Бесплатная электронная книга

УЧУСЬ

azure-webjobs

Free unaffiliated eBook created from
Stack Overflow contributors.

#azure-webjobs

Около

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

глава 1: Начало работы с azure-webjobs

замечания

Azure WebJobs предоставляют простой способ запускать сценарии или программы в качестве фоновых процессов в контексте веб-приложения App App, приложения API или мобильного приложения. Вы можете загрузить и запустить исполняемый файл, например:

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

Эти программы выполняются как WebJobs по расписанию (cron) или непрерывно.

Вы можете использовать SDK WebJobs, чтобы упростить код, который вы пишете, для общих задач, которые может выполнять WebJob, таких как обработка изображений, обработка очереди, агрегация RSS, обслуживание файлов и отправка писем. SDK WebJobs имеет встроенные функции для работы с Azure Storage и Service Bus для планирования задач и ошибок обработки, а также для многих других распространенных сценариев.

Версии

Azure WebJobs SDK

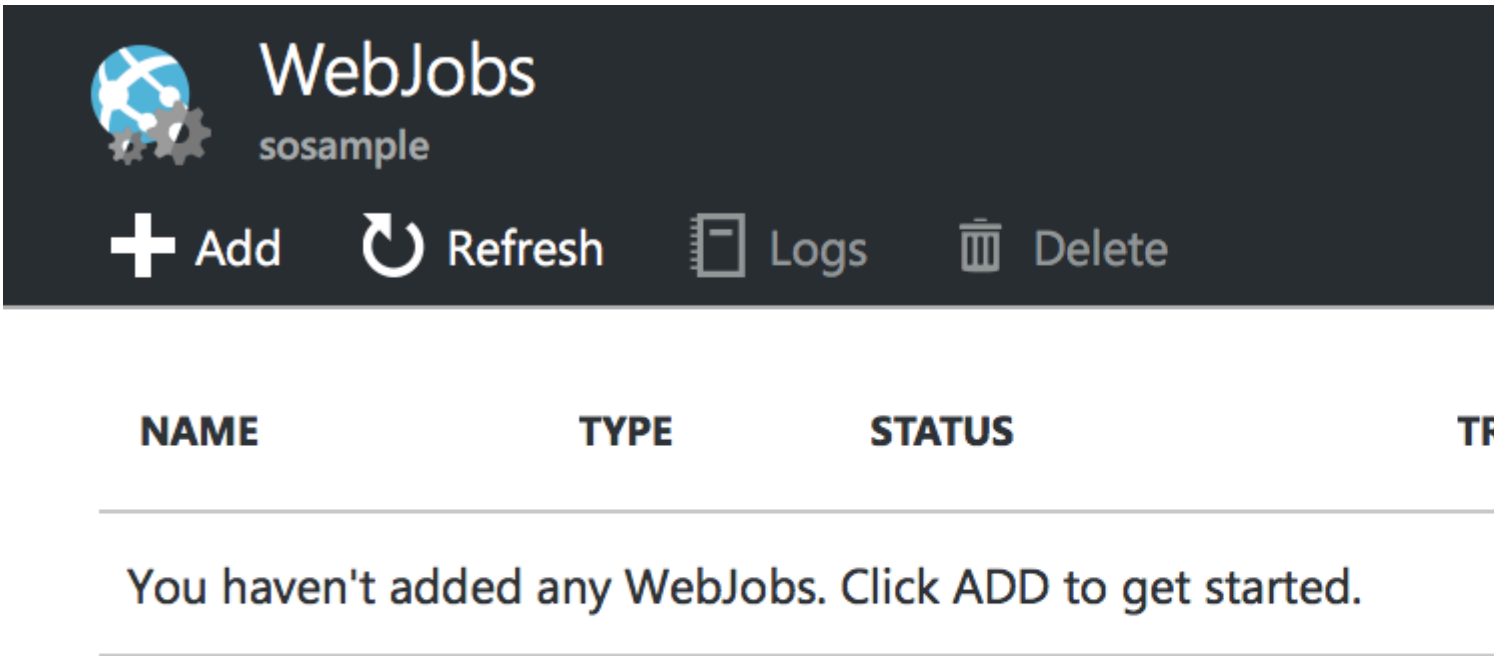
Версия	Дата выхода
2.0.0-beta1	2016-07-14
1.1.2	2016-04-22
1.1.1	2016-01-13
1.1.0	2015-11-19
1.1.0-rc1	2015-11-02
1.1.0-beta1	2015-09-16
1.1.0-альфа2	2015-08-12

Версия	Дата выхода
1.1.0-альфа1	2015-07-10
1.0.1	2015-03-19
1.0.1-альфа1	2015-02-18
1.0.0	2014-10-17
1.0.0-rc1	2014-09-22
0.6.0-бета	2014-09-13
0.5.0-бета	2014-09-05
0.4.1-бета	2014-08-30
0.4.0-бета	2014-08-21

