# Table des matières

[Table des matières 1](#_Toc176451511)

[1. Termes à savoir 3](#_Toc176451512)

[1.1. Qu’est-ce qu’un container ? 3](#_Toc176451513)

[1.2. Qu’est-ce que la containerisation ? 3](#_Toc176451514)

[1.3. Qu’est-ce que Docker ? 3](#_Toc176451515)

[1.4. Qu’est-ce qu’une image Docker ? 3](#_Toc176451516)

[1.5. Qu’est-ce qu’un Dockerfile ? 3](#_Toc176451517)

[2. Points à retenir en utilisant Docker dans des projets ASP.NET Core 4](#_Toc176451518)

[3. Dockeriser une nouvelle application ASP.NET Core 5](#_Toc176451519)

[4. Ajouter le support Docker à une application ASP.NET Core existante 9](#_Toc176451520)

[5. Créer un Dockerfile 10](#_Toc176451521)

[5.1. Créer une image pour exécuter une application ASP.NET Core 10](#_Toc176451522)

[5.1.1. Choisir l’image de base 10](#_Toc176451523)

[5.1.2. Spécifier le repertoire de travail 10](#_Toc176451524)

[5.1.3. Exposer les ports 10](#_Toc176451525)

[5.2. Créer une image pour développer une application ASP.NET Core 11](#_Toc176451526)

[5.2.1. Choisir l’image de base 11](#_Toc176451527)

[5.2.2. Spécifier le répertoire de travail 11](#_Toc176451528)

[5.2.3. Ajouter le fichier du projet à l’image Docker 11](#_Toc176451529)

[5.2.4. Restaurer les dépendances dans l’image 11](#_Toc176451530)

[5.2.5. Ajouter les fichiers de l’application à l’image 11](#_Toc176451531)

[5.2.6. Construire le projet ASP.NET Core dans l’image 12](#_Toc176451532)

[5.3. Publier l’application ASP.NET Core dans l’image 12](#_Toc176451533)

[5.4. Spécifier le point d'entrée d'exécution de l'application 12](#_Toc176451534)

[5.5. Dockerfile complet 13](#_Toc176451535)

[6. Construire une image pour un application ASP.NET Core 14](#_Toc176451536)

[7. Exécuter une application ASP.NET Core dans des containers 15](#_Toc176451537)

[8. Nettoyer les images et les containers 16](#_Toc176451538)

[8.1. Arrêter des containers 16](#_Toc176451539)

[8.2. Supprimer des containers 16](#_Toc176451540)

[8.3. Supprimer des images 16](#_Toc176451541)

[8.3.1. Supprimer une image spécifique 16](#_Toc176451542)

[8.3.2. Supprimer toutes les images 16](#_Toc176451543)

[8.3.3. Supprimer toutes les ressources inutiles 16](#_Toc176451544)

[9. Qu’est-ce que Docker Compose ? 17](#_Toc176451545)

[10. Ajouter le support de docker-compose dans Visual Studio 17](#_Toc176451546)

# Termes à savoir

## Qu’est-ce qu’un container ?

Un container est une unité logicielle portable et autonome comprenant tout ce qui est nécessaire pour exécuter une application (Code, dépendances, runtime, outils, paramètres, bibliothèques). Il permet de s’assurer que le code s’exécute de manière fiable et cohérente, quelle que soit l’infrastructure sous-jacente.

## Qu’est-ce que la containerisation ?

La containerisation est le processus de créer et de gérer des containers. Cela augmente la scalabilité (Capacité à rencontrer une demande d’utilisation plus grande) et la portabilité des applications.

## Qu’est-ce que Docker ?

Docker est un système de gestion de containers open-source permettant de développer, de tester et de déployer leurs applications dans des containers.

## Qu’est-ce qu’une image Docker ?

Une image Docker est un fichier autonome et en lecture seule agissant comme un modèle pour créer des containers. Elle inclut un OS allégé, le code de l’application, les bibliothèques, les outils système et les dépendances. Elles sont créées en couches, chaque couche représentant un ensemble spécifique de modifications ou d'ajouts à l'image de base.

