# 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, which will include both a numerical score (good/bad) as well as free-form comments about the trip.

### References

### General Notes

### Architecture for this Lab

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



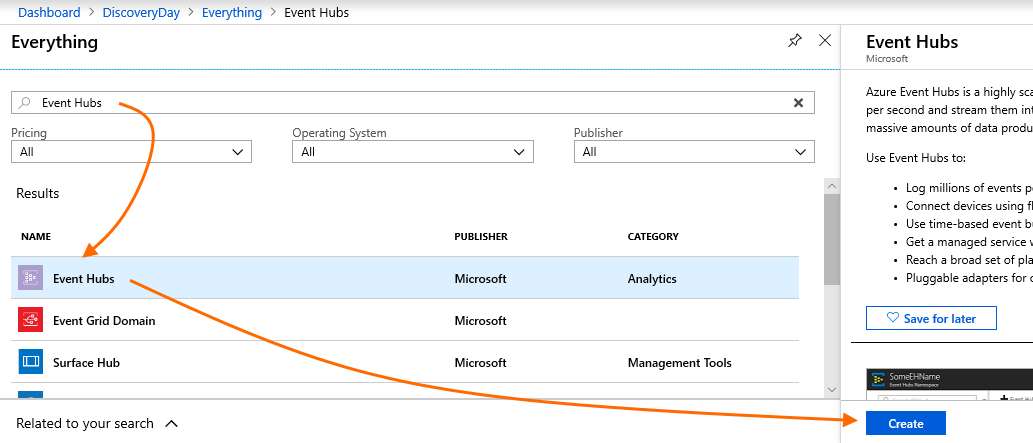
### Task 1 – Set up two stream ingestion endpoints

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.

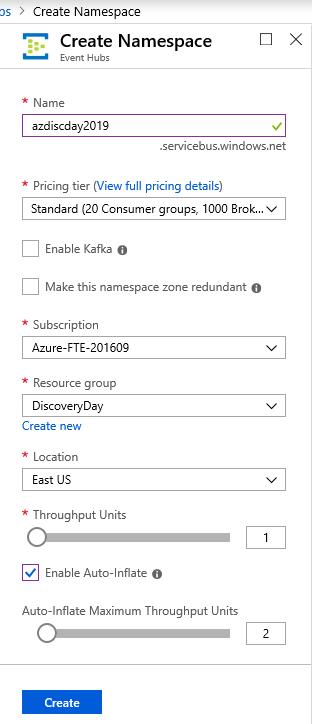
In the search box, type “Event Hubs” followed by Enter. Click on the Event Hubs entry, then “Create” on its product blade.



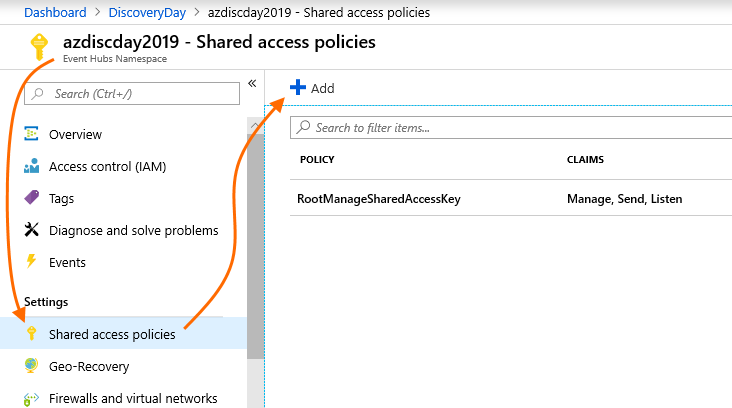
Provide appropriate values on the blade to create an Event Hubs Namespace. Specifically:

* 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.

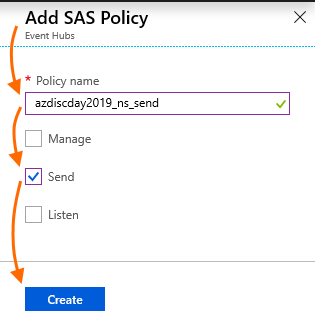
Then click Create.



When deployment completes, click on the new Event Hubs Namespace resource in your Resource Group. Then click on “Shared access policies”, and “+ Add” to add a new policy.

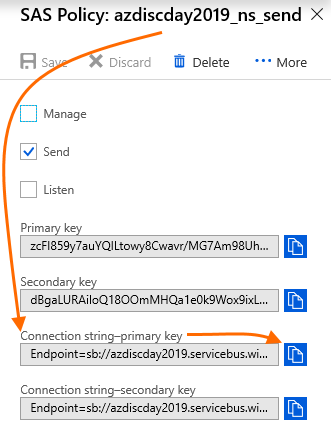


On the create blade, provide an appropriate name for the policy and ensure only “Send” is checked, then click “Create”.



