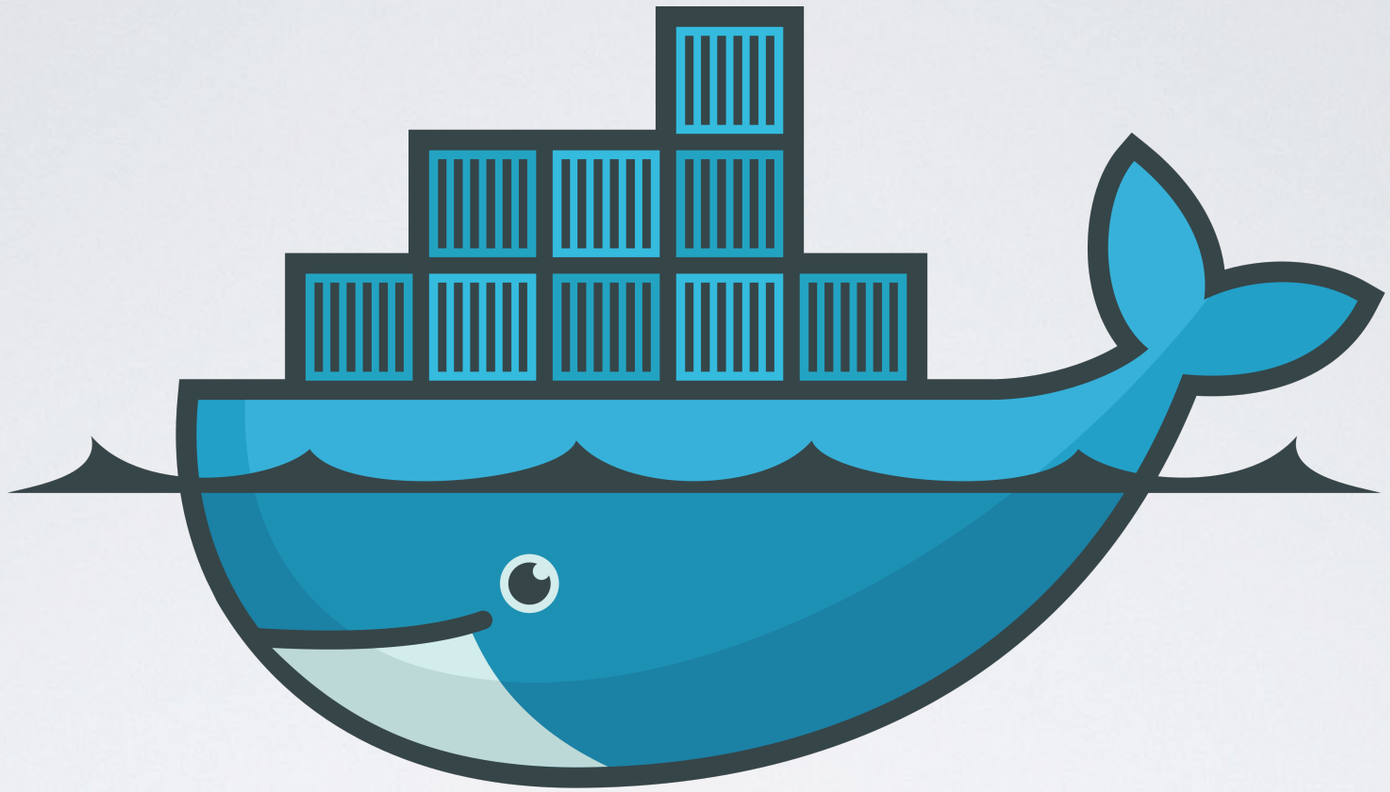


docker

PARA JAVEROS

¿Y TU QUIÉN ERES?

- Daniel Rodríguez Gil
- Programador senior en Optare Solutions
- Me encanta aprender cosas nuevas

Java - Docker - Node.js - Groovy
Gradle - PHP - Angular - Vue.js - ¿VIM?

@danybmx

¿QUÉ ES DOCKER?

