Design Advanced Applications Lab

Create a Stream Analytics Pipeline

The lab creates a simple stream analytics pipeline using patterns and services that are common for IoT and other stream input scenarios. Four Azure services are involved – two of which you dealt with in the previous lab:

1. Stream Analytics
2. Service Bus (topics)
3. Function App
4. Azure Storage

This pipeline starts with mock incoming sensor data representing temperature measurements; these are stored in an external blob storage account for you to access, but in a true IoT application, the input would likely be coming from an IoT Hub instance to which remote sensors are reporting data.

The Temperature Validator is a Stream Analytics job that accepts the temperature data and forwards any data considered out-of-bounds to a Service Bus topic for further handling. In this case, the Stream Analytics job is hard-coded to reject any temperature above 125 degrees. Note that Stream Analytics does have a mechanism for such range thresholds to be provided dynamically (namely Reference Data store in a Blob).

When a message arrives at the Service Bus, it triggers an Azure Function to then copy the out-of-range record to a Azure storage account table in your own subscription. Since this architecture uses a topic, there could be other subscribers to that message that might alert a manager via text or push an alert to a dashboard display.

Download the Lab Files

To get the lab files navigate to

http:// rebrand.ly/70-534-AdvAppsLab

and save the downloaded file. Copy the file to a directory of your choice on your local machine, and extract the contents.

You should see four files:

`provision-70-534.ps1`: PowerShell script to deploy the lab services

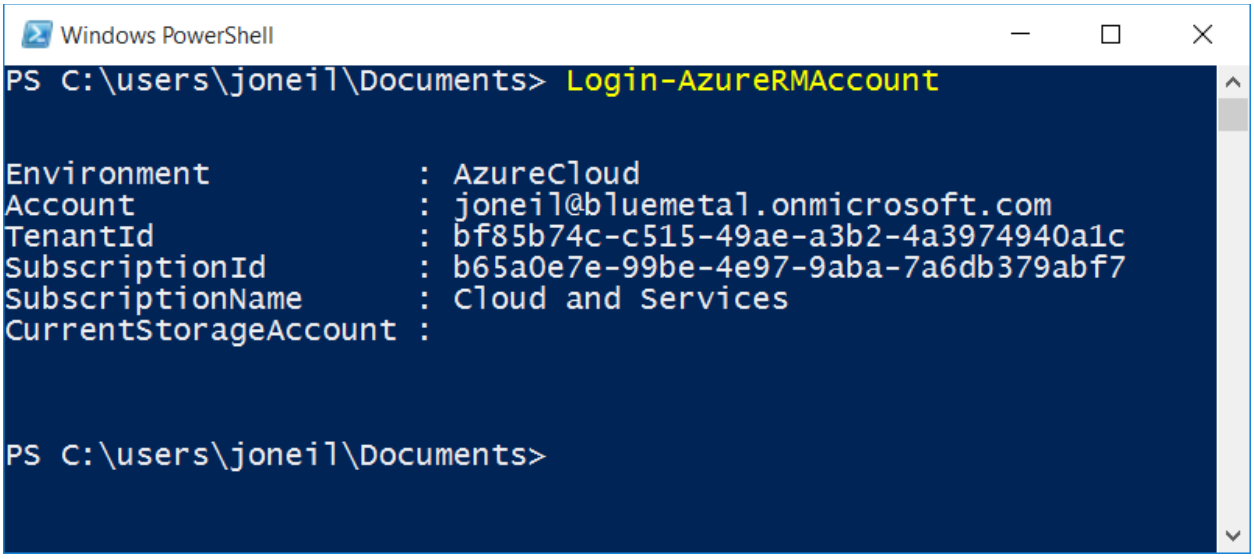
`arm-advapps.json:`	Azure ARM template for the lab services
`RecordAlert.zip:`	Function app implementation files
`LabInstructions.pdf:`	this document

Provision the Services

Rather than use the Azure Portal to create the services, you'll leverage an Azure Resource Template and a PowerShell script to provision all the needed services.

Start a PowerShell session and navigate to the directory where you unzipped the lab files, specifically the directory that contains the file `provision-70-534.ps1.`

First login to your Azure Account using the **`Login-AzureRMAccount`** cmdlet.



