Building and Deploying a WebJob using the WebJobs SDK

The goal of this lab is to give you a step by step walk through of creating and deploying a WebJob using C# and Visual Studio 2015.

In this hands-on lab, you will explore the basics of WebJobs SDK and the WebJobs SDK Extensions by creating a simple console application that mimics the general steps of an order processing system.

This lab starts with building a simple WebJob, and then make several improvements using some of the customization points in the WebJobs SDK to accomplish things that a real production WebJob will need to do.

Prerequisites

- Visual Studio 2015 Installed
- Azure Subscription
- Send Grid API Key
- Internet connection

Assumptions

- You have experience developing C# applications

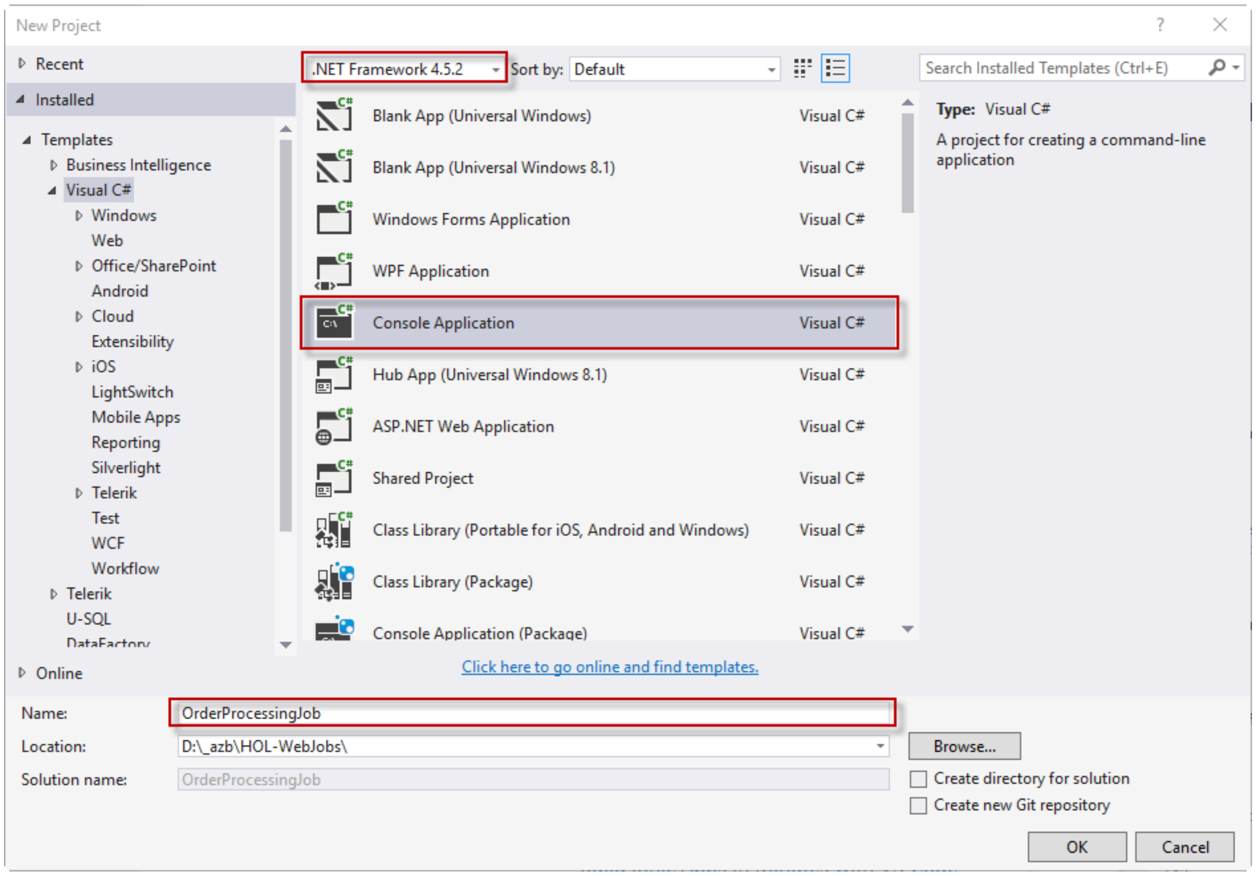
This lab includes the following sections:

1. Create a Console Application project for the WebJob functionality
2. Setup a storage account to use with the WebJob
3. Add a function to the WebJob to watch a queue for Orders and then save to Blob storage
4. Add a test function to verify the queue and blob logic works
5. Modify the logic to use a custom NameResolver
6. Modify the logic to use a custom trace writer
7. Add functionality to save the Products to a storage table
8. Add functionality to send an email once the order is processed
9. Add a function to handle poison messages on the Orders queue to send out an email
10. Add a function that emails when a certain number of error have happened
11. Deploy the WebJob to Azure

Create a Console Application project for the WebJob functionality

In this section we will create the console application that we will be adding functionality to throughout this hands-on lab.

1. Open **Visual Studio 2015** and click the **New Project** link on the start page (or File -> New -> Project.
2. Create a new **Console Application** using **C#, .NET Framework 4.5.2** and name it **OrderProcessingJob**.


New Project

Installed > Templates > Visual C#

.NET Framework 4.5.2 | Sort by: Default

Console Application — Visual C#

Type: Visual C#

A project for creating a command-line application

Name: OrderProcessingJob

Location: D:_azb\HOL-WebJobs\

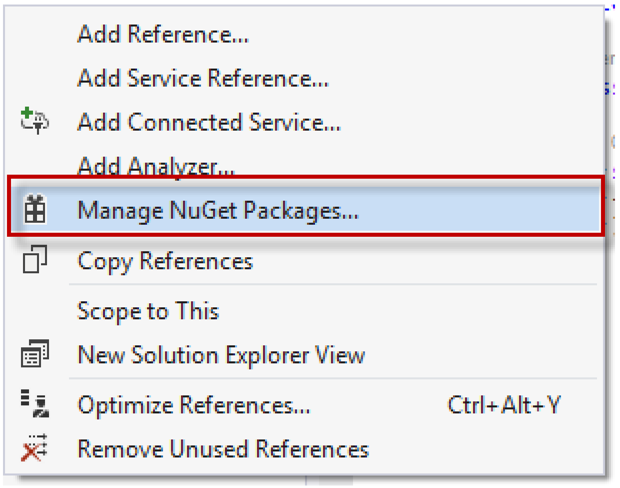
Solution name: OrderProcessingJob

Recent
Business Intelligence
Windows
Web
Office/SharePoint
Android
Cloud
Extensibility
iOS
LightSwitch
Mobile Apps
Reporting
Silverlight
Telerik
Test
WCF
Workflow
Telerik
U-SQL
DataFactory
Online
Blank App (Universal Windows)
Blank App (Universal Windows 8.1)
Windows Forms Application
WPF Application
Hub App (Universal Windows 8.1)
ASP.NET Web Application
Shared Project
Class Library (Portable for iOS, Android and Windows)
Class Library (Package)
Console Application (Package)
Search Installed Templates (Ctrl+E)
Click here to go online and find templates.
Browse...
Create directory for solution
Create new Git repository
OK
Cancel

Before starting the WebJobs logic, we need to add a NuGet package (which will add all the other dependencies we need along with it):

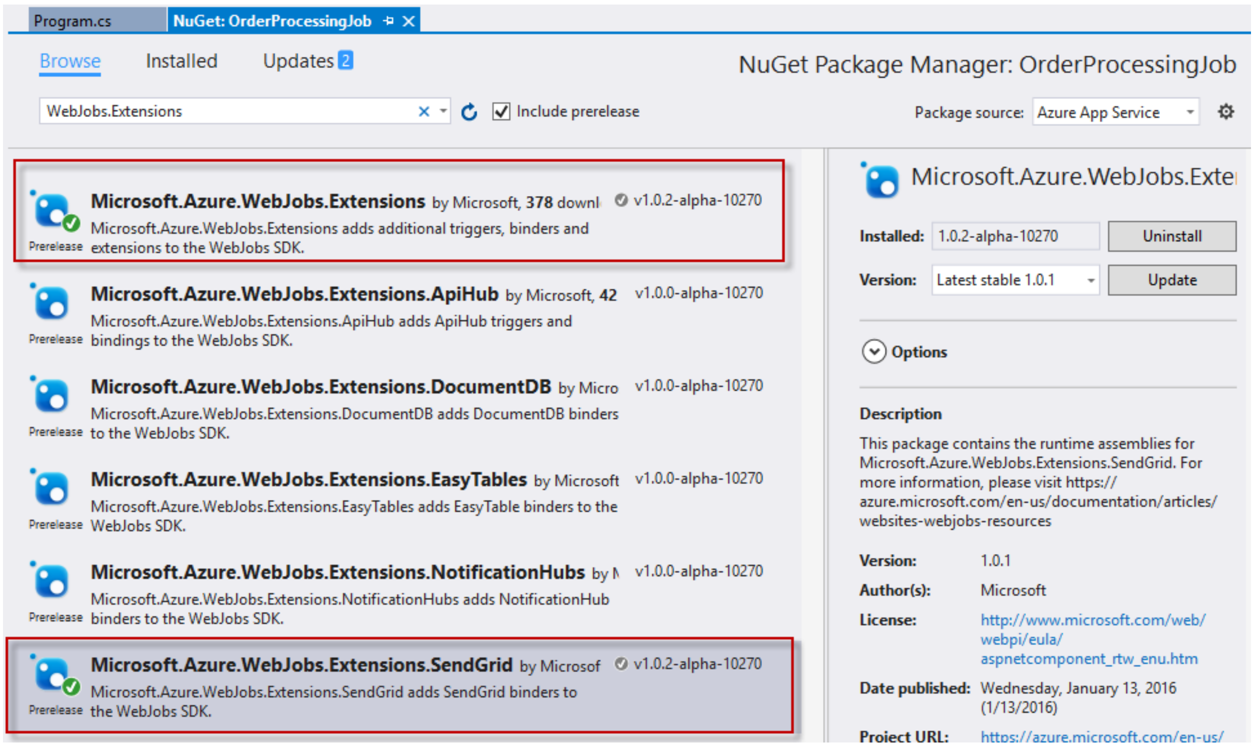
- Microsoft.Azure.WebJobs.Extensions.SendGrid

3. In the **Solution Explorer**, **right-click the References** node and select **Manage NuGet Packages** ...



4. In the **NuGet window**, select the **Browse** tab, check the **Include prerelease** checkbox and type **WebJobs.Extensions** in the search box.

5. Select **Microsoft.Azure.WebJobs.Extensions.SendGrid** and click **Install** button. This will ask you to confirm some license boxes


Program.cs
NuGet: OrderProcessingJob
Browse
Installed
Updates 2
NuGet Package Manager: OrderProcessingJob
WebJobs.Extensions
Include prerelease
Package source: Azure App Service
Uninstall
Update
Options

Microsoft.Azure.WebJobs.Extensions by Microsoft, 378 downl v1.0.2-alpha-10270
Microsoft.Azure.WebJobs.Extensions adds additional triggers, binders and extensions to the WebJobs SDK.
Prerelease

Microsoft.Azure.WebJobs.Extensions.ApiHub by Microsoft, 42 v1.0.0-alpha-10270
Microsoft.Azure.WebJobs.Extensions.ApiHub adds ApiHub triggers and bindings to the WebJobs SDK.
Prerelease

Microsoft.Azure.WebJobs.Extensions.DocumentDB by Micro v1.0.0-alpha-10270
Microsoft.Azure.WebJobs.Extensions.DocumentDB adds DocumentDB binders to the WebJobs SDK.
Prerelease

Microsoft.Azure.WebJobs.Extensions.EasyTables by Microsoft v1.0.0-alpha-10270
Microsoft.Azure.WebJobs.Extensions.EasyTables adds EasyTable binders to the WebJobs SDK.
Prerelease

Microsoft.Azure.WebJobs.Extensions.NotificationHubs by M v1.0.0-alpha-10270
Microsoft.Azure.WebJobs.Extensions.NotificationHubs adds NotificationHub binders to the WebJobs SDK.
Prerelease

Microsoft.Azure.WebJobs.Extensions.SendGrid by Microsof v1.0.2-alpha-10270
Microsoft.Azure.WebJobs.Extensions.SendGrid adds SendGrid binders to the WebJobs SDK.
Prerelease

Microsoft.Azure.WebJobs.Exte

Installed: 1.0.2-alpha-10270
Version: Latest stable 1.0.1

Description
This package contains the runtime assemblies for Microsoft.Azure.WebJobs.Extensions.SendGrid. For more information, please visit https://azure.microsoft.com/en-us/documentation/articles/websites-webjobs-resources

Version:	1.0.1
Author(s):	Microsoft
License:	http://www.microsoft.com/web/webpi/eula/aspnetcomponent_rtw_enu.htm
Date published:	Wednesday, January 13, 2016 (1/13/2016)
Project URL:	https://azure.microsoft.com/en-us/

You should now see the green check by the NuGet packages that were installed.

