**Introduction to Azure Logic Apps**

Student Lab Manual

V1.4, July 28, 2019

Information in this document is subject to change without notice. The example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarked, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2019 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, MS, Windows, Windows NT, MSDN, Active Directory, BizTalk, SQL Server, SharePoint, Outlook, PowerPoint, FrontPage, Visual Basic, Visual C++, Visual J++, Visual InterDev, Visual SourceSafe, Visual C#, Visual J#, and Visual Studio are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

Other product and company names herein may be the trademarks of their respective owners.

Contents

[Introduction to Azure Logic Apps 4](#_Toc15223466)

[Introduction 4](#_Toc15223467)

[Objectives 4](#_Toc15223468)

[Prerequisites 4](#_Toc15223469)

[Exercise 1: Setting up the infrastructure needed for the lab 5](#_Toc15223470)

[Task 1 – Create Azure resources using an ARM template 5](#_Toc15223471)

[Task 2 – Collect your Azure resource keys 9](#_Toc15223472)

[Task 3 – Configure CosmosDB and the Azure Function app settings 10](#_Toc15223473)

[Exercise 2: Creating your Logic App 12](#_Toc15223474)

[Scenario 12](#_Toc15223475)

[Task 1 – Create your Logic App 12](#_Toc15223476)

[Task 2 – Build the logic in your Logic App 14](#_Toc15223477)

[Task 3 – Testing your Logic App 25](#_Toc15223478)

[Task 4 – Reviewing your message in the Service Bus queue (optional) 29](#_Toc15223479)

[Appendix A – How to Package up your Azure Function and deploy to Storage 32](#_Toc15223480)

# Introduction to Azure Logic Apps

## Introduction

Estimated time to complete this lab

60 minutes

#### Overview

[Azure Logic Apps](https://azure.microsoft.com/services/logic-apps) is a cloud service that helps you automate and orchestrate tasks, business processes, and [workflows](https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-overview#logic-app-concepts) when you need to integrate apps, data, systems, and services across enterprises or organizations. Logic Apps simplifies how you design and build scalable solutions for app integration, data integration, system integration, enterprise application integration (EAI), and business-to-business (B2B) communication, whether in the cloud, on premises, or both.

### Objectives

In this hands-on lab, you will learn how to:

* Create a logic app that has a listen trigger for an http request
* Use a flow control for-each connector to loop through a list of orders
* Use a logic app connector to call a pre-existing Azure Function to insert records into a pre-existing CosmosDB database
* Use a logic app connector to insert data in to a Service Bus queue

## Prerequisites

The following is required to complete this hands-on lab:

* Microsoft V[isual Studio 2017 Professional or Enterprise edition](http://www.microsoft.com/visualstudio/), version 15.9.13+
* [Microsoft Azure SDK for .NET for Visual Studio 2017](http://www.microsoft.com/windowsazure/sdk/)
* A Microsoft Azure subscription
* Azure Storage Explorer <https://azure.microsoft.com/en-us/features/storage-explorer/>
* Service Bus Explorer <https://github.com/paolosalvatori/ServiceBusExplorer>

# Exercise 1: Setting up the infrastructure needed for the lab

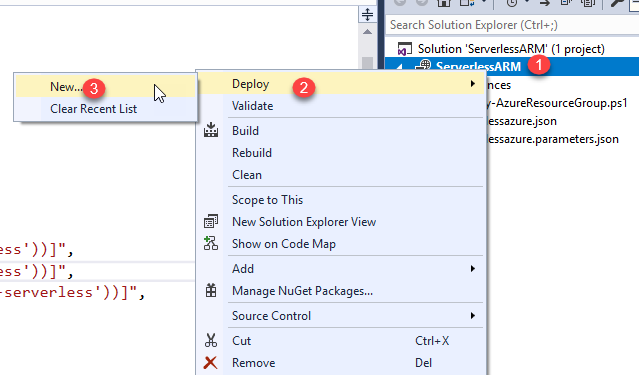
In this exercise, you will perform several different operations such as deploying an ARM template and executing the Azure CLI via PowerShell in order to setup the infrastructure needed for your logic app.

## Task 1 – Create Azure resources using an ARM template

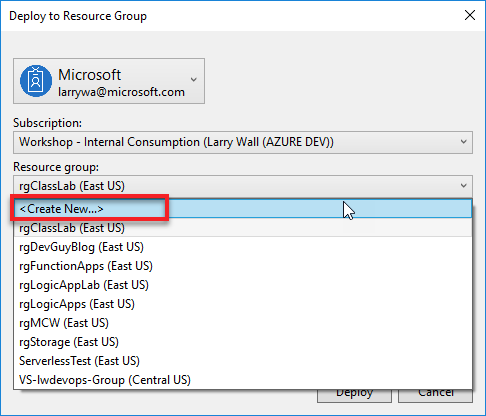
ARM templates are typically used to deploy the infrastructure that other Azure Services require. In your case, the ARM template will create:

* An Azure Storage account
* An Azure CosmosDB account
* An Azure Function App
* An Azure Service Bus namespace with a Queue
* A server hosting plan, used by the Azure Function app

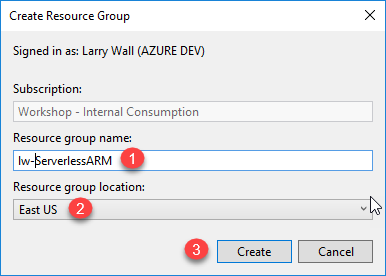
1. Using Visual Studio, open the **.\LogicApps\Labs\IntroToLogicApps\ServerlessARM\ServerlessARM.sln** solution.
2. This ARM template is ready to go except for one parameter that you need to enter.
3. Right-click on the **ServerlessARM** ***project*** (not the ServerlessARM solution) and select **Deploy | New**.



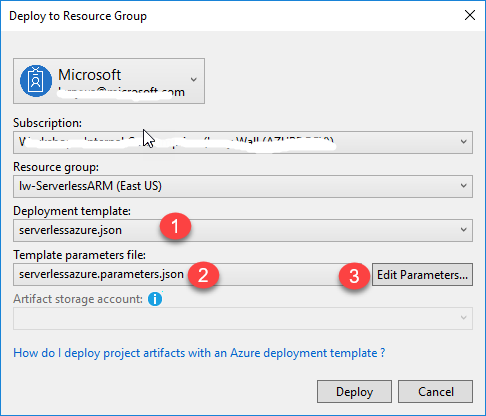
1. When the *Deploy to Azure Resource* group dialog box appears, you will need to make sure you log in to your Azure subscription (you should be prompted to do so if you have not already done so). Choose your subscription and then in the Resource Group drop-down, select the Create New menu item.



