

Trabajo Práctico 6

Materia: Ingeniería de Software 3

Alumno: Vietto Santiago

Docente: Fernando Bono

Institución: UCC

Año: 2022

Ejercicio 1:

FROM: esta instrucción inicializa una nueva etapa de construcción y establece la imagen base para las instrucciones posteriores. Como tal, un Dockerfile válido debe comenzar con una instrucción FROM. La imagen puede ser cualquier imagen válida; es especialmente fácil comenzar extrayendo una imagen de los repositorios públicos.

RUN: esta instrucción va a ejecutar cualquier comando en una nueva capa encima de la imagen actual y confirmará los resultados. La imagen confirmada resultante se usará para el siguiente paso en el Dockerfile.

ADD: esta instrucción copia nuevos archivos, directorios o direcciones URL de archivos remotos desde <src> y los agrega al sistema de archivos de la imagen en la ruta <dest>.

COPY: esta instrucción copia nuevos archivos o directorios desde <src> y los agrega al sistema de archivos del contenedor en la ruta <dest>.

EXPOSE: esta instrucción informa a Docker que el contenedor escucha en los puertos de red especificados en tiempo de ejecución. Puede especificar si el puerto escucha en TCP o UDP, y el valor predeterminado es TCP si no se especifica el protocolo. La instrucción EXPOSE en realidad no publica el puerto. Funciona como un tipo de documentación entre la persona que construye la imagen y la persona que ejecuta el contenedor, sobre qué puertos se pretende publicar.

CMD: solo puede haber una instrucción CMD en un Dockerfile. Si incluye más de un CMD, sólo tendrá efecto el último CMD. El objetivo principal de esta instrucción es proporcionar valores predeterminados para un contenedor en ejecución. Estos valores predeterminados pueden incluir un ejecutable o pueden omitir el ejecutable, en cuyo caso también debe especificar una instrucción ENTRYPOINT.

ENTRYPOINT: esta instrucción permite configurar un contenedor que correrá como un ejecutable. Solo tendrá efecto la última instrucción ENTRYPOINT en el Dockerfile.

Ejercicio 2:

1)_ Clonamos el repositorio y compilamos:

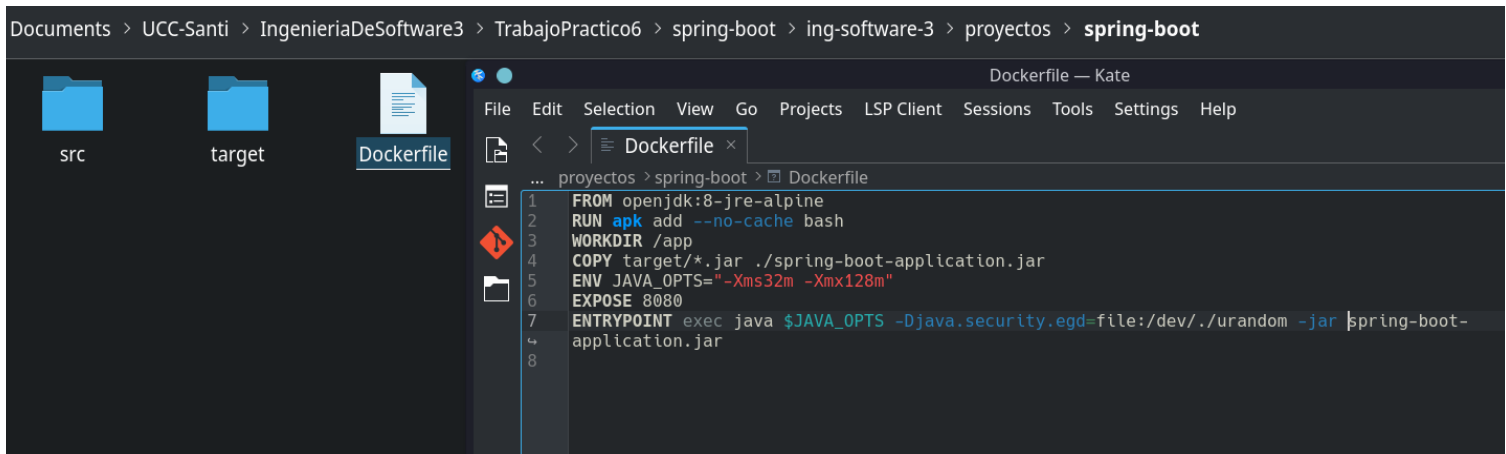
```
~/Documents/UCC-Santi/IngenieriaDeSoftware3/TrabajoPractico6/spring-boot git config --global user.name "santi2019"
~/Documents/UCC-Santi/IngenieriaDeSoftware3/TrabajoPractico6/spring-boot git config --global user.email "santiagovietto5@hotmail.com"
~/Documents/UCC-Santi/IngenieriaDeSoftware3/TrabajoPractico6/spring-boot git clone https://github.com/fernandobono/ing-software-3.git
Cloning into 'ing-software-3'...
remote: Enumerating objects: 231, done.
remote: Counting objects: 100% (231/231), done.
remote: Compressing objects: 100% (183/183), done.
remote: Total 231 (delta 111), reused 116 (delta 38), pack-reused 0
Receiving objects: 100% (231/231), 1.69 MiB | 2.18 MiB/s, done.
Resolving deltas: 100% (111/111), done.
~/Documents/UCC-Santi/IngenieriaDeSoftware3/TrabajoPractico6/spring-boot cd proyectos
cd: no such file or directory: proyectos
~/Documents/UCC-Santi/IngenieriaDeSoftware3/TrabajoPractico6/spring-boot cd ing-software-3
~/Doc/U/I/TrabajoPractico6/spring-boot/ing-software-3 P master cd proyectos
~/Doc/U/I/TrabajoPractico6/spring-boot/ing-software-3/proyectos P master cd spring-boot
~/Doc/U/I/TrabajoPractico6/s/ing-software-3/proyectos/spring-boot P master mvn clean package spring-boot:repackage
```

