Formation Docker : Manipulations pratiques avec C#

El Hadji Gaye

Auteur	El Hadji Gaye
Pour	Formations
Date	06/11/2024
Objet	Formation Docker : Manipulations pratiques avec C#.

I) Vocabulaire

Conteneurisation: En informatique, un conteneur est une structure de données, une classe, ou un type de données abstrait, dont les instances représentent des collections d'autres objets. Autrement dit, les conteneurs sont utilisés pour stocker des objets sous une forme organisée qui suit des règles d'accès spécifiques. On peut implémenter un conteneur de différentes façons, qui conduisent à des complexités en temps et en espace différentes. On choisira donc l'implémentation selon les besoins.

Un conteneur est une enveloppe virtuelle qui permet de distribuer une application avec tous les éléments dont elle a besoin pour fonctionner : fichiers source, environnement d'exécution, librairies, outils et fichiers. Ils sont assemblés en un ensemble cohérent et prêt à être déployé sur un serveur et son système d'exploitation (OS). Contrairement à la virtualisation de serveurs et à une machine virtuelle, le conteneur n'intègre pas de noyau, il s'appuie directement sur le noyau de l'ordinateur sur lequel il est déployé.

Virtualisation : La virtualisation consiste, en informatique, à exécuter sur une machine hôte, dans un environnement isolé, des systèmes d'exploitation — on parle alors de virtualisation système — ou des applications— on parle alors de virtualisation applicative. Ces ordinateurs virtuels sont appelés serveur privé virtuel (Virtual Private Server ou VPS) ou encore environnement virtuel (Virtual Environment ou VE).

II) Les commandes Docker à connaître

1. Commande docker ps

docker ps vous affiche toutes les instances de docker qui tournent actuellement sur votre environnement. Si vous ajoutez l'option -a, alors vous verrez mêmes les containers stoppés.

docker ps -a

2. *Commande docker images*

docker images est une commande qui vous montre les images que vous avez construites, et le -a vous montre les images intermédiaires.

docker images -a

3. *Commande docker network*

docker network ls est la commande docker qui liste les différents réseaux.

docker network ls

4. *Commande docker de runtime*

docker-compose up (-d) (--build)
docker-compose stop

La docker-compose est la plus simple car vous n'avez besoin que de 2 commandes : up et stop. stop est assez explicite et stop (mais ne supprime pas) vos conteneurs, mais up nécessite plus d'explications : cela va construire vos images si elles ne le sont pas déjà, et va démarrer vos dockers.

docker build (-t NAME) PATH /URL

Si vous voulez re-build vos images, utilisez l'option --build (vous pouvez aussi utiliser la commande docker-compose build pour uniquement construire des images). L'option -d, qui signifie "detach" fait tourner les conteneurs en tâche de fond.

Avec Docker, vous avez besoin d'une commande séparée pour construire votre image, où vous pouvez spécifier le nom de votre image et vous devez spécifier le PATH ou URL selon votre contexte (cela peut être un repo git).

docker run (-d) (-p hostPort :containerPort) (--name NAME)

run crée le conteneur en utilisant l'image que vous indiquez. Vous pouvez spécifier de nombreux paramètres. Nous vous recommandons d'ajouter un nom à votre conteneur et vous pourriez avoir besoin de spécifier quelques ports à exposer. Comme pour docker-compose, le -d lance le conteneur en tâche de fond.

docker start ID /NAME

docker stop ID /NAME

Le start and stop ne devraient pas être trop compliqués à comprendre, mais il faut noter que vous pouvez "start" uniquement des conteneurs qui sont déjà arrêtés, donc déjà build avec la commande run.

docker exec -it NAME /ID "sh" /"/bin/bash"

Cette commande vous permet de lancer un shell sur votre container. Je préfère utiliser "/bin/bash" mais votre conteneur peut ne pas avoir bash d'installé, et seulement "sh" qui est plus courant (surtout sur les alpines). Si vous avez des configurations spéciales dans votre conteneur, vous aurez peut-être besoin d'utiliser des arguments supplémentaires pour vous y connecter.

5. *Commandes docker de suppression*

Ces commandes permettent de supprimer vos conteneurs et vos images. Vous en aurez probablement besoin pour libérer de l'espace disque.

docker rm ID /NAME

docker-compose rm

Le docker rm supprime seulement un conteneur alors que docker-compose rm supprime tous les conteneurs démarrés avec une commande docker-compose.

docker rmi ID /NAME

Docker rmi supprime l'image que vous passez en paramètre et récursivement toutes les images intermédiaires utilisées pour la construire.

6. *Commandes docker de logs*

Les commandes suivantes sont utiles quand vous devez débugger certains de vos conteneurs (ou, plus souvent, l'application que vous déployez à l'intérieur).

docker logs ID /NAME (-f --tail NBLINE)

Cette commande affiche les logs du container passé en paramètre. Si vous utilisez l'option -f --tail NBLINE vous pouvez suivre en live le flux de vos logs (NBLINE est le nombre de lignes que vous souhaitez afficher). Gardez à l'esprit de choisir un nombre de lignes que vous serez capable de gérer, pour ne pas être dépassé par vos logs.

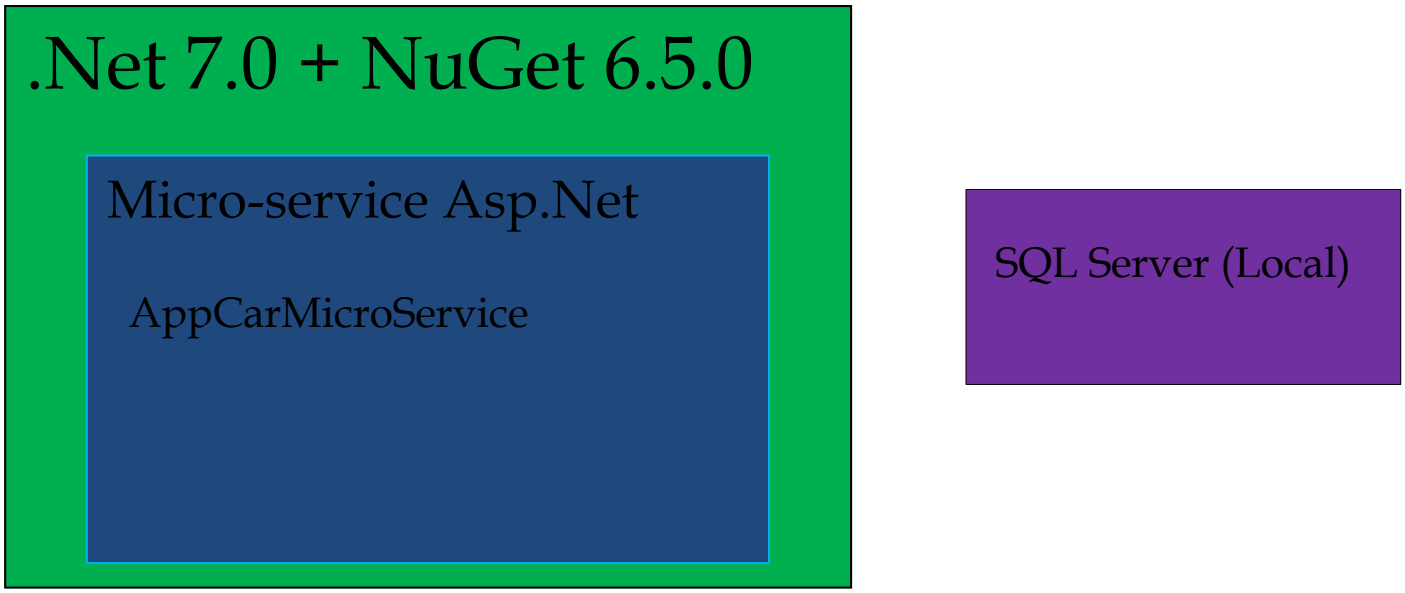
docker-compose logs (ID /NAME)

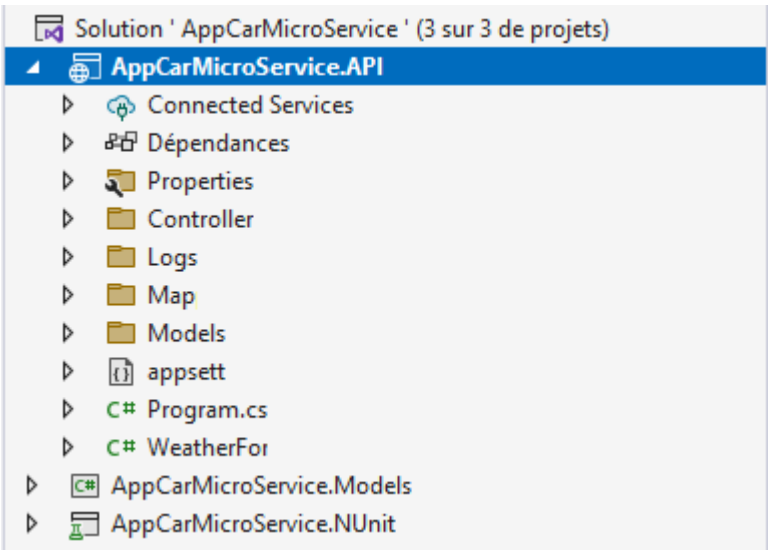
L'option (ID /NAME) avec docker-compose logs vous permet de voir les logs d'un conteneur uniquement, au lieu de voir tous les logs. L'astuce ici est que si vous n'utilisez pas l'option -d quand vous utilisez docker run ou docker-compose up vous verrez vos logs directement (mais vous aurez besoin d'arrêter le conteneur pour quitter la vue). Cela peut toujours être utile pour débugger des applications au démarrage.

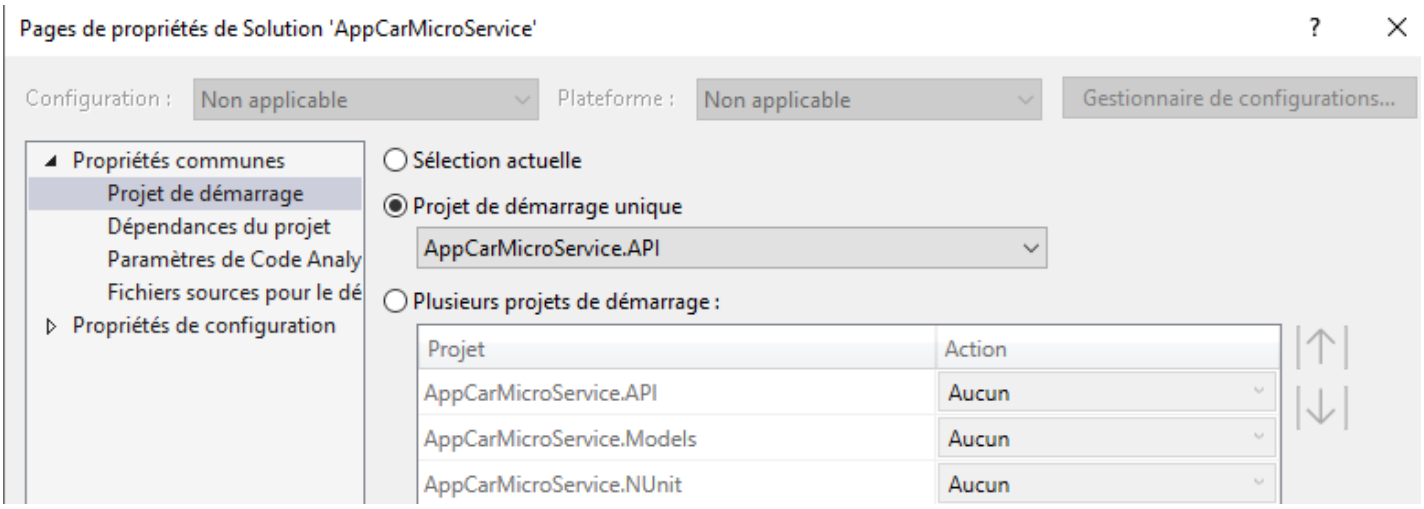
III) Déployer un Micro-Service Asp.Net Core avec Docker