You should see one of your subscriptions returned as the current context. If you haven't already run **`Add-AzureRMAccount`** in PowerShell, you may need to do that first.

If the subscription that is show after you login is not the subscription you wish to use for this lab, you can run **`Get-AzureRMSubscription`** to get a list of your subscriptions and then **`Select-AzureRMSubscription`** with either the `-SubscriptionID` or the `-SubscriptionName` argument to switch the context.

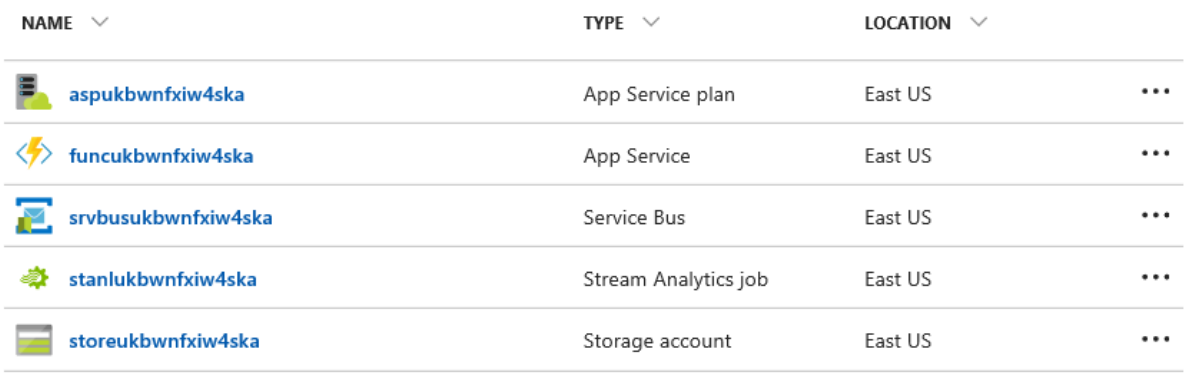
Next, run the command `.\ provision-70-534.ps1` (don't forget the .\ at the front!). The script prompts you for a Resource Group name where you would like to store the Azure services. If the Resource Group does not exist, it will be created; if it does exist, the services will be added to that group, but other existing services in that group will not be modified or deleted.

Note that the Resource Group and services will be created in the East US region by default. You can modify the `-Location` parameter in the last line of code in `.\ provision-70-534.ps1` to specify a different region is you prefer.

The script performs three primary steps:

1. Validates the Resource Group to which to provision the Azure services
2. Provisions the services
3. Uses the Azure ARM REST API to upload the Azure function code

When the script has completed, visit the Azure Portal and you should see six services deployed in the selected Resource Group, such as below. The 'random' 13-character suffix is a generated ID (based on the Resource Group ID) that helps avoid (but not absolutely eliminates) the chances of namespace collisions for services that require globally unique endpoints – as the Azure Storage Service, Service Bus, and Function Apps do.

