Introducción a Docker

CiDAEN

Jacinto Arias

@cidaen

@jacintoarias

Curso de Especialista en Ciencia de Datos y Aplicaciones Escalables en la Nube

https://www.docker.com/get-docker

```
git clone git@github.com:cidaen/tutorial-docker.git
```

Introducción a Docker

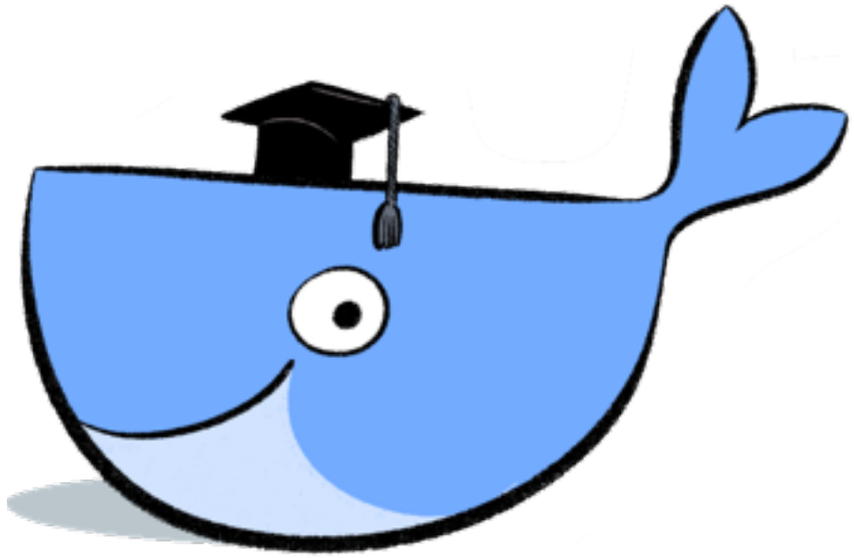
- ¿Qué es docker, qué es un contenedor?
- Conceptos Básicos y primeros pasos
- Herramientas, ecosistema y casos de uso
- CiDAEN

Introducción a Docker

- **¿Qué es docker, qué es un contenedor?**
- Conceptos Básicos y primeros pasos
- Herramientas, ecosistema y casos de uso
- CiDAEN

¿Porqué aprender Docker?

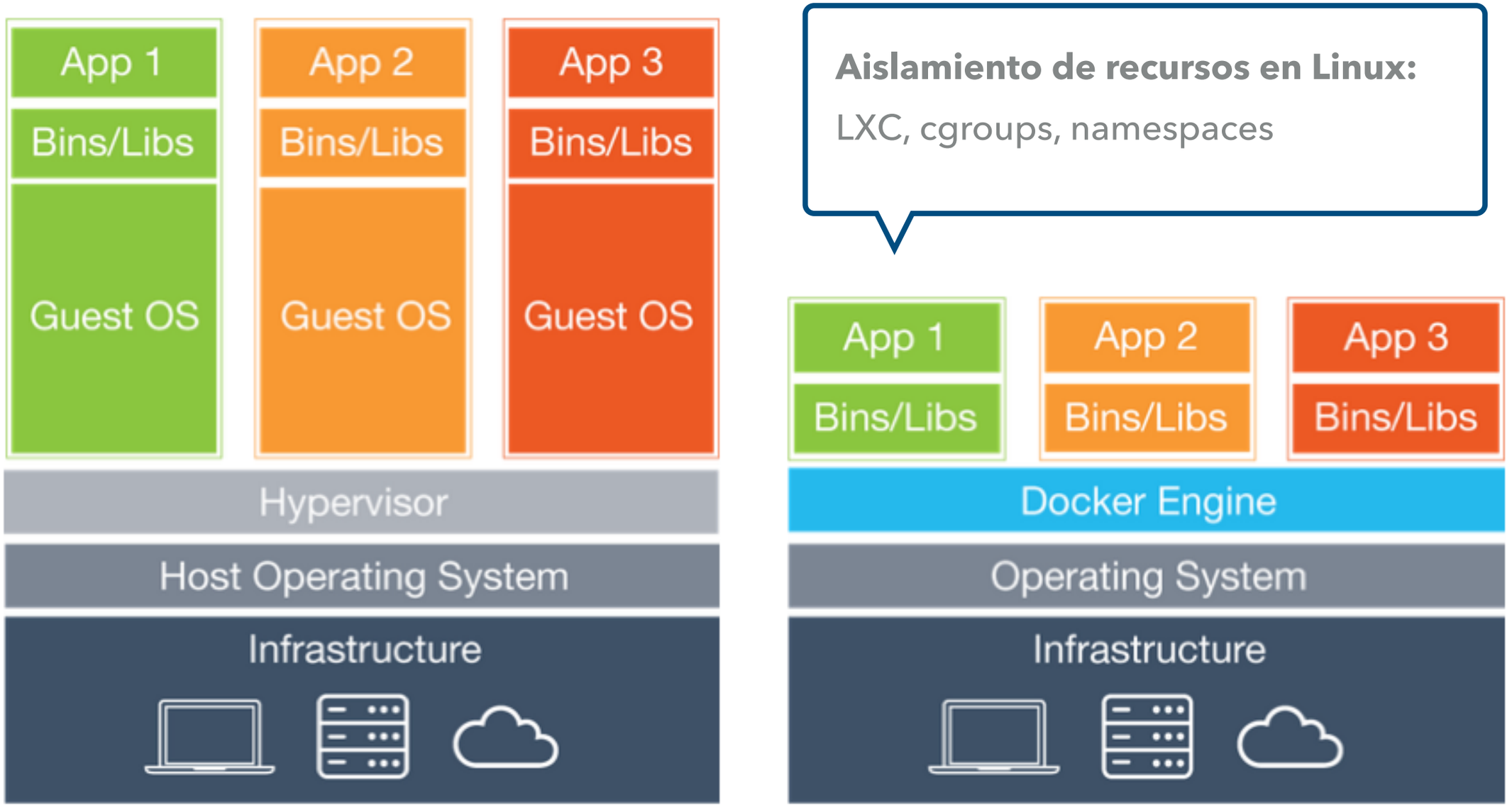
- Mejoramos como desarrolladores
- Mejor entorno, menos mantenimiento y menos bugs
- Mejor integración y colaboración en equipos
- Acceso a infraestructura/ contenedores como servicio en cloud



¿Qué es un contenedor?

- Un contenedor **encapsula** software en un sistema de ficheros que contiene todo lo **necesario para su ejecución**
- Garantiza **estabilidad y portabilidad** a todos los sistemas
- Docker es:
  - **Ligero** a la hora de consumir recursos
  - **Abierto** y compatible (docker es software libre)
  - **Seguro** permitiendo el aislamiento entre contenedores

Contenedores vs Máquinas Virtuales



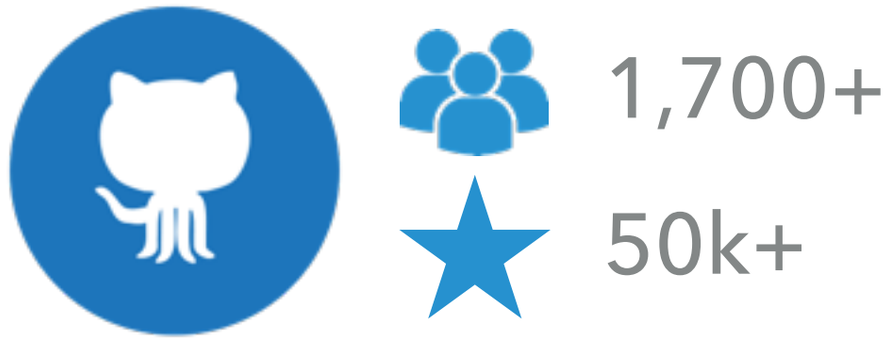
Docker y tu infraestructura

- https://www.docker.com/products/overview
- ¡Aviso! ¡La compatibilidad depende de tu infraestructura!
  - Contenedor Windows Server solo para windows
  - *Contenedores Linux nativos sobre en Windows
  - La arquitectura de la CPU también influye (e.g Raspberry Pi ARM)

* Docker nativo (desde 2016) para Windows 10 y Mac OS 10.10. La alternativa es docker-toolbox, mucho más limitada ya es virtualizada.

