Azure Functions Lab

# Prereqs

* Visual Studio Code
* Azure functions core tools
* Postman (make sure to dismiss welcome screen after installation)

# Lab 1: Creating your first function app

**Creating your first function app: in this section you will learn how to your first function app, we will be using Azure Functions Core Tools and VS code**

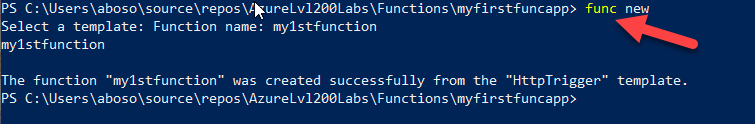
## Creating function app

* Open windows PowerShell on your machine
* Type CD:\path
* Type: func init and hit enter key
* Select dotnet as worker runtime
* Once the function app creation is completed
* Type: dir and hit enter key
* Take a look at the created file structure

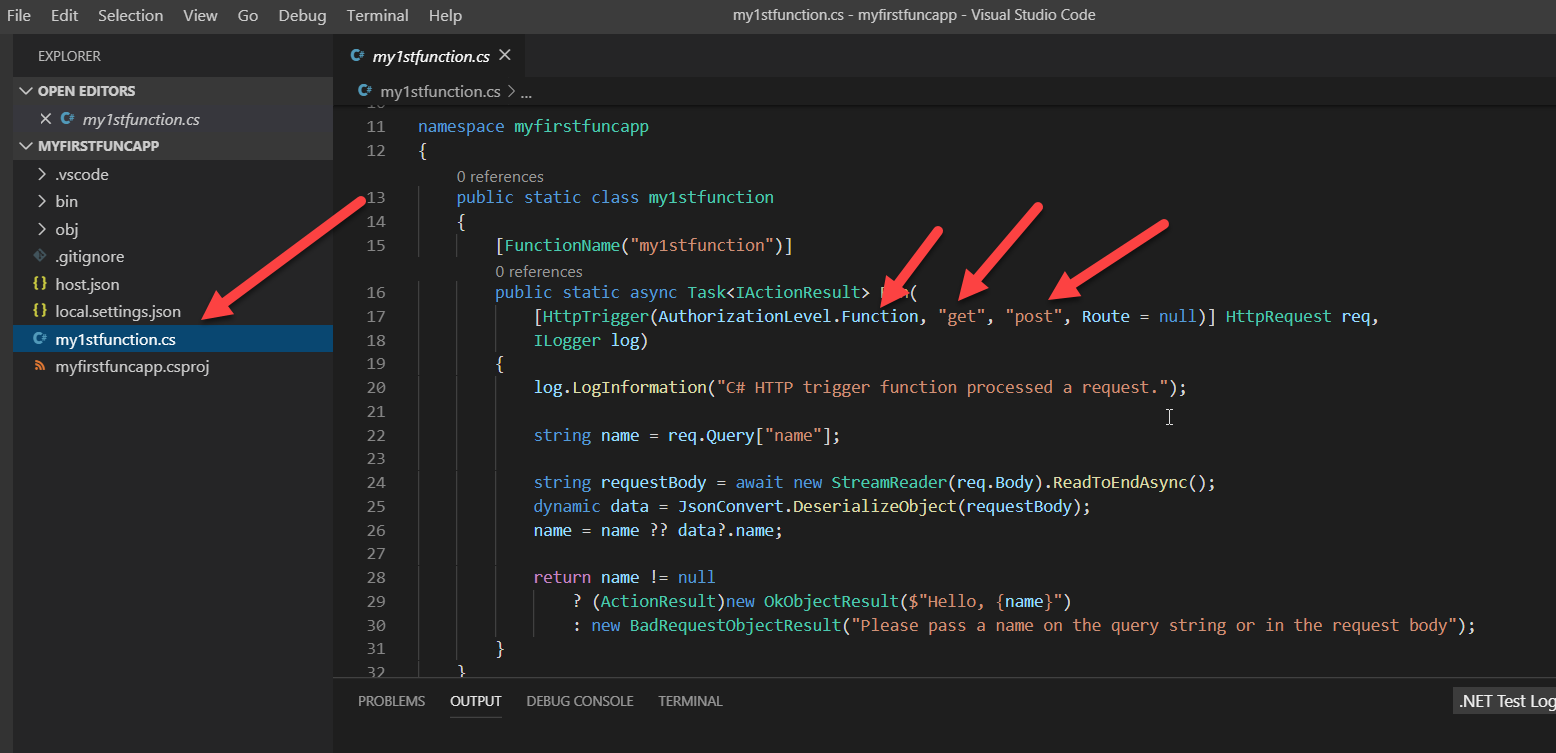
Machine generated alternative text:
PS C: cd . 
PS C: func init 
Select a worker runtime: 
dotnet 
Writing C: vscode\extensions.json 
PS C: dir 
Directory: C: 
Mode 
LastWriteTime 
10/14/2019 
10/14/2019 
10/14/2019 
10/14/2019 
10/14/2019 
4 
4 
4 
4 
4 
: 26 
: 26 
: 26 
: 26 
: 26 
Al.l 
Al.l 
Al.l 
Al.l 
Length 
4626 
26 
163 
634 
Name 
.vscode 
. gitignore 
host. j son 
local. settings . j son 
myfirstfuncapp. csproj 

