FREE eBook

LEARNING azure-webjobs

Free unaffiliated eBook created from
Stack Overflow contributors.

#azure-webjobs

Table of Contents

About

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

Chapter 1: Getting started with azure-webjobs

Remarks

Azure WebJobs provide an easy way to run scripts or programs as background processes in the context of an App Service web app, API app, or mobile app. You can upload and run an executable file such as:

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

These programs run as WebJobs on a schedule (cron) or continuously.

You can use the WebJobs SDK is to simplify the code you write for common tasks that a WebJob can perform, such as image processing, queue processing, RSS aggregation, file maintenance, and sending emails. The WebJobs SDK has built-in features for working with Azure Storage and Service Bus, for scheduling tasks and handling errors, and for many other common scenarios.

Versions

Azure WebJobs SDK

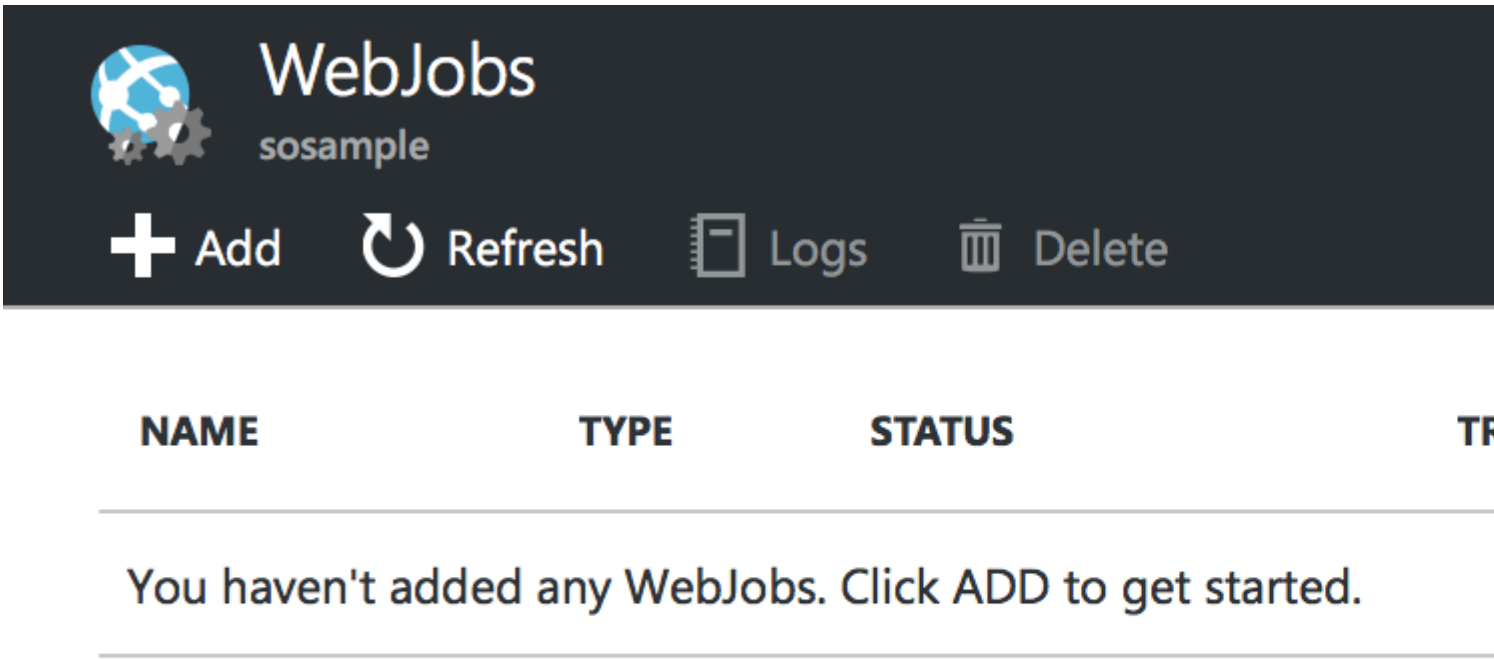
Version	Release Date
2.0.0-beta1	2016-07-14
1.1.2	2016-04-22
1.1.1	2016-01-13
1.1.0	2015-11-19
1.1.0-rc1	2015-11-02
1.1.0-beta1	2015-09-16
1.1.0-alpha2	2015-08-12
1.1.0-alpha1	2015-07-10