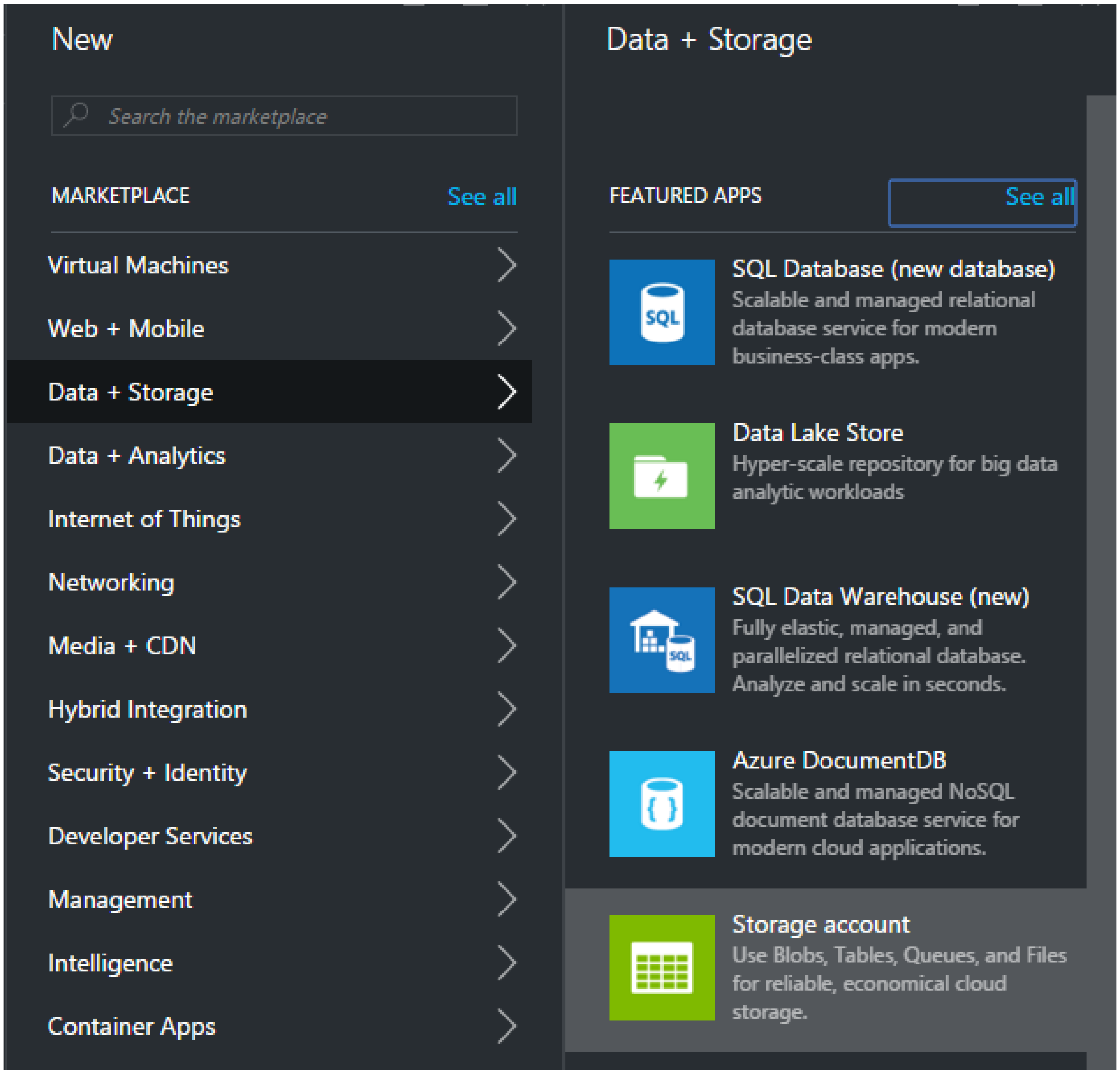
6. In the Solution Explorer, right-click on the **OrderProcessingJob project** and select **Add -> Class**. Name it **Order.cs** and click **Add**.
7. Open the **Order** class and insert the following C#

```
using System;
using System.Collections.Generic;

namespace OrderProcessingJob
{
    public class Order
    {
        public string Id { get; set; }
        public string BuyerName { get; set; }
        public string BuyerEmail { get; set; }
        public string BuyerCell { get; set; }
        public DateTime OrderPlacedUtc { get; set; }
        public List<Product> Products { get; set; }
    }

    public class Product
    {
        public string Id { get; set; }
        public string Name { get; set; }
        public int Quantity { get; set; }
        public double Price { get; set; }
    }
}
```

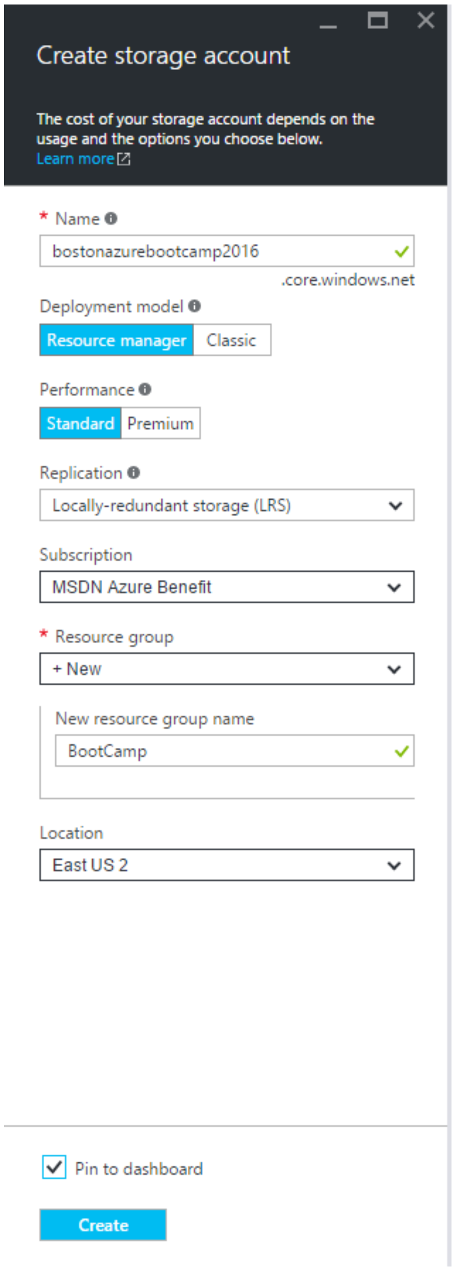
Setup a storage account to use with the WebJob

Since the WebJob is going to use queues, tables and blobs, we need to create a storage account to use.

1. Open a browser and go to https://portal.azure.com and sign in with your Azure credentials.
2. On the top left, click the **+ New button**


New
Search the marketplace
MARKETPLACE
See all
Virtual Machines
Web + Mobile
Data + Storage
Data + Analytics
Internet of Things
Networking
Media + CDN
Hybrid Integration
Security + Identity
Developer Services
Management
Intelligence
Container Apps
Data + Storage
FEATURED APPS
See all
SQL Database (new database)
Scalable and managed relational database service for modern business-class apps.
Data Lake Store
Hyper-scale repository for big data analytic workloads
SQL Data Warehouse (new)
Fully elastic, managed, and parallelized relational database. Analyze and scale in seconds.
Azure DocumentDB
Scalable and managed NoSQL document database service for modern cloud applications.
Storage account
Use Blobs, Tables, Queues, and Files for reliable, economical cloud storage.

3. On the Create blade, select **Data + Storage -> Storage account**



Create storage account

The cost of your storage account depends on the usage and the options you choose below.
Learn more

* Name: bostonazurebootcamp2016 .core.windows.net

Deployment model: [✓ Resource manager] Classic

Performance: [✓ Standard] Premium

Replication: Locally-redundant storage (LRS)

Subscription: MSDN Azure Benefit

* Resource group: + New

New resource group name: BootCamp

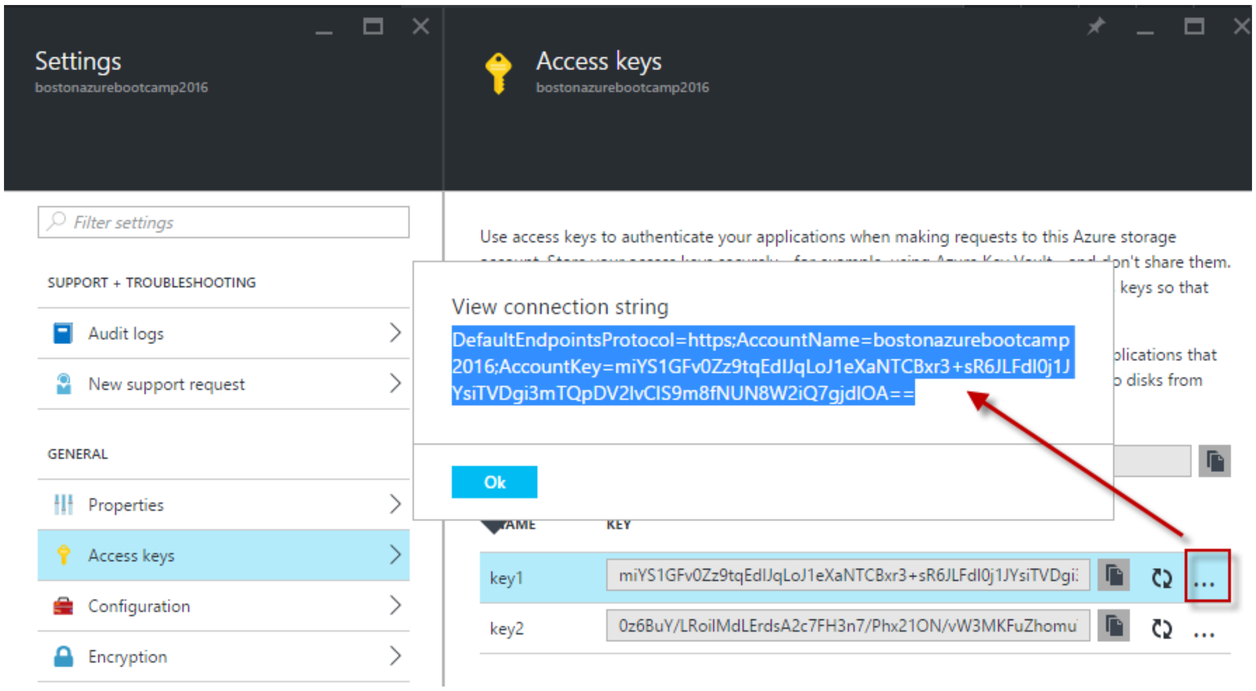
Location: East US 2

[✓] Pin to dashboard

4. On the **Create storage account** blade, enter a **unique name** for the storage account
5. Give the storage account a **resource name**
6. Select **Pin to dashboard** (this will save you a few steps later)
7. Click the **Create** button

Once the storage account creates, the panel should open for you. If not, you can click the storage account tile on the dashboard.

8. In the storage account panel -> **All Settings** (blade should be open). Select **Access Keys**



Use access keys to authenticate your applications when making requests to this Azure storage ... don't share them. ... keys so that ... plications that ... disks from

View connection string

DefaultEndpointsProtocol=https;AccountName=bostonazurebootcamp2016;AccountKey=miYS1GFv0Zz9tqEdIJqLoJ1eXaNTCBxr3+sR6JLFdl0j1JYsiTVDgi3mTQpDV2lvCIS9m8fNUN8W2iQ7gjdlOA==

key1 miYS1GFv0Zz9tqEdIJqLoJ1eXaNTCBxr3+sR6JLFdl0j1JYsiTVDgi:

key2 0z6BuY/LRoilMdLErdsA2c7FH3n7/Phx21ON/vW3MKFuZhomu

9. Click on the … beside the key1 and **copy the connection string** from the dialog to the **clipboard** (Ctrl + C)
10. Back in Visual Studio, open the **app.config** file
11. Add two connection strings named AzureWebJobsDashboard and AzureWebJobsStorage and set their connectionString to the value you copied from the portal.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="AzureWebJobsDashboard" connectionString="DefaultEndpointsProtocol=https;AccountName=bostonazurebootcamp2016;AccountKey=miYS1GFv0Zz9tqEdIJqLoJ1eXaNTCBxr3+sR6JLFdl0j1JYsiTVDgi3mTQpDV2lvCIS9m8fNUN8W2iQ7gjdlOA==" />
    <add name="AzureWebJobsStorage" connectionString="DefaultEndpointsProtocol=https;AccountName=bostonazurebootcamp2016;AccountKey=miYS1GFv0Zz9tqEdIJqLoJ1eXaNTCBxr3+sR6JLFdl0j1JYsiTVDgi3mTQpDV2lvCIS9m8fNUN8W2iQ7gjdlOA==" />
  </connectionStrings>
```

Make sure you use your storage account - the key I show above will be changed by the time you read this ☺

Add a function to the WebJob to watch a queue for Orders and then save to Blob storage

You are now ready to create the WebJob functionality to use a QueueTrigger and a Blob binding. Since we want the blob to work with the Order type (not a Stream type or one of the other options) we will need to also create an implementation of the ICloudBlobStreamBinder to create the custom serialization/deserialization of an Order type.

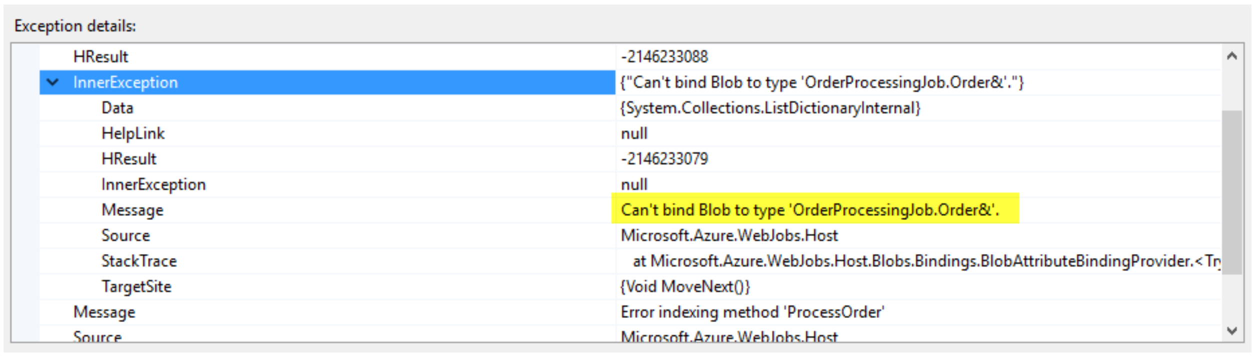
1. Open the **Program.cs** file and insert the following C#

```
using Microsoft.Azure.WebJobs;

namespace OrderProcessingJob
{
    class Program
    {
        static void Main(string[] args)
        {
            var config = new JobHostConfiguration();

            config.UseDevelopmentSettings();

            var host = new JobHost(config);
            host.RunAndBlock();
        }
    }
}
```

2. In the Solution Explorer, right-click on the **OrderProcessingJob project** and select **Add -> Class**. Name it **Functions.cs** and click **Add**.
3. Open the **Functions** class and insert the following C#

```
using System.IO;
using Microsoft.Azure.WebJobs;

namespace OrderProcessingJob
{
    public class Functions
    {
        public static void ProcessOrder(
           [QueueTrigger("orders")] Order order,
           [Blob("orders/{Id}")] out Order orderToSave,
           TextWriter log
           )
        {
            // Save the order in blob storage
            orderToSave = order;