- "Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización a nivel de sistema operativo en Linux." (Wikipedia)

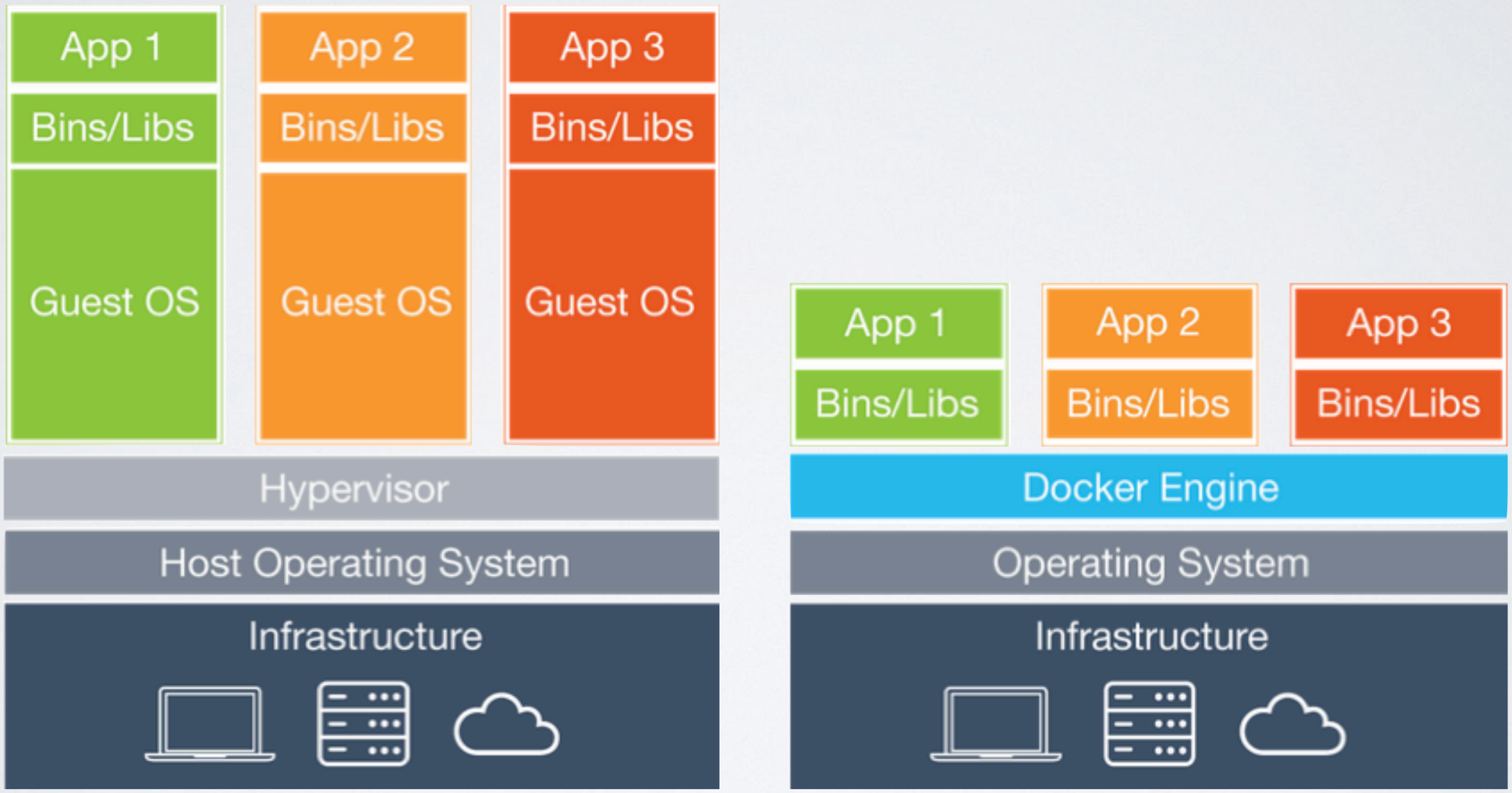
- Aparece en **2013** y se basa en la ideología de **Build**, **Ship** and **Run Anywhere.**

- **Build** es la fase en la que construimos una imagen que contiene el aplicativo y sus dependencias.

