

# Workshop: Connecting IoT devices to the cloud with Google Cloud and mbed Cloud

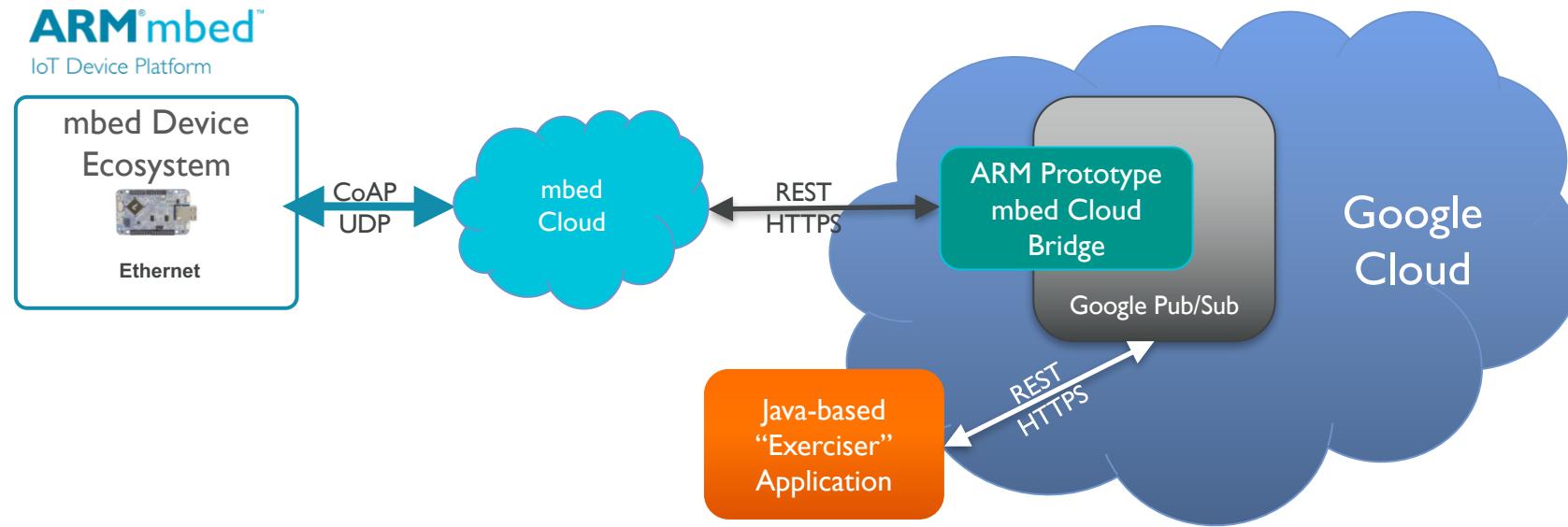
**ARM**

Doug Anson  
Solutions Architect / IoT BU / ARM

Brian Daniels  
Applications Engineer / IoT BU / ARM

Dec / 03 / 2017 – v2.1

# What will we build in this Workshop?



- Connect your mbed device into Google Cloud through mbed Cloud and ARMs Prototype Google Cloud Bridge
  - Create a simple mbed device and connect it to mbed Cloud
  - Create an instance of ARM's Prototype Google Cloud bridge and bind it to your Google and mbed Cloud accounts
  - Exploration of the device data telemetry using a java-based “exerciser” application utilizing Google’s Pub/Sub APIs

# Workshop: Let's get started!

- Create and setup your Google account (should be completed prior to workshop)
- Install the necessary tools/drivers PC/Mac/Linux (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...

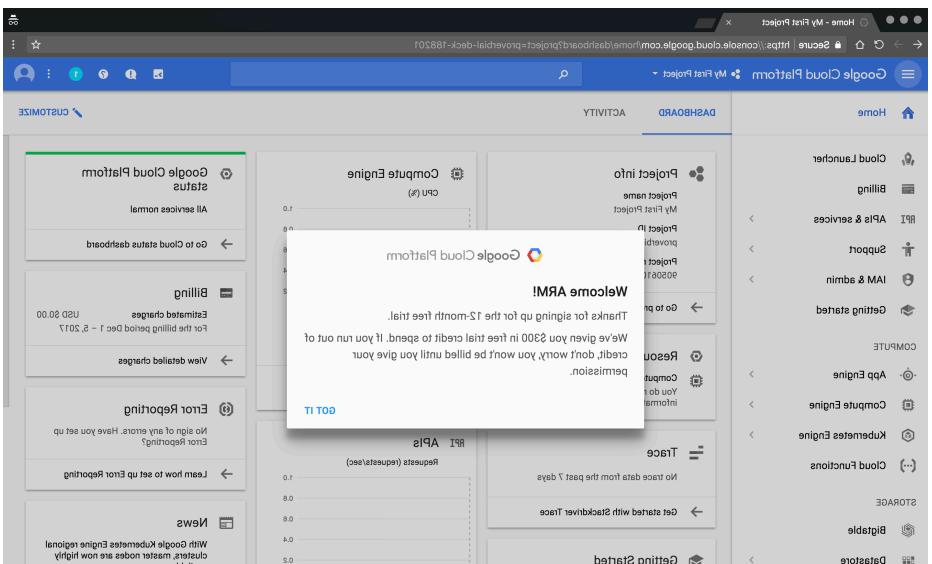
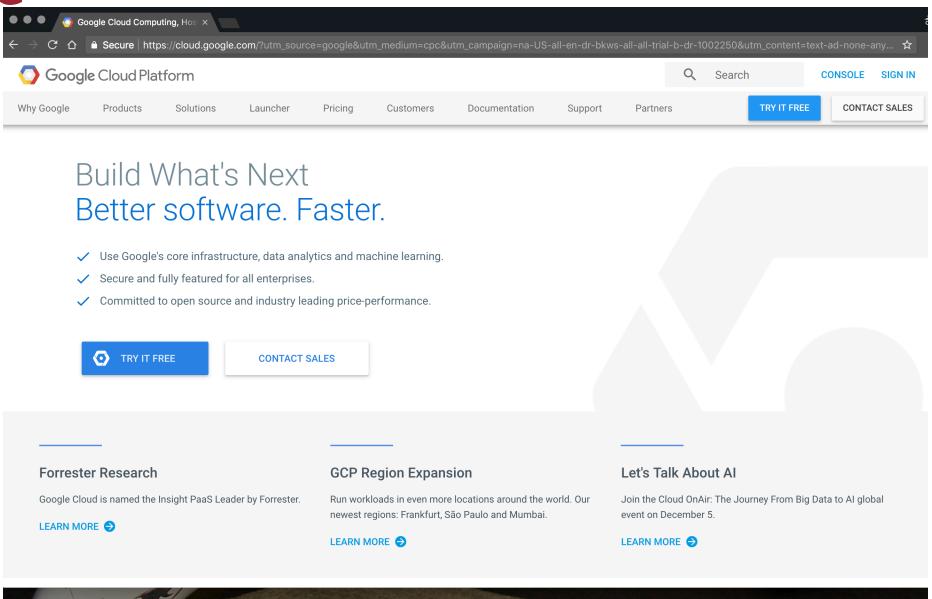
Quick Links: Visit [https://github.com/ARMmbed/anson\\_workshop\\_google\\_cloud\\_2017](https://github.com/ARMmbed/anson_workshop_google_cloud_2017)

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

# Create our Google Cloud Account

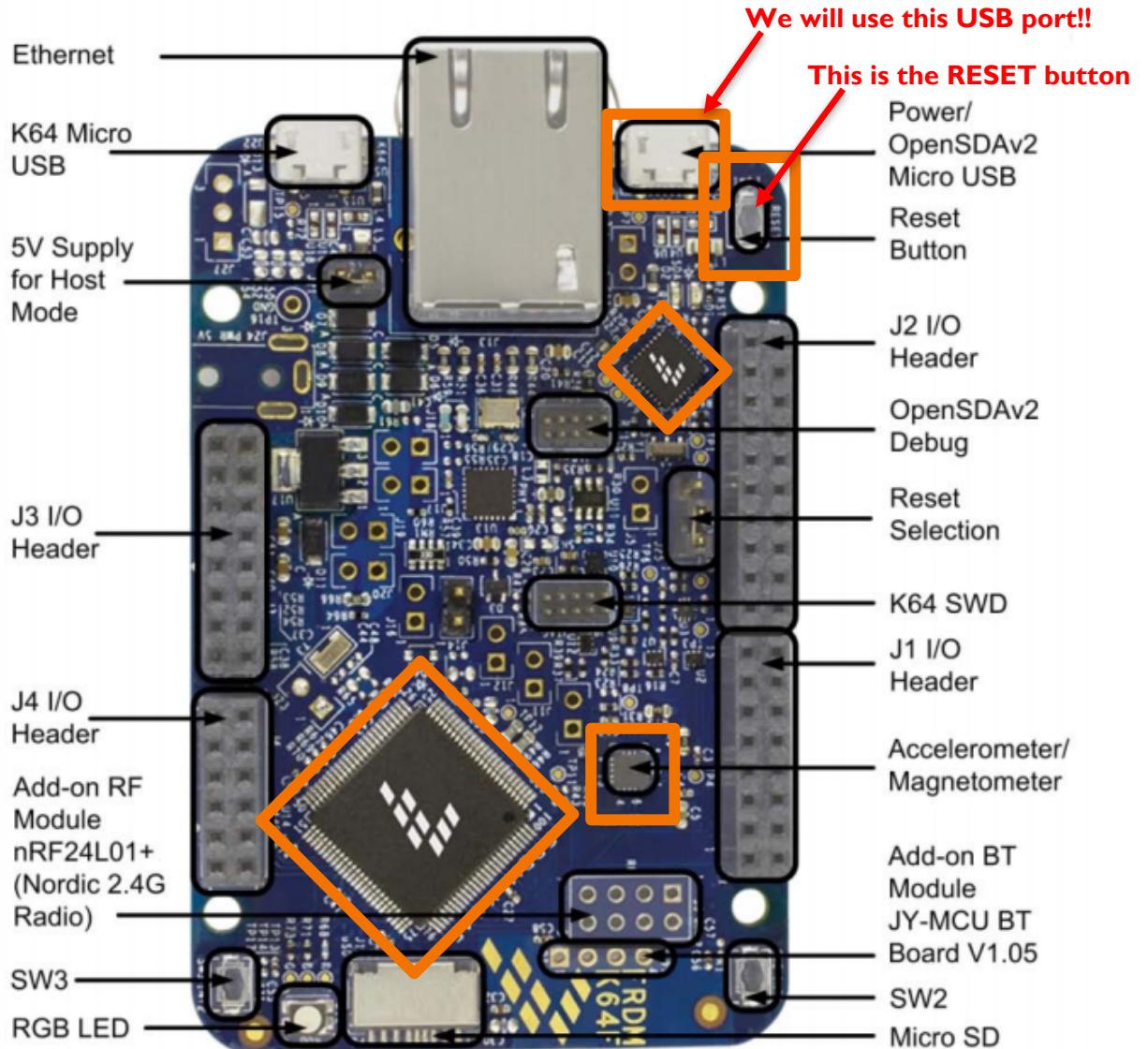
- Navigate to the Google Console:  
<https://cloud.google.com>
  - Press "Try It Free"
  - Complete the signup process (you'll need to supply a credit card – but the trial is for 1 year)
  - You'll then be at the main dashboard

