Voice Academy Versión 01

Documento de Evidencia de Despliegue Docker

Equipo “Rusteze” conformado por:

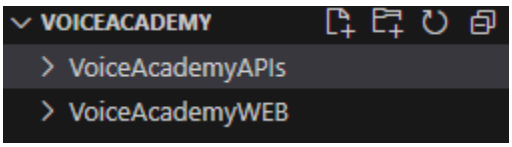
- Eduart Ussiel Dircio Cayón
- Dan Joshua Segura Domínguez
- Cristopher Rodríguez Salamanca

Universidad Veracruzana
Facultad de Estadística e Informática

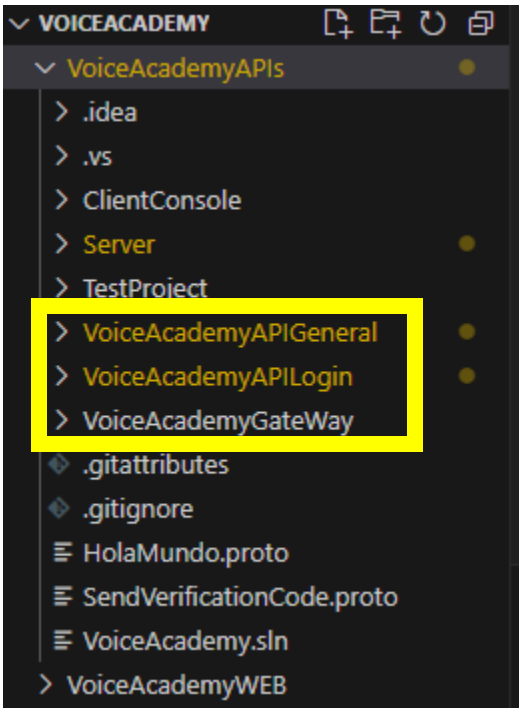
Experiencia Educativa: Desarrollo de Aplcaciones

Académico: Juan Luis López Herrera

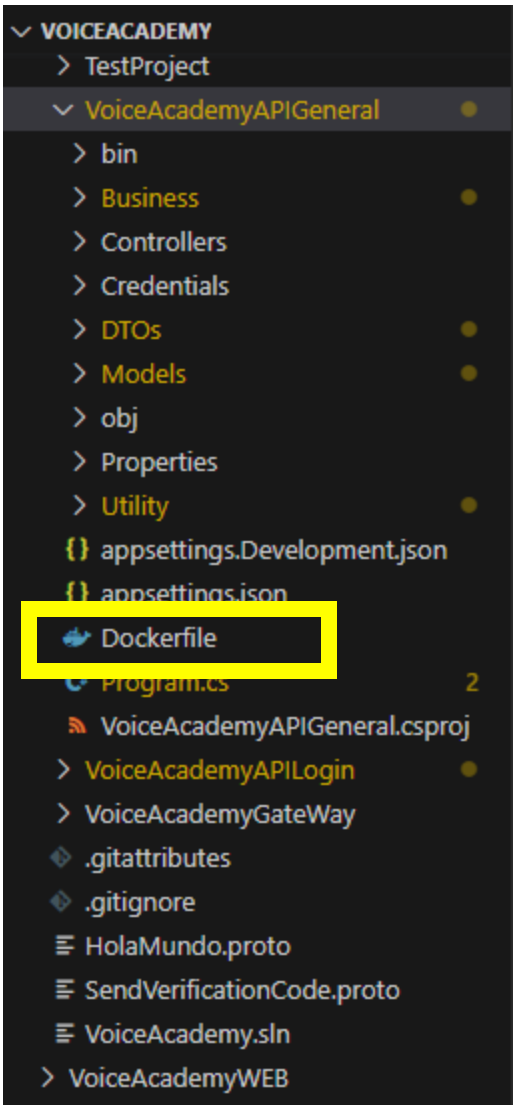
Se creó una carpeta “contenedora” de los proyectos creados para nuestro sistema:

