mangOH to GCloud Tutorial

Overview

This tutorial illustrates how to send data to a **Google Cloud IoT platform** from a **Legato** application running on a **AirPrime WP module** on **mangOH Green** or **mangOH Red** open source hardware platforms.

The tutorial details the source files of the Legato application and shows how to build, install and run the Legato application that sends MangOH native sensor data [temperature + gyroscope] to the cloud through MQTT protocol.

We'll build, install and run the application with the open source Linux-based embedded platform **Legato** toolchain and installation command line.

To quickly test this sample, just download the source code and follow the guidelines here;

Prerequisites

Setting up your mangOH board & Legato development environment

Before starting this tutorial, you must have followed mangoh Getting Started documentation on getting-started and ensured that your development environment is properly set up by building, compiling and running the Hello World Legato application.

Install MQTT client Legato application

MQTT is the protocol used to communicate with the Cloud in this tutorial.
You can find a MQTT client Legato application here.

You must build, install and run the client on your board before running any application using MQTT protocol.

Note that you will find all information in the MQTT client repository.

Setting up your Google IoT Cloud account

Furthermore, you must follow Google Cloud IoT documentation to setup your **gcloud** account and create your **project**, **registry**, **devices** and **channels**.

You must generate all public/private key pair registered in your Google Cloud IoT project.

Good to know

The **mangOH** board is using the **I2C Dev interface** module which allow accelerometer, gyroscope data and more others sensors acquisition.

Thanks to the full open hardware philosophy of project MangOH, you can find all design resources in the official website.

Theses are somes features available through the I2c interface:

- **Accelerometer** for super accurate movement sensing
- **Gyroscope** data
- Temperature
- Built-in **FIFO buffer**
- **Pedometer**
- **Shocks detection**
- **Drive interrupt** pins by embedded functions or by FIFO low-capacity/overflow warning.

Theses are interested links around this usecase:

- I2c device interface
- I2c specification

Application skeleton

```
~/mhirba/sws/mangoh-to-gcloud ❯❯❯ tree .
.
├──── Makefile
├──── ReadMe.md
├──── mangohToGCloud.adef
├──── mqtt.api
└──── myComponent
    ├──── Component.cdef
    ├──── config
    │     ├──── ec_private.pem
    │     ├──── ec_public.pem
    │     ├──── rsa_cert.pem
    │     └──── rsa_private.pem
     ├──── main.cc
    ├──── deviceToCloud.cc
    ├──── deviceToCloud.hh
    └──── utils
        ├──── nativeSensor.cc
        └──── nativeSensor.hh

2 directories, 10 files
```

You have **myComponent,** the main Legato component of our application and some utils (native sensor class) that you can use with other applications.

MPORTANT: the config directory must contain your generated public/private keys generated using openssl for **RSA key pair** or **Elliptic Curve key pair** and registered in you Google Cloud IoT during the last step -- cf. the chapter prerequisites.

SHORT CUT: the following chapters explain the structure and source code of the Legato application. To may want to jump immediately to the Compile, Install, Run section and come back here later to understand how the code works.

Application definition file

Let's take a look at our **mangohToGcloud.adef** file:

```
// application version
version: 1.0.0

// maximum amount of RAM that can be consumed by the temporary (volatile) file
system at runtime.
maxFileSystemBytes: 512K

// the app should start automatically at start-up
start: manual

// the app will be launched inside a sandbox.
sandboxed: true

executables:
{
  mangohToGCloud = ( myComponent )
}

processes:
{
  envVars:
  {
    LE_LOG_LEVEL = DEBUG
  }

  run:
  {
    ( mangohToGCloud )
  }

  // maximum size of core dump files.
  maxCoreDumpFileBytes: 512K

  // files are not allowed to grow bigger than this.
  maxFileBytes: 512K
}
```

```
bindings:
{
  // we want to use mqtt services (from an external app)
  mangohToGCloud.myComponent.mqtt -> mqttService.mqtt

  // we want to use the radio WAN app so that we can start an IP session over WAN
(3G)
  mangohToGCloud.myComponent.le_data -> dataConnectionService.le_data

  // we want to use le_mdc because we need to know if the WAN interface is up
  mangohToGCloud.myComponent.le_mdc -> modemService.le_mdc
}
```

Pay attention to the **'bindings'** section that allows us to connect the application to others services (**MQTT**, Legato data + mdc) and the **'requires'** section that allows access to the **I2C** bus.

