DOCUMENTACIÓN DE DESPLIEGUE

Grupo 6:
Erik Ranea, Ibai Saenz de Buruaga, Jason Varas

INDICE

1. Propuesta de despliegue

La arquitectura basada en API que utilizamos requiere de dos servicios indenpendientes que provean tanto el **Frontend** como el **Backend**.

Para ahorrar recursos públicos hemos utilizado contenedores **Docker** para desplegar los servicios.

La base de datos que vamos a utilizar es la que nos brinda Egibide. Gracias a su labor social, no será necesario invertir recursos en este apartado.

1.1 Creación de imagenes

En primer lugar vamos a crear el **Dockerfile** del **Backend.** Para ello utilizaremos una imagen de **PHP** con **Apache** para poder desplegar el servicio de **API**.

Está imagen contendra un servicio web gracias a la herramienta de **Apache** la cual hará que se puedan realizar peticiones a la **API**. Y luego expondremos el puerto **80** para poder ser accesible desde otros contenedores.

A continuación se podrá ver el Dockerfile con la líneas comentadas para poder ver las anidanciones necesarias. **(imagen-1)**

```
# Usa una imagen base de PHP con Apache (Debian)
FROM php:8.2-apache

# Instala dependencias necesarias (para PDO y MySQL, y otras)
RUN apt-get update && apt-get install -y \
    $PHPIZE_DEPS \
    libzip-dev \
    zip \
    && docker-php-ext-install pdo pdo_mysql zip

# Instala Composer
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer

# Establece el directorio de trabajo en el contenedor
WORKDIR /var/www/html

# Copia todos los archivos de la aplicación Laravel
COPY . /var/www/html

# Instala las dependencias de Laravel usando Composer
RUN composer install

# Establece la propiedad de la carpeta bootstrap/cache al usuario www-data
RUN chown -R www-data:www-data bootstrap/cache

# Establece la propiedad de la carpeta storage al usuario www-data
RUN chown -R www-data:www-data storage

# Establece la propiedad de los archivos dentro de la carpeta storage al usuario www-data, incluyendo la carpeta app/public
RUN chown -R www-data:www-data storage/app/public

# Establece permisos para la carpeta storage y sus subcarpetas
RUN chmod -R 775 /var/www/html/storage

# Diagnóstico: Verifica que el directorio storage/app/public existe y sus permisos
RUN echo "Verificando existencia y permisos de storage/app/public:"
RUN ls -l storage/app
RUN ls -l storage/app/public

# Diagnóstico: Verifica permisos del directorio public
RUN echo "Verificando permisos del directorio public:"
RUN ls -l public

# Intenta crear el directorio storage/app/public si no existe (aunque ya debería existir al copiar el proyecto)
RUN echo "Creando el directorio storage/app/public (si no existe):"
RUN mkdir -p storage/app/public

# Ejecuta el comando para crear el enlace simbólico
RUN echo "Creando enlace simbólico:"
RUN php artisan storage:link

# Expone el puerto 80
EXPOSE 80

# Configura Apache para que sirva Laravel (modificar el VirtualHost)
RUN sed -i 's!/var/www/html!/var/www/html/public!g' /etc/apache2/sites-available/000-default.conf

# Habilita mod_rewrite para URLs amigables de Laravel
RUN a2enmod rewrite


# ---------------------------------- Comando por defecto modificado ----------------------------------
# Ejecuta Apache en primer plano
CMD apache2-foreground
```

Imagen-1

El siguiente paso a seguir será crear la imagen del **Frontend**.
Para ello es dividir está imagen en dos etapas.

En la primera etapa, crearemos una imagen de Node.js para **buildear** la página estática que necesita el servidor web para funcionar. Al haber desarrollado el **Frontend** con **Vue** es necesario realizar un **compilado/empaquetado** del proyecto para tener los archivos listos para desplegar en un servidor web.

En la segunda etapa, recogemos el resultado de la compilación, un directorio **/dist** donde se encuentran los archivos estáticos, compilados, minificados, empaquetados y optimizados para su despliegue. **Desplegamos** en un servidor web con **Nginx** requiriendo un **nginx.conf** para poder hacer funcionar el servicio.