## Creating you first function

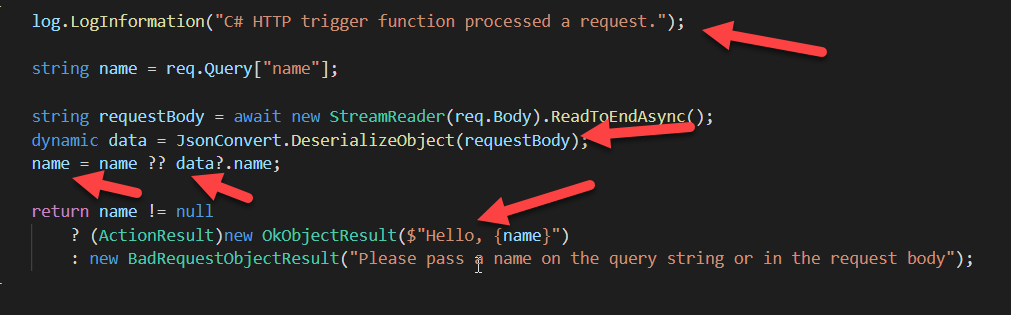
* Type func new and hit enter key
* Choose HttpTrigger and hit enter key
* Provide a name for the function (e.g. my1stfunction)



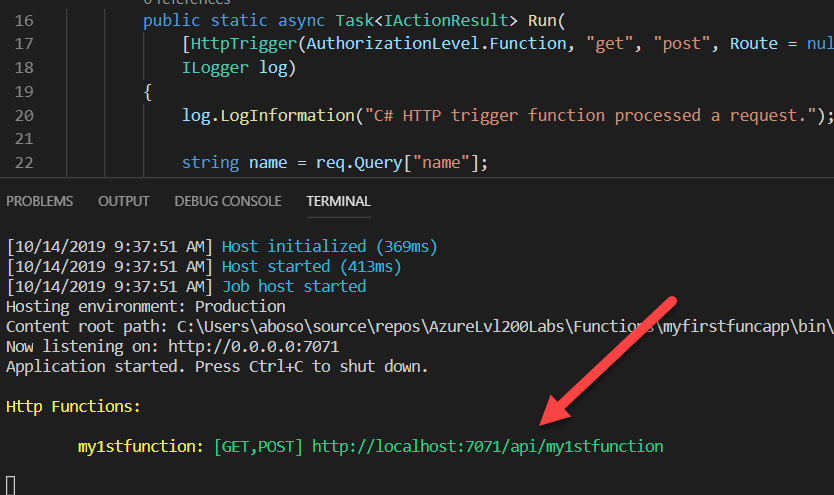
* Type code . (including the dot) and hit enter to open the function app in VS Code.
* Notice the following
  + On the left side.. Under explorer you can find the function you have just created
  + Notice that **AuthorizationLevel** is set to function which means you need to include a secret when calling the function
  + Check the allowed http request are both "get" and "post"