Version	Release Date
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

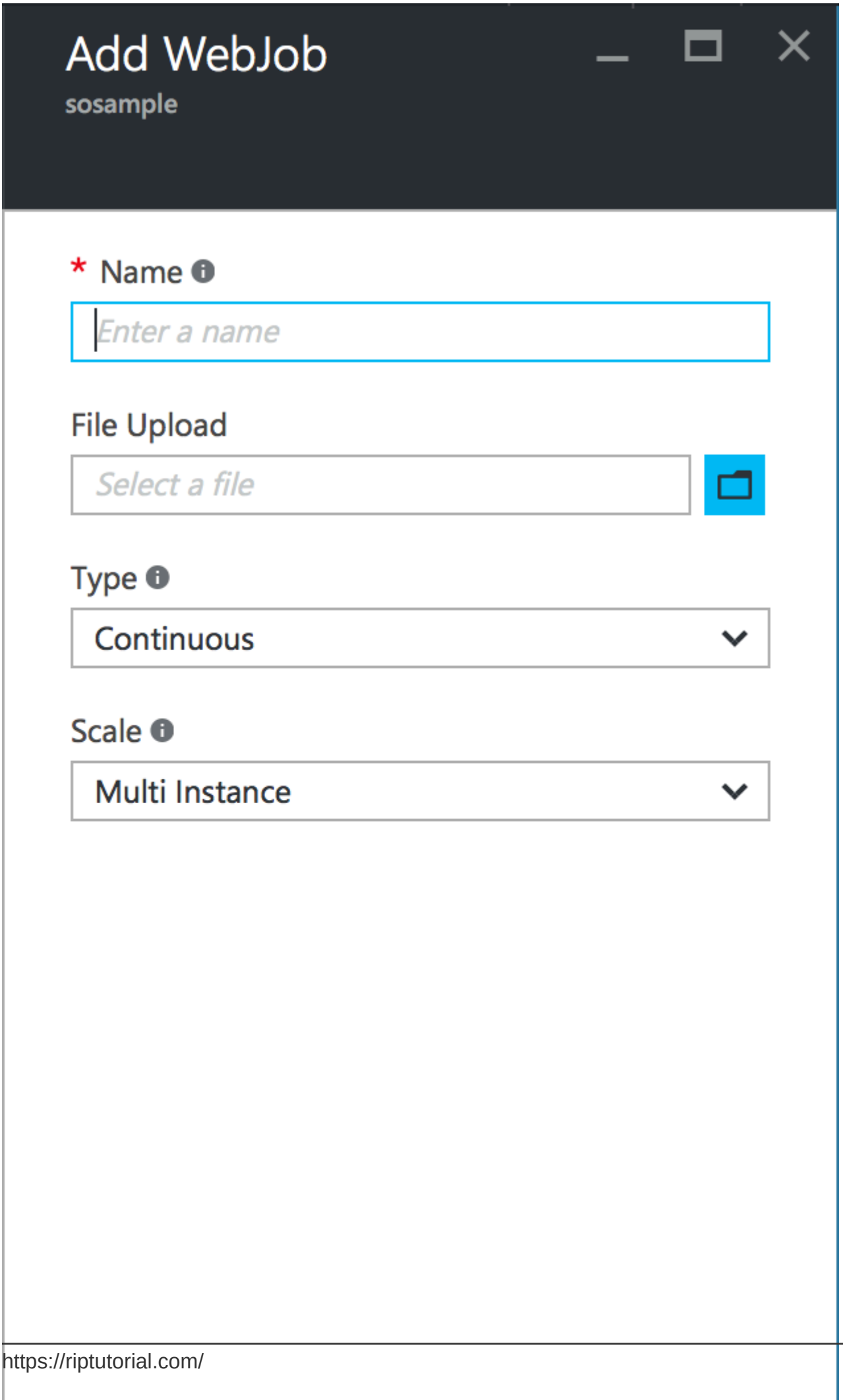
Examples

Creating a WebJob in the Azure Portal

1. In the **Web App** blade of the Azure Portal, click **All settings** > **WebJobs** to show the WebJobs blade:



2. Click **Add**. The **Add WebJob** dialog appears.


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

, provide a name for the WebJob. The name must start with a letter or a number and cannot contain any special characters other than "-" and "_".

