# Project “Johannesburg”

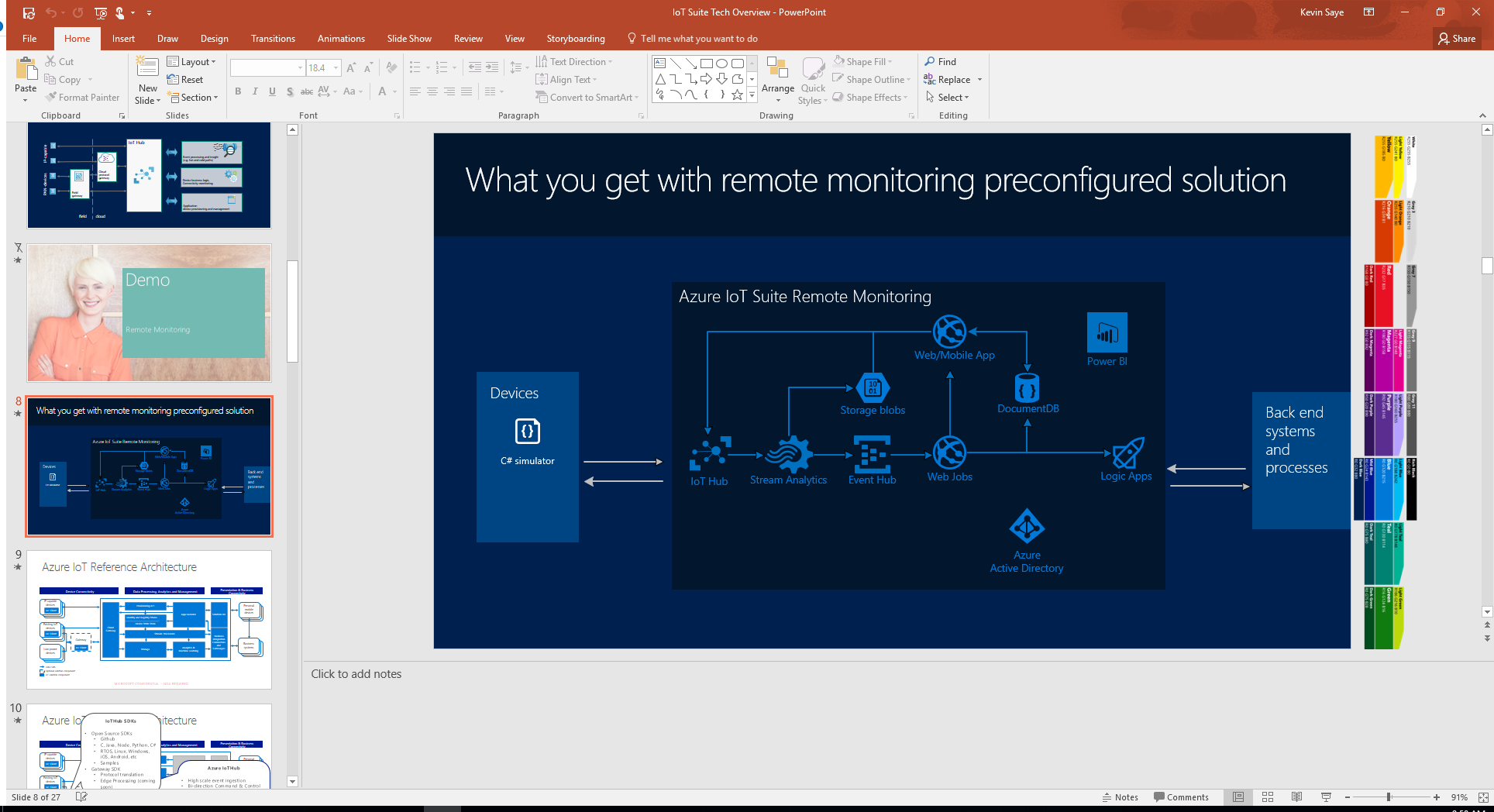
# Hub Hands on Lab

## Level 300

By: Kevin Saye

Azure IoT Black Belt, Microsoft Corporation

Version: May 2017



# Overview:

IoT brings the power and data of millions of devices and sensors to make critical business decisions. Project “Johannesburg” brings routing capabilities to industrial and professional trucks, considering unique aspects like hazardous chemicals, truck height and road inclines. Together, we can automatically plan, route, report and manage safety of industrial trucks while also ensuring public safety.

In this hands-on lab, we will have a simulated truck with GPS coordinates and truck attributes, such as height and cargo. We will then allow the driver to request a route, either using facial recognition, voice recognition or manually determining the destination.

# Requirements:

The student will need:

1. Access to an Azure subscription.
2. Access to a Linux device (for device emulation)
3. Basic programming knowledge. We will program in Python and C# using both libraries and REST endpoints

# Architecture Overview:

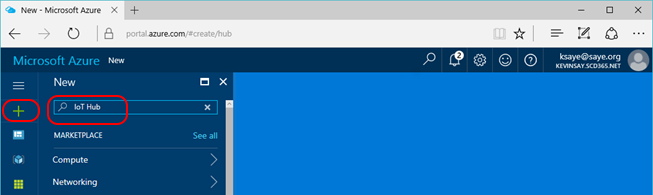
This lab will demonstrate a device communicating with Azure IoT Hub. Messages on the hub are processed by Azure Functions and interact with: 3 Cognitive Services, Bing Maps and a representative scheduling database.



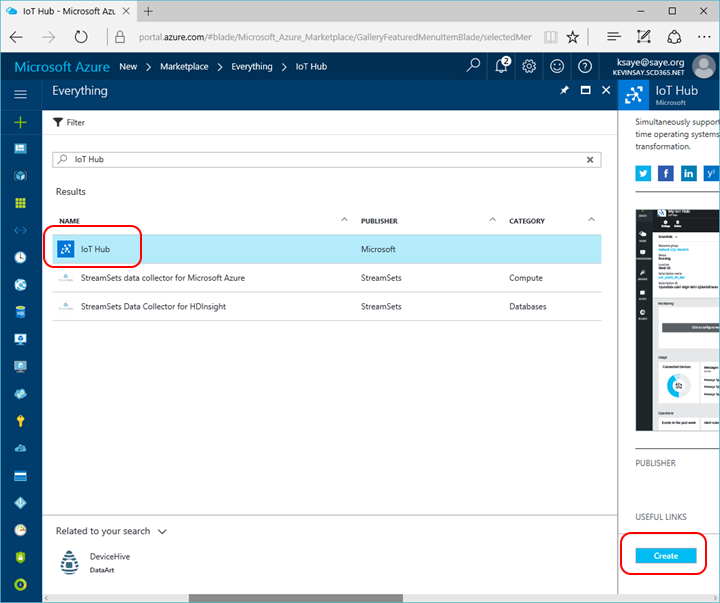
For this lab, the Bing Maps API, the Speech Cognitive Service and Face Cognitive Service API keys will be provided.

# Setting up IoT Hub

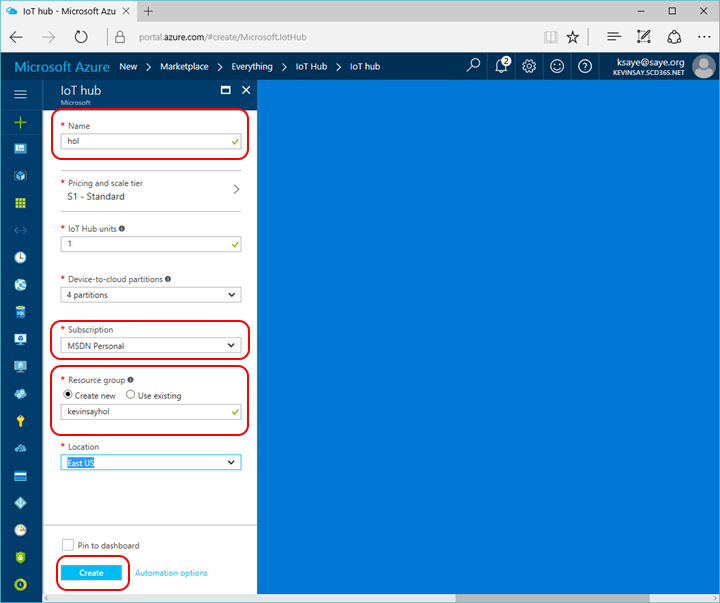
1. Go to <http://portal.azure.com>, click the Plus sign on the left and search for IoT Hub



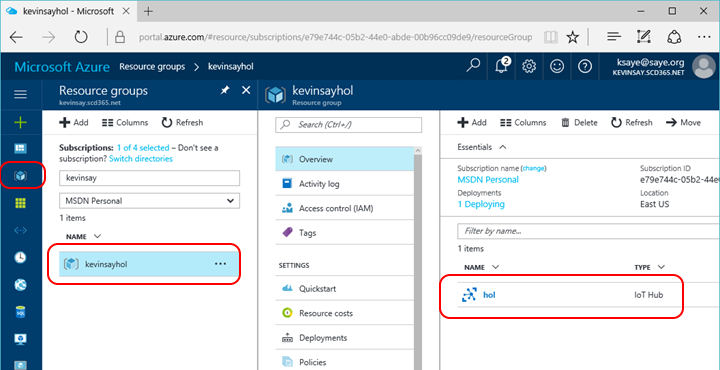
1. Select the IoT Hub by Microsoft and click Create.



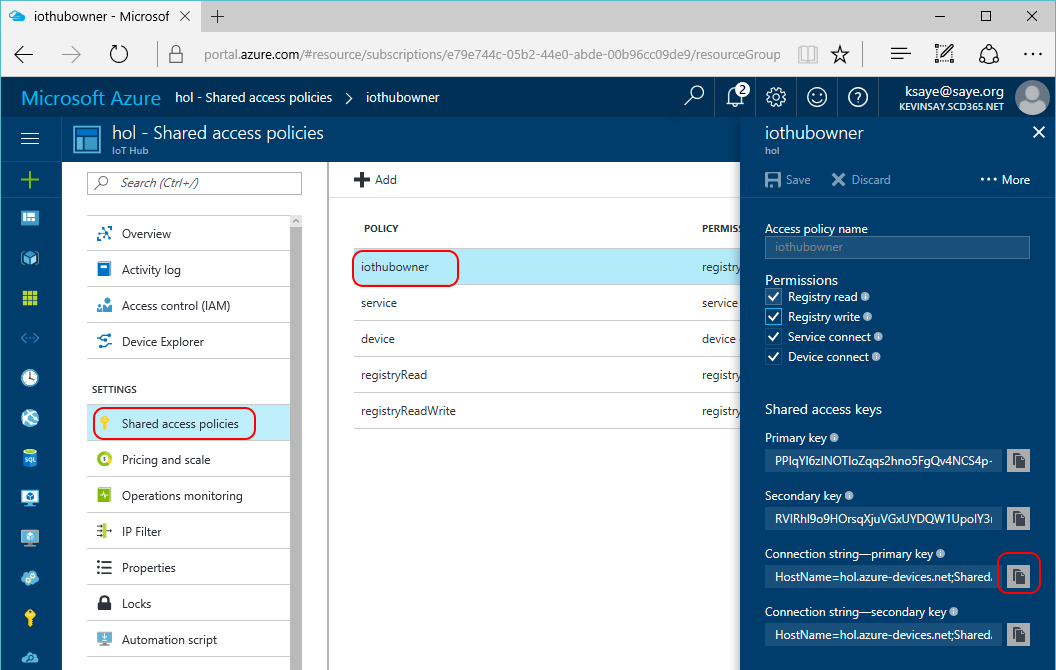
1. Give your hub a unique name, select the correct Azure Subscription, create a new (or use an existing Resource Group) and click create.



