03 NEST MICROSERVICIOS DOCKERFILE

- En api-gateway crearemos (en la raíz) el Dockerfile, partiremos de node:20
- Que trabaje en el directorio /app
- Copiamos el package.json. El . es mi WORKDIR
- Que instale las dependencias necesarias
- Que copie todo en mi directorio
- Que haga el build
- Que ejecute el main.js

```
FROM node:20.10.0

WORKDIR /app

COPY package.json .

RUN npm install --legacy-peer-deps

COPY . .

RUN npm run build

CMD ["node", "dist/main.js"]
```

- El resto de microservicios van a usar el mismo Dockerfile (copio y pego)

Creación de docker-compose

- En la raíz de ms-superflights tendré el docker-compose.dev y el docker-compose.prod
- Debo copiar en el .env el JWT_SECRET y el EXPIRES_IN
- En las variables de entorno, en el string de conexión de mongo **CAMBIO LOCALHOST POR mongodb**
- Para el string de conexión de RabbitMQ usaremos **amqps://rabbitmq:5672**
- docker-compose.dev

```
version: '3.7'

services:
  app: ## aplicación principal
    image: app_vuelos:v2
    container_name: app_vuelos
    build:      ## hacemos el build
      context: ./api-gateway ## el primer build será el de api-gateway
      dockerfile: Dockerfile ## usaremos Dockerfile
    env_file: .env.example ## le indico donde estan las variables de entorno
    ports:
```

```
      - 80:3000  ## mapeo el puerto 80 del pc con el 3000 del contenedor
    depends_on: ## los servicios correrán siempre y cuando mongodb y rabbitmq se
encuentren corriendo
      - mongodb
      - rabbitmq
    restart: always ## siempre reiniciaremos el servicio
    networks:
      - ms_nestjs   ## la red se va a llamar así

    ## lo mismo con los microservicios, pero no escucharemos a través de ningún
puerto
    ## la comunicación será a través de rabbitMQ
  microservice-flights:
    image: microservice-flights:v2
    container_name: microservice-flights
    build:
      context: ./microservice-flights
      dockerfile: Dockerfile
    env_file: .env.example
    depends_on:
      - mongodb
      - rabbitmq
    restart: always
    networks:
      - ms_nestjs

  microservice-passengers:
    image: microservice-passengers:v2
    container_name: microservice-passengers
    build:
      context: ./microservice-passengers
      dockerfile: Dockerfile
    env_file: .env.example
    depends_on:
      - mongodb
      - rabbitmq
    restart: always
    networks:
      - ms_nestjs

  microservice-users:
    image: microservice-users:v2
    container_name: microservice-users
    build:
      context: ./microservice-users
      dockerfile: Dockerfile
    env_file: .env.example
    depends_on:
      - mongodb
      - rabbitmq
    restart: always
    networks:
      - ms_nestjs
```

```
  ## descargo la imagen de rabbitmq
  rabbitmq:
    image: rabbitmq:3-management
    container_name: rabbitmq
    expose:
      - 5672  ## expongo el puerto del string de conexión
      - 15672
    restart: always
    networks:
      - ms_nestjs

  ## la imagen de mongo!
  mongodb:
    image: mongo:4.4.6
    container_name: mongodb
    restart: always
    environment: ## le indico donde almacenará la data
      - MONGO_DATA_DIR=/data/db
      - MONGO_LOG_DIR=/dev/null
    volumes: ## creamos un volumen para la persistencia de datos
      - mongodb:/data/db
    expose: ## expongo el puerto
      - 27017
    networks: ## pertenece a la red que hemos creado
      - ms_nestjs
## indico el volumen
volumes:
  mongodb:

## indico la red
networks:
  ms_nestjs:
```

Despliegue de contenedores con microservicios

- Hago el build de api-gateway

```
npm run build
```

- Hago lo mismo para los microservicios
- En las .env de ms-superflights le quitaremos la s a la conexion de rabbitmq . No usaremos ssl, haremos la conexión dentro de nuestro contenedor

```
amqp://rabbitmq:5672
```

- Uso docker compose up --build -d para levantar los contenedores en la raíz principal
- En Docker puedo ver que app_vuelos (api-gateway) está en el puerto 80
- En POSTMAN ya no estamos en el puerto 3000, si no el 80

Push de imágenes a DockerHub

- Me loggeo en docker desde la terminal

```
docker login
```

- Para subir la imágen coloco tag seguido del nombre de la aplicación, mi nombre de usuario/el nombre de la imagen como la deseo nombrar

```
docker tag app_vuelos:2 pepe2000/app_vuelos:2
```