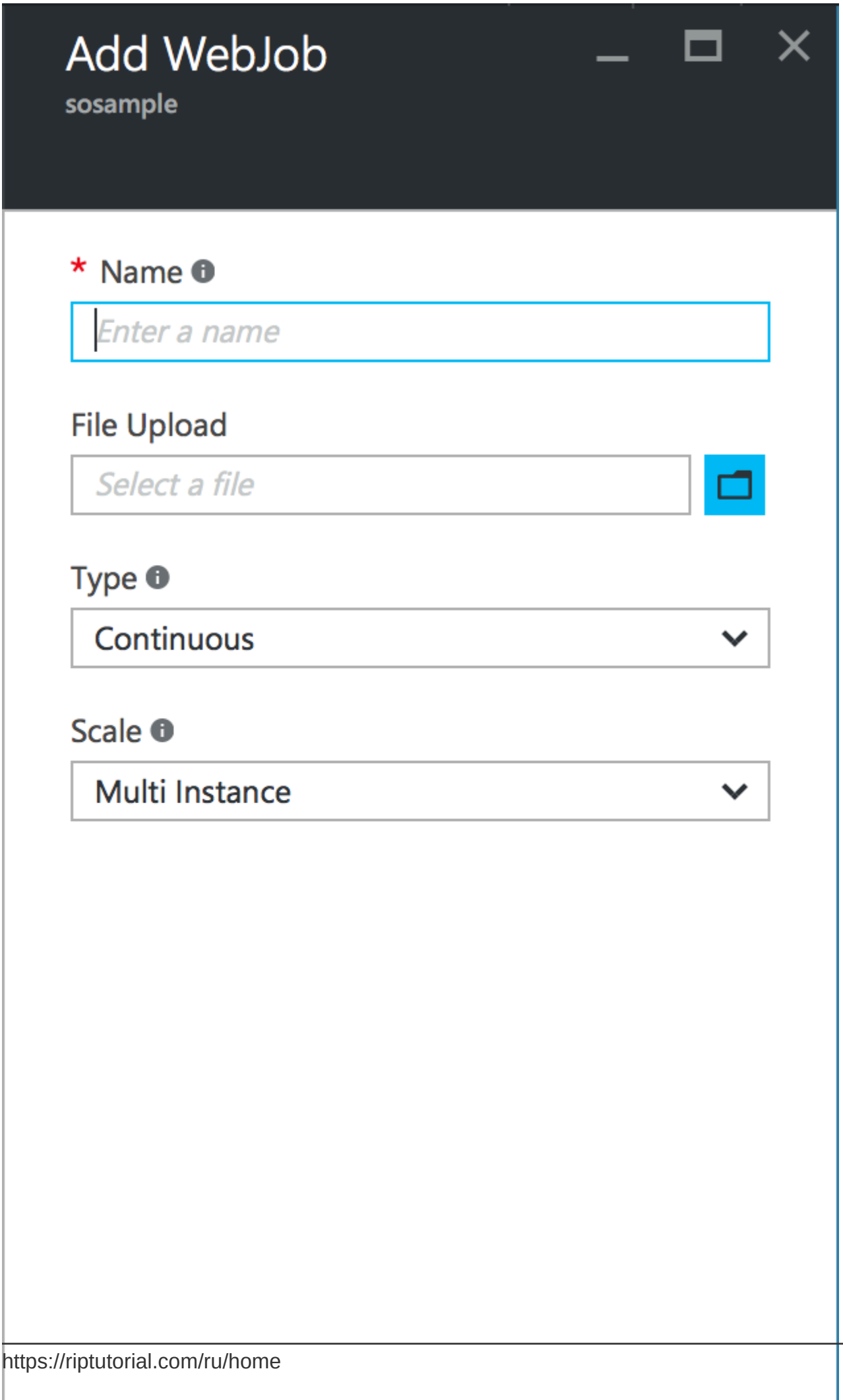
Examples

Создание веб-сайта на Azure Portal

1. В клике **веб-приложения** Azure Portal нажмите « **Все настройки**» > « **Веб-ссылки**», чтобы отобразить клик WebJobs:



2. Нажмите « **Добавить»** . **Откроется** диалоговое окно **Add WebJob** .


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

Имя должно начинаться с буквы или числа и не может содержать никаких специальных символов, кроме «-» и «_».

