Table des matières

Objectif du TP

L'objectif de ce TP est de concevoir des microservices en ASP.NET Core et de les conteneuriser en les mettant sous Docker

Préparation de l'environnement de développement

Docker

Windows

L'ensemble des installers des logiciels ci-dessous sont dans le dossier Tools-Windows

Dans le cas où vous auriez W10 Enterprise, W10 Professionnel ou W10 Education, installez Docker Desktop for Windows. Cette installation va nécessiter l'activation d'Hyper-V et un redémarrage.

https://hub.docker.com/editions/community/docker-ce-desktop-windows

Dans les autres cas, utilisez Docker Toolbox.

https://docs.docker.com/toolbox/toolbox_install_windows/

MacOS

Dans le cas où vous auriez une version égale ou supérieure à Mac OS Sierra 10.12, installez Docker Desktop for Mac.

https://hub.docker.com/editions/community/docker-ce-desktop-mac

Dans les autres cas, utilisez Docker Toolbox.

https://docs.docker.com/toolbox/toolbox_install_mac/

Remarques Docker Toolbox

La VM créée par Docker Toolbox ne possède que 1GO de RAM, il faut donc augmenter cette RAM à 2Go pour ce TP.

Pour cela, il faut suivre les étapes suivantes :

1. Ouvrir Oracle VM VirtualBox
2. Eteindre la machine « default »
3. Ouvrir la configuration de la machine
4. Dans la section système, augmentez la RAM à 2048Mo

VS Code

Installer ou utiliser le VS Code fourni en fonction de votre OS.

https://code.visualstudio.com/

Pour les utilisateurs linux, il faudra installer les 2 extensions suivantes :

- C# (Microsoft)
- C# Extensions
- ASP.NET Core Snippets
- Essential ASP.NET Core 3 Snippets
- Docker (Microsoft)

Framework .NET Core

Installer le Framework .NET Core en installant ou suivant les instructions en fonction de votre OS.

```
https://dotnet.microsoft.com/download/dotnet-core/3.1
```

BDD Docker

Installation et lancement du container docker

```
docker run -d -p 27017:27017 -v mongo-data:/data/db --name mongo-db mongo:latest
```

Travail à effectuer

Consignes

Le but de ce TP est de construire de développer 2 microservices et une interface web consommant ces 2 microservices. L'ensemble de ces microservices développés en ASP.NET Core devront tourner dans des conteneurs Dockers indépendants. Ce sont les 2 microservices qui accèdent à la BDD.

Microservices :

1. StudentManager : Microservice permettant d'afficher et de gérer le référentiel des étudiants
2. CourseManager : Microservice permettant d'afficher les formations et d'y inscrire un étudiant.

Pour chaque microservice, il faut donc :

1. Créer le modèle objet,
2. Créer le contrôleur permettant les actions en API REST
3. Mettre en place le swagger
4. Créer et déployer vos conteneurs docker en local

L'application web avec la techno de votre choix permet donc d'effectuer les actions attendues sur les microservices par une IHM Web.

Compléments d'information

Définition des classes :

Student
Id : Guid
Lastname : string
Firstname : string
Birthdate : DateTime
Email : string

Course
Id : Guid
Label : string
Duration : TimeSpan
Theme : string
Teacher : string
Students : List<Guid>

Remarques

- Un exemple de projet vous a été mis à disposition dans le dossier src.
- Analysez le bien et n'hésitez pas à réutiliser la couche d'accès aux données (DAL)
- Le projet utilise de l'injection de dépendance, ça se passe dans Startup.cs
- Les chaines de connexion se trouvent dans les 2 fichiers appsettings(.*).json
- Lorsque vous runnez vous conteneurs, n'oublier pas de les lier à celui de la BDD

Trucs & astuces

ASP.NET Core

Lignes de commande

- Création d'un nouveau projet API REST (Dans un dossier projet)

```
dotnet new webapi
```

- Création d'un nouveau projet Web MVC (Dans un dossier projet)

```
dotnet new mvc
```

- Lancer l'exécution de votre programme

```
dotnet run
```

- Installation d'un package nuget

```
dotnet add package <NomDuPackage>
```

Packages Nuget utiles (cf http://nuget.org)

1. Microsoft.VisualStudio.Web.CodeGeneration.Design
2. Swashbuckle.AspNetCore
3. Mongo.DB.Driver

Divers

- Par convention les classes POCO du modèle objet se situe dans un dossier Models

Docker

Lignes de commande

1. Build son application dans un conteneur

```
docker build --rm -f "Dockerfile" -t <NomDeLImage>:<Tag> .
```

   - A Faire dans le dossier de l'application
   - --rm : Enlève les conteneurs intermédiaires après la fin du build
   - -f : spécifie le fichier de définition
   - -t : target
   - <NomDeLImage> : nom de l'image Docker cible (ex : studentmanager)
   - <NomDuTag> : nom du tag lié à la version (ex : 1.0.0-stable / 2.3.2-beta / latest)

2. Exécuter un conteneur

```
docker run --rm -it -p <HostPort>:<ContainerPort>/tcp --name <Name> --link=<ConainerName>:<Alias> <NomDeLImage>:<Tag>
```

   - --rm : Nettoie l'ancien conteneur
   - -it : Interactif
   - -p : Port mapping.
     - <HostPort> : Port sur la machine hôte
     - <Container> : Port sur le conteneur cible
   - --name <Name> : Nommage du conteneur
   - --link=<ConainerName>:<Alias> : Etablissement d'un lien entre conteneurs où <ContainerName> est le nom ou l'id du conteneur auquel on veut accéder et <Alias> est l'alias par lequel sera accessible le conteneur.
   - <NomDeLImage> : nom de l'image Docker cible (ex : studentmanager)
   - <NomDuTag> : nom du tag lié à la version (ex : 1.0.0-stable / 2.3.2-beta / latest)

3. Accéder à l'intérieur du conteneur

```
docker exec -it <Name> /bin/bash
```

   - <Name> : Nom ou id du conteneur

Visual Studio Code

1. F1 ouvre la ligne de commande interne
2. Docker : Add Docker Files to Workspace permet d'ajouter les fichiers nécessaires à la conteneurisation dans votre projet.