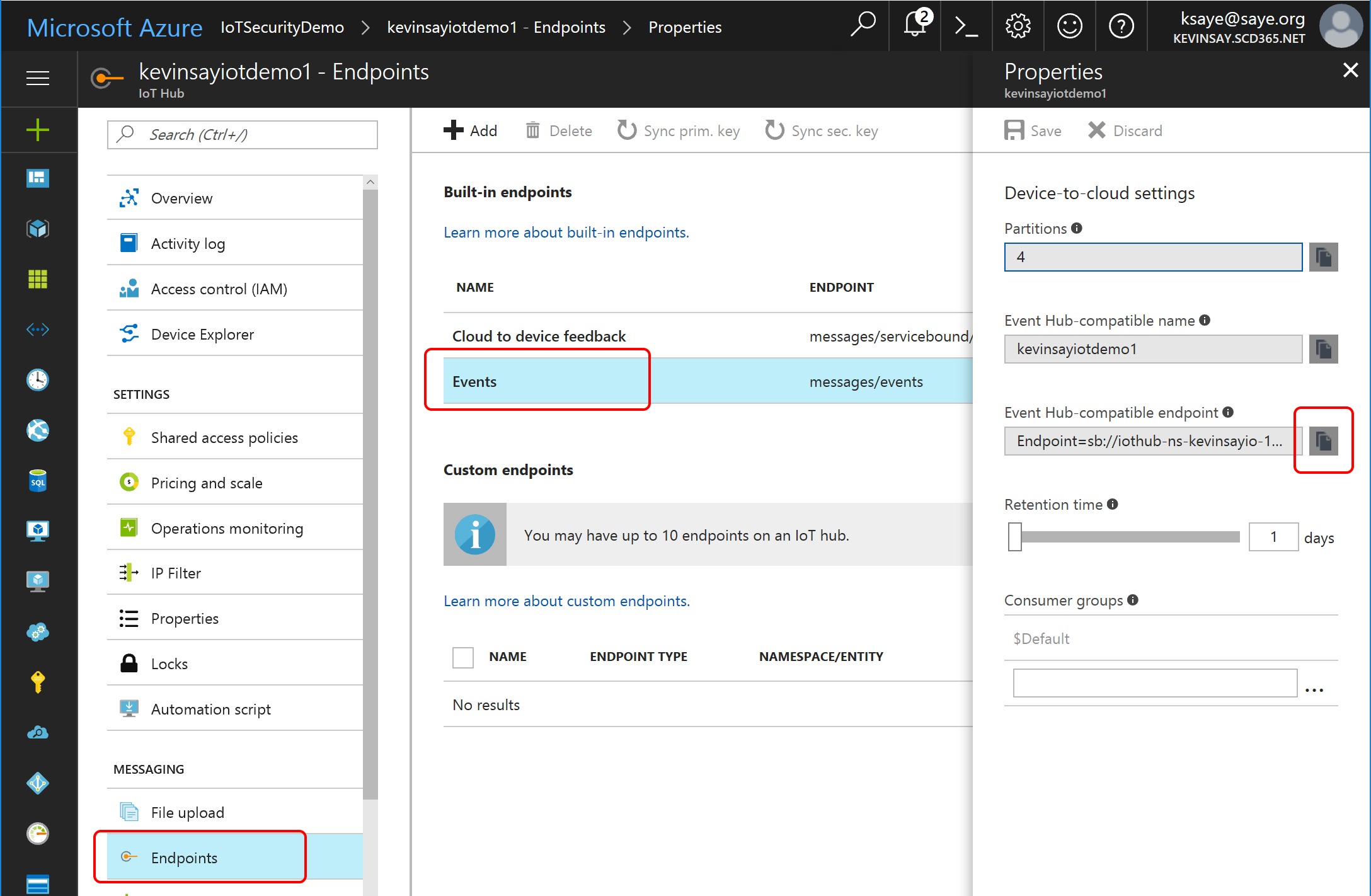
1. Click the Resource Group Icon on the left, select the resource group you just created and click on your IoT Hub just created.



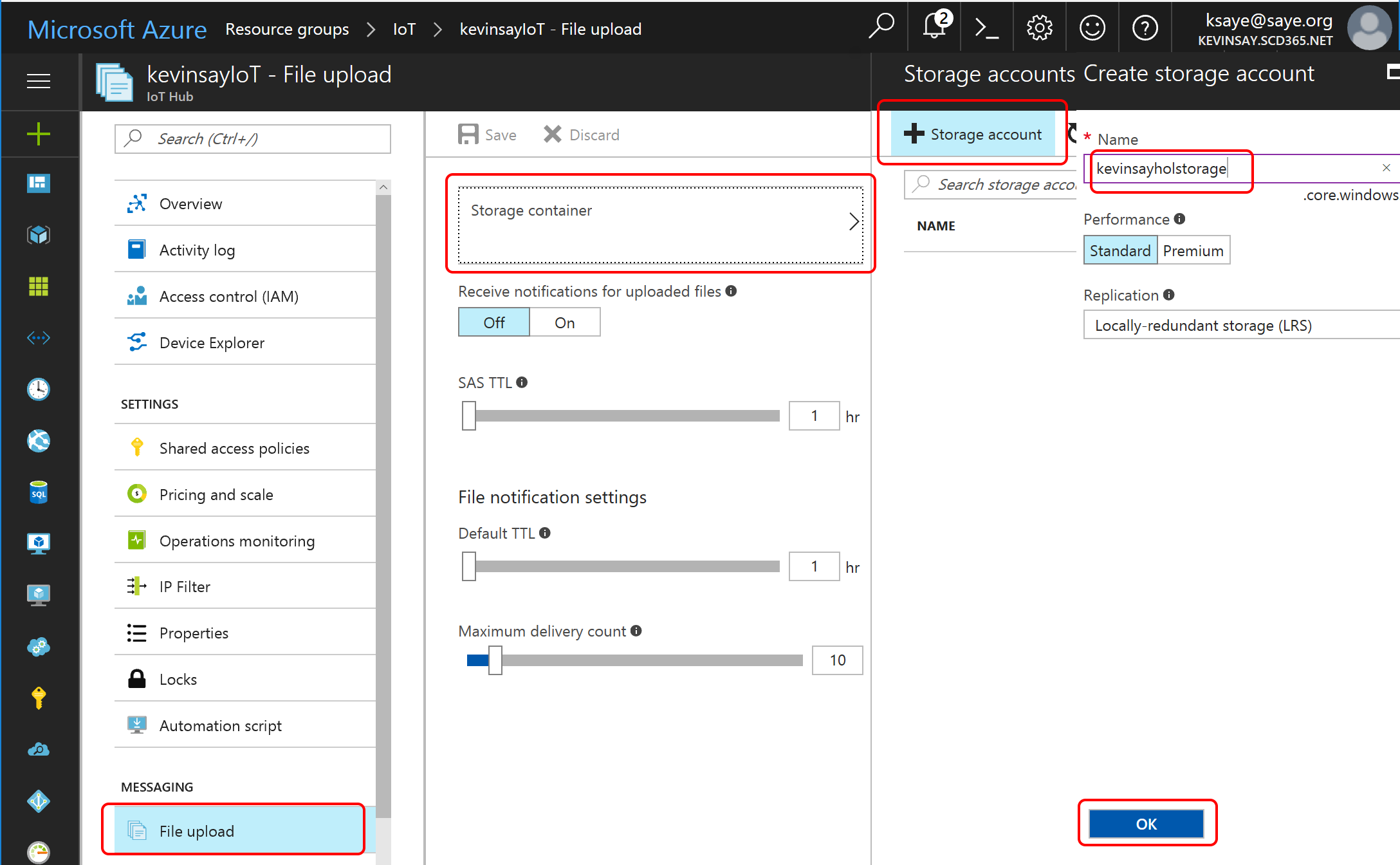
1. Once the Hub is crated (about 2 minutes), click on the Shared Access Policies and then the iothubowner and then the copy icon to copy the connection string to the clipboard and save in a note file for later reference.



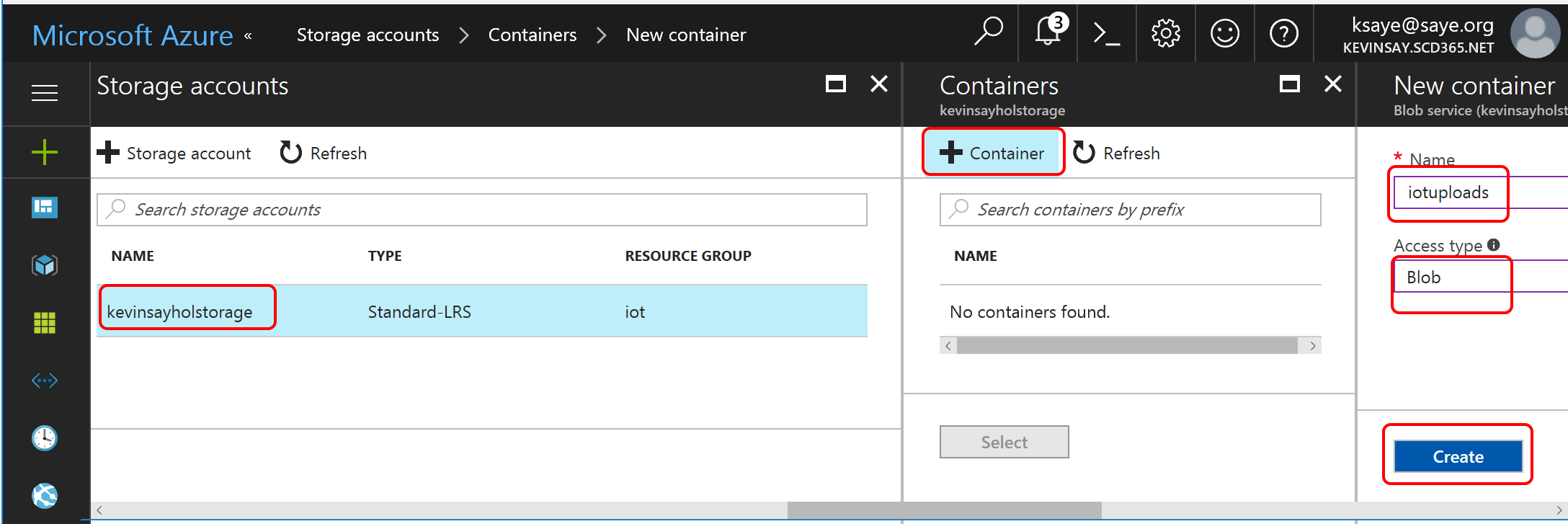
1. Select the Endpoints 🡪 events. Copy the Event Hub Compatible Connections String and save it to your note file for use in the Azure Function.



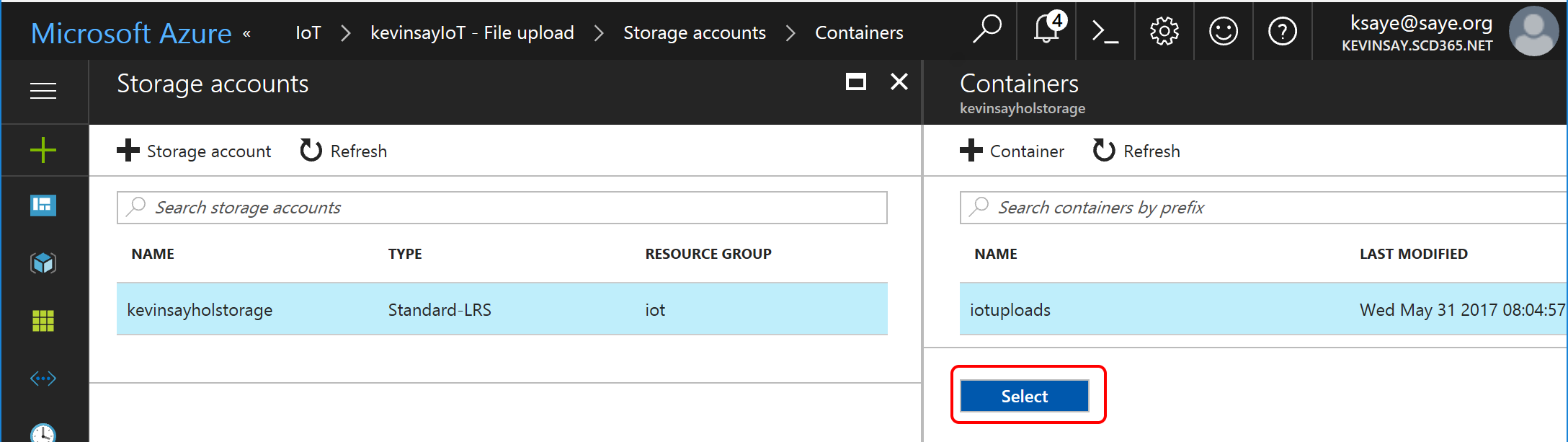
1. Add an Upload Azure Storage Account. Click File Upload 🡪 Storage Container 🡪 Storage Account. Name the Storage Account and click ok.