Docker en la comunidad

- Adoptado por las grandes compañías, completamente estandarizado.
- Comparte tu código + Dockerfile para que otros puedan colaborar o desplegar con mayor facilidad
- Un gran catálogo de imágenes y recetas disponibles
- Libre

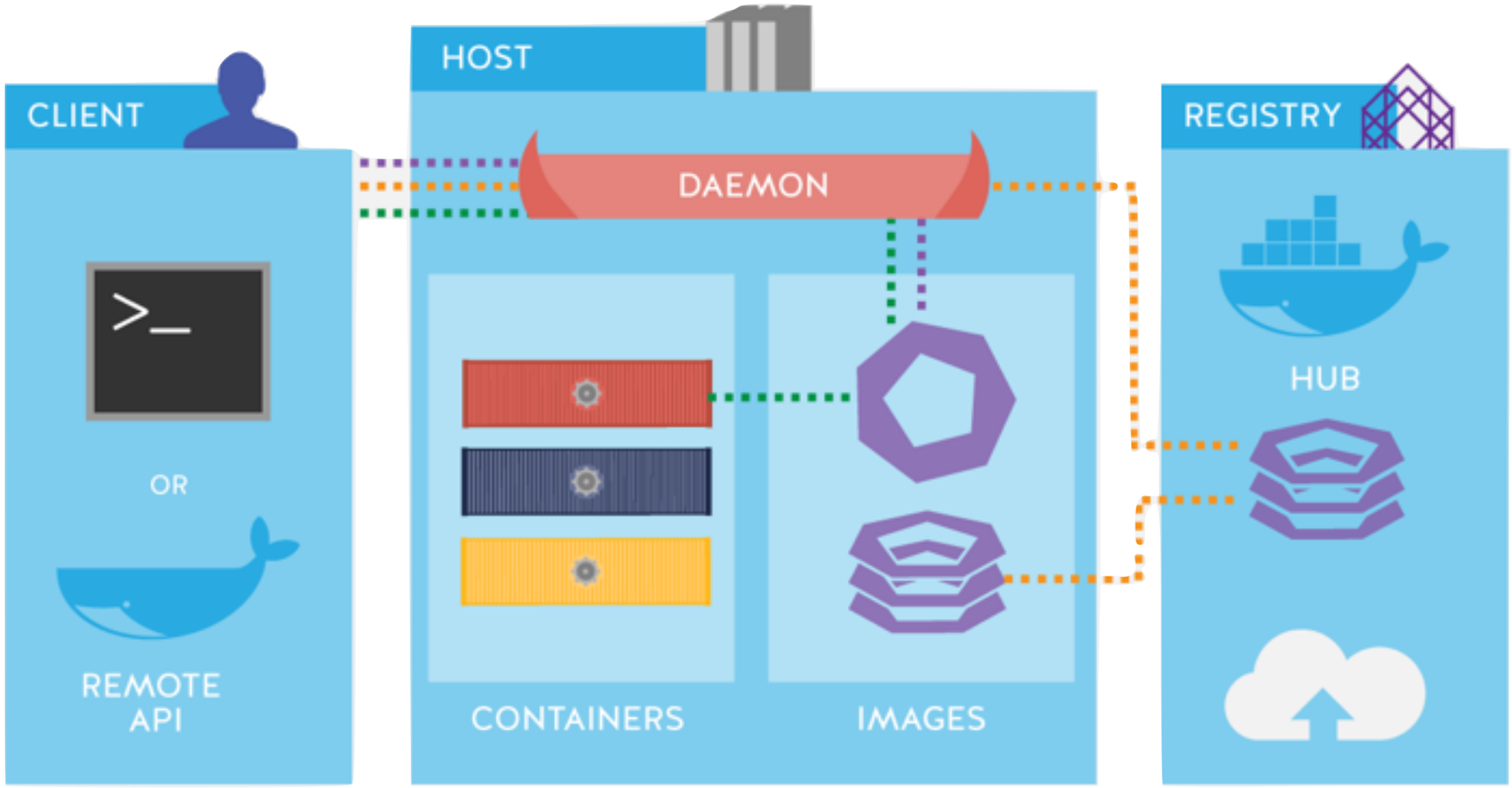


Introducción a Docker

- ¿Qué es docker, qué es un contenedor?
- **Conceptos Básicos y primeros pasos**
- Herramientas, ecosistema y casos de uso
- CiDAEN

Conceptos básicos

- **Docker Engine/Daemon**
- Container
- Image
- Registry
- DockerHub
- Network
- Volume
- Dockerfile



Docker Engine CLI

- El cliente principal de docker es una CLI
- También dispone de una API remota
- El demonio de docker se ejecuta en segundo plano y gestiona la ejecución de los contenedores.

```
$ docker
        | info
        | run
        | kill
        | stop
        | start
        | ps
        | logs
        | rm
        | images
        | build
        | rmi
        | network
        | volumes
        | ...
```

DEMO 1: DOCKER CLI

Primer contenedor Ubuntu

```
docker info
```

Muestra el estado del servicio

```
docker run hello-world
```

Contenedor de test

Contenedor real e interactivo

```
docker run -ti ubuntu /bin/bash
```

```
docker ps
```

```
docker ps -a
```

Listar contenedores en ejecución

-a contenedores parados

```
docker run –rm -t -i <image> <command>
```

Alterar el estado de contenedores

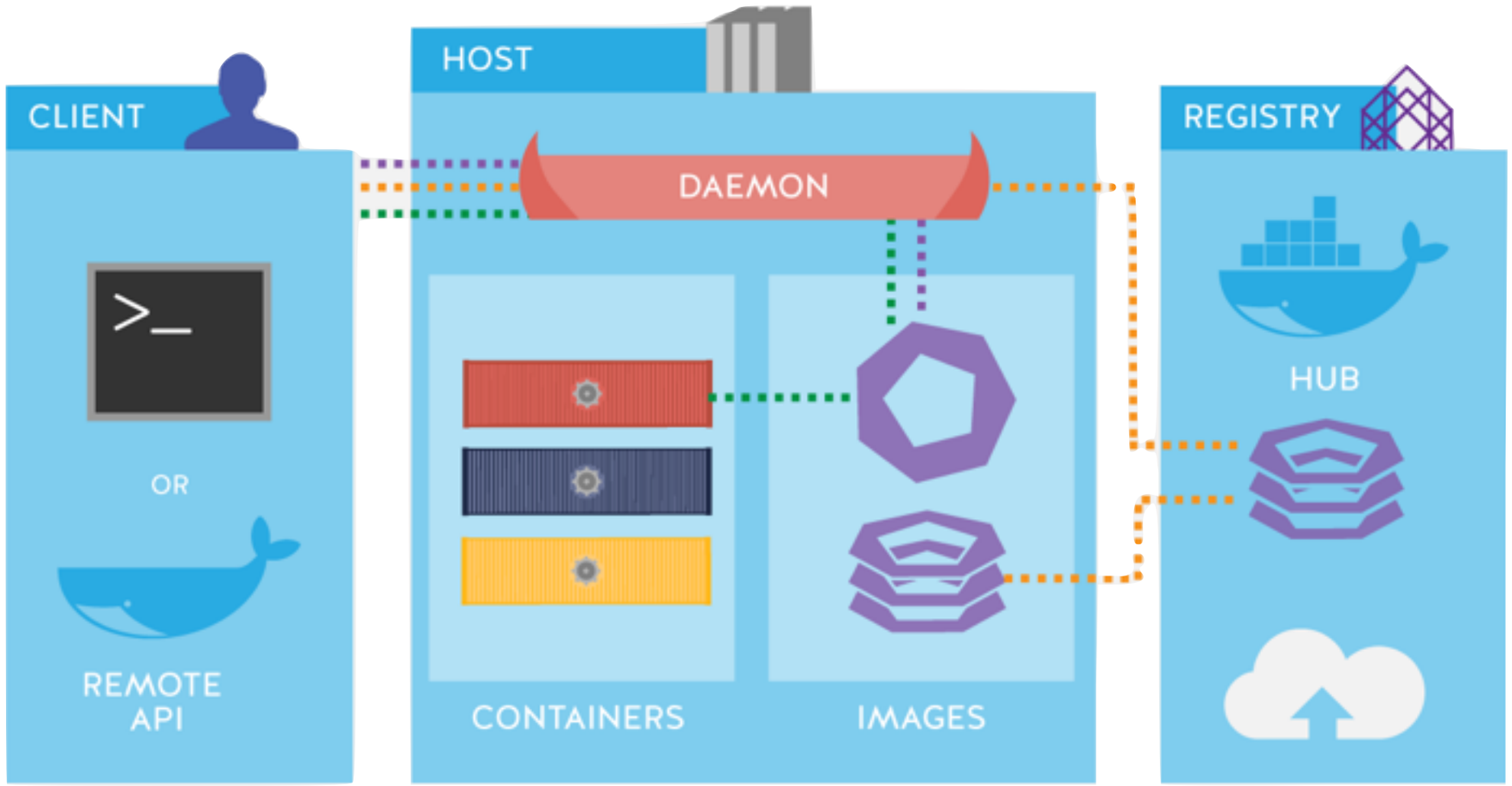
```
docker stop <container>
```

```
docker kill <container>
```

```
docker rm <container>
```

