eBook Gratuit

APPRENEZ azure-webjobs

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#azure-webjobs

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: azure-webjobs

It is an unofficial and free azure-webjobs ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official azure-webjobs.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec azure-webjobs

Remarques

Azure WebJobs permet d'exécuter facilement des scripts ou des programmes en tant que processus d'arrière-plan dans le contexte d'une application Web, d'une application API ou d'une application mobile App Service. Vous pouvez télécharger et exécuter un fichier exécutable tel que:

- .cmd, .bat, .exe (en utilisant Windows cmd)
- .ps1 (en utilisant PowerShell)
- .sh (en utilisant bash)
- .php (en utilisant PHP)
- .py (en utilisant Python)
- .js (en utilisant Node.js)
- .jar (en Java)

Ces programmes s'exécutent en tant que WebJobs selon un calendrier (cron) ou en continu.

Vous pouvez utiliser le SDK WebJobs pour simplifier le code que vous écrivez pour les tâches courantes qu'un WebJob peut effectuer, comme le traitement des images, le traitement des files d'attente, l'agrégation RSS, la maintenance des fichiers et l'envoi d'e-mails. WebJobs SDK possède des fonctionnalités intégrées pour travailler avec Azure Storage and Service Bus , pour planifier des tâches et gérer des erreurs, ainsi que pour de nombreux autres scénarios courants.

Versions

Azure WebJobs SDK

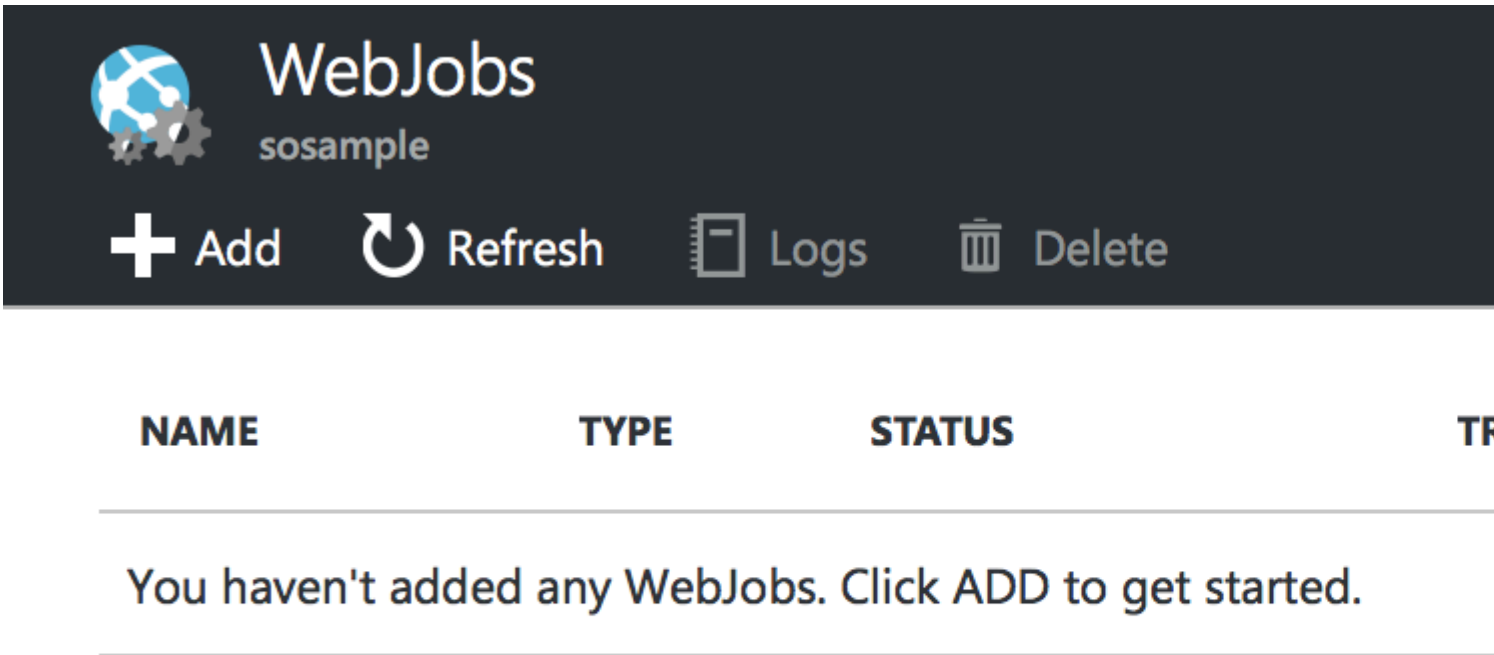
Version	Date de sortie
2.0.0-beta1	2016-07-14
1.1.2	2016-04-22
1.1.1	2016-01-13
1.1.0	2015-11-19
1.1.0-rc1	2015-11-02
1.1.0-beta1	2015-09-16
1.1.0-alpha2	2015-08-12
1.1.0-alpha1	2015-07-10