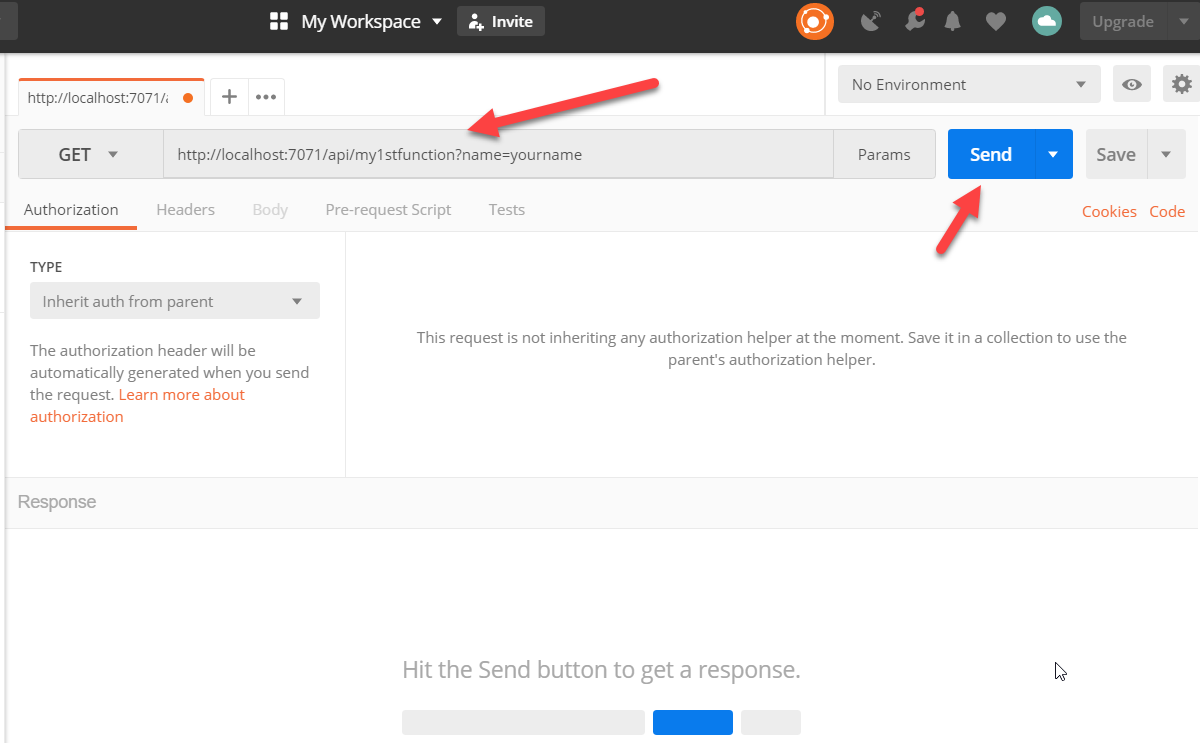
* Notice the following
  + Log.loginformation is used to save a log entry
  + Notice that the function expect a name variable either in the query string or the request body
  + The it will respond with a Hello, {name variable value}



* Now select from top menu select debug and then select start debugging
* From terminal window copy the function url to test it



* Start Postman (provide your credentials if not logged-in already)
* Add send a GET request to the function providing your name as a parameter and check the response



* Check the response in the **Body** tab

Machine generated alternative text:
the request. Learn more about 
authorization 
Body 
Pretty 
Cookies 
Raw 
Headers (4) 
Preview 
Test Results 
Text 
1 Hello, 
yourname 

* Stop the debugger

# Lab 2: Cyberbullying free chat zone

**Description**: This lab shows you how to use Functions with Logic Apps and Cognitive Services on Azure to run sentiment analysis on chat messages sent using the Service Bus you created in previous labs. An HTTP triggered function categorizes chat messages based on the sentiment score. An email is sent when poor sentiment is detected.

## Setup the lab environment

To setup the message bus required for the lab

* Open a PowerShell window
* Execute the **Create-CacheLab** PowerShell script found in the **0\_setup** folder, with your lab ID:  
  .\Create-ServerlessLab.ps1 -participantId {your\_id}

This will create a new resource group, az{your\_id}-cache-rg

The resource group will contain a service bus with two queues.

## Create a Cognitive Services API Resource

* + The Cognitive Services APIs are available in Azure as individual resources. Use the Text Analytics API to detect the sentiment of the tweets being monitored.
    - Sign in to the [Azure portal](https://portal.azure.com/).
    - Click + **Create a resource** in the upper left-hand corner of the Azure portal.
    - Click **AI + Machine Learning** > **Text Analytics**. Then, use the settings as specified in the table to create the resource.

|  |  |
| --- | --- |
| Setting | Suggested value |
| Name | az{your\_id}TextAnalytics |
| Location | West Europe |
| Pricing tier | F0 |
| Resource group | az{your\_id}-serverless-rg |

* + - Click **Create** to create your resource.
  + Once created, navigate to the Text Analytics resource blade. On **Overview**, copy the value of the **Endpoint** to a text editor.  
    This endpoint is used when creating a connection to the Cognitive Services API.
  + In the left navigation column, click Keys, and then copy the value of **Key 1** and set it aside in a text editor.  
    You use the key to connect to your Cognitive Services API.