            // log order processed
            log.WriteLine($"Processed {order.Products.Count} products");
        }
    }
}
```

If you try and run the console application now, you will get an exception:



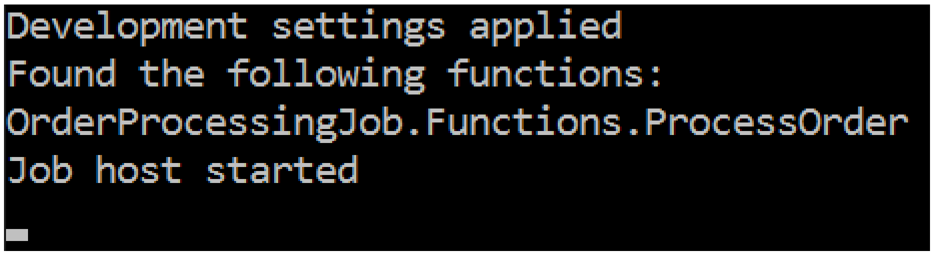
Exception details:	
HResult	-2146233088
InnerException	{"Can't bind Blob to type 'OrderProcessingJob.Order&'."}
Data	{System.Collections.ListDictionaryInternal}
HelpLink	null
HResult	-2146233079
InnerException	null
Message	Can't bind Blob to type 'OrderProcessingJob.Order&'.
Source	Microsoft.Azure.WebJobs.Host
StackTrace	at Microsoft.Azure.WebJobs.Host.Blobs.Bindings.BlobAttributeBindingProvider.<Tr
TargetSite	{Void MoveNext()}
Message	Error indexing method 'ProcessOrder'
Source	Microsoft.Azure.WebJobs.Host

This is due to the Blob binding not knowing how we want the Order type serialized to the blob storage.

4. In the Solution Explorer, right-click on the **OrderProcessingJob project** and select **Add -> Class**. Name it **OrderBlobBinder.cs** and click **Add**.
5. Open the **OrderBlobBinder**class and insert the following C#

```
using System.IO;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Azure.WebJobs;
using Newtonsoft.Json;

namespace OrderProcessingJob
{
    public class OrderBlobBinder : ICloudBlobStreamBinder<Order>
    {
        public Task<Order> ReadFromStreamAsync(Stream input,
            CancellationToken cancellationToken)
        {
            using (StreamReader reader = new StreamReader(input))
            {
                var jsonString = reader.ReadToEnd();
                var order = JsonConvert.DeserializeObject<Order>(jsonString);
                return Task.FromResult(order);
            }
        }

        public Task WriteToStreamAsync(Order value, Stream output,
            CancellationToken cancellationToken)
        {
            var serializer = new JsonSerializer();
            using (var writer = new StreamWriter(output))
            using (var jsonTextWriter = new JsonTextWriter(writer))
            {
                serializer.Serialize(jsonTextWriter, value);
            }
            return Task.FromResult<object>(null);
        }
    }
}
```

Now if you run the console application you will see the job starts up fine.



```
Development settings applied
Found the following functions:
OrderProcessingJob.Functions.ProcessOrder
Job host started
```

Add a test function to verify the queue and blob logic works

We now need to create a function that will add an Order object to the orders queue so we can verify the WebJob functionality is working as designed so far. This new test function will use the NoAutomaticTrigger attribute to allow us to use the WebJobs SDK bindings without needing a trigger.

1. Open the **Functions** class and insert the following C#

```csharp
[NoAutomaticTrigger]
public static void PutTestOrderInQueue(
    [Queue("orders")] out Order testOrder,
    TextWriter log)
{
    testOrder = new Order()
    {
        Id = Guid.NewGuid().ToString("N"),
        BuyerEmail = "<your email here>",
        BuyerName = "<your name here>",
        OrderPlacedUtc = DateTime.UtcNow,
        Products = new List<Product>
        {
            new Product()
            {
                Id = "Product1",
                Name = "Product #1",
                Price = 1.5,
                Quantity = 5
            },
            new Product()
            {
                Id = "Product2",
                Name = "Product #2",
                Price = 2.1,
                Quantity = 2
            }
        }
    };

    // log message
    log.WriteLine("Queued test order");
}
```

2. You may need to add the following using statement:

```
using System;

```

Now we need to add a call to invoke this function when the WebJob starts up

3. Open the **Program.cs** file and add the highlighted line to your code:

```
static void Main(string[] args)
{
    var config = new JobHostConfiguration();

    config.UseDevelopmentSettings();

    var host = new JobHost(config);

    host.Call(typeof(Functions).GetMethod("PutTestOrderInQueue"));

    host.RunAndBlock();
}
```

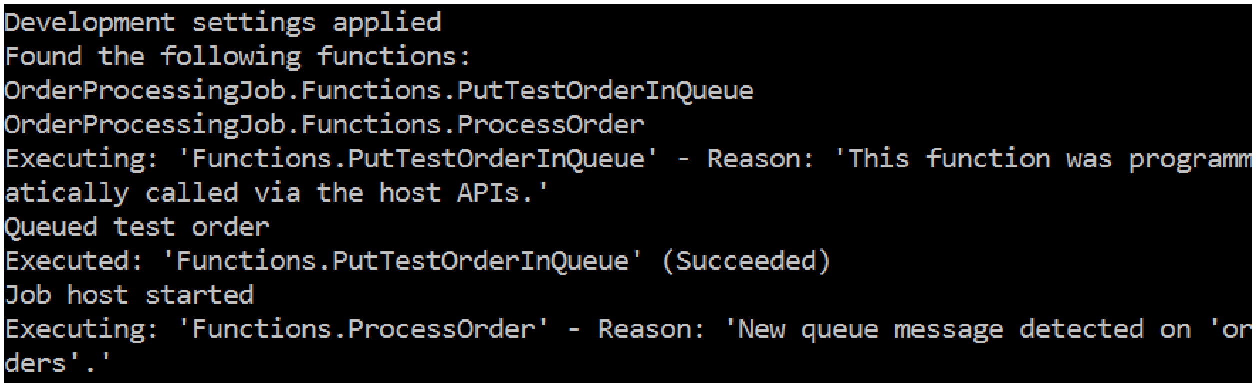
One more thing to do before testing it out:

4. Open the **Functions** class and put a breakpoint in the first line off the ProcessOrder method.

```
public static void ProcessOrder(
    [QueueTrigger("orders")] Order order,
    [Blob("orders/{Id}")] out Order orderToSave,
    TextWriter log
    )
{
    // Save the order in blob storage
    orderToSave = order;

    // log order processed
    log.WriteLine($"Processed {order.Products.Count} products");
}
```

5. Run the application

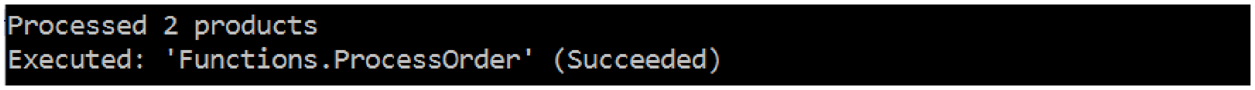
Once the WebJob starts, you should see the logging in your console that looks like this:


```
Development settings applied
Found the following functions:
OrderProcessingJob.Functions.PutTestOrderInQueue
OrderProcessingJob.Functions.ProcessOrder
Executing: 'Functions.PutTestOrderInQueue' - Reason: 'This function was programm
atically called via the host APIs.'
Queued test order
Executed: 'Functions.PutTestOrderInQueue' (Succeeded)
Job host started
Executing: 'Functions.ProcessOrder' - Reason: 'New queue message detected on 'or
ders'.'
```

The debugger should now be stopped in the ProcessOrder method.

6. Hit **F5** to **continue execution** of the method

Once the method has completed the console should add the following lines:


```
Processed 2 products
Executed: 'Functions.ProcessOrder' (Succeeded)
```

At this point the logic has completed, so you can stop the debugger.

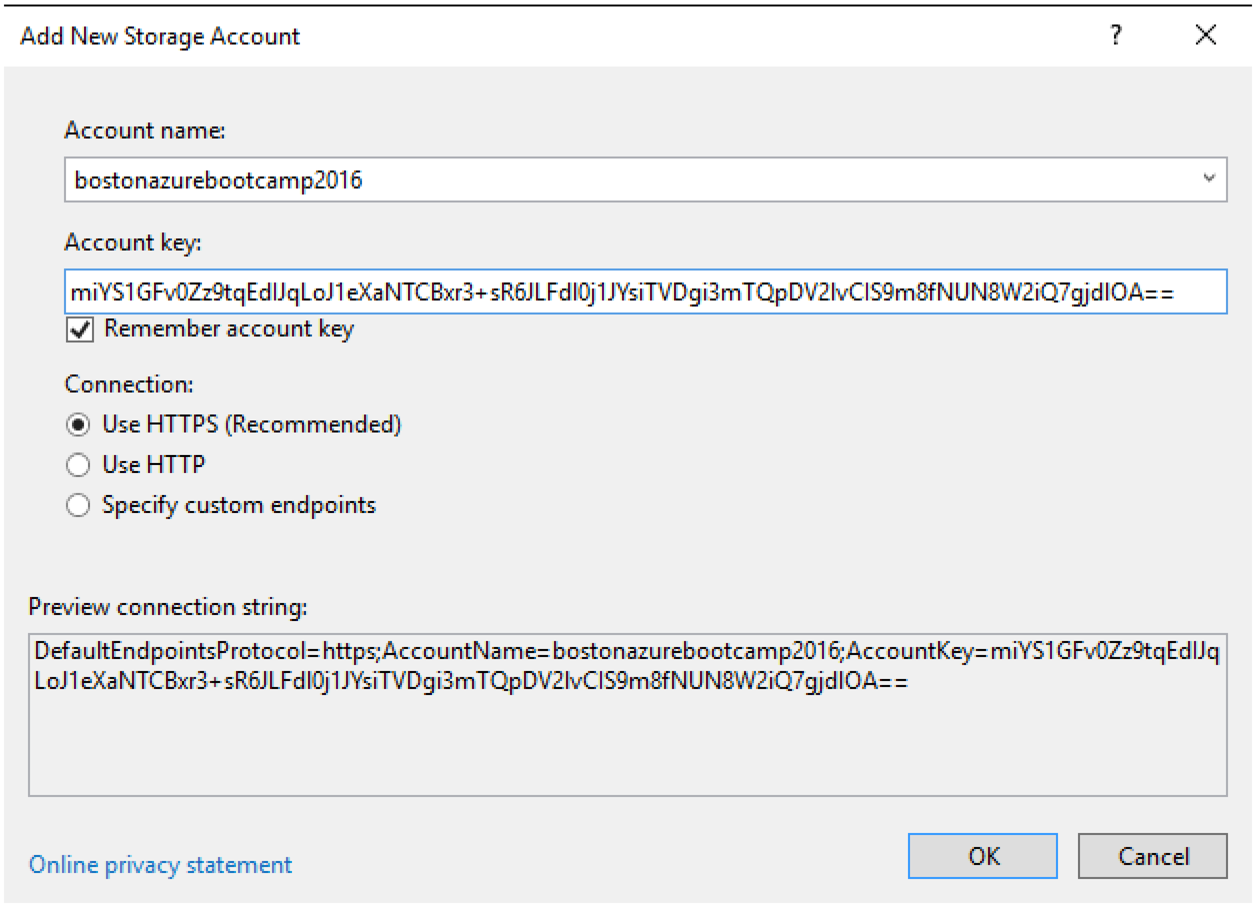
Verify the blob was saved to storage

7. Open **Server Explorer** and click on the **Azure** node. Login to your account if you need to
8. Locate the **Storage** node and look for your **WebJob's storage account** you setup earlier

If you don't see the storage account under the storage node:

9. Right-click the **Storage node** -> **Attach External Storage**

This will show the Add New Storage Account dialog


Add New Storage Account

Account name: bostonazurebootcamp2016

Account key: miYS1GFv0Zz9tqEdlJqLoJ1eXaNTCBxr3+sR6JLFdl0j1JYsiTVDgi3mTQpDV2lvClS9m8fNUN8W2iQ7gjdlOA==

[✓] Remember account key

Connection:

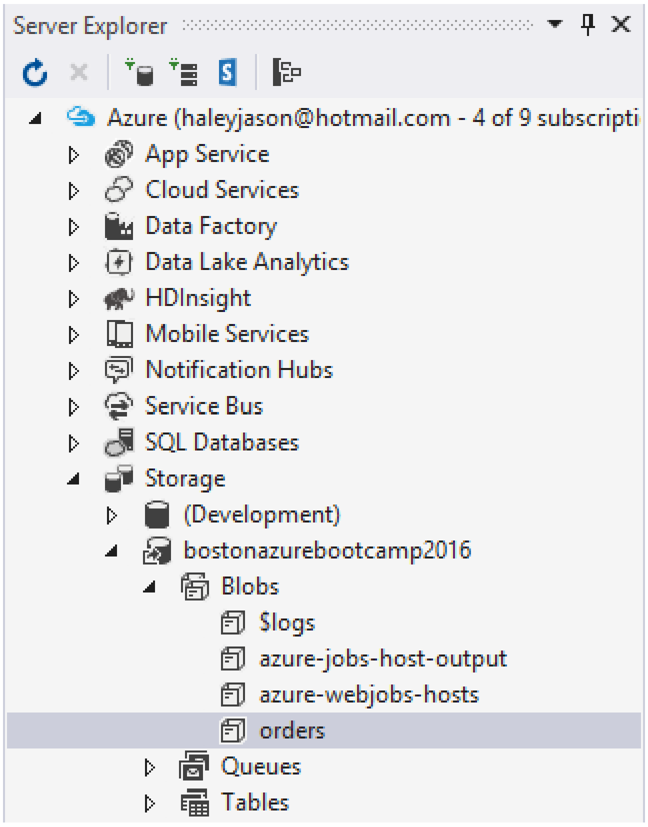
- [✓ Use HTTPS (Recommended)]
- Use HTTP
- Specify custom endpoints

Preview connection string:

```
DefaultEndpointsProtocol=https;AccountName=bostonazurebootcamp2016;AccountKey=miYS1GFv0Zz9tqEdlJq
LoJ1eXaNTCBxr3+sR6JLFdl0j1JYsiTVDgi3mTQpDV2lvClS9m8fNUN8W2iQ7gjdlOA==
```

10. Enter your **storage account name** and **account key** (these are in the connection string you added to the app.config earlier).
11. Click Ok

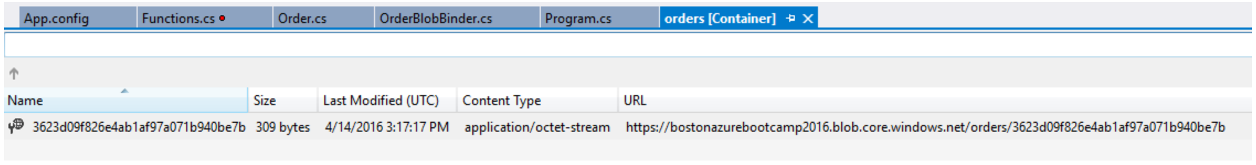
Your storage account should now show.


Server Explorer

- Azure (haleyjason@hotmail.com - 4 of 9 subscripti
  - App Service
  - Cloud Services
  - Data Factory
  - Data Lake Analytics
  - HDInsight
  - Mobile Services
  - Notification Hubs
  - Service Bus
  - SQL Databases
  - Storage
    - (Development)
    - bostonazurebootcamp2016
      - Blobs
        - $logs
        - azure-jobs-host-output
        - azure-webjobs-hosts
        - orders
      - Queues
      - Tables

12. Click on the **orders** blob container to see the contents.

You should see one item in the container that was uploaded recently.



Name	Size	Last Modified (UTC)	Content Type	URL
3623d09f826e4ab1af97a071b940be7b	309 bytes	4/14/2016 3:17:17 PM	application/octet-stream	https://bostonazurebootcamp2016.blob.core.windows.net/orders/3623d09f826e4ab1af97a071b940be7b

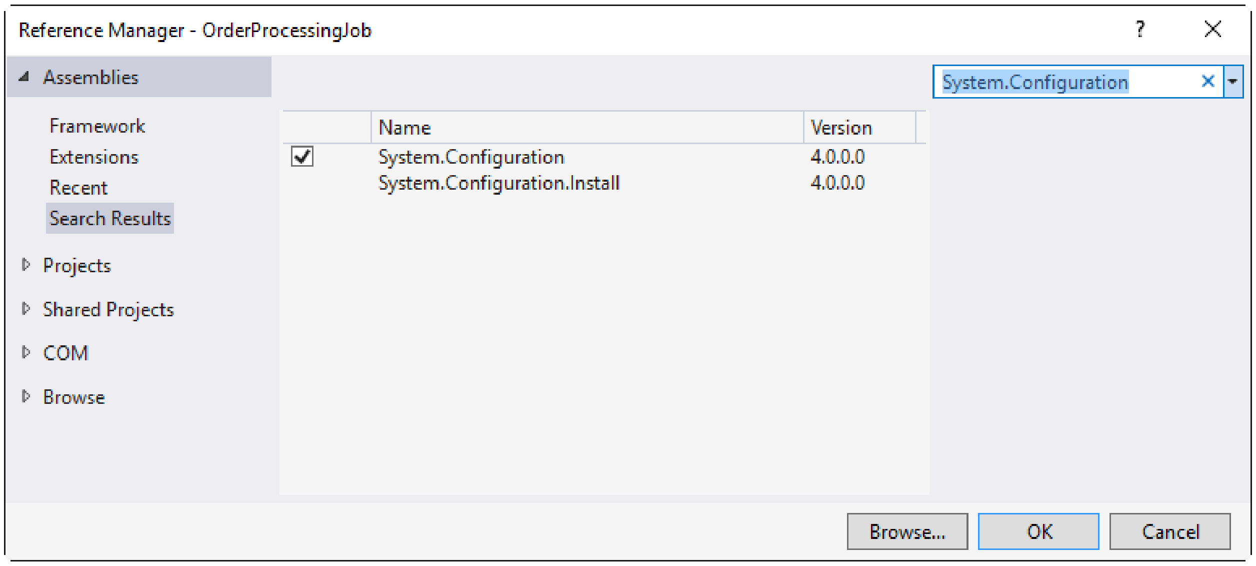
Modify the logic to use a custom NameResolver

One thing that you may notice in the ProcessOrder function, is the QueueTrigger and Blob attributes have hard coded queue and container names. This can be limiting when you have multiple environments and may want those queue and/or blob containers to differ between the environments. The solution to this is to create a custom implementation of the INameResolver interface.

1. In the Solution Explorer, right-click on the **OrderProcessingJob project** and select **Add -> Class**. Name it **NameResolver.cs** and click **Add**.
2. Open the **NameResolver** class and insert the following C#

```
using System;
using System.Configuration;
using Microsoft.Azure.WebJobs;

namespace OrderProcessingJob
{
    public class NameResolver : INameResolver
    {
        public string Resolve(string name)
        {
            var settingValue = ConfigurationManager.AppSettings[name];
            if (string.IsNullOrEmpty(settingValue))
            {
                settingValue = Environment.GetEnvironmentVariable(name);
            }
            return settingValue;
        }
    }
}
```

3. In the Solutions Explorer, right-click on References -> Add Reference
4. Select Assemblies, enter System.Configuration in the search box and check the box to add the reference.



	Name	Version
✓	System.Configuration	4.0.0.0
	System.Configuration.Install	4.0.0.0

5. Click OK

Now that you have a custom name resolver, you need to configure it, add application settings and modify the code to use the name resolver.

6. Open the Program.cs file and add the highlighted line of C#

```
static void Main(string[] args)
{
    var config = new JobHostConfiguration();

    config.NameResolver = new NameResolver();

    config.UseDevelopmentSettings();

    var host = new JobHost(config);

    host.Call(typeof(Functions).GetMethod("PutTestOrderInQueue"));
    host.RunAndBlock();
}
```

7. Open the Functions class and modify the attributes to use the special syntax that indicates what name to resolve as highlighted below:

```
public static void ProcessOrder(
    [QueueTrigger("%OrdersQueueName%")] Order order,
    [Blob("%OrdersContainerName%/{Id}")] out Order orderToSave,
    TextWriter log
    )
{
    // Save the order in blob storage
    orderToSave = order;

    // log order processed
    log.WriteLine($"Processed {order.Products.Count} products");
}
```

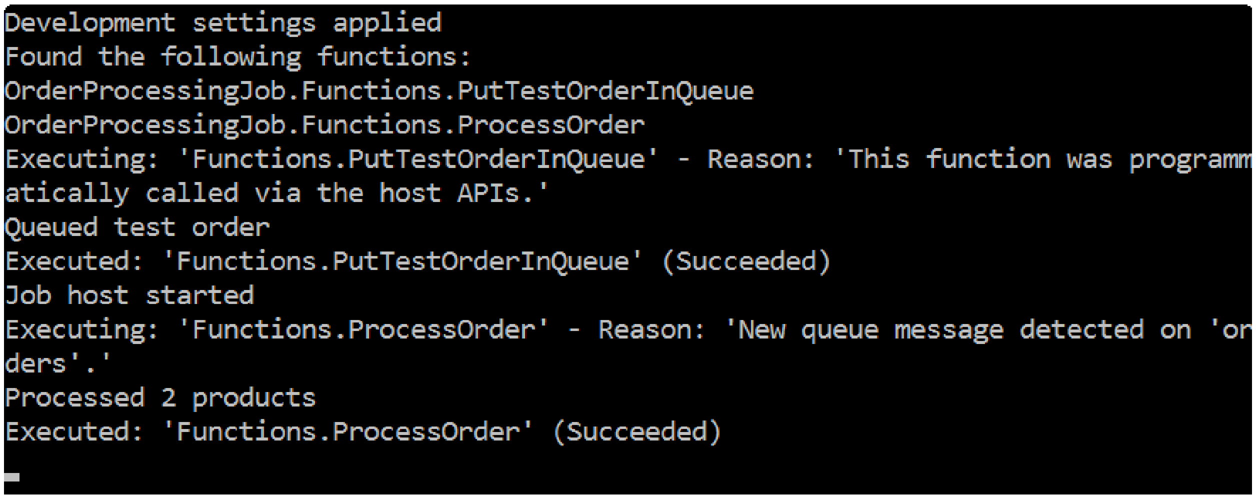
8. Open App.config and add an appSettings section and two keys named:

- OrdersQueueName
- OrdersContainerName

```
<configuration>
  <connectionStrings>
    <add name="AzureWebJobsDashboard" connectionString="…" />
    <add name="AzureWebJobsStorage" connectionString="…" />
  </connectionStrings>
  <appSettings>
    <add key="OrdersQueueName" value="orders" />
    <add key="OrdersContainerName" value="orders" />
  </appSettings>
```

9. Run the application

You should get the same console output as before – but this time the code is getting the queue and container names from the configuration file.



```
Development settings applied
Found the following functions:
OrderProcessingJob.Functions.PutTestOrderInQueue
OrderProcessingJob.Functions.ProcessOrder
Executing: 'Functions.PutTestOrderInQueue' - Reason: 'This function was programm
atically called via the host APIs.'
Queued test order
Executed: 'Functions.PutTestOrderInQueue' (Succeeded)
Job host started
Executing: 'Functions.ProcessOrder' - Reason: 'New queue message detected on 'or
ders'.'
Processed 2 products
Executed: 'Functions.ProcessOrder' (Succeeded)
```

Modify the logic to use a custom trace writer

Sometimes you may find you want to customize your logging. For instance, you may want to log certain things to a database or log4net or some other location besides the WebJobs dashboard.

Logging can be customized by creating and configuring your own implementation of a TraceWriter.

For this lab, we will just create a trace writer that writes to the console in a different color to differentiate the messages coming from our logger vs. the WebJobs logging.

1. In the Solution Explorer, right-click on the **OrderProcessingJob project** and select **Add -> Class**. Name it **ColorConsoleTraceWriter.cs** and click **Add**.
2. Open the **ColorConsoleTraceWriter** class and insert the following C#

```
using System;
using System.Diagnostics;
using Microsoft.Azure.WebJobs.Host;

namespace OrderProcessingJob
{
    public class ColorConsoleTraceWriter : TraceWriter
    {
        public ColorConsoleTraceWriter(TraceLevel level)
            : base(level)
        {
        }

        public override void Trace(TraceEvent traceEvent)
        {
            var holdColor = Console.ForegroundColor;

            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.WriteLine(traceEvent.Message);

            Console.ForegroundColor = holdColor;
        }
    }
}
```

3. Open the Program.cs file and add in the highlighted line of C#

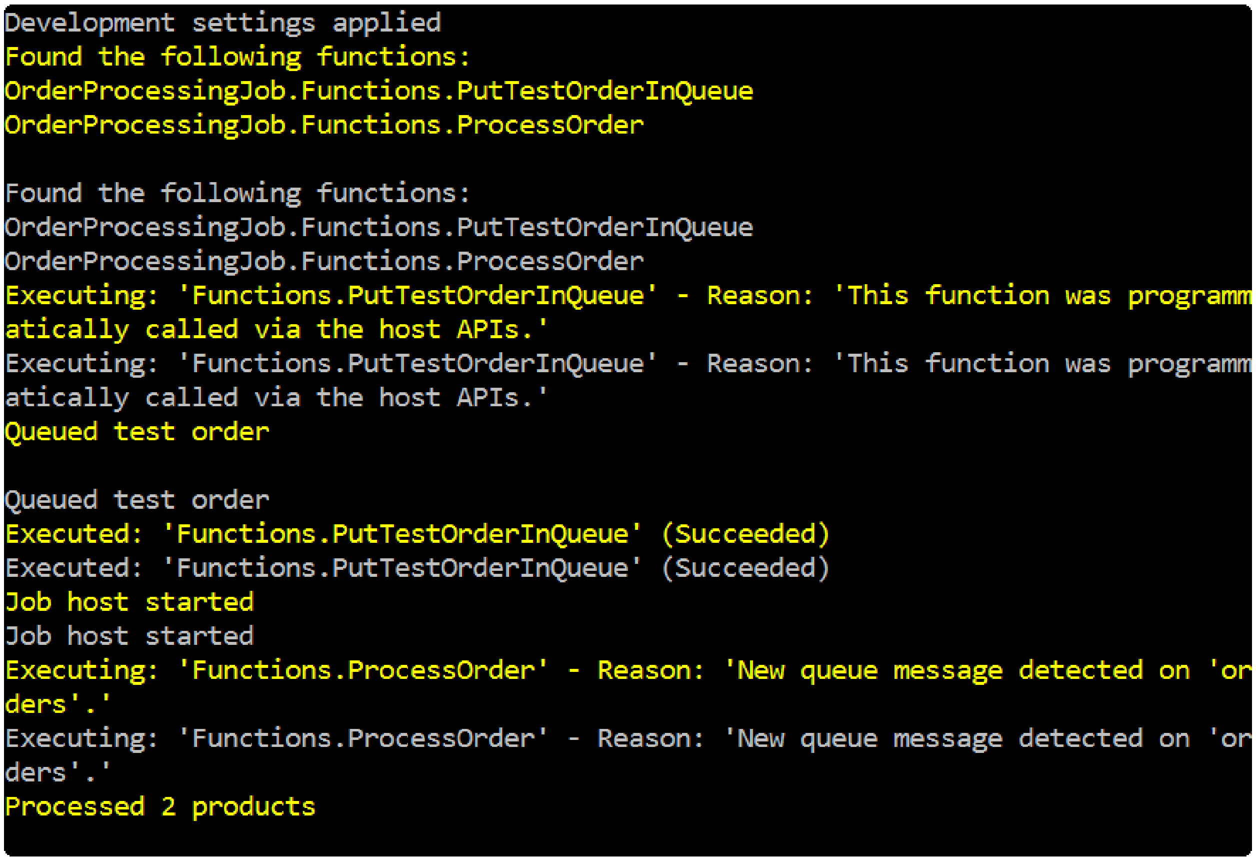
```
static void Main(string[] args)
{
    var config = new JobHostConfiguration();

    config.NameResolver = new NameResolver();
    config.Tracing.Tracers.Add(new ColorConsoleTraceWriter(TraceLevel.Info));

    config.UseDevelopmentSettings();

    var host = new JobHost(config);
    host.Call(typeof(Functions).GetMethod("PutTestOrderInQueue"));
    host.RunAndBlock();
}
```

4. You may need to add the following using statement:

```
using System.Diagnostics;
```

5. Run the application

You should now see duplicated messages in the console in yellow – those are the ones being logged by the custom trace writer … which means in real life you could be putting those into the database or some other location instead of the console window.