4. In the **How to Run** box, choose your preferred option **Continuous** or **Triggered** (the trigger can be using a cron schedule or a WebHook).
5. In the **File Upload** box, click the folder icon and browse to the zip file that contains your script. The zip file should contain your executable (.exe .cmd .bat .sh .php .py .js) as well as any supporting files needed to run the program or script.
6. Check **Create** to upload the script to your web app. The name you specified for the WebJob appears in the list on the WebJobs blade.

Read Getting started with azure-webjobs online: https://riptutorial.com/azure-webjobs/topic/1311/getting-started-with-azure-webjobs

Chapter 2: Azure Webjobs SDK

Examples

JobHost

The Azure Webjobs SDK is a **framework** distributed as a Nuget package aimed at helping you define **Functions** that are run by **Triggers** and use **Bindings** to other Azure services (like Azure Storage and Service Bus) in a **declarative** fashion.

The SDK uses a **JobHost** to coordinate your coded Functions. In a tipical scenario, your Webjob is a Console Application that initializes the JobHost this way:

```
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();
    }
}
```

The JobHostConfiguration lets you personalize more settings for different triggers:

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

Triggers for Queues

A simple example defining a Function that gets triggered by a Queue message:

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

It also supports POCO serialization:

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

Triggers for Blobs

A simple example of a Function that gets triggered when a Azure Storage Blob is modified:

```
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();

    }
}
```

Triggers by time

The SDK supports time triggered based on CRON expressions with **6 fields** (`{second} {minute} {hour} {day} {month} {day of the week}`). It requires an **extra setting** on the `JobHostConfiguration`:

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

Your time triggered functions respond to this 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)
{

}
```

Triggers by errors

Error handling is extremely important, we can define functions to be triggered when an execution error happens in one of your triggered functions:

```
//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);
}
```

It's specially useful for centralizing error handling.

ErrorTrigger requires an **additional setting** on the JobHostConfiguration:

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

You also must install the NuGet package **Microsoft.Azure.WebJobs.Extensions**.

Scaling

Azure Webjobs run on an Azure App Service. If we scale our App Service horizontally (add new instances), **each instance** will have **its own JobHost**.

Note that this only applies to WebJobs running in **Continuous** mode. On-demand and scheduled WebJobs are not affected by horizontal scaling, they always run a single instance.

If you have a continuous WebJob processing queue messages, and you scale the App Service Plan to 3 instances, you will have 3 instances of the WebJob running.

There might be WebJobs that you want to run in a single instance, because you might need to ensure that exactly one processing pipeline exists. For those WebJobs, you can add the **Singleton** attribute.

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

This is achieved by Azure Blob Leases for distributed locking.

Writing Logs

The WebJobs Dashboard shows logs in two places: the page for the WebJob, and the page for a particular WebJob invocation.

Output from Console methods that you call in a function or in the `Main()` method appears in the Dashboard page for the WebJob, not in the page for a particular method invocation. Output from the TextWriter object that you get from a parameter in your method signature appears in the Dashboard page for a method invocation.

To write application tracing logs, use `Console.Out` (creates logs marked as INFO) and `Console.Error` (creates logs marked as 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);
}
```

Which will result in these messages in the Dashboard for the 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!
```

And this message in the Dashboard page for the method:

```
TextWriter - Hello world!
```

Dependency Injection using Ninject

The following example shows how to set up Dependency Injection using Ninject as an IoC container.

First add a CustomModule class to your WebJob project, and add any dependency bindings there.

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

Then create a JobActivator class:

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

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

When you set up the JobHost in the Program class' Main function, add the JobActivator to the 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();
    }
}
```

Finally in the Functions.cs class, inject your 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);
    }
}
```

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

Credits

S. No	Chapters	Contributors
1	Getting started with azure-webjobs	Community, gbellmann
2	Azure Webjobs SDK	gbellmann, juunas, lopezbertoni, Matias Quaranta