

Workshop: Connecting IoT devices to the cloud with Azure IoT Hub and mbed Cloud

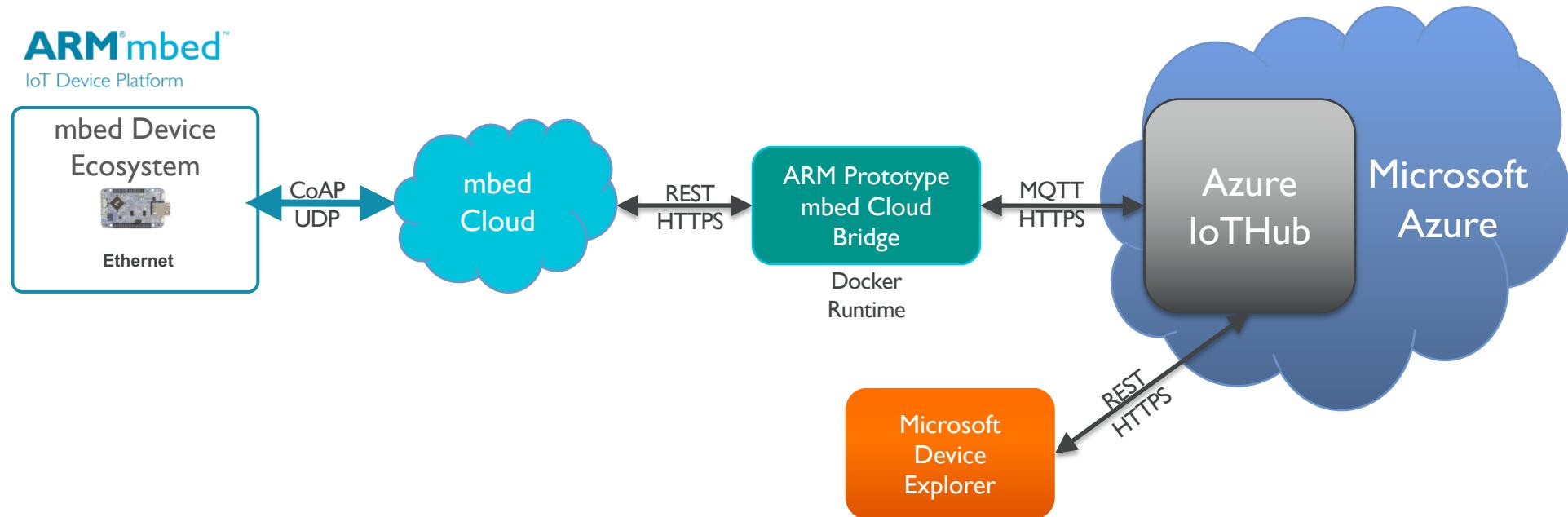


Doug Anson
Solutions Architect / IoT BU / ARM

Brian Daniels
Applications Engineer / IoT BU / ARM

August 22, 2018 - v2.5

What will we build in this Workshop?



- Connect your mbed device into Azure IoTHub through mbed Cloud and ARMs Prototype IoTHub Cloud Bridge
 - Create a simple mbed device and connect it to mbed Cloud
 - Create an instance of ARM's Prototype Azure IoTHub Cloud bridge and bind it to your Azure and mbed Cloud accounts
 - Exploration of the device data telemetry using Microsoft's Device Explorer

Workshop: Let's get started!

- Create and setup your Azure account (should be completed prior to workshop)
- Install the necessary tools/drivers into your Windows PC (should be completed prior to workshop)
- Create mbed developer and mbed Cloud accounts (should be completed prior to workshop)
- Retrieve a set of provisioning credentials (mbed_cloud_dev_credentials.c)
- Import our mbed sample project into the online IDE
- Update the sample project with the provisioning credentials (mbed_cloud_dev_credentials.c)
- Compile, Install, Download, and Copy into the mbed device
- Connect a Serial Terminal (115200,8NI, proper mbed COM port chosen for Windows users...)
- Send the “Break” command to reset the mbed device
- See the device output on our Serial Terminal (PTSOOI method...)

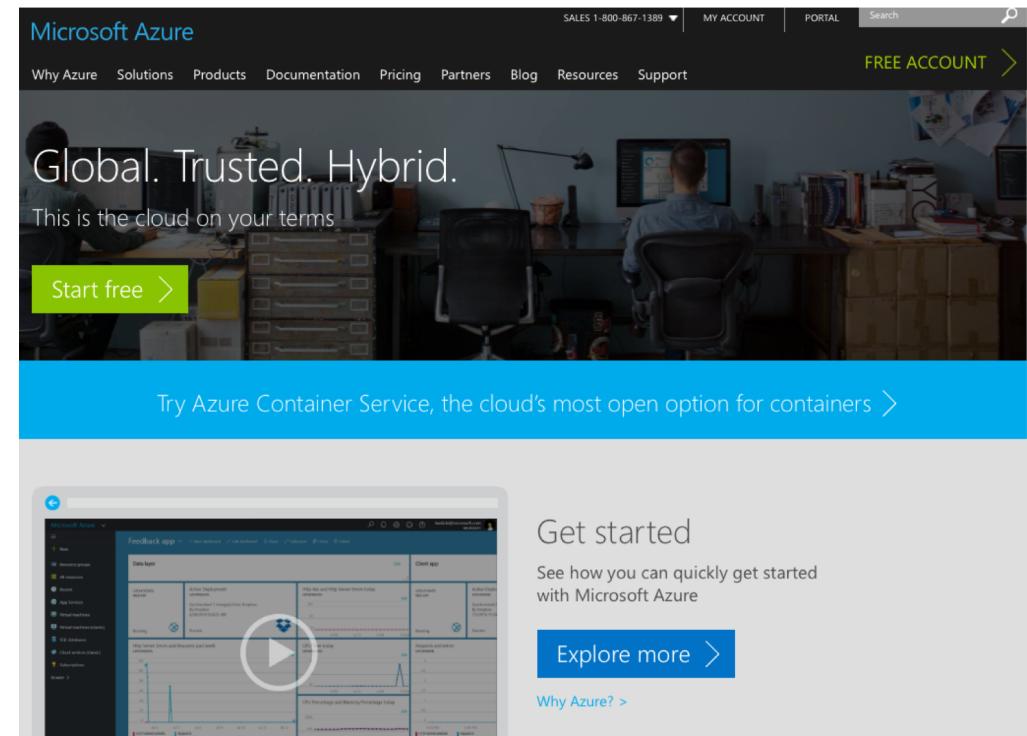
NOTE: During the workshop, be sure to double-check your copy/paste operations...

Quick Links: Visit https://github.com/ARMmbed/bridge_workshop_ms_iothub_cloud

- README.md has all of the links in the workshop + this presentation in PDF format

Create our Azure Account

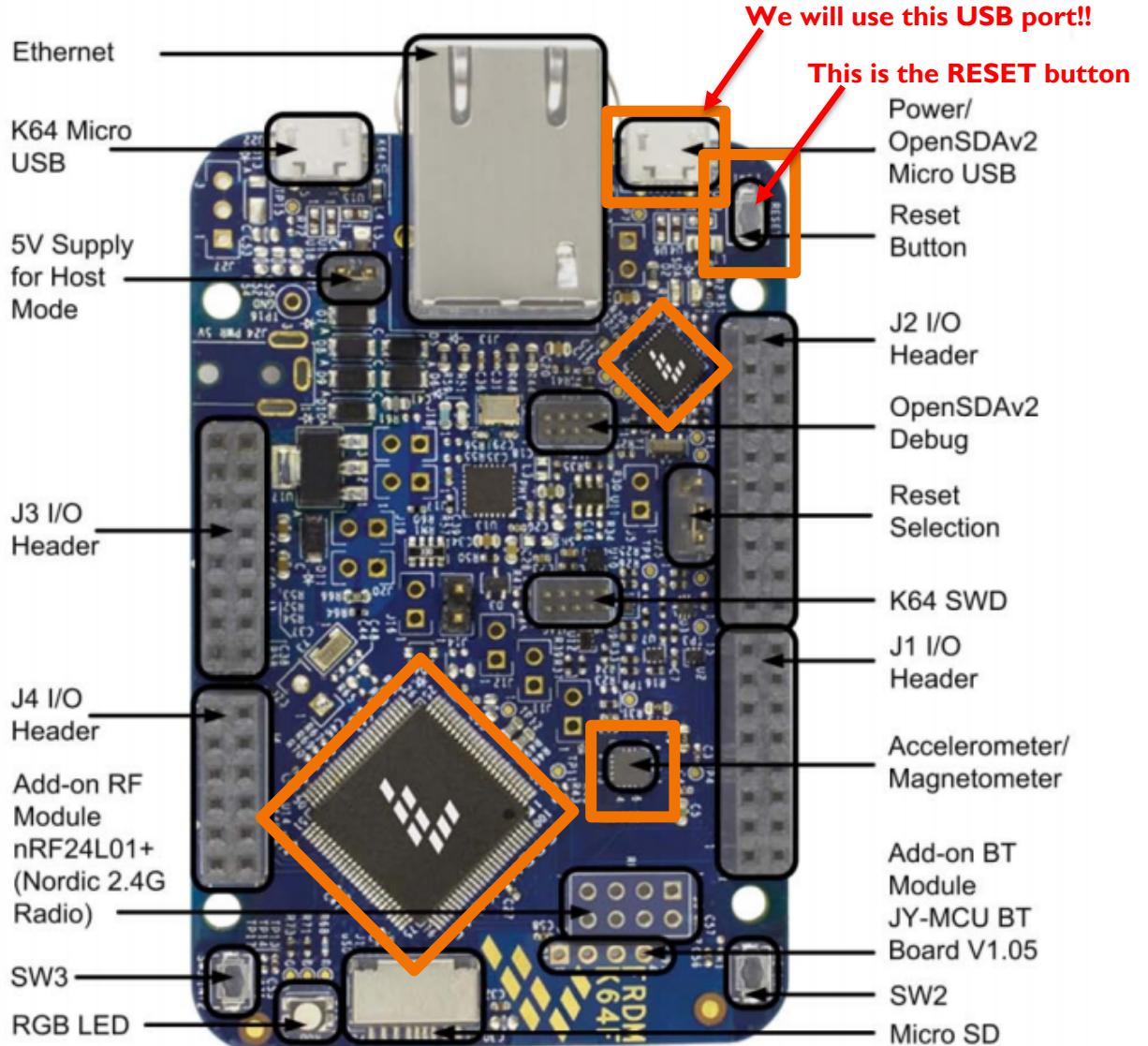
- Navigate to the Azure Portal:
<https://www.azure.com>
- Press “Free Account”, “Start Free”
- Complete the sign-up process
- A confirmation email will be sent that must be acknowledged
- Log into your Azure account

