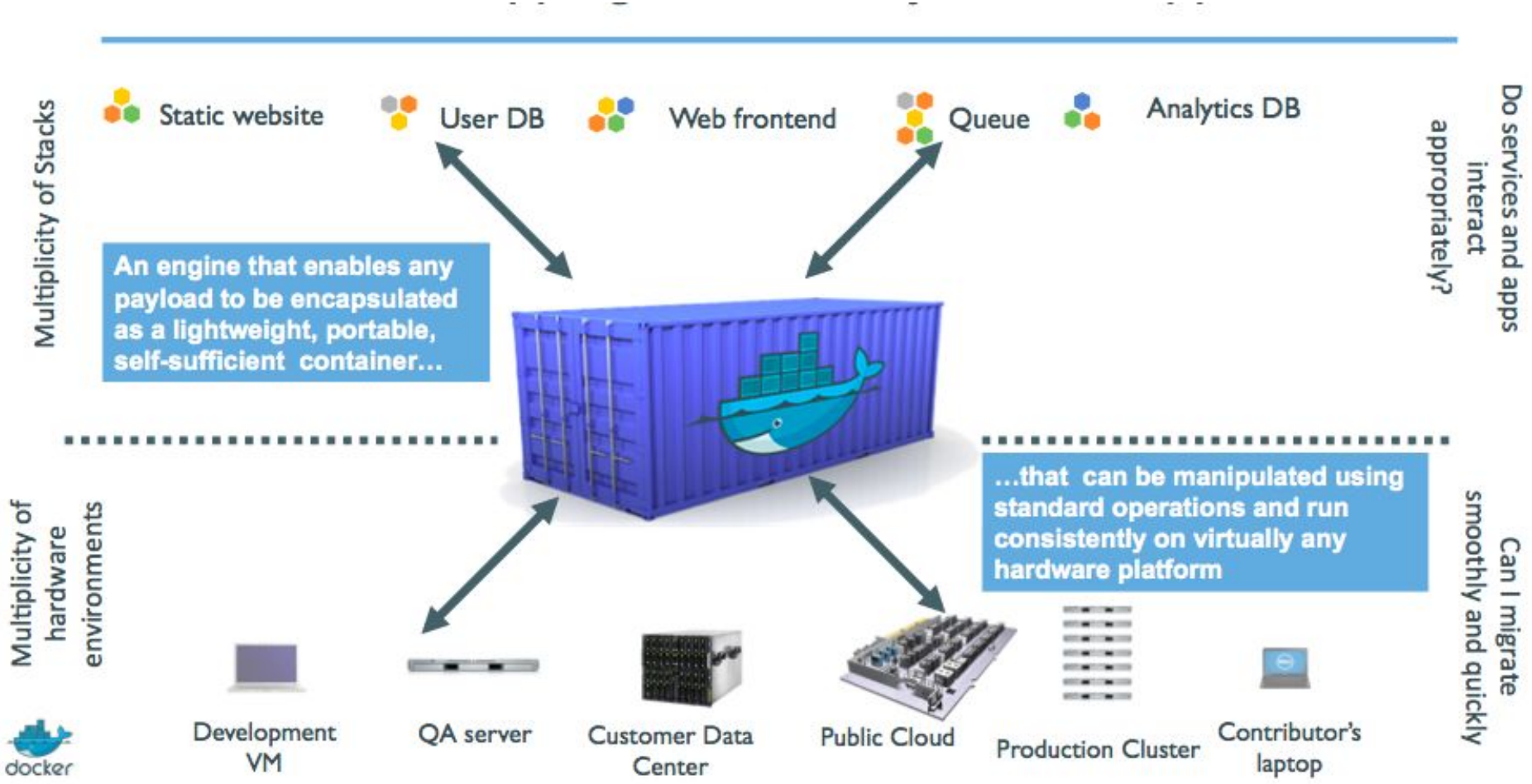
“Build, ship and run any app anywhere”

Por qué Docker? (cont.)



MANTIKA

Por qué Docker? (cont.)

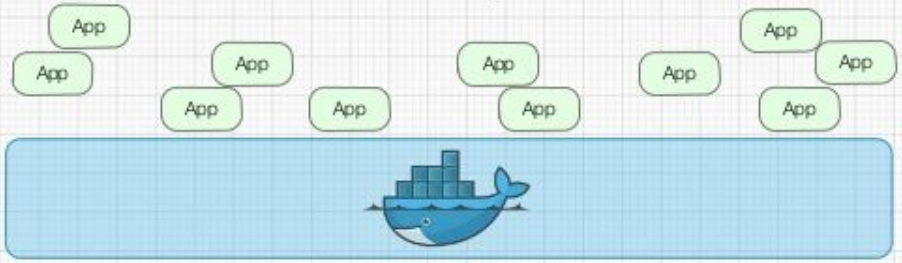


MANTIKA

Por qué Docker? (cont.)