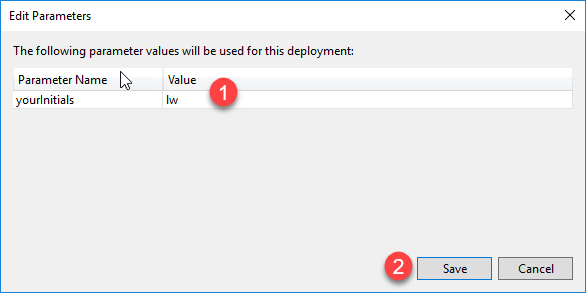
1. You can name the resource group anything you want but name it in such a way that it describes what you are working on. You can just add your ‘initials-‘ in front of the name that is provided if you want, then choose the region and select **Create**.



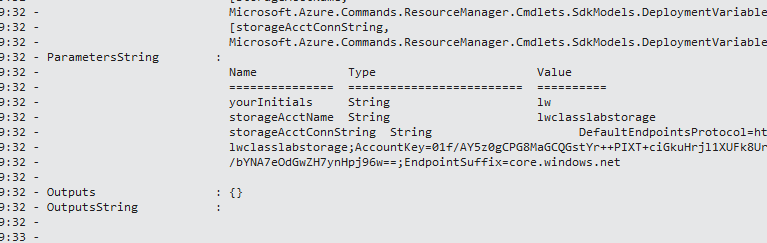
1. By default, the ARM template for deployment should already be selected in the *Deployment template* drop-down field and the parameters file that will be used with the template should show in the *Template parameters file* drop-down. Click on the **Edit Parameters** button.



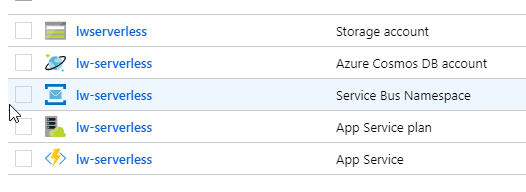
1. In the Edit parameters dialog box, you need to enter your Initials. When you have entered your parameters, select the **Save** button.



1. Select the **Deploy** button in the **Deploy to Resource Group** dialog box.
2. When the deployment has completed, you should see something similar to this in your Visual Studio output window.



1. Go back to the Azure portal to the resource group where you deployed the resources to and then confirm that your resources have been created. You may need to refresh the screen if you do not see your resources.



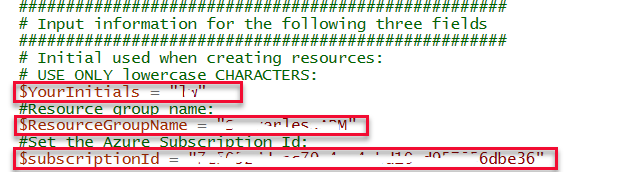
## Task 2 – Collect your Azure resource keys

As stated earlier, executing an ARM template sets up the infrastructure for other Azure services to use. Some things that are not setup with the ARM template, include:

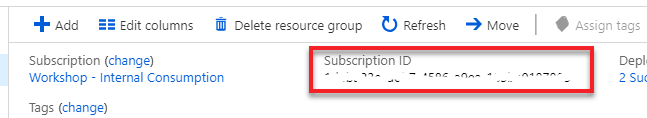
* Application settings for your Function App
* Storage account connection string used by the Function App
* Your CosmosDB database and collection to hold data

While setting up your logic app in the portal, it will need information about the resources you created, such as connection strings etc. You could go in to each one of the Azure resources and copy/paste all of these settings out to Notepad, but that is error prone. Instead, we have a PowerShell script that will extract all that information for you into a text file.

1. Open PowerShell ISE as an administrator.
2. Select **File | Open** and browse to the **.\LogicApps\Labs\IntroToLogicApps\Assets** folder and open the **WriteDataSettingsToFile.ps1**.
3. You will notice at the top of the file, you will need to put in 3 parameters, your initials, the resource group you created earlier and then your Azure subscription ID:



To find your subscriptionID, you can click on any of your Azure resources in the portal and then you can copy the subscription ID from that location.



1. Once you have entered the parameters, save the file and then select **F5** to run the PowerShell script. You will be prompted to log in to Azure to complete the running of the script.
2. When the script has completed executing, you will see that a new folder has been created on the C: drive named ‘*serverless’*. Using Windows Explorer, browse to that folder and open the **configuration.txt** file. You will use some of the values in the configuration.txt file in the next task.

## Task 3 – Configure CosmosDB and the Azure Function app settings

An ARM template does not have the capability to create a CosmosDB database and collection, nor can it deploy Azure Function code like we need for it to do in this lab. To create these items, you are going to be using Azure CLI (command line interface) commands, that are inside of a PowerShell script.

1. Open PowerShell ISE as an administrator and open the **.\LogicApps\Labs\IntroToLogicApps\Assets\ConfigureDeployment.ps1**.

