Kostenloses eBook

LERNEN azure-webjobs

Free unaffiliated eBook created from
Stack Overflow contributors.

#azure-webjobs

Inhaltsverzeichnis

Über

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

Kapitel 1: Erste Schritte mit Azure-Webjobs

Bemerkungen

Azure WebJobs bieten eine einfache Möglichkeit, Skripts oder Programme als Hintergrundprozesse im Kontext einer App Service-Webanwendung, einer API-App oder einer mobilen App auszuführen. Sie können eine ausführbare Datei hochladen und ausführen, z.

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

Diese Programme werden als WebJobs nach einem Zeitplan (cron) oder kontinuierlich ausgeführt.

Sie können das WebJobs-SDK verwenden , um den Code zu vereinfachen, den Sie für allgemeine Aufgaben, die ein WebJob ausführen kann, wie Bildverarbeitung, Warteschlangenverarbeitung, RSS-Aggregation, Dateiverwaltung und Senden von E-Mails, vereinfachen. Das WebJobs-SDK verfügt über integrierte Funktionen zum Arbeiten mit Azure Storage und Service Bus , zum Planen von Aufgaben und zum Behandeln von Fehlern sowie für viele andere gängige Szenarien.

Versionen

Azure WebJobs SDK

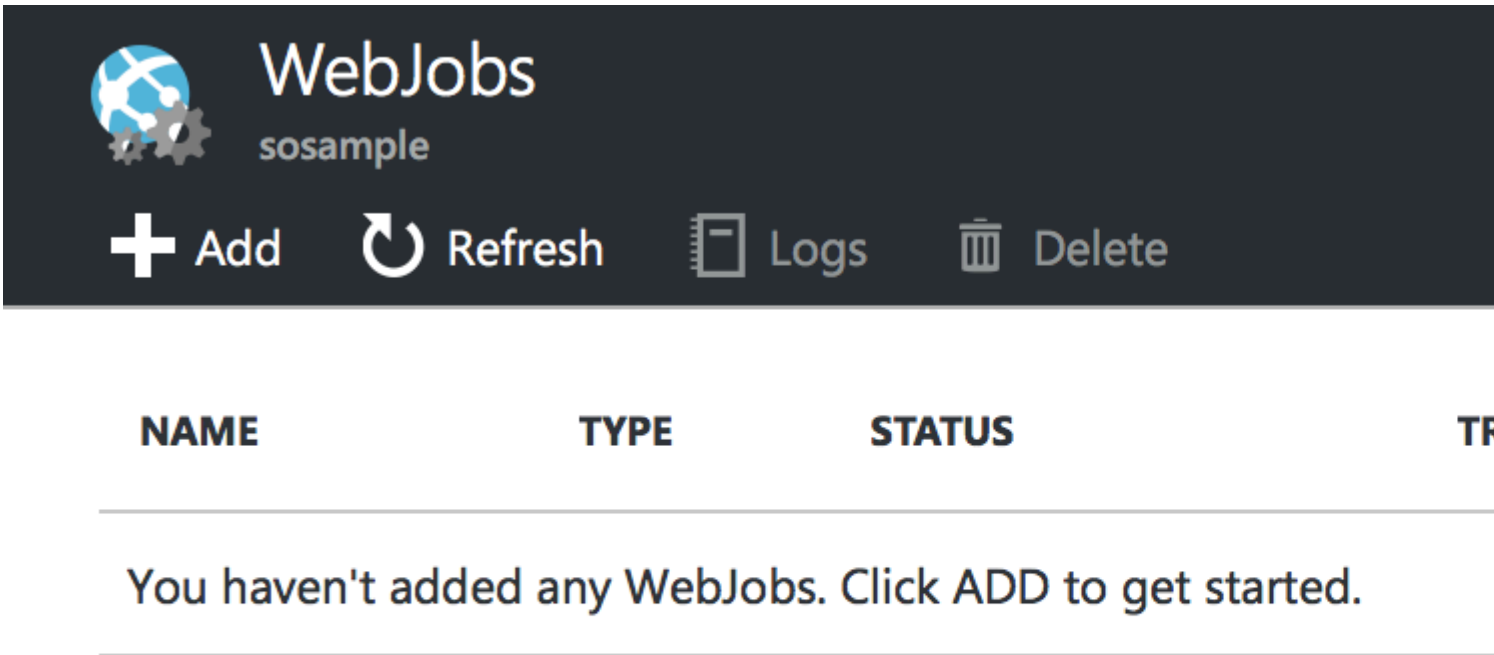
Ausführung	Veröffentlichungsdatum
2.0.0-beta1	2016-07-14
1.1.2	2016-04-22
1.1.1	2016-01-13
1.1.0	2015-11-19
1.1.0-rc1	2015-11-02
1.1.0-beta1	2015-09-16
1.1.0-alpha2	2015-08-12
1.1.0-alpha1	2015-07-10

