

An aerial photograph showing several people riding bicycles on a paved surface with a distinct diamond-shaped grid pattern. The people are scattered across the frame, some in groups and some alone, all moving in various directions. The overall scene is a mix of blue and grey tones.

arm

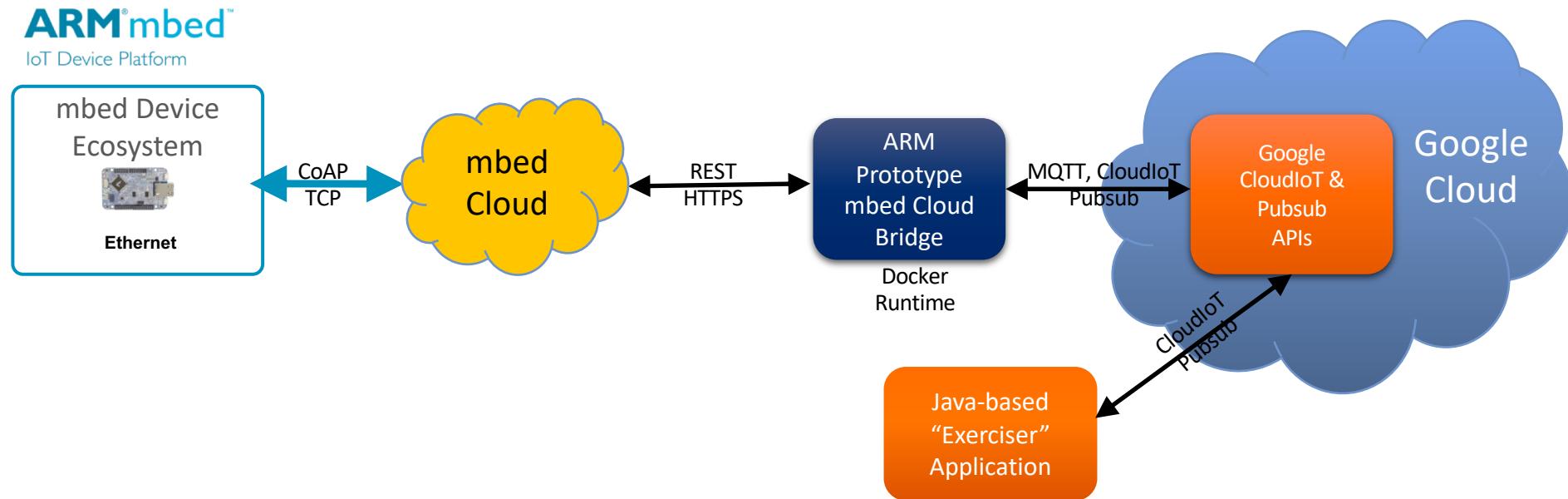
Workshop

Connecting IoT devices to the cloud with Google CloudIoT & mbed Cloud

Doug Anson
v2.5 (GCR Edition)

August 22, 2018

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 CloudIoT & Pubsub 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 (PTSOI method...)

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

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

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

Create our Google Cloud Account

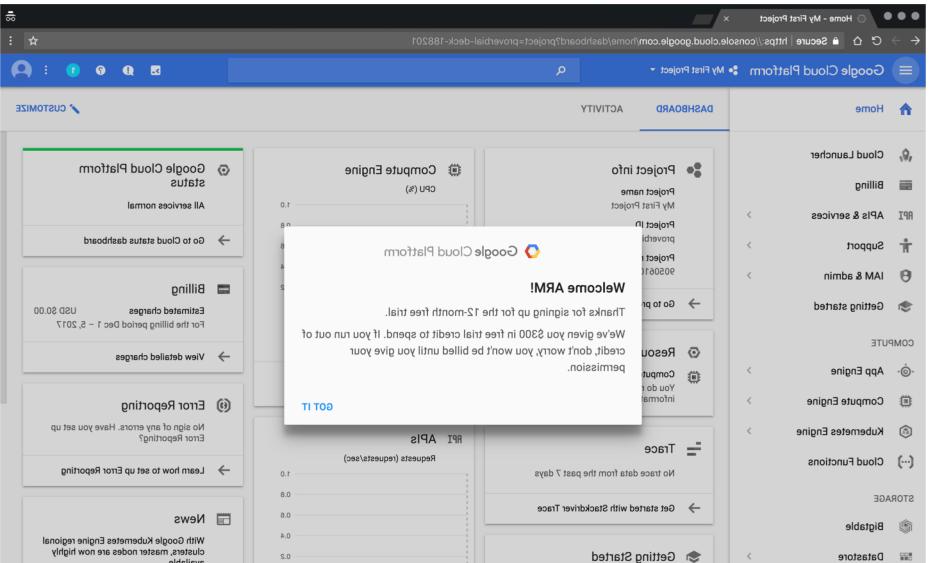
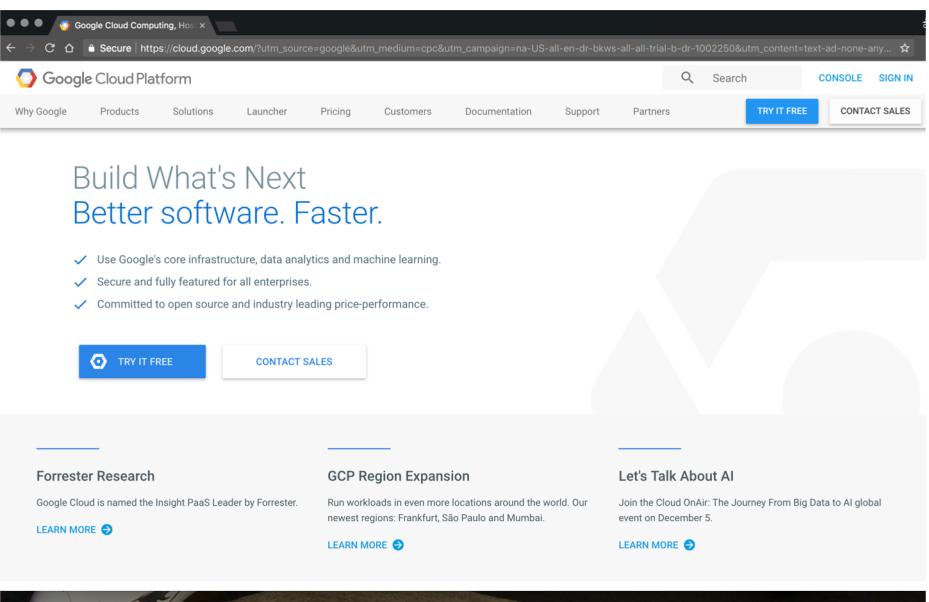
Navigate to the Google Console:
<https://cloud.google.com>

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

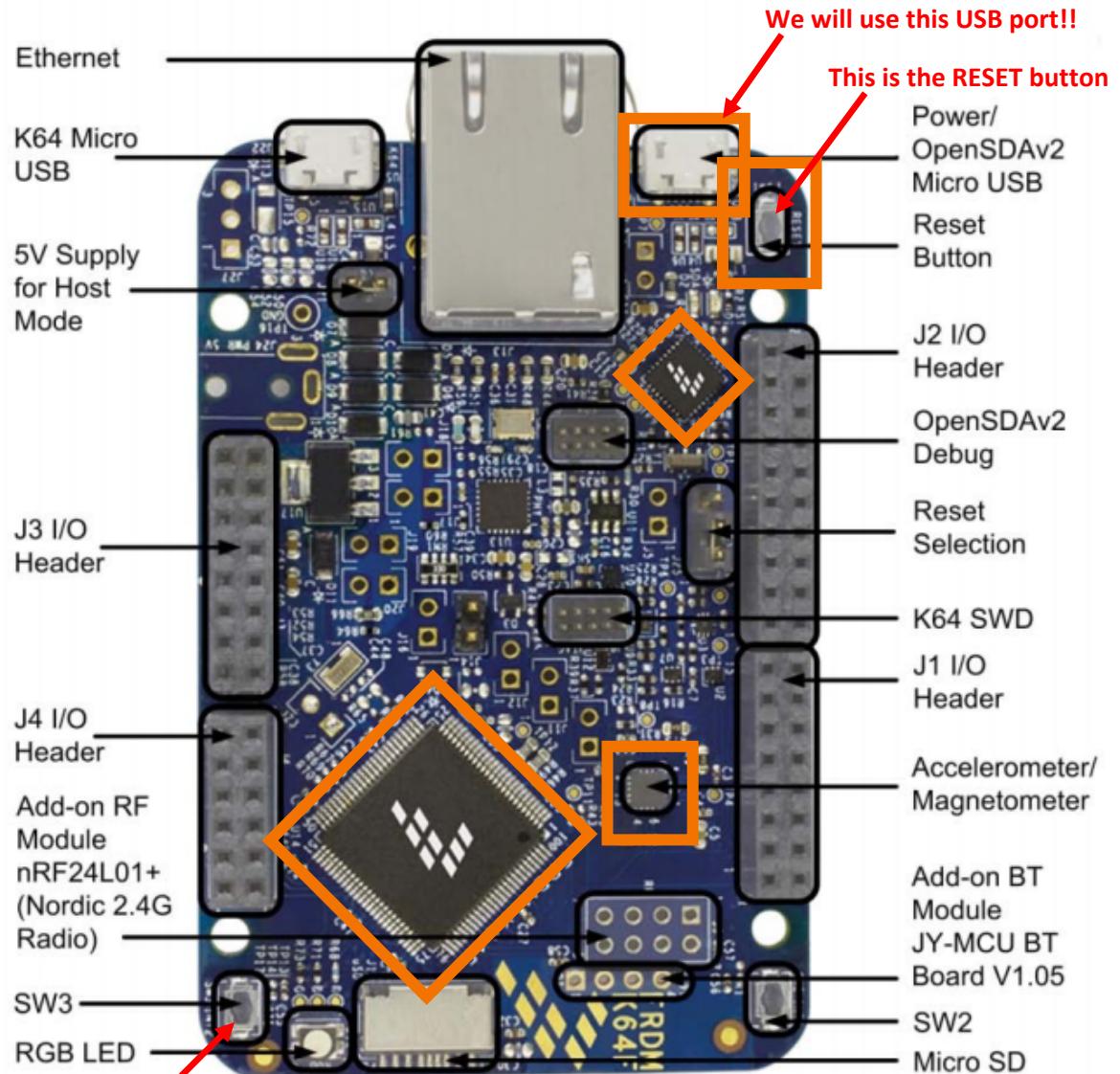
- Cortex-M4, 120MHz, 1MB Flash, 256KB SRAM
- Easy access to MCU I/O
- 3-axis **accelerometer**/3-axis **magnetometer**
- RGB LED
- Add-on **Bluetooth** Module
- Built-in Ethernet/Add-on **Wireless** Module
- Micro SD