## Create the function app

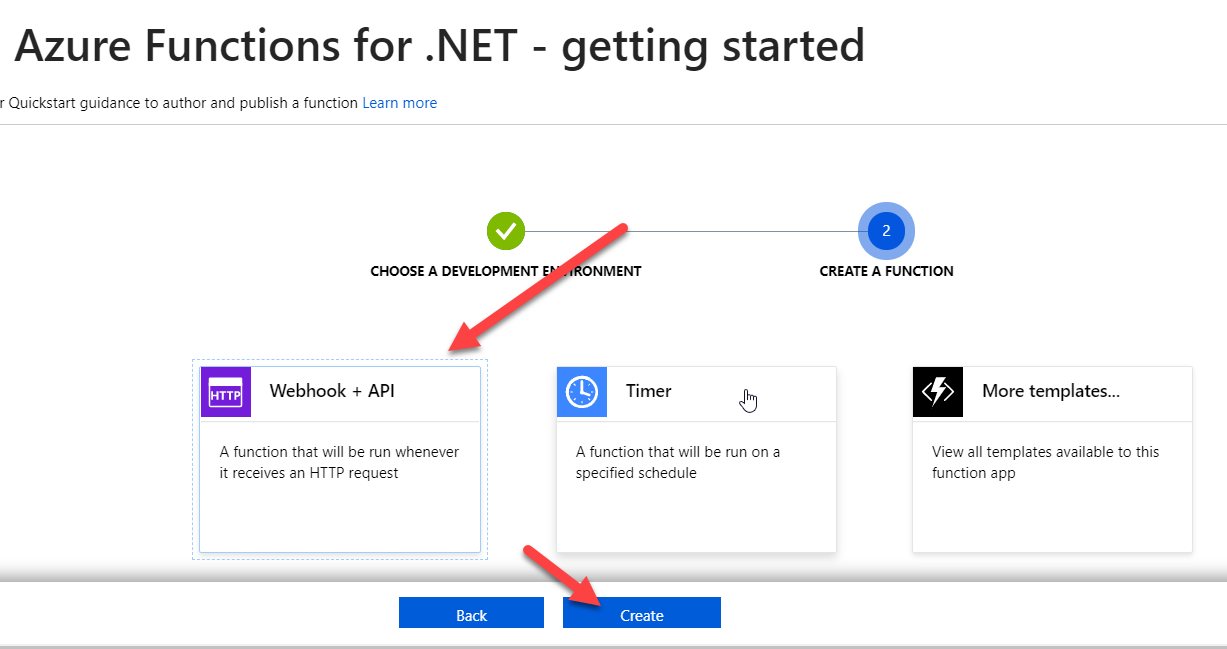
* + Select the Create a resource button found on the upper left-hand corner of the Azure portal, then select **Compute** > **Function App**.
  + Use the function app settings as specified in the table below

|  |  |
| --- | --- |
| Setting | Suggested value |
| App name | az{you\_id}-serverless-funcapp |
| Subscription | *Your subscription* |
| Resource Group | **Use Existing**  az{your\_id}-serverless-rg |
| OS | Windows |
| Hosting plan | Consumption plan |
| Location | West Europe |
| Runtime stack | .NET Core |
| Storage | *Leave as default* |
| Application Insights | *Leave as default* |

* + Select Create to provision and deploy the function app.
  + Once the Function App has been created go to its resource blade to view your new function app.  
    **Note**: Do not use the "Go to resource" button to view the Function App. Rather, search for your function app, or navigate using the left-hand menu.

## Create an http triggered function that categorizes chat message sentiment.

* + Expand your function app and create a new function by click the **+** button next to **Functions**.
  + Next, select **Webhook + API** and click **Create**.