# Prior to Workshop



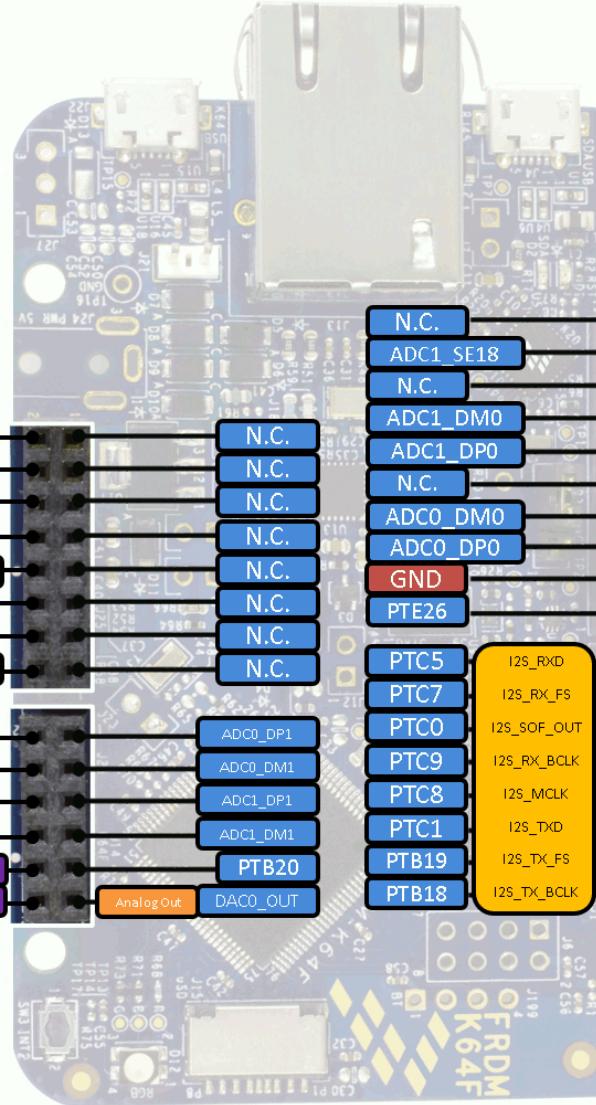
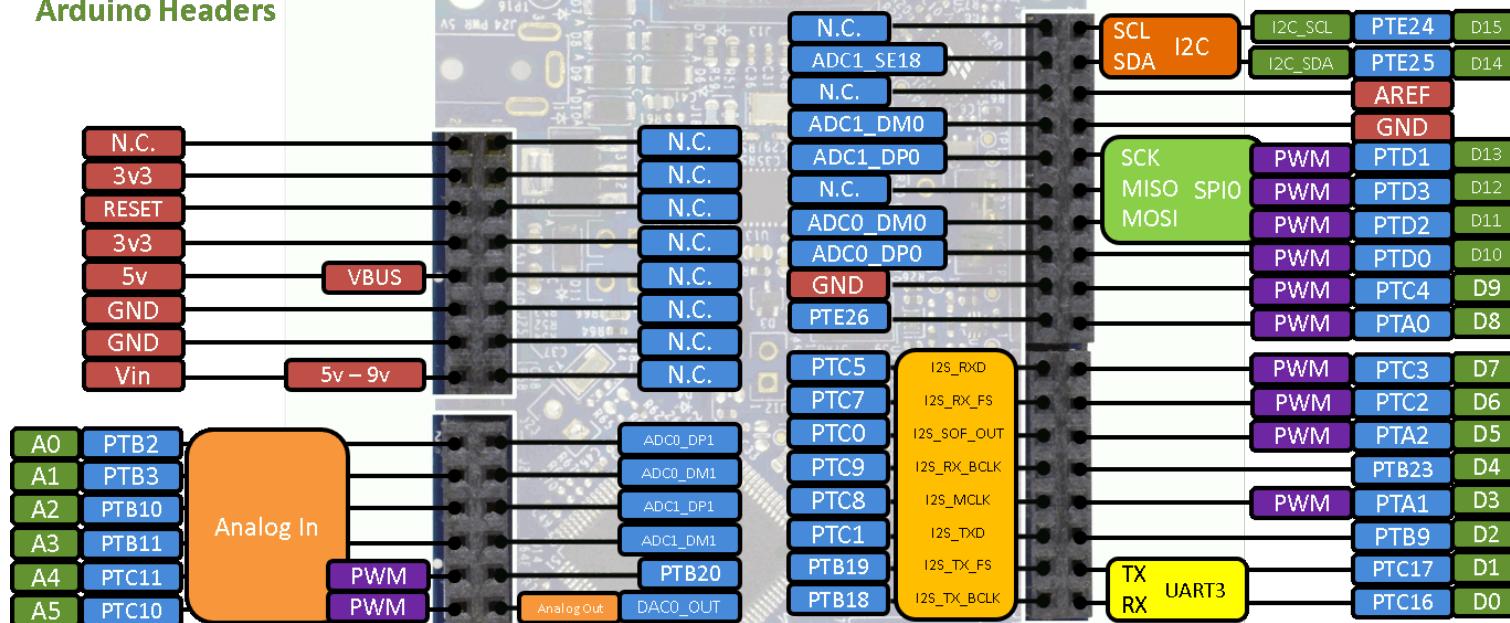
# NXP- FRDM-K64F Overview

- **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





**freescale™**  
FRDM-K64F  
Arduino Headers



ARM  
mbed  
enabled

# Install the Necessary Tools

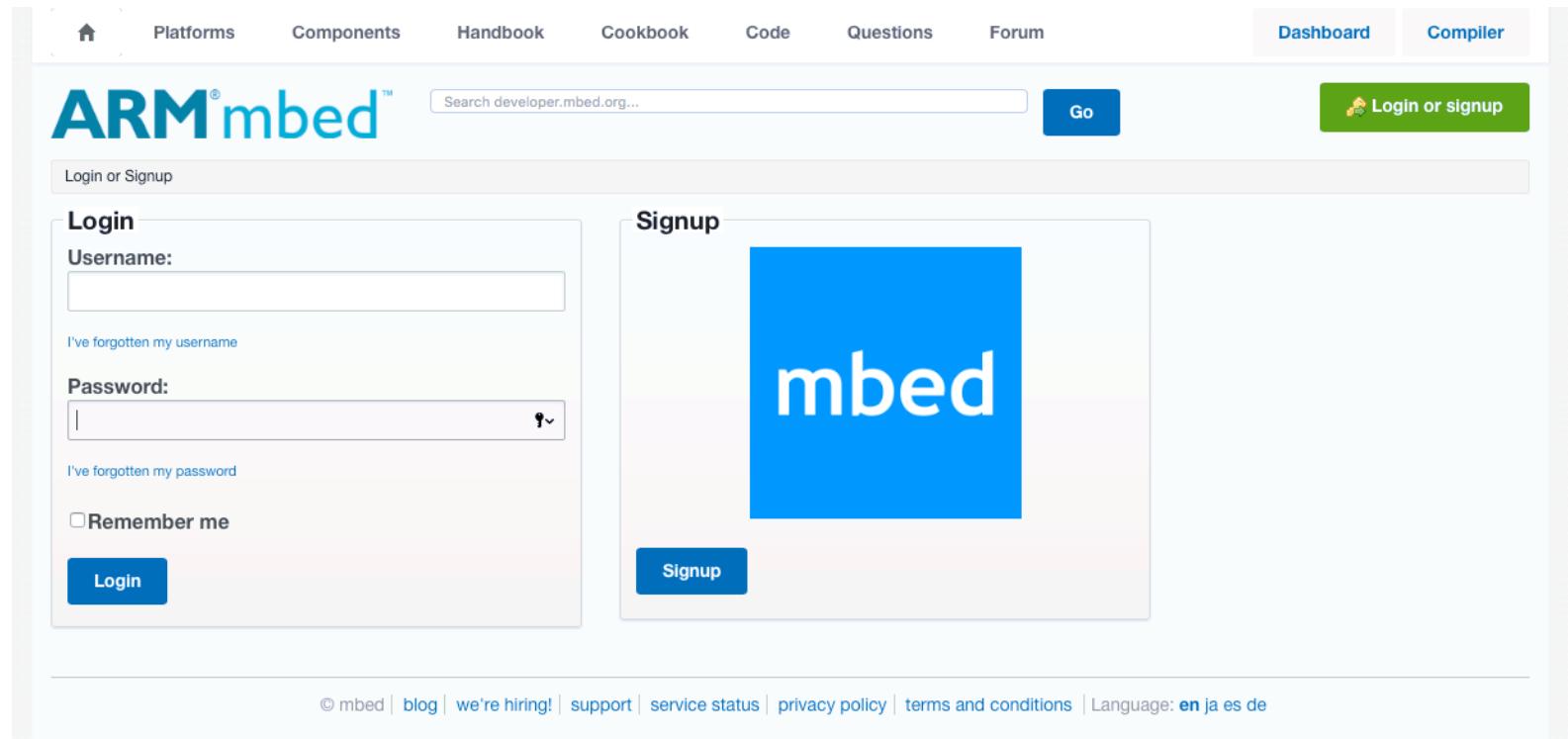
- **Windows IMPORTANT:** Install the mbed USB Serial driver -  
[https://developer.mbed.org/media/downloads/drivers/mbedWinSerial\\_16466.exe](https://developer.mbed.org/media/downloads/drivers/mbedWinSerial_16466.exe)
  - Insert the USB cable and mbed device BEFORE running the Serial Driver installation...
  - 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: <https://github.com/docker/toolbox/releases/tag/v1.12.2>
- Docker Engine installed on your Mac - <https://docs.docker.com/engine/installation/mac/> (*not* Toolbox)
- Docker Engine installed on your Linux - <https://docs.docker.com/engine/installation/linux/>
- Mac/Linux: install "git" (mac: <https://git-scm.com/download/mac>, Ubuntu: use "apt-get install git")
- Mac Serial Terminal - [http://freeware.the-meiers.org/CoolTerm\\_Mac.zip](http://freeware.the-meiers.org/CoolTerm_Mac.zip)
- Linux Serial Terminal - [http://freeware.the-meiers.org/CoolTerm\\_Linux.zip](http://freeware.the-meiers.org/CoolTerm_Linux.zip)