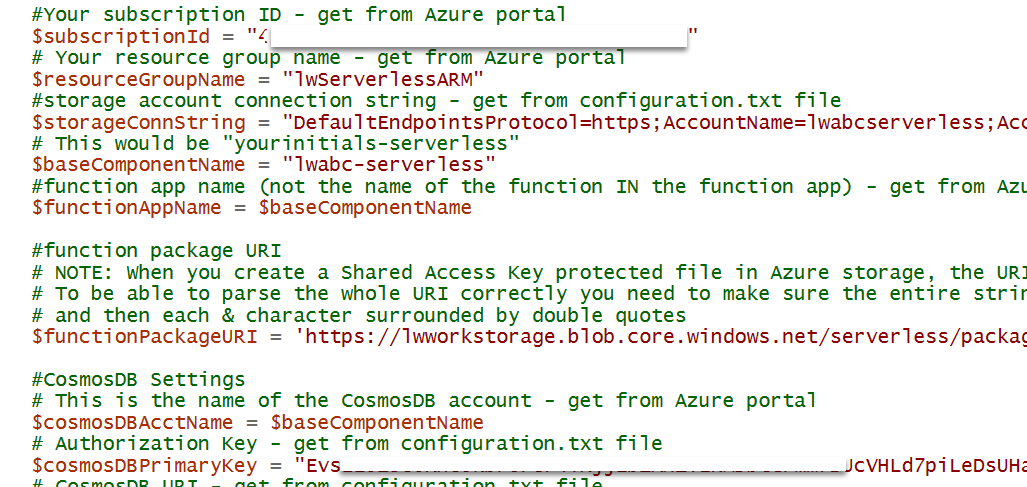
There are several parameters that you need to enter into the PowerShell script:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| $subscriptionId | The ID of your Azure subscription |
| $resourceGroupName | The name of the resource group where this labs Azure resources reside |
| $storageConnString | The Primary connection string for the storage account being used for this lab |
| $functionAppName | The name of the function app that was deployed by the ARM template deployment |
| $functionPackageURI | The location of the package that contains the source code for the Azure Function. **For this lab, the URI will already be in the PS script, do not modify it.** |
| $cosmosDBAcctName | The name of the CosmosDB account that was created by the ARM template deployment |
| $cosmosDBAuthKey | The authentication key that allows you to log in to CosmosDB |
| $cosmosDBAcctURI | The URI of your CosmosDB account |
| $cosmosDBDatabaseName | The name of the database that will be used by the Azure Function called from the logic app that holds orders |
| $cosmosDBAcctURI | The URI endpoint of your CosmosDB account. You can find this in the configuration.txt file or in the Azure portal |

Other parameters you’ll see in the script that you **DON’T** need to change

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| $cosmosDBCollectionName | The name of the collection in the $cosmosDBDatabaseName database that holds the order records |
| $cosmosDBPartition | The partition in the $cosmosDBCollectionName where orders are stored |
| $app\_websiteRunFromPackage | The Azure Function that will be called by your logic app is stored as a package in Azure storage. This appsetting is the address of the location of the package |
| $app\_cosmosDBURI | The appsetting that contains the URI of your CosmosDB account. This is used by the Azure Function |
| $app\_collectionName | The appsetting that contains the name of the collection to store the order records in. This is used by the Azure Function. |
| $app\_databaseName | The appsetting that contains the name of the database to store order records in. This is used by the Azure Function. |
| $app\_authorizationKey | The appsetting that contains the authorization key that allows the Azure Function to insert records in to the orders database. |
| $app\_websiteContentAzureFileConnectionString | This appsetting represents the file path to the function app code, which is stored in an package in Azure storage |
| $app\_azureWebJobsStorage | Typically used for not http triggered functions, a connection string to the projects storage account |
| $app\_azureWebJobsDashboard | Typically used for not http triggered functions, a connection string to the projects storage account |

1. Set the above parameters to the values associated with your currently deployed Azure resources. Here is an example screenshot of what these parameter settings may look like:



1. Select **F5** to setup the Azure resource configuration.

# Exercise 2: Creating your Logic App

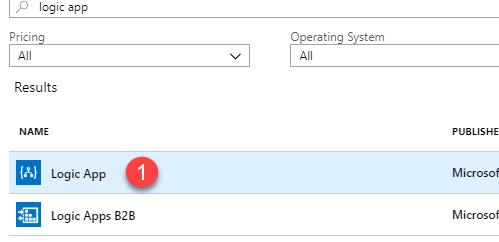
## Scenario

In this scenario, you will build a Logic app using the Azure portal that:

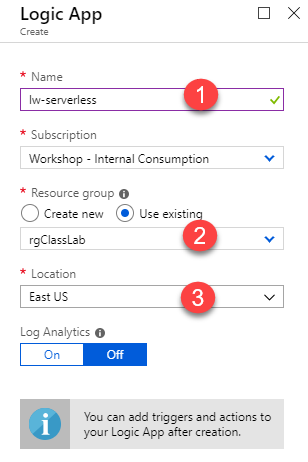
* Uses an http trigger to accept a JSON document that represents a collection of product orders from various customers
* For each order in the collection, the logic app will send the order to an Azure Function that will insert the record in to a CosmosDB database
* Upon successful completion of the insertion of the order into the CosmosDB database, you will then insert a record into a Service Bus queue

## Task 1 – Create your Logic App

1. Log in to the Azure portal if you have not already done so.
2. Go to the resource group that contains the resources you deployed via the ARM template earlier in this lab.
3. Click on the **+Add** toolbar button.
4. In the Marketplace blade, type in ‘*logic app’* and select **Enter**. You should see an icon for a **Logic App**. Select the Logic App icon and then click on the **Create** button.

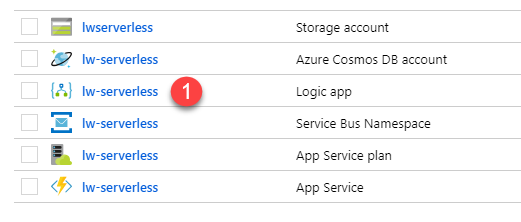


1. In the Logic app creation blade, enter a name for the logic app, choose the resource group you are currently in and a location. Click the **Create** button.

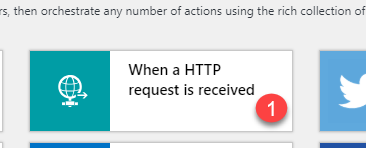


## Task 2 – Build the logic in your Logic App

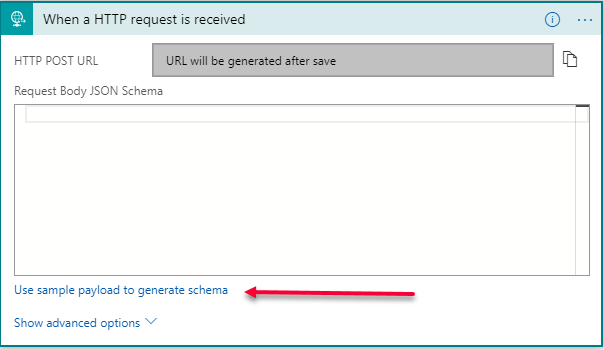
1. Once Azure has finished creating the empty logic app, you should see its name listed in the list of resources in your resource group. Click on the name of the logic app.



1. You will be taken directly to the Logic App designer. From this window, you can select pre-existing templates for logic apps or create a blank logic app and start from scratch. For this lab, you will be using an http trigger to kick off the logic app. Select the tile ‘**When an HTTP request is received**’.

