

Lab 1: Writing an Azure Function

Hands-on Lab

Released:

Terms of Use

© 2018 Microsoft Corporation. All rights reserved.

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

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

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

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

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

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

Table of Contents

[Lab Overview 1](#_Toc505693021)

[Exercise 1: Creating Azure Function to Azure Storage Queue for inbound trigger 2](#_Toc505693022)

[Exercise 2: Connecting to Cognitive Services for Content Moderation 6](#_Toc505693023)

# Lab Overview

###### Abstract

In this lab, we will be working with Azure Functions and Cognitive Services APIs to create an extension to an eCommerce website that will accept content posts by users and use the Content Moderator service in Cognitive Services to determine if there is any profanity within the content.

###### Learning Objectives

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

* Create an Azure Function using the Azure Portal editor and bind it to an Azure Storage event.
* Create a Cognitive Services account and call the Content Moderator API from an Azure Function.

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

# Exercise : Creating Azure Function to Azure Storage Queue for inbound trigger

#### Scenario

In this exercise, you will create your first Azure Function and bind it to an Azure Storage Queue for inbound trigger. In this exercise one can assume that this would be connected to the website where the website would put new product reviews on the Queue to be reviewed and this Azure Function would retrieve the content and process it.

After completing this exercise, you will understand:

* How to create an Azure Function
* Basic input (Queue) binding syntax for Queues for Azure Functions
* Using Web Editor to edit and test a Function using C# script syntax

#### Prerequisites:

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

Download Azure Storage Explorer from <https://azure.microsoft.com/en-us/features/storage-explorer/>

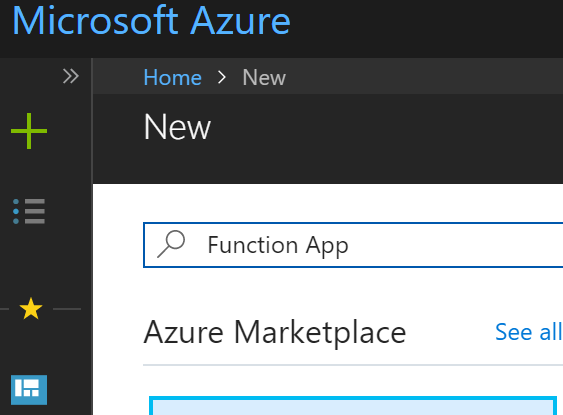
#### Install and configure:

You will need:

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

#### Creating first Function with inbound connection

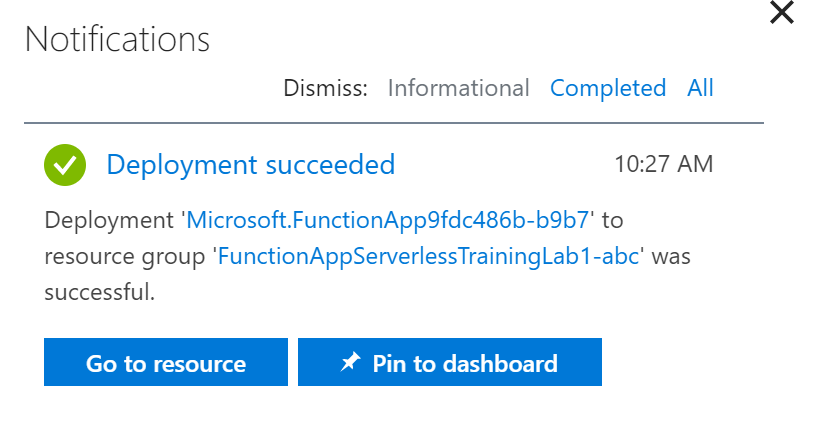
1. Navigate to the Azure Management Portal (<http://portal.azure.com>) and search for: **Function App**, after locating Function App in the search results, click **Create** to start the Function App Create blade.



1. In the Create pane prompts:
   * In the **App name** field, use a unique name for your function
   * In the **Subscription** field, select your subscription
   * In the **Resource Group** field, for simplicity of management use a consistent resource group for all of the labs
   * In the **OS** field, use **Windows**
   * In the **Hosting Plan** field, use **Consumption Plan**
   * In the **Location** field, use the default of **Central US** or whatever might be appropriate for you.
   * In the **Storage** field, create a new storage account for this Function App, it should default to match the name of the app with an extension, which is fine.
   * In the **Application Insights** field, leave the default to off, normally you will want App Insights but for this demo we will leave it off.

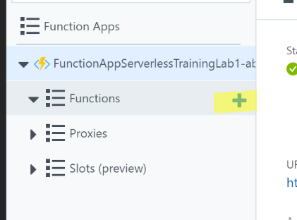
After this click **Create** to accept these values and move to the next step