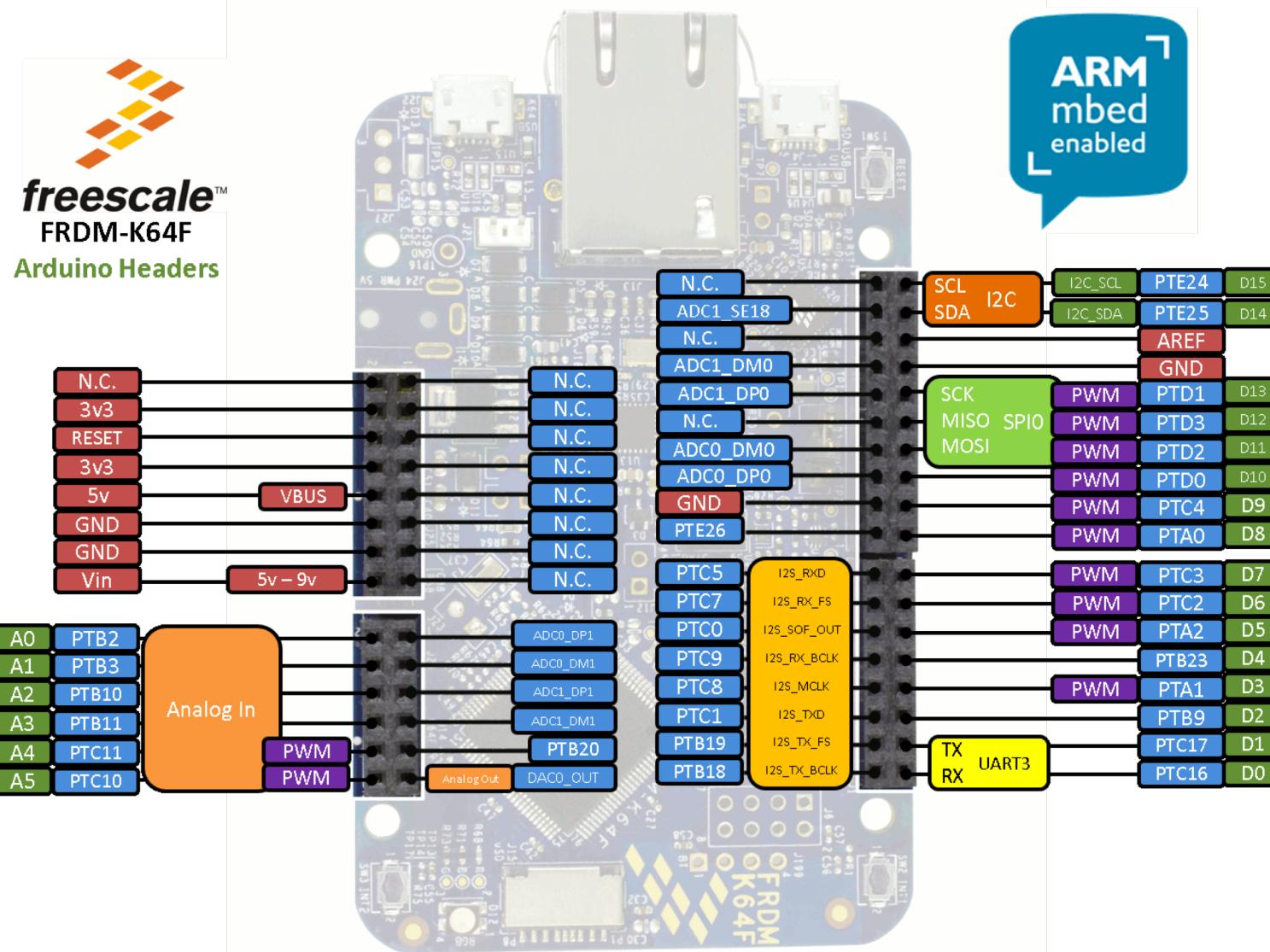
Arduino shield compatible

Flash programming functionality

Enabled by OpenSDA debug interface

De-Registration Button!
(SW3)





Install the Necessary Tools

- **Windows IMPORTANT:** Install the mbed USB Serial driver -
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
- Linux Serial Terminal - http://freeware.the-meiers.org/CoolTerm_Linux.zip

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

Prior to
workshop

Create Your mbed Account

Navigate to: <https://os.mbed.com/>

Prior to
Workshop

Create an Account

The screenshot shows the ARM mbed developer website's login and signup interface. At the top, there is a navigation bar with links for Platforms, Components, Handbook, Cookbook, Code, Questions, and Forum. On the right side of the header, there are buttons for Dashboard and Compiler. A search bar is located at the top center. Below the header, there is a large "ARM mbed™" logo. The main content area has two side-by-side forms: a "Login" form on the left and a "Signup" form on the right. The "Login" form includes fields for "Username" and "Password", both with "I've forgotten my [username/password]" links below them. It also has a "Remember me" checkbox and a "Login" button. The "Signup" form features a large blue "mbed" logo and a "Signup" button. At the bottom of the page, there is a footer with links for blog, we're hiring!, support, service status, privacy policy, terms and conditions, and language selection (en ja es de).

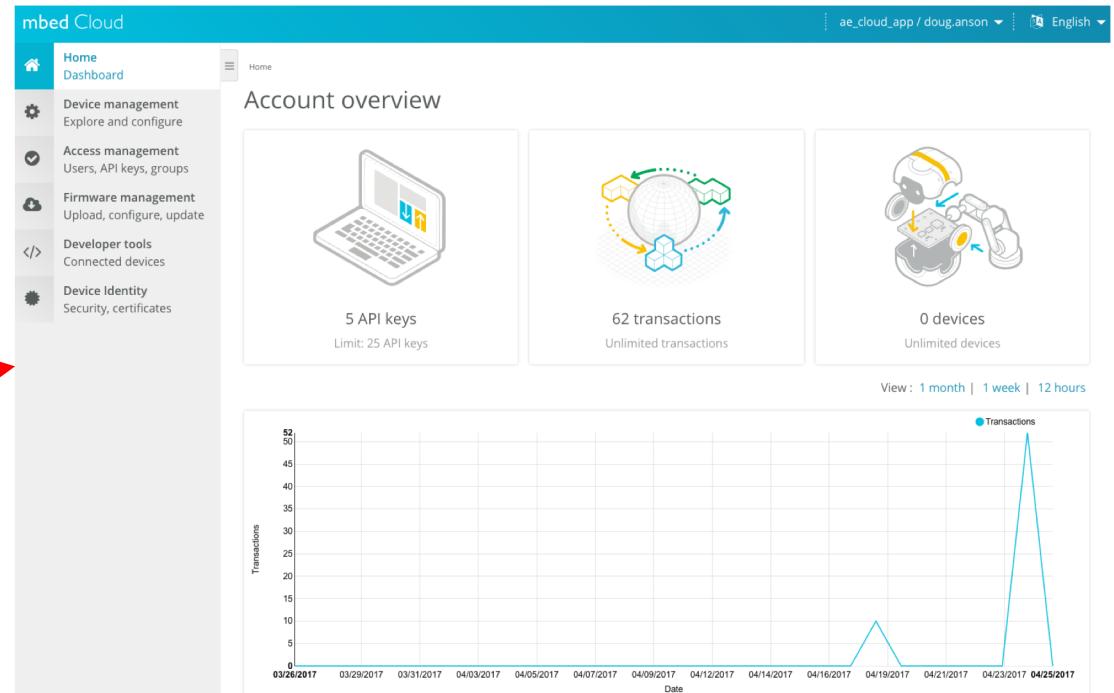
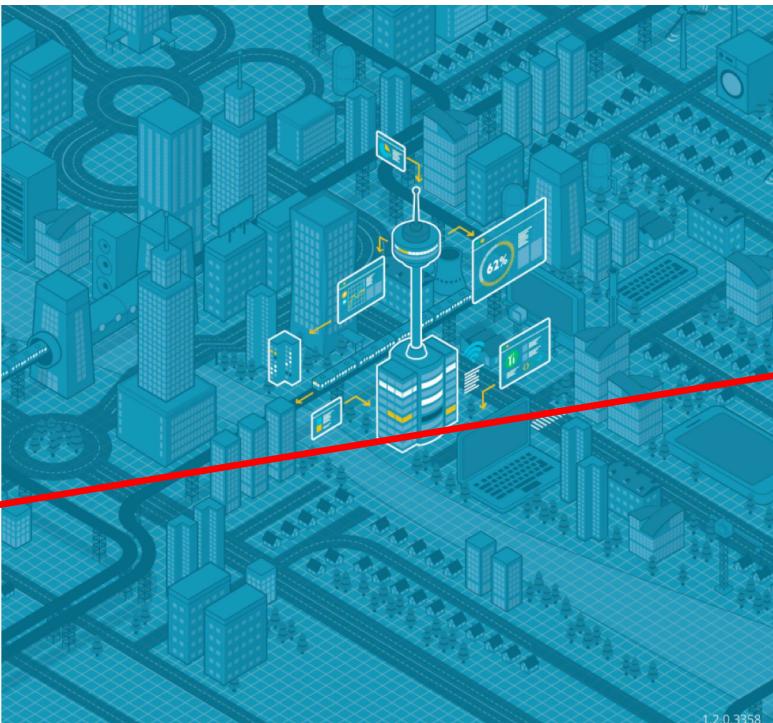
Create Your mbed Cloud Account...

