EBook Gratuito

APPRENDIMENTO azure-webjobs

Free unaffiliated eBook created from
Stack Overflow contributors.

#azure-webjobs

Sommario

Di

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

Capitolo 1: Iniziare con azure-webjobs

Osservazioni

I WebJob di Azure forniscono un modo semplice per eseguire script o programmi come processi in background nel contesto di un'app Web del servizio app, un'app API o un'app mobile. Puoi caricare ed eseguire un file eseguibile come:

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

Questi programmi vengono eseguiti come WebJob su una pianificazione (cron) o continuamente.

È possibile utilizzare l' SDK di WebJob per semplificare il codice che si scrive per le attività comuni che un WebJob può eseguire, come l'elaborazione delle immagini, l'elaborazione della coda, l'aggregazione di RSS, la manutenzione dei file e l'invio di e-mail. L'SDK di WebJobs ha funzionalità integrate per lavorare con Archiviazione e Bus di servizio di Azure, per pianificare attività e gestire errori e per molti altri scenari comuni.

Versioni

SDK di Azure WebJobs

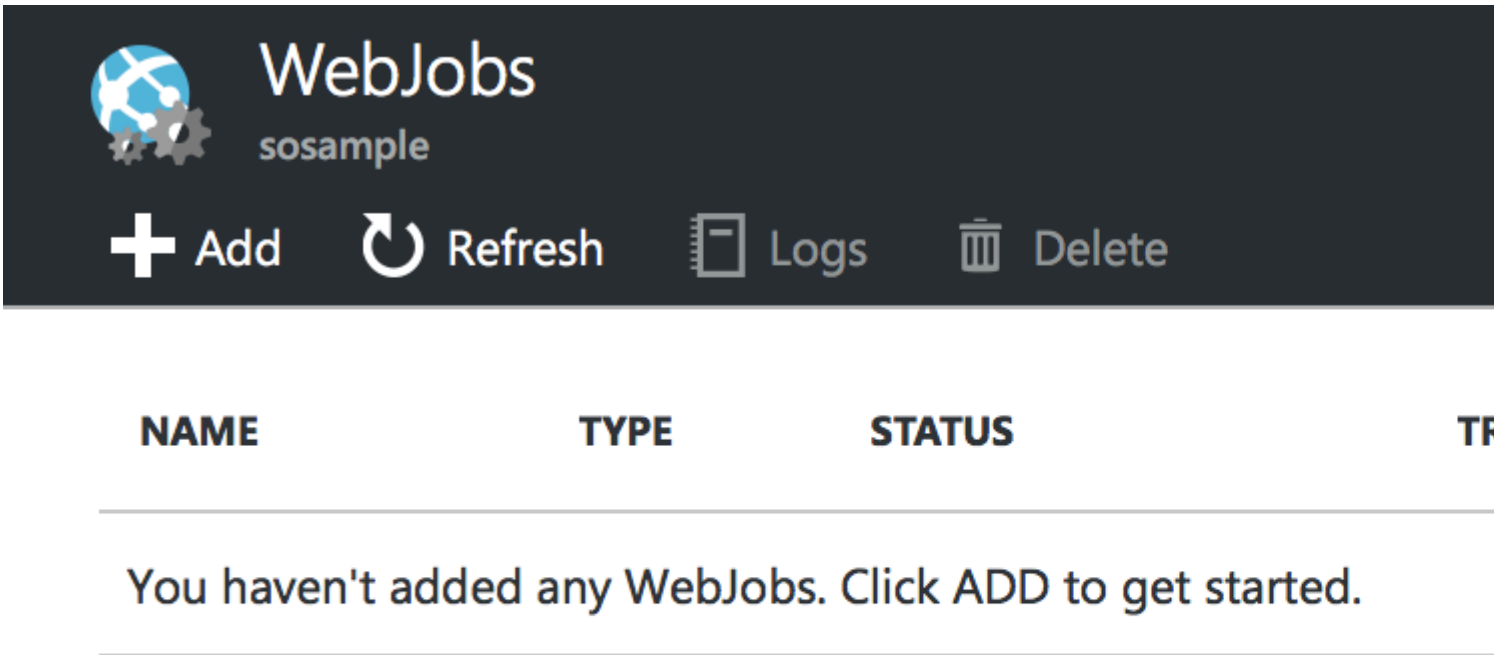
Versione	Data di rilascio
2.0.0-beta1	2016/07/14
1.1.2	2016/04/22
1.1.1	2016/01/13
1.1.0	2015/11/19
1.1.0-rc1	2015/11/02
1.1.0-beta1	2015/09/16
1.1.0-alpha2	2015/08/12
1.1.0-alpha1	2015/07/10