1. Wait for the deployment to complete

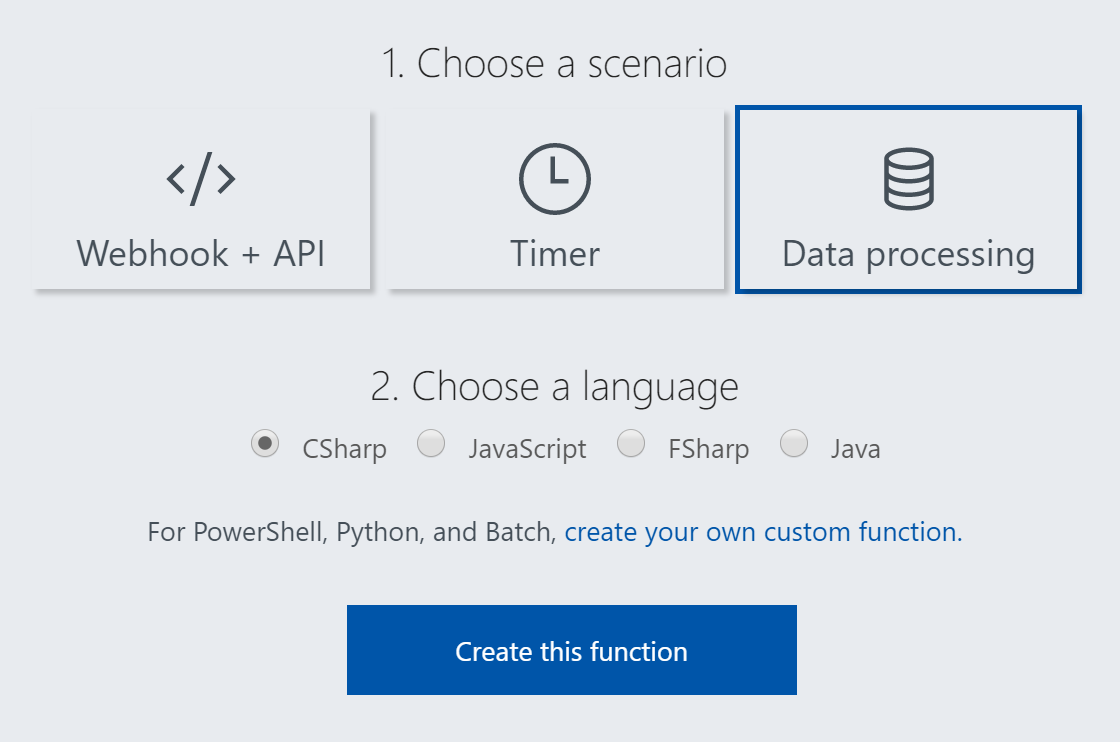


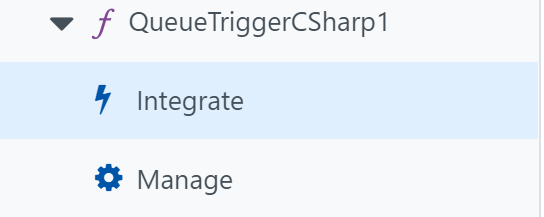
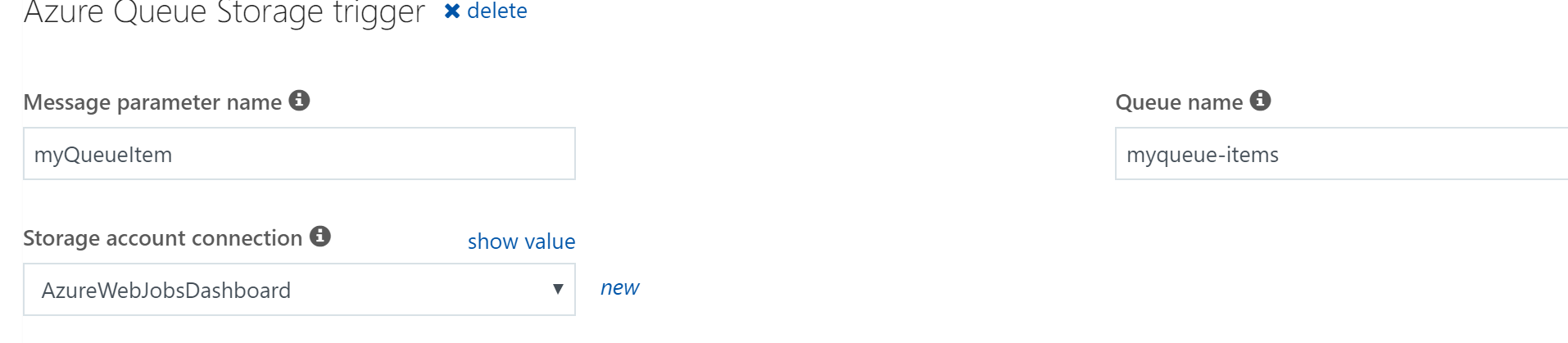
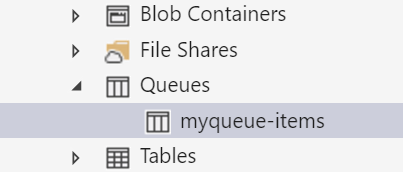
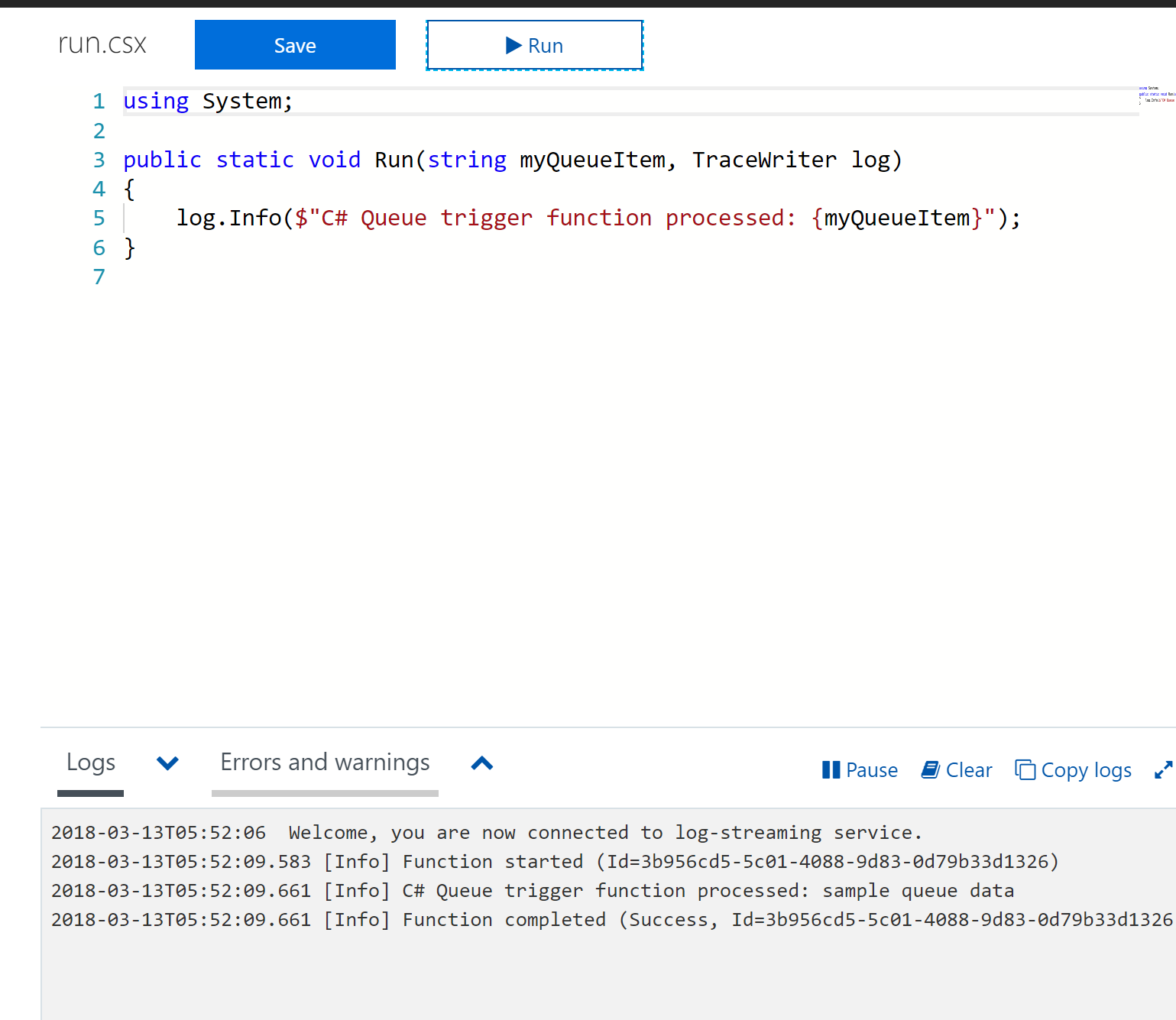