This policy will be used by the taxi device simulator to send in messages; it does not need “Manage” or “Listen” capabilities (as we do not want our taxi devices to be able to manage our stream ingestion endpoints, or access sent-in messages).

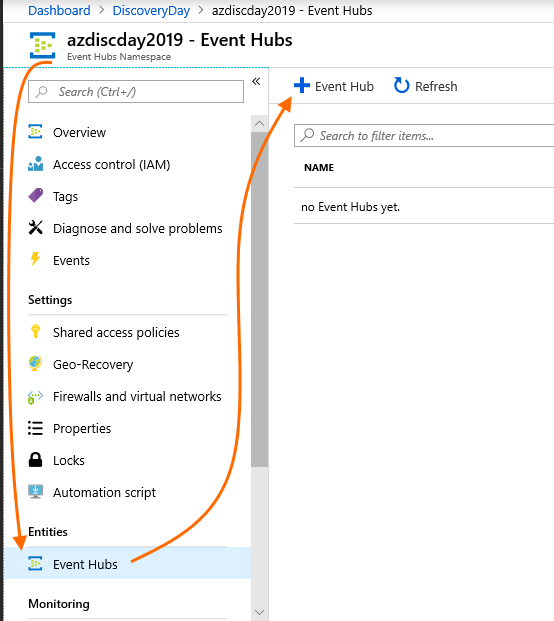
After creation completes, you will see the new policy in the Shared access policies view where you clicked “+Add”. Click on the new policy to view its properties. Copy one of its connection strings to a scratch area; you will need this later to configure the device simulator.



Close the SAS Policy blade to return to the Event Hubs Namespace view.

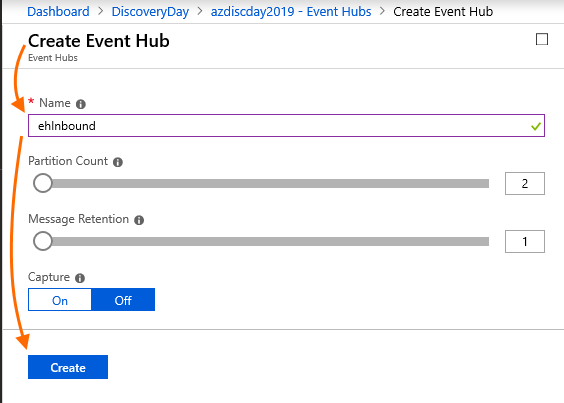
You will now create two Event Hubs in this Event Hubs Namespace. This first Event Hub will be the endpoint to which the taxi device simulator will send messages.

To start, click on “Entities/Event Hubs”. On the Event Hubs view, click “+ Event Hub”.

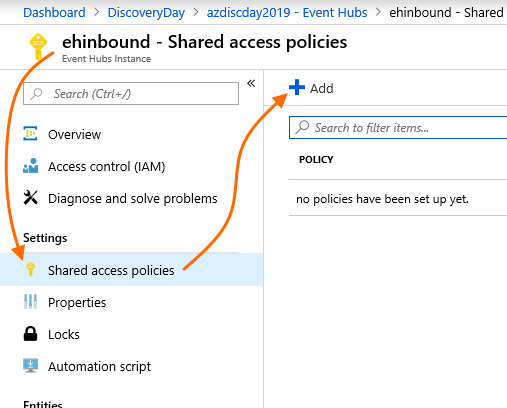


In the “Create Event Hub” view, provide a name for this Event Hub. Other inputs are good with their default values.

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”.

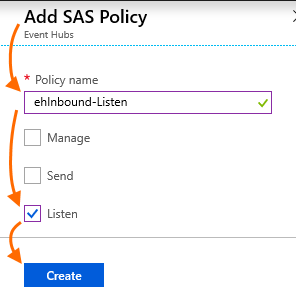


After the Event Hub is created, you will see it in the Event Hubs list on the view where you just clicked “+ Event Hub”. Now click the Event Hub you just created, then click “Shared access policies”, then “+ Add”.



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.