```
Development settings applied
Found the following functions:
OrderProcessingJob.Functions.PutTestOrderInQueue
OrderProcessingJob.Functions.ProcessOrder

Found the following functions:
OrderProcessingJob.Functions.PutTestOrderInQueue
OrderProcessingJob.Functions.ProcessOrder
Executing: 'Functions.PutTestOrderInQueue' - Reason: 'This function was programm
atically called via the host APIs.'
Executing: 'Functions.PutTestOrderInQueue' - Reason: 'This function was programm
atically called via the host APIs.'
Queued test order

Queued test order
Executed: 'Functions.PutTestOrderInQueue' (Succeeded)
Executed: 'Functions.PutTestOrderInQueue' (Succeeded)
Job host started
Job host started
Executing: 'Functions.ProcessOrder' - Reason: 'New queue message detected on 'or
ders'.'
Executing: 'Functions.ProcessOrder' - Reason: 'New queue message detected on 'or
ders'.'
Processed 2 products
```

Add functionality to Save the Order Products to a Storage Table

Sometimes you want to save things in an Azure storage table. When you use the WebJobs SDK, it is just as easy as saving the Order to blob storage. In this exercise we'll add a Table binding to the ProcessOrder function to save the individual products on the Order to a table.

1. In the Solution Explorer, right-click on the **OrderProcessingJob project** and select **Add -> Class**. Name it **ProductsOrderedEntity.cs** and click **Add**.
2. Open the **ProductsOrderedEntity** class and insert the following C#

```
using System;
using Microsoft.WindowsAzure.Storage.Table;

namespace OrderProcessingJob
```

```
{
    public class ProductsOrderedEntity : TableEntity
    {
        public ProductsOrderedEntity()
        { }

        public ProductsOrderedEntity(string orderId,
            Product product,
            DateTime orderedUtc)
        {
            SetRowKey(orderId, product, orderedUtc);

            OrderId = orderId;
            Id = product.Id;
            Name = product.Name;
            Quantity = product.Quantity;
            Price = product.Price;
        }

        public string OrderId { get; set; }
        public string Id { get; set; }
        public string Name { get; set; }
        public int Quantity { get; set; }
        public double Price { get; set; }
        public void SetRowKey(string orderId,
            Product product,
            DateTime orderedUtc)
        {
            // Use a reverse data schema to keep rows sorted chronologically
            long ticks = DateTimeOffset.MaxValue.Ticks - orderedUtc.Ticks;

            RowKey = $"{ticks}_{orderId}_{product.Id}";
            PartitionKey = orderId;
        }
    }
}
```

Since we created a customer name resolver, we should add a new application setting for the name of the storage table.

3. Open app.config and add the highlighted app setting:

```
  <appSettings>
    <add key="OrdersQueueName" value="orders" />
    <add key="OrdersContainerName" value="orders" />
    <add key="ProductsTableName" value="OrderedProducts"/>
  </appSettings>
```

4. Open the Functions class and locate the ProcessOrder method

With the table, we want to be able to add multiple rows inside the ProcessOrder method. We do this by using a parameter that is an ICollector.

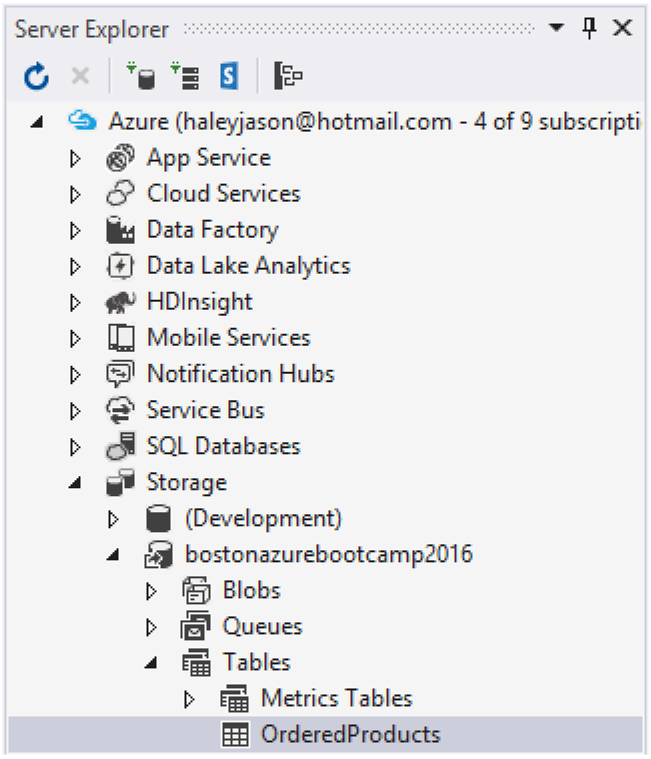
5. Add the highlighted changes to the ProcessOrder method:

```
public static void ProcessOrder(
    [QueueTrigger("%OrdersQueueName%")] Order order,
    [Blob("%OrdersContainerName%/{Id}")] out Order orderToSave,
    [Table("%ProductsTableName%")] ICollector<ProductsOrderedEntity> productsOrderedTable,
    TextWriter log
    )
{
    // Save the order in blob storage
    orderToSave = order;

    // Save the ordered product quantities to table storage
    foreach (var product in order.Products)
    {
        productsOrderedTable.Add(
            new ProductsOrderedEntity(order.Id, product, order.OrderPlacedUtc));
    }

    // log order processed
    log.WriteLine($"Processed {order.Products.Count} products");
}
```

6. Build and Run the application

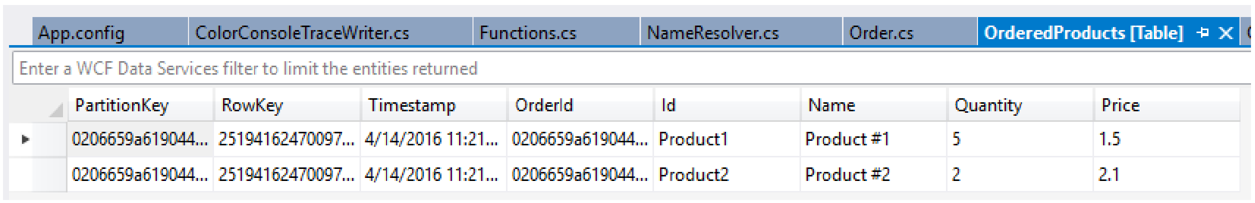
The console output will look the same as last time. In order to verify the table was created and rows were added, we need to look at Azure storage.

7. Open the Server Explorer and select Azure -> Storage. Find your storage account for the WebJob and expand the Tables node underneath. You should see the OrderedProducts table



8. Double click the OrderedProducts table name.

This will open the ProductsTable, where you should see two rows.


App.config
ColorConsoleTraceWriter.cs
Functions.cs
NameResolver.cs
Order.cs
OrderedProducts [Table]
Enter a WCF Data Services filter to limit the entities returned

PartitionKey	RowKey	Timestamp	OrderId	Id	Name	Quantity	Price
0206659a619044...	25194162470097...	4/14/2016 11:21...	0206659a619044...	Product1	Product #1	5	1.5
0206659a619044...	25194162470097...	4/14/2016 11:21...	0206659a619044...	Product2	Product #2	2	2.1

Add Functionality to Send an Email Once the Order is Processed

In this exercise you will use the SendGrid binding from the WebJobs SDK Extensions to send an email inside the WebJob.

The assumption is that you already have a SendGridApiKey by this point. If you do not, you can go here to sign up for a free one: https://sendgrid.com/free

1. Open app.config and create a new setting to hold your SendGridApiKey and an email to have in the from address.

```
<appSettings>
    <add key="OrdersQueueName" value="orders" />
    <add key="OrdersContainerName" value="orders" />
    <add key="ProductsTableName" value="OrderedProducts"/>
    <add key="AdminEmail" value="<your email here>"/>
    <add key="AzureWebJobsSendGridApiKey" value="…"/>
</appSettings>
```

Now you need to configure the SendGrid extension.

2. Open Program.cs and add the highlighted changes to your C#

```
using System.Configuration;
using System.Diagnostics;
using System.Net.Mail;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.SendGrid;

namespace OrderProcessingJob
{
    class Program
    {
        static void Main(string[] args)
        {
            var config = new JobHostConfiguration();

            config.NameResolver = new NameResolver();
               config.Tracing.Tracers.Add(new ColorConsoleTraceWriter(TraceLevel.Info)
            );
            config.UseSendGrid(new SendGridConfiguration()
            {
                FromAddress = new MailAddress(
                    ConfigurationManager.AppSettings["AdminEmail"])
            });

            config.UseDevelopmentSettings();

            var host = new JobHost(config);
            host.Call(typeof(Functions).GetMethod("PutTestOrderInQueue"));
            host.RunAndBlock();
        }
    }
}
```

3. Open the Functions class, and add the highlighted C#

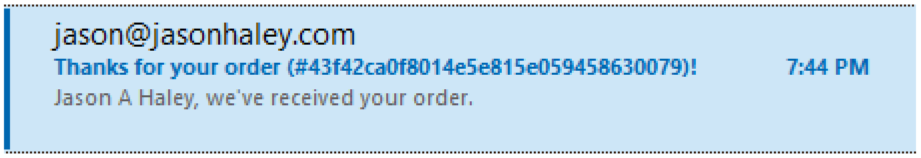
```
public static void ProcessOrder(
    [QueueTrigger("%OrdersQueueName%")] Order order,
    [Blob("%OrdersContainerName%/{Id}")] out Order orderToSave,
    [Table("%ProductsTableName%")] ICollector<ProductsOrderedEntity> productsOrderedTable,
    [SendGrid] SendGridMessage message,
    TextWriter log
    )
{
    // Save the order in blob storage
    orderToSave = order;

    // Save the ordered product quantities to table storage
    foreach (var product in order.Products)
    {
        productsOrderedTable.Add(
            new ProductsOrderedEntity(order.Id, product, order.OrderPlacedUtc));
    }

    // Send and email to the buyer
    message.AddTo(order.BuyerEmail);
    message.Subject = $"Thanks for your order (#{order.Id})!";
    message.Text = $"{order.BuyerName}, we've received your order.";

    // log order processed
    log.WriteLine($"Processed {order.Products.Count} products");
}
```

4. You will need to make sure you have the following using statement as well:

```
using SendGrid;
```

5. In the PutTestOrderInQueue method of the Functions class, make sure your BuyerEmail and BuyerName have good values (like your name and your email address).
6. Run the application

The console output will again be the same, however in a minute or two you should receive and email.


jason@jasonhaley.com
Thanks for your order (#43f42ca0f8014e5e815e059458630079)! 7:44 PM
Jason A Haley, we've received your order.

Add a function to handle poison messages on the Orders queue to send out an email

Sometimes messages can be full of unexpected data and cause a function to fail. This is managed in WebJobs by moving the message to a poison queue once the message has hit the MaxDequeueCount attemps. The poison queue has the same name as the other queue, but with a '-poison' appended to it. Example: 'orders-poison'

In this exercise we will create a function to handle and poison messages and also write a function to test it.

1. Open the Functions class and add the following C# to it:

```
public static void HandlePoisonMessage(
    [QueueTrigger("%OrdersQueueName%-poison")] string orderJson,
    [SendGrid] SendGridMessage message,
    TextWriter log)
{
    // Send and email to the admin
    message.AddTo(ConfigurationManager.AppSettings["AdminEmail"]);
    message.Subject = "Order Failed Notification";
    message.Text = orderJson;

    // log poison message
    log.WriteLine($"Poison Order: {orderJson}");
}
```

2. Add the following using statement to the top:

```
using System.Configuration;
```

This method will watch for messages on the orders-poison queue. When a message gets added to it, the function will send an email with text contents of the poison message.

3. Add the following method to the Functions class

```
[NoAutomaticTrigger]
public static void PutPoisonOrderInQueue(
    [Queue("orders")] out Order testOrder,
    TextWriter log)
{
    testOrder = new Order()
    {
        Id = Guid.NewGuid().ToString("N"),
        BuyerEmail = "<your email here>",
        BuyerName = "<your name here>",
        OrderPlacedUtc = DateTime.UtcNow
    };

    // log message
    log.WriteLine("Queued order");
}
```

This Order doesn't contain a Products list, which will cause errors and the removal of the message after 5 attempts it will get moved to the poison queue and run the method we wrote earlier.

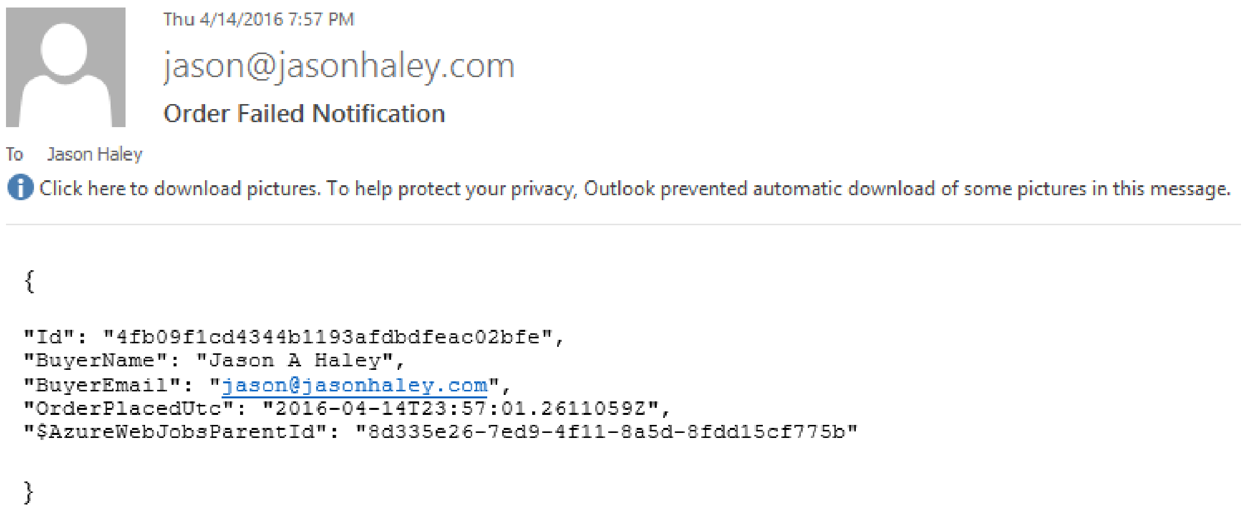
4. Open Program.cs and add the highlighted line to the Main method:

```
static void Main(string[] args)
{
    var config = new JobHostConfiguration();
    config.NameResolver = new NameResolver();
    config.Tracing.Tracers.Add(new ColorConsoleTraceWriter(TraceLevel.Info));
    config.UseSendGrid(new SendGridConfiguration()
    {
        FromAddress = new MailAddress(
                ConfigurationManager.AppSettings["AdminEmail"])
    });

    config.UseDevelopmentSettings();

    var host = new JobHost(config);
    host.Call(typeof(Functions).GetMethod("PutTestOrderInQueue"));
    host.Call(typeof(Functions).GetMethod("PutPoisonOrderInQueue"));
    host.RunAndBlock();
}
```

This new line will invoke the test method you just created.

5. Run the application.

This time you should see quite a bit of red messages and eventually an email will get sent.



Thu 4/14/2016 7:57 PM

jason@jasonhaley.com

Order Failed Notification

To Jason Haley

Click here to download pictures. To help protect your privacy, Outlook prevented automatic download of some pictures in this message.