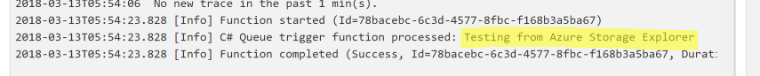
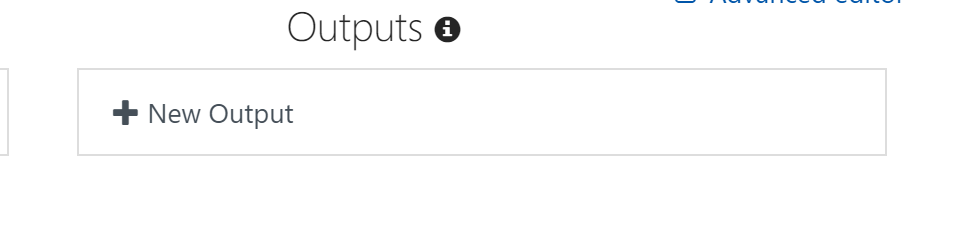
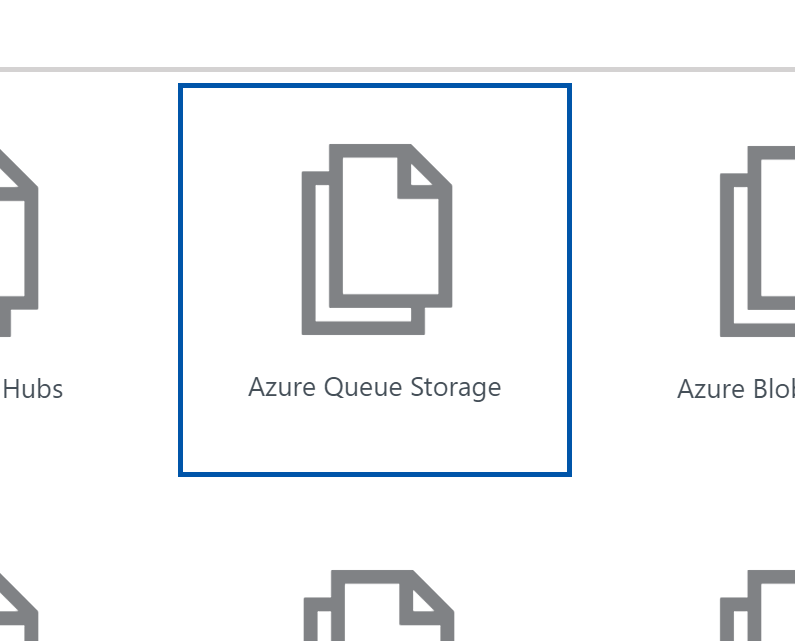
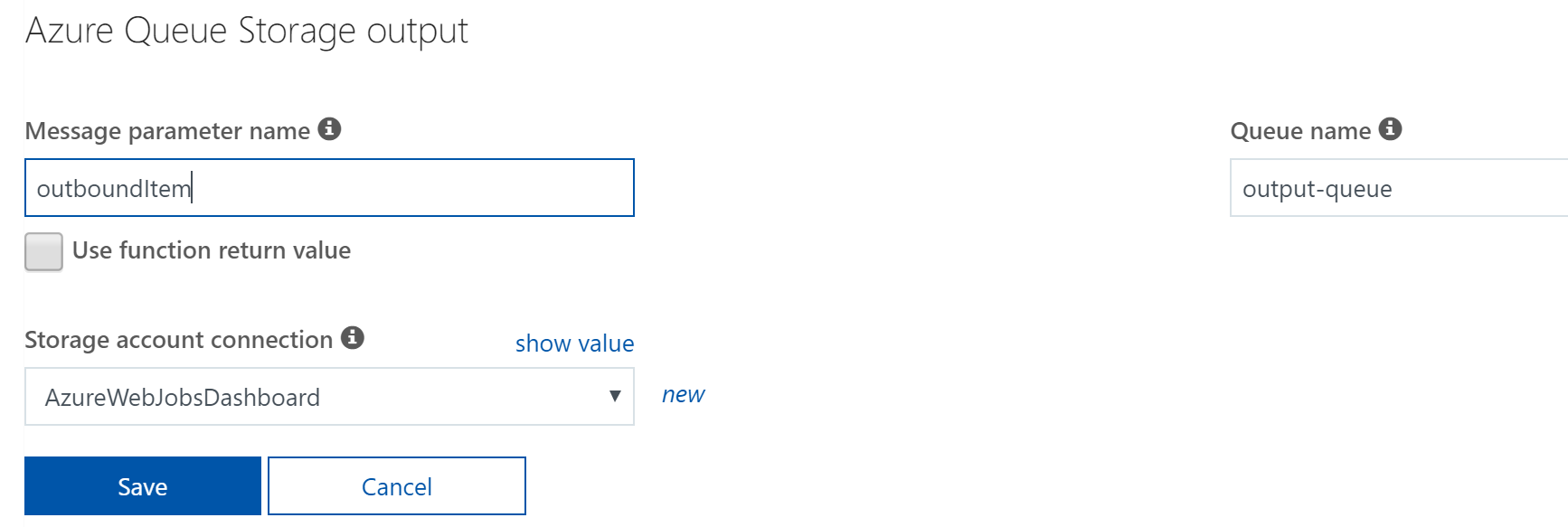
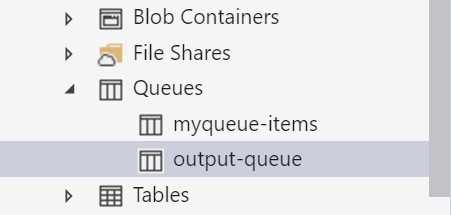
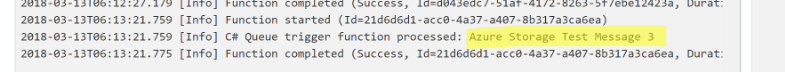
And then click **Go to Resource.**

