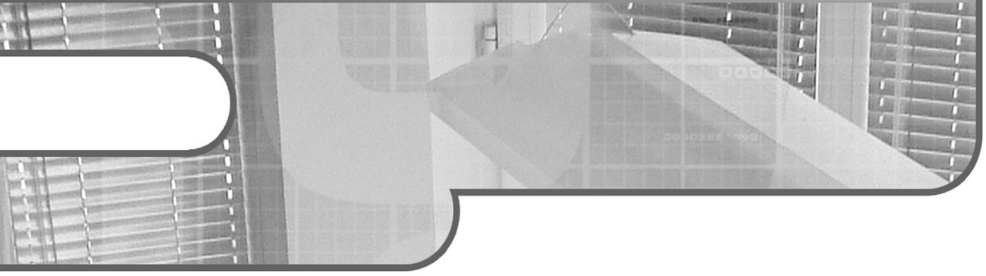




# Chapitre 3
# Les nouveautés d'ASP.NET Core

## 1. Introduction

ASP.NET existe depuis 2002 et bien des changements ont été effectués sur le framework depuis sa première version. Il faut rappeler une chose sur ASP.NET Core : la nouvelle plateforme web de Microsoft n'est en aucun cas une suite de la version 4.6 du framework que nous connaissions, mais bien un renouveau qui doit marquer une nouvelle ère de la technologie de Microsoft dans le Web moderne.

Certains diront que le framework n'a pas tant changé que cela (surtout la partie MVC), pourtant c'est bien "sous le capot" que les changements ont été les plus profonds, en commençant par le namespace `System.Web` qui n'existe plus. Ensuite, annoncé comme cross-platform, ASP.NET Core est plus modulable qu'il ne l'a été dans les années précédentes. Via ***NuGet*** pour les composants serveurs, puis via ***Grunt*** ou ***Gulp*** pour la partie cliente du site web, le nouveau framework bénéficie également d'un nouveau runtime, appelé ***CoreCLR***, permettant l'exécution d'une application web Microsoft sur Linux ou Mac.

## 2. Les nouveaux outils open source

ASP.NET Core arrive avec un tout nouveau panel d'outils open source permettant la gestion des nouveaux projets web. Fort heureusement pour le développement, l'ensemble de ces outils est réuni autour d'une seule et même interface en ligne de commande : dotnet.

### 2.1 L'environnement d'exécution dotnet

dotnet a été conçu afin de faire fonctionner des applications .NET de manière cross-platform sur Windows, Mac et Linux et ceci sans développer un runtime différent pour chaque plateforme. C'est à la fois un environnement d'exécution et un SDK embarquant ainsi tout ce qui est nécessaire pour le bon fonctionnement des applications web ASP.NET cross-platform.

Totalement orienté ***package-first***, Microsoft a poussé le concept de modularité très loin, permettant même à l'environnement d'exécution d'embarquer, de gérer et de créer de lui-même les packages dont il a besoin, et ceci de manière automatique via NuGet. dotnet est capable de cibler plusieurs frameworks (.NET Core ou le framework .NET Full), et ainsi générer les packages NuGet directement. De plus, dotnet embarque le nouveau moteur d'exécution CoreCLR conçu spécialement pour les problématiques de compatibilité sur les autres plateformes.

dotnet est intégré à Visual Studio 2015 pour offrir une expérience développeur enrichie, mais l'environnement d'exécution est également pilotable via les lignes de commande. Dans un projet ASP.NET Core, il est possible de rajouter des outils Microsoft et ainsi piloter son projet avec dotnet en ligne de commande. Ces outils se rajoutent en même temps que le paquet NuGet dans le fichier *.csproj* :

```
<ItemGroup>
    <DotNetCliToolReference Inclu-de="Microsoft.VisualStudio.Web.
CodeGeneration.Tools" Version="2.0.4" />
    <DotNetCliToolReference Include="BundlerMinifier.Core"
Version="2.0.238" />
  </ItemGroup>
```

Les commandes déclarées permettent par exemple de lancer un outil de génération de code, anciennement appelé *scaffolding*. Ce dernier peut permettre de générer le contrôleur et les vues qui correspondent à un modèle de données bien précis. La commande est utilisable via `dotnet <command>` :

```
dotnet Microsoft.VisualStudio.Web.CodeGeneration.Tools
```

Au sein de l'application elle-même, il est possible d'utiliser les services de dotnet via certaines interfaces C# disponibles via injection de dépendances. On retrouve des services comme `IServiceEnvironment` permettant de modifier la configuration de l'environnement d'exécution pendant que l'application elle-même fonctionne.