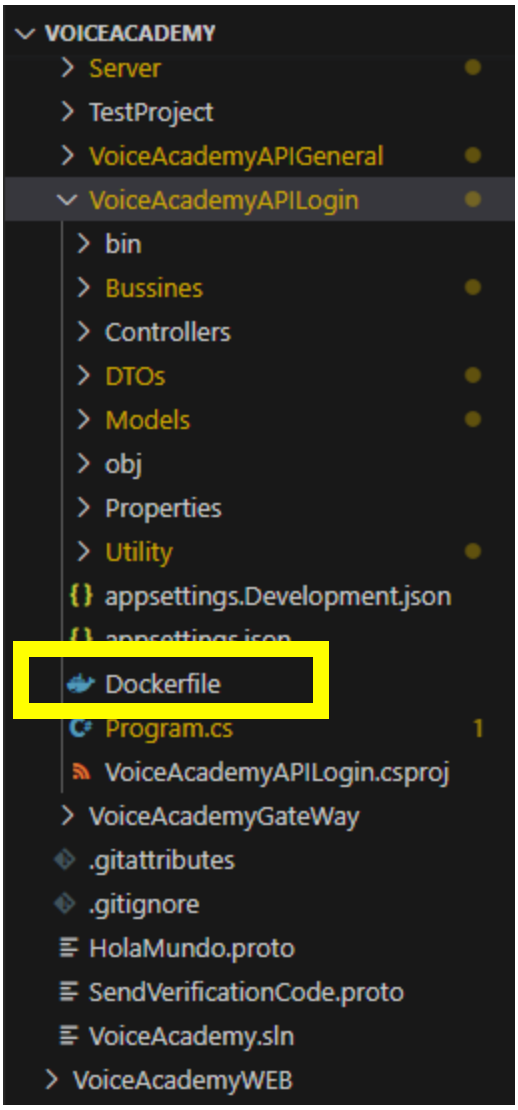


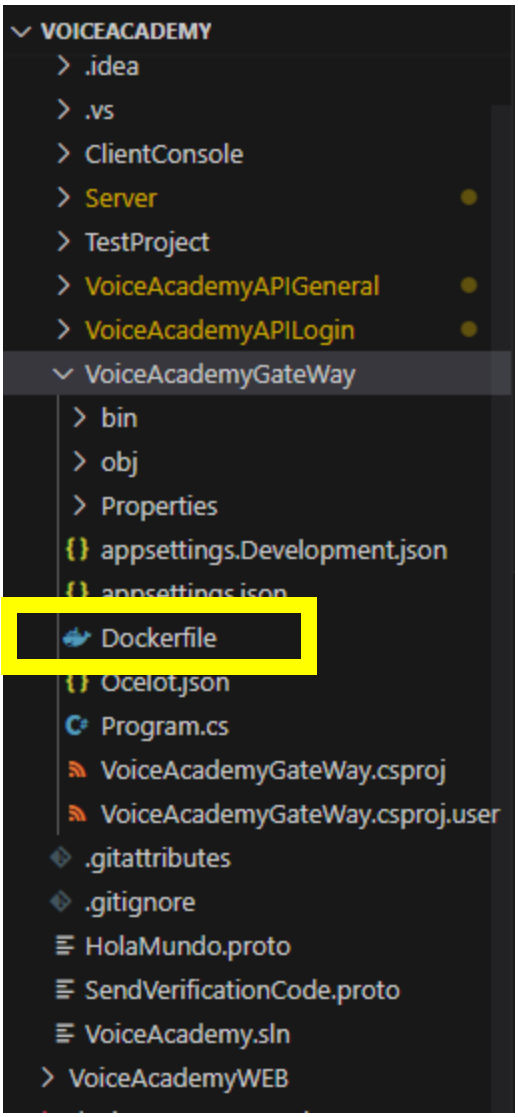
Dentro de la carpeta VoiceAcademyAPIs se encuentran nuestras API:



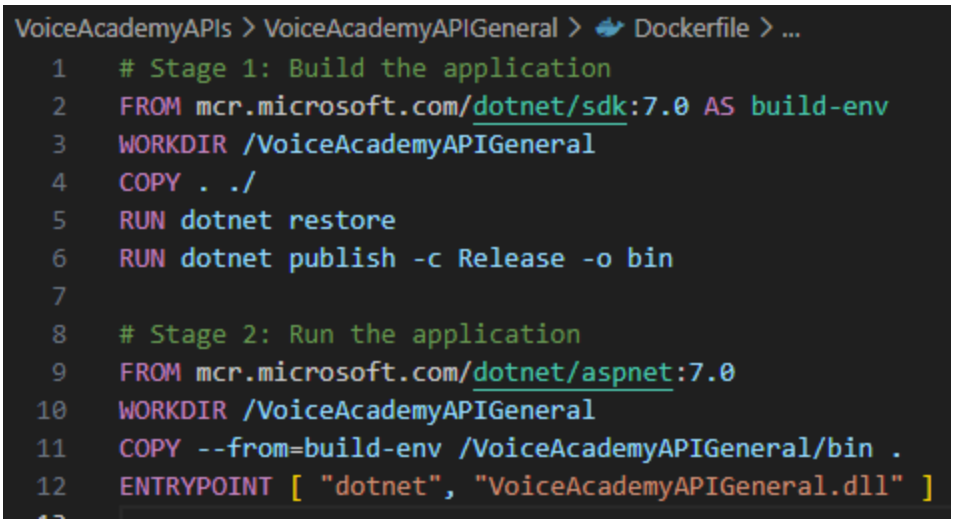
En cada API colocamos un archivo Docker:


VOICEACADEMY
TestProject
VoiceAcademyAPIGeneral
bin
Business
Controllers
Credentials
DTOs
Models
obj
Properties
Utility
appsettings.Development.json
appsettings.json
Dockerfile
Program.cs
2
VoiceAcademyAPIGeneral.csproj
VoiceAcademyAPILogin
VoiceAcademyGateWay
.gitattributes
.gitignore
HolaMundo.proto
SendVerificationCode.proto
VoiceAcademy.sln
VoiceAcademyWEB


VOICEACADEMY
Server
TestProject
VoiceAcademyAPIGeneral
VoiceAcademyAPILogin
bin
Bussines
Controllers
DTOs
Models
obj
Properties
Utility
appsettings.Development.json
Dockerfile
Program.cs
VoiceAcademyAPILogin.csproj
VoiceAcademyGateWay
.gitattributes
.gitignore
HolaMundo.proto
SendVerificationCode.proto
VoiceAcademy.sln
VoiceAcademyWEB



Contenido de archivos Docker:



Etapa 1: Construir la aplicación

- Se utiliza la imagen base "mcr.microsoft.com/dotnet/sdk:7.0", que proporciona el SDK de .NET Core 7.0 para compilar la aplicación.
- Se establece el directorio de trabajo dentro del contenedor en "/VoiceAcademyAPIGeneral".
- Se copian todos los archivos y carpetas del directorio actual al directorio de trabajo del contenedor.
- Se pide que ejecute el comando "dotnet restore" para restaurar las dependencias del proyecto.
- Se pide que ejecute el comando "dotnet publish" para compilar y publicar la aplicación en modo de lanzamiento ("Release") en el directorio "bin".

Etapa 2: Ejecutar la aplicación

- Se utiliza igualmente la imagen base "mcr.microsoft.com/dotnet/aspnet:7.0", que proporciona el entorno de ejecución de .NET Core 7.0 para ejecutar la aplicación.
- Se establece el directorio de trabajo dentro del contenedor en "/VoiceAcademyAPIGeneral".
- Se copian los archivos generados en la etapa anterior desde el directorio "/VoiceAcademyAPIGeneral/bin" del contenedor al directorio de trabajo actual del contenedor.
- Se establece el punto de entrada ("ENTRYPOINT") para el contenedor, indicando que se debe ejecutar el archivo "VoiceAcademyAPIGeneral.dll" utilizando el comando "dotnet".

Para los demás archivos es el mismo proceso, se adjunta únicamente capturas por las 2 API restantes:

VoiceAcademyAPIs > VoiceAcademyAPILogin > Dockerfile > ...