1. At this point you can create a new Function in the app by clicking the plus sign next to the Functions name.



1. Pick **Data processing** and **CSharp** and click **Create this function**.



1. By default this will create a function bound to a queue. Looking at the CSX file you’ll see that it processes a queue and writes a log message about the item found on the queue.
2. Click on the **Integrate** setting  
   
3. Check that the **Storage account connection** is set to AzureWebJobsDashboard. Functions defaults it to AzureWebJobsDashboard. This is used by the AzureWebJobs infrastructure and are not good choices for production systems, but we will use them right not for this little example lab.  
   
4. Using Azure Storage Explorer, create the Storage Queue, which should be named to match the **Queue name** seen in the settings, in this case it will be **myqueue-items**.   
   
5. Go back and **Run** the function, so we can then put an item in the queue and make sure it is correctly processing.  
   
6. Return to the Azure Storage Explorer and add an item to the queue, you should then see it printed in the log in the browser log window.  
   
7. Next, create the output binding, to send the output from the Function to another Storage Queue. Set it to the same Azure Storage connection, AzureWebJobsDashboard but use a new Queue name, . Use the parameter name of outboundItem. Click on **+ New Output**
8. Select **Azure Queue Storage** and click **Select.**  
   
9. Set **Message parameter name** to **outboundItem** and **Queue name** to **output-queue**. Then click **Save**.  
   
10. Using Azure Storage Explorer create a new **Queue** called **output-queue**.  
    
11. Update the code to match, <https://gist.github.com/criter/513f59f2d661bac0626c6034ca7ab7dd>
    1. You’ll see that there is a new parameter in the Run method to match the output binding you added.
    2. There is a new class added to allow for formatting of the outbound parameter.
    3. And a new line to populate the outboundItem variable.
12. At this point you should be able to put an item into the input queue (myqueue-items) and that will be processed through the function and then placed into the output queue (output-queue) with information in a JSON format. You should be able to **Run** the Function and test that with Azure Storage Explorer by putting some text into the input queue. You’ll see it in the Log and then also in the output-queue.  
    
13. Next we can move on to integrating the Cognitive Services API.

# Exercise : Connecting to Cognitive Services for Content Moderation

#### Scenario

In this exercise, you will be using the Content Moderation API from an Azure Function. This will demonstrate how you can make external connections from a Function App.

After completing this exercise, you will understand:

* Include external assemblies in C# script
* Use Application Settings in Function Apps
* Using Cognitive Services Content Moderation API

#### Prerequisites:

To complete this lab, you should have a basic understanding of C# script and using REST APIs.

#### Install and configure:

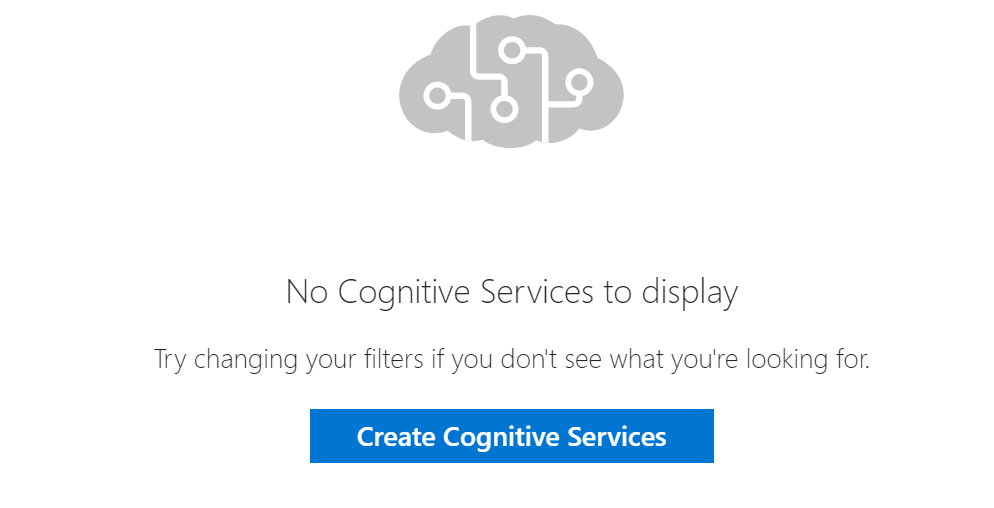
You will need:

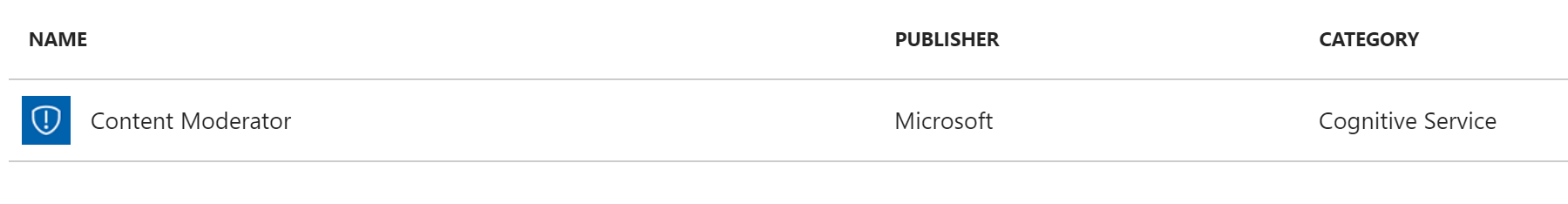
* A desktop computer (host) running Windows 10
* An active Azure subscription (MSDN credits will suffice)
* An active Internet connection