* + Replace the contents of the run.csx file with the following contents, then click **Save**:

#r "Newtonsoft.Json"

using System;

using System.Net;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Logging;

using Microsoft.Extensions.Primitives;

using Newtonsoft.Json;

public static async Task<IActionResult> Run(HttpRequest req, ILogger log)

{

string category = "Allow";

string requestBody = await new StreamReader(req.Body).ReadToEndAsync();

log.LogInformation(string.Format("The sentiment score received is '{0}'.", requestBody));

double score = Convert.ToDouble(requestBody);

if(score < .4)

{

category = "Block";

}

else if (score < .7)

{

category = "Audit";

}

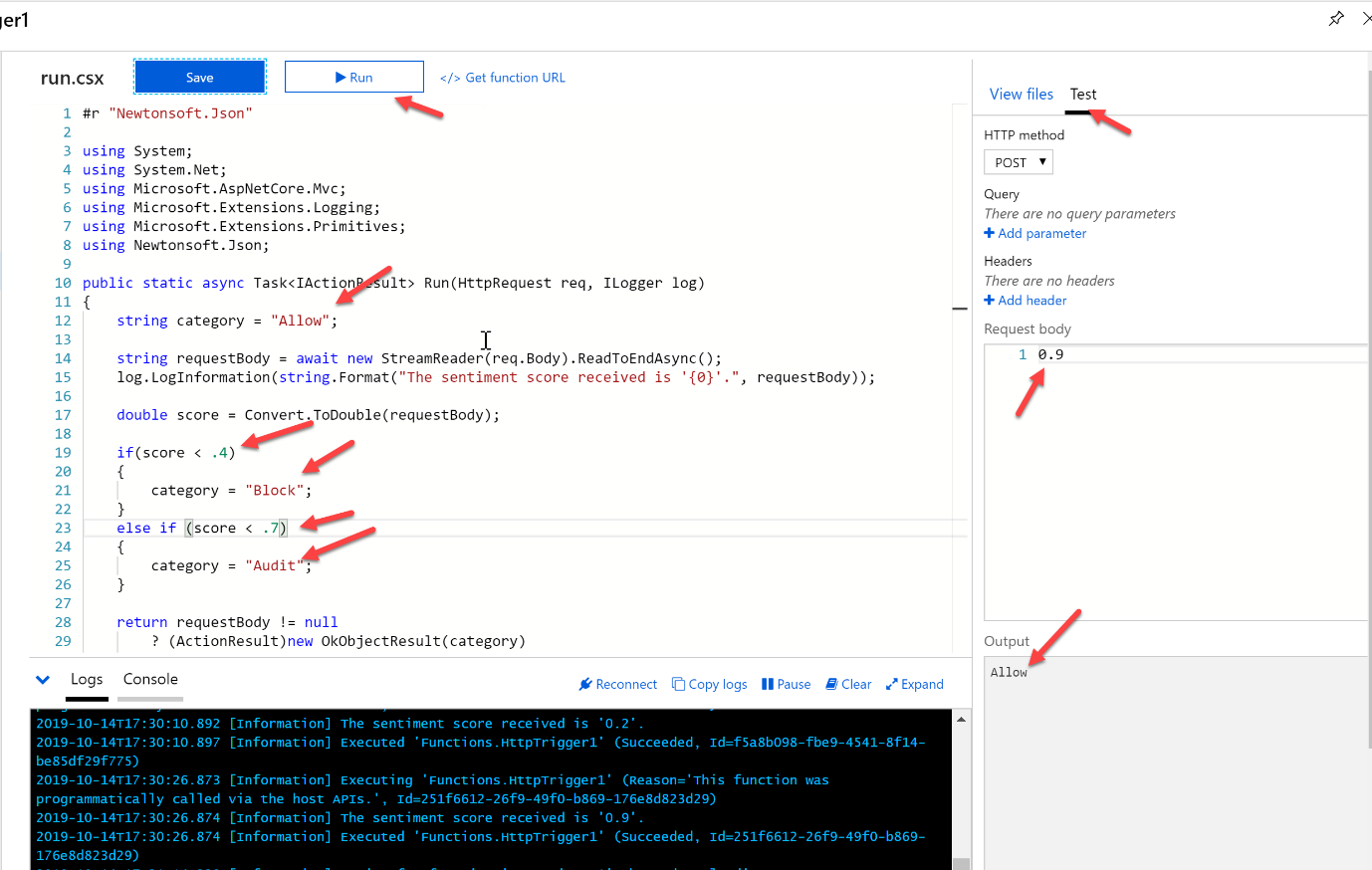
return requestBody != null

? (ActionResult)new OkObjectResult(category)

: new BadRequestObjectResult("Please pass a value on the query string or in the request body");

}

* + From under test on the right side.. In the request body type 0.9 and then click on run and notice the output.. A sentiment of 0.9 means a positive sentiment has been detected.