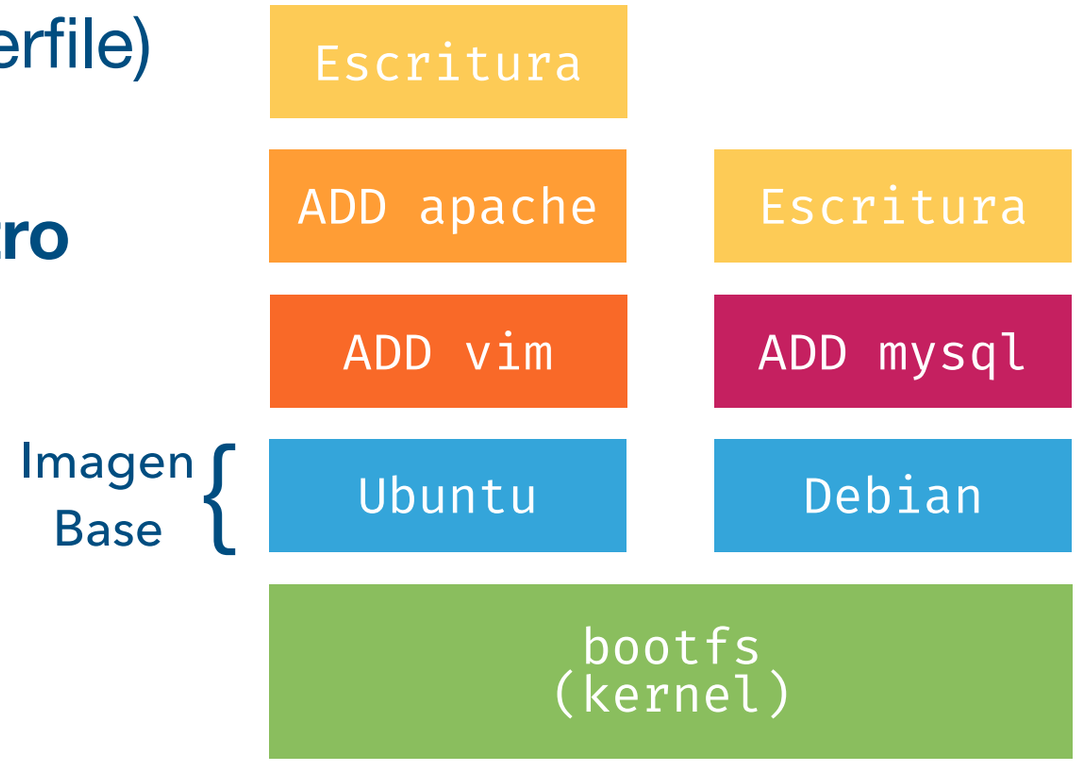
Conceptos básicos

- ~~Docker Engine/Daemon~~
- **Container**
- **Image**
- **Registry**
- **DockerHub**
- Network
- Volume
- Dockerfile



Imágenes

- Las utilizaremos como punto de inicio de nuestros contenedores
- Definen una serie de capas que añaden funcionalidad a partir de una **imagen base** (un SO)
- Podemos **gestionar** imágenes:
  - Definidas por nosotros (Dockerfile)
  - Descargadas desde un **registro**



Contenedores

- Se inician desde una imagen y creando su estructura de ficheros.
- Pueden encontrarse en **ejecución** o **parados**.
- **Persisten los cambios** realizados sobre la imagen base (cuidado!)
- Suelen ejecutarse como **servicios**, pero podemos acceder de manera interactiva
- Docker puede configurar redes y volúmenes compartidos entre contenedores y host

Registro

- **Repositorios** a los que podemos subir imágenes para más tarde utilizarlas nosotros, colaboradores o terceros

```
docker push
```

```
docker pull
```

- Las imágenes están **disponibles** a través del cliente docker

- Permite **control de versiones, etiquetado…**

```
<image-name>:<tag>
```

```
ubuntu:latest
```

```
ubuntu:xenial
```

Dockerhub

- https://hub.docker.com
- Registro **oficial** y **gratuito** de docker
- +400.000 repositorios
- Imágenes oficiales de las plataformas más importantes: distros de linux, servidores BBDD, Web, librerías…
- Posibilidad de cuenta **privada** para compañías

DEMO 2: DOCKERHUB

Sing Up

Explorar imágenes base

Explorar imágenes usuarios

Visita y crea una cuenta en

http://hub.docker.com

`docker login`

Inicia sesión con tus credenciales

No te olvides de hacer logout cuando te vayas si no es tu máquina!

`docker logout`

```
docker run --rm -ti python:3.6-slim
```

```
docker run --rm -ti node:8-slim
```

```
docker run --rm nginx
```

```
docker images ls
```