#### 

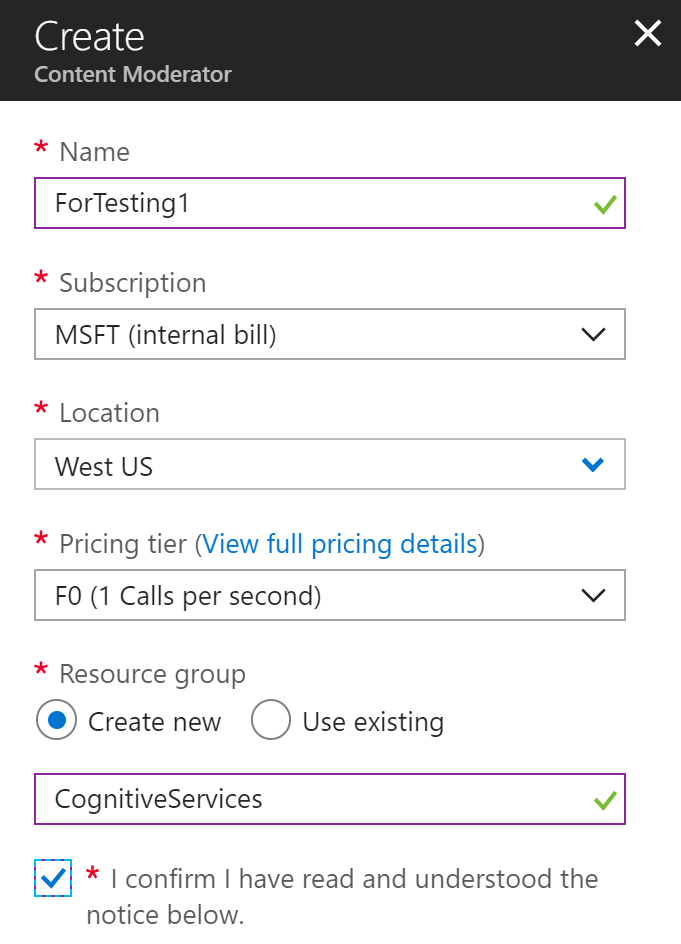
#### Connecting to Cognitive Services

1. Create a Cognitive Services account. After logging into the Azure Portal, use the following URL to create a Cognitive Services API.  
   <https://ms.portal.azure.com/#blade/HubsExtension/BrowseResourceBlade/resourceType/Microsoft.CognitiveServices%2Faccounts>

