

Workshop: Connecting IoT devices to the cloud with IBM Watson IoT and mbed Cloud **(EA-Prototype Bridge)**

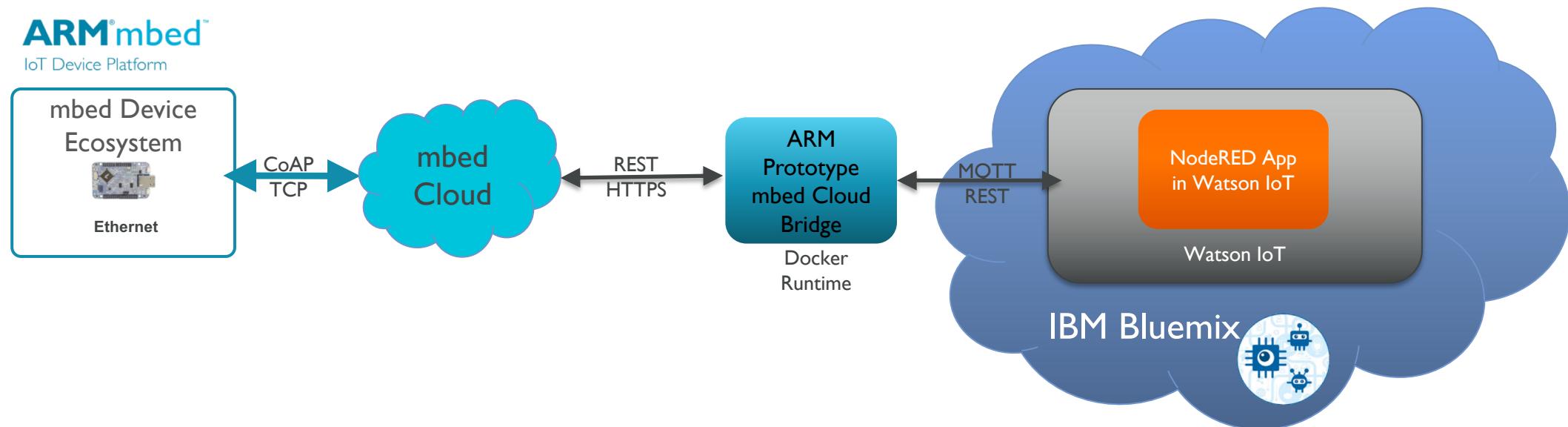
ARM

Doug Anson
Solutions Architect / IoT BU / ARM

Brian Daniels
Applications Engineer / IoT BU / ARM

April 3, 2018 – v2.4

What will we build in this Workshop?



- Connect your mbed device into Watson IoT through mbed Cloud and Watson Prototype Cloud Bridge
 - Create a simple mbed device and connect it to mbed Cloud
 - We will use the ARM Prototype Bridge to connect mbed Cloud to IBM Watson IoT
 - Exploration of the device data telemetry using NodeRED flows within Watson IoT

Workshop: Let's get started!

- Create and setup your Bluemix account (should be completed prior to workshop)
- Install the necessary tools/drivers into your Windows/Mac/Linux 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 (115200,8N1, 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...

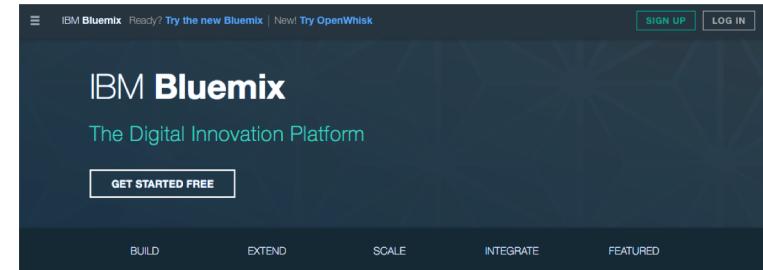
Quick Links: Visit https://github.com/ARMmbed/bridge_workshop_ibm_watson_cloud

- README.md has most of the links in the workshop...

Create our Bluemix Account

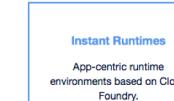
- Navigate to the Bluemix Dashboard:
<https://console.bluemix.net/>
- Press “Sign Up”
- Complete the sign-up process
- A confirmation email will be sent that must be acknowledged
- Log into your Bluemix account and establish your default organization and space

Prior to
Workshop



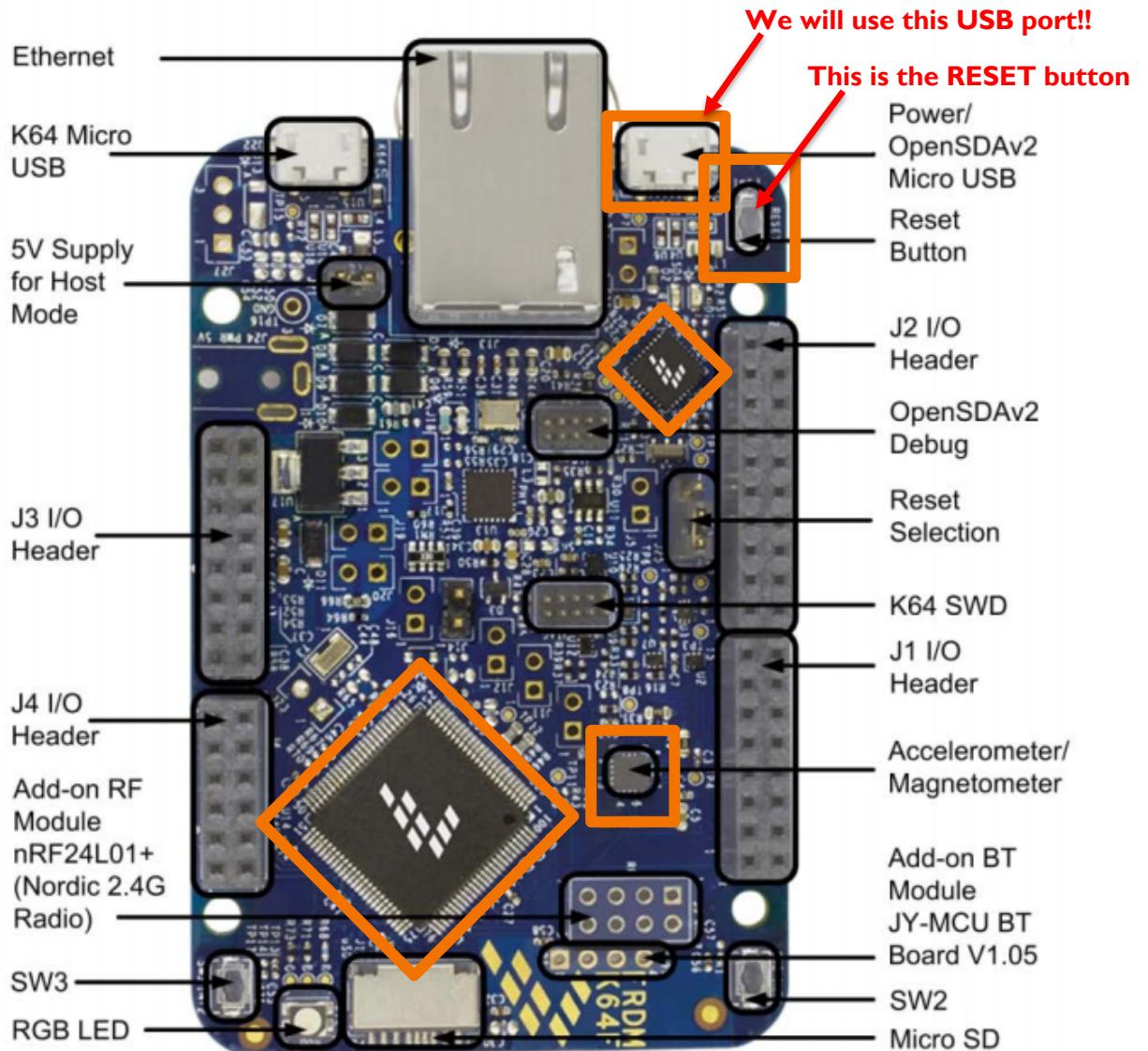
Build your apps, your way.

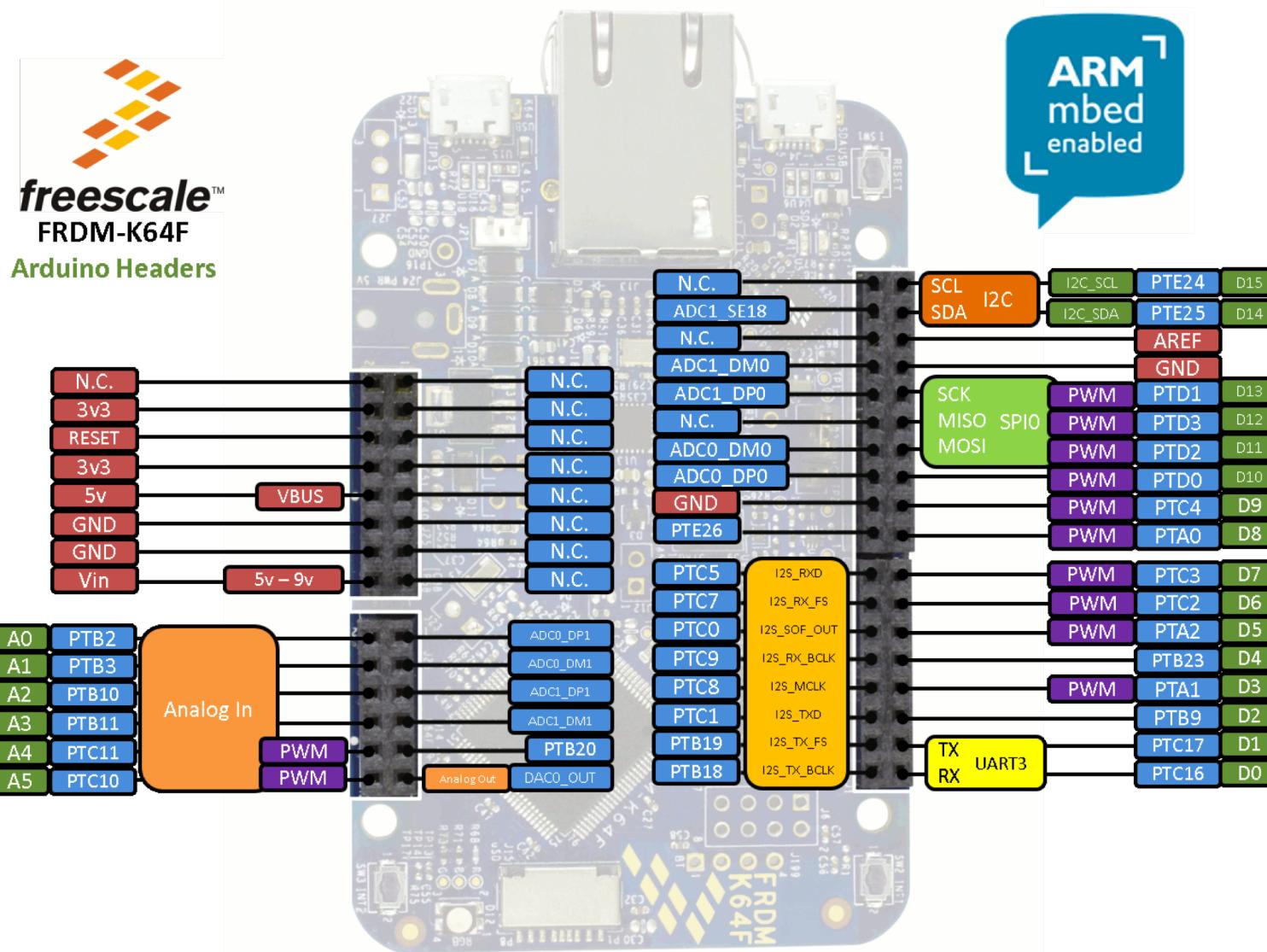
Use a combination of the most prominent open-source compute technologies to power your apps. Then, let Bluemix handle the rest.

