



Microsoft Cloud Workshop

Serverless architecture

Hands-on lab step-by-step

January 2018

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example 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.

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 names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2018 Microsoft Corporation. All rights reserved.

Microsoft and the trademarks listed at <https://www.microsoft.com/en-us/legal/intellectualproperty/Trademarks/Usage/General.aspx> are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

Contents

Serverless architecture hands-on lab step-by-step.....	1
Abstract and learning objectives.....	1
Overview	1
Solution architecture.....	2
Requirements	3
Before the hands-on lab.....	4
Task 1: Set up a development environment.....	4
Task 2: Disable IE Enhanced Security.....	4
Task 3: Validate connectivity to Azure	5
Task 4: Download and explore the TollBooth starter solution	5
Task 5: Create a new Azure Resource group.....	6
Exercise 1: Azure data, storage, and serverless environment setup	7
Help references.....	7
Task 1: Provision the storage account.....	7
Task 2: Provision the Function Apps.....	10
Task 3: Provision the Event Grid topic.....	11
Task 4: Provision the Azure Cosmos DB account	14
Task 5: Provision the Computer Vision API service	18
Exercise 2: Develop and publish the photo processing and data export functions.....	21
Help references.....	21
Task 1: Configure application settings.....	21
Task 2: Finish the ProcessImage function	23
Task 3: Publish the Function App from Visual Studio	25
Exercise 3: Create functions in the portal	29
Help references.....	29
Task 1: Create function to save license plate data to Azure Cosmos DB.....	29
Task 2: Add an Event Grid subscription to the SavePlateData function.....	32
Task 3: Add an Azure Cosmos DB output to the SavePlateData function.....	34
Task 4: Create function to save manual verification info to Azure Cosmos DB	35
Task 5: Add an Event Grid subscription to the QueuePlateForManualCheckup function	38
Task 6: Add an Azure Cosmos DB output to the QueuePlateForManualCheckup function.....	39
Task 7: Configure custom event types for the new Event Grid subscriptions	41
Exercise 4: Monitor your functions with Application Insights	45
Help references.....	45
Task 1: Provision an Application Insights instance.....	45
Task 2: Enable Application Insights integration in your Function Apps.....	46
Task 3: Use the Live Metrics Stream to monitor functions in real time.....	48

Task 4: Observe your functions dynamically scaling when resource-constrained.....	53
Exercise 5: Explore your data in Azure Cosmos DB.....	57
Help references.....	57
Task 1: Use the Azure Cosmos DB Data Explorer	57
Exercise 6: Create the data export workflow.....	60
Help references.....	60
Task 1: Create the Logic App	60
Exercise 7: Configure continuous deployment for your Function App.....	69
Help references.....	69
Task 1: Create a GitHub repository	69
Task 2: Add GitHub repository to your Visual Studio solution	70
Task 3: Configure your Function App to use your GitHub repository for continuous deployment.....	73
Task 4: Finish your ExportLicensePlates function code and push changes to GitHub to trigger deployment	76
Exercise 8: Rerun the workflow and verify data export.....	80
Task 1: Run the Logic App.....	80
Task 2: View the exported CSV file.....	81
After the hands-on lab	84
Task 1: Delete the Resource group in which you placed your Azure resources.....	84

Serverless architecture hands-on lab step-by-step

Abstract and learning objectives

Setup and configure a serverless architecture within Azure using a combination of Azure Functions, Logic Apps, Event Grid, Cosmos DB, and Azure Storage. The focus is on removing server management from the equation, breaking down the solution into smaller components that are individually scalable, and allowing the customer to only pay for what they use.

Learning Objectives:

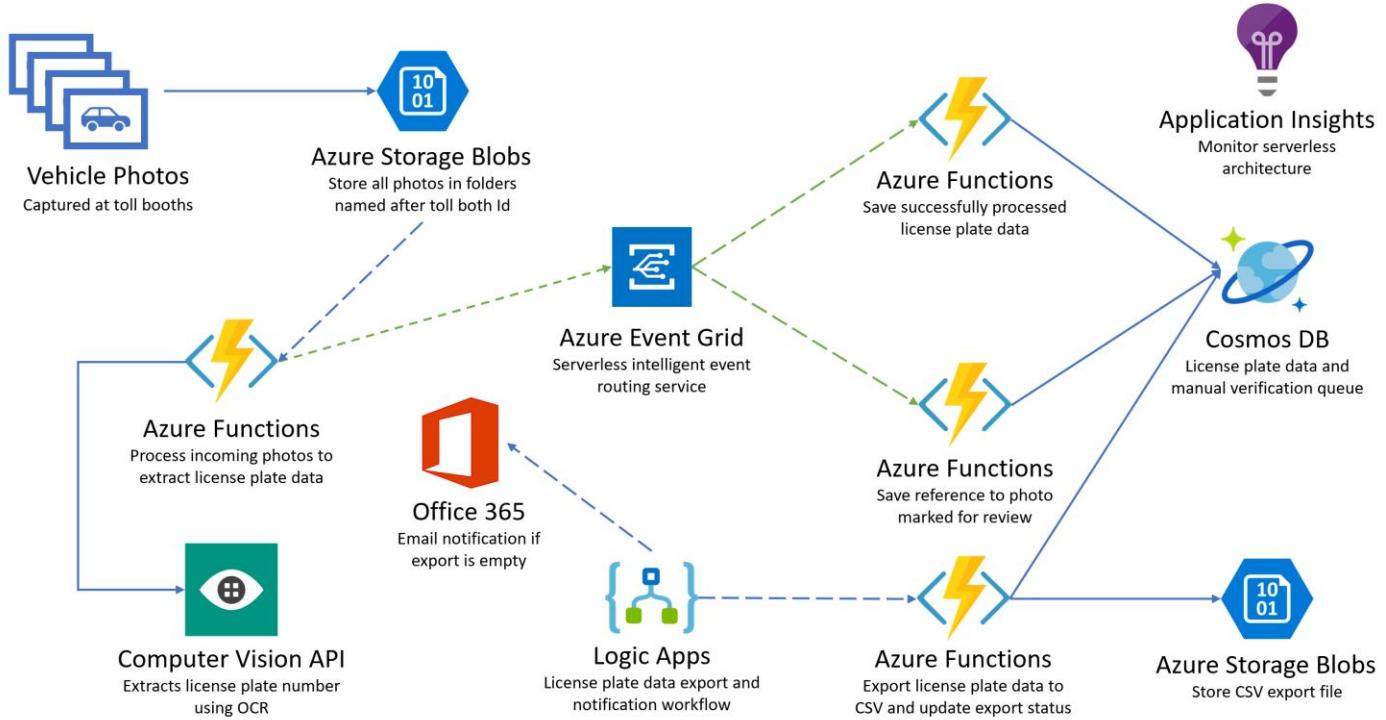
- Use a series of Azure Functions that independently scale and break down business logic to discrete components.
- Use computer vision algorithms within an Azure Function to accurately detect license plates in car images at scale.
- Provision and use Cosmos DB as a highly available NoSQL data store for processed data.
- Create a Logic App that contains a workflow to export processed license plates and conditionally send alerts based on successful or unsuccessful operation.
- Use App Insights to monitor the serverless topology, observing how well the solution scales when under load.
- Implement a Continuous Deployment DevOps process to automatically publish changes to Function Apps.

Overview

The Serverless architecture hands-on lab is an exercise that will challenge you to implement an end-to-end scenario using a supplied sample that is based on Microsoft Azure Functions, Azure Cosmos DB, Event Grid, and related services. The scenario will include implementing compute, storage, workflows, and monitoring, using various components of Microsoft Azure. The hands-on lab can be implemented on your own, but it is highly recommended to pair up with other members at the lab to model a real-world experience and to allow each member to share their expertise for the overall solution.

Solution architecture

Below is a diagram of the solution architecture you will build in this lab. Please study this carefully, so you understand the whole of the solution as you are working on the various components.



The solution begins with vehicle photos being uploaded to an Azure Storage blobs container, as they are captured. A blob storage trigger fires on each image upload, executing the photo processing **Azure Function** endpoint (on the left-hand side of the diagram), which in turn sends the photo to the **Cognitive Services Computer Vision API OCR** service to extract the license plate data. If processing was successful and the license plate number was returned, the function submits a new Event Grid event, along with the data, to an Event Grid topic with an event type called "savePlateData". However, if the processing was unsuccessful, the function submits an Event Grid event to the topic with an event type called "queuePlateForManualCheckup". Two separate functions are configured to trigger when new events are added to the Event Grid topic, each filtering on a specific event type, both saving the relevant data to the appropriate **Azure Cosmos DB** collection for the outcome, using the Cosmos DB output binding. A **Logic App** that runs on a 15-minute interval executes an Azure Function via its HTTP trigger, which is responsible for obtaining new license plate data from Cosmos DB and exporting it to a new CSV file saved to Blob storage. If no new license plate records are found to export, the Logic App sends an email notification to the Customer Service department via their Office 365 subscription. **Application Insights** is used to monitor all of the Azure Functions in real-time as data is being processed through the serverless architecture. This real-time monitoring allows you to observe dynamic scaling first-hand and configure alerts when certain events take place.

Requirements

1. Microsoft Azure subscription (non-Microsoft subscription)
2. Local machine or a virtual machine configured with (**complete the day before the lab!**):
 - a. Visual Studio Community 2017 or greater, **version 15.4** or later
 - i. <https://www.visualstudio.com/vs/>
 - b. Azure development workload for Visual Studio 2017
 - i. <https://docs.microsoft.com/azure/azure-functions/functions-develop-vs#prerequisites>
 - c. .NET Framework 4.7 runtime (or higher)
 - i. <https://www.microsoft.com/net/download/windows>
3. Office 365 account. If required, you can sign up for an Office 365 trial at:
 - d. <https://portal.office.com/Signup/MainSignup15.aspx?Dap=False&QuotId=79a957e9-ad59-4d82-b787-a46955934171&ali=1>
4. GitHub account. You can create a free account at <https://github.com>.

Before the hands-on lab

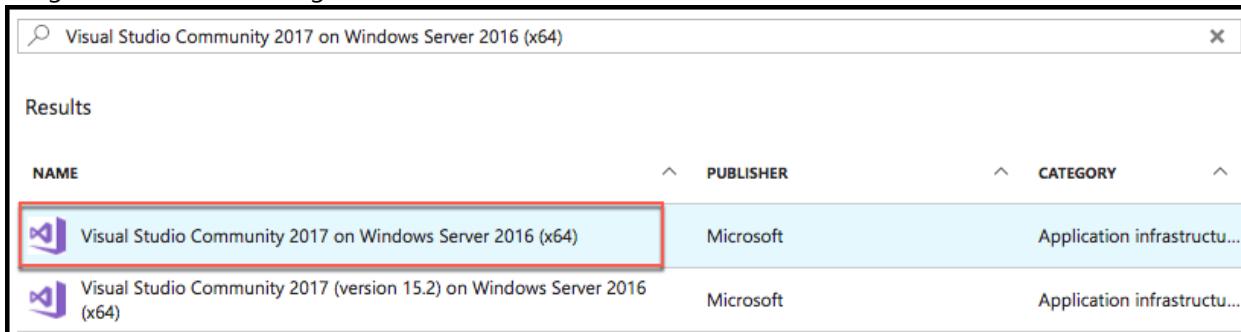
Duration: 10 minutes

In this exercise, you will set up your environment you will use for the rest of the exercises. This will involve downloading the sample application and creating your Azure resource group for the lab.

Task 1: Set up a development environment

If you do not have a machine with Visual Studio Community 2017 (or greater) and the Azure development workload, complete this task.

1. Create a virtual machine (VM) in Azure using the Visual Studio Community 2017 on Windows Server 2016 (x64) image. A Windows 10 image will work as well.



Note: It is highly recommended to use a DS2 or D2 instance size for this VM.

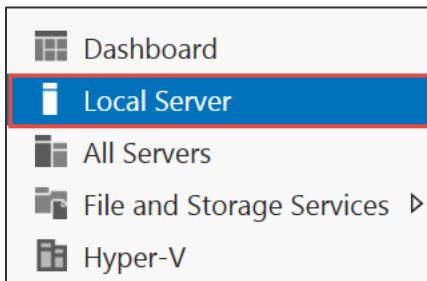
Task 2: Disable IE Enhanced Security

Note: Sometimes this image has IE ESC disabled. Sometimes it does not.

1. On the new VM you just created, select the **Server Manager** icon.



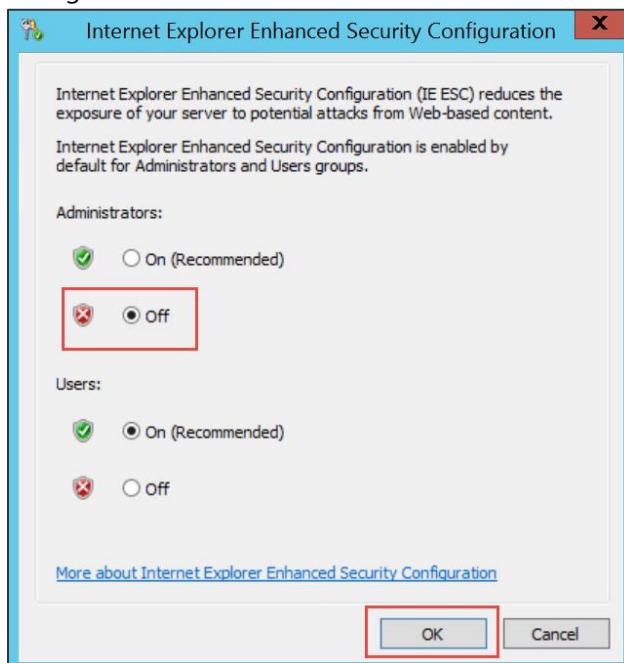
2. Select **Local Server**.



3. On the right side of the pane, select **On** by **IE Enhanced Security Configuration**.

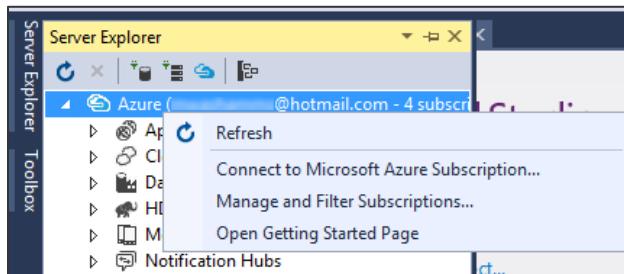


4. Change to **Off** for Administrators and select **OK**.



Task 3: Validate connectivity to Azure

1. From within the virtual machine, launch Visual Studio and validate that you can log in with your Microsoft Account when prompted.
2. To validate connectivity to your Azure subscription, launch Visual Studio, open **Server Explorer** from the **View** menu, and ensure that you can connect to your Azure subscription.



Task 4: Download and explore the TollBooth starter solution

1. Create a new folder on your C: drive named **Hackathon**.
2. Download the sample application from here: <http://bit.ly/2D0uo6z> and extract to the **Hackathon** folder. You will need to copy and paste the URL into your browser, as Ctrl+Click does not honor casing in the URL.

Note: The link above is case sensitive.

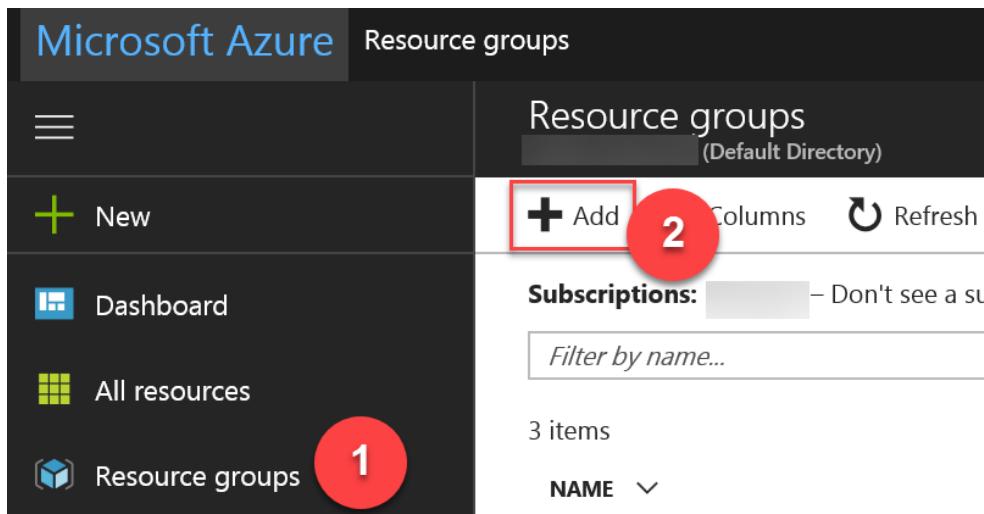
- From the **TollBooth** folder under **Hackathon**, open the Visual Studio Solution file: **TollBooth.sln**.
The solution contains the following projects:

TollBooth	The TollBooth Azure Function App project
UploadImages	Local console project for uploading either a handful of car photos, or 1,000 for testing scalability of the serverless architecture

- Go back to the Hackathon folder and open the **license plates** subfolder. It contains sample license plate photos used for testing out the solution. One of the photos is guaranteed to fail OCR processing, which is meant to show how the workload is designed to handle such failures. The **copyfrom** folder is used by the UploadImages project as a basis for the 1,000 photo upload for testing scalability.

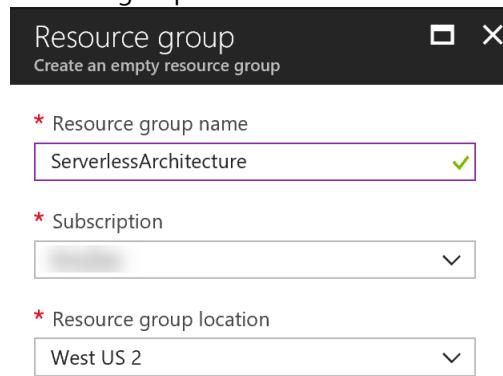
Task 5: Create a new Azure Resource group

- Within the Azure Management Portal, open the **Resource groups** tile and select **Add**.



The screenshot shows the Azure Management Portal interface. On the left, there is a sidebar with icons for 'New', 'Dashboard', 'All resources', and 'Resource groups'. The 'Resource groups' icon is highlighted with a red circle containing the number 1. On the right, the main area is titled 'Resource groups (Default Directory)'. It shows a list of resource groups with a red circle containing the number 2 over the 'Add' button, which is highlighted with a red box.

- Specify the name of the resource group as **ServerlessArchitecture**, and choose the Azure region to which you want to deploy the lab. This resource group will be used throughout the rest of the lab. Select **Create** to create the resource group.



The screenshot shows the 'Resource group' creation dialog. It has fields for 'Resource group name' (containing 'ServerlessArchitecture' with a checkmark), 'Subscription' (a dropdown menu), 'Resource group location' (containing 'West US 2' with a dropdown arrow), and a 'Create' button.

Exercise 1: Azure data, storage, and serverless environment setup

Duration: 30 minutes

You must provision a few resources in Azure before you start developing the solution. Ensure all resources use the same resource group for easier cleanup.

In this exercise, you will provision a blob storage account using the Hot tier, and create two containers within to store uploaded photos and exported CSV files. You will then provision two Function Apps instances, one you will deploy from Visual Studio, and the other you will manage using the Azure portal. Next, you will create a new Event Grid topic. After that, you will create an Azure Cosmos DB account with two collections. Finally, you will provision a new Cognitive Services Computer Vision API service for applying object character recognition (OCR) on the license plates.

Help references

Creating a storage account (blob hot tier)	https://docs.microsoft.com/en-us/azure/storage/common/storage-create-storage-account?toc=%2fazure%2fstorage%2fblobs%2ftoc.json#create-a-storage-account
Creating a function app	https://docs.microsoft.com/en-us/azure/azure-functions/functions-create-first-azure-function#create-a-function-app
Concepts in Event Grid	https://docs.microsoft.com/en-us/azure/event-grid/concepts
Creating an Azure Cosmos DB account	https://docs.microsoft.com/en-us/azure/cosmos-db/tutorial-develop-sql-api-dotnet#create-an-azure-cosmos-db-account

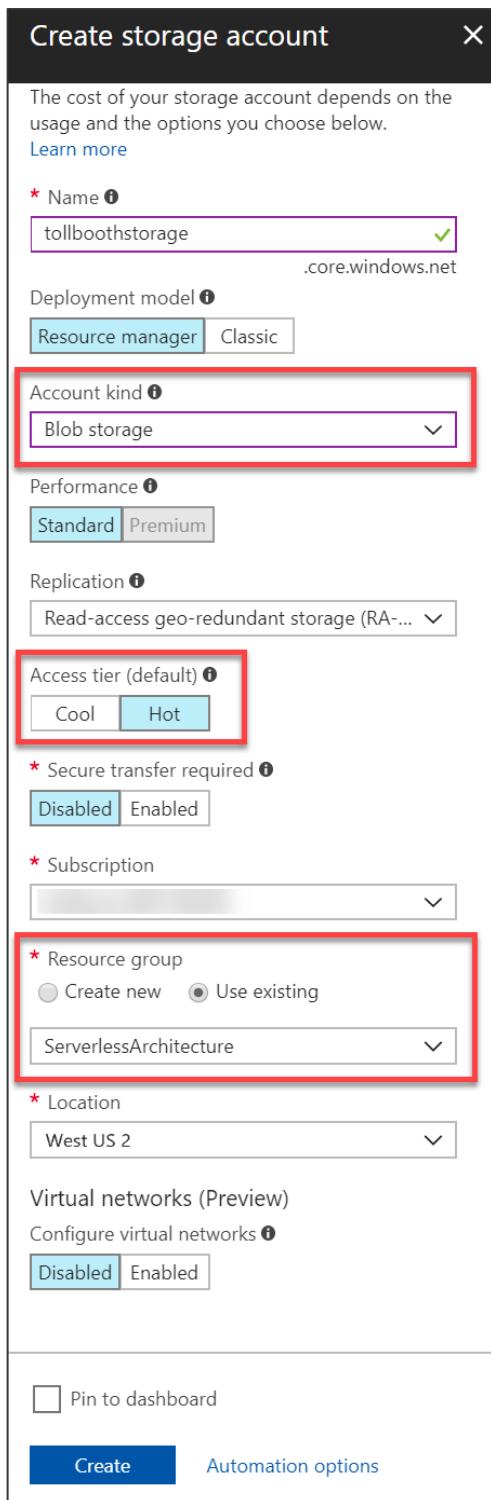
Task 1: Provision the storage account

1. Using a new tab or instance of your browser, navigate to the Azure Management portal, <http://portal.azure.com>.
2. Select **+New, Storage, Storage account – blob, file, table, queue**.

The screenshot shows the Azure Management Portal. On the left, the 'New' blade is open, with a red box highlighting the 'New' button. Below it is a list of service categories: Dashboard, Resource groups, All resources, Recent, App Services, SQL databases, and Virtual machines (classic). On the right, the 'Storage' section of the Azure Marketplace is displayed. A red box highlights the 'Storage account - blob, file, table, queue' item, which includes a 'Quickstart tutorial' link. Another red box highlights the 'Storage' category in the marketplace navigation bar. Other items listed in the marketplace include 'Azure File Sync (preview)', 'Data Lake Store', and 'Containers'.

3. On the **Create storage account** blade, specify the following configuration options:
 - a. **Name:** enter a unique value for the storage account (ensure the green check mark appears)
 - b. Select **Blob storage** as the account kind.
 - c. Select the **Hot** access tier.

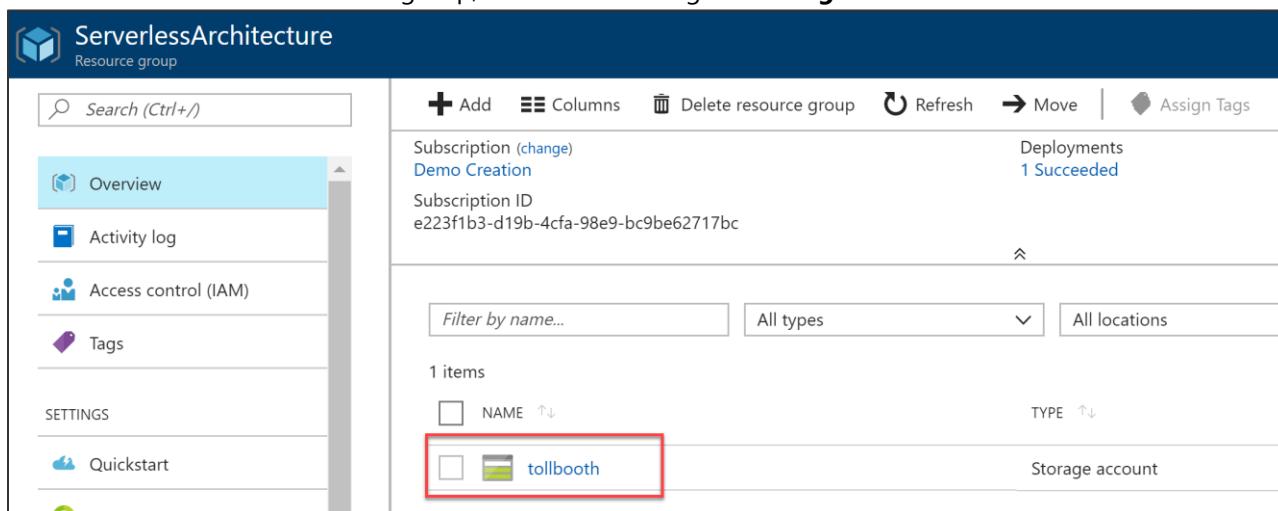
- d. For **Resource group**, select the **Use existing** radio button, and specify the **ServerlessArchitecture** resource group.
- e. Ensure the **Location** is the same region as the resource group.



4. Select **Create**.

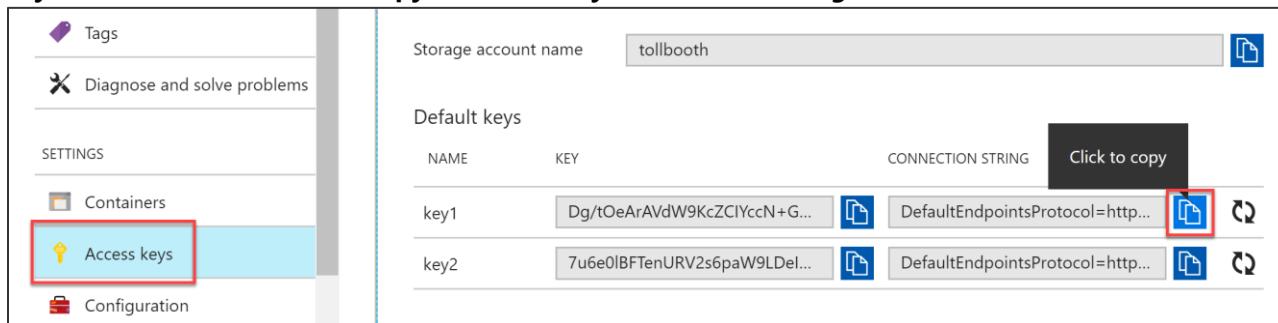


5. After the storage account has completed provisioning, open the storage account by opening the **ServerlessArchitecture** resource group, and then selecting the **storage account** name.



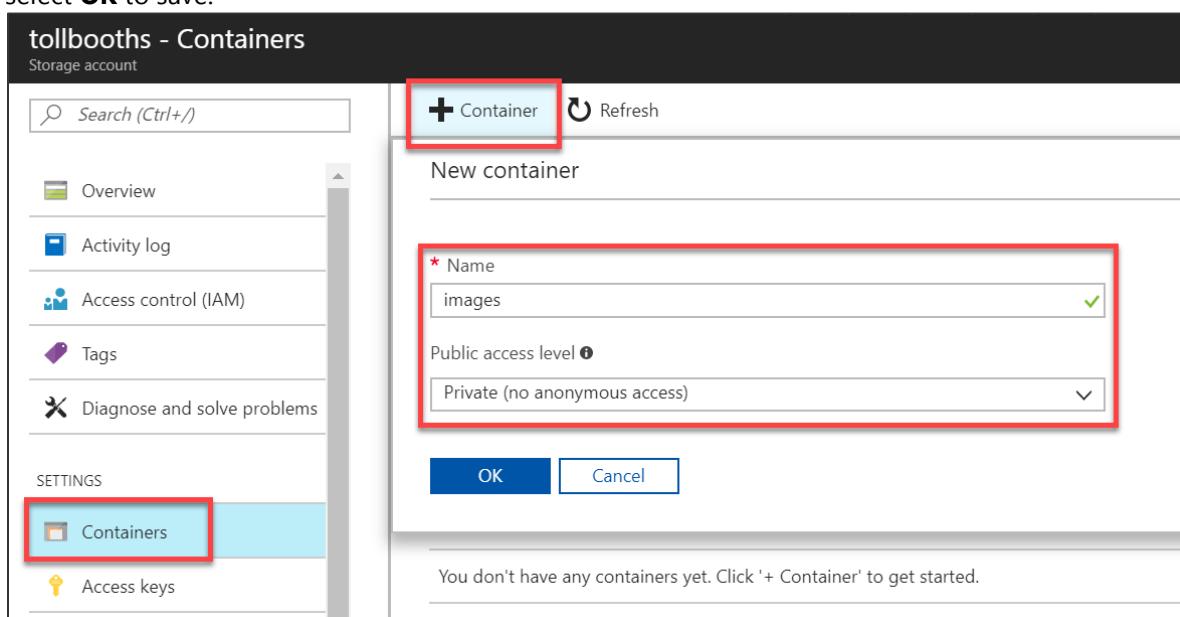
The screenshot shows the Azure portal's resource group interface. The left sidebar lists 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'SETTINGS', and 'Quickstart'. The main area shows a resource group named 'ServerlessArchitecture' with a 'Subscription (change)' of 'Demo Creation' and a 'Subscription ID' of 'e223f1b3-d19b-4cfa-98e9-bc9be62717bc'. A deployment '1 Succeeded' is listed. The search bar at the top contains 'Search (Ctrl+ /)'. The 'tollbooth' storage account is listed in the results, with its name highlighted by a red box.

6. On the **Storage account** blade, select **Access Keys**, under Settings in the left-hand menu. Then on the **Access keys** blade, select the **Click to copy** button for **key1 connection string**.



The screenshot shows the 'tollbooth' storage account blade. The left sidebar shows 'Tags', 'Diagnose and solve problems', 'SETTINGS', 'Containers' (highlighted with a red box), 'Access keys' (highlighted with a red box), and 'Configuration'. The main area shows the storage account name 'tollbooth'. Under 'Default keys', there are two entries: 'key1' and 'key2'. The 'key1' entry has a 'Click to copy' button, which is also highlighted with a red box.

7. Paste the value into a text editor, such as Notepad, for later reference.
8. Select **Containers** under **Settings** in the left-hand menu. Then on the **Containers** blade, select the **+ Container** button to add a new container. In the **Name** field, enter **images**, select **Private** for the public access level, then select **OK** to save.

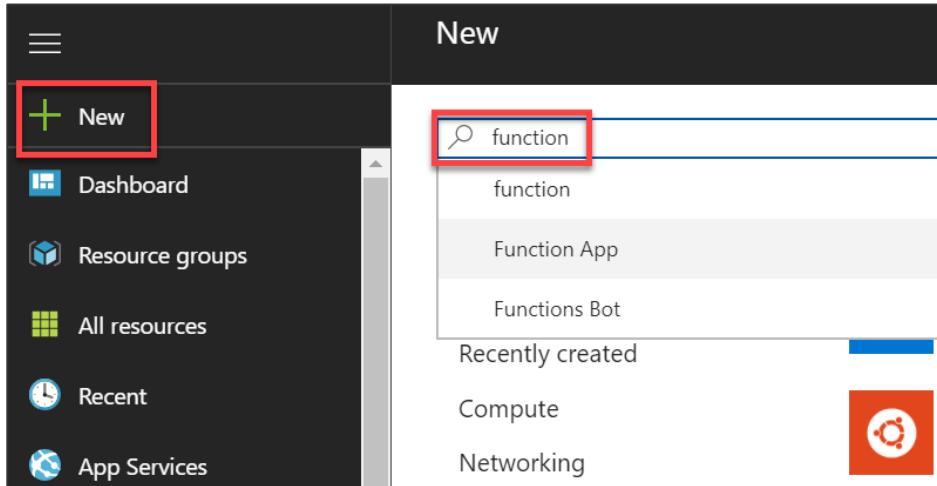


The screenshot shows the 'tollbooths - Containers' blade. The left sidebar shows 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'SETTINGS', 'Containers' (highlighted with a red box), and 'Access keys'. The main area shows a '+ Container' button highlighted with a red box. A 'New container' dialog is open, containing fields for 'Name' (set to 'images') and 'Public access level' (set to 'Private (no anonymous access)'). At the bottom of the dialog are 'OK' and 'Cancel' buttons.

9. Repeat these steps to create a container named **export**.

Task 2: Provision the Function Apps

1. Navigate to the Azure Management portal, <http://portal.azure.com>.
2. Select **+ New**, then enter **function** into the search box on top. Select **Function App** from the results.



3. Select the **Create** button on the **Function App overview** blade.
4. On the **Create Function App** blade, specify the following configuration options:
 - a. **Name:** unique value for the App name (ensure the green check mark appears). Provide a name similar to **TollBoothEvents**.
 - b. Specify the **Resource Group** **ServerlessArchitecture**.
 - c. Select the **Consumption Plan** Hosting Plan.
 - d. Select the same **location** as your Resource Group.
 - e. Leave the **storage** option as **create new**.

- f. Ensure **Off** is selected for **Application Insights** (we'll add this later).