Ausführung	Veröffentlichungsdatum
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-beta	2014-09-05
0,4,1-beta	2014-08-30
0,4,0-beta	2014-08-21

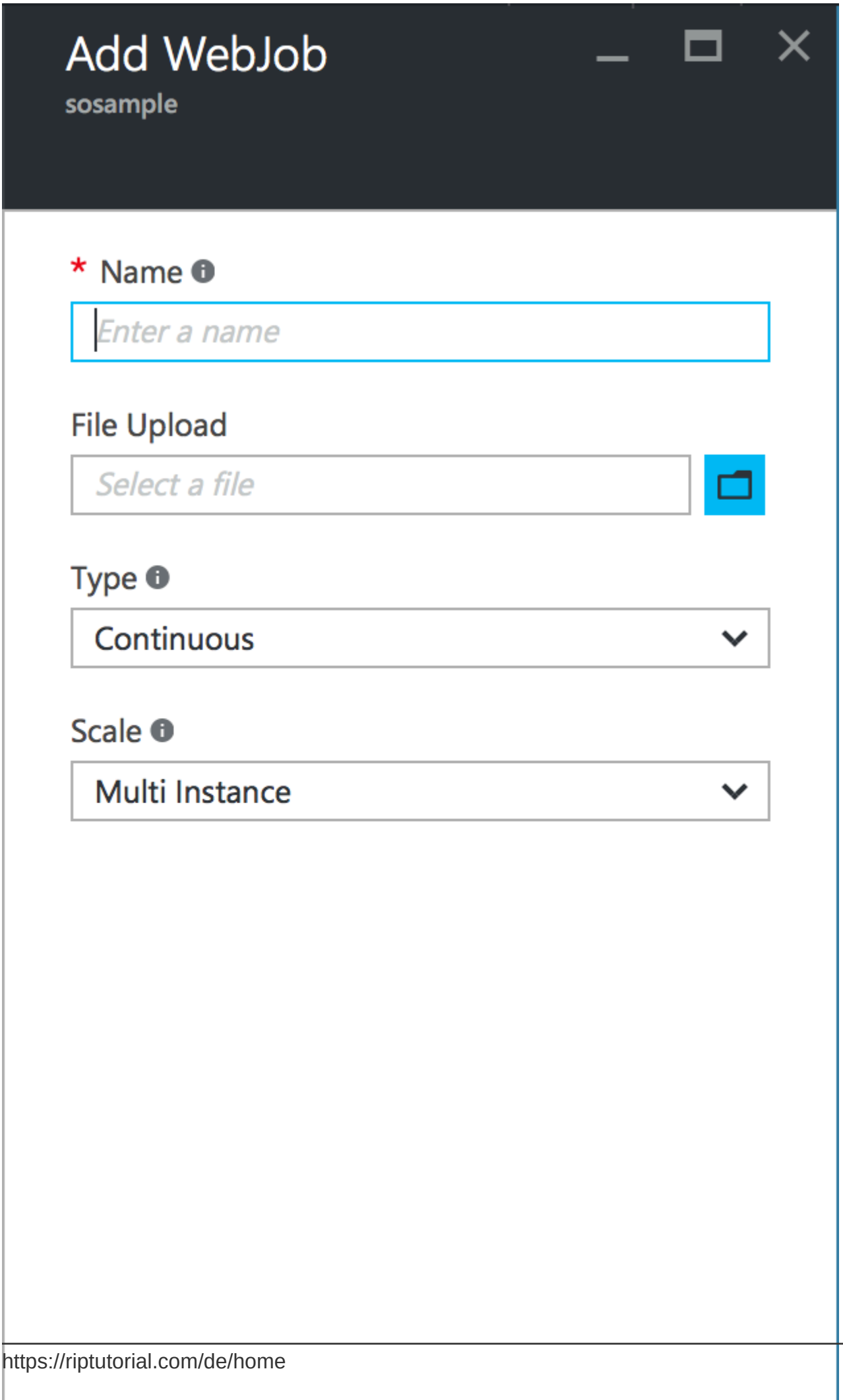
Examples

Erstellen eines WebJob im Azure-Portal

1. Klicken Sie im **Web-App-** Blade des Azure-Portals auf **Alle Einstellungen** > **WebJobs** , um das WebJobs-Blade anzuzeigen:



2. Klicken Sie auf **Hinzufügen** . Das Dialogfeld " **WebJob hinzufügen** " wird angezeigt.


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

einen Namen für den WebJob an. Der Name muss mit einem Buchstaben oder einer Zahl beginnen und darf keine Sonderzeichen außer "-" und "_" enthalten.

4. Wählen Sie im Feld **How to Run** die bevorzugte Option **Continuous** oder **Triggered aus** (der Auslöser kann einen Cron-Zeitplan oder einen WebHook verwenden).
5. In dem **Datei** - **Upload** - Feld, klicken Sie auf das Ordnersymbol und navigieren Sie zu der ZIP - Datei , die das Skript enthält. Die ZIP-Datei sollte Ihre ausführbare Datei (.exe .cmd .bat .sh .php .py .js) sowie alle unterstützenden Dateien enthalten, die zum Ausführen des Programms oder Skripts erforderlich sind.
6. Aktivieren Sie **Erstellen** , um das Skript in Ihre Webanwendung hochzuladen. Der Name, den Sie für den WebJob angegeben haben, wird in der Liste auf dem WebJobs-Blade angezeigt.

Erste Schritte mit Azure-Webjobs online lesen: https://riptutorial.com/de/azure-webjobs/topic/1311/erste-schritte-mit-azure-webjobs

Kapitel 2: Azure Webjobs SDK

Examples

JobHost

Die Azure Webjobs SDK ist ein **Framework** als verteiltes Paket Nuget sollen Ihnen einige **Funktionen** definieren , die von **Trigger** ausgeführt werden und verwenden **Bindungen** zu anderen Azure - Dienste (wie Azure Storage und Service Bus) in einer **deklarativen** Weise.

