

Azure Discovery Days 2019

Data Analytics & Near Real Time Intelligence with Azure - Hands-On Lab Guide

Lab 3: Stream Enrichment

Summary

In this hands-on lab, you will:

1. Set up two stream ingestion endpoints
2. Set up a streaming event simulator that will send events to the first stream ingestion endpoint
3. Enrich the stream with insights from a pre-trained AI Cognitive Service
4. Send the enriched stream on to the second stream ingestion endpoint

About this Lab

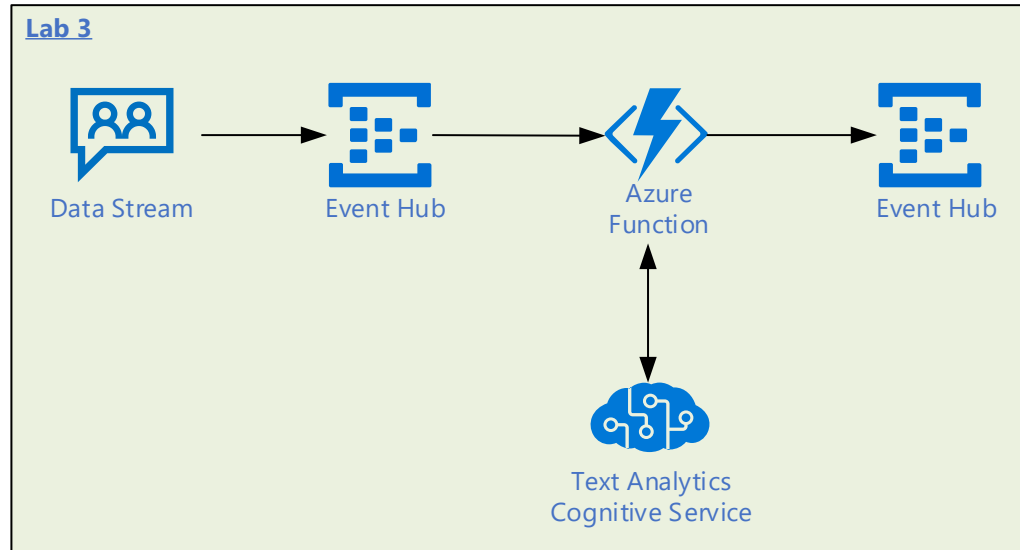
The streaming event simulator is meant to simulate the flow of data from taxis in the future, to include both trip data as well as user feedback as free-form comments about the trip.

References

- Azure Event Hubs Documentation: <https://docs.microsoft.com/azure/event-hubs/>
- Azure Functions Documentation: <https://docs.microsoft.com/azure/azure-functions/>
- Text Analytics Cognitive Service Documentation: <https://docs.microsoft.com/azure/cognitive-services/text-analytics/>
- This lab also uses some open-source helper libraries, including to wrap around the Text Analytics Cognitive Service REST API.
 - Text Analytics helper library: <https://github.com/plzm/azure-cognitive/tree/master/src/textanalytics>
 - Text Analytics helper Nuget: <https://www.nuget.org/packages/pelazem.azure.cognitive.textanalytics/>

Architecture for this Lab

The tasks in this lab cover the following components of the overall architecture.



Task 1 – Set up two stream ingestion endpoints

In this lab, we will use Azure Event Hubs for stream ingestion.

The first stream ingestion endpoint will receive inbound trip messages from taxi devices. The stream will then be enriched (see Task 2), and then forwarded to a second stream ingestion endpoint. (Lab 4 includes further work with the enriched stream after the second stream ingestion endpoint.)

Begin in the Resource Group where you have been working so far, and click “+ Add”, similarly to previous deployments, to deploy new resources.

In the search box, type “Event Hubs” followed by Enter. Click on the Event Hubs entry, then “Create” on its product blade.

The screenshot shows the Azure portal search results for 'Event Hubs'. The breadcrumb navigation at the top reads 'Dashboard > DiscoveryDay > Everything > Event Hubs'. The search bar contains 'Event Hubs'. Below the search bar are filters for 'Pricing' (All), 'Operating System' (All), and 'Publisher' (All). The results table has columns 'NAME', 'PUBLISHER', and 'CATEGORY'. The first result is 'Event Hubs' by Microsoft, categorized under 'Analytics'. An orange arrow points from the search bar to this result. Below it are 'Event Grid Domain' and 'Surface Hub'. At the bottom, there is a 'Related to your search' section and a 'Create' button. An orange arrow points from the 'Event Hubs' result to the 'Create' button. On the right side, the 'Event Hubs' product blade is partially visible, showing a description and a 'Save for later' button.

NAME	PUBLISHER	CATEGORY
Event Hubs	Microsoft	Analytics
Event Grid Domain	Microsoft	
Surface Hub	Microsoft	Management Tools

Provide appropriate values on the blade to create an Event Hubs Namespace. Specifically:

- Pricing Tier: Standard
- Subscription, Resource Group and Location: choose the ones you have been using so far.
- Throughput Units (TUs):
 - Set to 1
 - Enable Auto-Inflate: yes (checked)
 - Auto-Inflate Maximum TUs: 2
 - Each TU provides 1MB/sec or 1,000 events/sec ingress. These settings will be ample for this lab.

Then click Create.

ps > Create Namespace

Create Namespace

Event Hubs

* Name

azdiscdays2019 ✓

.servicebus.windows.net

* Pricing tier ([View full pricing details](#))

Standard (20 Consumer groups, 1000 Brok... ▼

☐ Enable Kafka ⓘ

☐ Make this namespace zone redundant ⓘ

* Subscription

Azure-FTE-201609 ▼

* Resource group

DiscoveryDay ▼

[Create new](#)

* Location

East US ▼

* Throughput Units

1

☒ Enable Auto-Inflate ⓘ

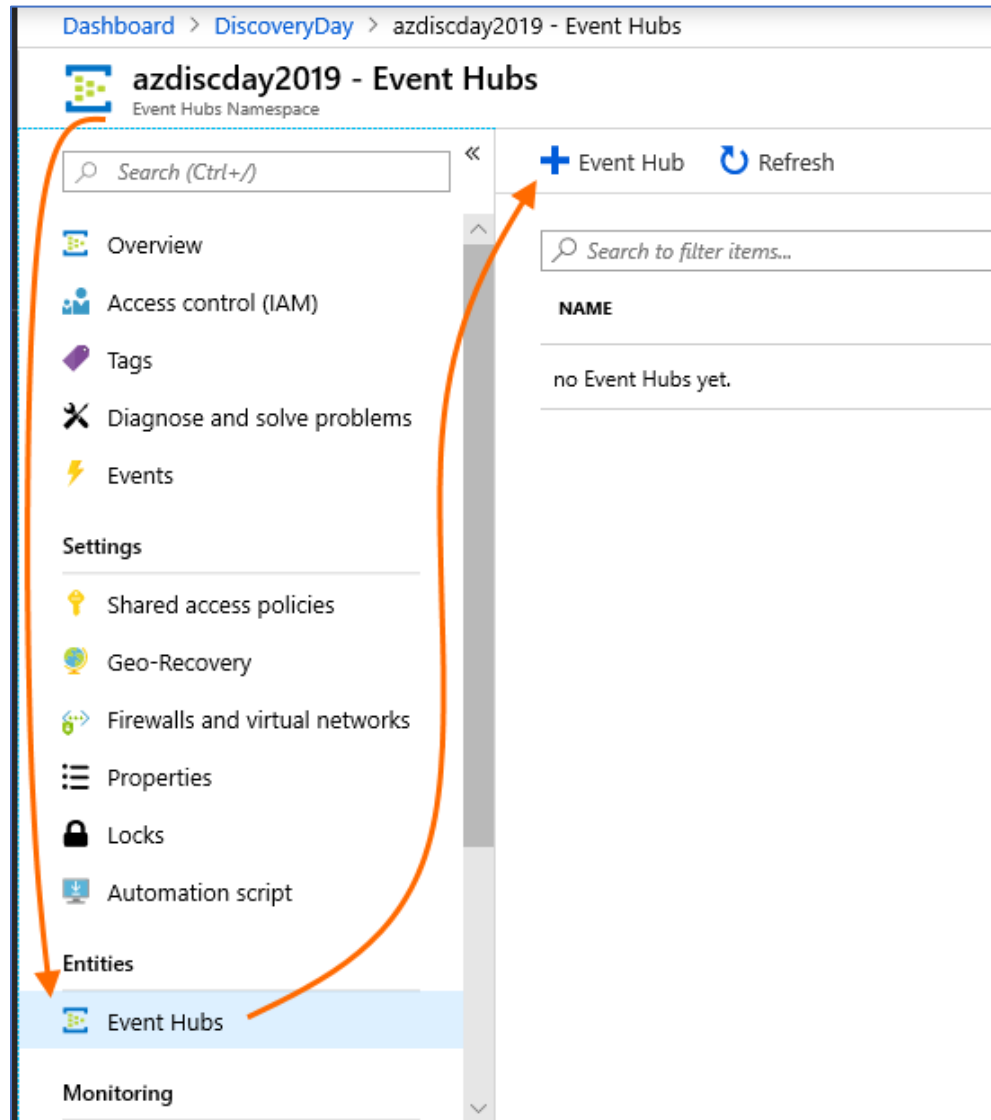
Auto-Inflate Maximum Throughput Units

2

Create

When deployment completes, click on the new Event Hubs Namespace resource in your Resource Group.

You will now create two Event Hubs in this Event Hubs Namespace. This first Event Hub will be the endpoint to which the taxi device simulator will send messages. To start, click on “Entities/Event Hubs”. On the Event Hubs view, click “+ Event Hub”.



In the “Create Event Hub” view, provide a name for this Event Hub. Other inputs are good with their default values.

Optionally, you can configure Capture so that inbound messages, in addition to being routed to the Azure Function you will create later in this lab, are also stored in Azure Storage. However, Capture is not required for this or other labs. Then click “Create”.

Dashboard > DiscoveryDay > azdiscday2019 - Event Hubs > Create Event Hub

Create Event Hub

Event Hubs

* Name ⓘ
ehInbound ✓

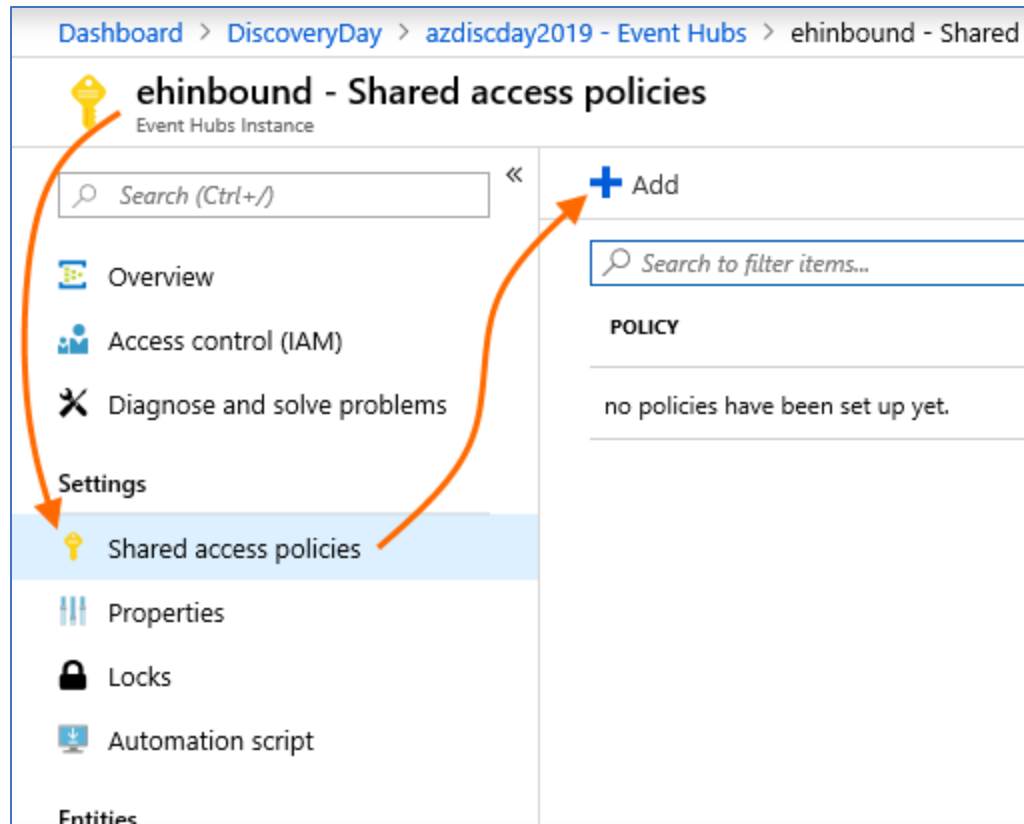
Partition Count ⓘ
2

Message Retention ⓘ
1

Capture ⓘ
On Off

Create

After the Event Hub is created, you will see it in the Event Hubs list on the view where you just clicked “+ Event Hub”. Now click the Event Hub you just created, then click “Shared access policies”, then “+ Add”.



Specify a name for the policy and check only “Listen”, then click “Create”.

This policy will allow the Azure Function you will create in a later task to listen for event messages sent to this endpoint by the taxi device simulator. As with the namespace-level policy you created earlier, in this case the Azure Function will only need to Listen for messages – it does not need to Send messages to this endpoint, nor does it need to Manage it.

Add SAS Policy
Event Hubs

* Policy name
ehInbound-Listen ✓

☐ Manage

☐ Send

☒ Listen

Create

After the Listen policy is created, create a second policy following the same process. This time, it will be a **Send** policy: name it accordingly, check only the “**Send**” checkbox, and then click “Create”. You will need the connection string for this **Send** policy to deploy the taxi device simulator later in this lab.

Next, return to the Event Hubs Namespace “Event Hubs” view, where your first Event Hub is shown in the list.

Now, create another Event Hub. This will be the second messaging endpoint shown in the architecture diagram (see page 1 of this document). The Azure Function you will create in a later task will send the enriched message stream to this second Event Hub.

Similarly to when you created policies for the first Event Hub, create a Shared access policy for this second Event Hub. **Create a policy with both Send and Listen.** The Azure Function will use this policy to **send** enriched messages onward, and in lab 4 you will create an Azure Stream Analytics resource that will use this policy to **listen** for the enriched messages.

When you are done, you should see two Event Hubs listed in the Event Hubs Namespace list.

azdisccday2019 - Event Hubs
Event Hubs Namespace

Search (Ctrl+ /)

Event Hub Refresh

Search to filter items...

NAME	STATUS
ehenriched	Active
ehinbound	Active

Entities

Event Hubs

Monitoring

Alerts

Metrics

To conclude this task, you can optionally click into each of the two Event Hubs. In each Event Hub, click “Shared access policies”. Then click the policies you created to show their properties. Copy the policy connection strings and save them in a scratch pad area for later use - or, of course, you can always come back to this view when you need one of the connection strings.

Dashboard > DiscoveryDay > azdisccday2019 - Event Hubs > ehInbound-Listen

ehinbound - Shared access policies

Event Hubs Instance

Search (Ctrl+ /) << + Add

Search to filter item

POLICY

ehInbound-Listen

Settings

- Shared access policies
- Properties
- Locks
- Automation script

Entities

SAS Policy: ehInbound-Listen

Save Discard More

☐ Manage

☐ Send

☒ Listen

Primary key

PW4U2gsGD2nH0apVpTPQEDTXsZJaC...

Secondary key

VV23zK5CBJQeaQMvVzkW6ErHm+nuh...

Connection string-primary key

Endpoint=sb://azdisccday2019.serviceb...

Connection string-secondary key

Endpoint=sb://azdisccday2019.serviceb...

ehenriched - Shared access policies
Event Hubs Instance

Search (Ctrl+/)

Overview
Access control (IAM)
Diagnose and solve problems
Settings
Shared access policies
Properties
Locks
Automation script
Entities

+ Add

Search to filter items...