Navigate to the mbed Cloud Dashboard:

<https://portal.mbedcloud.com>

Confirm that you can log into your mbed device mbed Cloud dashboard

Prior to
Workshop



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”

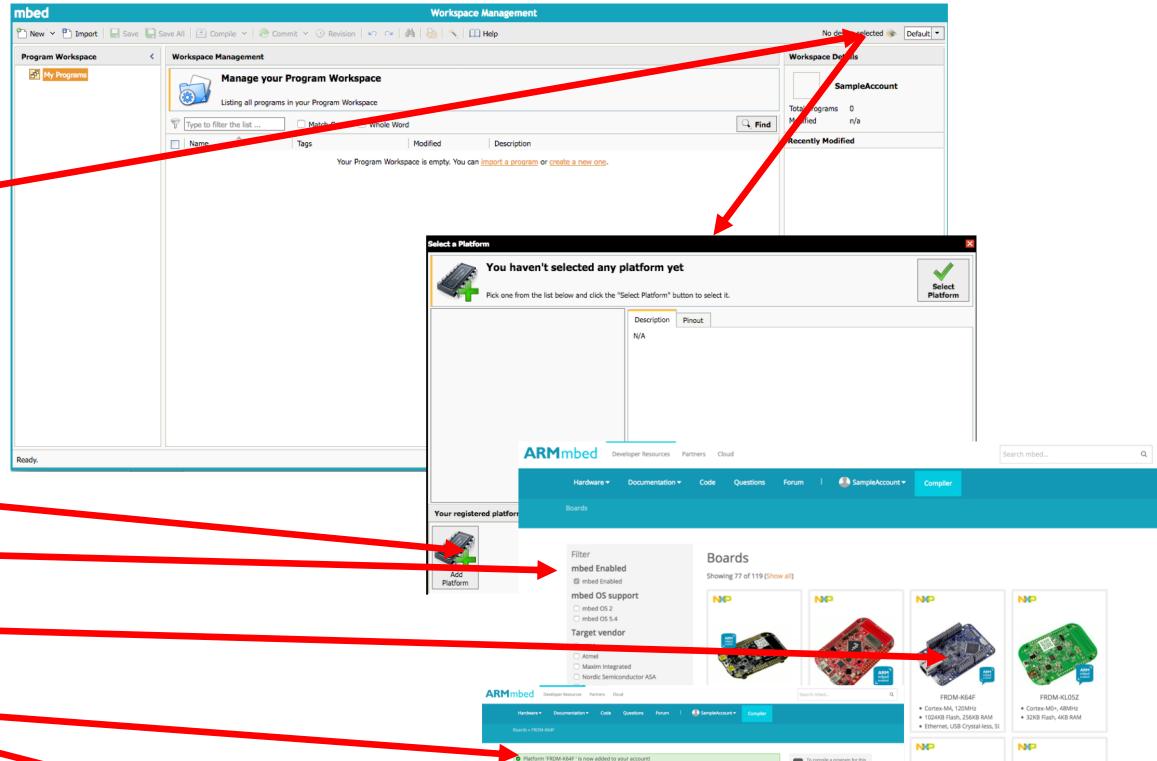
Click on “Add to Compiler”... note addition.

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

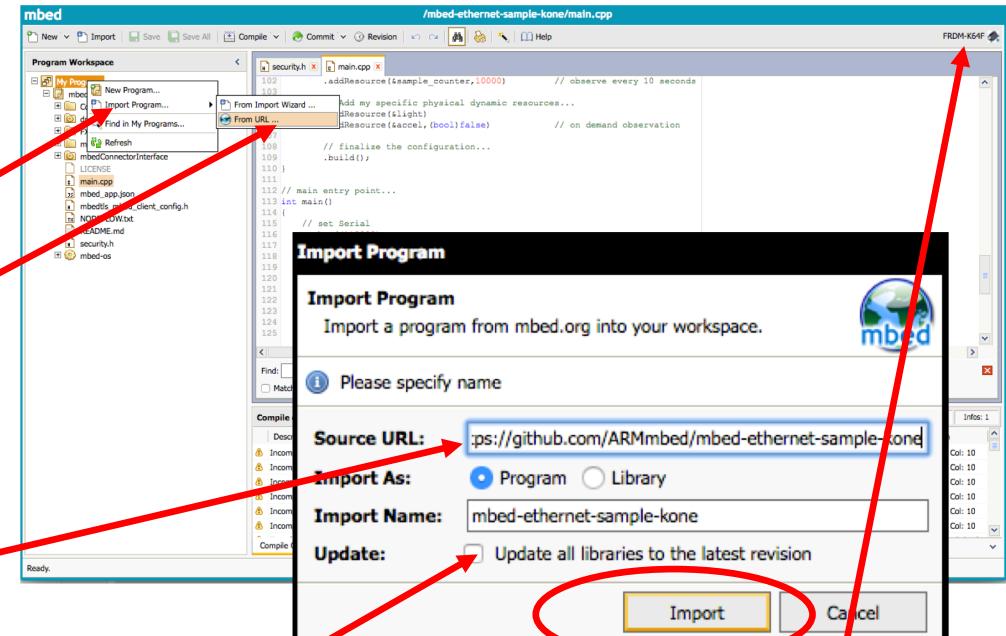
Right-click on “My Programs” → “Import Program”

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 three CoAP resources: accelerometer , monotonic counter, and LED

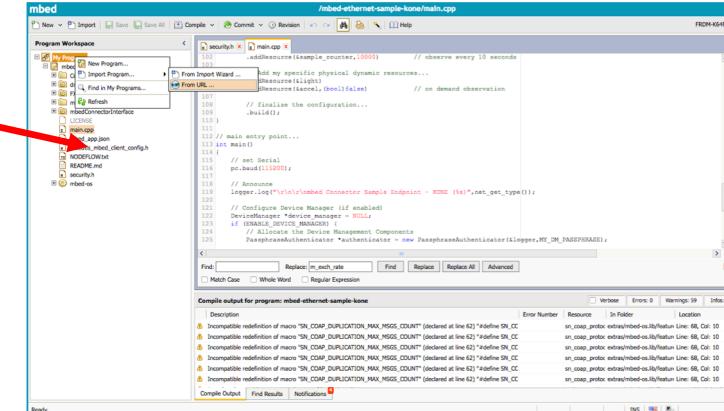
Select the project name and press the “Compile” button

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 "DAPLINK")

K64F green LED will flicker for a bit, then stop, and dismount/remount...



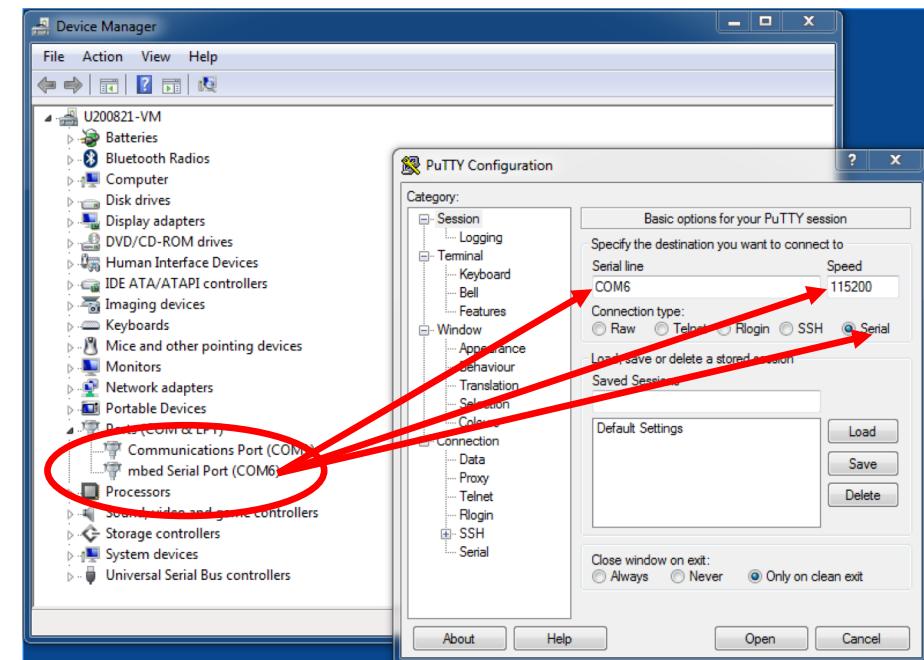
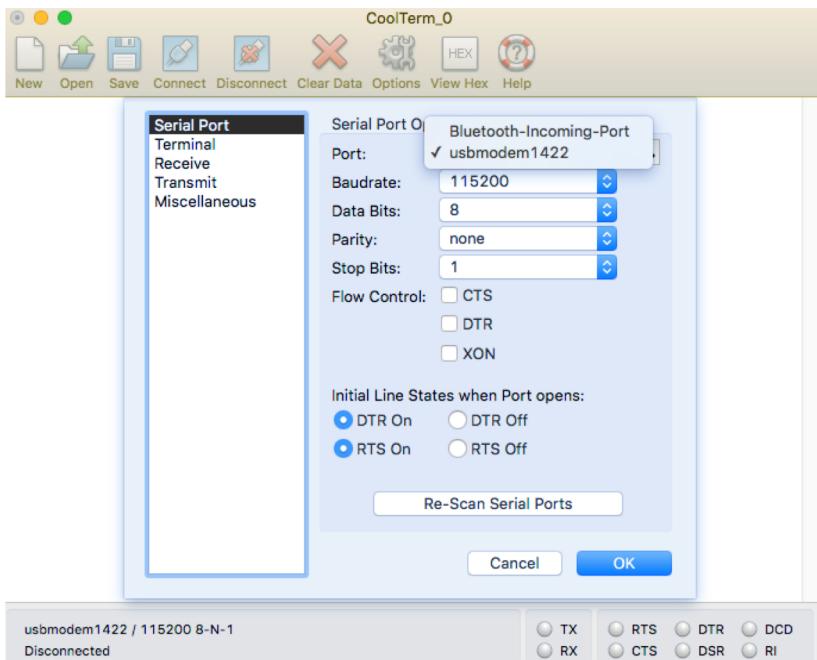
Running your Endpoint Code

