

Lab 3: Event Grid

Hands-on Lab

Released:

Terms of Use

© 2018 Microsoft Corporation. All rights reserved.

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, 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.

For more information, see **Microsoft Copyright Permissions** at [**http://www.microsoft.com/permission**](http://www.microsoft.com/permission)

Microsoft may have patents, patent applications, trademarks, 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.

The Microsoft company name and Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

**This document reflects current views and assumptions as of the date of development and is subject to change.  Actual and future results and trends may differ materially from any forward-looking statements.  Microsoft assumes no responsibility for errors or omissions in the materials.**

**THIS DOCUMENT IS FOR INFORMATIONAL AND TRAINING PURPOSES ONLY AND IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.**

Table of Contents

[Lab Overview 1](#_Toc508661967)

[Exercise 1: Creating Event Grid Topic 2](#_Toc508661968)

[Exercise 2: Updating Azure SQL with Order 8](#_Toc508661969)

[Exercise 3: Creating Function to subscribe to Event Grid 12](#_Toc508661970)

# Lab Overview

###### Abstract

In this lab, we will be working with Azure Event Grid to connect the order submission process for the eShopOnWeb eCommerce website to the backend processing systems. The Event Grid will provide the fan out event of the order being received to send it to multiple subscribers. In this case one of the subscribers will be a Logic App that will have a workflow to update a SQL Server order system and then send an email to the customer. The second subscriber to the event will be an Azure Function that will update a Cosmos DB representing the customer’s order history.

In this scenario the Event Grid provides a means to immediately update multiple subscribers about the ordering event. Also, those subscribers can apply some filtering when creating the subscriptions as needed.

###### Learning Objectives

After completing the exercises in this lab, you will be able to:

* Create an Event Grid Topic so that custom event can be sent through the Event Grid.
* Create multiple subscription for an Event Grid using both a Logic App and an Azure Function with Filtering.
* Connect the Serverless components to additional backend services such as Azure SQL and Cosmos DB.

**Estimated time to complete this lab: *30* minutes**

Be sure to save all the components used in this Lab, they will be used in a future lab.

# Exercise : Creating Event Grid Topic

#### Scenario

In this exercise, you will be creating the Event Grid Topic a sample .NET application to POST event to the Topic and an initial Logic App to create an Event Subscription.

After completing this exercise, you will understand:

* An Event Grid Topic and how to submit custom events to that topic.
* How to subscribe to an Event Grid from a Logic App

#### Prerequisites:

To complete this lab, you should have a basic understanding of Azure Portal, Events and C#.