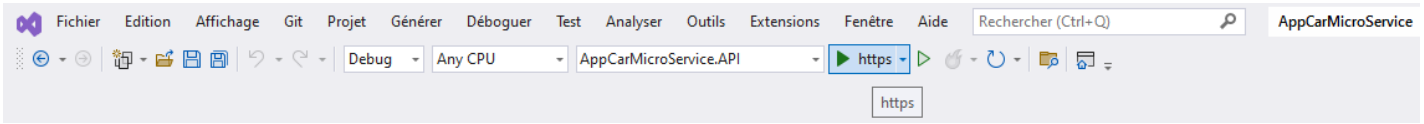
1. *Version 1 : Micro Service Asp.Net Core sans Docker*

Recupérer le projet **AppCarMicroService**. Ce projet était développé sous .Net 7.0 avec une base de donnée SQL Server qui était installé sur votre poste en local, l'architecture de l'application sera :

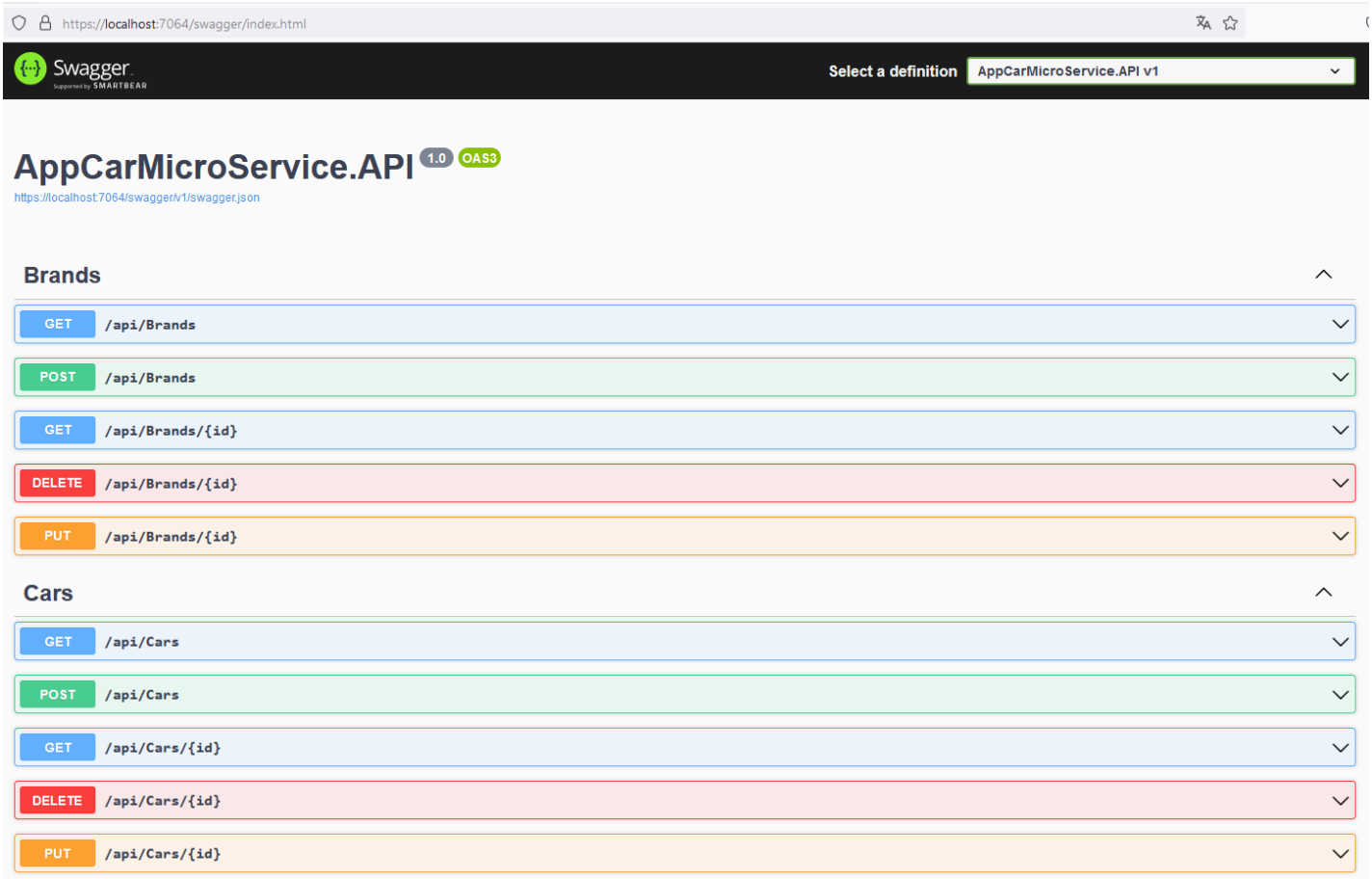


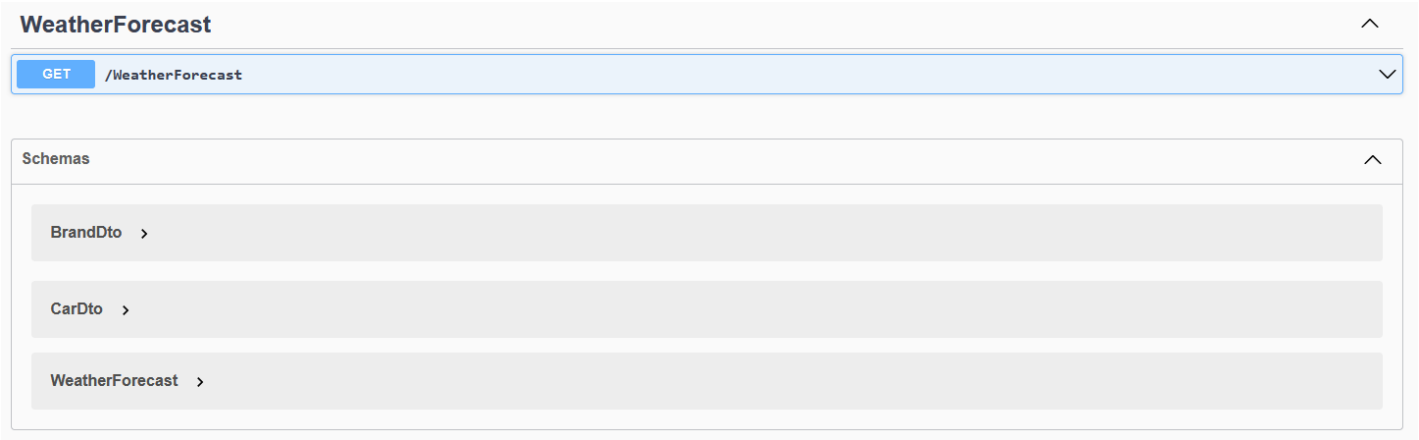




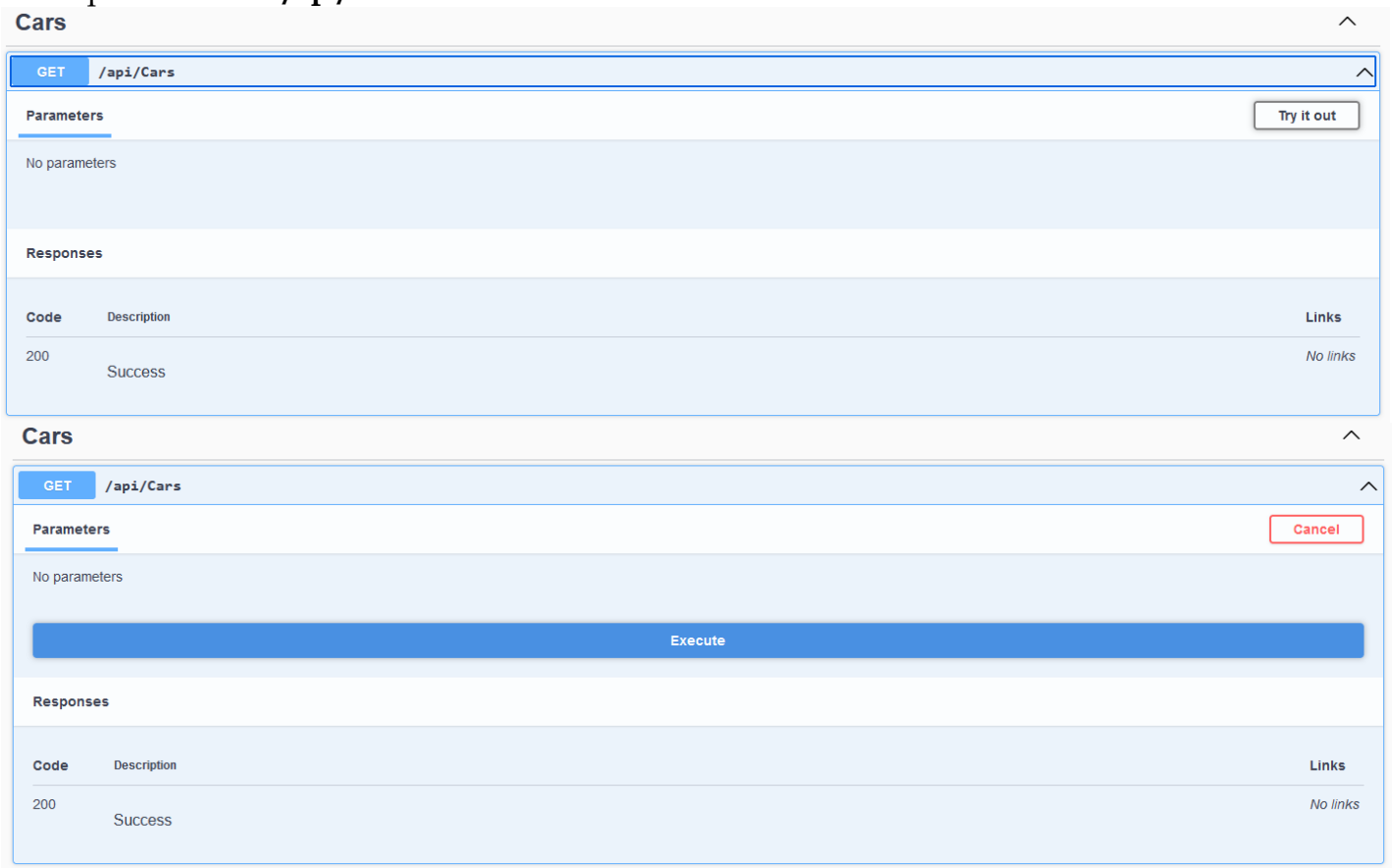


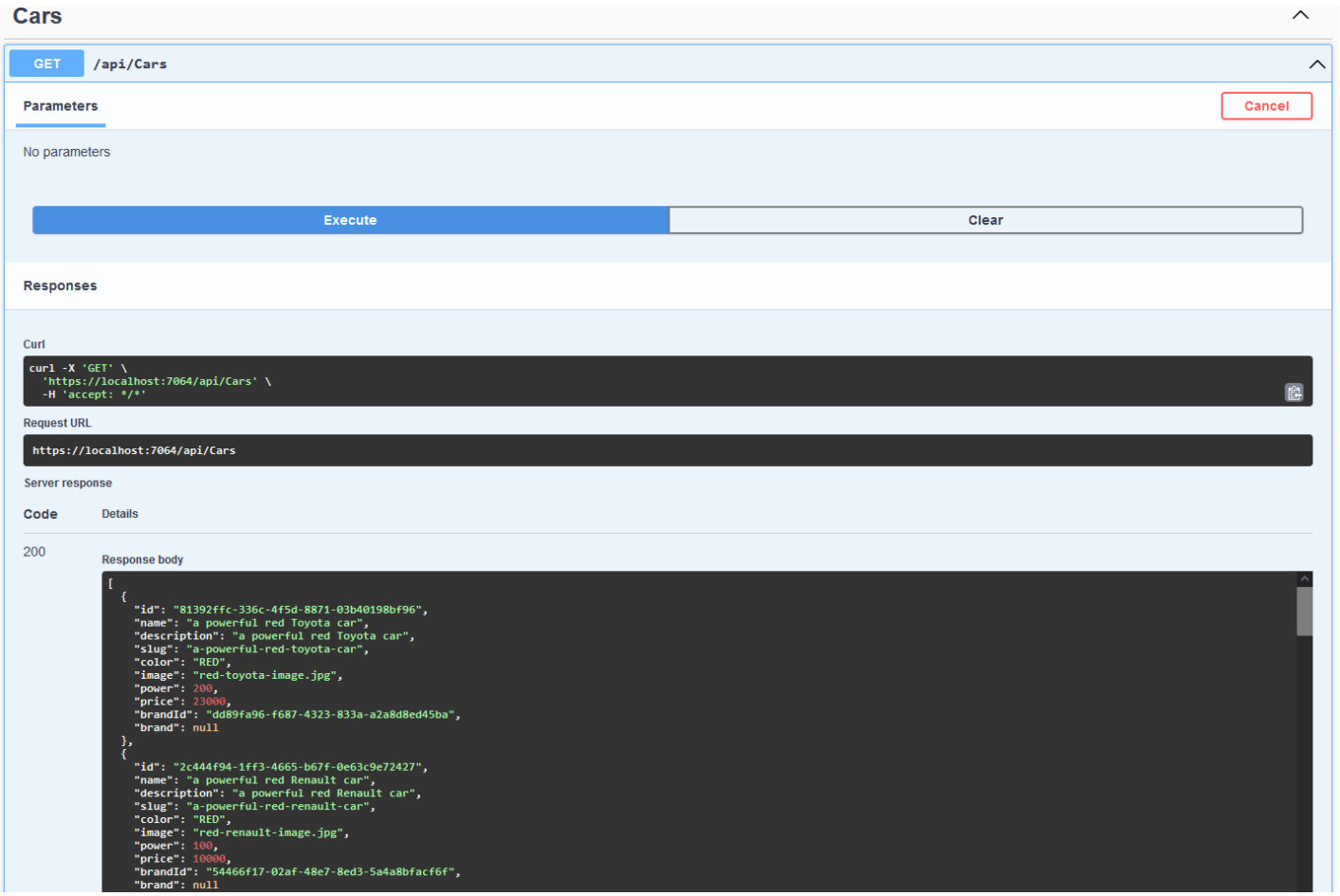
Appuyer sur le bouton « **https** » et l ’application lancer l’URL
https://localhost:7064/swagger/index.html