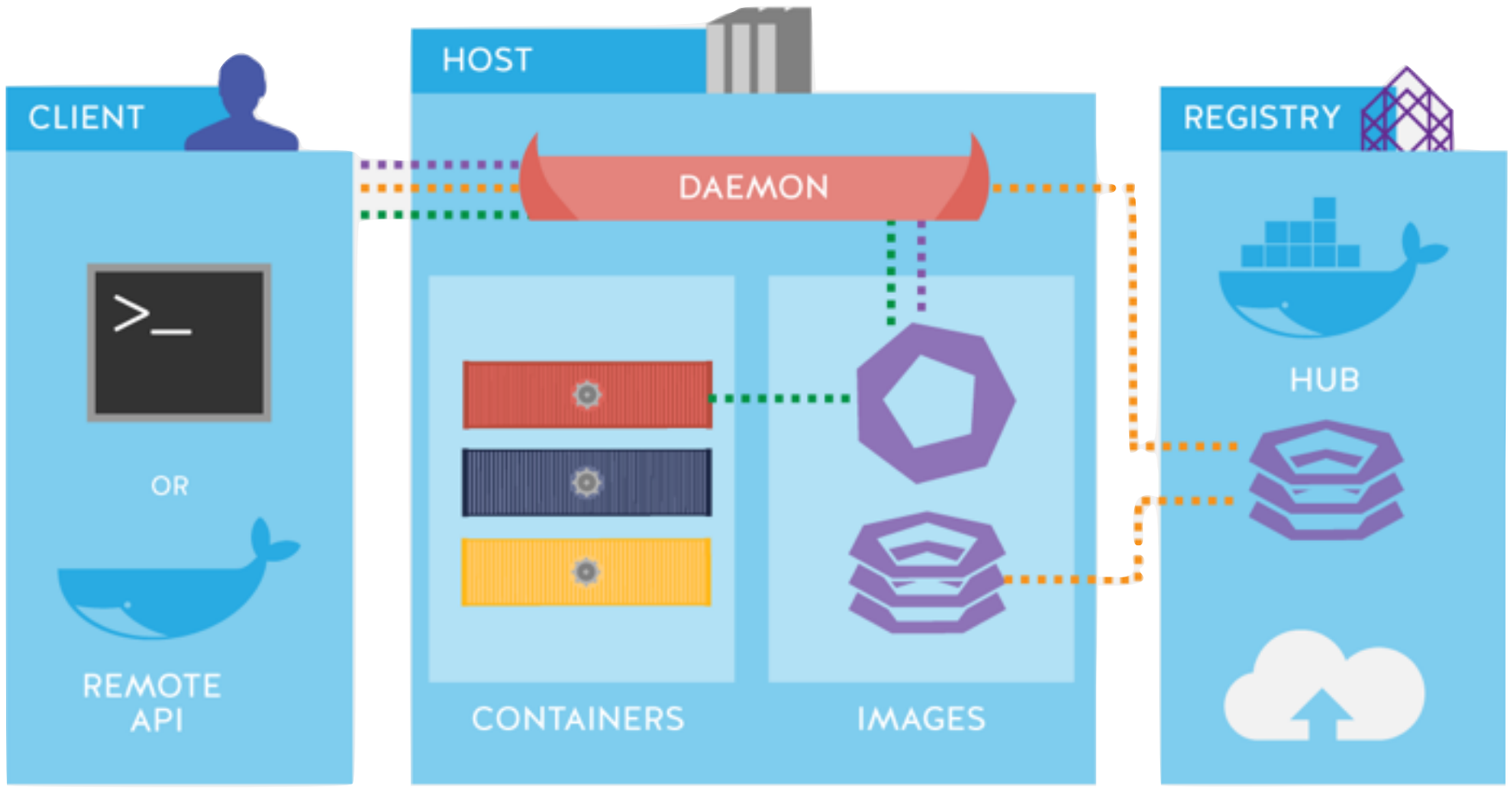
```
docker images rm / docker rmi
```

Listar/borrar imagenes en nuestro sistema local

Es importante mantener nuestra instalación de docker limpia

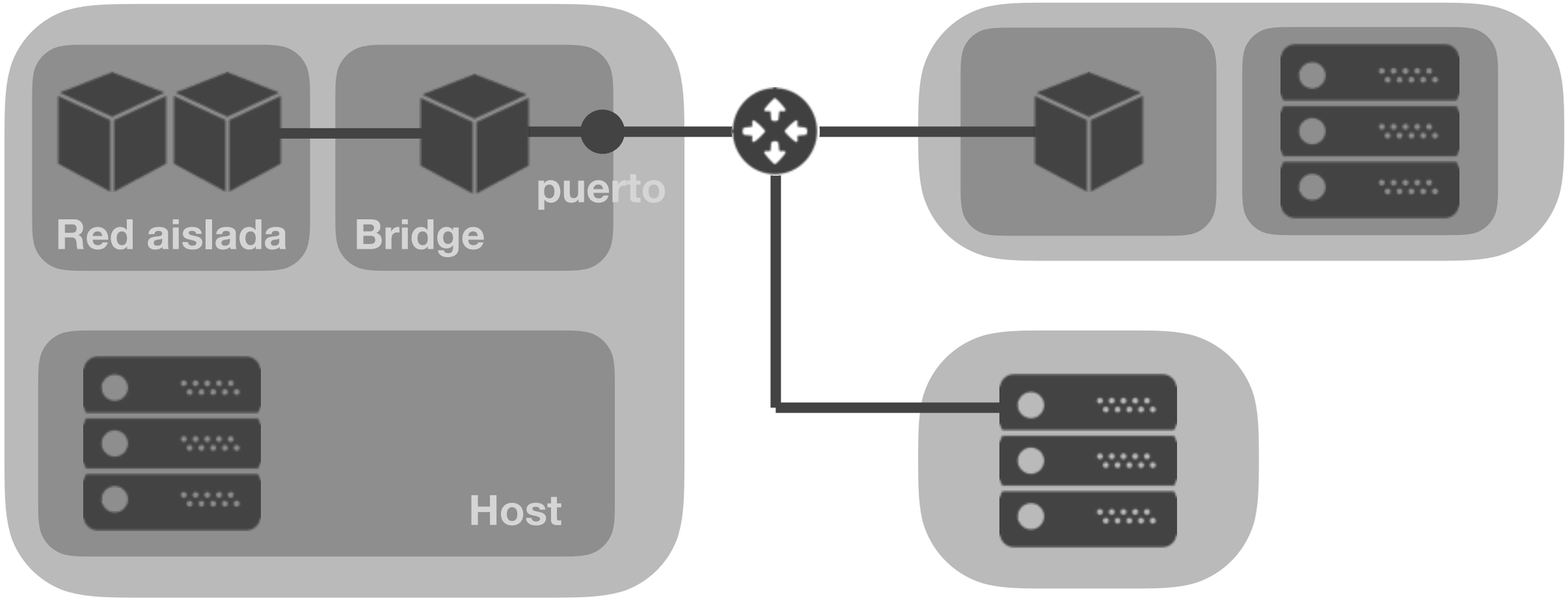
Conceptos básicos

- ~~Docker Engine/Daemon~~
- ~~Container~~
- ~~Image~~
- ~~Registry~~
- ~~DockerHub~~
- **Network**
- **Volume**
- Dockerfile



Comunicación entre contenedores: Redes

- Los contenedores pueden conectarse entre ellos o a otros sistemas mediante redes
- Podemos crear múltiple adaptadores y controlar los puertos accesibles a cada contenedor para crear diversas arquitecturas



Persistencia de datos: Volúmenes

- Docker puede crear **volúmenes de datos persistentes** que se montan sobre el sistema de ficheros de un contenedor
- Los volúmenes puedes **compartirse** entre contenedores para establecer **comunicación**
- Podemos montar un **directorio local** como un volumen de datos

DEMO 3: REDES&VOLÚMENES

Binding de puertos y directorios

Servidores Web

Ejecutar código desarrollo

```
docker run -p <host>:<container> <image>
```

Bind de un puerto del contenedor al host

```
docker run -p 80:80 —rm nginx
```

Podemos desplegar un servidor web en nuestro loop local.

Visita http://localhost:80

```
docker run -p 8000:80 --rm nginx
```

Despliegue en un puerto diferente

```
docker run -p 80:80 -d nginx
```

Despliegue en modo detach

```
docker stop
```

```
docker kill
```

```
docker run -v <local-dir>:<remote-dir> <image>
```

Bind de un directorio local a un contenedor

```
docker run -v $(pwd):/root --rm -ti ubuntu
```

Montar la carpeta actual en el directorio /root del contenedor.

Aviso: $(pwd) puede no funcionar en tu shell

La ruta debe ser absoluta

```
cd demo3
```

```
docker run --rm -d -p 80:80 \
-v $(pwd)/webA:/usr/share/nginx/html:ro \
nginx
```

La imagen de Nginx viene preparada para servir desde

/usr/share/nginx/

```
docker run --rm -d -p 8000:80 \
-v $(pwd)/webB:/usr/share/nginx/html:ro \
nginx
```

Podemos desplegar más servidores.