The screenshot shows the 'Sign up for IBM Bluemix' form. It includes fields for First Name*, Email Address*, Last Name*, Password*, Re-enter Password*, Company, Security Question*, and Security Answer*. There is also a dropdown for Select your country or region (set to UNITED KINGDOM) and a checkbox for 'Keep me informed of products, services, and offerings from IBM companies worldwide'. At the bottom is a 'CREATE ACCOUNT' button.

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

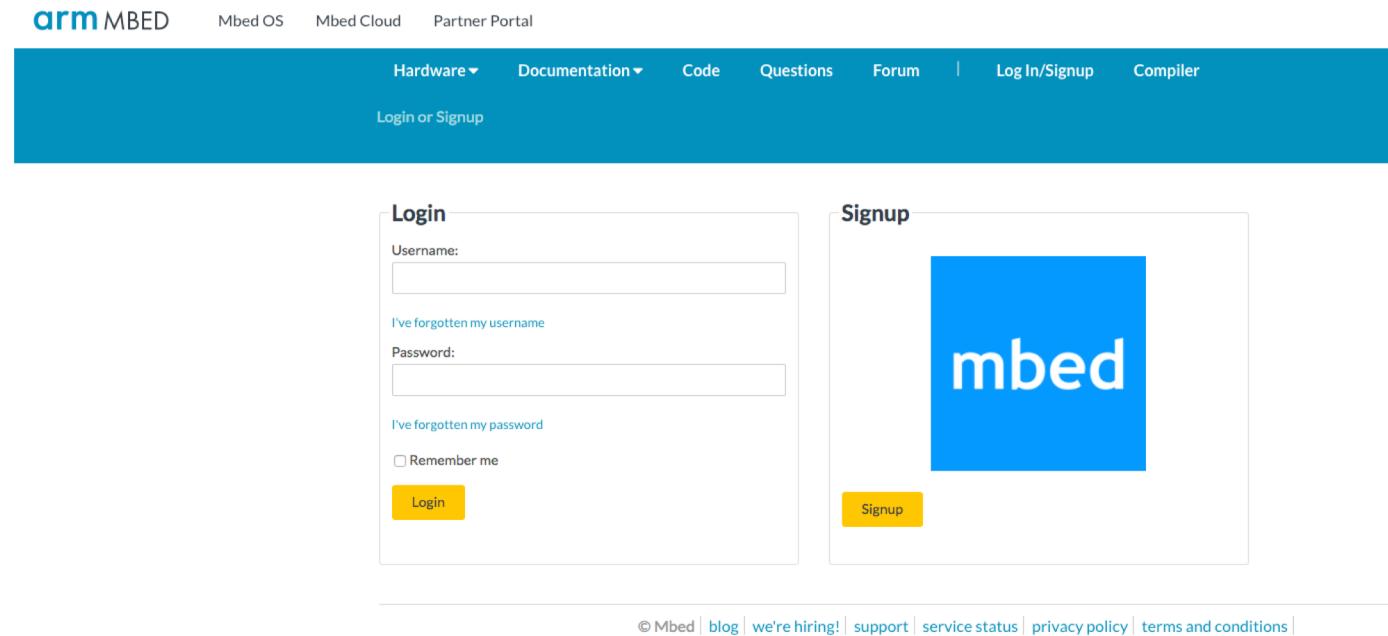
Prior to
Workshop

- **Windows**
 - **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
 - Serial Terminal: Putty - <http://www.putty.org/>
 - Option: Install CoolTerm - http://freeware.the-meiers.org/CoolTerm_Win.zip
 - Chrome and Firefox Browsers installed – **NOTE: IE will NOT WORK... Please use Chrome and/or Firefox**
- **Mac**
 - Serial Terminal: CoolTerm - http://freeware.the-meiers.org/CoolTerm_Mac.zip
 - Chrome and Firefox Browsers installed
- **Linux**
 - Serial Terminal: CoolTerm - http://freeware.the-meiers.org/CoolTerm_Linux.zip
 - Chrome and Firefox Browsers installed

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

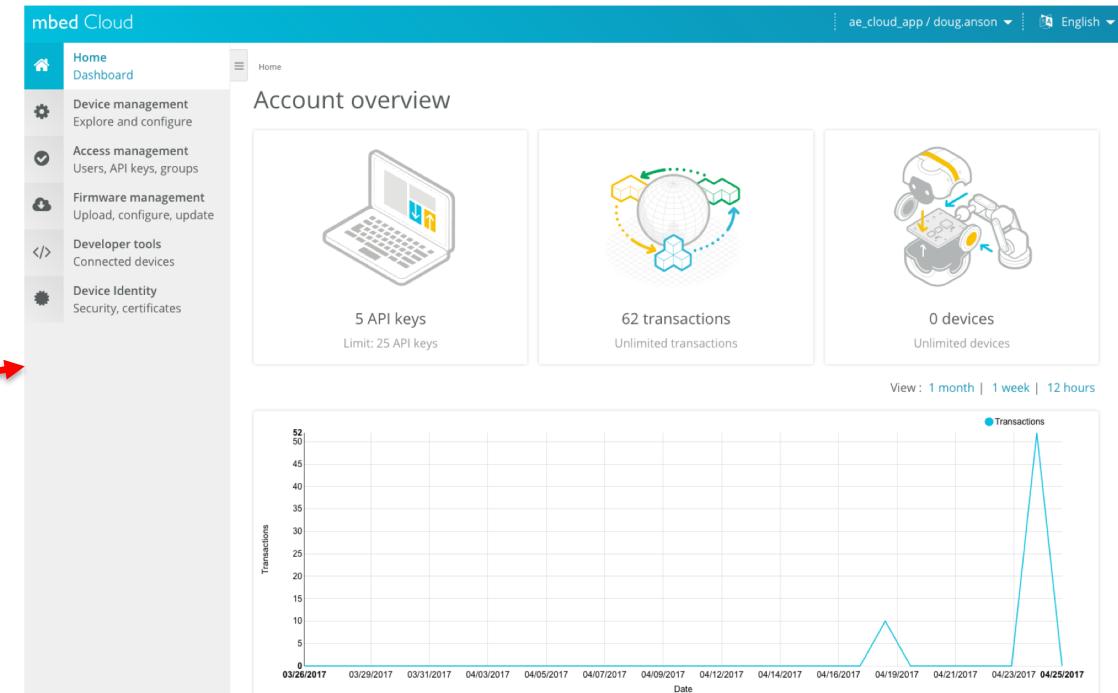
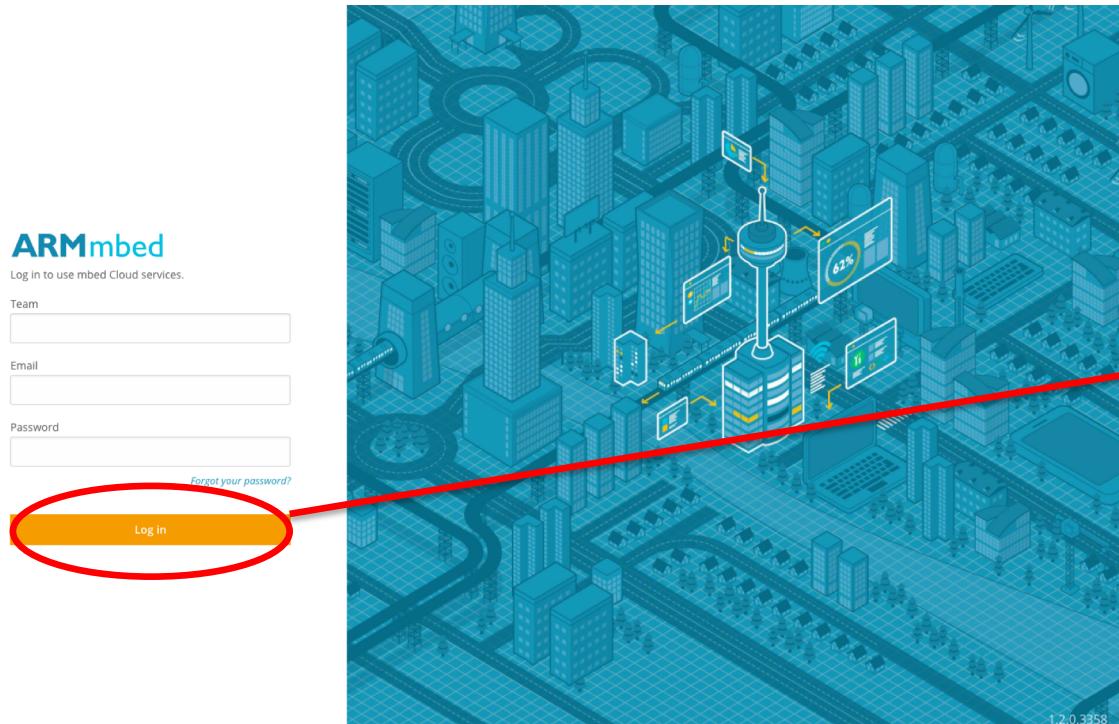


Prior to
Workshop

Create Your mbed Cloud Account...