Das SDK verwendet einen **JobHost** , um Ihre codierten Funktionen zu koordinieren. In einem typischen Szenario ist Ihr Webjob eine Konsolenanwendung, die den JobHost folgendermaßen initialisiert:

```
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();
    }
}
```

Mit der JobHostConfiguration können Sie weitere Einstellungen für verschiedene Auslöser anpassen:

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

Trigger für Warteschlangen

Ein einfaches Beispiel zur Definition einer Funktion, die durch eine Warteschlangennachricht ausgelöst wird:

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

Es unterstützt auch die POCO- Serialisierung:

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

```
}
```

Trigger für Blobs

Ein einfaches Beispiel für eine Funktion, die ausgelöst wird, wenn ein Azure Storage Blob geändert wird:

```
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();

    }
}
```

Triggert nach Zeit

Das SDK unterstützt die Zeit, die basierend auf CRON- Ausdrücken mit **6 Feldern** ausgelöst wird (`{second} {minute} {hour} {day} {month} {day of the week}`). Es erfordert eine **zusätzliche Einstellung** in der `JobHostConfiguration` :

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

Ihre zeitgesteuerten Funktionen reagieren auf diese Syntax:

```
// 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)
{

}
```

Triggert durch Fehler

Fehlerbehandlung ist äußerst wichtig. Wir können Funktionen definieren, die ausgelöst werden sollen, wenn in einer Ihrer ausgelösten Funktionen ein Ausführungsfehler auftritt:

```
//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);
}
```

Es ist besonders nützlich, um die Fehlerbehandlung zu zentralisieren.