POLICY	CLAIMS
ehenriched_send_listen	Send, Listen

Manage

☒ Send

☒ Listen

Primary key
oRgK/cLvya3bZSC

Secondary key
FhtRo29k4jWuVm

Connection string-p
Endpoint=sb://az

Connection string-s
Endpoint=sb://az

In summary, when you have successfully deployed the following, then this task is complete:

1. Event Hub for inbound taxi device messages
 - a. Send Shared Access Policy for the Event Hub, to be used by the taxi device simulator to send in messages.
 - b. Listen Shared Access Policy for the Event Hub, to be used by the Azure Function to listen for incoming messages.
2. Event Hub to forward enriched taxi device messages
 - a. Shared Access Policy with Send and Listen for the Event Hub, to be used by the Azure Function to **send** messages it has enriched and by Stream Analytics(in lab 4) to **listen** for these enriched messages.

Note: you may be wondering why you created *two distinct policies* (one for Send, one for Listen) for the inbound Event Hub, but only *one policy* (with both Send and Listen) for the enriched Event Hub?

1. To show you different approaches
2. Because the inbound Event Hub needs external message senders to *send* to it, but we probably don't want to configure those devices with a policy that would also allow them to *listen* to sent-in messages! The devices would be out of our immediate control in taxi environments, so we are providing the least privilege possible to them: send only, with no ability to listen for what was sent.
By contrast, the enriched Event Hub is only being used by components entirely in our environment and under our control, so a combined policy presents much less of a security concern there.

Task 2 – Deploy Text Analytics Cognitive Service

In this task, you will deploy an Azure Text Analytics Cognitive Service. This will be used by the Azure Function (which you will deploy in task 3) to perform text analytics on customer comments that are part of the messages sent in by the taxi devices.

In your Resource Group, click “+ Add” again to add a new resource. Type “Text Analytics” in the search box, then Enter.

The screenshot shows the Azure Marketplace interface. At the top, the breadcrumb navigation is "Dashboard > DiscoveryDay > Marketplace > Text Analytics". The main heading is "Marketplace". Below it, there is a search bar containing "Text Analytics" and three filter dropdowns: "Pricing" (set to "All"), "Operating System" (set to "All"), and "Publisher" (set to "All"). Below the filters, a table displays the search results. The table has three columns: "NAME", "PUBLISHER", and "CATEGORY". One result is shown: "Text Analytics" by "Microsoft" in the "AI + Machine Learning" category. An orange arrow points from the search bar to this result. To the right of the table, there is a detailed view for "Text Analytics" by Microsoft. It includes a description: "Use a few lines of code to easily analyze any kind of text." and lists features: "Sentiment analysis", "Key phrase extraction", and "Language detection". At the bottom of this panel, there is a "Save for later" button and a "Create" button. An orange arrow points from the "Text Analytics" result in the table to the "Create" button.

NAME	PUBLISHER	CATEGORY
Text Analytics	Microsoft	AI + Machine Learning

Text Analytics
Microsoft

Use a few lines of code to easily analyze any kind of text.

Sentiment analysis
Find out what users think about your text.

Key phrase extraction
Automatically extract key phrases from text.

Language detection
Determine what language a piece of text is written in.

[Save for later](#)

[Create](#)

On the Create blade, provide a name. Select the S0 pricing tier, and ensure that your Resource Group is selected. Then click “Create”.

The screenshot shows the 'Create' blade for Text Analytics in the Azure portal. The breadcrumb navigation at the top reads 'Dashboard > DiscoveryDay > Marketplace > Text Analytics'. The blade title is 'Create Text Analytics'. Below the title, there are five required fields, each marked with a red asterisk: 'Name' (containing 'discdays-ta' with a green checkmark), 'Subscription' (containing 'Azure-FTE-201609'), 'Location' (containing 'East US'), 'Pricing tier (View full pricing details)' (containing 'S0 (25K Transactions per 30 days)'), and 'Resource group' (containing 'DiscoveryDay'). Below the 'Resource group' field is a link 'Create new'. At the bottom of the blade are two buttons: 'Create' (a blue button) and 'Automation options' (a text link). Five orange arrows are overlaid on the image, pointing to the 'Name' field, the 'Subscription' field, the 'Location' field, the 'Pricing tier' field, and the 'Create' button.

After creation completes, locate the new cognitive service resource in your Resource Group and click on it. Begin on the “Overview” blade. Locate the Endpoint and copy its value to a scratchpad area. Then click on the “Keys” blade and copy either API key value to a scratchpad area. You will need both pieces of information when creating the Azure Function that will use this cognitive service.

Dashboard > Resource groups > DiscoveryDay > discdays-ta

discdays-ta

Cognitive Services

Search (Ctrl+/) << Unavailable setting Delete

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems

RESOURCE MANAGEMENT

- Keys

Resource group [\(change\)](#)
DiscoveryDay

Status
Active

Location
East US

Subscription [\(change\)](#)
Azure-FTE-201609

Subscription ID
e61e4c75-268b-4c94-ad48-237aa3231481

API type
Text Analytics

Pricing tier
Standard

Endpoint
<https://eastus.api.cognitive.microsoft.com/text/analytics/v2.0>

Manage keys
[Show access keys ...](#)

Dashboard > Resource groups > DiscoveryDay > discdays-ta - Keys

discdays-ta - Keys

Cognitive Services

Search (Ctrl+/) << Regenerate Key1 Regenerate Key2

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems

RESOURCE MANAGEMENT

- Keys

NAME
discdays-ta

KEY 1
620acdd81aa349ceb1fe413df187a876

KEY 2
8dd348910347440d8d7e020f1fa70b38

When you have obtained both the API Endpoint URL and the API key, this task is complete. Please return to your Resource Group for the next task.

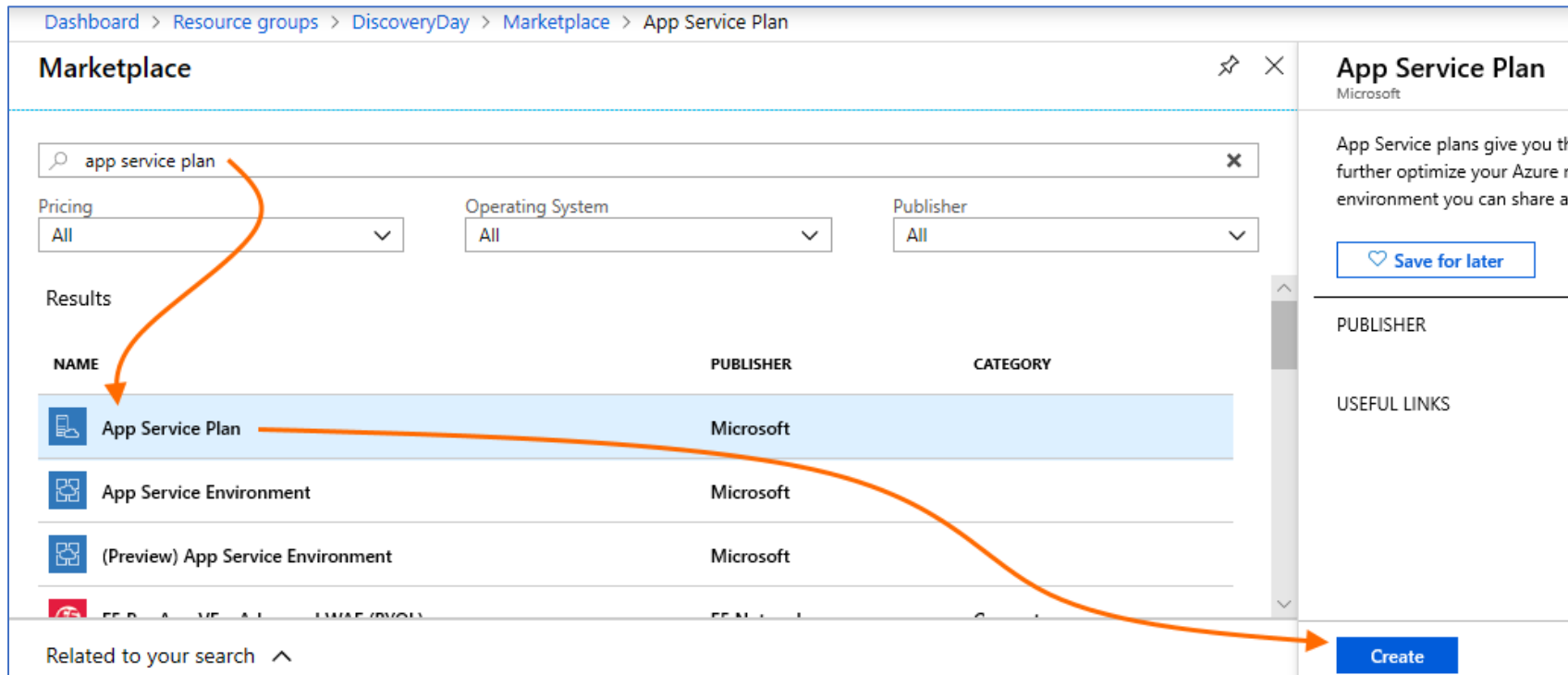
Task 3 – Deploy Azure Function to Process and Enrich Taxi Messages

In this task, you will deploy the Azure Function shown in the architecture diagram on page 1.

This Function will be triggered to run every time a taxi device message is received by the first Event Hub you deployed in task 1. It will then get the customer's comments from the message, and run those comments through the text analytics cognitive service you created in task 2. Then, the text analytics results will be added to the taxi message – i.e. the inbound message is enriched with text analytics results. Lastly, the enriched message is sent to the second Event Hub you deployed, where it will be further processed in lab 4.

Azure Functions can run either in "Consumption" plans or in App Service Plans. You will deploy an App Service Plan, which provides the ability to keep a Function running ("Always On"), which is consistent with our scenario of taxi devices sending in messages around the clock. After the App Service Plan is deployed, you will then deploy the Azure Function that uses it.

In your Resource Group, click "+ Add" and type "App Service Plan" into the search box, then Enter. Select "App Service Plan", then click "Create".



On the create blade, provide a name for the App Service Plan. Then, ensure your Resource Group is selected; leave Operating System at Windows; set the Location to the Azure region you have been using so far; then click Pricing Tier. On that view, select P1V2 and click “Apply”. Then, click “Create”.

Dashboard > Resource groups > DiscoveryDay > Marketplace > App Service Plan > New App Service Plan >

New App Service Plan

Create a plan for the web app

* App Service plan
azdisccday-asp ✓

* Subscription
Azure-FTE-201609


* Resource Group ⓘ
☐ Create new ☒ Use existing
DiscoveryDay


* Operating System
Windows

* Location
East US

* Pricing tier
P1v2 PremiumV2 >

Create Automation options

**Dev / Test**
For less demanding workloads

**Production**
For most production workloads

Recommended pricing tiers

S1 100 total ACU 1.75 GB memory A-Series compute equivalent 74.40 USD/Month (Estimated)	P1V2 210 total ACU 3.5 GB memory Dv2-Series compute equivalent 148.80 USD/Month (Estimated)	P2V2
P3V2 840 total ACU 14 GB memory Dv2-Series compute equivalent 595.20 USD/Month (Estimated)		

[See additional options](#)

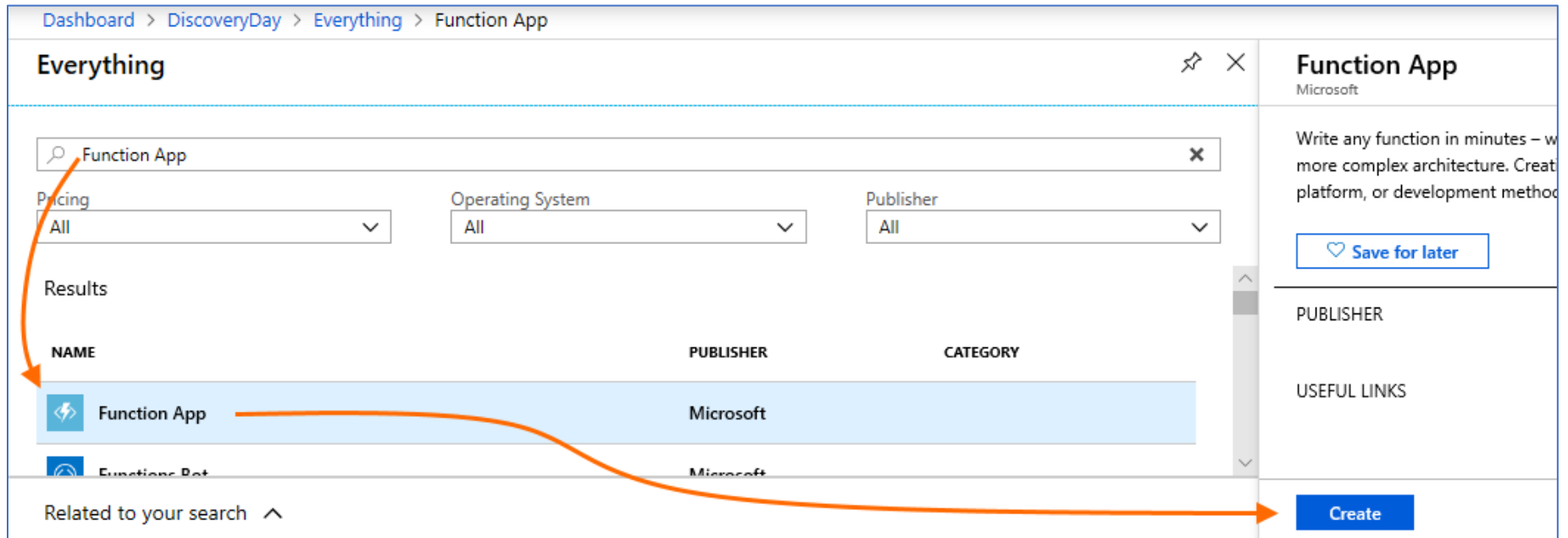
Included features

Every app hosted on this App Service plan will have access to these features.

Apply

Return to your Resource Group. When the App Service Plan has completed deployment (reminder – use the bell glyph at the top of the portal view to monitor deployment status and other events) and you can see the new App Service Plan in your Resource Group (you may need to click “Refresh”), click “+ Add” again.

Type “Function App” into the search box. Click “Function App” in the search results, then click “Create” on its info blade.



On the Function App’s create blade, ensure that all the following are correctly entered!

- App Name: enter a name for your Azure Function App
- Resource Group: ensure the Resource Group you have been using so far is selected
- OS: ensure Windows is selected
- Hosting Plan: select “App Service Plan”, then select the App Service Plan you just deployed
- Runtime Stack: ensure .NET is selected
- Storage: select “Use existing” and select the storage account you deployed in lab 1
- Application Insights: this is an optional Application Performance Monitoring solution that is helpful in issue analysis and debugging. You can leave this set to deploy, or you can click and disable deployment if desired (it does not add materially to deployment time).

When all information is correctly entered, click “Create”. Return to your Resource Group while deployment proceeds.

Dashboard > DiscoveryDay > Everything > Function App

Function App

Create

* App name
azdiscday2019-fn ✓
.azurewebsites.net

* Subscription
Azure-FTE-201609

* Resource Group ⓘ
☐ Create new ☒ Use existing
DiscoveryDay

* OS
Windows Linux (Preview)

* Hosting Plan ⓘ
App Service Plan

* App Service plan/Location
azdiscday-asp(East US)

* Runtime Stack
.NET

* Storage ⓘ
☐ Create new ☒ Use existing
azurediscday2019

Application Insights
azdiscday2019-fn

Create Automation options

When the Function App has completed deployment, find it in your Resource Group. Remember to monitor notifications and to click “Refresh” in the Resource Group view. Click on the Function App you just deployed and work through the following steps.