- **Ship** existen repositorio de imágenes de los que podemos descargar y enviar estas imágenes.

- **Run Anywhere** siempre que docker engine esté disponible, podremos correr cualquier imagen.

DIFERENCIAS CON LAS MÁQUINAS VIRTUALES



- Menor tiempo de arranque que una VM, de varios minutos a pocos segundos.

- Cada una de las instancias tienen un tamaño mucho menor ya que no necesitan tener el SO completo.

- Mejor aprovechamiento de los recursos, no solo del filesystem, también CPU y Memoría al no tener que correr un SO por cada instancia.

- Por otro lado, docker-engine debe correr sobre Linux ya que usa propiedades del kernel como namespaces, cgroups, etc… en windows / macOS él mismo creará una máquina virtual para esto.

¿POR QUÉ USAR JAVA EN CONTENDORES DOCKER?

- Java es agnóstico en cuanto al Hardware y Sistema Operativo.
- Es más seguro al ser la JVM la que gestiona la seguridad y protección de los recursos.
- La JVM es capaz de adaptarse a los cambios en el entorno para asegurar una ejecución estable.
- Es posible observar un progreso en Java en cuanto a su ejecución sobre contenedores (como vemos en java 10).
- Existe un gran ecosistema y comunidad.

¿POR DÓNDE EMPEZAMOS?

Lo primero es descargar docker e instalarlo en nuestra máquina

https://www.docker.com/community-edition

DOCKER ENGINE

- docker ps -all
- docker build -t imagename:tag .
- docker tag imagename:tag newimagename:newtag
- docker login
- docker push/pull [repo/]user/imagename:tag
- docker run imagename/imageid
- docker exec containerid command
- docker history imagename/imageid
- docker start/stop containerid
- docker images -f "reference=imagename"
- docker rm containerid
- docker rmi imageid

https://docs.docker.com/engine/reference/commandline/docker/#description

BUILD, SHIP & RUN ANYWHERE

BUILD

REFERENCIA DOCKERFILE

- FROM
- LABEL
- RUN
- ENTRYPOINT
- CMD
- ARG
- EXPOSE
- ENV
- COPY
- ADD
- USER
- WORKDIR
- ~~MAINTAINER~~ *(deprecated)*

```
LABEL maintainer="daniel@dpstudios.es"
```

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

BUILD

Dockerfile

```
# Definimos que imagen vamos a utilizar como base
FROM alpine:3.7

# Definimos el comando que se va a lanzar al iniciar la instancia
ENTRYPOINT ["echo"]

# Definimos los parámetros que se enviarán al entrypoint
CMD ["Hello VigoJUG!"]
```

Build

```
$ docker build -t hello-vigojug:latest .
```

List

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-vigojug       latest              73aae8d14c4f        41 seconds ago      4.15MB
alpine              3.7                 3fd9065eaf02        2 months ago        4.15MB
```

SHIP

Prepare for push

```
docker tag hello-vigojug:latest danybmx/hello-vigojug:latest
```

Login on docker hub

```
docker login
```

Push

```
docker push danybmx/hello-vigojug:latest
```

Pull Esto se hace automáticamente al hacer un run

```
docker pull danybmx/hello-vigojug:latest
```

RUN ANYWHERE

```
$ docker run danybmx/hello-vigojug:latest
Hello VigoJUG!
```

DOCKER, TE PRESENTO A JAVA

Structure

```
$ tree
├── Dockerfile
├── bin
│   └── MainApplication.class
└── src
    └── MainApplication.java
```

MainApplication.java

```
class MainApplication {
    public static void main(String[] args) {
        System.out.println("This is a Java hello for the VigoJug!");
    }
}
```

Dockerfile