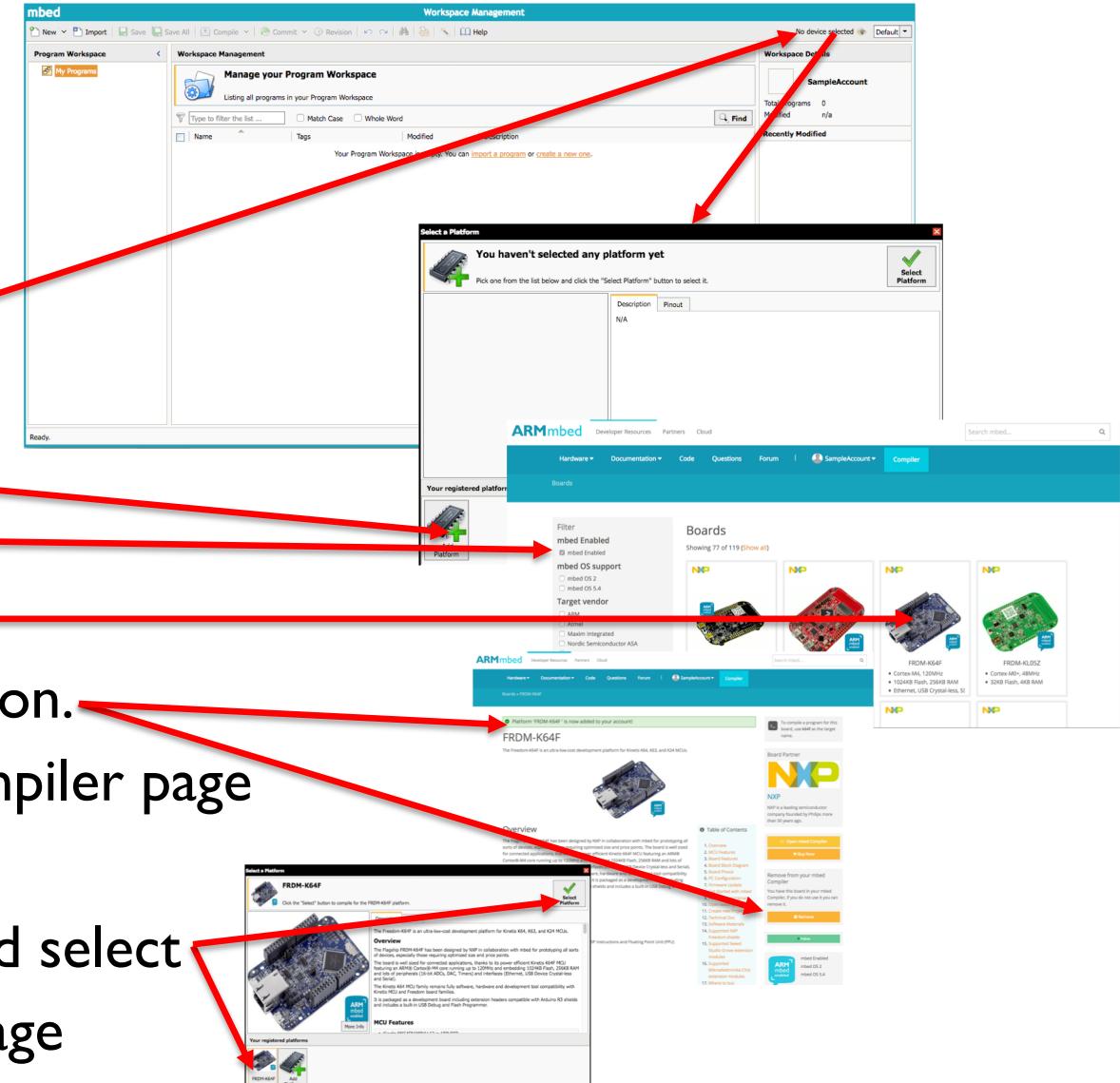
Prior to
Workshop

- Navigate to the mbed Cloud Dashboard: <https://portal.us-east-1.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/>

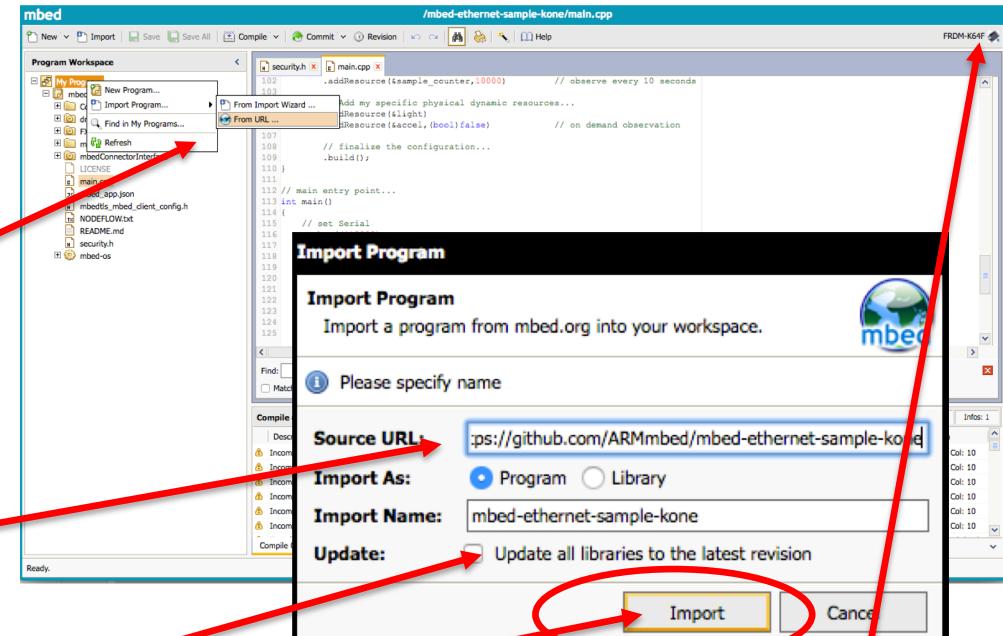
- Select the “Compiler” page

- 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



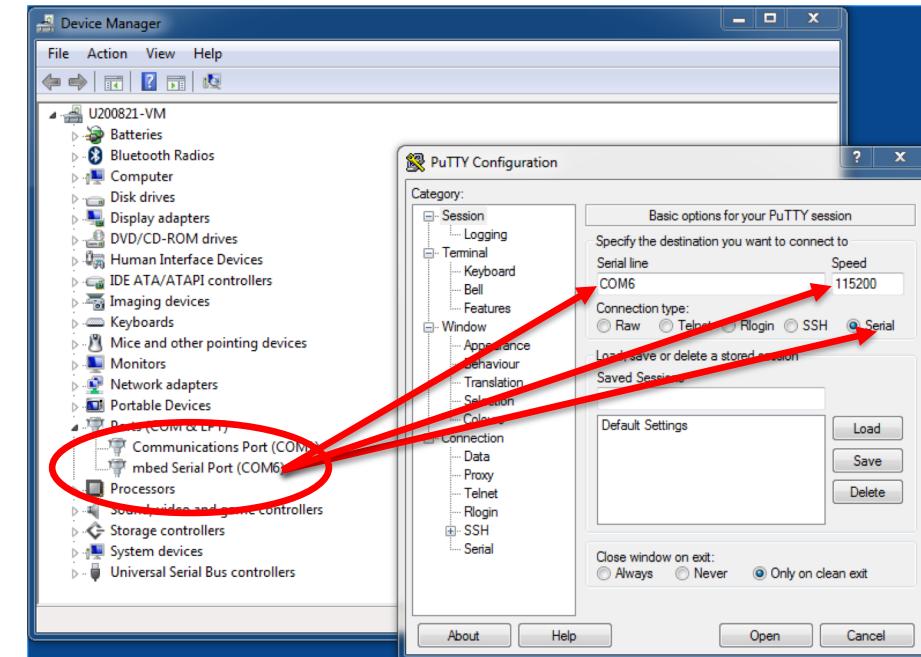
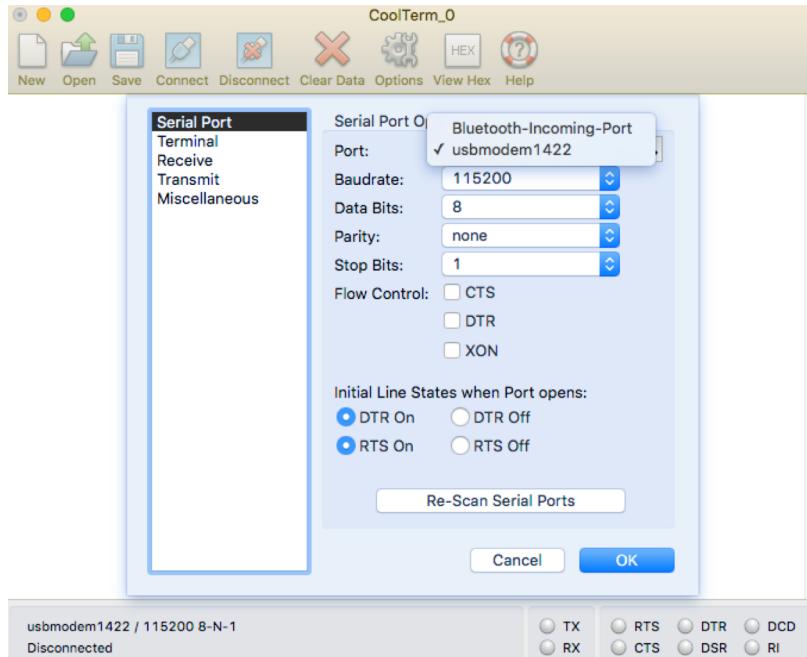
Set Provisioning Credentials in Your Endpoint Code

- Go back to the mbed Device mbed Cloud dashboard: <https://portal.us-east-1.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...

The screenshot shows the mbed online IDE interface. The 'Program Workspace' tab is active, displaying a list of files including `main.cpp`, `security.h`, and `security.c`. The 'Code Editor' tab is also visible, showing the contents of `main.cpp`. The code in `main.cpp` includes comments about observing a counter every 10 seconds, initializing configuration, and announcing a device endpoint. It also defines a serial port and a device manager. The 'Code Editor' tab has search and replace tools at the bottom.

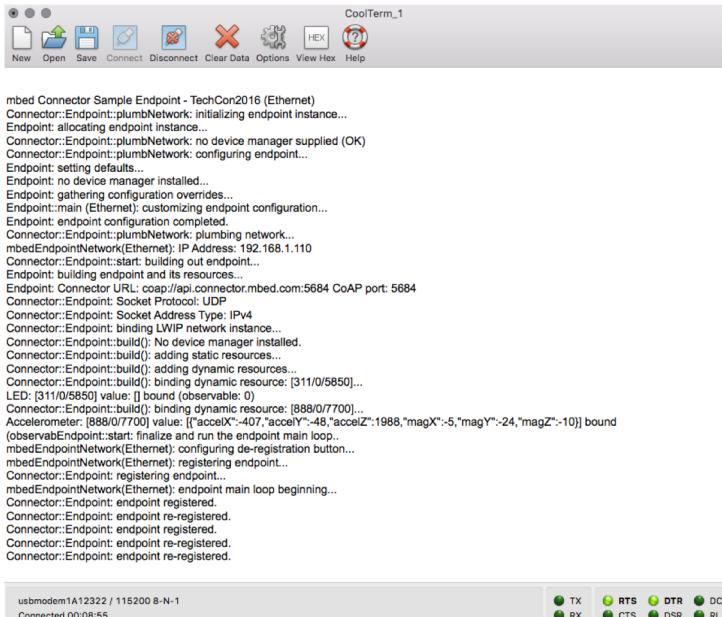
Running your Endpoint Code

- Bring up your Serial Terminal
 - CoolTerm for Windows, Mac, Linux: Select: “Options→”Re-scan serial ports”. Select the mbed one found.
 - For MAC, you may need to “authorize” the CoolTerm Application under your “System Preferences”-> “Security & Privacy”... you may have to allow apps to run from “any developer”, then authorize the launch of CoolTerm.
 - 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...