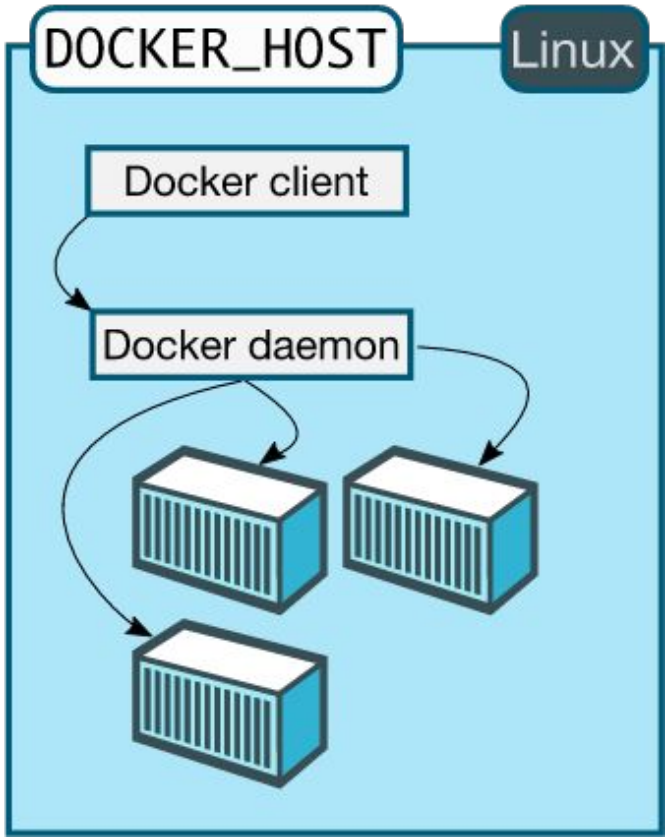
Happy developer

Open software layer



The Internet

Configuración avanzada Docker Engine



Configuración avanzada Docker Engine

`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

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
```

CLIENTE

SERVICIO / DEMONIO

Configuración avanzada Docker Engine

- La forma en la que iniciamos / detenemos el demonio de Docker depende de distintos factores
  - Se encuentra corriendo como servicio ?
  - La distribución de Linux utilizada

- El comando `service` vs `systemctl`

- Iniciar el demonio de manera interactiva desde la terminal (`docker daemon …`)

Parámetros de inicio del Docker Engine

- Cómo configurar los parámetro de inicio dependerá de:
  - Si nos encontramos ejecutando el demonio de manera interactiva o como un servicio(`docker daemon …`)
  - Si se encuentra corriendo como servicio, el sistema operativo (Ubuntu, Debian, CentOS or Fedora etc…)
- Si ejecutamos el demonio de docker de manera interactiva, especificar los parámetros de inicio es tan sencillo como:

```
sudo docker daemon [options] &
```

- En el caso de utilizar *"upstart"* localizar el archivo en /etc/default/docker y modificar *DOCKER_OPTS* (sudo service docker restart es necesario)

- En *"systemd"* localizar el archivo docker.service (y modificar el archivo referenciado por la propiedad `EnvironmentFile`

Qué podemos configurar?

- Decidir cómo escucha el demonio de docker (tcp / unix socket)
- Especificar un servidor DNS
- Definir el nivel de logging
- Habilitar el modo de DEBUG
- Cambiar el registro por defecto
- Configuraciones de seguridad (TLS)

- Referencia completa https://docs.docker.com/reference/commandline/cli/#daemon

Logs del demonio de Docker

- Especificar el parámetro `--log-level` en cualquiera de sus variantes (DOCKER_OPTS / interactiva)
- En caso de ejecutarse como servicio es necesario reiniciar el demonio para que el mismo tome los cambios
- Ubicaciones por defecto:
  - Upstart: /var/log/upstart/docker.log
  - Interactivo: stdout
  - Systemd: journalctl -f -u docker.service

Conectándose a un demonio de forma remota

- Hasta el momento nuestro cliente y servidor se encuentran en el mismo equipo
- Qué sucede si necesitamos conectarnos a un demonio externo?
- Es necesario cambiar unas configuraciones:
  - Primero es necesario decirle al demonio de docker que escuche en una conexión TCP
  - Luego es necesario configurar el cliente de docker para indicarle la dirección remota de nuestro demonio.

- El demonio de docker soporta conectividad mediante 3 tipos de APIs
  - unix (por defecto en /var/run/docker.sock - root)
  - tcp
  - fd (systemd)

- Finalmente configurar el cliente de manera correspondiente

Aspectos de seguridad en Docker

- Docker permite ejecutar aplicaciones de manera segura debido a los controles y privilegios que utiliza
- Los Namespaces proveen una vista aislada del sistema. Cada contenedor utiliza su propio entorno de:
  - IPC, Stack de red, root file system etc…
- Los procesos que se ejecutan en un contenedor no pueden ver o afectar a procesos en otros contenedores
- Los Control groups (Cgroups) aislan los recursos del sistema utilizador por cada contenedor (memoria / cpu / red / io)
  - Aseguran que un contenedor no pueda hacer fallar el host por hacer mal uso de sus recursos.

Consideraciones importantes

- El demonio de docker debe correr como usuario root
- Es importante controlar aquellos usuarios que pueden utilizar el demonio de docker
  - Revisar quién tiene acceso al grupo docker
- Si el demonio de docker escucha en una conexión TCP, securizarla utilizando TLS
- Se pueden utilizar aspectos de seguridad avanzados
  - Apparmor
  - SELinux
  - GRSEC

Configuración de TLS

- Evolution de SSL
- El protocolo que utiliza la web segura (https)
- Utiliza criptografía de clave pública y privada para encriptar las conexiones
- Las claves son cifradas con certificados que son mantenidos emitidos y mantenidos por una entidad confiable.
- Los certificados aseguran que el servidor es quien dice ser
- La comunicación en consecuencia se encuentra encriptada y autenticada

Configuración de TLS (cont.)

- Docker provee mecanismos para autenticar el cliente y el demonio entre ellos.
- Agrega autenticación y autorización y encriptación para las conexiones a través de la red
- Las claves pueden ser distribuidas a clientes autorizados
- **Requisitos**
  - Tener instalado OpenSSL 1.0.1 o superior
  - Crear una carpeta donde guardar las claves protegidas (`chmod 700`)

Pasos a seguir

1. Crear una Autoridad de Certificación (**CA**)
   a. Es necesario una clave privada y un certificado.
2. Configurar la clave privada del servidor
3. Crear un requerimiento de firma (**CSR**) de certificado para el servidor
4. Firmar la clave del servidor con el **CSR** mediante nuestro **CA**
5. Crear una clave privada y un **CSR** para el cliente
6. Firmar la clave del cliente con el **CSR** mediante nuestro **CA**
7. Iniciar el demonio de docker con la opción de TLS y especificar la ubicación de la clave privada del **CA**, el certificado del servidor y la clave privada del servidor
8. Configurar el cliente de docker para que utilice la clave privada del cliente y la clave privada del **CA**

Asegurando nuestras claves

1. Asegurarse que las claves del cliente y del servidor sólo puedan ser leídas por el usuario actual
   ```
   chmod -v 0400 ca-key.pem client-key.pem server-key.pem
   ```
2. Remover el acceso de escritura a todos los certificados
   ```
   chmod -v 0444 ca.pem server-cert.pem client-cert.pem
   ```
3. Crear la carpeta `/etc/docker` en caso que no exista.
4. Cambiar los permisos de la carpeta `/etc/docker`.
   ```
   sudo chown <username>:docker /etc/docker
   sudo chmod 700 /etc/docker
   ```
5. Copiar las claves de servidor a la nueva carpeta
   ```
   sudo cp ~/docker-ca/{ca,server-key,server-cert}.pem /etc/docker
   ```

Usando Docker con TLS

1. Iniciar el demonio con los siguientes parámetros:

```
DOCKER_OPTS="-H tcp://0.0.0.0:2376 --tlsverify
--tlscacert=/etc/docker/ca.pem
--tlscert=/etc/docker/server-cert.pem
--tlskey=/etc/docker/server-key.pem”
```

2. Reiniciar el servicio de Docker en caso de ser necesario

```
sudo service docker restart
```

3. Utilizar las credenciales correspondientes en el cliente (carpeta docker-ca)

```
docker --tlsverify \
       --tlscacert=ca.pem \
       --tlscert=client-cert.pem \
       --tlskey=client-key.pem \
     -H tcp://127.0.0.1:2376 \
```

Tips

- Podemos emitir especificar todas las claves en el cliente creando una carpeta `.docker` en el home de nuestro usuario

- Sin embargo, es necesario renombrar nuestros archivos a: `ca.pem`, `cert.pem` and `key.pem`

- Una vez realizados los pasos anteriores cada vez que utilicemos el comando `docker`, el cliente utilizará las claves automáticamente. Sólo es necesario especificar el comando `--tlsverify` y `-H`.

```
docker --tlsverify -H 127.0.0.1:2376 ps -a
```

- Para simplificar aún más el uso del cliente, podemos utilizar las variables de entorno *DOCKER_HOST* y *DOCKER_TLS_VERIFY*.