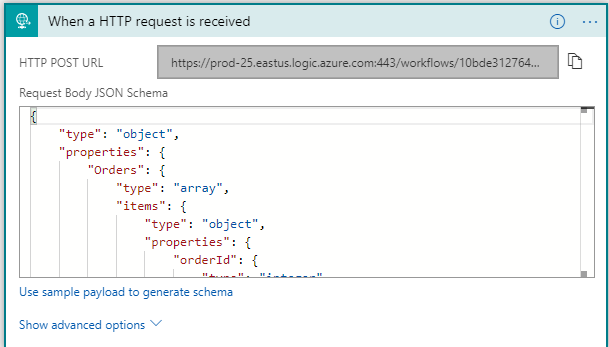


1. In the Logic Apps designer window, you will see the http trigger.

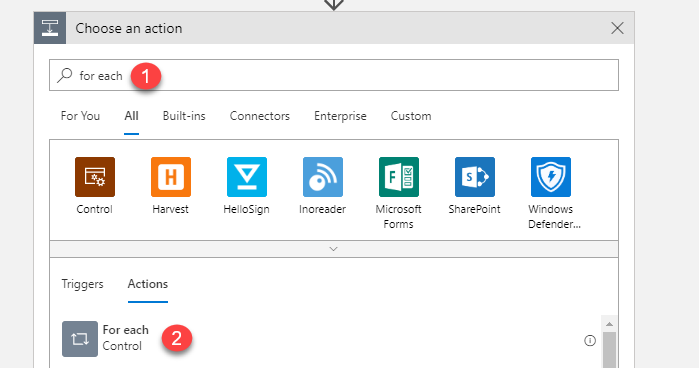


The http trigger gives us the capability to either paste in the JSON schema of the message that is going to be passed in from outside, or we can point to an actual JSON document and the schema will be inferred from that live document. That is what you will do in this lab.

1. Click on the **Use sample payload to generate schema** link.
2. Using the Windows Explorer, browse to and open **.\LogicApps\Labs\IntroToLogicApps\Assets\Orders.json**. Open this file in Notepad or your favorite editor.
3. Copy the contents of this file to the clipboard.
4. Paste the contents of the **Order.json** file in to the ‘**Enter or paste a sample JSON payload’** edit field and select the **Done** button.
5. You will now see the inferred JSON schema. Click the **Save** button on the toolbar. When you click Save, a URL will be created for the http listener address.

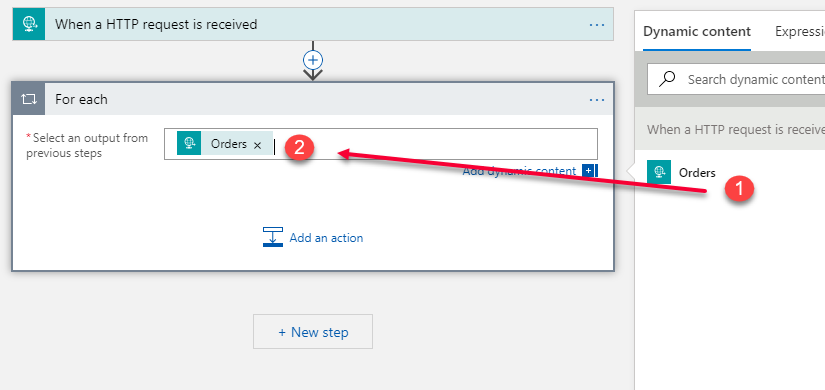


1. Below the http trigger, you will see a button named ‘+New step’. Click the **+New step** button.
2. Type ‘*for each*’ in to the *Choose an action* search field and then click on the **For each Control** icon.

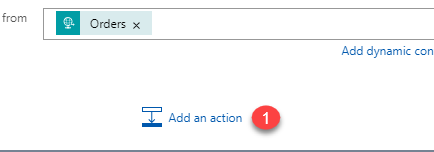


1. For the purposes of the lab, you are going to be focused on the Orders object that will be passed in to the Http trigger. The *for each* control has the capability to either sequentially loop through the array of orders, or it can be setup to run each order in parallel to improve performance. The *for each* loop executes in parallel by default.

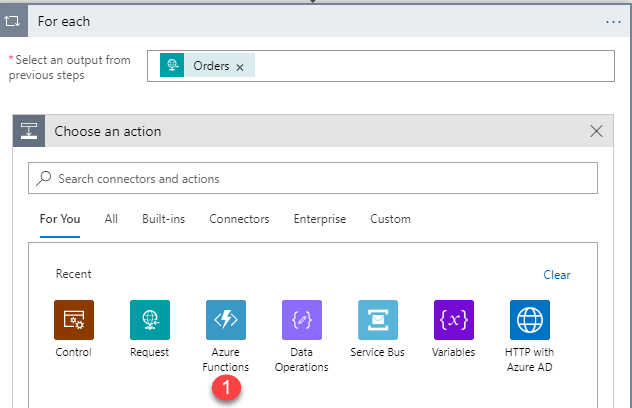
Over to the right of the for each action, click on the **Orders** icon. When you click on the Orders icon in the *Dynamic content* window, it will copy the Orders icon over to the edit field of the for each action. **Save** the logic app in the designer after this step.



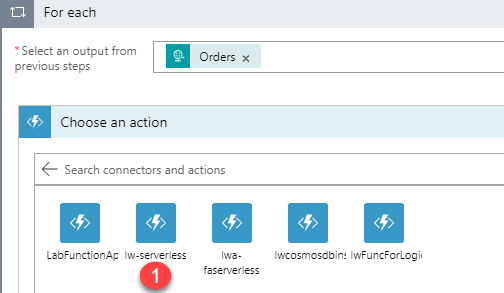
1. Inside of the For each action, you will see an ‘**Add an action**’ icon. Click on this icon.



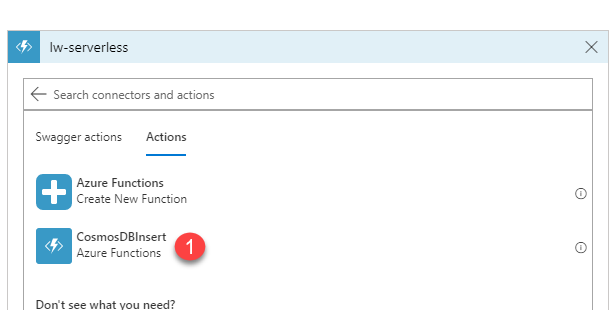
1. In this step, you are going to add a Functions connector in order to call the pre-existing Azure Function that will pass the order record to the Azure CosmosDB database you created. In the *Choose an action* dialog, click on the **Azure Functions** icon.