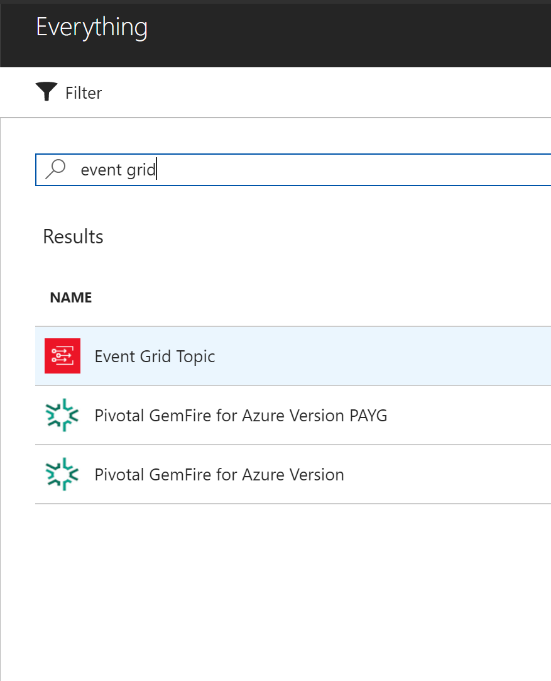
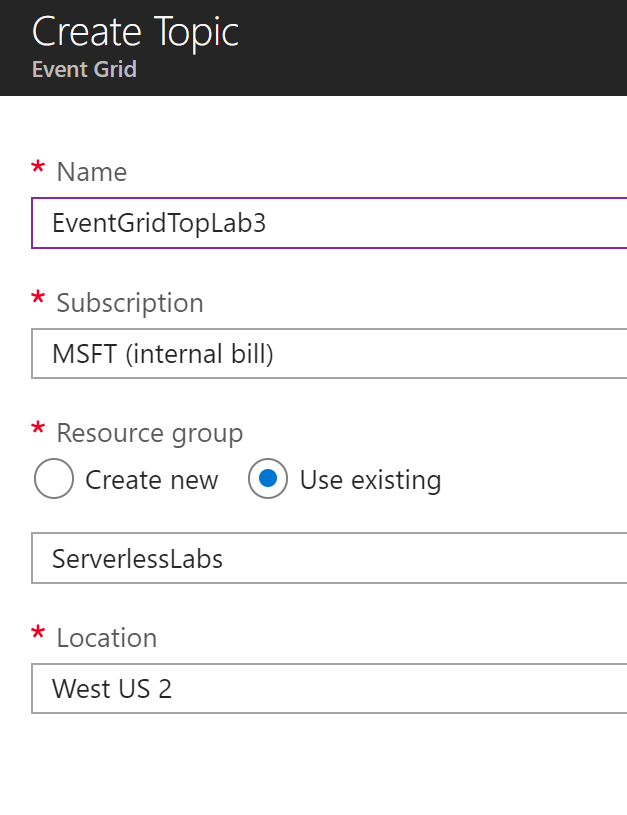
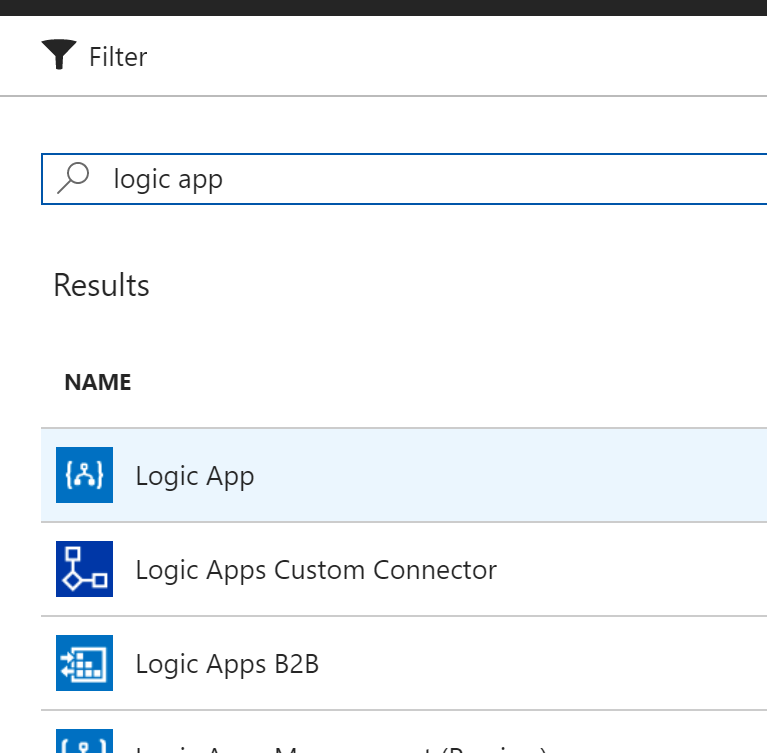
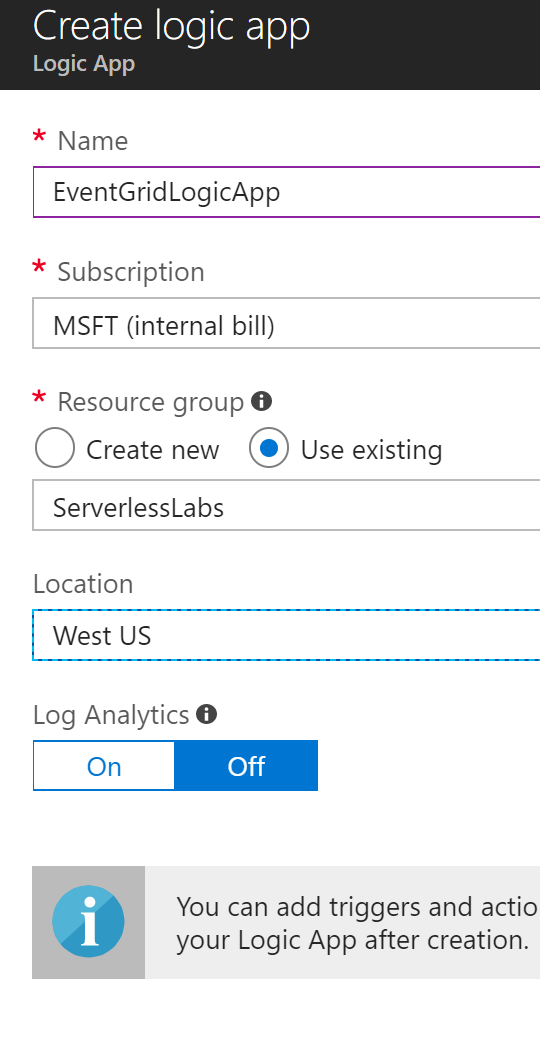
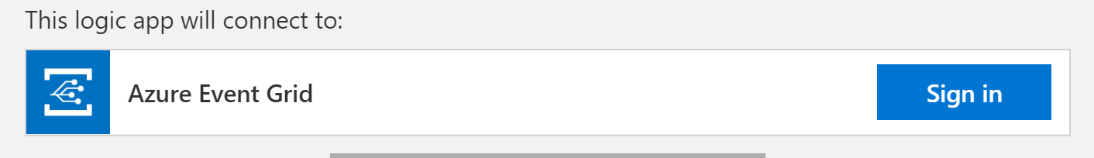
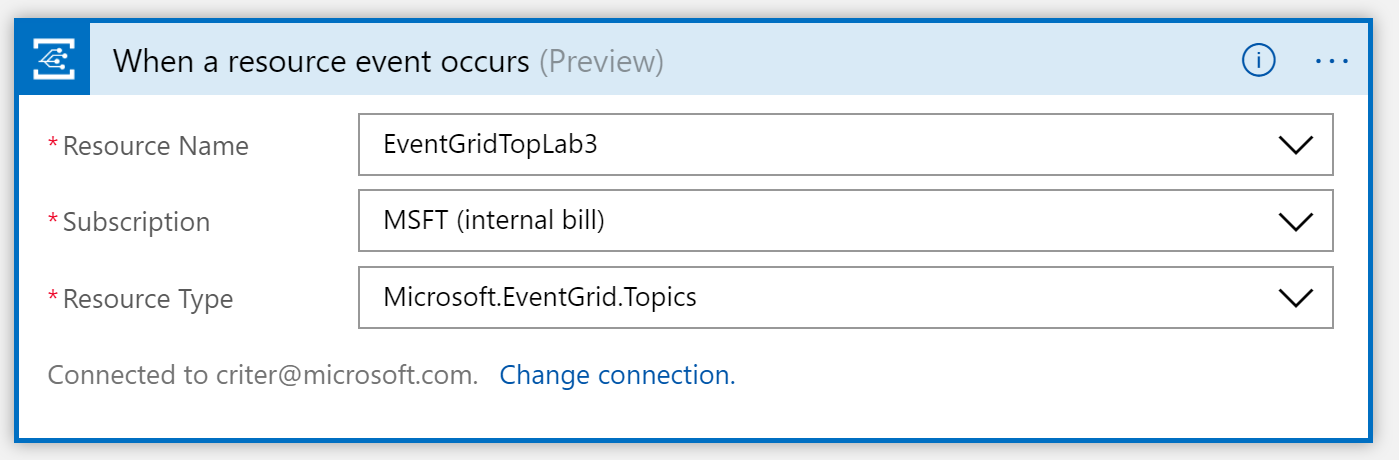
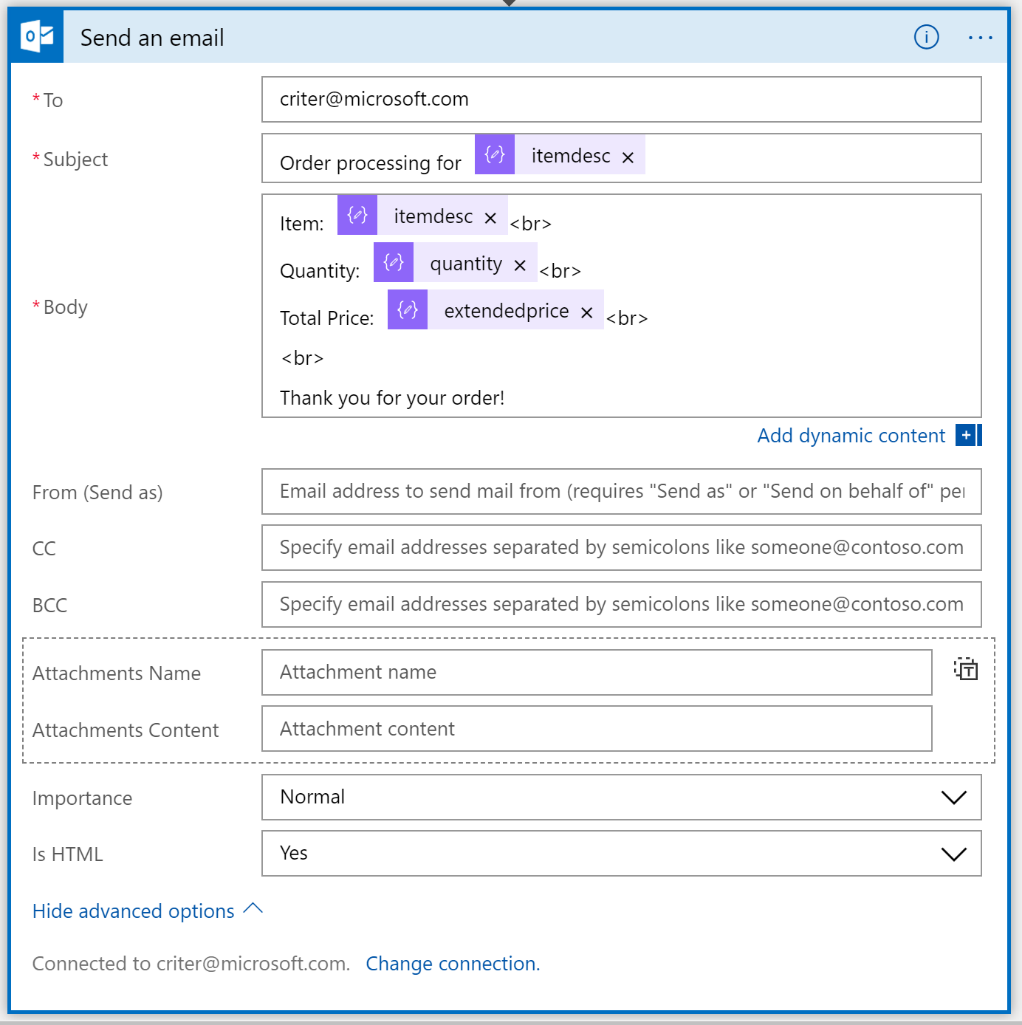
#### Install and configure:

You will need:

* A desktop computer (host) running Windows 10
* An active Azure subscription (MSDN credits will suffice)
* An active Internet connection
* Visual Studio 2017 (or earlier version is fine), this is only needed if you would like to review the code for the app that sends messages to the Event Grid

#### Create Event Grid Topic

Be sure to save this Event Grid since it will be used in a future lab.