## Create a logic app that connects to service bus

* + In the Azure portal, click the **+ Create a resource** button found on the upper left-hand corner of the Azure portal.
  + Click **Web** > **Logic App**.
  + Use the settings as specified in the table.

|  |  |
| --- | --- |
| **Setting** | **Suggested value** |
| **Name** | az{your\_id}-serverless-logicapp |
| **Resource group** | az{your\_id}-serverless-rg |
| **Location** | West Europe |

* + Once you have entered the proper settings values, click **Create** to create your logic app.
  + After the app is created. Then in the Logic Apps Designer, scroll down and click the **Blank Logic App** template.

You can now use the Logic Apps Designer to add services and triggers to your app.

## Connect to ServiceBus

We will now connect the logic app to the service bus we created in the lab setup.

* + Navigate to your Logic App's blade. This should open by default with the Logic App Designer. If not go to **Development Tools** -> **Logic app designer**.
  + Select **When a message is received in a service bus** as the starting trigger for the workflow.
  + Click Create to create a connection to a message bus.
  + Provide a name for the connection: ServiceBusConnection.
  + Select az{your\_id}-cache-servicebus, which was created previously during setup…
  + and utilise the RootManageSharedAccessKey., as the credentials to connect to the bus.
  + Click **Create**
  + Click **Continue**
* Select the Queue name **All messages**.
* Change the polling interval to every 1 second.

Now your app is connected to ServiceBus. Next, you connect to text analytics to detect the sentiment of collected chat messages.

## Add sentiment detection to the logic app.

* + Click **New Step**, and then **Add an action**.
  + In Choose an action, type **Text Analytics**, and then click the **Detect sentiment** action.
  + Type a connection name such as sentimentAnalysisConnection, paste the key for your Cognitive Services API and endpoint which you set aside in a text editor, and click **Create**.
  + Add a new parameter for **Text**, and in the textbox labeled **The text to analyze**, select **Content** from the popup window.

This will extract the content from the service bus message, and send it to Text Analytics for sentiment detection. Now that sentiment detection is configured, we can add a connection to our function that consumes the sentiment score output.

## Connect the logic app to the function.

* + In the Logic Apps Designer, click **New step**.
  + Filter on **Azure Functions** and select **Choose an Azure function**.
  + Select the function you created for this lab.
  + In **Request Body**, click **Score** (from Detect Sentiment) and then **Save**.

Now, your function is triggered when a sentiment score is sent from the logic app. A decision is returned to the logic app by the function whether to Allow, Block, or Audit the chat message.

Next, we will add an email notification that is sent when a sentiment value of block is returned from the function.

## Forward all but Blocked messages to the chat

* + In the Logic Apps Designer, click **New step**.
  + Filter on **Control** and select **Condition**.
    - Click **Choose a value**, then click **Body**. Select **is not equal to**, and in **Choose a value** type **Block**, and click **Save**.
  + In **If true**, click **Add an action**,
    - Filter on **Control** and select **Send message**.
    - This will automatically re-use the service bus connection we set up earlier.
    - In **Queue/Topic name** select **filteredmessages**
    - From the **Add new parameter** drop-down, select **Content**.
    - Click into the **Content of the message** text box, select the **Content** from the message bus trigger.
* Click **Save**.

## Run the chat application

Copy the connection string from Service Bus:

* In the Azure portal, go to your service bus's blade.
* Select **Settings** -> **Shared access policies** from the left menu
* Select the RootManageSharedAccessKey from the list
* Copy the Primary Connection String.

Open the **ServiceBus.sln** solution found in the **1\_lab** folder in Visual Studio.

* Paste the service bus connection string in the values for SendServiceBusConnectionString and ReceiveServiceBusConnectionString.

Build and run the sample.

After typing a "nice" message you should see it being echoed back to you. Typing a nasty message gets filtered out.

# Going further

Explore the logic app and see the run history for each message. Work through the workflow to see the results at each step.

Try setting up a second condition and sending a message to an email account (using the Outlook 365, Outlook.com, or GMail logic app connectors) when a message is marked for Audit.

Swap you receive connection string with that of the person sitting next to you to create a non-abusive chat application.