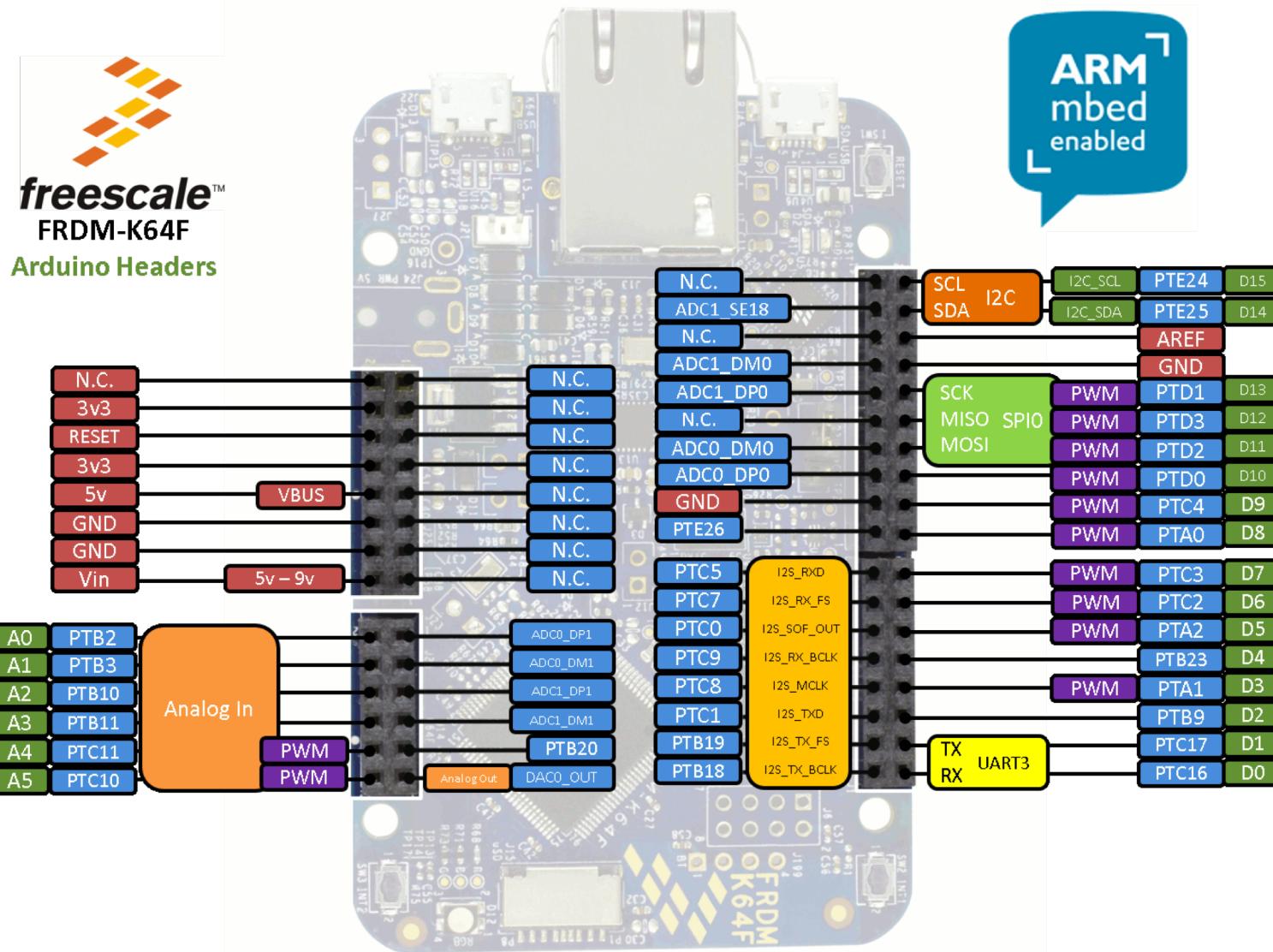


Prior to
Workshop

NXP- FRDM-K64F Overview

- **Freedom Development Platform**
 - Quick, simple development experience with rich features
 - Cortex-M4, 120MHz, 1MB Flash, 256KB SRAM
 - Easy access to MCU I/O
 - 3-axis **accelerometer**/3-axis **magnetometer**
 - RGB LED
 - Add-on **Bluetooth** Module
 - Built-in Ethernet/Add-on **Wireless** Module
 - Micro SD
- **Arduino shield compatible**
- Flash programming functionality
- Enabled by OpenSDA debug interface





Install the Necessary Tools

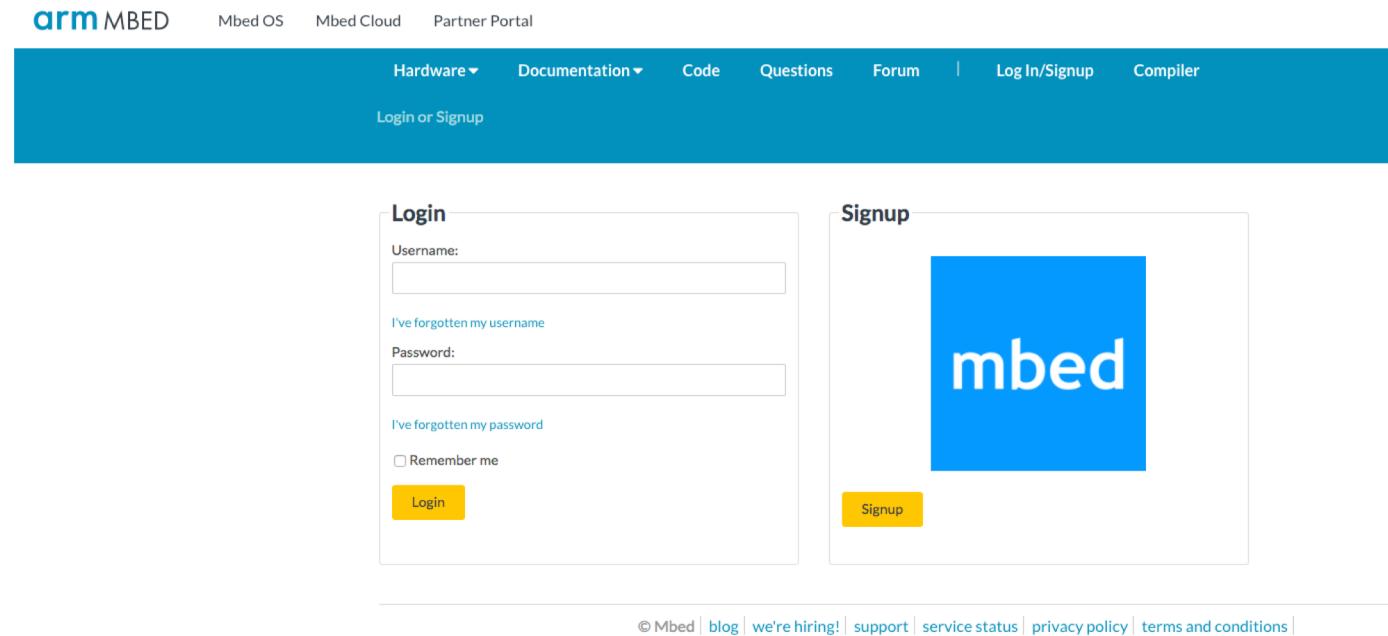
- **This workshop requires a Windows PC (for MS IoTHub Device Explorer)**
 - **IMPORTANT:** Install the mbed USB Serial driver -
https://os.mbed.com/media/downloads/drivers/mbedWinSerial_16466.exe
 - Insert the USB cable and mbed device BEFORE running the Serial Driver installation...
 - the installer MUST see the device first
 - *You will need administrator access to your Windows PC to install this driver*
 - Serial Terminal: Putty - <http://www.putty.org/>
 - Chrome and Firefox Browsers may also be used
 - Docker Toolbox installed on your Windows PC (v1.12.2 or later):
 - <https://github.com/docker/toolbox/releases/tag/v1.12.2>
 - Microsoft Device Explorer (latest version) installed:
 - https://github.com/Azure/azure-iot-sdks/blob/master/tools/DeviceExplorer/doc/how_to_use_device_explorer.md
 - Device Explorer may require additional .NET redistributable downloads... the setup installer will guide you.

Prior to
Workshop

Connect both your USB cable and Ethernet cable to the K64F and leave it there for now

Create Your mbed Account

- Navigate to: <https://os.mbed.com/account/login/?next=/>
- Create an Account

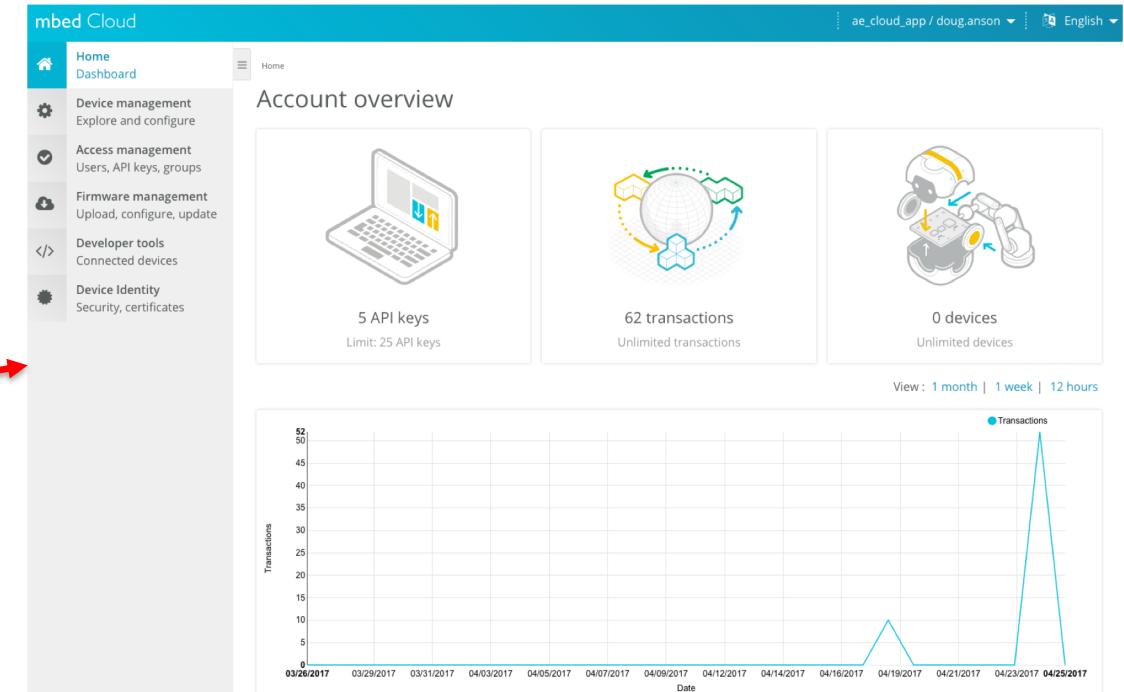
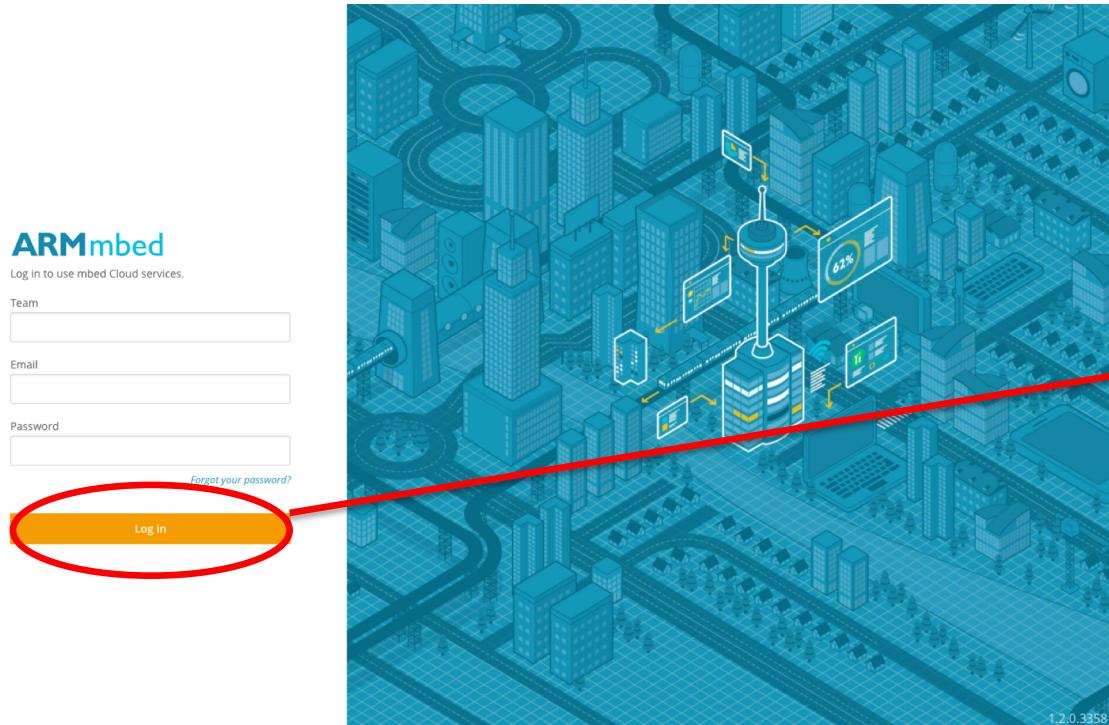