En mi caso tiro un error en la compilación, pero lo solucioné añadiendo la información de este sitio en el archivo POM:

- <https://facingissuesonit.com/2021/05/08/maven-error-failed-to-execute-goal-org-apache-maven-plugins-maven-surefire-plugin-2-12-4-test-default-test-on-project-maven-junit-code-coverage-jacoco-there-are-test-failures/>

```
[INFO]
[INFO] --- spring-boot-maven-plugin:2.0.2.RELEASE:repackage (default-cli) @ spring-boot-sample-actuator ---
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 01:50 min
[INFO] Finished at: 2022-09-29T09:42:27-03:00
[INFO]
```

2)_ Modificamos la primera línea del archivo Dockerfile en la parte de FROM de java:8-jre-alpine a openjdk:8-jre-alpine, ya que salta error.



3)_ Generamos la imagen de docker:

```
~/Doc/U/I/TrabajoPractico6/s/ing-software-3/p/spring-boot master !1 ?1 sudo docker build -t test-spring-boot .
Sending build context to Docker daemon 22.4MB
Step 1/7 : FROM openjdk:8-jre-alpine
--> f7a292bbb70c
Step 2/7 : RUN apk add --no-cache bash
--> Using cache
--> da3e9e0d4362
Step 3/7 : WORKDIR /app
--> Using cache
--> 5fad4bb1739e
Step 4/7 : COPY target/*.jar ./spring-boot-application.jar
--> Using cache
--> a0fbb9362c93
Step 5/7 : ENV JAVA_OPTS=\"-Xms32m -Xmx128m\"
--> Using cache
--> 52fef0d96ba7
Step 6/7 : EXPOSE 8080
--> Using cache
--> 0607671d1fb7
Step 7/7 : ENTRYPOINT exec java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar spring-boot-application.jar
--> Running in 63b2d7a7b9e5
Removing intermediate container 63b2d7a7b9e5
--> 8fd8de2a3c0c
Successfully built 8fd8de2a3c0c
Successfully tagged test-spring-boot:latest
```