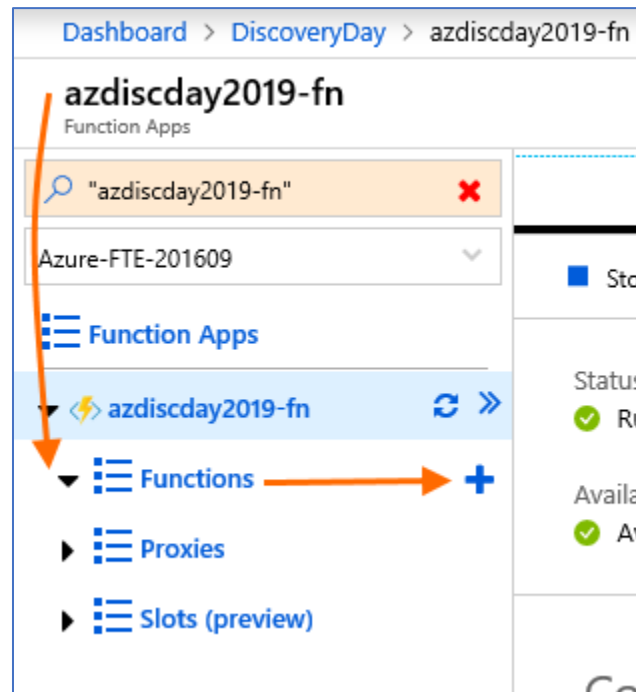
There are several ways to develop and deploy Azure Functions, including from Visual Studio and Visual Studio Code as well as the Azure portal. In this lab, we will use the Azure portal and deploy a C# script Function, but you are encouraged to learn about the other ways to develop and deploy Functions.

This task has several steps:

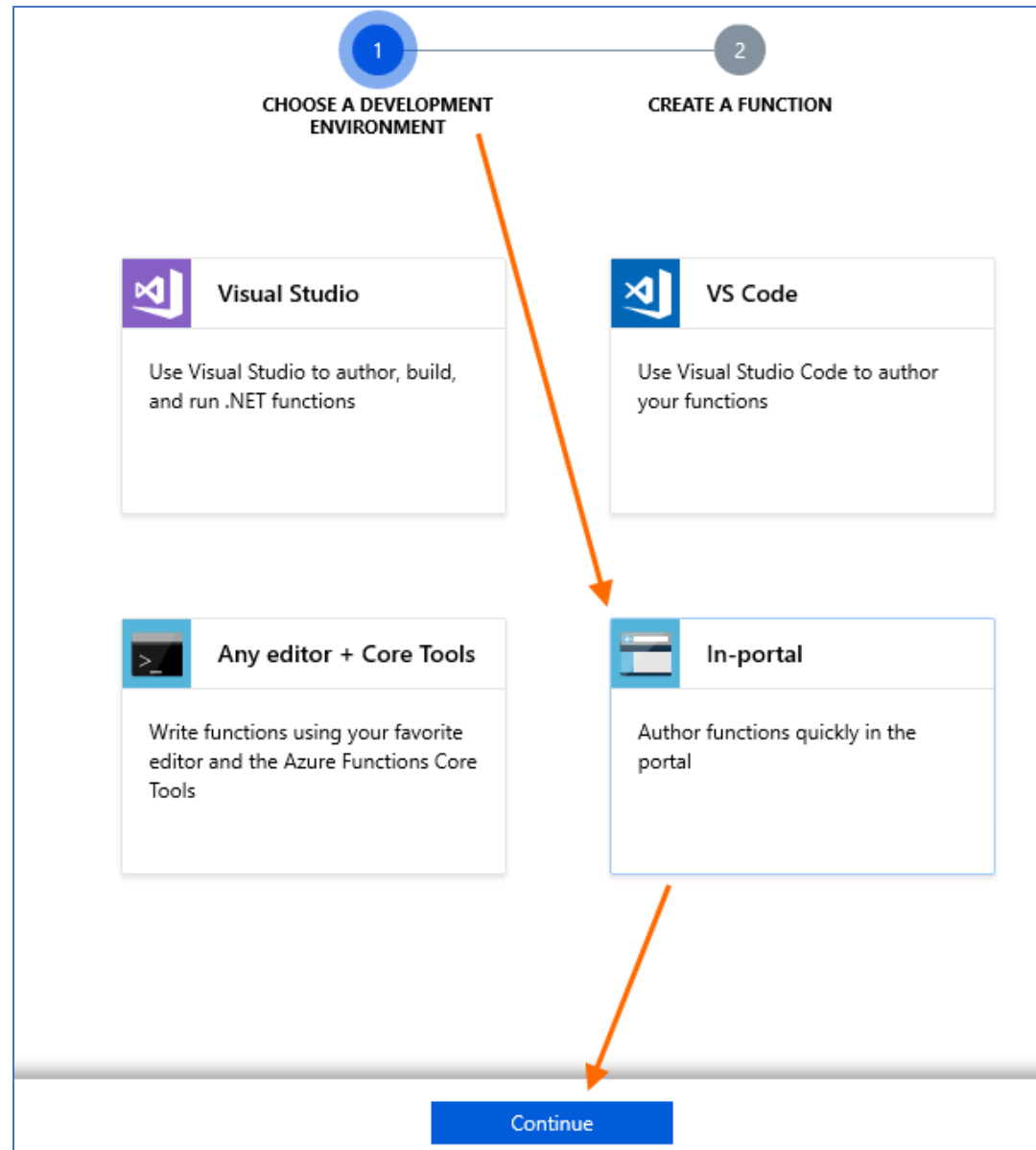
1. You will create a new Function and add an appropriate trigger binding that will cause the Function to run
2. You will add an output binding, which will let your Function pass the results of its work to the next step in the pipeline you are building in this lab
3. You will import packages from Nuget, to provide functionality you will need in the Function code
4. You will add Application Settings to store values from other resources you deployed earlier in this lab so that your Function code can access these values
5. You will conclude by adding Function code to process, enrich, and pass on the inbound messages. You will also test your Function before the taxi device simulator is live.

Create a Function and add a Trigger binding

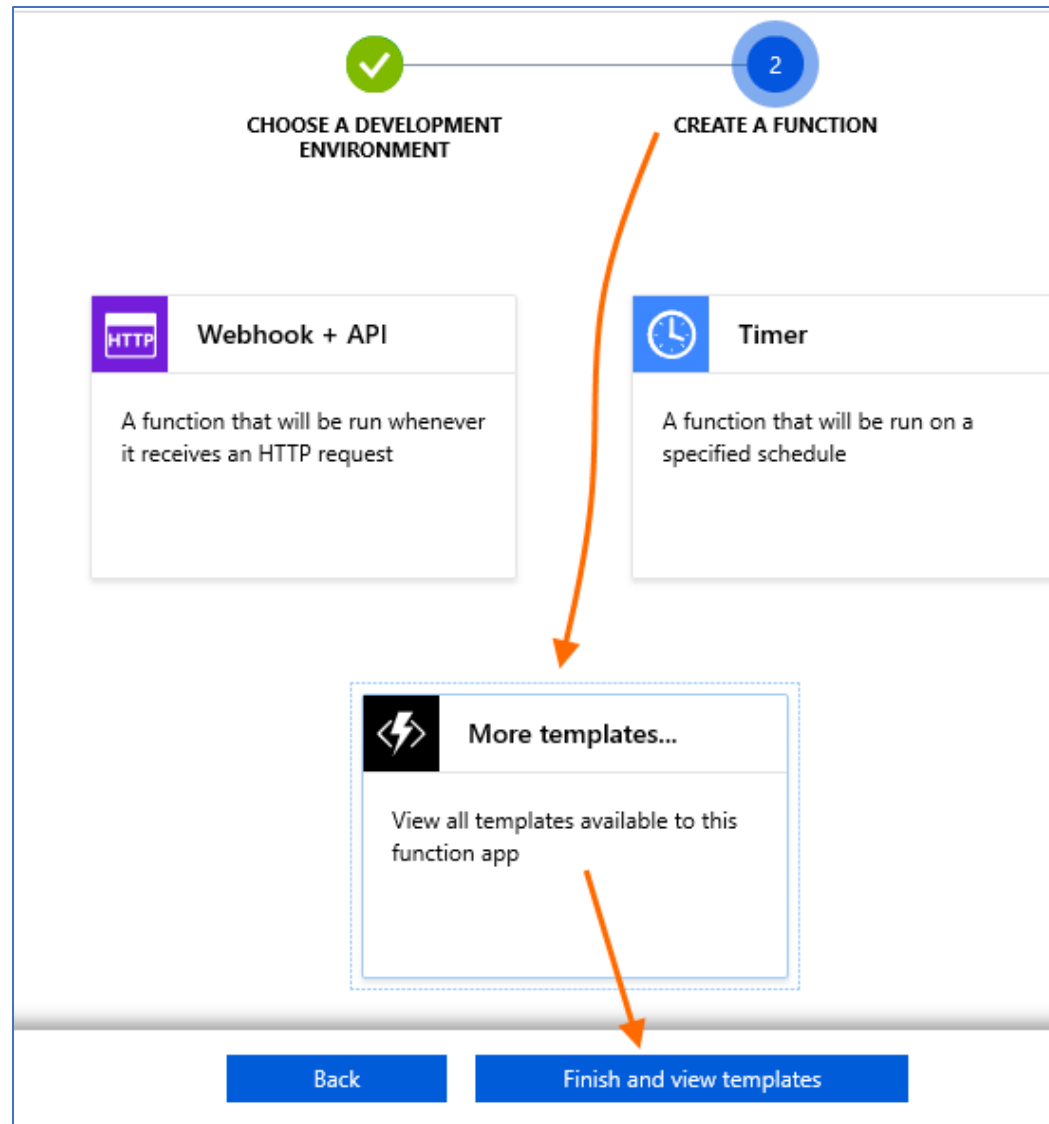
First, create a new Function. To do this, in the Function App left navigation bar, locate “Functions” and click “+” next to it.



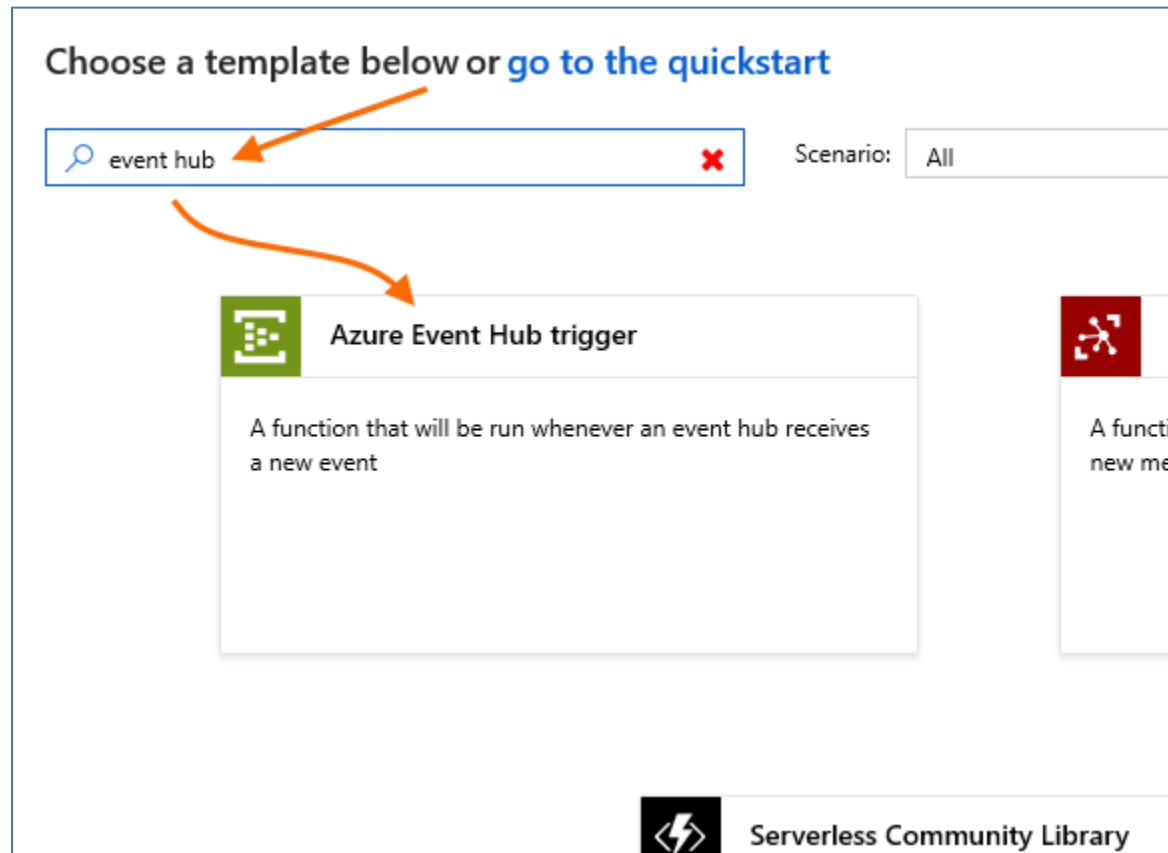
Next, you will be asked to pick your Development Environment. Click “In-portal”, then click “Continue”.



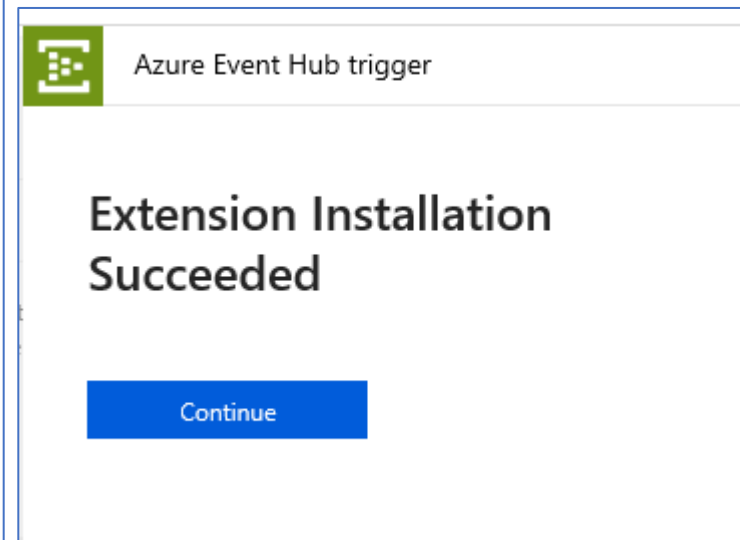
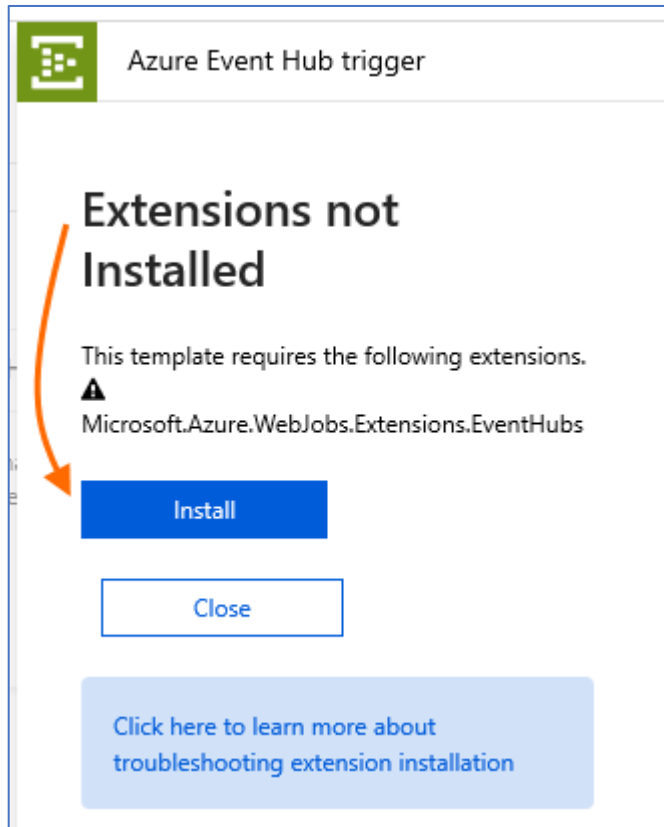
Next, click “More templates...” and “Finish and view templates”.