Prior to  
Workshop

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

# Create Your mbed Account

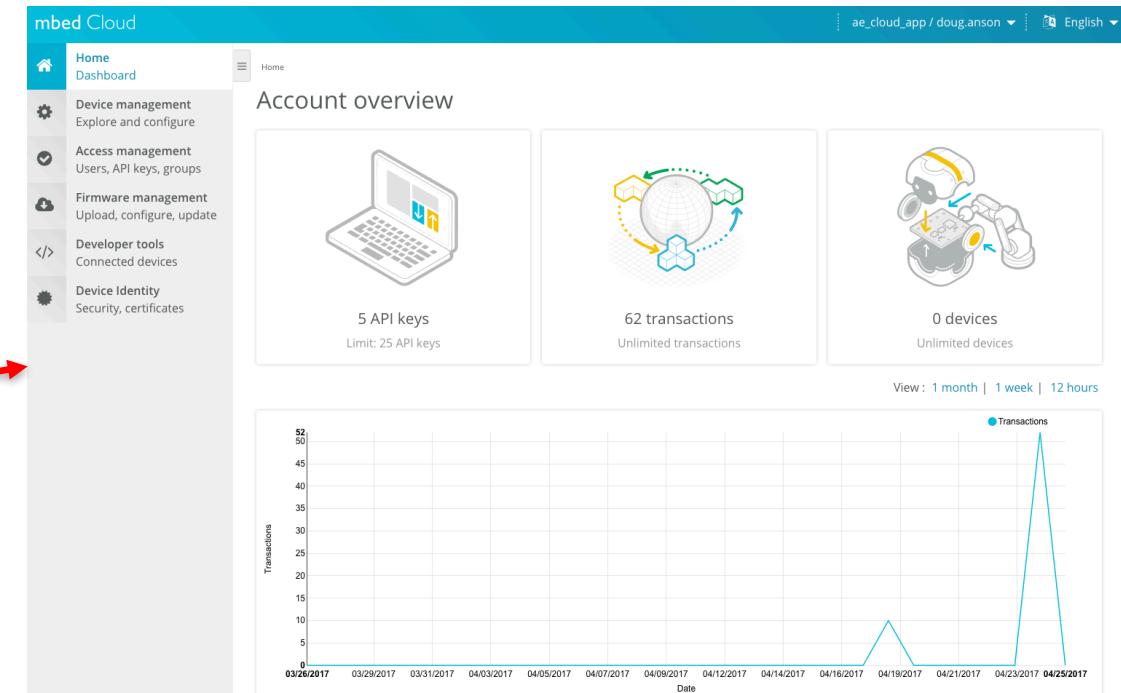
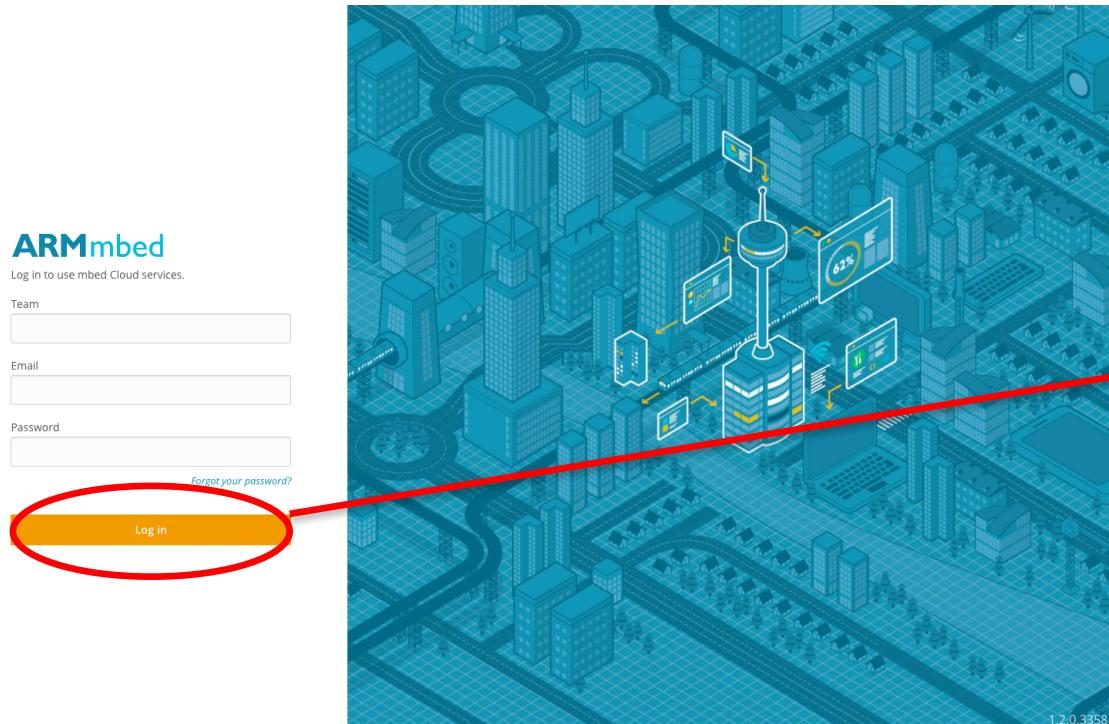
- Navigate to: <https://os.mbed.com/>
- Create an Account



# Create Your mbed Cloud Account...

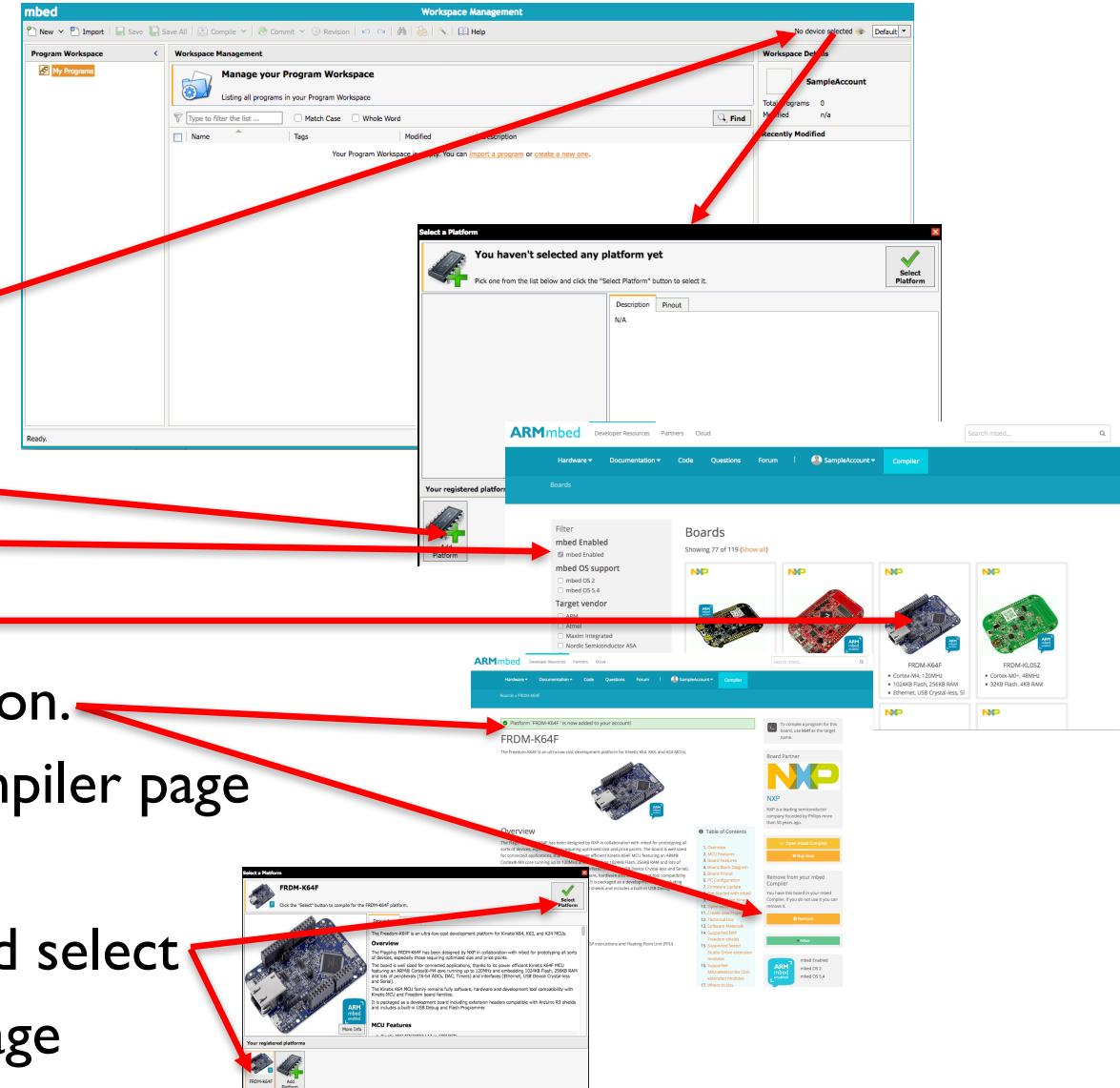
Prior to  
Workshop

- 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://developer.mbed.org>



- Select the “Compiler” page
- Click on “No device selected”
- Click on “Add Device”
- Check “mbed Enabled”
- Click on the “FRDM-K64F”
- Click on “Add to Compiler”... note addition.
- Dismiss all windows... go back to the compiler page
- Click on “No device selected”
- Now, select the “FRDM-K64F” to add and select
- Dismiss and confirm on your compiler page

