Introducción a Containers

Container Engines - ECS

Mauricio Améndola / Sebastián Orrego – Profesor Adjunto
Escuela de Tecnología – Facultad de Ingeniería

UNIVERSIDAD ORT
Uruguay

AGENDA

1. Mastering building images
    a. Buenas prácticas
    b. Buildkit
    c. Multi-stage

Mastering building images

Construyendo imágenes

Cómo DevOps una de las tareas que tenemos que hacer es ensamblar imágenes que después darán vida a containers.

Buenas imágenes -> Containers saludables -> aplicaciones performantes -> Usuarios felices

Buenas prácticas

1 - El orden de las instrucciones

Por ejemplo, tenemos la instrucción COPY para copiar archivos desde el host a la imagen. Las instrucciones se deben especificar según su frecuencia de cambio.

```
FROM debian
COPY . /app
RUN apt-get update
RUN apt-get -y install openjdk-8-jdk ssh vim
COPY . /app
CMD ["java", "-jar", "/app/target/app.jar"]
```

Order from least to most frequently changing content.

2 - Copias específicas

Evitar copiar archivos que no serán usados ni por la compilación ni por la aplicación.

```
FROM debian
RUN apt-get update
RUN apt-get -y install openjdk-8-jdk ssh vim
COPY . /app
COPY target/app.jar /app
CMD ["java", "-jar", "/app/target/app.jar"]
```

Only copy what's needed. Avoid "COPY ." if possible

3 - Reducir cantidad de layers

Una imagen saludable, es una imagen con el mínimo de layers posible y lo más liviana posible. Reducir a one-liners las instrucciones posibles.

```
FROM debian
RUN apt-get update
RUN apt-get -y install openjdk-8-jdk ssh vim
RUN apt-get update \
 && apt-get -y install \
    openjdk-8-jdk ssh vim
COPY target/app.jar /app
CMD ["java", "-jar", "/app/app.jar"]
```

Prevents using an outdated package cache

4 - Instalar solo paquetes necesarios

Evitar instalar paquetes que no serán necesarios ni para el proceso de construcción, ni para la aplicación.

```
FROM debian
RUN apt-get update \
 && apt-get -y install --no-install-recommends \
    openjdk-8-jdk ssh vim
COPY target/app.jar /app
CMD ["java", "-jar", "/app/app.jar"]
```

5 - Limpiar dependencias / cache

Al instalar paquetes o actualizar versiones base de las imágenes, estamos descargando temporales que quedan en la imagen. Es saludable limpiar esto.

```
FROM debian
RUN apt-get update \
 && apt-get -y install --no-install-recommends \
    openjdk-8-jdk \
 && rm -rf /var/lib/apt/lists/*
COPY target/app.jar /app
CMD ["java", "-jar", "/app/app.jar"]
```

6 - Usar imágenes oficiales

En la medida de lo posible, usar imágenes oficiales o imágenes cuyo footprint sea bajo, por ejemplo, *alpine*. A veces esto no es posible, pero nos puede ayudar a generar imágenes más saludables.

```
FROM debian
RUN apt-get update \
 && apt-get -y install --no-install-recommends \
    openjdk-8-jdk\
 && rm -rf /var/lib/apt/lists/*
FROM openjdk
COPY target/app.jar /app
CMD ["java", "-jar", "/app/app.jar"]
```

7 - Usar tags específicos

Conviene usar tags específicos donde sepamos exactamente qué incluye esa imagen. Evitar usar el tag *latest* dado que las imágenes se actualizan y este tag puede introducir cambios que rompen la aplicación.

```
~~FROM openjdk:latest~~
FROM openjdk:8
COPY target/app.jar /app
CMD ["java", "-jar", "/app/app.jar"]
```

8 - Usar “sabores” livianos

Muchas imágenes oficiales contienen varios “sabores” de imágenes, que están basados en distintas distribuciones.

REPOSITORY	TAG	SIZE
openjdk	8	624MB
openjdk	8-jre	443MB
openjdk	8-jre-slim	204MB
openjdk	8-jre-alpine	83MB

9 - Build from source

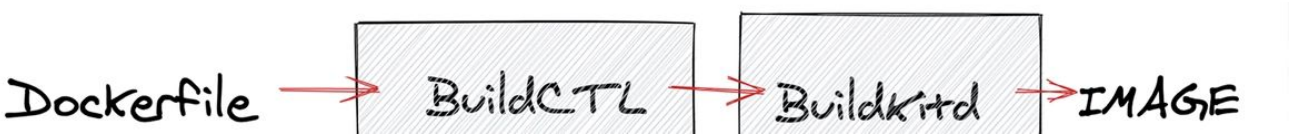
En la medida de lo posible, usar imágenes base con las tools necesarias para construir la app. En el ejemplo que venimos siguiendo, podemos usar una imagen basada en maven para compilar.

```
~~FROM openjdk:8-jre-alpine~~
FROM maven:3.6-jdk-8-alpine
WORKDIR /app
~~COPY app.jar /app~~
COPY pom.xml .
COPY src ./src
RUN mvn -e -B package
CMD ["java", "-jar", "/app/app.jar"]
```

10 - Usar Buildkit

A partir de Docker 18.09 se introduce este sistema para mejorar la performance al momento de buildear las imágenes. Entre otras cosas, buildkit permite:

- ➔ Buildear layers en paralelo
- ➔ Mejora manejo de storage
- ➔ Mejora procesamiento y caching



10 - Usar Buildkit

Para usar buildkit se puede activar al momento de invocar al comando de build

- ***DOCKER_BUILDKIT=1 docker build -t*** ……
- Usando el comando ***docker buildx***
- Configurar el docker daemon para que siempre use buildkit

```
cat /etc/docker/daemon.json
{
    "features": {
        "buildkit": true
    }
}
```

11 - Usar multi-stages

Si consideramos el siguiente Dockerfile...

El comando ***go build -o*** genera un binario con la aplicación.

11 - Usar multi-stages

Esta imagen ya tiene un footprint bastante alto porque esta basada en una imagen que contiene todas las herramientas necesarias para compilar y ejecutar Go.

Entonces, cómo hacemos para compilar el código y que el resultado sea una imagen minimal?

11 - Usar multi-stages

Multi-stages permite crear etapas dentro de un mismo Dockerfile. De esta forma, generamos un stage para compilar el código y un stage con los pasos para ejecutar la app. Finalmente, la imagen se creará a partir de este último stage.

11 - Usar multi-stages

Siguiendo el ejemplo de Golang.

Bloque con las instrucciones para compilar el código

Bloque con las instrucciones para ejecutar el código compilado


SHOW TIME!!!
imgflip.com