L'utilitaire `dotnet` contient également une liste de commandes prédéfinies permettant d'effectuer certaines actions sur le projet ASP.NET Core :

– `dotnet new` : initialise un projet C# console simple.

– `dotnet restore` : restaure les dépendances du projet selon le *.csproj*.

– `dotnet build` : construit l'application .NET Core.

– `dotnet publish` : publie une application .NET Core.

– `dotnet run` : lance l'application depuis les sources.

– `dotnet test` : lance les tests utilisant le *test runner* défini dans le *.csproj*.

– `dotnet pack` : crée un paquet NuGet depuis le code source.

Cet outil supporte ainsi plusieurs processus de lancement de l'application ASP.NET Core. Le premier consiste simplement à restaurer les packages nécessaires à l'application, puis à lancer le projet depuis les sources.

```
dotnet new
dotnet restore
dotnet run
```

Cependant, l'utilitaire permet également de lancer l'application directement depuis une DLL après avoir lancé la génération du projet.

```
dotnet build
dotnet run bin/Debug/netcoreapp1.0/test-app.dll
```

Avec les deux processus ci-dessus, l'outil montre qu'il est capable d'exécuter plusieurs types d'applications :

– Des applications "portables", c'est-à-dire des applications qui dépendent de la version de .NET Core installée sur la machine. Cela veut dire que ce type d'application sera capable de fonctionner sur différentes installations de .NET Core, peu importe les systèmes d'exploitation utilisés. Les applications "portables" permettent également de centraliser les librairies de .NET : elles embarqueront donc uniquement les librairies externes.
– Des applications "autonomes", c'est-à-dire des applications contenant toutes les dépendances dont elles ont besoin pour fonctionner, incluant le runtime .NET Core faisant entièrement partie de l'application. Cela permet de faciliter le déploiement de l'application mais alourdit le package. De plus, il convient à l'application de spécifier la version du runtime à utiliser dans le *.csproj*.

## 2.2 L'utilitaire dotnet restore

Un projet web ASP.NET Core possède plusieurs dépendances à des packages externes provenant souvent de NuGet. Pour faire fonctionner une application web, il va tout d'abord falloir restaurer les packages nécessaires au bon fonctionnement du projet. C'est une des nouveautés d'ASP.NET Core : le projet embarque uniquement ce dont il a besoin, et donc il a fallu concevoir un outil permettant de restaurer ces dépendances, ceci de manière cross-platform et totalement transparente. dotnet restore permet de faire cette restauration.

Intégré à .NET Core et installé avec dotnet, dotnet restore permet de scruter le projet ASP.NET Core et de récupérer les packages dont il a besoin dans le cloud. Visual Studio 2015 utilise automatiquement cet utilitaire lorsque les dépendances du projet sont mises à jour. Il est cependant possible de lancer le processus à la main grâce à la commande suivante :

```
> dotnet restore
```

Remarque

*Il faut noter toutefois que, dans la console depuis laquelle la commande est lancée, il faut se placer à la racine du projet. En effet,* `dotnet restore` *va inspecter le fichier .csproj pour identifier les dépendances à récupérer.*

Grâce à dotnet restore et à sa simplicité d'utilisation, il est très facile de restaurer les dépendances d'un projet ASP.NET Core, et ceci indépendamment de la plateforme.