## Qu’est-ce qu’un Dockerfile ?

Un Dockerfile est un fichier texte contenant les instructions ou commandes pour construire un image Docker. Il permet d’automatiser le processus de création d’images Docker et d’assurer la consistance et la facilité de développement à travers différents environnements. Il contient des commandes pour définir l’image de base, ajouter des dépendances, copier des fichiers dans l’image, définir des variables d’environnement, exécuter des commandes, exposer des ports, … Chaque commande dans un Dockerfile représente une couche dans l’image => Caching efficace et builds incrémentaux.

# Points à retenir en utilisant Docker dans des projets ASP.NET Core

* Pour construire, debugger et exécuter des applications ASP.NET Core pour .NET Core, utiliser VS 2017 ou plus.
* Pour containeriser un projet ASP.NET Core, il doit cibler .NET Core.
* Windows et Linux sont tous deux supportés pour des application ASP.NET Core.
* Pour ajouter le support Docker à un projet utilisant des containers Windows ou Linux, s’assurer qu’un container du même type est en train de s’exécuter sur l’hôte Docker (Docker Desktop).
* VS Container Tools ne supporte pas l’ajout de Docker à des projets ASP.NET Core existants ciblant le framework .NET.
* Le support Docker peut être ajouté à des nouveaux projets ASP.NET Core ou à des projets ASP.NET Core existants dans VS.
* Il est possible de debugger une application ASP.NET Core s’exécutant dans un container en utilisant VS.
* Un Dockerfile contenant des instructions pour construire une image peut être ajoutée dans des applications ASP.NET Core.
* Pour simplifier le processus d’exécution d’applications à plusieurs containers : fichier docker-compose.
* Les projets ASP.NET Core peuvent être publiés dans des registres de containers (Docker Hub, Azure Container Registry).

# Dockeriser une nouvelle application ASP.NET Core

* Créer un nouveau projet sur VS :

A screenshot of a computer

Description automatically generated

* Sélectionner le modèle *ASP.NET Core Web App* et cliquer sur *Next*:

A screenshot of a computer program

Description automatically generated

* Donner un nom au projet et définir son emplacement :

A screenshot of a computer

Description automatically generated

* Checker *Enable container support* puis sélectionner *Windows* dans le dropdown *Container OS*:

A screenshot of a computer

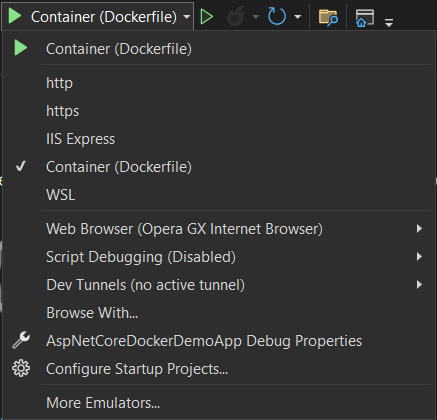
Description automatically generated

* Structure du projet créé :

A screenshot of a computer

Description automatically generated

Présence d’un Dockerfile et du package *Microsoft.VisualStudio.Azure.Containers.Tools.Targets*, et bouton *Run* remplacé par *Container (Dockerfile)* avec des options liés à Docker:



* Lancer l’application en cliquant le bouton *Container (Dockerfile)*. Après quelques commandes Docker. Le navigateur s’ouvre et l’application s’exécutant à l’intérieur d’un container Docker Windows s’affiche :

A screenshot of a computer

Description automatically generated

Ainsi que la fenêtre *Containers* qui affiche la liste des containers locaux sur l’ordinateur sur la gauche :

A screenshot of a computer

Description automatically generated

Le container est également affiché dans Docker Desktop :

A screenshot of a computer

Description automatically generated

# Ajouter le support Docker à une application ASP.NET Core existante

* Clic droit sur le projet
* Cliquer sur *Add*
* Cliquer sur *Docker support…*

A screenshot of a computer

Description automatically generated

* Choisir un OS cible (Windows ou Linux) => Un Dockerfile va apparaître ainsi que le package *Microsoft.VisualStudio.Azure.Containers.Tools.Targets.*

# Créer un Dockerfile

Un Dockerfile à la base du projet est nécessaire.