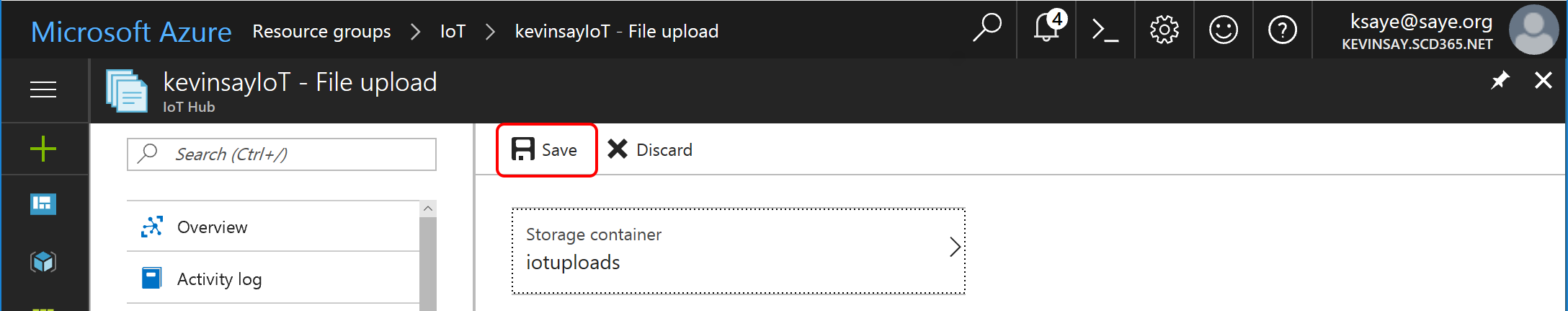
1. Once the Storage Account is created, Select the storage Account Click + Container and name the container. Select the Access Type to Blob and click Create.



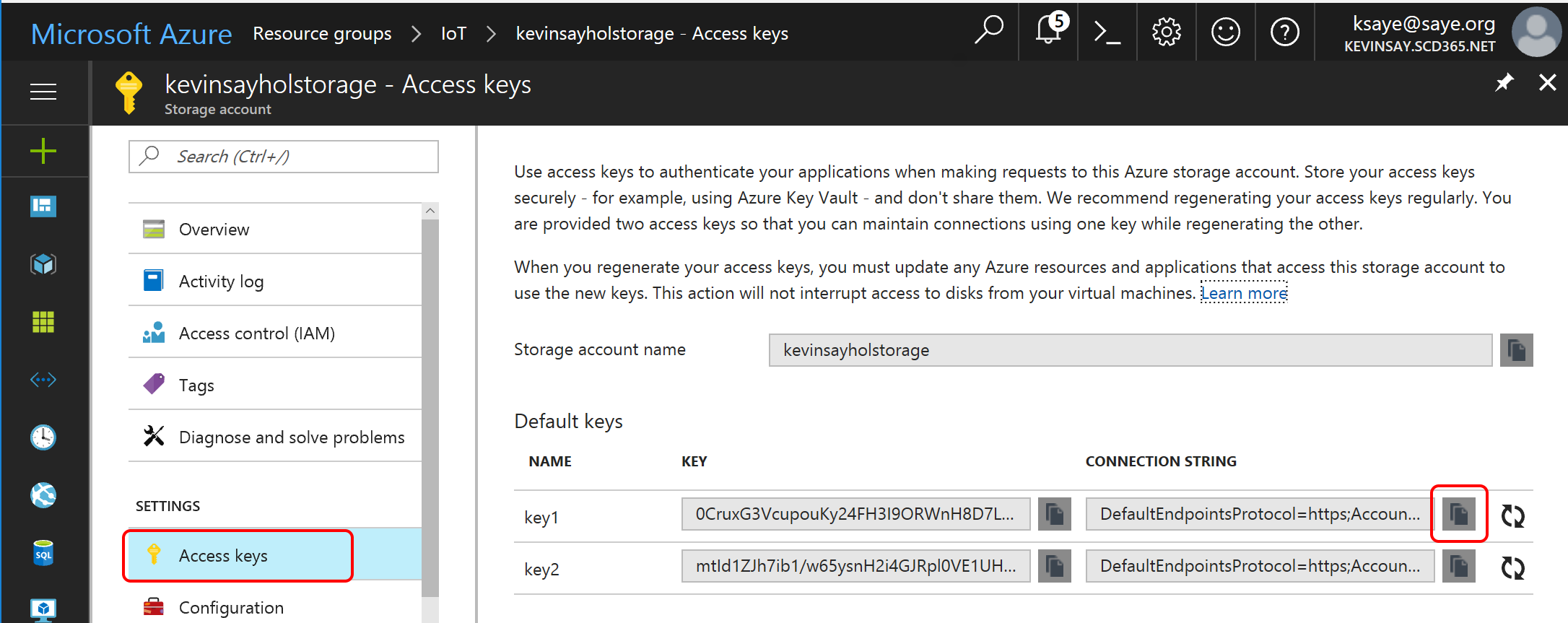
1. Once the Container is created, Select the storage Account Click + Container and name the container. Select the Access Type to Blob and click Create.



1. Finally, click Save to apply the new container in the new storage account.

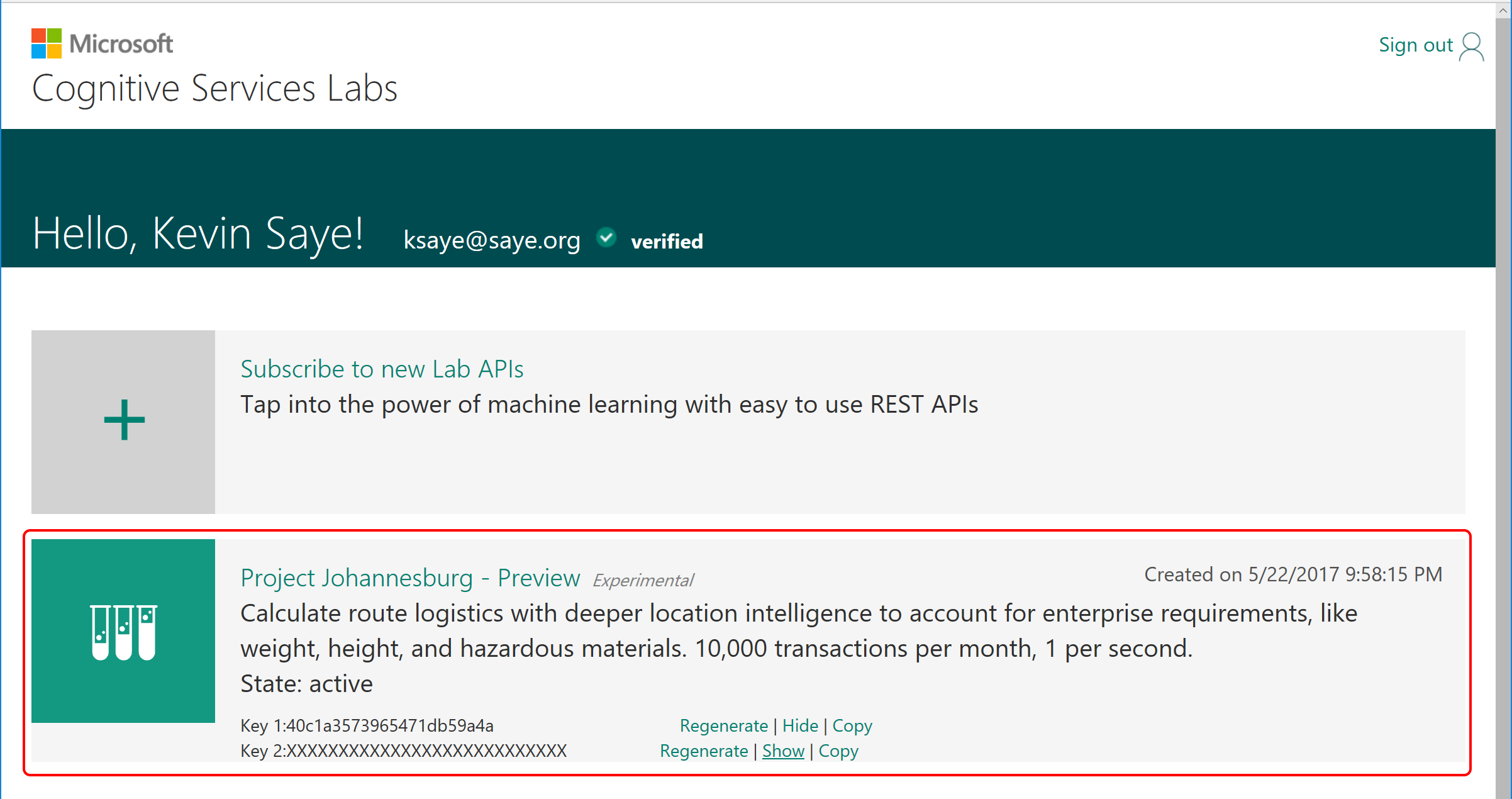


1. Next, we need the connection string of the new storage container. Select the new Storage Account you created in the resource group. Select Access Keys and copy the copy the connection string to save in your note file.



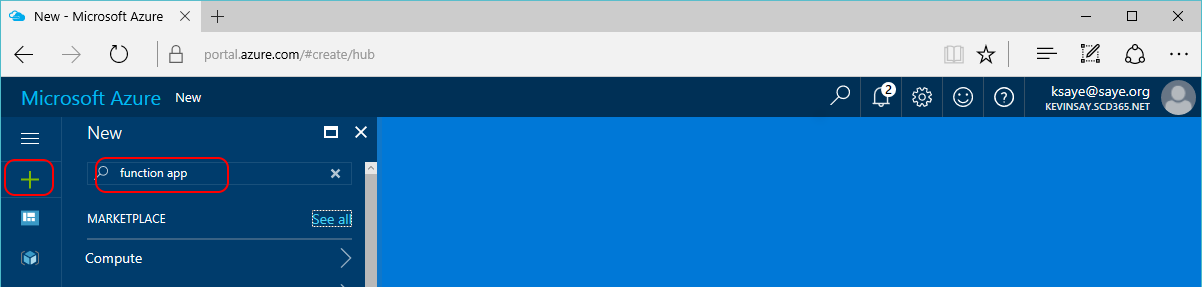
# Setting up the Project “Johannesburg” Cognitive Service

1. Go to <https://labs.cognitive.microsoft.com/en-us/subscriptions> and signup for Project Johannesburg Cognitive Service. You will have to verify an email before you can continue.
2. Request a Project “Johannesburg” API Key, save this to your note file for use in the Azure Function setup.