```
{

"Id": "4fb09f1cd4344b1193afdbdfeac02bfe",
"BuyerName": "Jason A Haley",
"BuyerEmail": "jason@jasonhaley.com",
"OrderPlacedUtc": "2016-04-14T23:57:01.2611059Z",
"$AzureWebJobsParentId": "8d335e26-7ed9-4f11-8a5d-8fdd15cf775b"

}
```

Add a function that emails when a certain number of error have happened

In the last exercise, we dealt with the poison message scenario. In this exercise we will create a function that will use the ErrorTrigger to cause a function to run once a certain error threshold has been reached – in our case 10 errors in 5 minutes.

1. Open the Functions class and add the following C# method to it:

```
public static void HandleErrors(
    [ErrorTrigger("00:05:00", 10)] TraceFilter filter,
    [SendGrid] ref SendGridMessage message,
    TextWriter log)
{
    // Send and email to the admin
    message.AddTo(ConfigurationManager.AppSettings["AdminEmail"]);
    message.Subject = "WebJob Error Notification";
    message.Text = filter.GetDetailedMessage(5);

    // log last 5 detailed errors to the Dashboard
    log.WriteLine(filter.GetDetailedMessage(5));

}
```

This function will execute when 10 errors have been reached in a 5 minute period of time, then send an email with the last 5 errors to the administrator.

2. Add the following test method to the Functions class:

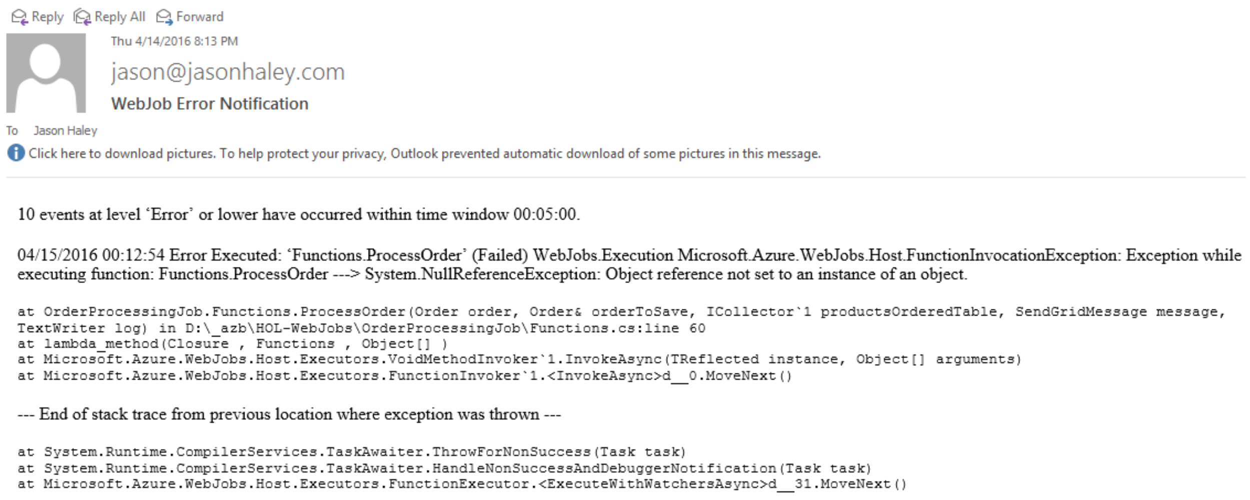
```
[NoAutomaticTrigger]
public static void PutTwoPoisonOrdersInQueue(
    [Queue("orders")] ICollector<Order> testOrders,
    TextWriter log)
{
    for (int i = 0; i < 2; i++)
    {
        testOrders.Add(new Order()
        {
            Id = Guid.NewGuid().ToString("N"),
            BuyerEmail = "<your email here>",
            BuyerName = "<your name here>",
            OrderPlacedUtc = DateTime.UtcNow
        });
    }

    // log message
    log.WriteLine("Queued order");
}
```

3. Open Program.cs and add the highlighted changes

```
static void Main(string[] args)
{
    var config = new JobHostConfiguration();

    config.NameResolver = new NameResolver();
    config.Tracing.Tracers.Add(new ColorConsoleTraceWriter(TraceLevel.Info));
    config.UseSendGrid(new SendGridConfiguration()
    {
        FromAddress = new MailAddress(
                ConfigurationManager.AppSettings["AdminEmail"])
    });
    config.UseCore();

    config.UseDevelopmentSettings();

    var host = new JobHost(config);
    host.Call(typeof(Functions).GetMethod("PutTestOrderInQueue"));
    host.Call(typeof(Functions).GetMethod("PutPoisonOrderInQueue"));
    host.Call(typeof(Functions).GetMethod("PutTwoPoisonOrdersInQueue"));
    host.RunAndBlock();
}
```

4. Run the application

Eventually you will start getting emails from both the poison queue and the error handler. Once you've proven the error handler works, you can stop the debugging.



Thu 4/14/2016 8:13 PM

jason@jasonhaley.com

WebJob Error Notification

To Jason Haley

Click here to download pictures. To help protect your privacy, Outlook prevented automatic download of some pictures in this message.

10 events at level 'Error' or lower have occurred within time window 00:05:00.

04/15/2016 00:12:54 Error Executed: 'Functions.ProcessOrder' (Failed) WebJobs.Execution Microsoft.Azure.WebJobs.Host.FunctionInvocationException: Exception while executing function: Functions.ProcessOrder ---> System.NullReferenceException: Object reference not set to an instance of an object.

```
at OrderProcessingJob.Functions.ProcessOrder(Order order, Order& orderToSave, ICollector`1 productsOrderedTable, SendGridMessage message,
TextWriter log) in D:\_azb\HOL-WebJobs\OrderProcessingJob\Functions.cs:line 60
at lambda_method(Closure , Functions , Object[] )
at Microsoft.Azure.WebJobs.Host.Executors.VoidMethodInvoker`1.InvokeAsync(TReflected instance, Object[] arguments)
at Microsoft.Azure.WebJobs.Host.Executors.FunctionInvoker`1.<InvokeAsync>d__0.MoveNext()
```

--- End of stack trace from previous location where exception was thrown ---

```
at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)
at System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)
at Microsoft.Azure.WebJobs.Host.Executors.FunctionExecutor.<ExecuteWithWatchersAsync>d__31.MoveNext()
```

Deploy the WebJob to Azure

Now that you have a WebJob and have proven the functionality works by testing it locally, it is time to deploy it to Azure.

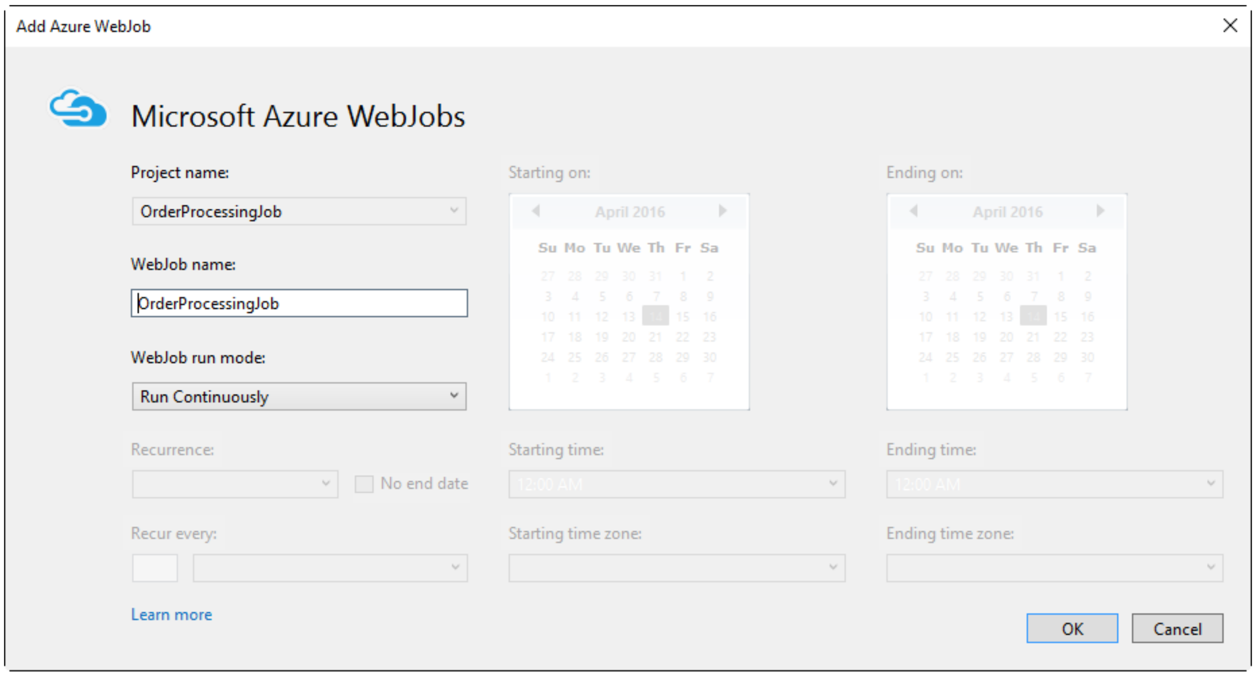
Before deploying it to Azure, you need to disable the testing methods.

1. Open Program.cs and delete or comment out the highlighted lines:

```
static void Main(string[] args)
{
    var config = new JobHostConfiguration();

    config.NameResolver = new NameResolver();
    config.Tracing.Tracers.Add(new ColorConsoleTraceWriter(TraceLevel.Info));
    config.UseSendGrid(new SendGridConfiguration()
    {
        FromAddress = new MailAddress(
                ConfigurationManager.AppSettings["AdminEmail"])
    });
    config.UseCore();

    config.UseDevelopmentSettings();