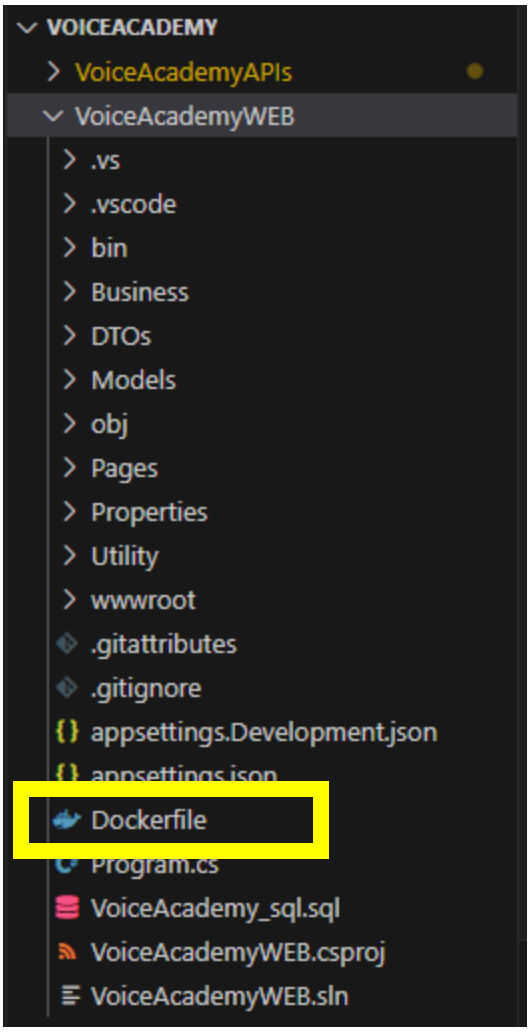
```
1   # Stage 1: Build the application
2   FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build-env
3   WORKDIR /VoiceAcademyAPILogin
4   COPY . ./
5   RUN dotnet restore
6   RUN dotnet publish -c Release -o bin
7
8   # Stage 2: Run the application
9   FROM mcr.microsoft.com/dotnet/aspnet:7.0
10  WORKDIR /VoiceAcademyAPILogin
11  COPY --from=build-env /VoiceAcademyAPILogin/bin .
12  ENTRYPOINT [ "dotnet", "VoiceAcademyAPILogin.dll" ]
```

VoiceAcademyAPIs > VoiceAcademyGateWay > Dockerfile > ...

```
1   # Stage 1: Build the application
2   FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build-env
3   WORKDIR /VoiceAcademyAPIGateWay
4   COPY . ./
5   RUN dotnet restore
6   RUN dotnet publish -c Release -o bin
7
8   # Stage 2: Run the application
9   FROM mcr.microsoft.com/dotnet/aspnet:7.0
10  WORKDIR /VoiceAcademyAPIGateWay
11  COPY --from=build-env /VoiceAcademyAPIGateWay/bin .
12  ENTRYPOINT [ "dotnet", "VoiceAcademyAPIGateWay.dll" ]
```

Dentro de nuestra aplicación web se realiza el mismo procedimiento:

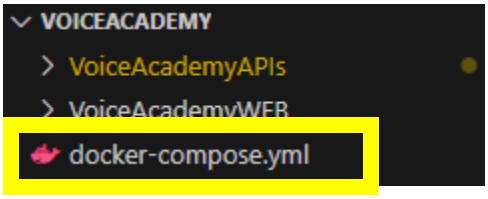
Colocamos el archivo Docker dentro:



Contenido archivo Docker:

```
VoiceAcademyWEB > Dockerfile > ...
1   # Stage 1: Build the application
2   FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build-env
3   WORKDIR /VoiceAcademyWEB
4   COPY . ./
5   RUN dotnet restore
6   RUN dotnet publish -c Release -o bin
7
8   # Stage 2: Run the application
9   FROM mcr.microsoft.com/dotnet/aspnet:7.0
10  WORKDIR /VoiceAcademyWEB
11  COPY --from=build-env /VoiceAcademyWEB/bin .
12  ENTRYPOINT [ "dotnet", "VoiceAcademyWEB.dll" ]
```

Ahora, colocamos el archivo Docker Compose en la carpeta “Contenedora” de nuestros proyectos:



¿Por qué usamos Docker Compose?, para definir y ejecutar nuestros servicios relacionados entre sí, como la aplicación web con su base de datos.

Aparte, usar Docker Compose nos resuelve automáticamente las dependencias entre servicios, así que si un servicio depende de otro, este nos ayuda de que se inicie en el orden adecuado.

Contenido de Docker Compose:

docker-compose.yml

```yaml
version: '3'
services:
  api_gateway:
    build:
      context: ./VoiceAcademyAPIs/VoiceAcademyGateWay
      dockerfile: Dockerfile
    ports:
      - 5000:5000
    depends_on:
      - apilogin
      - apigeneral

  apilogin:
    build:
      context: ./VoiceAcademyAPIs/VoiceAcademyAPILogin
      dockerfile: Dockerfile
    ports:
      - 5186:5186
    depends_on:
      - mysql

```

```yaml
mysql:
  image: mysql:latest
  environment:
    - MYSQL_ROOT_PASS
    - MYSQL_DATABASE=
    - MYSQL_USER=Voic
    - MYSQL_PASSWORD=
  ports:
    - 3306:3306
  volumes:
    - mysql-data:/var/lib/mysql
    - ./VoiceAcademyWEB\VoiceAcademy_sql.sql:/docker-entrypoint-initdb.d/init.sql

apigeneral:
  build:
    context: ./VoiceAcademyAPIs/VoiceAcademyAPIGeneral
    dockerfile: Dockerfile
  ports:
    - 5003:5003
  depends_on:
    - mysql
```