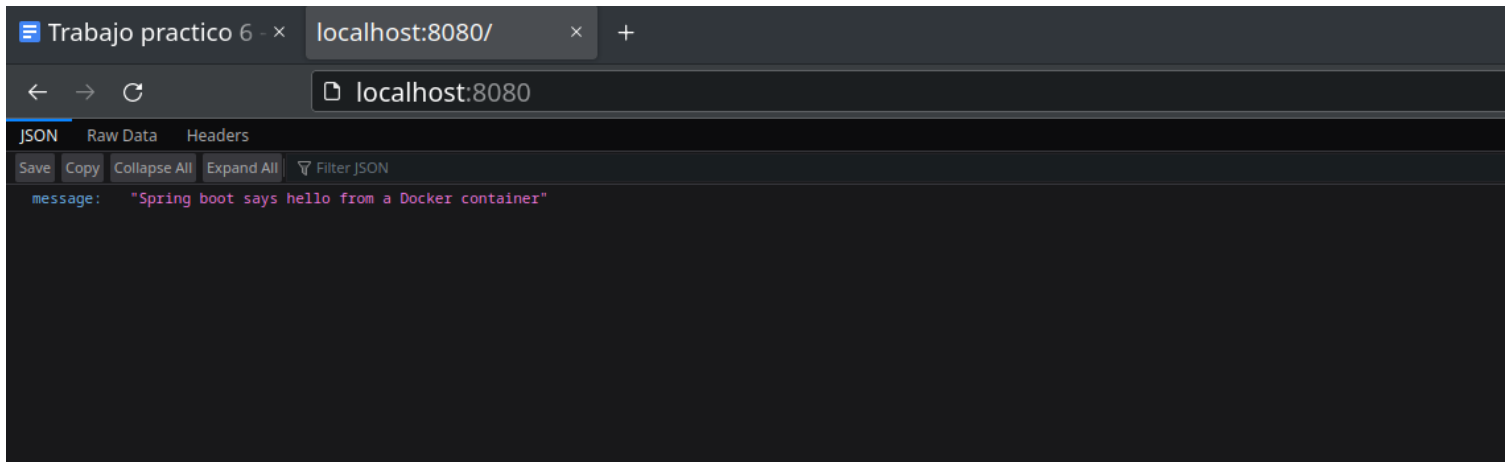
4)_ Ejecutamos el contenedor:

```
~/Doc/U/I/TrabajoPractico6/s/ing-software-3/p/spring-boot master !1 ?1 sudo docker run -p 8080:8080 test-spring-boot

:: Spring Boot :: (v2.0.2.RELEASE)

2022-09-29 14:08:07.168 INFO 1 --- [main] s.actuator.SampleActuatorApplication : Starting SampleActuatorApplication v2.0.2 on 580baf9c2671 with PID 1 (/app/spring-boot-application.jar started by root in /app)
2022-09-29 14:08:07.171 INFO 1 --- [main] s.actuator.SampleActuatorApplication : No active profile set, falling back to default profiles: default
2022-09-29 14:08:07.212 INFO 1 --- [main] ConfigServletWebServerApplicationContext : Refreshing org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplication
2022-09-29 14:08:08.225 INFO 1 --- [main] trationDelegate$BeanPostProcessorChecker : Bean 'org.springframework.transaction.annotation.ProxyTransactionManagementConfiguration' of type [org.springframework.transaction.annotation.ProxyTransactionManagementConfiguration$EnhancerBySpringGLIB$$257b5d18] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
```

Verificamos en la URL con <http://localhost:8080/>

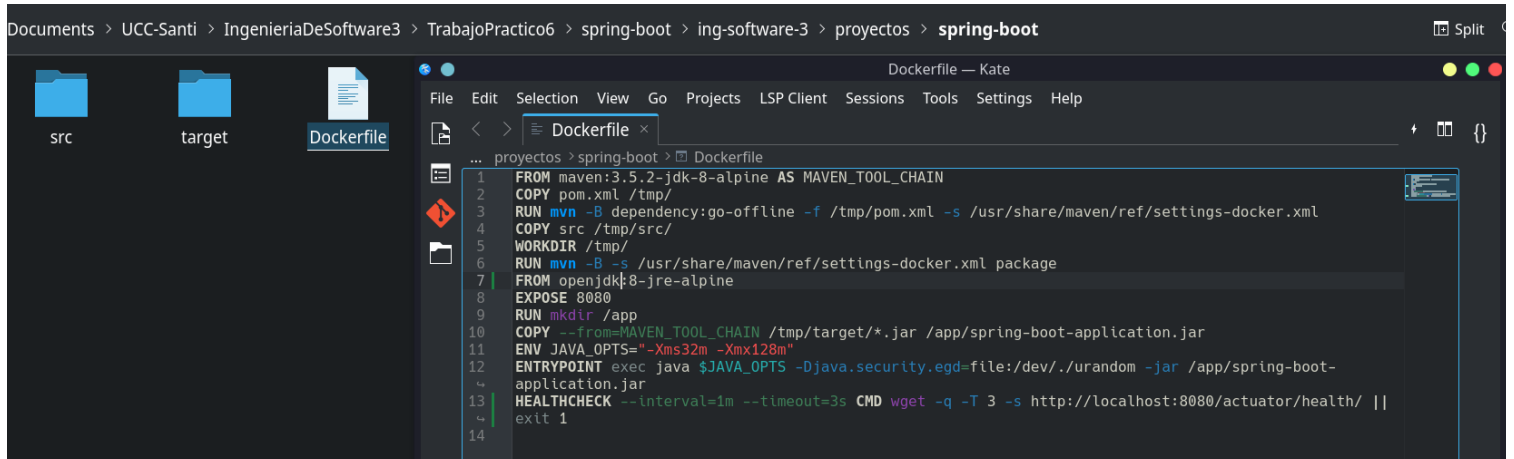


5)_ En otra pestaña de la terminal ejecutamos:

```
~/Doc/U/I/TrabajoPractico6/spring-boot/ing-software-3/proyectos/spring-boot master !1 ?1 curl -v localhost:8080
* Trying 127.0.0.1:8080...
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET / HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.85.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Content-Type: application/json;charset=UTF-8
< Transfer-Encoding: chunked
< Date: Thu, 29 Sep 2022 14:15:50 GMT
<
* Connection #0 to host localhost left intact
{"message":"Spring boot says hello from a Docker container"}
}
```

