**EBook Gratis**

# APRENDIZAJE azure-webjobs

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#azure-webjobs

# Tabla de contenido

# Acerca de

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

# Capítulo 1: Empezando con azure-webjobs

## Observaciones

Azure WebJobs proporciona una manera fácil de ejecutar scripts o programas como procesos en segundo plano en el contexto de una aplicación web de App Service, una aplicación API o una aplicación móvil. Puede cargar y ejecutar un archivo ejecutable como:

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

Estos programas se ejecutan como WebJobs en un horario (cron) o continuamente.

Puede usar el SDK de WebJobs para simplificar el código que escribe para las tareas comunes que puede realizar un WebJob, como el procesamiento de imágenes, el procesamiento de colas, la agregación de RSS, el mantenimiento de archivos y el envío de correos electrónicos. El SDK de WebJobs tiene características integradas para trabajar con Azure Storage y Service Bus , para programar tareas y manejar errores, y para muchos otros escenarios comunes.

## Versiones

## Azure WebJobs SDK

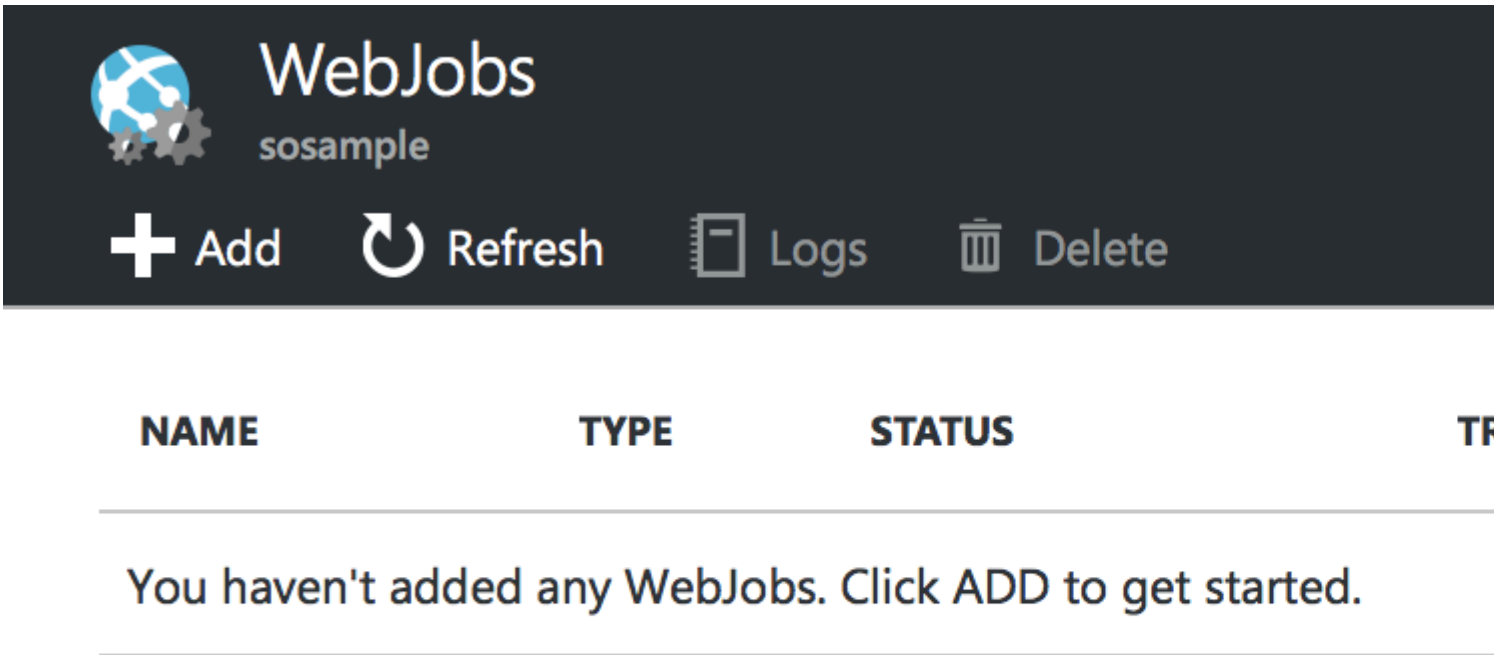
| Versión | Fecha de lanzamiento |
|---|---|
| 2.0.0-beta1 | 2016-07-14 |
| 1.1.2 | 2016-04-22 |
| 1.1.1 | 2016-01-13 |
| 1.1.0 | 2015-11-19 |
| 1.1.0-rc1 | 2015-11-02 |
| 1.1.0-beta1 | 2015-09-16 |
| 1.1.0-alfa2 | 2015-08-12 |
| 1.1.0-alfa1 | 2015-07-10 |

| Versión | Fecha de lanzamiento |
|---|---|
| 1.0.1 | 2015-03-19 |
| 1.0.1-alfa1 | 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 |