On the Function template view, find the “Azure Event Hub trigger” template. You can scroll to find it, or you can type “event hub” in the search box to narrow the selections. Click it to continue.



You may now be prompted to install missing extensions. If so, click “Install” and wait for installation to complete, then click “Continue”.



Next, on the “New Function” blade, provide a Function name. Fill in the first Event Hub name you created in task 1.

Leave “Event Hub consumer group” at its default value (“\$Default”). This is a more advanced option for when more than one group of resources “listens” to an Event Hub; it is not needed for this lab.

Then, configure a connection to the first Event Hub: that is where inbound messages from taxi devices will come, and you will now configure this Function to listen for those messages on that Event Hub. Next to “Event Hub connection”, click “new”.

Azure Event Hub trigger

New Function

Name:

StreamEnricher

Azure Event Hubs trigger

Event Hub connection ⓘ [new](#) [show value](#)

Event Hub consumer group ⓘ

\$Default

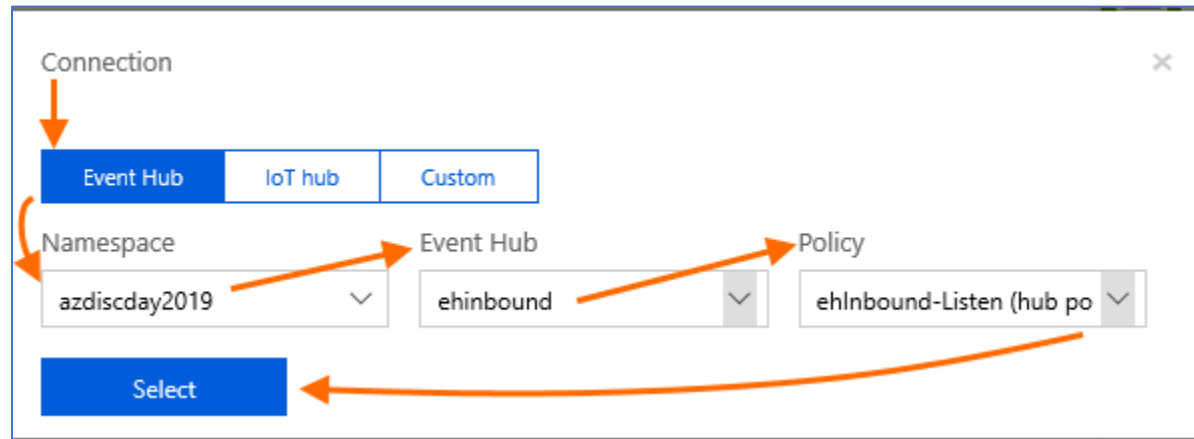
Event Hub name ⓘ

ehinbound

Create

Cancel

On the Event Hub connection view, select the Event Hub namespace, the first Event Hub and the Listen policy you created on the Event Hub. You created all of these in task 1. Then click “Select”.



The screenshot shows a 'Connection' dialog box with a close button (X) in the top right corner. At the top, there are three tabs: 'Event Hub' (selected), 'IoT hub', and 'Custom'. Below the tabs, there are three dropdown menus labeled 'Namespace', 'Event Hub', and 'Policy'. The 'Namespace' dropdown is set to 'azdisccday2019', the 'Event Hub' dropdown is set to 'ehinbound', and the 'Policy' dropdown is set to 'ehInbound-Listen (hub po'. Below these dropdowns is a blue 'Select' button. Orange arrows are drawn on the image to guide the user: one points to the 'Event Hub' tab, another points to the 'Namespace' dropdown, a third points to the 'Event Hub' dropdown, a fourth points to the 'Policy' dropdown, and a fifth points to the 'Select' button.

You will now be returned to the “New Function” view, which now shows the Event Hub connection that you just created. Click “Create”, which is now enabled, to create the Function.

You will now see the Function's code window with the default code for an Event Hub. Note the Logs tab at the bottom; when in this code view, you should always click this to open it, as it will be very useful for debugging and monitoring your Function's activity.

The screenshot displays the Azure Portal interface for a Function App named "azdisccday2019-fn - StreamEnricher". The left-hand navigation pane shows the hierarchy: "azdisccday2019-fn" > "Functions" > "StreamEnricher". The main area shows the C# code for the "run.csx" file. The code is as follows:

```
1 #r "Microsoft.Azure.EventHubs"
2
3 using System;
4 using System.Text;
5 using Microsoft.Azure.EventHubs;
6
7 public static async Task Run(EventData[] events, ILogger log)
8 {
9     var exceptions = new List<Exception>();
10
11     foreach (EventData eventData in events)
12     {
13         try
14         {
15             string messageBody = Encoding.UTF8.GetString(eventData.Body.ToArray());
16
17             // Replace these two lines with your processing logic
18             log.LogInformation($"C# Event Hub trigger function processed a message.");
19             await Task.Yield();
20         }
21         catch (Exception e)
22         {
23             // We need to keep processing the rest of the batch, so we add the exception to the list.
24             // Also, consider capturing details of the message to help with debugging.
25             exceptions.Add(e);
26         }
27     }
28
29     // Once processing of the batch is complete, if any messages failed, throw an exception.
30
31     if (exceptions.Count > 1)
32     {
33         throw new AggregateException(exceptions);
34     }
35     if (exceptions.Count == 1)
36     {
37         throw exceptions.Single();
38     }
39 }
```

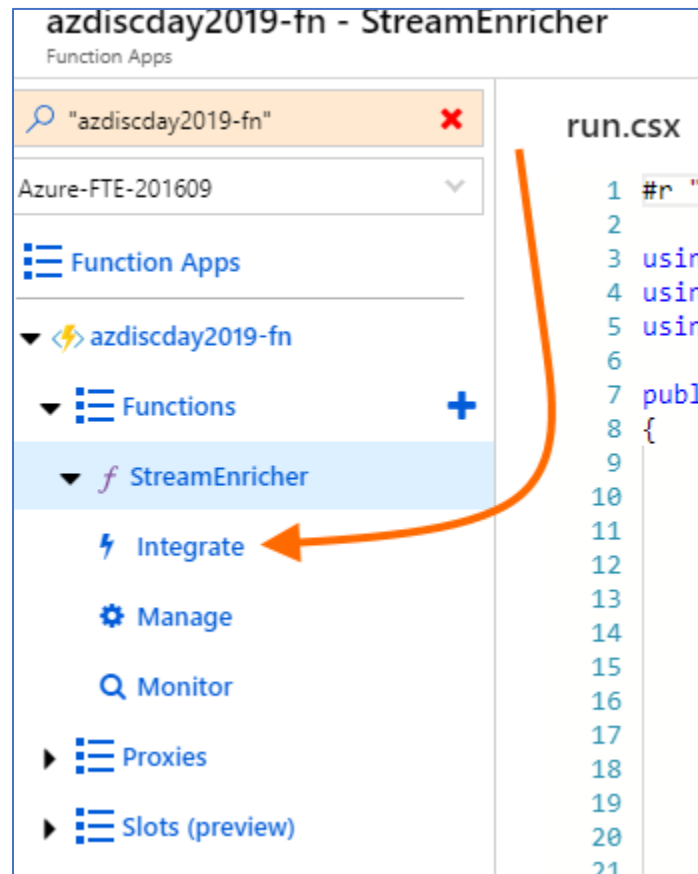
At the bottom of the interface, the "Logs" tab is selected, showing a list of log entries:

```
2019-02-18T20:49:57 Welcome, you are now connected to log-streaming service.
2019-02-18T20:50:57 No new trace in the past 1 min(s).
2019-02-18T20:51:57 No new trace in the past 2 min(s).
2019-02-18T20:52:57 No new trace in the past 3 min(s).
2019-02-18T20:53:57 No new trace in the past 4 min(s).
```

An orange arrow points from the "StreamEnricher" function in the left pane to the "Logs" tab at the bottom.

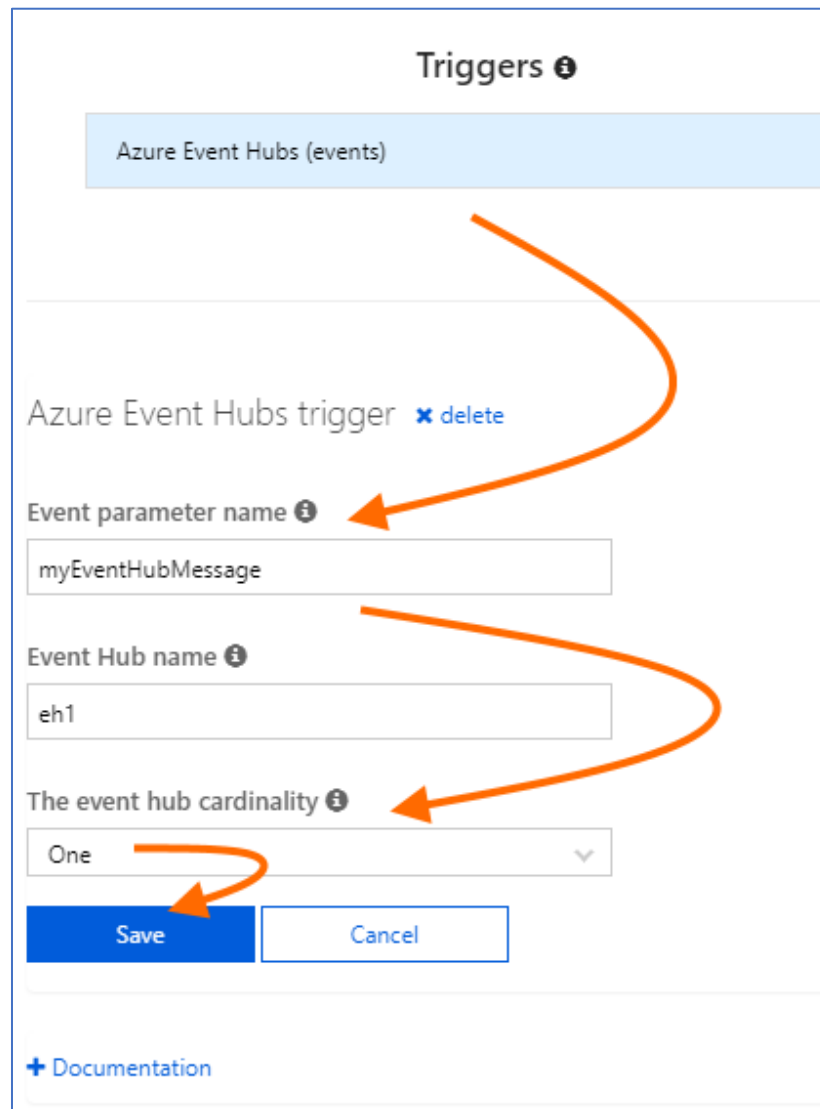
First, you need to make some changes to the default Function code.

In the left menu, click “Integrations” under the function tree.



Update the “Azure Event Hubs trigger” as follows, then click “Save”.

- Change “Event parameter name” to “myEventHubMessage”
- Change “The event hub cardinality” to “One”.

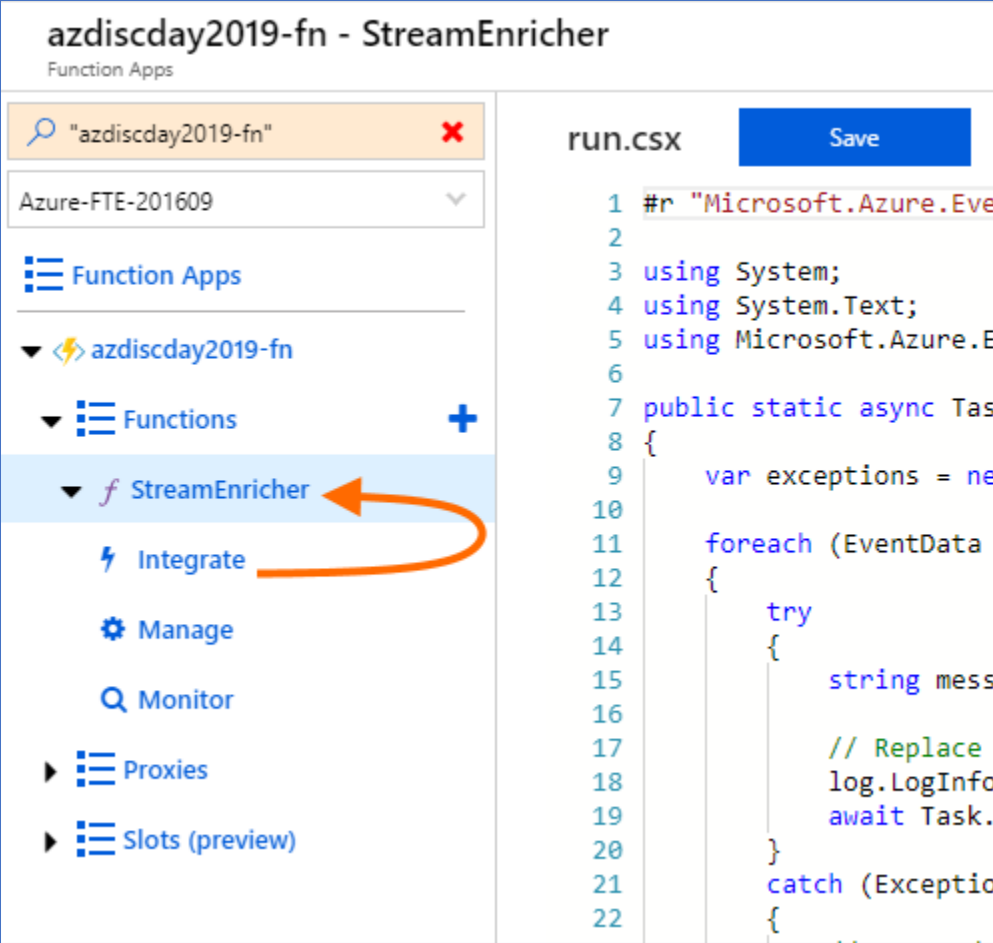


The screenshot shows the 'Triggers' configuration page for an Azure Event Hubs trigger. The title 'Triggers' is at the top with an information icon. Below it is a light blue header bar labeled 'Azure Event Hubs (events)'. The main configuration area contains the following elements:

- The trigger name 'Azure Event Hubs trigger' followed by a blue 'x delete' link.
- The 'Event parameter name' field, which contains 'myEventHubMessage'. An orange arrow points from the text 'myEventHubMessage' in the instructions to this field.
- The 'Event Hub name' field, which contains 'eh1'. An orange arrow points from the text 'eh1' in the instructions to this field.
- The 'The event hub cardinality' dropdown menu, which is currently set to 'One'. An orange arrow points from the text 'One' in the instructions to this dropdown.
- At the bottom are two buttons: a blue 'Save' button and a white 'Cancel' button with a blue border. An orange arrow points from the text 'Save' in the instructions to the 'Save' button.
- At the very bottom is a link that says '+ Documentation'.

Three orange curved arrows originate from the instruction list and point to the 'Event parameter name' field, the 'Event Hub name' field, and the 'The event hub cardinality' dropdown menu. A fourth orange arrow points from the 'Save' button in the instructions to the 'Save' button on the form.

Next, click on the function node to return to the code view.