En cliquant sur **GET/api/Cars** on obtient :



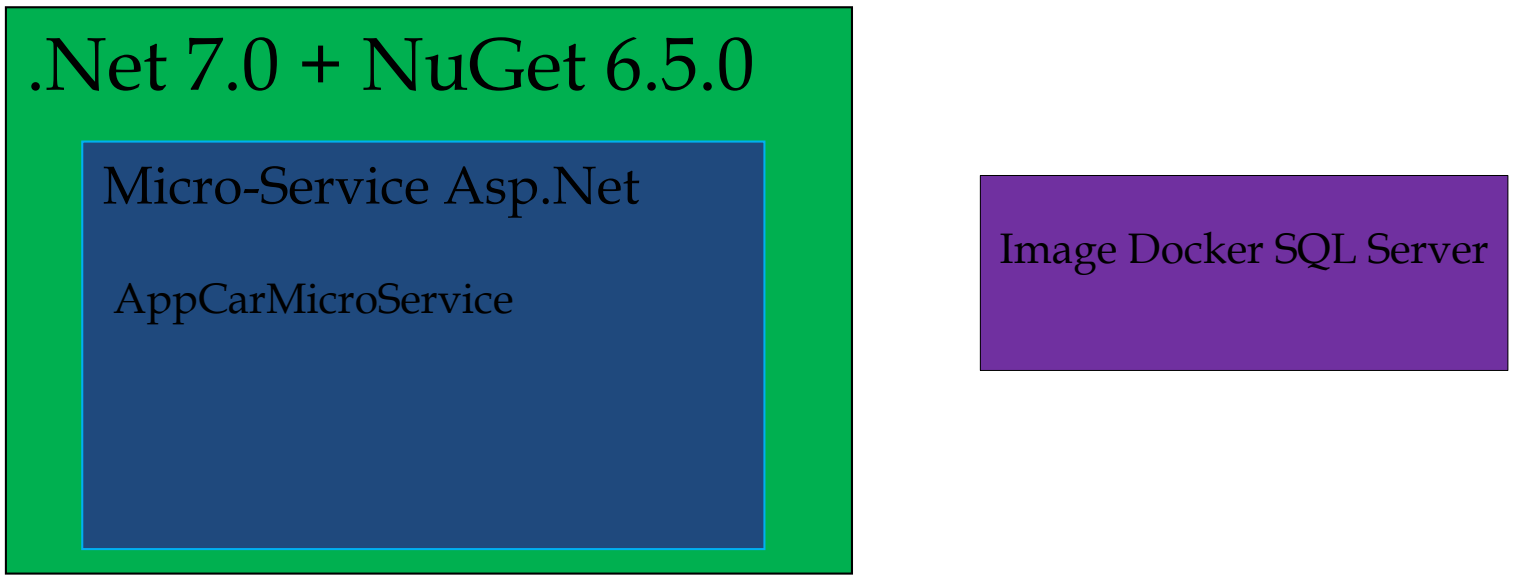


Nous allons faire une première amélioration dans cette application en utilisant une image docker de SQL SERVER.

2. *Version 2 : Micro Service Asp.Net Core avec une image docker SQL Server*

Nous allons améliorer notre architecture micro service en utilisant une image Docker SQL Server à la place d'une base de données installé phisiquement dans la machine local.

L'achitecture de l'application resemblera à :



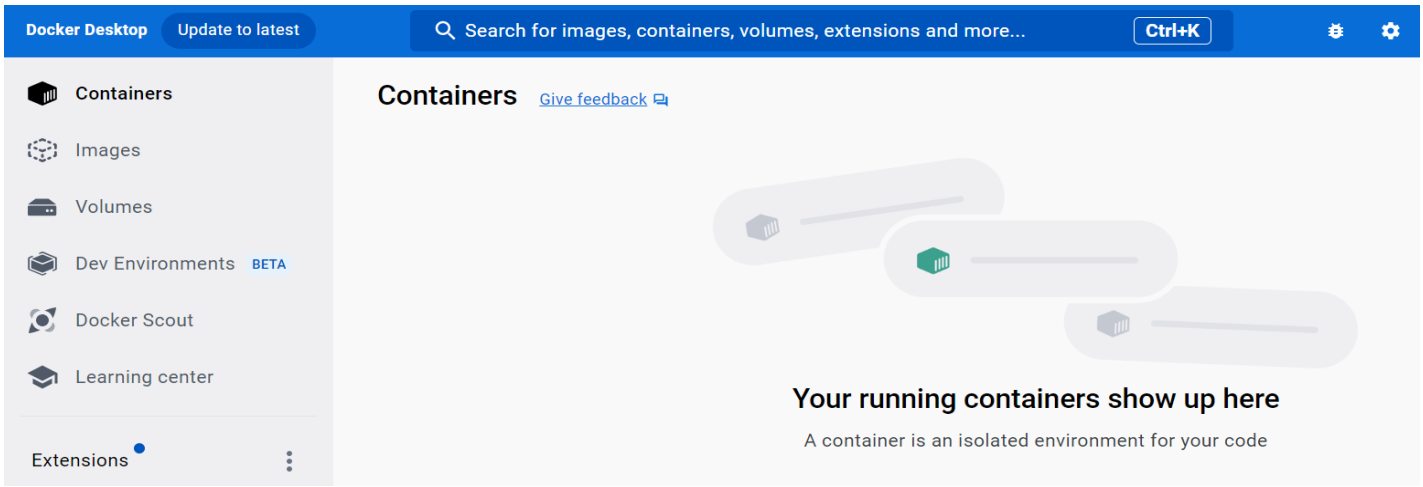
a. Le fichier init_my_data_base.sql

Créer le fichier **AppCarMicroService/init/init_my_data_base.sql** dont le contenu sera :

```
/* Base de données: CarMicroServiceDb*/
USE CarMicroServiceDb;
```

b. Commande Docker pour une image SQL Server

Verifier que vous n'avez pas de container Docker **SQL Server** qui est en cours d'éxécution si tel est le cas il faut la supprimer.

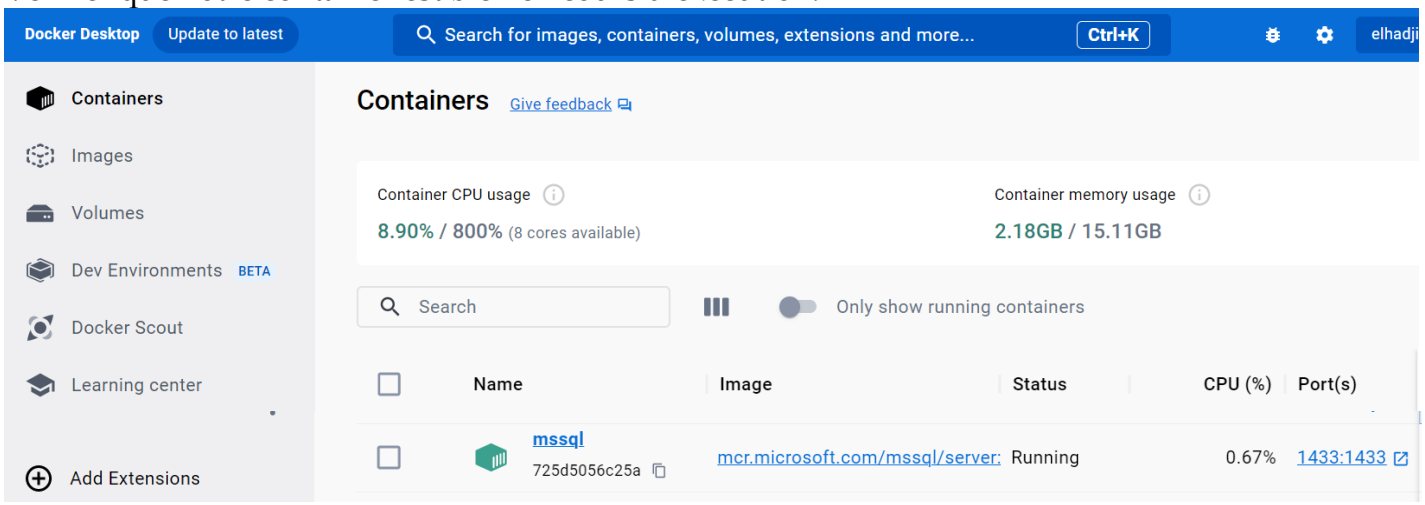


Nous allons maintenant créer la base de données **SQL Server** avec un container Docker avec la commande ci-dessous :

docker run -d --name mssql --hostname mssql --env ACCEPT_EULA=Y --env MSSQL_PID=Express --env "MSSQL_SA_PASSWORD=yourStrong(!)Password" --env "SA_PASSWORD=yourStrong(!)Password" --volume mssql:/var/opt/mssql --publish 1433:1433 mcr.microsoft.com/mssql/server:2022-latest

```
C:\Users\elhad\Desktop\AutoEntrepreneur\Formations\DOTNET\ASP.NET\Projects\AppCarMicroService>docker run -d --name mssql --hostname mssql --env ACCEPT_EULA=Y --env MSSQL_PI
D=Express --env "MSSQL_SA_PASSWORD=yourStrong(!)Password" --env "SA_PASSWORD=yourStrong(!)Password" --volume mssql:/var/opt/mssql --publish 1433:1433 mcr.microsoft.com/mssq
l/server:2022-latest
Unable to find image 'mcr.microsoft.com/mssql/server:2022-latest' locally
2022-latest: Pulling from mssql/server
e7945123d2a2: Pull complete
18a53d1b3bd7: Pull complete
d2a9a15297bf: Pull complete
Digest: sha256:c1aa8afe9b06eab64c9774a4802dcd032205d1be785b1fd51e1c0151e7586b74
Status: Downloaded newer image for mcr.microsoft.com/mssql/server:2022-latest
725d5056c25a73cb8a89e736a40188cfba5bf602771fb4e4406e081e78f292c7
```

Verifier que votre container est bien en cours d'execution.