Ejercicio 3:

1)_ Modificamos el archivo Dockerfile:

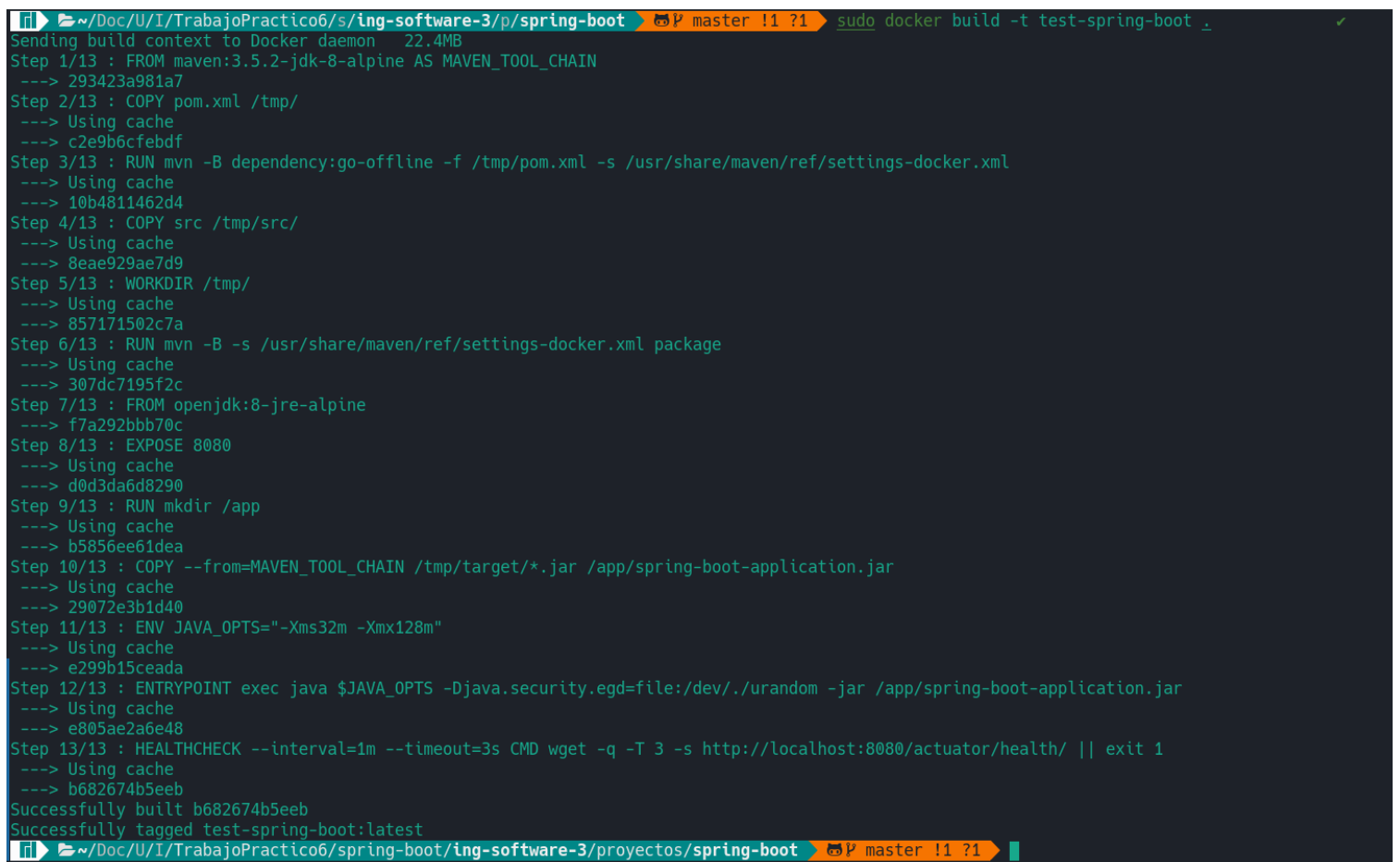


The screenshot shows a code editor with the following Dockerfile content:

```
1 FROM maven:3.5.2-jdk-8-alpine AS MAVEN_TOOL_CHAIN
2 COPY pom.xml /tmp/
3 RUN mvn -B dependency:go-offline -f /tmp/pom.xml -s /usr/share/maven/ref/settings-docker.xml
4 COPY src /tmp/src/
5 WORKDIR /tmp/
6 RUN mvn -B -s /usr/share/maven/ref/settings-docker.xml package
7 FROM openjdk:8-jre-alpine
8 EXPOSE 8080
9 RUN mkdir /app
10 COPY --from=MAVEN_TOOL_CHAIN /tmp/target/*.jar /app/spring-boot-application.jar
11 ENV JAVA_OPTS="-Xms32m -Xmx128m"
12 ENTRYPOINT exec java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar /app/spring-boot-
application.jar
13 HEALTHCHECK --interval=1m --timeout=3s CMD wget -q -T 3 -s http://localhost:8080/actuator/health/ ||
exit 1
14
```

Además, en el FROM cambiamos java por openjdk.

2)_ Construimos nuevamente la imagen:



The screenshot shows a terminal window with the following output for the command `sudo docker build -t test-spring-boot .`:

```
Sending build context to Docker daemon 22.4MB
Step 1/13 : FROM maven:3.5.2-jdk-8-alpine AS MAVEN_TOOL_CHAIN
--> 293423a981a7
Step 2/13 : COPY pom.xml /tmp/
--> Using cache
--> c2e9b6cfebfdf
Step 3/13 : RUN mvn -B dependency:go-offline -f /tmp/pom.xml -s /usr/share/maven/ref/settings-docker.xml
--> Using cache
--> 10b4811462d4
Step 4/13 : COPY src /tmp/src/
--> Using cache
--> 8eae929ae7d9
Step 5/13 : WORKDIR /tmp/
--> Using cache
--> 85717150c7a
Step 6/13 : RUN mvn -B -s /usr/share/maven/ref/settings-docker.xml package
--> Using cache
--> 307dc7195f2c
Step 7/13 : FROM openjdk:8-jre-alpine
--> f7a292bbb70c
Step 8/13 : EXPOSE 8080
--> Using cache
--> d0d3da6d8290