Component

The main directory is our **myComponent**

```
// things the component needs from its runtime environment.
requires:
{
  // IPC APIs used by this component.
  api:
  {
    // \*.{h,c} files are created by building tools
    mqtt.api [manual-start]

    // allow WAN IP interface
    le_data.api [manual-start]

    // allow check on IP address and connection state
    modemServices/le_mdc.api
  }
  file:
  {
      /sys/devices/i2c-0/0-0068/iio:device0/in_accel_x_raw
/sys/devices/i2c-0/0-0068/iio:device0/
      /sys/devices/i2c-0/0-0068/iio:device0/in_accel_y_raw
/sys/devices/i2c-0/0-0068/iio:device0/
      /sys/devices/i2c-0/0-0068/iio:device0/in_accel_z_raw
/sys/devices/i2c-0/0-0068/iio:device0/
      /sys/devices/i2c-0/0-0076/iio:device1/in_temp_input
/sys/devices/i2c-0/0-0076/iio:device1/
  }
}

sources:
{
  utils/nativeSensor.cc
  sensorToGoogleCloud.cc
  main.cc
}


// command-line arguments to pass to the c++ compiler
cxxflags:
{
  -std=c++11
}
```

Pay attention to **IPC APIs** definition **'api'** section: MQTT, Data, etc.

Application code explained

This chapter focuses on the explanation of the main classes.

Utils class like NativeSensor is well commented and are simply wrappers from common specifications. Let's explore our simple entry point:

Main

```
COMPONENT_INIT
{
  LE_INFO("Hi, from mangohToGoogleCloud app!");

   // instantiate our application as a singleton
  SensorToGoogleCloud* s = SensorToGoogleCloud::getInstance();
  s->start();
}
```

We decide to create a singleton for more flexibility and to handle callback stuff with ease.

DeviceToCloud class

- The first thing to do is managing termination/cleaning handlers.

```
SensorToGoogleCloud::SensorToGoogleCloud() : _is_mqtt_connected(false)
{
  // blocks SIGTERM in the calling thread
  le_sig_Block(SIGTERM);

  // sets SIGTERM event handler for the calling thread.
  le_sig_SetEventHandler(SIGTERM, DeviceToCloud::appTerminationCallback);

  // register appCleanup as the method to be called on exit
  ::atexit(DeviceToCloud::appCleanUp);
}
```

- Generate Json Web Token for authentication:

```
void SensorToGoogleCloud::generateJWT()
{
    // Reading the private key file
    std::ifstream ifs("config/rsa_private.pem");
    std::string key((std::istreambuf_iterator<char>(ifs)),
                    (std::istreambuf_iterator<char>()));


    // Validate private key with RS256 encryption
    RS256Validator signer(key.c_str());

    // About expiration time
    std::time_t iat = std::time(nullptr);

    std::tm expiration_date = *std::localtime(&iat);
    expiration_date.tm_year += 1;

    std::time_t exp = mktime(&expiration_date);

    // Creating the json payload that expires 1 year from today
    json_ptr json(json_pack("iat", iat, "exp", exp, "aud",
GOOGLE_CLOUD_PROJECT_ID));

    // Let's encode the token to a char[]
    _jwt = JWT::Encode(&signer, json.get());

    LE_INFO("GENERATED TOKEN: %s", _jwt.get());
}
```