# Import our K64F Endpoint Project

- Go to <https://developer.mbed.org>

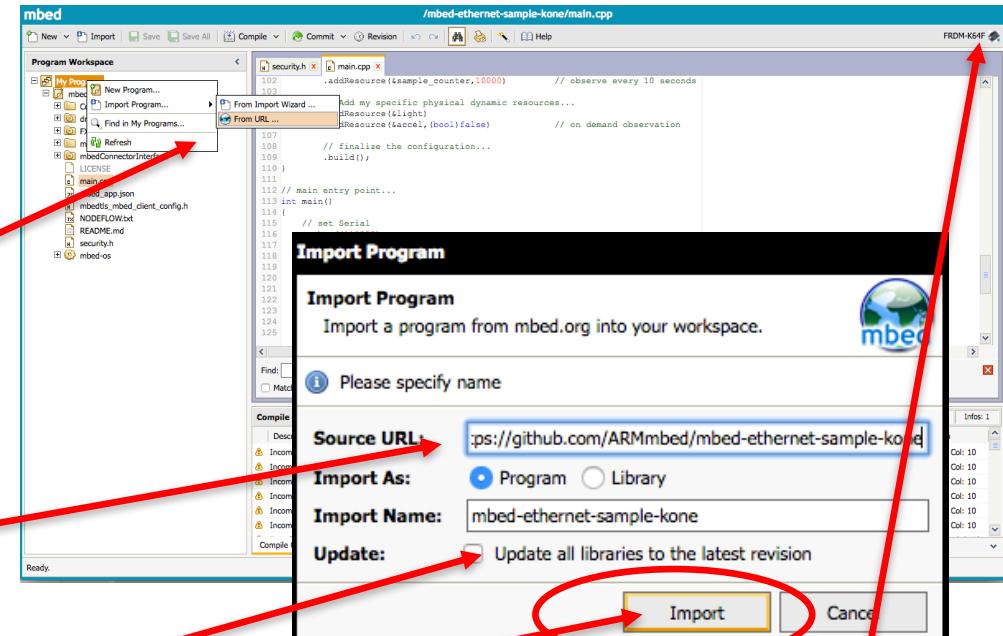
- Select the “Compiler” page

- 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



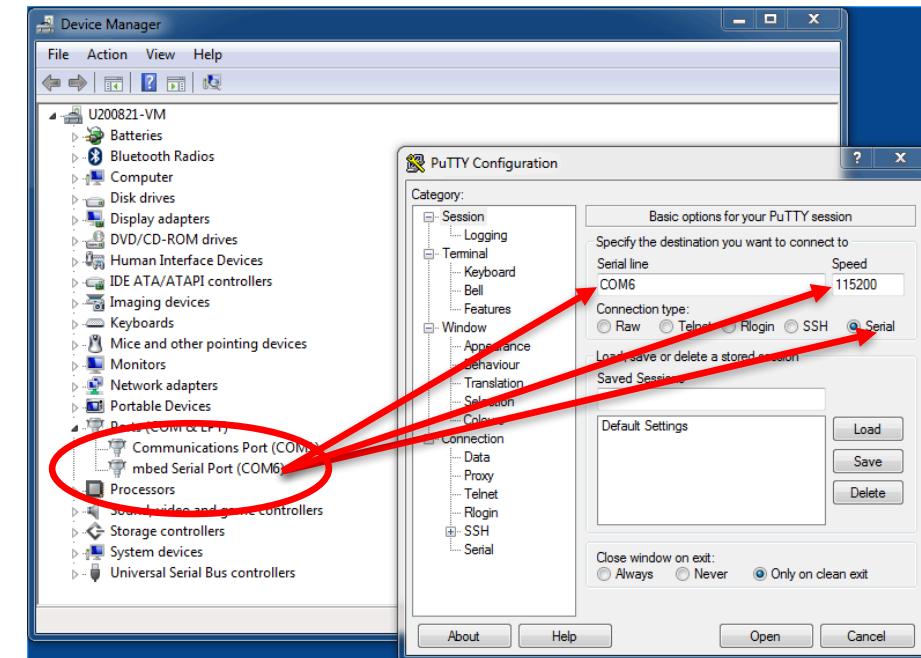
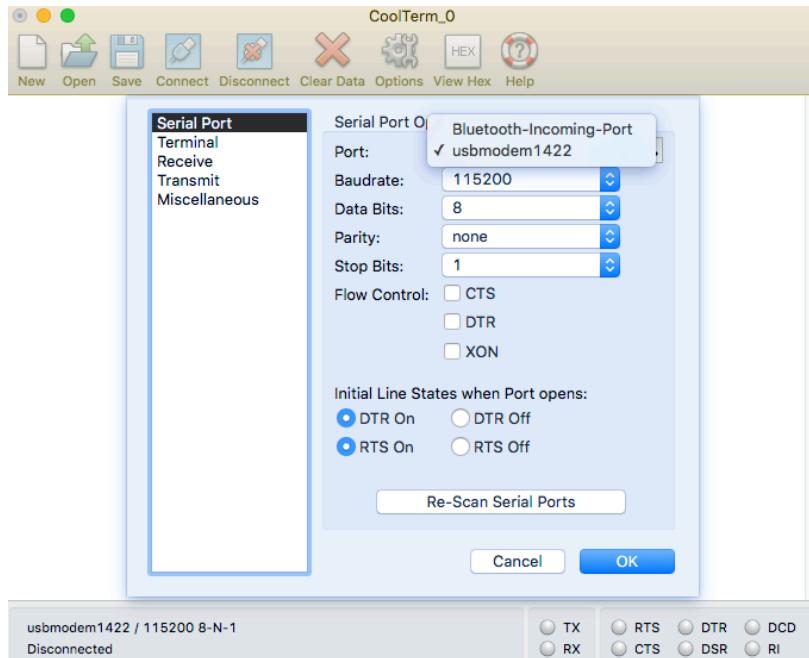
# Set Provisioning Credentials in Your Endpoint Code

- 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...

The screenshot shows the mbed online IDE interface. The 'Program Workspace' pane on the left lists files like `main.cpp`, `security.h`, and `accelerometer.h`. The 'Code Editor' pane on the right contains the source code for `main.cpp`. A red arrow originates from the 'Replace' button in the code editor's search bar and points towards the bottom right corner of the screen, where a status bar displays the message 'Compiling successful for program: mbed-ethernet-sample-kone'.

# 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 screenshots show the output of running endpoint code on a K64F board. The log messages include:

```
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::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::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": -407, "accelY": -48, "accelZ": 1988, "magX": -5, "magY": -24, "magZ": -10}] bound
(observeEndpoint::start: finalizing and run the endpoint main loop.
mbedEndpointNetwork(Ethernet): configuring de-registration button...
mbedEndpointNetwork(Ethernet): registering endpoint...
Connector::Endpoint::register endpoint...
Connector::Endpoint::re-register 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.
```

At the bottom of the CoolTerm window, there is a status bar with the text "usbmodem1A12322 / 115200 8-N-1" and "Connected 00:08:55". Below the status bar, there is a row of seven small circular indicators labeled TX, RTS, DTR, DCD, RX, CTS, DSR, and RI.

# 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 Google Prototype mbed Cloud Bridge...

# ARM Google mbed Cloud Bridge Setup

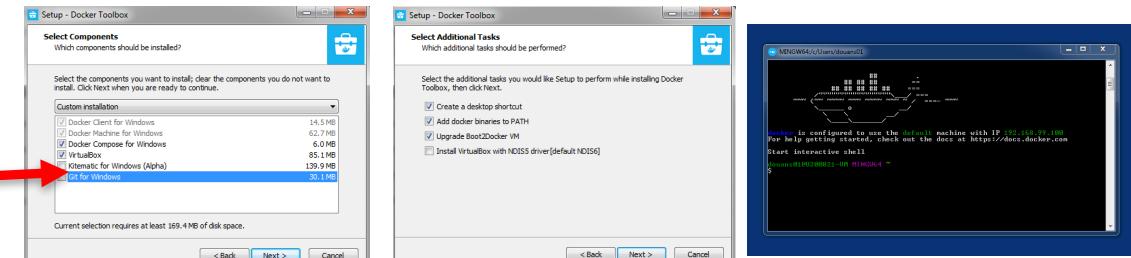
# What we will do next...

- Ensure that we have the necessary pre-requisites setup on our PC
- Create our mbed Cloud API Key/Token
- Create our Google Service Account & Authentication JSON
- Enable the Pub/Sub API
- Import our Prototype mbed Cloud Bridge instance as a Docker image into our Docker runtime
- Configure our Prototype mbed Cloud Bridge for Google Cloud

# 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

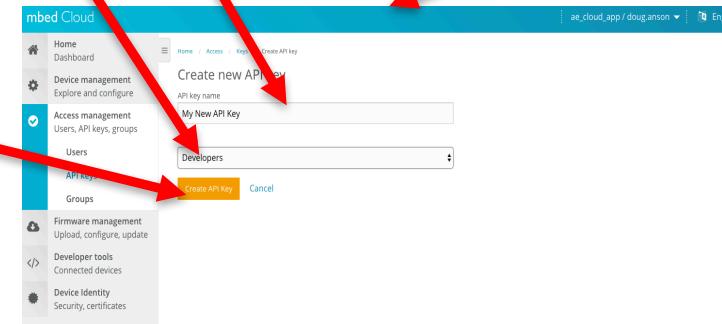
- Chrome and/or Firefox installed

# 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

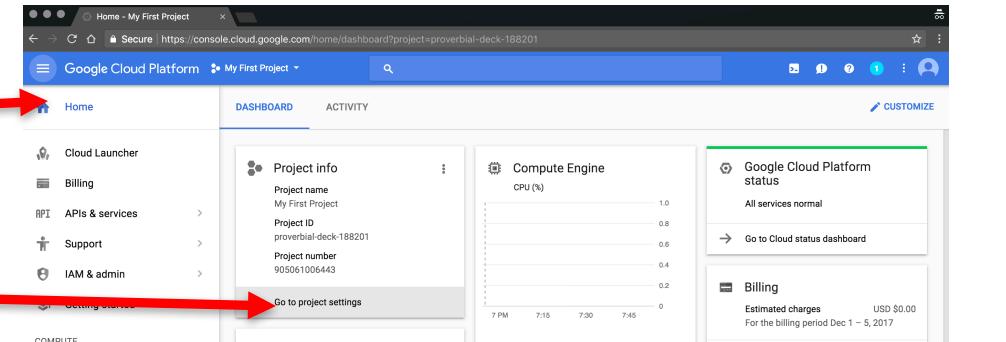
# Create our mbed Cloud Access/API Token