Aller voir votre SGBD concretement avec la commande :

docker exec -it mssql /opt/mssql-tools18/bin/sqlcmd -U sa -P "yourStrong(!)Password" -C

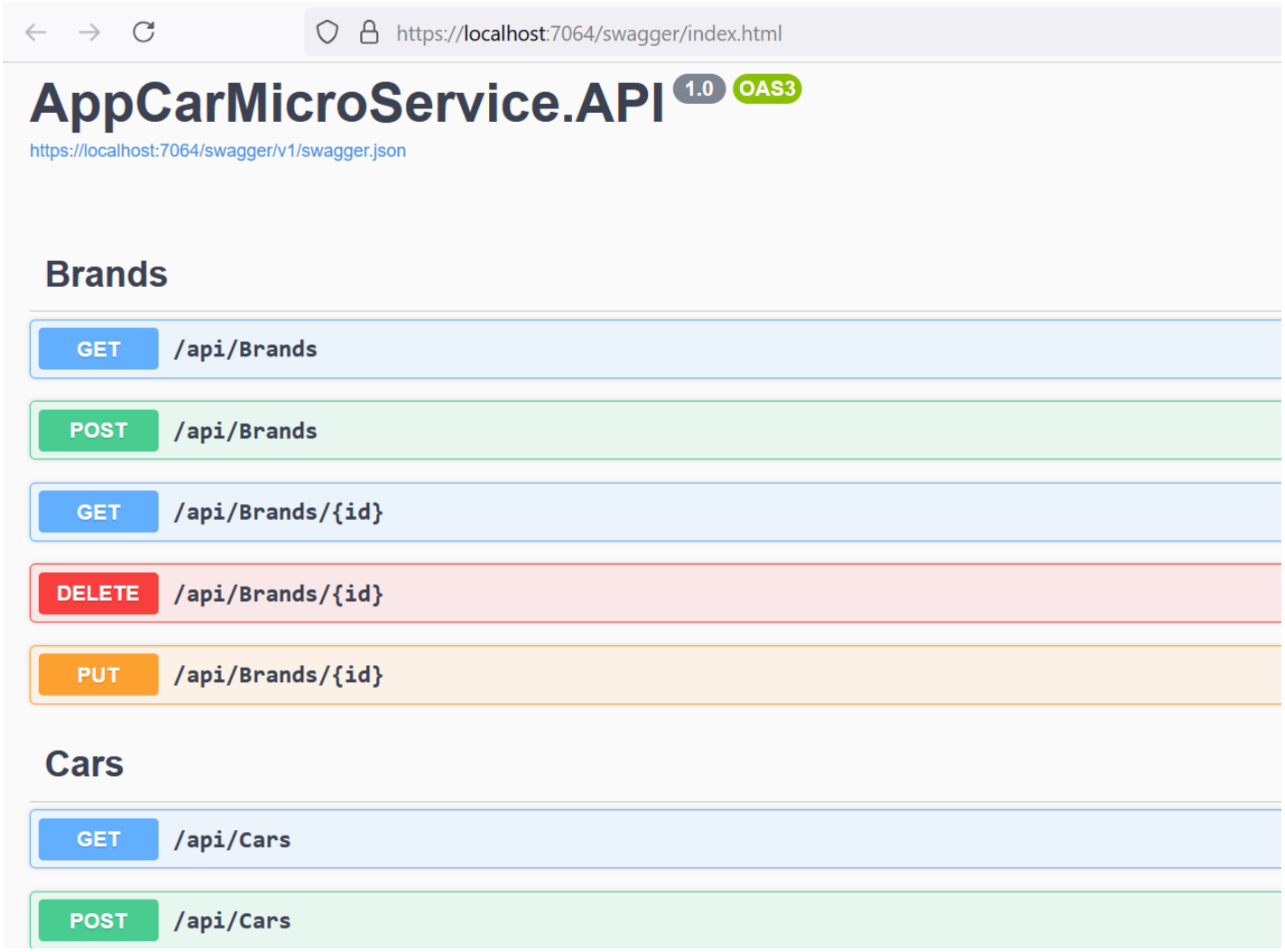
Une fois sur le Prompt 1> taper à la suite :
SELECT Name from sys.databases;
Go

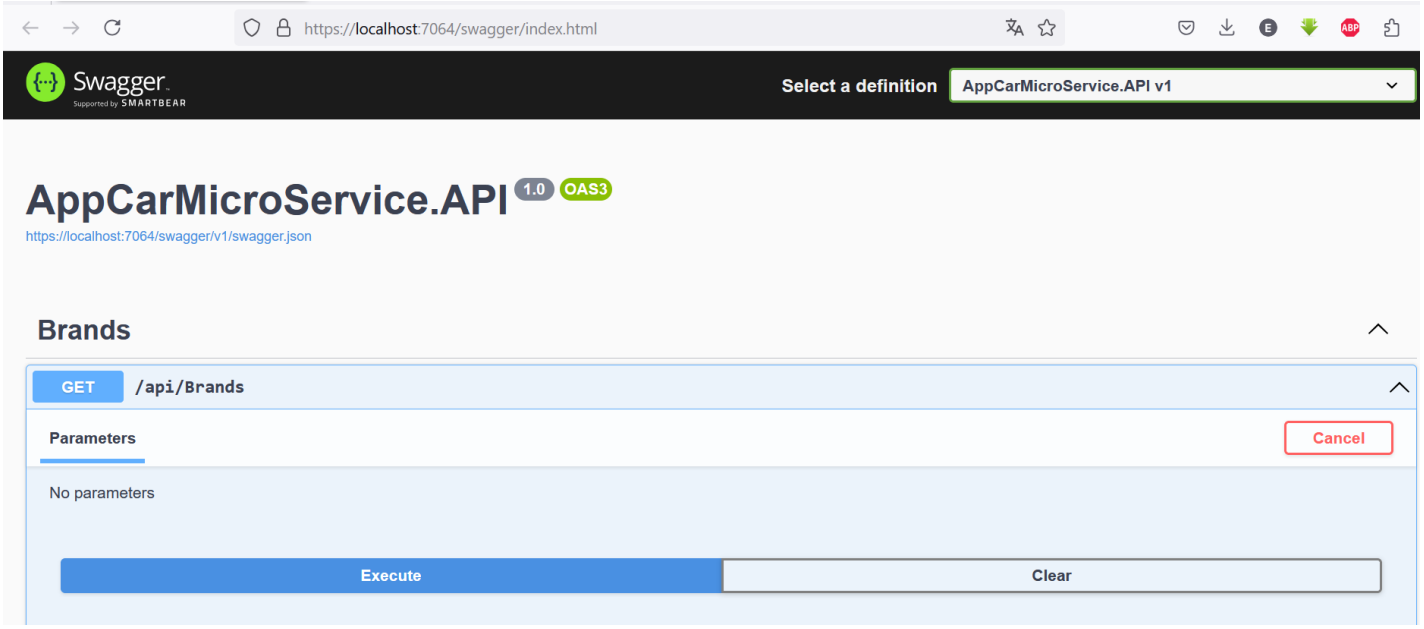
```
C:\Users\elhad>docker exec -it mssql /opt/mssql-tools18/bin/sqlcmd -U sa -P "yourStrong(!)Password" -C
1> SELECT Name from sys.databases;
2> Go
Name

--------------------------------------------------------------------------------------------------------------------------------
master

tempdb

model

msdb
```

Le fichier **appsettings.json** devient :

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "CarMicroServiceDbConnectionString": "Server=localhost,1433;Database=CarMicroServiceDb;User
Id=SA;Password=yourStrong(!)Password;Encrypt=True;TrustServerCertificate=True"
  },
  "Serilog": {
    "MinimumLevel": {
      "Default": "Information",
      "Override": {
        "Default": "Information",
        "Microsoft": "Warning",
        "Microsoft.Hosting.Lifetime": "Information"
      }
    },
    "WriteTo": [
      {
        "Name": "Console",
        "Args": {
          "outputTemplate": "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Level}] ({SourceContext}.{Method})
{Message}{NewLine}{Exception}"
        }
      },
      {
        "Name": "File",
        "Args": {
          "path": "Logs/API_Car_Services_Log.txt",
          "outputTemplate": "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Level}] ({SourceContext}.{Method})
{Message}{NewLine}{Exception}",
          "rollingInterval": "Hour"
        }
      }
    ]
  }
}
```

Le lancement de l'application donne sur https://localhost:7064/swagger/index.html





Curl

```
curl -X 'GET' \
  'https://localhost:7064/api/Brands' \
  -H 'accept: */*'
```

Request URL

```
https://localhost:7064/api/Brands
```

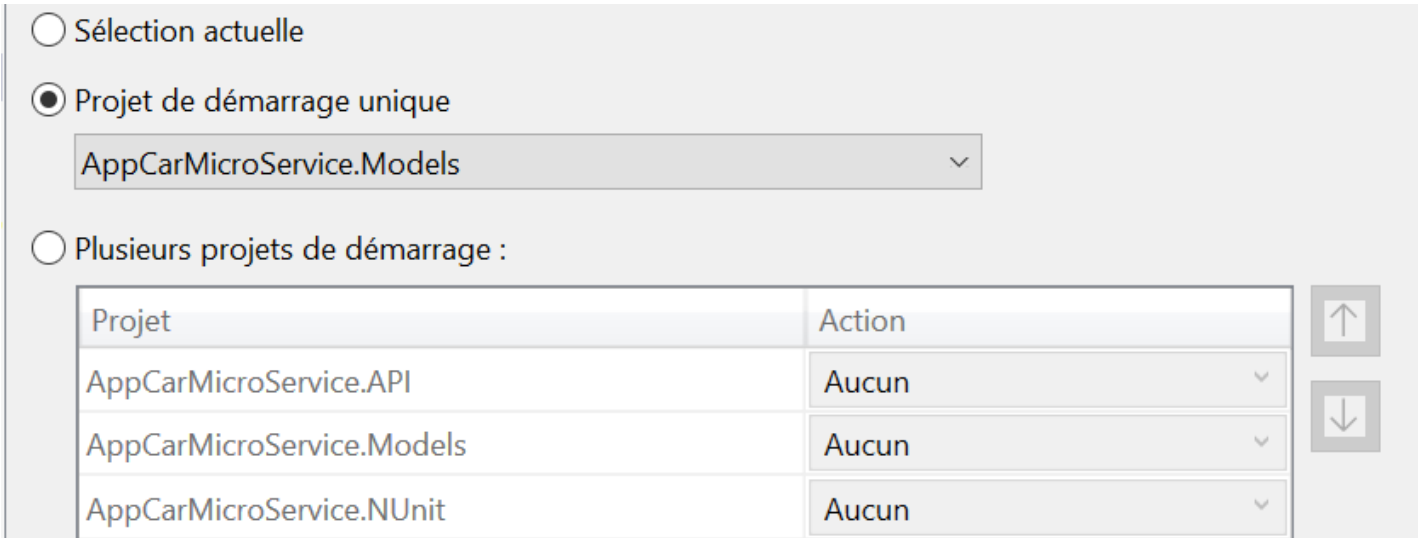
Server response

Code	Details
500 *Undocumented*	Error: response status is 500

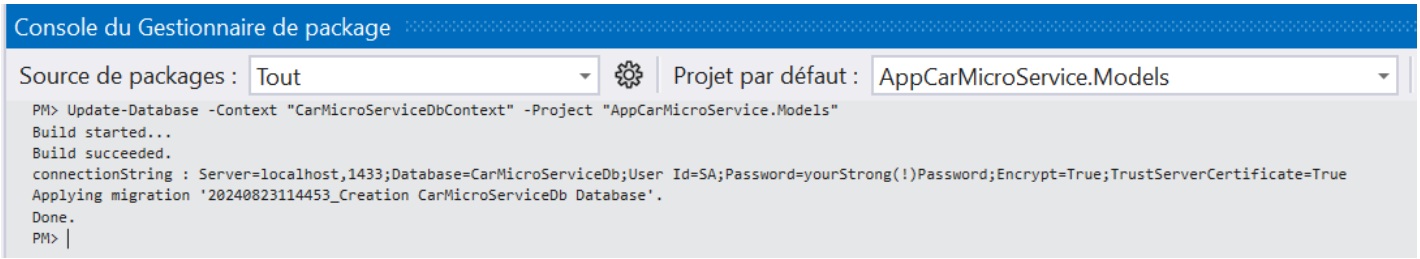
Response body