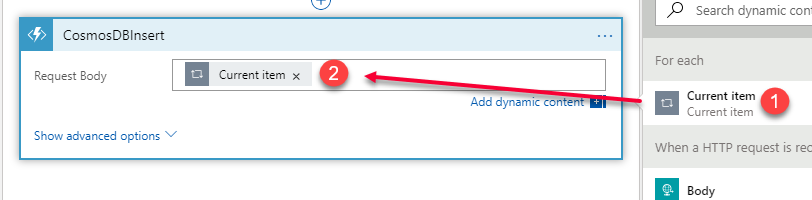
1. Assuming that your Azure Function is up and running in the same subscription as your logic app, you should see the name of your *Azure Function App* (not the name of the logic app) in an icon. Click on the icon that represents that name of your Azure Function app.



1. What you should see next are two icons, one for creating a new Azure Function (in the Function App you just selected) and the other for choosing a Function that is inside of the Function app. Choose the **CosmosDBInsert** icon.

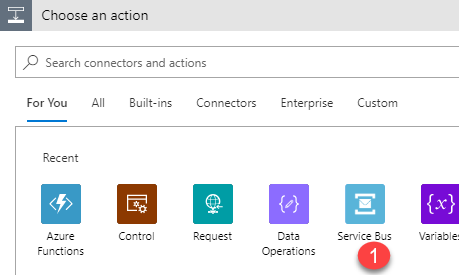


1. Once the function is inserted, click in the **Request Body** edit field. This will bring up the Dynamic content window and from there, choose **Current Item**. Basically, you want the whole order record inserted in to the CosmosDB dbOrders database and that is represented by ‘Current Item’.



1. Still within the For each action, click on the **Add an action** icon again.
2. The final thing you are going to do, is take part of the order object and send it to a Service Bus queue (the Service Bus queue was created by the ARM template deployment earlier). The Service Bus queue will hold on to the information for other Azure services to retrieve (you won’t be building that part in this lab).

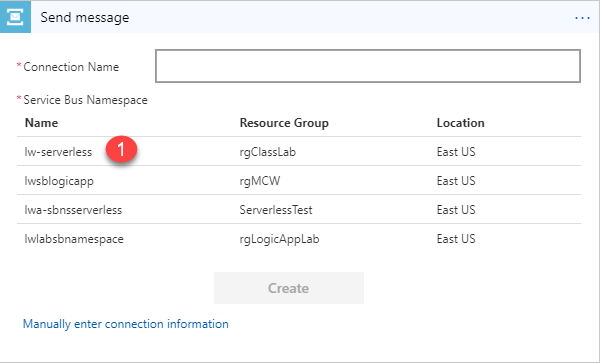
If you do not see the Service Bus icon, type Service Bus into the Search edit field. Choose the Service Bus icon.



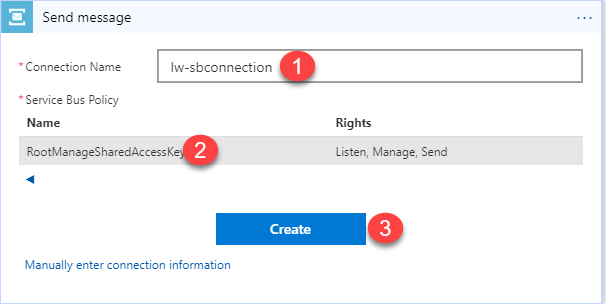
1. Scroll down until you see the Send message icon and click on the icon.



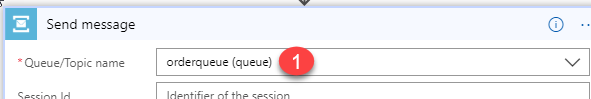
1. What you should see in the *Send message* action is at least one Service Bus namespace. The Service Bus connector will show all Service Bus namespaces in your subscription. Click on the Service Bus namespace that was created for this lab. Note that you can also see the Resource Group listed, which may help you find the right Service Bus namespace.

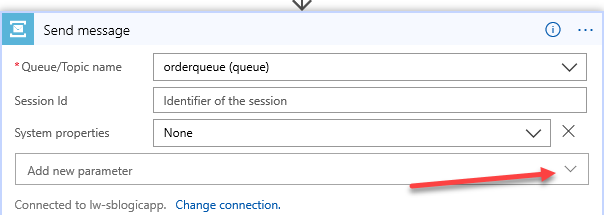


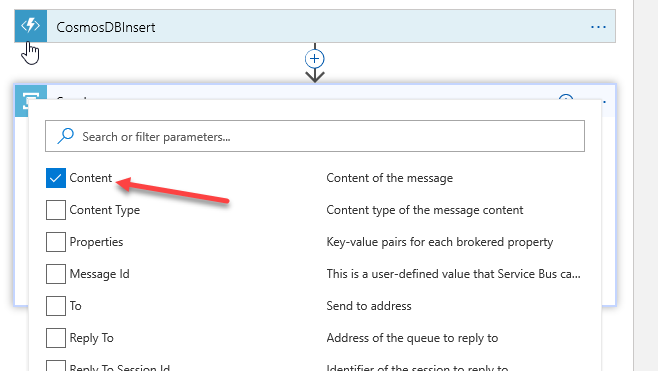
1. Provide a name for the Service Bus connection. It doesn’t matter what you name the connection, but you would be able to use the same connection later if you added other Service Bus connectors. Once you enter the connection name, click on the **RootManageSharedAccessKey** policy name, which will enable the Create button. Then click on the **Create** button.