The screenshot shows the 'Function App' creation dialog. Key fields include:

- App name:** TollBoothEvents (selected)
- Subscription:** (dropdown menu)
- Resource Group:** ServerlessArchitecture (selected)
- OS:** Windows (selected)
- Hosting Plan:** Consumption Plan
- Location:** West US 2
- Storage:** tollboothevents94e9 (selected)
- Application Insights:** Off (selected)
- Pin to dashboard:** (checkbox)

At the bottom are the **Create** and **Automation options** buttons.

5. Select **Create**.

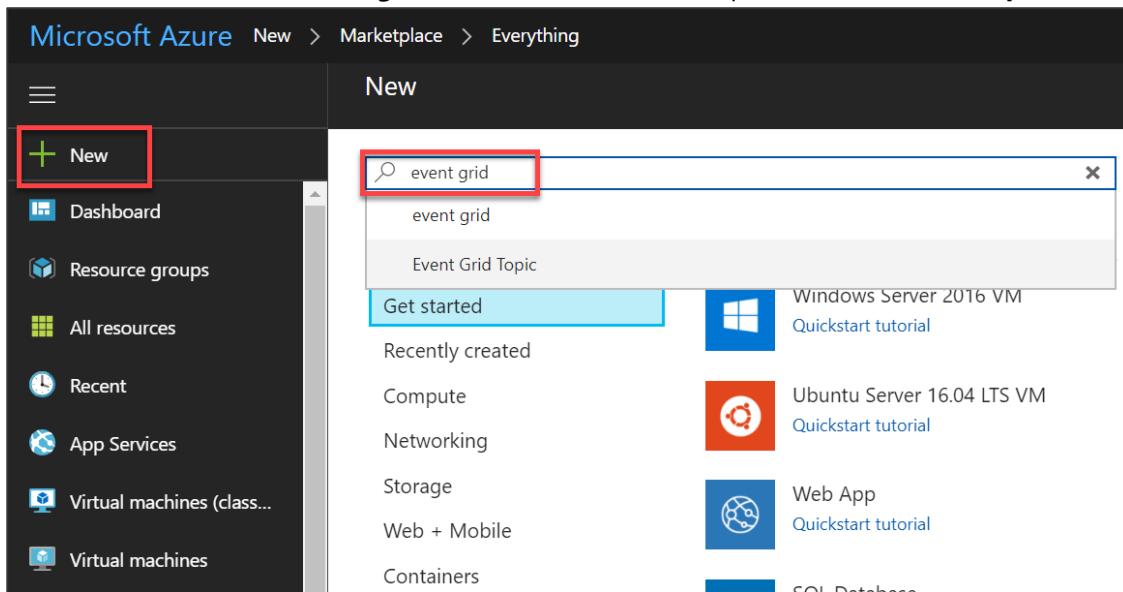
Create

6. **Repeat steps 1-5** to create a second Function App, named TollBooth**FunctionApp** or similar.

Task 3: Provision the Event Grid topic

1. Navigate to the Azure Management portal, <http://portal.azure.com>.

2. Select **+ New**, then enter **event grid** into the search box on top. Select **Event Grid Topic** from the results.



3. Select the **Create** button on the **Event Grid Topic overview** blade.
4. On the **Create Topic** blade, specify the following configuration options:

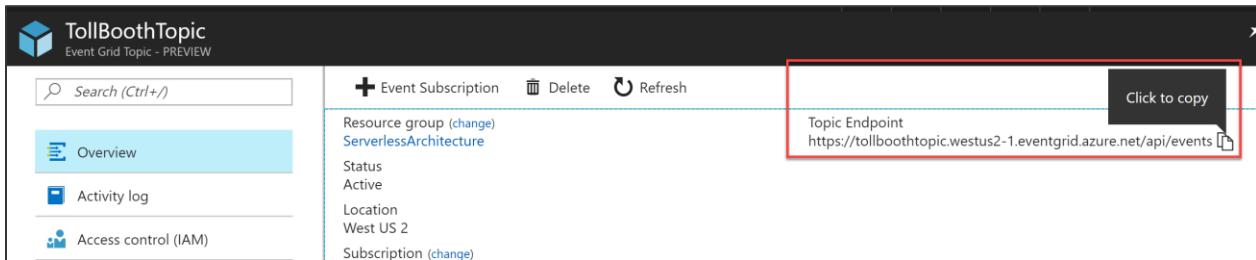
- Name:** unique value for the App name (ensure the green check mark appears).
- Specify the Resource Group** ServerlessArchitecture.
- Select the same **location** as your Resource Group.

The screenshot shows the 'Create Topic' blade for 'Event Grid - PREVIEW'. It has the following fields:

- Name:** 'TollBoothTopic' (highlighted with a red box)
- Subscription:** A dropdown menu (highlighted with a red box)
- Resource group:** A dropdown menu with options 'Create new' and 'Use existing'. 'Use existing' is selected, and 'ServerlessArchitecture' is chosen (highlighted with a red box)
- Location:** A dropdown menu with 'West US 2' selected (highlighted with a red box)

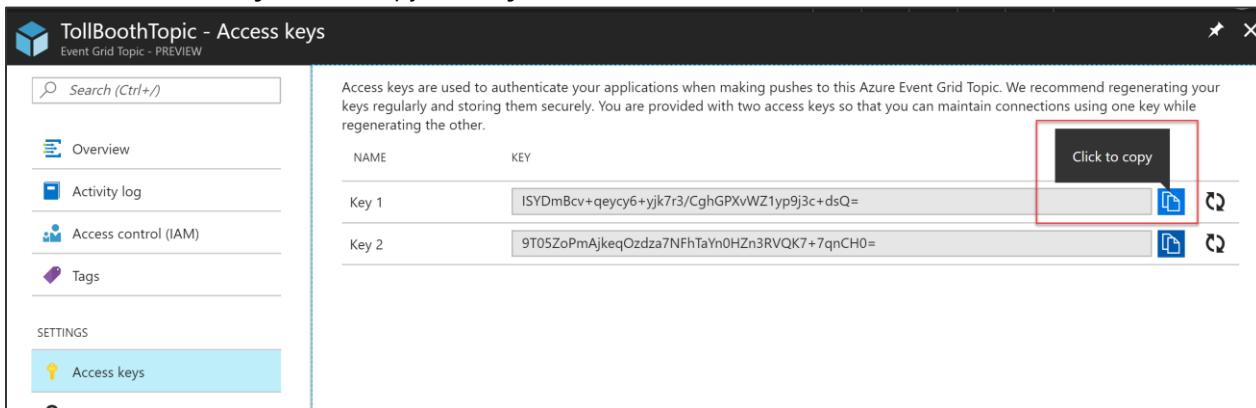
5. Select **Create**.
6. After the Event Grid topic has completed provisioning, open the account by opening the **ServerlessArchitecture** resource group, and then selecting the **Event Grid** topic name.

7. Select **Overview** in the left-hand menu, and then copy the **Topic Endpoint** value.



The screenshot shows the 'TollBoothTopic' Event Grid Topic - PREVIEW page. The left sidebar has 'Overview' selected. The main area shows the 'Topic Endpoint' as `https://tollboothtopic.westus2-1.eventgrid.azure.net/api/events`, with a 'Click to copy' button next to it. Other details shown include Resource group: ServerlessArchitecture, Status: Active, Location: West US 2, and Subscription: (change).

8. Select **Access Keys** under Settings in the left-hand menu.
9. Within the **Access Keys** blade, copy the **Key 1** value.

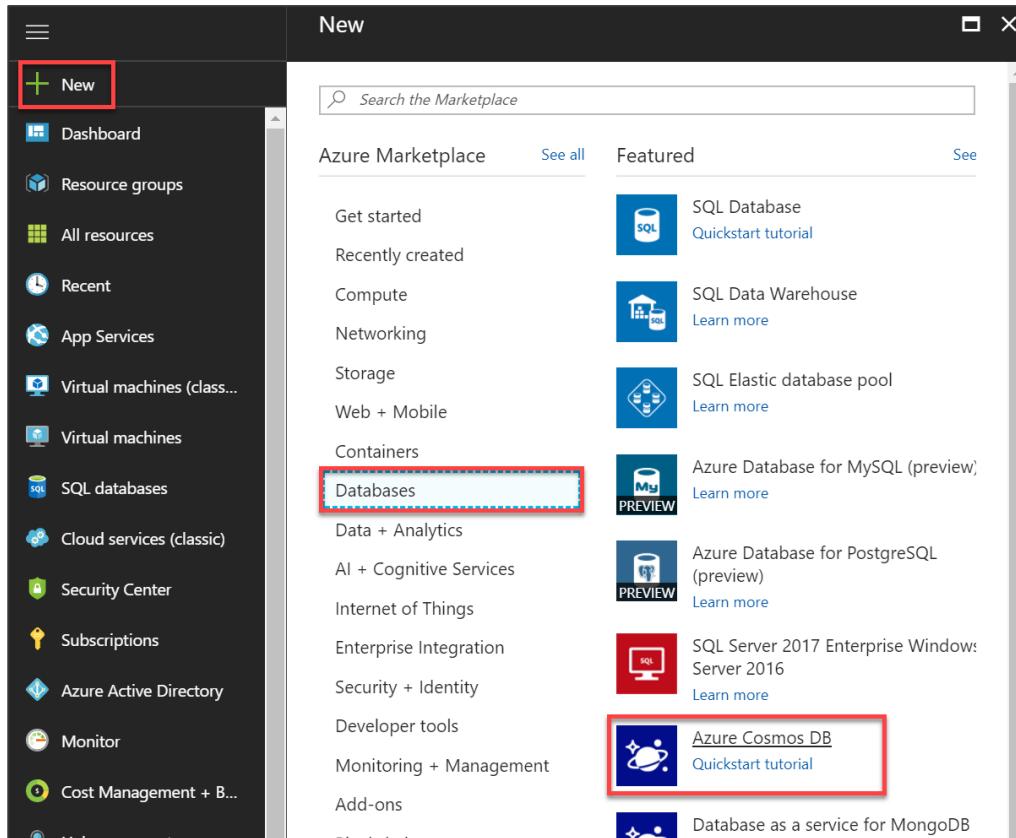


The screenshot shows the 'TollBoothTopic - Access keys' blade. The left sidebar has 'Access keys' selected. The main area displays two access keys: 'Key 1' with value `ISYDmBcv+qeycy6+yjk7r3/CghGPXvWZ1yp9j3c+dsQ=` and 'Key 2' with value `9T05ZoPmAjkeqOzdza7NFhTaYn0HZN3RVQK7+7qnCH0=`. Each key has a 'Click to copy' button next to it.

10. Paste the values into a text editor, such as Notepad, for later reference.

Task 4: Provision the Azure Cosmos DB account

1. Navigate to the Azure Management portal, <http://portal.azure.com>.
2. Select **+ New, Databases, Azure Cosmos DB**.



3. On the **Create new Azure Cosmos DB account** blade, specify the following configuration options:
 - a. **Name**: unique value for the App name (ensure the green check mark appears).
 - b. Select the **SQL API**.
 - c. Specify the **Resource Group** ServerlessArchitecture.
 - d. Select the same **location** as your Resource Group if available. Otherwise, select the next closest **region**.

e. Check **geo-redundancy**.

Azure Cosmos DB

New account

* ID: tollbooth

* API: SQL

* Subscription: [dropdown]

* Resource Group: Use existing: ServerlessArchitecture

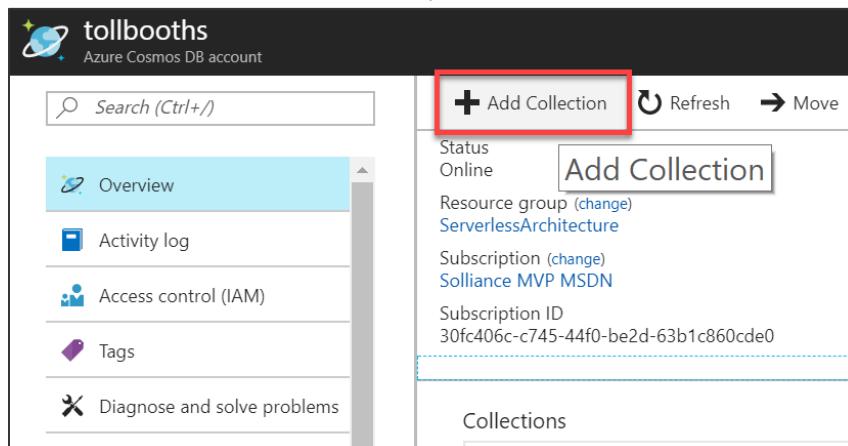
* Location: West US

Enable geo-redundancy

4. Select **Create**.

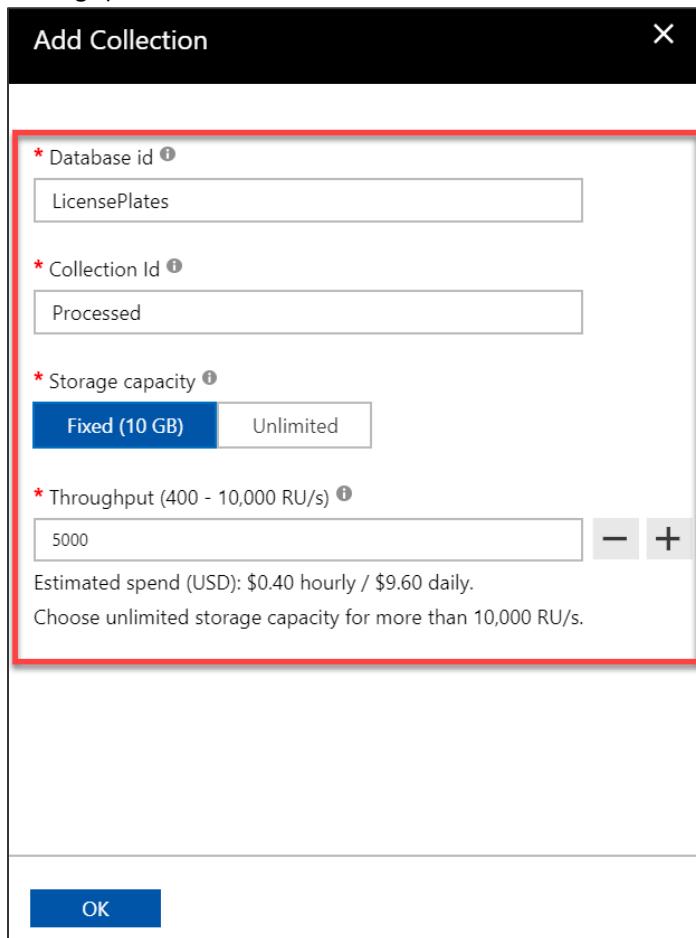


5. After the Azure Cosmos DB account has completed provisioning, open the account by opening the **ServerlessArchitecture** resource group, and then selecting the **Azure Cosmos DB** account name.
6. On the **Cosmos DB overview** blade, select **+ Add Collection**.



7. On the **Add Collection** blade, specify the following configuration options:
 - Enter **LicensePlates** for the **database id**.
 - Enter **Processed** for the **collection id**.
 - Storage capacity: **Fixed**.

- d. Throughput: **5000**.



8. Select **OK**.
9. Select **New Collection** to add another collection.
10. On the **Add Collection** blade, specify the following configuration options:
 - a. Enter **LicensePlates** for the database id.
 - b. Enter **NeedsManualReview** for the collection id.
 - c. Storage capacity: **Fixed**.

- d. Throughput: **5000**.

The screenshot shows the 'Add Collection' dialog box. It has a black header bar with the text 'Add Collection' and a close button 'X'. The main area contains several input fields and buttons. A red box highlights the 'Database id' field, which contains 'LicensePlates'. Below it is the 'Collection Id' field with 'NeedsManualReview'. Under 'Storage capacity', there are two options: 'Fixed (10 GB)' (selected) and 'Unlimited'. The 'Throughput' field is also highlighted with a red box and contains the value '5000'. Below the throughput field, there is a note: 'Estimated spend (USD): \$0.40 hourly / \$9.60 daily.' and 'Choose unlimited storage capacity for more than 10,000 RU/s.' At the bottom of the dialog is a blue 'OK' button.

11. Select **OK**.
12. Select **Keys** under Settings in the left-hand menu.

13. Underneath the **Read-write Keys** tab within the **Keys** blade, copy the **URI** and **Primary Key** values.

The screenshot shows the 'tollbooth - Keys' blade in the Azure portal. The left sidebar lists various account settings like Overview, Activity log, and Keys. The 'Keys' item is selected and highlighted with a red box. The main content area shows the 'Read-write Keys' tab selected. It displays the 'URI' (https://tollbooth.documents.azure.com:443/), 'PRIMARY KEY' (a long string of characters), and 'SECONDARY KEY' (another long string). Below these are 'PRIMARY CONNECTION STRING' and 'SECONDARY CONNECTION STRING', each with its own copy icon. The 'Read-only Keys' tab is also visible.

14. Paste the values into a text editor, such as Notepad, for later reference.

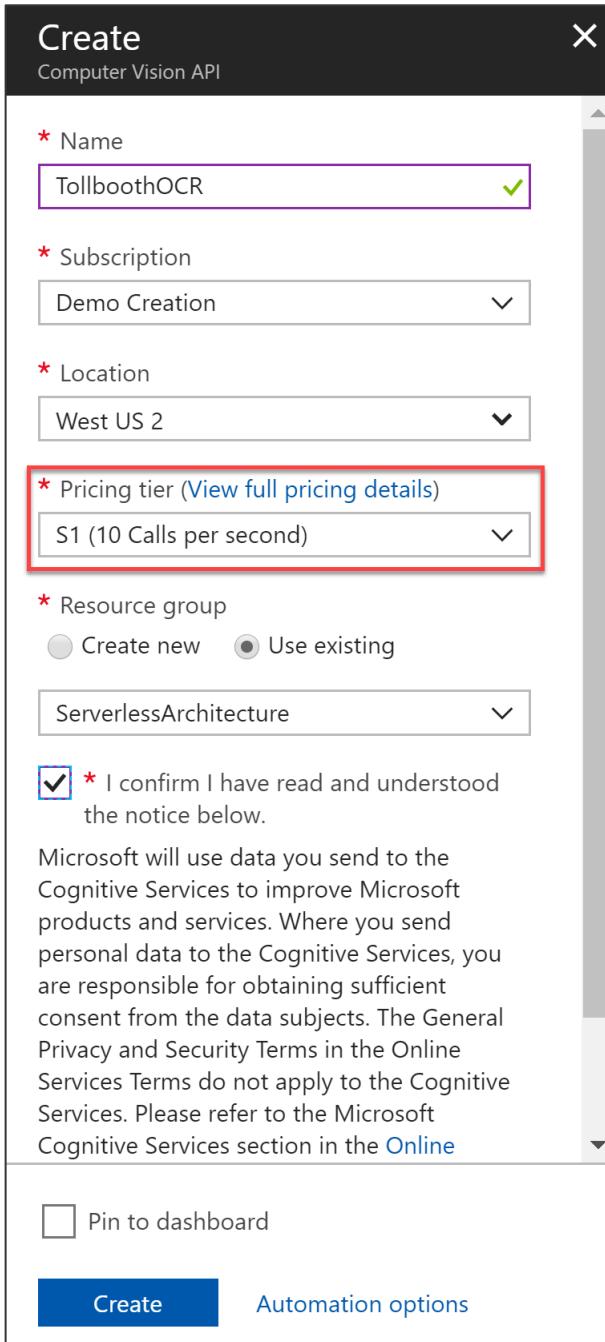
Task 5: Provision the Computer Vision API service

1. Navigate to the Azure Management portal, <http://portal.azure.com>.
2. Select **+ New**, then enter **computer vision** into the search box on top. Select **Computer Vision API** from the results.

The screenshot shows the 'Microsoft Azure' 'New' blade. The left sidebar has a 'New' button highlighted with a red box. The main area has a search bar with 'computer vision' typed in, and a list of results. The 'Computer Vision API' result is selected and highlighted with a red box. Other results include 'Windows Server 2016 VM' and 'Ubuntu Server 16.04 LTS VM'.

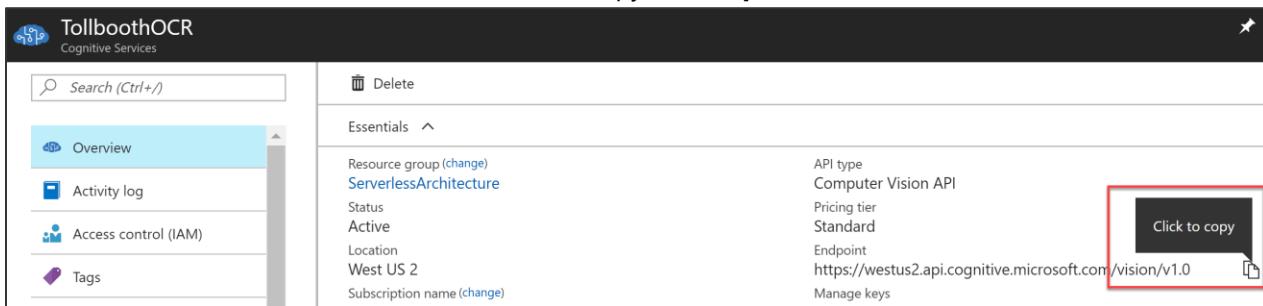
3. Select the **Create** button on the **Computer Vision API overview** blade.

4. On the **Create Computer Vision API** blade, specify the following configuration options:
 - a. **Name**: unique value for the App name (ensure the green check mark appears).
 - b. Specify the **Resource Group ServerlessArchitecture**.
 - c. Select the same **location** as your Resource Group.
 - d. Select the **S1 pricing tier**.



5. Select **Create**.
6. After the Computer Vision API has completed provisioning, open the service by opening the **ServerlessArchitecture** resource group, and then selecting the **Computer Vision API** service name.

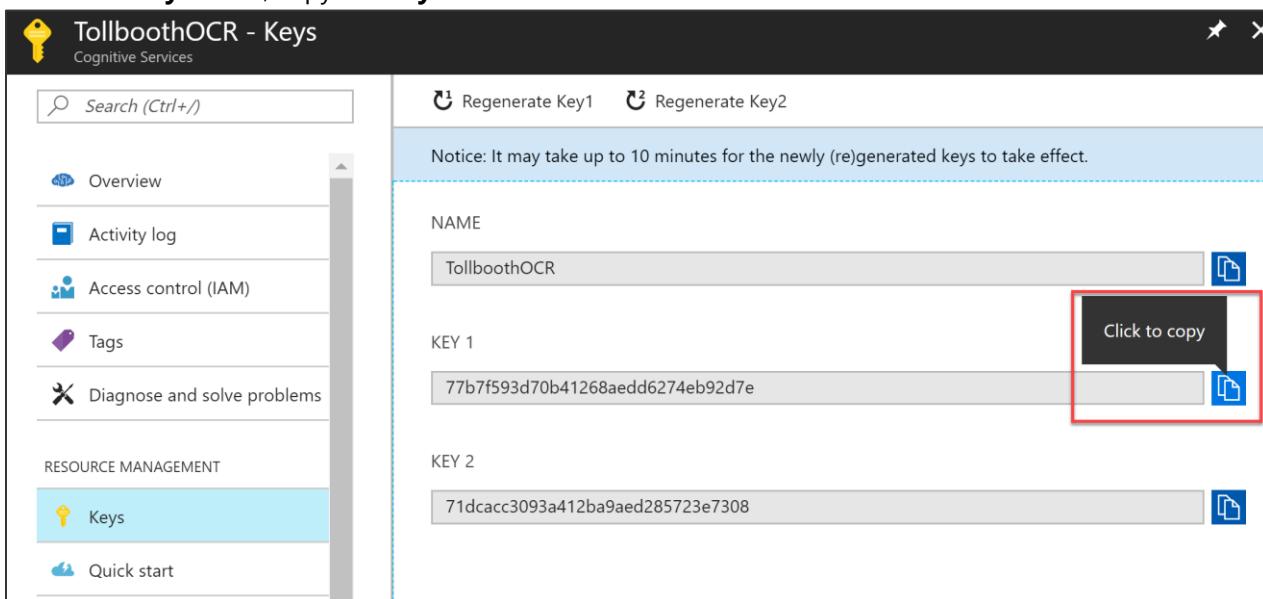
7. Select **Overview** in the left-hand menu, and then copy the **Endpoint** value.



The screenshot shows the Azure portal interface for the 'TollboothOCR' Cognitive Services resource. The left sidebar has 'Overview' selected. The main content area shows the resource details: Resource group (change) to 'ServerlessArchitecture', Status 'Active', Location 'West US 2', and Subscription name (change). On the right, the 'Endpoint' field is displayed as 'https://westus2.api.cognitive.microsoft.com/vision/v1.0', with a red box and a 'Click to copy' button overlaid.

8. Select **Keys** under **Resource Management** in the left-hand menu.

9. Within the **Keys** blade, copy the **Key 1** value.



The screenshot shows the 'TollboothOCR - Keys' blade in the Azure portal. The left sidebar has 'Keys' selected. The main content area shows the key details: NAME 'TollboothOCR', KEY 1 '77b7f593d70b41268aedd6274eb92d7e', and KEY 2 '71dcacc3093a412ba9aed285723e7308'. The 'KEY 1' value is highlighted with a red box and a 'Click to copy' button.

10. Paste the values into a text editor, such as Notepad, for later reference.

Exercise 2: Develop and publish the photo processing and data export functions

Duration: 45 minutes

Use Visual Studio 2017 and its integrated Azure Functions tooling to develop and debug the functions locally, and then publish them to Azure. The starter project solution, TollBooths, contains most of the code needed. You will add in the missing code before deploying to Azure.

Help references

Code and test Azure Functions locally	https://docs.microsoft.com/en-us/azure/azure-functions/functions-run-local
---------------------------------------	---

Task 1: Configure application settings

In this task, you will apply application settings using the Microsoft Azure Portal. You will then add the application settings to the TollBooth Starter Project.

1. Using a new tab or instance of your browser navigate to the Azure Management portal, <http://portal.azure.com>.
2. Open the **ServerlessArchitecture** resource group, and then select the Azure Function App you created whose name ends with **FunctionApp**. If you did not use this naming convention, that's fine. Just be sure to make note of the name so you can distinguish it from the Function App you will be developing using the portal later on.

9 items	
<input type="checkbox"/>	NAME ↑↓
<input type="checkbox"/>	TYPE ↑↓
<input type="checkbox"/>	 tollbooth
<input type="checkbox"/>	Azure Cosmos DB account
<input type="checkbox"/>	 tollbooth
	Storage account
<input type="checkbox"/>	 TollBoothEvents2
	App Service
<input type="checkbox"/>	 tollboothevents94e9
	Storage account
<input type="checkbox"/>	 tollboothfunctib90e
	Storage account
<input type="checkbox"/>	 TollBoothFunctionApp
	App Service
<input type="checkbox"/>	 TollboothOCR
	Cognitive Services
<input type="checkbox"/>	 TollBoothTopic
	Event Grid Topic
<input type="checkbox"/>	 WestUS2Plan
	App Service plan

3. Select on **Application settings** on the Overview pane.

4. Scroll down to the **Application settings** section. Use the **+ Add new setting** link to create the following additional Key / Value pairs (the key names must exactly match those found in the table below):

Application Key	Value
computerVisionApiUrl	Computer Vision API endpoint you copied earlier. Append /ocr to the end. Example: https://westus2.api.cognitive.microsoft.com/vision/v1.0/ocr
computerVisionApiKey	Computer Vision API key
eventGridTopicEndpoint	Event Grid Topic endpoint
eventGridTopicKey	Event Grid Topic access key
cosmosDBEndPointUrl	Cosmos DB URI
cosmosDBAuthorizationKey	Cosmos DB Primary Key
cosmosDBDatabaseId	Cosmos DB database id (LicensePlates)
cosmosDBCollectionId	Cosmos DB processed collection id (Processed)
exportCsvContainerName	Blob storage CSV export container name (export)
blobStorageConnection	Blob storage connection string

Application settings	
AzureWebJobsDashboard	DefaultEndpointsProtocol=https;AccountName=tollboothfunctib90e;AccountKey=
AzureWebJobsStorage	DefaultEndpointsProtocol=https;AccountName=tollboothfunctib90e;AccountKey=
FUNCTIONS_EXTENSION_VERSION	~1
WEBSITE_CONTENTAZUREFILECONNECTI...	DefaultEndpointsProtocol=https;AccountName=tollboothfunctib90e;AccountKey=
WEBSITE_CONTENTSHARE	tollboothfunctionappb90e
WEBSITE_NODE_DEFAULT_VERSION	6.5.0
computerVisionApiUrl	https://westus2.api.cognitive.microsoft.com/vision/v1.0/ocr
computerVisionApiKey	77b7f593d70b41268aedd6274eb92d7e
eventGridTopicEndpoint	https://tollboothtopic.westus2-1.eventgrid.azure.net/api/events
eventGridTopicKey	ISYDmBcv+qeycy6+yjk7r3/CghGPXvWZ1yp9j3c+dsQ=
cosmosDBEndPointUrl	https://tollbooth.documents.azure.com:443/
cosmosDBAuthorizationKey	3xmUtmc8neWZytbBGSTKhcVNaSvIWurNI2gbUU0nnnkgXskplv5J
cosmosDBDatabaseId	LicensePlates
cosmosDBCollectionId	Processed
exportCsvContainerName	export
blobStorageConnection	DefaultEndpointsProtocol=https;AccountName=tollbooth;AccountKey=
+ Add new setting	

5. Select **Save**.



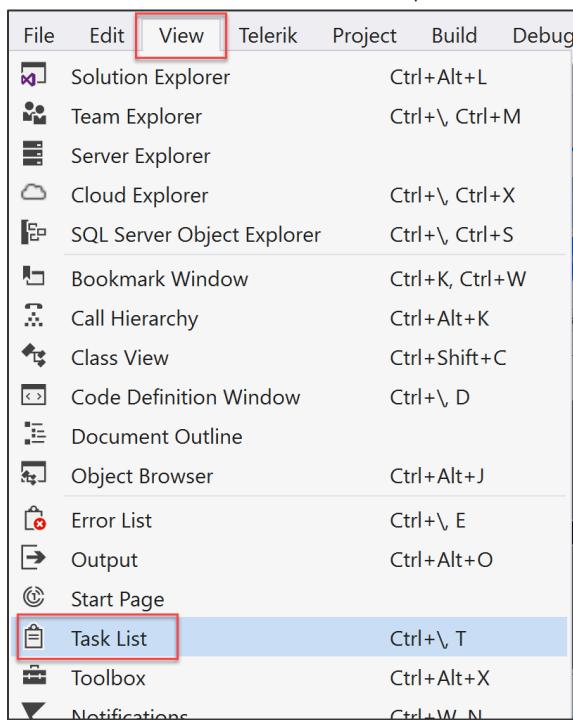
6. *Optional steps*, only if you wish to debug the functions locally on your development machine:
- Update the local.settings.json file with the same values.
 - Update the AzureWebJobsStorage and AzureWebJobsDashboard values in local.settings.json with those found under Application settings for the Function App. Save your changes.

Task 2: Finish the ProcessImage function

There are a few components within the starter project that must be completed, marked as TODO in the code. The first set of TODO items we will address are in the ProcessImage function, the FindLicensePlateText class that calls the Computer Vision API, and finally the SendToEventGrid.cs class, which is responsible for sending processing results to the Event Grid topic you created earlier.

Important Note: Do **not** update the version of any NuGet package. This solution is built to function with the NuGet package versions currently defined within. Updating these packages to newer versions could cause unexpected results.

1. Navigate to the **TollBooth** project using the Solution Explorer of Visual Studio.
2. From the Visual Studio **View** menu, select **Task List**.



3. There you will see a list of TODO tasks, where each task represents one line of code that needs to be completed.

The image shows the 'Task List' window in Visual Studio. The window displays a list of TODO tasks with columns for Description, Project, File, and Line. The tasks listed are:

Description	Project	File	Line
TODO 5: Retrieve a List of LicensePlateDataDocument objects from the collectionLink where the exported value is false.	TollBooth	DatabaseMethods.cs	42
TODO 6: Remove the line below.	TollBooth	DatabaseMethods.cs	44
TODO 7: Asynchronously upload the blob from the memory stream.	TollBooth	FileMethods.cs	60
TODO 2: Populate the below two variables with the correct AppSettings properties.	TollBooth	FindLicensePlateText.cs	45
TODO 1: Set the licensePlateText value by awaiting a new FindLicensePlateText.GetLicensePlate method.	TollBooth	ProcessImage.cs	34
TODO 3: Modify send method to include the proper eventType name value for saving plate data.	TollBooth	SendToEventGrid.cs	31
TODO 4: Modify send method to include the proper eventType name value for queuing plate for manual review.	TollBooth	SendToEventGrid.cs	36

4. Open **ProcessImage.cs**. Notice that the Run method is decorated with the `FunctionName` attribute, which sets the name of the Azure Function to "ProcessImage". This has a BlobTrigger that watches for new blobs being added to the images container of the storage account that was created in Exercise 1.
5. The following code represents the completed task in `ProcessImage.cs`:

```
// TODO 1: Set the licensePlateText value by awaiting a new FindLicensePlateText.GetLicensePlate method.
licensePlateText = await new FindLicensePlateText(log,
_client).GetLicensePlate(licensePlateImage);
```

6. Open **FindLicensePlateText.cs**. This class is responsible for contacting the Computer Vision API to find and extract the license plate text from the photo, using OCR. Notice that this class also shows how you can implement a resiliency pattern using Polly. This is useful for ensuring that you do not overload downstream services, in this case, the Computer Vision API. This will be demonstrated later on when visualizing the Function's scalability.