Step 9/13 : RUN mkdir /app
--> Using cache
--> b5856ee61dea
Step 10/13 : COPY --from=MAVEN_TOOL_CHAIN /tmp/target/*.jar /app/spring-boot-application.jar
--> Using cache
--> 29072e3b1d40
Step 11/13 : ENV JAVA_OPTS="-Xms32m -Xmx128m"
--> Using cache
--> e299b15ceada
Step 12/13 : ENTRYPOINT exec java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar /app/spring-boot-application.jar
--> Using cache
--> e805ae2a6e48
Step 13/13 : HEALTHCHECK --interval=1m --timeout=3s CMD wget -q -T 3 -s http://localhost:8080/actuator/health/ || exit 1
--> Using cache
--> b682674b5eeb
Successfully built b682674b5eeb
Successfully tagged test-spring-boot:latest
```

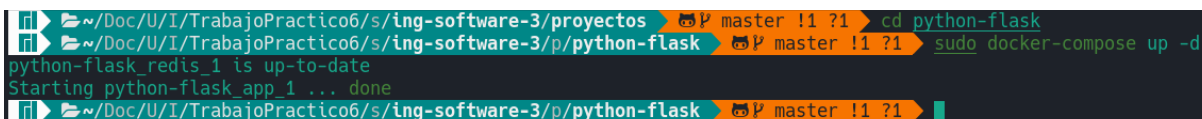
Luego de realizar los pasos anteriores, analizamos detalladamente el nuevo contenido del archivo Dockerfile. Por un lado vemos la creación de una imagen de maven, en donde con el FROM se trae la imagen de maven, en COPY, va a copiar el archivo pom.xml a la carpeta /tmp, el primer RUN ejecuta todas las dependencias, luego con COPY va a copiar el archivo src a la carpeta /tmp/src, WORKDIR especifica el directorio de trabajo del contenedor que en este caso es /tmp, y el segundo RUN ejecuta el comando mvn package para compilar el proyecto.

Como segunda parte tenemos la creación de la imagen de java, en donde con el FROM se trae la imagen de java, EXPOSE nos dice que se expone en el puerto 8080, con RUN creamos y corremos un subdirectorio /app, con COPY copiamos de la imagen de maven anterior los ejecutables especificados, ENV setea la variable de entorno que se observa, ENTRYPOINT indica que cada vez que corra el contenedor se ejecute el archivo jar especificado, y HEALTHCHECK realiza chequeos en el contenedor para saber si la app es saludable.