Version	Date de sortie
1.0.1	2015-03-19
1.0.1-alpha1	2015-02-18
1.0.0	2014-10-17
1.0.0-rc1	2014-09-22
0.6.0-beta	2014-09-13
0.5.0-bêta	2014-09-05
0.4.1-bêta	2014-08-30
0.4.0-bêta	2014-08-21

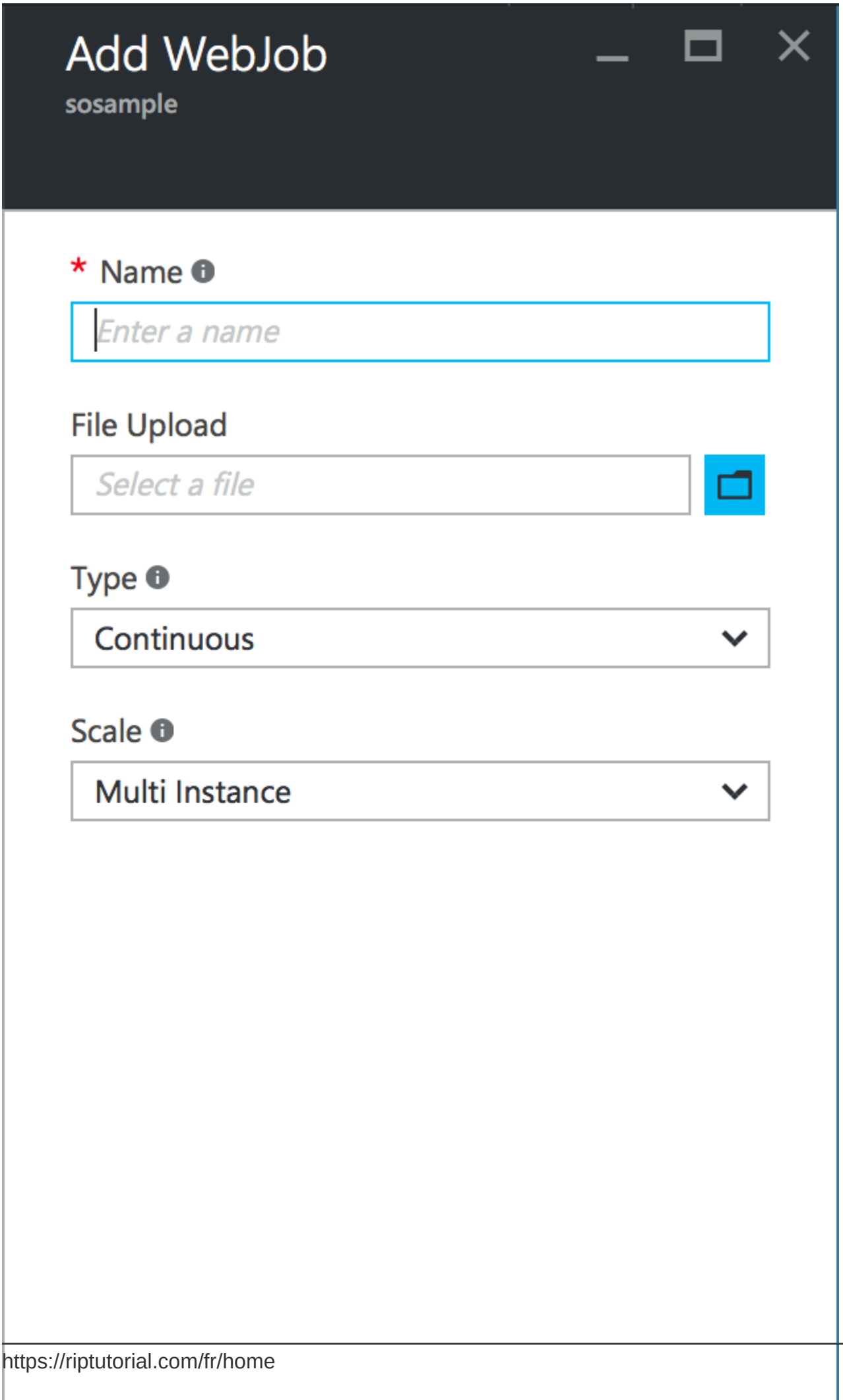
Examples

Création d'un WebJob dans le portail Azure

1. Dans la lame **Web App** du portail Azure , cliquez sur **Tous les paramètres** > **WebJobs** pour afficher la lame WebJobs:



2. Cliquez sur **Ajouter** . La boîte de dialogue **Ajouter un WebJob** apparaît.


Add WebJob
sosample
* Name
Enter a name
File Upload
Select a file
Type
Continuous
Scale
Multi Instance

, nommez le WebJob. Le nom doit commencer par une lettre ou un chiffre et ne peut contenir aucun caractère spécial autre que "-" et "_".