    var host = new JobHost(config);
    //host.Call(typeof(Functions).GetMethod("PutTestOrderInQueue"));
    //host.Call(typeof(Functions).GetMethod("PutPoisonOrderInQueue"));
    //host.Call(typeof(Functions).GetMethod("PutTwoPoisonOrdersInQueue"));
    host.RunAndBlock();
}
```

2. Right-click the OrderProcessingJob and select Publish as Azure Web Job…

This will open the Add Azure WebJob dialog



Project name: OrderProcessingJob

WebJob name: OrderProcessingJob

WebJob run mode: Run Continuously

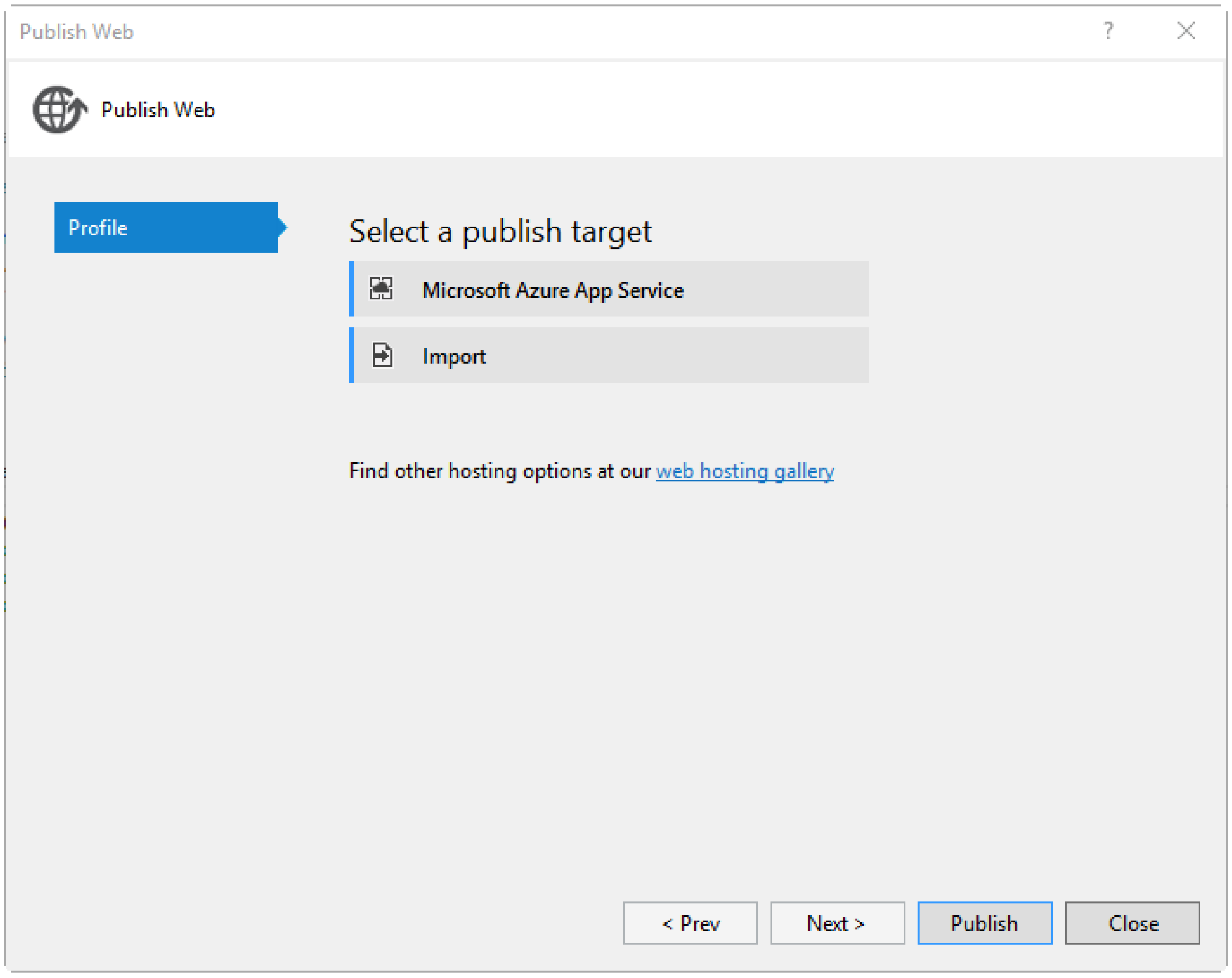
Recurrence:

No end date

Recur every:

3. Click OK

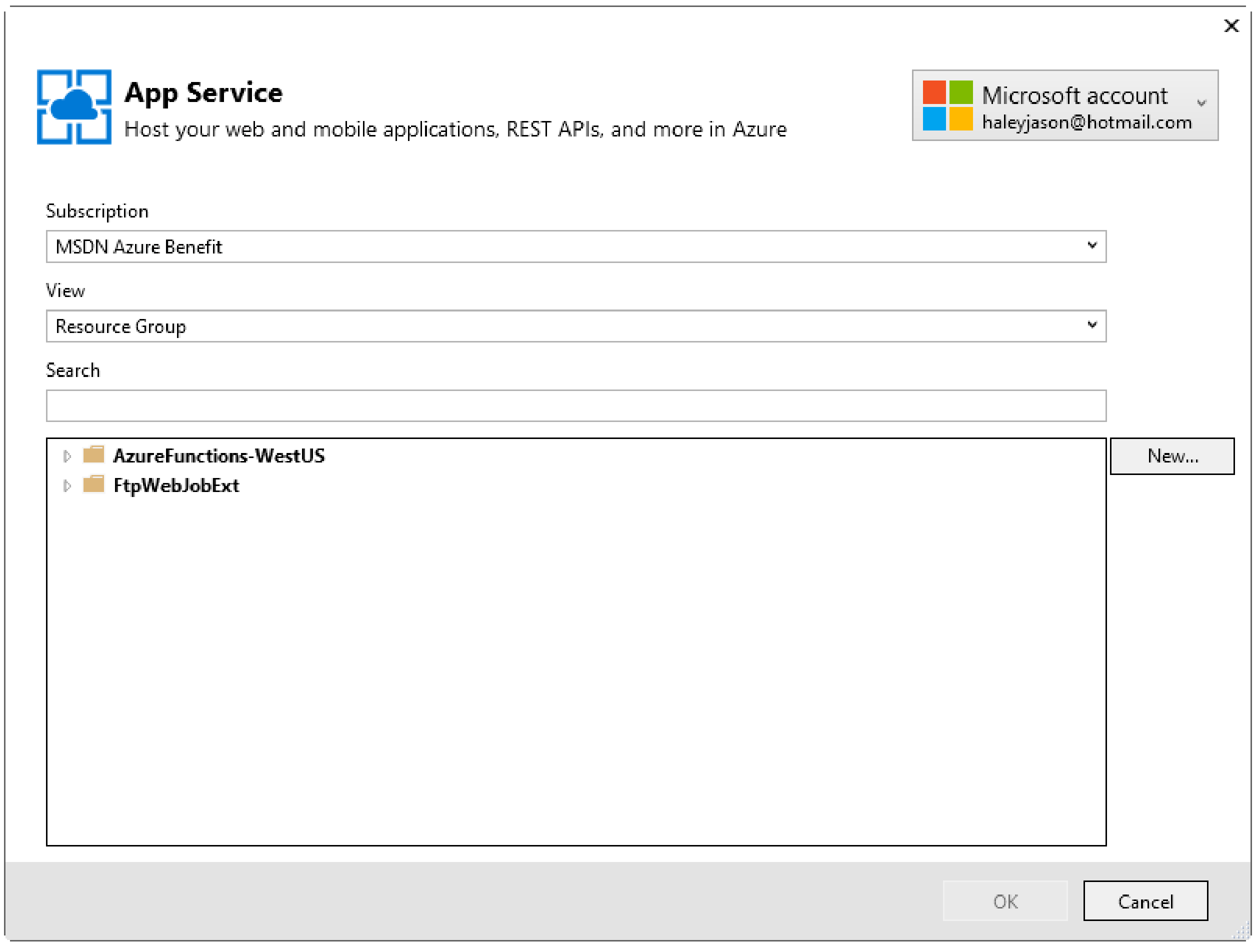
This will open the Publish Web dialog



Select a publish target

Find other hosting options at our web hosting gallery

4. Click the Microsoft Azure App Service button


App Service
Host your web and mobile applications, REST APIs, and more in Azure
Microsoft account
haleyjason@hotmail.com

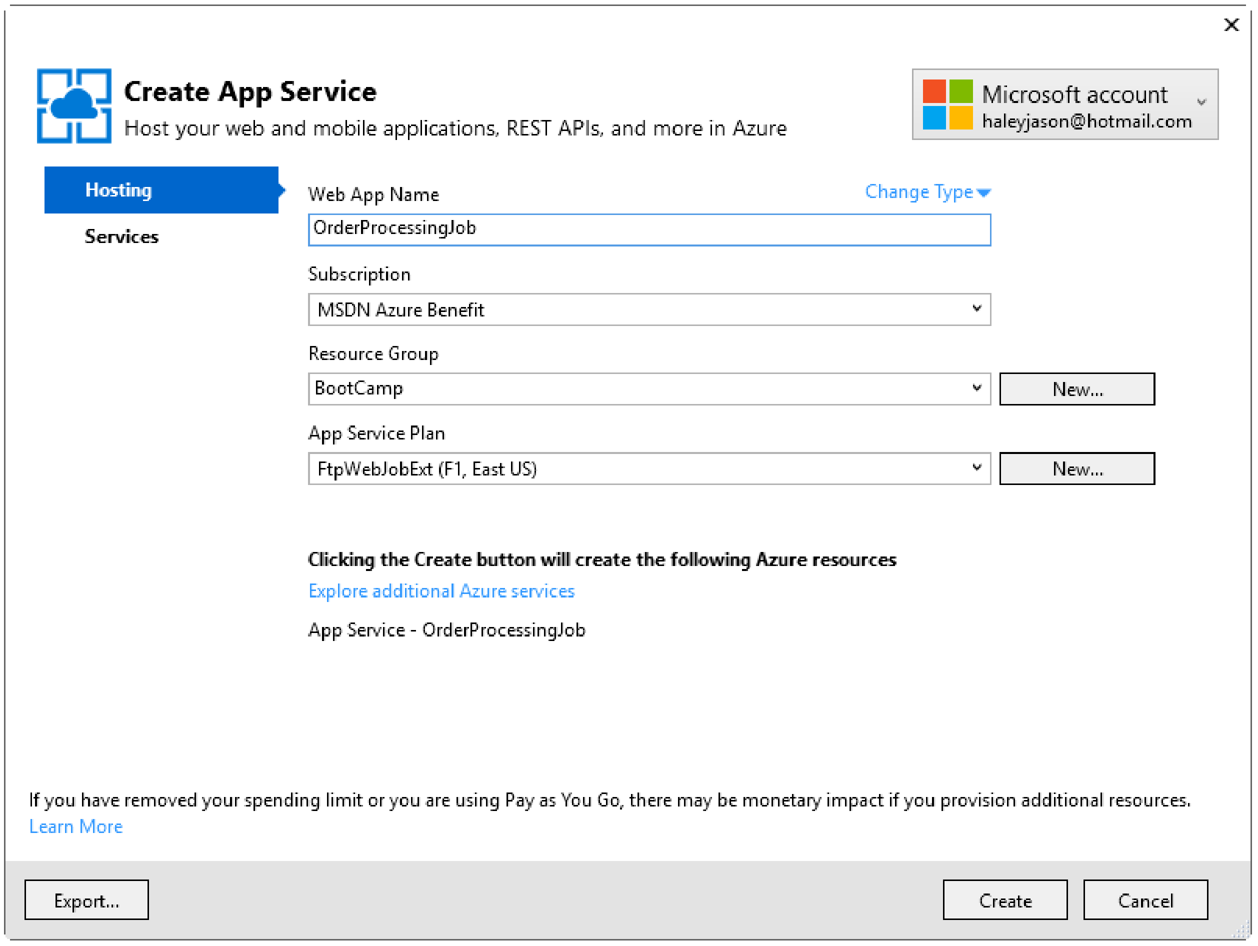
Subscription: MSDN Azure Benefit

View: Resource Group

Search:

- AzureFunctions-WestUS
- FtpWebJobExt

5. Select your subscription from the dropdown, click the New… button on the right



Web App Name: OrderProcessingJob

Subscription: MSDN Azure Benefit

Resource Group: BootCamp

App Service Plan: FtpWebJobExt (F1, East US)

Clicking the Create button will create the following Azure resources

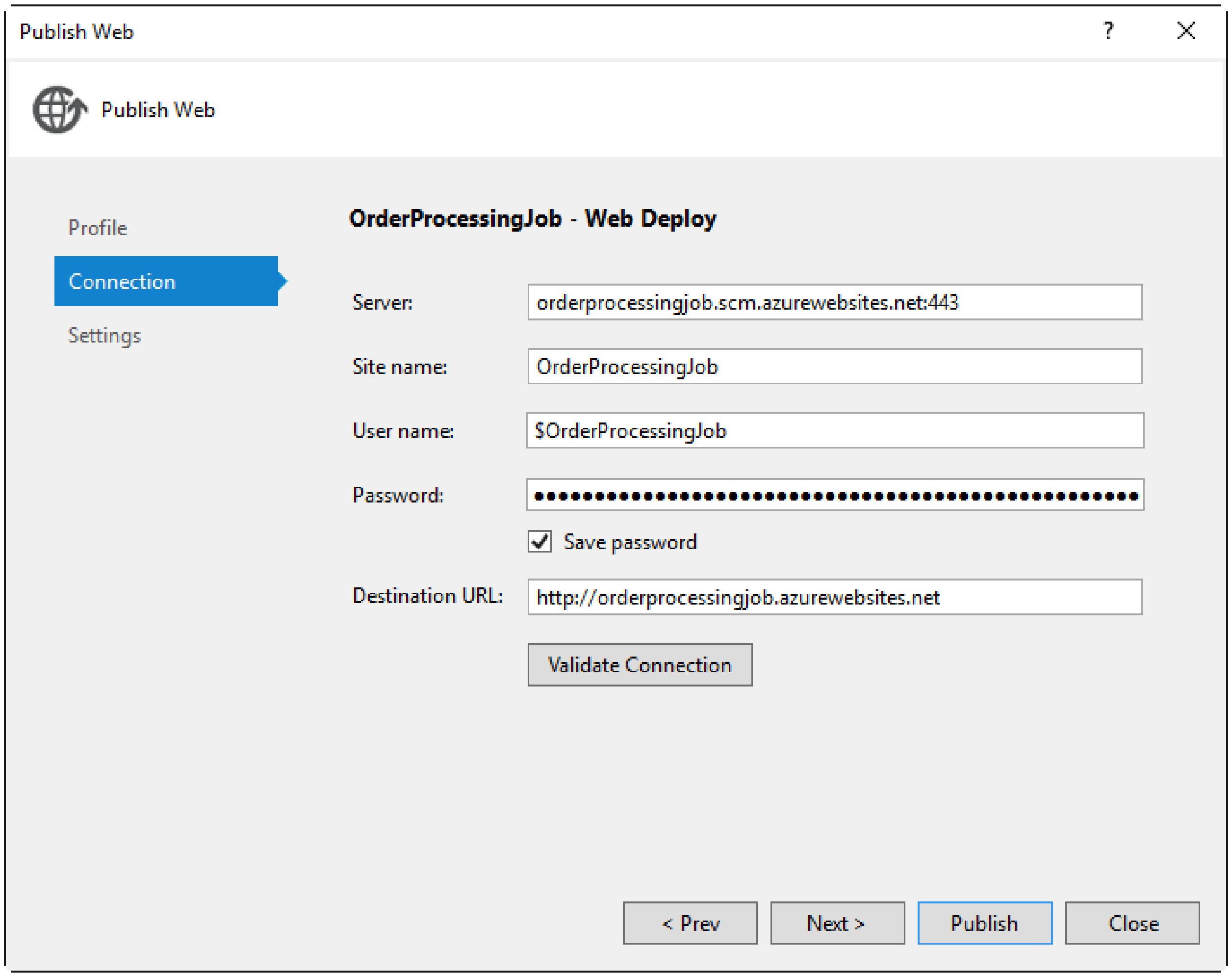
Explore additional Azure services

App Service - OrderProcessingJob

If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources.
Learn More

6. Fill in your Hosting details:
   - Web App Name: OrderProcessingJob
   - Subscription:
   - Resource Group: select an existing or click the New button to create a new one
   - App Service Plan: select an existing or click the New button to create a new one
7. Click Create

The dialog will go out to Azure and create an App Service to deploy the WebJob to. Once it is ready, the Publish Web dialog will show with all the Connection information populated.



Server: orderprocessingjob.scm.azurewebsites.net:443

Site name: OrderProcessingJob

User name: $OrderProcessingJob

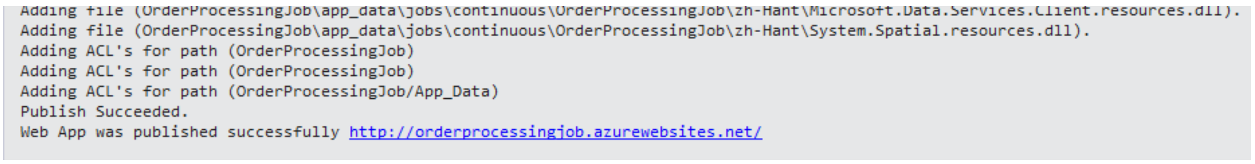
Password: ●●

✓ Save password

Destination URL: http://orderprocessingjob.azurewebsites.net

8. Click Publish

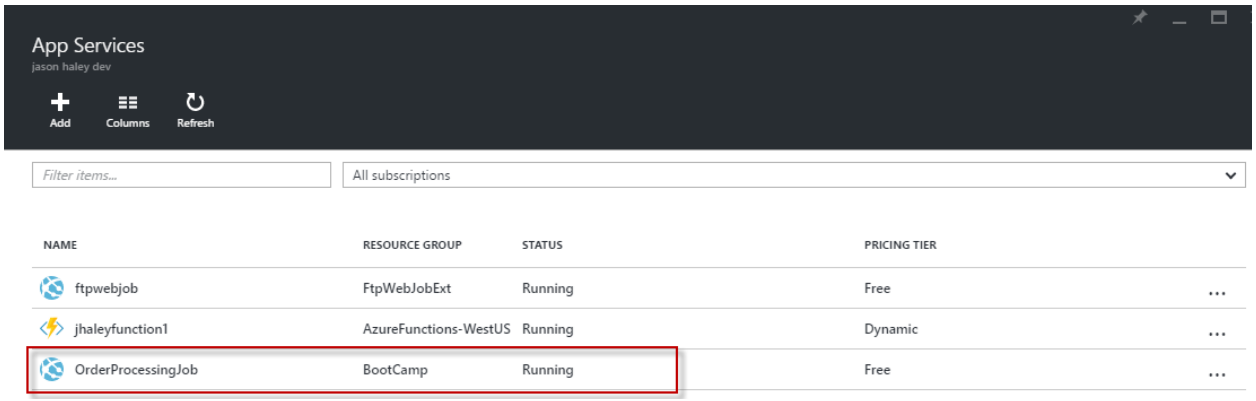
This will package and deploy the webjob out to the site.