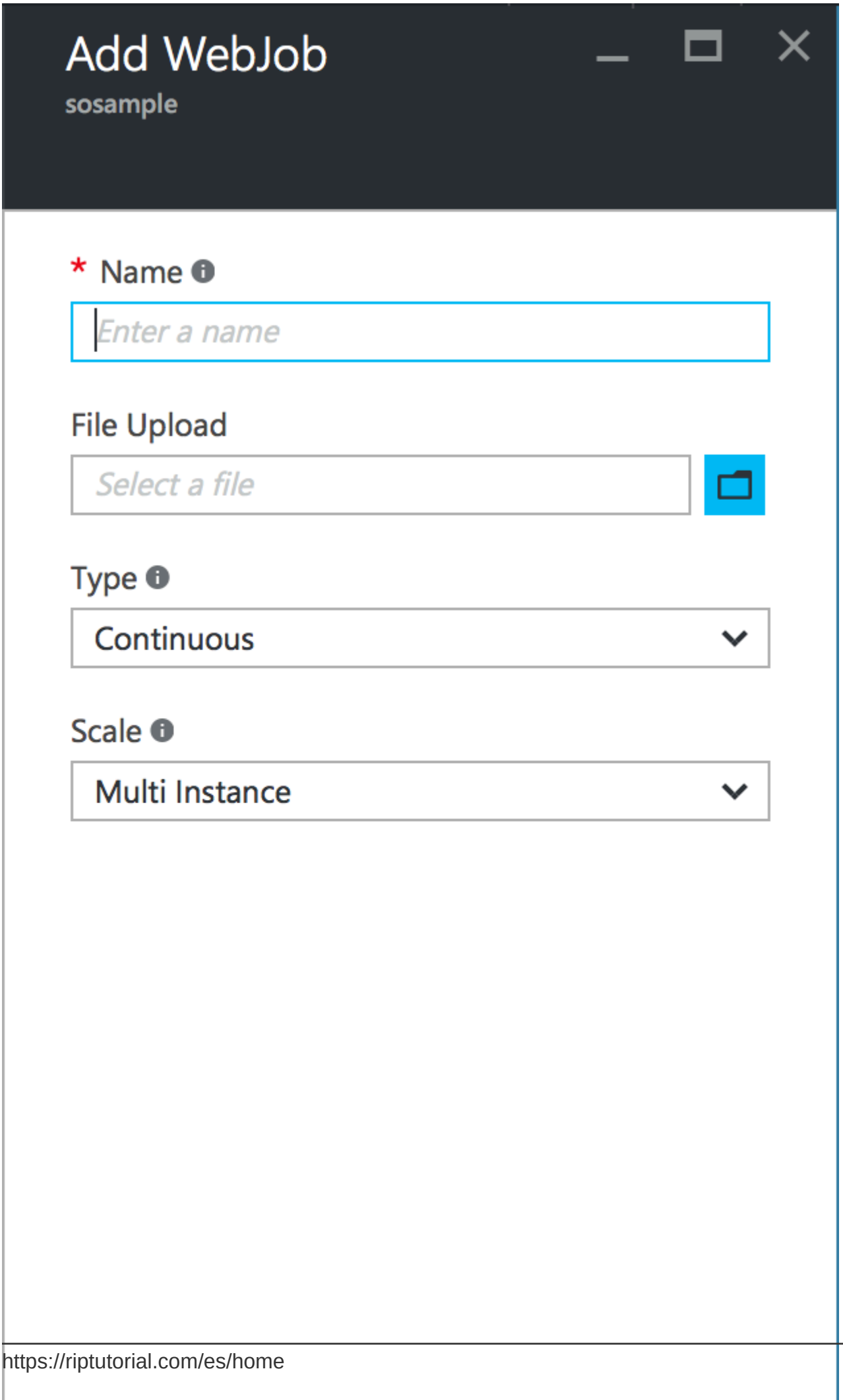
# Examples

### Creando un WebJob en el Portal de Azure

1. En la hoja de **aplicación web** del Portal de Azure , haga clic en **Todas las configuraciones** > **WebJobs** para mostrar la hoja de WebJobs:



2. Haga clic en **Agregar** . **Aparece el** cuadro de diálogo **Agregar WebJob** .


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

, proporcione un nombre para el WebJob. El nombre debe comenzar con una letra o un número y no puede contener ningún carácter especial que no sea "-" y "_".
4. En el cuadro **Cómo ejecutar** , elija su opción preferida **Continuo** o **Activado** (el activador puede usar un cronograma o un WebHook).
5. En el cuadro **Cargar archivo** , haga clic en el icono de la carpeta y busque el archivo zip que contiene su script. El archivo zip debe contener su archivo ejecutable (.exe .cmd .bat .sh .php .py .js) así como cualquier archivo de soporte necesario para ejecutar el programa o el script.
6. Marque **Crear** para cargar el script en su aplicación web. El nombre que especificó para el WebJob aparece en la lista en la hoja de WebJobs.