4. Dans la zone **Comment exécuter** , choisissez l'option préférée **Continu** ou **Déclenchée** (le déclencheur peut utiliser un programme cron ou un WebHook).
5. Dans la zone **Téléchargement de** fichier, cliquez sur l'icône du dossier et naviguez jusqu'au fichier zip contenant votre script. Le fichier zip doit contenir votre exécutable (.exe .cmd .bat .sh .php .py .js) ainsi que tous les fichiers de support nécessaires pour exécuter le programme ou le script.
6. Cochez **Créer** pour télécharger le script sur votre application Web. Le nom que vous avez spécifié pour le WebJob apparaît dans la liste de la lame WebJobs.

Lire Démarrer avec azure-webjobs en ligne: https://riptutorial.com/fr/azure-webjobs/topic/1311/demarrer-avec-azure-webjobs

Chapitre 2: Azure Webjobs SDK

Examples

JobHost

Azure Webjobs SDK est une **structure** distribuée sous la forme d'un package Nuget destiné à vous aider à définir des **fonctions** exécutées par des **déclencheurs** et à utiliser des **liaisons** vers d'autres services Azure (tels que Azure Storage and Service Bus) de manière **déclarative** .

Le SDK utilise un **JobHost** pour coordonner vos fonctions codées. Dans un scénario typique, votre Webjob est une application console qui initialise JobHost de la manière suivante:

```
class Program
{
    static void Main()
    {
        JobHostConfiguration config = new JobHostConfiguration();
        config.StorageConnectionString = "Your_Azure_Storage_ConnectionString";
        config.DashboardConnectionString = "Your_Azure_Storage_ConnectionString";
        JobHost host = new JobHost(config);
        host.RunAndBlock();
    }
}
```

JobHostConfiguration vous permet de personnaliser davantage de paramètres pour différents déclencheurs:

```
config.Queues.BatchSize = 8;
config.Queues.MaxDequeueCount = 4;
config.Queues.MaxPollingInterval = TimeSpan.FromSeconds(15);
config.JobActivator = new MyCustomJobActivator();
```

Déclencheurs pour les files d'attente

Un exemple simple définissant une fonction qui est déclenchée par un message de file d'attente:

```
public static void StringMessage([QueueTrigger("my_queue")] string plainText)
{
    //...
}
```

Il supporte également la sérialisation POCO :

```
public static void POCOMessage([QueueTrigger("my_queue")] MyPOCOClass aMessage)
{
    //...
}
```

Déclencheurs pour Blobs

Exemple simple d'une fonction déclenchée lors de la modification d'un blob de stockage Azure:

```
public static async Task BlobTrigger(
[BlobTrigger("my_container/{name}.{ext}")] Stream input,
string name,
string ext)
{
    //Blob with name {name} and extension {ext}

    using (StreamReader reader = new StreamReader(input))
    {
        //Read the blob content
        string blobContent = await reader.ReadToEndAsync();

    }
}
```

Déclencheurs par le temps

Le SDK prend en charge l'heure déclenchée sur la base d' expressions CRON avec **6 champs** (`{second} {minute} {hour} {day} {month} {day of the week}`). Il nécessite un **paramètre supplémentaire** sur `JobHostConfiguration` :

```
config.UseTimers();
```

Vos fonctions déclenchées par le temps répondent à cette syntaxe:

```
// Runs once every 5 minutes
public static void CronJob([TimerTrigger("0 */5 * * * *")] TimerInfo timer)
{

}

// Runs immediately on startup, then every two hours thereafter
public static void StartupJob([TimerTrigger("0 0 */2 * * *", RunOnStartup = true)] TimerInfo
timerInfo)
{

}
```

Déclencheurs par erreurs

La gestion des erreurs est extrêmement importante, nous pouvons définir des fonctions à déclencher lorsqu'une erreur d'exécution se produit dans l'une de vos fonctions déclenchées:

```
//Fires when 10 errors occur in the last 30 minutes (sliding)
public static void ErrorMonitor([ErrorTrigger("0:30:00", 10)] TraceFilter filter)
{
        // get the last 5 errors
        filter.GetDetailedMessage(10);
}
```

C'est particulièrement utile pour centraliser la gestion des erreurs.