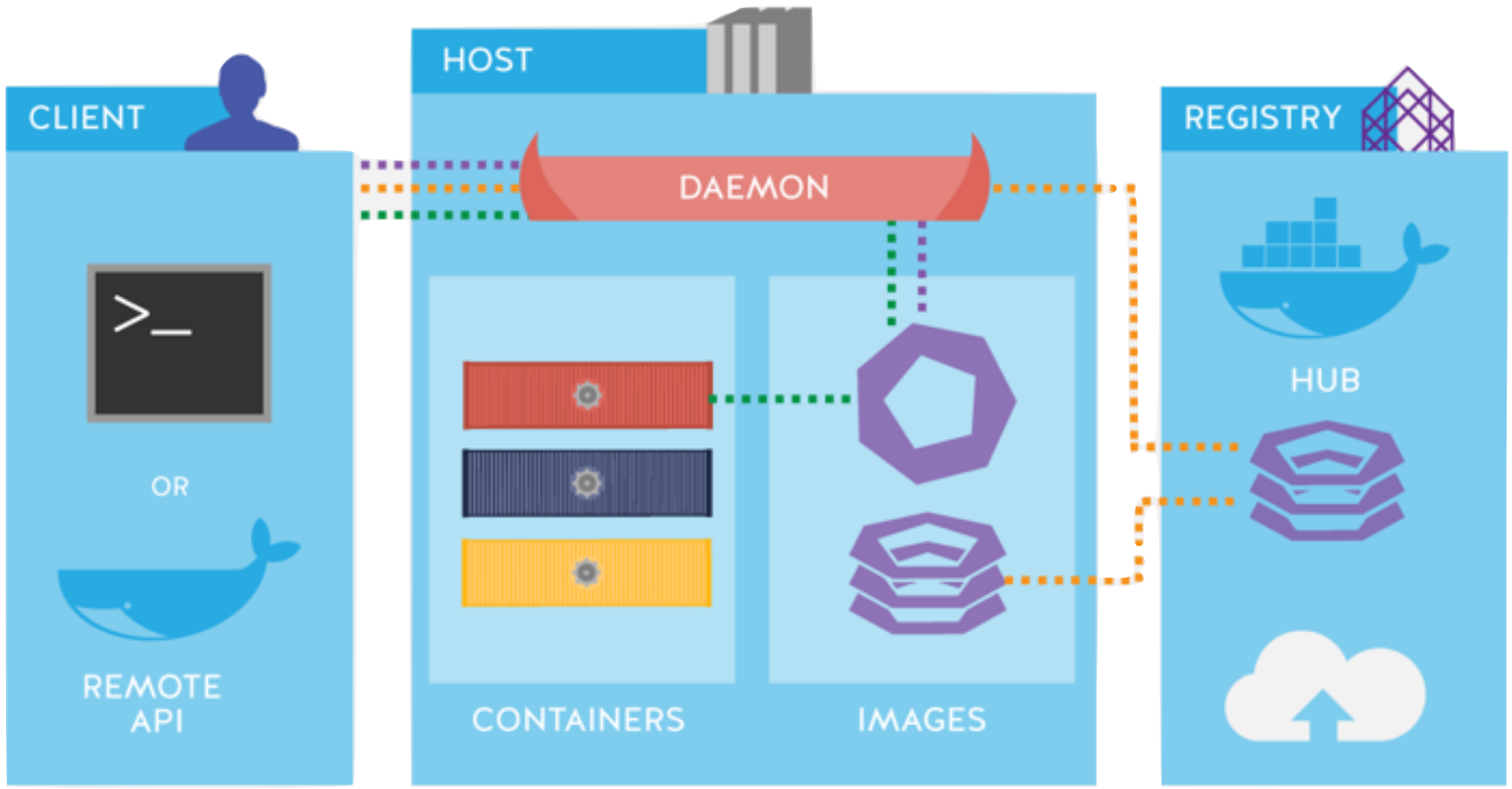
Probad a modificar el código de una web

```
docker run --rm \
-v $(pwd):/root/ \
python:3.6-slim python3 \
/root/very-useful.py cidaen
```

Podemos utilizar volúmenes para compartir y ejecutar código dentro de un contenedor

Conceptos básicos

- ~~Docker Engine/Daemon~~
- ~~Container~~
- ~~Image~~
- ~~Registry~~
- ~~DockerHub~~
- ~~Network~~
- ~~Volume~~
- **Dockerfile**



Dockerfiles

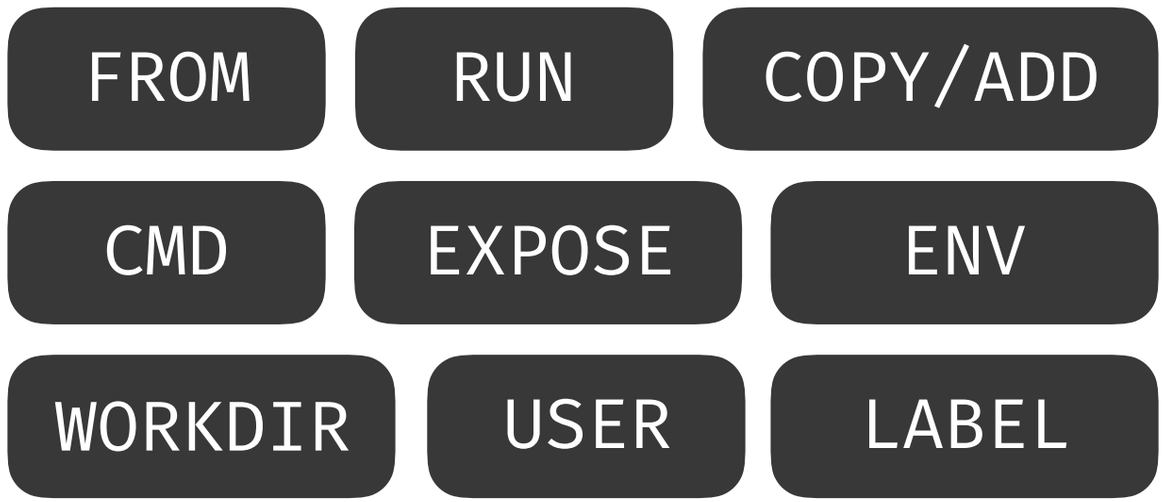
- Un **script** que define la construcción (**build**) de una imagen de docker

```
FROM ubuntu:14.04

LABEL maintainer=“jarias"

RUN apt-get update && \
    apt-get install -y apache2

COPY index.html /var/www/

EXPOSE 80

CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
```

```
docker build -t <namespace>/<image-name>:<tag> <dir>
```

Dockerfiles: Comandos

- Cada **comando** en la dockerfile crea una **capa** en un contenedor
- Cada capa se almacena en una **cache** para reutilizarla
- Las imágenes y contenedores pueden **compartir capas**



Dockerfiles: Comandos

`FROM`	**Añade las capas de una imagen base**
`RUN`	**Ejecuta un comando dentro del contenedor**
`COPY/ADD`	**Copia ficheros desde nuestro sistema**
`USER`	**Cambia al usuario que ejecuta el contenedor**
`WORKDIR`	**Cambia el directorio desde el que se ejecuta**
`ENV`	**Declara variables de entorno**
`CMD`	**Modifica el comando por defecto**
`EXPOSE`	**Especifica los puertos disponibles (no hace bind)**

Dockerfile: Buenas prácticas

- Diseña cada contenedor para una **función específica**
- Los contenedores deben ser **efímeros**
- Minimiza el número de capas y su **tamaño** (cache)
- Cuidado con la **seguridad**, evita usar root y vigila los puertos
- Cuidado con la **privacidad**, todas las capas están disponibles!

DEMO 4: DOCKERFILES

Desplegando Web estática
Desplegando aplicación python
Creando entorno desarrollo Data Science

```
cd demo4/webserver
```

```
docker build -t <dockerhub-user>/cidaenweb .
```

Hacemos un build del servidor y el código

```
docker run -p 80:80 --rm <dockerhub-user>/cidaenweb
```

Ahora podemos desplegar el servidor en cualquier parte

```
cd demo4/python-useful
```

```
docker build -t <dockerhub-user>/cidaen-usefull .
```

Hacemos un build del código y su entorno

```
docker run --rm <dockerhub-user>/cidaen-usefull
```

Ahora nuestro código es portable

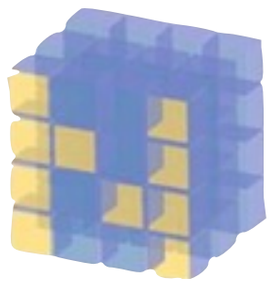
```
docker run --rm -e ARGS="cidaen" \
  <dockerhub-user>/cidaen-usefull
```