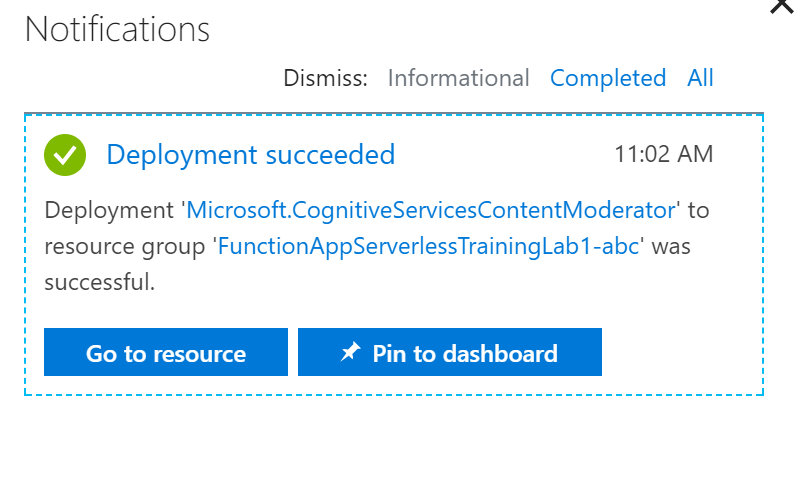


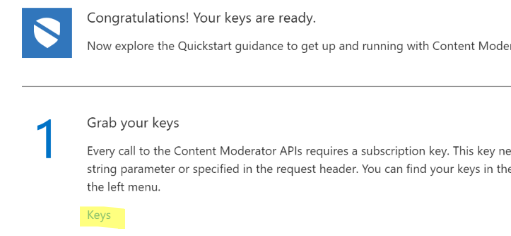
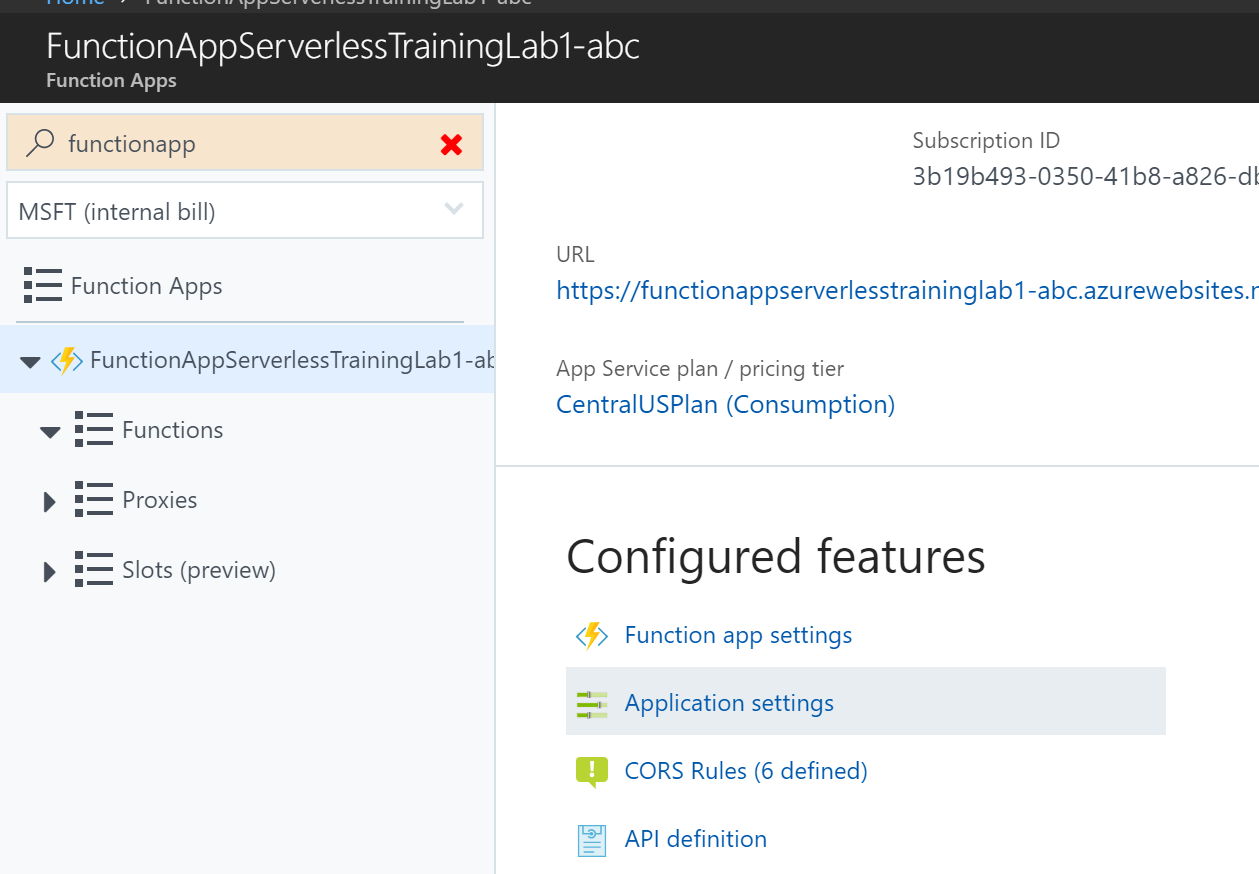
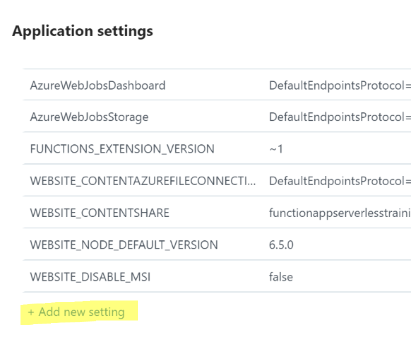
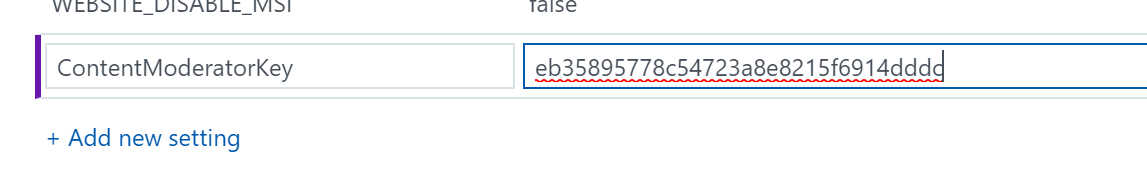