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...

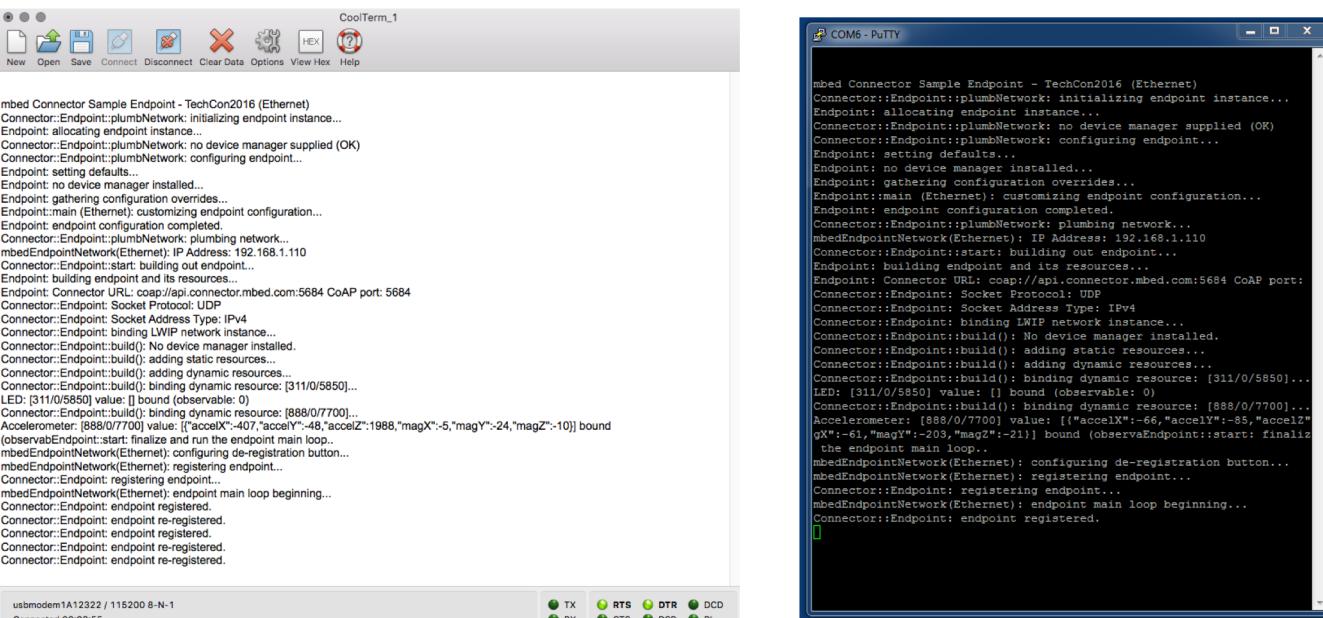
Connect your Serial Terminal to the K64F:

- 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 image shows two side-by-side screenshots of serial terminal windows. The left window is titled "CoolTerm_1" and the right window is titled "COM6 - PuTTY". Both windows display the same text output, which is the log of an mbed Connector Sample Endpoint running on a TechCon2016 (Ethernet) device. The log includes messages like "Connector::Endpoint::plumbNetwork: initializing endpoint instance...", "Connector::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": -407, "accelY": -48, "accelZ": 1988, "magX": -5, "magY": -24, "magZ": -10}] bound (observable: 0)", "mbedEndpointNetwork(Ethernet): configuring run the endpoint main loop.", "mbedEndpointNetwork(Ethernet): registering endpoint...", "Connector::Endpoint::register endpoint...", "mbedEndpointNetwork(Ethernet): endpoint main loop beginning...", "Connector::Endpoint::end point registered.", "Connector::Endpoint::end point re-registered.", "Connector::Endpoint::end point registered.", "Connector::Endpoint::end point re-registered.", "Connector::Endpoint::end point re-registered.", "Connector::Endpoint::end point 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 (115200,8N1, 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 (PTSOI 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...

Confirm all of our needed tools are installed

Create our mbed Cloud API Key/Token

Create our Google Service Account & Authentication JSON

Enable the Google Pubsub and CloudIoT API Instances

Import our prototype mbed Cloud Bridge instance as a Docker runtime from GCR

Configure our Prototype mbed Cloud Bridge for Google Cloud

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

Latest “git” installed

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

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”

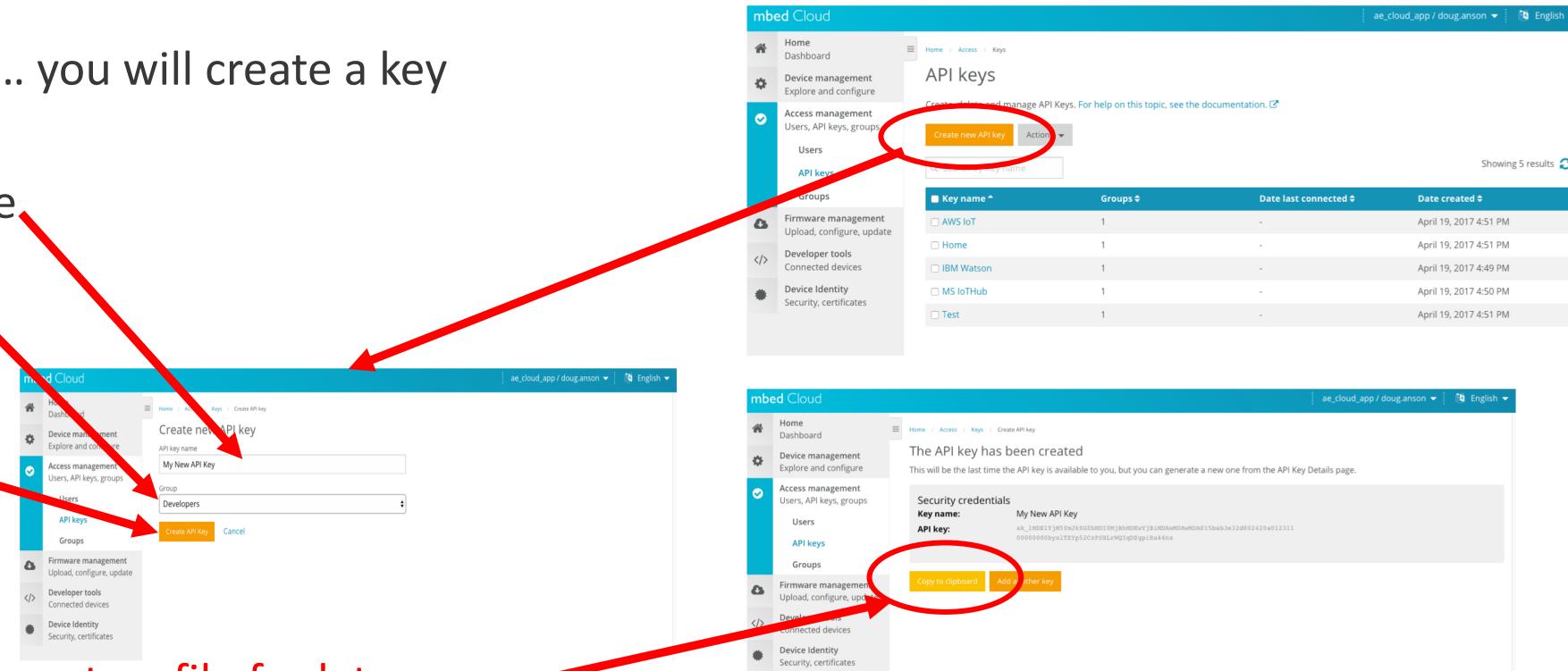
Press “Create new API Key”... you will create a key