The screenshot shows the Azure Functions portal interface for a function app named "azdisccday2019-fn - StreamEnricher". The left sidebar contains a navigation menu with the following items: "Function Apps", "azdisccday2019-fn", "Functions", "StreamEnricher" (highlighted with an orange arrow), "Integrate", "Manage", "Monitor", "Proxies", and "Slots (preview)". The main area displays the "run.csx" code file, which contains the following C# code:

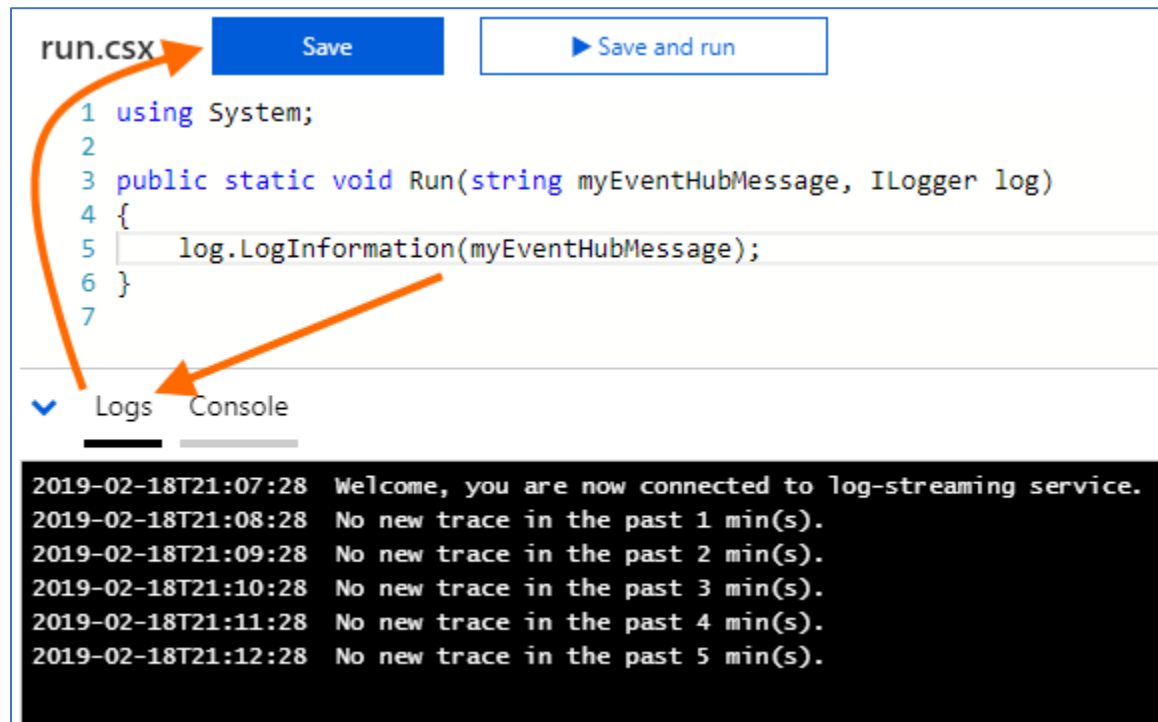
```
1 #r "Microsoft.Azure.EventHubs"
2
3 using System;
4 using System.Text;
5 using Microsoft.Azure.EventHubs;
6
7 public static async Task
8 {
9     var exceptions = new List<Exception>();
10
11     foreach (EventDataBatch batch in await client.ReceiveNextAsync())
12     {
13         try
14         {
15             string message = batch.GetMessage(0).Body.ToString();
16
17             // Replace
18             log.LogInformation(message);
19             await Task.Delay(1000);
20         }
21         catch (Exception ex)
22         {
23             exceptions.Add(ex);
24         }
25     }
26 }
```

Replace all of the code that was provided with the following code.

```
using System;

public static void Run(string myEventHubMessage, ILogger log)
{
    log.LogInformation(myEventHubMessage);
}
```

Make sure your Logs tab is expanded. Then click “Save” above the code. You should see a message in the Logs tab that “Compilation succeeded.”

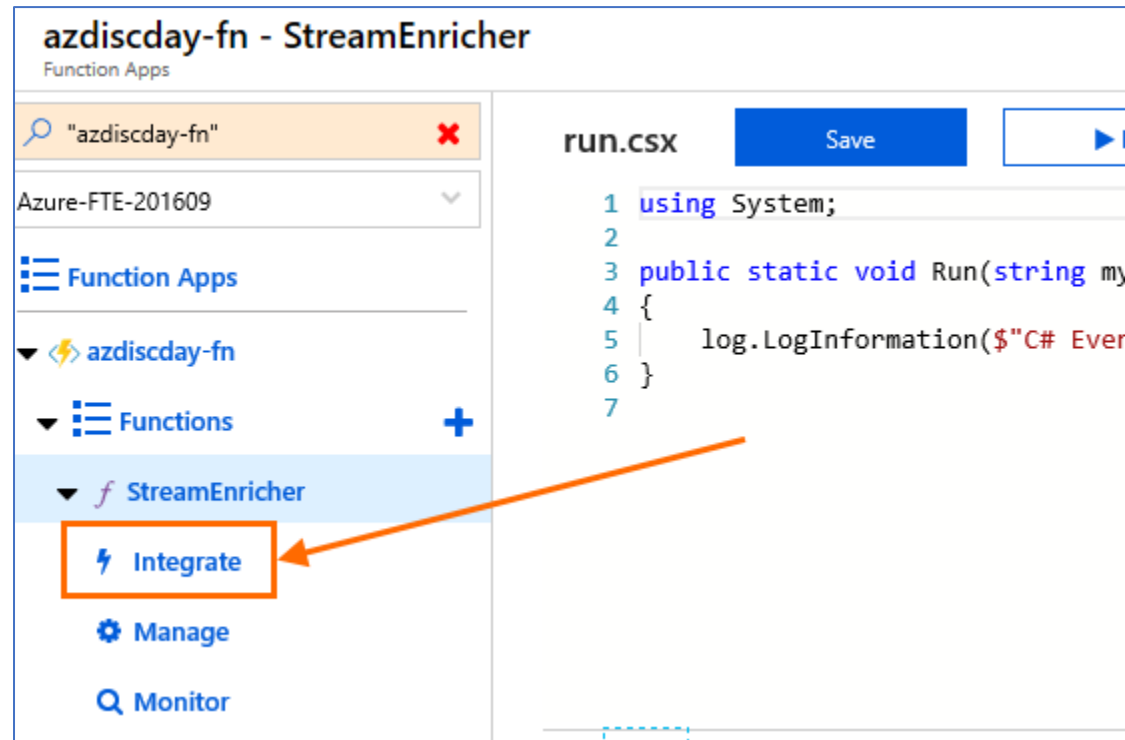


```
2019-02-18T21:14:28 No new trace in the past 7 min(s).
2019-02-18T21:14:34.347 [Information] Script for function 'StreamEnricher' changed. Reloading.
2019-02-18T21:14:36.050 [Information] Compilation succeeded.
```

Add an Output Binding

Now, you need to add an output binding. This is where your Function will write enriched stream messages after processing them.

In the left menu, find your Function and, below it, click “Integrate”.



On the Integrate view, under “Outputs” click “+ New Output”. Then click “Azure Event Hubs” from the choices. Then click “Select”.

The screenshot displays the Azure Functions portal interface for a function named "azdisccday-fn - StreamEnricher". The left-hand navigation pane shows the hierarchy: "Function Apps" > "azdisccday-fn" > "Functions" > "StreamEnricher". The "Integrate" tab is selected, showing the function's configuration. The "Outputs" section is active, displaying a list of available output providers. The "Azure Event Hubs" provider is highlighted with a blue border, indicating it is the selected option. Orange arrows illustrate the workflow: one arrow points from the "+ New Output" button in the "Outputs" section to the "Azure Event Hubs" provider, and another arrow points from the "Azure Event Hubs" provider to the "Select" button at the bottom. The "Triggers" and "Inputs" sections are also visible, each with a "+ New" button. The "Azure Event Hubs" provider is listed under the "Outputs" section, along with other providers like Azure Queue Storage, Azure Blob Storage, HTTP, Azure Service Bus, Azure Table Storage, SendGrid, and Twilio SMS.

On the output view, provide the second Event Hub name you created in task 1. This is the Event Hub to which the Function sends the enriched message after finishing processing. Then, under “Event Hub connection”, click “new” (ignore the pre-filled connection – that is for the inbound Event Hub which we used previously for the Function trigger).

Triggers **Inputs** **Outputs** [Advanced editor](#)

Azure Event Hubs (myEventHub...)

+ New Input

+ New Output

Azure Event Hubs output

Event parameter name ⓘ

outputEventHubMessage

☐ Use function return value

Event Hub connection ⓘ [show value](#)

azdisccday2019_ehInbound-Listen_EVENTHUB [new](#)

Event Hub name ⓘ

ehenriched

Save Cancel

For this Event Hub Connection, provide the same Event Hubs Namespace as previously. This time, specify the second Event Hub you created in task 1, and the Send policy you created on that Event Hub. Then click “Select”.

Connection

Event Hub

IoT hub

Custom

Namespace

azdisccday2019

Event Hub

ehenriched

Policy

enEnriched-Send (hub poli

Select

You are now back at the output creation view, which now shows the Event Hub connection you just created. After verifying that it looks correct, click “Save”.

Triggers ⓘ

Azure Event Hubs (myEventHub...

Inputs ⓘ

+ New Input

Outputs ⓘ

+ New Output

Azure Event Hubs output

Event parameter name ⓘ

outputEventHubMessage

☐ Use function return value

Event Hub connection ⓘ

azdisccday2019_enEnriched-Send_EVENTHUB

show value

new

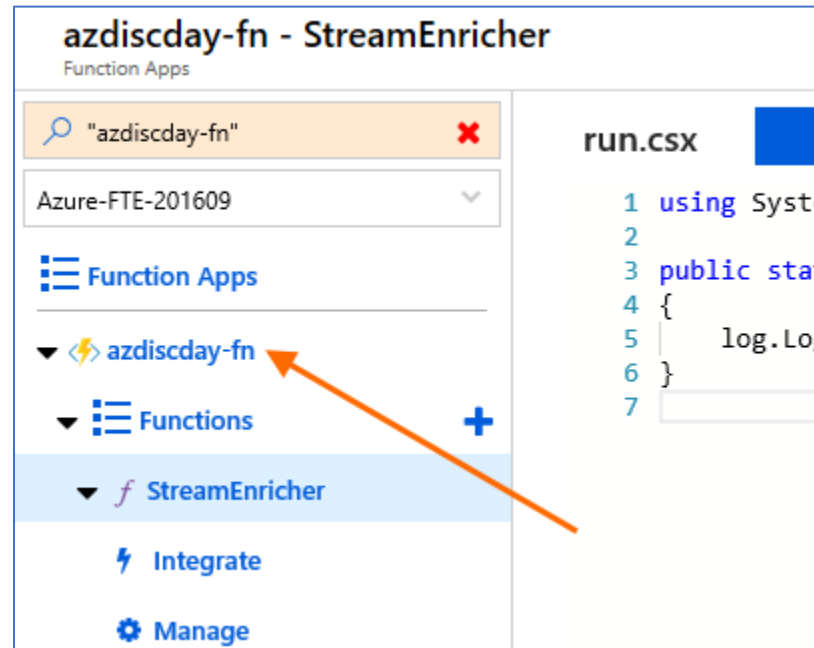
Event Hub name ⓘ

ehenriched

Save

Cancel

Next, in the left navigation bar, click on the Function App node (lightning-bolt icon with the name you used to create the Function App in the Azure portal). This will bring you back to the Overview. You can now move on to the next step.



To complete this task, you must make your Function code aware of this new output binding. Return to the code view and expand the “Logs” tab.

Add a parameter to the Run() method signature. The name of the parameter must be the “Event parameter name” you specified on the new output binding view – in this case, the default “outputEventHubMessage” is used (see previous screenshots).

Add the parameter as follows:

```
out string outputEventHubMessage.
```

Next, in the body of the function, add an initial assignment for the output parameter’s value (this is required for output parameters in C# functions). You will add more useful code later in this task.

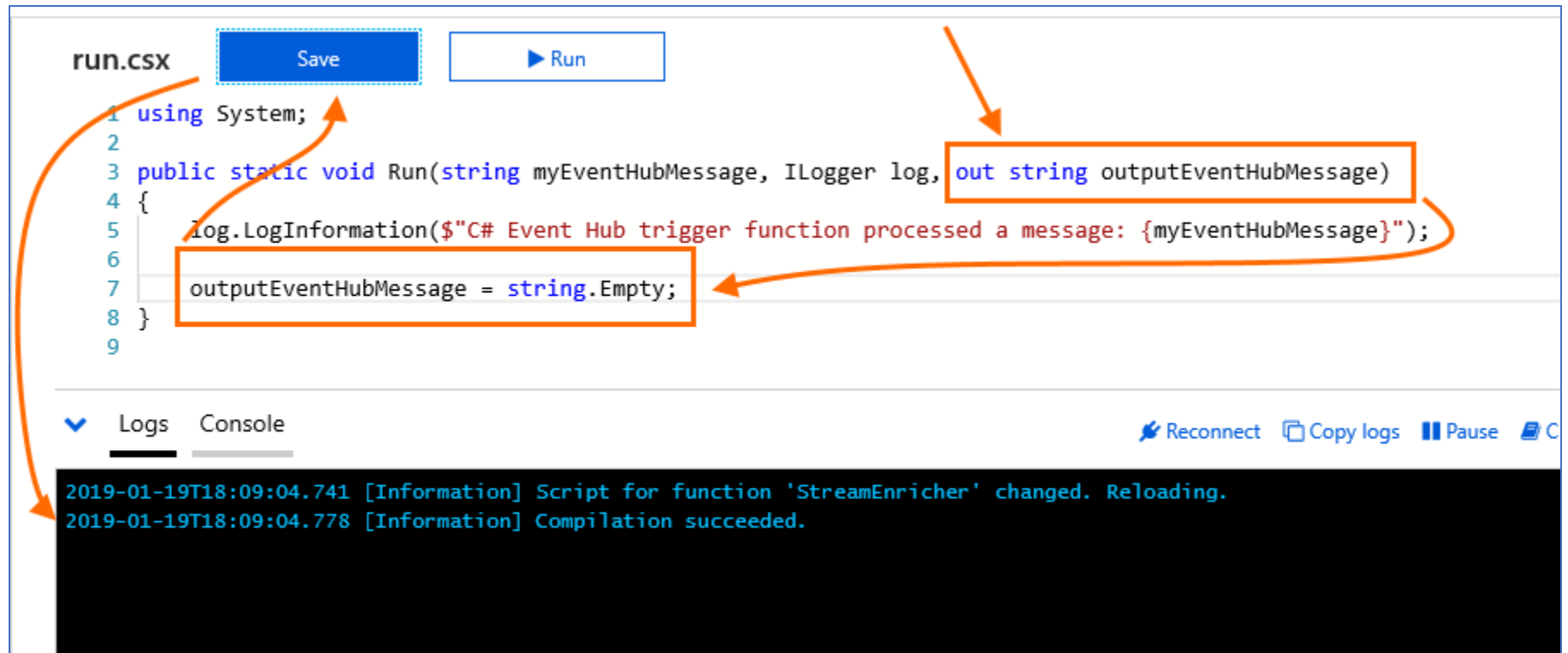
```
outputEventHubMessage = string.Empty;
```

Then click “Save” at the top of the code window. If the code is correct, you will see a “Compilation succeeded” message in the Logs tab.

Before:

```
1 using System;
2
3 public static void Run(string myEventHubMessage, ILogger log)
4 {
5     log.LogInformation($"C# Event Hub trigger function processed a message: {myEventHubMessage}");
6 }
7
```

After adding the out parameter and the new line of code and clicking “Save”:



```
run.csx
1 using System;
2
3 public static void Run(string myEventHubMessage, ILogger log, out string outputEventHubMessage)
4 {
5     log.LogInformation($"C# Event Hub trigger function processed a message: {myEventHubMessage}");
6
7     outputEventHubMessage = string.Empty;
8 }
9
```

Save Run

Logs Console

2019-01-19T18:09:04.741 [Information] Script for function 'StreamEnricher' changed. Reloading.
2019-01-19T18:09:04.778 [Information] Compilation succeeded.

You have now successfully added an output binding and made your Function code aware of it. You can now proceed to the next step.

Import Packages from Nuget

The Function's code will need to use two external libraries for its functionality. Your Function will need to download these libraries from nuget.org.