Versione	Data di rilascio
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

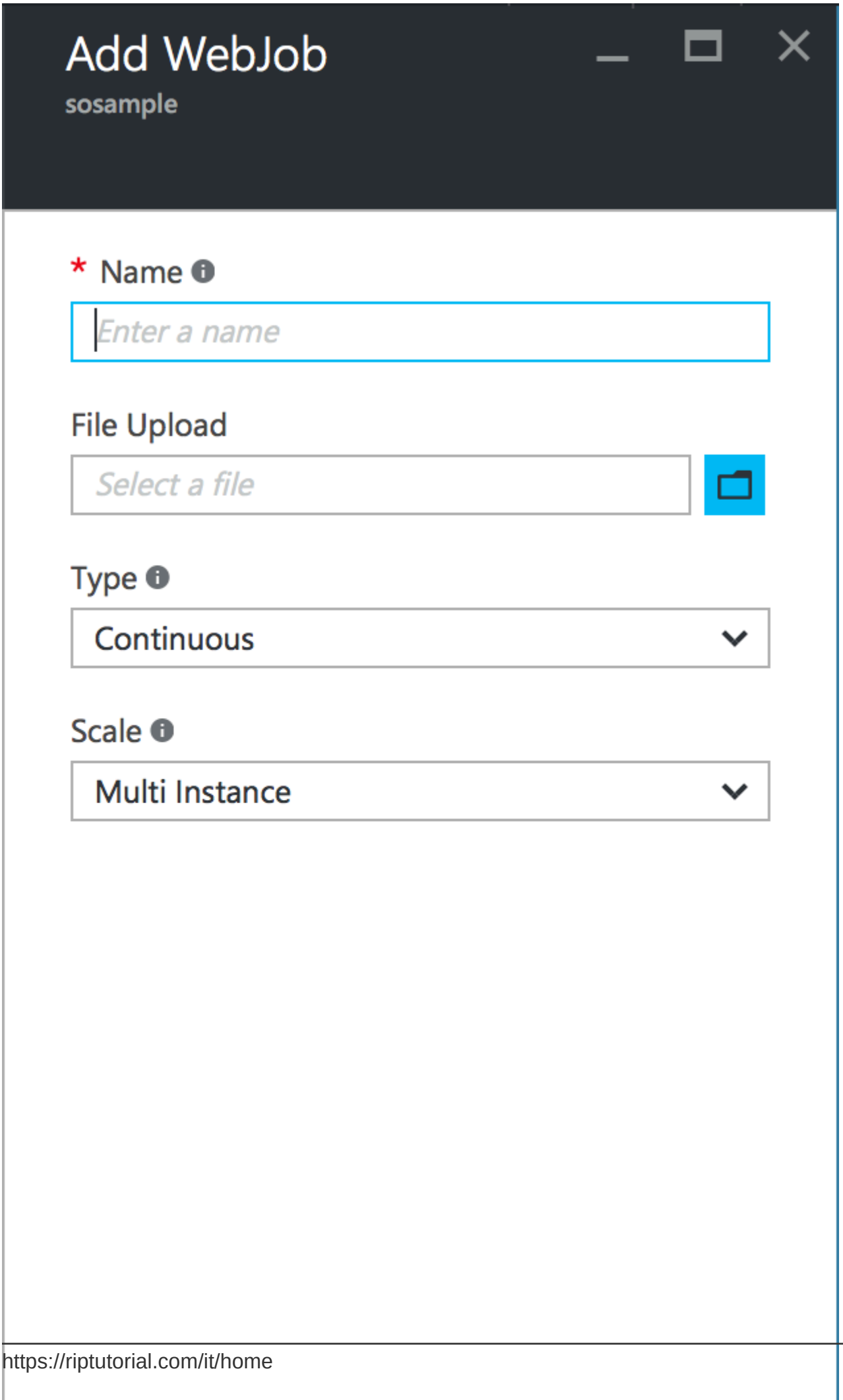
Examples

Creazione di un WebJob nel portale di Azure

1. Nel **pannello App Web** di Azure Portal , fare clic su **Tutte le impostazioni** > **WebJob** per visualizzare il blade WebJobs:



2. Fai clic su **Aggiungi** . **Viene** visualizzata la finestra di dialogo **Aggiungi WebJob** .


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

, fornire un nome per il WebJob. Il nome deve iniziare con una lettera o un numero e non può contenere caratteri speciali diversi da "-" e "_".

4. Nella casella **Come eseguire,** scegliere l'opzione preferita **continua** o **Innescato** (il trigger può essere utilizzando una pianificazione cron o un WebHook).
5. Nella casella **Caricamento file** , fai clic sull'icona della cartella e cerca il file zip che contiene il tuo script. Il file zip deve contenere il file eseguibile (.exe .cmd .bat .sh .php .py .js) e tutti i file di supporto necessari per eseguire il programma o lo script.
6. Seleziona **Crea** per caricare lo script nella tua app Web. Il nome specificato per WebJob viene visualizzato nell'elenco sul blade WebJobs.