7. The following code represents the completed task in `FindLicensePlateText.cs`:

```
// TODO 2: Populate the below two variables with the correct AppSettings properties.  
var uriBase = ConfigurationManager.AppSettings["computerVisionApiUrl"];  
var apiKey = ConfigurationManager.AppSettings["computerVisionApiKey"];
```

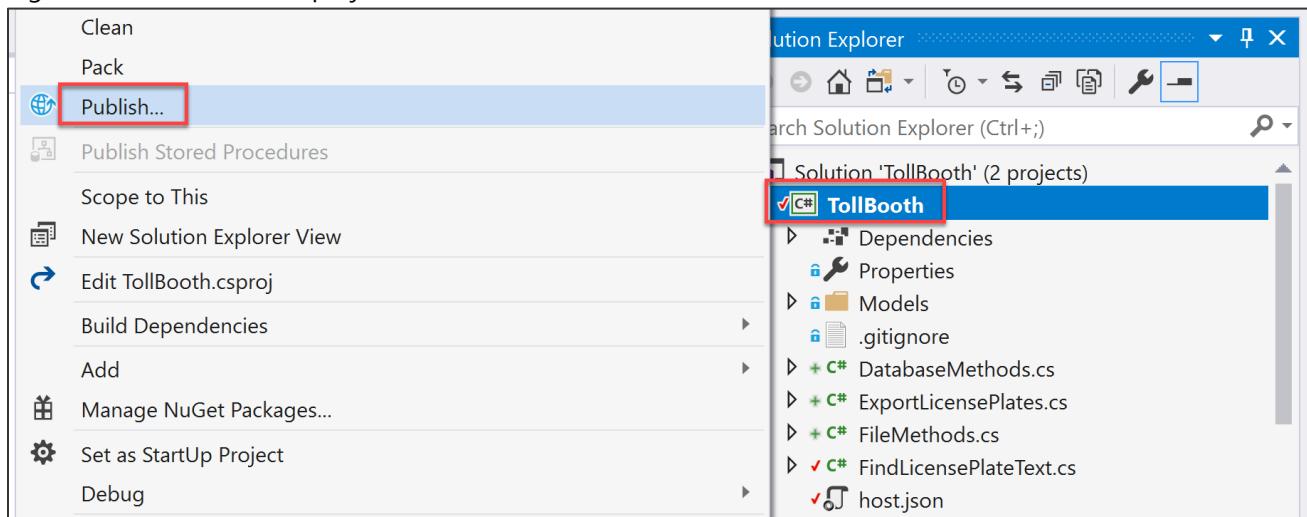
8. Open **SendToEventGrid.cs**. This class is responsible for sending an Event to the Event Grid topic, including the event type and license plate data. Event listeners will use the event type to filter and act on the events they need to process. Make note of the event types defined here (the first parameter passed into the `Send` method), as they will be used later on when creating new functions in the second Function App you provisioned earlier.
9. The following code represents the completed tasks in `SendToEventGrid.cs`:

```
// TODO 3: Modify send method to include the proper eventType name value for saving plate data.  
await Send("savePlateData", "TollBooth/CustomerService", data);  
  
// TODO 4: Modify send method to include the proper eventType name value for queuing plate for  
// manual review.  
await Send("queuePlateForManualCheckup", "TollBooth/CustomerService", data);
```

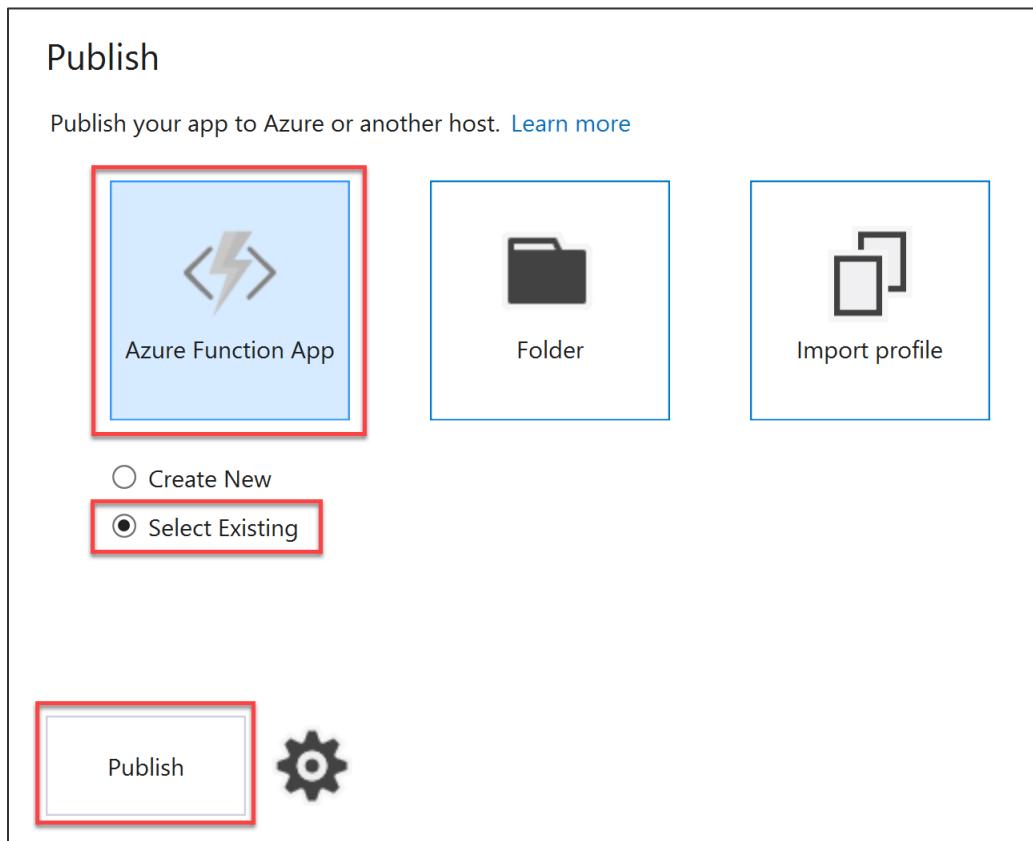
Task 3: Publish the Function App from Visual Studio

In this task, you will publish the Function App from the starter project in Visual Studio to the existing Function App you provisioned in Azure.

1. Navigate to the **TollBooth** project using the Solution Explorer of Visual Studio.
2. Right-click the **TollBooth** project and select **Publish...** from the context menu.

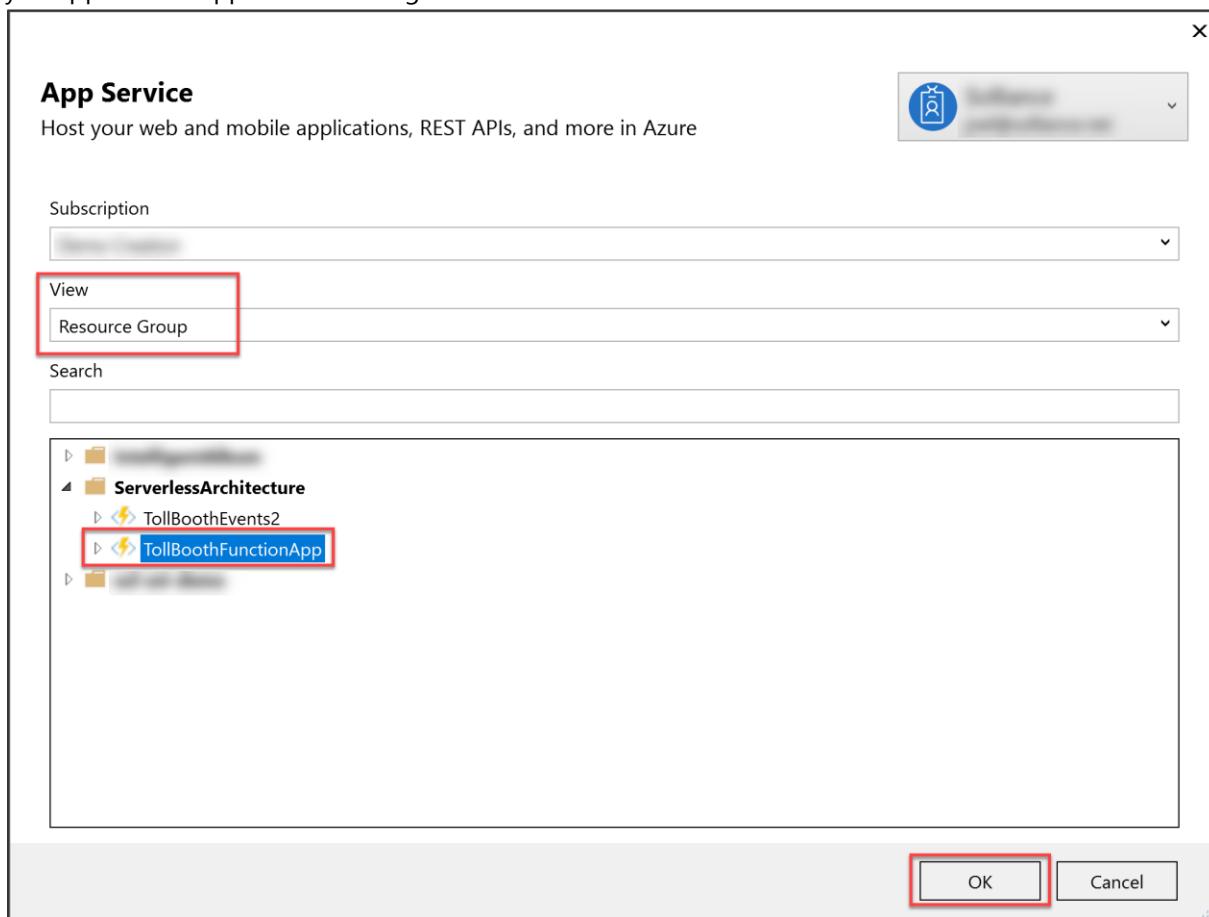


3. In the Publish window that appears, make sure **Azure Function App** is selected, choose the **Select Existing** radio button, then select **Publish**.



4. In the App Service form, select your **Subscription**, select **Resource Group** under **View**, then expand your **ServerlessArchitecture** resource group and select the Function App whose name ends with **FunctionApp**.

5. Whatever you named the Function App when you provisioned it is fine. Just make sure it is the same one to which you applied the Application Settings in Task 1 of this exercise.



6. After you select the Function App, select **OK**.
7. Watch the Output window in Visual Studio as the Function App publishes. When it is finished, you should see a message that says, "Publish Succeeded."
8. Using a new tab or instance of your browser navigate to the Azure Management portal, <http://portal.azure.com>.
9. Open the **ServerlessArchitecture** resource group, then select the Azure Function App to which you just published.

10. Expand the functions underneath your Function App in the left-hand menu. You should see both functions you just published from the Visual Studio solution listed.

The screenshot shows the Azure Functions Overview page for the 'TollBoothFunctionApp'. The left sidebar lists 'Function Apps' and the specific app 'TollBoothFunctionApp'. Under 'TollBoothFunctionApp', the 'Functions (Read Only)' section is expanded, showing two functions: 'ExportLicensePlates' and 'ProcessImage'. These two functions are highlighted with a red box. The right panel displays the 'Overview' tab, showing the status as 'Running' and the subscription as 'Demo Create'. There are also links for 'Function app settings' and 'Application settings'.

Exercise 3: Create functions in the portal

Duration: 45 minutes

Create two new Azure Functions written in Node.js, using the Azure portal. These will be triggered by Event Grid and output to Azure Cosmos DB to save the results of license plate processing done by the ProcessImage function.

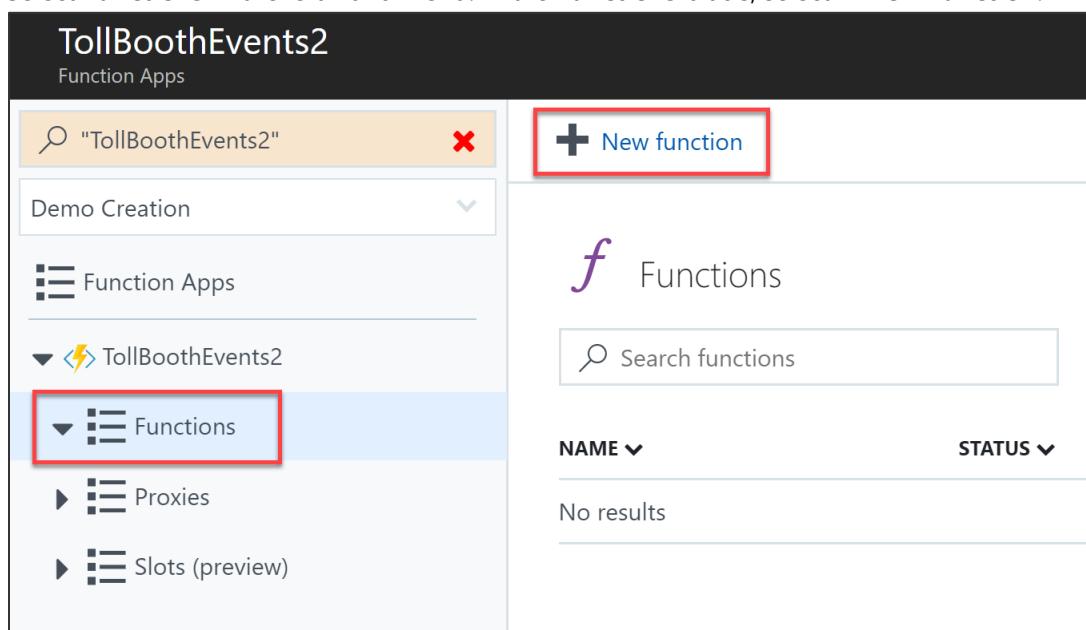
Help references

Create your first function in the Azure portal	https://docs.microsoft.com/en-us/azure/azure-functions/functions-create-first-azure-function
Store unstructured data using Azure Functions and Azure Cosmos DB	https://docs.microsoft.com/en-us/azure/azure-functions/functions-integrate-store-unstructured-data-cosmosdb

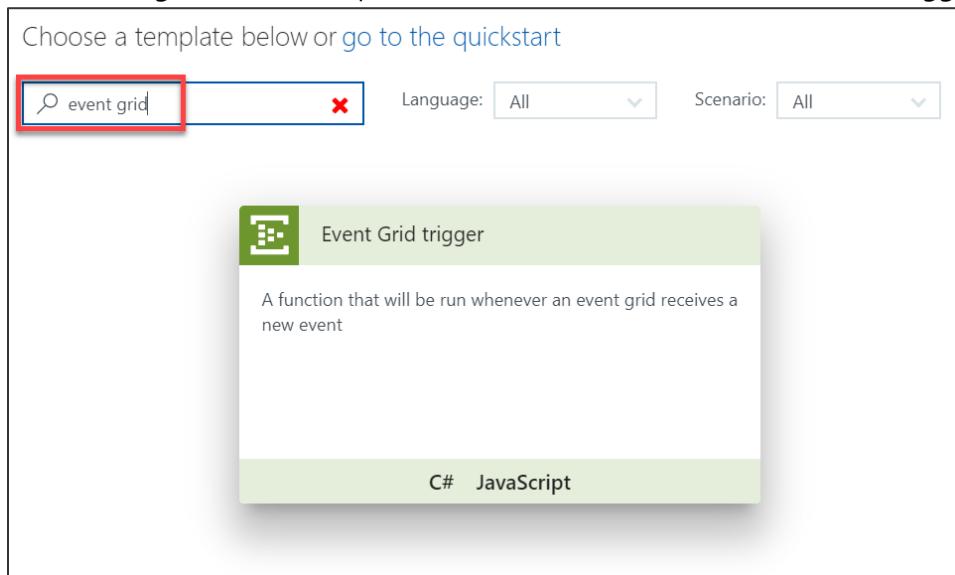
Task 1: Create function to save license plate data to Azure Cosmos DB

In this task, you will create a new Node.js function triggered by Event Grid and that outputs successfully processed license plate data to Azure Cosmos DB.

1. Using a new tab or instance of your browser navigate to the Azure Management portal, <http://portal.azure.com>.
2. Open the **ServerlessArchitecture** resource group, then select the Azure Function App you created whose name ends with **Events**. If you did not use this naming convention, make sure you select the Function App that you did not deploy to in the previous exercise.
3. Select **Functions** in the left-hand menu. In the **Functions** blade, select **+ New Function**.



4. Event **event grid** into the template search form, then select the **Event Grid trigger** template.



Choose a template below or [go to the quickstart](#)

event grid

Language: All Scenario: All

Event Grid trigger

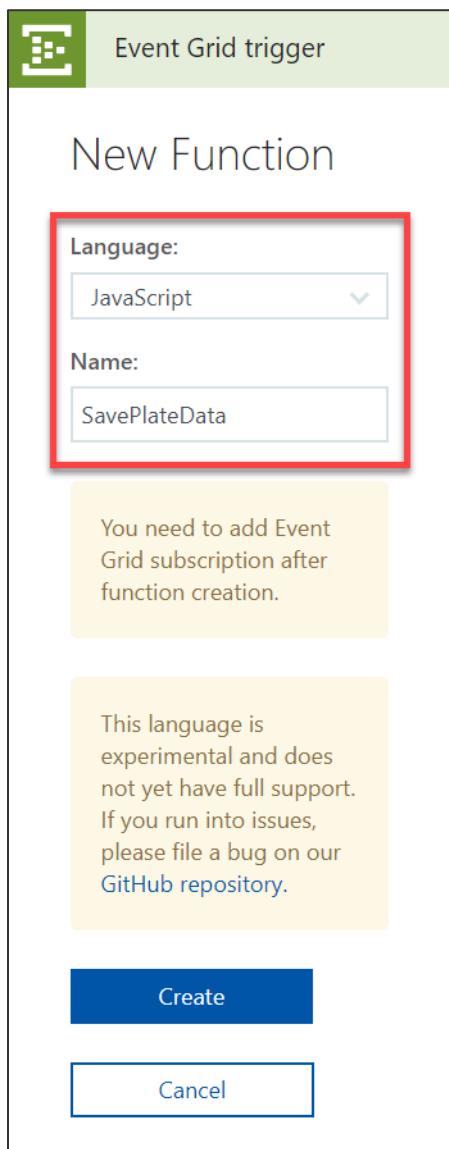
A function that will be run whenever an event grid receives a new event

C# JavaScript

5. In the New Function form, fill out the following properties:

- Language:** JavaScript

- b. **Name:** SavePlateData



6. Select **Create**.

7. Replace the code in the new SavePlateData function with the following:

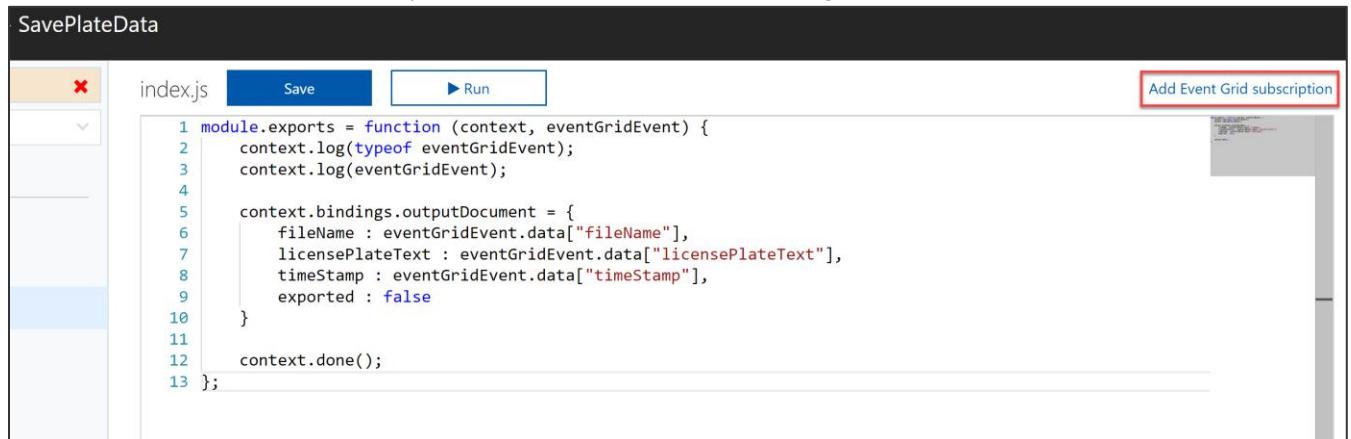
```
module.exports = function (context, eventGridEvent) {  
    context.log(typeof eventGridEvent);  
    context.log(eventGridEvent);  
  
    context.bindings.outputDocument = {  
        fileName : eventGridEvent.data["fileName"],  
        licensePlateText : eventGridEvent.data["licensePlateText"],  
        timeStamp : eventGridEvent.data["timeStamp"],  
        exported : false  
    };  
  
    context.done();  
};
```

8. Select **Save**.

Task 2: Add an Event Grid subscription to the SavePlateData function

In this task, you will add an Event Grid subscription to the SavePlateData function. This will ensure that the events sent to the Event Grid topic containing the savePlateData event type are routed to this function.

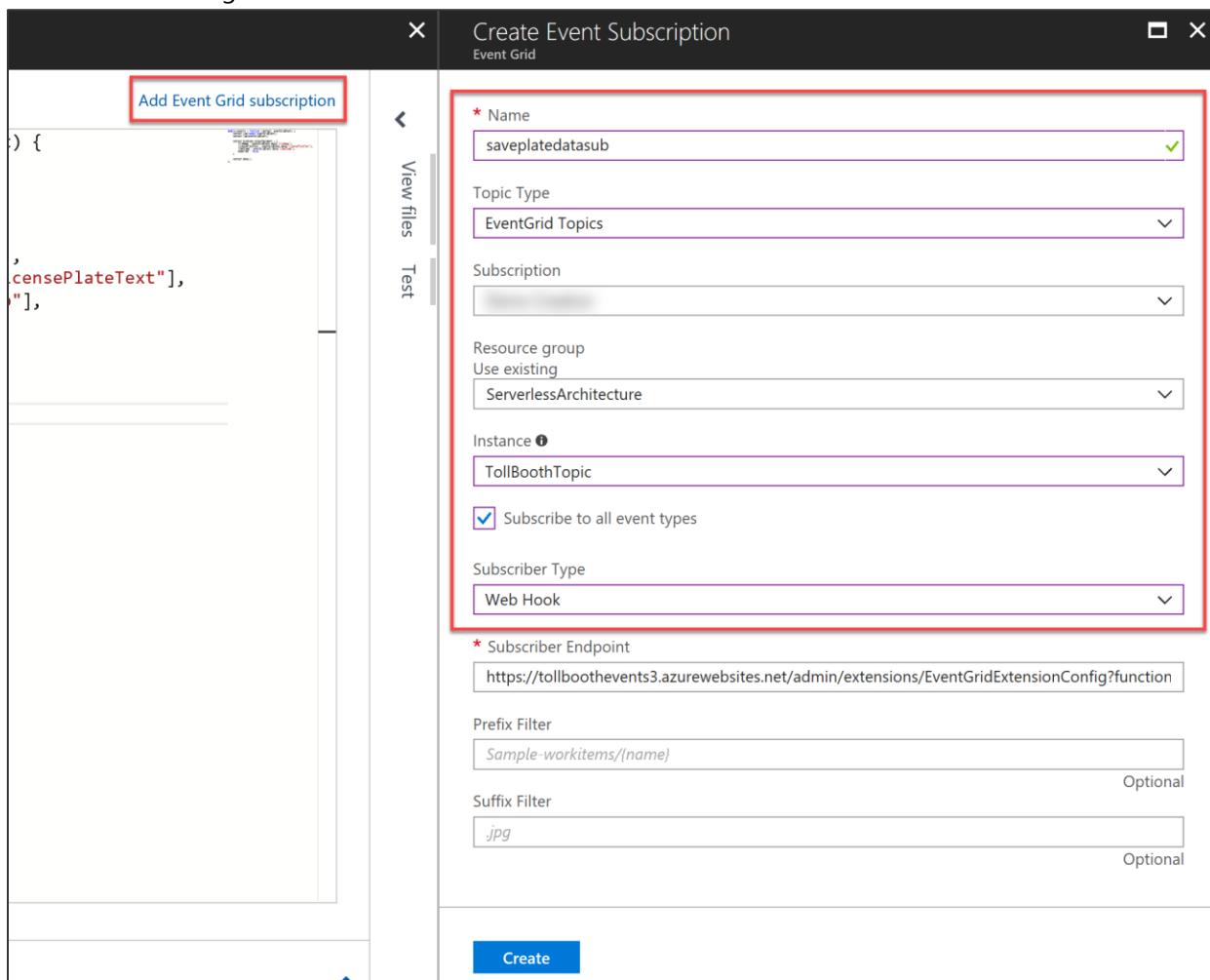
1. With the SavePlateData function open, select **Add Event Grid subscription**.



2. On the **Create Event Subscription** blade, specify the following configuration options:

- Name:** unique value for the App name (ensure the green check mark appears). Provide a name similar to `saveplatedatasub`.
- Select **Event Grid Topics** under **Topic Type**.
- Select your **subscription** and **ServerlessArchitecture** resource group.
- Select your Event Grid topic under **Instance**.
- Check **Subscribe to all event types**. You will enter a custom event type later.

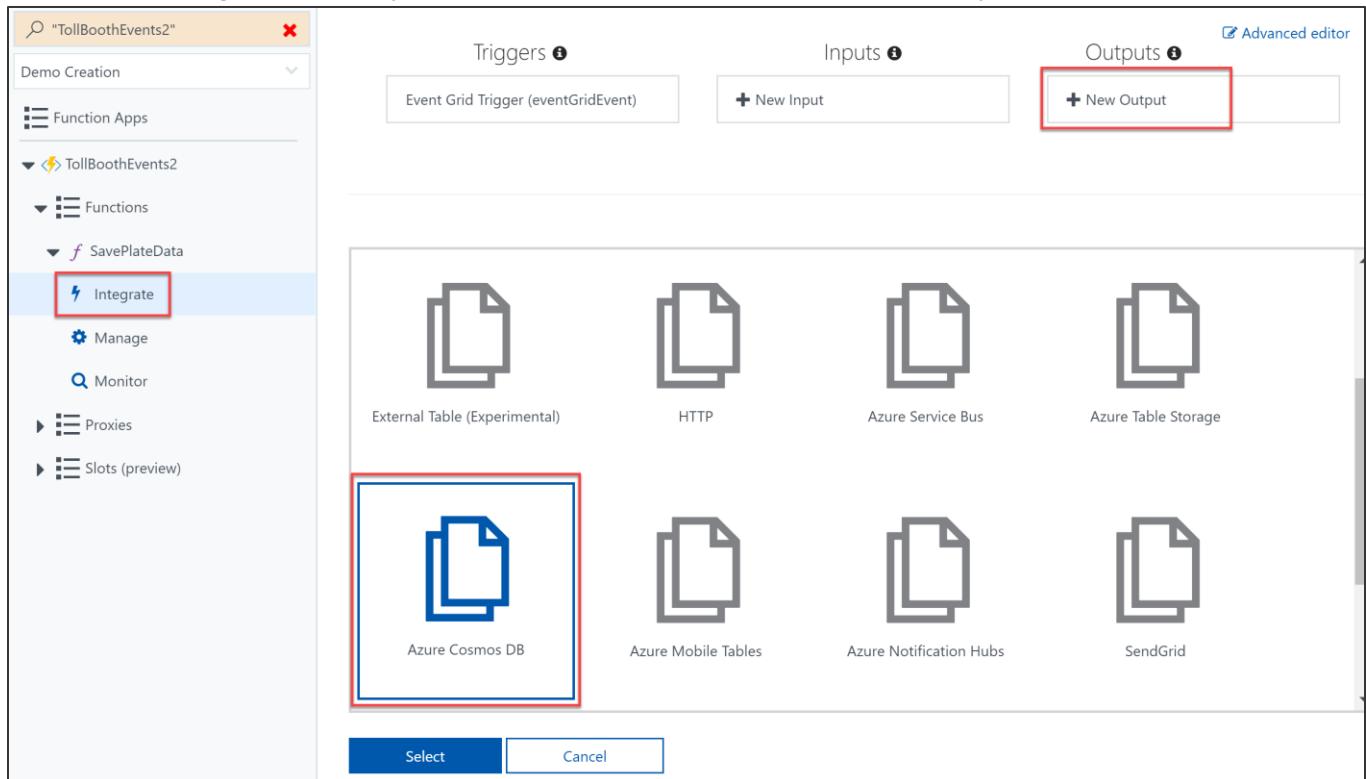
3. Leave the remaining fields at their default values and select **Create**.



Task 3: Add an Azure Cosmos DB output to the SavePlateData function

In this task, you will add an Azure Cosmos DB output binding to the SavePlateData function, enabling it to save its data to the Processed collection.

1. Expand the SavePlateData function in the left-hand menu, then select **Integrate**.
2. Select **+ New Output** under Outputs, select **Azure Cosmos DB** from the list of outputs, then select **Select**.

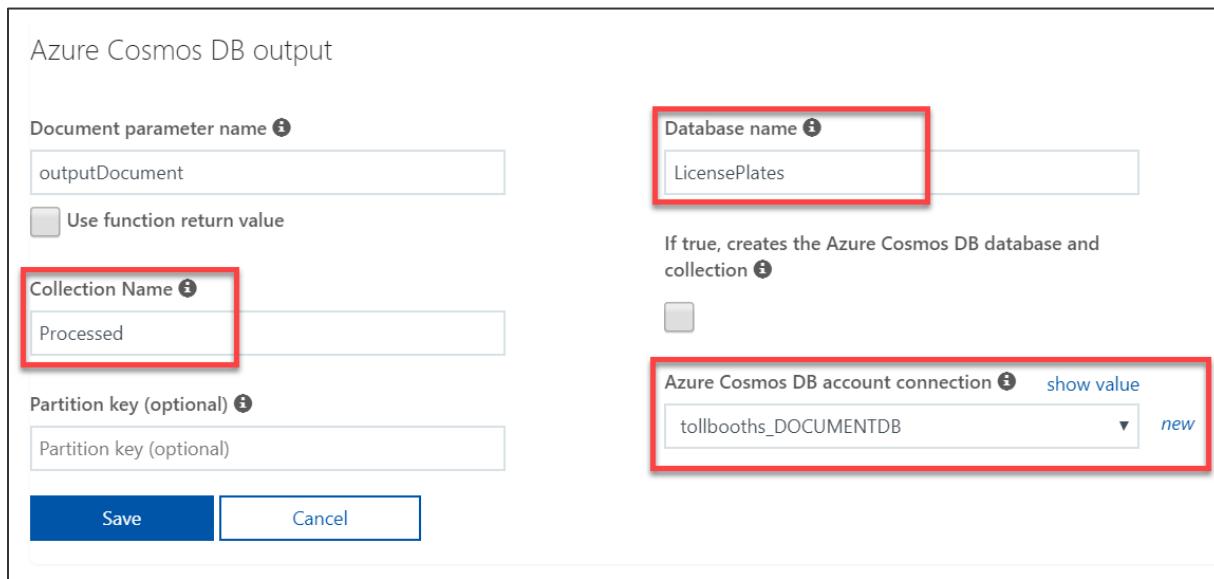


3. In the Azure Cosmos DB output form, select **new** next to the Azure Cosmos DB account connection field.



4. Select your Cosmos DB account from the list that appears.
5. Specify the following configuration options in the Azure Cosmos DB output form:
 - a. Enter **LicensePlates** into the **database name** field.

- b. Enter **Processed** into the **collection name** field.

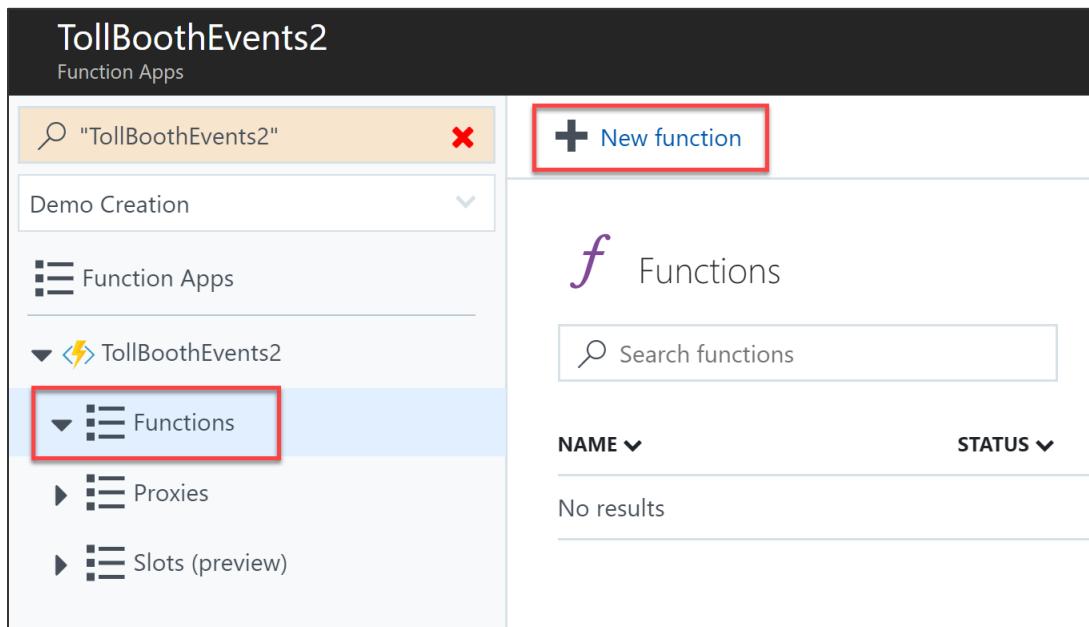


6. Select **Save**.

Task 4: Create function to save manual verification info to Azure Cosmos DB

In this task, you will create a new Node.js function triggered by Event Grid and outputs information about photos that need to be manually verified to Azure Cosmos DB.