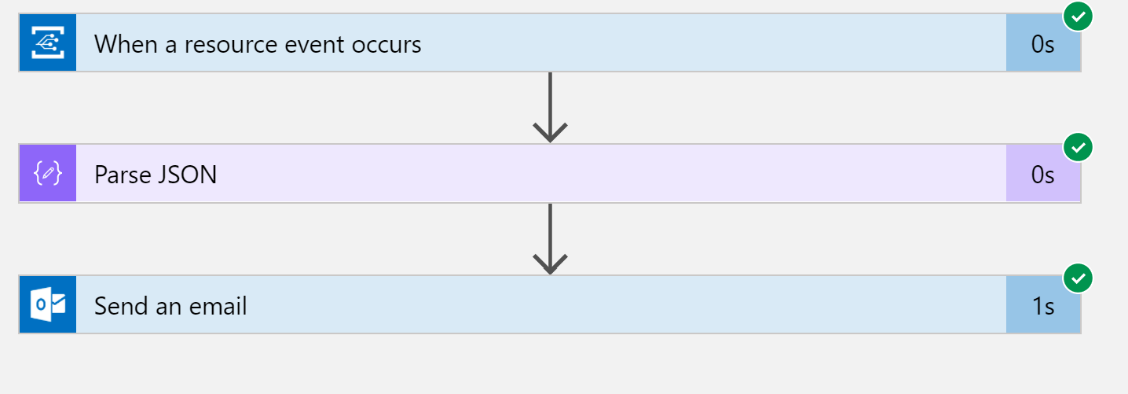
1. Navigate to the Azure Management Portal (http://portal.azure.com) and search for: **Event** **Grid**  
   
2. After locating **Event Grid Topic** in the search results and clicking the row, click the **Create** button to open up the Create Topic blade.
3. Fill out the Create Topic blade. **Name** the topic, associate it to your **Subscription** and use an existing **Resource Group** for ease of organization and place in whatever Location is **appropriate**. Click the **Create** button to create the resource.  
   
4. Click the **Create a resource** link in the upper left corner and search for **Logic App** and select the Logic App row and click the **Create** button  
   
5. On the Create logic app blade enter the values to create the Logic App. Make a unique descriptive **Name** and associate it to you **Subscription**. Use an existing **Resource Group** for ease of organization and place in whatever **Location** is appropriate. Finally click **Create**.  
   
6. Navigate to the Logic App and click the ‘Start with a common trigger’ **When a Event Grid event occurs** to start the designer.
7. Click the **Sign in** button to authenticate with your credentials to gain access to the list of Event Grid Topics. After sign in is complete, click **Continue**.  
   
8. Now you will need to fill out the event connector. First the **Subscription** attribute is first where you need to select the subscription where you created the Event Grid Topic in Step 3 above. Next select **Microsoft.EventGrid.Topics** for the **Resource Type** since this will determine the items to show in the **Resource Name** list. In the Resource Name list you should find the name of the Event Grid Topic that you created in Step 3. It is possible to apply some filtering conditions with the Prefix and Suffix filter but for this one we will leave it open.  
   
9. Click **New Step** to create a new action and then select **Add an action.**  Search for parse and select **Data Operations – Parse JSON**.
10. For **Content** select **Body** from the event and for the schema click the **Use sample payload to generate schema** link and in the file from this gist and click **Done**.  
    <https://gist.github.com/criter/e430aad52c9b69c570c08de39e41937b>
11. Click **New Step** to create a new action and then select **Add an action.**  Search for **outlook** and select **Office 365 Outlook – Send an email**. Likely you should already be connected to your Office 365 account but if not just sign-in.
12. Next is to fill out the action.
    * **To -**  this should be your address for testing purposes
    * **Subject –** As the screenshot shows it should indicate what was ordered by the customer.
    * **Body –** Format this field as if it was an HTML field using the information parsed by the previous action from the JSON body field from the event. Using the advanced field we will indicate that the body is being defined as HTML.
    * Click the **Show advanced options** to show the **Importance** and **Is HTML** fields. Set Importance to **Normal** and Is HTML to **Yes**.  
      
13. At this point we can save the Logic App and we can work on generating an Event to post to the Event Grid Topic for the Logic App to respond to. So click the **Save** button in the designer.

#### Create the App to POST to the Event Grid Topic

There are a few ways to have an Event Grid create an event. There are already a few services that create Event Grid events such as Blob Storage, Subscriptions, Resource Groups, Event Hubs, etc. One can also write Custom Topics and then write an application that will POST to the REST API of Event Grid to submit an event. That is what this application will do. There is also a Logic App connector that will POST to an Event Grid Topic but for the sake showing the code I thought this would be better.

1. You’ll want to download the source code and utility that can POST event to the Event Grid Topic. You can find that at [https://github.com/criter/PostToEventGridTopic/](https://github.com/criter/PostToEventGridTopic/tree/master/SubmitToEventGridTopic) or download the [tree as a zip file](https://github.com/criter/PostToEventGridTopic/archive/master.zip). Either way you can open it up with Visual Studio on your machine after you have downloaded it from github. If you do download the zip file make sure you Unblock the file by right clicking it in Explorer, selecting Properties and checking the Unblock checkbox at the bottom.
2. You can see via the README.txt file that there is a precompiled version of the program in the binn folder. Open a command prompt and navigate to the binn folder.
3. You’ll want to execute the **SubmitToEventGridTopic.exe**. As noted in the README.txt file this takes 3 parameters.

* Event Grid Endpoint URL – this can be found by going back into the properties of the Event Grid Topic that was created previously, click the **Overview** property and look for the **Topic EndPoint**, copy this value.
* SAS Key – this can be found by clicking the **Access keys** property on the same Event Grid Topic page and then copying either **Key 1** or Key 2.
* Repeat – this value is the number of times you want the program to repeat submitting the message.

1. Navigate back to your Logic App and put it into **Run** mode so that you can see if it executes successful and see the inputs and outputs and if there is an error.
2. Then execute the **SubmitToEventGridTopic.exe** with the parameters as noted.
3. You should receive an email and see success on your Logic App, troubleshoot if you have an error.  
   

# Exercise 2: Updating Azure SQL with Order

#### Scenario

In this exercise, you will be using the order from the Logic App to also update an Azure SQL database in addition to sending the email confirmation to the user.

After completing this exercise, you will understand:

* How to connect a Logic App to an Azure SQL database

#### Prerequisites:

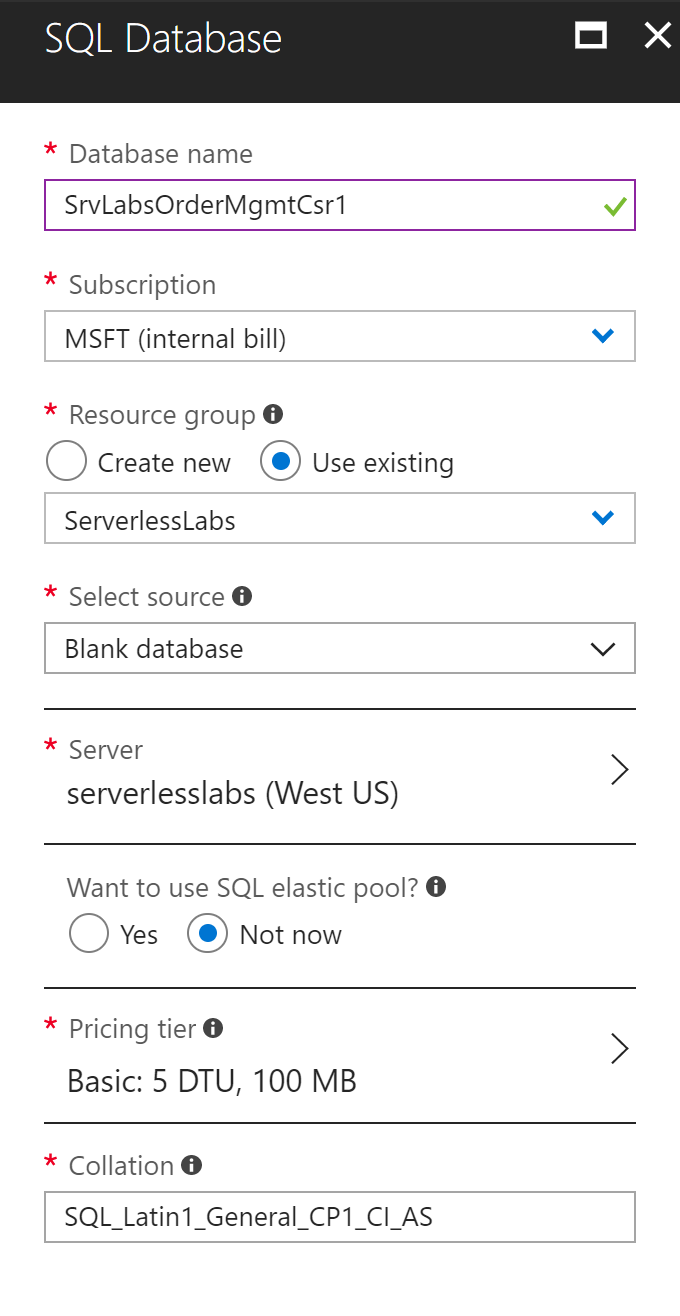
To complete this lab, you should have a basic understanding of Azure Portal and SQL table structures.

#### Install and configure:

You will need:

* A desktop computer (host) running Windows 10
* An active Azure subscription (MSDN credits will suffice)
* An active Internet connection

#### Create Azure SQL database

1. Navigate to the Azure Management Portal (http://portal.azure.com) and click **Create a resource** and search for **SQL Database** and click **Create**.
2. With the SQL Database blade open enter the fields to create the database. Follow the suggestions in the screenshot. Once complete, click **Create**. When creating a new server remember the **Username** and **Password** since you’ll need that to connect to the database in the later steps. If you use an existing database server already in your subscription, be sure you know the Username and Password.  
   
3. After the database provisions, navigate to the database and use the Query Editor and connect to the database using the **Login** button. In the Query window put the follow create table statement and click the **Run** button.