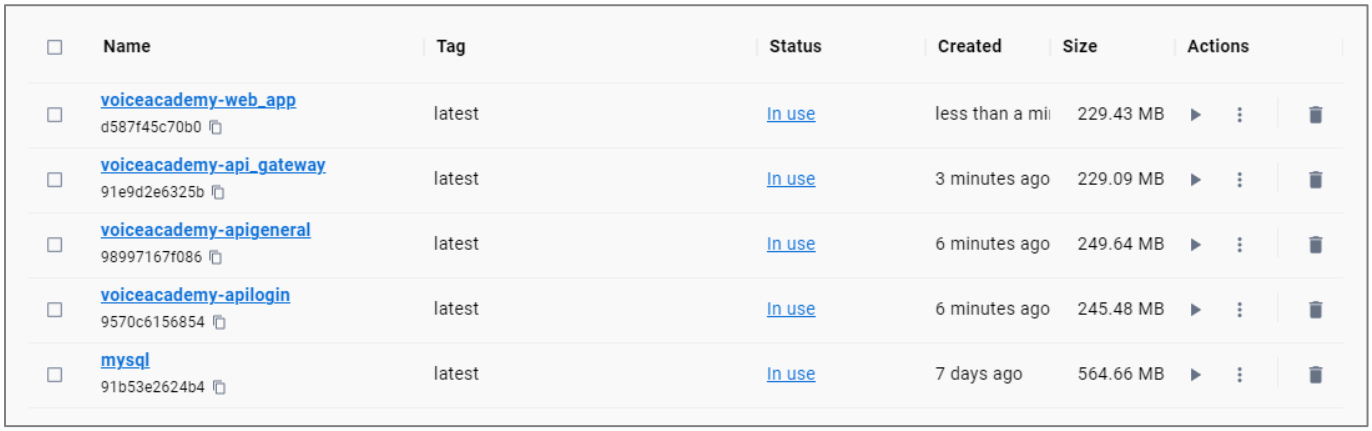
```yaml
  apigeneral:
    build:
      context: ./VoiceAcademyAPIs/VoiceAcademyAPIGeneral
      dockerfile: Dockerfile
    ports:
      - 5003:5003
    depends_on:
      - mysql

  web_app:
    build:
      context: ./VoiceAcademyWEB
      dockerfile: Dockerfile
    ports:
      - 5145:80
    depends_on:
      - api_gateway

volumes:
  mysql-data:
```

- Servicios:
    - api_gateway: Aquí definimos el servicio "api_gateway". El cual creamos a partir de la imagen generada en el directorio **./VoiceAcademyAPIs/VoiceAcademyGateWay** utilizando el archivo Dockerfile especificado. Exponemos el puerto 5000 en el host y lo mapeamos al puerto 5000 dentro del contenedor. Este API depende de los otros servicios, "apilogin" y "apigeneral".
    - apilogin: Aquí definimos el servicio "apilogin". Lo construimos a partir de una imagen generada en el contexto del directorio **./VoiceAcademyAPIs/VoiceAcademyAPILogin** utilizando el archivo Dockerfile especificado. Exponemos el puerto 5186 en el host y lo mapeamos al puerto 5186 dentro del contenedor. Dependiendo del servicio "mysql".
    - mysql: Define el servicio "mysql". Aquí usamos la imagen **mysql:latest** predefinida desde Docker Hub. Se establecen variables de entorno para la configuración de MySQL, como la contraseña de root,, usuario y la contraseña. Exponemos el puerto 3306 en el host y lo mapeamos al puerto 3306 dentro del contenedor. Además, creamos volúmenes para persistir los datos de MySQL en el directorio **/var/lib/mysql** y montamos el archivo **VoiceAcademy_sql.sql** en el contenedor para ejecutar scripts de inicialización.
    - apigeneral: Definimos el servicio "apigeneral". Lo construimos a partir de una imagen generada en el contexto del directorio **./VoiceAcademyAPIs/VoiceAcademyAPIGeneral** utilizando el archivo Dockerfile especificado. Exponemos el puerto 5003 en el host y lo mapeamos al puerto 5003 dentro del contenedor. Dependiendo del servicio "mysql".
    - web_app: Definimos nuestro servicio "web_app". Lo construimos a partir de una imagen generada en el contexto del directorio **./VoiceAcademyWEB** utilizando el archivo Dockerfile especificado. Exponemos el puerto 5145 en el host y lo mapeamos al puerto 80 dentro del contenedor. Dependiendo del servicio "api_gateway".
- Volúmenes:
    - mysql-data: Definimos el volumen "mysql-data" que utilizamos para persistir los datos de MySQL en el host.