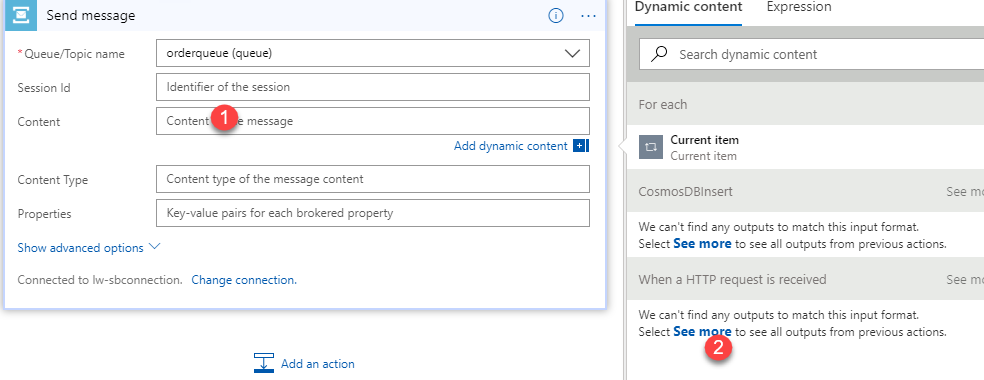
1. Now, the Send message action needs to know what queue you want to send the data to. Click on the *Queue/Topic name* drop-down and select the **orderqueue (queue)** option.



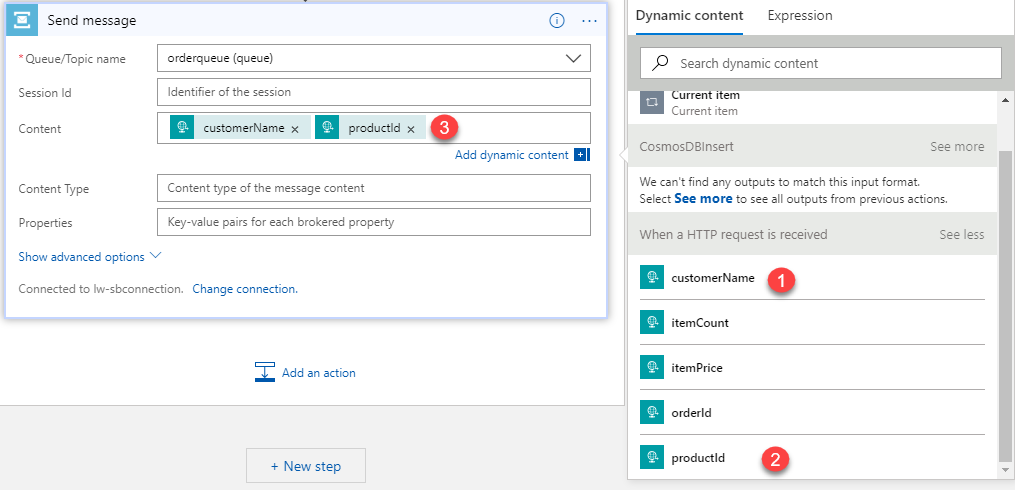
1. You will need to add the Content field to the UI of the Send message action. Click on the drop-down arrow in the **Add new parameter** field:  
     
   
2. A window will appear that has several built-in parameters listed. Check the checkbox for **Content** and then click outside of this window to close the pop-up window.



1. Click on the **Content** field and that will expand the dynamic content window over to the right. We only need to take two fields out of the order object and put those in to the Service Bus queue. To do this, click on the **See more** link under the ‘*When an Http request is received*’ section.



1. Select the **customerName** and **productId** from the list of fields in the dynamic content window.



1. Save the logic app.

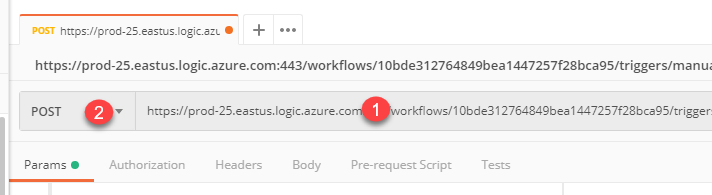
## Task 3 – Testing your Logic App

To test your logic app, you will need to use a tool that is able to send a POST command to your logic app. Although there are several free tools available, the next steps will include screenshots for using Postman <https://www.getpostman.com/>.

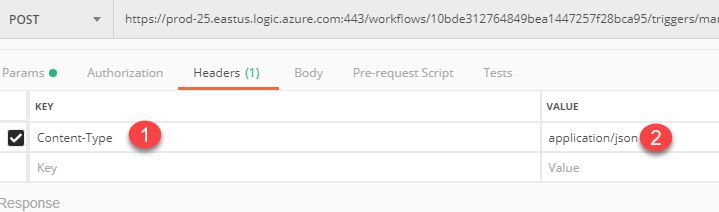
1. In the logic app designer, click on the http request trigger to expand it. Copy the URL that has been provided for the POST operation.



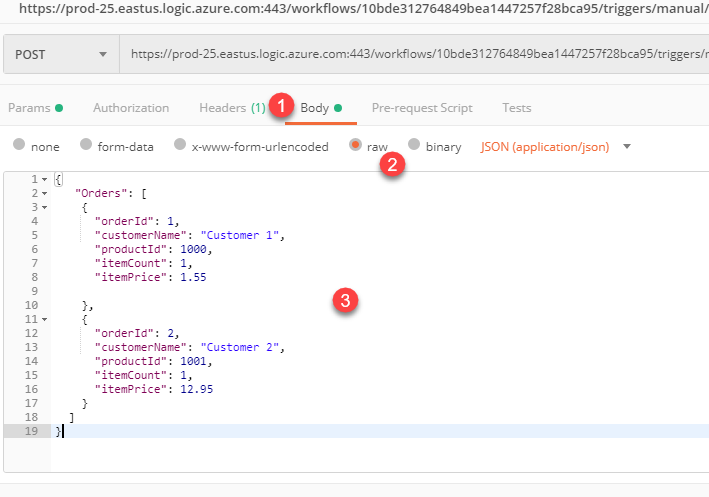
1. Open **Postman**.
2. In Postman, paste the URL in to the request field and set the method to **POST**.



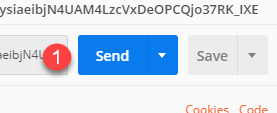
1. Click on the Headers menu item and enter in a header for **Content-Type** with **application/json**.



1. Click on the **Body** menu item and then select the ‘**raw’** option. Copy the contents of the **.\LogicApps\Labs\IntroToLogicApps\Assets\Orders.json** in to the edit field.