Prior to
Workshop

Create Your mbed Cloud Account...

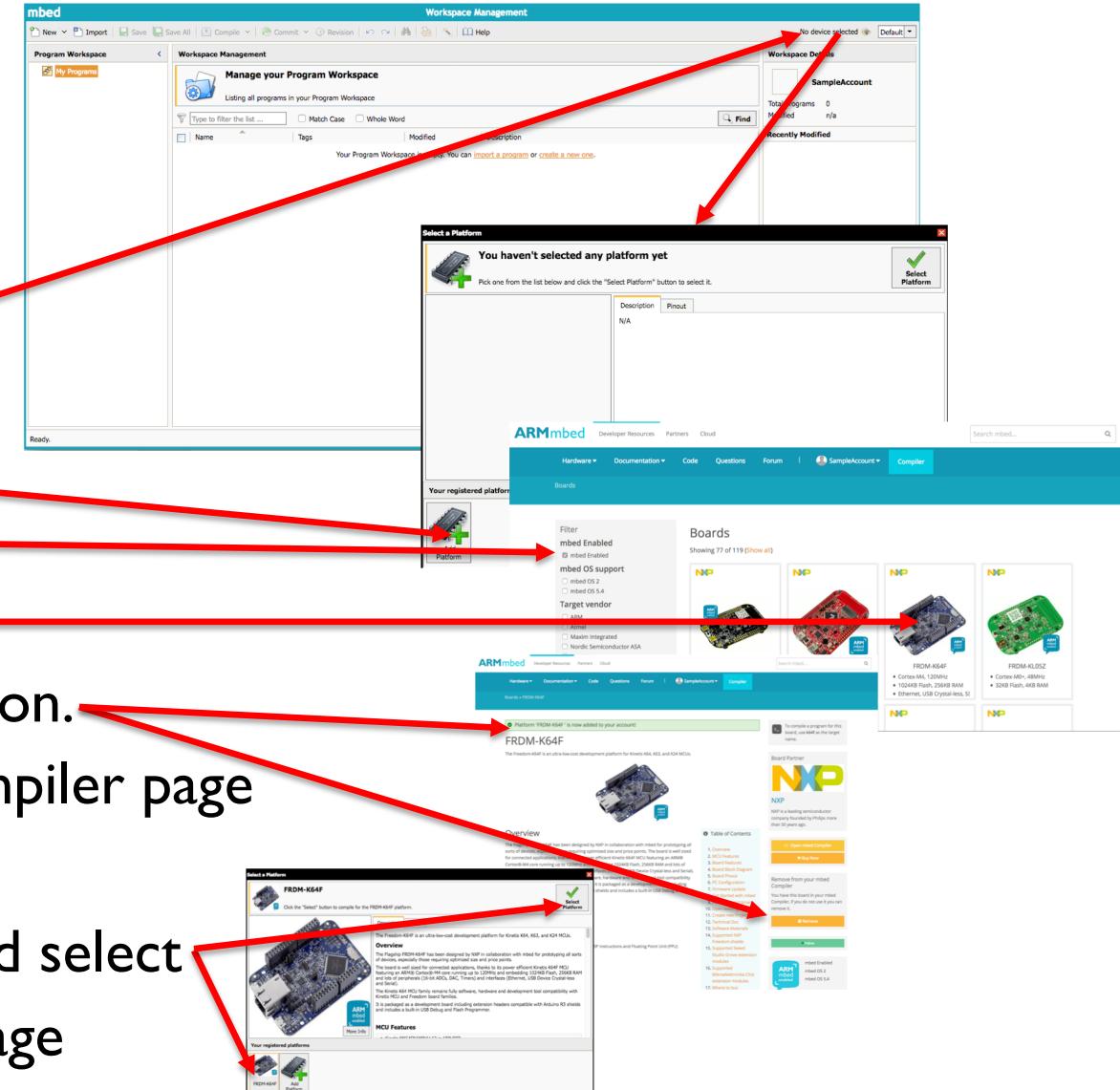
Prior to
Workshop

- Navigate to the mbed Cloud Dashboard: <https://portal.mbedcloud.com>
- Confirm that you can log into your mbed device mbed Cloud dashboard



Log into the Online IDE – Add the K64F Compile Target

- Go to <https://os.mbed.com/>



Import our K64F Endpoint Project

- Go to <https://os.mbed.com/>

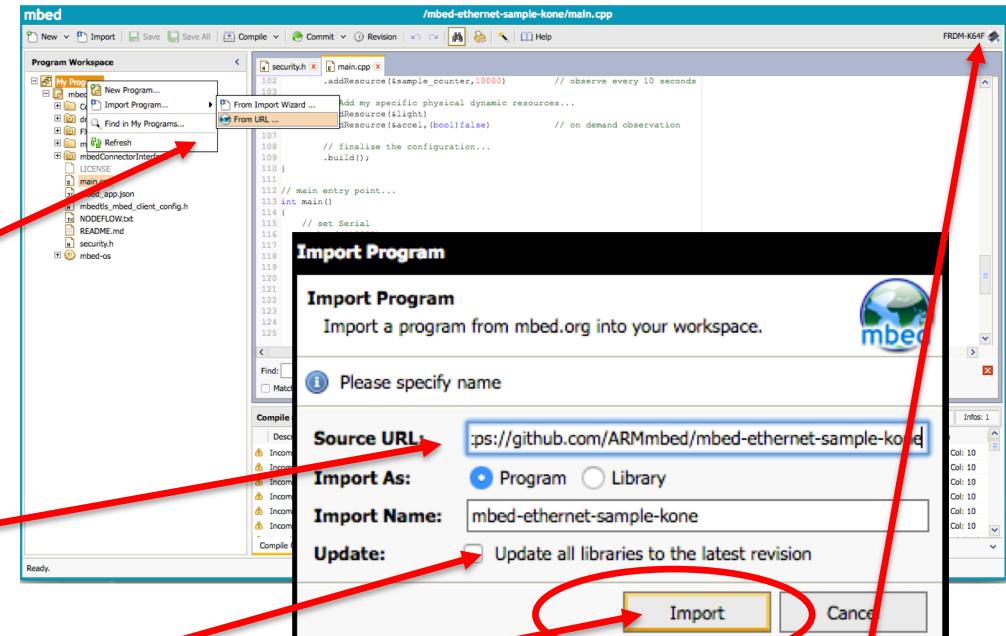
- Select the “Compiler” page

- Right-click on “My Programs” → “Import Program”

- Choose Select “from URL...”

- Enter this URL (Leave the “Update all libraries...” **unchecked**)
<https://github.com/ARMmbed/mbed-cloud-sample/>

- Press “Import”, the ensure that “FRDM-K64F” is selected



Set Provisioning Credentials in Your Endpoint Code

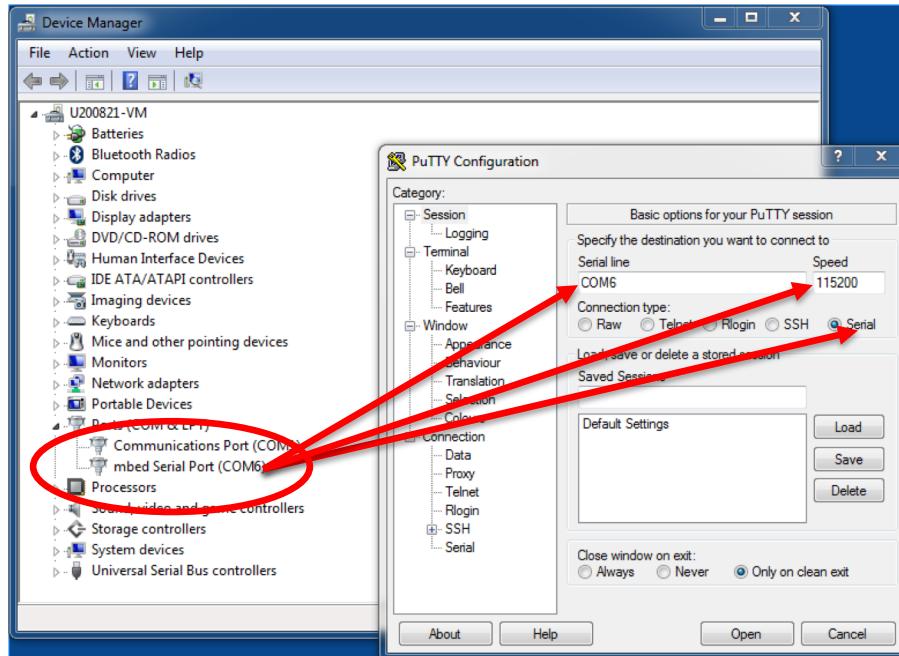
- Go back to the mbed Device mbed Cloud dashboard: <https://portal.mbedcloud.com>
 - In the left sidebar, select “Device Identity” → “Certificates”
 - Under “Actions”, select “create a developer certificate”
 - Give your developer certificate a name and description... press “create certificate”
 - Press “Download Developer C file” – this will deposit “mbed_cloud_dev_credentials.c” in your downloads directory
- Now, go back to the Compiler page of your online IDE
- Replace mbed_cloud_dev_credentials.c with newly downloaded one from the dashboard
- Save
 - Glance at main.cpp... a clean and simple mbed endpoint example
 - Exposes two CoAP resources: accelerometer and LED
- Select the project name and press the “Compile” button
- The endpoint code should compile up successfully
- The online IDE will deposit a “bin” file into your downloads directory
- Drag-n-Drop this bin file to your “MBED” flash drive (may also be called “L
- K64F green LED will flicker for a bit, then stop, and dismount/remount...

```
mbed /mbed-ethernet-sample-kone/main.cpp
Program Workspace
Code Editor
```

The screenshot shows the mbed online IDE interface. The 'Program Workspace' tab is active, displaying a file tree with files like 'main.cpp', 'security.h', 'security.c', 'app.json', 'client_config.h', and 'client_config.json'. The 'Code Editor' tab is also visible, showing the contents of 'main.cpp'. The code in 'main.cpp' includes comments for a CoAP server setup, resource definitions for an accelerometer and LED, and a main entry point. A red arrow points from the 'Program Workspace' tab towards the 'Code Editor' tab.