Your Function will use these Packages:

1. Newtonsoft.Json. This is a widely-used JSON processing library.
2. pelazem.azure.cognitive.textanalytics. This is a library that wraps several calls to the Azure Text Analytics Cognitive Service into one result set. The Text Analytics Cognitive Service requires a separate API call for each type of text analytics; this library abstracts away this and other complexities and returns one combined text analytics result, with which your code will enrich the inbound message.
Note that this library is a convenience. You could also interact directly with the Text Analytics Cognitive Service via HTTP REST calls; like all the Azure Cognitive Services, the service we will use exposes a REST API.

In this lab, you are authoring a Function in the portal. Portal-authored Functions use a specific file to initiate nuget package download and install. This file is called function.proj. This file is provided with this lab document.

Please download a copy to your machine from this URL:

<https://raw.githubusercontent.com/plzm/azure-discoveryday2019-mdw/master/labs/lab3/StreamEnricherFunction/function.proj>

Note: you can also go to the github repository for this workshop at <https://github.com/plzm/azure-discoveryday2019-mdw> and navigate all labs and documents there.

In your Function's code view, first expand the "Logs" tab at the bottom. Then expand the "View files" tab at the right.



You will now see both the Logs console at the bottom, and the “View files” list of Function files at the right. Click “Upload” above the list of files.

The screenshot displays the Azure Functions portal interface. At the top, the file `run.csx` is open in the editor, showing the following C# code:

```
1 using System;
2
3 public static void Run(string myEventHubMessage, ILogger log, out string outputEventHubMessage)
4 {
5     log.LogInformation($"C# Event Hub trigger function processed a message: {myEventHubMessage}");
6
7     outputEventHubMessage = string.Empty;
8 }
9
```

Below the code editor is the Logs console, which is currently showing the following log entries:

```
2019-01-20T03:01:34 Welcome, you are now connected to log-streaming service.
2019-01-20T03:02:34 No new trace in the past 1 min(s).
2019-01-20T03:03:34 No new trace in the past 2 min(s).
```

On the right side of the interface, the "View files" tab is active, showing a list of files: `StreamEnricher`, `function.json`, and `run.csx`. Above this list, there are buttons for `+ Add`, `↑ Upload`, and `🗑 Delete`. An orange arrow points from the `Run` button in the code editor to the `Upload` button in the file list.

Find the function.proj file you just downloaded and select it in the upload dialog. After it uploads, you will see several messages in the Console tab, concluding with installation of the two packages and successful compilation of your Function. (Note that versions may change from what the screenshots show.)

The screenshot displays the Azure Functions portal interface. At the top, there are buttons for 'run.csx', 'Save', and 'Run'. Below these are tabs for 'Logs' and 'Console'. The 'Console' tab is active, showing a series of log messages. On the right side, there is a 'View files' section with a list of files: 'StreamEnricher', 'function.json', 'function.proj', and 'run.csx'. The 'function.proj' file is highlighted with an orange box. Two orange arrows point from the 'function.proj' file to the console output. The first arrow points to the line 'Installing pelazem.azure.cognitive.textanalytics 1.3.0.' and the second arrow points to the line 'Compilation succeeded.'

```
2019-01-20T03:08:04.543 [Information] 3. Copy the .NET Core SDK to a non-protected location and use it from there.
2019-01-20T03:08:04.543 [Information]
2019-01-20T03:08:04.955 [Information] ASP.NET Core
2019-01-20T03:08:04.956 [Information] -----
2019-01-20T03:08:04.956 [Information] Successfully installed the ASP.NET Core HTTPS Development Certificate.
2019-01-20T03:08:04.956 [Information] To trust the certificate run 'dotnet dev-certs https --trust' (Windows and macOS only). For establishing trust on other platforms refer to the platform specific documentation.
2019-01-20T03:08:04.956 [Information] For more information on configuring HTTPS see https://go.microsoft.com/fwlink/?linkid=848054.
2019-01-20T03:08:07.570 [Information] Restoring packages for D:\local\Temp\cf52cee4-f85c-4418-b456-a3ea6d52f303\function.proj...
2019-01-20T03:08:08.516 [Information] Installing pelazem.azure.cognitive.textanalytics 1.3.0.
2019-01-20T03:08:08.561 [Information] Installing Newtonsoft.Json 12.0.1.
2019-01-20T03:08:11.288 [Information] Generating MSBuild file D:\local\Temp\cf52cee4-f85c-4418-b456-a3ea6d52f303\obj\function.proj.nuget.g.props.
2019-01-20T03:08:11.288 [Information] Generating MSBuild file D:\local\Temp\cf52cee4-f85c-4418-b456-a3ea6d52f303\obj\function.proj.nuget.g.targets.
2019-01-20T03:08:11.288 [Information] Restore completed in 3.72 sec for D:\local\Temp\cf52cee4-f85c-4418-b456-a3ea6d52f303\function.proj.
2019-01-20T03:08:11.502 [Information] Packages restored.
2019-01-20T03:08:11.810 [Information] Script for function 'StreamEnricher' changed. Reloading.
2019-01-20T03:08:11.905 [Information] Compilation succeeded.
```

When your Function has installed these two packages and compiled successfully, this step is complete. Please move on to the next step.

Add Application Settings

In the left navigation, click the node with the lightning bolt and your Function App's name to go to the Overview. In the Function App Overview, click "Application Settings".

The screenshot shows the Azure portal interface for a Function App named "azdisccday2019-fn". The breadcrumb navigation at the top reads "Dashboard > DiscoveryDay > azdisccday2019-fn". The left-hand navigation pane lists "Function Apps" with a search bar containing "azdisccday2019-fn". Under "Function Apps", the "azdisccday2019-fn" item is selected and highlighted in blue, featuring a lightning bolt icon and a refresh/expand icon. Below it are "Functions", "Proxies", and "Slots (preview)". The main content area is titled "Overview" and "Platform features". It displays the app's status as "Running" and "Available", along with its subscription, subscription ID, URL, and app service plan. At the bottom, the "Configured features" section lists "Function app settings", "Application settings" (highlighted with an orange arrow), and "Application Insights".

Dashboard > DiscoveryDay > azdisccday2019-fn

azdisccday2019-fn
Function Apps

Search: "azdisccday2019-fn" ✕

Azure-FTE-201609

Function Apps

▼ ⚡ azdisccday2019-fn ↻ >>

▼ Functions +

▶ Proxies

▶ Slots (preview)

Overview Platform features

Stop Swap Restart Get publish profile Reset

Status: Running

Availability: Available

Subscription: Azure-FTE-201609

Subscription ID: e61e4c75-268b-4c94-ad48-237aa3231481

URL: <https://azdisccday2019-fn.azurewebsites.net>

App Service plan / pricing tier: azdisccday-asp (PremiumV2)

Configured features

- ⚡ Function app settings
- ≡ Application settings
- 💡 Application Insights

You have creat

Now it is time

Scroll down to the “Application settings” section. Add two new application settings. For each, click “+ Add new setting”, then add the setting as follows. Note that as soon as you enter the value, the portal view will hide the value and show “Hidden value. Click to edit.” This is to protect sensitive data.

Name	Value
TextAnalyticsApiEndpoint	Enter the API Endpoint value you retrieved after deploying the Text Analytics Cognitive Service in task 2.
TextAnalyticsApiKey	Enter the API Key value you retrieved after deploying the Text Analytics Cognitive Service in task 2.

Dashboard > DiscoveryDay > azdisccday2019-fn

azdisccday2019-fn

Function Apps

azdisccday2019-fn

Functions

Proxies

Slots (preview)

Application settings

Application Settings are encrypted at rest and transmitted over an encrypted

Hide Values

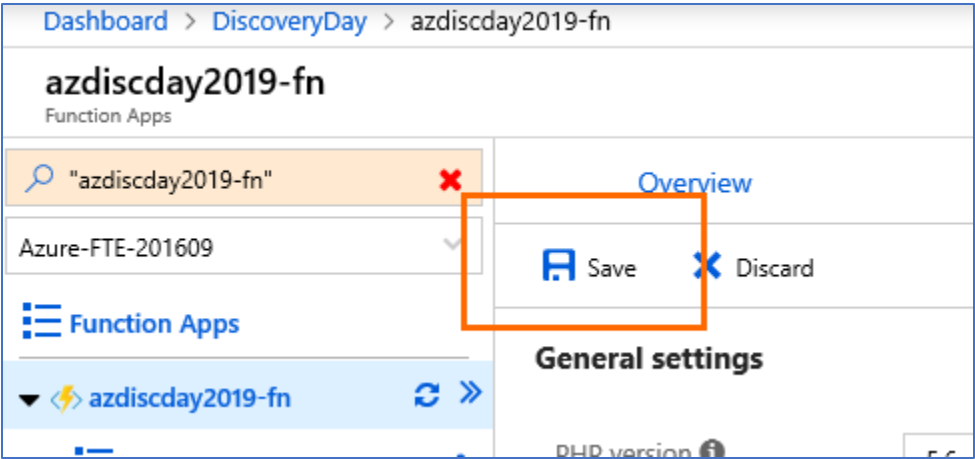
Show Values

APP SETTING NAME	VALUE
APPINSIGHTS_INSTRUMENTATIONKEY	Hidden value. Click to edit.
AzureWebJobsStorage	Hidden value. Click to edit.
FUNCTIONS_EXTENSION_VERSION	Hidden value. Click to edit.
FUNCTIONS_WORKER_RUNTIME	Hidden value. Click to edit.
WEBSITE_NODE_DEFAULT_VERSION	Hidden value. Click to edit.
TextAnalyticsApiEndpoint	Hidden value. Click to edit.
TextAnalyticsApiKey	Hidden value. Click to edit.

+ Add new setting

Page 48 of 62

Next, scroll back up and click “Save”.



After the Application Settings are saved, move to the next step.

Add and Test Function Code

The last step in this task is to add our Function's code, then to test it with a sample message.

Your Function code needs to accomplish the following tasks:

1. Receive a new taxi device message from the inbound Event Hub
2. Deserialize the message to structured JSON
3. Extract the customer's comments text from the taxi message JSON
4. Instantiate the text analytics library, pass in the customer's comments text for analysis, and receive text analytics results back
5. Add selected parts of the text analytics results to the received (inbound) taxi device message – this is where message enrichment occurs
6. Write the enriched message to the output, which points to the second Event Hub (which you will work with further in lab 4)

A complete, working version of the Function code is provided for you in this workshop's github repo. The full URL to the code file is:

<https://raw.githubusercontent.com/plzm/azure-discoveryday2019-mdw/master/labs/lab3/StreamEnricherFunction/run.csx>

Please download a copy of this file, then upload it into your Azure Function using the same process as you did with function.proj in a previous step. You should overwrite the existing run.csx file.

After the file is uploaded, confirm in the "Logs" tab (you did expand it, didn't you?) that the script for your Function changed and that compilation succeeded.

Let's examine a few key pieces of this code. You are encouraged to experiment and change the code, clicking "Save" after changes – look for compilation messages in the "Logs" tab when you do this.

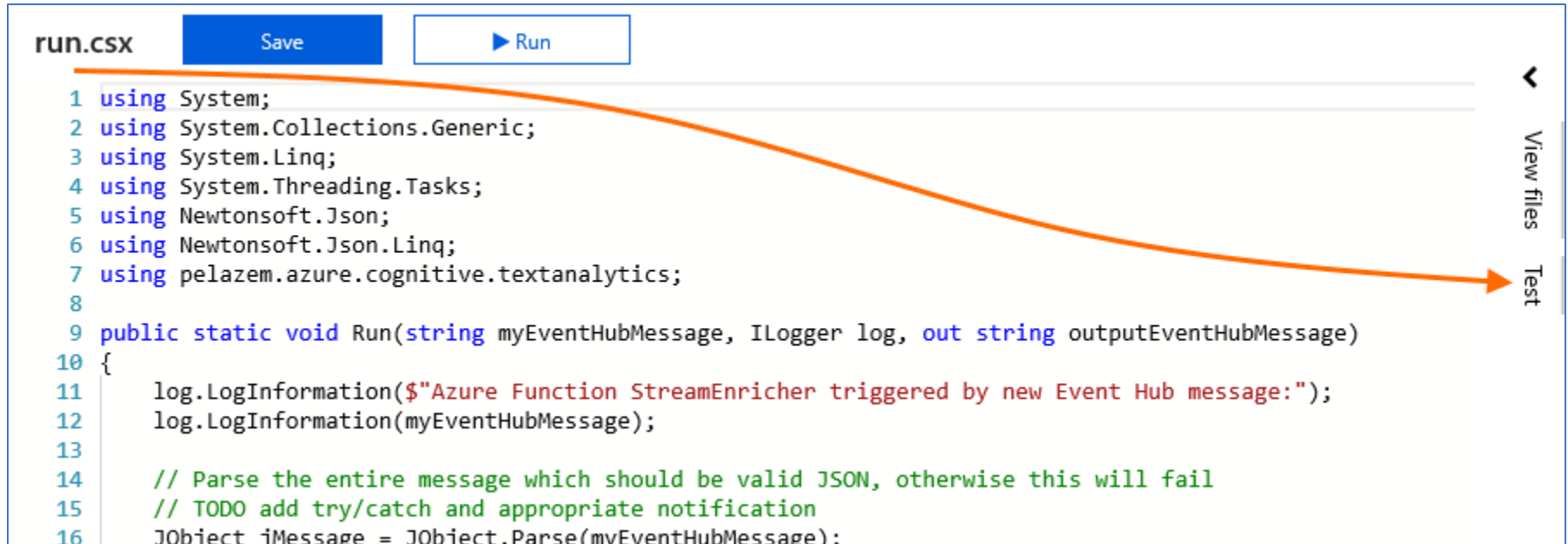
Note that the code contains many console output statements (lines beginning with `log.LogInformation`). Many of these are commented out with a leading `//`. These lines – when not commented out – will write to the Logs tab, providing very helpful real-time information about what your Function is doing.

Consider uncommenting (or commenting) some of these lines as you examine, test, and change the code.

Can you match what the code is doing to the list of six tasks above?

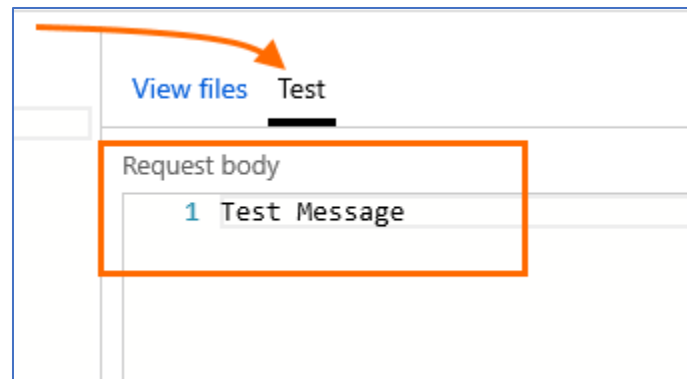
Note the lines that use `Environment.GetEnvironmentVariable("...")`. These lines go to the Application Settings and retrieve the values you pasted there in an earlier step. It's a good idea to uncomment the immediately following `log.LogInformation()` calls that echo those values to the "Logs" tab, to ensure you pasted the right keys and values into Application Settings.

Now, let's test the Function code. Still in your Function code window, expand the "Test" tab on the right side.



```
run.csx Save Run
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5 using Newtonsoft.Json;
6 using Newtonsoft.Json.Linq;
7 using pelazem.azure.cognitive.textanalytics;
8
9 public static void Run(string myEventHubMessage, ILogger log, out string outputEventHubMessage)
10 {
11     log.LogInformation($"Azure Function StreamEnricher triggered by new Event Hub message:");
12     log.LogInformation(myEventHubMessage);
13
14     // Parse the entire message which should be valid JSON, otherwise this will fail
15     // TODO add try/catch and appropriate notification
16     JObject jMessage = JObject.Parse(myEventHubMessage);
```

You will see a "Request body" text box, pre-populated with sample text "Test Message".

