

Creando Imágenes Docker

Preparar el entorno de trabajo

Instalar Docker y Docker Compose según las bitácoras en sitio de la cátedra.

Crear un container

Creamos un directorio y dentro de él un archivo para definir nuestro container:

```
mkdir test-curl  
cd test-curl  
nano Dockerfile
```

Ingresar el siguiente texto en el archivo:

```
FROM ubuntu:18.04  
  
RUN apt-get update  
RUN apt-get install -y curl  
  
ENV SITE_URL http://ifconfig.co/  
  
CMD sh -c "curl -Lks $SITE_URL"
```

Guardar y cerrar (CTRL-O y luego CTRL-X)

Explique qué hace cada línea del Dockerfile.

En el mismo directorio donde está el archivo Dockerfile ejecutar

```
docker build -t test-curl .
```

¿Qué hace el comando anterior?

Ver la imagen creada con:

```
docker images -f reference='test-curl'
```

¿Qué hace el modificador -f?

Probamos el container con:

```
docker run --rm test-curl
```

¿Qué hace el modificador --rm?

Ahora probamos pasando otra URL mediante la variable de entorno:

```
docker run --rm -e SITE_URL=https://ifconfig.co/country test-curl
```

Crear una versión “alpine” del container

Modificar el archivo Dockerfile para que quede de la siguiente manera:

```
FROM alpine:3.11

RUN apk add curl

ENV SITE_URL http://ifconfig.co/

CMD sh -c "curl -Lks $SITE_URL"
```

En el mismo directorio donde está el archivo Dockerfile ejecutar **docker build -t test-curl-alpine .**

Ver la imagen creada con:

docker images -f reference='test-curl*'

Compare el tamaño con la imagen anterior basada en ubuntu.

Probamos el container con:

docker run --rm test-curl-alpine

Ahora probamos pasando otra URL mediante la variable de entorno:

docker run --rm -e SITE_URL=https://ifconfig.co/country test-curl-alpine

Las imágenes basadas en [Alpine](https://www.alpinelinux.org/) son sensiblemente más pequeñas, pero hay que tener cuidado al utilizarlas (especialmente para compilar código) porque tienen diferencias que pueden provocar errores o pérdida de performance en algunos casos.

- <https://www.youtube.com/watch?v=e2pAkqYCG8>
- <https://pythonspeed.com/articles/alpine-docker-python/>

Aplicaciones de prueba

Para las siguientes actividades del práctico utilizaremos dos aplicaciones de ejemplo: un backend REST (usando [Micronaut](#)) y un frontend (usando [ReactJS](#)).

Para esta etapa vamos a necesitar instalar varias herramientas:

- [cURL](#) (en debian/ubuntu/mint: **sudo apt install curl**).
- [Git](#) (en debian/ubuntu/mint: **sudo apt install git**).
- [OpenJDK](#) 8 (en debian/ubuntu/mint: **sudo apt install openjdk-8-jdk**).
- [NodeJS](#) y [NPM](#) (en debian/ubuntu/mint: **sudo apt install nodejs npm**).

Descargar, compilar y probar el backend

Descargar el código del proyecto backend con:

```
cd ~  
git clone https://github.com/tics-frcu/docker-sample-backend-micronaut.git
```

Ingresar al directorio y ejecutar la aplicación con:

```
cd docker-sample-backend-micronaut  
./gradlew run
```

Este comando va a demorar unos minutos la primera vez, ya que en ese caso descargará gradle y todas las dependencias del proyecto.

Al iniciar el backend quedará esperando solicitudes en: <http://localhost:8080>

Una vez inicie la aplicación, abrir otra consola y ejecutar en la misma los siguientes comandos:

```
curl http://localhost:8080/hola  
curl http://localhost:8080/hola/Motoko
```

La funcionalidad es muy elemental: el backend acepta llamadas HTTP y responde con un mensaje de texto.

Descargar, compilar y probar el frontend

En una nueva consola descargar el código del proyecto frontend con:

```
cd ~  
git clone https://github.com/tics-frcu/docker-sample-frontend-react.git
```

Ingresar al directorio y ejecutar la aplicación con:

```
cd docker-sample-frontend-react  
yarn install  
yarn start
```

El comando install va a demorar unos minutos la primera vez, ya que en ese caso descargará todas las dependencias del proyecto.

Una vez inicie, nos abrirá un navegador con la página <http://localhost:3000>

En esta app escribimos un nombre (o dejamos en blanco el campo) y presionamos el botón "Enviar". Debajo nos aparecerá la respuesta del backend a la solicitud.

Crear imagenes de containers para aplicaciones

Detener con CTRL+C las aplicaciones en caso de que estén corriendo.

Imagen para el proyecto backend

Crear un Dockerfile para el proyecto backend:

```
cd ~/docker-sample-backend-micronaut  
nano Dockerfile
```

Ingresar el siguiente texto en el archivo:

```
FROM gradle:5.3-jdk8  
  
WORKDIR /home/gradle  
  
ADD . /home/gradle  
  
# Compilo la aplicación (como un fat jar)  
RUN gradle assemble --no-build-cache --no-daemon  
  
# Copio el fat jar a la raíz del Workdir  
RUN cp build/libs/*all.jar application.jar  
  
EXPOSE 8080  
  
CMD ["java", "-jar", "-XX:+UseContainerSupport", \  
    "-Djava.security.egd=file:/dev/./urandom", "application.jar"]
```

Guardar y cerrar (CTRL-O y luego CTRL-X)

Explique que hacen los comandos de este Dockerfile.

¿Por qué para copiar el archivo jar uso **RUN** y no **COPY**?