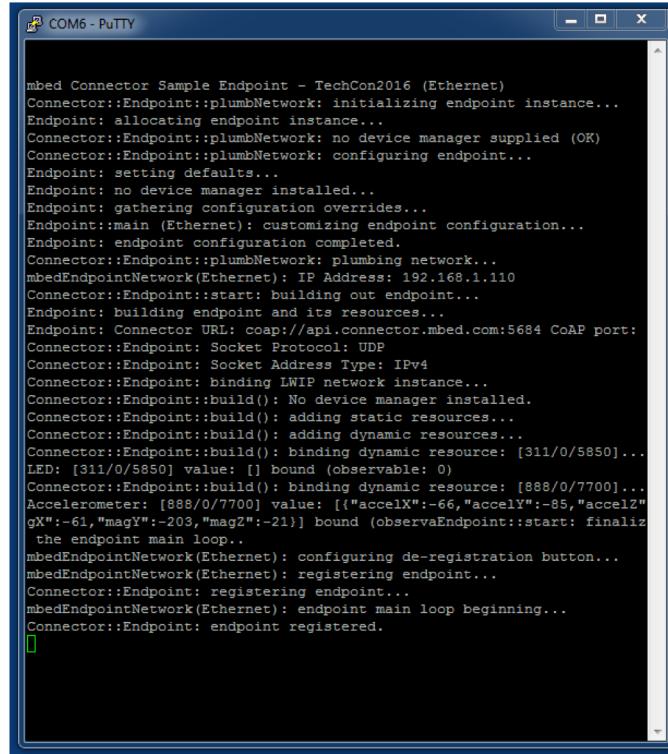
Running your Endpoint Code

- Bring up your Serial Terminal
 - PuTTY for Windows : You must determine what COM port your mbed device is. Look in the Windows Device Manager FYI
- For the endpoint serial configuration, set the baud rate to 115200 baud, defaults for everything else: (8, N, one)
 - PuTTY for Windows: Ensure that you have “Serial” radio button selected...



Running your Endpoint Code...

- Connect your Serial Terminal to the K64F:
 - PuTTY for Windows: Press the “Open” button
- Send a “Break” command from the serial terminal (or press the RESET button on the K64F)
 - PuTTY for Windows: Right-click on top of Window, Select “Special Command” → “Break”
- Look at the output – you need to confirm that you see “endpoint registered” in the output:



```
mbed Connector Sample Endpoint - TechCon2016 (Ethernet)
Connector::Endpoint::plumbNetwork: initializing endpoint instance...
Endpoint: allocating endpoint instance...
Connector::Endpoint::plumbNetwork: no device manager supplied (OK)
Connector::Endpoint::plumbNetwork: configuring endpoint...
Endpoint: setting defaults...
Endpoint: no device manager installed...
Endpoint: gathering configuration overrides...
Endpoint::main (Ethernet): customizing endpoint configuration...
Endpoint: endpoint configuration completed.
Connector::Endpoint::plumbNetwork: plumbing network...
mbedEndpointNetwork(Ethernet): IP Address: 192.168.1.110
Connector::Endpoint::start: building out endpoint...
Endpoint: building endpoint and its resources...
Endpoint: Connector URL: coap://api.connector.mbed.com:5684 CoAP port:
Connector::Endpoint: Socket Protocol: UDP
Connector::Endpoint: Socket Address Type: IPv4
Connector::Endpoint: binding LWIP network instance...
Connector::Endpoint::build(): No device manager installed.
Connector::Endpoint::build(): adding static resources...
Connector::Endpoint::build(): adding dynamic resources...
Connector::Endpoint::build(): binding dynamic resource: [311/0/5850]...
LED: [311/0/5850] value: [] bound (observable: 0)
Connector::Endpoint::build(): binding dynamic resource: [888/0/7700]...
Accelerometer: [888/0/7700] value: [{"accelX": -66, "accelY": -85, "accelZ": -61, "magX": -203, "magY": -21}]] bound (observaEndpoint::start: finaliz
the endpoint main loop..
mbedEndpointNetwork(Ethernet): configuring de-registration button...
mbedEndpointNetwork(Ethernet): registering endpoint...
Connector::Endpoint: registering endpoint...
mbedEndpointNetwork(Ethernet): endpoint main loop beginning...
Connector::Endpoint: endpoint registered.
```

Status Check

So far we've completed the following

- Setup our accounts and PC with appropriate tools (prior to workshop)...
- Retrieved a set of provisioning credentials (mbed_cloud_dev_credentials.c)
- Imported our mbed sample project into the online IDE
- Updated the sample project with the provisioning credentials (mbed_cloud_dev_credentials.c)
- Compiled, Installed, Downloaded, and Copied into the mbed device
- Connected a Serial Terminal (115200,8NI, proper mbed COM port chosen for Windows users...)
- Sent the “Break” command to reset the mbed device
- Saw the device output on our Serial Terminal (PTSOOI method...)

Next, we will import and configure the ARM IoT Hub Prototype mbed Cloud Bridge...

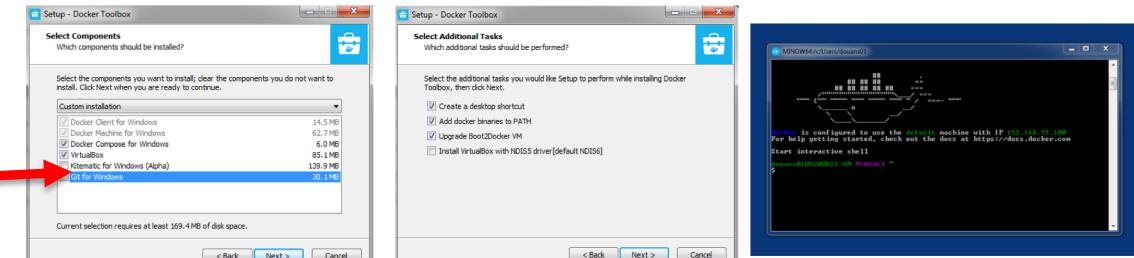
ARM IoTHub mbed Cloud Bridge Setup

What we will do next...

- Ensure that we have the necessary pre-requisites setup on our Windows PC
- Create our mbed Cloud API Key/Token
- Create our IoTHub instance in Azure
- Using DeviceExplorer, we create an IoTHubOwner SAS Token
- Import our Prototype mbed Cloud Bridge instance as a Docker image into our Docker runtime
- Configure our Prototype mbed Cloud Bridge for IoTHub

Double check: all our needed Tools Installed

- Latest Docker Toolbox runtime installed
 - Ensure that “Git for Windows” is checked!
 - <https://github.com/docker/toolbox/releases/tag/v1.12.2>



- MS DeviceExplorer installed and available.
 - https://github.com/Azure/azure-iot-sdks/blob/master/tools/DeviceExplorer/doc/how_to_use_device_explorer.md
- Windows mbed USB driver installed and functioning properly.
 - “DAPLINK” flash drive present
 - “mbed Serial Port” seen in Windows Device Manager when K64F USB is connected
- Putty installed and operational

Create our mbed Cloud Access/API Token

- Navigate to the mbed Cloud Dashboard: <https://portal.mbedcloud.com>

- Log in, Select “Access Management”, then “API keys”

- Press “Create new API Key ”... you will create a key

- Give the new API Key a name

- Select “Developers” group

- Create the API Key

Create new API key

Key name: My New API Key

Groups: Developers

Create API Key Cancel

Showing 5 results

Key name	Groups	Date last connected	Date created
AWS IoT	1	-	April 19, 2017 4:51 PM
Home	1	-	April 19, 2017 4:51 PM
IBM Watson	1	-	April 19, 2017 4:49 PM
MS IoTHub	1	-	April 19, 2017 4:50 PM
Test	1	-	April 19, 2017 4:51 PM

The API key has been created

This will be the last time the API key is available to you, but you can generate a new one from the API Key Details page.