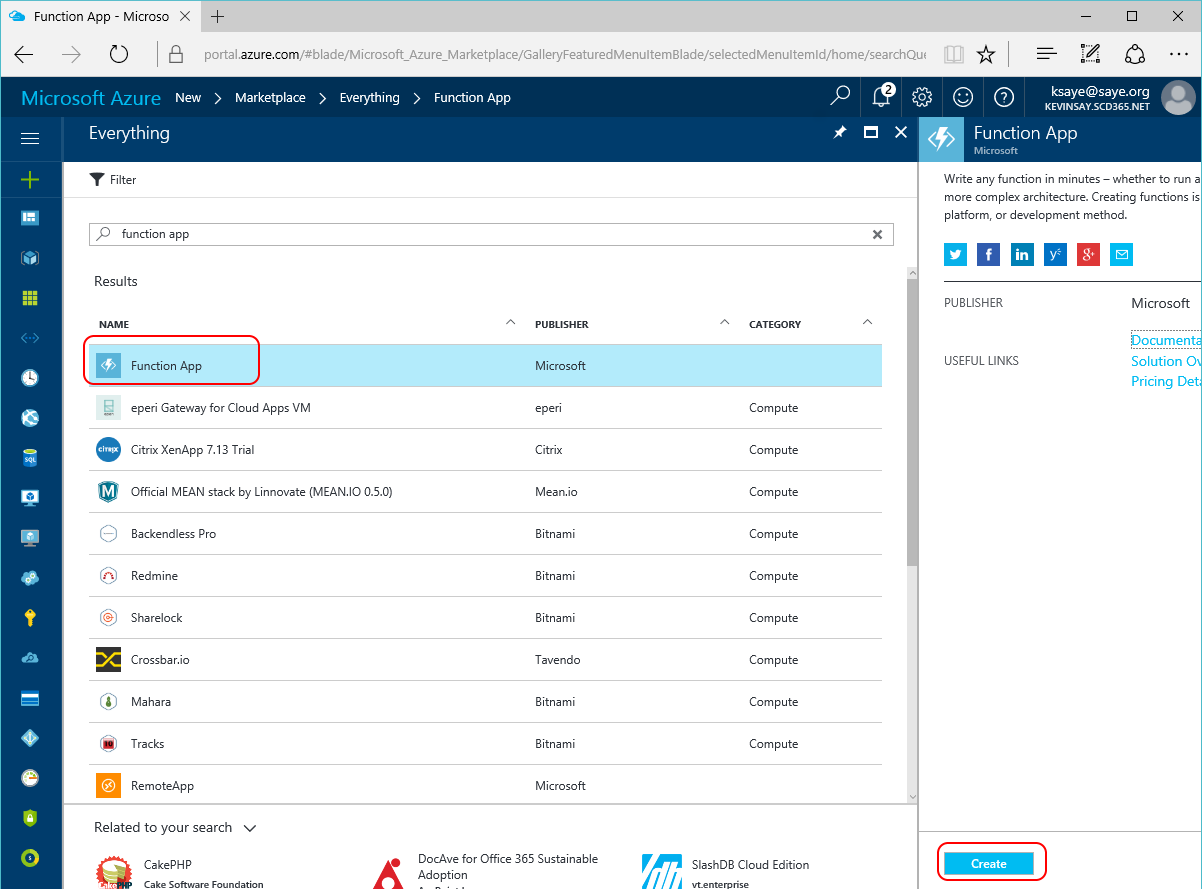


# Setting up the Azure Function

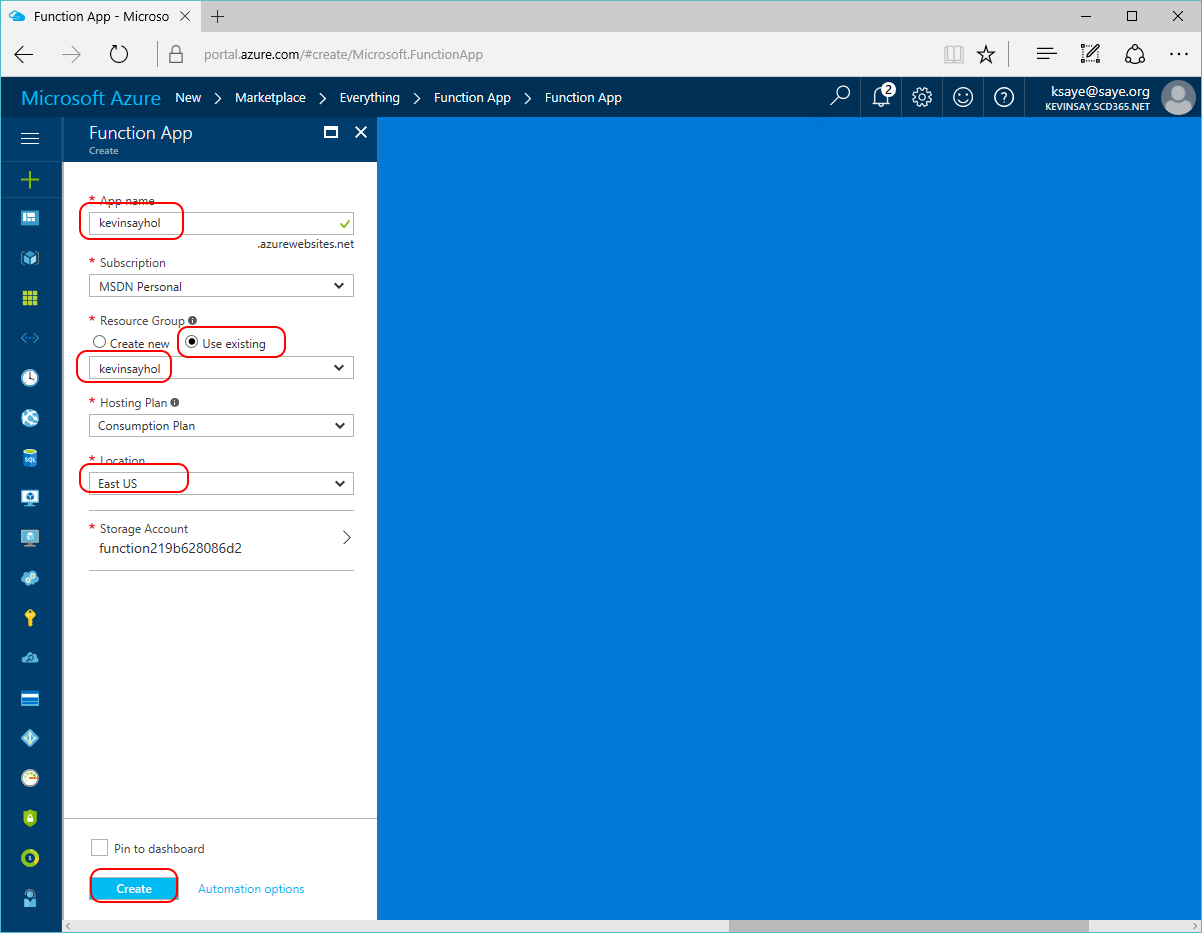
1. In the Azure Portal, click the Plus sign on the left and search for Function App.



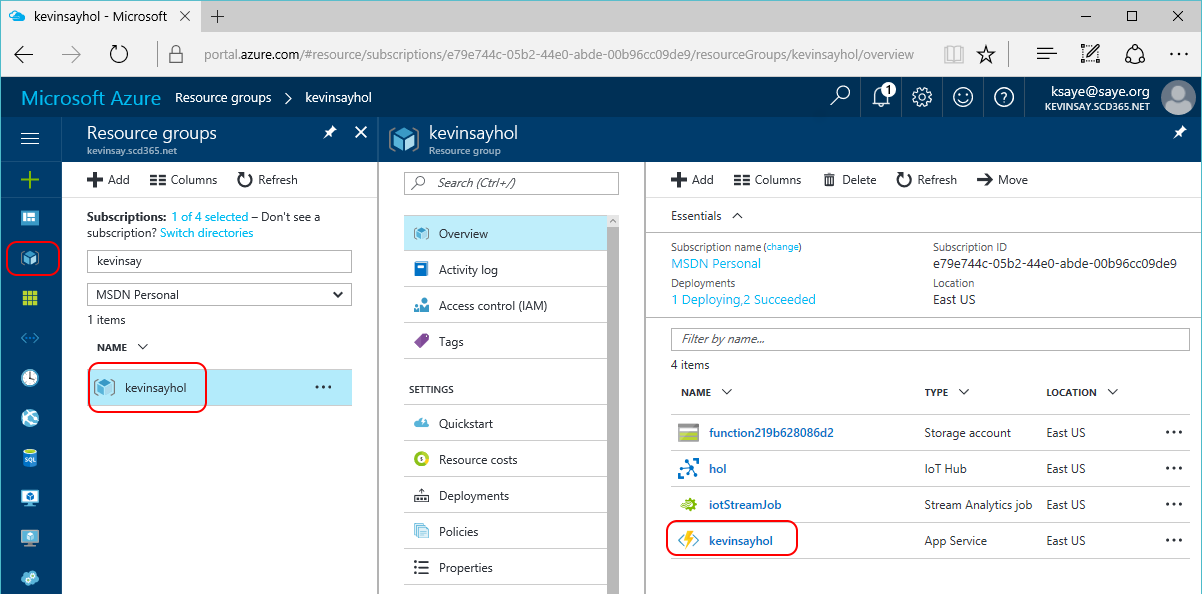
1. Select the Function App by Microsoft and click Create.



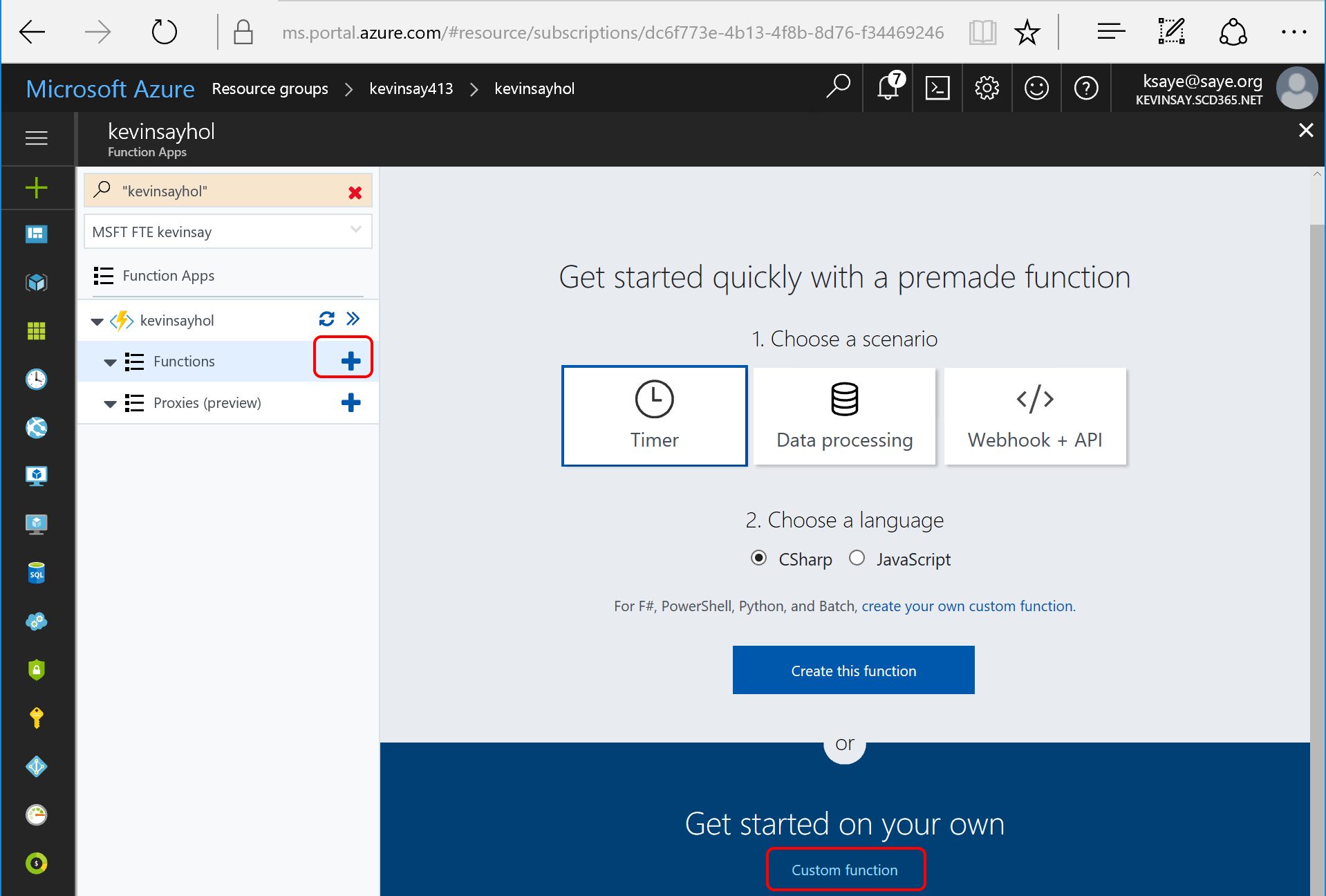
1. Give your function a unique name, select the correct Azure Subscription, select the Resource Group created in the prior lab, make sure it is in the desired location and click create.