Give the new API Key a name

Select “Developers” group

Create the API Key

Press “copy to clipboard” – save to a file for later use



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

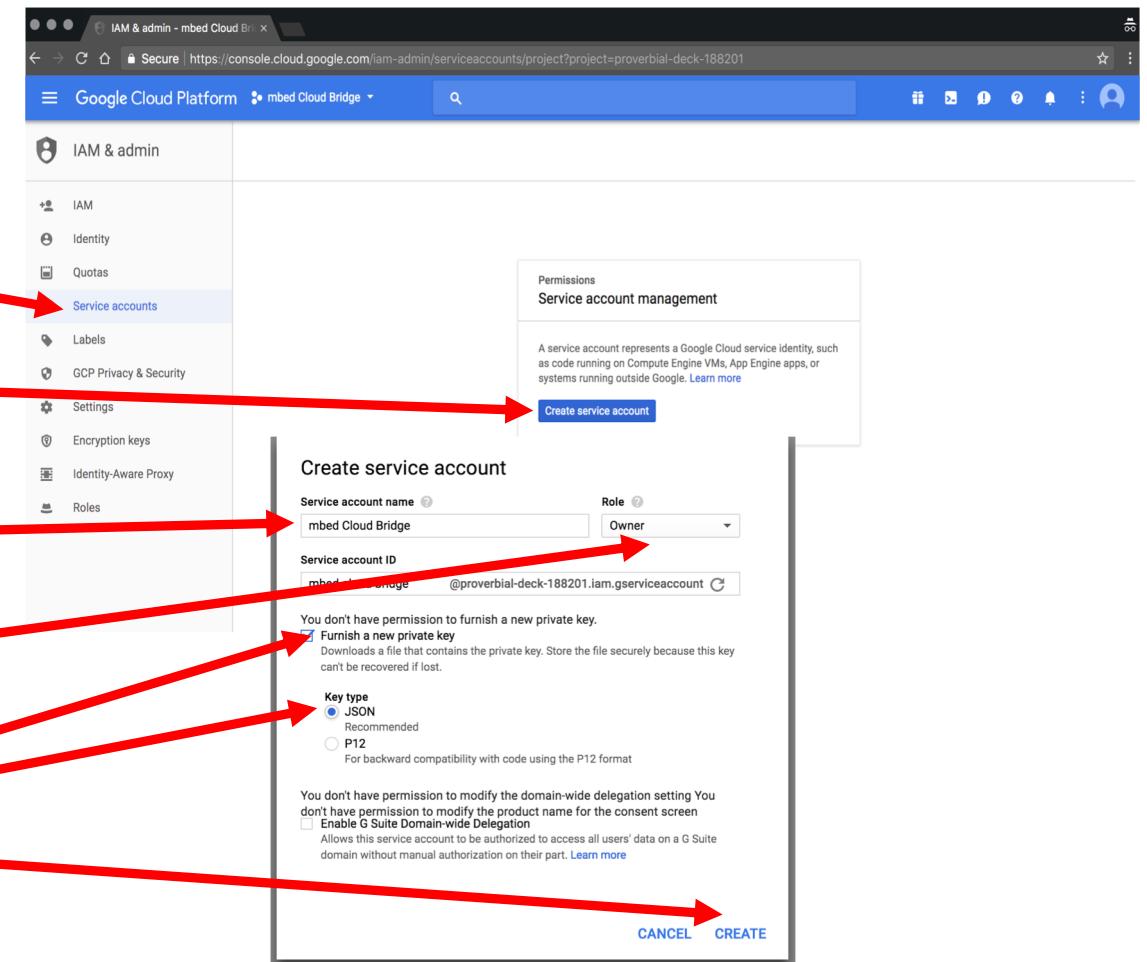
Select “Service Account”

Press “Create a Service Account”

Name the service account

Select Project “Owner”

Confirm “JSON” selected, press Create



The JSON file down auto download - Save off this file – you'll need it later.

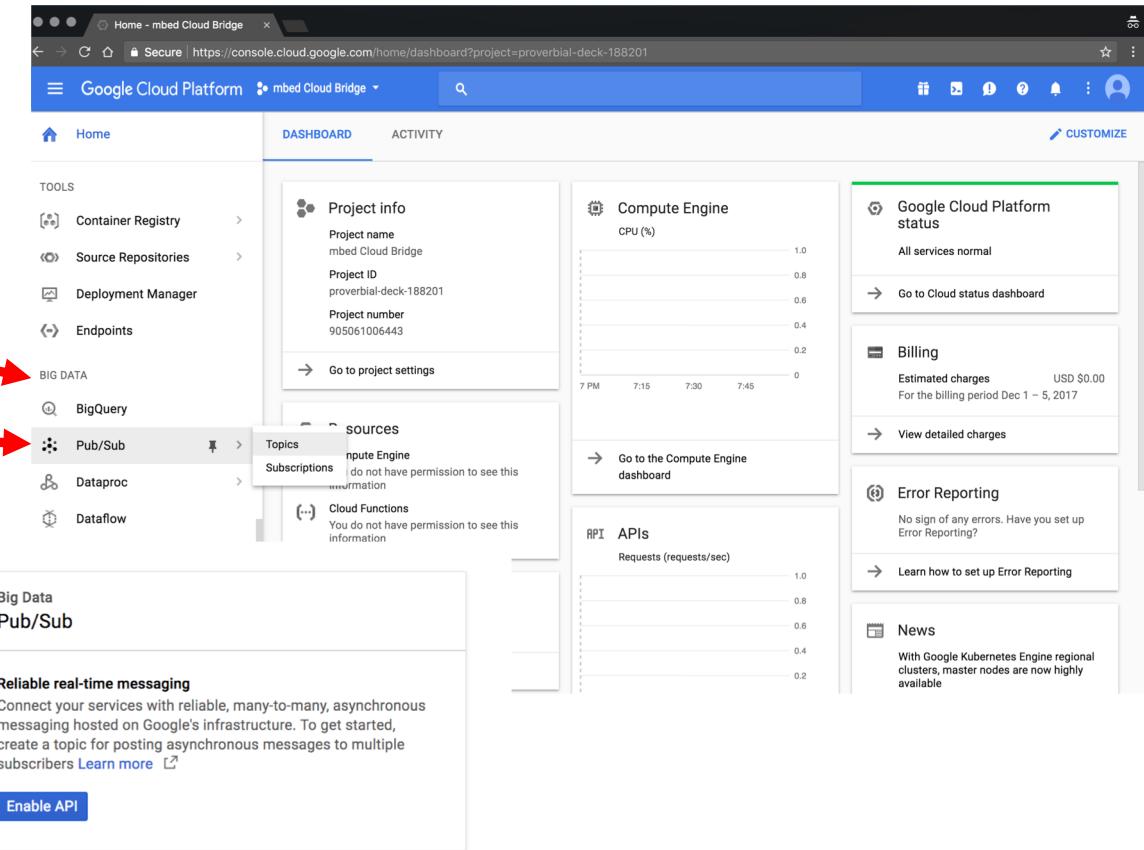
Create a Google Pubsub API Instance

Go to the Google Cloud Dashboard

Scroll down to the “Big Data” section

Select “Pub/Sub”

Press “Enable API”



Next, we have to create our Google CloudIoT API instance...

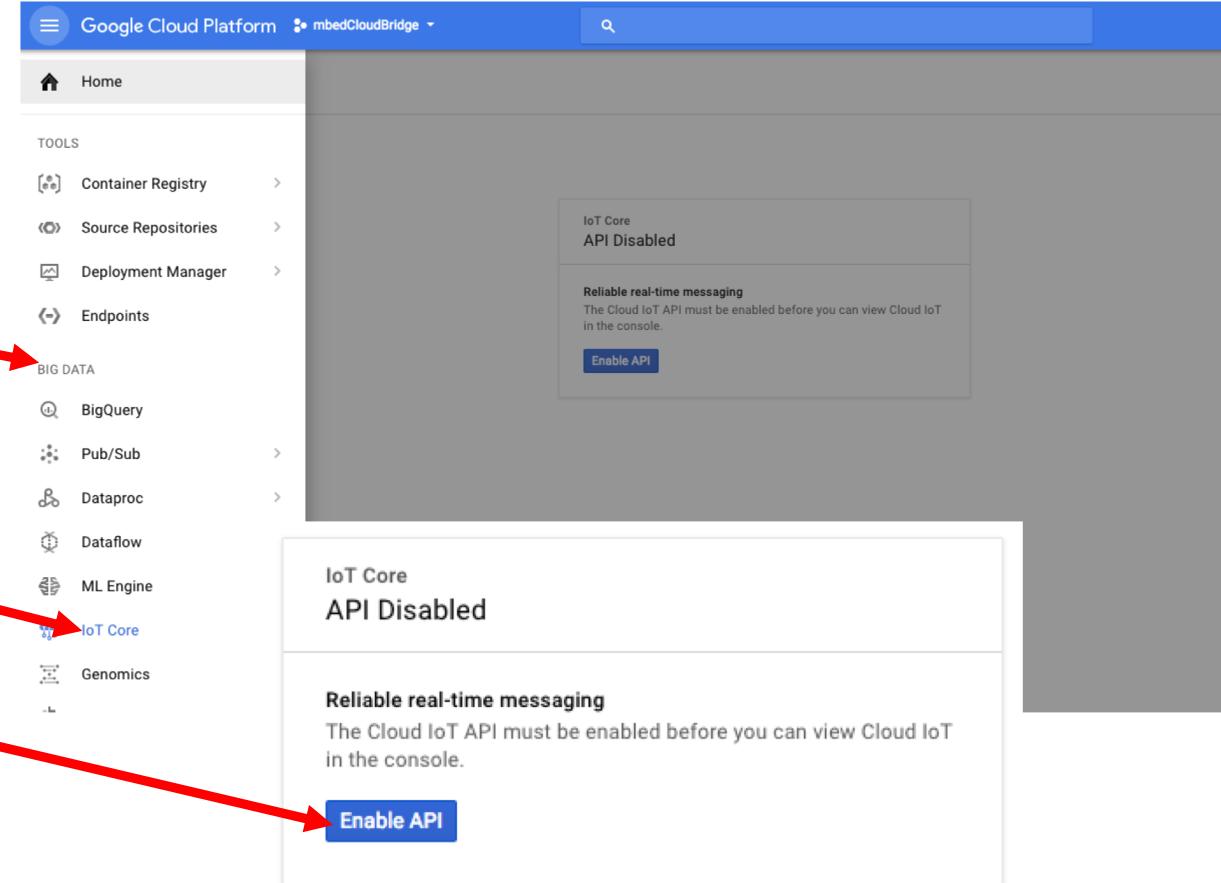
Create a Google Cloud IoT API Instance

Go to the Google Cloud Dashboard

Scroll down to the “Big Data” section

Select “IoT Core”