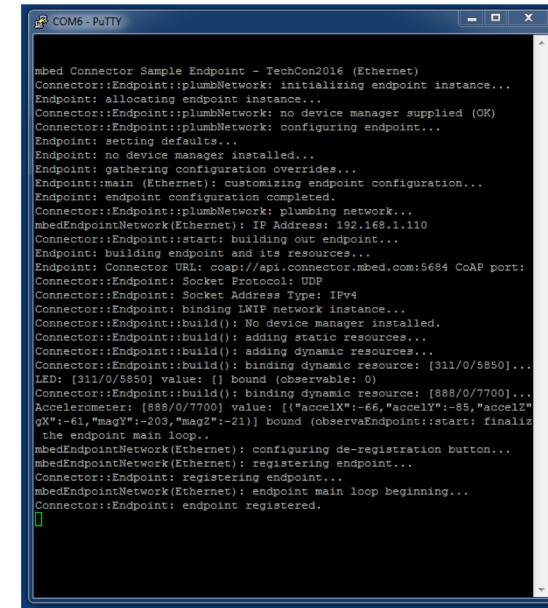
- Connect your Serial Terminal to the K64F:
 - CoolTerm for Windows, Mac, Linux: Simply press the “Connect” button on the top part of the CoolTerm GUI
 - PuTTY for Windows: Press the “Open” button
- Send a “Break” command from the serial terminal (or press the RESET button on the K64F)
 - CoolTerm for Windows, Mac, Linux: “Connection” → “Send Break”
 - 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:



The screenshot shows the CoolTerm application window titled "CoolTerm_1". The menu bar includes "File", "Open", "Save", "Data", "Connect", "Disconnect", "Clear Data", "Options", "View Hex", and "Help". The main pane displays the following log 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 services...
Endpoint: (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::bind dynamic resource: [311/0/5850]...
LED: [311/0/5850] value: [] bound (observable: 0)
Connector::Endpoint::bind dynamic resource: [888/0/7700]...
Accelerometer: [888/0/7700] value: [{"accelX": -48, "accelY": -85, "accelZ": 10}, {"magX": -61, "magY": -203, "magZ": -211}] bound (observable: 1)
mbedEndpointNetwork(Ethernet): configuring de-registration button...
mbedEndpointNetwork(Ethernet): registering endpoint...
Connector::Endpoint::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 registered.
```

At the bottom, there are status indicators for TX, RX, RTS, CTS, DTR, DSR, DCD, RI, and DSR.



The screenshot shows the PuTTY application window titled "COM6 - PUTTY". The main pane displays the following log 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: 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::bind dynamic resource: [311/0/5850]...
LED: [311/0/5850] value: [] bound (observable: 0)
Connector::Endpoint::bind dynamic resource: [888/0/7700]...
Accelerometer: [888/0/7700] value: [{"accelX": -48, "accelY": -85, "accelZ": 10}, {"magX": -61, "magY": -203, "magZ": -211}] bound (observable: 1)
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 Prototype mbed Cloud Bridge for Watson IoT...

Prototype mbed Cloud Bridge Import for Watson IoT

What we will do next...

- Ensure that we have the necessary pre-requisites setup on our PC
- Import our Prototype mbed Cloud Bridge instance as a Docker image into our Docker runtime

Double check (Windows) :All our needed tools are installed

- Latest Docker Toolbox runtime installed

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



- 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

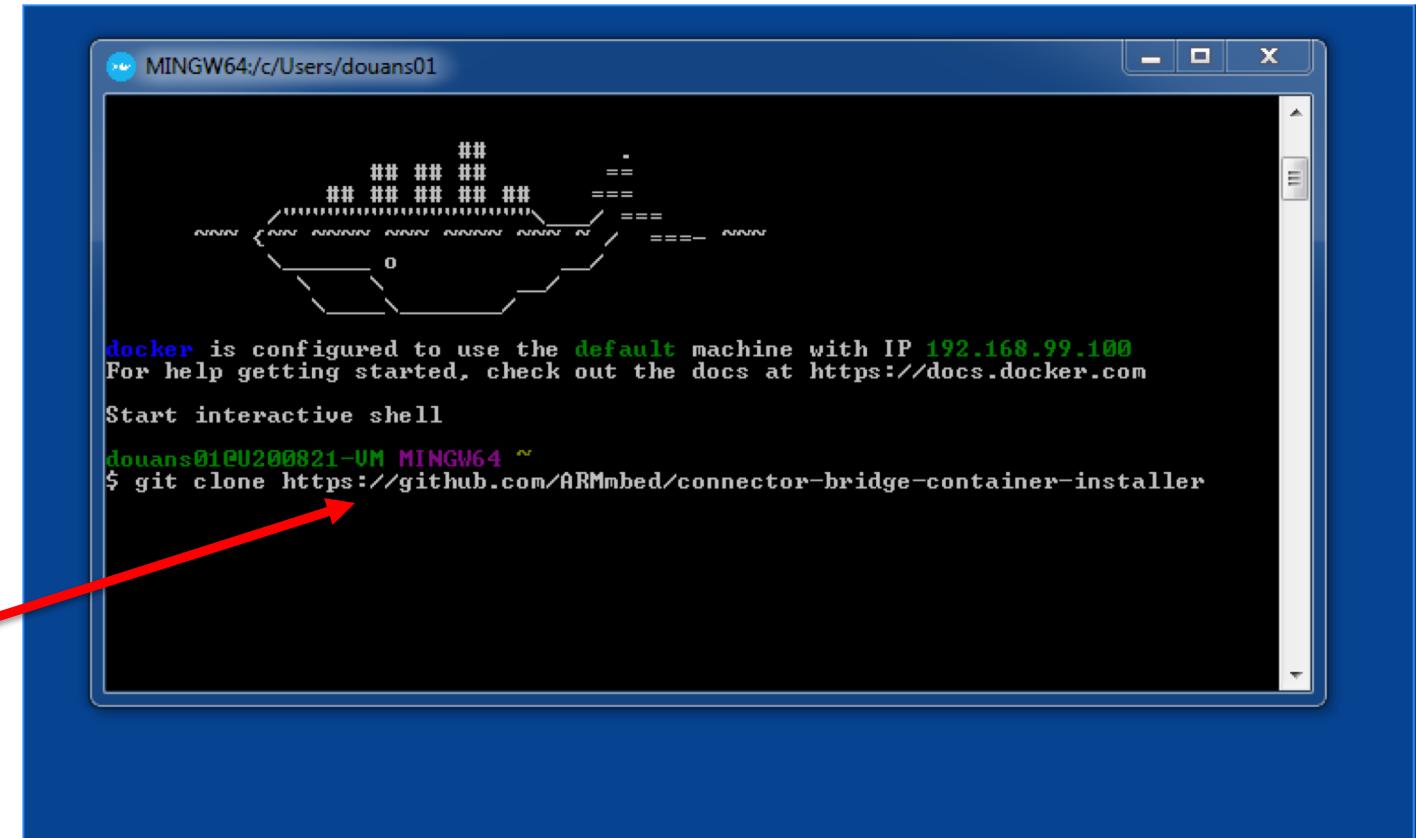
- Chrome and/or Firefox installed

Double check (Mac/Linux) :All our needed tools are installed

- Docker Engine installed (do not install the "Toolbox" version on Mac... use the native engine)
 - Docker Engine for Mac (do not install the "Toolbox" version on the mac):
 - <https://docs.docker.com/engine/installation/mac/>
 - Docker Engine for Linux:
 - <https://docs.docker.com/engine/installation/linux/>
- Suitable serial terminal installed and operational
- Chrome and/or Firefox installed
- "git" installed
- Access to a command line terminal

Import our mbed Prototype Connector Bridge Installer

- Windows: Launch the Docker QuickStart Terminal
 - It may take a few minutes to initialize if launched the first time... it will have to download additional stuff
 - Allow VirtualBox to make changes to your network interface
- Mac/Linux: Open a terminal
- Type the following “git” command (below)
- For all platforms (Windows/Mac/Linux), this is the "git" command to run:
\$ git clone <https://github.com/ARMmbed/connector-bridge-container-installer>



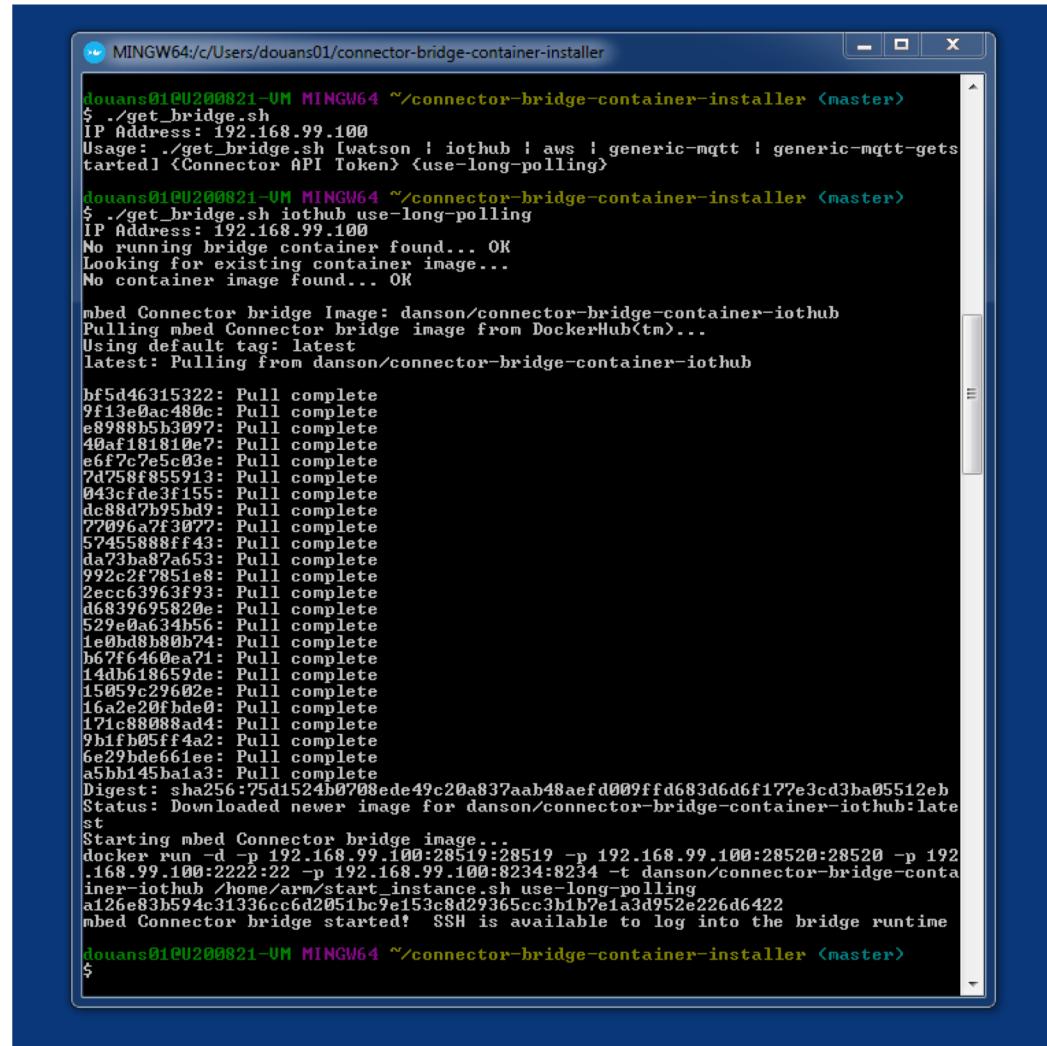
The screenshot shows a terminal window titled 'MINGW64:/c/Users/douans01'. It displays the Docker configuration message:

```
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
```

At the bottom, the command \$ git clone https://github.com/ARMmbed/connector-bridge-container-installer is visible, with a red arrow pointing to it.

Import our Prototype 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 watson use-long-polling
- Bridge container will import from DockerHub



```
MINGW64:/c/Users/douans01/connector-bridge-container-installer
$ ./get_bridge.sh
IP Address: 192.168.99.100
Usage: ./get_bridge.sh [watson | iothub | aws | generic-mqtt | generic-mqtt-gets
tarted] <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
1e0fd8b80b74: 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:late
st
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-conta
iner-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>
$
```

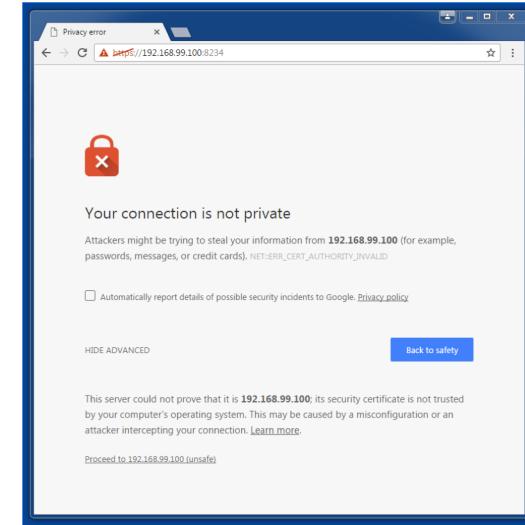
Configure our Prototype mbed Cloud Bridge (Windows)

- Your Container IP address will be:

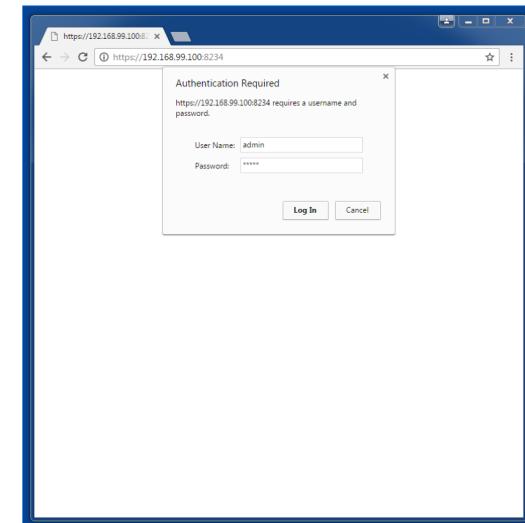
192.168.99.100

- Bring up Firefox/Chrome and go to:

<https://192.168.99.100:8234>



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



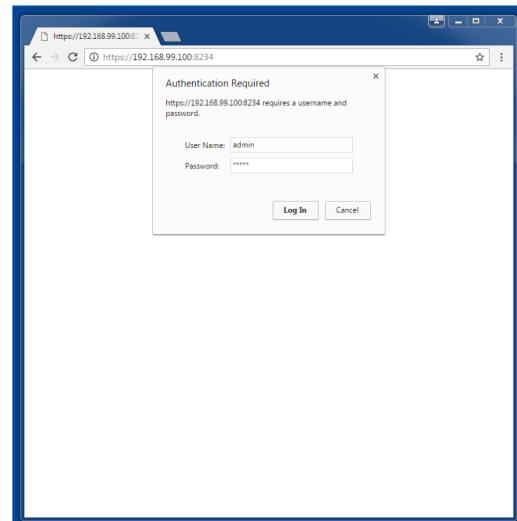
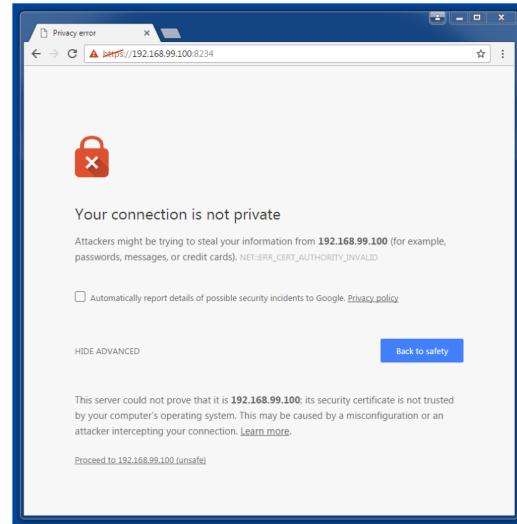
Configure our Prototype mbed Cloud Bridge (Mac/Linux)

- Your Container IP address be "localhost" if installed on your local box, otherwise, make note of it if installed remotely

- Bring up Firefox/Chrome and go to:

<https://<your container IP address>:8234>
(typically: <https://localhost:8234>)

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



Watson IoT Configuration

What we will do next...

- Create our own Watson IoT Instance within our Bluemix account
- Create our Watson IoT Application Credentials (used in the NodeRED application...)
- Create our sample Watson IoT NodeRED Application
- Bind the Watson NodeRED Application to our Watson IoT instance
- Create a mbed Cloud API Key/Token
- Configure the Prototype mbed Cloud Bridge for Watson IoT

Create our Watson IoT Instance

- Go to the Bluemix dashboard & click:
(<https://console.bluemix.net>)
- Select “Services”, then “Internet of Things”
- Press “Get Started”
- Configure your Watson IoT Instance
 - Leave “unbound”
 - Select the “Free” plan
(scroll down...)
- Select “Create”

This will create your Watson IoT instance

The image consists of five screenshots illustrating the process of creating a Watson IoT instance:

- Bluemix Dashboard:** Shows the "IBM Bluemix Apps" section with a "Create Application" button.
- Services Catalog:** Shows the "Services" menu with "Internet of Things" selected. A red arrow points to the "Internet of Things" link.
- Internet of Things Overview:** Shows the "Internet of Things" service page with a "Get Started" button.
- Catalog Page:** Shows the "Internet of Things Platform" catalog entry. It includes a description, a "Service name" input field set to "Internet of Things Platform-90", and a "Features" section. A red arrow points to the "Leave unbound" dropdown under "Connect to".
- Create Button:** A large red arrow points to the "Create" button at the bottom right of the catalog page.

Create Watson IoT Application Credentials

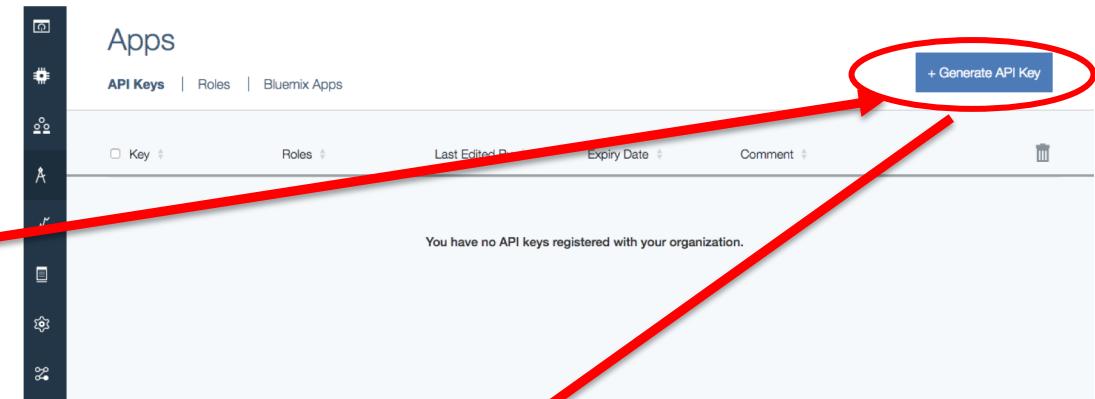
- Once your Watson IoT Instance is created, you should see the Watson IoT dashboard
- Launch the “Launch”
- Press “Apps”

The screenshot shows the 'Manage' tab of the Watson IoT Platform instance 'my-watson-iot-instance'. At the top right, there's a 'Launch' button, which is highlighted with a large red arrow. Below it, there's a 'Docs' button. The main area features a 'Welcome to Watson IoT Platform' message with a subtext about securely connecting devices. There are also links to 'Learn about Watson IoT Platform' and 'Expand using step-by-step recipes'.

The screenshot shows the 'APPS' section of the Watson IoT Platform. On the left, a sidebar lists navigation items: BOARDS, DEVICES, MEMBERS, APPS (which is highlighted with a red arrow), USAGE, RULES, SECURITY, SETTINGS, and EXTENSIONS. The main area displays four cards: 'DEVICE-CENTRIC ANALYTICS' (5 Cards, Owned by you), 'USAGE OVERVIEW' (3 Cards, Owned by you), and 'RULE-CENTRIC ANALYTICS' (6 Cards, Owned by you). At the bottom right of the sidebar, there's a '+ Create New Board' button.

Create Watson IoT Application Credentials...

- Press “Generate API Key”



- Record the API Key

Generate API Key

API Key

Authentication Token

a-bo522z-smaoymkkzn

7Pkf_tv@?K@cpO3(1k

- Record the Authentication Token

- Press “Generate” after adding a comment

Select API Role(s)

Standard Application

+ Add another role

Comment

My NodeRED Application Key

Set API key expiry

10/13/2016

- Save these two values! You will need them later...

Creating our Sample Watson IoT NodeRED Application

Please DO NOT USE Internet Explorer for this task!

- Go back to our Bluemix Dashboard: <https://console.bluemix.net>

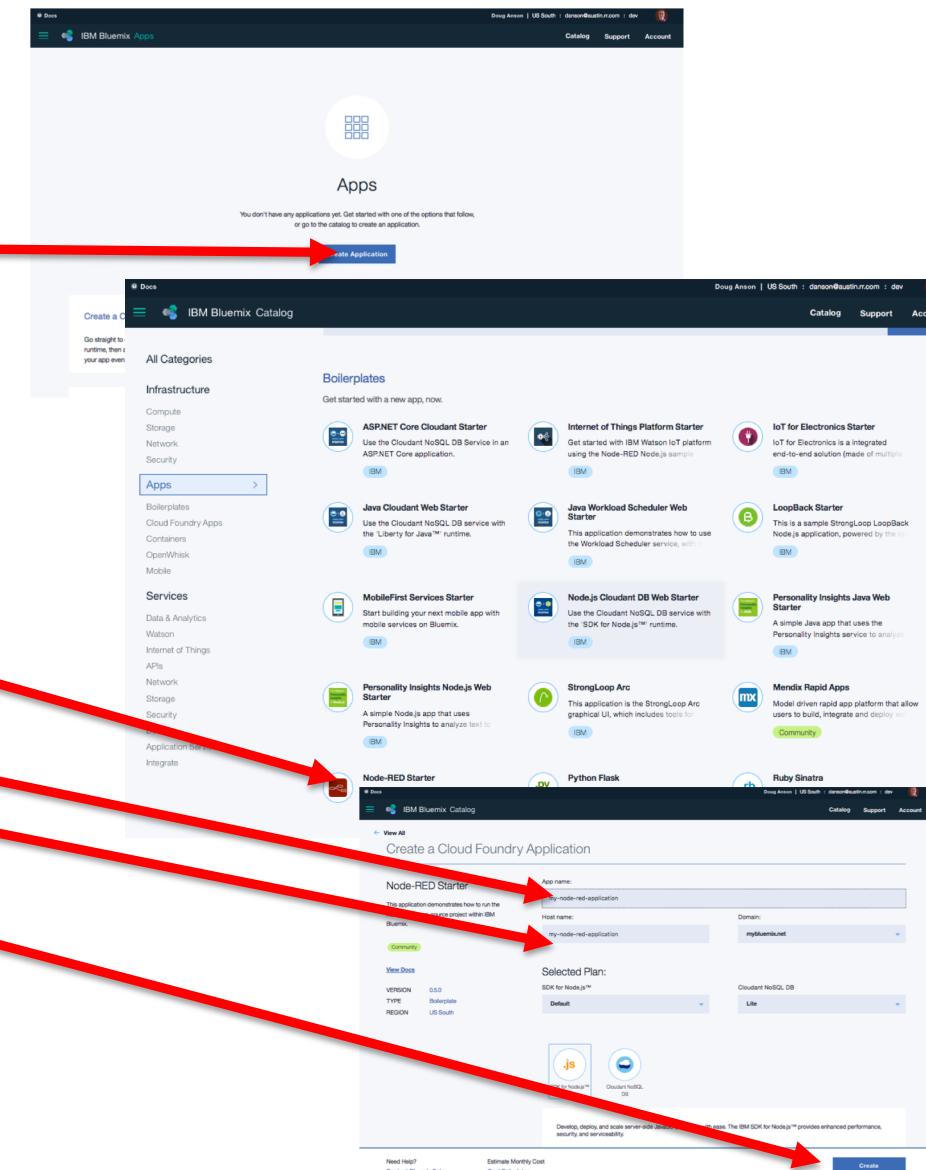
- Click on “Create Application”

- Select “NodeRED Starter”

- Enter an “App Name” (must be unique)
 - Make a note that your app URL is <app name>.mybluemix.net
 - Press “Create” to finalize creating the application

- Application will “Stage”... this will take a few minutes...

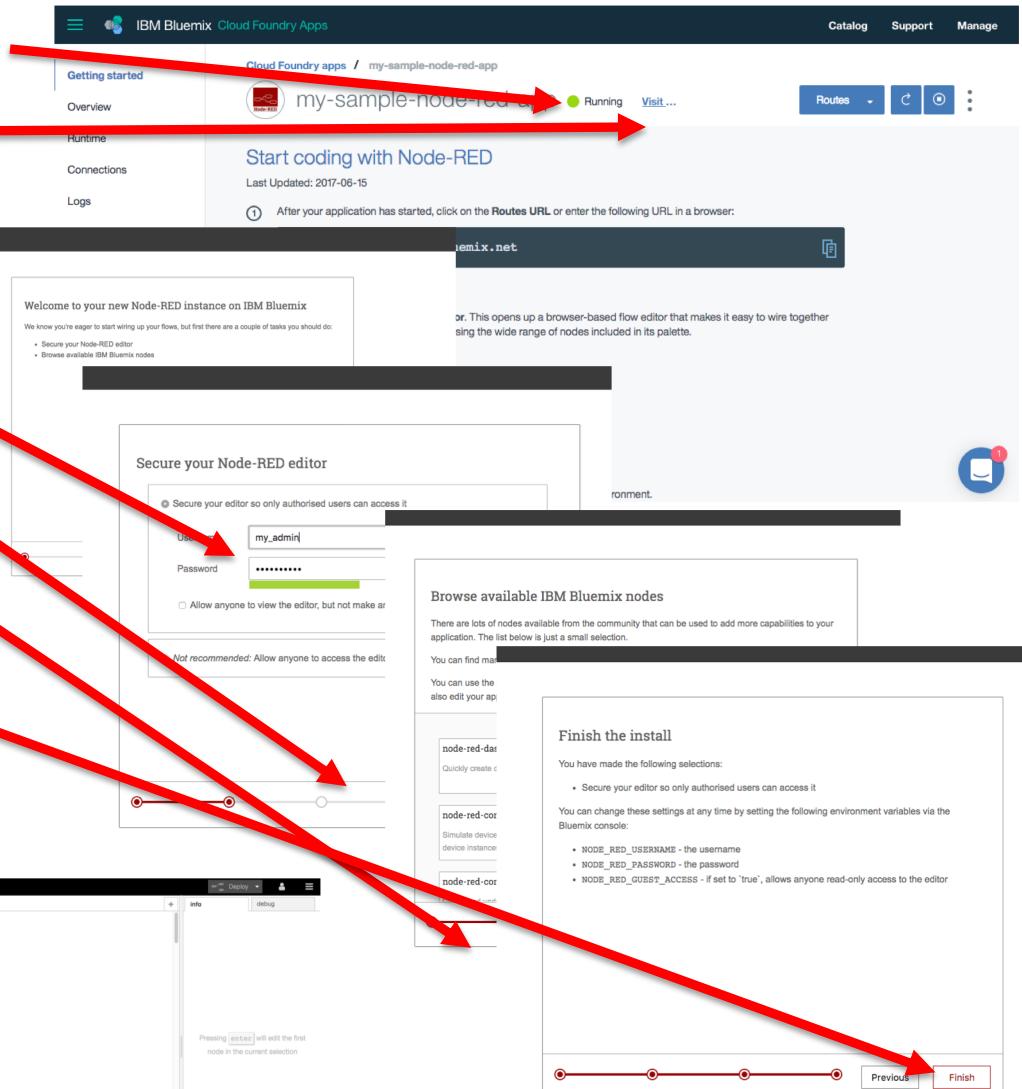
- Next we will configure the NodeRED application (next slide)



Configure Watson IoT NodeRED Application

Please DO NOT USE Internet Explorer for this task!

- Your NodeRED app should fully "stage" and report "running"
- Click on "Visit..."
- Next, you must configure your NodeRED app
 - Create an admin username/password for the dashboard
 - Press "Next"
 - Press "Next" (no additional nodes need to be added)
 - Press "Finish"
- Now, login to the dashboard



Bind the Watson NodeRED Application to our Watson IoT instance

Please DO NOT USE Internet Explorer for this task!

- Dashboard -> Apps, select your NodeRED app

IBM Bluemix Cloud Foundry Apps

Cloud Foundry apps / my-sample-node-red-app

my-sample-node-red-app • Running [Visit...](#)

Runtime

Node-RED

BUILDPACK Node-RED Starter

INSTANCES 1

MB MEMORY PER INSTANCE 512

TOTAL MB ALLOCATION 512 MB still available

Connections (1) my-sample-node-red-app-cloudantNoSQLDB

Connect new Connect existing

Routes \$0.00 \$0.00

- Press “Connect Existing”
- Click on your Watson IoT Instance
- Press “Connect”
- Your NodeRED application will “restage”... this will take awhile.
Wait for the “running” state:

IBM Bluemix Cloud Foundry Apps

Connect existing service

Services

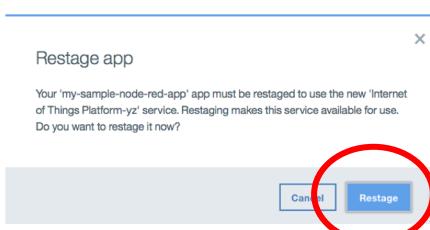
Internet of Things Platform-yz

my-watson-node-red-app-cloud