```
System.InvalidOperationException: An exception has been raised that is likely due to a transient failure. Consider enabling transient error resiliency by adding
'EnableRetryOnFailure' to the 'UseSqlServer' call.
 ---> Microsoft.Data.SqlClient.SqlException (0x80131904): Cannot open database "CarMicroServiceDb" requested by the login. The login failed.
Login failed for user 'sa'.
   at Microsoft.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean breakConnection, Action`1 wrapCloseInAction)
   at Microsoft.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj, Boolean callerHasConnectionLock, Boolean asyncClose)
   at Microsoft.Data.SqlClient.TdsParser.TryRun(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopySimpleResultSet bulkCopyHandler
, TdsParserStateObject stateObj, Boolean& dataReady)
   at Microsoft.Data.SqlClient.TdsParser.Run(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopySimpleResultSet bulkCopyHandler, T
dsParserStateObject stateObj)
   at Microsoft.Data.SqlClient.SqlInternalConnectionTds.CompleteLogin(Boolean enlistOK)
   at Microsoft.Data.SqlClient.SqlInternalConnectionTds.AttemptOneLogin(ServerInfo serverInfo, String newPassword, SecureString newSecurePassword, Boolean ignore
SniOpenTimeout, TimeoutTimer timeout, Boolean withFailover)
   at Microsoft.Data.SqlClient.SqlInternalConnectionTds.LoginNoFailover(ServerInfo serverInfo, String newPassword, SecureString newSecurePassword, Boolean redire
ctedUserInstance, SqlConnectionString connectionOptions, SqlCredential credential, TimeoutTimer timeout)
   at Microsoft.Data.SqlClient.SqlInternalConnectionTds.OpenLoginEnlist(TimeoutTimer timeout, SqlConnectionString connectionOptions, SqlCredential credential, St
ring newPassword, SecureString newSecurePassword, Boolean redirectedUserInstance)
   at Microsoft.Data.SqlClient.SqlInternalConnectionTds..ctor(DbConnectionPoolIdentity identity, SqlConnectionString connectionOptions, SqlCredential credential,
```

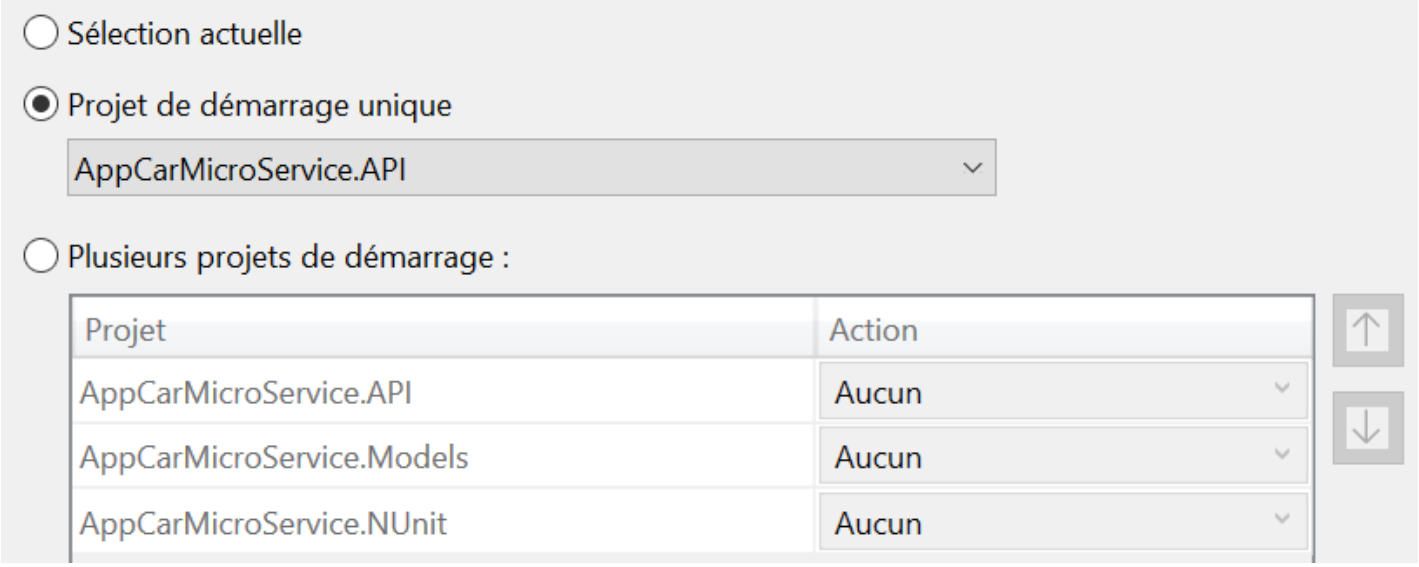
Il faut donc jouer la migration :

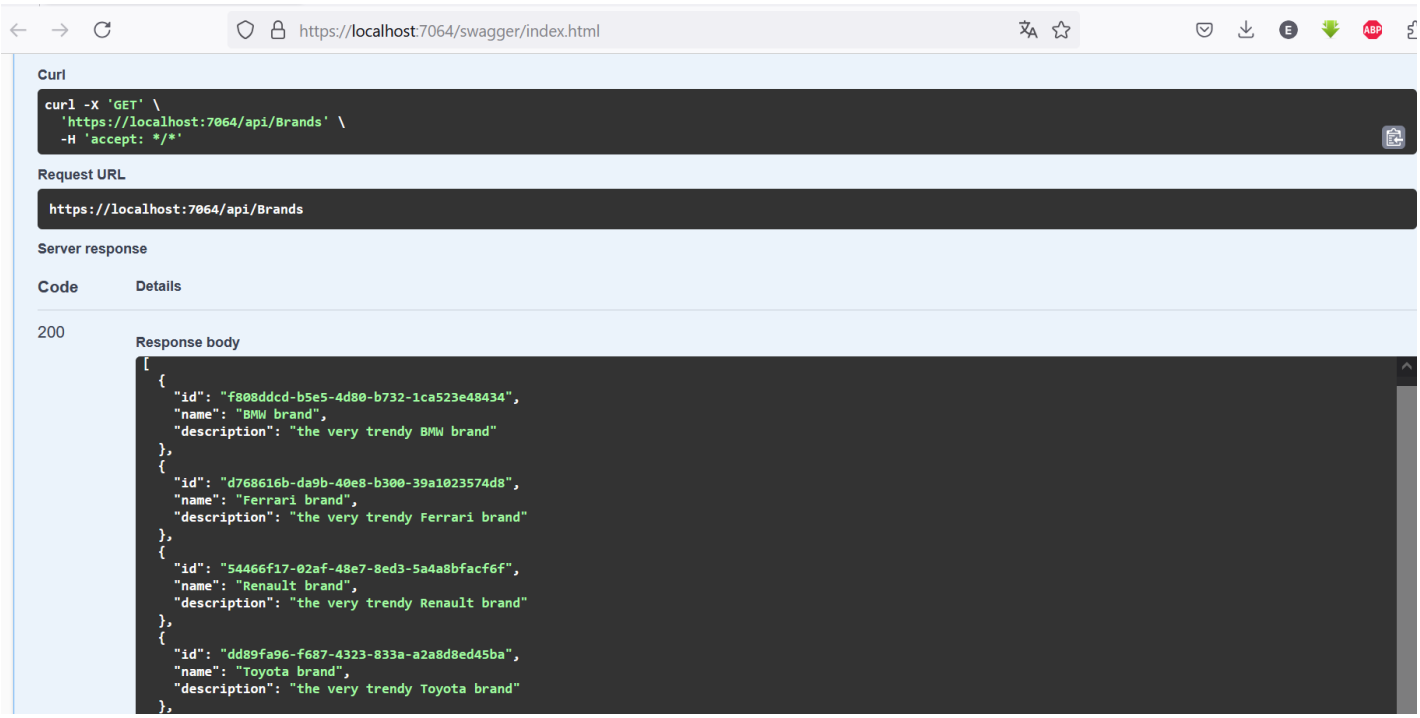


Update-Database -Context "CarMicroServiceDbContext" -Project "AppCarMicroService.Models"



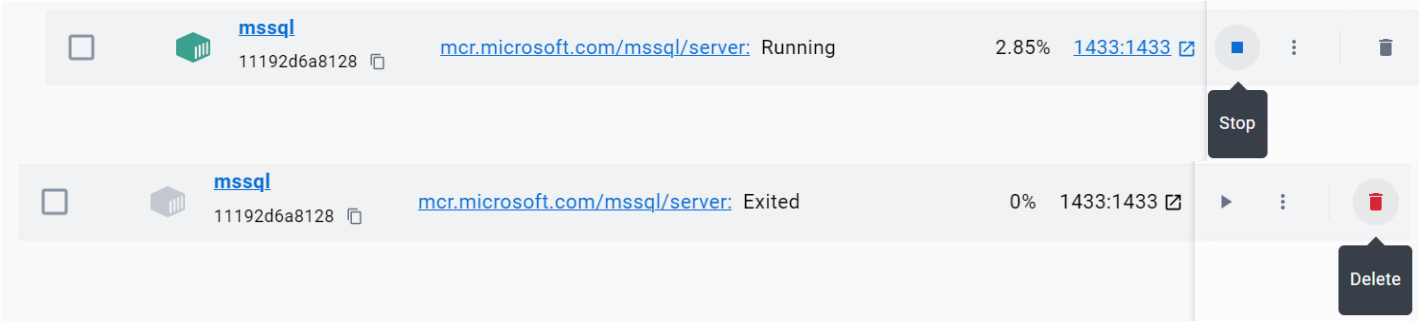
Relancer l'application pour obtenir :

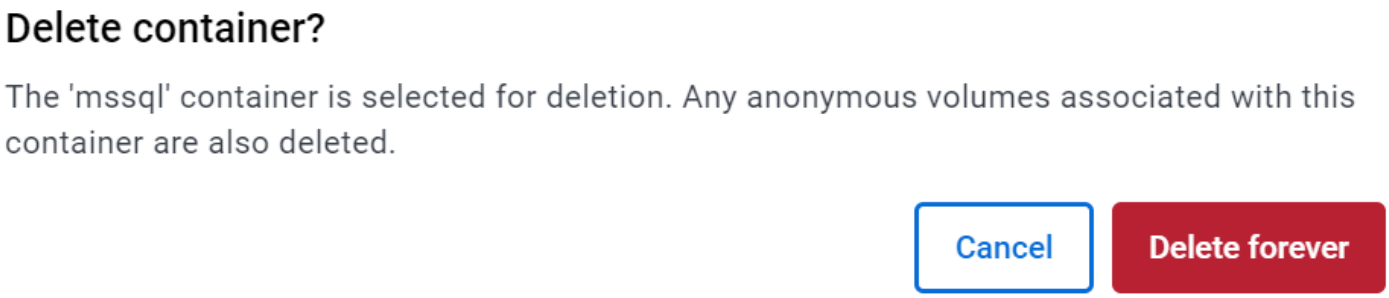




c. Le fichier docker-compose.yaml pour une image SQL Server

Arreter et supprimer le précédent conteneur :





Créer le fichier **sql-server-database/docker-compose.yaml** dont le contenu sera :