Press “Enable API”

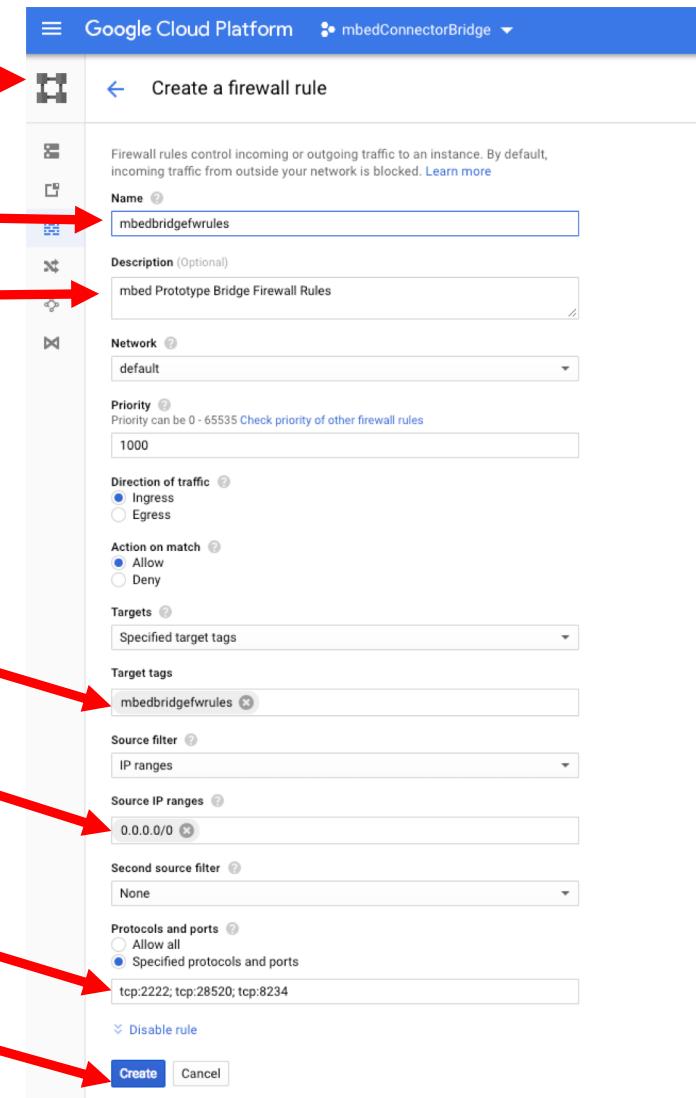


Once created, our Google Cloud Environment is ready for our bridge!

Import the mbed Cloud prototype Bridge for Google from GCR

First we have to create a special firewall rule via VPC Networks

- Name the firewall rule
- Provide a description
- Create a tag (remember this!)
- 0.0.0.0/0 for the port range
- Specify these ports
- Press “Create”



Import the mbed Cloud prototype Bridge for Google from GCR...

With “Compute Engine” select “VM Instances”

Select “Create”

Provide a name for your bridge

Choose “micro” machine type

Check the “deploy container image”

Container image entered here

gcr.io/mbedconnectorbridge-155920/github-douganson-connector-bridge-container-google:latest

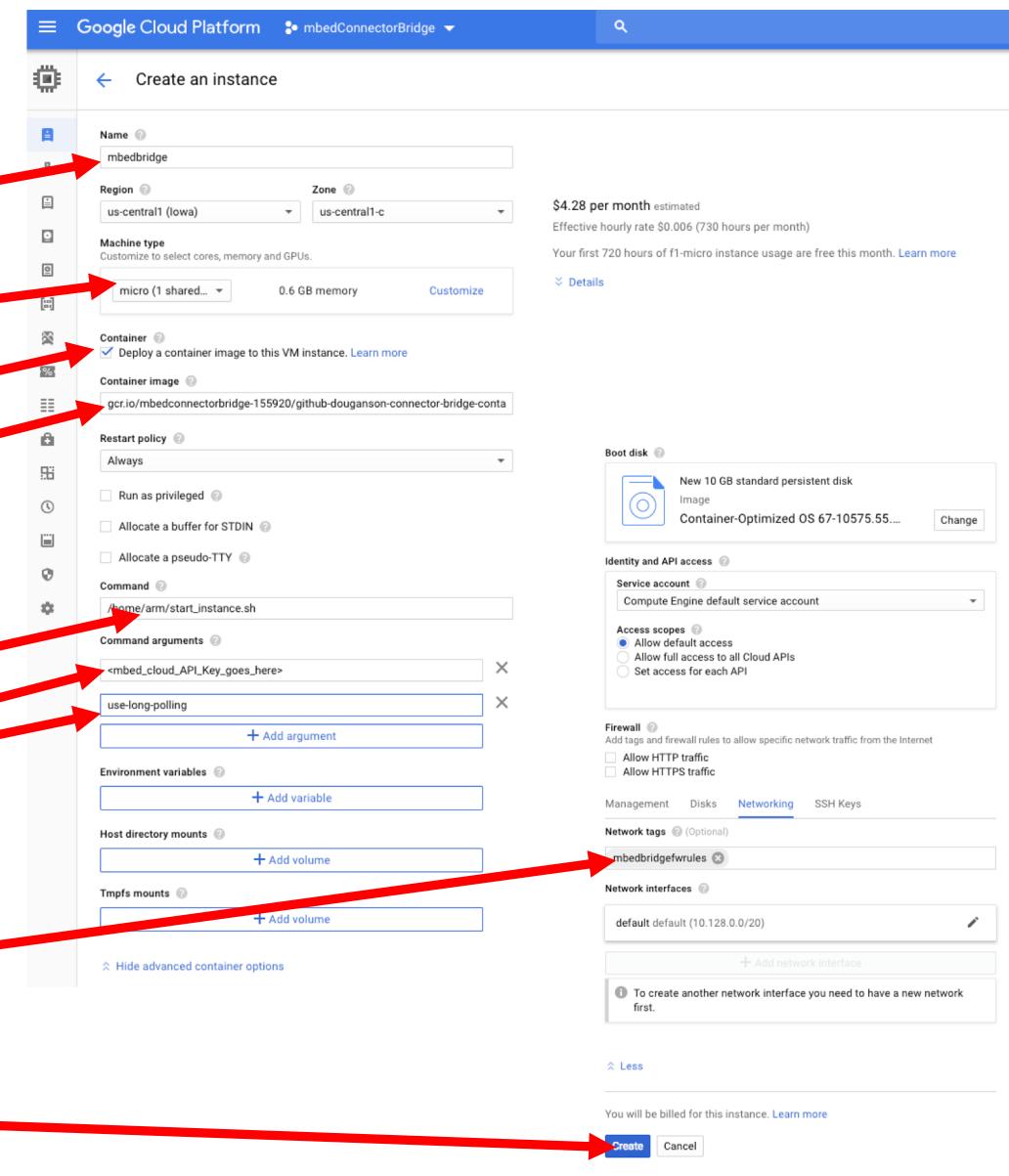
Command entered here

/home/arm/start_instance.sh

Command arguments

Network tag created from firewall rule

Press “Create”



Import the mbed Cloud prototype Bridge for Google from GCR...

Once Created, you should see this:

Give this instance about 5 minutes
to spin up and ready itself

Next, record this IP address

The screenshot shows the Google Cloud Platform interface for managing VM instances. The title bar says "Google Cloud Platform" and "mbedConnectorBridge". The main area is titled "VM instances" with a "CREATE INSTANCE" button. A red arrow points from the text "Next, record this IP address" to the "External IP" column of the table. The table has columns: Name, Zone, Recommendation, Internal IP, External IP, and Connect. One row is listed: "mbedbridge" in the Name column, "us-central1-c" in the Zone column, "10.128.0.2 (mbed)" in the Internal IP column, and "35.193.64.126" in the External IP column. The "Connect" column shows an "SSH" dropdown.