```
Adding file (OrderProcessingJob\app_data\jobs\continuous\OrderProcessingJob\zh-Hant\Microsoft.Data.Services.Client.resources.dll).
Adding file (OrderProcessingJob\app_data\jobs\continuous\OrderProcessingJob\zh-Hant\System.Spatial.resources.dll).
Adding ACL's for path (OrderProcessingJob)
Adding ACL's for path (OrderProcessingJob)
Adding ACL's for path (OrderProcessingJob/App_Data)
Publish Succeeded.
Web App was published successfully http://orderprocessingjob.azurewebsites.net/
```

9. Open the Azure portal (https://portal.azure.com)
10. Select App Services from the menu

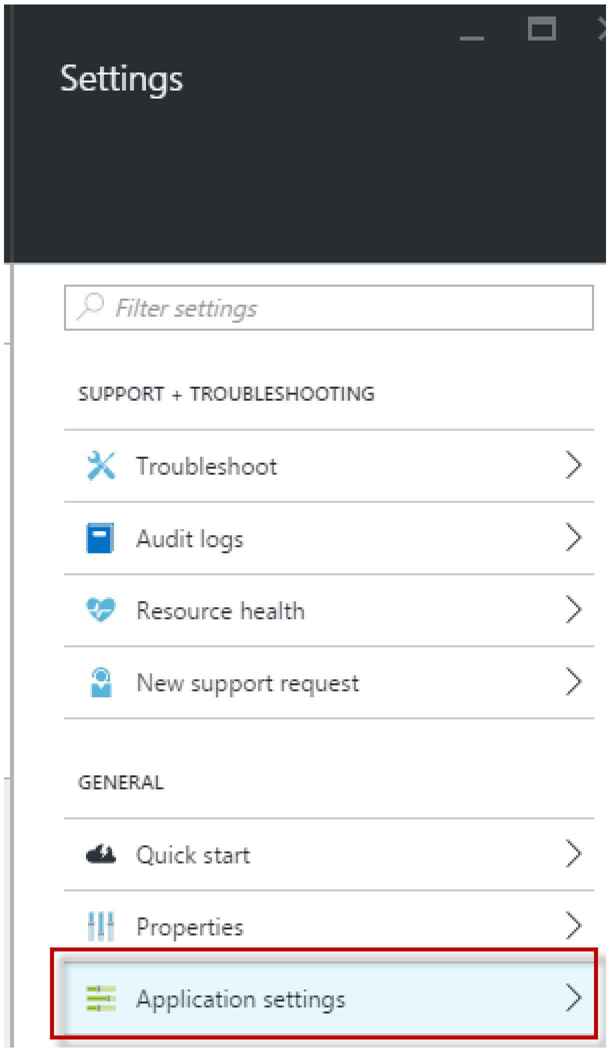
You should see your app service in the listing


App Services
jason haley dev

NAME	RESOURCE GROUP	STATUS	PRICING TIER
ftpwebjob	FtpWebJobExt	Running	Free
jhaleyfunction1	AzureFunctions-WestUS	Running	Dynamic
OrderProcessingJob	BootCamp	Running	Free

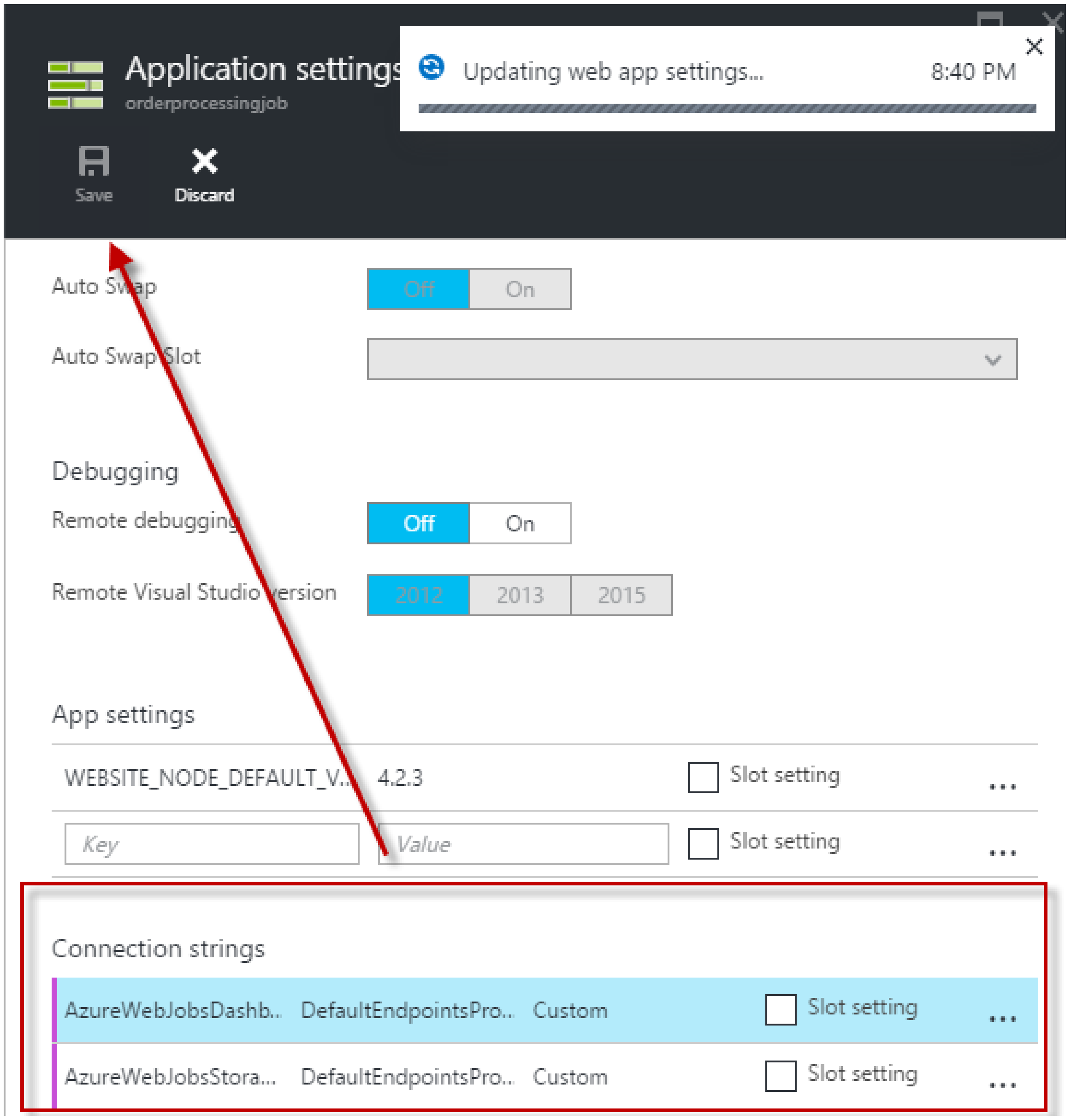
The WebJobs dashboard needs the two app settings that you added to your config file added to the website in order to open up all the functionality it provides.

11. Select the App Service. Once the Web app panel is open and the Settings blade, scroll to the Application settings menu item and click it.



12. Scroll to the Connection Strings area of the Application Settings panel

13. In Visual Studio, open the app.config file and copy the AzureWebJobsDashboard and AzureWebJobsStorage over to Connection Strings in the panel.
14. Click the Save button at the top


Application settings
orderprocessingjob
Updating web app settings... 8:40 PM

Auto Swap: Off
Auto Swap Slot

Debugging
Remote debugging: Off
Remote Visual Studio version: 2012

App settings

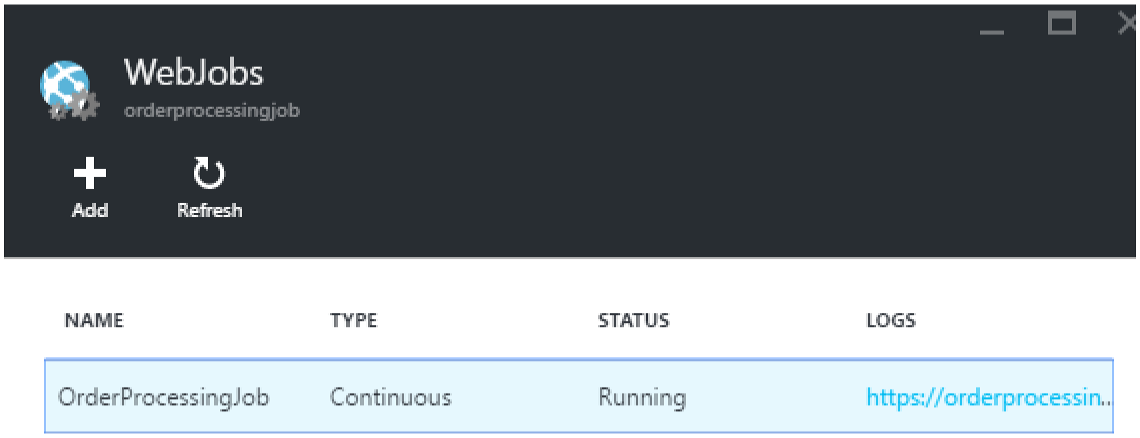
WEBSITE_NODE_DEFAULT_V...	4.2.3	Slot setting	...
Key	Value	Slot setting	...

Connection strings

AzureWebJobsDashb...	DefaultEndpointsPro...	Custom	Slot setting	...
AzureWebJobsStora...	DefaultEndpointsPro...	Custom	Slot setting	...

15. In the Settings menu, scroll down to the WebJobs menu and click it.

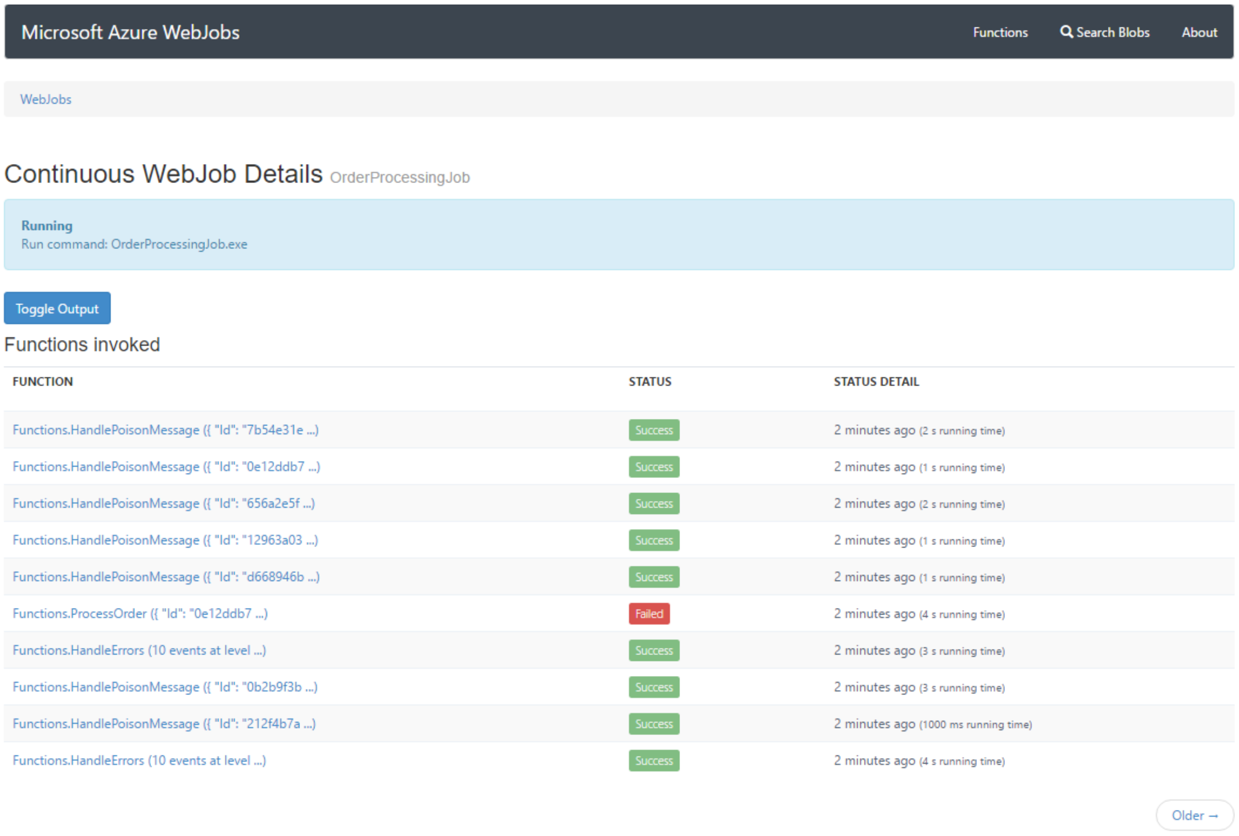
This will show the list of WebJobs running.



NAME	TYPE	STATUS	LOGS
OrderProcessingJob	Continuous	Running	https://orderprocessin..

16. Double click the link in the logs column.

This will take you to the WebJobs dashboard, where you can navigate and see all the logging information for the different runs of the WebJobs functions and their status.



Continuous WebJob Details OrderProcessingJob

Running
Run command: OrderProcessingJob.exe

Functions invoked

FUNCTION	STATUS	STATUS DETAIL
Functions.HandlePoisonMessage ({ "Id": "7b54e31e ...)	Success	2 minutes ago (2 s running time)
Functions.HandlePoisonMessage ({ "Id": "0e12ddb7 ...)	Success	2 minutes ago (1 s running time)
Functions.HandlePoisonMessage ({ "Id": "656a2e5f ...)	Success	2 minutes ago (2 s running time)
Functions.HandlePoisonMessage ({ "Id": "12963a03 ...)	Success	2 minutes ago (1 s running time)
Functions.HandlePoisonMessage ({ "Id": "d668946b ...)	Success	2 minutes ago (1 s running time)
Functions.ProcessOrder ({ "Id": "0e12ddb7 ...)	Failed	2 minutes ago (4 s running time)
Functions.HandleErrors (10 events at level ...)	Success	2 minutes ago (3 s running time)
Functions.HandlePoisonMessage ({ "Id": "0b2b9f3b ...)	Success	2 minutes ago (3 s running time)
Functions.HandlePoisonMessage ({ "Id": "212f4b7a ...)	Success	2 minutes ago (1000 ms running time)
Functions.HandleErrors (10 events at level ...)	Success	2 minutes ago (4 s running time)