- 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

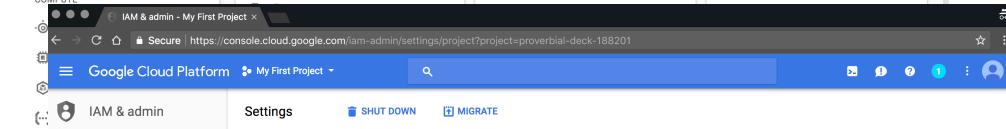


# Create/Name our Google Cloud Default Project

- Go to your Google Cloud Console

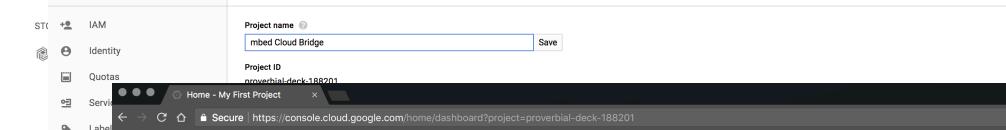


- Select “Project Settings”

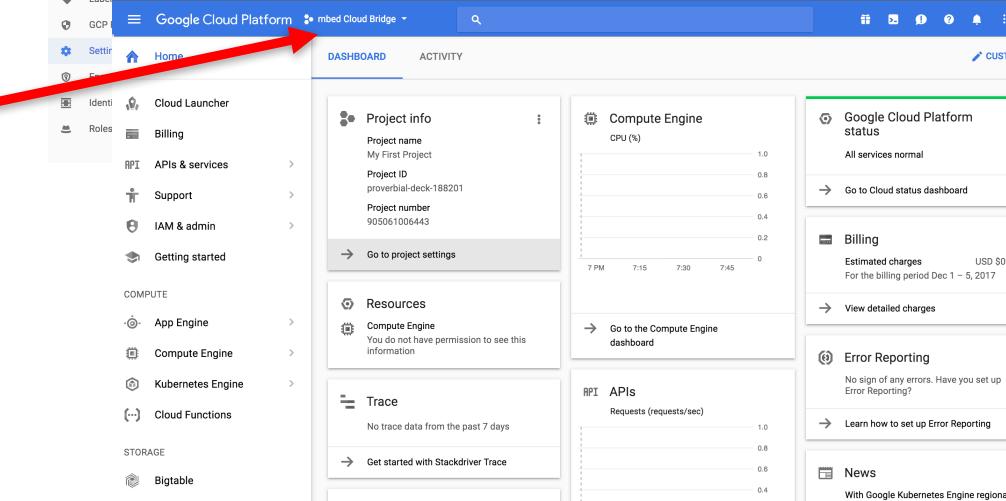


- New Project name:“mbed Cloud Bridge”

- Press "Save"
- Refresh your browser!



- Confirm that the default project name has changed



# Create a Service Account

- Go to your Google Cloud Dashboard

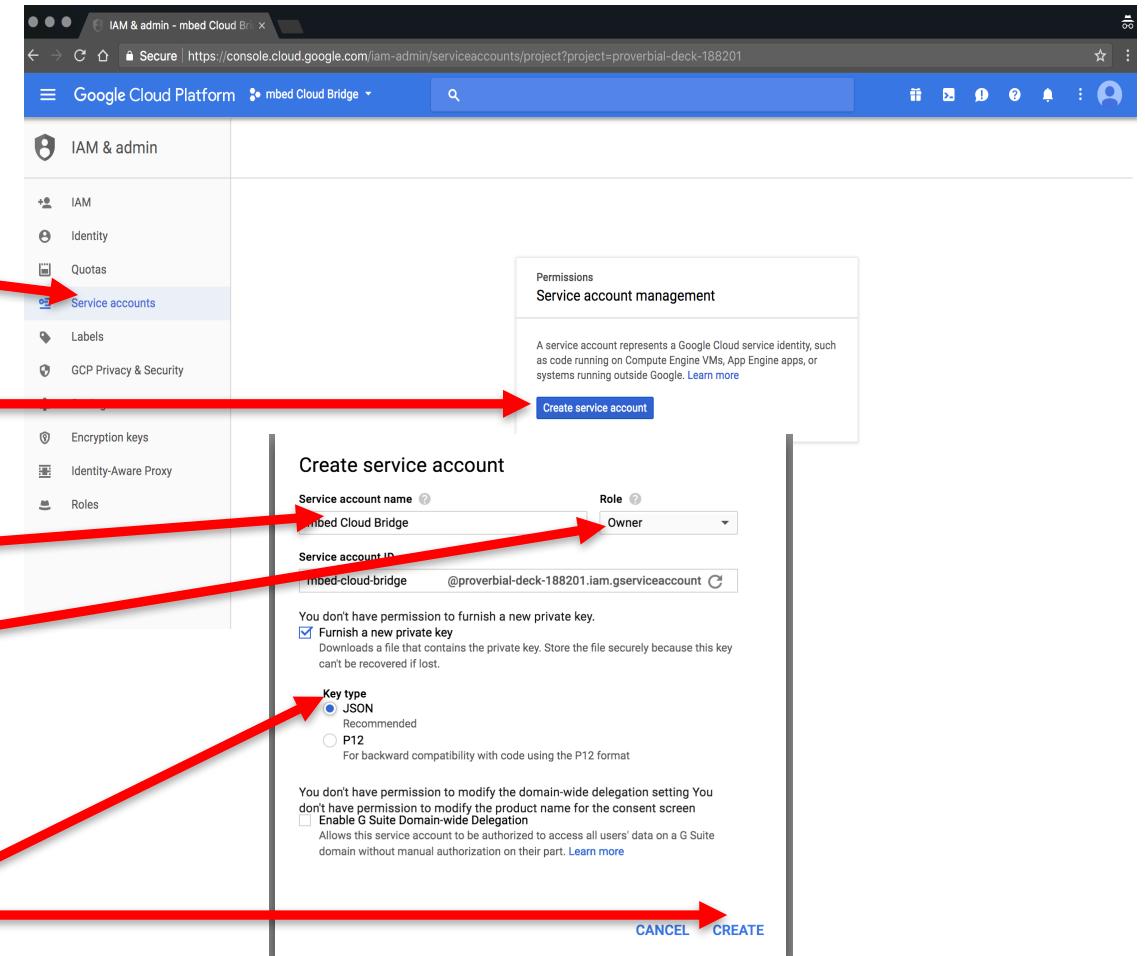
- Select “Service Account”

- Press “Create a Service Account”

- Name the service account

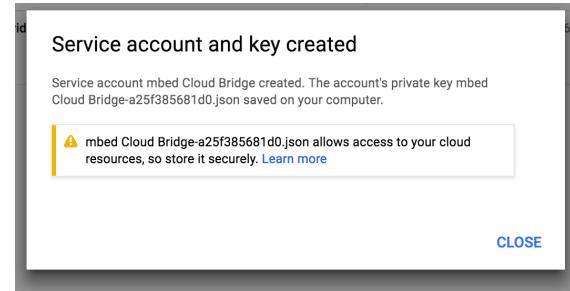
- Select “Owner”

- Confirm “JSON” selected, press Create



# Create a Service Account... “project\_id” value

- A JSON file will be downloaded as a result of creating the service account



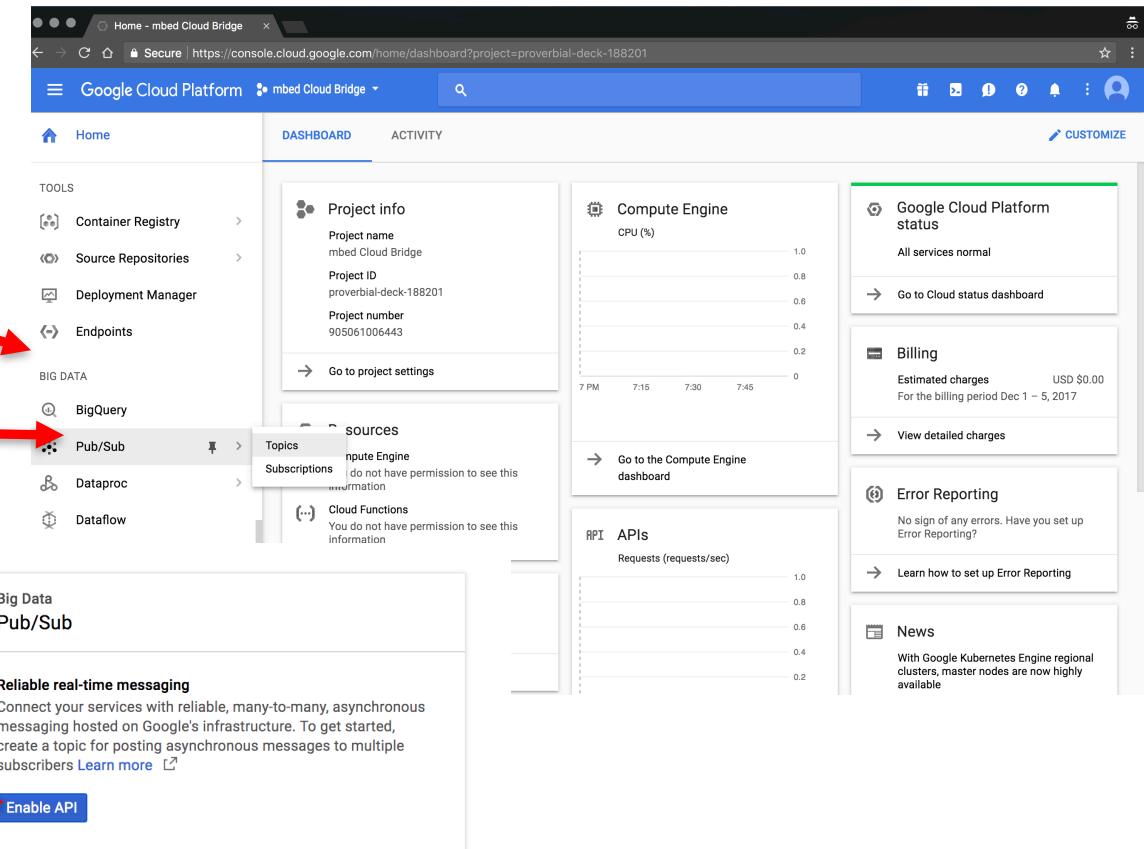
- View the JSON file
- Save the “project\_id” value

```
{  
  "type": "service_account",  
  "project_id": "proverbial-deck-188201",  
  "private_key_id": "a25f385681d04c59ca04b91044aad46752c90b1c",  
  "client_email": "mbed-cloud-bridge@proverbial-deck-188201.iam.gserviceaccount.com",  
  "private_key": "-----BEGIN PRIVATE KEY-----<private key is long and deleted>-----END PRIVATE KEY-----",  
  "client_id": "105206108098722267993",  
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",  
  "token_uri": "https://accounts.google.com/o/oauth2/token",  
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",  
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/mbed-cloud-bridge%40proverbial-deck-188201.iam.gserviceaccount.com"  
}
```

- The “project\_id” value is the “google\_cloud\_app\_name” value for the bridge config
  - Save it off for now... we’ll use it later.