1. Using a new tab or instance of your browser navigate to the Azure Management portal, <http://portal.azure.com>.
2. Open the **ServerlessArchitecture** resource group, then select the Azure Function App you created whose name ends with **Events**. If you did not use this naming convention, make sure you select the Function App that you did not deploy to in the previous exercise.
3. Select **Functions** in the left-hand menu. In the **Functions** blade, select **+ New Function**.



4. Enter **event grid** into the template search form, then select the **Event Grid trigger** template.

Choose a template below or [go to the quickstart](#)

XLanguage: AllScenario: All

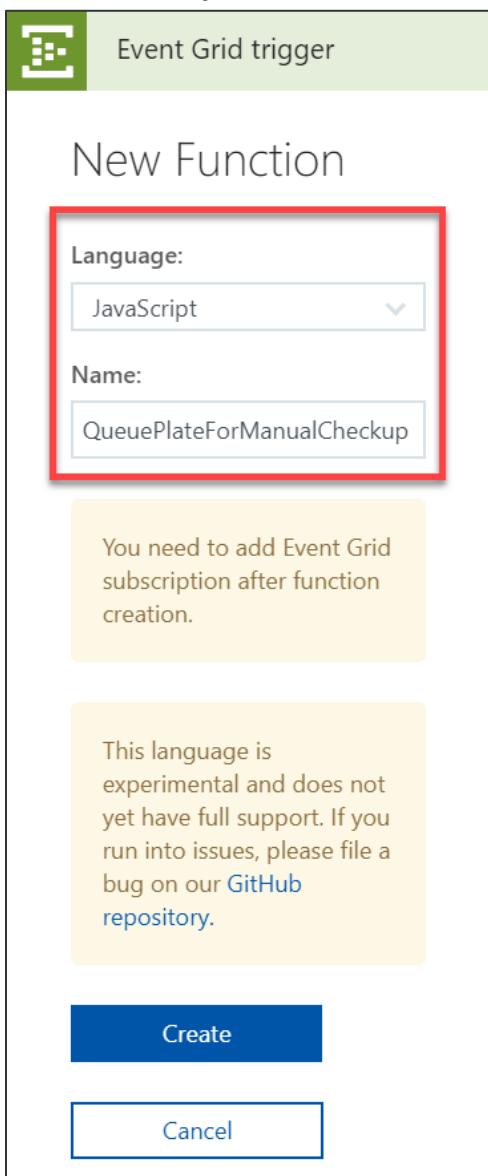
Event Grid trigger

A function that will be run whenever an event grid receives a new event

C#JavaScript

5. In the **New Function** form, fill out the following properties:

- Language:** JavaScript
- Name:** QueuePlateForManualCheckup



6. Select **Create**.

7. Replace the code in the new QueuePlateForManualCheckup function with the following:

```
module.exports = function (context, eventGridEvent) {
    context.log(typeof eventGridEvent);
    context.log(eventGridEvent);

    context.bindings.outputDocument = {
        fileName : eventGridEvent.data["fileName"],
        licensePlateText : "",
        timeStamp : eventGridEvent.data["timeStamp"],
        resolved : false
    }

    context.done();
};

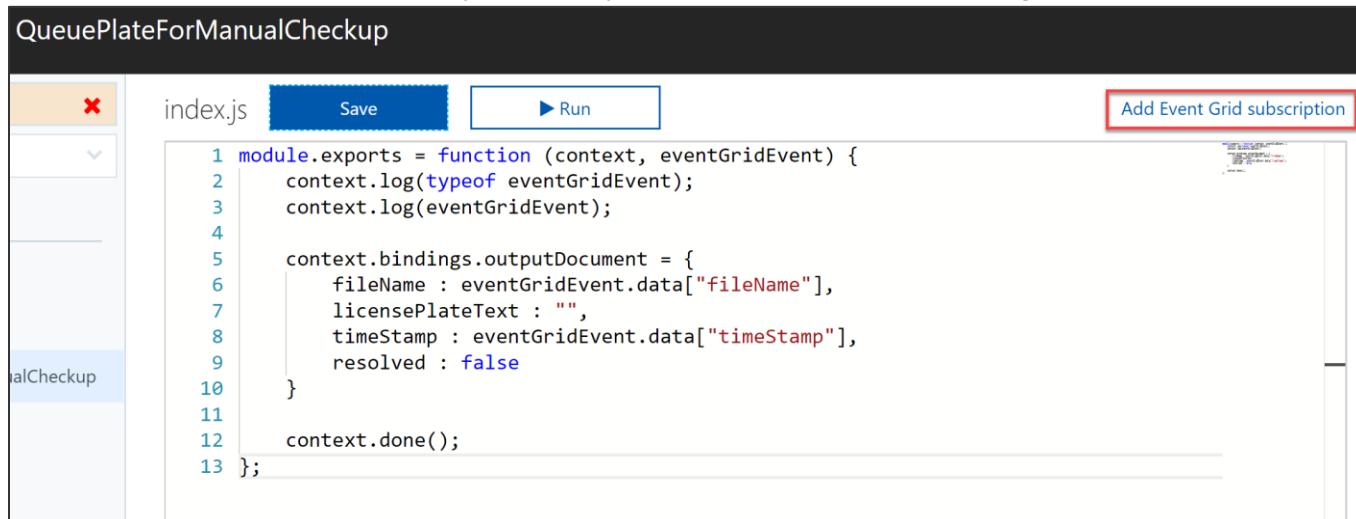

```

8. Select **Save**.

Task 5: Add an Event Grid subscription to the QueuePlateForManualCheckup function

In this task, you will add an Event Grid subscription to the QueuePlateForManualCheckup function. This will ensure that the events sent to the Event Grid topic containing the queuePlateForManualCheckup event type are routed to this function.

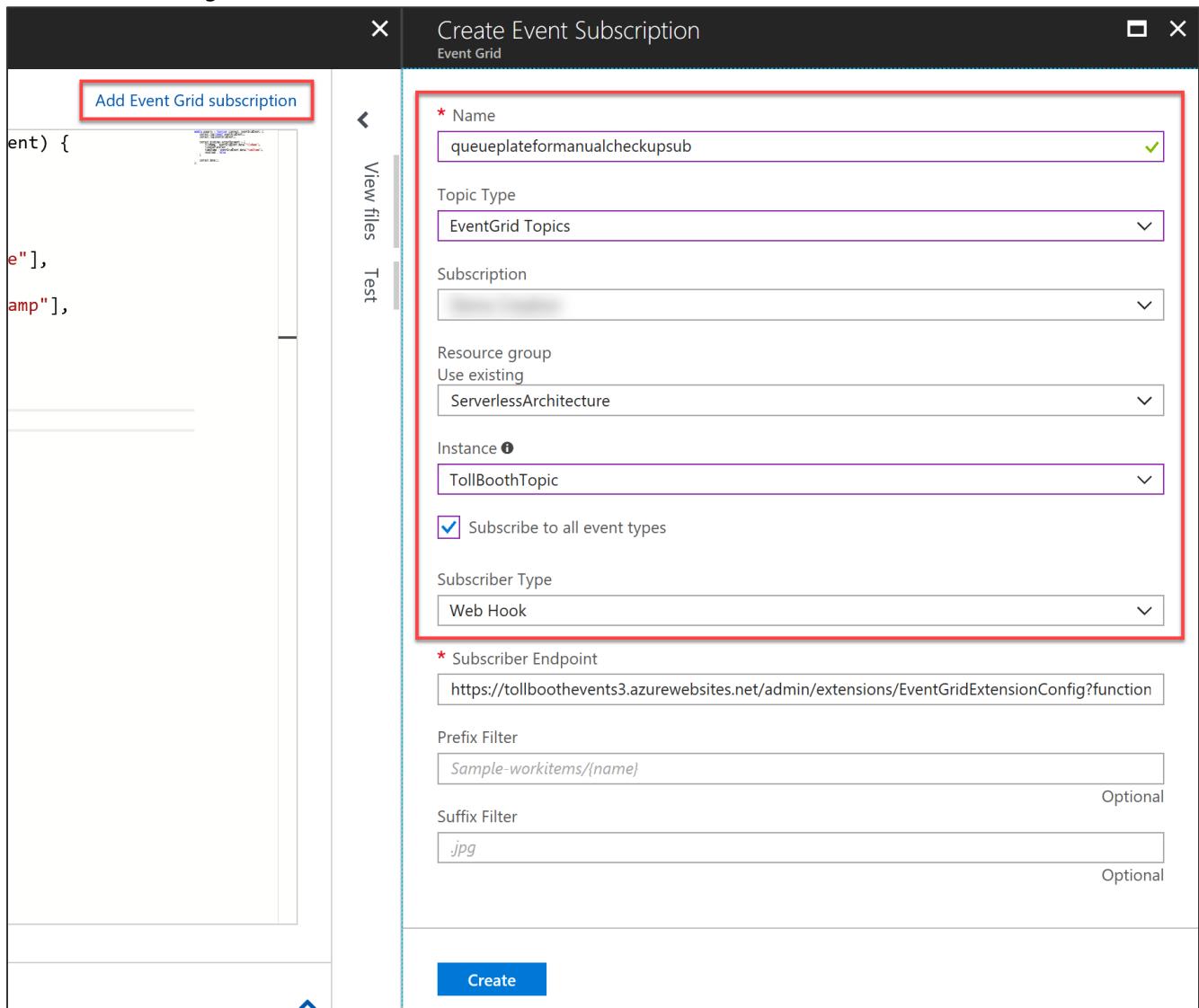
1. With the QueuePlateForManualCheckup function open, select **Add Event Grid subscription**.



2. On the **Create Event Subscription** blade, specify the following configuration options:

- Name:** unique value for the App name (ensure the green check mark appears). Provide a name similar to queueplateformanualcheckupsb.
- Select **Event Grid Topics** under Topic Type.
- Select your **subscription** and **ServerlessArchitecture** resource group.
- Select your Event Grid topic under **Instance**.
- Check **Subscribe to all event types**. You will enter a custom event type later.

3. Leave the remaining fields at their default values and select **Create**.

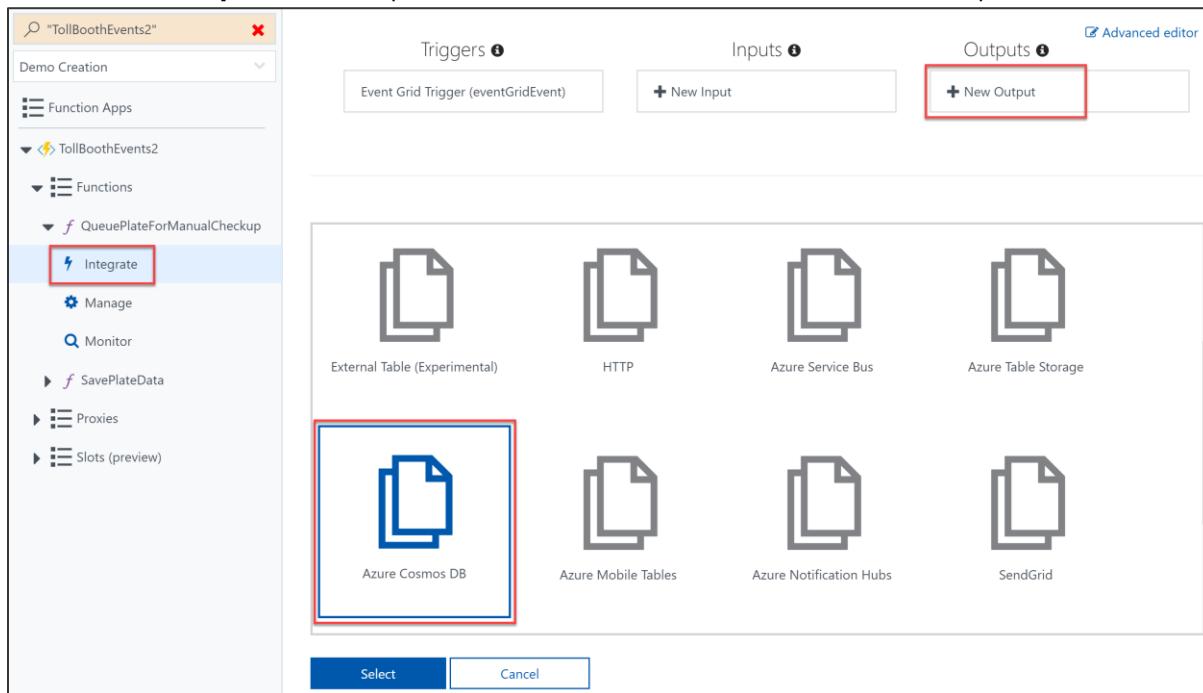


Task 6: Add an Azure Cosmos DB output to the QueuePlateForManualCheckup function

In this task, you will add an Azure Cosmos DB output binding to the QueuePlateForManualCheckup function, enabling it to save its data to the NeedsManualReview collection.

1. Expand the QueuePlateForManualCheckup function in the left-hand menu, then select **Integrate**.

2. Select **+ New Output** under Outputs, select **Azure Cosmos DB** from the list of outputs, then select **Select**.



3. Specify the following configuration options in the Azure Cosmos DB output form:

- Enter **LicensePlates** into the **database name** field.
- Enter **NeedsManualReview** into the **collection name** field.
- Select the **Azure Cosmos DB account connection** you created earlier.

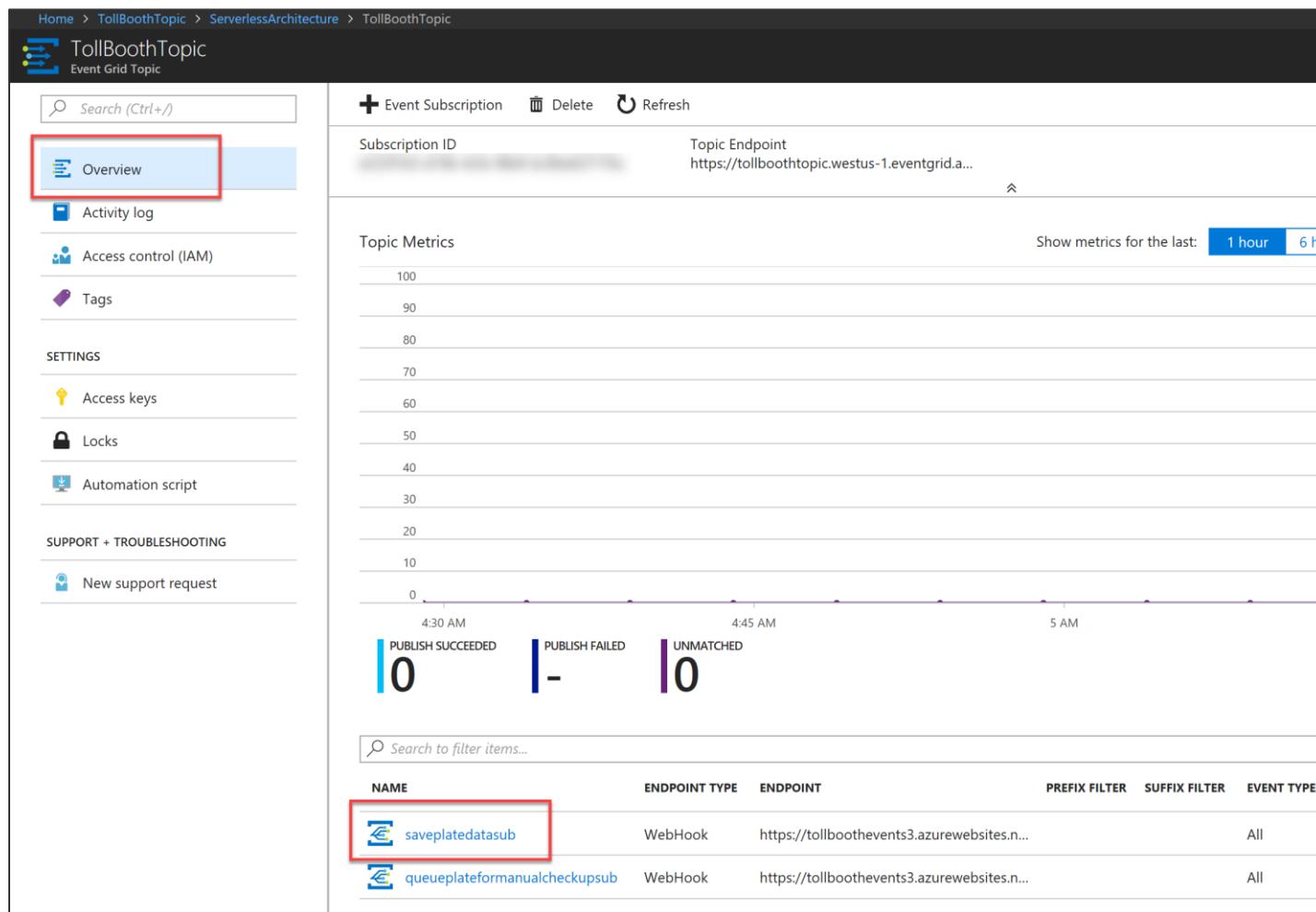
The screenshot shows the "Azure Cosmos DB output" configuration form. The "Collection Name" field (containing "NeedsManualReview") and the "Azure Cosmos DB account connection" dropdown (containing "tollbooths_DOCUMENTDB") are both highlighted with red boxes. The "Database name" field (containing "LicensePlates") is also highlighted with a red box. At the bottom are "Save" and "Cancel" buttons.

4. Select **Save**.

Task 7: Configure custom event types for the new Event Grid subscriptions

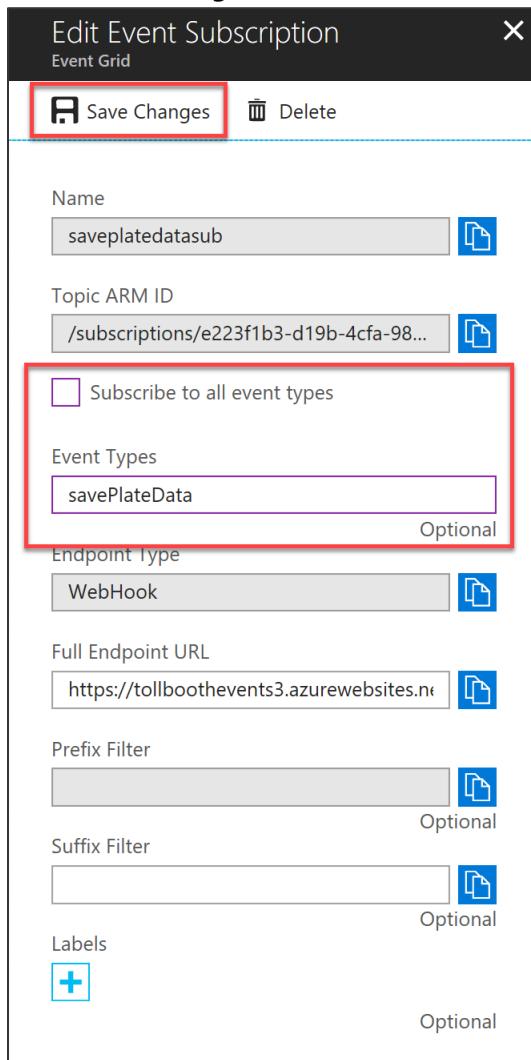
In this task, you will configure a custom event type for each new Event Grid subscription you created for your functions in the previous steps of this exercise. Currently the event types are set to All. We want to narrow this down to only the event types specified within the `SendToEventGrid` class in the TollBooth solution. This will ensure that all other event types are ignored by your functions.

1. Using a new tab or instance of your browser navigate to the Azure Management portal, <http://portal.azure.com>.
2. Open the **ServerlessArchitecture** resource group, then select the Event Grid Topic.
3. At the bottom of the **Overview** blade, you will see both Event Grid subscriptions created from the functions. Select **saveplatedatasub**.

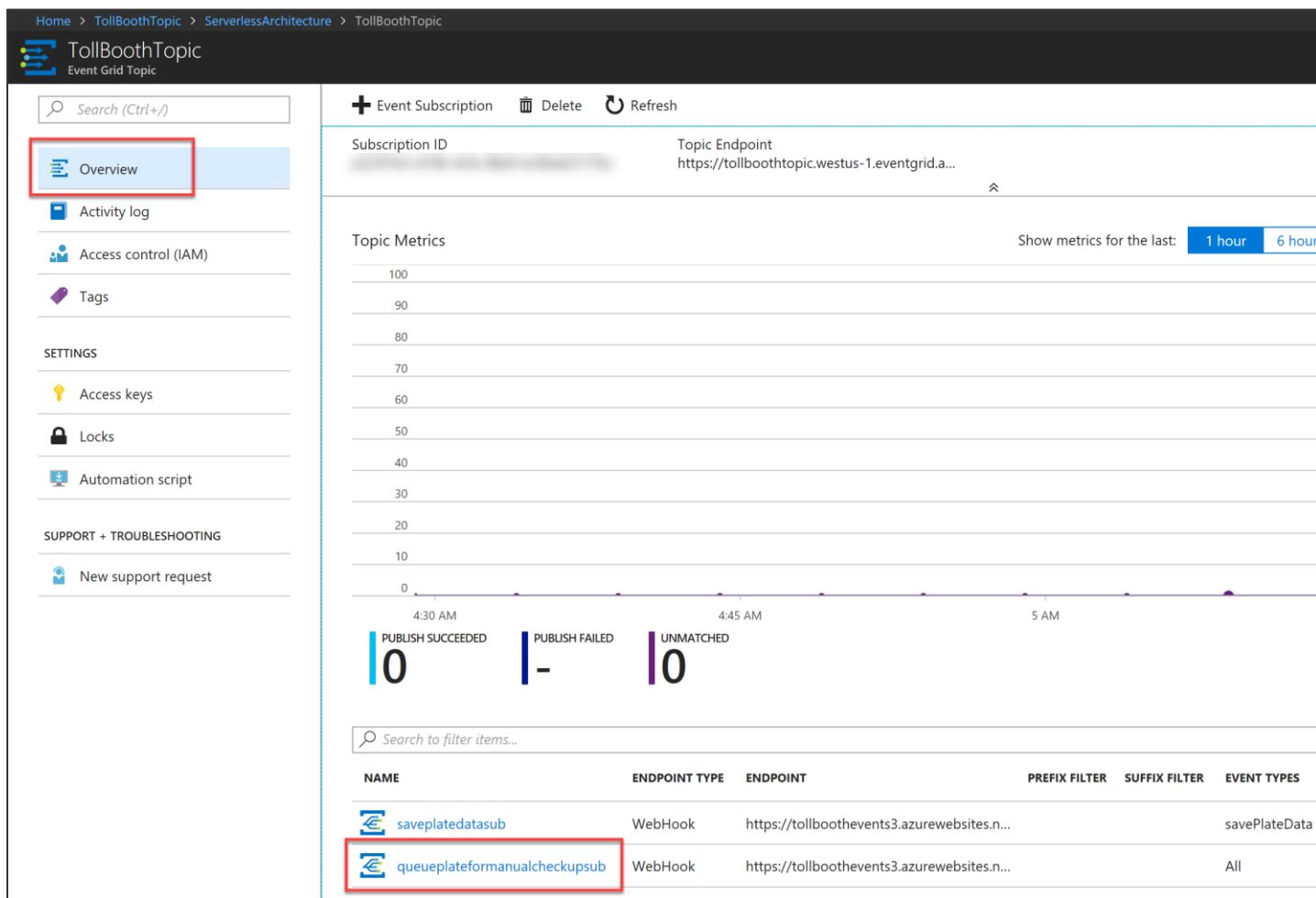


NAME	ENDPOINT TYPE	ENDPOINT	PREFIX FILTER	SUFFIX FILTER	EVENT TYPES
saveplatedatasub	WebHook	https://tollboothevents3.azurewebsites.n...			All
queueplateformmanualcheckupsub	WebHook	https://tollboothevents3.azurewebsites.n...			All

4. Uncheck **Subscribe to all event types**.
5. Enter **savePlateData** into the event types field. If you specified a different name in the `SendToEventGrid` class in the TollBooth solution, use that instead.

6. Select **Save Changes**.

7. Select the **queueplateformanualcheckupsub** Event Grid subscription at the bottom of the **Overview** blade.



Home > TollBoothTopic > ServerlessArchitecture > TollBoothTopic

TollBoothTopic
Event Grid Topic

Search (Ctrl+ /)

Overview (highlighted with a red box)

Event Subscription Delete Refresh

Subscription ID [REDACTED] Topic Endpoint <https://tollboothtopic.westus-1.eventgrid.azure.net/>

Topic Metrics

Show metrics for the last: 1 hour 6 hours

100
90
80
70
60
50
40
30
20
10
0

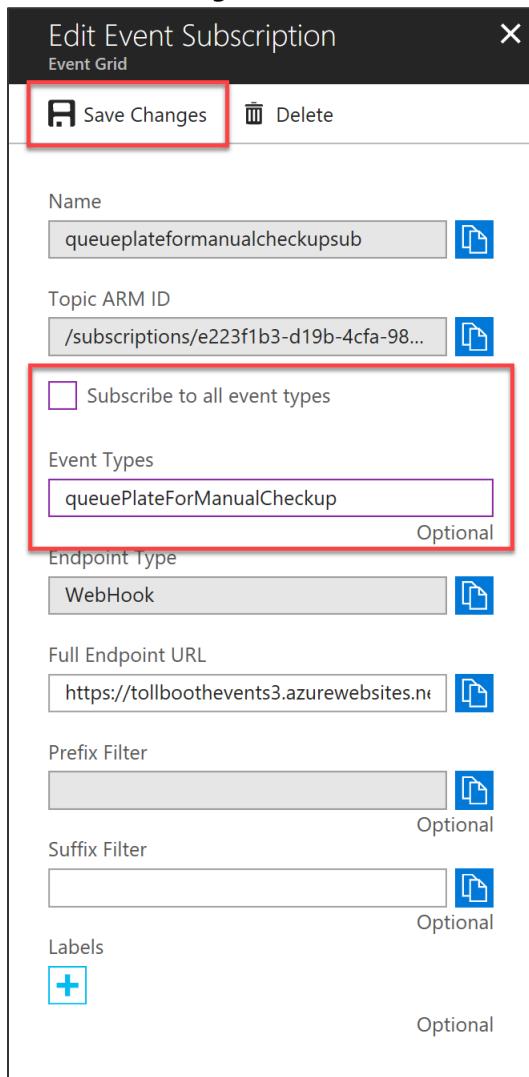
4:30 AM 4:45 AM 5 AM

PUBLISH SUCCEEDED: 0 PUBLISH FAILED: - UNMATCHED: 0

Search to filter items...

NAME	ENDPOINT TYPE	ENDPOINT	PREFIX FILTER	SUFFIX FILTER	EVENT TYPES
saveplatedatasub	WebHook	https://tollboothevents3.azurewebsites.net/	savePlateData		
queueplateformanualcheckupsub (highlighted with a red box)	WebHook	https://tollboothevents3.azurewebsites.net/		All	

8. Uncheck **Subscribe to all event types**.
9. Enter **queuePlateForManualCheckup** into the event types field. If you specified a different name in the `SendToEventGrid` class in the TollBooth solution, use that instead.

10. Select **Save Changes**.

Exercise 4: Monitor your functions with Application Insights

Duration: 45 minutes

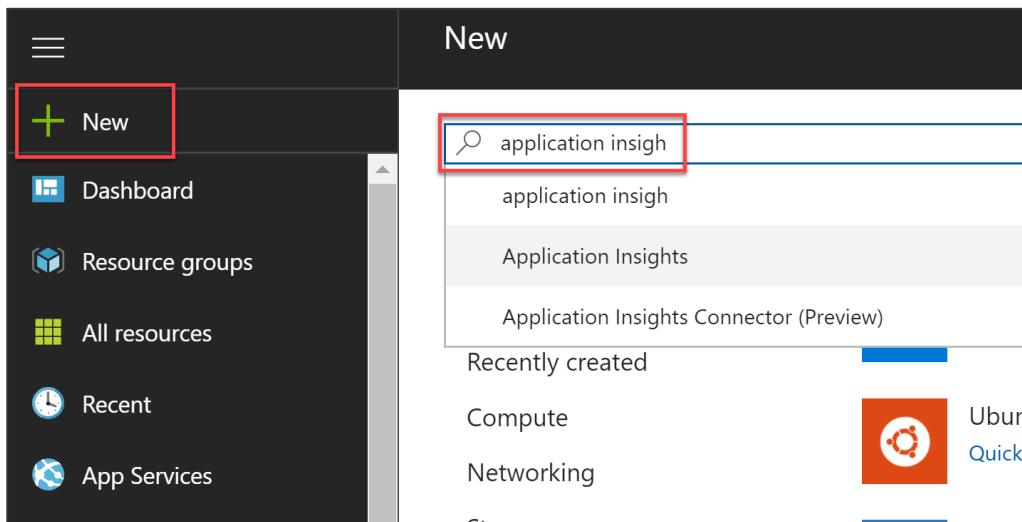
Application Insights can be integrated with Azure Function Apps to provide robust monitoring for your functions. In this exercise, you will provision a new Application Insights account and configure your Function Apps to send telemetry to it.

Help references

Monitor Azure Functions using Application Insights	https://docs.microsoft.com/en-us/azure/azure-functions/functions-monitoring
Live Metrics Stream: Monitor & Diagnose with 1-second latency	https://docs.microsoft.com/en-us/azure/application-insights/app-insights-live-stream

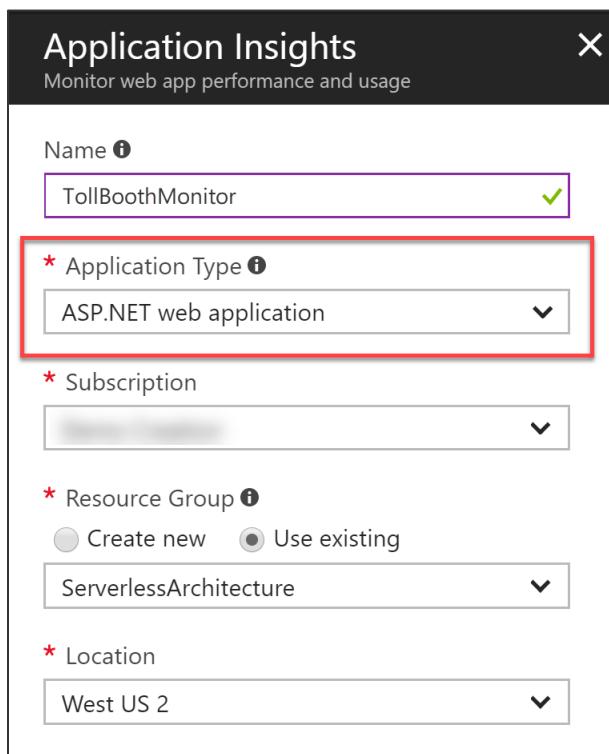
Task 1: Provision an Application Insights instance

1. Navigate to the Azure Management portal, <http://portal.azure.com>.
2. Select **+ New**, then type **application insights** into the search box on top. Select **Application Insights** from the results.



3. Select the **Create** button on the **Application Insights overview** blade.
4. On the **Create Function App** blade, specify the following configuration options:
 - a. **Name:** unique value for the App name (ensure the green check mark appears).
 - b. Select **ASP.NET web application** for the **Application Type**.
 - c. Specify the **Resource Group ServerlessArchitecture**.

- d. Select the same **location** as your Resource Group.

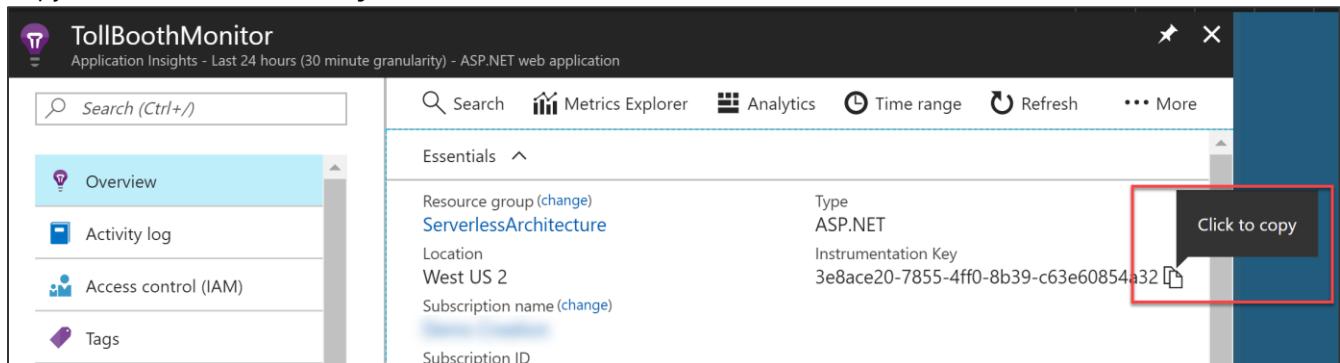


5. Select **Create**.

Task 2: Enable Application Insights integration in your Function Apps

Both of the Function Apps need to be updated with the Application Insights instrumentation key so they can start sending telemetry to your new instance.

1. After the Application Insights account has completed provisioning, open the instance by opening the **ServerlessArchitecture** resource group, and then selecting the **Application Insights** instance name.
2. Copy the **instrumentation key** from the Essentials section of the **Overview** blade.



3. Open the Azure Function App you created whose name ends with **FunctionApp**, or the name you specified for the Function App containing the `ProcessImage` function.

4. Select on **Application settings** on the **Overview** pane.

TollBoothFunctionApp
Function Apps

Overview Platform features

Demo Creation

Function Apps

TollBoothFunctionApp

- Functions
- Proxies
- Slots (preview)

Status: Running Subscription: [Subscription ID]

Configured features

- Function app settings
- Application settings**

5. Scroll down to the **Application settings** section. Use the **+ Add new setting** link and name the new setting **APPINSIGHTS_INSTRUMENTATIONKEY**. Paste the copied instrumentation key into its value field.