Lea Empezando con azure-webjobs en línea: https://riptutorial.com/es/azure-webjobs/topic/1311/empezando-con-azure-webjobs

# Capítulo 2: Azk Webjobs SDK

## Examples

### JobHost

El SDK de Azure Webjobs es un **marco** distribuido como un paquete de Nuget destinado a ayudarlo a definir las **funciones** que son ejecutadas por **Triggers** y usan los **enlaces** a otros servicios de Azure (como Azure Storage y Service Bus) de manera **declarativa** .

El SDK usa un **JobHost** para coordinar sus funciones codificadas. En un escenario típico, su Webjob es una aplicación de consola que inicializa JobHost de esta manera:

```
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 le permite personalizar más configuraciones para diferentes disparadores:

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

### Activadores de colas

Un ejemplo simple que define una función que se desencadena por un mensaje de cola:

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

También soporta la serialización POCO :

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

### Disparadores para Blobs

Un ejemplo simple de una función que se activa cuando se modifica un blob de almacenamiento de 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();

    }
}
```

## Gatillos por tiempo

El SDK admite el tiempo activado basado en expresiones CRON con **6 campos** ( `{second} {minute} {hour} {day} {month} {day of the week}` ). Requiere una **configuración adicional** en la **configuración** de `JobHostConfiguration` :

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

Las funciones activadas por el tiempo responden a esta sintaxis:

```
// 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)
{

}
```

## Disparadores por errores.

El manejo de errores es extremadamente importante, podemos definir las funciones que se activarán cuando ocurra un error de ejecución en una de sus funciones activadas:

```
//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 especialmente útil para centralizar el manejo de errores.

ErrorTrigger requiere una **configuración adicional** en JobHostConfiguration:

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

También debe instalar el paquete NuGet **Microsoft.Azure.WebJobs.Extensions** .

## Escalada

Azure Webjobs se ejecuta en un servicio de aplicaciones de Azure. Si escalamos nuestro Servicio de aplicaciones horizontalmente (agregamos nuevas instancias), **cada instancia** tendrá **su propio JobHost** .

Tenga en cuenta que esto solo se aplica a los WebJobs que se ejecutan en modo **Continuo** . Los WebJobs a pedido y programados no se ven afectados por la escala horizontal, siempre ejecutan una sola instancia.

Si tiene mensajes de cola de procesamiento de WebJob continuos y escala el Plan de Servicio de Aplicaciones a 3 instancias, tendrá 3 instancias de WebJob en ejecución.

Es posible que haya WebJobs que desee ejecutar en una sola instancia, ya que es posible que deba asegurarse de que exista exactamente una canalización de procesamiento. Para esos WebJobs, puede agregar el atributo **Singleton** .

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

Esto se logra mediante Azure Blob Leases para el bloqueo distribuido.

## Escribiendo Registros

El panel de control de WebJobs muestra los registros en dos lugares: la página para el WebJob y la página para una invocación particular de WebJob.

La salida de los métodos de la consola a los que llama en una función o en el método `Main()` aparece en la página Panel de control para el WebJob, no en la página para la invocación de un método en particular. La salida del objeto TextWriter que obtiene de un parámetro en su firma de método aparece en la página Panel para una invocación de método.

Para escribir registros de seguimiento de aplicaciones, use `Console.Out` (crea registros marcados como INFO) y `Console.Error` (crea registros marcados como 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);
}
```

Lo que resultará en estos mensajes en el Tablero para el 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!
```

Y este mensaje en la página de Dashboard para el método:

```
TextWriter - Hello world!
```

## Inyección de dependencia usando Ninject

El siguiente ejemplo muestra cómo configurar la inyección de dependencia utilizando Ninject como un contenedor IoC.

Primero agregue una clase CustomModule a su proyecto WebJob, y agregue cualquier enlace de dependencia allí.

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

Luego crea una clase JobActivator:

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

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

Cuando configure el JobHost en la función principal de la clase de programa, agregue el JobActivator a la configuración de JobHost

```
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();
    }
}
```

Finalmente en la clase Functions.cs, inyecte sus servicios.

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

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

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

Lea Azk Webjobs SDK en línea: https://riptutorial.com/es/azure-webjobs/topic/2662/azk-webjobs-sdk

# Creditos

| S. No | Capítulos | Contributors |
|---|---|---|
| 1 | Empezando con azure-webjobs | Community, gbellmann |
| 2 | Azk Webjobs SDK | gbellmann, juunas, lopezbertoni, Matias Quaranta |