Hemos parametrizado la interfaz con variables de entorno

Entorno de trabajo para Data Scientist

Requiere instalar dependencias de python

Podemos instalarlo en local, pero reduce la reproducibilidad



NumPy

matplotlib

pandas

$y_i t = \beta' x_{it} + \mu_i + \epsilon_{it}$

```
cd demo4/datascience
```

```
docker build -t <dockerhub-user>/jupyter .
```

```
docker run --rm -p 8888:8888 \
  <dockerhub-user>/jupyter
```

Hemos creado un entorno portable

Podemos compartirlo con nuestros colaboradores

```
docker push <dockerhub-user>/jupyter
```

Introducción a Docker

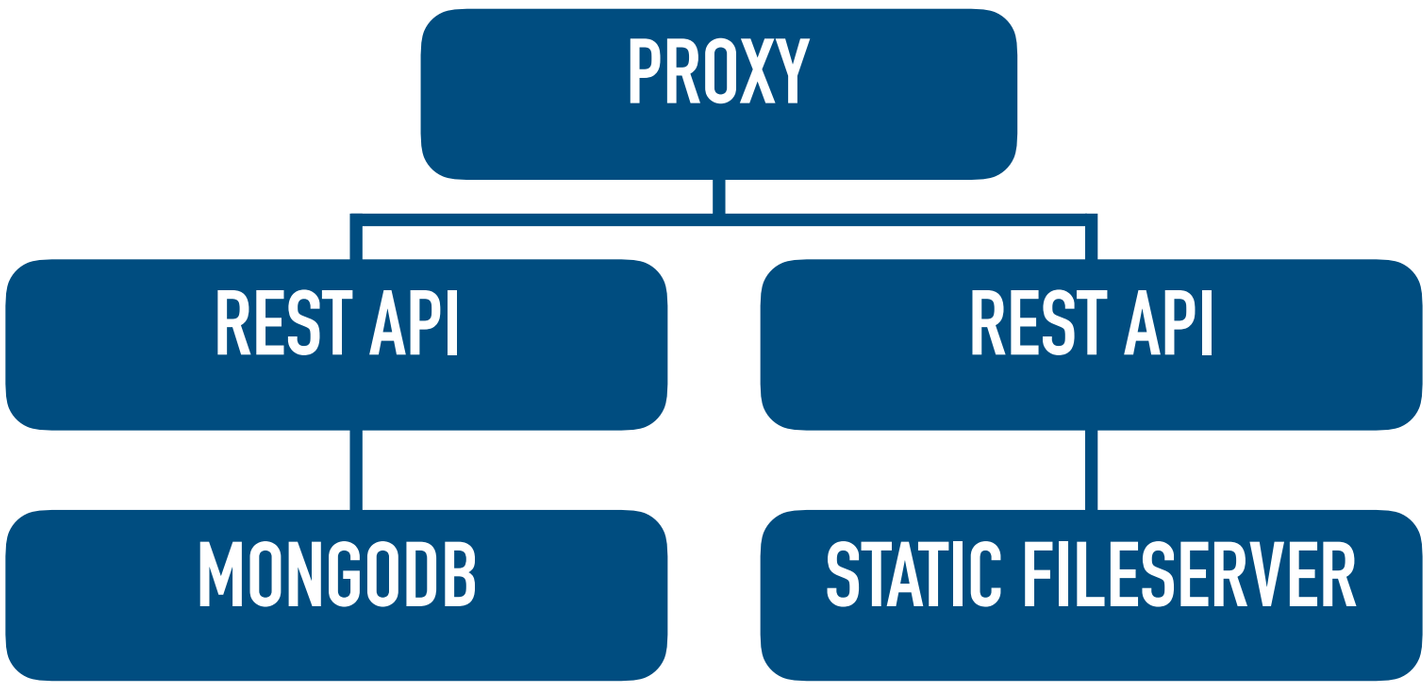
- ¿Qué es docker, qué es un contenedor?
- Conceptos Básicos y primeros pasos
- **Herramientas, ecosistema y casos de uso**
- CiDAEN

Ecosistema y herramientas

- Es una de las tecnologías de mayor y más rápida adopción en mundo del desarrollo software
- Ha permitido un nuevo nivel de **desarrollo ágil** y es el soporte de la c**omputación en la nube**

Caso de uso: Microservicios

- Elimina las restricciones de la infraestructura subyacente
- Agiliza la orquestación del sistema completo
- Independiza cada componente del sistema



Caso de uso: Microservicios

- Un fichero de configuración que nos permite definir contenedores a modo de servicios y sus recursos compartidos:

```
services:
   webserver:
      image: cidaen/website
      ports:
       - 80:80
      volumes:
       - app:/var/www/
   database:
      image: mysql/mysql
   environment:
      MYSQL_USER: user
      MYSQL_PASSWORD: unbreakablepass
      MYSQL_DATABASE: cidaen
```

```
docker-compose up
```

```
docker-compose down
```

```
docker-compose run
```

DEMO 5: DOCKER-COMPOSE

API REST con MongoDB

Wordpress

```
cd demo5/rest-tutorial/flask1
```

app.py es una aplicación que despliega una API REST en python

Para desplegarla en modo host podemos instalar los requirements sobre un virtualenv

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
python app.py
```

```
cd demo5/rest-tutorial/flask2-mongo
```

Esta versión utiliza mongodb, que podemos desplegar con docker

```
docker run --rm –name mongo \
    -p 27017:27017 mongo
```

```
python app.py
```

Nuestra aplicación se conecta por localhost a mongodb. ¿Y si queremos desplegar la app en docker? En lugar de usar dos imágenes por separado usaremos docker-compose

```
cd demo5/rest
```

Esta versión es una aplicación de docker-compose toda la información esta en docker-compose.yml

```
docker-compose build
```

```
docker-compose up
```

Desplegamos nuestros microservicios

Eliminamos los recursos creados

```
docker-compose down
```

```
cd demo5/wordpress
```

```
docker-compose up
```

Desplegamos nuestros microservicios

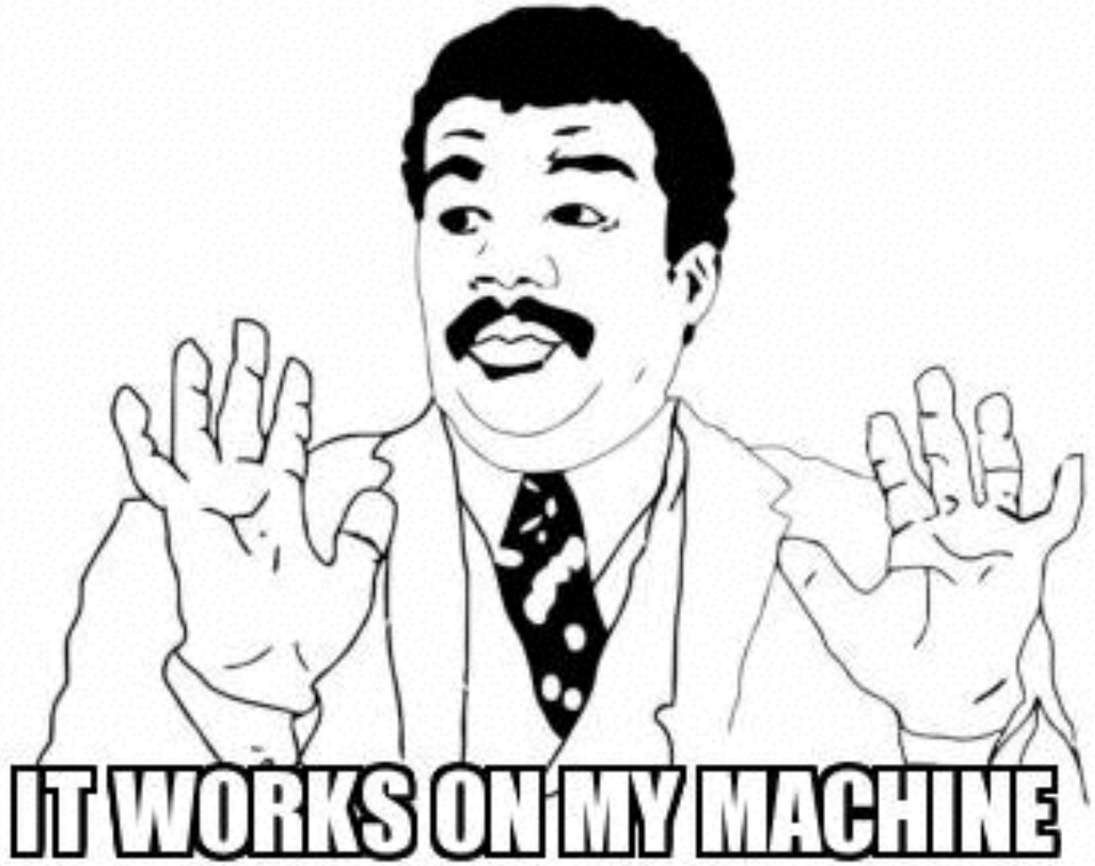
Eliminamos los recursos creados (incluso los datos)