TollBoothFunctionApp
Function Apps

eventGridTopicKey: ISYDmBcv+qeycy6+yjk7r3/CghGPXvWZ1yp9j3c+dsQ=

cosmosDBEndPointUrl: https://tollbooth.documents.azure.com:443/

cosmosDBAuthorizationKey: 3xmUmc8neWZytbBGSTKhcVNaSvlWurNI2gbUU0nnnkgsXsk

cosmosDBDatabaseId: LicensePlates

cosmosDBCollectionId: Processed

exportCsvContainerName: export

blobStorageConnection: DefaultEndpointsProtocol=https;AccountName=tollbooth;A

APPINSIGHTS_INSTRUMENTATIONKEY: ceeaa095-ca90-4236-9ba4-1b89dcf86b1a

+ Add new setting

6. Select **Save**.

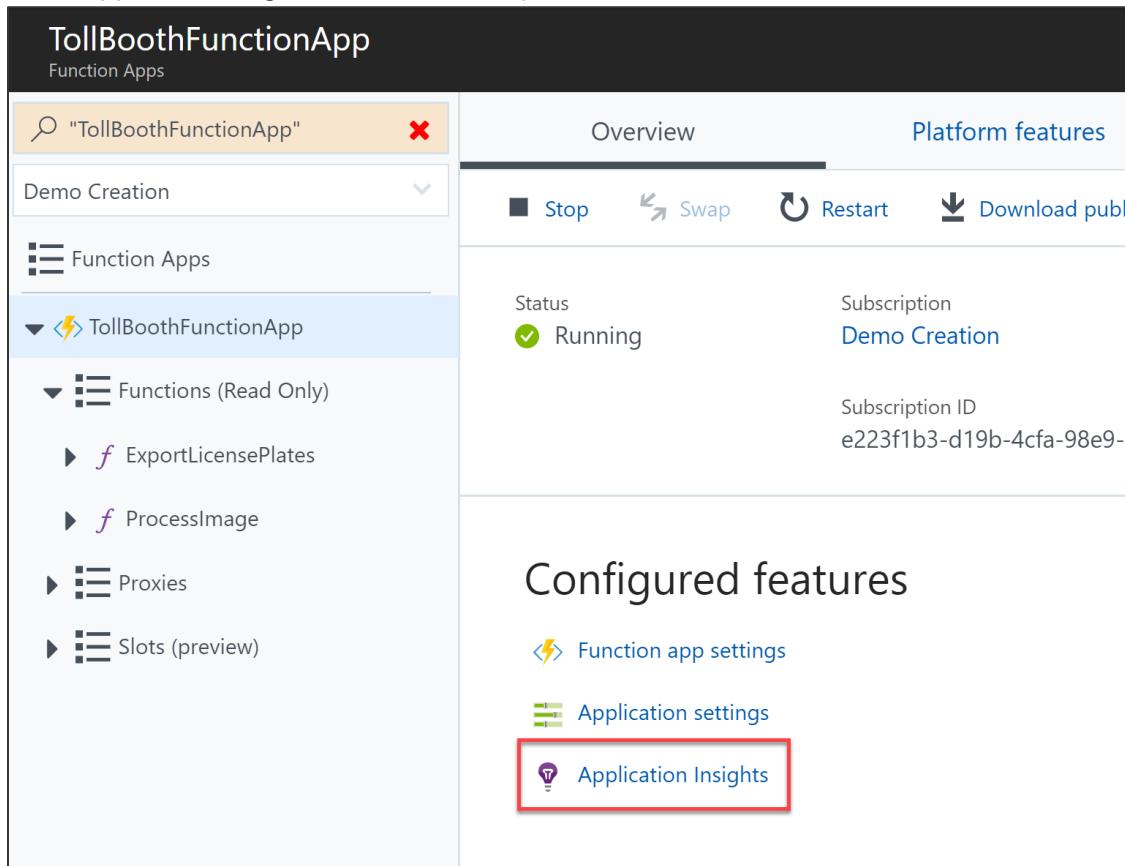


7. Follow the steps above to add the **APPINSIGHTS_INSTRUMENTATIONKEY** setting to the **other Function App**.

Task 3: Use the Live Metrics Stream to monitor functions in real time

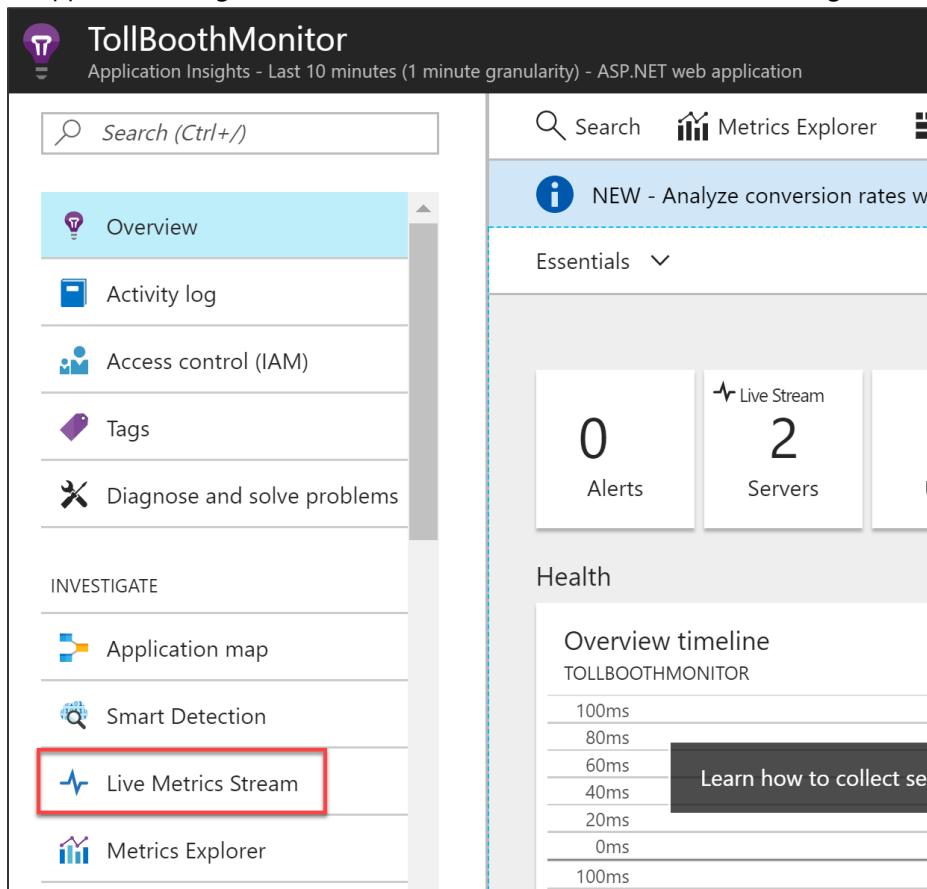
Now that Application Insights has been integrated into your Function Apps, you can use the Live Metrics Stream to see the functions' telemetry in real time.

1. Open the Azure Function App you created whose name ends with **FunctionApp**, or the name you specified for the Function App containing the ProcessImage function.
2. Select Application Insights on the Overview pane.



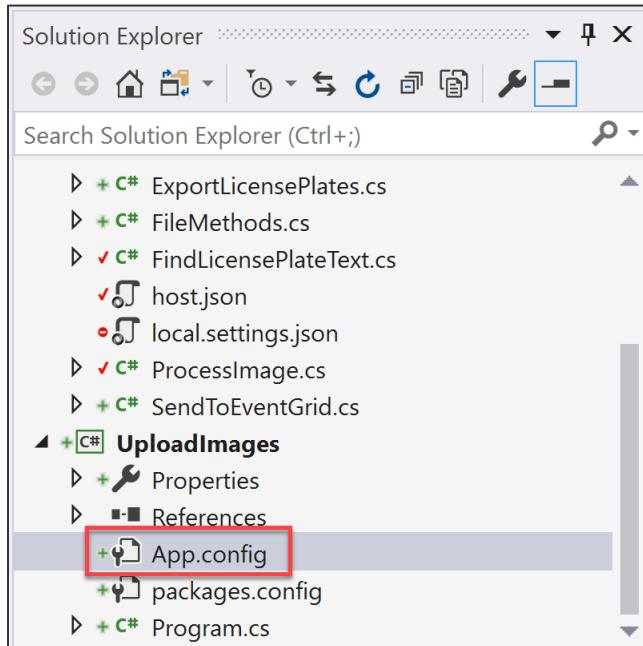
The screenshot shows the Azure Function App Overview page for the 'TollBoothFunctionApp'. The left sidebar shows the app name and a list of functions: 'ExportLicensePlates', 'ProcessImage', 'Proxies', and 'Slots (preview)'. The 'ProcessImage' function is highlighted with a purple icon. The main 'Overview' tab is selected, showing the status as 'Running' and the subscription ID as 'e223f1b3-d19b-4cfa-98e9-'. In the 'Configured features' section, the 'Application Insights' link is highlighted with a red box.

3. In Application Insights, select **Live Metrics Stream** underneath Investigate in the left-hand menu.



The screenshot shows the Microsoft Application Insights portal for the 'TollBoothMonitor' application. The left sidebar has a red box around the 'Live Metrics Stream' option under the 'INVESTIGATE' section. The main content area shows a summary with 0 Alerts and 2 Servers, and a timeline showing response times from 100ms down to 0ms.

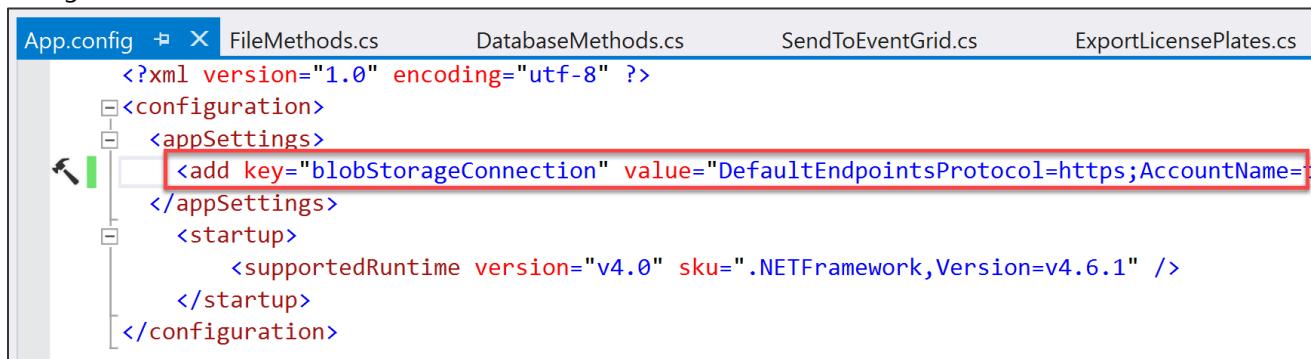
4. Leave the Live Metrics Stream open and go back to the starter app solution in Visual Studio.
5. Navigate to the **UploadImages** project using the Solution Explorer of Visual Studio.



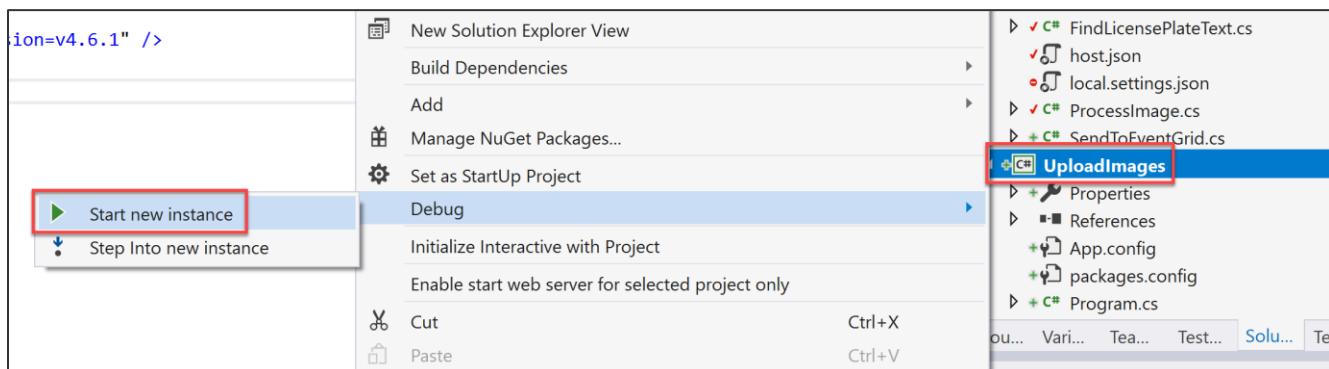
The screenshot shows the Microsoft Visual Studio Solution Explorer. The 'UploadImages' project is expanded, showing its structure. The 'App.config' file is highlighted with a red box.

6. Open **App.config**.

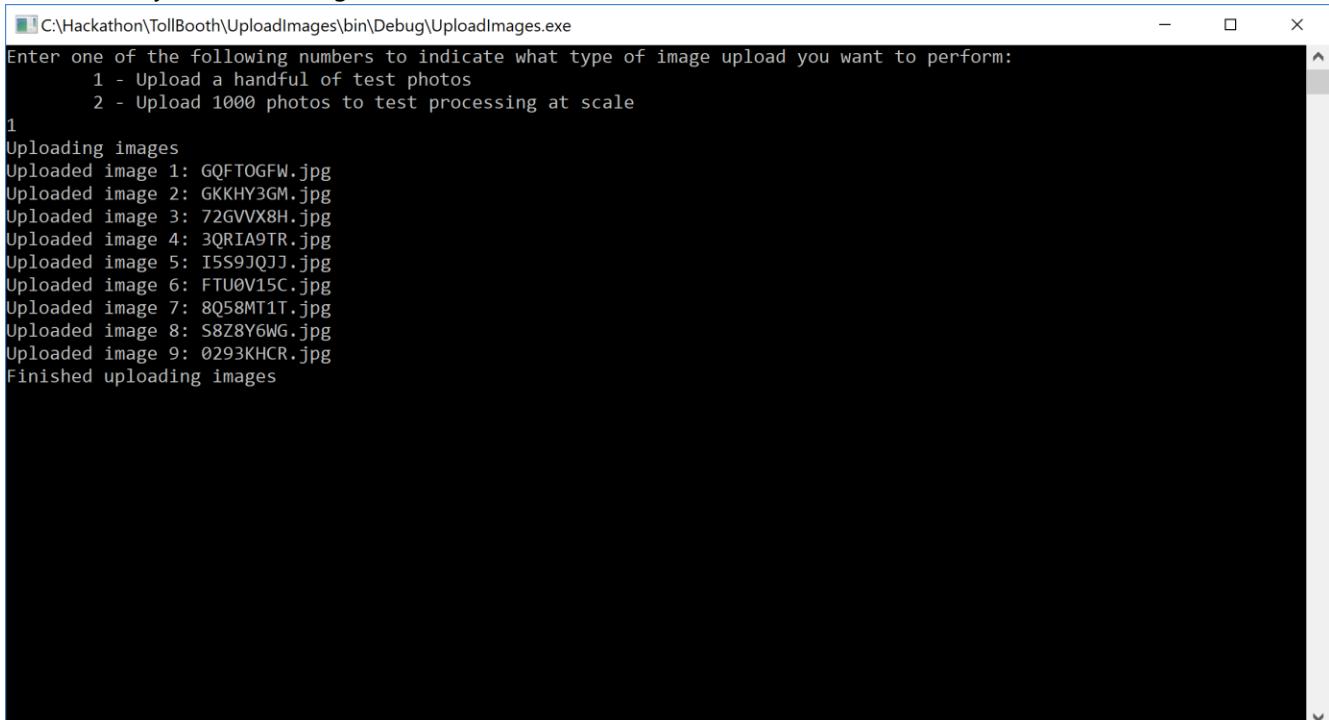
7. In App.config, update the **blobStorageConnection** appSetting value to the connection string for your Blob storage account.



8. Save your changes to the file.
 9. Right-click the UploadImages project in the Solution Explorer, then select **Start new instance** under the **Debug** menu.

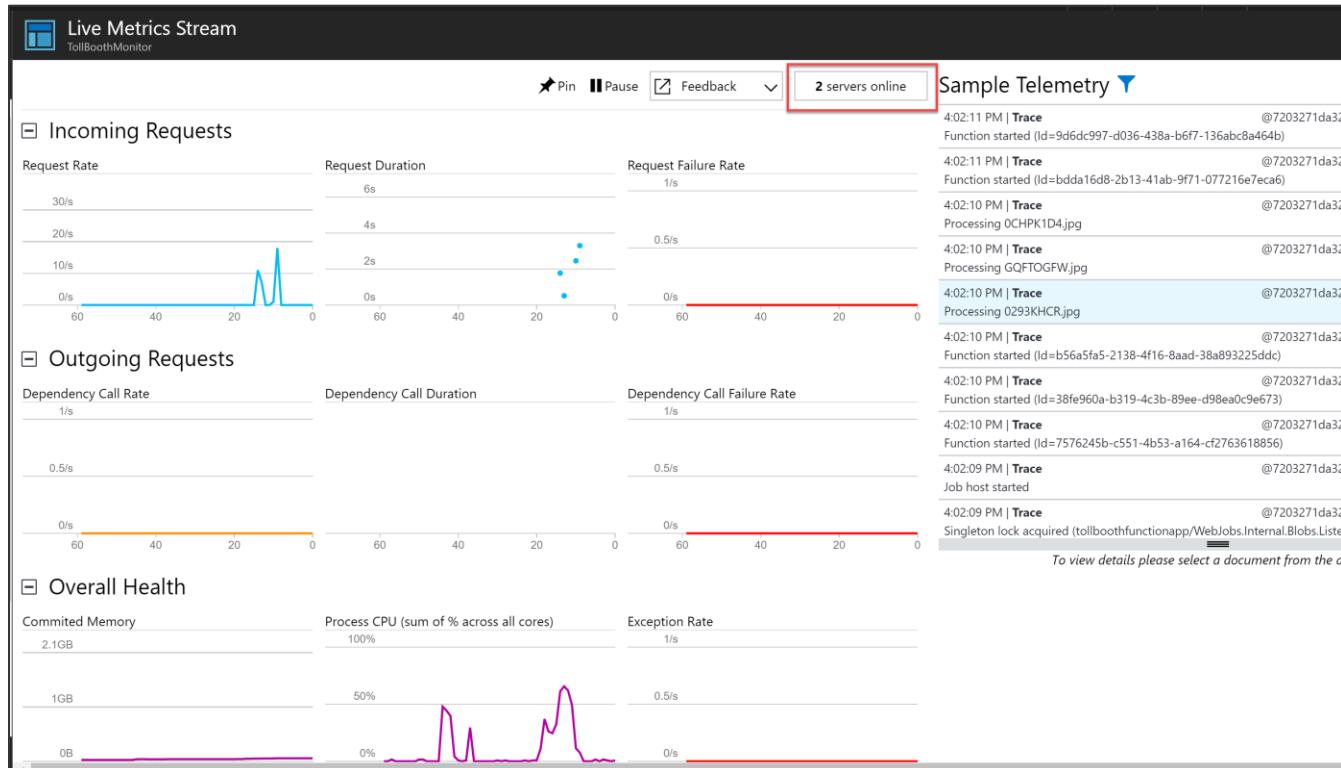


10. When the console window appears, enter **1** and press **ENTER**. This uploads a handful of car photos to the images container of your Blob storage account.

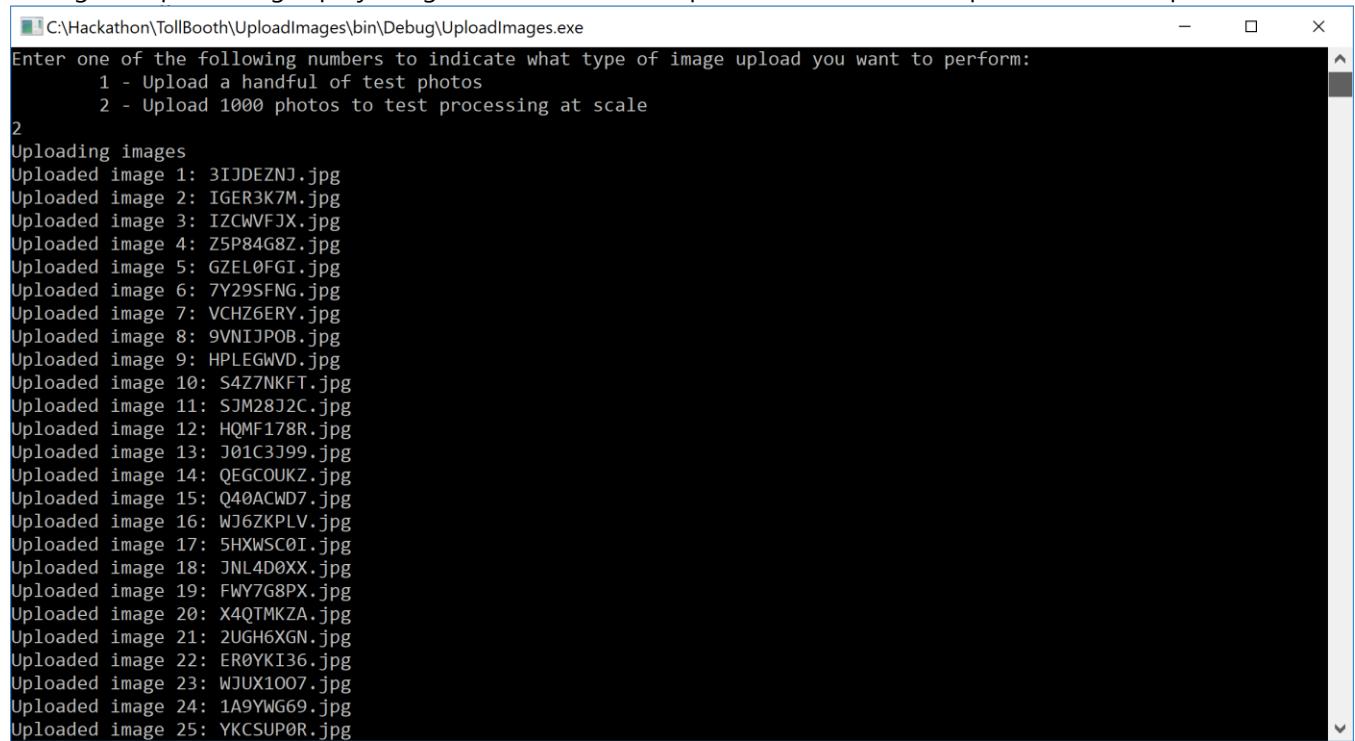


11. Switch back to your browser window with the Live Metrics Stream still open within Application Insights. You should start seeing new telemetry arrive, showing the number of servers online, the incoming request rate, CPU process amount, etc. You can select some of the sample telemetry in the list to the right to view output data.

Please note, since our function with the blob trigger is running on a Consumption plan, there can be up to a 10-minute delay in processing new blobs after the function app has gone idle. After the function app is running, blobs are processed immediately. To avoid this initial delay, consider one of the following options: Use an App Service plan with Always On enabled, or use another mechanism to trigger the blob processing, such as a queue message that contains the blob name or an Event Grid trigger. Sometimes, simply opening the function on the portal speeds up the process.



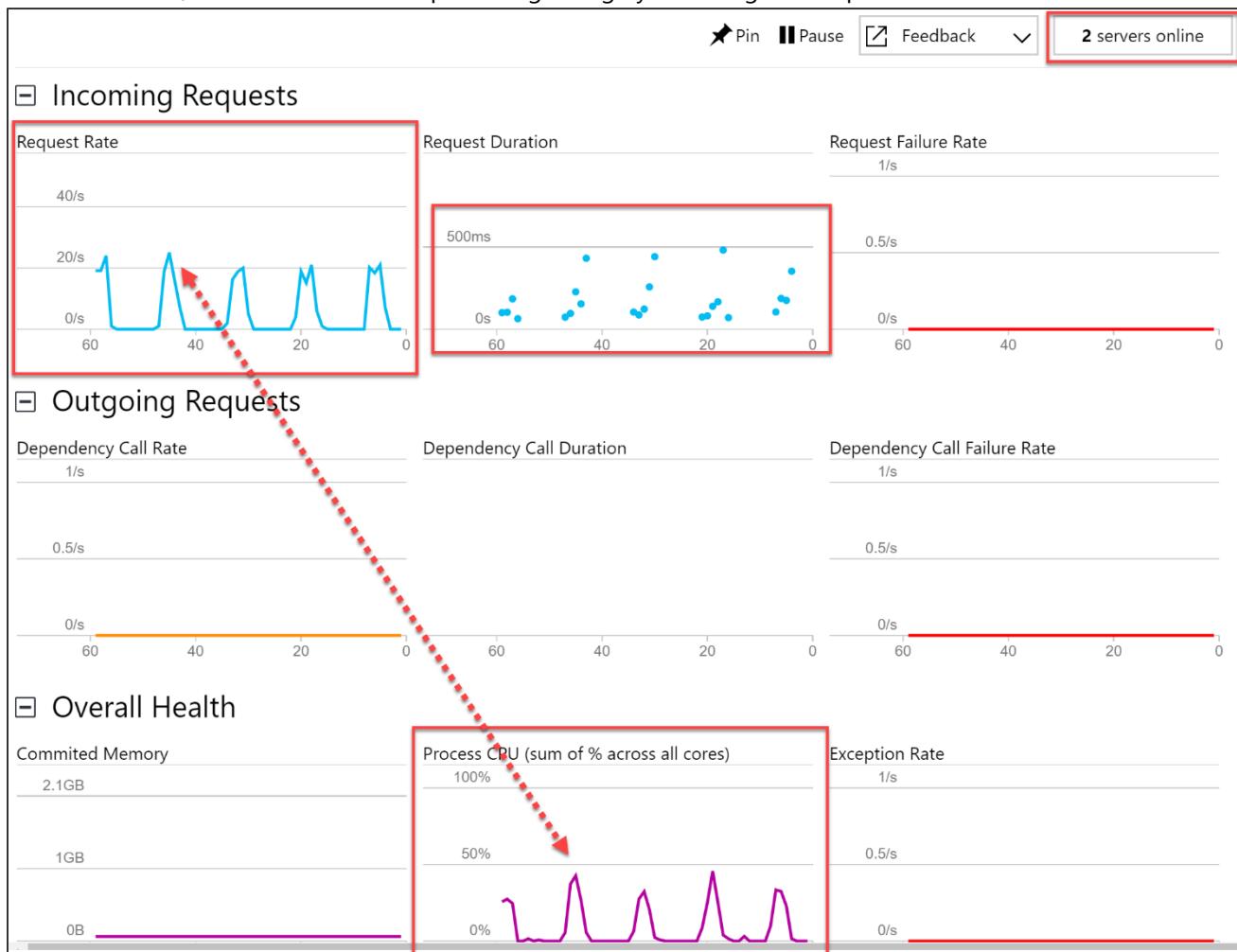
12. Leave the Live Metrics Stream window open once again, and close the console window for the image upload. Debug the UploadImages project again, then enter **2** and press **ENTER**. This will upload 1,000 new photos.



The screenshot shows a Windows command-line window with the following text:

```
C:\Hackathon\TollBooth\UploadImages\bin\Debug\UploadImages.exe
Enter one of the following numbers to indicate what type of image upload you want to perform:
  1 - Upload a handful of test photos
  2 - Upload 1000 photos to test processing at scale
2
Uploading images
Uploaded image 1: 3IJDEZNJ.jpg
Uploaded image 2: IGER3K7M.jpg
Uploaded image 3: IZCWVFJX.jpg
Uploaded image 4: Z5P84G8Z.jpg
Uploaded image 5: GZEL0FGI.jpg
Uploaded image 6: 7Y29SFNG.jpg
Uploaded image 7: VCHZ6ERY.jpg
Uploaded image 8: 9VNIJPOB.jpg
Uploaded image 9: HPLEGWVD.jpg
Uploaded image 10: S4Z7NKFT.jpg
Uploaded image 11: SJM28J2C.jpg
Uploaded image 12: HQMF178R.jpg
Uploaded image 13: J01C3J99.jpg
Uploaded image 14: QEGCOUKZ.jpg
Uploaded image 15: Q40ACWD7.jpg
Uploaded image 16: WJ6ZKPLV.jpg
Uploaded image 17: 5HXWSC0I.jpg
Uploaded image 18: JNL4D0XX.jpg
Uploaded image 19: FWY7G8PX.jpg
Uploaded image 20: X4QTMKZA.jpg
Uploaded image 21: 2UGH6XGN.jpg
Uploaded image 22: ER0YKI36.jpg
Uploaded image 23: WJUX1007.jpg
Uploaded image 24: 1A9YWG69.jpg
Uploaded image 25: YKCSUP0R.jpg
```

13. Switch back to the Live Metrics Stream window and observe the activity as the photos are uploaded. It is possible that the process will run so efficiently that no more than two servers will be allocated at a time. You should also notice things such as a steady cadence for the Request Rate monitor, the Request Duration hovering below ~500ms second, and the Process CPU percentage roughly matching the Request Rate.



14. After this has run for a while, close the image uploader console window once again, but leave the Live Metrics Stream window open.

Task 4: Observe your functions dynamically scaling when resource-constrained

In this task, you will change the Computer Vision API to the Free tier. This will limit the number of requests to the OCR service to 10 per minute. Once changed, run the UploadImages console app to upload 1,000 images again. The resiliency policy programmed into the `FindLicensePlateText.MakeOCRRequest` method of the `ProcessImage` function will begin exponentially backing off requests to the Computer Vision API, allowing it to recover and lift the rate limit. This intentional delay will greatly increase the function's response time, thus causing the Consumption plan's dynamic scaling to kick in, allocating several more servers. You will watch all of this happen in real time using the Live Metrics Stream view.

1. Open your Computer Vision API service by opening the **ServerlessArchitecture** resource group, and then selecting the **Computer Vision API** service name.

2. Select **Pricing tier** under Resource Management in the left-hand menu. Select the **F0 Free** pricing tier, then select **Select**.

The screenshot shows the Azure portal interface for the 'TollboothOCR' Cognitive Services resource. On the left, the 'Resource Management' blade is open, listing various management options. The 'Pricing tier' option is highlighted with a red box. On the right, a 'Choose your pricing tier' dialog box is displayed, comparing the 'F0 Free' tier (0.00 USD per call) with the 'S1 Standard' tier (1.00 USD per 1000 calls). The 'F0 Free' tier is selected and highlighted with a red box. A 'Select' button at the bottom of the dialog box is also highlighted with a red box.