CREATE TABLE [dbo].[Orders](

[customerid] [uniqueidentifier] NOT NULL,

[orderid] [nvarchar](50) NOT NULL,

[itemid] [bigint] NOT NULL,

[itemdesc] [nvarchar](200) NOT NULL,

[quantity] [int] NOT NULL,

[itemprice] [money] NOT NULL,

[extendedprice] [money] NOT NULL,

CONSTRAINT [PK\_Table\_1] PRIMARY KEY CLUSTERED (

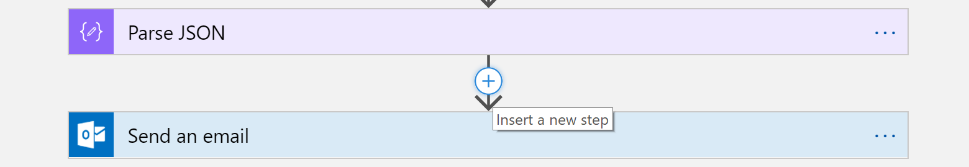
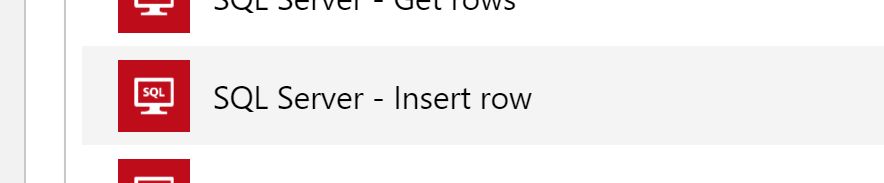
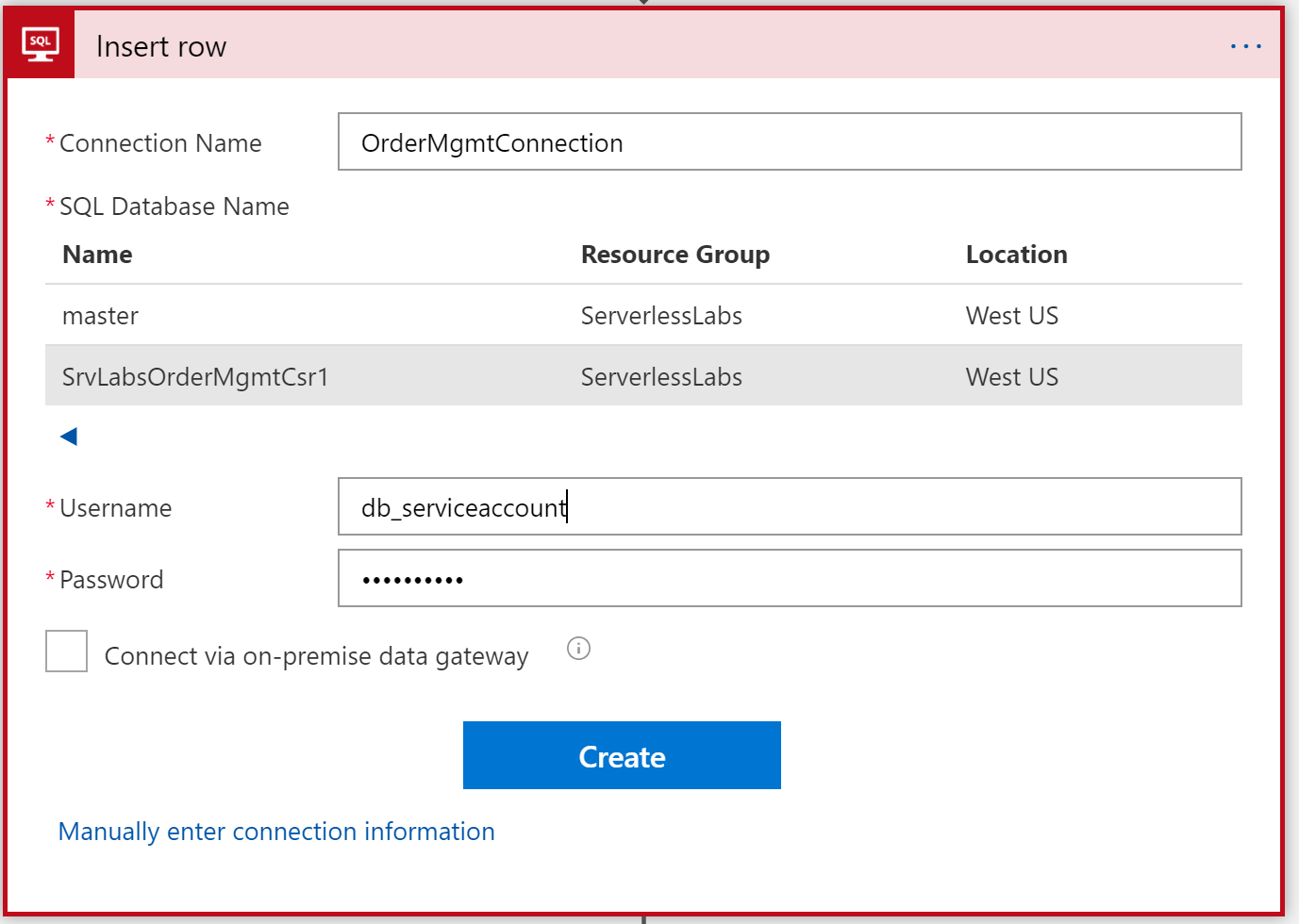
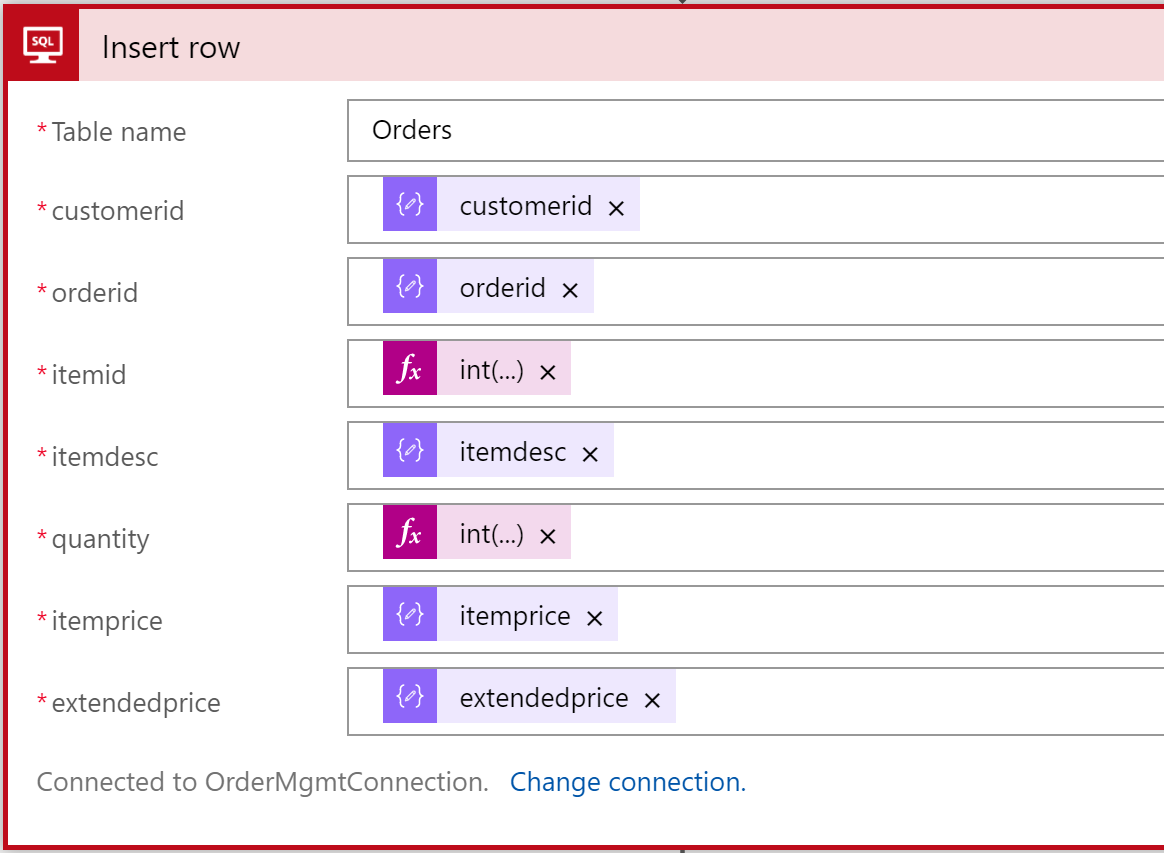
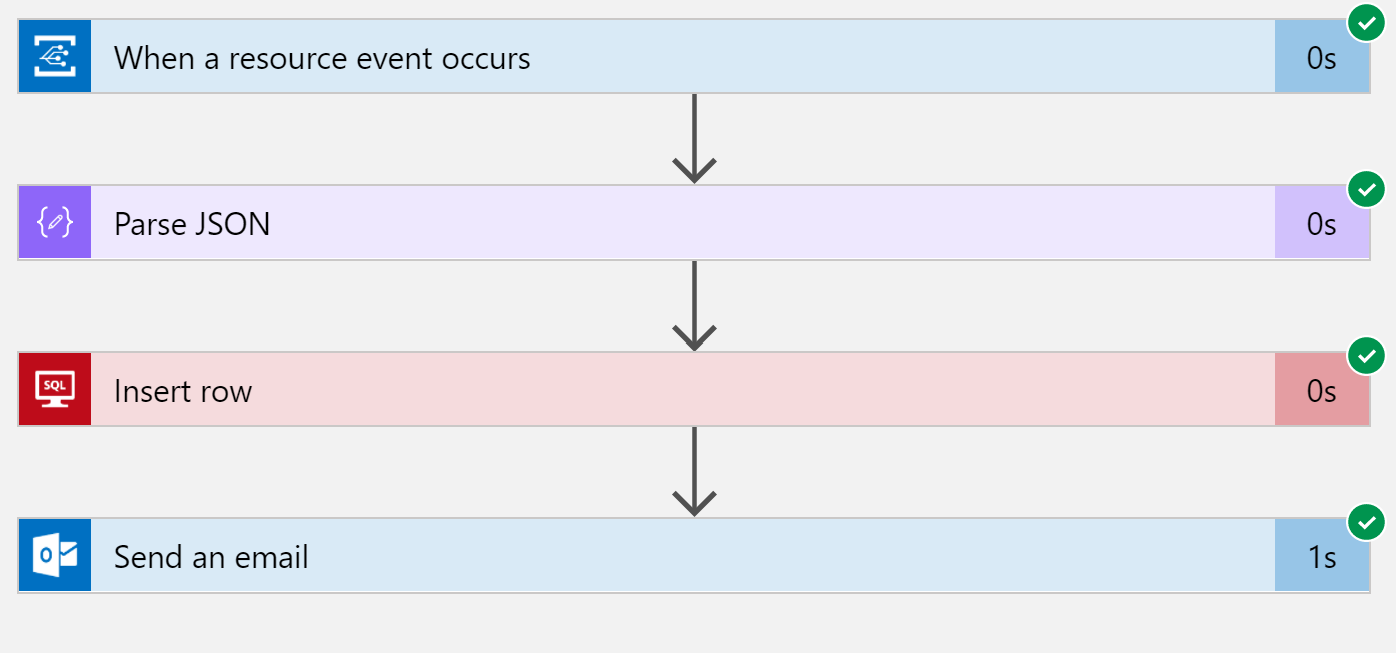
[customerid] ASC

) )

1. This will be the table that will be populated by the Logic App.

#### Update Logic App to connect to Azure SQL

Be sure to save this Logic App since it will be used in a future lab.

1. Navigate back to the Logic App designer with the Logic App from Exercise 1.
2. Insert an action before the email by hovering over the line between the boxes and clicking the + button and then clicking **Add an action**.  
   
3. Search for **SQL** and select **SOL Server – Insert row**.  
   
4. Then select the **Server** and **Database** that you created. Enter a unique **Connection Name** and **Username** and **Password** that can connect to the database with permissions to insert a record.  
   
5. Next you’ll be prompted for a **Table name**. This will then read the table structure and display all of the fields from the table to be populated. Work through each of the fields to populate them from the Parse JSON activity. Some of these will need to have a converson applied to them since there is a type change from the Javascript JSON type and the SQL type.  
   
6. **Save** your Logic App. Once complete you can test the integration by running the **SubmitToEventGrid.exe** to validate that the Logic App does run successful and sends the email and adds the record to the database. Be sure to leave the Logic App in read mode to see if it successful ran or if not where it fails and the inputs and outputs.  
   