Visual Recognition-Id

Select compatible service to connect to my-sample-node-red-app

Connect



Note: keep this window open!!!

Create our mbed Cloud Access/API Token

- Navigate to the mbed Cloud Dashboard: <https://portal.us-east-1.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
- Press “copy to clipboard” – save to a file for later use

The screenshots illustrate the process of creating a new API key in the mbed Cloud. The first screenshot shows the 'Create new API key' dialog with 'My New API Key' entered in the 'Key name' field and 'Developers' selected in the 'Groups' dropdown. The second screenshot shows the 'API keys' list with the new key added. The third screenshot shows the 'Create API key' confirmation page with the 'API key' copied to the clipboard.

Screenshot 1: Create new API key dialog

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

Screenshot 2: API keys list

Screenshot 3: Create API key confirmation

Configure the mbed Cloud Bridge for Watson IoT...

- In a browser - Mac/Linux: <https://localhost:8234> Windows: <https://192.168.99.100:8234>

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

- Enter your Watson API Key, Press “Save”

- Enter your Watson Authentication Token, Press “Save”

- Ensure you have the right API endpoint

- Cloud: api.us-east-1.mbedcloud.com
- Connector: api.connector.mbed.com

- Press “RESTART”

- Your mbed Cloud Prototype Bridge is now configured for Watson IoT

The screenshot shows the 'mbed Connector Bridge Configuration' interface. It includes sections for 'Connector Bridge Control' (with a 'RESTART' button) and 'Connector Bridge Configuration'. The configuration section contains the following fields:

Setting	Value	Action
mds_api_token	mbed_connector_api_token_goes_here	SAVE
iotf_api_key	a-__ORG_ID__-__ORG_KEY__	SAVE
iotf_authentication_token	iotf_authentication_token_goes_here	SAVE
mds_address	api.connector.mbed.com	SAVE
iotr_legacy_bridge	false	SAVE
mds_enable_long_poll	true	SAVE

Below this is the 'Configurator Admin Settings' section:

Setting	Value	Action
admin_username	admin	SAVE
admin_password	admin	SAVE
default_port	8234	SAVE
keystore_password	arm1234	SAVE

Status Check

So far we've completed the following

- Created our own Watson IoT Instance within our Bluemix account
- Created our Watson IoT Application Credentials (used in the NodeRED application...)
- Created our sample Watson IoT NodeRED Application
- Bound the Watson IoT Application to our Watson IoT instance/service
- Created a mbed Cloud API Token
- Configured the Prototype mbed Cloud Bridge for Watson IoT

Next, we will Import our NodeRED flow sample and restart the Endpoint...We should then see device telemetry flowing from the device, through mbed Cloud, through the Bridge, and into Watson IoT!

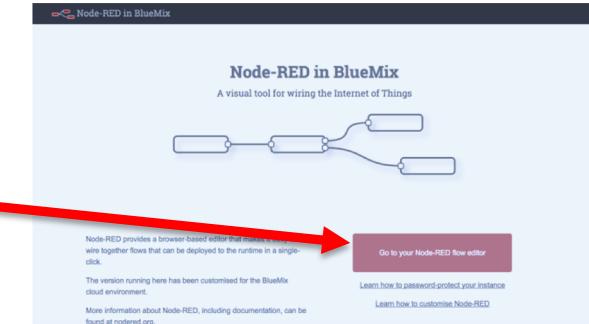
Importing the NodeRED Flow Example

Copy the Sample NodeRED Flow JSON

- Go back to your Online IDE workspace
- Go to your “mbed-cloud-sample” project
- Double-click on NODEFLOW.txt
- Copy all of that file...

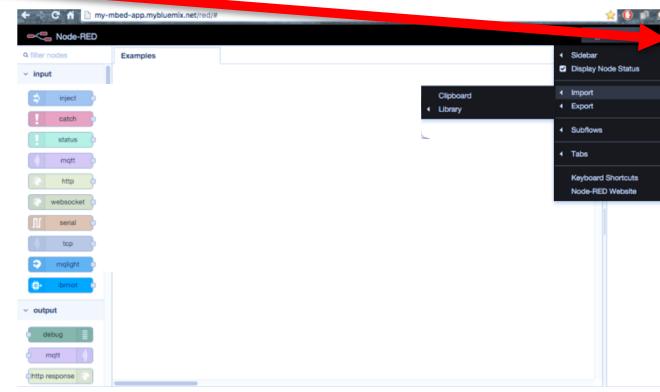
Import the NodeRED Flow

- Navigate to the URL that we recorded earlier for your Watson IoT NodeRED Application (i.e. `http://<app name>.mybluemix.net`)



- Select “go to your NodeRED flow editor”

- Select the menu (far right)



- Select “Import”

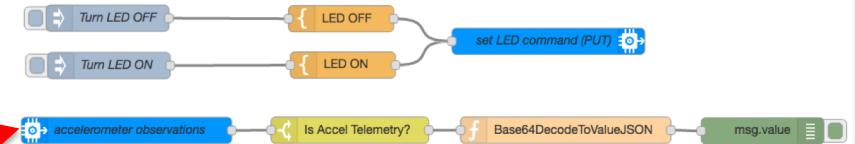
- Select “Clipboard”

- Paste the entire contents of the JSON code you copied in the previous slide into the “paste nodes here” window... then press “OK”.

- If you have trouble copy/pasting the NODEFLOW.txt file contents, try copying it from <https://github.com/ARMmbed/mbed-cloud-sample> (NODEFLOW.txt is listed there...click & copy...)

Your new Watson IoT Application Node Flow: Configure it

- Configure your NodeRED flow input and output nodes (Blue) and link them to your Watson IoT instance



- Click on the observation node

Edit ibmiot in node

Authentication: API Key
API Key: MyWatsonIoT

Input Type: Device Event

Device Type: mbed-endpoint

Device Id: cc69e7c5-c24f-43cf-8365-8d23bb01c

Event: observation

Format: json

Name: connector-bridge source (observations)

Use the Input Type property to configure this node to receive Events sent by IoT Devices, Commands sent to IoT Devices, Status Messages referring to IoT Devices, or Status Messages referring to IoT Applications
Check the info tab, to get more information about each of the fields

Ok Cancel

- Click on the “edit” button

- Provide a name

- Insert your Watson API Key from slide 28

- Insert your Watson Auth Token from slide 28

- Press “Update” to save

Edit ibmiot config node

* Ethernet

Name: MyWatsonIoT

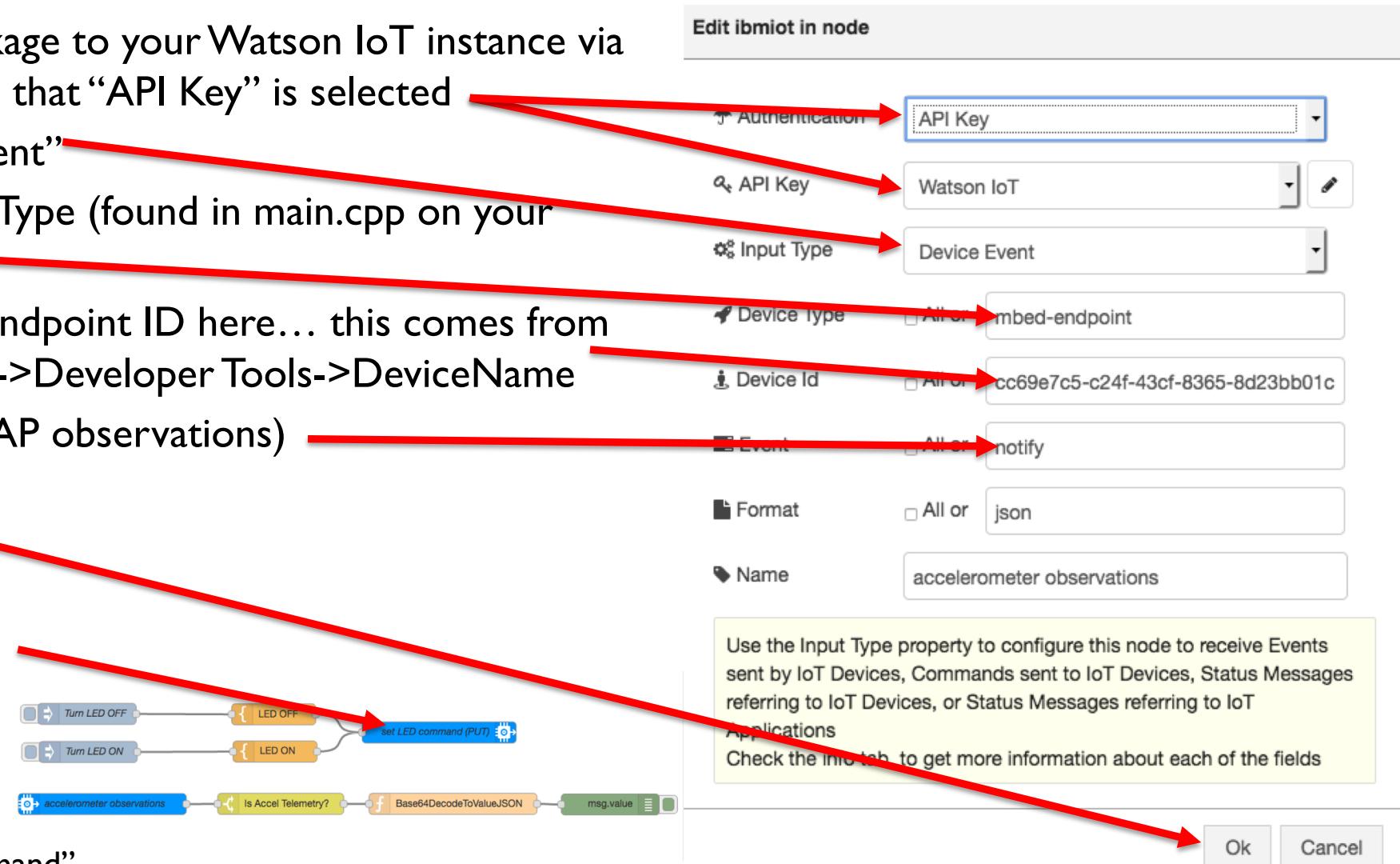
API Key: a-uzttt4-28lb0hgffy

API Token:
5 nodes use this config

Delete Update Cancel

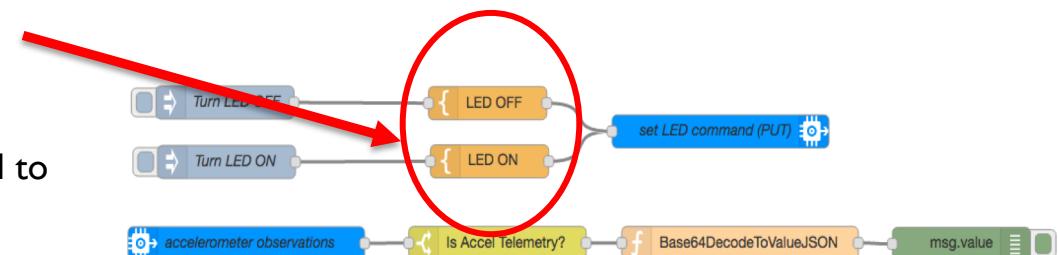
Your new Watson IoT Application Node Flow: Configure it...

- Now, select the new linkage to your Watson IoT instance via the drop down here and that “API Key” is selected
- Input Type is “Device Event”
- You can filter by Device Type (found in main.cpp on your endpoint) or “all”
- Ensure you have your endpoint ID here... this comes from mbed Cloud Dashboard->Developer Tools->DeviceName
- Event is “notify” (i.e. CoAP observations)
- Press “OK”
- Edit the blue PUT node
 - API Key Auth
 - Select same API Key
 - Device Type, Device ID
 - Event is “PUT” (all caps!)
 - Input Type is “Device Command”



Your new Watson IoT Application Node Flow: Configure it...

- Lastly, we have to look at the remaining “Orange” command builder nodes and update them as well
- For both LED OFF/ON nodes, select and note the JSON payload created. “deviceld” needs to be the value you have for your **Device Name** (from mbed Cloud Dashboard→ Developer Tools)
 - “deviceld” will be on the right-hand side in the JSON structure... you have to scroll to the right to see it... You can also enlarge the edit window...
- Locate MBED_ENDPOINT_NAME_Goes_Here and replace it with your actual **Device Name** value ... press “Done” when complete.
- FYI, Each LED ON/OFF has a Base64 encoded value
 - It’s a “1” for ON, “0” for OFF... each must be Base64 encoded
- Press the red “Deploy” button in the upper right hand side of your NodeRED console



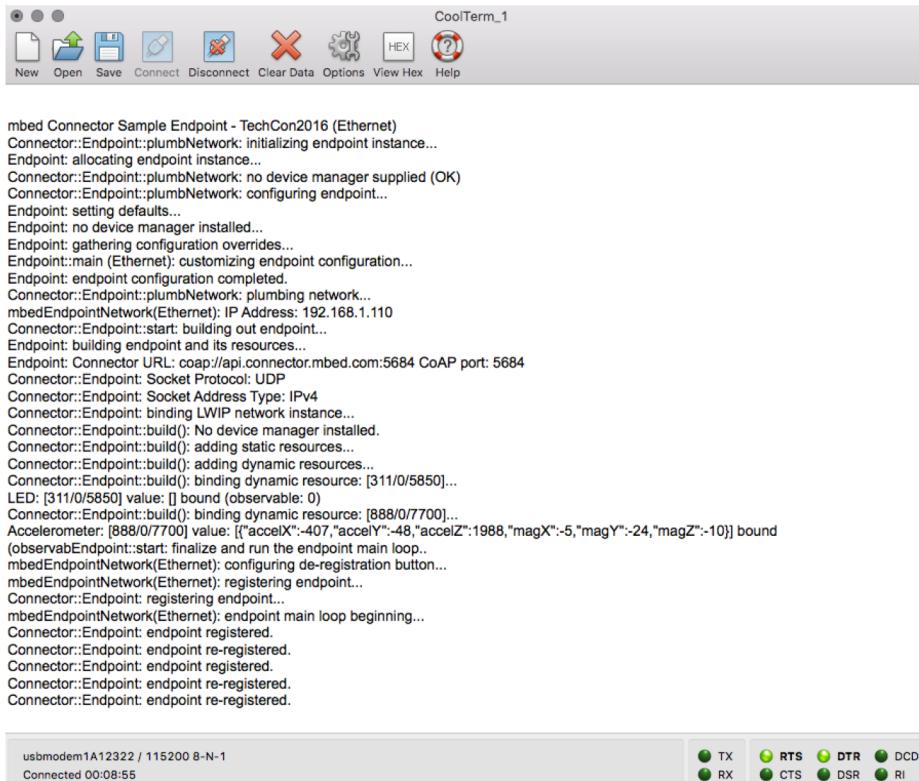
Putting it all Together

- Press the “reset” button on your mbed device (next to the USB port)
- On your NodeRED Editor, select the “debug” window

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”



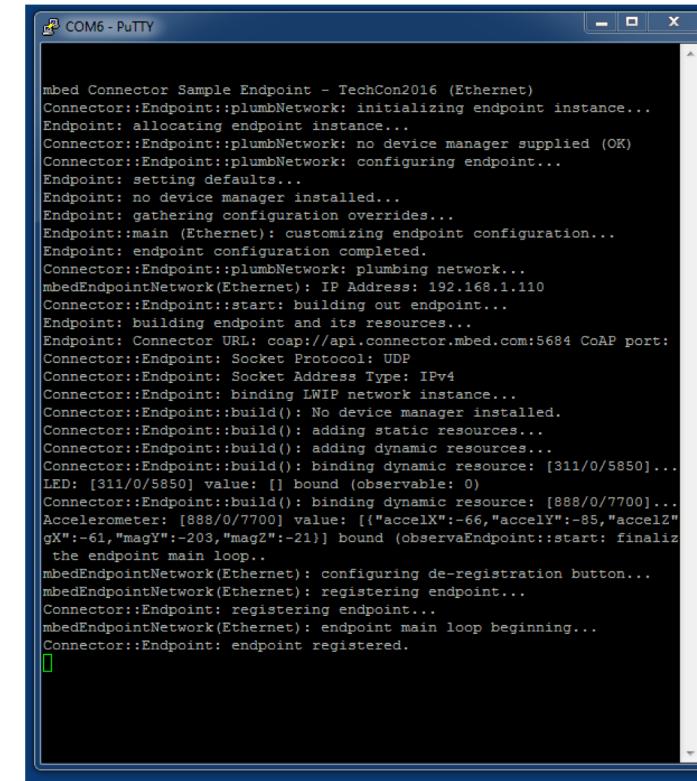
CoolTerm_1

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.
```