# Create a Pub/Sub Service Instance

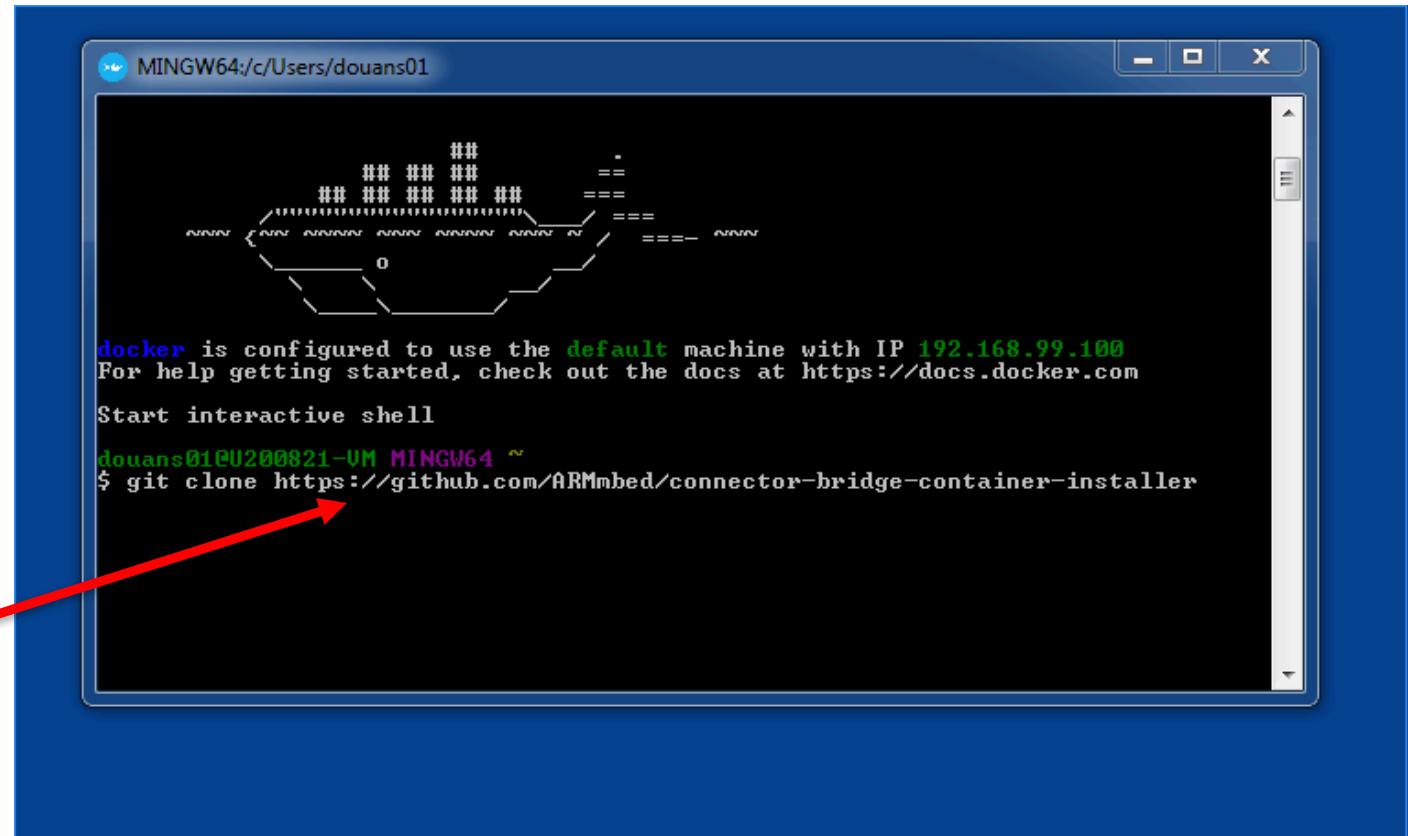
- Go to the Google Cloud Dashboard
- Scroll down to the “Big Data” section
- Select “Pub/Sub”
- Press “Enable API”



Once created, our Google Cloud Environment is ready for our bridge!

# Import our mbed Cloud Google 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:

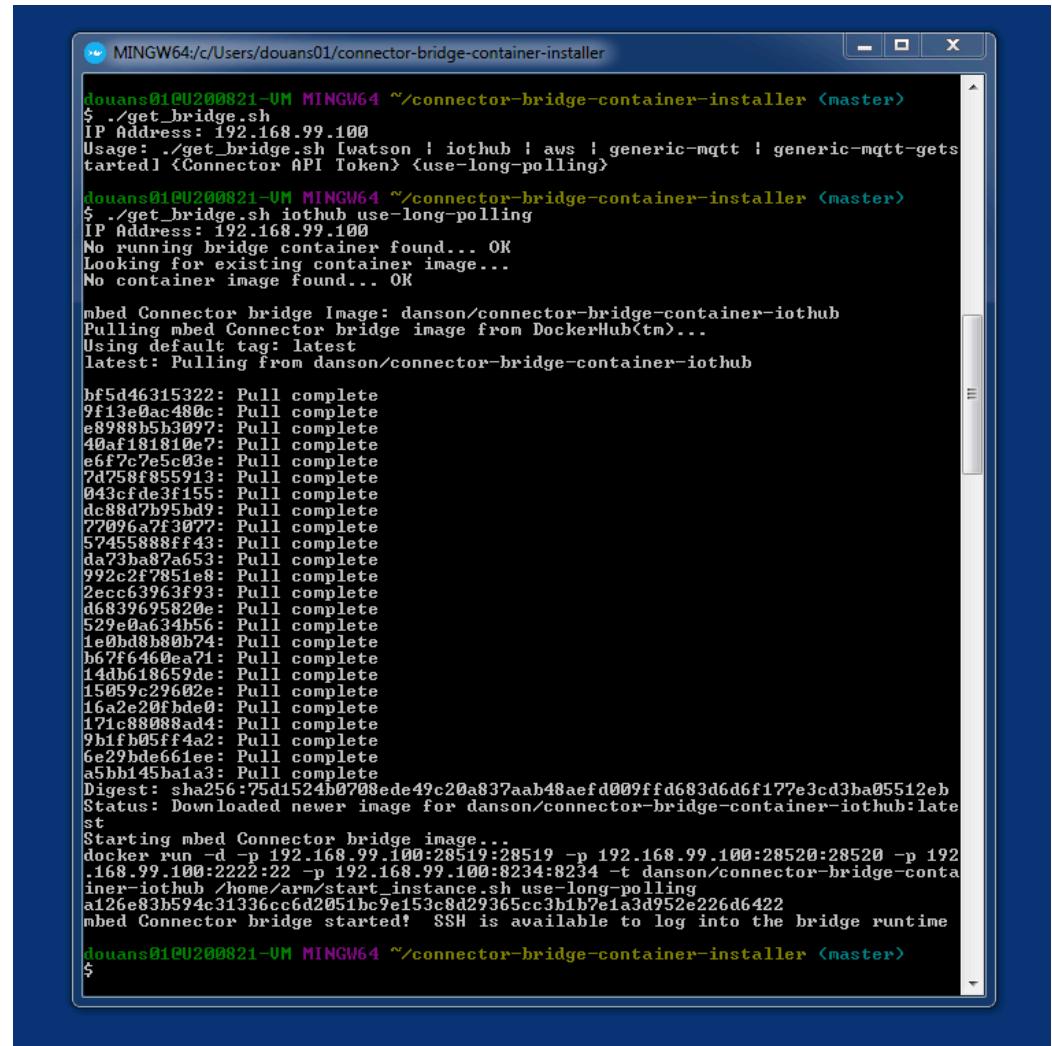
```
docker is configured to use the default machine with IP 192.168.99.100
For help getting started, check out the docs at https://docs.docker.com
Start interactive shell
douans01@PU200821-VM MINGW64 ~
```

A red arrow points to the command line where the 'git clone' command is being typed:

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

# Import our mbed Cloud Google 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 google use-long-polling
- Bridge container will import from DockerHub



```
MINGW64:/c/Users/douans01/connector-bridge-container-installer
douans01@EU200821-VM MINGW64 ~/connector-bridge-container-installer <master>
$ ./get_bridge.sh
IP Address: 192.168.99.100
Usage: ./get_bridge.sh [aws | iothub | generic-mqtt | generic-mqtt-gets]
      [Connector API Token] [use-long-polling]

douans01@EU200821-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
77096a7f3077: Pull complete
5745588ff43: Pull complete
da73ba87a653: Pull complete
992c2f7851e8: Pull complete
2ecc63963f93: Pull complete
d6839695820e: Pull complete
529e0a634b56: Pull complete
1e0hd8b80b74: Pull complete
b67f6460ea71: Pull complete
14db618659de: Pull complete
15059c29602e: Pull complete
16a2e20fbd0: Pull complete
171c88088ad4: Pull complete
91fb05ff4a2: 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@EU200821-VM MINGW64 ~/connector-bridge-container-installer <master>
$
```

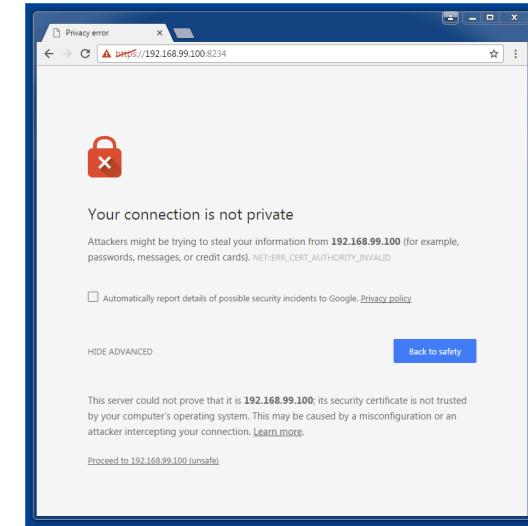
# Configure our Google Bridge (Windows)

- Your Container IP address will be:

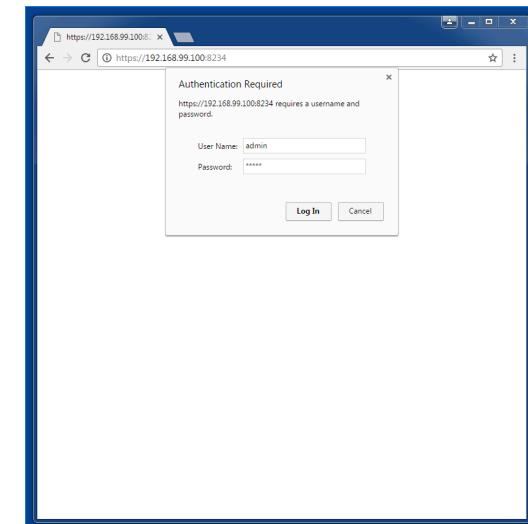
**192.168.99.100**

- Bring up Firefox/Chrome and go to:

<https://192.168.99.100:8234>



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



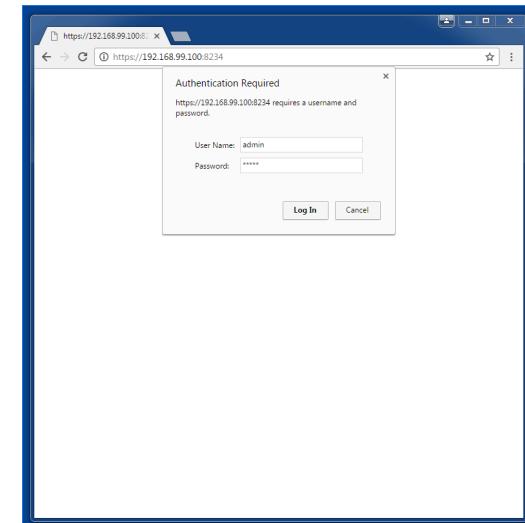
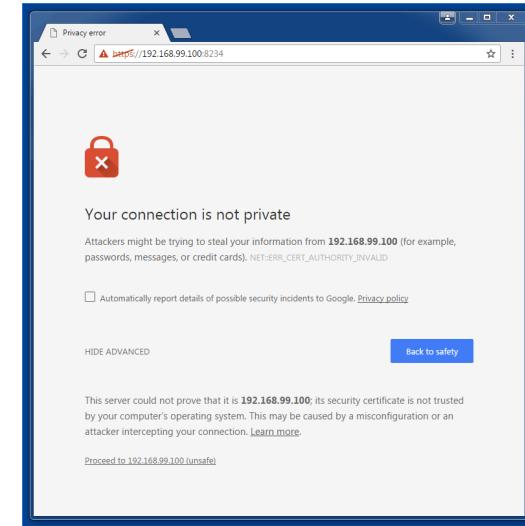
# Configure our Google Bridge (Mac/Linux)

- 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



# Configure the mbed Cloud Bridge for Google...

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

- Enter your “project\_id” value, Press “Save”

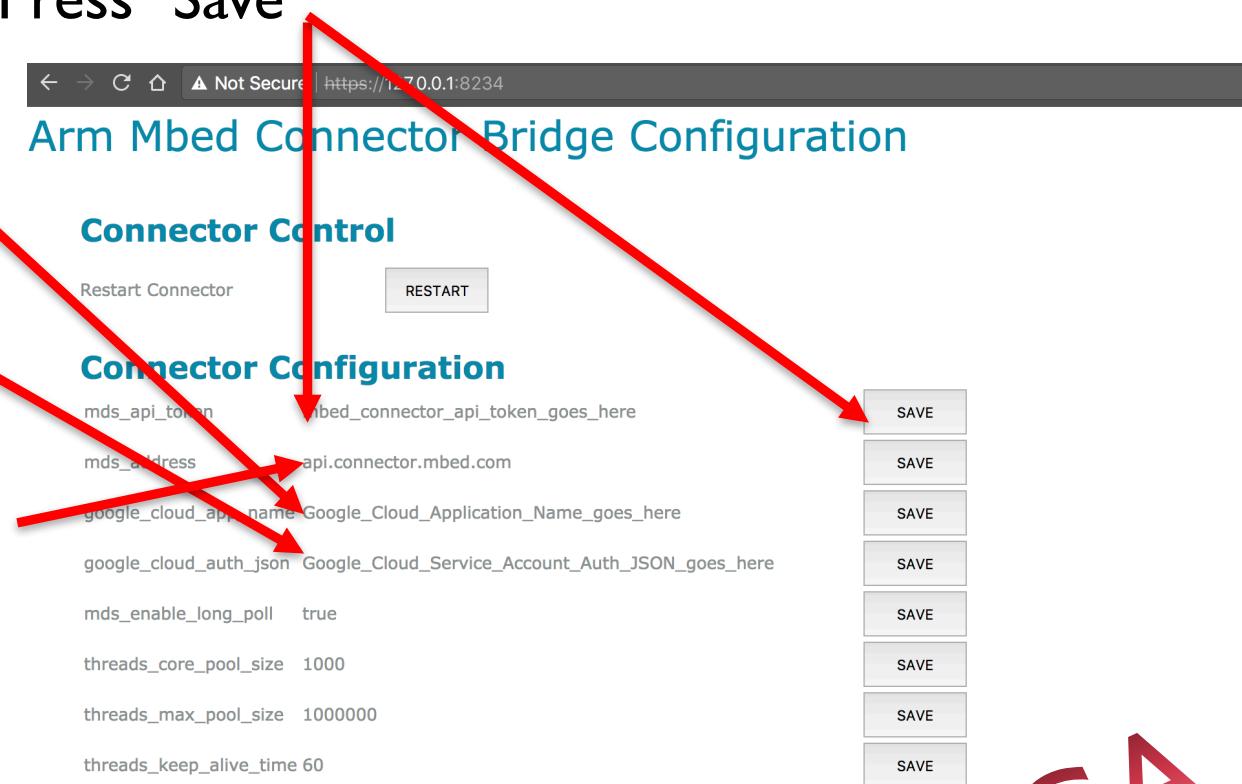
- Enter your Google Auth JSON, Press "Save"

- Ensure you have the right mbed API endpoint

- Cloud: [api.us-east-1.mbedcloud.com](https://api.us-east-1.mbedcloud.com)
- Connector: [api.connector.mbed.com](https://api.connector.mbed.com)
- If changed... be sure to press “Save”

- Press “RESTART”

- Your mbed Cloud Bridge is now configured for Google Cloud



# Status Check

## **So far we've completed the following**

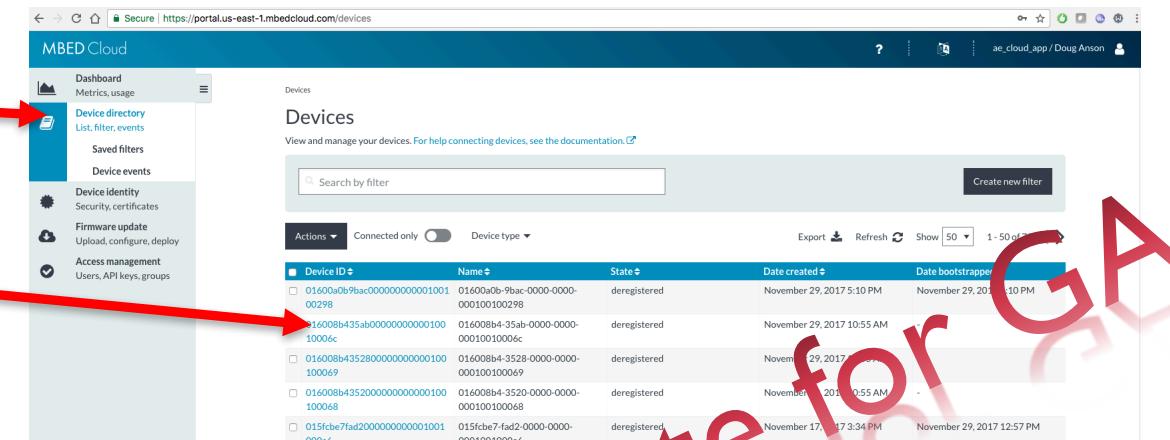
- Ensure that we have the necessary pre-requisites setup on our PC
- Create our mbed Cloud API Key/Token
- Create our Google Service Account & Authentication JSON
- Enable the Pub/Sub API
- Import our bridge instance: a Docker image into our Docker runtime
- Configure our mbed Cloud Bridge for Google

With our bridge now operational, we should be able to restart our endpoint and see messages coming into our Google via a java-based “Exerciser” Application

# 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
  - Go to the mbed Cloud Portal dashboard: <https://portal.us-east-1.mbed.cloud.com>

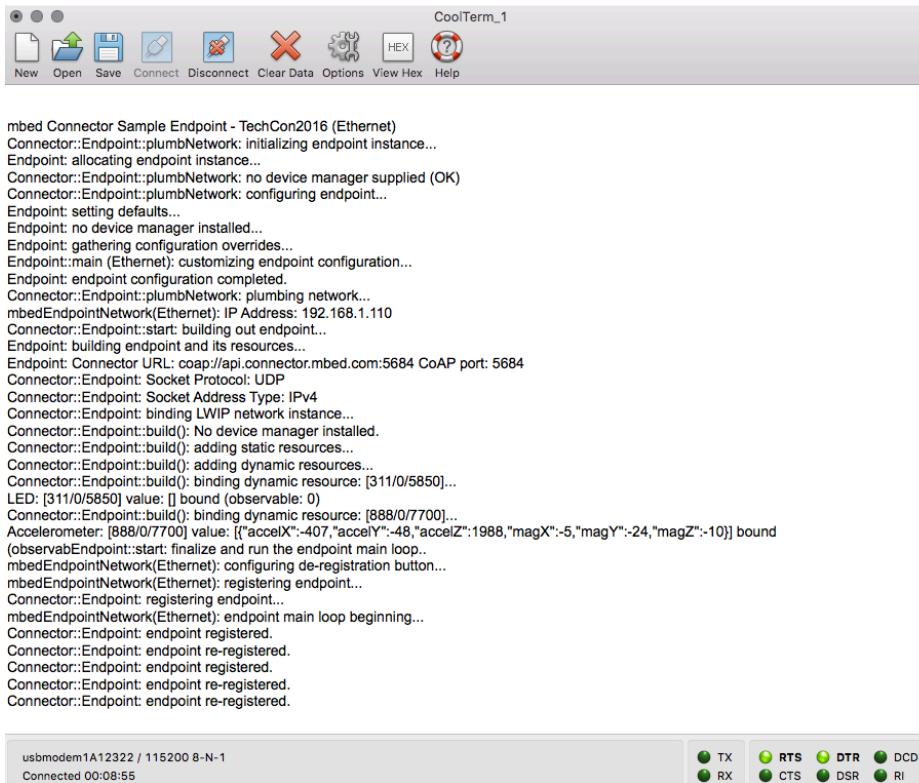
- Look for your endpoint in the list
  - Record your endpoint's Device ID  
You'll need this in the next steps...