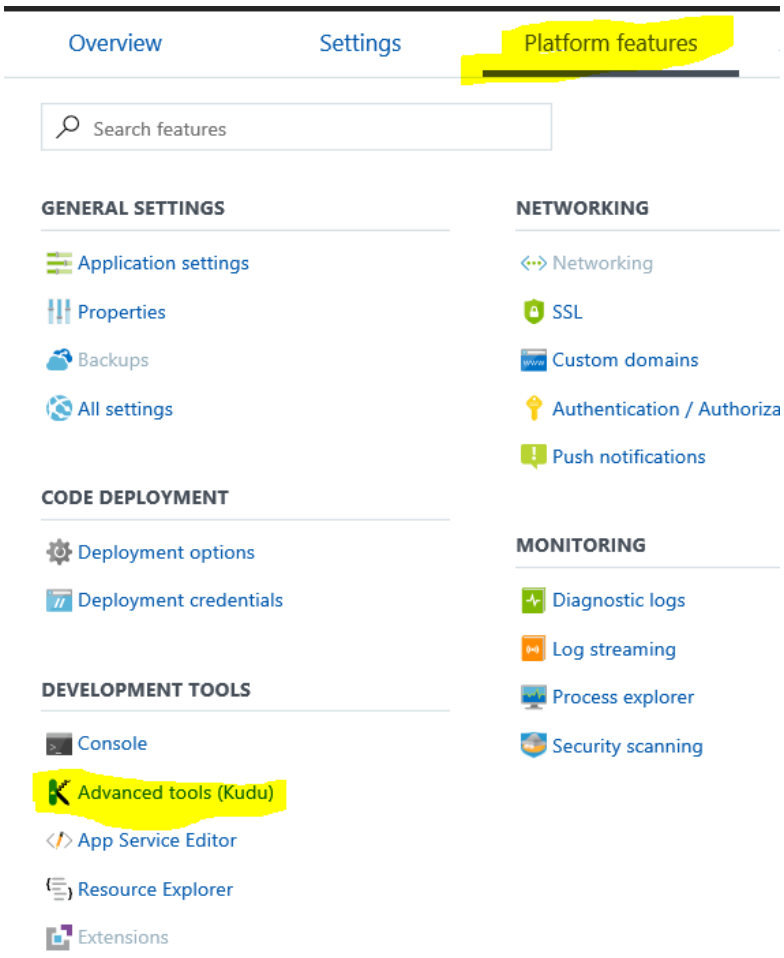


NAME	TYPE	LOCATION	
aspukbwnfxiw4ska	App Service plan	East US	•••
funcukbwnfxiw4ska	App Service	East US	•••
srvbusukbwnfxiw4ska	Service Bus	East US	•••
stanlukbwnfxiw4ska	Stream Analytics job	East US	•••
storeukbwnfxiw4ska	Storage account	East US	•••

These services map to the diagram provided earlier in this lab document with two exceptions:

1. The App Service plan is created to provide the resources on which to run the Azure Function App (noted as App Service in the listing). The function was created using a Consumption plan, so this service is automatically created, and you can essentially ignore (but not delete!) it.
2. The Sensor Data input from the diagram comes from an external 'hard-coded' Azure account in a separate subscription.

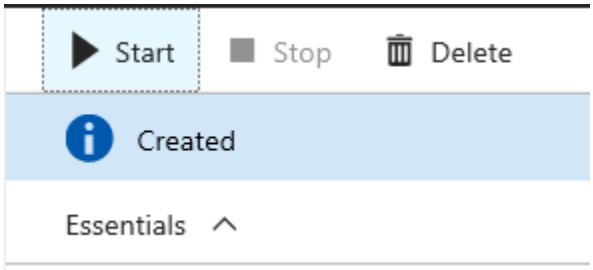
If you receive an error during "Deploying Function Application" step, you will need to manually deploy the function application directory (RecordAlert) below the `/wwwroot` directory in the Function app before proceeding. You can use Kudu for this; it is accessible via the *Platform features* item of your Function app.