Name	Zone	Recommendation	Internal IP	External IP	Connect
mbedbridge	us-central1-c		10.128.0.2 (mbed)	35.193.64.126	SSH

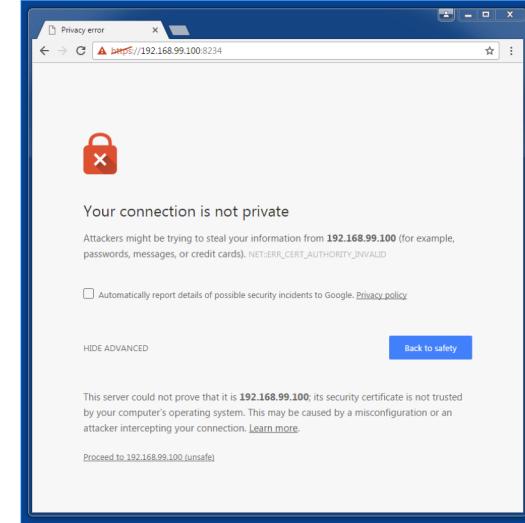
This IP address is your public-facing container IP
address... You'll need it in the next step...

Configure the mbed Cloud prototype Bridge for Google

Using your containers public-facing IP address:

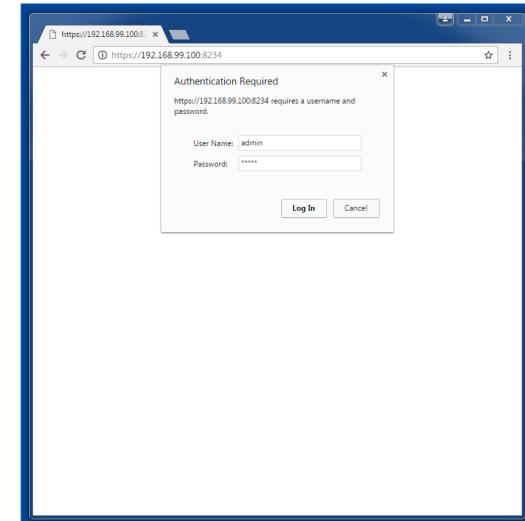
Bring up Firefox/Chrome and go to:

<https://<your public container IP address>:8234>



Accept the self signed certificate

Username: admin PW: admin



Configure the mbed Cloud prototype Bridge for Google...

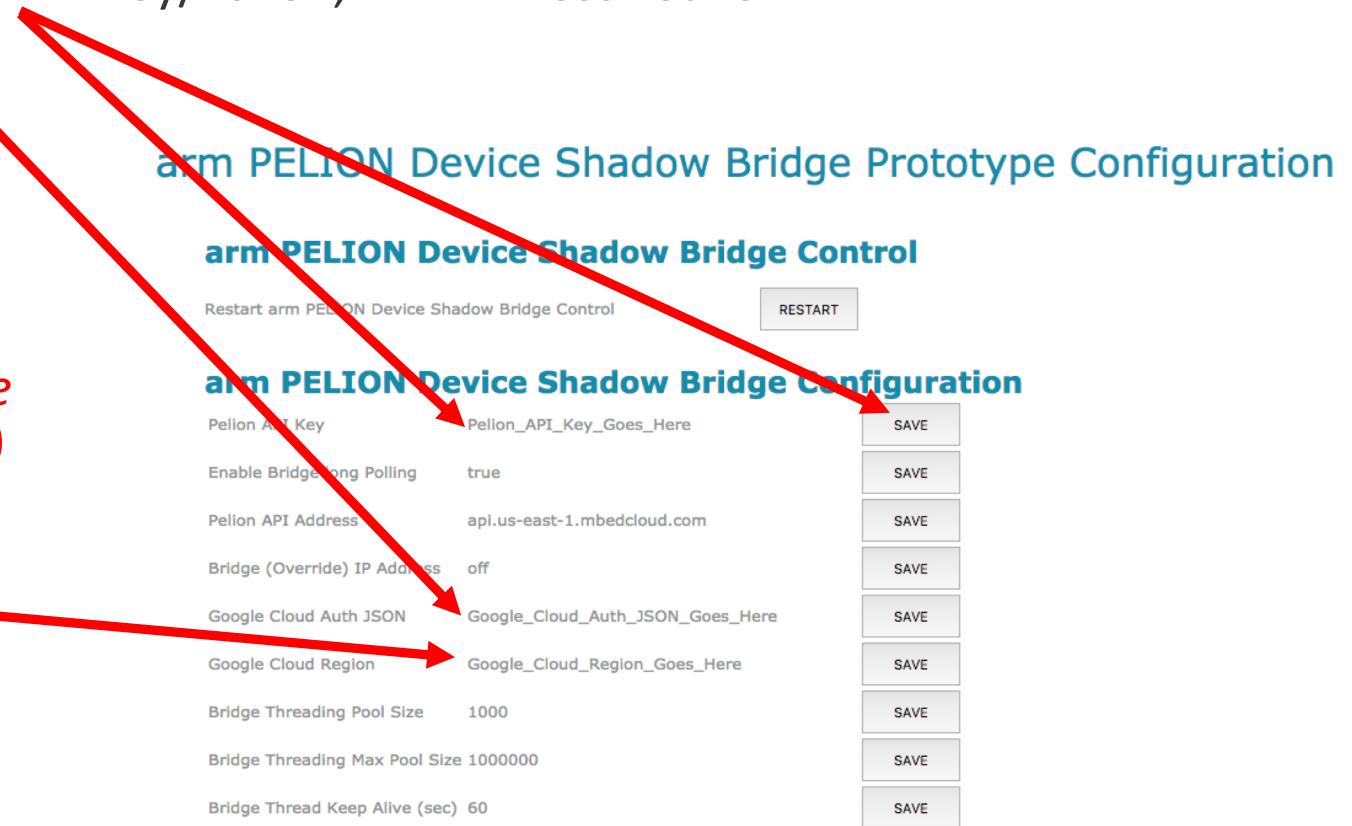
If not already entered, enter the mbed Cloud API Key/Token, *THEN* Press “Save”

Enter your Google Auth JSON, Press “Save”

*NOTE: make sure you are pasting raw text!
-html markup will botch the config file...you
then have to “./remove_bridge.sh”
and “./get_bridge.sh” to retry...(slide 26)
-if, after saving part of the configuration page
disappears, you'll have to re-pull per above)*

Enter your Google Cloud Region, then “Save”

Press “RESTART”



Your mbed Cloud Bridge is now configured for Google Cloud

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...

Device ID	Name	State	Date created	Date booted
01600a0b9bac00000000000000001001	01600a0b-9bac-0000-0000-000000000098	deregistered	November 29, 2017 5:10 PM	November 29, 2017 5:10 PM
01600bb435ab000000000000000010005c	01600bb4-35ab-0000-0000-00000000005c	deregistered	November 29, 2017 10:55 AM	-
01600bb4352800000000000000000100069	01600bb4-3528-0000-0000-000000000069	deregistered	November 29, 2017 10:55 AM	-
01600bb4352000000000000000000000000000068	01600bb4-3520-0000-0000-000000000000000068	deregistered	November 29, 2017 10:55 AM	-
015fce7fad2000000000000000000000000000000000000006	015fce7-fad2-0000-0000-0000000000000000000000000000000000000006	deregistered	November 17, 2017 3:34 PM	November 29, 2017 12:57 PM

Install the Google Exerciser Application

Install Java 8 JDK: <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

Install Java and the NetBeans IDE <http://www.netbeans.org> (Java SE support or “All”... either should be OK)

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

Import into NetBeans as a Maven Project

Select this source file

```
// CHANGE ME: Our Exerciser needs to know the device ID we wish to exercise...
// - Assumption: our device is running this project: https://github.com/ARMmbed/mbed-cloud-sample
// - Just Set the ENDPOINT_NAME below to the created device id for your endpoint
//   - look at the device attributes for the device id in the mbed Cloud portal
//
//private static final String ENDPOINT_NAME = "mbed Cloud device ID goes here";
private static final String ENDPOINT_NAME = "162457deaba0000000000010010021d";
// Registry Region - typically "us-central1" ... but yours may be different.
private static final String DEVICE_REGISTRY_REGION = "us-central1";
// Enable/Disable: We can Toggle the LED resource depending on /123/0/4567 being even or odd...
private static final boolean TOGGLE_LED = false;
// if your device is running https://github.com/ARMmbed/mbed-cloud-sample,
// nothing below needs to be changed... just leave it
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";
// Google Auth JSON gets copied into this file
// File Location: top-level directory for this project (ordinary text file with JSON copied/pasted)
private static final String GOOGLE_AUTH_JSON_FILE = "google_auth.json";
// Device registry connector-bridge uses this name specifically... so no need to change
private static final StringMBED_DEVICE_REGISTRY_NAME = "mbedDeviceRegistry";
// LED ON/OFF payloads
private String m_connector_led_put_ON = null;
private String m_connector_led_put_OFF = null;
```

Enter your mbed Cloud Endpoint Device ID you saved!