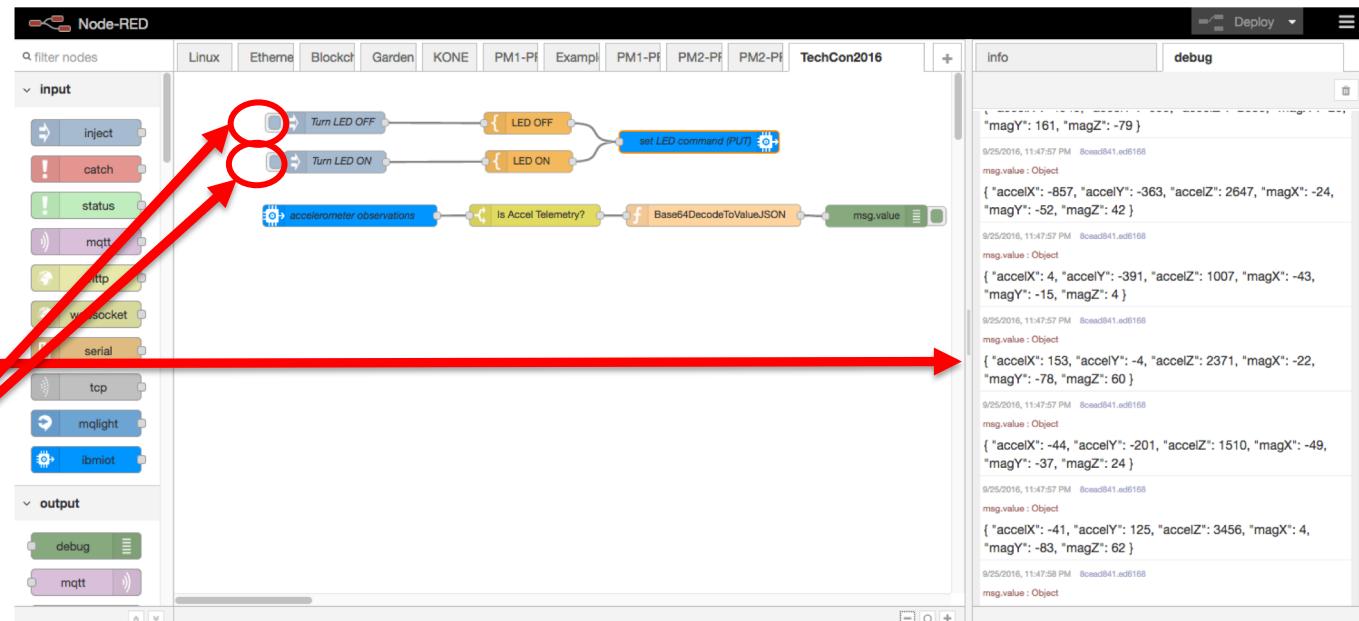
Putting it all Together: NodeRED debug info...