4. В поле **Как запустить,** выбрать нужный вариант **Непрерывный** или **Срабатывает** (триггер может использовать график хрон или WebHook).
5. В поле « **Загрузка файла»** щелкните значок папки и перейдите к zip-файлу, содержащему ваш скрипт. Zip-файл должен содержать исполняемый файл (.exe .cmd .bat .sh .php .py .js), а также любые поддерживающие файлы, необходимые для запуска программы или скрипта.
6. Установите **флажок «Создать»,** чтобы загрузить сценарий в свое веб-приложение. Имя, указанное вами для WebJob, отображается в списке на клипе WebJobs.

Прочитайте Начало работы с azure-webjobs онлайн: https://riptutorial.com/ru/azure-webjobs/topic/1311/начало-работы-с-azure-webjobs

глава 2: Azure Webjobs SDK

Examples

JobHost

Azure Webjobs SDK - это **платформа,** распространяемая как пакет Nuget, которая помогает вам определять **функции** , запускаемые **триггерами,** и использовать **привязки** к другим службам Azure (например, Azure Storage и Service Bus) **декларативным** образом.

SDK использует **JobHost** для координации ваших кодированных функций. В типичном сценарии ваш Webjob представляет собой консольное приложение, которое инициализирует JobHost следующим образом:

```
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 позволяет персонализировать дополнительные настройки для разных триггеров:

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

Триггеры для очередей

Простой пример, определяющий функцию, которая запускается сообщением «Очередь»:

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

Он также поддерживает сериализацию POCO :

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

```
}
```

Триггеры для Blobs

Простой пример функции, которая запускается при изменении Azure Storage Blob:

```
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();

    }
}
```

Триггеры по времени

SDK поддерживает время срабатывания на основе выражений CRON с **6 полями** (`{second} {minute} {hour} {day} {month} {day of the week}`). Для **настройки** `JobHostConfiguration` требуется **дополнительная настройка** :

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

Функции, вызванные временем, реагируют на этот синтаксис:

```
// 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)
{

}
```

Триггеры по ошибкам

Обработка ошибок чрезвычайно важна, мы можем определить функции, которые будут запускаться, когда ошибка выполнения происходит в одной из ваших запущенных функций:

```
//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);
}
```

Это особенно полезно для централизации обработки ошибок.

ErrorTrigger требует **дополнительной настройки** в JobHostConfiguration:

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

Вы также должны установить пакет NuGet **Microsoft.Azure.WebJobs.Extensions** .

пересчет

Azure Webjobs работают в Azure App Service. Если мы масштабируем нашу службу приложений по горизонтали (добавим новые экземпляры), **каждый экземпляр** будет иметь **свой собственный JobHost** .

Обратите внимание, что это относится только к веб-приложениям, работающим в **непрерывном** режиме. Горизонтальное масштабирование не зависит от запросов по запросу и запланированных веб-приложений, они всегда запускают один экземпляр.

Если у вас есть непрерывные сообщения очереди обработки WebJob, и вы масштабируете план обслуживания приложений до 3 экземпляров, у вас будет 3 экземпляра WebJob.

Может быть, WebJobs, который вы хотите запустить в одном экземпляре, потому что вам может потребоваться обеспечить ровно один конвейер обработки. Для этих веб-приложений вы можете добавить атрибут **Singleton** .

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

Это достигается за счет использования Azure Blob Leases для распределенной блокировки.

Письменные журналы

Панель инструментов WebJobs показывает журналы в двух местах: странице для WebJob и странице для конкретного вызова WebJob.

Вывод методов консоли, которые вы вызываете в функции или в методе `Main()` отображается на странице Dashboard для WebJob, а не на странице для конкретного вызова метода. Вывод объекта TextWriter, который вы получаете из параметра в вашей

сигнатуре метода, отображается на странице Dashboard для вызова метода.

Чтобы записывать журналы трассировки приложений, используйте `Console.Out` (создает журналы, помеченные как INFO) и `Console.Error` (создает журналы, помеченные как 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);
}
```

Это приведет к появлению этих сообщений в панели инструментов для 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!
```

И это сообщение на странице Dashboard для метода:

```
TextWriter - Hello world!
```

Инъекция зависимостей с использованием Ninject

В следующем примере показано, как настроить инъекцию зависимостей с использованием Ninject в качестве контейнера IoC.

Сначала добавьте класс CustomModule в проект WebJob и добавьте туда привязки зависимостей.

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

Затем создайте класс JobActivator:

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

    public T CreateInstance<T>()
```

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

Когда вы устанавливаете JobHost в классе «Основная функция» программы, добавьте JobActivator в 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();
    }
}
```

Наконец, в классе Functions.cs введите свои службы.

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

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

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

Прочитайте Azure Webjobs SDK онлайн: https://riptutorial.com/ru/azure-webjobs/topic/2662/azure-webjobs-sdk

кредиты

S. No	Главы	Contributors
1	Начало работы с azure-webjobs	Community, gbellmann
2	Azure Webjobs SDK	gbellmann, juunas, lopezbertoni, Matias Quaranta