## Créer une image pour exécuter une application ASP.NET Core

### Choisir l’image de base

La première instruction est d’habitude une instruction *FROM*, initiant un nouveau stage de construction définissant l’image de base pour les instructions suivantes. Nécessite un nom d’image de conteneur Docker qualifiée. Un nom peut être optionnellement donné au stage via *AS*. Ce nom pourra être utilisé dans les instructions *FROM* et *COPY* (*-from=<nom>*) qui suivront.



### Spécifier le repertoire de travail

L’instruction *WORKDIR* spécifie le répertoire de travail pour les instructions suivantes (*RUN*, *CMD*, *ENTRYPOINT*, *COPY*, *ADD*). Si non spécifié, elle est créée automatiquement. Peut être utilisée plusieurs fois dans un Dockerfile.



### Exposer les ports

L’instruction *EXPOSE* informe Docker que le container écoute sur les ports spécifiés durant l’exécution. Ne publie pas le port : Sert plutôt de documentation entre celui qui construit l’image et celui qui exécute le container, au sujet de quels ports sont destinés à être publiés.

A black background with white text

Description automatically generated

## Créer une image pour développer une application ASP.NET Core

### Choisir l’image de base

Spécifier l’image de base .NET SDK qui contient le SDK .NET le runtime .NET SDK, et le CLI .NET.



### Spécifier le répertoire de travail



### Ajouter le fichier du projet à l’image Docker

L’instruction *COPY* copie les fichiers ou répertoires du système de fichiers local et l’ajoute au système de fichiers du container. Spécifier les chemins source et destination.

Pour copier le fichier *AspNetCoreDockerDemoApp.csproj* dans le répertoire *src* du container :



### Restaurer les dépendances dans l’image

Un projet .NET fait référence à des librairies externes dans des packages NuGet. Ces dépendances externes sont reprises dans le fichier .csproj. Pour demander au CLI .NET d’utiliser NuGet pour chercher ces dépendances et de les télécharger si nécessaire : *dotnet restore*. Permet également de s’assurer qu’il n’y a pas de conflit entre les dépendances.

L’instruction *RUN* est utilisée pour exécuter toute commande dans une nouvelle couche au-dessus de l’image courante. L’image résultant sera utilisée dans la suite du Dockerfile.



### Ajouter les fichiers de l’application à l’image

Une fois que les dépendances sont restaurées, copier tous les fichiers et dossiers du projet dans l’image.



### Construire le projet ASP.NET Core dans l’image

La commande du CLI .NET *dotnet build* construit le projet et ses dépendances en langage machine.

Argument *-c Release* : Spécifie le configuration (Dans ce cas, elle est spécifiée sur *Release*).

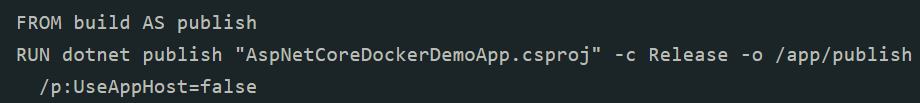
Argument *-o /app/build*: Spécifie le répertoire de sortie où l’enseble du code binaire sera copié.





## Publier l’application ASP.NET Core dans l’image

Initialiser un nouveau stage de build *publish* en faisant référence à un stage de build précédent (*build*). La commande *dotnet publish* compile l’application, lis les dépendances présentes dans le fichier du projet et publie l’ensemble des fichiers dans un répertoire.



## Spécifier le point d'entrée d'exécution de l'application

Pour exécuter l’application dans des containers Docker, il est nécéssaire de créer un nouveau stage de build (Nommé *final*) en faisant référence au stage de build précédent *base* :



Le répertoire de travail est ensuite défini sur */app* Et le code machine de l’application est copié de /app/publish vers le répertoire de travail :

A black background with white text

Description automatically generated

Puis la commande *ENTRYPOINT* est exécutée. Elle définit le processus ou le script principal qui sera exécuté quand un conteneur basé sur cette image sera démarré.



