DigitalHouse>

Certified Tech Developer
The Ultimate Degree

Infraestructura II

Inmutabilidad en Docker

En este documento vamos a demostrar el principio de inmutabilidad aplicado a contenedores.

Requisitos

Para poder ejecutar este ejemplo vamos a necesitar una aplicación dockerizada para ejecutar localmente. Vamos a utilizar tres archivos:

- index.js, que será la aplicación en NodeJS que modificaremos para obtener varias versiones.

```
'use strict';



const express = require('express');

// Constantes

const PORT = 8080;

const HOST = '0.0.0.0';



// App

const app = express();
```

```
app.get('/', (req, res) => {

 res.send('Hello World');

});



app.listen(PORT, HOST);

console.log(`Running on http://${HOST}:${PORT}`);
```

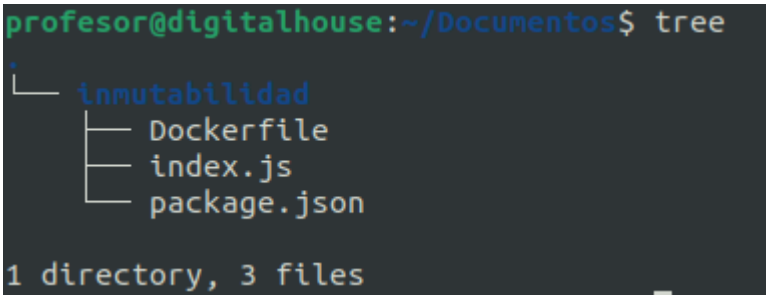
- package.json con el listado de las dependencias y descripción de nuestra aplicación.

```
{

   "name": "mi_app_nodejs",

   "version": "1.0.0",

   "description": "Node.js en Docker",

   "author": "Digital House <digitalhouse@example.com>",

   "main": "index.js",

   "scripts": {

     "start": "node index.js"

   },

   "dependencies": {

     "express": "^4.16.1"

   }

 }
```

- Dockerfile donde estará la definición del contenedor con los requerimientos mínimos para ejecutar nuestra aplicación.

```
FROM node:14


# Creamos directorio para la aplicación

WORKDIR /usr/src/app


# Instalamos dependencias

COPY package*.json ./


RUN npm install


# Copiamos los sources de la aplicación

COPY . .


# Ejecutamos la aplicación

EXPOSE 8080

CMD [ "node", "index.js" ]
```

La estructura de la carpeta nos quedará así:



Compilar un Dockerfile

Al compilar una aplicación con Docker, vamos a poder realizar esta acción las veces que necesitemos publicar un cambio de la aplicación misma, pero si siempre lo hacemos con el comando estándar:

```
docker build . -t node-app
```

Esto va a generar una imagen de nuestra aplicación, pero etiquediéndola con “latest” al final.

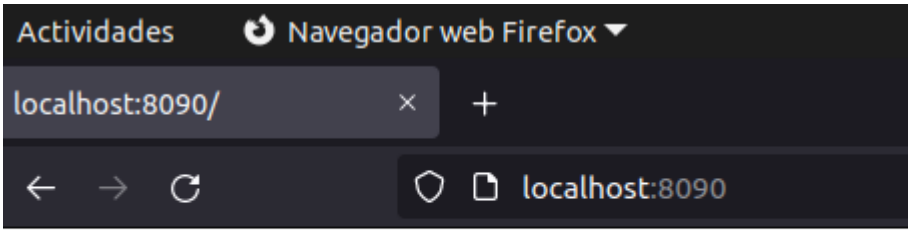
```
profesor@digitalhouse:~/Documentos/inmutabilidad$ docker build . -t node-app
Sending build context to Docker daemon  4.096kB
Step 1/7 : FROM node:14
14: Pulling from library/node
08224db8ce18: Pull complete
abd3caf86f5b: Pull complete
71c316554a55: Pull complete
721081de66bf: Pull complete
239fb482263d: Pull complete
26d24e5f0efd: Pull complete
4a43fffd53dd: Pull complete
4e10c266ec1a: Pull complete
6c4e1d6ce241: Pull complete
Digest: sha256:adbbb61dab70ea6e5a6c2ad7fba60e4d1047ba98ad1afcd631c15553163b22b7
Status: Downloaded newer image for node:14
 ---> e0ab58ea4a4f
Step 2/7 : WORKDIR /usr/src/app
 ---> Running in 1ae36b1510dd
Removing intermediate container 1ae36b1510dd
 ---> d2dd64b633ca
Step 3/7 : COPY package*.json ./
 ---> 860d2c27fcd7
Step 4/7 : RUN npm install
 ---> Running in 619bb9684970
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN mi_app_nodejs@1.0.0 No repository field.
npm WARN mi_app_nodejs@1.0.0 No license field.

added 50 packages from 37 contributors and audited 50 packages in 3.397s
found 0 vulnerabilities

Removing intermediate container 619bb9684970
 ---> e8fa71f3748f
Step 5/7 : COPY . .
 ---> 6c15f23299c5
Step 6/7 : EXPOSE 8080
 ---> Running in 8731e92a3eac
Removing intermediate container 8731e92a3eac
 ---> ffcef5db0417
Step 7/7 : CMD [ "node", "index.js" ]
 ---> Running in 407cb1b091cb
Removing intermediate container 407cb1b091cb
 ---> 1a474395eebd
Successfully built 1a474395eebd
Successfully tagged node-app:latest
profesor@digitalhouse:~/Documentos/inmutabilidad$
```