Security credentials

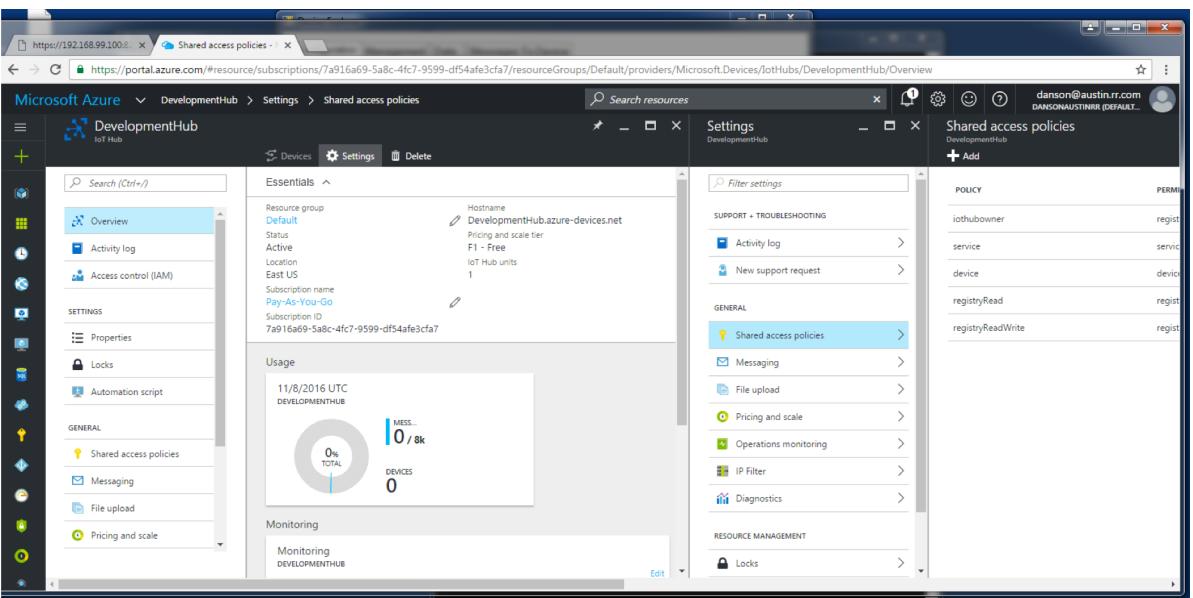
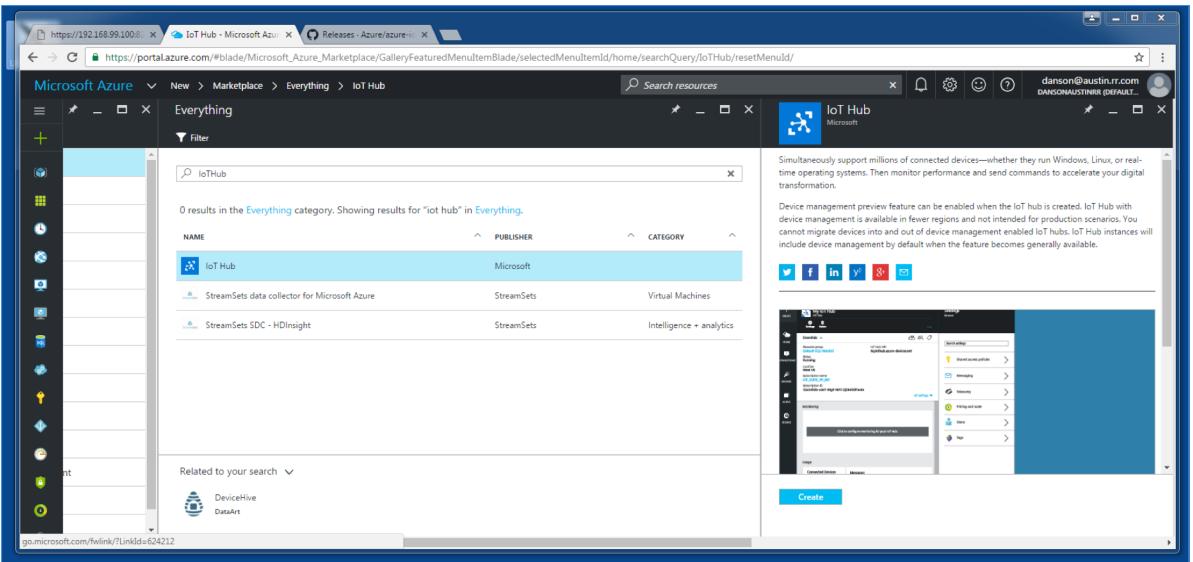
Key name: My New API Key

API key: `ak_1HDEIYj051x3z2G00C09j8M00w4fJALDNaRQWHDAD015bab3e32d02420a01231100000000key12T2tp12C4P3Lx#Wtq0gsp1Ra444xz`

Copy to clipboard Add another key

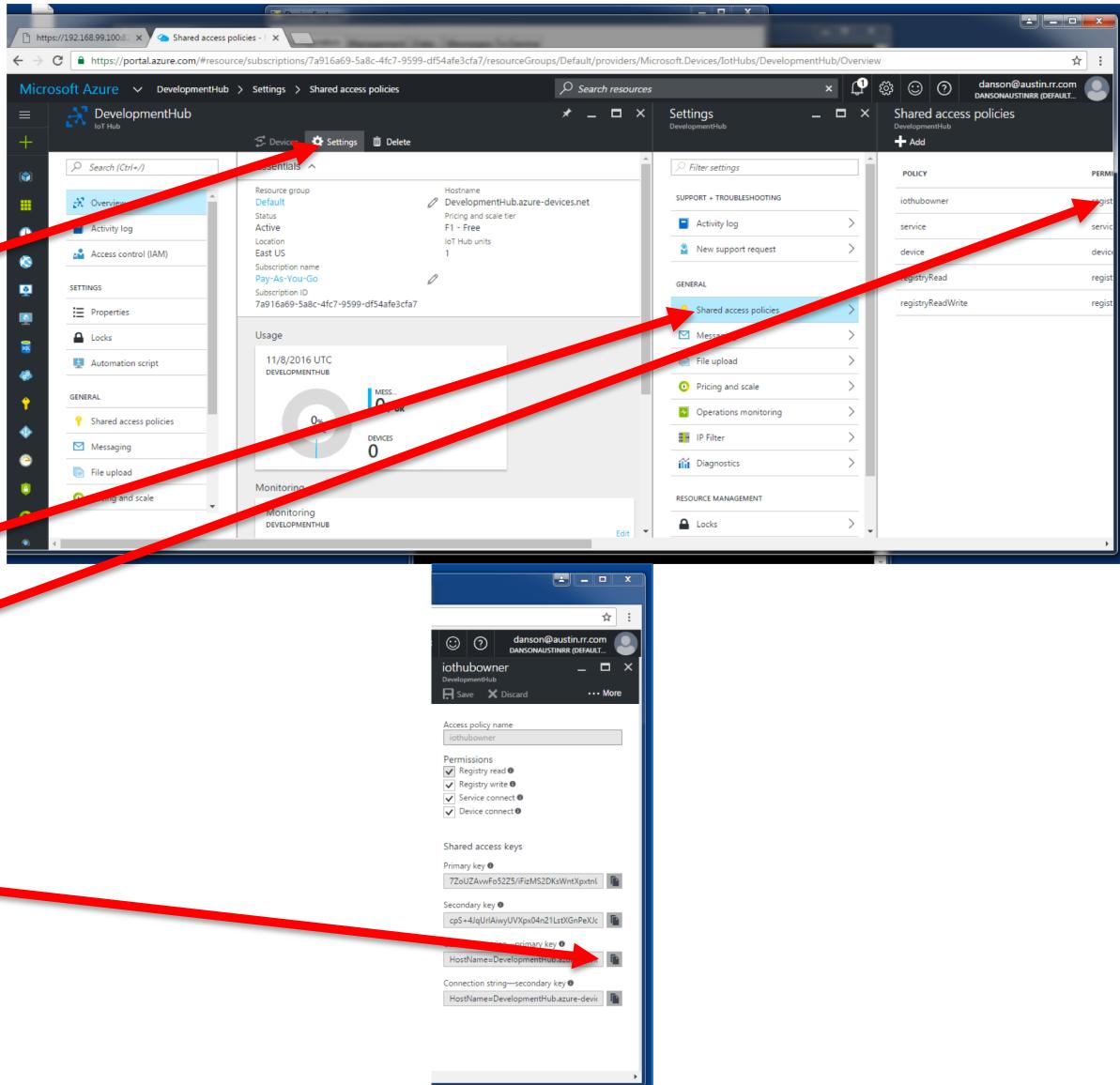
Create our Azure IoT Hub Instance

- Log into your Azure Portal
- Search for IoT Hub
- Select and “Create”
- Give your IoT Hub a name
- Be sure to choose “Free” Tier
- Azure will create the IoT Hub



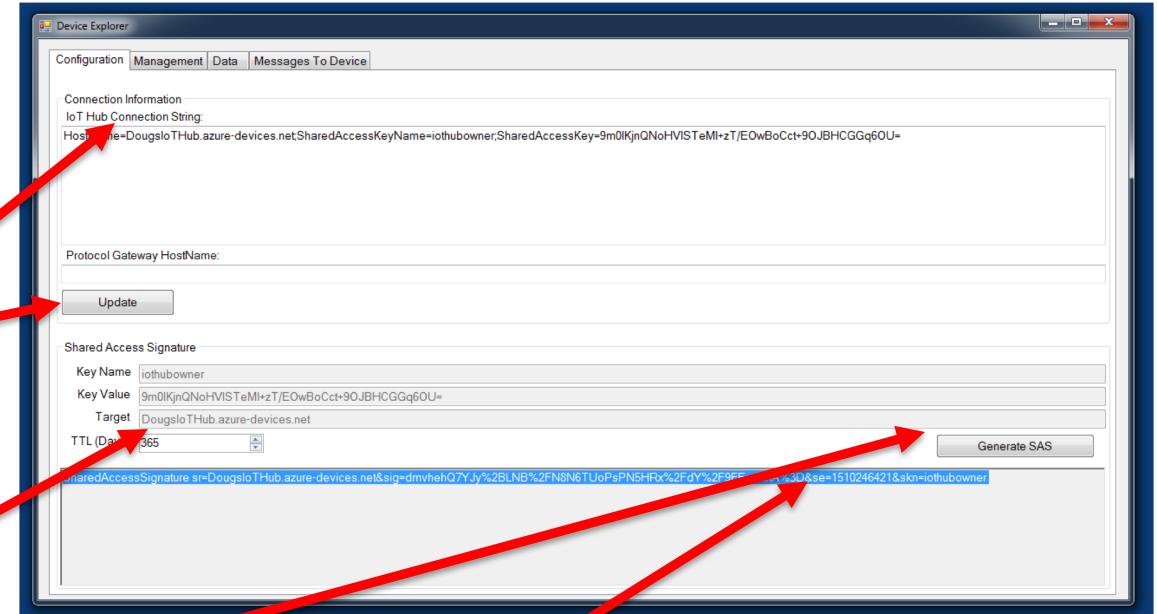
Copy our IoT Hub Connection String

- In the IoT Hub dashboard
- Select your IoT Hub Resource
- Press “Settings”
- Select “Shared Access Policies”
- Select “iothubowner”
- Select Copy to Clipboard
 - Primary Connection String