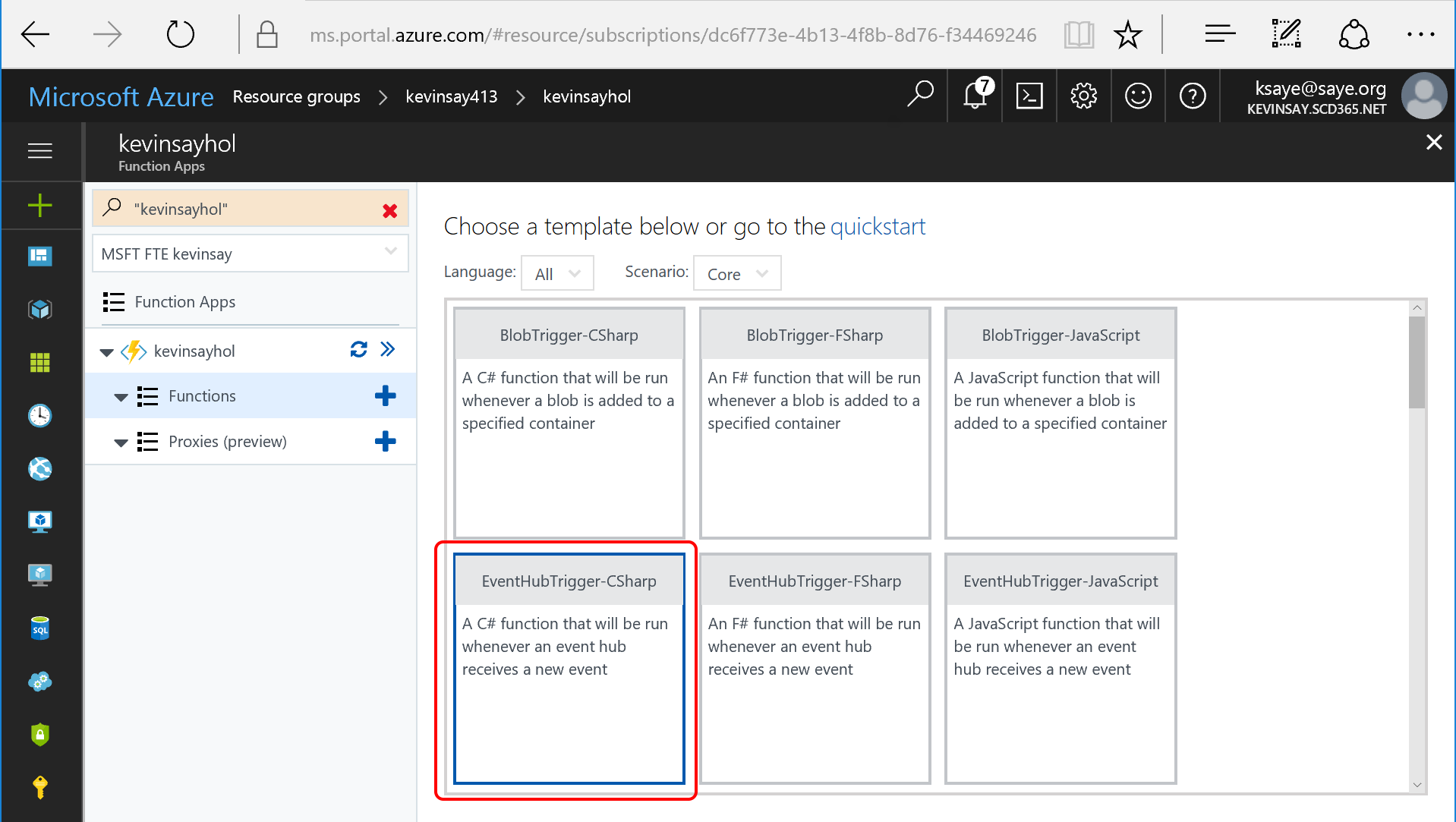
1. After about a minute, click the Resource Group Icon on the left, select the resource group you created and click on the Function just created.



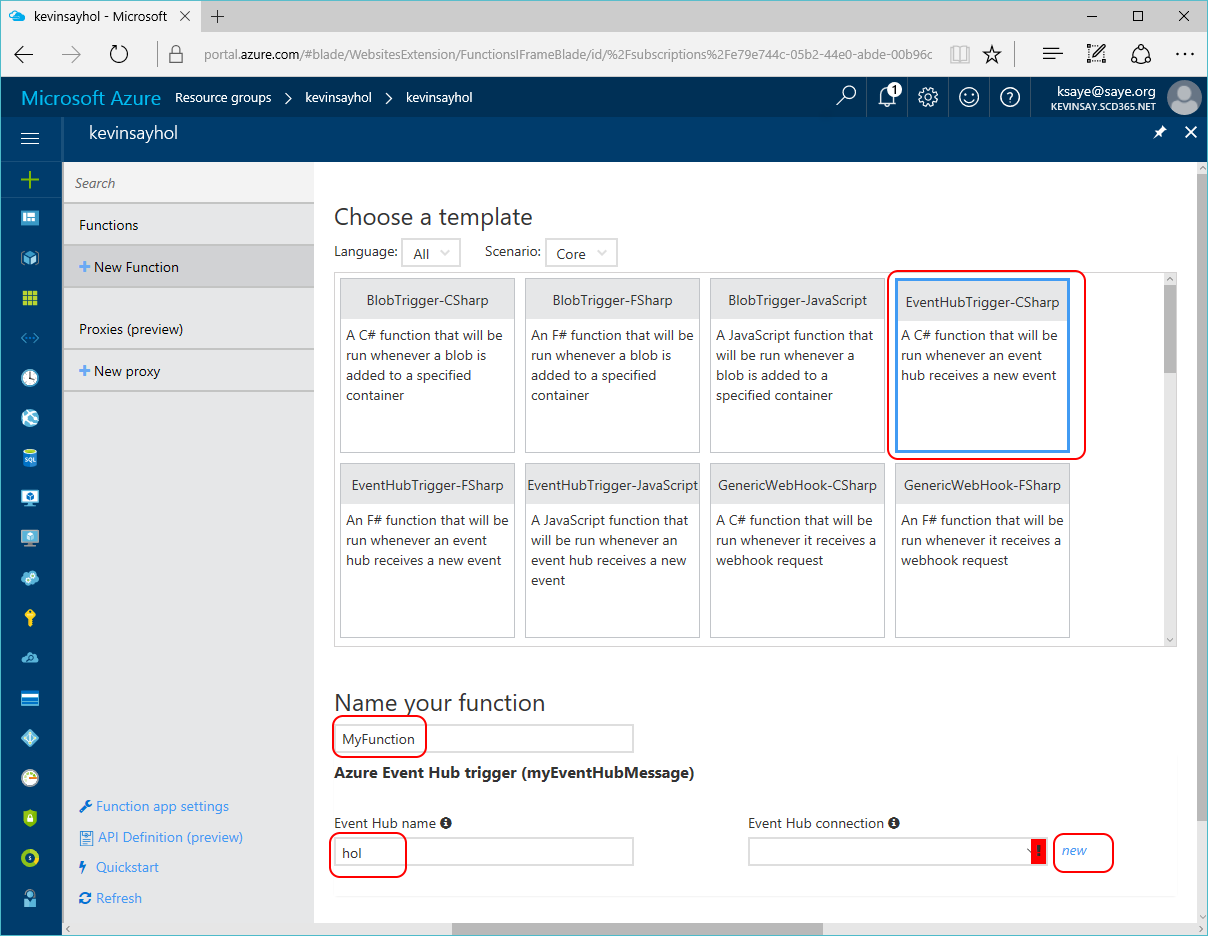
1. Under your function name, click the + sign and click on Custom function.



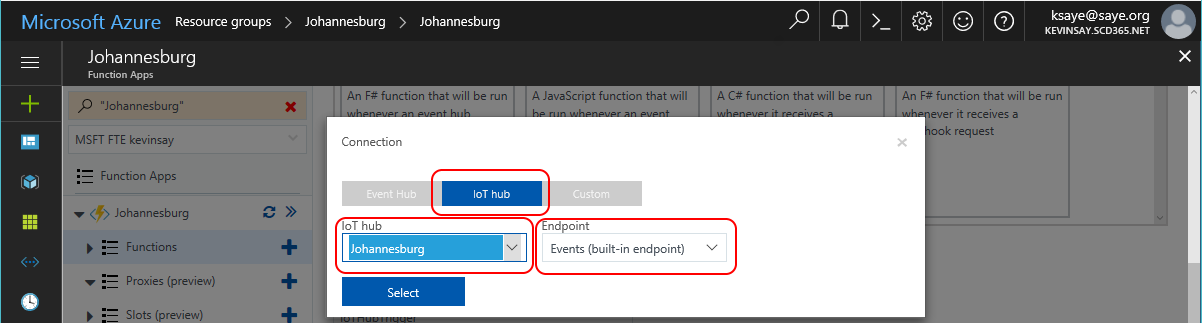
1. Click the “EventHubTrigger-CSharp” template.



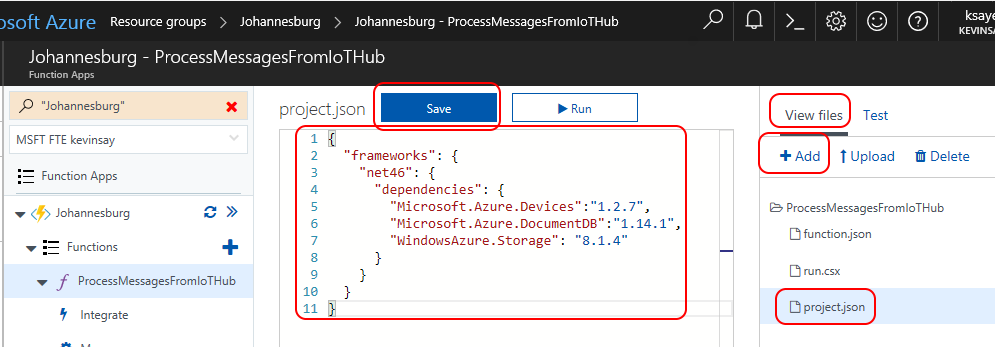
1. Name your function and type in the name of the IoT Hub in the event Hub Name area. Next click new on the Event Hub Connection.