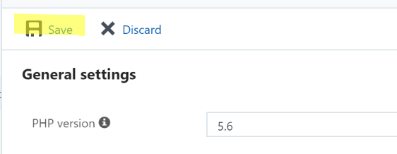
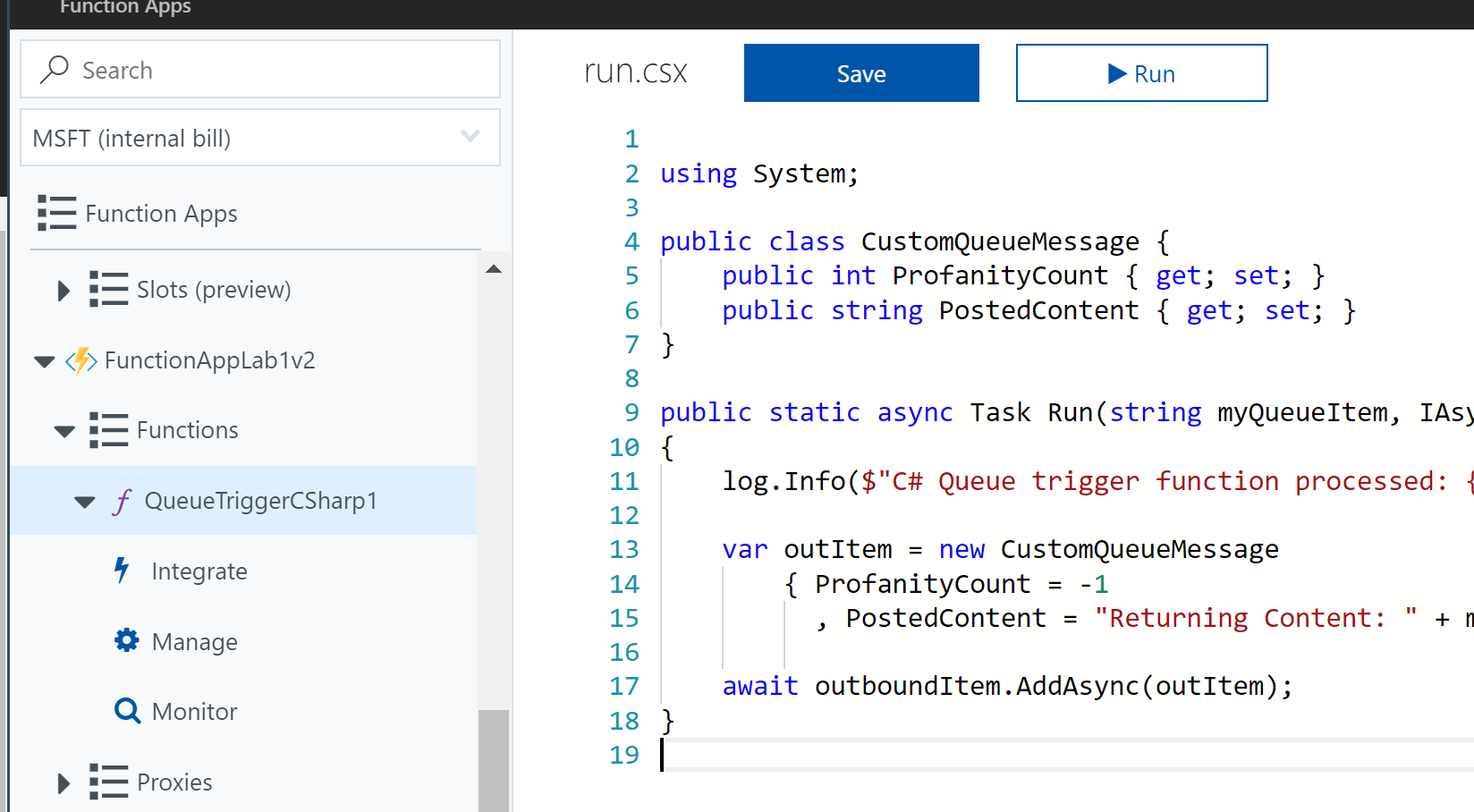
1. Search for **Content Moderator** and select Content Moderator and click **Create**.  
   