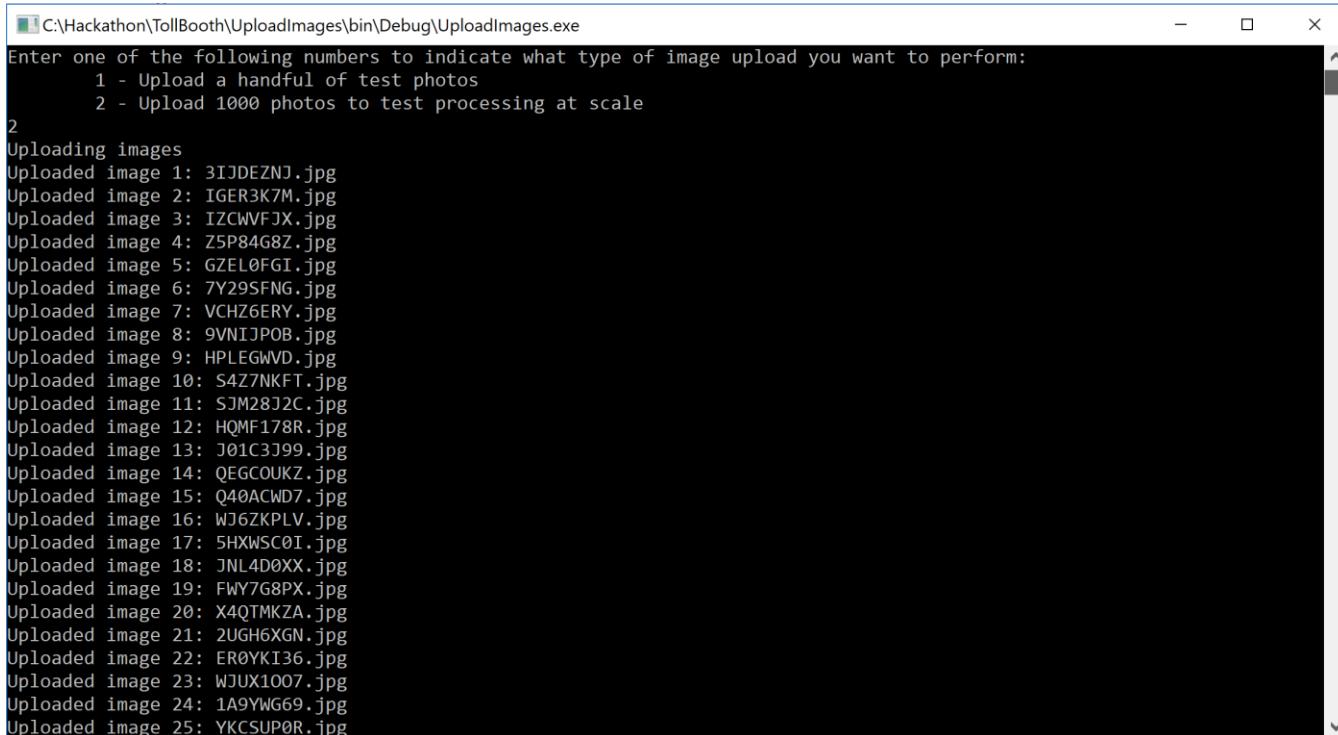
F0 Free		S1 Standard	
20	Calls per minute	10	Calls per second
5K	Calls per month		
	AnalyzeImage		AnalyzeImage
	TagImage		TagImage
	DescribeImage		DescribeImage
	OCR		OCR
	GetThumbnail		GetThumbnail

0.00 USD PER CALL (ESTIMATED)

1.00 STARTING USD/1000 CALLS (ESTIMATED)

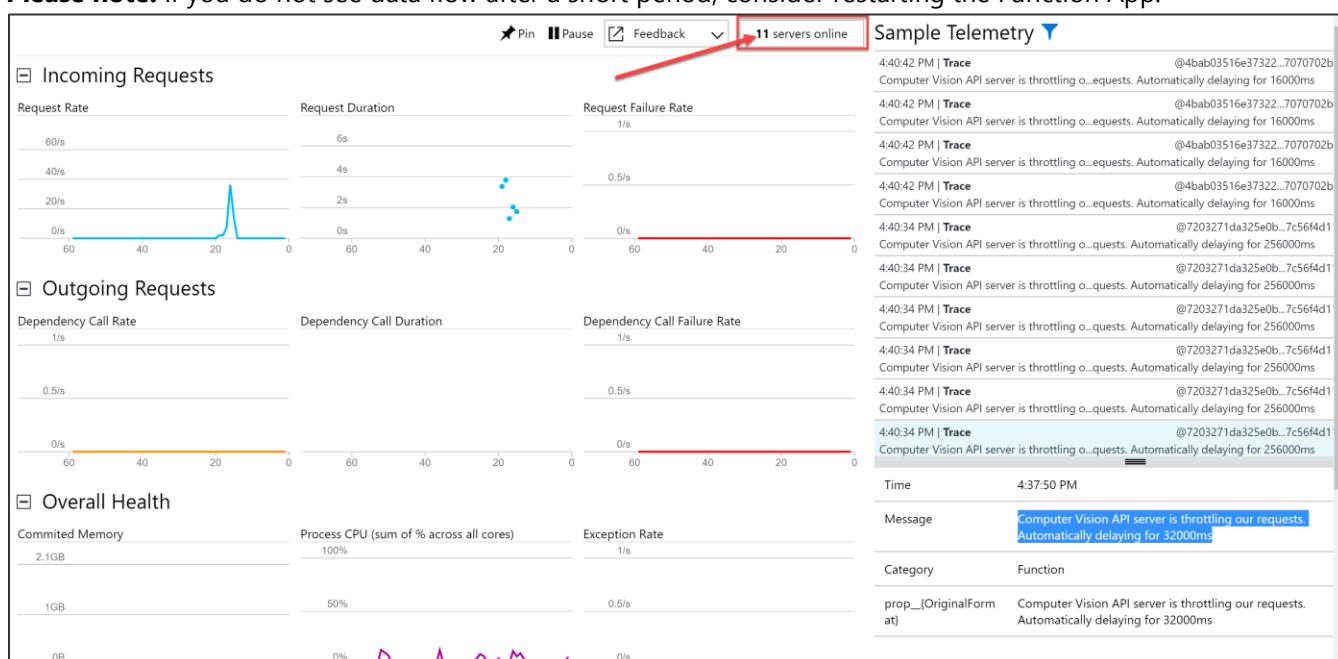
Select

3. Debug the UploadImages project again, then enter **2** and press **ENTER**. This will upload 1,000 new photos.



4. Switch back to the Live Metrics Stream window and observe the activity as the photos are uploaded. After running for a couple of minutes, you should start to notice a few things. The Request Duration will start to increase over time. As this happens, you should notice more servers being brought online. Each time a server is brought online, you should see a message in the Sample Telemetry stating that it is "Generating 2 job function(s)", followed by a Starting Host message. You should also see messages logged by the resiliency policy that the Computer Vision API server is throttling the requests. This is known by the response codes sent back from the service (429). A sample message is "Computer Vision API server is throttling our requests. Automatically delaying for 32000ms".

Please note: if you do not see data flow after a short period, consider restarting the Function App.



5. After this has run for some time, close the UploadImages console to stop uploading photos.
6. Navigate back to the **Computer Vision** API and set the pricing tier back to **S1 Standard**.

Exercise 5: Explore your data in Azure Cosmos DB

Duration: 15 minutes

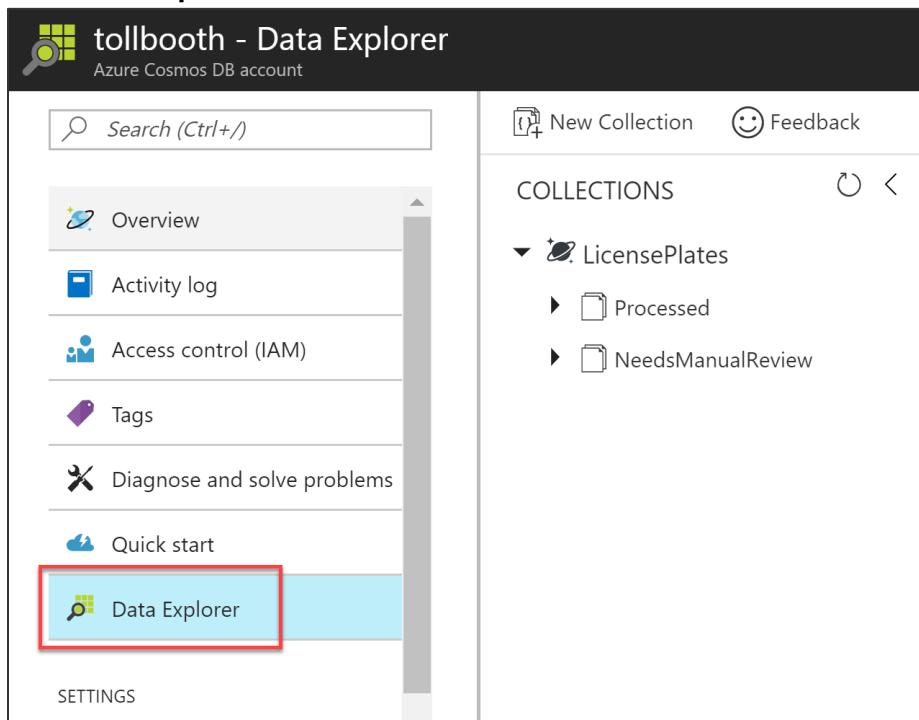
In this exercise, you will use the Azure Cosmos DB Data Explorer in the portal to view saved license plate data.

Help references

About Azure Cosmos DB	https://docs.microsoft.com/en-us/azure/cosmos-db/introduction
-----------------------	---

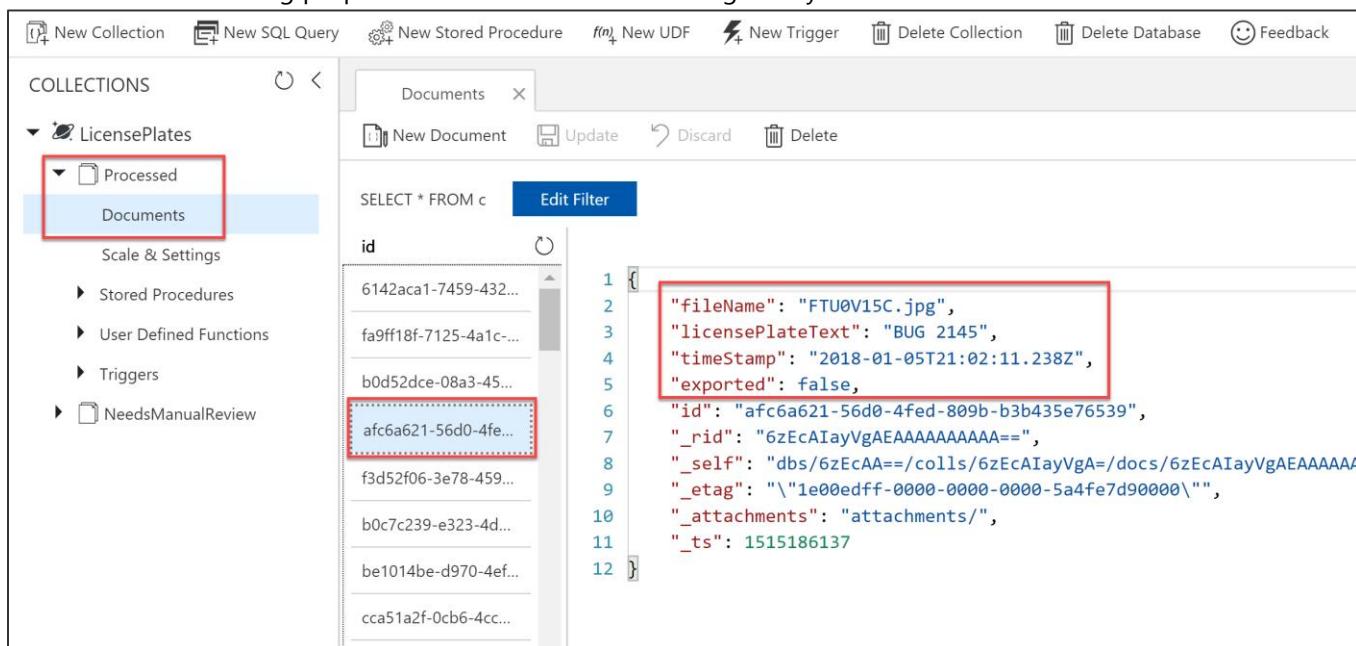
Task 1: Use the Azure Cosmos DB Data Explorer

1. Open your Azure Cosmos DB account by opening the **ServerlessArchitecture** resource group, and then selecting the **Azure Cosmos DB account** name.
2. Select **Data Explorer** from the left-hand menu.



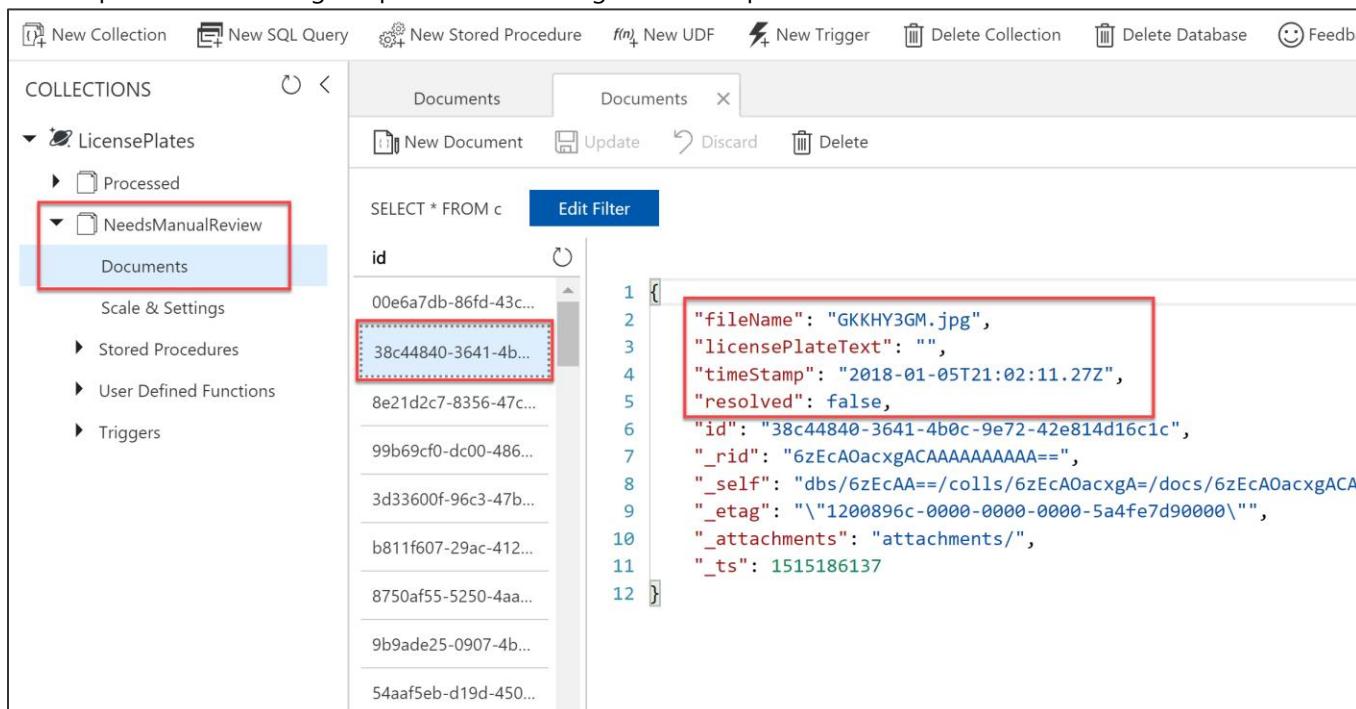
3. Expand the **Processed** collection, then select **Documents**. This will list each of the JSON documents added to the collection.

4. Select one of the documents to view its contents. The first four properties are ones that were added by your functions. The remaining properties are standard and are assigned by Cosmos DB.



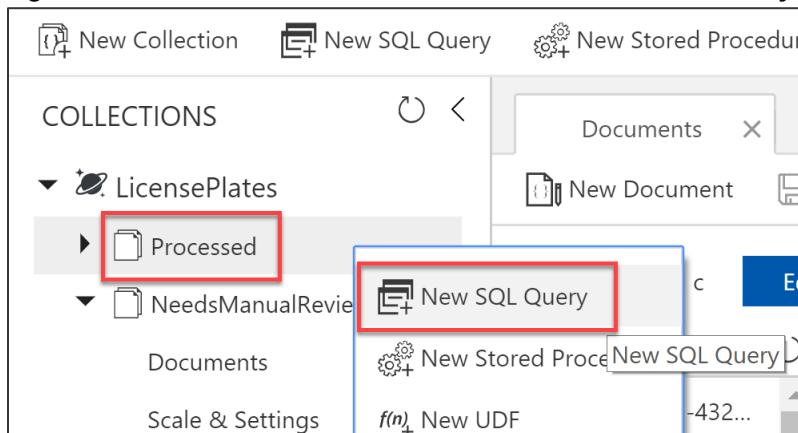
```
1 {
2   "fileName": "FTU0V15C.jpg",
3   "licensePlateText": "BUG 2145",
4   "timeStamp": "2018-01-05T21:02:11.238Z",
5   "exported": false,
6   "id": "afc6a621-56d0-4fed-809b-b3b435e76539",
7   "_rid": "6zEcAIayVgAEAAAAAAA==",
8   "_self": "dbs/6zEcAA==/colls/6zEcAIayVgA=/docs/6zEcAIayVgAEAAAAAA",
9   "_etag": "\"1e00edff-0000-0000-0000-5a4fe7d90000\"",
10  "_attachments": "attachments/",
11  "_ts": 1515186137
12 }
```

5. Expand the **NeedsManualReview** collection, then select **Documents**.
6. Select one of the documents to view its contents. Notice that the filename is provided, as well as a property named "resolved". While this is out of scope for this lab, those properties can be used together to provide a manual process for viewing the photo and entering the license plate.



```
1 {
2   "fileName": "GKKHY3GM.jpg",
3   "licensePlateText": "",
4   "timeStamp": "2018-01-05T21:02:11.27Z",
5   "resolved": false,
6   "id": "38c44840-3641-4b0c-9e72-42e814d16c1c",
7   "_rid": "6zEcAOacxgACAAAAAAA==",
8   "_self": "dbs/6zEcAA==/colls/6zEcAOacxgA=/docs/6zEcAOacxgACAA",
9   "_etag": "\"1200896c-0000-0000-0000-5a4fe7d90000\"",
10  "_attachments": "attachments/",
11  "_ts": 1515186137
12 }
```

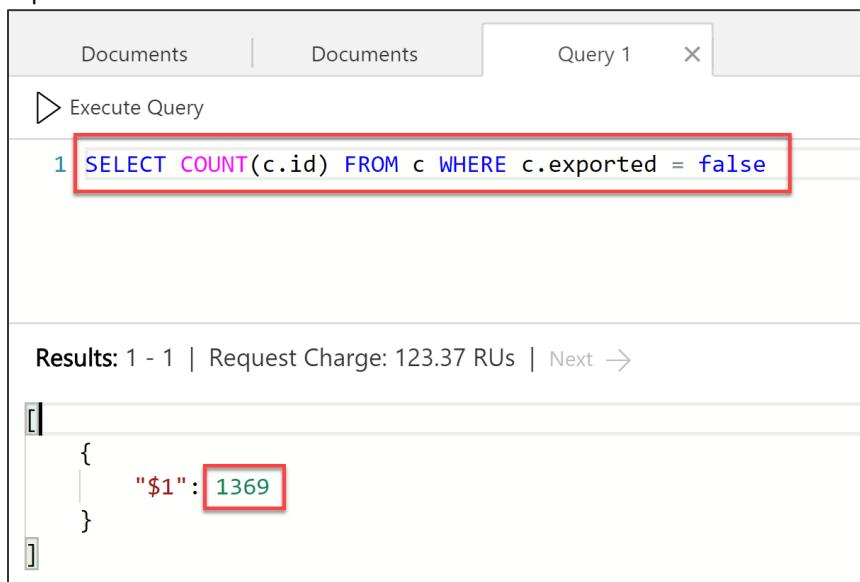
7. Right-click on the **Processed** collection and select **New SQL Query**.



8. Modify the SQL query to count the number of processed documents that have not been exported:

```
SELECT COUNT(c.id) FROM c WHERE c.exported = false
```

9. Execute the query and observe the results. In our case, we have 1,369 processed documents that need to be exported.



Exercise 6: Create the data export workflow

Duration: 30 minutes

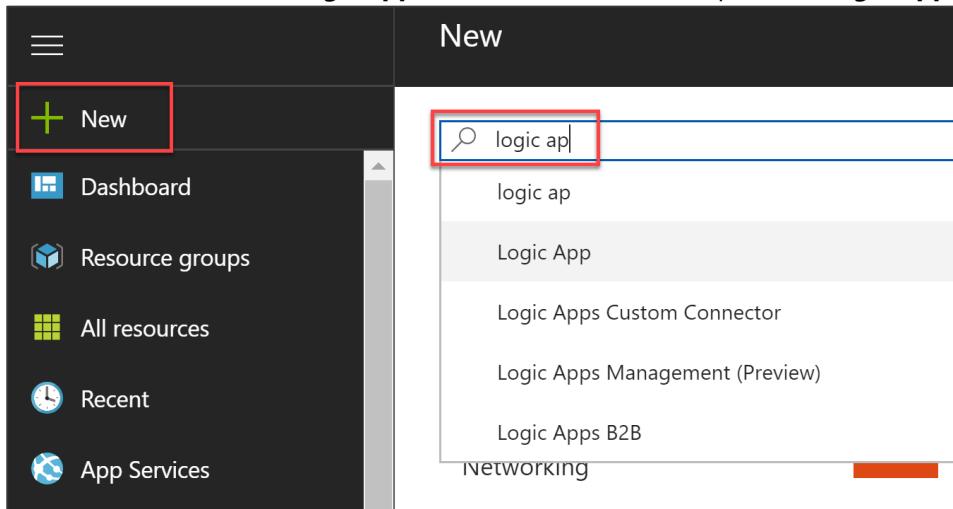
In this exercise, you create a new Logic App for your data export workflow. This Logic App will execute periodically and call your ExportLicensePlates function, then conditionally send an email if there were no records to export.

Help references

What are Logic Apps?	https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-what-are-logic-apps
Call Azure Functions from logic apps	https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-azure-functions#call-azure-functions-from-logic-apps

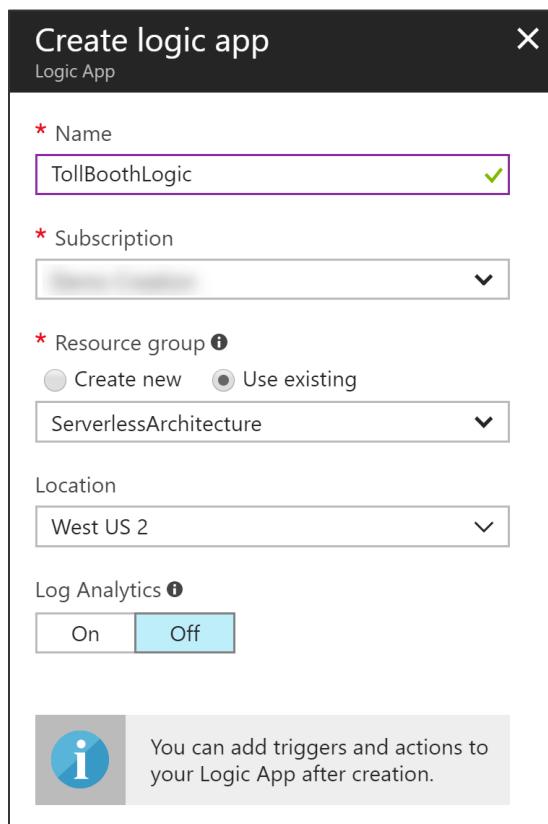
Task 1: Create the Logic App

1. Navigate to the Azure Management portal, <http://portal.azure.com>.
2. Select **+ New**, then enter **logic app** into the search box on top. Select **Logic App** from the results.

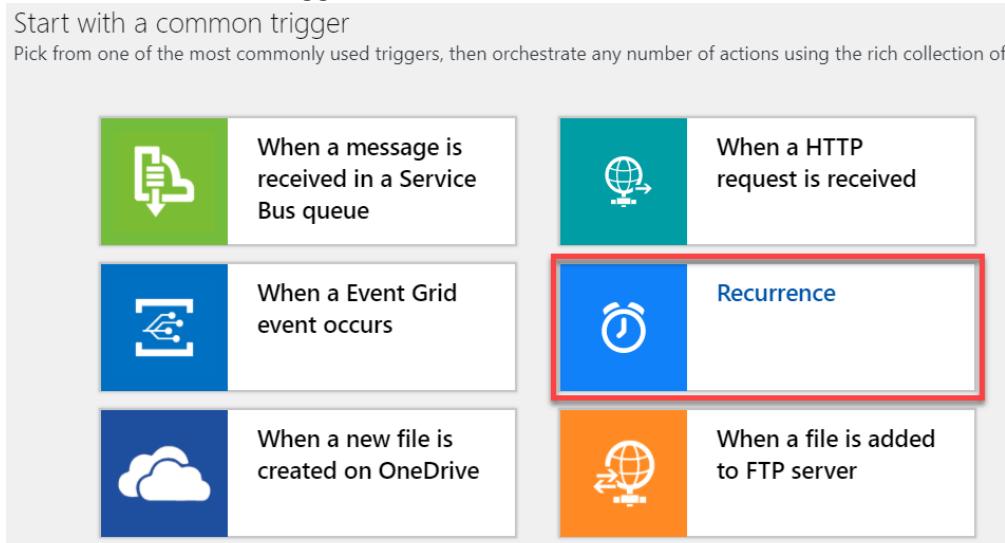


3. Select the **Create** button on the Logic App overview blade.
4. On the **Create Logic App** blade, specify the following configuration options:
 - a. **Name:** unique value for the App name (ensure the green check mark appears).
 - b. Specify the **Resource Group ServerlessArchitecture**.

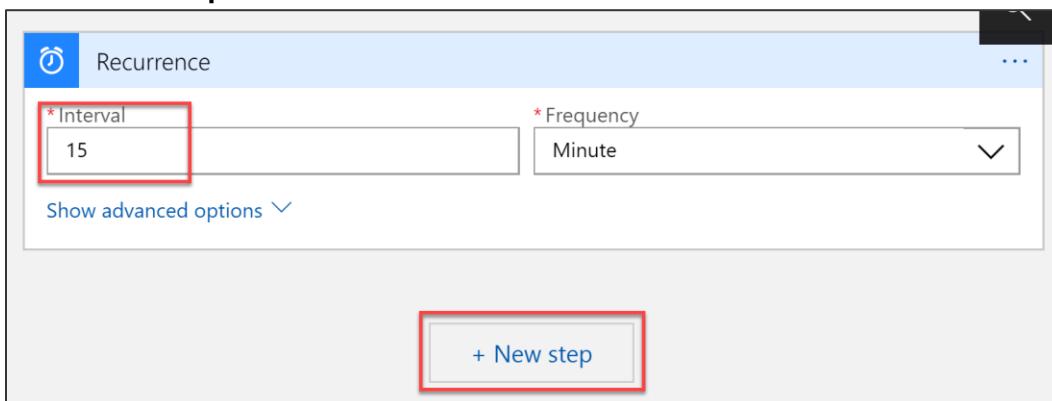
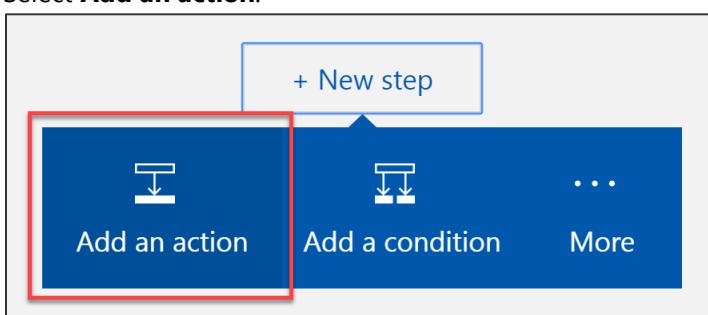
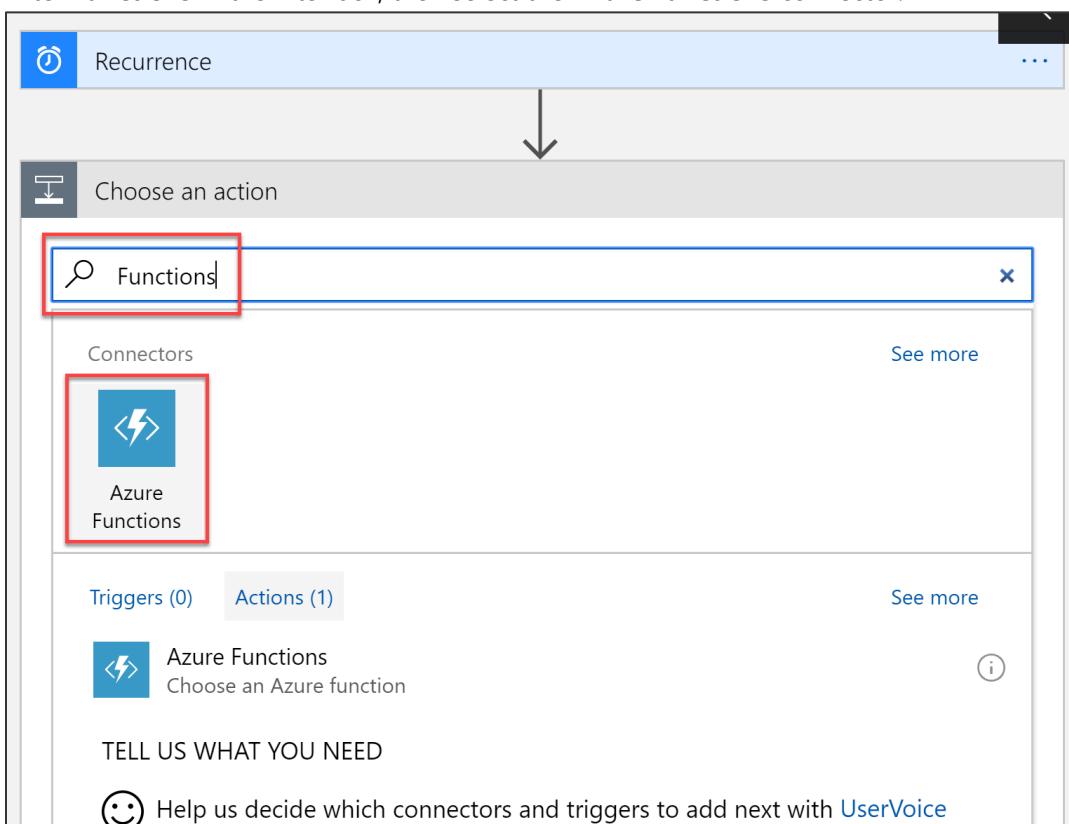
- c. Select the same **location** as your Resource Group.



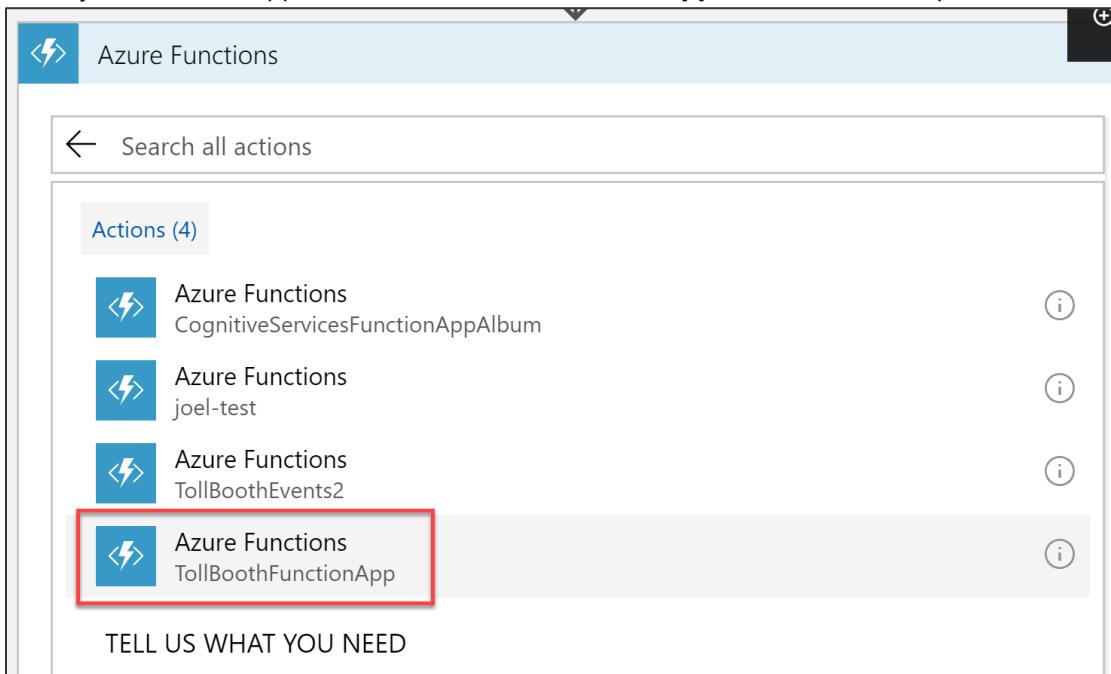
5. Open the Logic App once it has been provisioned.
6. In the Logic App Designer, scroll through the page until you located the *Start with a common trigger* section. Select the **Recurrence** trigger.



7. Enter **15** into the **Interval** box, and make sure Frequency is set to Minute. This can be set to an hour or some other interval, depending on business requirements.

8. Select **+ New step**.9. Select **Add an action**.10. Enter **Functions** in the filter box, then select the **Azure Functions** connector.

