

Table des matières

[Introduction 1](#_Toc66287579)

[Dans Visual Studio Code 1](#_Toc66287580)

[Dans Visual Studio 2](#_Toc66287581)

[Dans le portail Azure 2](#_Toc66287582)

[Slot de déploiement : 2](#_Toc66287583)

[Scale Up 3](#_Toc66287584)

[Scale Out 3](#_Toc66287585)

[Configuration 3](#_Toc66287586)

[Advanced Tools 3](#_Toc66287587)

# Introduction

Dans cette démonstration, nous allons voir comment créer l’infrastructure nécessaire sur Azure afin de pouvoir déployer notre application Woodgrove

# Dans Visual Studio Code

Il existe plusieurs manières et outils pour créer une infrastructure sur Azure, que ce soit en mode impératif à base de script, ou déclaratif à base de modèle, soit propre à Azure comme le modèle ARM (Azure Resource Manager) ou via du Terraform, ansible et autre.

Ici nous allons nous allons utiliser un mode impératif à base de script en utilisant l’outil az cli, outil disponible sur Windows, Linux et MacOs.

Dans un 1er temps nous allons préparer le code afin de pouvoir le déployer

1. Nous publions l’ensembles des composants qui composent notre application
2. Nous créons un package de déploiement au format Zip

Ensuite nous allons créer l’infrastructure nécessaire sur Azure afin d’héberger notre application

1. Nous allons créer un groupe de ressources, dans la région francecentral
2. Créer un Service Plan, qui permettra d’héberger notre application. Le service Plan, fourni les capacités hardware (CPU, MEMOIRE, RESEAU, DISQUE, etc..) pour notre ou nos applications Web. En effet sur un service plan peut résider plusieurs Apps.
3. Nous allons ensuite créer l’application Web, ou si vous préférez le serveur Web associé à notre service plan
4. Enfin nous allons déployer notre application sous forme d’un package zip.

# Dans Visual Studio

Bien évidemment, si on est en phase de développement, on peut utiliser directement notre éditeur de code préféré,

Par exemple, dans Visual studio, il est possible de créer l’infrastructure et de publier directement dans Azure

1. On clique sur le bouton droit au niveau du projet
2. Commande Publish
3. On choisit Azure
4. Next
5. Notez que comme l’application est développé avec ASP.NET CORE elle est de ce faite multi-plateforme, Windows/Linux
6. Pour l’exemple, je vais prendre linux, car dans le script par défaut il prend windows
7. Ici il me liste les ressources groupes qui contiennent déjà des App Service Linux
8. Nous allons en créer un nouveau
9. Nous allons choisir un nom
10. Choisir l’abonnement azure
11. Créer un nouveau ressource groupe
12. Créer un nouveau plan
13. Choisir la région
14. Et créer l’app service

Allons sur le portail

# Dans le portail Azure

Comme je vous le disais nous avons donc deux composants Azure.

Notre Service Plan et notre web App

Je ne vais pas aborder toutes les fonctionnalités disponibles pour une app service, mais voyons les plus utiles pour démarrer

## Slot de déploiement :

Les slots de déploiement permettent de déployer sur slot de type staging, à des fins de tests en production et si tout fonctionne correctement de swapper de l’un vers l’autre

Ici vous voyez que le service plan que j’ai choisi au départ étant de dev & Test, ne me permet pas de rajouter un slot.

Pas de problème, je vais pouvoir à la volée, upgrader le service plan associé à ma Web app

1. Je vais aller dans Scale Up,
2. Choisir Production
3. Prendre par exemple S1 comme taille
4. Apply
5. Une fois ceci fait, je peux revenir sur Deployments slots
6. Créer un slot staging que je clone

Ensuite si je développe une nouvelle fonctionnalité dans mon application, je peux la déployer sur ce slot staging, tester si cela fonctionne correctement et swapper les environnements

## Scale Up

Cette fonctionnalité de scale up est utile également si jamais vous constatez que la puissance ou les ressources mémoire/disque et autre ne sont pas suffisantes pour faire tourner votre application.

## Scale Out

En complément, si jamais votre application subit une forte charge, il est possible d’augmenter le nombre d’instances de l’application soit manuellement, soit en fonction de règles bien précise basées sur des métriques d’utilisation de la CPU, l’utilisation de la mémoire et autres.

Par exemple ici je peux ajouter une règle qui en fonction d’un pourcentage d’utilisation de la CPU supérieur à 70%

**Question.**  
Vaut-il mieux par exemple deux instances avec des machines très performantes, ou 8 instances de machines plus petites ?  
En réalité, il faut surveiller continuellement l’application et si les métriques indiquent que votre système est peu chargé et qu’il dispose d’importantes capacités inutilisées, vous pouvez effectuer un scale-in pour réduire les coûts.  
Dans les deux cas, vous devez continuer à superviser les statistiques de l’application web. Veillez à stabiliser le système. Si les métriques indiquent que l’application est toujours sous-alimentée (ou suralimentée), ajoutez ou supprimez des instances selon les besoins.

## Configuration

Ici on va pouvoir ajouter des paramètres à l’application, vérifier ou modifier les paramètres générales du service en lui-même, la pile de développement utilisée, la plate-forme, la possibilité de faire du debug à distance etc..

## Advanced Tools

Kudu : [Home · projectkudu/kudu Wiki (github.com)](https://github.com/projectkudu/kudu/wiki)

Enfin je finirai ce rapide tour de App Service, par les outils avancés qui vont vous permettre d’investiguer plus en profondeur les éventuelles problèmes que vous pourriez rencontrer avec votre application.

En effet derrière chaque web app, se cache un site nommé kudu, qui vous permet d’accéder à l’environnement ou votre application est déployée.

Comme vous le voyez, on a accès à un certain nombre d’outils qui vous permettent de lister les variables d’environnements, d’accéder à la console, de lister

## Montrer l’application woodgroove