Como se puede ver a continuación, está sería la configuración del archivo **nginx.conf** (**imagen-2**)

```
server {
    listen 80;
    index index.html;
    root /usr/share/nginx/html;

    location / {
        try_files $uri $uri/ /index.html;
    }

    # Sirve la carpeta /assets/ directamente
    location /assets/ {
        alias /usr/share/nginx/html/assets/;
        expires 30d; # Opcional: Cache de navegador por 30 días para assets
    }

}
```

Imagen-2

La siguiente imagen se aprecia la configuración del **Dockerfile** del **Frontend (imagen-3)**

```
# Usa una imagen base de Node.js para la construcción de la app
FROM node:18-alpine as builder

# Establece el directorio de trabajo
WORKDIR /app

# Copia el archivo package.json y package-lock.json
COPY package*.json ./

# Instala las dependencias
RUN npm install

# Copia todo el resto del código
COPY . .

# Construye la aplicación Vue
RUN npm run build

# Utiliza una imagen nginx para servir los archivos estáticos
FROM nginx:alpine

# Copia los archivos de la construcción del paso anterior l servidor Nginx
COPY --from=builder /app/dist /usr/share/nginx/html
# Copia el archivo de configuración de Nginx personalizado
COPY nginx.conf /etc/nginx/conf.d/default.conf

# Expone el puerto 80
EXPOSE 80

# Ejecuta el servidor Nginx
CMD ["nginx", "-g", "daemon off;"]
```

Imagen-3

1.2 Creación del Docker-Compose

Para la creación del **Docker-compose** hemos planteado lo siguiente. Con la intención de hacer un despliegue sencillo y poliforme, hemos indicado la variables de entorno que necesita cada uno de los contenedores en el archivo **docker-compose.yml**

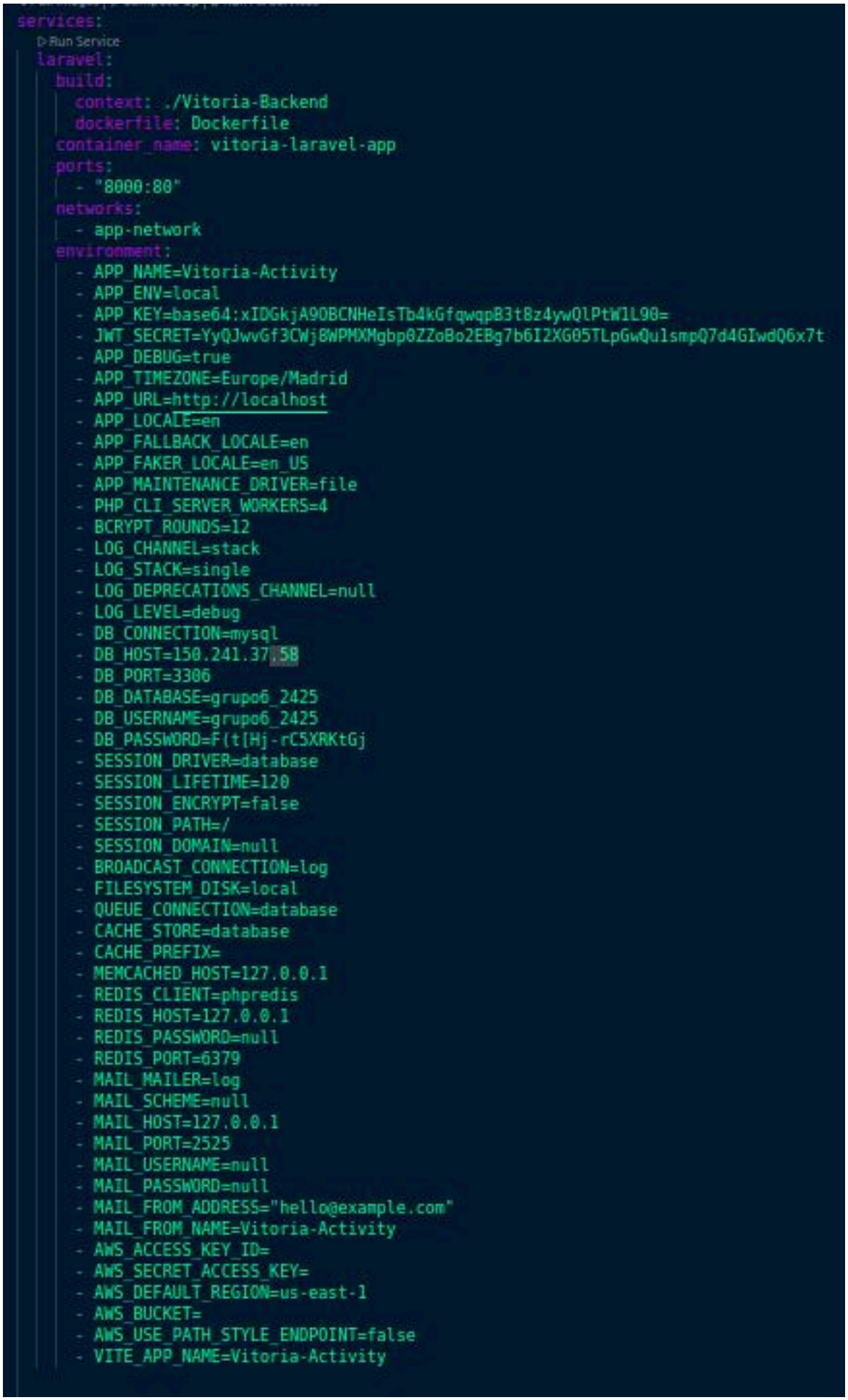
A la hora de **buildear** el contenedor del **Backend** lo hacemos las siguientes acciones:

- Realizar una redirección de puertos, del puerto **8000** al **80** del contenedor. Haciendo que el puerto que tiene expuesta la imagen del **Backend** reciba el tráfico a tráves del puerto **8000.** Permitiendo no colisionar con el servicio web que creamos en el **Frontend.**
- Le pasamos todas las variables de entorno necesarias para el funcionamiento del proyecto del **Laravel.** Permitiendo que puedas personalizar donde quieres que este la base de datos.

En cambio en la etapa del **buildeo** del contenedor del **Frontend** hemos hecho lo siguiente:

- Abrimos el puerto **80** al **80** del contenedor, ya que tiene el servicio web a la escucha en este puerto.
- Y le pasamos las variables de entorno que necesita para conectar con la **API**.

A continuación se puede ver el archivo **docker-compose.yml(imagen-5)**



```yaml
services:
  ▷ Run Service
  laravel:
    build:
      context: ./Vitoria-Backend
      dockerfile: Dockerfile
    container_name: vitoria-laravel-app
    ports:
      - "8000:80"
    networks:
      - app-network
    environment:
      - APP_NAME=Vitoria-Activity
      - APP_ENV=local
      - APP_KEY=base64:xIDGkjA9OBCNHeIsTb4kGfqwqpB3t8z4ywQlPtWlL90=
      - JWT_SECRET=YyQJwvGf3CWj8WPMXMgbp0ZZoBo2EBg7b6I2XG05TLpGwQulsmpQ7d4GIwdQ6x7t
      - APP_DEBUG=true
      - APP_TIMEZONE=Europe/Madrid
      - APP_URL=http://localhost
      - APP_LOCALE=en
      - APP_FALLBACK_LOCALE=en
      - APP_FAKER_LOCALE=en_US
      - APP_MAINTENANCE_DRIVER=file
      - PHP_CLI_SERVER_WORKERS=4
      - BCRYPT_ROUNDS=12
      - LOG_CHANNEL=stack
      - LOG_STACK=single
      - LOG_DEPRECATIONS_CHANNEL=null
      - LOG_LEVEL=debug
      - DB_CONNECTION=mysql
      - DB_HOST=150.241.37.58
      - DB_PORT=3306
      - DB_DATABASE=grupo6_2425
      - DB_USERNAME=grupo6_2425
      - DB_PASSWORD=F(t[Hj-rC5XRKtGj
      - SESSION_DRIVER=database
      - SESSION_LIFETIME=120
      - SESSION_ENCRYPT=false
      - SESSION_PATH=/
      - SESSION_DOMAIN=null
      - BROADCAST_CONNECTION=log
      - FILESYSTEM_DISK=local
      - QUEUE_CONNECTION=database
      - CACHE_STORE=database
      - CACHE_PREFIX=
      - MEMCACHED_HOST=127.0.0.1
      - REDIS_CLIENT=phpredis
      - REDIS_HOST=127.0.0.1
      - REDIS_PASSWORD=null
      - REDIS_PORT=6379
      - MAIL_MAILER=log
      - MAIL_SCHEME=null
      - MAIL_HOST=127.0.0.1
      - MAIL_PORT=2525
      - MAIL_USERNAME=null
      - MAIL_PASSWORD=null
      - MAIL_FROM_ADDRESS="hello@example.com"
      - MAIL_FROM_NAME=Vitoria-Activity
      - AWS_ACCESS_KEY_ID=
      - AWS_SECRET_ACCESS_KEY=
      - AWS_DEFAULT_REGION=us-east-1
      - AWS_BUCKET=
      - AWS_USE_PATH_STYLE_ENDPOINT=false
      - VITE_APP_NAME=Vitoria-Activity
```