11. Select your Function App whose name ends in **FunctionApp**, or contains the ExportLicensePlates function.



Azure Functions

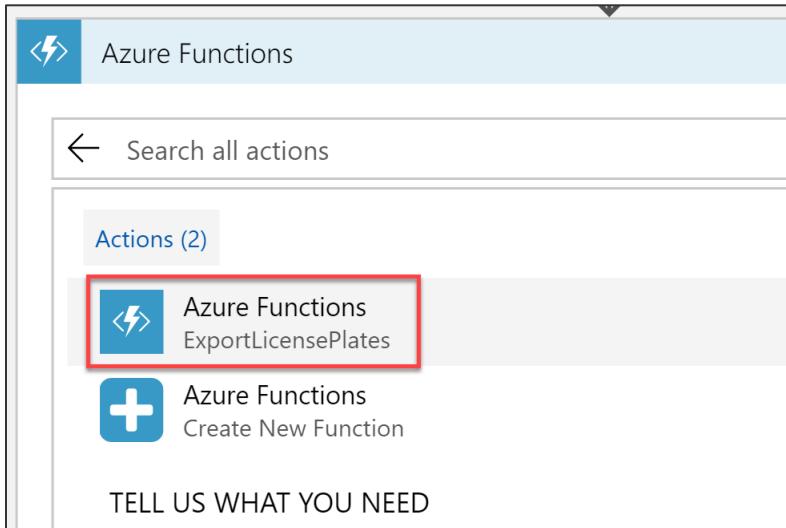
← Search all actions

Actions (4)

- Azure Functions CognitiveServicesFunctionAppAlbum
- Azure Functions joel-test
- Azure Functions TollBoothEvents2
- Azure Functions TollBoothFunctionApp

TELL US WHAT YOU NEED

12. Select the **ExportLicensePlates** function from the list.



Azure Functions

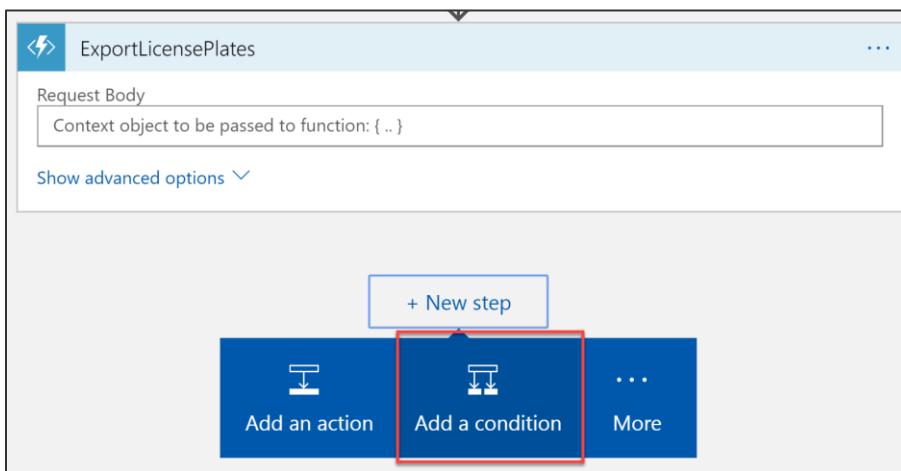
← Search all actions

Actions (2)

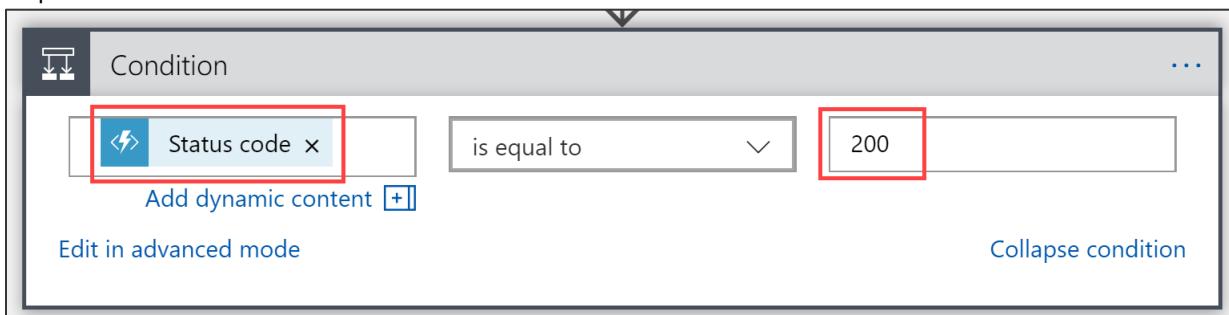
- Azure Functions ExportLicensePlates
- Azure Functions Create New Function

TELL US WHAT YOU NEED

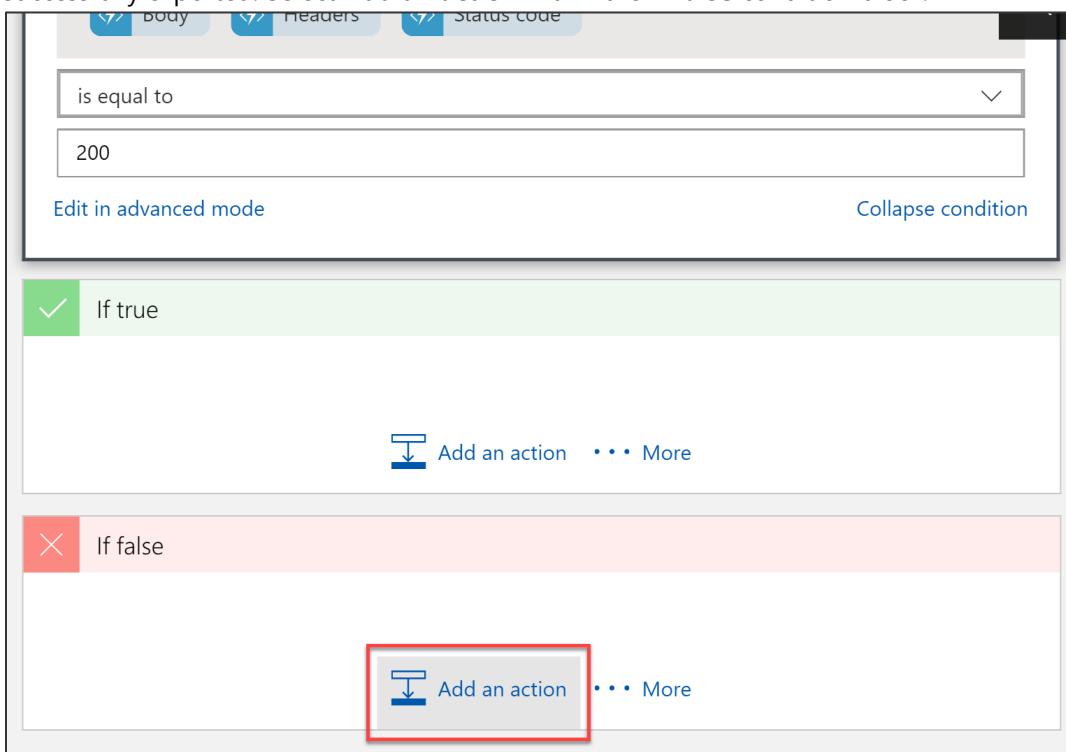
13. This function does not require any parameters that need to be sent when it gets called. Select **+ New step**, then **Add a condition**.



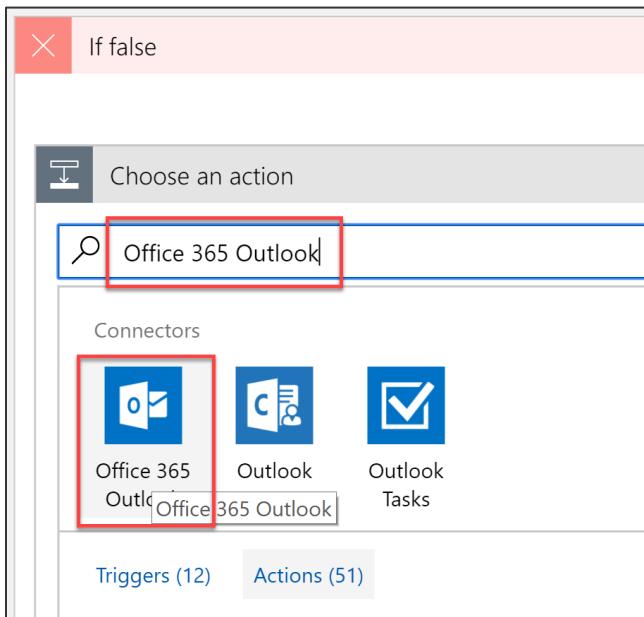
14. Select the **Status code** parameter in the first field. Make sure **is equal to** is selected in the second field, then enter **200** in the third field. This evaluates the status code returned from the ExportLicensePlates function, which will return a 200 code when license plates are found and exported. Otherwise, it sends a 204 (NoContent) status code when no license plates were discovered that need to be exported. We will conditionally send an email if any response other than 200 is returned.



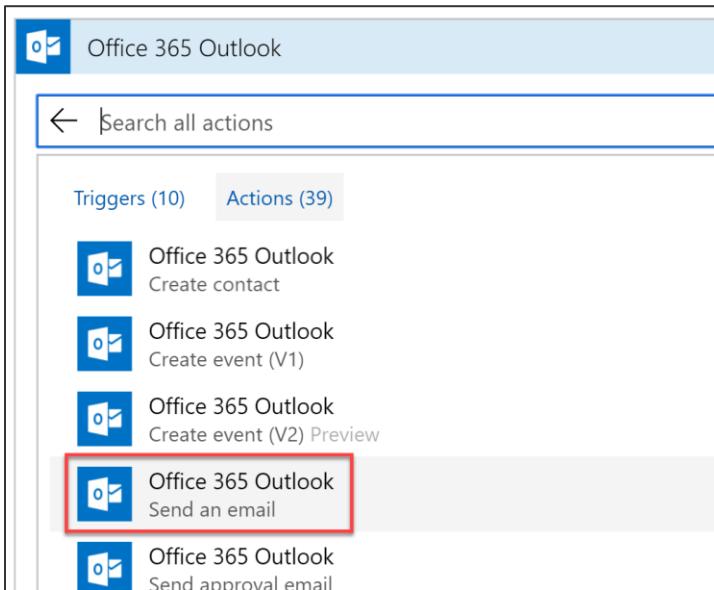
15. We will ignore the If true condition because we don't want to perform an action if the license plates are successfully exported. Select **Add an action** within the **If false** condition block.



16. Enter **Office 365 Outlook** in the filter box, then select the **Office 365 Outlook** connector from the results.

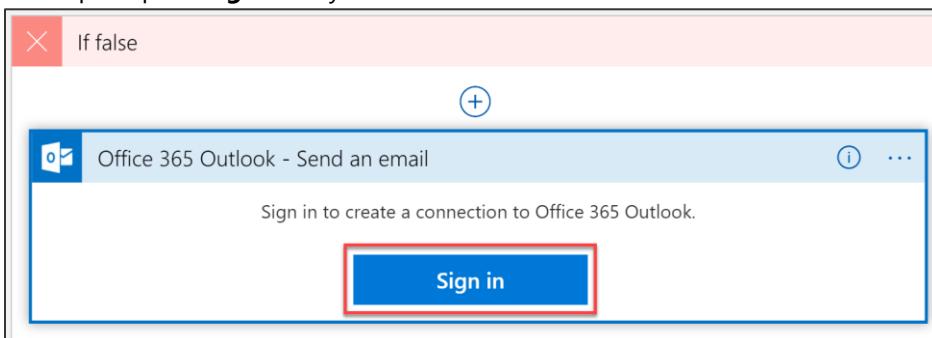


17. Select the **Send an email** action.



The screenshot shows the Microsoft Flow interface with the 'Actions' tab selected. Under the 'Actions (39)' category, the 'Office 365 Outlook' section is expanded. The 'Send an email' action is listed and highlighted with a red box. Other actions in the list include 'Create contact', 'Create event (V1)', 'Create event (V2) Preview', 'Send approval email', and 'Send an email' (which is the one selected).

18. When prompted **Sign in** to your Office 365 Outlook account.

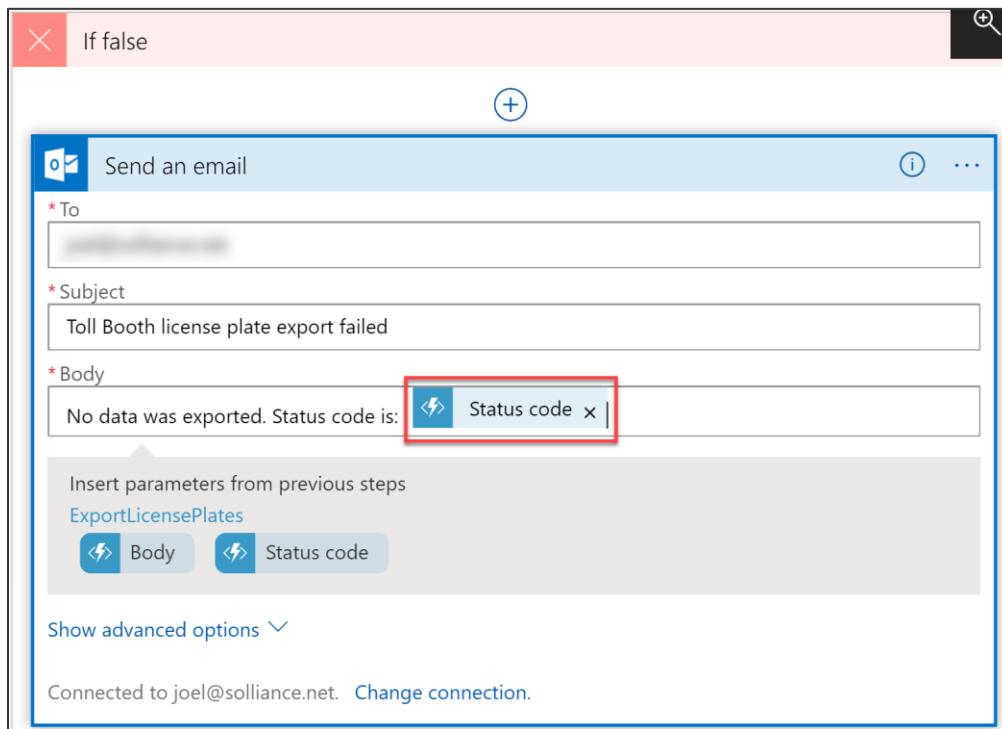


The screenshot shows the Microsoft Flow interface with an 'If false' condition step. The condition is set to 'Office 365 Outlook - Send an email'. Below the condition, there is a prompt: 'Sign in to create a connection to Office 365 Outlook.' A 'Sign in' button is present, and it is highlighted with a red box.

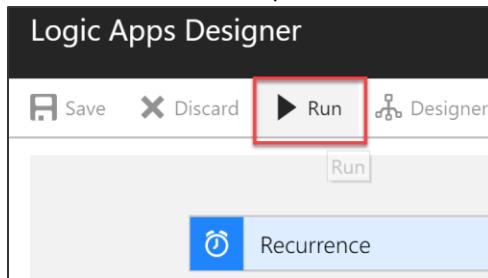
19. In the Send an email form, provide the following values:

- Enter your email address in the **To** box.
- Provide a **subject**, such as "Toll Booth license plate export failed".

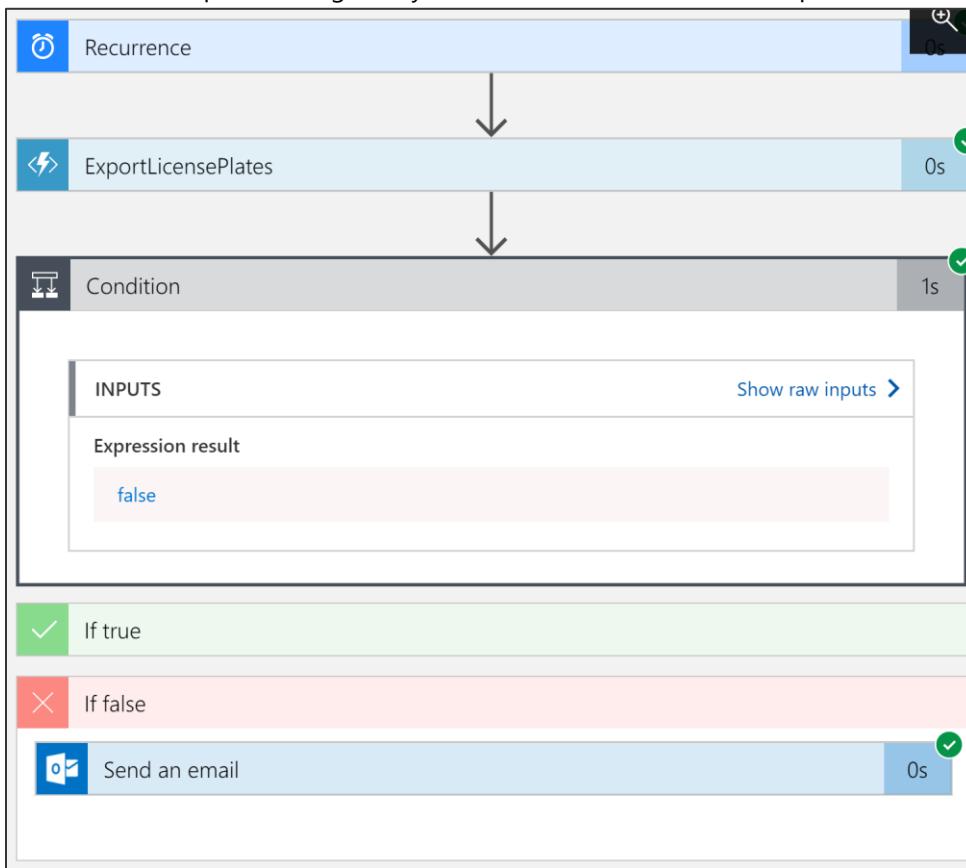
- c. Enter a message into the **body**, and select the **Status code** from the ExportLicensePlates function so that it is added to the email body.



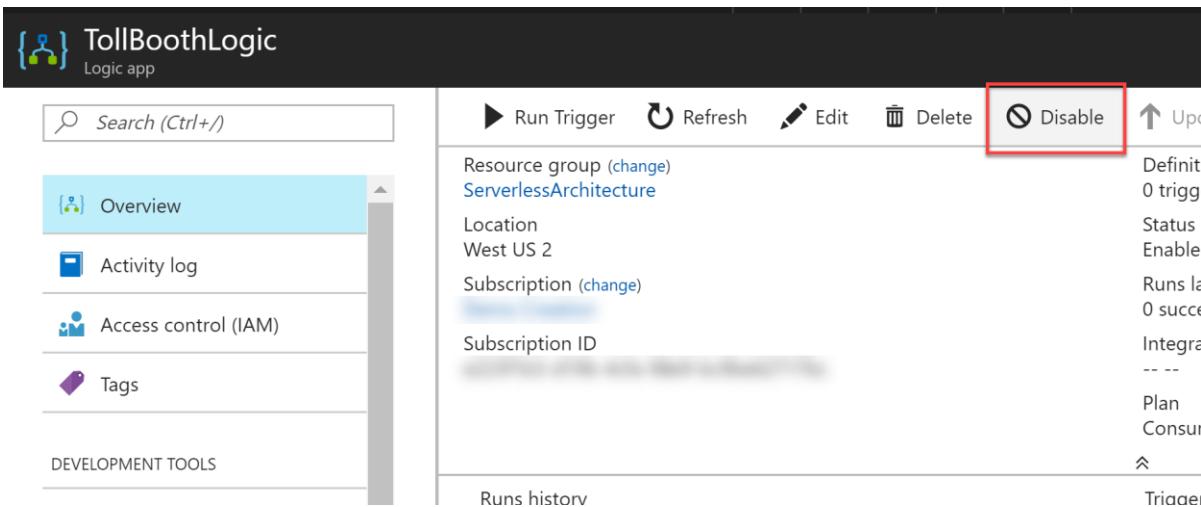
20. Select **Save** in the tool bar to save your Logic App.
21. Select **Run** to execute the Logic App. You should start receiving email alerts because the license plate data is not being exported. This is because we need to finish making changes to the ExportLicensePlates function so that it can extract the license plate data from Azure Cosmos DB, generate the CSV file, and upload it to Blob storage.



22. While in the Logic Apps Designer, you will see the run result of each step of your workflow. A green checkmark is placed next to each step that successfully executed, showing the execution time to complete. This can be used to see how each step is working, and you can click on the executed step and see the raw output.



23. The Logic App will continue to run in the background, executing every 15 minutes (or whichever interval you set) until you disable it. To disable the app, go to the **Overview** blade for the Logic App and select the **Disable** button on the toolbar.



Exercise 7: Configure continuous deployment for your Function App

Duration: 40 minutes

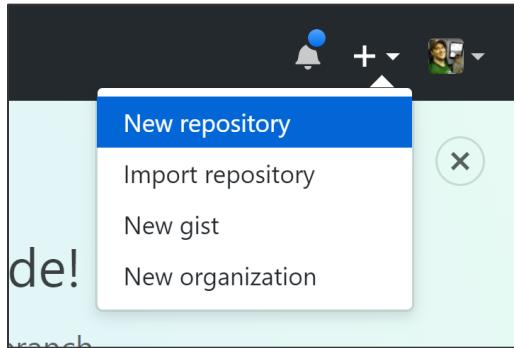
In this exercise, configure your Function App that contains the ProcessImage function for continuous deployment. You will first set up a GitHub source code repository, then set that as the deployment source for the Function App.

Help references

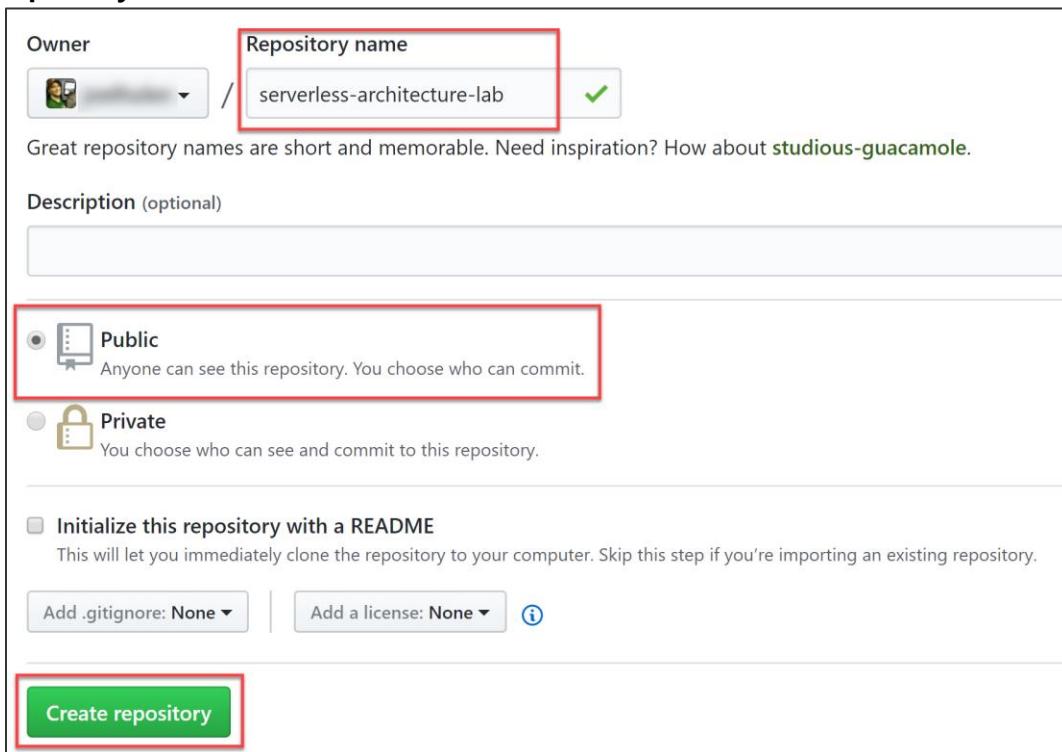
Creating a new GitHub repository	https://help.github.com/articles/creating-a-new-repository/
Continuous deployment for Azure Functions	https://docs.microsoft.com/en-us/azure/azure-functions/functions-continuous-deployment

Task 1: Create a GitHub repository

1. Navigate to <https://github.com> and sign in.
2. From the top-right corner of any page (once signed in), select the + menu item, then select **New repository**.



3. Select your owner account, enter a unique **repository name**, make sure it is set to **Public**, and then select **Create repository**.



Owner: [Profile Picture] / Repository name: **serverless-architecture-lab** ✓

Great repository names are short and memorable. Need inspiration? How about [studious-guacamole](#).

Description (optional):

Public
Anyone can see this repository. You choose who can commit.

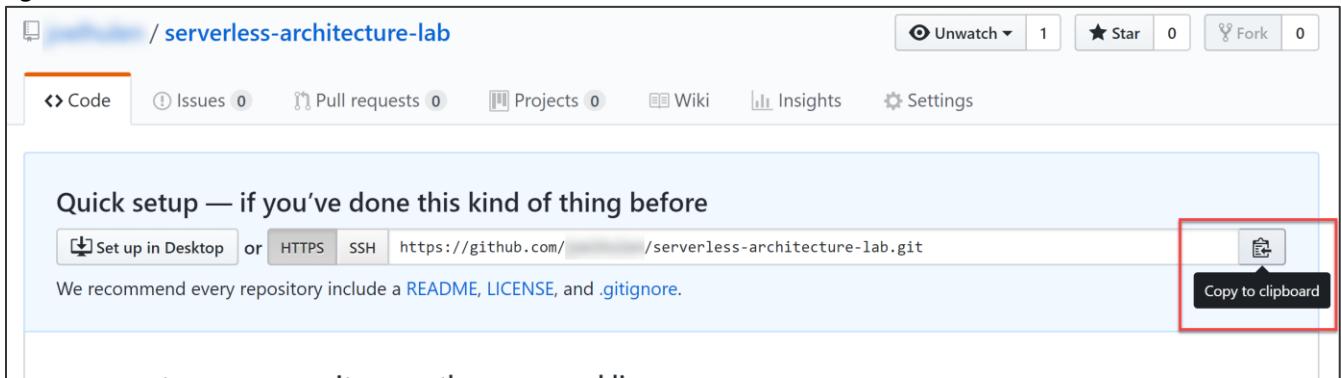
Private
You choose who can see and commit to this repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** Add a license: **None** ⓘ

Create repository

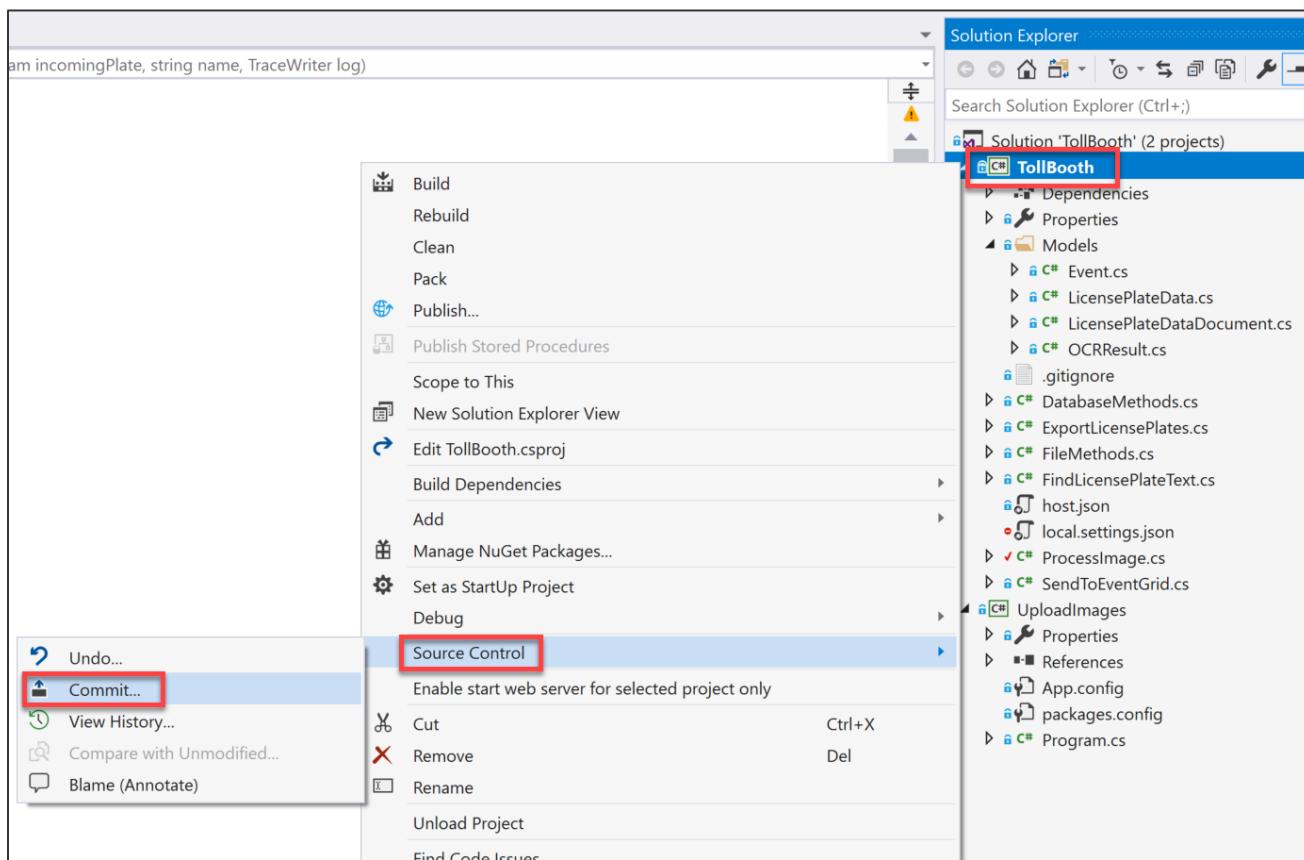
4. On your new repository page, copy the **HTTPS git path** to your clipboard, using the **button** provided on the right.



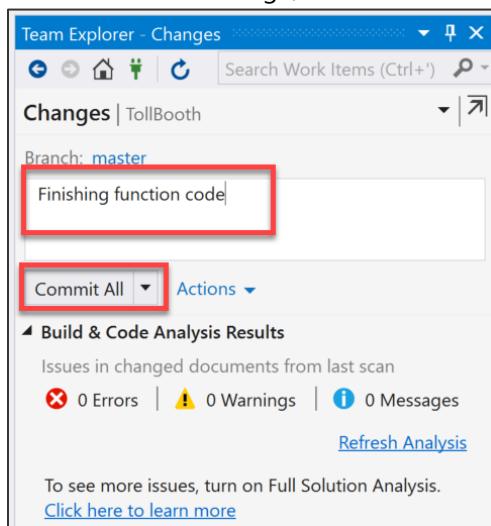
Task 2: Add GitHub repository to your Visual Studio solution

1. Open the **TollBooth** project in Visual Studio.

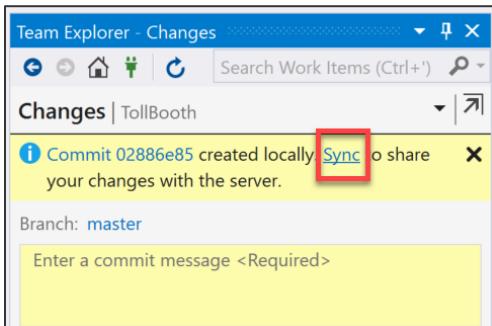
2. Right-click the **TollBooth** project in Solution Explorer, then select **Commit...** under the **Source Control** menu item.



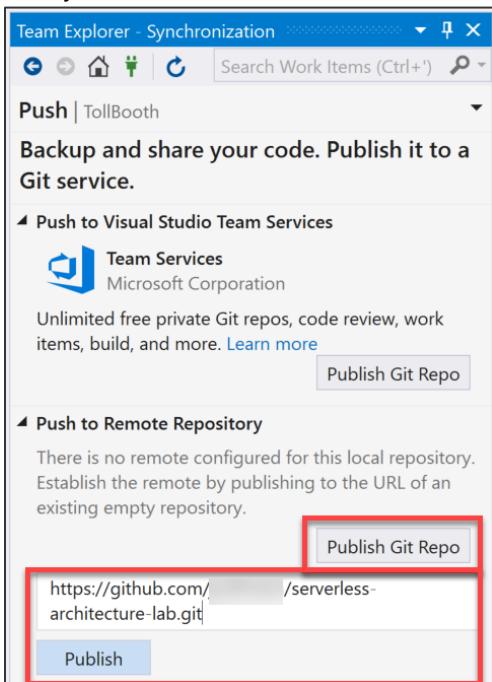
3. Enter a commit message, then select **Commit All**.



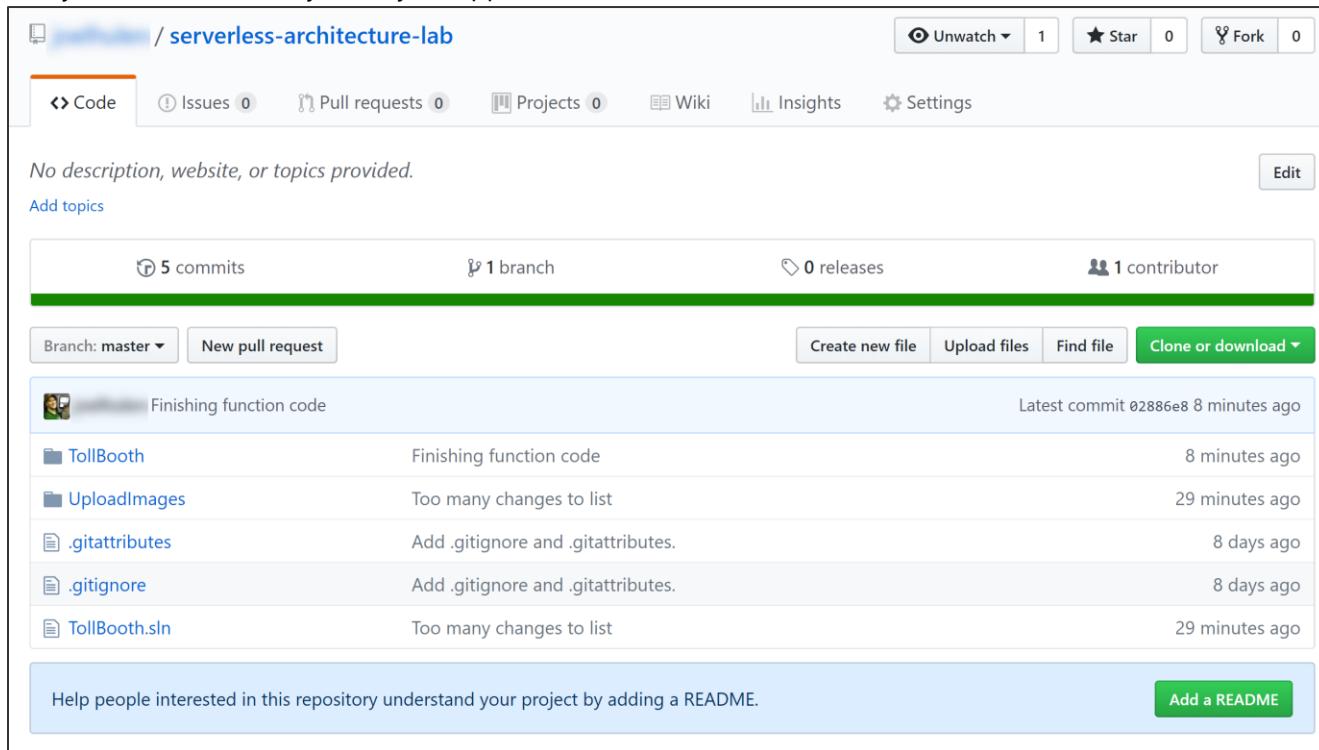
4. After committing, select the **Sync** link. This will allow us to add the remote GitHub repository.



5. Select the **Publish Git Repo** button, then paste the git URL for your new repository you copied from GitHub. Finally, select **Publish**.



6. Refresh your GitHub repository page in your browser. You should see that the project files have been added. Navigate to the **TollBooth** folder of your repo. Notice that the local.settings.json file has not been uploaded. That's because the .gitignore file of the TollBooth project explicitly excludes that file from the repository, making sure you don't accidentally share your application secrets.



No description, website, or topics provided.