- Hago lo mismo con el resto de microservicios
- rabbitmq y mongo ya los estoy descargando desde docker
- Para hacer el push

```
docker push pepe2000/app_vuelos:2
```

- Hago push del resto

Docker compose para producción

- Usamos las imágenes de DockerHub
- En .env tendremos un token que expirará en 12 horas, coloco en EXPIRES_IN=12h

```
version: '3.7'

services:
  app:
    image: acordova200/app_vuelos:v2
    container_name: app_vuelos
    env_file: .env.example
    ports:
      - 80:3000
    depends_on:
      - mongodb
      - rabbitmq
    restart: always
    networks:
      - ms_nestjs

  microservice-flights:
    image: acordova200/microservice-flights:v2
    container_name: microservice-flights
    env_file: .env.example
    depends_on:
      - mongodb
      - rabbitmq
    restart: always
    networks:
      - ms_nestjs

  microservice-passengers:
    image: acordova200/microservice-passengers:v2
```

```
    container_name: microservice-passengers
    env_file: .env.example
    depends_on:
      - mongodb
      - rabbitmq
    restart: always
    networks:
      - ms_nestjs

  microservice-users:
    image: acordova200/microservice-users:v2
    container_name: microservice-users
    env_file: .env.example
    depends_on:
      - mongodb
      - rabbitmq
    restart: always
    networks:
      - ms_nestjs

  rabbitmq:
    image: rabbitmq:3-management
    container_name: rabbitmq
    expose:
      - 5672
      - 15672
    restart: always
    networks:
      - ms_nestjs

  mongodb:
    image: mongo:latest
    container_name: mongodb
    restart: always
    environment:
      - MONGO_DATA_DIR=/data/db
      - MONGO_LOG_DIR=/dev/null
    volumes:
      - mongodb:/data/db
    expose:
      - 27017
    networks:
      - ms_nestjs

volumes:
  mongodb:

networks:
  ms_nestjs:
```

---

04 NEST MICROSERVICIOS AWS

- Nos logueamos en AWS
- En All Services acced EC2
- Aquí es donde crearemos las instancias. Voy a instances running
- Le doy a Launch instances (free only)
- Seleccionamos Ubuntu server LTS en 64 bits
- Next, Next, le coloco un espacio de 20 GB en Add Storage
- Next, Next, En el 6: Configure Security group
    - Le coloco de nombre internet
    - Description: ssh
- Tengo SSH TCP 22 Custom 0.0.0.0/0
- Agrego HTTP TCP 80 Custom 0.0.0.0/0, ::/0
- HTTPS TCP 443 Custom 0.0.0.0/0,. ::/0
- Le damos a Launch
- Uso mi llave privada (si no creo una) (hay que tenerla descargada)
- Launch instances
- En la pantalla de instances, doy click en mi instance ID o selecciono y doly click a conectar
- En EC2 instances doy click a conectar
- Estoy en el ubuntu server
- Hago un apt update && upgrade

Despliegue de contenedores

- El despliegue no lo haremos desde la consola, lo haremos desde un programa que se llama **MobaXterm**
- En SSH, pego la ip publica de mi instancia en remote host, en specify user name le digo ubuntu
- En use private key copio mi llave privada
- OK
- Para instalar Docker uso sudo apt install docker.io

```
cd /opt
```

- Creo la carpet microservices-superflights

```
sudo mkdir microservices-superflights cd microservices-superflights sudo nano docker-compose.yml
```

- Copiamos el docker-compose.prod.yml y lo pegamos en el editor de la consola de ubuntu que hemos abierto (nano)
- Guardamos como docker-compose.yml
- Creamos el archivo de variables de entorno con sudo nano .env
- Pegamos las env

```
# API
APP_URL=https://superflights.com
PORT=3000

# JWT
```

```
JWT_SECRET=JWTCl4v3S3cr3t4@Api
EXPIRES_IN=12h

#Database Connection
URI_MONGODB=mongodb://mongodb:27017/superFlights

#RabbitMQ
AMQP_URL=amqp://rabbitmq:5672
```

- Guardo
- Agrego $USER al grupo de docker

```
sudo usermod -aG docker $USER
```

- Reinicio docker service

```
sudo service docker restart
```

- Corro los contenedores

```
sudo docker compose up -d
```

- Copiamos la direccion IP pública de AWS en la pantalla de instance summary de mi instancia
- La pego en el navegador y agrego /api/docs y tengo la documentación de Swagger