```
# Base image
FROM openjdk:8-jre

# Copy main class
COPY bin/MainApplication.class /app/MainApplication.class

# Define the working directory
WORKDIR "/app"

# Define command to run
ENTRYPOINT ["java"]
CMD ["MainApplication"]
```

Build & Run

```
Build & Run$ docker build -t hello-from-java:jre8 .
[…]
Successfully tagged hello-from-java:jre8

$ docker run hello-from-java:jre8
This is a Java hello for the VigoJug!
```

LO "MALO", EL TAMAÑO

Show image

```
$ docker images -f "reference=hello-from-java:jre8"
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-from-java     jre8                1e851968f737        27 seconds ago      527MB
```

Image history

```
$ docker history 1e851968f737
IMAGE               CREATED             CREATED BY                                      SIZE
1e851968f737        55 seconds ago      /bin/sh -c #(nop)  CMD ["MainApplication"]      0B
b3f0241a74ad        55 seconds ago      /bin/sh -c #(nop)  ENTRYPOINT ["java"]          0B
d6754c8320e1        55 seconds ago      /bin/sh -c #(nop) WORKDIR /app                  0B
51cc7e78fa26        55 seconds ago      /bin/sh -c #(nop) COPY file:2307f1b07375aea3…   461B
1b56aa0fd38c        12 days ago         /bin/sh -c /var/lib/dpkg/info/ca-certificate…   394kB
<missing>           12 days ago         /bin/sh -c set -ex;   if [ ! -d /usr/share/m…   393MB
<missing>           12 days ago         /bin/sh -c #(nop)  ENV CA_CERTIFICATES_JAVA_…   0B
<missing>           12 days ago         /bin/sh -c #(nop)  ENV JAVA_DEBIAN_VERSION=8…   0B
<missing>           12 days ago         /bin/sh -c #(nop)  ENV JAVA_VERSION=8u162       0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  ENV JAVA_HOME=/docker-jav…   0B
<missing>           2 weeks ago         /bin/sh -c ln -svT "/usr/lib/jvm/java-8-open…   33B
<missing>           2 weeks ago         /bin/sh -c {   echo '#!/bin/sh';   echo 'set…   87B
<missing>           2 weeks ago         /bin/sh -c #(nop)  ENV LANG=C.UTF-8             0B
<missing>           2 weeks ago         /bin/sh -c apt-get update && apt-get install…   2.05MB
<missing>           2 weeks ago         /bin/sh -c set -ex;  if ! command -v gpg > /…   7.8MB
<missing>           2 weeks ago         /bin/sh -c apt-get update && apt-get install…   23.8MB
<missing>           2 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>           2 weeks ago         /bin/sh -c #(nop) ADD file:b380df301ccb5ca09…   100MB
```

CUSTOM JRE

- El JDK o JRE completo añade mucho peso a nuestras imágenes.

- En Java 9 se añade el sistema de módulos que nos permite crear JREs solo con los módulos que necesitamos para la aplicación.

- Dockerfile multistage.

Dockerfile.jre9

```
# Base image for use as the builder
FROM openjdk:9-jdk as java-builder

WORKDIR /jlink/outputdir

# Run jlink tool to generate the custom jre
RUN jlink --module-path /docker-java-home/jmods --strip-debug \
    --compress=2 --output java --add-modules java.base

# Base image for the custom jre
# The same that the openjdk:8-jre uses
FROM buildpack-deps:stretch-curl

# Copy custom jre built
COPY --from=java-builder /jlink/outputdir /jre

# Copy the class
COPY bin/MainApplication.class /app/MainApplication.class

# Add java the the path
ENV PATH /jre/java/bin:$PATH

# Define the working directory
WORKDIR /app

# Define command to run
ENTRYPOINT ["java"]
CMD ["MainApplication"]
```

Build stage

Build it!

```
$ docker build -t hello-from-java:jre9 -f Dockerfile.jre9 .
```

Comprobamos el tamaño

527MB

```
$ docker images -f "reference=hello-from-java:jre9"
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-from-java     jre9                66773b743eba        4 minutes ago       162MB
```

Más detalle

393MB