2. In the Create pane prompts:
   * In the **Name** field, use a unique name for your function
   * In the **Subscription** field, select your subscription
   * In the **Location** field, use the default of **Central US** (or whichever location is closest for you)
   * In the **Pricing tier** field, select **F0** which is free at 1 call per second, also you can only have one of these per subscription.
   * In the **Resource Group** field, Use existing group previously selected for the Function App for ease of management.
   * Check the **I confirm** notice box.

After this click **Create** to accept these values and move to the next step

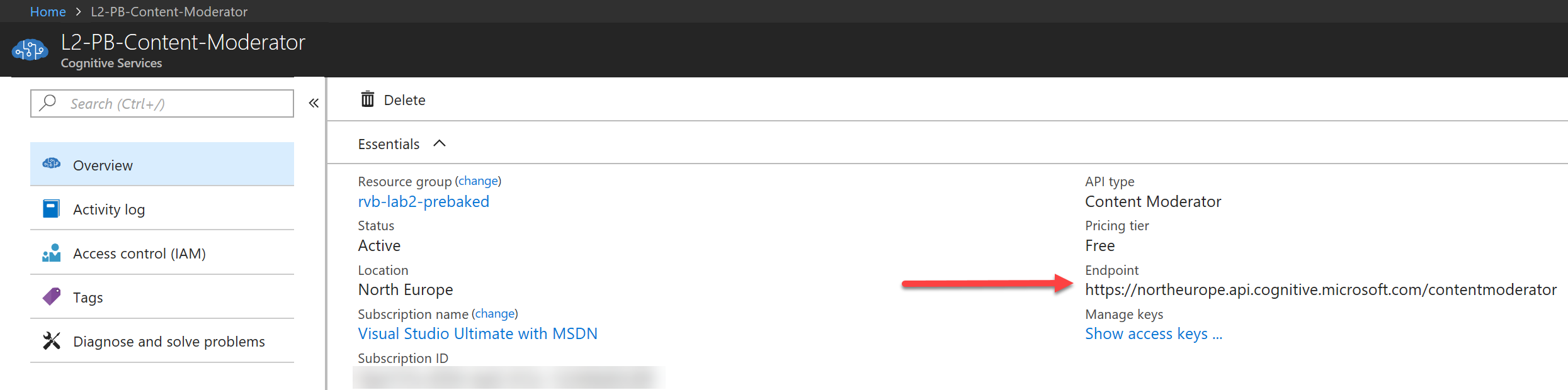


1. Once the Content Moderator is created you can click **Go to resource**.



1. Then click the **Keys** link  
   
2. You will need a Key to be used in the Function app. So, **copy** the Key 1 and keep it in a safe place to be used later.  
   
3. Next, go back to the Function App and you can start editing the app to include the calls to the Content Moderator. First let’s add the Key into the **Application settings**. Select the Function and then click on the **Application settings** link on the right.  
   
4. Under Application settings click **Add new setting**.  
   
5. Use a name for the setting such as **ContentModeratorKey** and paste the **Key value from Step 5**.  *Don’t use the Key shown. It will not be valid.*  
   
6. Then click on the **Save** button on the top to commit the setting.  
   
7. Next, edit the code to call the ContentModerator API. So, navigate back to the editor for the function  
   
8. Make your Function look like the gist, <https://gist.github.com/criter/c3aa22561d9a8d451fc93cd0d4ab4d90#file-functionapplab1-csx-L2> . You’ll see in here code:
   1. That uses the API key
   2. Calls the Content Moderator API
   3. Parses the Profanity Count
   4. Creates the JSON structure
   5. Returns the outbound Item

NOTE: Please make sure that the Cognitive Services URL in the code points to the region where you created your Content Moderator.



1. Compile and Run it. Test it with different inputs, include good content and negative content.
2. View the Output with Azure Storage Explorer in the output queue.