Dans le cas de l’application, elle est comme tel :



Cela signifie que la commande *dotnet AspNetCoreDockerDemoApp.dll* va être exécutée au démarrage du container.

## Dockerfile complet

A screenshot of a computer program

Description automatically generated

# Construire une image pour un application ASP.NET Core

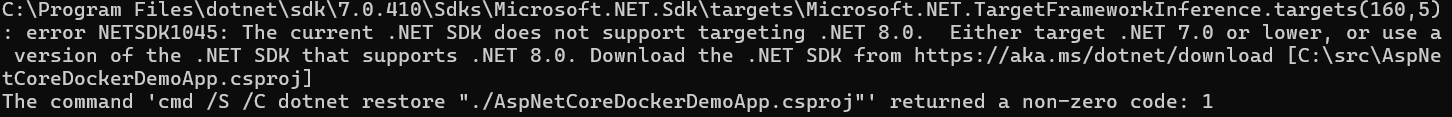
Pur créer une image, utiliser la commande *docker build* dans le répertoire où le Dockerfile est situé.

Argument *-t myaspnetdockerapp*: Définit le nom de l’image

Argument *.*: Précise l’emplacement du Dockerfile, dans ce cas dans le répertoire courant.



Remarque : Si le message d’erreur suivant s’affiche :



Ouvrir le fichier *AspNetCoreDockerDemoApp.csproj* et changer la balise *<TargetFramework>net****8****.0</TargetFramework>* en *<TargetFramework>net****7****.0</TargetFramework>.*

# Exécuter une application ASP.NET Core dans des containers

Pour créer un container Docker à partir de l’image créée précédemment, afin d’exécuter l’application sur celui-ci :



Argument *-d*: Spécifie que le container doit s’exécuter en arrière-plan.

Argument *-p 8080 :80* : Lie le port interne du container 80 au port externe de la machine hôte 8080

Argument *–name myapp* : Donne le nom *myapp* au container.

Argument *myaspnetdockerapp*: Nom de l’image Docker utilisée.

Après l’exécution de cette commande, checker si le container est créé et en cours d’exécution via la commande *docker ps*.

Pour accéder à l’application, taper <http://localhost:8080> (8080 : Port situé à gauche du « : ») dans le navigateur internet.

# Nettoyer les images et les containers

## Arrêter des containers



## Supprimer des containers



## Supprimer des images

### Supprimer une image spécifique



### Supprimer toutes les images



### Supprimer toutes les ressources inutiles

Pour supprimer toutes les ressources inutiles comme les images, containers, volumes et networks non tagués ou associés à un container (= vielles ressources qui ne sont plus utilisées) :



# Qu’est-ce que Docker Compose ?

Docker Compose est un outil simplifiant la gestion et le déploiement d’applications à plusieurs containers. Il permet de définir et de configurer tous les services, networks et volumes requis pour une application en utilisant un seul fichier YAML (*docker-compose.yml*).

# Ajouter le support de docker-compose dans Visual Studio

* Clic droit sur le projet > *Add* > *Container Orchestrator Support…*

A screenshot of a computer

Description automatically generated

* Choisir *Docker Compose* > *OK*.

A screenshot of a computer

Description automatically generated

* Choisir Linux comme OS cible.

A screenshot of a computer program

Description automatically generated

Un fichier *docker-compose.yml* est créé dans le projet :

A screenshot of a computer program

Description automatically generated

# Création d’une application ASP.NET Core et SQL Server Dockerisée

## Installation de packages NuGet

Installer les packages suivants :

* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.EntityFrameworkCore.Design
* Microsoft.EntityFrameworkCore.Tools

## Créer un modèle

Dans le dossier *Models*, créer la classe *Product.cs*.

A screen shot of a computer program

Description automatically generated

## Créer un contexte de base de données

Créer un dossier *Data* puis créer une classe *OnlineShopDbContext* dans celui-ci.

A screen shot of a computer program

Description automatically generated