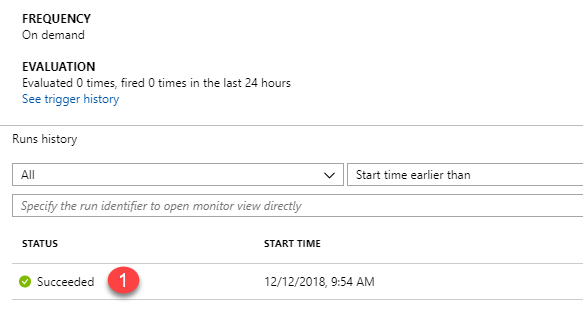


1. Click on the **Send** button.

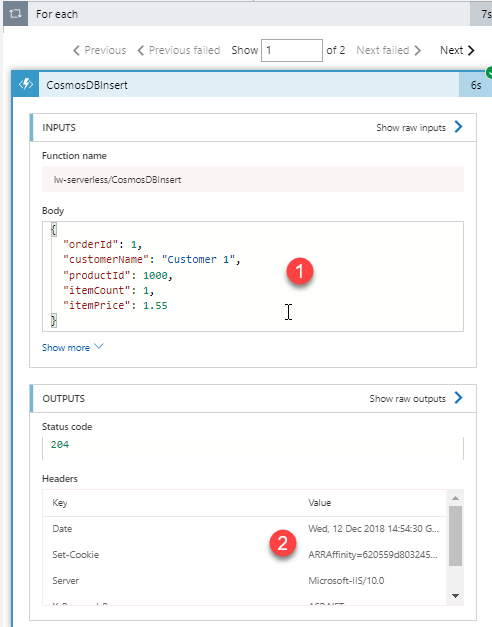


1. Assuming you get back no errors in Postman, go back to your logic app in the Azure portal. At this point, you don’t want to be in the logic app designer, so you may need to go back to the resource group and click on the name of your logic app just to get back to the logic app overview blade.

On the logic app overview page, you will see a Runs history section. What you should see is a green succeeded list item. Click on this list item.



1. You should now be in the Logic app run blade. Here, you can click on each action/trigger and see data flow results. For example, click on and expand the CosmosDBInsert action. When you do this, you will be able to see both the input and output to the Azure function.



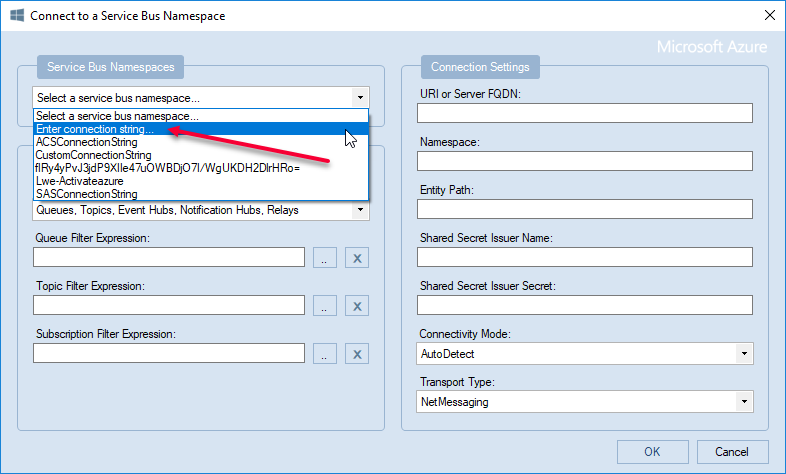
More importantly though, if you take a look at the full workflow in the run window, you can see the path that the workflow took while processing. If you had a more complex workflow, you would be able to trace the route of the data.

## Task 4 – Reviewing your message in the Service Bus queue (optional)

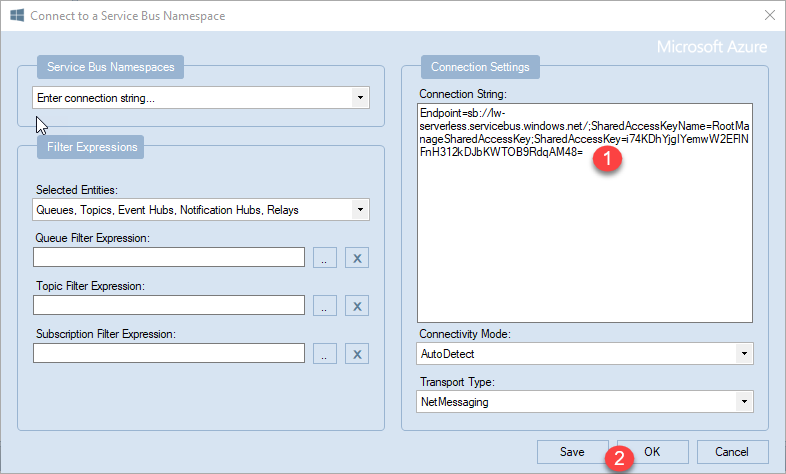
Although you can browse to your Azure Service Bus queue using the Azure portal, the portal won’t actually show you the contents of the message in the queue. To see the queue contents, you will need to use a tool like the Service Bus Explorer <https://github.com/paolosalvatori/ServiceBusExplorer>.

There are multiple ways to install/execute the Service Bus Explorer, on the Microsoft hosted lab environment, it has been installed using Chocolatey (instructions are on the Github page).