```
docker-compose down
```

Caso de uso: Devops

- Se pierde mucho tiempo en **normalizar** los entornos de desarrollo y producción

- Impone **restricciones** a la **agilidad** del ciclo de despliegue



Devops

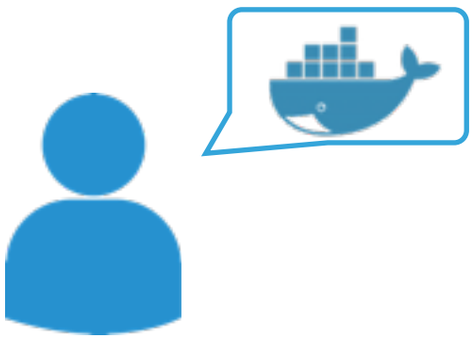
- **Unifica** entornos de desarrollo, test y producción
- **Acelera** el ciclo de desarrollo y despliegue
- **Incrementa** la automatización y la productividad

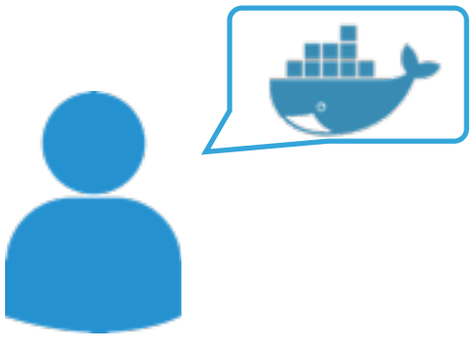
CI y CD (Integración y despliegue contínuos)

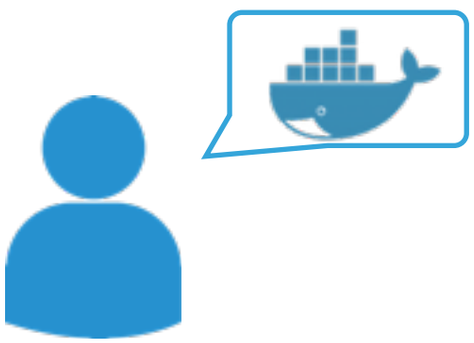
DEVELOP

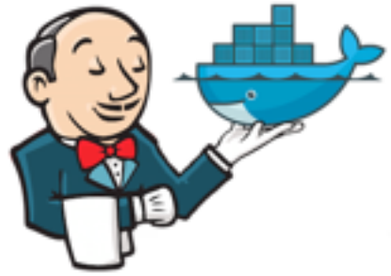
CI

CD









TEST TEST

BUILD BUILD

DEPLOY

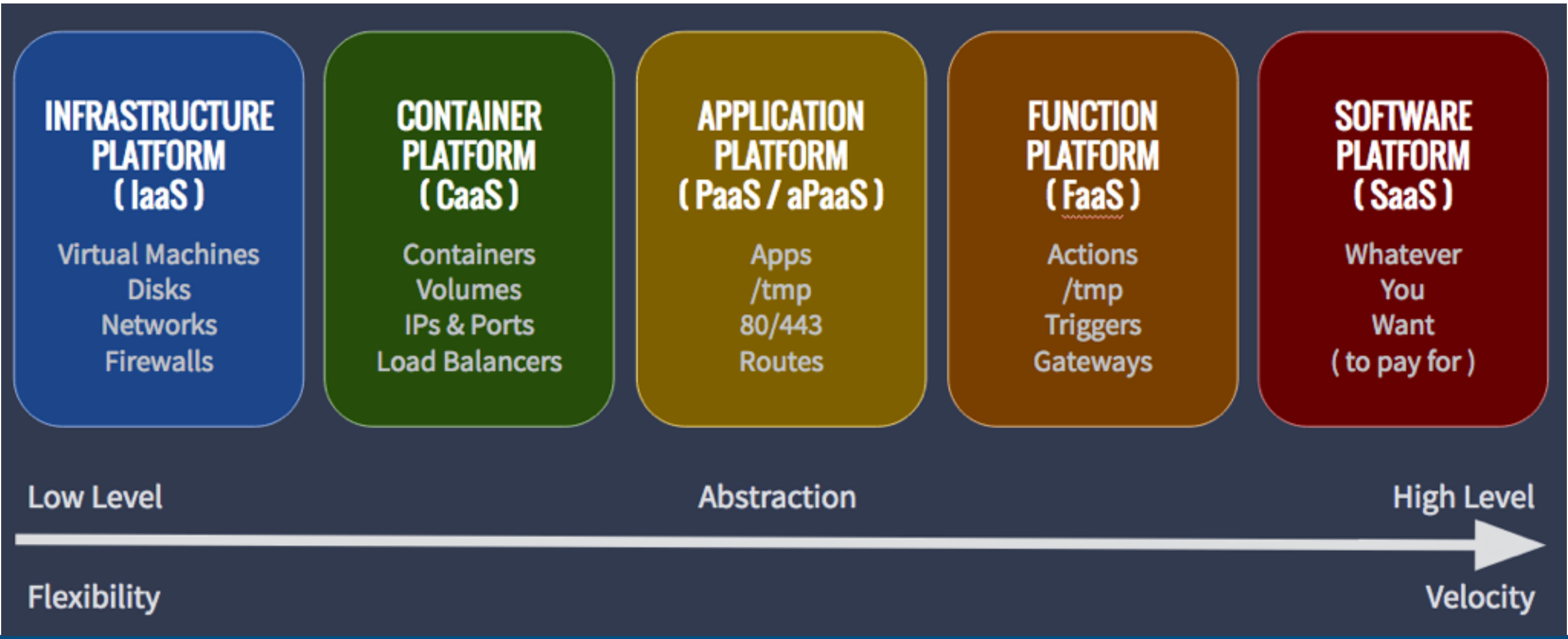
DEMO: DEVOPS

CI con travis-ci

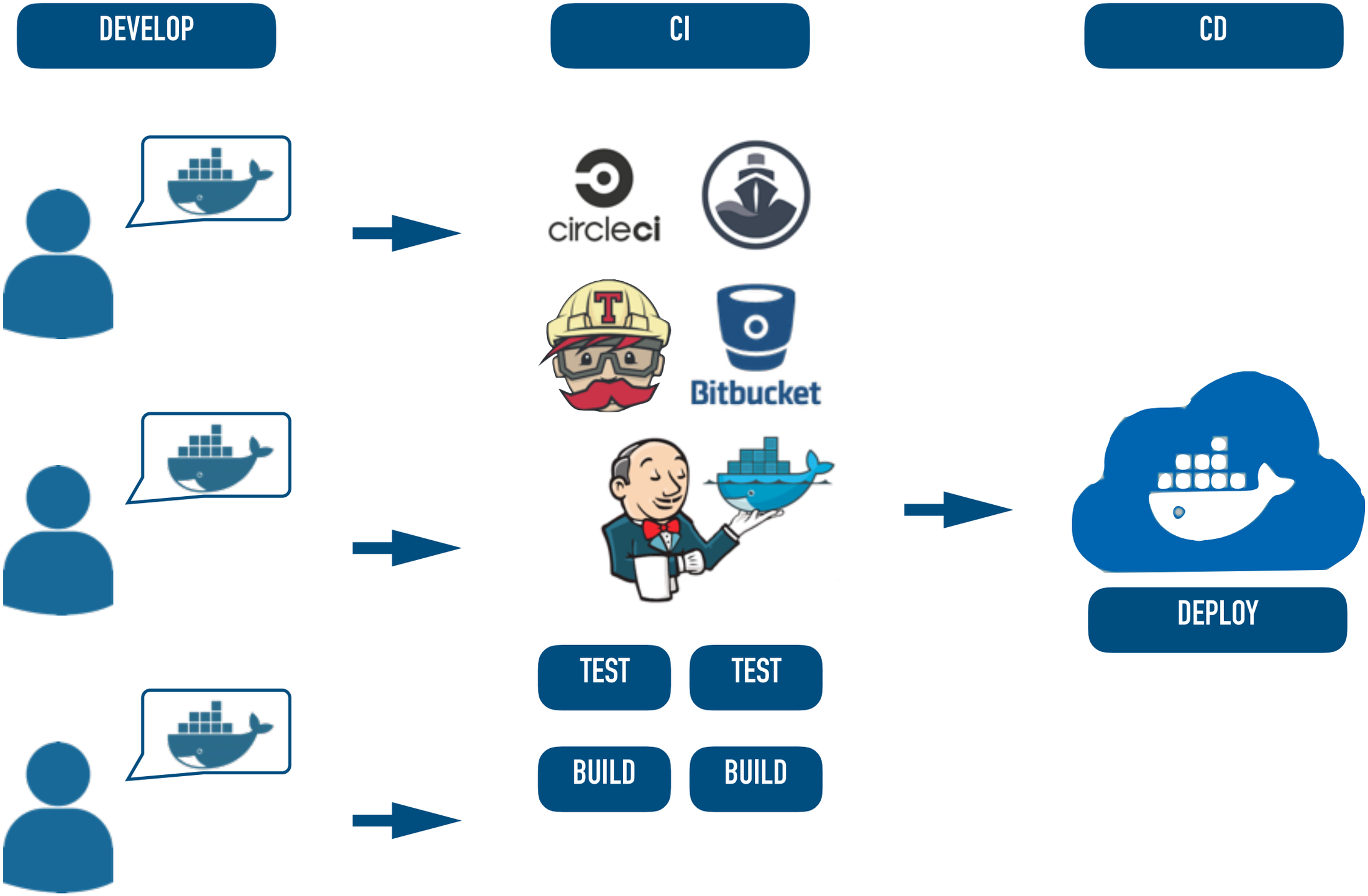
Caso de uso: Containers As A Service (PaaS)

- Proveedores de cloud público (AWS, Azure, Google, IBM) permiten desplegar contenedores directamente

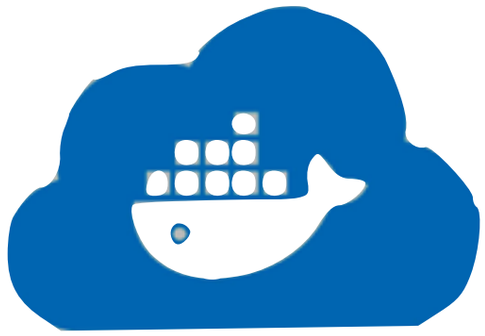