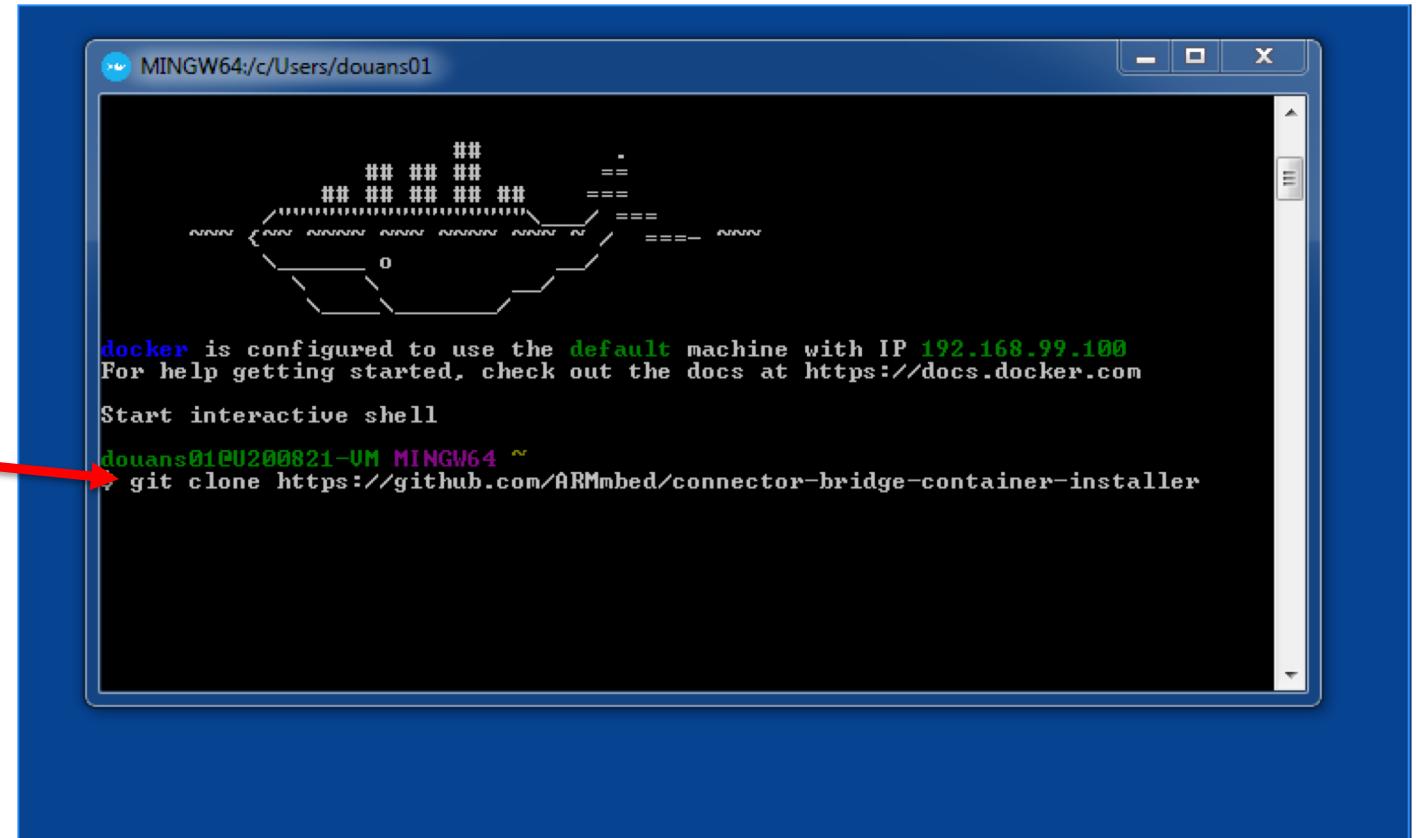
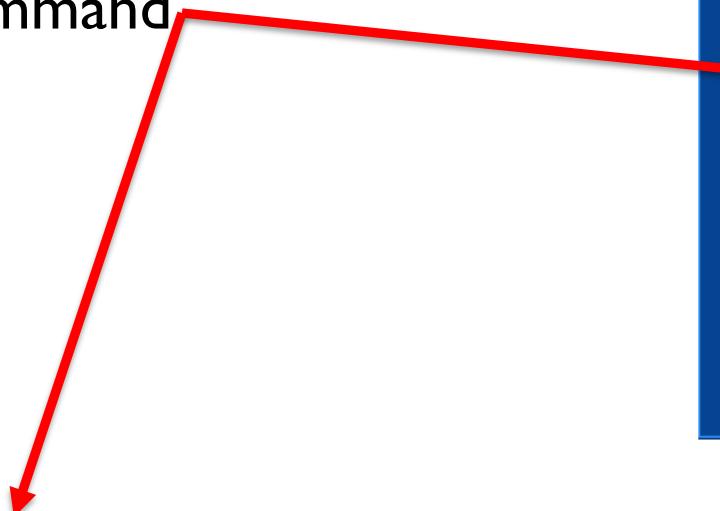
Create our SAS Token with DeviceExplorer

- Launch DeviceExplorer
- Paste your connection string here
- Press “Update”
- Ensure “365” is entered here
- Press “Generate SAS”
- Copy ALL contents of your SAS Token
 - Save this for later!!



Import our mbed Cloud IoT Hub Bridge Installer

- Launch the Docker QuickStart Terminal
- Type the following “git” command

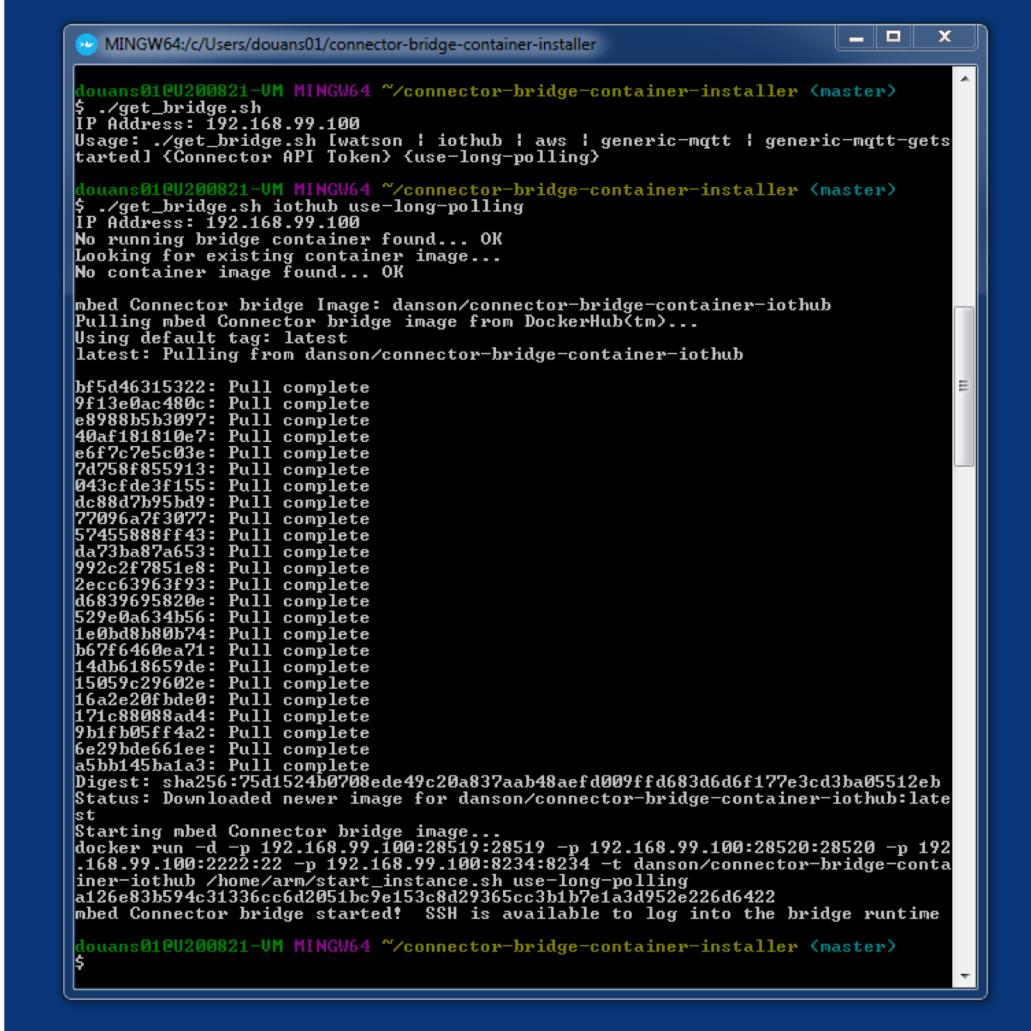


```
MINGW64:/c/Users/douans01
.
.
.
docker is configured to use the default machine with IP 192.168.99.100
For help getting started, check out the docs at https://docs.docker.com
Start interactive shell
douans01@U200821-U MINGW64 ~
> git clone https://github.com/ARMmbed/connector-bridge-container-installer
```

```
% git clone https://github.com/ARMmbed/connector-bridge-container-installer
```

Import our mbed Cloud IoT Hub Bridge Container

- cd into the installer repo
% cd connector-bridge-container-installer
- Run the installer script (look at options)
% ./get_bridge.sh
- Run the installer script with options
% ./get_bridge.sh iothub use-long-polling
- Bridge container will import from DockerHub



```
MINGW64:/c/Users/douans01/connector-bridge-container-installer
$ ./get_bridge.sh
Usage: ./get_bridge.sh [watson | iothub | aws | generic-mqtt | generic-mqtt-gets]
                     [Connector API Token] [use-long-polling]

douans01@U200821-VM MINGW64 ~/connector-bridge-container-installer <master>
$ ./get_bridge.sh iothub use-long-polling
IP Address: 192.168.99.100
No running bridge container found... OK
Looking for existing container image...
No container image found... OK

mbed Connector bridge Image: danson/connector-bridge-container-iothub
Pulling mbed Connector bridge image from DockerHub(tm)...
Using default tag: latest
latest: Pulling from danson/connector-bridge-container-iothub

bf5d46315322: Pull complete
9f13e0ac480c: Pull complete
e89885b3097: Pull complete
40af181810e7: Pull complete
e6f7c7e5c03e: Pull complete
7d758f855913: Pull complete
043cfde3f155: Pull complete
dc88d7b95bd9: Pull complete
72096a7f3077: Pull complete
5745588ff43: Pull complete
da73ba87a653: Pull complete
992c2f7851e8: Pull complete
2ecc63963f93: Pull complete
d6839695820e: Pull complete
529e0a634b56: Pull complete
1e0fd880b74: Pull complete
b67f6460ea71: Pull complete
14db618659de: Pull complete
15059c29602e: Pull complete
16a2e20fbde0: Pull complete
171c88088ad4: Pull complete
9b1fb05ff4a2: Pull complete
6e29bde661ee: Pull complete
a5bb145baba3: Pull complete
Digest: sha256:75d1524b0708ede49c20a837aab4aef009ffd683d6d6f177e3cd3ba05512eb
Status: Downloaded newer image for danson/connector-bridge-container-iothub:latest

Starting mbed Connector bridge image...
docker run -d -p 192.168.99.100:28519 -p 192.168.99.100:28520 -p 192.168.99.100:2222:22 -p 192.168.99.100:8234:8234 -t danson/connector-bridge-container-iothub /home/arm/start_instance.sh use-long-polling
a126e83b594c31336cc6d2051bc9e153c8d29365cc3b1b7e1a3d952e226d6422
mbed Connector bridge started! SSH is available to log into the bridge runtime

douans01@U200821-VM MINGW64 ~/connector-bridge-container-installer <master>
$
```

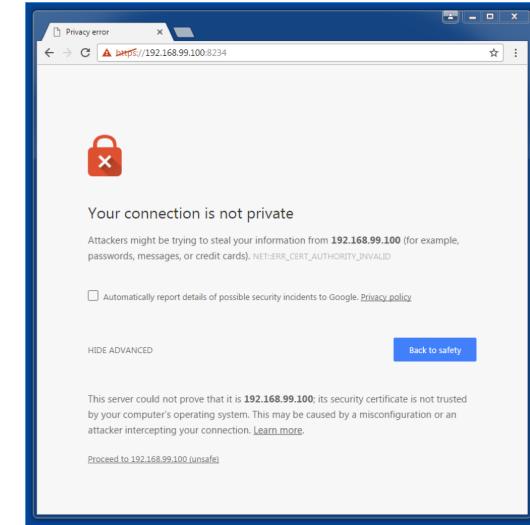
Configure our IoT Hub Bridge (Windows)