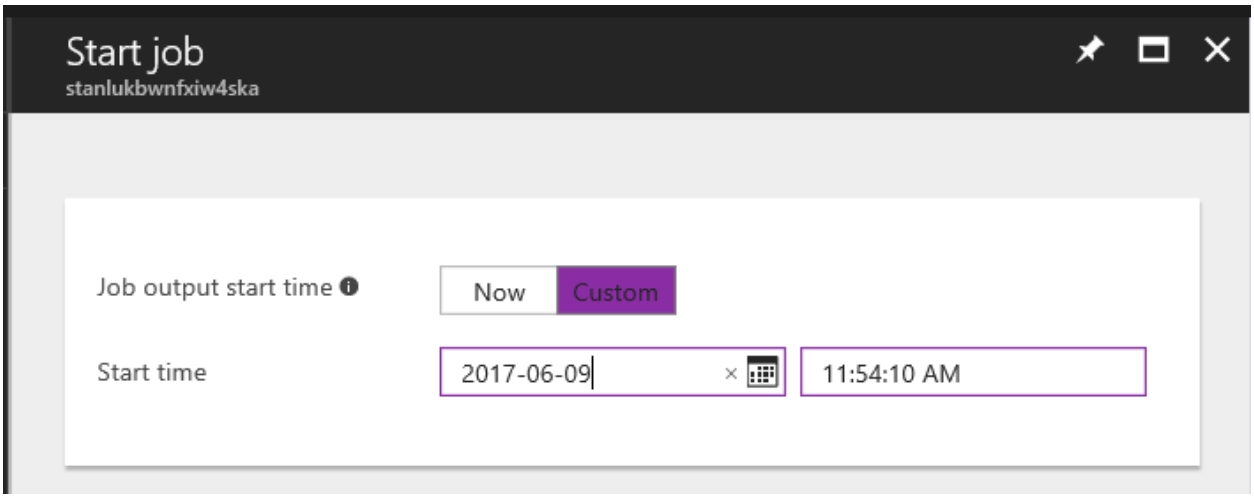
Start the Stream Analytics Job

In the portal, open the Stream Analytics job that was just provisioned. It includes all of the necessary input, output, and transformation properties for the lab.

To start the job, press the Start button/link at the top of the Stream Analytics Blade



Because Stream Analytics is designed for streaming data, when the job starts it need to know when to start sampling the incoming stream. Select the "Now" option, which will process the input blob. Note that the time stamp data within the blob records is opaque to Stream Analytics; however, the blob creation date plays a role here, so select a date earlier than Jun 12, 2017.



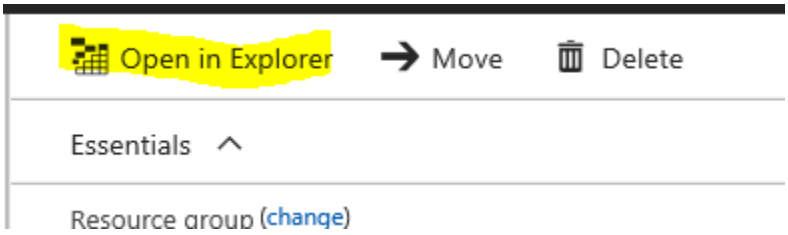
Note, if you stop and start this job a second time. You'll see another 'start time' option, but continue to provide the a custom date for subsequent runs.

It takes several minutes for a Stream Analytics job to start processing data, so in the meantime…

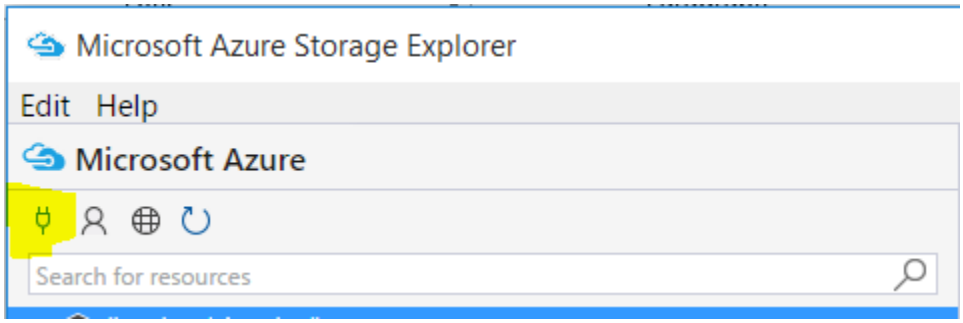
Configure Azure Storage Explorer to View the Output

The output data from this job (temperature data that exceeds 125 degrees) is directed to a table called `alerts` in your Azure Storage account. If you currently have a utility to explore Azure Storage accounts (such as Cloudberry or Cerebrata) configure it to support the storage account for this lab, and you should see 11 records populated in the table each time you run the analytics job.