Finalmente ejecutamos el Docker Compose con el comando: **docker-compose up -d**

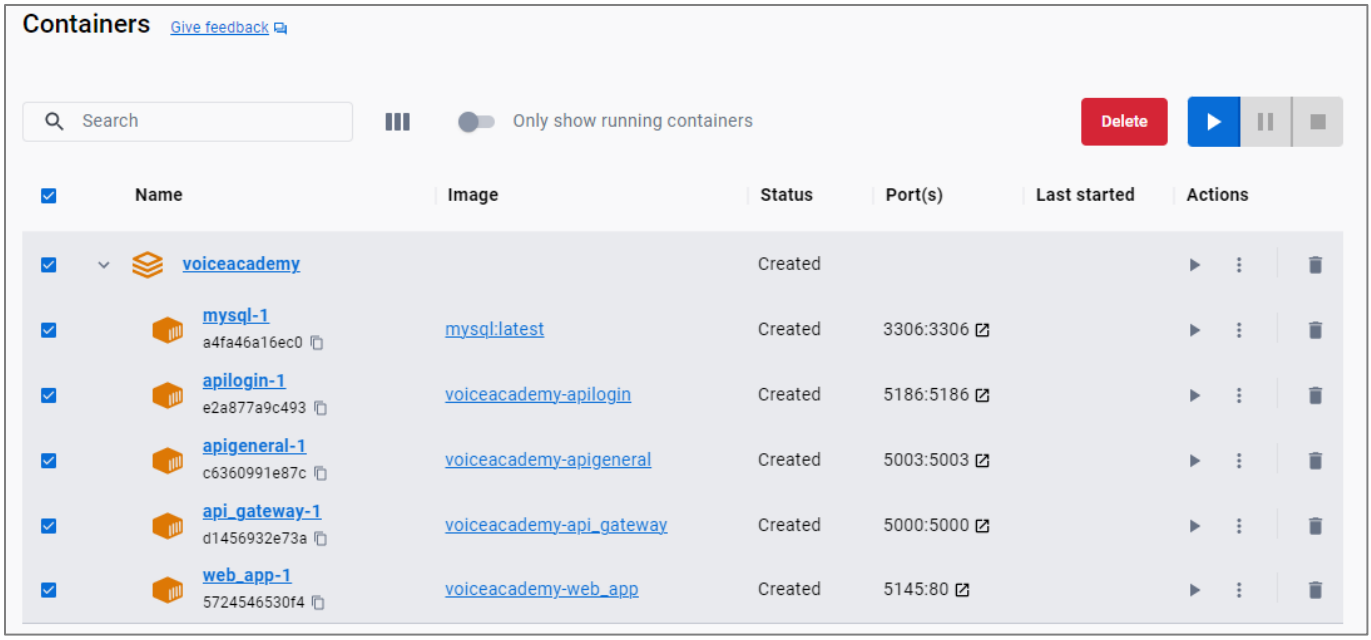
- **docker-compose**: Lo usamos porque tenemos varios contenedores como servicios utilizando un archivo de configuración.
- **up**: Lo utilizamos para crear y ejecutar los contenedores de los servicios definidos en el archivo de Docker Compose.
- **-d**: Lo usamos para indicarle a Docker Compose que los contenedores deben ejecutarse en “modo de demonio”, es decir, en segundo plano. Para que los servicios se ejecuten de forma continua sin bloquear la terminal.

RESULTADOS:



Name	Tag	Status	Created	Size	Actions
voiceacademy-web_app d587f45c70b0	latest	In use	less than a mi	229.43 MB	
voiceacademy-api_gateway 91e9d2e6325b	latest	In use	3 minutes ago	229.09 MB	
voiceacademy-apigeneral 98997167f086	latest	In use	6 minutes ago	249.64 MB	
voiceacademy-apilogin 9570c6156854	latest	In use	6 minutes ago	245.48 MB	
mysql 91b53e2624b4	latest	In use	7 days ago	564.66 MB	

IMÁGENES CREADAS.



Contenedores Creados.