Entonces, a diferencia del archivo Dockerfile anterior, acá tenemos una primera parte en la que partimos de la imagen de maven donde se compila el código y genera los archivos .jar necesarios para poder ser ejecutado. Luego en una segunda parte, partimos de la imagen de java. Esto reduce el tamaño de la imagen considerablemente, copiando de la etapa anterior los archivos .jar generados que son necesarios para ejecutar la aplicación y genera una imagen similar a la del ejercicio 2, en donde al final nos permite crear un contenedor con la app spring-boot en funcionamiento.

Ejercicio 4:

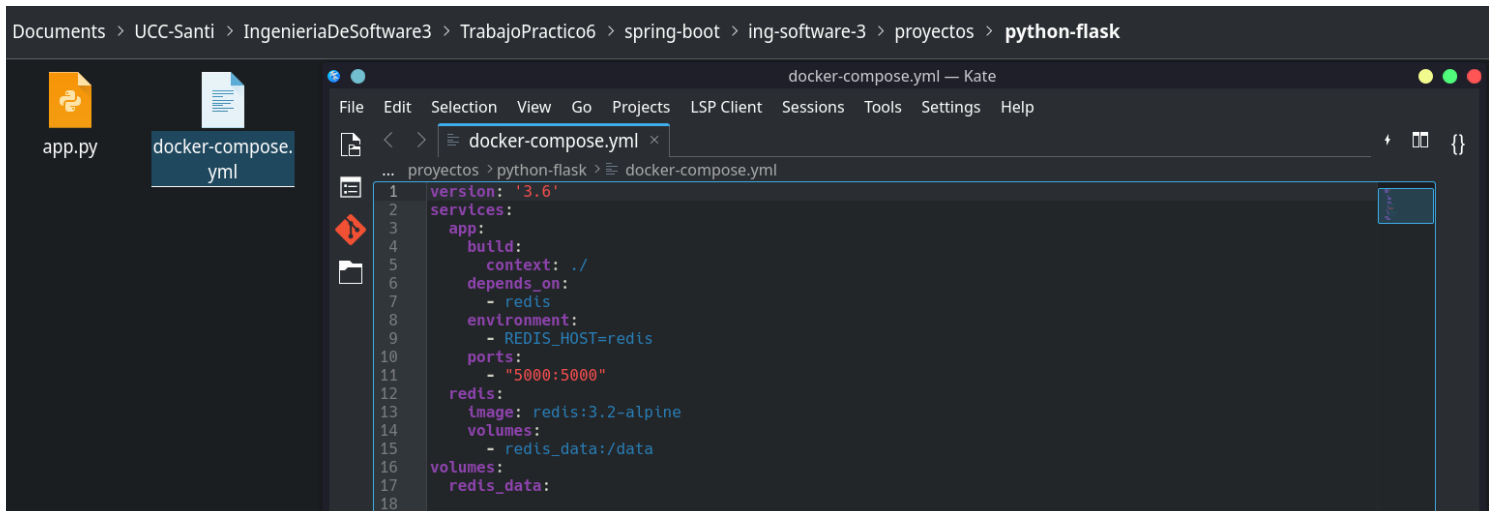
1)_



```
~/Doc/U/I/TrabajoPractico6/s/ing-software-3/proyectos master !1 ?1 cd python-flask
~/Doc/U/I/TrabajoPractico6/s/ing-software-3/p/python-flask master !1 ?1 sudo docker-compose up -d
python-flask_redis_1 is up-to-date
Starting python-flask_app_1 ... done
~/Doc/U/I/TrabajoPractico6/s/ing-software-3/p/python-flask master !1 ?1
```

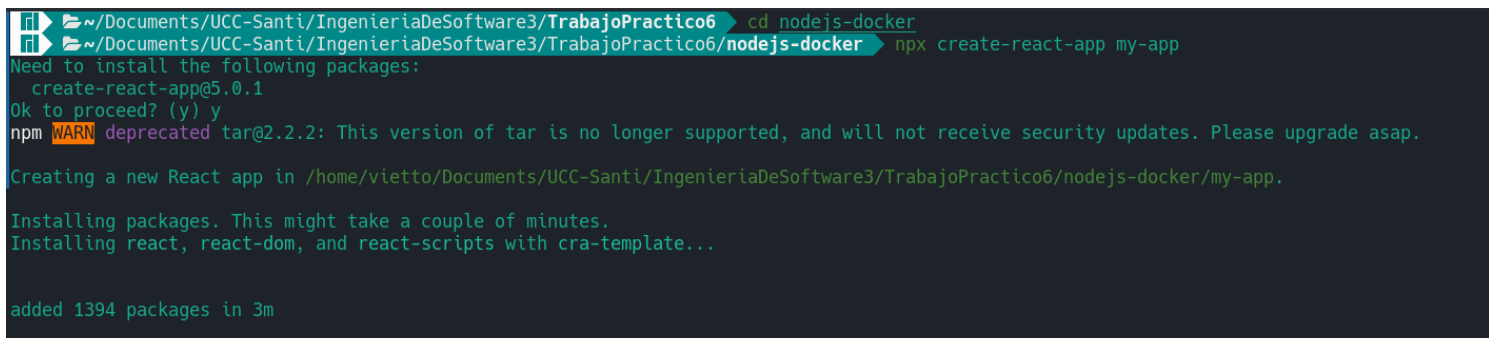
Al ejecutar este comando lo que sucedió es que se crearon dos contenedores, uno llamado “app” y el otro “redis” como podemos observar en el archivo docker-compose.yml. En donde se descargaron y se iniciaron los servicios de estas imágenes.