Leggi Iniziare con azure-webjobs online: https://riptutorial.com/it/azure-webjobs/topic/1311/iniziare-con-azure-webjobs

Capitolo 2: SDK di Webjobs di Azure

Examples

JobHost

L'SDK Webjobs di Azure è un **framework** distribuito come pacchetto Nuget che aiuta a definire le **funzioni** eseguite da **Trigger** e ad utilizzare **Bind** per altri servizi di Azure (come Azure Storage e Service Bus) in modo **dichiarativo** .

L'SDK utilizza un **JobHost** per coordinare le tue funzioni codificate. In uno scenario tipico, il tuo Webjob è un'applicazione console che inizializza JobHost in questo modo:

```
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 consente di personalizzare più impostazioni per diversi trigger:

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

Trigger per le code

Un semplice esempio che definisce una funzione che viene attivata da un messaggio in coda:

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

Supporta anche la serializzazione POCO :

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

Trigger per Blob

Un semplice esempio di una funzione che viene attivata quando viene modificato un BLOB di archiviazione di 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();

    }
}
```

Trigger di volta

L'SDK supporta il tempo attivato in base alle espressioni CRON con **6 campi** (`{second} {minute} {hour} {day} {month} {day of the week}`). Richiede un'impostazione **aggiuntiva** su `JobHostConfiguration` :

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

Le funzioni attivate per il tempo rispondono a questa sintassi:

```
// 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)
{

}
```

Trigger per errori

La gestione degli errori è estremamente importante, possiamo definire le funzioni da attivare quando si verifica un errore di esecuzione in una delle funzioni attivate:

```
//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);
}
```

È particolarmente utile per centralizzare la gestione degli errori.

ErrorTrigger richiede un'impostazione **aggiuntiva** su JobHostConfiguration:

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

È inoltre necessario installare il pacchetto NuGet **Microsoft.Azure.WebJobs.Extensions** .

scalata

I Webjobs di Azure vengono eseguiti su un servizio app di Azure. Se ridimensioniamo il nostro servizio app orizzontalmente (aggiungi nuove istanze), **ogni istanza** avrà **il proprio JobHost** .

Si noti che questo si applica solo ai WebJob in esecuzione in modalità **Continua** . I WebJob su richiesta e pianificati non sono interessati dal ridimensionamento orizzontale, eseguono sempre una singola istanza.

Se si dispone di una coda di elaborazione WebJob continua e si ridimensiona il piano del servizio app su 3 istanze, si avranno 3 istanze del WebJob in esecuzione.

Potrebbero esserci WebJob che si desidera eseguire in una singola istanza, poiché potrebbe essere necessario assicurarsi che esista esattamente una pipeline di elaborazione. Per quei WebJob, è possibile aggiungere l'attributo **Singleton** .

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

Questo risultato è ottenuto dai contratti di locazione BLOB di Azure per il blocco distribuito.

Registri di scrittura

La dashboard di WebJobs mostra i registri in due punti: la pagina per il WebJob e la pagina per una particolare chiamata WebJob.

L'output dai metodi Console chiamati in una funzione o nel metodo `Main()` viene visualizzato nella pagina Dashboard per WebJob, non nella pagina per un particolare richiamo del metodo. L'output dell'oggetto TextWriter ottenuto da un parametro nella firma del metodo viene visualizzato nella pagina Dashboard per un richiamo del metodo.

Per scrivere i registri di traccia dell'applicazione, utilizzare `Console.Out` (crea i log contrassegnati come INFO) e `Console.Error` (crea i log contrassegnati come 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);
}
```

Quale risultato in questi messaggi in Dashboard per 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!
```

E questo messaggio nella pagina Dashboard per il metodo:

```
TextWriter - Hello world!
```

Iniezione delle dipendenze con Ninject

L'esempio seguente mostra come configurare Dependency Injection usando Ninject come contenitore IoC.

Innanzitutto aggiungi una classe CustomModule al tuo progetto WebJob e aggiungi i collegamenti delle dipendenze.

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

Quindi creare una classe JobActivator:

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

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

Quando si imposta JobHost nella funzione principale della classe Program, aggiungere JobActivator a 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();
    }
}
```

Infine, nella classe Functions.cs, inserisci i tuoi servizi.

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

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

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

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

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con azure-webjobs	Community, gbellmann
2	SDK di Webjobs di Azure	gbellmann, juunas, lopezbertoni, Matias Quaranta