Notemos el “TAG” latest, que se aplica de forma automática si no hacemos explícita la etiqueta de la versión de nuestra aplicación.

```
profesor@digitalhouse:~/Documentos/inmutabilidad$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
node-app            latest              1a474395eebd        23 seconds ago      947MB
```

Ejecutamos nuestra aplicación en el puerto 8090 de nuestra computadora:

```
profesor@digitalhouse:~/Documentos/inmutabilidad$ docker run -p 8090:8080 -d node-app
f3e7da7fe3a4566524d2f6f38e57bf554b80afd24b618610c3fb84d62d7827b9
```



Hello World

Pero, ¿qué sucede si quiero modificar ese mensaje?. Solo modificamos nuestro código dentro de index.js y compilamos nuevamente nuestra imagen de Docker. ¡Vamos por ello!

Modificamos el mensaje por “Hola Digital House!”

index.js

```
'use strict';

const express = require('express');

// Constantes
const PORT = 8080;
const HOST = '0.0.0.0';

// App
const app = express();
app.get('/', (req, res) => {
  res.send('Hola Digital House!');
});

app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);
```

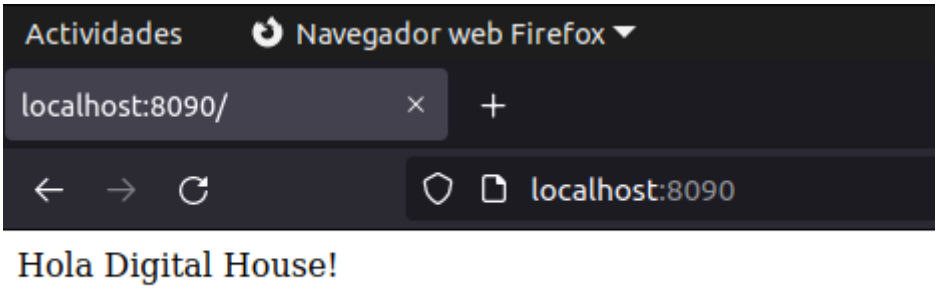
Compilamos nuestro dockerfile para obtener la nueva versión.

```
profesor@digitalhouse:~/Documentos/inmutabilidad$ docker build . -t node-app
Sending build context to Docker daemon  4.096kB
Step 1/7 : FROM node:14
 ---> e0ab58ea4a4f
Step 2/7 : WORKDIR /usr/src/app
 ---> Using cache
 ---> d2dd64b633ca
Step 3/7 : COPY package*.json ./
 ---> Using cache
 ---> 860d2c27fcd7
Step 4/7 : RUN npm install
 ---> Using cache
 ---> e8fa71f3748f
Step 5/7 : COPY . .
 ---> 21340fe8e8cf
Step 6/7 : EXPOSE 8080
 ---> Running in a67baed6add4
Removing intermediate container a67baed6add4
 ---> cc67093c1b4d
Step 7/7 : CMD [ "node", "index.js" ]
 ---> Running in d22648d02013
Removing intermediate container d22648d02013
 ---> dfefbaa1e324
Successfully built dfefbaa1e324
Successfully tagged node-app:latest
```

Lo negativo de lo que acabamos de hacer es que reemplazamos nuestra imagen anterior quitándole la etiqueta. La primera versión de nuestra aplicación está desetiquetada y solo vemos su ID, pero no es fácilmente identificable en un proceso de una aplicación escalable.

```
profesor@digitalhouse:~/Documentos/inmutabilidad$ docker images
REPOSITORY          TAG         IMAGE ID       CREATED              SIZE
node-app            latest      dfefbaa1e324   About a minute ago   947MB
<none>              <none>      1a474395eebd   14 minutes ago       947MB
```

Aunque funciona igual:



Buenas prácticas

Para realizar esta tarea aplicamos buenas prácticas. Es aquí donde implementamos el **principio de inmutabilidad** al respetar ese atributo de cada imagen de Docker en donde es único.

Notemos que la imagen anterior no se borró, aunque sí perdió su nombre y etiqueta, pero el ID sigue pudiendo recuperarse. Esto es porque Docker en sí es inmutable.

Para aprovechar esas bondades de la plataforma tenemos que realizar cada cambio de nuestra aplicación con etiquetas que representen la versión de nuestra aplicación. Para esto, vamos a realizar lo siguiente:

- V1: es nuestra versión con “Hello World”.
- V2: es la nueva versión mostrando “Hola Digital House!”.