En el mismo directorio donde está el archivo Dockerfile ejecutar:

```
docker build -t backend:single-stage .
```

Probamos el container con:

```
docker run --rm -p 8080:8080 -h localhost backend:single-stage
```

¿Qué hacen los modificadores **-p** y **-h**?

Una vez inicie el container, abrir otra consola y ejecutar en la misma los siguientes comandos:

```
curl http://localhost:8080/hola  
curl http://localhost:8080/hola/Motoko
```

Imagen para el proyecto frontend

Crear un Dockerfile para el proyecto backend:

```
cd ~/docker-sample-frontend-react  
nano Dockerfile
```

Ingresar el siguiente texto en el archivo:

```
FROM node:8.10.0  
  
WORKDIR /app  
  
ADD . /app  
  
# Instalo las dependencias  
RUN yarn install --silent  
  
EXPOSE 3000  
  
CMD ["yarn", "start"]
```

Guardar y cerrar (CTRL-O y luego CTRL-X)

Explique que hacen los comandos de este Dockerfile.

En el mismo directorio donde está el archivo Dockerfile ejecutar:

```
docker build -t frontend:single-stage .
```

Probamos el container con:

```
docker run --rm -p 3000:3000 -h localhost frontend:single-stage
```

Una vez inicie el container, abrir un navegador e ingresar a: <http://localhost:3000>

Crear imágenes de múltiples etapas

Detener los containers en caso de que estén corriendo.

Proyecto backend

Modificar el Dockerfile del proyecto backend:

```
cd ~/docker-sample-backend-micronaut  
nano Dockerfile
```

El Dockefile deberá quedar de la siguiente manera:

```
# Primera etapa - container para build  
FROM gradle:5.3-jdk8 as builder  
  
WORKDIR /home/gradle  
  
ADD . /home/gradle  
  
RUN gradle assemble --no-build-cache --no-daemon  
  
# Segunda etapa - container de aplicación  
FROM openjdk:8-jre-alpine  
  
RUN mkdir /app  
  
COPY --from=builder /home/gradle/build/libs/*all.jar \  
    /app/application.jar  
  
EXPOSE 8080  
  
CMD ["java", "-jar", "-XX:+UseContainerSupport", \  
    "-Djava.security.egd=file:/dev/./urandom", \  
    "/app/application.jar"]
```

Guardar y cerrar (CTRL-O y luego CTRL-X)

Explique que hacen los comandos de este Dockerfile.

En el mismo directorio donde está el archivo Dockerfile ejecutar:

```
docker build -t backend:multi-stage .
```

Ver la imagen creada con:

```
docker images -f reference='backend:*
```

Compare el tamaño con de la nueva imagen con anterior con una sola etapa.

Probamos el container con:

```
docker run --rm -p 8080:8080 -h localhost backend:multi-stage
```

Podemos probar el container abriendo otra consola y ejecutando en la misma los siguientes comandos:

```
curl http://localhost:8080/hola/Motoko
```

Proyecto frontend

Modificar el Dockerfile del proyecto frontend:

```
cd ~/docker-sample-frontend-react  
nano Dockerfile
```

El Dockefile deberá quedar de la siguiente manera:

```
# Primera etapa - container para build  
FROM node:8.10.0 as builder  
  
WORKDIR /app  
  
ENV PATH /app/node_modules/.bin:$PATH  
  
# Instalo las dependencias  
COPY package.json ./  
RUN npm install react-scripts@2.1.5 -g --silent  
COPY yarn.lock ./  
RUN yarn install --silent  
  
ADD . /app  
  
RUN yarn run build  
  
# Segunda etapa - container de aplicación  
FROM nginx:stable-alpine  
  
COPY --from=builder /app/build /usr/share/nginx/html  
  
EXPOSE 80  
  
CMD ["nginx", "-g", "daemon off;"]
```

Guardar y cerrar (CTRL-O y luego CTRL-X)

Explique que hacen los comandos de este Dockerfile.

En el mismo directorio donde está el archivo Dockerfile ejecutar:

```
docker build -t frontend:multi-stage .
```

Ver la imagen creada con:

```
docker images -f reference='frontend:*
```

Compare el tamaño con de la nueva imagen con anterior con una sola etapa.

Probamos el container con:

```
docker run --rm -p 80:80 -h localhost frontend:multi-stage
```

Una vez inicie el container, abrir un navegador e ingresar a: <http://localhost:80>

Subir imagenes a Docker Hub

Crearse un usuario en [Docker Hub](#).

Identificarse con sus credenciales de Docker Hub mediante el comando:

docker login

Ingresar al directorio del backend

cd ~/docker-sample-backend-micronaut

Agrego el usuario y el tag "latest" a la imagen actual

docker build -t ticsfrcu/backend:latest .

(reemplace **ticsfrcu** por su usuario de Docker Hub).

Subo la imagen al hub con el comando:

docker push ticsfrcu/backend:latest

(reemplace **ticsfrcu** por su usuario de Docker Hub).

Ingresar al directorio del frontend

cd ~/docker-sample-frontend-react

Agrego el usuario y el tag "latest" a la imagen actual

docker build -t ticsfrcu/frontend:latest .

(reemplace **ticsfrcu** por su usuario de Docker Hub).

Subo la imagen al hub con el comando:

docker push ticsfrcu/frontend:latest

(reemplace **ticsfrcu** por su usuario de Docker Hub).

Las imágenes se pueden ahora usar desde cualquier lugar y como base de otros containers.

Compare las versiones de etapa simple y dos etapas de cada container. Describa las mejoras que pudieran haberse incorporado y si es posible sugiera nuevas.

Cree un archivo docker-compose.yml para el stack frontend/backend.