```yaml
# docker-compose file
services:

 mssql:
  hostname: mssql
  container_name: mssql
  image: mcr.microsoft.com/mssql/server:2022-latest
  ports:
    - "1433:1433"
  environment:
    - ACCEPT_EULA=Y
    - MSSQL_SA_PASSWORD=yourStrong(!)Password
  volumes:
     - ./data/:/var/opt/mssql/data/
     - ./log/:/var/opt/mssql/log/
  networks:
    - my_network

networks:
  my_network:
    driver: bridge
```

Lancer les commandes :

cd Projects\AppCarMicroService\sql-server-database

docker-compose build

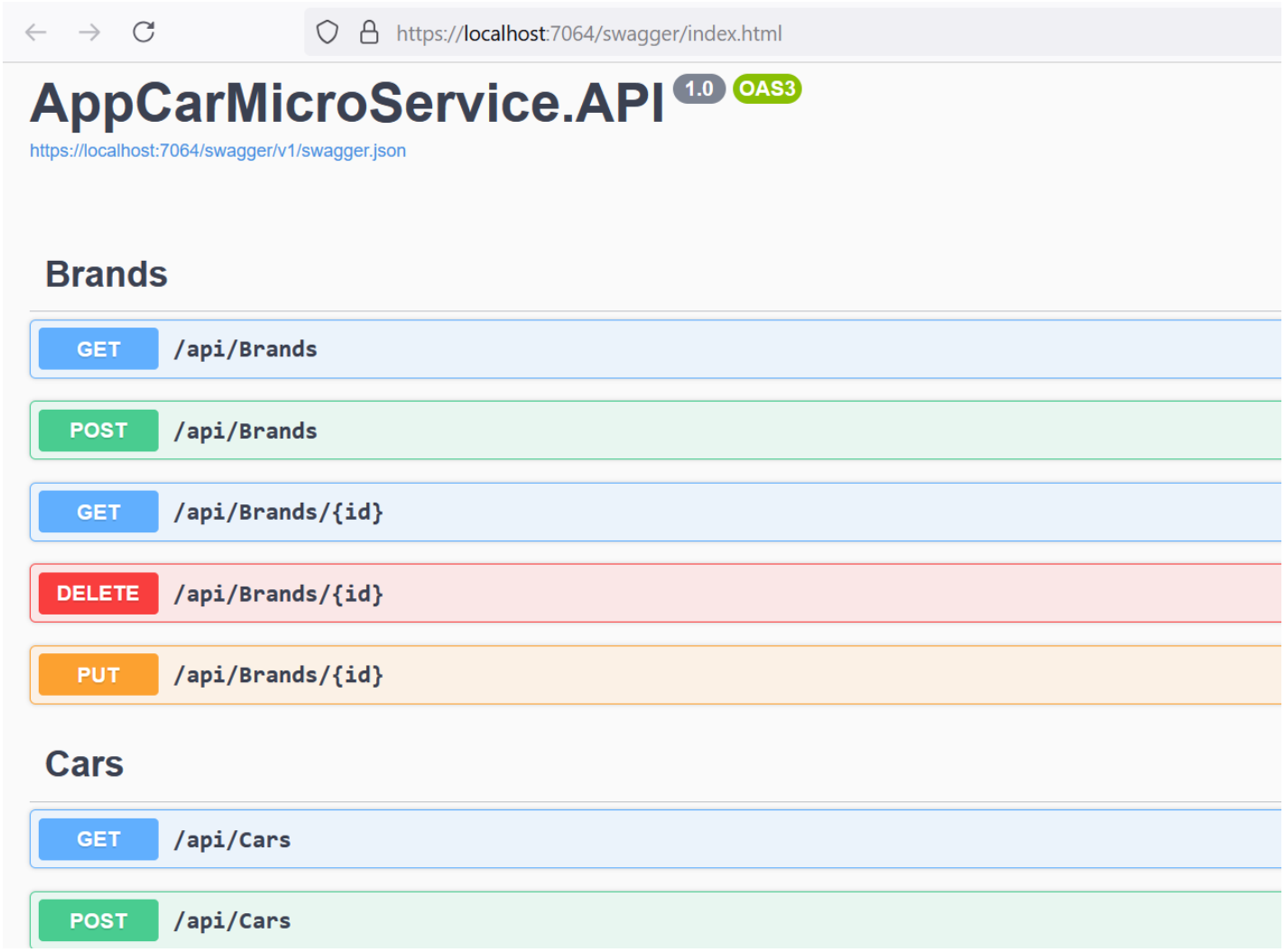
docker-compose up

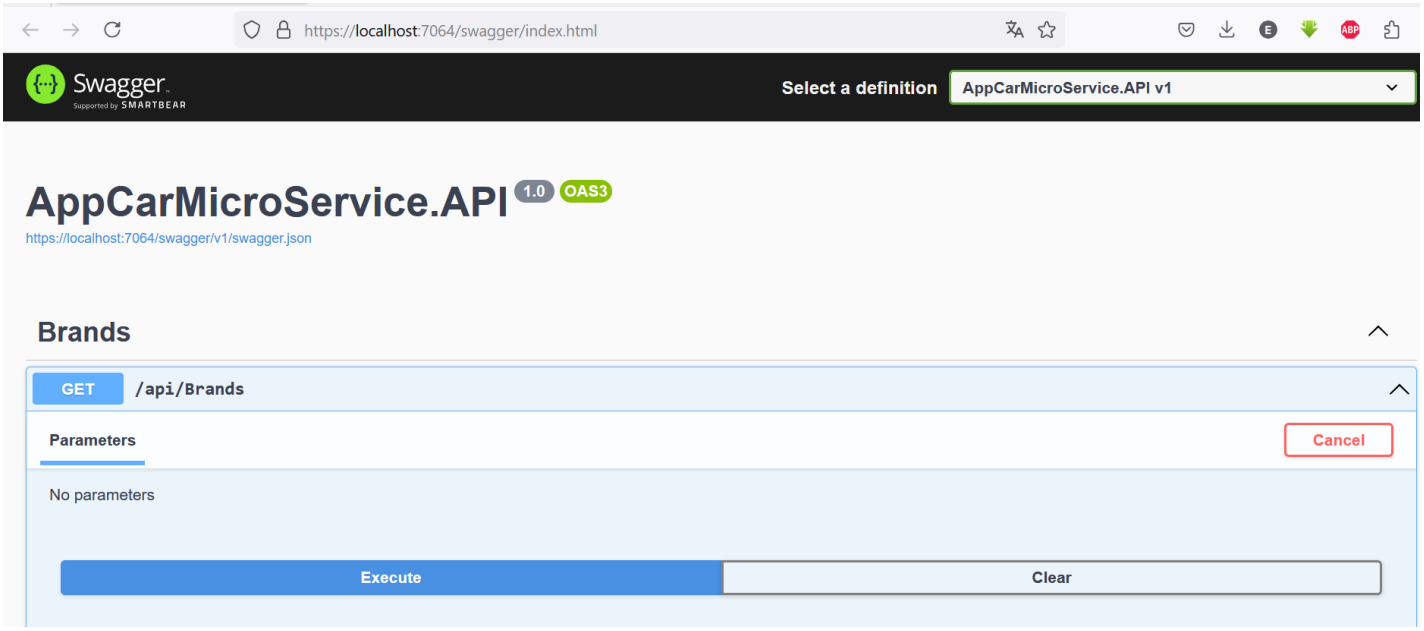
```
C:\Users\elhad\Desktop\AutoEntrepreneur\Formations\DOTNET\ASP.NET\Projects\AppCarMicroService\sql-server-database>docker-compose build
[+] Building 0.0s (0/0)                                                                                                                                 docker:default

C:\Users\elhad\Desktop\AutoEntrepreneur\Formations\DOTNET\ASP.NET\Projects\AppCarMicroService\sql-server-database>docker-compose up
[+] Building 0.0s (0/0)                                                                                                                                 docker:default
[+] Running 1/1
 ⠿ Container mssql  Created                                                                                                                                      0.1s
Attaching to mssql
mssql  | SQL Server 2022 will run as non-root by default.
mssql  | This container is running as user mssql.
mssql  | To learn more visit https://go.microsoft.com/fwlink/?linkid=2099216.
mssql  | 2024-08-26 04:03:13.00 Server      Setup step is copying system data file 'C:\templatedata\master.mdf' to '/var/opt/mssql/data/master.mdf'.
2024-08-26 04:03:13.20 Server      Did not find an existing master data file /var/opt/mssql/data/master.mdf, copying the missing default master and other system database fi
les. If you have moved the database location, but not moved the database files, startup may fail. To repair: shutdown SQL Server, move the master database to configured loc
ation, and restart.
2024-08-26 04:03:13.21 Server      Setup step is copying system data file 'C:\templatedata\mastlog.ldf' to '/var/opt/mssql/data/mastlog.ldf'.
2024-08-26 04:03:13.38 Server      Setup step is copying system data file 'C:\templatedata\model.mdf' to '/var/opt/mssql/data/model.mdf'.
2024-08-26 04:03:13.58 Server      Setup step is copying system data file 'C:\templatedata\modellog.ldf' to '/var/opt/mssql/data/modellog.ldf'.
2024-08-26 04:03:13.80 Server      Setup step is copying system data file 'C:\templatedata\msdbdata.mdf' to '/var/opt/mssql/data/msdbdata.mdf'.
2024-08-26 04:03:14.01 Server      Setup step is copying system data file 'C:\templatedata\msdblog.ldf' to '/var/opt/mssql/data/msdblog.ldf'.
2024-08-26 04:03:14.21 Server      Setup step is FORCE copying system data file 'C:\templatedata\model_replicatedmaster.mdf' to '/var/opt/mssql/data/model_replicatedmaster.
mdf'.
2024-08-26 04:03:14.42 Server      Setup step is FORCE copying system data file 'C:\templatedata\model_replicatedmaster.ldf' to '/var/opt/mssql/data/model_replicatedmaster.
ldf'.
2024-08-26 04:03:14.69 Server      Setup step is FORCE copying system data file 'C:\templatedata\model_msdbdata.mdf' to '/var/opt/mssql/data/model_msdbdata.mdf'.
2024-08-26 04:03:14.92 Server      Setup step is FORCE copying system data file 'C:\templatedata\model_msdblog.ldf' to '/var/opt/mssql/data/model_msdblog.ldf'.
2024-08-26 04:03:15.16 Server      Microsoft SQL Server 2022 (RTM-CU14) (KB5038325) - 16.0.4135.4 (X64)
mssql  |Jul 10 2024 14:09:09
mssql  |Copyright (C) 2022 Microsoft Corporation
mssql  |Developer Edition (64-bit) on Linux (Ubuntu 22.04.4 LTS) <X64>
2024-08-26 04:03:15.17 Server      UTC adjustment: 0:00
2024-08-26 04:03:15.17 Server      (c) Microsoft Corporation.
2024-08-26 04:03:15.17 Server      All rights reserved.
2024-08-26 04:03:15.17 Server      Server process ID is 468.
2024-08-26 04:03:15.17 Server      Logging SQL Server messages in file '/var/opt/mssql/log/errorlog'.
2024-08-26 04:03:15.17 Server      Registry startup parameters:
```

Le fichier **appsettings.json** devient :

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "CarMicroServiceDbConnectionString": "Server=localhost,1433;Database=CarMicroServiceDb;User
Id=SA;Password=yourStrong(!)Password;Encrypt=True;TrustServerCertificate=True"
  },
  "Serilog": {
    "MinimumLevel": {
      "Default": "Information",
      "Override": {
        "Default": "Information",
        "Microsoft": "Warning",
        "Microsoft.Hosting.Lifetime": "Information"
      }
    },
    "WriteTo": [
      {
        "Name": "Console",
        "Args": {
          "outputTemplate": "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Level}] ({SourceContext}.{Method})
{Message}{NewLine}{Exception}"
        }
      },
      {
        "Name": "File",
        "Args": {
          "path": "Logs/API_Car_Services_Log.txt",
          "outputTemplate": "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Level}] ({SourceContext}.{Method})
{Message}{NewLine}{Exception}",
          "rollingInterval": "Hour"
        }
      }
    ]
  }
}
```

Le lancement de l'application donne sur https://localhost:7064/swagger/index.html





Curl

```
curl -X 'GET' \
  'https://localhost:7064/api/Brands' \
  -H 'accept: */*'
```