```
$ docker history 66773
130
IMAGE               CREATED             CREATED BY                                      SIZE
COMMENT
66773b743eba        5 minutes ago       /bin/sh -c #(nop)  CMD ["MainApplication"]      0B
41dda20d6c97        5 minutes ago       /bin/sh -c #(nop)  ENTRYPOINT ["java"]          0B
911b616927f3        5 minutes ago       /bin/sh -c #(nop) WORKDIR /app/                 0B
0b304f5d5c60        5 minutes ago       /bin/sh -c #(nop)  ENV PATH=/jre/java/bin:/u…   0B
8d582393f8c8        5 minutes ago       /bin/sh -c #(nop) COPY file:2307f1b07375aea3…   461B
ee78c04e5017        5 minutes ago       /bin/sh -c #(nop) COPY dir:9eee70976e994bc5c…   29.9MB
339c13797659        5 minutes ago       /bin/sh -c #(nop) WORKDIR /jre/                 0B
2fd722140cac        2 weeks ago         /bin/sh -c set -ex;  if ! command -v gpg > /…   7.8MB
<missing>           2 weeks ago         /bin/sh -c apt-get update && apt-get install…   23.8MB
<missing>           2 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>           2 weeks ago         /bin/sh -c #(nop) ADD file:b380df301ccb5ca09…   100MB
```

¿Se puede afinar más?

Podemos buscar imágenes base más ligeras.
Alpine usa la librería musl y JDK no es compatible.
Más info: http://openjdk.java.net/projects/portola/

DOCKER-COMPOSE

- Docker compose es una herramienta para definir y gestionar servicios con múltiples contenedores docker.
- La configuración es sencilla y rápida, tan solo se usar un archivo docker-compose.yml.
- Por defecto crea una red privada para los contenedores que gestione.

https://docs.docker.com/compose/overview/

DOCKER-COMPOSE.YML