- Configure and connecting MQTT publisher is pretty simple

```
void DeviceToCloud::configureMQTT()
{
  // connect to external services
  le_data_ConnectService();

  // connect to mqtt service
  mqtt_ConnectService();

  LE_INFO("MQTT CONNECTED!");
  // an experiment to see if we can close mqtt if has been left connected
  if(mqtt_GetConnectionState()) {
    DeviceToCloud::appCleanUp();
  }

  // lambda for mqtt session
  auto mqttSessionCallback = [](bool is_connected, int32_t connect_err_code,
int32_t sub_err_code, void* ctxt_ptr) {
    if(is_connected) {
      DeviceToCloud::getInstance()->setMqttSession(true);
    }
    else {
      LE_INFO("MQTT DISCONNECTED - error:%d and sub erorr: %d", connect_err_code,
sub_err_code);
    }
  };

  // setting up the mqtt service
  mqtt_AddSessionStateHandler(mqttSessionCallback, NULL);

  // configure mqtt session with AirVantage informations
  mqtt_Config(MQTT_BRIDGE_HOSTNAME, MQTT_BRIDGE_PORT, MQTT_KEEP_ALIVE, MQTT_QOS);
}
```

Thanks to the **mqtt.api** we can use all **MQTT** client functions.

For configuration we are using values defined in our header, this is a sample where the password is stored as **JWT** generated our device with private key

- Time to check our main loop

```
void DeviceToCloud::start()
{
  // create timer with an interval of every 10 seconds, repeating forever (0)
  le_clk_Time_t repeat_interval;

  repeat_interval.sec  = 10;
  repeat_interval.usec = 0;
  // create our timer object
  _timer_ref = le_timer_Create("mainLoopTimer");

  // lambda for our loop acquisition
  auto callback = [](le_timer* timer) {
    DeviceToCloud* s = DeviceToCloud::getInstance();

    // data acquisition
    s->fetch();
    s->dump();

    // -->> ☁
    s->send();
  };

  // about timer configuration
  le_timer_SetHandler(_timer_ref, callback);
  le_timer_SetInterval(_timer_ref, repeat_interval);
  le_timer_SetRepeat(_timer_ref, 0);
  le_timer_Start(_timer_ref);

  return;
}
```

We just create a timer calling a simple loop to fetch, display and send acquired data.

- Get native sensor data

```
void SensorToGoogleCloud::fetch()
{
    // get last temperature from LSM6DS3
     _ns.readTemperature(&_last_temperature);

    // same for X->Z gyroscope axis
    _ns.readGyro(&_last_gyroscope.Ax, &_last_gyroscope.Ay,
&_last_gyroscope.Az);
}
```

- Send data to the cloud with MQTT

```
void DeviceToCloud::send()
{
  // reference to the modem data connection profile (here: default profile)
  le_mdc_ProfileRef_t profile = le_mdc_GetProfile(LE_MDC_DEFAULT_PROFILE);

  // connection state value
  le_mdc_ConState_t mdc_state;

  char json_payload[2048];
  int32_t json_err_code;

  // get WAN connected info
  le_mdc_GetSessionState(profile, &mdc_state);

  // check that we have WAN
  if(mdc_state == LE_MDC_DISCONNECTED) {
    le_data_Request();
    LE_INFO("WAN CONNECTED");
  }
  else if(!_is_mqtt_connected) {
    // check that mqttService is up
    mqtt_Connect(MQTT_BROKER_PASSWORD);
    LE_INFO("MQTT CONNECTED");
  }
  else {
    // format our data into json
    sprintf
      (json_payload,
       "{"
       "\"sensor.temperature\":%f,"
       "\"sensor.AX\":%f,"
       "\"sensor.AY\":%f,"
       "\"sensor.AZ\":%f"
       "}",
       _last_temperature,
       _last_gyroscope.Ax,
       _last_gyroscope.Ay,
        _last_gyroscope.Az);

    // send playload to AirVantage
    mqtt_SendJson(json_payload, &json_err_code);
  }
}
```

The purpose is to simply verify connection, format last acquired data and send a playload as a **MQTT** publisher.