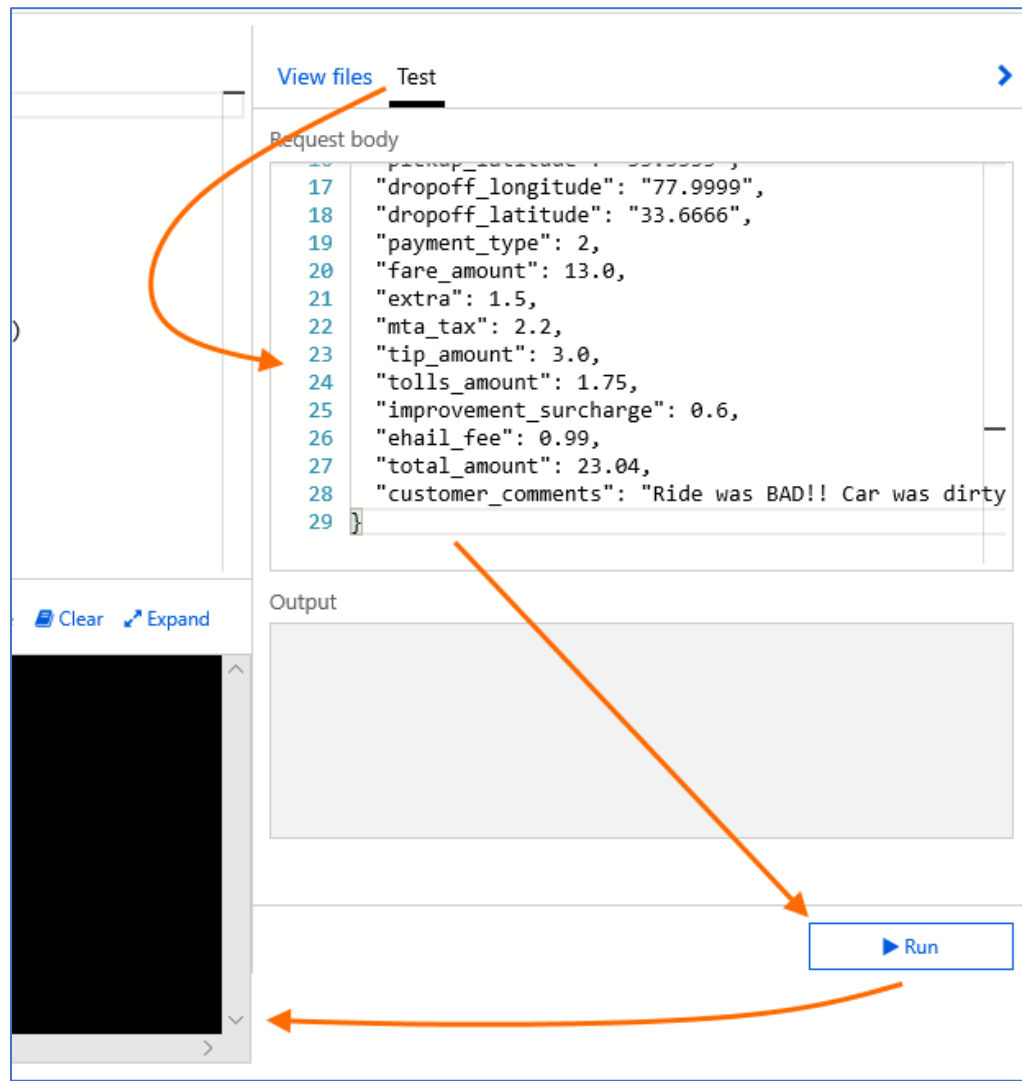


Select and delete that test message.

A sample message is provided for you in the github repo for this workshop. Please navigate to this URL and copy/paste the sample taxi device message into the Test "Request body" text box where you just deleted "Test Message".

https://raw.githubusercontent.com/plzm/azure-discoveryday2019-mdw/master/labs/lab3/StreamEnricherFunction/test_message.txt

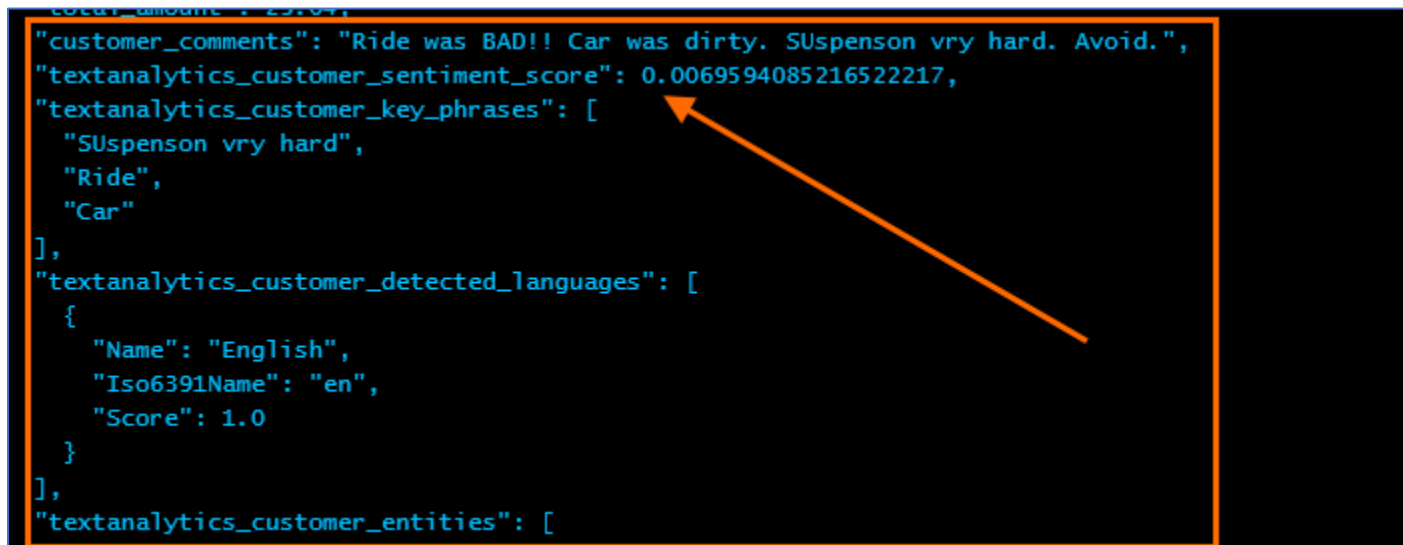
After you paste the test message into the “Request body” text box, find and click “Run” at bottom right. This will run your Function with a simulated Event Hub input message using the JSON sample message you just pasted. The “Logs” tab (you did expand it, right?) will show you output from running the Function with this message.



The “Logs” tab will show you output from running the customer comments text through the Text Analytics Azure Cognitive Service!

Note the `textanalytics_customer_sentiment_score` output. You will use this in lab 4.

This value is in a range from zero (negative) to one (positive). For the sample text, note how a clearly negative comment resulted in a numerical sentiment score very close to zero.



```
total_amount": 23.84,  
"customer_comments": "Ride was BAD!! Car was dirty. SUsension vry hard. Avoid.",  
"textanalytics_customer_sentiment_score": 0.0069594085216522217,  
"textanalytics_customer_key_phrases": [  
  "SUsension vry hard",  
  "Ride",  
  "Car"  
],  
"textanalytics_customer_detected_languages": [  
  {  
    "Name": "English",  
    "Iso6391Name": "en",  
    "Score": 1.0  
  }  
],  
"textanalytics_customer_entities": [
```

This task is now complete. You can now exit the Azure Function view and return to your Resource Group. Please move on to the final task of lab 3.

Task 4 – Deploy Taxi Device Simulator and Initiate Message Stream

We have provided a taxi device simulator for this lab. Both a downloadable runner as well as the source code are provided.

To use the simulator, you will need the connection string for the inbound Event Hub's Send policy (since the simulator will -send- messages to the inbound Event Hub". This connection string will have a format like this (sensitive details have been removed):

```
"Endpoint=sb://azdisday2019.servicebus.windows.net/;SharedAccessKeyName=ehinbound_Send;SharedAccessKey={removed};EntityPath=ehinbound"
```

Two methods are provided for getting the device simulator running. **DO NOT USE BOTH – PICK ONLY ONE!**

Method 1 is much simpler than method 2 and we recommend you use this method unless any difficulties are encountered, in which case try method 2 instead.

Method 1 – Deploy device simulator in a container

The device simulator is available as a Docker container image which can be deployed to an Azure Container Instance using a single line command. You need to provide several pieces of information in this Azure Command Line Interface command:

- Your resource group name (should be same as you are using elsewhere in these labs); `--resource-group` parameter
- Your Azure region (should be same as you are using elsewhere in these labs); `--location` parameter
- A name for your Azure container instance (see example command line below, `--name` parameter; you can leave that unchanged, or provide a different name if desired)
- Your event hub connection string. Please be careful to replace only the token [PROVIDE] in the `--environment-variables` parameter with your Event Hubs connection string!

Starter command line which you need to prepare with your specific values. Paste this into a text editor in order to prepare it.

```
az container create --resource-group [PROVIDE] --location [PROVIDE] --image pelazem/azurediscday-danrt:latest --name danrt-simulator --environment-variables 'EventHubConnString=[PROVIDE]'
```

Sample command line (with security details removed):

```
az container create --resource-group azurediscday --location eastus --image pelazem/azurediscday-danrt:latest --name danrt-simulator --environment-variables 'EventHubConnString=Endpoint=sb://pzazdisday.servicebus.windows.net/;SharedAccessKeyName=eh1send;SharedAccessKey=REMOVED;EntityPath=eh1'
```

When you have prepared the command line, copy-paste it into the Azure portal cloud shell and hit Enter. The shell will show “Running...” while the container image is retrieved from the Docker Hub and the container deployment is prepared. **Expect the deployment to take a minute or two.** When the deployment is complete, you will see a JSON status message in the cloud shell.

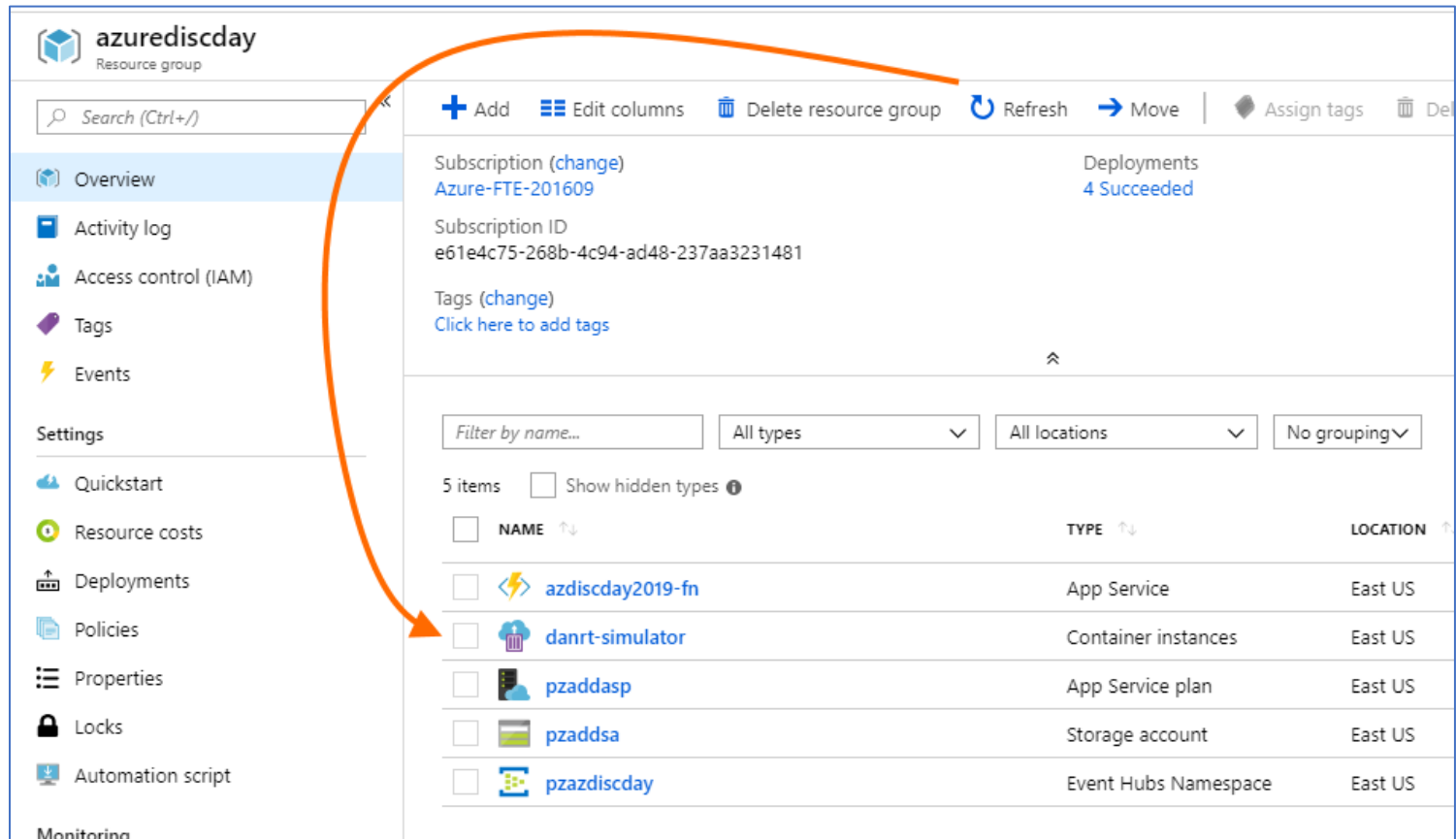
```
Bash
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

Welcome to Azure Cloud Shell

Type "az" to use Azure CLI 2.0
Type "help" to learn about Cloud Shell

patrick@Azure:~$ az container create --resource-group azuredisccday --image pelazem/azurediscday-danrt:latest --name danrt-simulator --environment-variables 'EventHubConnString=Endpoint=sb://pzazdiscday.servicebus.windows.net/;SharedAccessKeyName=eh1send;SharedAccessKey=B8G0ioKXpMrwFIGkwyLpER5bQLaTj65U0e6Meo0BtPo=;EntityPath=eh1'
{
  "containers": [
    {
      "command": null,
      "environmentVariables": [
        {
          "name": "EventHubConnString",
          "secureValue": null,
          "value": "Endpoint=sb://pzazdiscday.servicebus.windows.net/;SharedAccessKeyName=eh1send;SharedAccessKey=B8G0ioKXpMrwFIGkwyLpER5bQLaTj65U0e6Meo0BtPo=;EntityPath=eh1"
        }
      ],
      "image": "pelazem/azurediscday-danrt:latest",
      "instanceView": {
```

After the deployment completes and you see the JSON in the cloud shell, you can then also click “Refresh” in your Resource Group to see the new Container instance, and click on it to verify it’s running.



The screenshot displays the Azure portal interface for a resource group named 'azurediscday'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Events, Settings, Quickstart, Resource costs, Deployments, Policies, Properties, Locks, and Automation script. The main content area shows the resource group details, including the subscription (Azure-FTE-201609), subscription ID (e61e4c75-268b-4c94-ad48-237aa3231481), and tags. Below this, there are filters for name, type, location, and grouping. A table lists 5 items in the resource group:

NAME	TYPE	LOCATION
azdiscday2019-fn	App Service	East US
danrt-simulator	Container instances	East US
pzaddasp	App Service plan	East US
pzaddsa	Storage account	East US
pzazdiscday	Event Hubs Namespace	East US

An orange arrow originates from the 'Refresh' button in the top toolbar and points to the 'danrt-simulator' resource in the table.