```
version: "2"
services:
  api:
    image: 'org.vigojug/api:latest'
    ports:
      - 8080
    environment:
      VIRTUAL_HOST: "*/api*"
  app:
    image: 'org.vigojug/app:latest'
    ports:
      - 8080
    environment:
      VIRTUAL_HOST: "*/web*"
  load_balancer:
    image: 'dockercloud/haproxy:latest'
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      - BALANCE=roundrobin
    links:
      - app
      - api
    ports:
      - 80:80
      - 1936:1936
```

https://docs.docker.com/compose/compose-file/compose-file-v2/#compose-and-docker-compatibility-matrix

DOCKER-COMPOSE COMMANDS

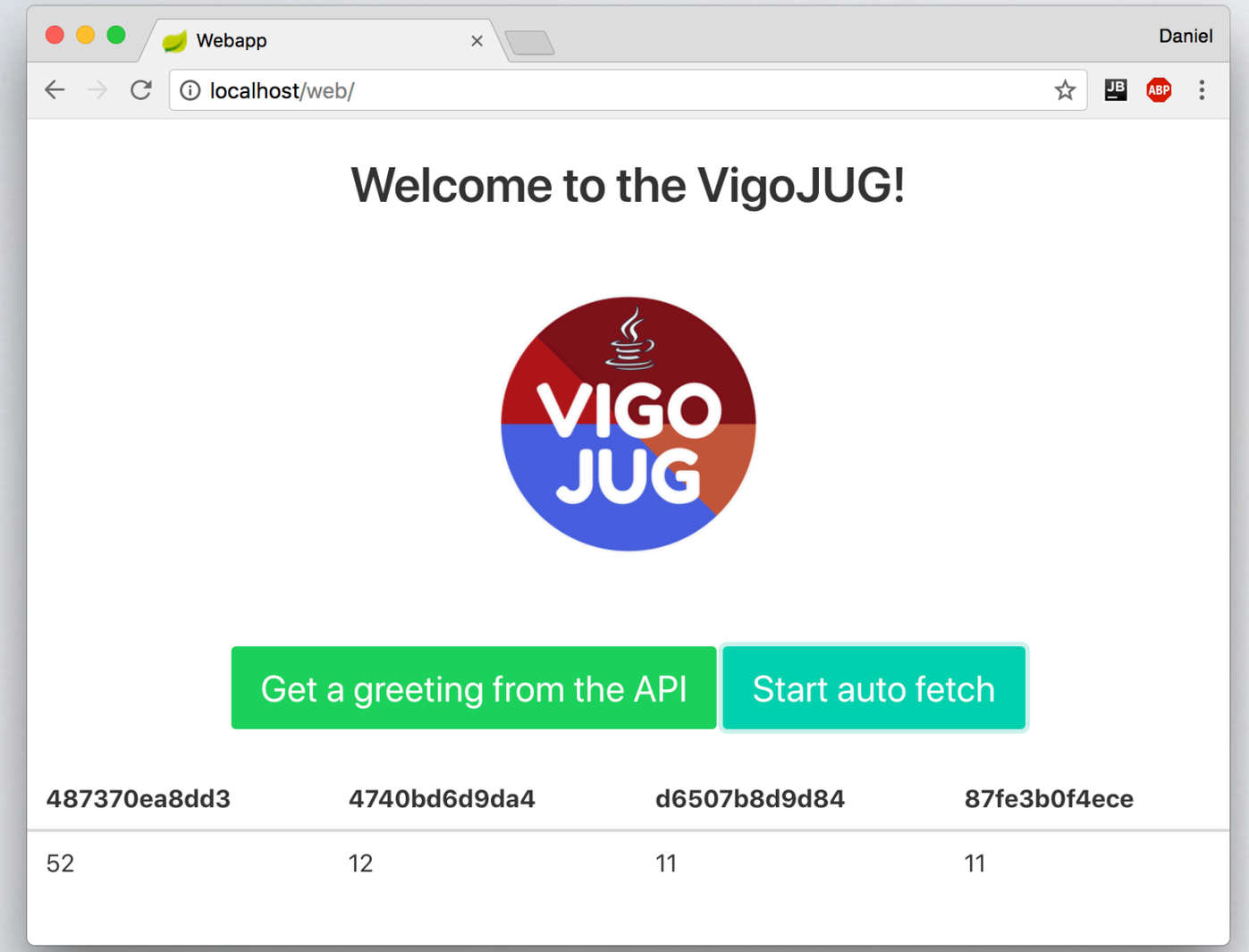
```
build              Build or rebuild services
config             Validate and view the Compose file
create             Create services
down               Stop and remove containers, networks, images, and volumes
exec               Execute a command in a running container
kill               Kill containers
logs               View output from containers
ps                 List containers
pull               Pull service images
push               Push service images
restart            Restart services
rm                 Remove stopped containers
scale              Set number of containers for a service
start              Start services
stop               Stop services
top                Display the running processes
up                 Create and start containers
```

https://docs.docker.com/compose/reference/overview/

MEJORAS DE JAVA 10 PARA DOCKER

- Tamaño del Heap
  - Por defecto la JVM usa 1/4 de la memoria física de la máquina como MaxHeap, en Java 10 la JVM tiene en cuenta el límite de memoria del contenedor. (en Java 9 hay varios workarounds)
- CPUs disponibles
  - Por defecto la JVM cuenta el numero de procesadores total de la máquina física, por lo que si estos están limitados en el contenedor, la JVM creará los threads, pools, etc… teniendo en cuenta un valor incorrecto.
- Attach to the JVM
  - La JVM, a través del Attach API, permite que una JVM se conecte a otra, esto es útil para ver el estado de la JVM, hacer profiling, tareas de diagnóstico, etc…
  - En versiones anteriores esto no funciona correctamente y el problema es que la JVM no es consciente de cual es su PID en la root namespace. En Java 10, se arregla este comportamiento.

CASO DE USO



- API
  - Springboot
- APP
  - Springboot
  - Angular5
- LOAD BALANCER
  - ha_proxy