Request URL

```
https://localhost:7064/api/Brands
```

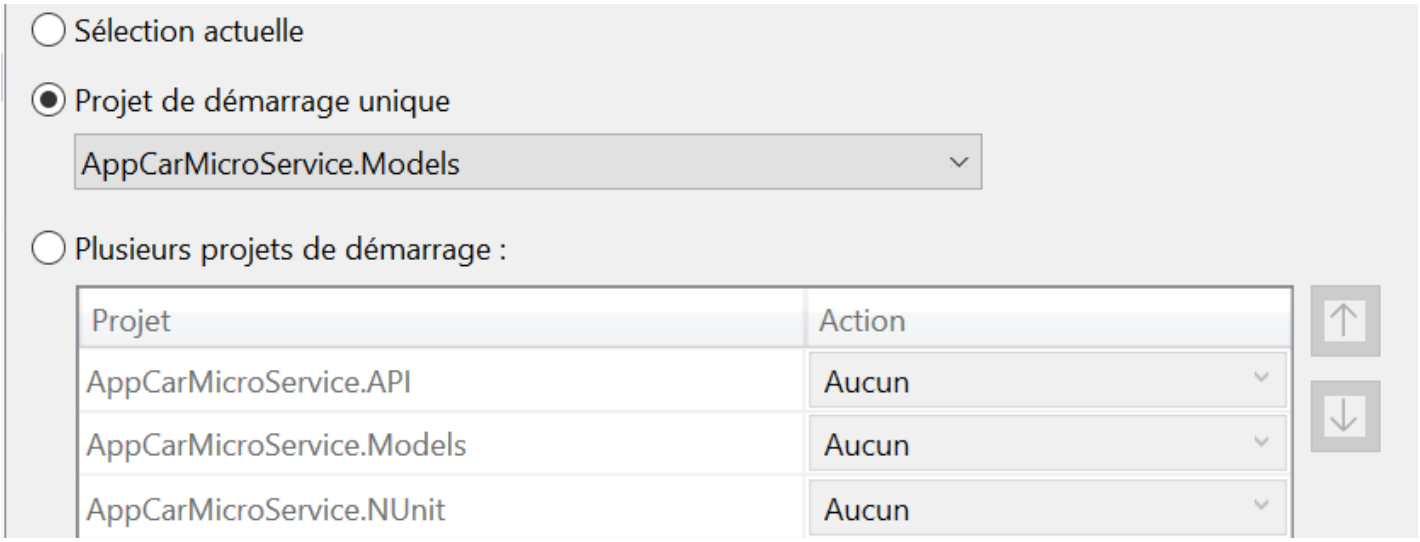
Server response

Code	Details
500 *Undocumented*	Error: response status is 500

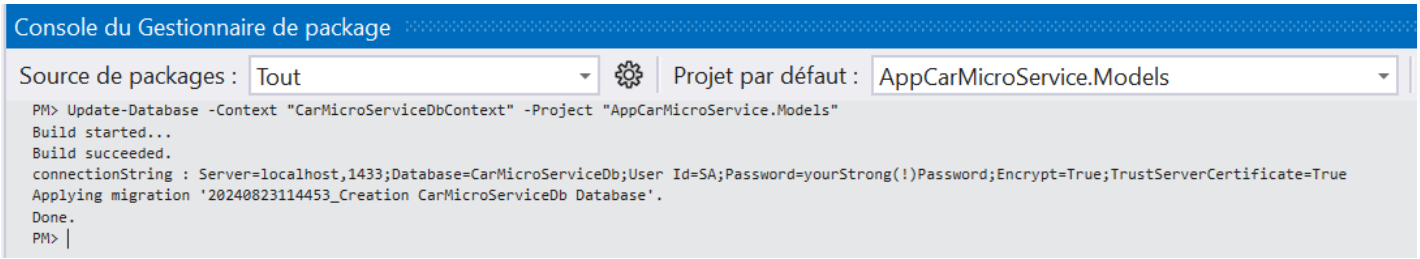
Response body

```
System.InvalidOperationException: An exception has been raised that is likely due to a transient failure. Consider enabling transient error resiliency by adding
'EnableRetryOnFailure' to the 'UseSqlServer' call.
 ---> Microsoft.Data.SqlClient.SqlException (0x80131904): Cannot open database "CarMicroServiceDb" requested by the login. The login failed.
Login failed for user 'sa'.
   at Microsoft.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean breakConnection, Action`1 wrapCloseInAction)
   at Microsoft.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj, Boolean callerHasConnectionLock, Boolean asyncClose)
   at Microsoft.Data.SqlClient.TdsParser.TryRun(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopySimpleResultSet bulkCopyHandler
, TdsParserStateObject stateObj, Boolean& dataReady)
   at Microsoft.Data.SqlClient.TdsParser.Run(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopySimpleResultSet bulkCopyHandler, T
dsParserStateObject stateObj)
   at Microsoft.Data.SqlClient.SqlInternalConnectionTds.CompleteLogin(Boolean enlistOK)
   at Microsoft.Data.SqlClient.SqlInternalConnectionTds.AttemptOneLogin(ServerInfo serverInfo, String newPassword, SecureString newSecurePassword, Boolean ignore
SniOpenTimeout, TimeoutTimer timeout, Boolean withFailover)
   at Microsoft.Data.SqlClient.SqlInternalConnectionTds.LoginNoFailover(ServerInfo serverInfo, String newPassword, SecureString newSecurePassword, Boolean redire
ctedUserInstance, SqlConnectionString connectionOptions, SqlCredential credential, TimeoutTimer timeout)
   at Microsoft.Data.SqlClient.SqlInternalConnectionTds.OpenLoginEnlist(TimeoutTimer timeout, SqlConnectionString connectionOptions, SqlCredential credential, St
ring newPassword, SecureString newSecurePassword, Boolean redirectedUserInstance)
   at Microsoft.Data.SqlClient.SqlInternalConnectionTds..ctor(DbConnectionPoolIdentity identity, SqlConnectionString connectionOptions, SqlCredential credential,
```

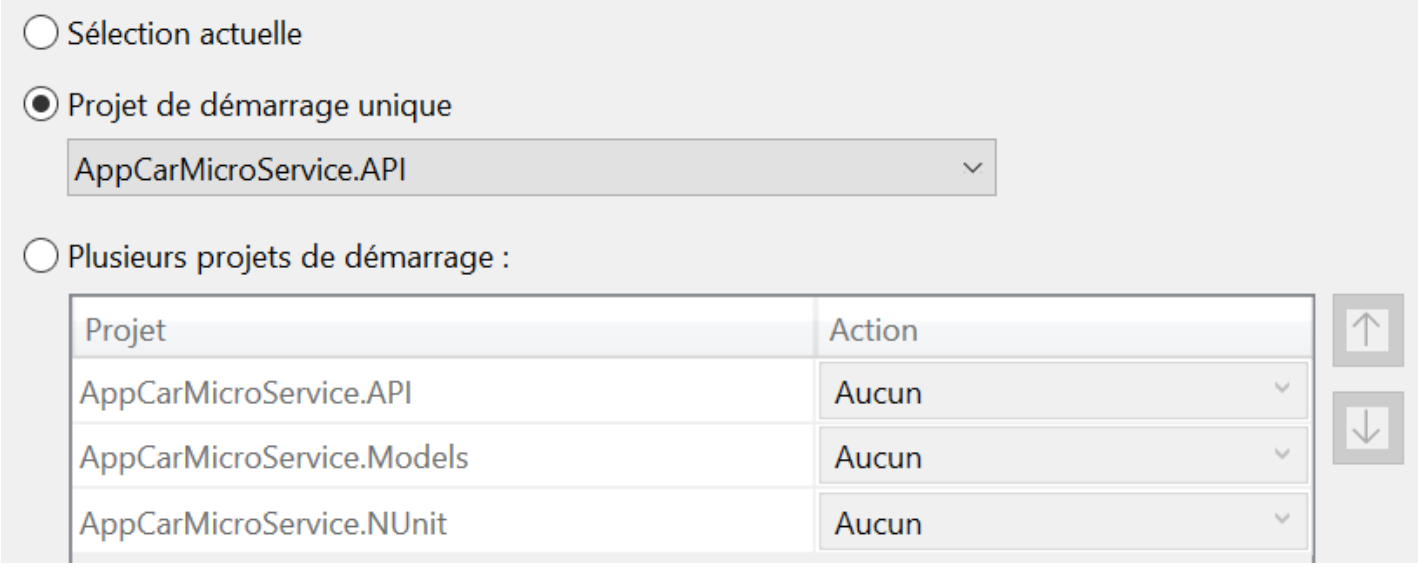
Il faut donc jouer la migration :

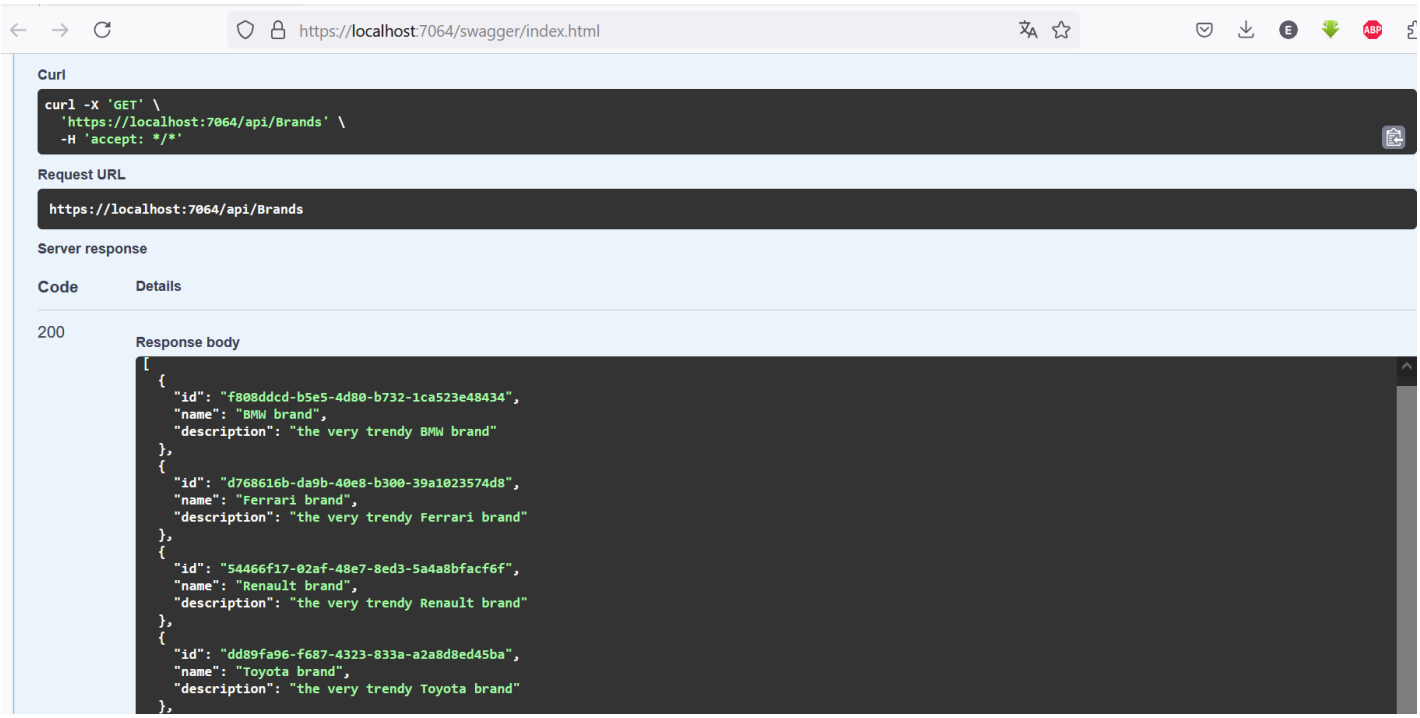


Update-Database -Context "CarMicroServiceDbContext" -Project "AppCarMicroService.Models"



Relancer l'application pour obtenir :

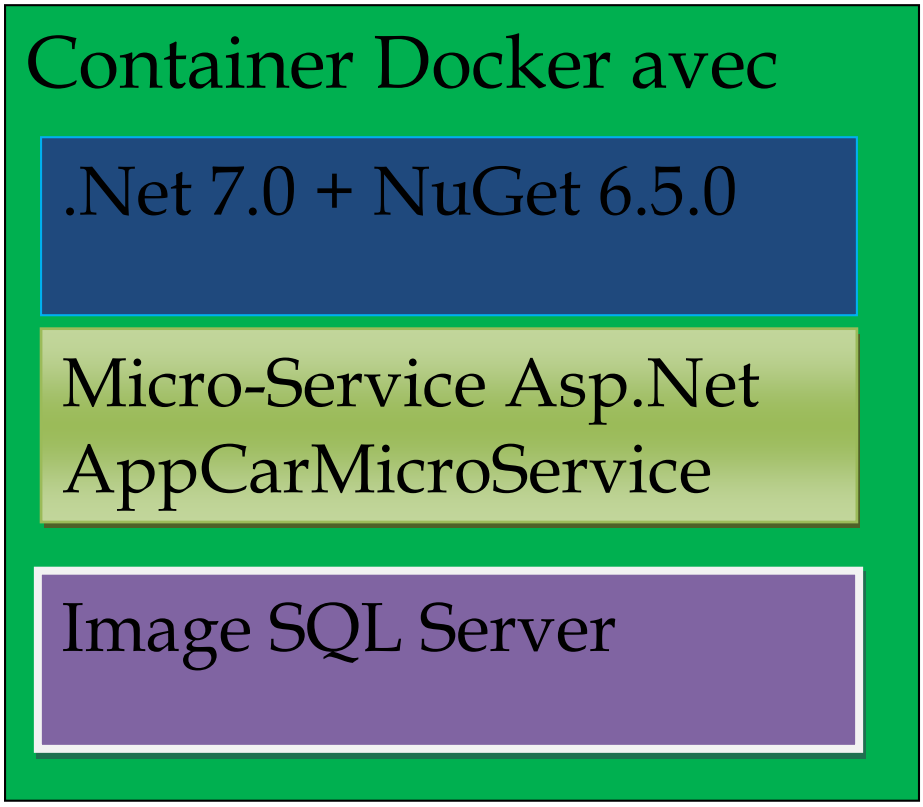




3. *Version 3 : Micro Service Asp.Net Core avec une image docker SQL Server*

Nous allons améliorer notre architecture micro service en utilisant une image docker de l'application.