5 commits 1 branch 0 releases 1 contributor

Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

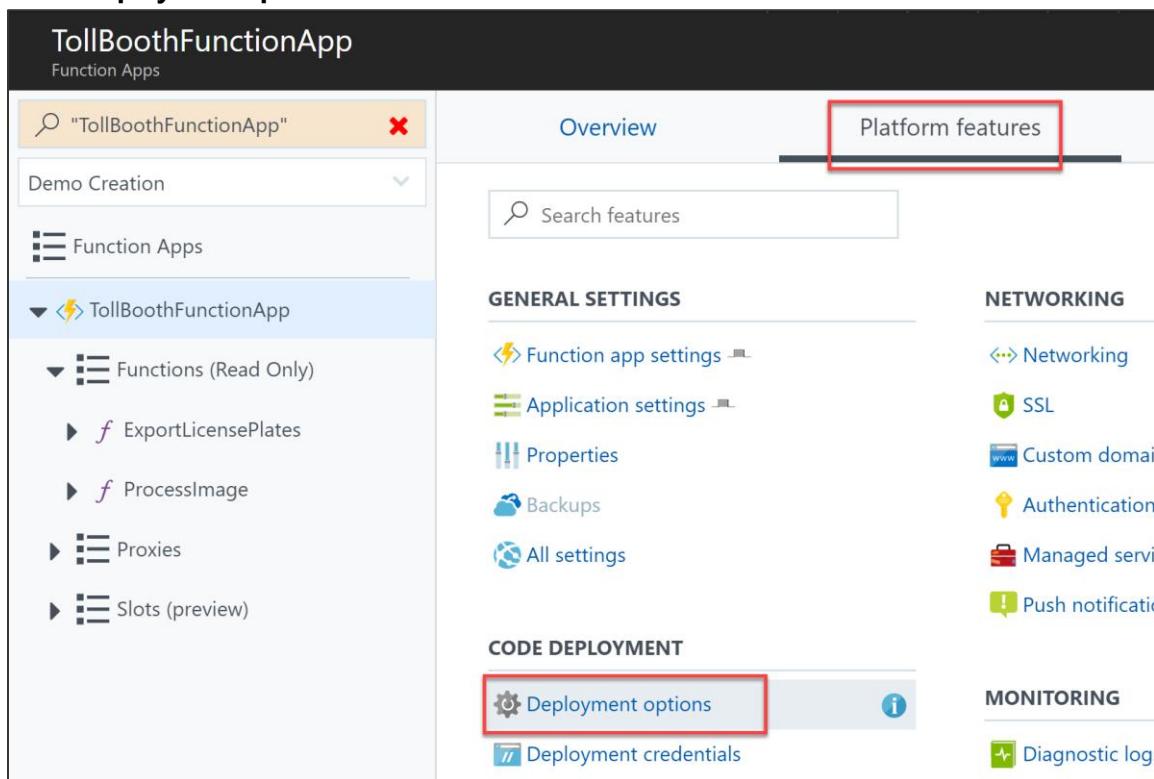
Author	Commit Message	Time Ago
	Finishing function code	Latest commit 02886e8 8 minutes ago
	TollBooth Finishing function code	8 minutes ago
	UploadImages Too many changes to list	29 minutes ago
	.gitattributes Add .gitignore and .gitattributes.	8 days ago
	.gitignore Add .gitignore and .gitattributes.	8 days ago
	TollBooth.sln Too many changes to list	29 minutes ago

Help people interested in this repository understand your project by adding a README. Add a README

Task 3: Configure your Function App to use your GitHub repository for continuous deployment

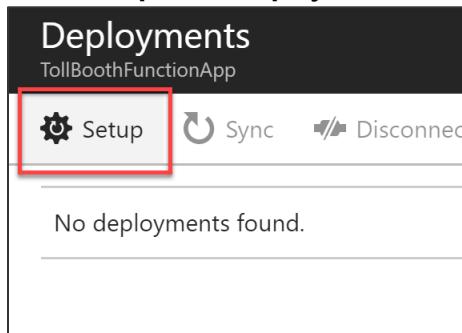
1. Open the Azure Function App you created whose name ends with **FunctionApp**, or the name you specified for the Function App containing the ProcessImage function.

2. Select **Deployment options** underneath the **Platform features** tab.



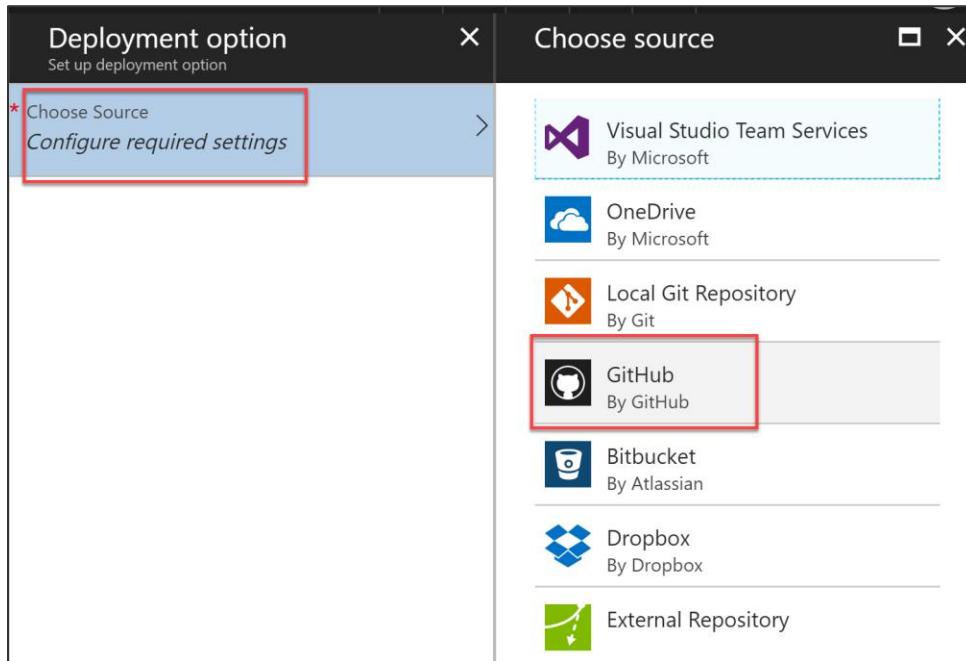
The screenshot shows the Azure portal interface for a Function App named 'TollBoothFunctionApp'. The left sidebar shows a tree structure with 'Function Apps' selected, and a sub-tree for 'TollBoothFunctionApp' showing 'Functions (Read Only)' with items like 'ExportLicensePlates' and 'ProcessImage'. The main content area has a tab bar with 'Overview' (selected), 'Platform features' (highlighted with a red box), and 'Deployment options' (highlighted with a red box). The 'Platform features' tab contains sections for 'GENERAL SETTINGS' (Function app settings, Application settings, Properties, Backups, All settings), 'NETWORKING' (Networking, SSL, Custom domain, Authentication, Managed service, Push notifications), and 'MONITORING' (Diagnostic logs). The 'Deployment options' section is part of the 'CODE DEPLOYMENT' group.

3. Select **Setup** in the **Deployments** blade.



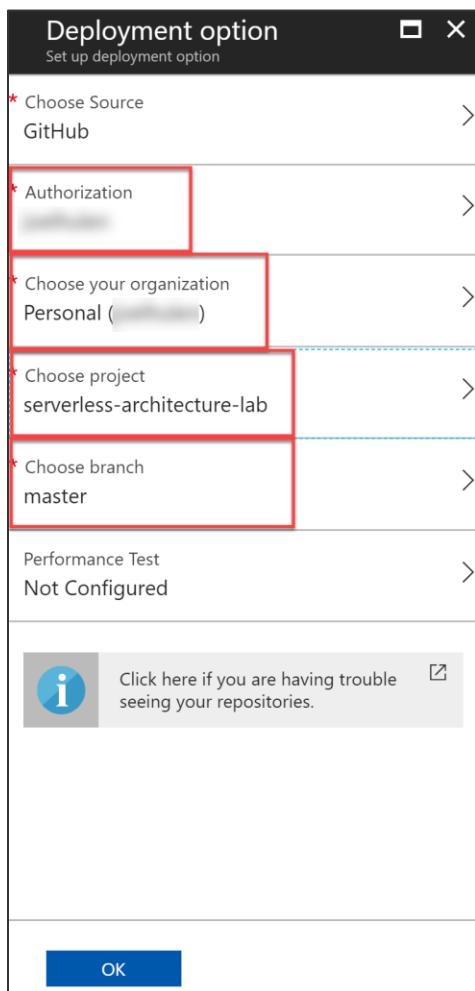
The screenshot shows the 'Deployments' blade for the 'TollBoothFunctionApp'. It has a top bar with 'Setup' (highlighted with a red box), 'Sync' (Sync icon), and 'Disconnected' (Disconnected icon). Below this, a message says 'No deployments found.'.

4. Select **Choose Source**, then **GitHub** in the list of sources.



5. Select **Authorization**, then enter your GitHub credentials when prompted.
6. **Choose your organization.**

7. Choose your new repository under **Choose project**. Make sure the **master branch** is selected.

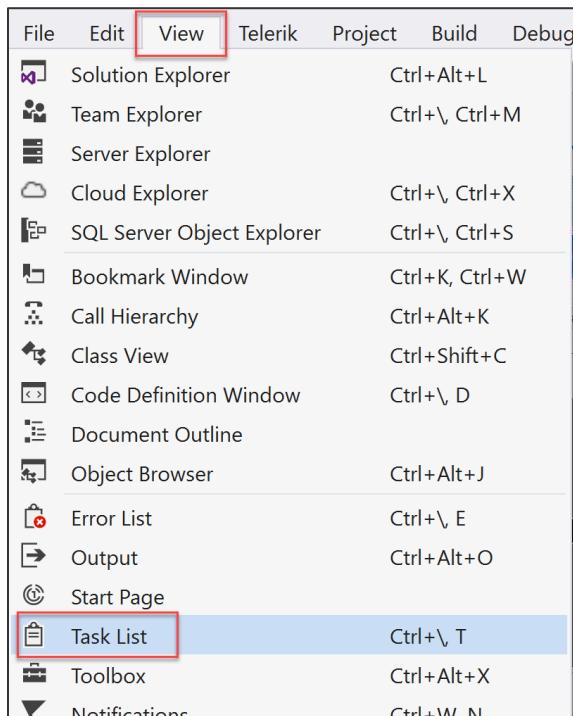


8. Select **OK**.
9. After continuous deployment is configured, all file changes in your deployment source are copied to the function app and a full site deployment is triggered. The site is redeployed when files in the source are updated.

Task 4: Finish your ExportLicensePlates function code and push changes to GitHub to trigger deployment

1. Navigate to the **TollBooth** project using the Solution Explorer of Visual Studio.

2. From the Visual Studio **View** menu, select **Task List**.



3. There you will see a list of TODO tasks, where each task represents one line of code that needs to be completed.
4. Open **DatabaseMethods.cs**.
5. The following code represents the completed task in DatabaseMethods.cs:

```
// TODO 5: Retrieve a List of LicensePlateDataDocument objects from the collectionLink where the
// exported value is false.
licensePlates = _client.CreateDocumentQuery<LicensePlateDataDocument>(collectionLink,
    new FeedOptions() { MaxItemCount = 100 })
    .Where(l => l.exported == false)
    .ToList();

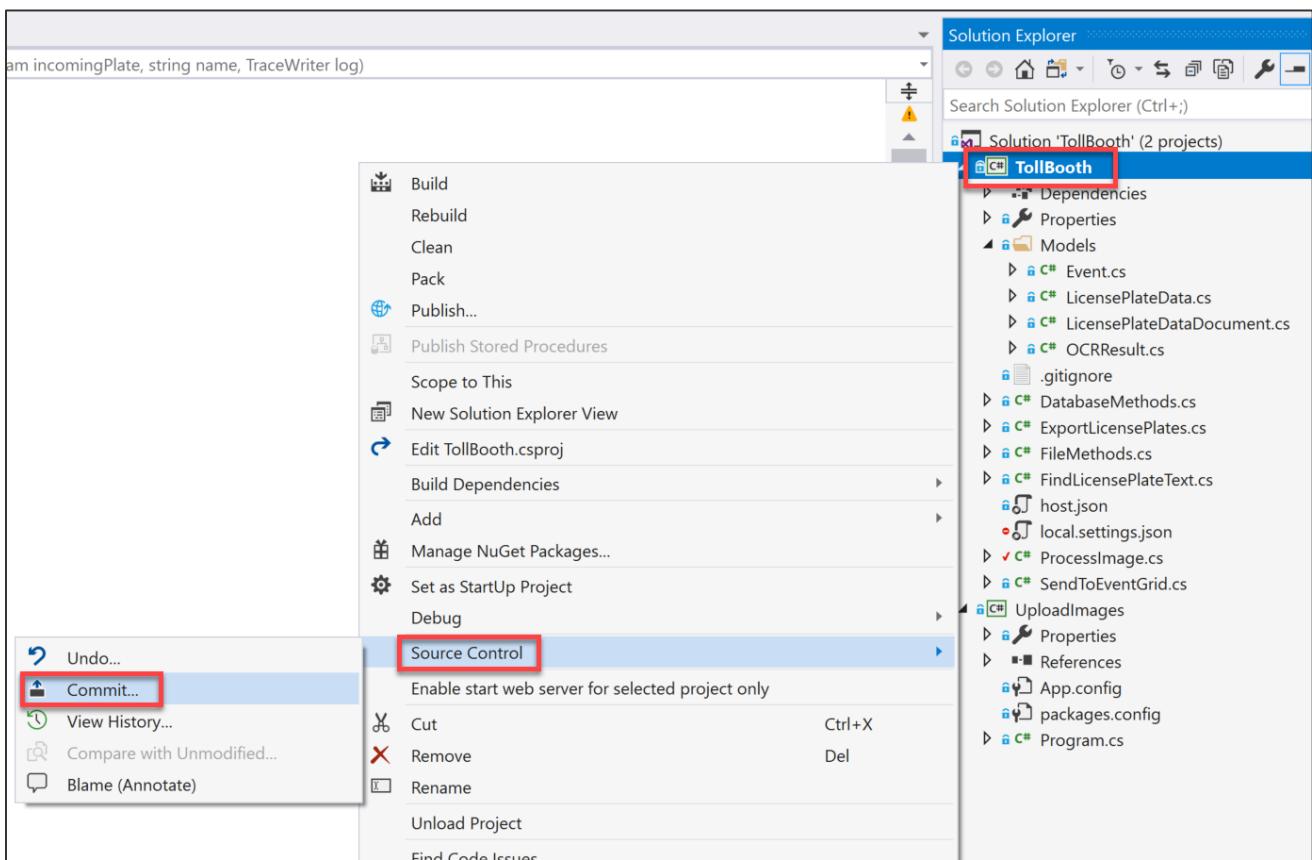
// TODO 6: Remove the line below.
```

6. Make sure that you deleted the following line under TODO 6: `licensePlates = new List<LicensePlateDataDocument>();`
7. Save your changes then open **FileMethods.cs**.
8. The following code represents the completed task in DatabaseMethods.cs:

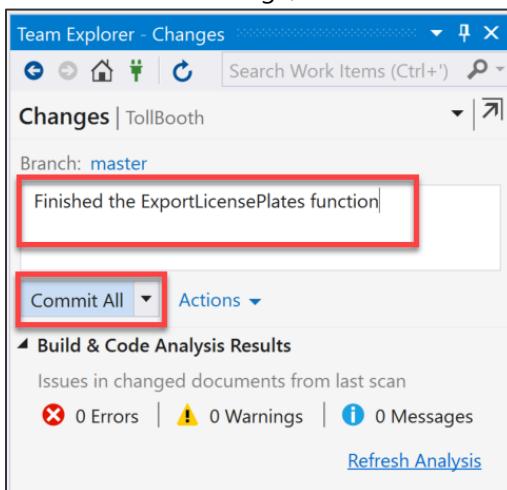
```
// TODO 7: Asynchronously upload the blob from the memory stream.
await blob.UploadFromStreamAsync(stream);
```

9. Save your changes.

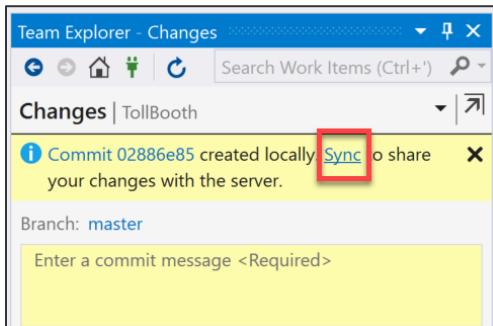
10. Right-click the **TollBooth** project in Solution Explorer, then select **Commit...** under the **Source Control** menu item.



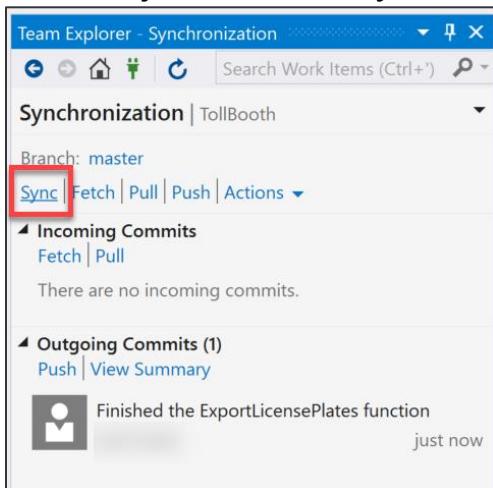
11. Enter a commit message, then select **Commit All**.



12. After committing, select the **Sync** link. This will allow us to add the remote GitHub repository.

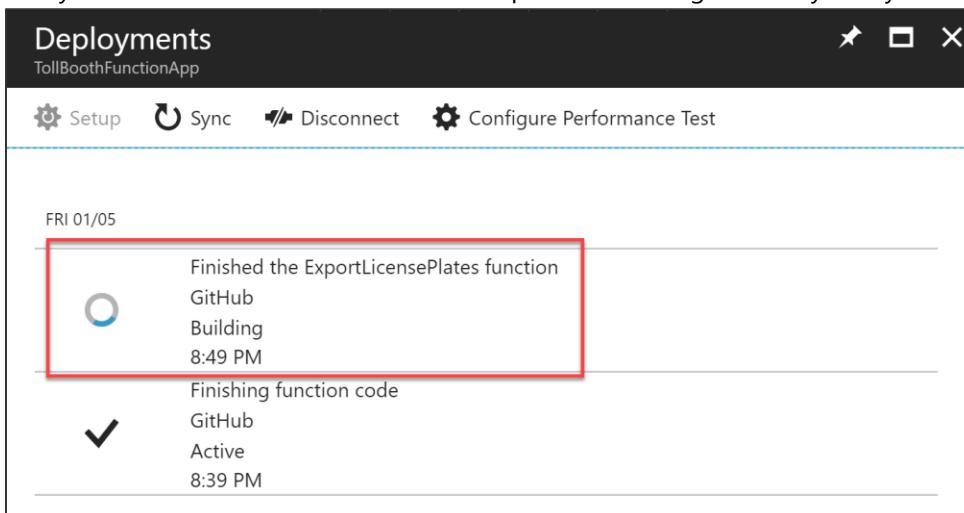


13. Select the **Sync** button on the **Synchronization** step.



Afterward, you should see a message stating that the incoming and outgoing commits were successfully synchronized.

14. Go back to Deployments for your Function App in the portal. You should see an entry for the deployment kicked off by this last commit. Check the timestamp on the message to verify that you are looking at the latest one.



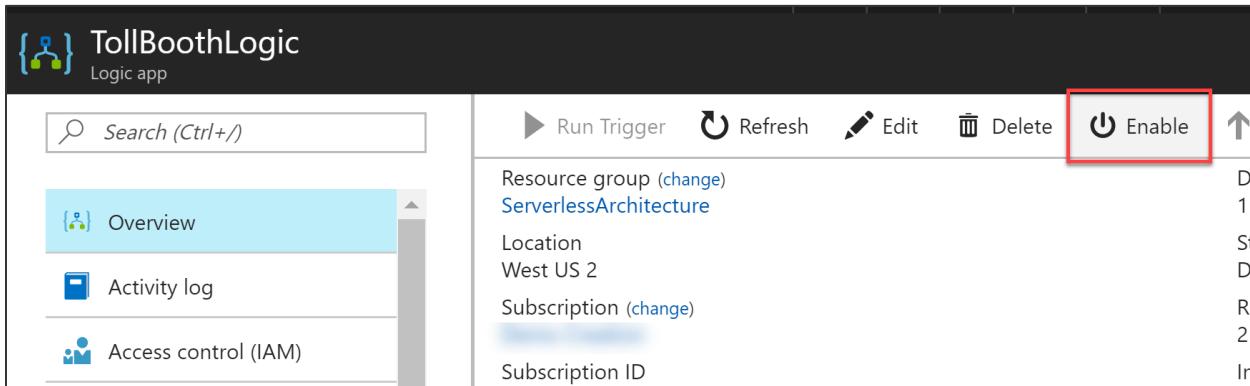
Exercise 8: Rerun the workflow and verify data export

Duration: 10 minutes

With the latest code changes in place, run your Logic App and verify that the files are successfully exported.

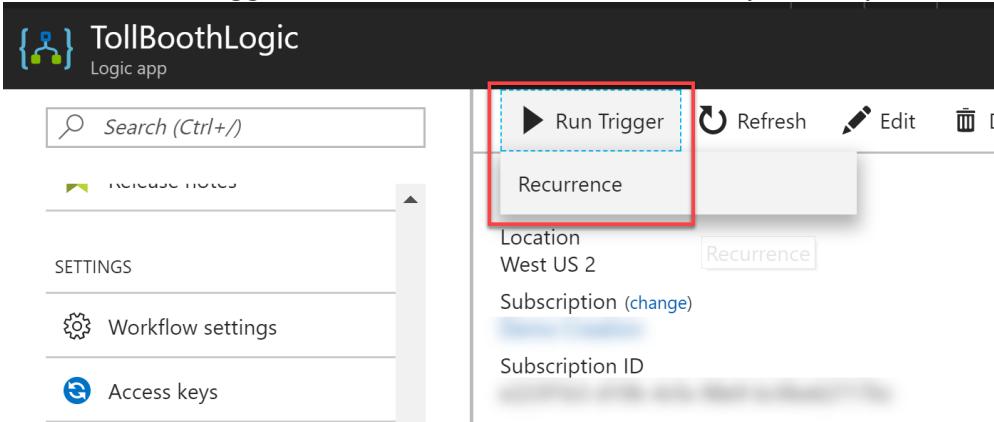
Task 1: Run the Logic App

1. Open your ServerlessArchitecture resource group in the Azure portal, then select your Logic App.
2. From the **Overview** blade, select **Enable**.



The screenshot shows the Azure Logic App Overview blade for a resource group named 'ServerlessArchitecture'. The 'TollBoothLogic' logic app is selected. The top navigation bar includes 'Run Trigger', 'Refresh', 'Edit', 'Delete', and 'Enable'. The 'Enable' button is highlighted with a red box. The main content area displays the logic app's configuration: Resource group (change), Location (West US 2), Subscription (change), and Subscription ID. On the left, a sidebar lists 'Overview', 'Activity log', and 'Access control (IAM)'.

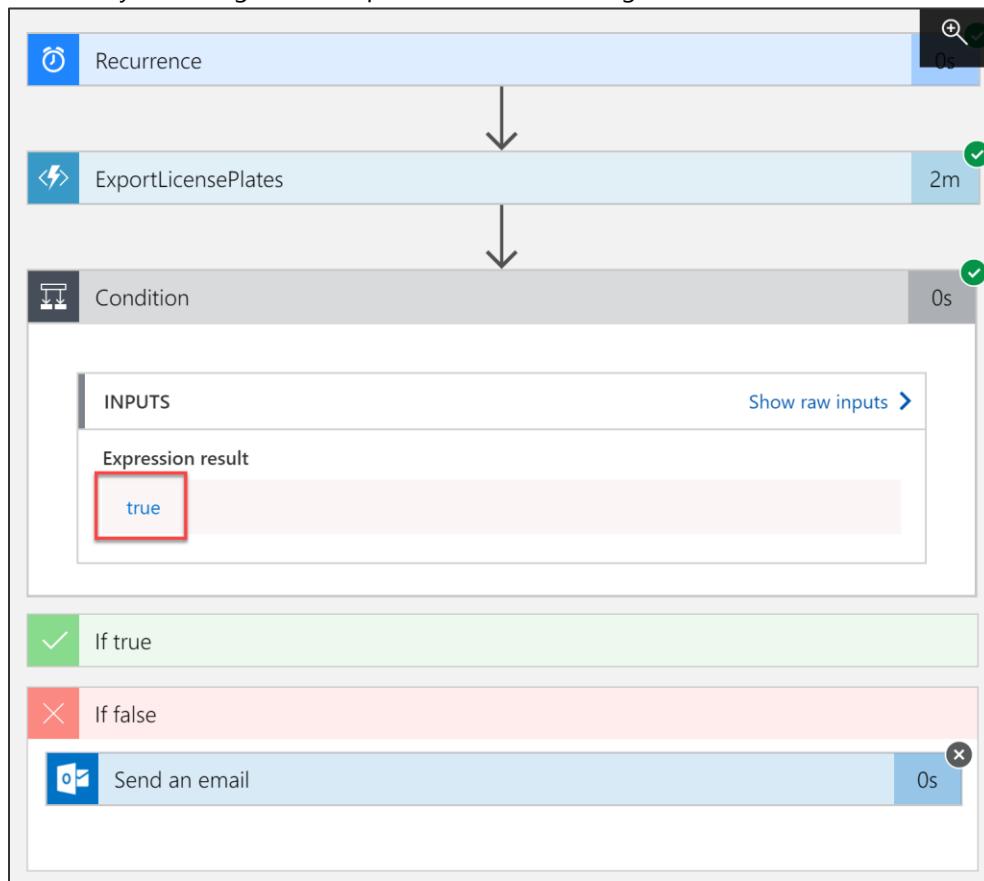
3. Now select **Run Trigger**, then select **Recurrence** to immediately execute your workflow.



The screenshot shows the Azure Logic App Overview blade for the same logic app. The 'Run Trigger' button is highlighted with a red box. A dropdown menu is open, showing 'Run Trigger' and 'Recurrence', with 'Recurrence' also highlighted with a red box. The main content area shows the logic app's configuration: Location (West US 2), Subscription (change), and Subscription ID. On the left, a sidebar lists 'Release notes', 'SETTINGS', 'Workflow settings', and 'Access keys'.

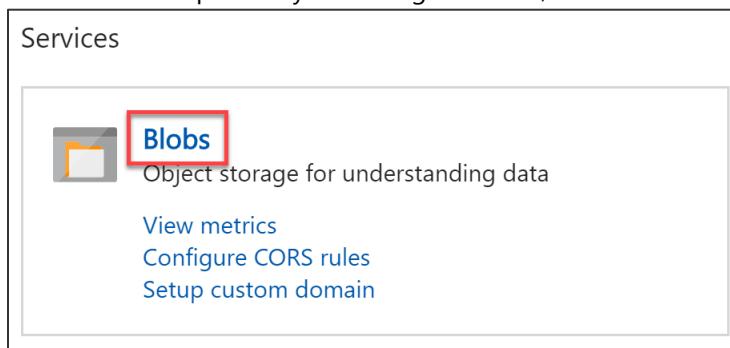
4. Select the **Refresh** button next to the Run Trigger button to refresh your run history. Select the latest run history item. If the expression result for the condition is **true**, then that means the CSV file should've been exported to

Blob storage. Be sure to disable the Logic App so it doesn't keep sending you emails every 15 minutes. Please note that it may take longer than expected to start running, in some cases.

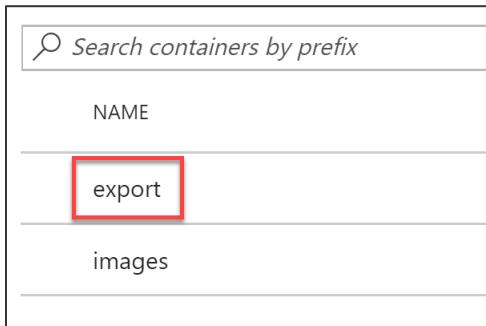


Task 2: View the exported CSV file

1. Open your ServerlessArchitecture resource group in the Azure portal, then select your **Storage account** you had provisioned to store uploaded photos and exported CSV files.
2. In the Overview pane of your storage account, select **Blobs**.

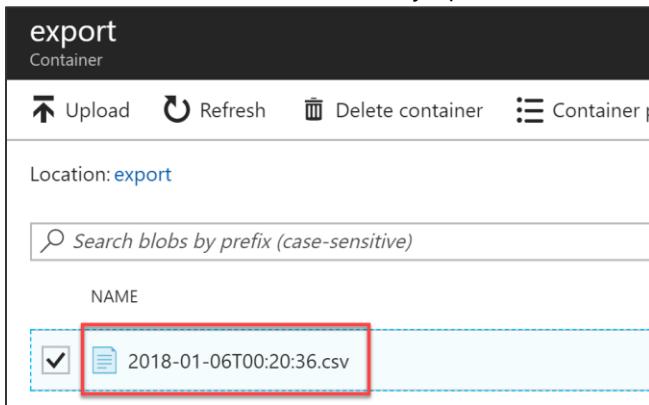


3. Select the **export** container.



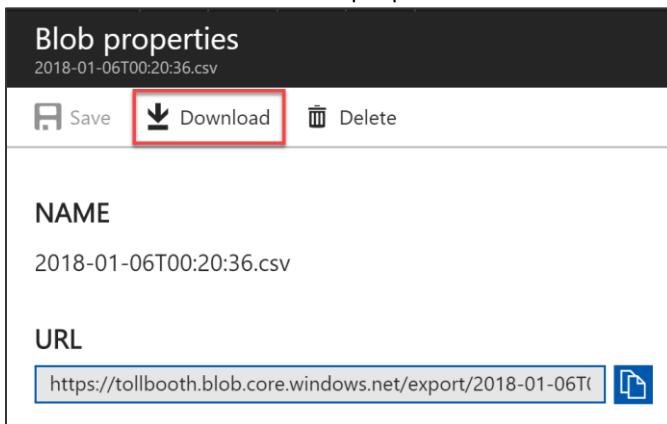
A screenshot of the Azure Storage Explorer interface. The left pane shows a list of containers. At the top is a search bar labeled 'Search containers by prefix'. Below it, the 'NAME' column lists three items: 'export' (which is highlighted with a red box), 'images', and another unnamed item. The 'export' item is selected.

4. You should see at least one recently uploaded CSV file. Click on the filename to view its properties.



A screenshot of the Azure Storage Explorer interface, showing the contents of the 'export' container. The left pane shows a list of blobs. At the top is a search bar labeled 'Search blobs by prefix (case-sensitive)'. Below it, the 'NAME' column lists one item: '2018-01-06T00:20:36.csv' (which is highlighted with a red box). The '2018-01-06T00:20:36.csv' item is selected.

5. Select **Download** in the blob properties window.



A screenshot of the 'Blob properties' window for the blob '2018-01-06T00:20:36.csv'. The window has a dark header bar with the text 'Blob properties' and '2018-01-06T00:20:36.csv'. Below the header are three buttons: 'Save' (with a save icon), 'Download' (with a download icon, which is highlighted with a red box), and 'Delete' (with a delete icon). The main content area shows the 'NAME' of the blob as '2018-01-06T00:20:36.csv' and the 'URL' as 'https://tollbooth.blob.core.windows.net/export/2018-01-06T00:20:36.csv'.

The CSV file should look similar to the following:

A	B	C	D	E
1	FileName	LicensePlateText	TimeStamp	LicensePlateFound
2	0293KHCR.jpg	THECAR	1/5/2018 21:02	TRUE
3	ZIM7HYST.jpg	SORRY	1/5/2018 21:02	TRUE
4	ABAHYTPD.jpg	JP THRLS	1/5/2018 21:02	TRUE
5	FTU0V15C.jpg	BUG 2145	1/5/2018 21:02	TRUE
6	8Q58MT1T.jpg	ROCKY 58	1/5/2018 21:02	TRUE
7	S8Z8Y6WG.jpg	TWAWSI	1/5/2018 21:02	TRUE
8	0CHPK1D4.jpg	THECAR	1/5/2018 21:02	TRUE
9	CUOAGXNU.jpg	ZOOMN65	1/5/2018 21:02	TRUE
10	H68UYLWM.jpg	BUG 2145	1/5/2018 21:02	TRUE
11	B49EWF5P.jpg	TWAWSI	1/5/2018 21:02	TRUE
12	8Y0IS5G0.jpg	127 RFS	1/5/2018 21:02	TRUE
13	I5S9JQJJ.jpg	127 RFS	1/5/2018 21:02	TRUE
14	GIX9EAKO.jpg	ROCKY 58	1/5/2018 21:02	TRUE
15	72GVVX8H.jpg	SORRY	1/5/2018 21:02	TRUE
16	GQFTOGFW.jpg	ZOOMN65	1/5/2018 21:02	TRUE
17	3ORIA9TR.jpg	IP THRLS	1/5/2018 21:02	TRUE

6. The ExportLicensePlates function updates all of the records it exported by setting the exported value to true. This makes sure that only new records since the last export are included in the next one. Verify this by re-executing the script in Azure Cosmos DB that counts the number of documents in the Processed collection where exported is false. It should return 0 unless you've subsequently uploaded new photos.

After the hands-on lab

Duration: 10 minutes

In this exercise, attendees will deprovision any Azure resources that were created in support of the lab.

Task 1: Delete the Resource group in which you placed your Azure resources.

1. From the Portal, navigate to the blade of your **Resource Group** and select **Delete** in the command bar at the top.
2. Confirm the deletion by re-typing the **resource group name** and selecting **Delete**.

You should follow all steps provided *after* attending the hands-on lab.