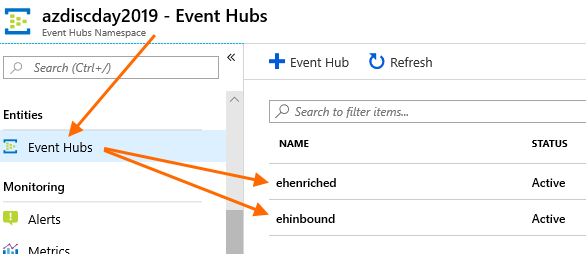


After the policy is created, return to the Event Hubs Namespace “Event Hubs” view, where your first Event Hub is shown in the list.

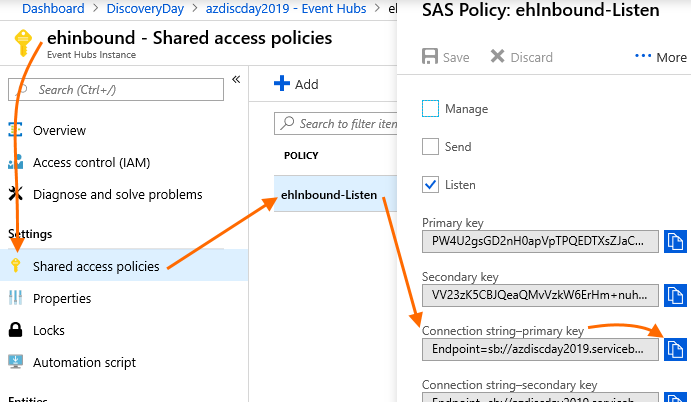
Now, create another Event Hub. This will be the second messaging endpoint shown in the architecture diagram (see page 1 of this document). The Azure Function you will create in a later task will send the enriched message stream to this second Event Hub.

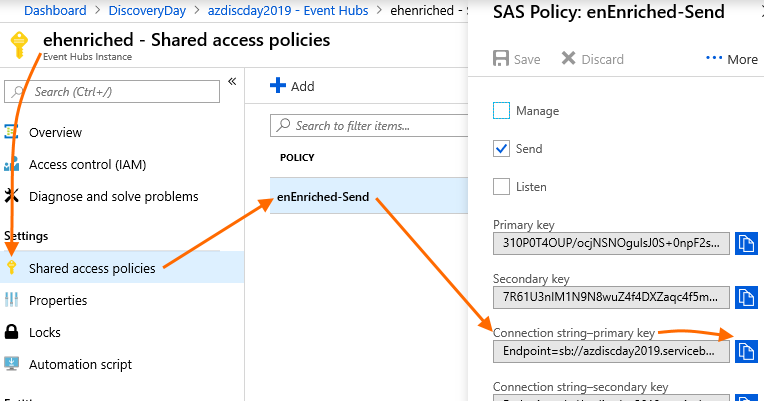
**Proceed as with the first Event Hub, but with this important difference:** create a Shared access policy for this second Event Hub, but with Send permission only. The Azure Function will use this policy to send enriched messages onward, and Listen and Manage are not needed.

When you are done, you should see two Event Hubs listed in the Event Hubs Namespace list. Each should have one SAS Policy.



To conclude this task, click into each of the two Event Hubs. In each Event Hub, click “Shared access policies”. Then click the policy you created to show its properties. Copy the policy’s connection string and save it in a scratch pad area for later use (or, of course, you can always come back to this view when you need the connection string).





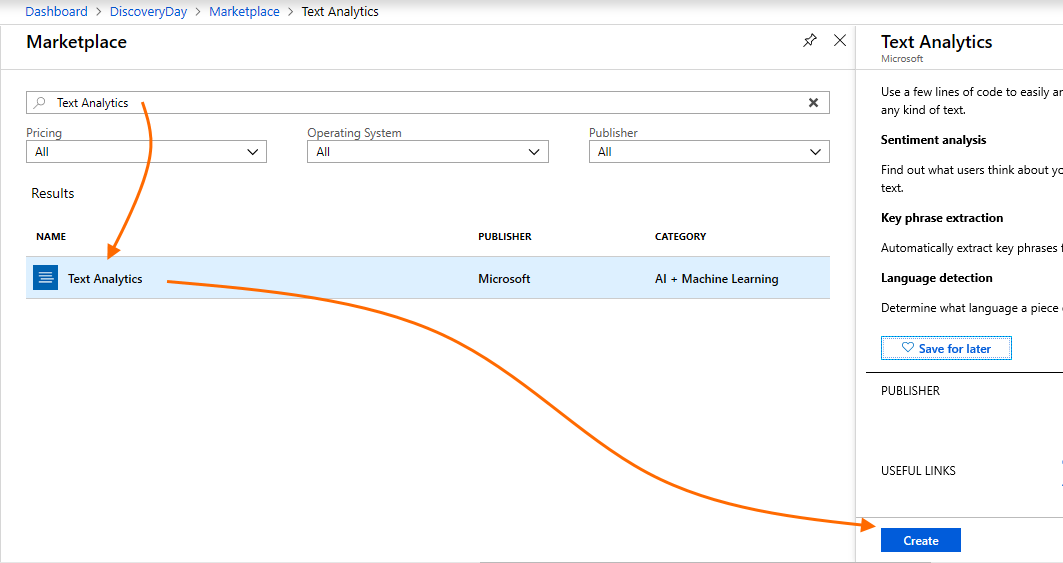
In summary, when you have successfully deployed the following, then this task is complete:

1. Event Hubs Namespace
   1. Send Shared Access Policy for the Namespace, to be used by the taxi device simulator to send in messages.
2. Event Hub for inbound taxi device messages
   1. Listen Shared Access Policy for the Event Hub, to be used by the Azure Function to listen for incoming messages.
3. Event Hub to forward enriched taxi device messages
   1. Send Shared Access Policy for the Event Hub, to be used by the Azure Function to send on messages it has enriched.

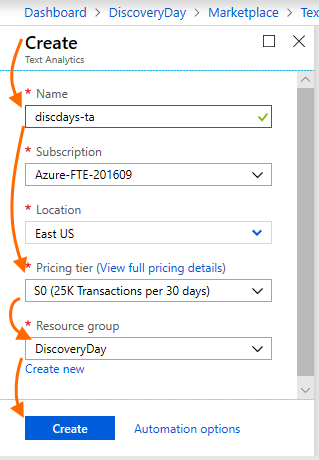
### Task 2 – Deploy Text Analytics Cognitive Service

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.

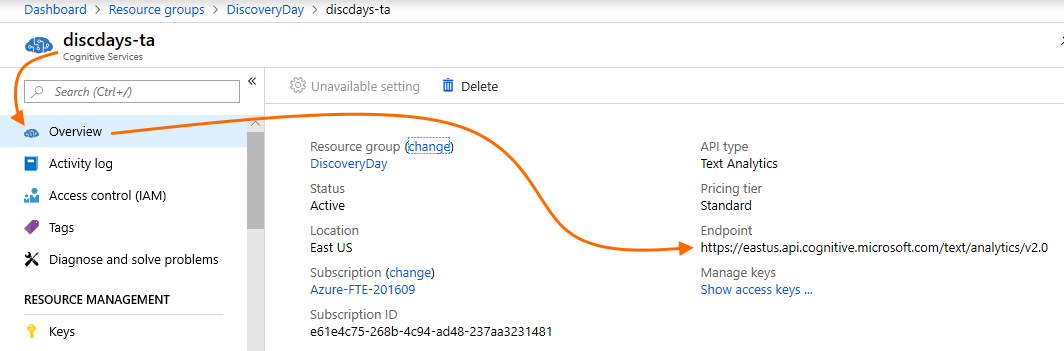
In your Resource Group, click “+ Add” again to add a new resource. Type “Text Analytics” in the search box, then Enter.

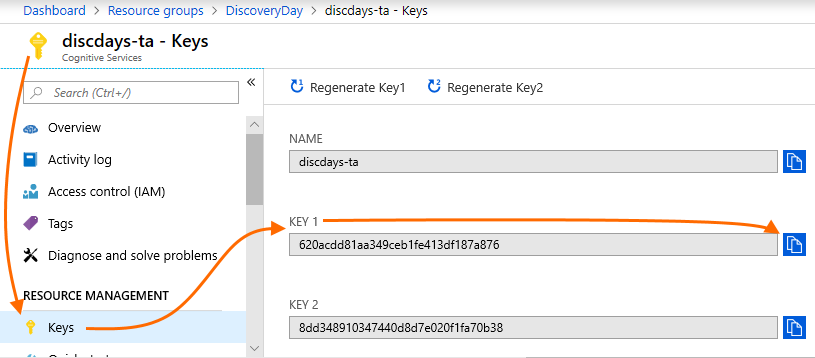


On the Create blade, provide a name. Select the S0 pricing tier, and ensure that your Resource Group is selected. Then click “Create”.



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.





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.