- Nos abstraemos de la infraestructura, que es gestionada por ellos. Sin servidores, gestión a nivel de docker.

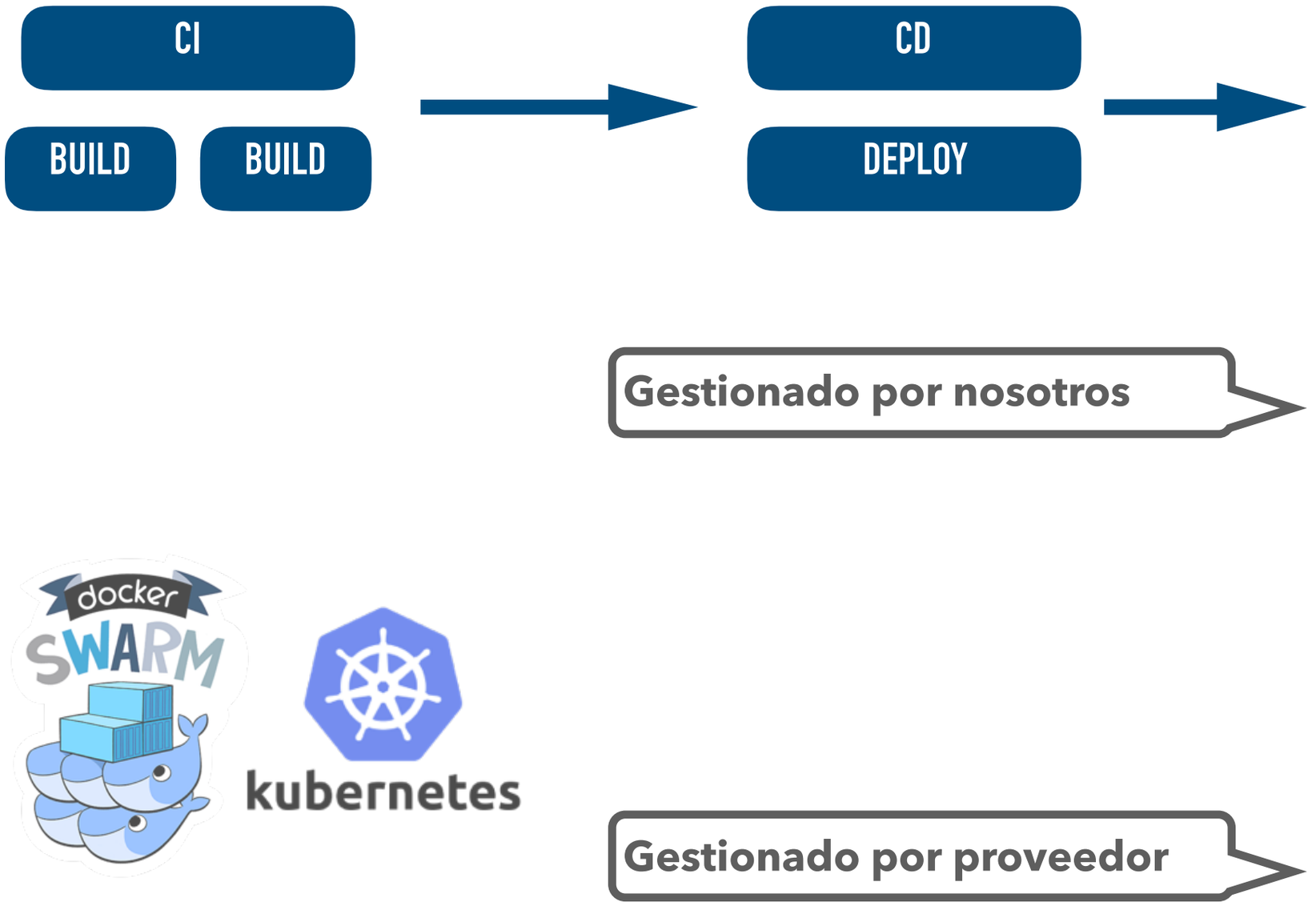


Delivery a nivel cloud



Delivery a nivel cloud





kubernetes

Introducción a Docker

- ¿Qué es docker, qué es un contenedor?
- Conceptos Básicos y primeros pasos
- Herramientas, ecosistema y casos de uso
- **CiDAEN**

http://www.cidaen.es (1ª ed)

- **Segunda edición anual Sep 2018 / Jun 2019**
- **Curso orientado a currículum profesional**
- **Contenidos destacados:**
  - Data Science con python: numpy, pandas, jupyter
  - Exploración y visualización de datos
  - Bases de datos NoSQL
  - Machine Learning
  - Big Data con Apache Hadoop y Apache Spark
  - Servicios Cloud en AWS
  - Creación de aplicaciones escalables (docker, serverless)

Introducción a Docker

CiDAEN

Jacinto Arias

@cidaen

@jacintoarias

Curso de Especialista en Ciencia de Datos y Aplicaciones Escalables en la Nube