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”



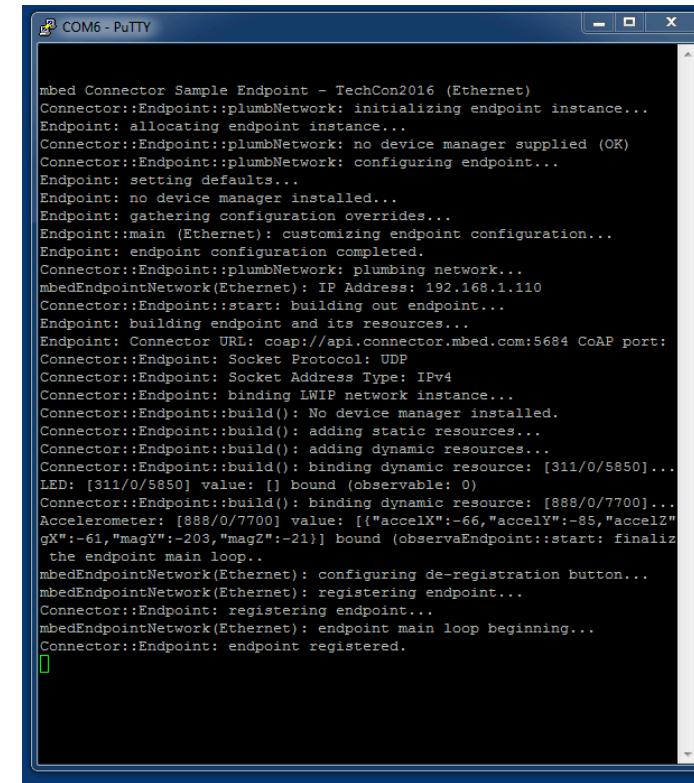
CoolTerm\_1

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 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": -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.
```

# Install the Java Exerciser Application

- Install Java and the NetBeans IDE <http://www.netbeans.org>

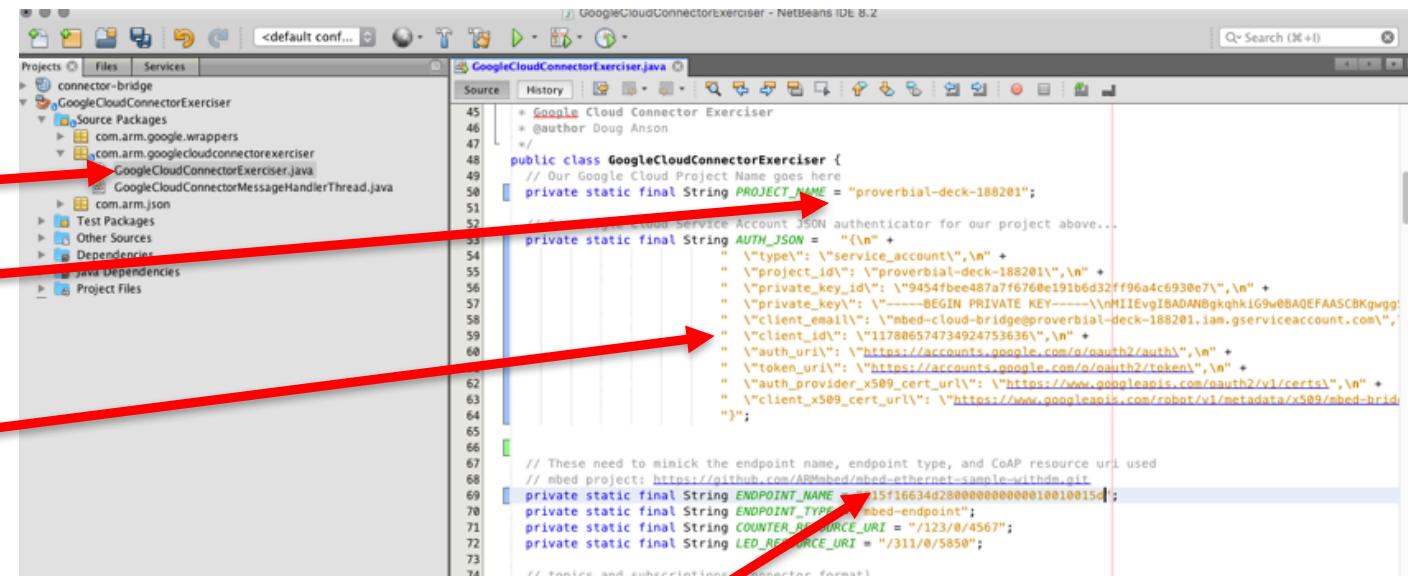
- Clone the following GitHub repo:  
<https://github.com/ARMmbed/GoogleCloudConnectorExerciser>

- Open the project in NetBeans  
Select this source file

- Enter “project\_id” value

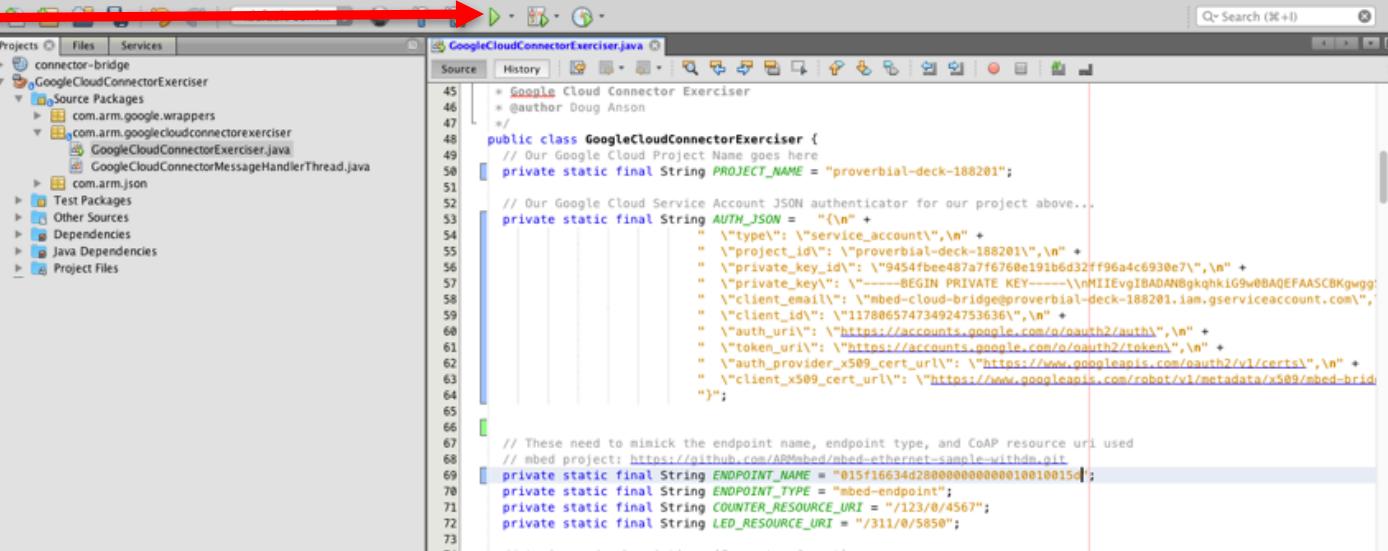
- Enter your JSON string  
(note “string” formatting!)

- Enter your mbed Cloud Endpoint Device ID you saved!



```
* Google Cloud Connector Exerciser
* @author Doug Anson
*/
public class GoogleCloudConnectorExerciser {
    // Our Google Cloud Project Name goes here
    private static final String PROJECT_NAME = "proverbial-deck-188201";
    // This is the Cloud Service Account JSON authenticator for our project above..
    private static final String AUTH_JSON = "("n" +
        "  \"type\": \"service_account\",\\n" +
        "  \"project_id\": \"proverbial-deck-188201\",\\n" +
        "  \"private_key\": \"-----BEGIN PRIVATE KEY-----\\nHIEvgIBADANBgkqhkiG9w0BAQEFAASCBKggg:" +
        "  \"private_key\": \"-----BEGIN PRIVATE KEY-----\\nHIEvgIBADANBgkqhkiG9w0BAQEFAASCBKggg:" +
        "  \"client_email\": \"mbed-cloud-bridge@proverbial-deck-188201.iam.gserviceaccount.com\",\\n" +
        "  \"client_id\": \"117806574734924753636\",\\n" +
        "  \"auth_uri\": \"https://accounts.google.com/o/oauth2/auth\",\\n" +
        "  \"token_uri\": \"https://accounts.google.com/o/oauth2/token\",\\n" +
        "  \"auth_provider_x509_cert_url\": \"https://www.googleapis.com/oauth2/v1/certs\",\\n" +
        "  \"client_x509_cert_url\": \"https://www.googleapis.com/robot/v1/metadata/x509/mbed-brid" +
        ")";
    // These need to mimick the endpoint name, endpoint type, and CoAP resource uri used
    // mbed project: https://github.com/ARMmbed/mbed-ethernet-sample-withdm.git
    private static final String ENDPOINT_NAME = "15166342800000000010010015d";
    private static final String ENDPOINT_TYPE = "mbed-endpoint";
    private static final String COUNTER_RESOURCE_URI = "/123/0/4567";
    private static final String LED_RESOURCE_URI = "/311/0/5850";
    // topics and subscriptions connector format
}
```

# Running the Java Exerciser Application

- Press the “Run” button 
- The project will compile
- If the compile succeeds without error, the project will run...
- You should see output down here 
- The Exerciser will turn the LED resource on and off depending on whether the monotonic resource observation value is even or odd... this demonstrates the bi-directional nature of the bridge with your endpoint! **CONGRATS!! You are finished!**

# Summary

We have completed the following – congratulations!

- Created our Google Cloud and mbed environments and PC tool setup
- Imported our mbed endpoint, customized, installed, and ran it
- Created our own Google Pub/Sub instance
- Imported and configured our ARM Google Prototype mbed Cloud Bridge
- Examined live telemetry with the java-based "Exerciser" application
- Interacted with our device through the java-based "Exerciser" application

Great JOB! Thanks for your time.