Dashboard > azurediscday > danrt-simulator

danrt-simulator

Container instances

Search (Ctrl+/)

Start Restart Stop Delete Refresh

Overview

Activity log

Access control (IAM)

Tags

Settings

Containers

Properties

Locks

Resource group (change) [azurediscday](#)

Status Running

Location East US

Subscription (change) [Azure-FTE-201609](#)

Subscription ID e61e4c75-268b-4c94-ad48-237aa3231481

Tags (change) [Click here to add tags](#)

OS type Linux

IP address ---

FQDN ---

Container count 1

Note: you do NOT need to stop this Azure container instance during this lab, and at the end of the session, it will be cleaned up with other resources. However, if needed, you can stop the container using the “Stop” button shown in the portal, or the CLI command

```
az container stop --resource-group azurediscday --name danrt-simulator
```

Substitute your Resource Group name and Azure Container Instance name in that command before running it in the cloud shell.

Method 2 – Deploy device simulator on a Virtual Machine (VM)

NOTE: do NOT do the steps outlined in this task (Method 2) if you successfully completed the Method 1 task above.

On your computer or on a VM in Azure (your lab environment may have come with an Azure VM) or elsewhere, follow these steps:

1. Download and install the latest .NET Core Runtime from <https://dotnet.microsoft.com/download> (see screenshot below)
2. Navigate to <https://github.com/plzm/azure-discoveryday2019-mdw/tree/master/labs/lab3/StreamEventSender>. Download and unzip simulator.zip.
 - a. You may also be able to download it directly from <https://github.com/plzm/azure-discoveryday2019-mdw/blob/master/labs/lab3/StreamEventSender/simulator.zip?raw=true>
3. Open a command prompt in the folder where you unzipped the simulator.
4. Run this command: `dotnet Sender.dll "your Event Hub connection string"`
 - a. Provide your inbound Event Hub connection string in double quotes instead of the placeholder text shown

If you completed all the previous steps and the simulator starts successfully, you will see messages that are being sent to your inbound Event Hub echoed to the console (see screenshot below). You can also return to your Azure Function's "Logs" tab and see Executing/Executed status messages being written to the console (and remember that you can uncomment or add log statements to send more status messages to the "Logs" console).

https://dotnet.microsoft.com/download

Microsoft | .NET About Learn Architecture Docs Downloads Community

.NET Downloads

Not sure where to start? See [Hello World in 10 minutes](#) to install .NET and build

Windows Linux macOS

.NET Core

.NET Framework

.NET Core 2.2

.NET Core is a cross-platform version of .NET for building websites, services, and console apps.

Build Apps ⓘ [Download .NET Core SDK](#)

Run Apps ⓘ [Download .NET Core Runtime](#)

.NET Framework

.NET Framework is a Windows-only platform for building any type of app that runs on Windows.

Build Apps ⓘ [Download .NET Framework SDK](#)

Run Apps ⓘ [Download .NET Framework Runtime](#)

Download the .NET Core Runtime

```

C:\Users\paellaz\Desktop\discday19sim>dotnet Sender.dll "Endpoint=sb://azdiscday2019.servicebus.windows.net/;SharedAccessKeyName=ehinbound_Send;SharedAccessKey=
Running. Press Ctrl-C to end.
{"trip_type":2,"trip_year":"2019","trip_month":"01","taxi_type":"Yellow","vendor_id":2,"pickup_datetime":"2019-01-21T15:38:48.8681066-05:00","dropoff_datetime":
43084531,"rate_code_id":1,"store_and_fwd_flag":"","pickup_location_id":146,"dropoff_location_id":102,"pickup_longitude":-72.301969,"pickup_latitude":40.8428
":16,"fare_amount":16.58,"extra":0.0,"mta_tax":0.82,"tip_amount":2.48,"tolls_amount":0.0,"improvement_surcharge":0.0,"ehail_fee":0.0,"total_amount":19.88,"custo

{"trip_type":1,"trip_year":"2019","trip_month":"01","taxi_type":"Green","vendor_id":3,"pickup_datetime":"2019-01-21T15:41:50.8571846-05:00","dropoff_datetime":
814785787,"rate_code_id":6,"store_and_fwd_flag":"","pickup_location_id":230,"dropoff_location_id":100,"pickup_longitude":-74.580670,"pickup_latitude":39.913
e":2,"fare_amount":28.03,"extra":0.0,"mta_tax":1.4,"tip_amount":5.6,"tolls_amount":0.0,"improvement_surcharge":0.0,"ehail_fee":0.0,"total_amount":35.03,"custom
food, and a blankie - wow send him every time!"}

{"trip_type":2,"trip_year":"2019","trip_month":"01","taxi_type":"Green","vendor_id":2,"pickup_datetime":"2019-01-21T15:29:24.8518762-05:00","dropoff_datetime":
82168152832,"rate_code_id":6,"store_and_fwd_flag":"","pickup_location_id":240,"dropoff_location_id":109,"pickup_longitude":-72.116189,"pickup_latitude":39.8
ype":2,"fare_amount":12.62,"extra":0.0,"mta_tax":0.63,"tip_amount":3.15,"tolls_amount":0.0,"improvement_surcharge":0.0,"ehail_fee":0.0,"total_amount":16.4,"cus

{"trip_type":2,"trip_year":"2019","trip_month":"01","taxi_type":"Yellow","vendor_id":3,"pickup_datetime":"2019-01-21T15:33:07.0812753-05:00","dropoff_datetime":
6803643549,"rate_code_id":1,"store_and_fwd_flag":"","pickup_location_id":186,"dropoff_location_id":35,"pickup_longitude":-72.815551,"pickup_latitude":40.7450
e":2,"fare_amount":12.81,"extra":0.0,"mta_tax":0.64,"tip_amount":1.92,"tolls_amount":0.0,"improvement_surcharge":0.0,"ehail_fee":0.0,"total_amount":15.37,"custo

{"trip_type":2,"trip_year":"2019","trip_month":"01","taxi_type":"Green","vendor_id":2,"pickup_datetime":"2019-01-21T15:00:08.3630954-05:00","dropoff_datetime":
7316801742,"rate_code_id":6,"store_and_fwd_flag":"","pickup_location_id":25,"dropoff_location_id":200,"pickup_longitude":-73.706504,"pickup_latitude":40.9710
e":2,"fare_amount":18.31,"extra":0.0,"mta_tax":0.91,"tip_amount":2.74,"tolls_amount":0.0,"improvement_surcharge":0.0,"ehail_fee":0.0,"total_amount":21.96,"custo

{"trip_type":2,"trip_year":"2019","trip_month":"01","taxi_type":"Yellow","vendor_id":1,"pickup_datetime":"2019-01-21T15:01:34.4649766-05:00","dropoff_datetime":
995731482279,"rate_code_id":1,"store_and_fwd_flag":"","pickup_location_id":74,"dropoff_location_id":87,"pickup_longitude":-74.853981,"pickup_latitude":40.95
pe":1,"fare_amount":13.78,"extra":0.0,"mta_tax":0.68,"tip_amount":2.75,"tolls_amount":0.0,"improvement_surcharge":0.0,"ehail_fee":0.0,"total_amount":17.21,"cus
got me and my friends laughing"}

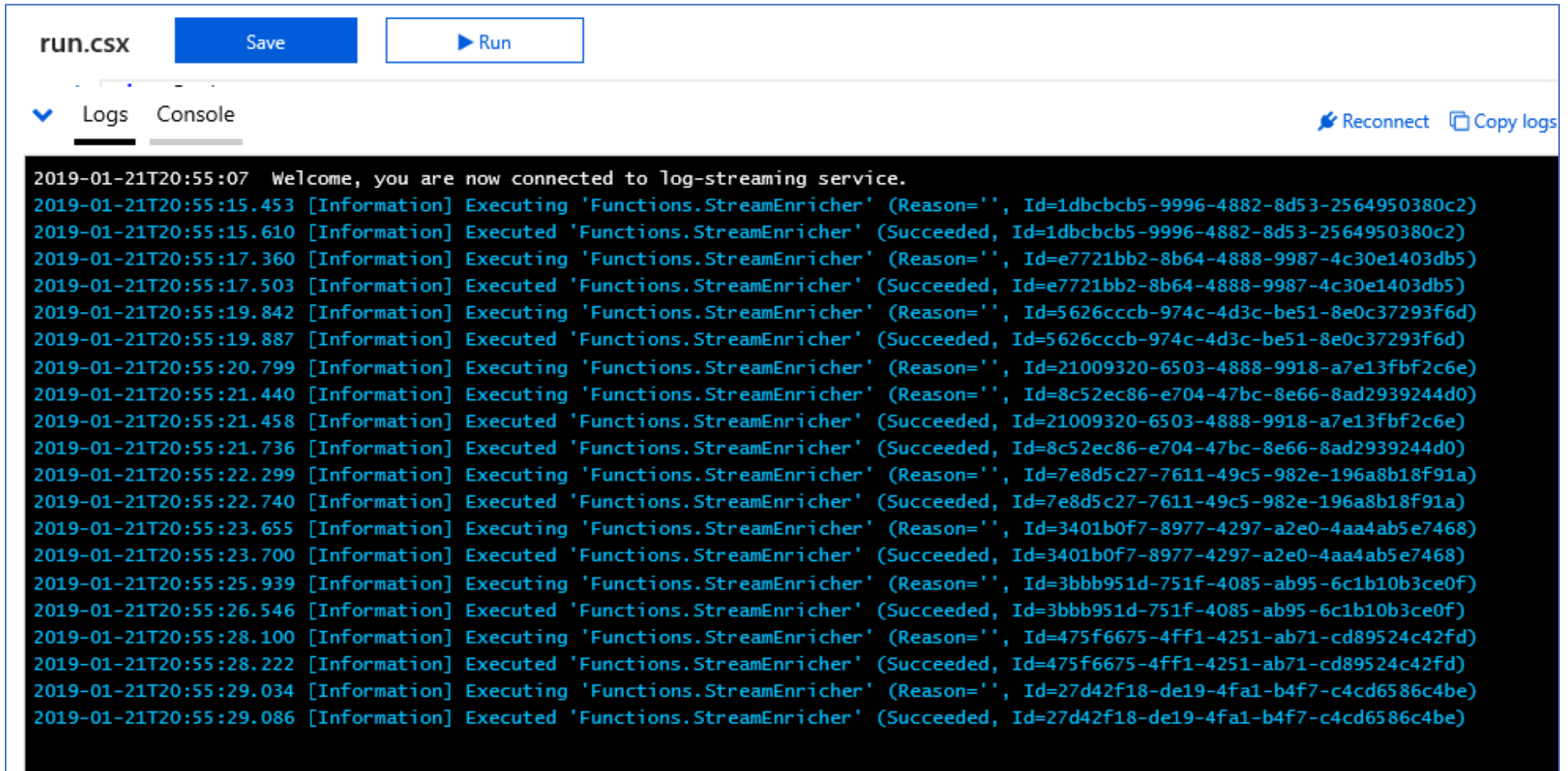
{"trip_type":1,"trip_year":"2019","trip_month":"01","taxi_type":"Yellow","vendor_id":3,"pickup_datetime":"2019-01-21T15:30:07.8357447-05:00","dropoff_datetime":
1411354406,"rate_code_id":6,"store_and_fwd_flag":"","pickup_location_id":76,"dropoff_location_id":176,"pickup_longitude":-74.340522,"pickup_latitude":39.707
":1,"fare_amount":22.00,"extra":0.0,"mta_tax":1.1,"tip_amount":5.53,"tolls_amount":0.0,"improvement_surcharge":0.0,"ehail_fee":0.0,"total_amount":28.71,"custo

```

Message Simulator running

Monitoring

Whichever method you used to deploy the device simulator, you should now be able to switch to your Azure Function's Log tab and see incoming messages streaming in.



The screenshot shows the Azure Functions portal interface. At the top, there's a file named 'run.csx' with 'Save' and 'Run' buttons. Below this, the 'Logs' tab is selected, showing a stream of log messages. The messages are timestamped and include the function name 'Functions.StreamEnricher' and its status (Executing or Executed). The status is followed by a reason (empty string) and a unique ID. The logs show a continuous stream of messages, indicating that the function is processing inbound messages.

```
run.csx Save Run

Logs Console Reconnect Copy logs

2019-01-21T20:55:07 Welcome, you are now connected to log-streaming service.
2019-01-21T20:55:15.453 [Information] Executing 'Functions.StreamEnricher' (Reason='', Id=1dbcbcb5-9996-4882-8d53-2564950380c2)
2019-01-21T20:55:15.610 [Information] Executed 'Functions.StreamEnricher' (Succeeded, Id=1dbcbcb5-9996-4882-8d53-2564950380c2)
2019-01-21T20:55:17.360 [Information] Executing 'Functions.StreamEnricher' (Reason='', Id=e7721bb2-8b64-4888-9987-4c30e1403db5)
2019-01-21T20:55:17.503 [Information] Executed 'Functions.StreamEnricher' (Succeeded, Id=e7721bb2-8b64-4888-9987-4c30e1403db5)
2019-01-21T20:55:19.842 [Information] Executing 'Functions.StreamEnricher' (Reason='', Id=5626cccb-974c-4d3c-be51-8e0c37293f6d)
2019-01-21T20:55:19.887 [Information] Executed 'Functions.StreamEnricher' (Succeeded, Id=5626cccb-974c-4d3c-be51-8e0c37293f6d)
2019-01-21T20:55:20.799 [Information] Executing 'Functions.StreamEnricher' (Reason='', Id=21009320-6503-4888-9918-a7e13fbf2c6e)
2019-01-21T20:55:21.440 [Information] Executing 'Functions.StreamEnricher' (Reason='', Id=8c52ec86-e704-47bc-8e66-8ad2939244d0)
2019-01-21T20:55:21.458 [Information] Executed 'Functions.StreamEnricher' (Succeeded, Id=21009320-6503-4888-9918-a7e13fbf2c6e)
2019-01-21T20:55:21.736 [Information] Executed 'Functions.StreamEnricher' (Succeeded, Id=8c52ec86-e704-47bc-8e66-8ad2939244d0)
2019-01-21T20:55:22.299 [Information] Executing 'Functions.StreamEnricher' (Reason='', Id=7e8d5c27-7611-49c5-982e-196a8b18f91a)
2019-01-21T20:55:22.740 [Information] Executed 'Functions.StreamEnricher' (Succeeded, Id=7e8d5c27-7611-49c5-982e-196a8b18f91a)
2019-01-21T20:55:23.655 [Information] Executing 'Functions.StreamEnricher' (Reason='', Id=3401b0f7-8977-4297-a2e0-4aa4ab5e7468)
2019-01-21T20:55:23.700 [Information] Executed 'Functions.StreamEnricher' (Succeeded, Id=3401b0f7-8977-4297-a2e0-4aa4ab5e7468)
2019-01-21T20:55:25.939 [Information] Executing 'Functions.StreamEnricher' (Reason='', Id=3bbb951d-751f-4085-ab95-6c1b10b3ce0f)
2019-01-21T20:55:26.546 [Information] Executed 'Functions.StreamEnricher' (Succeeded, Id=3bbb951d-751f-4085-ab95-6c1b10b3ce0f)
2019-01-21T20:55:28.100 [Information] Executing 'Functions.StreamEnricher' (Reason='', Id=475f6675-4ff1-4251-ab71-cd89524c42fd)
2019-01-21T20:55:28.222 [Information] Executed 'Functions.StreamEnricher' (Succeeded, Id=475f6675-4ff1-4251-ab71-cd89524c42fd)
2019-01-21T20:55:29.034 [Information] Executing 'Functions.StreamEnricher' (Reason='', Id=27d42f18-de19-4fa1-b4f7-c4cd6586c4be)
2019-01-21T20:55:29.086 [Information] Executed 'Functions.StreamEnricher' (Succeeded, Id=27d42f18-de19-4fa1-b4f7-c4cd6586c4be)
```

Monitoring Azure Function processing inbound messages

Conclusion

Congratulations! You have completed lab 3.

In this lab, you built a streaming pipeline and added near real-time processing and message enrichment, then forwarded the enriched messages for further processing downstream. That's what you'll work on in lab 4.