- Navigate to your Watson IoT application and NodeRED flow editor

- Examine the debug window
 - Shake your K64F!

- You should see output similar to this... telemetry in Watson IoT

- You can toggle your LED On and off (left-most block) by clicking each of these... look in the serial terminal for output from the endpoint



Putting it all Together: Check Watson IoT Devices

- Go to the last window you had open from slide 27
- Select "Devices"

The screenshot shows the 'Extensions' section of the IBM Watson IoT Platform. It lists four extensions: 'Single Sign On' (Status: Not Configured), 'Email' (Status: Not Configured), 'ARM mbed Connector' (Status: Configured), and 'Historical Data Storage' (Status: Not Configured). Each extension has a 'Setup' button.

- The bridge automatically creates Watson IoT devices for you

The screenshot shows the 'Devices' page with a single result. The table has columns: Device ID, Device Type, Class ID, Date Added, and Location. The first row shows a device with Device ID 'cc69e7c5-c24f-43cf-8365-8d23bb01c707', Device Type 'mbed-endpoint', Class ID 'Device', Date Added 'Oct 31, 2016 3:58:23 PM', and Location ' '. A red circle highlights the Device ID column for the first row.

Device ID	Device Type	Class ID	Date Added	Location
cc69e7c5-c24f-43cf-8365-8d23bb01c707	mbed-endpoint	Device	Oct 31, 2016 3:58:23 PM	

Putting it all Together: Check Watson IoT Devices

- Select your device

The screenshot shows the Watson IoT Platform's Devices dashboard. A red arrow points from the 'Select your device' bullet point to the device list. The list displays one device entry:

Device ID	Device Type	Class ID	Date Added	Location
cc69e7c5-c24f-43cf-8365-8d23bb01c707	mbed-endpoint	Device	Oct 31, 2016 3:58:23 PM	

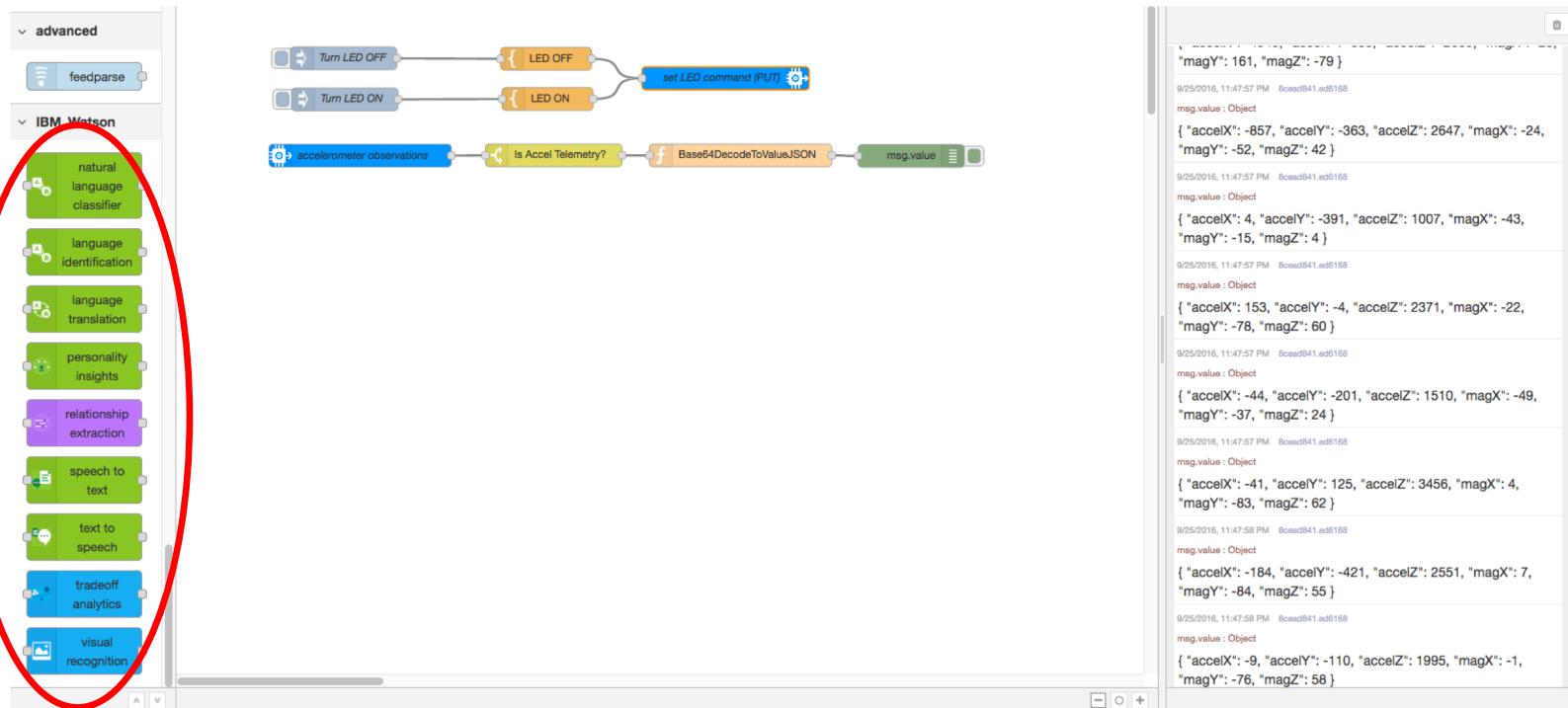
- You can watch CoAP
notify events occurring
via your bridge
(observations)

The screenshot shows the 'Device cc69e7c5-c24f-43cf-8365-8d23bb01c707' details page. A red arrow points from the 'Observations' bullet point to the 'Recent Events' section. This section lists three notifications:

Time Received	Format	Message
Oct 12, 2016 1:19:51 PM	json	notify
Oct 12, 2016 1:19:52 PM	json	notify
Oct 12, 2016 1:19:53 PM	json	notify

Ah Ha!

This is SUPER COOL



- CONGRATS! You have now finished getting mbed device data into IBM Watson IoT.
- We can deliver mbed device data into Watson IoT analytics via our NodeRED flow!
- The bigger challenge remains... What can/do you do with your data telemetry?

Summary

We have completed the following – congratulations!

- Created our Bluemix mbed environments and PC tool setup
- Imported our mbed endpoint, customized, installed, and ran it
- Created our own Watson IoT Service Instance and NodeRED application
- Configured our Prototype mbed Cloud Bridge for Watson
- Imported a NodeRED flow and customized it
- Examined live telemetry and some bi-directional capabilities of our bridge

Great JOB! Thanks for your time.