Práctica Tema 4

Despliegue de la práctica 3 en el VPS contratado

Se debe subir la aplicación realizada en la práctica 3 a un repositorio, como GitHub o similar, **sin incluir** la carpeta **node_modules**, para a posteriori descargarla en nuestro VPS.

Se recomienda la contratación de OVH con Debian 10 ya que viene Docker instalado y por tanto nos facilitará la tarea. Ver.

Pasos a realizar

¡Advertencia!: Importante ir guandando los pasos de cada punto para la documentación.

1. Accede vía **ssh** al VPS y pasa a instalar docker-compose:

```
sudo apt install docker-compose
```

Nota: Si estás usando ovh con Debian 10 y Docker preinstalado no es necesario instalar previamente docker, pero en caso contrario sí debes hacerlo ya que lanzaremos el despliegue usando docker. **2.** Una vez que tenemos instalado docker-compose, pasa a clonar tu proyecto de la práctica 3.

```
git clone urlRepositorio
```

3. Accede al directorio donde se encuentra tu proyecto. Crea dentro el fichero Dockerfile y configúralo.

4. Antes de generar la imagen, vamos a modificar dentro del archivo index.js de nuestro proyecto (src/app.module.ts si tu proyecto ha sido hecho con Nest) la ruta a la cual se conecta la base de datos:

Antes con Node:

```
mongoose.connect('mongodb://127.0.0.1:27017/playREST_v3',
{ useNewUrlParser: true, useUnifiedTopology: true });
```

Después con Node:

```
mongoose.connect('mongodb://mongodb:27017/playREST_v3',
{ useNewUrlParser: true, useUnifiedTopology: true });
```

Antes con Nest:

```
imports: [JuegoModule,
MongooseModule.forRoot('mongodb://127.0.0.1/playrest_V3'),
WebModule],
```

Después con Nest:

```
imports: [JuegoModule,
MongooseModule.forRoot('mongodb://mongodb/playrest_V3'),
WebModule],
```

Lo que estamos haciendo con esto es decirle a la base de datos que debe conectarse con el container de mongo que vamos a configurar en nuestro archivo docker-compose.yml en lugar de a nuestro localhost (127.0.0.1)

Nota: Esto mismo lo deberéis hacer en el fichero que tenéis en el proyecto para generar los usuarios de la aplicación.

5. Antes de generar la imagen vamos a cambiar de nuestro proyecto dos cosas:

> **5.1** En el archivo de package.json vamos a indicar en la zona de script que se ejecute el proyecto ("start": "node ."), y para ello tendremos en el bloque scripts:

```
"scripts": {
    "start": "node .",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
```

> **5.2** El segundo paso será importar el archivo generar_usuarios.js para que antes de lanzarse la aplicación se ejecute, y esto lo haremos en nuestro index.js:

```
...
// Ejecutamos el script de los usuarios primero
import './utils/generar_usuarios.js';
// Resto de nuestro código

...
```

6. Generamos la imagen de nuestro servicio web

```
sudo docker build -t <usuario docker-hub>/<nombre de la imagen> .
```

Y comprobamos que la imagen ha sido creada correctamente con:

```
sudo docker images
```

7. Creamos y configuramos nuestro fichero de despliegue docker-compose.yml, recuerda que este fichero debe estar fuera de la raíz de nuestro proyecto:

|-practica4
|--docker-compose.yml
|--practica3
|---Dockerfile
|---package.json
|--- ... \

Nota Recuerda que en la configuración le debes decir que tu aplicación depende del servicio de mongo para que se ejecute primero la bbdd. Que estamos usando un tipo de red bridge para que puedan comunicarse los contenedores y que mongo está usando un volumen /data/db. (Ver ejercicio propuesto tema 4.2)

8. Lanza la aplicación con el comando

```
sudo docker-compose up
```

> Tras estos pasos tu aplicación deberá estar operativa. Prueba que todo funciona correctamente antes de entregar.

Entrega y calificación

Se debe entregar un documento de texto explicando **todo** el proceso seguido (incluid imágenes). Además hay que indicar el nombre de dominio donde esté la aplicación, y el usuario y contraseña de prueba que se puede utilizar para probarla. De forma opcional, también se puede indicar la IP, por si el nombre de dominio diera algún problema.