2)_ En el archivo docker-compose.yml, el build se utiliza para definir las opciones de configuración que serán aplicadas al momento de crear el contenedor. El context se refiere a un set de archivos que se encuentran en un PATH o URL, por lo tanto, el build utilizará los archivos de ese contexto para saber donde debe conseguir las imágenes y este directorio será también el contexto de creación enviado al daemon de docker. Entonces, en este caso el contenedor “app” se creó a partir de una imagen a su vez creada por el archivo Dockerfile que se encuentra en la raíz del directorio del proyecto, por ende docker-compose permite esto gracias a que se especifica la ubicación del archivo Dockerfile mediante la key build.context.

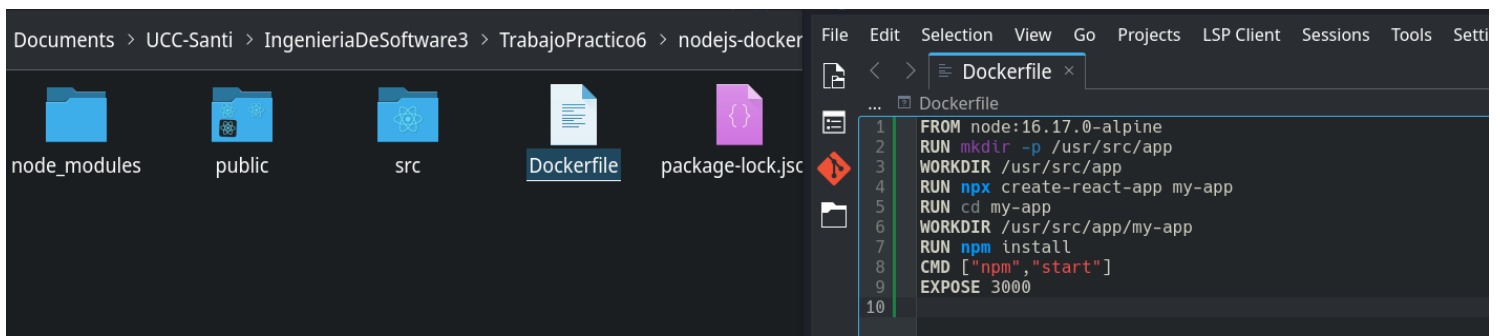


Ejercicio 5:

1)_ Generamos el proyecto nodejs:



2)_ Creamos el archivo Dockerfile dentro del proyecto:



3)_ Construimos la imagen ejecutando:

- `docker build -t test-node .`

```
~/Doc/U/I/TrabajoPractico6/nodejs-docker/my-app git master ?1 sudo docker build -t test-node .
Sending build context to Docker daemon 261.3MB
Step 1/9 : FROM node:16.17.0-alpine
16.17.0-alpine: Pulling from library/node
213ec9aee27d: Already exists
864b973d1bf1: Pull complete
80fe61ad56f5: Pull complete
e3887ab559e6: Pull complete
Digest: sha256:2c405ed42fc0fd6aacbe5730042640450e5ec030bada7617beac88f742b6997b
Status: Downloaded newer image for node:16.17.0-alpine
--> 5dcd1f6157bd
Step 2/9 : RUN mkdir -p /usr/src/app
--> Running in 489a3a4b3b57
Removing intermediate container 489a3a4b3b57
--> b751a8fe179a
Step 3/9 : WORKDIR /usr/src/app
--> Running in 2d2b6f8afea8
Removing intermediate container 2d2b6f8afea8
--> f5d8aa33e764
Step 4/9 : RUN npx create-react-app my-app
```

4)_ Ejecutamos la imagen mediante:

- `docker run -p 3000:3000 test-node`

```
~/Doc/U/I/TrabajoPractico6/nodejs-docker/my-app git master ?1 sudo docker run -p 3000:3000 test-node
[sudo] password for vietto:
> my-app@0.1.0 start
> react-scripts start

(node:26) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
(Use 'node --trace-deprecation ...' to show where the warning was created)
(node:26) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
Starting the development server...

Compiled successfully!

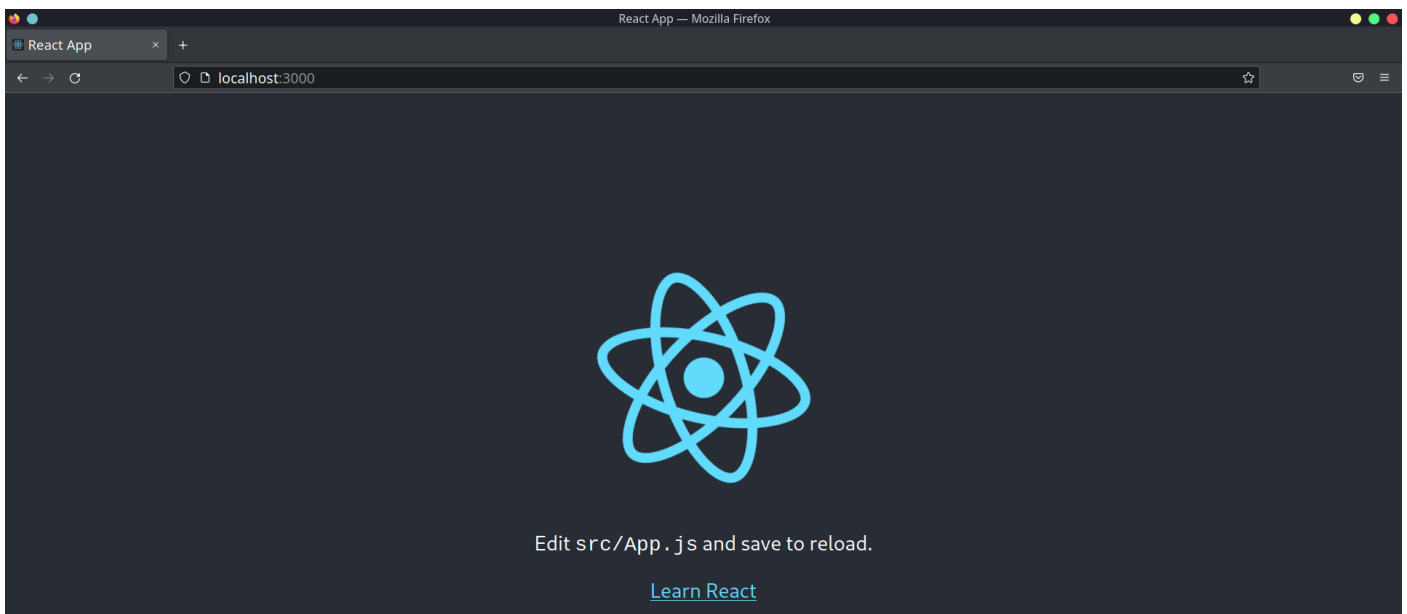
You can now view my-app in the browser.

   Local:            http://localhost:3000
   On Your Network:  http://172.17.0.2:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
Compiling...
Compiled successfully!
webpack compiled successfully
```

Verificamos en la URL con <http://localhost:3000/>



Ejercicio 6:

1)_ Como ya estoy registrado en docker-hub, procedo a loguearme:

```
❯ sudo docker login
[sudo] password for vietto:
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: 1802890
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```


```
❯ sudo docker tag test-node 1802890/test-node:latest
❯ sudo docker push 1802890/test-node:latest
The push refers to repository [docker.io/1802890/test-node]
91349a87be7c: Pushed
b71a2571e507: Pushed
02439544cda5: Pushed
f1ed0bba6314: Mounted from library/node
2808ff9120f2: Mounted from library/node
cb6eda6d73f0: Mounted from library/node
994393dc58e7: Mounted from library/node
latest: digest: sha256:54223e10e219e57058f2bf8f1f1b5ebf2950061f8bd77bbeac6ec758e4cbac9b size: 1788
```

2)_ Verificamos en la cuenta que la imagen se subió correctamente:

Browser address bar: <https://hub.docker.com/u/1802890>


Please check your inbox to verify the email associated with this account. You won't be able to create a repository or configure your Docker Hub without verifying your email address.

dockerhub Search for great content (e.g., mysql) Explore Repositories Organizations Help Upgrade 1802890

 **1802890** [Edit profile](#)
Community User Joined August 18, 2021

Repositories Starred Contributed

Displaying 1 of 1 repository

**1802890/test-node** 0 Stars
By [1802890](#) · Updated a few seconds ago
Image