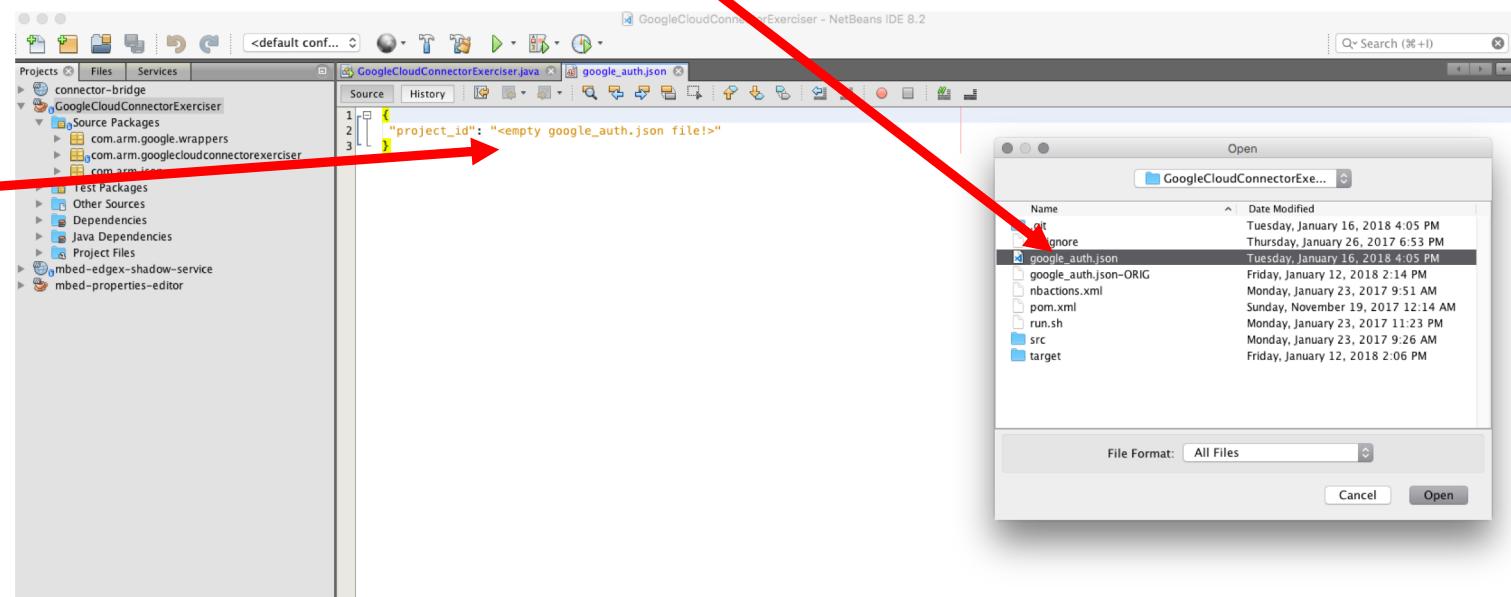
You can enable/disable the LED toggle (default is “false” or OFF)

Configuring the Google Exerciser Application (Auth)

In NetBeans, select File->Open.

Navigate to the project root directory and open “google_auth.json”

Paste the contents of your
Google auth json and save.



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 and CloudIoT API Instances

Import our prototype mbed Cloud Bridge docker runtime for Google via GCR

Configure our prototype mbed Cloud Bridge for Google

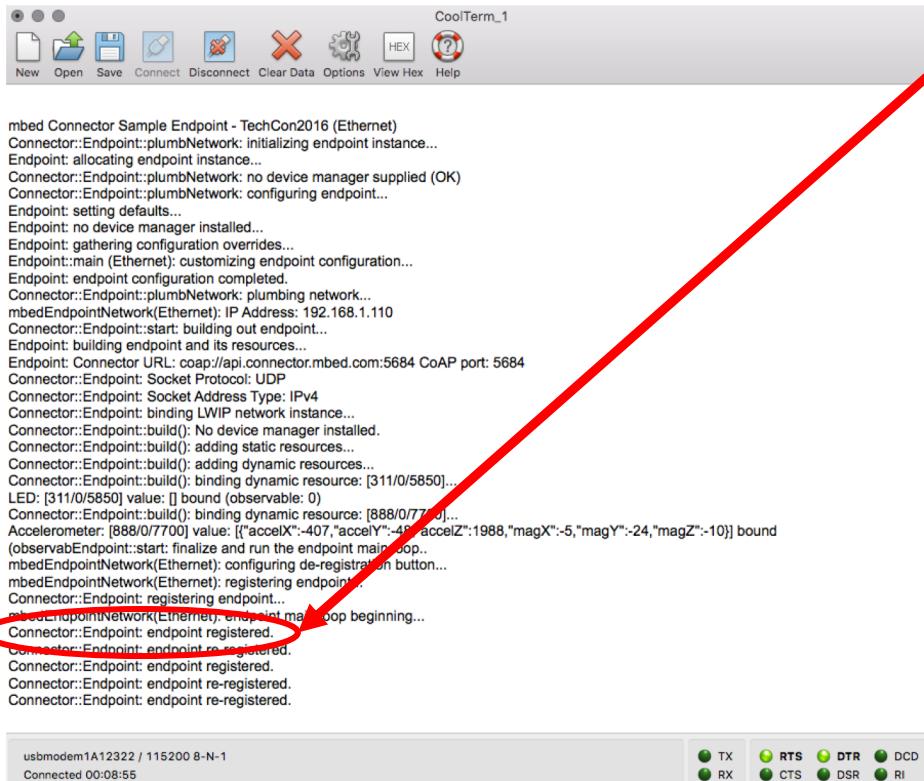
Installed and configured the “Exerciser” Application

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: Check the Serial Terminal

Press the deregister button (**SW3, slide 6**)... wait about a minute or so... now Reboot your endpoint.... You should see output that looks something like this:

Make sure that you see a message like “endpoint registered”



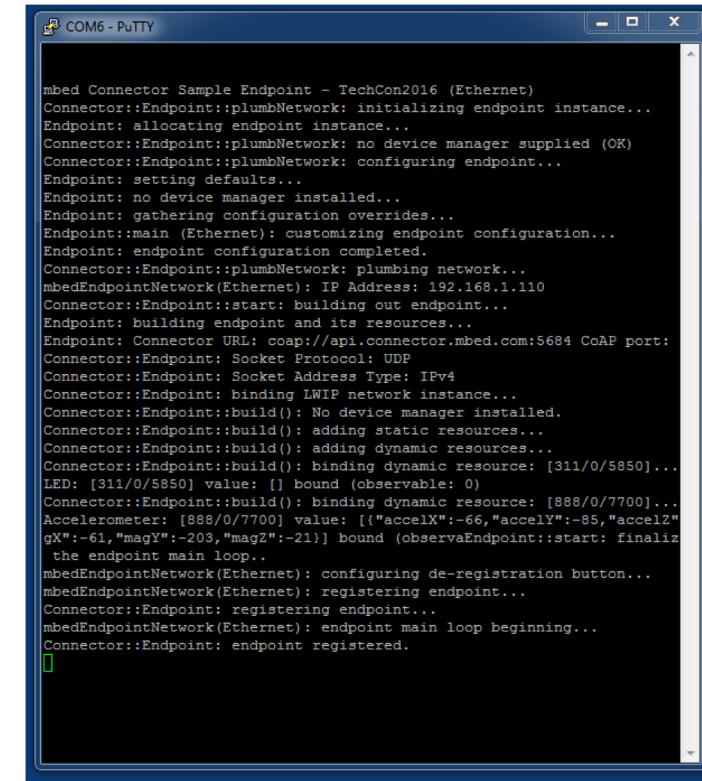
CoolTerm_1

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": -41, "accelZ": 1988, "magX": -5, "magY": -24, "magZ": -10}] bound
(observabEndpoint::start: finalize and run the endpoint main loop...
mbedEndpointNetwork(Ethernet): configuring de-registration button...
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": -21}] bound
(observabEndpoint::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.
```

Running the Google 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

GoogleCloudConnectorExerciser - NetBeans IDE 8.2

Projects Files Services

Source Packages

GoogleCloudConnectorExerciser

Source Packages

com.arm.google.wrappers

com.arm.googlecloudconnectorexerciser

com.arm.googlecloudconnectorexerciser

GoogleCloudConnectorExerciser.java

GoogleCloudConnectorMessageHandlerTh

com.arm.json

Test Packages

Other Sources

Dependencies

Java Dependencies

mbed-edgex-shadow-service

mbed-properties-editor

```
47 /**
48 * Google Cloud Connector Exerciser
49 *
50 * @author Doug Anson
51 */
52 public class GoogleCloudConnectorExerciser {
53     // Google Auth JSON gets copied into this file
54     // File Location: top-level directory for this project (ordinary text file with JSON copied/pasted)
55
56     private static final String GOOGLE_AUTH_JSON_FILE = "google_auth.json";
57
58     // These need to mimick the endpoint name, endpoint type, and CoAP resource uri used
59     // - for https://github.com/ARMmbed/mbed-cloud-sample, only set the ENDPOINT_NAME to
60     //   the created device name for your endpoint... all other settings are already OK.
61     //private static final String ENDPOINT_NAME = "<endpoint_name_goes_here>";
62     private static final String ENDPOINT_NAME = "015f168d318b0000000000001001003e3";
63     private static final String ENDPOINT_TYPE = "mbed-endpoint";
64     private static final String COUNTER_RESOURCE_URI = "/123/0/4567";
65     private static final String LED_RESOURCE_URI = "/311/0/5850";
66
67     // really nothing else below needs to be set... Please forward down to
68     // "IMPORTANT NOTE" to learn more about the expected subscription and topic
69     // mappings
70     // topics and subscriptions (Connector format)
71     private String m_connector_counter_topic = null;
72     private String m_connector_led_topic = null;
73     private String m_connector_led_put_ON = null;
74     private String m_connector_led_put_OFF = null;
75     private String m_connector_led_response_topic = null;
76 }
```

Search Results Output - Run (GoogleCloudConnectorExerciser)

Building GoogleCloudConnectorExerciser 1.0

exec-maven-plugin:1.2.1:exec (default-cli) @ GoogleCloudConnectorExerciser

Starting Idle Loop...

The Exerciser will turn the LED resource on and off depending on whether the monotonic resource observation value (resource /123/0/4567) is even or odd... this demonstrates the bi-directional nature of the bridge with your endpoint!

CONGRATS!! You are finished!



arm

Thank You!!