Be sure to save this Logic App since it will be used in a future lab.

# Exercise 3: Creating Function to subscribe to Event Grid

#### Scenario

In this exercise, you will be having an Azure Function subscribe to the Event Grid and update a Cosmos DB with the order information.

After completing this exercise, you will understand:

* How to create a Function that can respond to an Event Grid event
* Output an Cosmos DB document from a Function

#### Prerequisites:

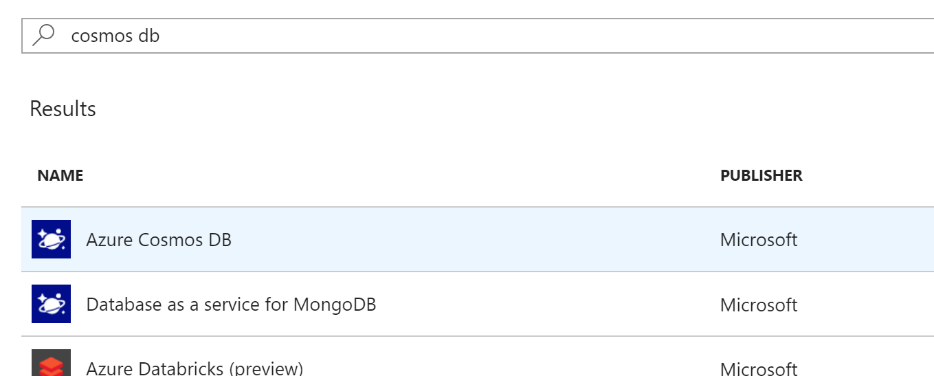
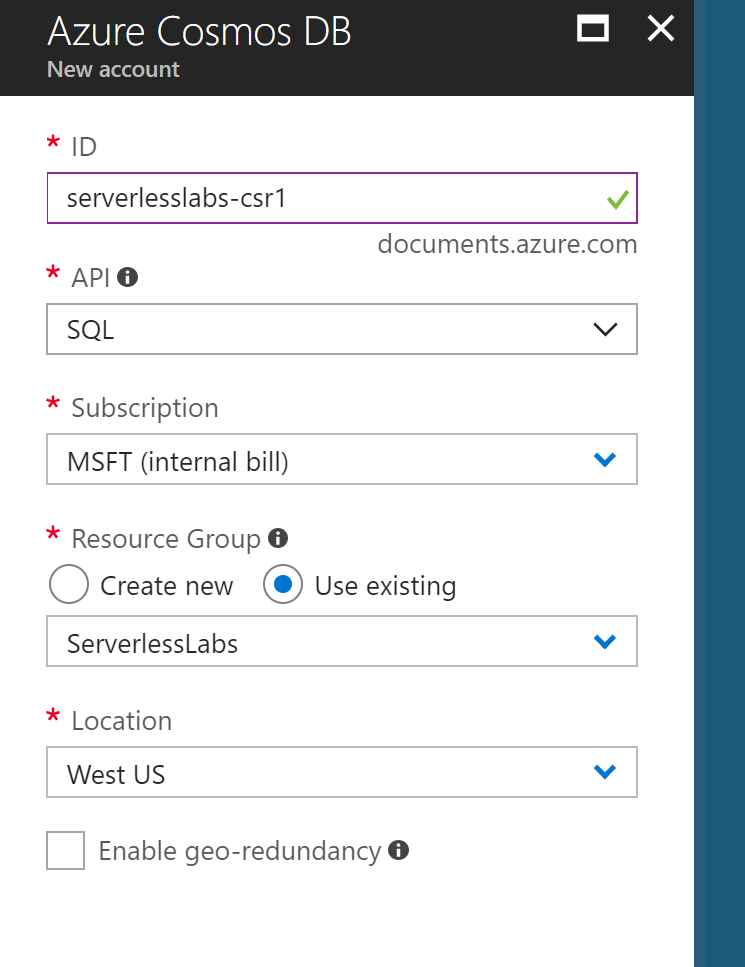
To complete this lab, you should have a basic understanding of Azure Portal, Cosmos DB