ErrorTrigger erfordert eine **zusätzliche Einstellung** in der JobHostConfiguration:

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

Sie müssen auch das NuGet-Paket **Microsoft.Azure.WebJobs.Extensions** installieren.

Skalieren

Azure-Webjobs werden auf einem Azure App Service ausgeführt. Wenn wir unseren App Service horizontal skalieren (neue Instanzen hinzufügen), hat **jede Instanz einen eigenen JobHost** .

Beachten Sie, dass dies nur für WebJobs gilt, die im **kontinuierlichen** Modus ausgeführt werden. On-Demand- und geplante WebJobs sind von der horizontalen Skalierung nicht betroffen. Sie führen immer eine einzige Instanz aus.

Wenn Sie eine Warteschlangenmeldung für die fortlaufende WebJob-Verarbeitung haben und den App Service Plan auf drei Instanzen skalieren, werden drei Instanzen des WebJob ausgeführt.

Es gibt möglicherweise Webjobs, die Sie in einer einzelnen Instanz ausführen möchten, da Sie möglicherweise sicherstellen müssen, dass genau eine Verarbeitungspipeline vorhanden ist. Für diese WebJobs können Sie das **Singleton-** Attribut hinzufügen.

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

Dies wird durch Azure-Blob-Leases für verteiltes Sperren erreicht.

Protokolle schreiben

Das WebJobs-Dashboard zeigt Protokolle an zwei Stellen an: der Seite für den WebJob und der Seite für einen bestimmten WebJob-Aufruf.

Ausgaben von Console-Methoden, die Sie in einer Funktion oder in der `Main()` Methode aufrufen, werden auf der Dashboard-Seite für den WebJob angezeigt, nicht auf der Seite für einen bestimmten Methodenaufruf. Die Ausgabe des TextWriter-Objekts, die Sie von einem Parameter in Ihrer Methodensignatur erhalten, wird auf der Dashboard-Seite für einen Methodenaufruf angezeigt.

Verwenden Sie `Console.Out` (erstellt als INFO markierte Protokolle) und `Console.Error` (erstellt als ERROR markierte Protokolle), um Anwendungsprotokollierungsprotokolle zu schreiben.

```
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);
}
```

Was führt zu folgenden Meldungen im Dashboard für den 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!
```

Und diese Nachricht auf der Dashboard-Seite für die Methode:

```
TextWriter - Hello world!
```

Abhängigkeitseinspritzung mit Ninject

Das folgende Beispiel zeigt, wie Sie Dependency Injection mit Ninject als IoC-Container einrichten.

Fügen Sie zunächst eine CustomModule-Klasse zu Ihrem WebJob-Projekt hinzu, und fügen Sie dort alle Abhängigkeitsbindungen hinzu.

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

Dann erstellen Sie eine JobActivator-Klasse:

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

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

Wenn Sie den JobHost in der Main-Funktion der Programmklasse einrichten, fügen Sie den JobActivator zur JobHostConfiguration hinzu

```
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();
    }
}
```

Fügen Sie schließlich in der Functions.cs-Klasse Ihre Dienste ein.

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

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

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

Azure Webjobs SDK online lesen: https://riptutorial.com/de/azure-webjobs/topic/2662/azure-webjobs-sdk

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Azure-Webjobs	Community, gbellmann
2	Azure Webjobs SDK	gbellmann, juunas, lopezbertoni, Matias Quaranta