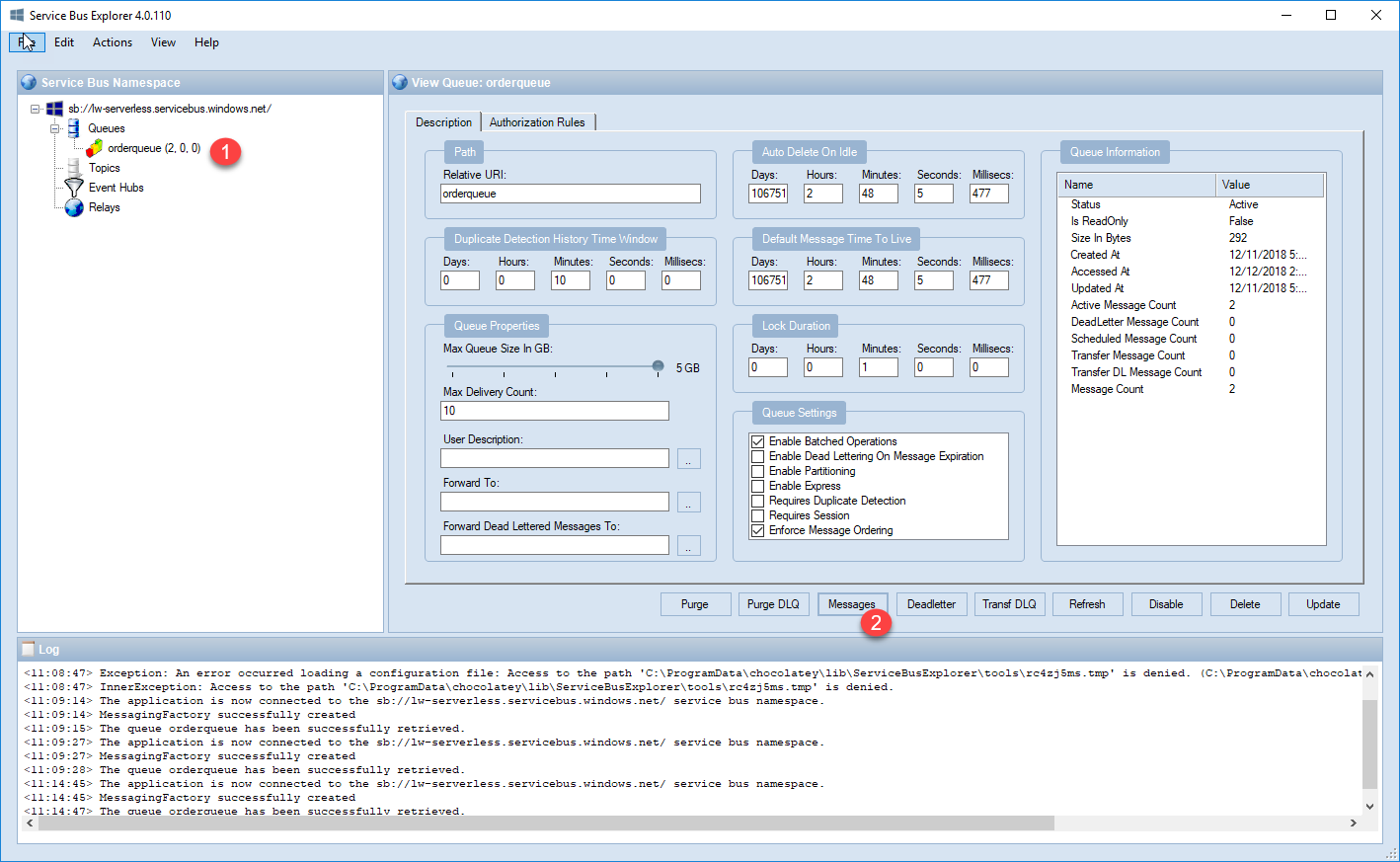
1. Assuming you have installed the Service Bus Explorer via Chocolatey, open up a command prompt window.
2. Type in ‘**ServiceBusExplorer’** and select Enter. Service Bus Explorer should appear.
3. From the Service Bus Explorer (SBE) choose **File | Connect**. Then in the *Service Bus Namespaces* dropdown, choose **Enter a Connection** string



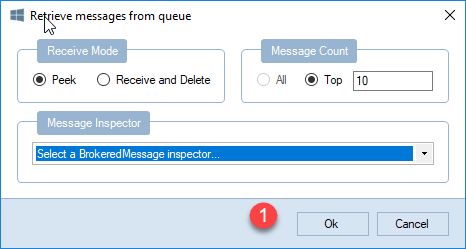
1. In the *Connection Settings* field, copy and paste the ServiceBusPublisherConnectionString Endpoint string from your **C:\serverless\configuration.txt** in to the field. Select the **Ok** button.



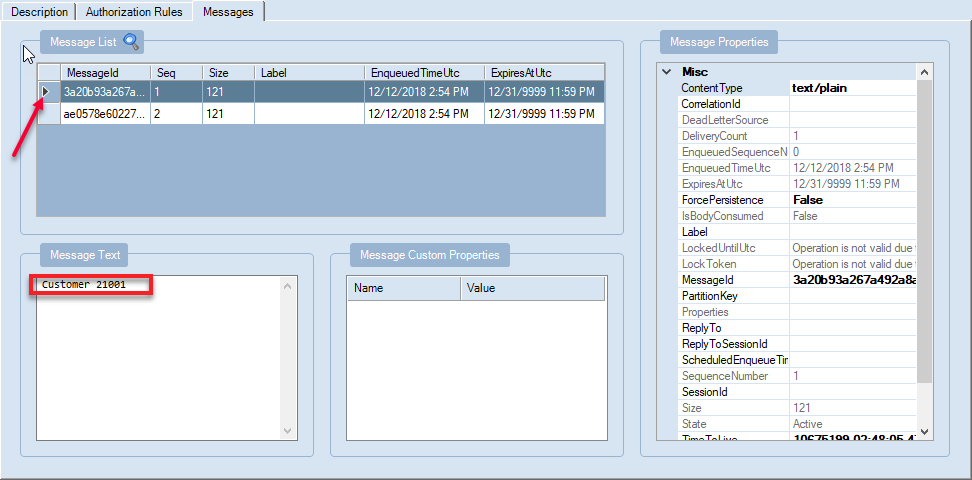
1. Click on the **orderqueue** item in the treeview and then click on the **Messages** button.



1. Leave all the settings as they are in the *Retrieve messages from queue* dialog box and select **Ok**.



1. In the messages window, you can see the content of the message in the Message Text window.



# Appendix A – How to Package up your Azure Function and deploy to Storage

This procedure assumes that you have created an Azure Function in Visual Studio and you want to package it up so that when you do your Azure resources deployment, the function can be referenced and executed.

1. Open the function app in Visual Studio. Compile the function app and function.
2. Right-click on the project and select **Pack** from the pop-up menu.
3. Note the location of the publish directory in the Visual Studio output window.
4. Use Windows Explorer to browse to the publish directory from step 3.
5. Select all files and folders in the publish directory, right-click and select Send to->Compressed (zip) folder.
6. Name the zip file **package.zip.**
7. Start Azure Storage Explorer and upload to the package.zip file to a blob storage container.
8. To create a link that includes a shared access signature, right-click on the file in Azure Storage Explorer, select Get Shared Access Signature…
9. Set the start and end time and leave other settings as they are. Click Create.
10. Select the value of the URL. This URL (with some modifications for the ‘&’ symbols) will be used in the ConfigureDeployment.ps1 file for the lab.