#### Install and configure:

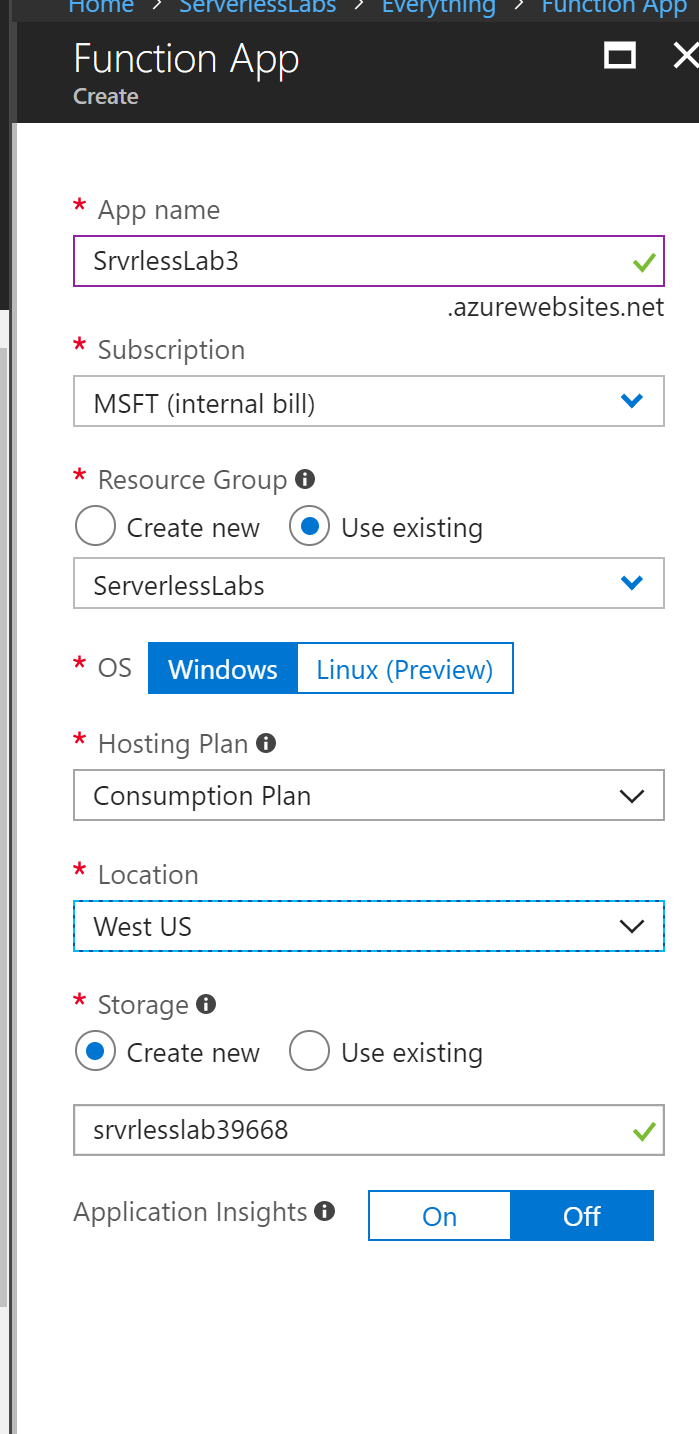
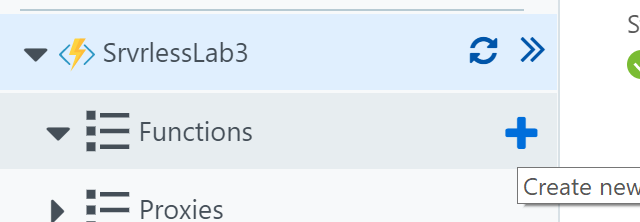
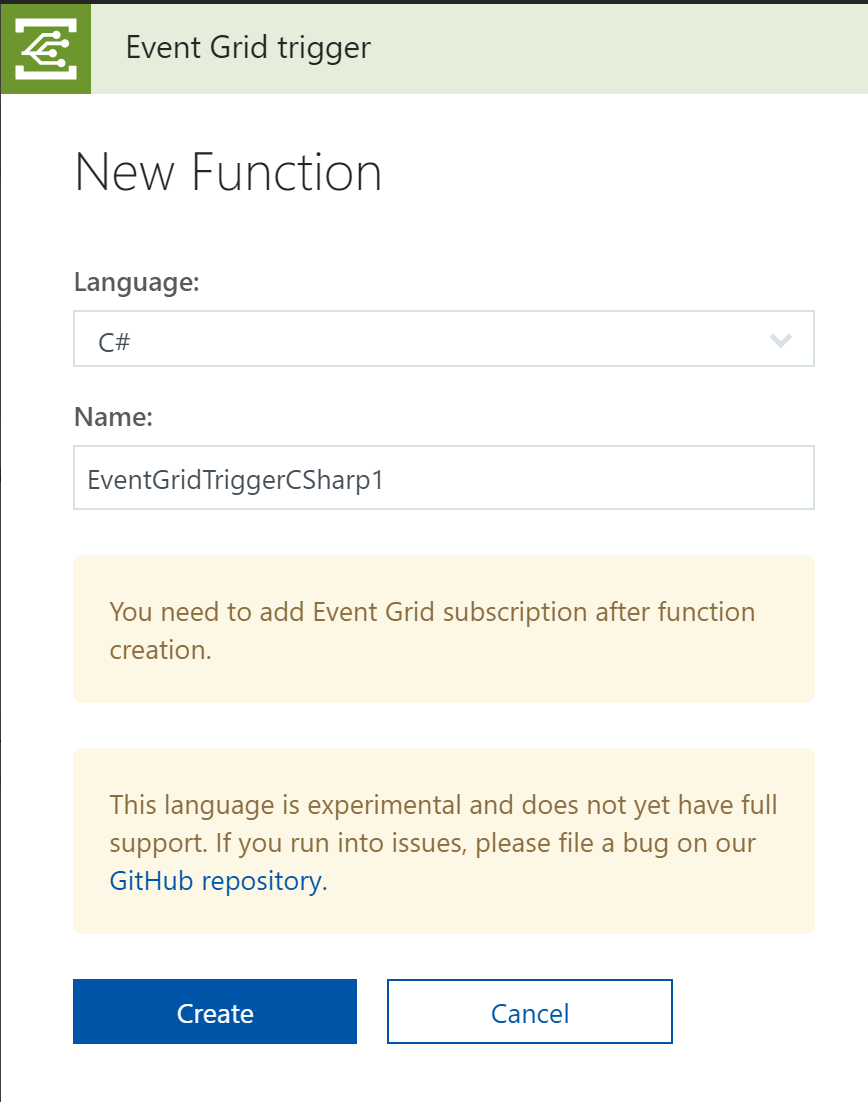
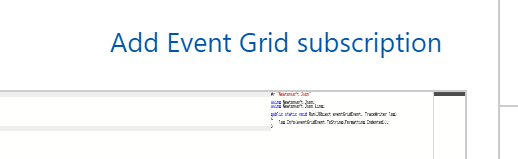
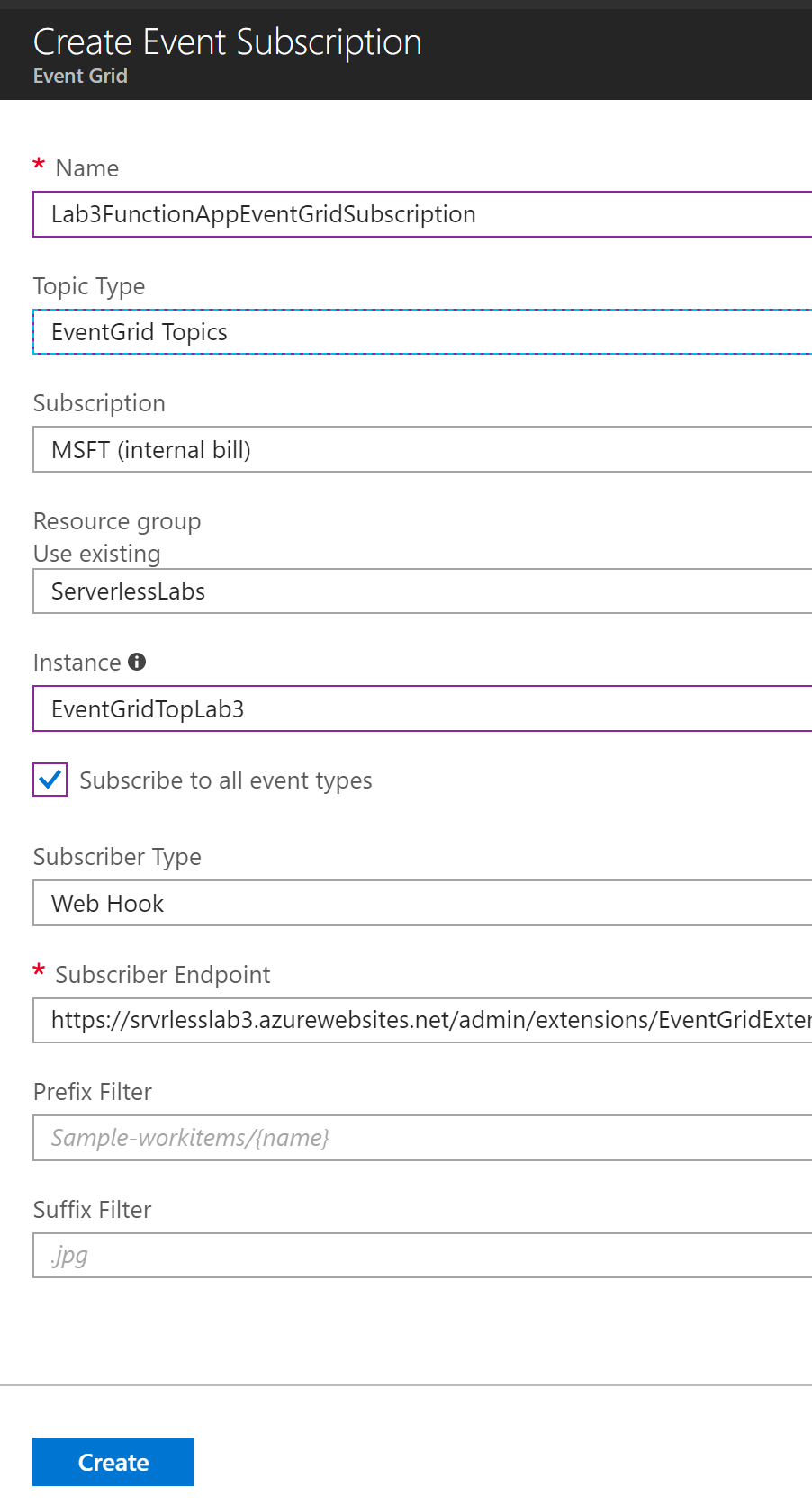
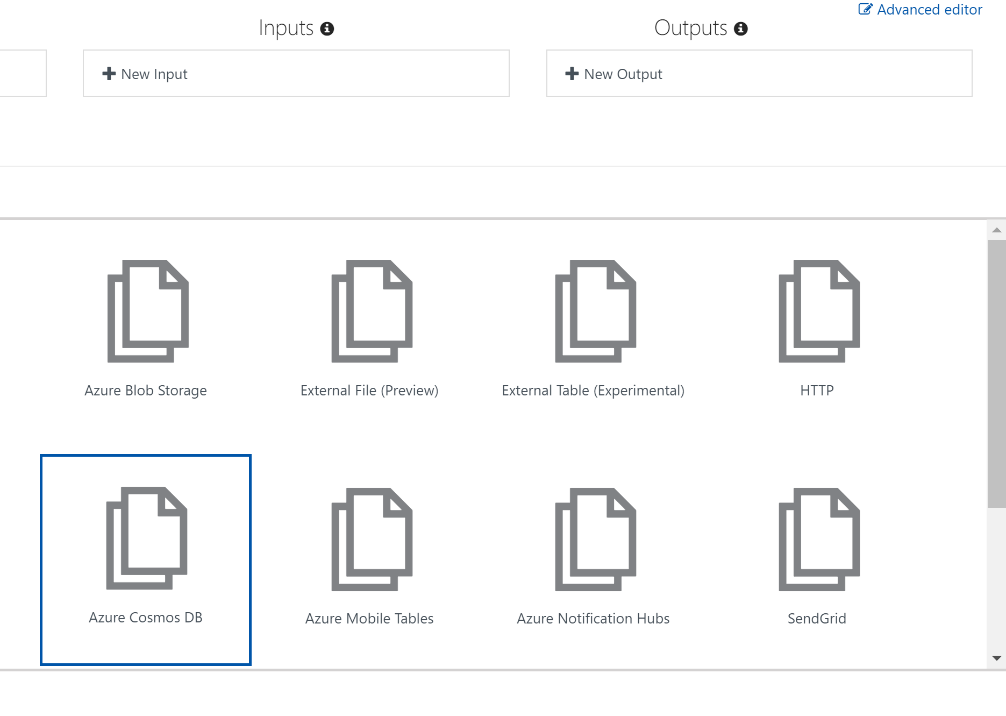
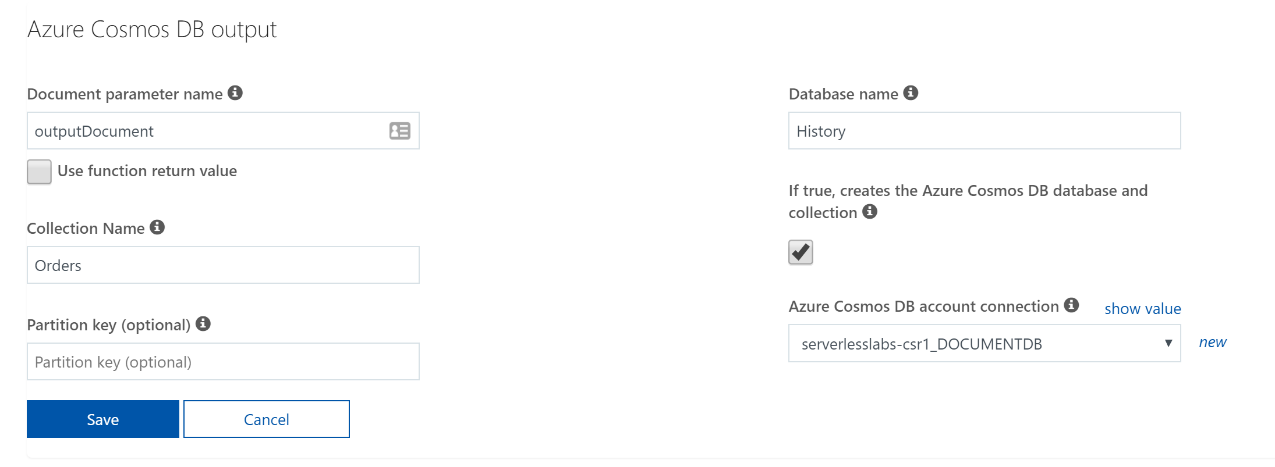
You will need:

* A desktop computer (host) running Windows 10
* An active Azure subscription (MSDN credits will suffice)
* An active Internet connection
* Not required by may be used instead of Data Explore in Azure Portal, download [Azure Storage Explorer](https://azure.microsoft.com/en-us/features/storage-explorer/)

#### Creating Cosmos DB

1. If you still have you Cosmos DB available from earlier labs then you can skip this section. If not then complete this section to create the Cosmos DB. In the next section you’ll create the Database and Collection.
2. Navigate to the Azure Management Portal (<http://portal.azure.com>) and search for: **Cosmos DB** and click on the row.  
   
3. Then click the **Create** button. With the blade open, enter the value to create it. The ID should be unique. **API** should be **SQL** since we aren’t trying to be compatible with other protocols. Use an existing **Resource Group** if you are trying to keep all of your resources together and whichever **Location** is appropriate for you. Then click **Create**.  
   
4. While that is creating move on to the next section and it should be finished by the time you need to create the database and collection.

#### Creating Function

1. Use the **Create a resource** link in the upper left corner of the portal, and search for **Function App**. Select the **Function App** row and click **Create** to open the blade.
2. Create a new Function App, similar to previously. Use an existing **Subscription Plan** for ease of maintenance.  
   
3. Once created navigate to the Function App and create a new function using the **Plus sign** next to the Functions heading.  
   
4. Click the link that says **Custom function** to bring up the list templates. Scroll down to find the one that says **Event Grid Trigger**. Select **C#** for the language. Leave the defaults and click **Create**.  
   
5. You have a default function and can do a test run to validate the scaffolding.
6. Next click the Add Event Grid subscription in the upper right corner  
   
7. Fill it out the Create Event Subscription blade like below. This is creating a WebHook type subscription against Event Grid that will call the Function. Once completed, click **Create**.  
   
8. Wait until the subscription is provisioned. With the test window still open, go back and run the console SubmitToEventGrid.exe with a Repeat value of 1, this will submit the event and the Function will execute and log the output of the JSON body in the log window. You can now copy that JSON text and put it into the Request body field in the upper right Test window for future executions of the Function to test as we integrate it with Cosmos DB.
9. Now click on the **Integrate** link under the Function name in the outline on the left hand side. Then click on the **+ New Output** under Outputs. Finally, select **Azure Cosmos DB** and click **Select**.  
   
10. Now fill out the output settings. Database name can be History, Collection Name can be Orders, **check** the box to provision the database and collection, leave partition key blank for simplicity, and connect it to the Cosmos DB you should still have from previous labs. Then click **Save**.  
    
11. Click on **EventGridTriggerCSharp1** to go back to the code. You will need to add the output parameter for the document to be updated into Cosmos DB, so the function declaration line should look like this.

public static void Run(JObject eventGridEvent, out object outputDocument, TraceWriter log)

1. Next you’ll need to convert the eventGridEvent variable into a dynamic type to make it easier to work with.

dynamic orderEvent = eventGridEvent;

1. Finally you’ll need to populate the outputDocument parameter.

outputDocument = new {  
 orderid = orderEvent.data.orderid,  
 itemid = orderEvent.data.itemid,  
 itemdesc = orderEvent.data.itemdesc,  
 quantity = orderEvent.data.quantity,  
 itemprice = orderEvent.data.itemprice,  
 extendedprice = orderEvent.data.extendedprice,  
 customerid = orderEvent.data.customerid,  
 orderdate = orderEvent.eventTime  
 };

1. You can execute a test run of the function to validate that it compiles and executes. Check the Cosmos DB to validate it has the order document. You can navigate to the Cosmos DB in the Azure Portal and use the Data Explorer property or use the [Azure Storage Explorer](https://azure.microsoft.com/en-us/features/storage-explorer/) which has a viewer for Cosmos DB.
2. With that test complete, you can run the command line SubmitToEventGrid.exe to test the whole pipeline with both the Logic App and the Function App running.