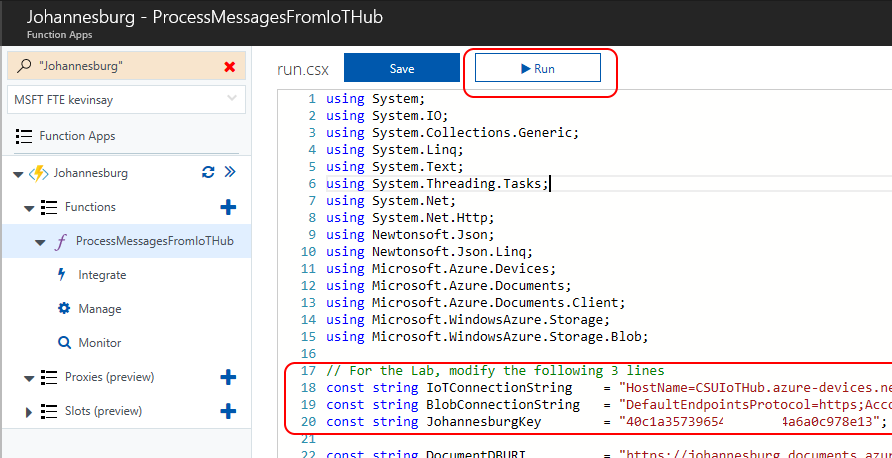
1. At the Connection screen, select IoT Hub and select the name of your hub and the Events endpoint, as shown below:

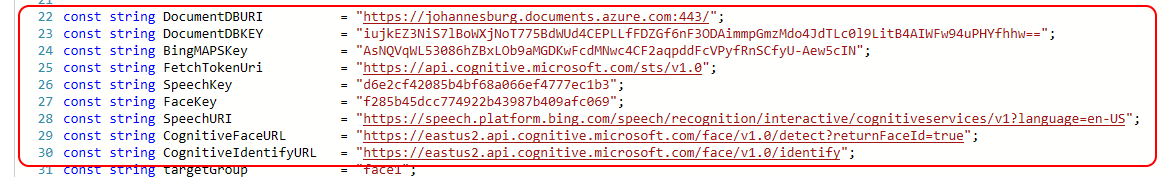


1. On the right side of the Function, click View Files and then add. Name the file project.json and type the following content in to load all the needed libraries.



1. Copy the code from: <https://github.com/ksaye/IoTDemonstrations/blob/master/Johannesburg/run.csx> and past it in the Azure Function, replacing all the C# code. Modify lines: 18, 19 and 20, inserting values from your note file and save the run.csx file. Modify lies 22 – 30 with the values provided by the instructor. Then click Run to compile and start the function.



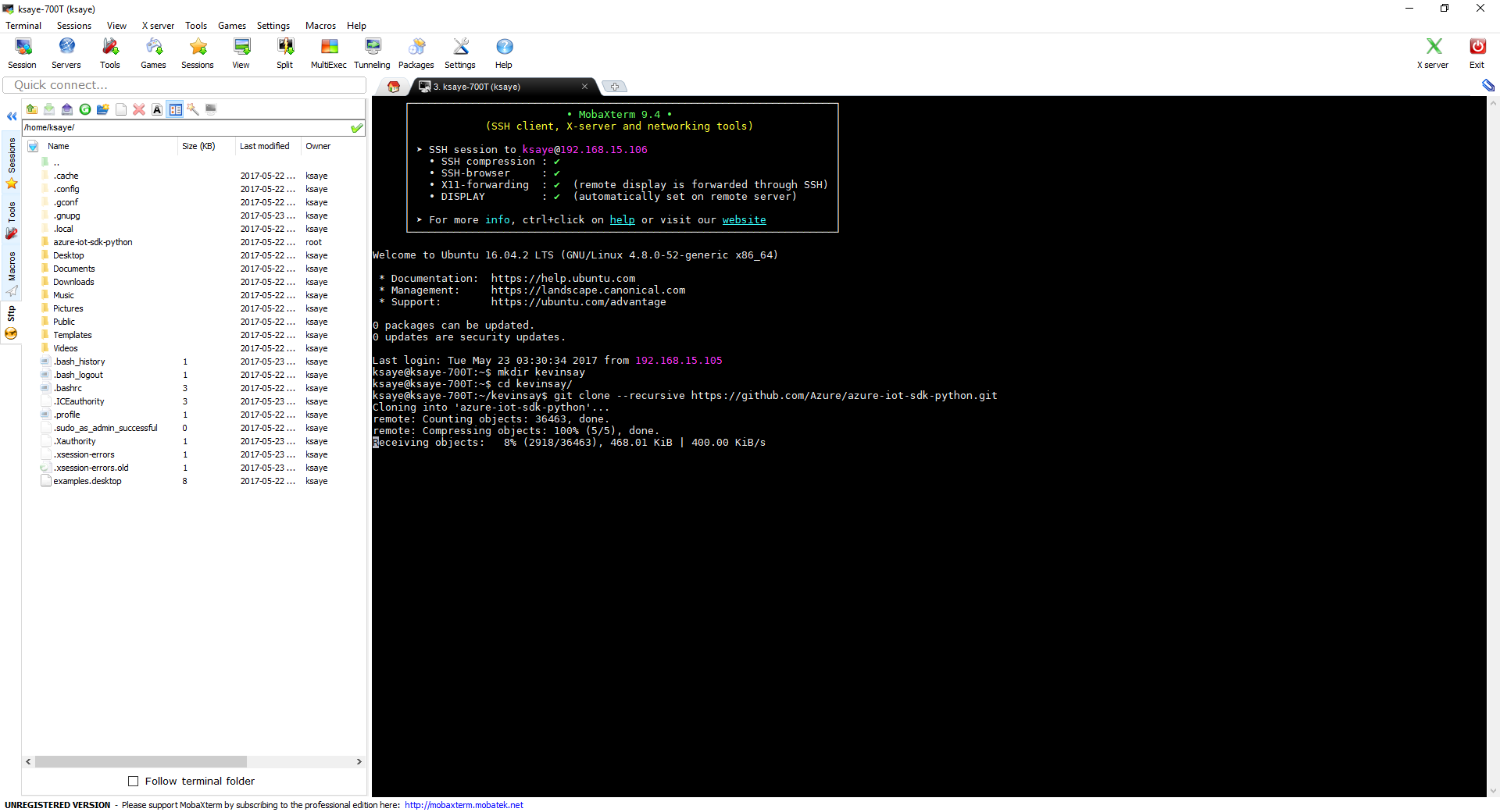


# Setting up the Device

1. Log into your Linux host as a standard user, using either Telnet or SSH. The instructor will provide the hostname, username and password for the host. Alternatively, you can use either Raspberry PI or Ubuntu.
2. Type the following commands to make a directory based on your name and change to that directory:

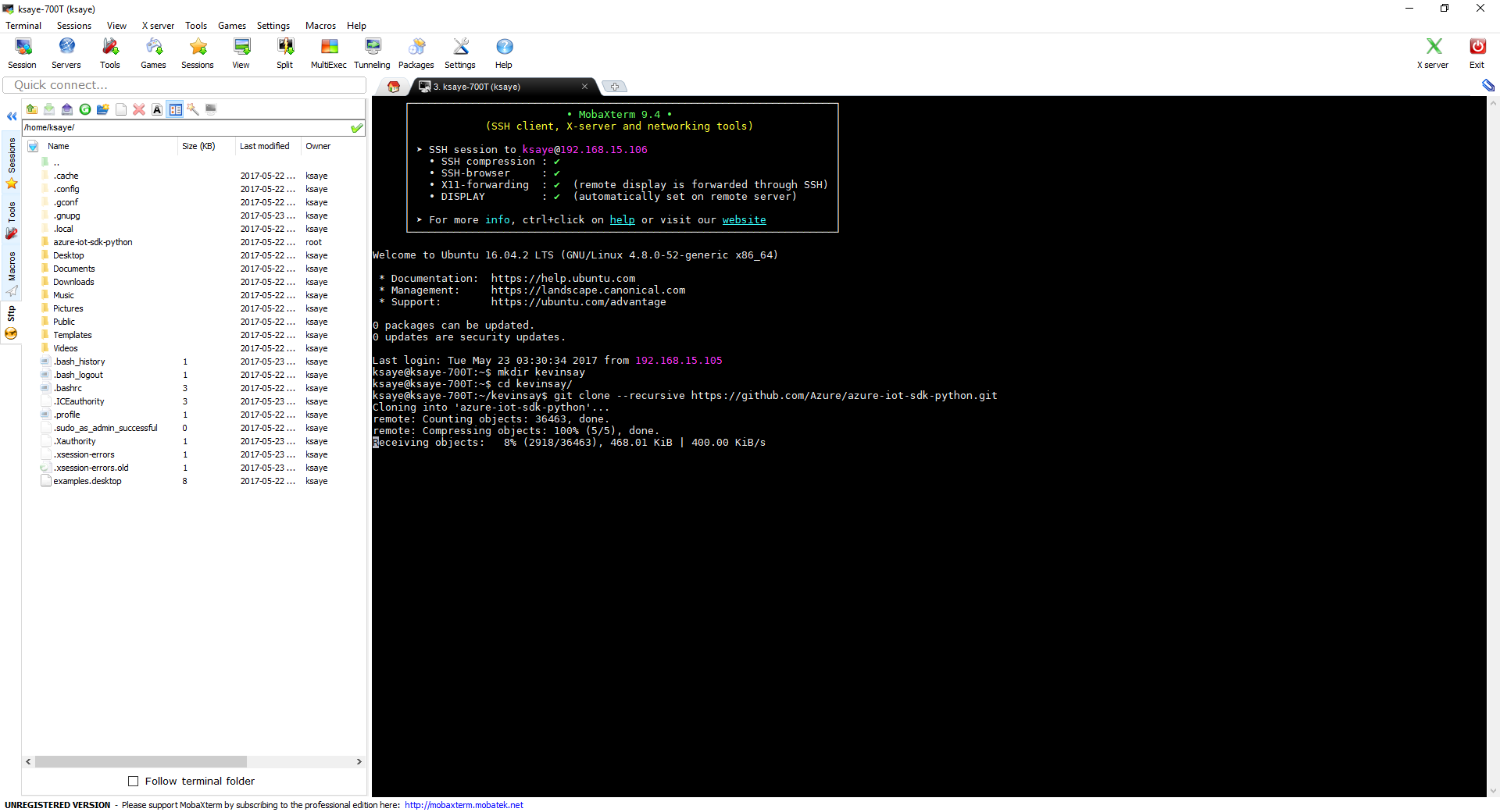
mkdir kevinsay

cd kevinsay



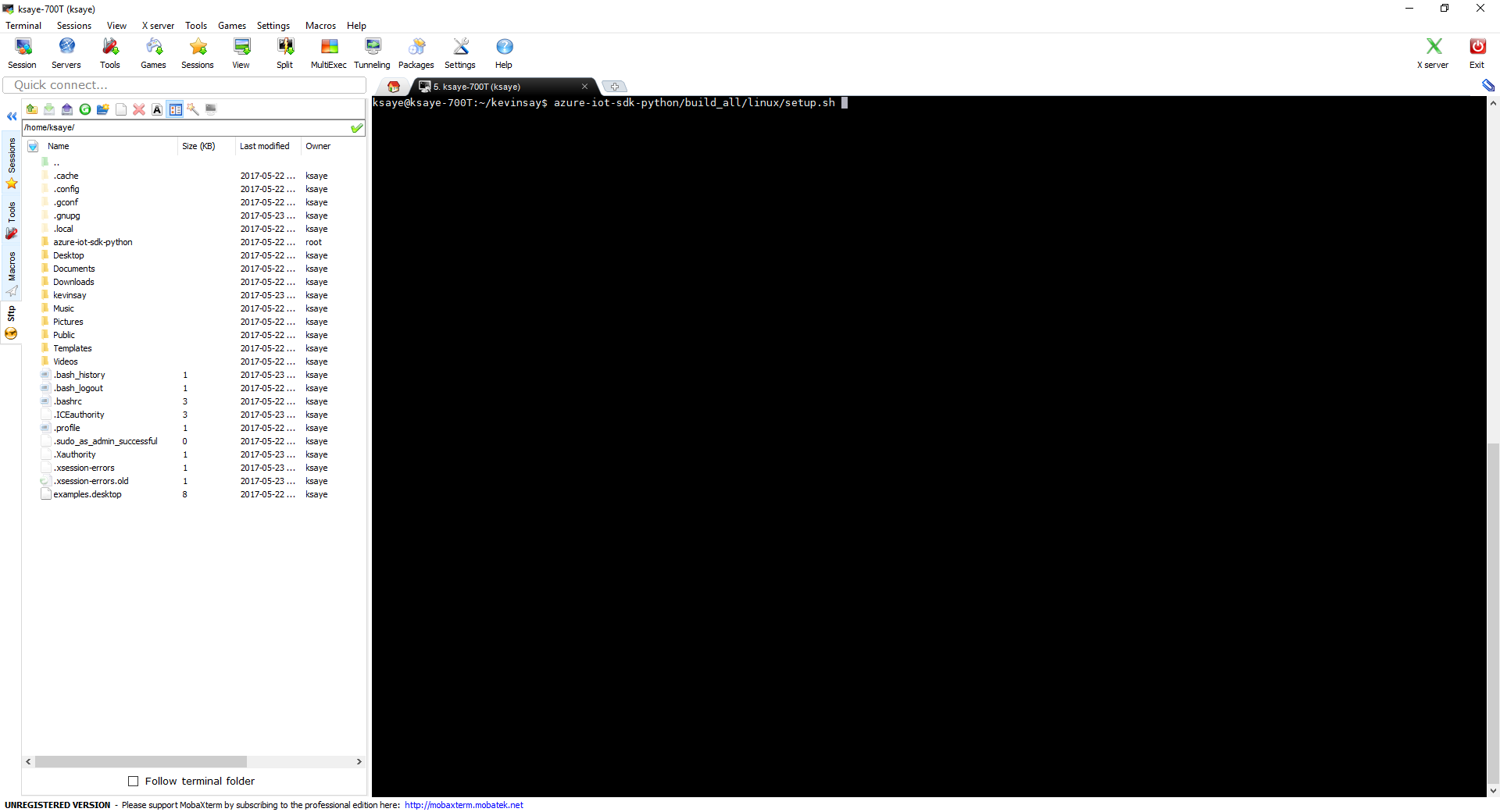
1. Type the following command, which will clone the source for the Azure IoT SDK for Python:

git clone --recursive https://github.com/Azure/azure-iot-sdk-python.git



1. Type the following command, which will setup any pre-requisites. If you are not root, sudo will ask you to verify your password:

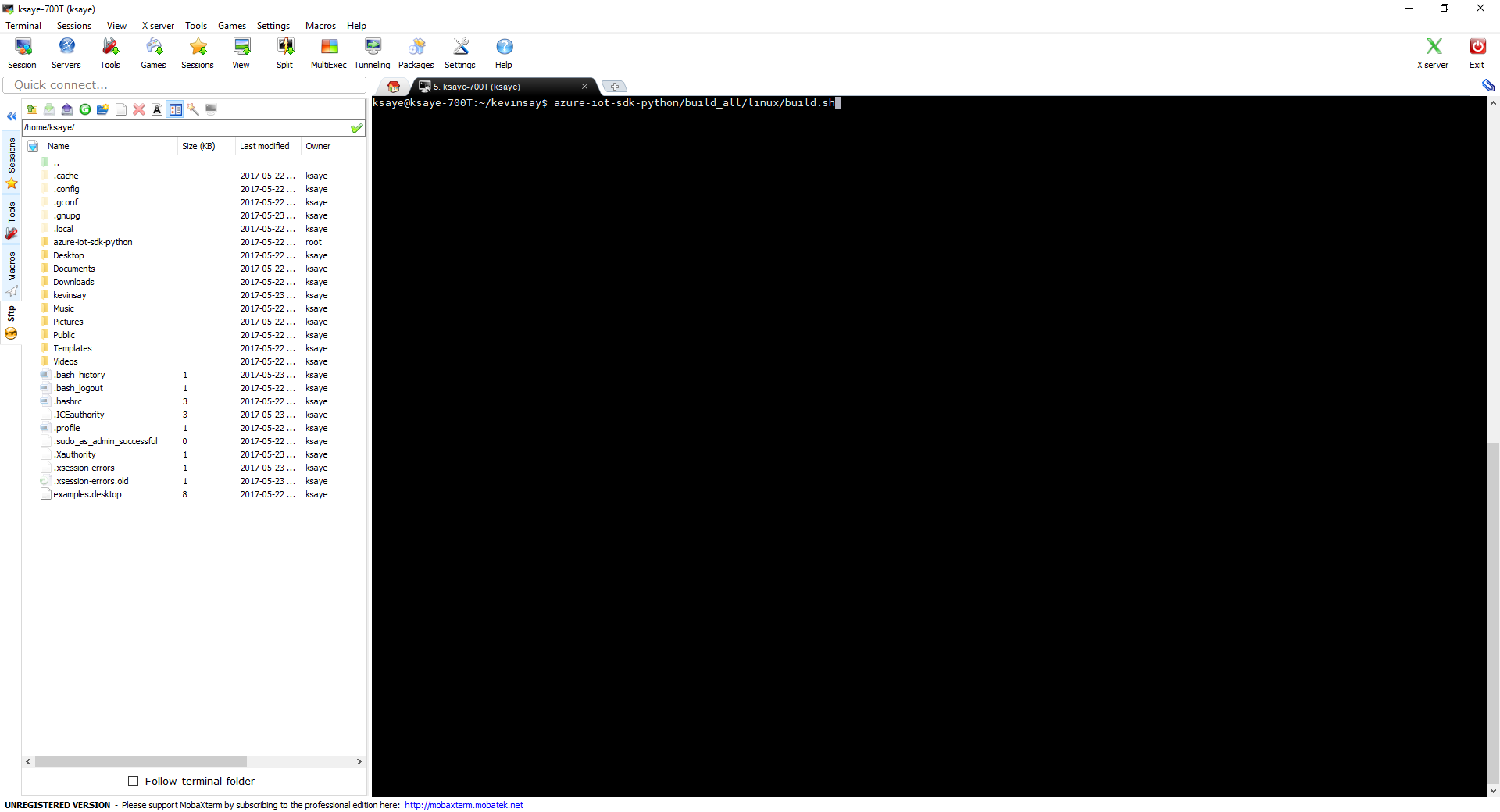
azure-iot-sdk-python/build\_all/linux/setup.sh



*Note, if this environment has already been setup, root or sudo access is not required.*

1. Type the following command, which will build the python library:

azure-iot-sdk-python/build\_all/linux/build.sh



1. Still in your username directory, make a new directory called truckdemo and copy the python library there.

mkdir truckdemo

cp azure-iot-sdk-python/device/samples/iothub\_client.so truckdemo/



1. Type the following command to change to the truckdemo directory, download the sample code and make it executable

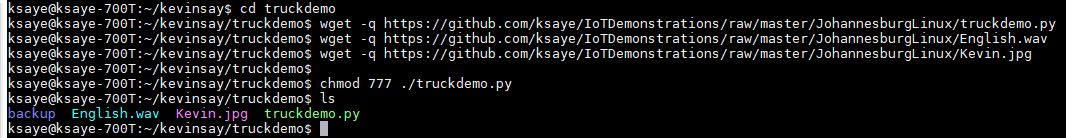
cd truckdemo

wget -q https://github.com/ksaye/IoTDemonstrations/raw/master/JohannesburgLinux/truckdemo.py

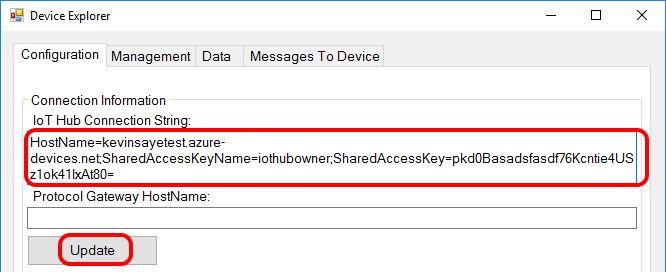
wget -q https://github.com/ksaye/IoTDemonstrations/raw/master/JohannesburgLinux/English.wav

wget -q https://github.com/ksaye/IoTDemonstrations/raw/master/JohannesburgLinux/Kevin.jpg

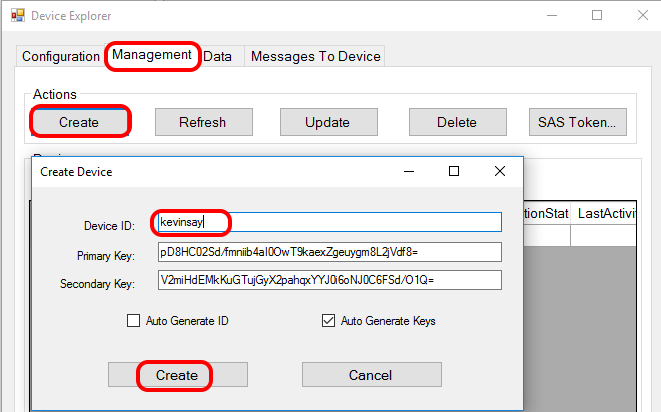
chmod 777 ./truckdemo.py



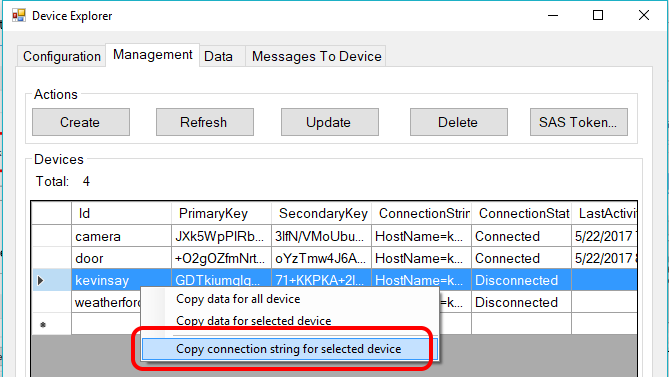
1. Download and Install Device Explorer from: <https://github.com/Azure/azure-iot-sdks/releases>
2. Using the IoT Hub connection string in your note file from the earlier steps, past it in Device Explorer and click Update.



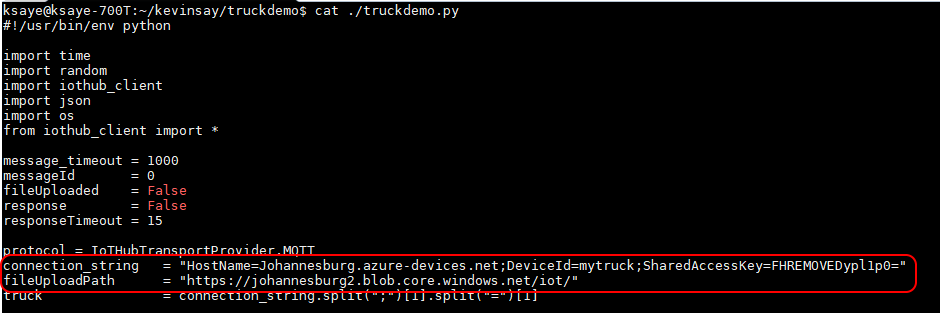
Click the Management Tab and Create. Then type in your truck name (your alias) and click Create.



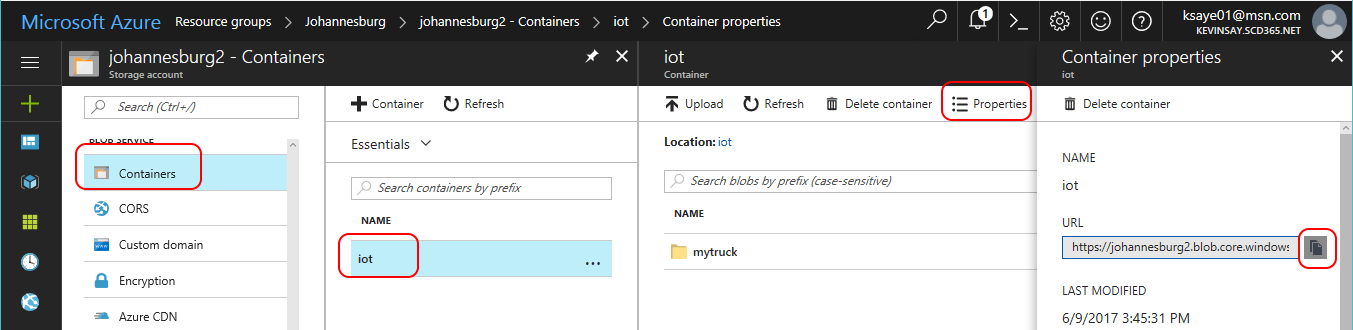
Next, select your device, right click and copy the device connection string and save it to be used for our device.