## 2.3 Gérer ses paquets NuGet avec dotnet pack

L'outil `dotnet pack` est utilisé pour générer un package NuGet à partir d'un projet .NET. Il peut être utilisé de différentes manières pour gérer ses paquets NuGet. Dans un premier temps, nous pouvons utiliser la commande suivante, à la racine d'un projet .NET, afin de simplement générer un paquet NuGet :

```
dotnet pack mon_projet.csproj
```

Cette commande générera un package NuGet à partir du projet spécifié et l'enregistrera dans le répertoire `bin\Debug` ou `bin\Release`, selon le mode de compilation utilisé.

Afin d'inclure les informations de version et de métadonnées dans le package NuGet généré, nous pouvons utiliser les options `--version` et `--metadata` :

```
dotnet pack mon_projet.csproj --version 1.2.3
--metadata authors="John Doe"
```

Dans cet exemple, nous avons spécifié la version `1.2.3` du package et ajouté une métadonnée `authors` avec la valeur "`John Doe`".

Pour publier le package NuGet généré sur un dépôt NuGet, nous pouvons utiliser l'option `--publish` en spécifiant l'URL du dépôt :

```
dotnet pack mon_projet.csproj --publish https://mynugetrepo.com
```

Cette commande publiera le package NuGet généré sur le dépôt NuGet spécifié.

Pour plus d'informations, nous pouvons consulter la documentation officielle ou utiliser la commande `dotnet pack --help` pour afficher la liste complète des options disponibles :

- `--output` : répertoire de sortie pour les paquets générés.
- `--no-build` : génère un paquet NuGet sans lancer la génération du projet .NET.
- `--no-restore` : génère un paquet NuGet sans lancer la restauration des paquets NuGet du projet.
- `--include-symbols` : inclue les symboles de compilations à côté du paquet généré dans le dossier de sortie.
- `--include-source` : inclut les fichiers PDB et les fichiers sources. Les sources iront dans le dossier `src`.
- `--serviceable` : définit le niveau de maintenance du paquet.
- `--nologo` : n'affiche pas le logo lors du démarrage de la commande.
- `--interactive` : permet d'attendre ou non une interaction utilisateur si nécessaire.
- `--verbosity` : indique le niveau de verbosité des logs affichés lors du lancement de la commande.
- `--version-suffix` : définit la valeur de la propriété `$(VersionSuffix)` à utiliser lors de la génération du projet.
- `--configuration` : définit la configuration à utiliser lors de la génération. Les valeurs peuvent être `Debug` ou `Release`.
- `--use-current-runtime` : définit s'il faut utiliser le runtime actuel comme runtime cible.

La commande `dotnet pack` est très utile pour la gestion des paquets NuGet des projets .NET et permet au développeur de créer et publier des paquets facilement.

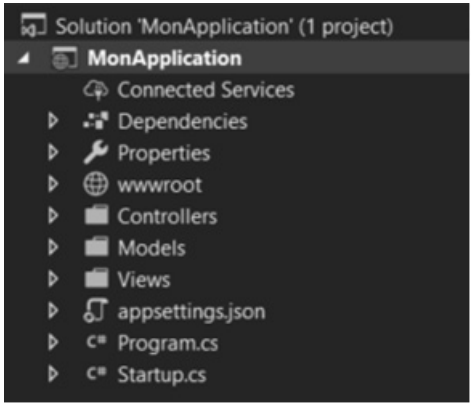
## 3. La structure d'une solution

Une solution ASP.NET Core est la base d'un projet web utilisant les technologies Microsoft. Elle permet rapidement de déployer un site et de structurer le code utilisé pour faire fonctionner l'application. Une solution peut comporter à la fois le code côté serveur et le code côté client, tout en incluant des mécanismes afin de bien séparer les deux parties. Ce chapitre va traiter des différents composants d'une solution ASP.NET Core et expliquer leurs rôles dans la configuration ou le déploiement de l'application web.

### 3.1 Les fichiers .csproj

Un projet ASP.NET Core met en œuvre une toute nouvelle philosophie de conception d'applications web issue de chez Microsoft s'inspirant beaucoup de l'open source.

Le nouveau template ressemble à ceci :




*Nouveau template de projet ASP.NET Core*