ErrorTrigger nécessite un **paramètre supplémentaire** sur JobHostConfiguration:

```
config.UseCore();
```

Vous devez également installer le package NuGet **Microsoft.Azure.WebJobs.Extensions** .

Mise à l'échelle

Azure Webjobs s'exécute sur un service d'application Azure. Si nous mettons à l'échelle notre service d'application horizontalement (ajout de nouvelles instances), **chaque instance** aura **son propre JobHost** .

Notez que cela ne s'applique qu'aux WebJobs exécutés en mode **Continu** . Les WebJobs à la demande et planifiés ne sont pas affectés par la mise à l'échelle horizontale, ils exécutent toujours une seule instance.

Si vous avez une file d'attente continue de traitement des messages WebJob et que vous mettez le plan de service d'application à 3 instances, vous aurez 3 instances de WebJob en cours d'exécution.

Il se peut que WebJobs soit exécuté dans une seule instance, car vous devrez peut-être vous assurer qu'un seul pipeline de traitement existe. Pour ces WebJobs, vous pouvez ajouter l'attribut **Singleton** .

```
[Singleton]
public static void SingletonQueueProcessing([QueueTrigger("my_queue")] MyPOCOClass aMessage)
{
    //...
}
```

Ceci est réalisé par Azure Blob Leases pour le verrouillage distribué.

Ecriture des journaux

Le tableau de bord WebJobs affiche les journaux à deux endroits: la page du WebJob et la page pour un appel WebJob particulier.

La sortie des méthodes de console que vous appelez dans une fonction ou dans la méthode `Main()` apparaît dans la page Tableau de bord du WebJob et non dans la page d'un appel de méthode particulier. La sortie de l'objet TextWriter que vous obtenez d'un paramètre dans votre signature de méthode apparaît dans la page Tableau de bord pour un appel de méthode.

Pour écrire des journaux de suivi d'application, utilisez `Console.Out` (crée des journaux marqués comme INFO) et `Console.Error` (crée des journaux marqués comme ERROR).

```
public static void WriteLog([QueueTrigger("logqueue")] string message, TextWriter logger)
{
```

```
    Console.WriteLine("Console.Write - " + message);
    Console.Out.WriteLine("Console.Out - " + message);
    Console.Error.WriteLine("Console.Error - " + message);
    logger.WriteLine("TextWriter - " + message);
}
```

Qui se traduira par ces messages dans le tableau de bord pour le WebJob:

```
[07/28/2016 22:29:18 > 0a1c35: INFO] Console.Write - Hello world!
[07/28/2016 22:29:18 > 0a1c35: INFO] Console.Out - Hello world!
[07/28/2016 22:29:18 > 0a1c35: ERR ] Console.Error - Hello world!
```

Et ce message dans la page Tableau de bord de la méthode:

```
TextWriter - Hello world!
```

Injection de dépendance à l'aide de Ninject

L'exemple suivant montre comment configurer l'injection de dépendance à l'aide de Ninject en tant que conteneur IoC.

Ajoutez d'abord une classe CustomModule à votre projet WebJob et ajoutez-y toutes les liaisons de dépendance.

```
public class CustomModule : NinjectModule
{
    public override void Load()
    {
        Bind<IMyInterface>().To<MyService>();
    }
}
```

Ensuite, créez une classe JobActivator:

```
class JobActivator : IJobActivator
{
    private readonly IKernel _container;
    public JobActivator(IKernel container)
    {
        _container = container;
    }

    public T CreateInstance<T>()
    {
        return _container.Get<T>();
    }
}
```

Lorsque vous configurez JobHost dans la fonction principale de la classe Program, ajoutez JobActivator à JobHostConfiguration.

```
public class Program
```

```
{
    private static void Main(string[] args)
    {
        //Set up DI
        var module = new CustomModule();
        var kernel = new StandardKernel(module);

        //Configure JobHost
        var storageConnectionString = "connection_string_goes_here";
        var config = new JobHostConfiguration(storageConnectionString) { JobActivator = new
JobActivator(kernel) };

        //Pass configuration to JobJost
        var host = new JobHost(config);

        // The following code ensures that the WebJob will be running continuously
        host.RunAndBlock();
    }
}
```

Enfin, dans la classe Functions.cs, injectez vos services.

```
public class Functions
{
    private readonly IMyInterface _myService;

    public Functions(IMyInterface myService)
    {
        _myService = myService;
    }

    public void ProcessItem([QueueTrigger("queue_name")] string item)
    {
        _myService .Process(item);
    }
}
```

Lire Azure Webjobs SDK en ligne: https://riptutorial.com/fr/azure-webjobs/topic/2662/azure-webjobs-sdk

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec azure-webjobs	Community, gbellmann
2	Azure Webjobs SDK	gbellmann, juunas, lopezbertoni, Matias Quaranta