1. Modify the connection string setting and the fileUploadPath URL in the truckdemo.py file. Use the connection string from the step above, and make sure the fileUpload URL matches the storage account and container that IoT Hub is uploading to.



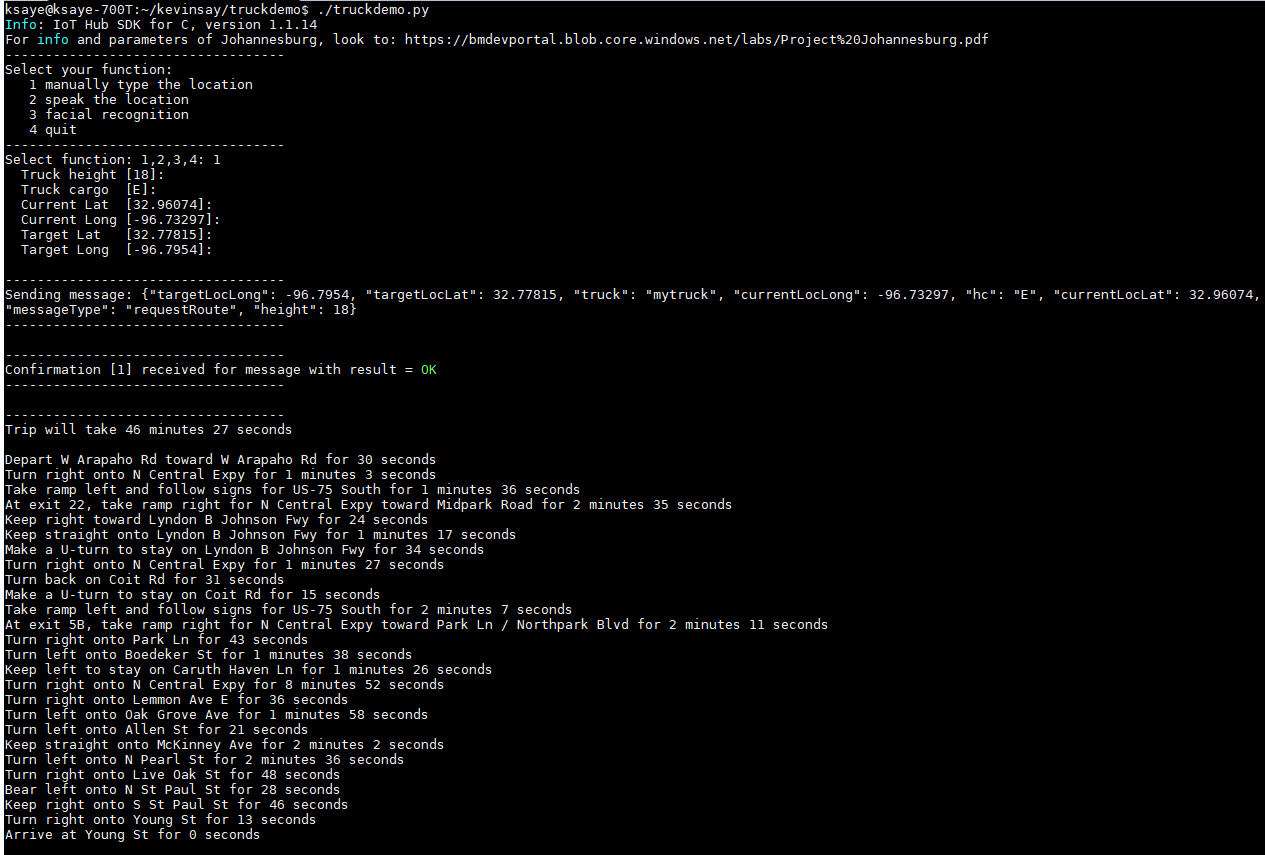
You can always get the URL of the container by selecting your storage account 🡪 Containers 🡪 {select the container} 🡪 Properties and click the copy icon next to the URL.



# Driving the Truck – Manual

Here we will manually type the GPS coordinates of where we want to go and that our truck is large and has explosive chemicals.

1. Start the python script by typing ./truckdemo.py
2. Type 1 and just accept (or change the default) as shown below

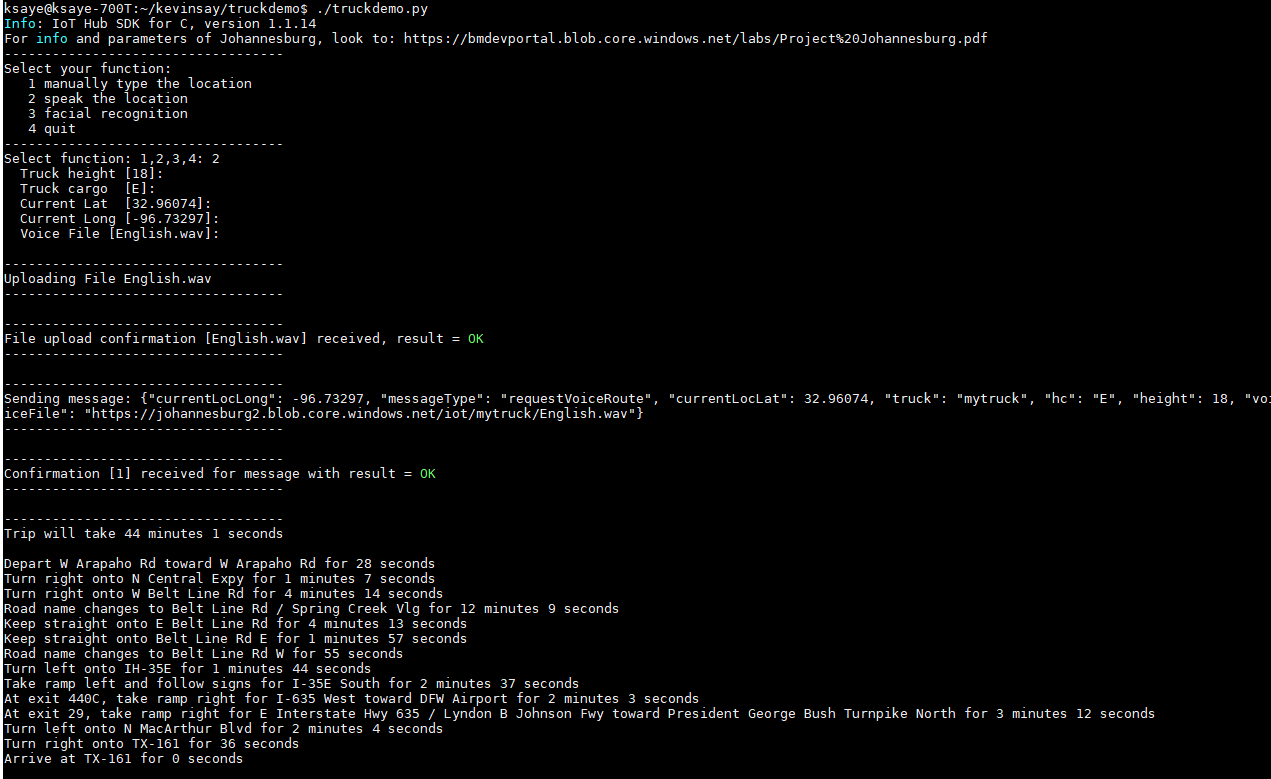


# Driving the Truck – Voice

Here we will just tell the truck (upload a wave file) telling the truck where we want to go. Azure will use Cognitive Services to convert speech to text, then use Bing Maps to calculate the destination latitude / longitude and finally use Project “Johannesburg” to calculate the route.

*Note, you can upload your own wave file to test other locations*

1. Start the python script by typing ./truckdemo.py
2. Type 2 and just accept (or change the default) as shown below

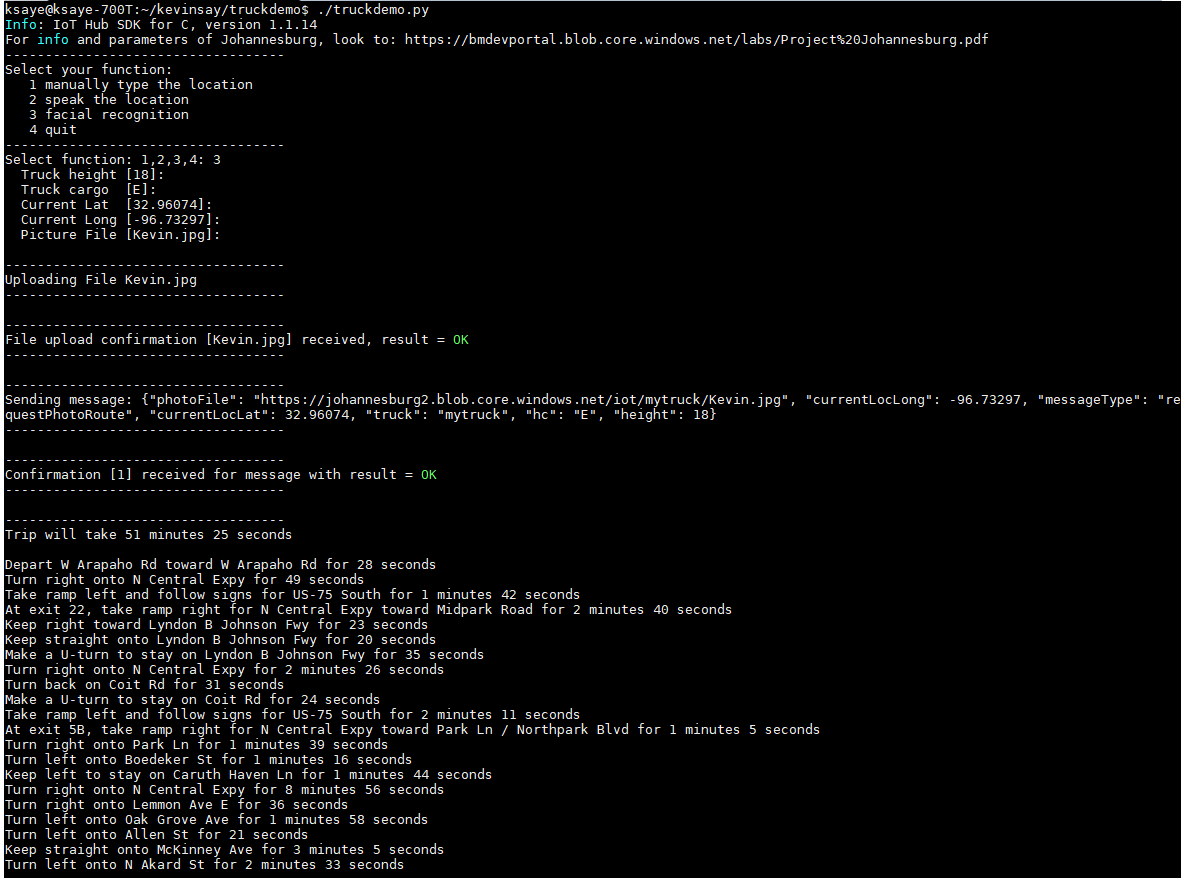


# Driving the Truck – Picture

Here we the truck will perform facial recognition (upload a jpeg picture) to calculate where we want to go. Azure will use Cognitive Services to verify the person, then use lookup the person in the route database to calculate the destination latitude / longitude and finally use Project “Johannesburg” to calculate the route.

*Note, you can upload other pictures to see if you can trick the Face Cognitive Service.*

1. Start the python script by typing ./truckdemo.py
2. Type 3 and just accept (or change the default) as shown below



Note, the “Route” Cosmos DB only contains one entry, but it derives the lat long off of the personID.