- Your Container IP address will be:

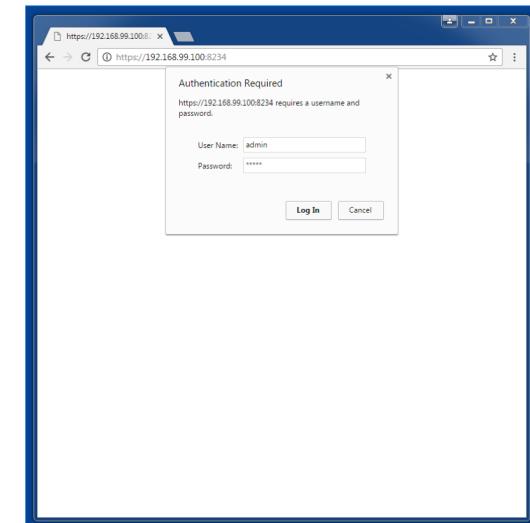
192.168.99.100

- Bring up Firefox/Chrome and go to:

<https://192.168.99.100:8234>

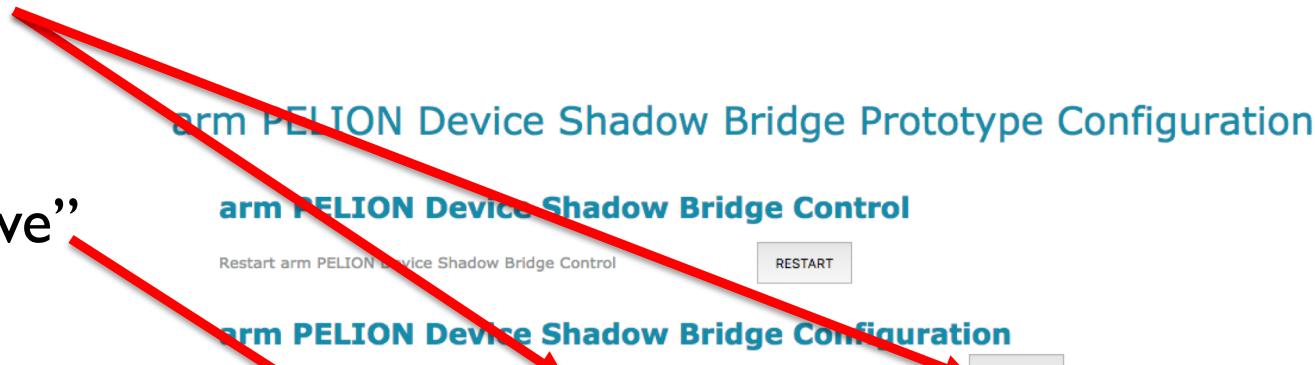


- Accept the self signed certificate
- Username: admin PW: admin



Configure the mbed Cloud Bridge for IoT Hub...

- Enter the mbed Cloud API Key/Token, then Press “Save”



arm PELION Device Shadow Bridge Prototype Configuration

arm PELION Device Shadow Bridge Control

Restart arm PELION Device Shadow Bridge Control

RESTART

arm PELION Device Shadow Bridge Configuration

Setting	Value	Action
Pelion API Key	Pelion_API_Key_Goes_Here	SAVE
Enable Bridge Local Polling	true	SAVE
Pelion API Address	api.us-east-1.mbedcloud.com	SAVE
Bridge (Override) IP Address	off	SAVE
Microsoft IoT Hub Name	IoT_Hub_Name_Goes_Here	SAVE
Microsoft IoT Hub SAS Token (iothubowner)	IoT_Hub_Owner_SAS_Token_Goes_Here	SAVE
Bridge Threading Pool Size	1000	SAVE
Bridge Threading Max Pool Size	1000000	SAVE
Bridge Thread Keep Alive (sec)	60	SAVE

- Enter the name of your IoT Hub, Press “Save”
- Enter your iothubowner SAS Token, Press ”Save”
- Now, Press “RESTART”
- Your mbed Cloud Bridge is now configured for IoT Hub

Status Check

So far we've completed the following

- Ensure that we have the necessary pre-requisites setup on our Windows PC
- Create our mbed Cloud API Key/Token
- Create our IoTHub instance in Azure
- Using DeviceExplorer, we create an IoTHubOwner SAS Token
- Import our bridge instance: a Docker image into our Docker Toolbox runtime
- Configure our Prototype mbed Cloud Bridge for IoTHub

With our bridge now operational, we should be able to restart our endpoint and see messages coming into our IoTHub via DeviceExplorer

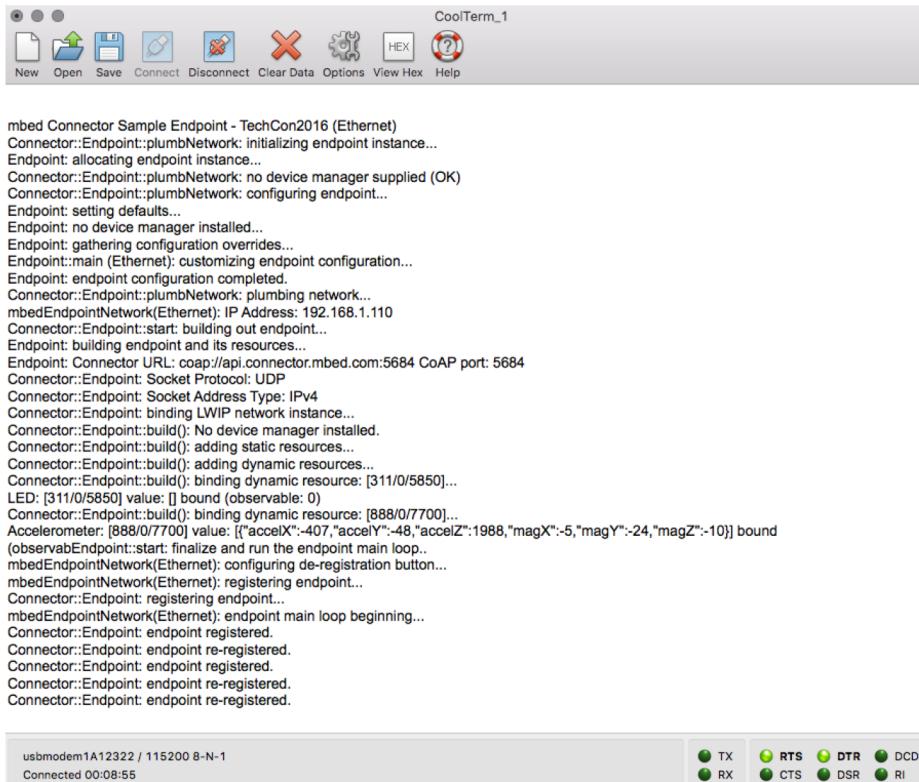
Putting it all Together

- Press the “reset” button on your mbed device (next to the USB port)
- Ensure that you see “endpoint registered” in the serial terminal
- Bring up DeviceExplorer

Lets check how things are working...

Putting it all Together: Check the Serial Terminal

- You should see output that looks something like this:
 - Make sure that you see a message like “endpoint registered”



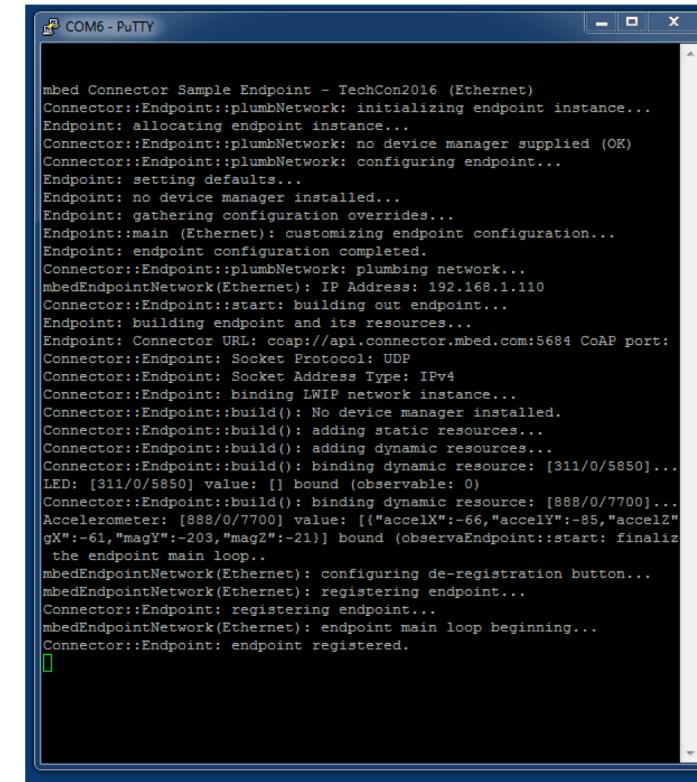
CoolTerm_1

New Open Save Connect Disconnect Clear Data Options View Hex Help

```
mbed Connector Sample Endpoint - TechCon2016 (Ethernet)
Connector::Endpoint::plumbNetwork: initializing endpoint instance...
Endpoint: allocating endpoint instance...
Connector::Endpoint::plumbNetwork: no device manager supplied (OK)
Connector::Endpoint::plumbNetwork: configuring endpoint...
Endpoint: setting defaults...
Endpoint: no device manager installed...
Endpoint: gathering configuration overrides...
Endpoint::main (Ethernet): customizing endpoint configuration...
Endpoint: endpoint configuration completed.
Connector::Endpoint::plumbNetwork: plumbing network...
mbedEndpointNetwork(Ethernet): IP Address: 192.168.1.110
Connector::Endpoint::start: building out endpoint...
Endpoint: building endpoint and its resources...
Endpoint: Connector URL: coap://api.connector.mbed.com:5684 CoAP port: 5684
Connector::Endpoint: Socket Protocol: UDP
Connector::Endpoint: Socket Address Type: IPv4
Connector::Endpoint: binding LWIP network instance...
Connector::Endpoint::build(): No device manager installed.
Connector::Endpoint::build(): adding static resources...
Connector::Endpoint::build(): adding dynamic resources...
Connector::Endpoint::build(): binding dynamic resource: [311/0/5850]...
LED: [311/0/5850] value: [] bound (observable: 0)
Connector::Endpoint::build(): binding dynamic resource: [888/0/7700]...
Accelerometer: [888/0/7700] value: [{"accelX": -407, "accelY": -48, "accelZ": 1988, "magX": -5, "magY": -24, "magZ": -10}] bound
(observaEndpoint::start: finalize and run the endpoint main loop...
mbedEndpointNetwork(Ethernet): configuring de-registration button...
mbedEndpointNetwork(Ethernet): registering endpoint...
Connector::Endpoint: registering endpoint...
mbedEndpointNetwork(Ethernet): endpoint main loop beginning...
Connector::Endpoint: endpoint registered.
Connector::Endpoint: endpoint re-registered.
Connector::Endpoint: endpoint registered.
Connector::Endpoint: endpoint re-registered.
Connector::Endpoint: endpoint re-registered.
```