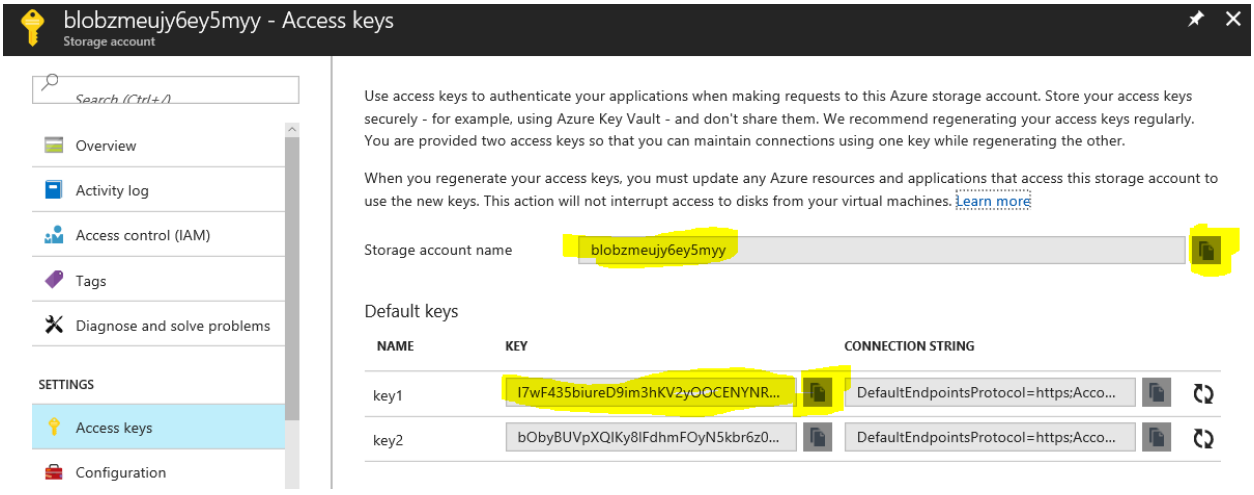
If you do not have a storage tool installed, got to the Azure Storage Account in the portal and select *Open in Explorer*



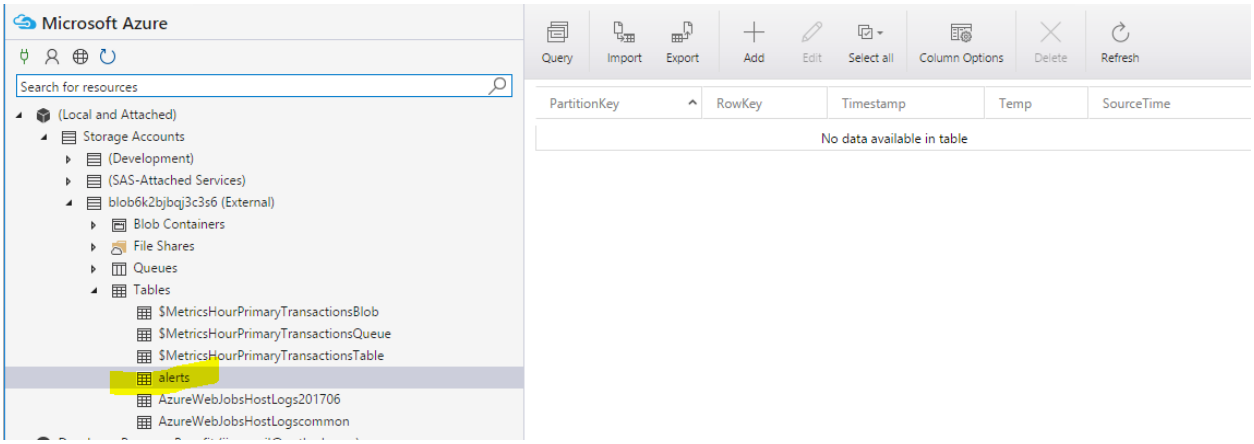
This will download the Microsoft Azure Storage Explorer. Configure a connection to your storage account by clicking the plug icon:



You will be prompted for your Azure Storage Key and the Storage Account Name, both available from the Azure Portal



Once you've set up your account, you can browse the files, blob, tables, and queues stored in that account. Find the alerts table and you can view and query the contents. Note there are additional other tables that are used to support the Azure Function App and monitoring. For the lab, we repurposed the Function App's storage account for our own application data, but in a production environment you would want to provision a separate Azure Storage Account.



If you find the data is not being populated, the `AzureWebJobsHostLog` table(s) may contain helpful information (assuming that data has arrived at the Service Bus topic). For further troubleshooting, via the Azure portal you can turn on diagnostics for each of the services in the pipeline to pinpoint the source of any issues.

Now that you have a running solution, you can dig deeper into the service implementation, Function App code, and even add more services to the pipeline by expanding the Azure ARM template.