L'achitecture de l'application resemblera à :



Le fichier **Projects/AppCarMicroService/Dockerfile** aura comme contenu :

```
# Use the official .NET Core SDK as a parent image
FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
WORKDIR /app


# Copy the project file and restore any dependencies (use .csproj for the project name)

WORKDIR /src
COPY . .
RUN dotnet restore AppCarMicroService.API/AppCarMicroService.API.csproj
WORKDIR "/src/AppCarMicroService.API"
RUN dotnet build "AppCarMicroService.API.csproj" -c Release -o /app

FROM build AS publish
WORKDIR "/src/AppCarMicroService.API"
RUN dotnet publish "AppCarMicroService.API.csproj" -c Release -o /app


# Build the runtime image
FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS runtime
# Expose the port your application will run on
EXPOSE 80
EXPOSE 443


FROM runtime AS final
WORKDIR /app
COPY --from=publish /app .

# Start Migrations

ENTRYPOINT dotnet ef database update --connection "Server=mssqldb,1433;Database=CarMicroServiceDb;User
Id=SA;Password=yourStrong(!)Password;Encrypt=True;TrustServerCertificate=True"

# Start the application
ENTRYPOINT ["dotnet", "AppCarMicroService.API.dll"]
```

```
# Use the official .NET Core SDK as a parent image
FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
WORKDIR /app
```

Commencer par l'image de base de votre conteneur Docker, spécifiée sous le nom mcr.microsoft.com/dotnet/sdk:7.0. Cette image utilise le SDK 7.0 officiel de .NET Core comme point de départ. Pour faire référence ultérieurement à cette étape, on lui donne l'alias « build ». Ensuite, définissez le répertoire de travail dans le conteneur sur /app. C'est là que vos fichiers d'application seront copiés et créés.

```
RUN dotnet publish "AppCarMicroService.API.csproj" -c Release -o /app
```

« RUN dotnet publish -c Release -o out » Cette commande restaure les dépendances du projet et tous les packages requis.

```
FROM runtime AS final
WORKDIR /app
COPY --from=publish /app .
```

FROM mcr.microsoft.com/dotnet/runtime:7.0 : cette section démarre une nouvelle phase à l'aide de l'image d'exécution. Cette image est petite et contient uniquement l'environnement d'exécution nécessaire pour exécuter les applications .NET Core, contrairement à l'image SDK, qui est utilisée pour les builds.

WORKDIR /app : définissez le répertoire de travail sur /app dans cette nouvelle section.

COPY — from=build /app/out . : copie l'application compilée du composant de build vers le composant d'exécution actuel. Cela garantit que seules les fonctionnalités nécessaires sont ajoutées à l'image d'exécution finale.

```
# Expose the port your application will run on
EXPOSE 80
EXPOSE 443
```

EXPOSE 80 : cette ligne spécifie que le conteneur exposera le port 80.

```
# Start the application
ENTRYPOINT ["dotnet", "AppCarMicroService.API.dll"]
```

ENTRYPOINT [“dotnet”, “AppCarMicroService.API.dll”] : spécifie la commande à exécuter lorsqu’un objet basé sur cette image est démarré. Dans ce cas, il crée votre application API Web .NET Core en appelant dotnet AppCarMicroService.API.dll.

Le fichier **Projects/AppCarMicroService/docker-compose.yaml** aura comme contenu :

```yaml
# docker-compose file
services:

 mssqldb:
  hostname: mssqldb
  container_name: mssqldb
  image: mcr.microsoft.com/mssql/server:2022-latest
  ports:
    - "1433:1433"
  environment:
    - ACCEPT_EULA=Y
    - MSSQL_SA_PASSWORD=yourStrong(!)Password
  volumes:
     - ./data/:/var/opt/mssql/data/
     - ./log/:/var/opt/mssql/log/
  networks:
    - car_micro_service_network

 myapi:
   hostname: myapi
   container_name: myapi
   build:
     context: .
     dockerfile: Dockerfile
   ports:
     - "5000:80"
     - "5001:443"
   depends_on:
     - mssqldb
   environment:
     - DB_HOST=host.docker.internal,1433
     - DB_NAME=CarMicroServiceDb
     - DB_SA_PASSWORD=yourStrong(!)Password
     - ASPNETCORE_ENVIRONMENT=Production
   networks:
    - car_micro_service_network

networks:
 car_micro_service_network:
   driver: bridge
```

Le fichier **appsettings.json** devient :

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "CarMicroServiceDbConnectionString": "Server=mssqldb,1433;Database=CarMicroServiceDb;User
Id=SA;Password=yourStrong(!)Password;Encrypt=True;TrustServerCertificate=True"
  },
  "DB_HOST": "mssqldb,1433",
  "DB_NAME": "CarMicroServiceDb",
  "DB_SA_PASSWORD": "yourStrong(!)Password",
  "ASPNETCORE_ENVIRONMENT": "Production",
  "Serilog": {
    "MinimumLevel": {
      "Default": "Information",
      "Override": {
        "Default": "Information",
        "Microsoft": "Warning",
        "Microsoft.Hosting.Lifetime": "Information"
      }
    },
    "WriteTo": [
      {
        "Name": "Console",
        "Args": {
          "outputTemplate": "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Level}] ({SourceContext}.{Method})
{Message}{NewLine}{Exception}"
        }
      },
      {
        "Name": "File",
        "Args": {
          "path": "Logs/API_Car_Micro_Services_Log.txt",
          "outputTemplate": "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Level}] ({SourceContext}.{Method})
{Message}{NewLine}{Exception}",
          "rollingInterval": "Hour"
        }
      }
    ]
  }
}
```

Arreter et supprimer les container inutiles et lancer la commande :

cd Projects/AppCarMicroService

docker-compose build

docker-compose up

```
C:\Users\elhad\Desktop\AutoEntrepreneur\Formations\DOTNET\ASP.NET\Projects\AppCarMicroService>docker-compose build
[+] Building 59.2s (18/18) FINISHED                                                                  docker:default
 => [myapi internal] load .dockerignore                                                                        0.0s
 => => transferring context: 2B                                                                                0.0s
 => [myapi internal] load build definition from Dockerfile                                                     0.0s
 => => transferring dockerfile: 1.20kB                                                                         0.0s
 => [myapi internal] load metadata for mcr.microsoft.com/dotnet/aspnet:7.0                                     0.7s
 => [myapi internal] load metadata for mcr.microsoft.com/dotnet/sdk:7.0                                        0.7s
 => [myapi runtime 1/1] FROM mcr.microsoft.com/dotnet/aspnet:7.0@sha256:c7d9ee6cd01afe9aa80642e577c7cec9f5d87f88e5d70bd36fd61072079bc55b  0.0s
 => [myapi internal] load build context                                                                        0.0s
 => => transferring context: 44.17kB                                                                           0.0s
 => [myapi build 1/7] FROM mcr.microsoft.com/dotnet/sdk:7.0@sha256:d32bd65cf5843f413e81f5d917057c82da99737cb1637e905a1a4bc2e7ec6c8d  0.0s
 => CACHED [myapi build 2/7] WORKDIR /app                                                                      0.0s
 => CACHED [myapi build 3/7] WORKDIR /src                                                                      0.0s
 => [myapi build 4/7] COPY . .                                                                                 1.4s
 => [myapi build 5/7] RUN dotnet restore AppCarMicroService.API/AppCarMicroService.API.csproj                 40.2s
 => [myapi build 6/7] WORKDIR /src/AppCarMicroService.API                                                      0.0s
 => [myapi build 7/7] RUN dotnet build "AppCarMicroService.API.csproj" -c Release -o /app                      8.4s
 => [myapi publish 1/2] WORKDIR /src/AppCarMicroService.API                                                    0.1s
 => [myapi publish 2/2] RUN dotnet publish "AppCarMicroService.API.csproj" -c Release -o /app                  7.3s
 => CACHED [myapi final 1/2] WORKDIR /app                                                                      0.0s
 => CACHED [myapi final 2/2] COPY --from=publish /app .                                                        0.0s
 => [myapi] exporting to image                                                                                 0.0s
 => => exporting layers                                                                                        0.0s
 => => writing image sha256:c2ba17b0bc8c79d759454cb2a97a17da37627bdb7b5fe748ef001fc2ebcd3605                   0.0s
 => => naming to docker.io/library/appcarmicroservice-myapi                                                    0.0s
```

```
C:\Users\elhad\Desktop\AutoEntrepreneur\Formations\DOTNET\ASP.NET\Projects\AppCarMicroService>docker-compose up
[+] Building 0.0s (0/0)                                                                              docker:default
[+] Running 3/3
 ✔ Network appcarmicroservice_car_micro_service_network  Created                                               0.2s
 ✔ Container mssqldb                                     Created                                               0.8s
 ✔ Container myapi                                       Created                                               0.2s
Attaching to mssqldb, myapi
mssqldb  | SQL Server 2022 will run as non-root by default.
mssqldb  | This container is running as user mssql.
mssqldb  | Your master database file is owned by mssql.
mssqldb  | To learn more visit https://go.microsoft.com/fwlink/?linkid=2099216.
myapi    | 2024-08-26 04:58:26.084 +00:00 [Information] (Microsoft.Hosting.Lifetime.) Now listening on: "http://[::]:80"
myapi    | 2024-08-26 04:58:26.172 +00:00 [Information] (Microsoft.Hosting.Lifetime.) Application started. Press Ctrl+C to shut down.
myapi    | 2024-08-26 04:58:26.174 +00:00 [Information] (Microsoft.Hosting.Lifetime.) Hosting environment: "Production"
myapi    | 2024-08-26 04:58:26.175 +00:00 [Information] (Microsoft.Hosting.Lifetime.) Content root path: "/app"
mssqldb  | 2024-08-26 04:58:34.97 Server      Setup step is FORCE copying system data file 'C:\templatedata\model_replicatedmaster.mdf' to '/var/opt/mssql/data/model_replic
atedmaster.mdf'.
2024-08-26 04:58:35.38 Server      Setup step is FORCE copying system data file 'C:\templatedata\model_replicatedmaster.ldf' to '/var/opt/mssql/data/model_replicatedmaster.
ldf'.
2024-08-26 04:58:35.62 Server      Setup step is FORCE copying system data file 'C:\templatedata\model_msdbdata.mdf' to '/var/opt/mssql/data/model_msdbdata.mdf'.
2024-08-26 04:58:35.96 Server      Setup step is FORCE copying system data file 'C:\templatedata\model_msdblog.ldf' to '/var/opt/mssql/data/model_msdblog.ldf'.
2024-08-26 04:58:36.43 Server      Microsoft SQL Server 2022 (RTM-CU14) (KB5038325) - 16.0.4135.4 (X64)
mssqldb Jul 10 2024 14:09:09
mssqldb Copyright (C) 2022 Microsoft Corporation
mssqldb Developer Edition (64-bit) on Linux (Ubuntu 22.04.4 LTS) <X64>
2024-08-26 04:58:36.44 Server      UTC adjustment: 0:00
2024-08-26 04:58:36.44 Server      (c) Microsoft Corporation.
2024-08-26 04:58:36.45 Server      All rights reserved.
2024-08-26 04:58:36.45 Server      Server process ID is 464.
2024-08-26 04:58:36.46 Server      Logging SQL Server messages in file '/var/opt/mssql/log/errorlog'.
2024-08-26 04:58:36.46 Server      Registry startup parameters:
mssqldb  -d /var/opt/mssql/data/master.mdf
mssqldb  -l /var/opt/mssql/data/mastlog.ldf
mssqldb  -e /var/opt/mssql/log/errorlog
2024-08-26 04:58:36.49 Server      SQL Server detected 1 sockets with 4 cores per socket and 8 logical processors per socket, 8 total logical processors; using 8 logical pr
ocessors based on SQL Server licensing. This is an informational message; no user action is required.
2024-08-26 04:58:36.51 Server      SQL Server is starting at normal priority base (=7). This is an informational message only. No user action is required.
```

On obtient sur http://localhost:5000/api/Brands