Compile, Install, Run

In your development environment, call **make** and specify the target (here ‘wp85’ for Sierra Wireless AirPrime WP85xx module) and install the Legato application onto the board:

```
❯❯❯ make wp85
mkapp -v -t wp85 \
-i myComponent/utils/i2c \
-i myComponent/utils/lsm6ds3 \
mangohToGCloud.adef
Command-line arguments from previous run not found.
Parsing file: '/srcstmp/mangohToGCloud.adef'.
Modelling application: 'mangohToGCloud'
  defined in: '/srcstmp/mangohToGCloud.adef'
Application 'mangohToGCloud' contains executable 'mangohToGCloud'.
...
❯❯❯ ls -l *update
-rw-r--r--@ 1 majdi  staff  30914 Feb 16 16:48 mangohToGCloud.wp85.update
❯❯❯ app install mangohToGCloud.wp85.update 192.168.2.2
Applying update from file 'mangohToGCloud.wp85.update' to device at address
'192.168.2.2'.
Unpacking package: 100% ++++++++++++++++++++++++++++++++++++++++++++++++++
Applying update: 100% ++++++++++++++++++++++++++++++++++++++++++++++++++
SUCCESS
Done
```

Finally, check that the application is running on your board by login in onto your mangOH board target:

```
❯❯❯ ssh root@192.168.2.2
root@swi-mdm9x15:~# app status
[running] audioService
...
[running] mqttService
[stopped] mangohToGCloud
root@swi-mdm9x15:~# app start mangohToGCloud
root@swi-mdm9x15:~# logread
...
Jul 25 17:06:37 swi-mdm9x15 user.info Legato:  INFO |
mangohToGCloud[8312]/myComponent T=main | main.cc _myComponent_COMPONENT_INIT() 24
| Hi, from mangohToGoogleCloud app!
...
```

After building and installing our new application, we can see on the logread terminal that we have some data outputs:

```
root@swi-mdm9x15:~# logread -f
Jul 25 17:07:57 swi-mdm9x15 user.info Legato:  INFO |
mangohToGCloud[8312]/myComponent T=main | deviceToCloud.cc dump() 143 | DUMP:
24.875000 0.017080 -0.025376 -0.990640
Jul 25 17:07:57 swi-mdm9x15 user.debug Legato:  DBUG |
mangohToGCloud[8312]/framework T=main | le_mdc_client.c le_mdc_GetProfile() 950 |
Sending message to server and waiting for response : 4 bytes sent
Jul 25 17:07:57 swi-mdm9x15 user.debug Legato:  DBUG |
mangohToGCloud[8312]/framework T=main | le_mdc_client.c le_mdc_GetSessionState()
1542 | Sending message to server and waiting for response : 4 bytes sent
Jul 25 17:07:57 swi-mdm9x15 user.debug Legato:  DBUG |
mangohToGCloud[8312]/framework T=main | mqtt_client.c mqtt_SendJson() 786 | Sending
message to server and waiting for response : 101 bytes sent
Jul 25 17:07:57 swi-mdm9x15 user.info Legato:  INFO |
mqttService[6918]/mqttServiceComponent T=main | mqttService.c mqtt_SendJson() 198 |
send
json({"sensor.temperature":24.875000,"sensor.ax":0.017080,"sensor.ay":-0.025376,"se
nsor.az":-0.990640})
Jul 25 17:07:57 swi-mdm9x15 user.info Legato:  INFO |
mqttService[6918]/mqttServiceComponent T=main | mqttService.c
mqttService_SendMessageJson() 83 | topic('359377060005641/messages/json')
payload('{"sensor.temperature":24.875000,"sensor.ax":0.017080,"sensor.ay":-0.025376
,"se
```

Result in Google Cloud console

After building and installing our new application, open the Google Cloud console and check the data is coming in.