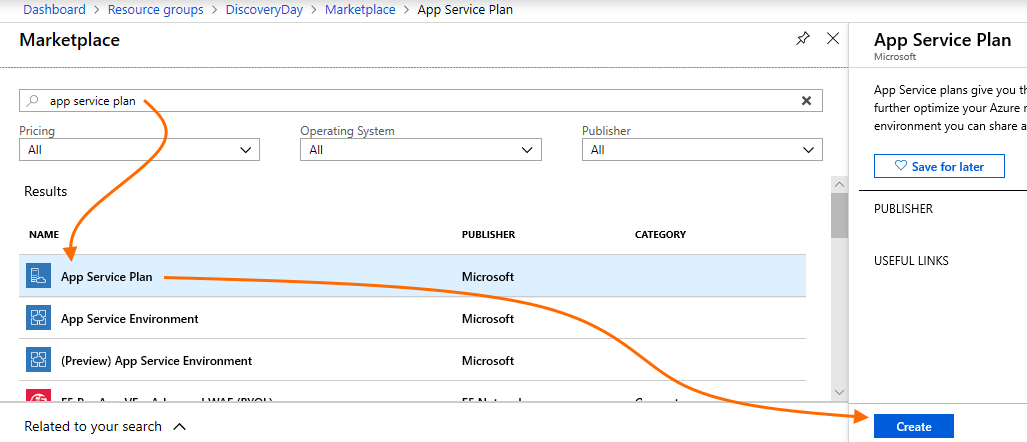
### Task 3 – Deploy Azure Function to Process and Enrich Taxi Messages

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

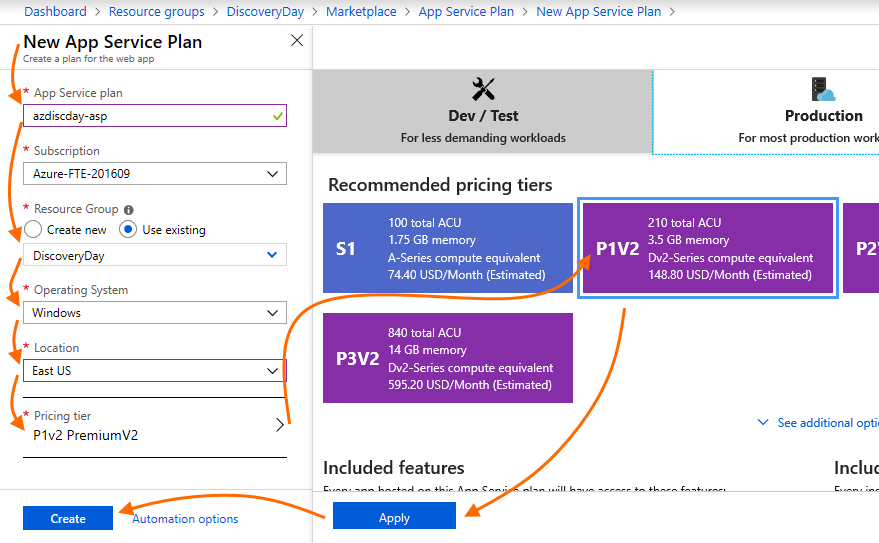
This Function will be triggered to run every time a taxi device message is received by the first Event Hub you deployed in task 1. It will then get the customer’s comments from the message, and run those comments through the text analytics cognitive service you created in task 2. Then, the text analytics 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”.

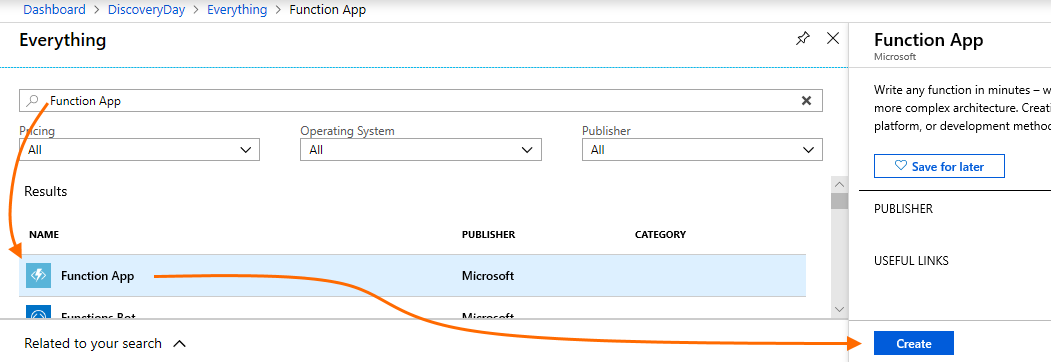


On the create blade, provide a name for the App Service Plan. Then, ensure your Resource Group is selected; leave Operating System at Windows; set the Location to the Azure region you have been using so far; then click Pricing Tier. On that view, select P1V2 and click “Apply”. Then, click “Create”.



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.