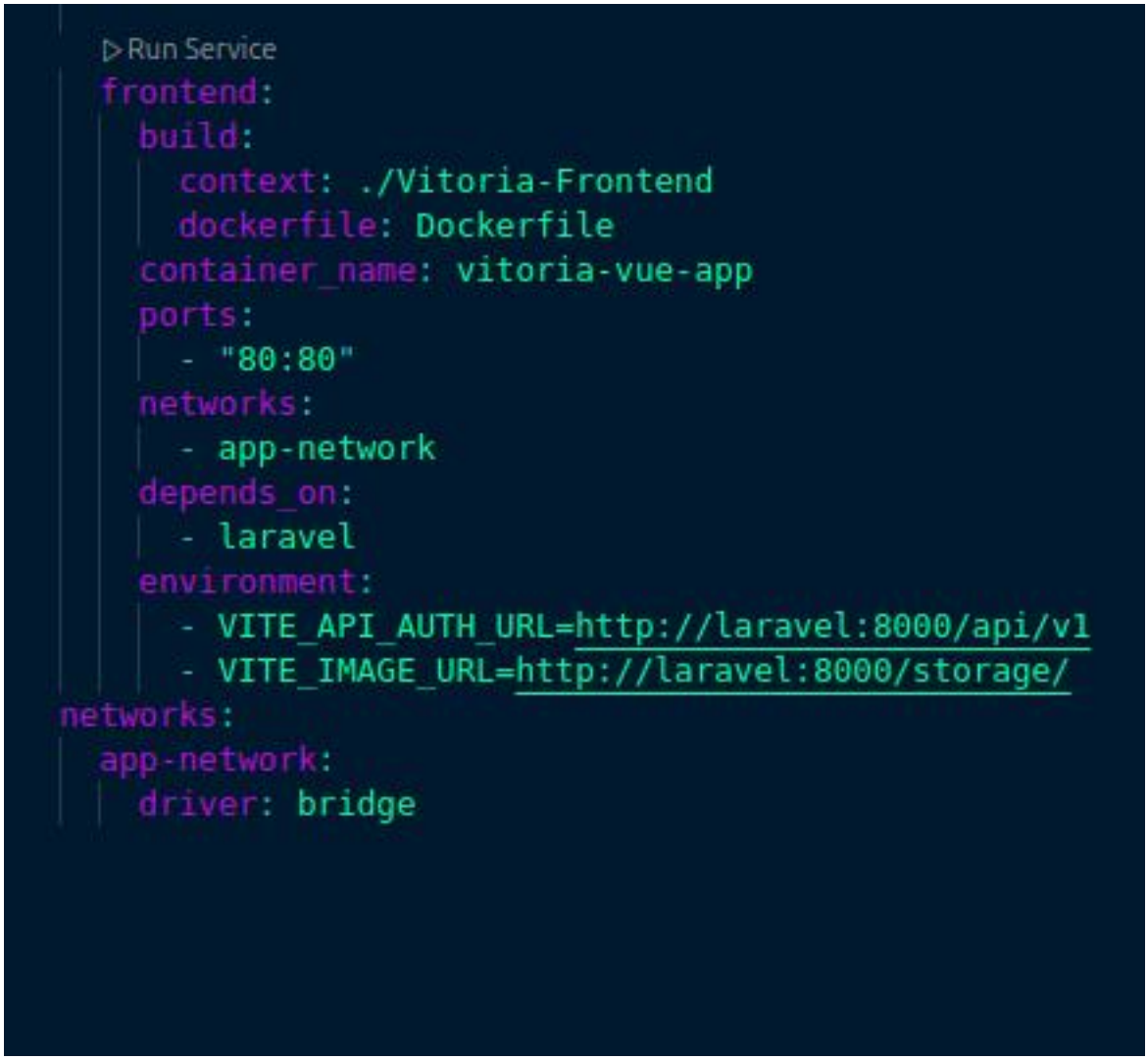

▷ Run Service
frontend:
build:
context: ./Vitoria-Frontend
dockerfile: Dockerfile
container_name: vitoria-vue-app
ports:
- "80:80"
networks:
- app-network
depends_on:
- laravel
environment:
- VITE_API_AUTH_URL=http://laravel:8000/api/v1
- VITE_IMAGE_URL=http://laravel:8000/storage/
networks:
app-network:
driver: bridge

Imagen-5

1.3 Desplagar la aplicación

Para desplegar la aplicació debemos seguir los siguientes pasos:

- Clonar el repositorio en una maquina preferiblemente **ubuntu** con **Docker Engine *instalado***.
- Entramos a la raíz del proyecto y ejecutamos **docker compose build**
- Para levantar los contenedores ejecutamos **docker compose up -d**

Una vez realizado esto, estaría listo para su uso :)