usbmodem1A12322 / 115200 8-N-1
Connected 00:08:55

TX RTS DTR DCD
RX CTS DSR RI

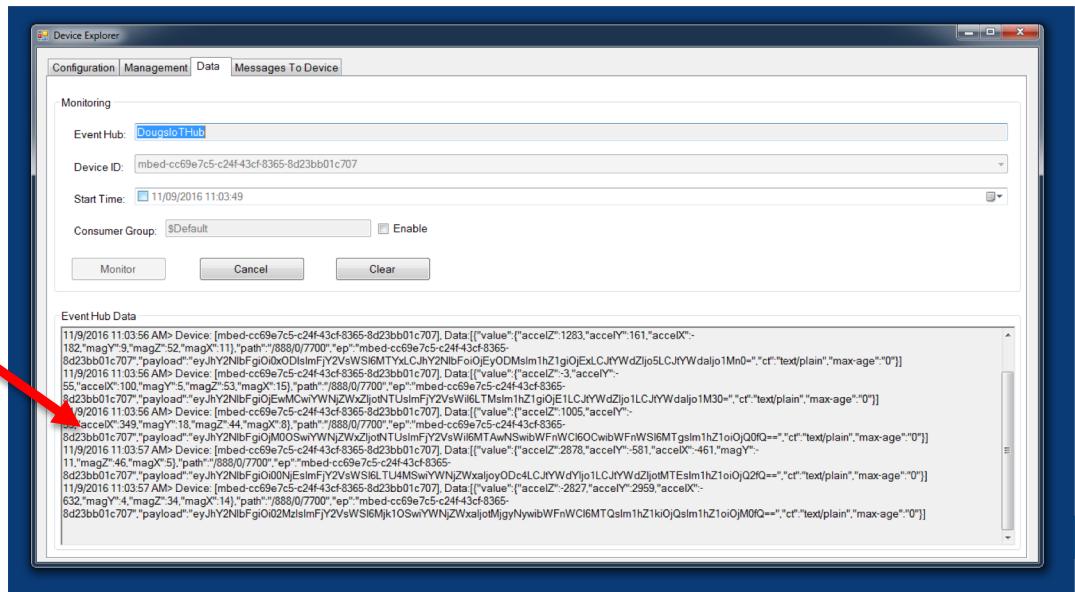
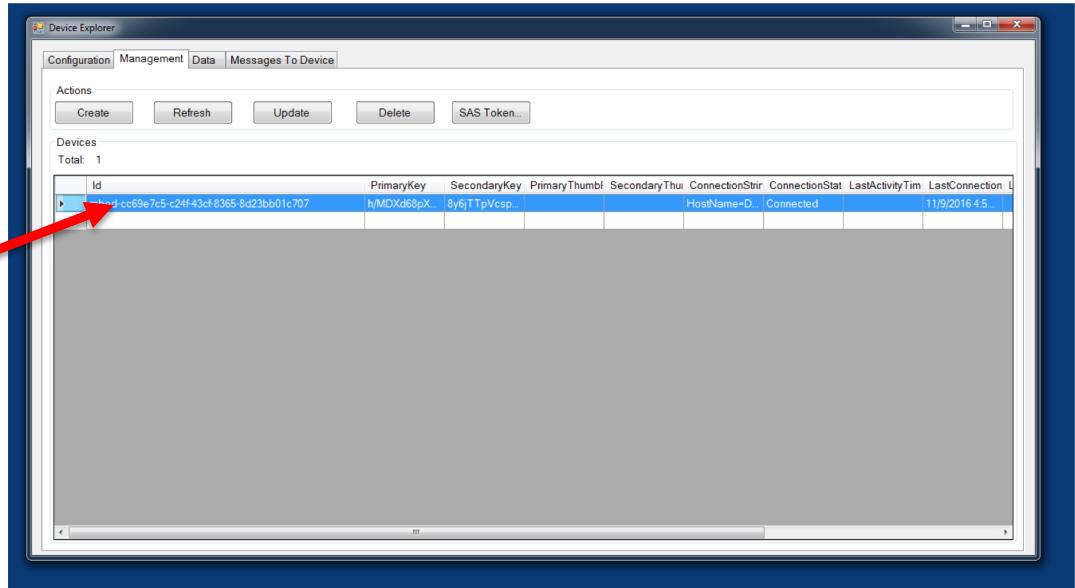


COM6 - PuTTY

```
mbed Connector Sample Endpoint - TechCon2016 (Ethernet)
Connector::Endpoint::plumbNetwork: initializing endpoint instance...
Endpoint: allocating endpoint instance...
Connector::Endpoint::plumbNetwork: no device manager supplied (OK)
Connector::Endpoint::plumbNetwork: configuring endpoint...
Endpoint: setting defaults...
Endpoint: no device manager installed...
Endpoint: gathering configuration overrides...
Endpoint::main (Ethernet): customizing endpoint configuration...
Endpoint: endpoint configuration completed.
Connector::Endpoint::plumbNetwork: plumbing network...
mbedEndpointNetwork(Ethernet): IP Address: 192.168.1.110
Connector::Endpoint::start: building our endpoint...
Endpoint: building endpoint and its resources...
Endpoint: Connector URL: coap://api.connector.mbed.com:5684 CoAP port: 5684
Connector::Endpoint: Socket Protocol: UDP
Connector::Endpoint: Socket Address Type: IPv4
Connector::Endpoint: binding LWIP network instance...
Connector::Endpoint::build(): No device manager installed.
Connector::Endpoint::build(): adding static resources...
Connector::Endpoint::build(): adding dynamic resources...
Connector::Endpoint::build(): binding dynamic resource: [311/0/5850]...
LED: [311/0/5850] value: [] bound (observable: 0)
Connector::Endpoint::build(): binding dynamic resource: [888/0/7700]...
Accelerometer: [888/0/7700] value: [{"accelX": -66, "accelY": -85, "accelZ": -61, "magX": -203, "magY": -24, "magZ": -10}] bound
(observaEndpoint::start: finalize and run the endpoint main loop...
mbedEndpointNetwork(Ethernet): configuring de-registration button...
mbedEndpointNetwork(Ethernet): registering endpoint...
Connector::Endpoint: registering endpoint...
mbedEndpointNetwork(Ethernet): endpoint main loop beginning...
Connector::Endpoint: endpoint registered.
```

View your device telemetry in DeviceExplorer

- Bring up your Device Explorer
- Under the “Management” tab you should see your device listed
- Under “Data” you should see output (shake your device...)
 - Press "monitor" to start monitoring



Interact with your device in DeviceExplorer

- Select “Messages to Device”

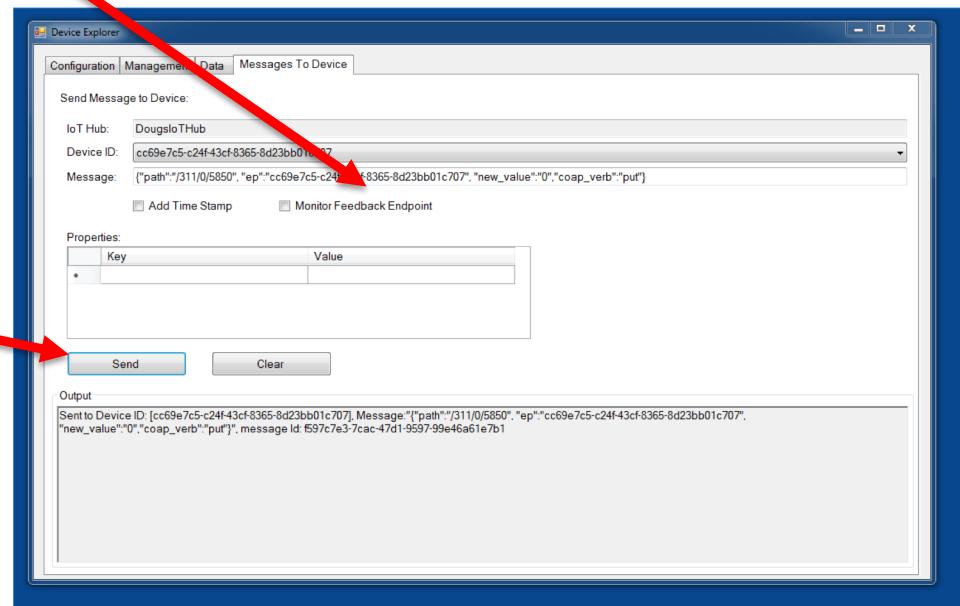
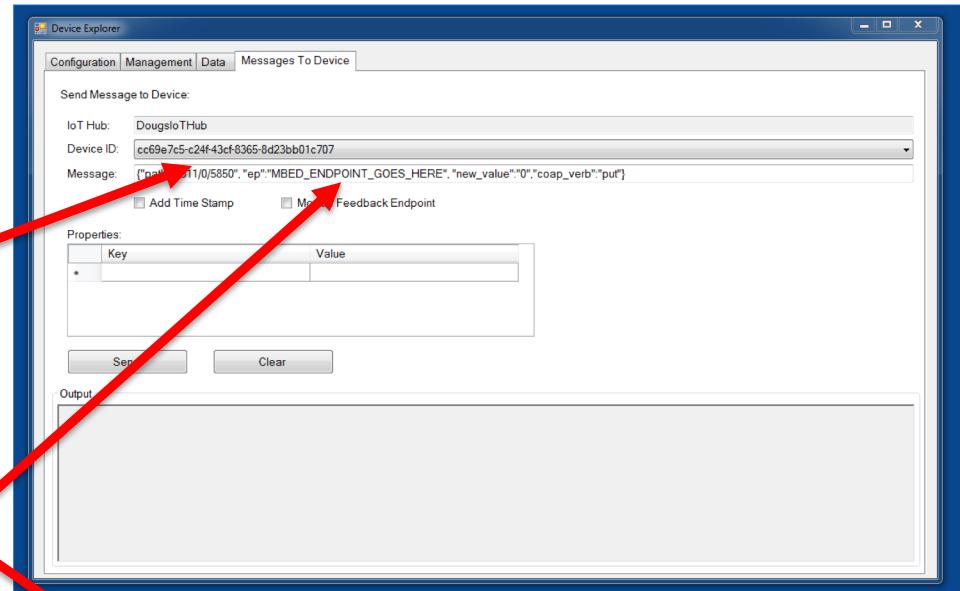
- Enter the following JSON here

```
{"path":"/311/0/5850", "ep":"MBED_ENDPOINT_GOES_HERE",  
"new_value":"0", "coap_verb":"put"}
```

- Replace **MBED_ENDPOINT_GOES_HERE** with
your actual device name (look in the Data you've
received for quick copy...)

- Press “Send”

- Blue LED should turn ON or OFF depending on
“new_value” being “0”, “1”



Summary

We have completed the following – congratulations!

- Created our Azure IoTHub and mbed environments and PC tool setup
- Imported our mbed endpoint, customized, installed, and ran it
- Created our own Azure IoTHub instance
- Imported and configured our ARM IoTHub Prototype mbed Cloud Bridge
- Examined live telemetry from within MS's DeviceExplorer
- Interacted with our device through MS's DeviceExplorer

Great JOB! Thanks for your time.