Veamos cómo funciona:

Realizamos la compilación agregando la etiqueta V1 en nuestra primera versión:

```
docker build . -t node-app:V1
```

```
profesor@digitalhouse:~/Documentos/inmutabilidad$ docker build . -t node-app:V1
Sending build context to Docker daemon  4.096kB
Step 1/7 : FROM node:14
 ---> e0ab58ea4a4f
Step 2/7 : WORKDIR /usr/src/app
 ---> Using cache
 ---> d2dd64b633ca
Step 3/7 : COPY package*.json ./
 ---> Using cache
 ---> 860d2c27fcd7
Step 4/7 : RUN npm install
 ---> Using cache
 ---> e8fa71f3748f
Step 5/7 : COPY . .
 ---> Using cache
 ---> 6c15f23299c5
Step 6/7 : EXPOSE 8080
 ---> Using cache
 ---> ffcef5db0417
Step 7/7 : CMD [ "node", "index.js" ]
 ---> Using cache
 ---> 1a474395eebd
Successfully built 1a474395eebd
Successfully tagged node-app:V1
```

Hacemos lo mismo con nuestra versión V2:

```
docker build . -t node-app:V2
```

Nuestras imágenes van a quedar compiladas así:

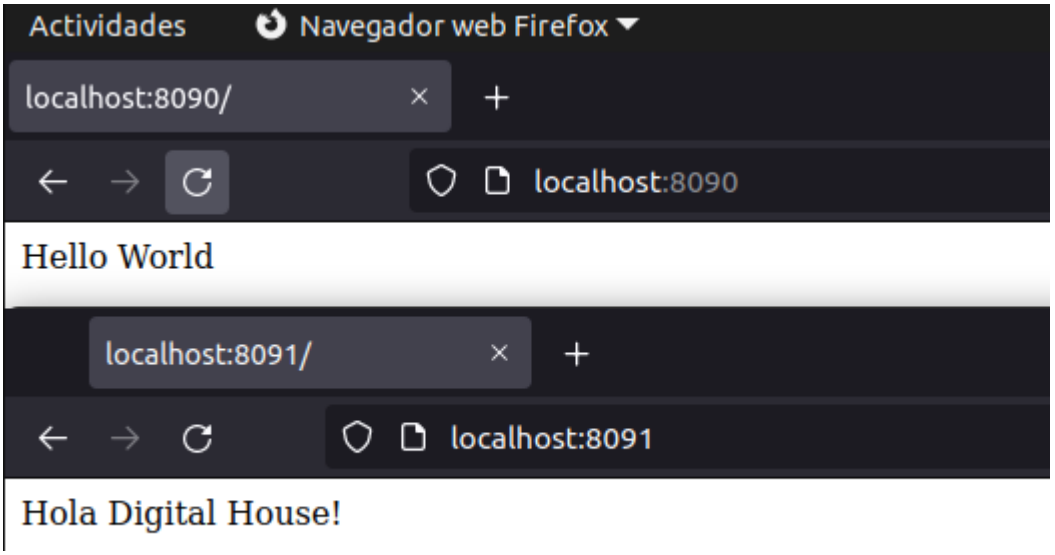
```
profesor@digitalhouse:~/Documentos/inmutabilidad$ docker images
REPOSITORY          TAG         IMAGE ID       CREATED          SIZE
node-app            V2          dfefbaa1e324   14 minutes ago   947MB
node-app            V1          1a474395eebd   26 minutes ago   947MB
```

Incluso, para realizar la prueba, vamos a poder ejecutar las dos versiones en paralelo. Una en el puerto 8090 y la segunda en el puerto 8091, a través de los siguientes comandos:

```
docker run -p 8090:8080 -d node-app:V1

docker run -p 8091:8080 -d node-app:V2
```

```
profesor@digitalhouse:~/Documentos/inmutabilidad$ docker run -p 8090:8080 -d node-app:V1
8c5b05ac946dd0f50bb20ee8b3e445b36c60e1c6dd840c50629aaeff10601cf0
profesor@digitalhouse:~/Documentos/inmutabilidad$ docker run -p 8091:8080 -d node-app:V2
edda5f42cca9b33cb1fdf25af5cb18f77c294298af9de840d6f4d4b7a1a0a2a3
profesor@digitalhouse:~/Documentos/inmutabilidad$
```

En el navegador lo comprobamos accediendo a cada ruta al mismo tiempo:



Conclusión

Los artefactos son productos de la compilación de una versión específica de una aplicación. Al aplicar una metodología DevOps en nuestros equipos de trabajo, el principio de inmutabilidad es un factor importante.

¿Por qué nos sirve tener presente este principio? Porque nos permite -al momento de escribir nuestros propios pipelines- mantener la trazabilidad de las versiones de cada aplicación para:

- ejecutar pruebas en ambientes controlados,
- poder implementar la versión que necesitemos,
- realizar vueltas atrás por errores detectados (rollback),

Y, en definitiva, ¡contar con varias versiones en paralelo porque así lo requiere el cliente final!