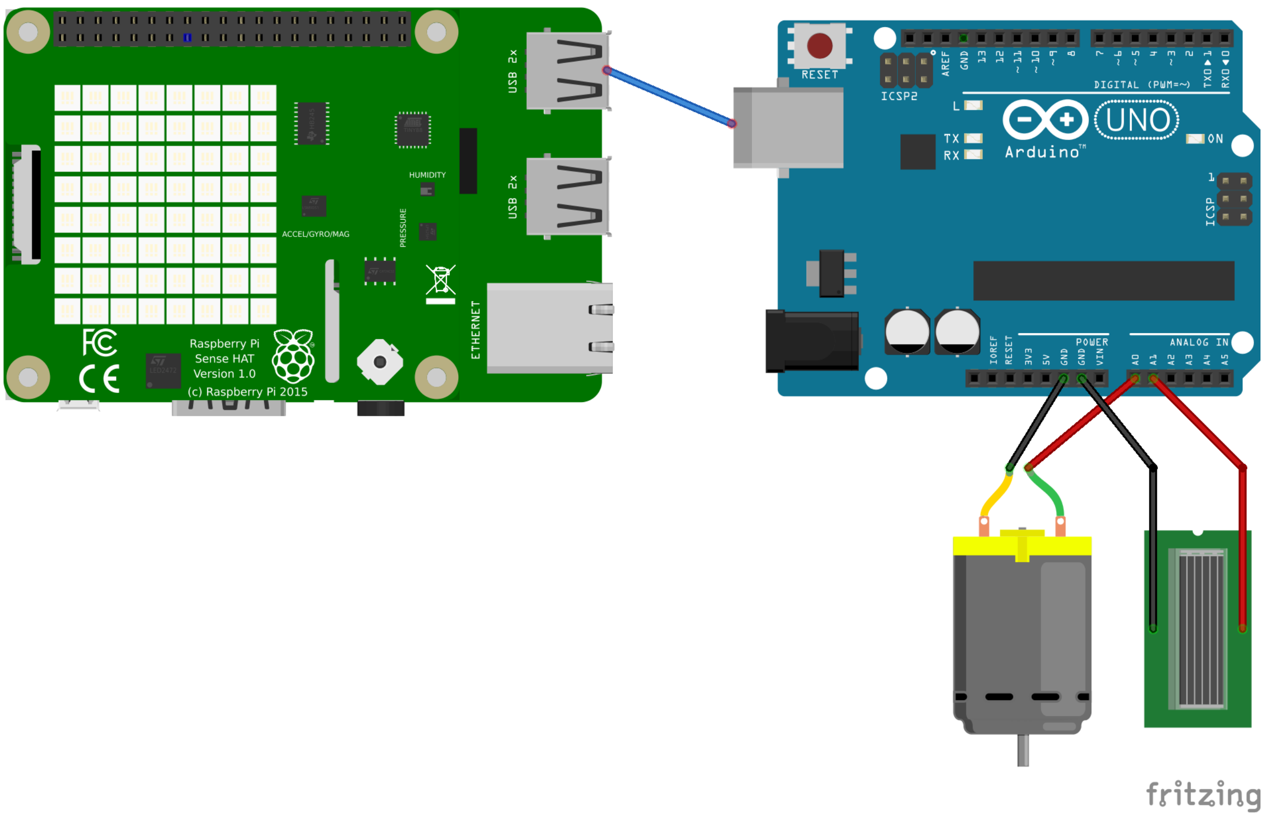
Markku Lepisto | Solutions Architect | Google Cloud

This two-part tutorial demonstrates how to control an Arduino Microcontroller (https://www.arduino.cc/) with a Raspberry Pi (https://www.raspberrypi.org/), connect the devices to Google Cloud IoT Core (https://cloud.google.com/iot-core/), post sensor data from the devices, and analyze the data in real time. Part 1 of the tutorial creates a 'hybrid' device, combining the strengths of a Linux-based microprocessor with internet connectivity and TLS stack, together with a constrained microcontroller for analog I/O.

Part 1 objectives

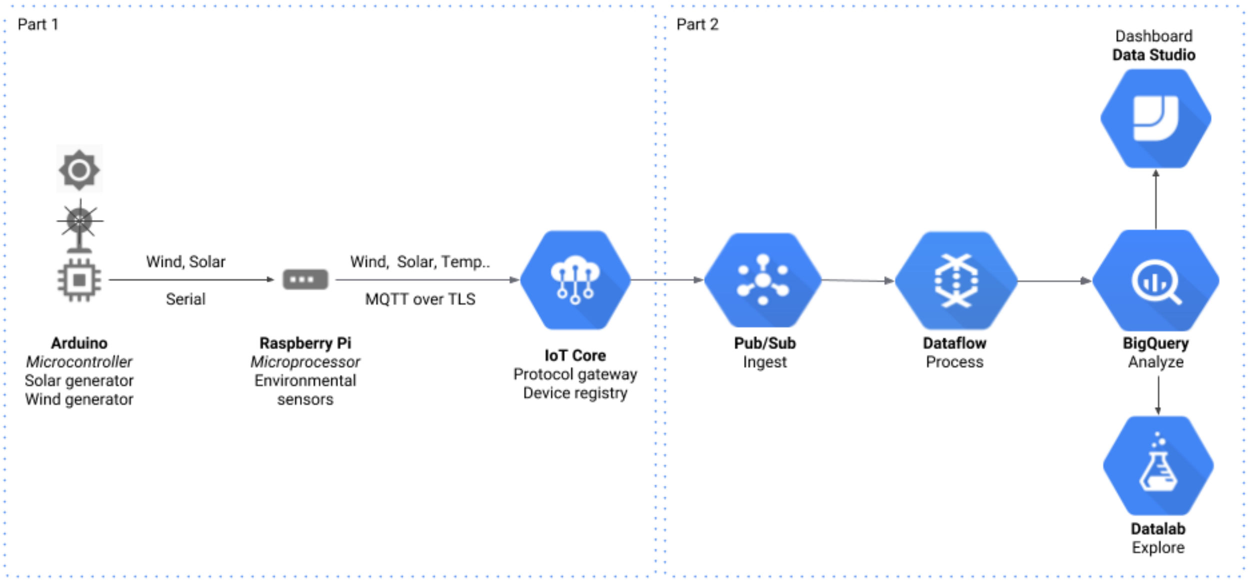
- Program the Arduino with the Sketch (https://www.arduino.cc/en/tutorial/sketch) language
- Read analog sensor values
- Transmit the data over a serial connection to the Raspberry Pi
- Program the Pi in Python (https://www.python.org/), to read digital sensor values
- Post the data to Google Cloud IoT Core over a secure MQTT (http://mqtt.org/) connection

Figure 1. *IoT boards and components*



In part 2 of the tutorial, you will learn how to process, store and analyze the streaming data in real time.

Figure 2. *End to end architecture*



Before you begin

This tutorial assumes you already have a Cloud Platform (https://console.cloud.google.com/freetrial) account set up. You will also need to download and install the Arduino IDE (https://www.arduino.cc/en/Main/Software) on your local development environment.

Costs

This tutorial uses billable components of GCP, including:

- Cloud IoT Core
- Cloud Pub/Sub

This tutorial should not generate any usage that would not be covered by the free tier (https://cloud.google.com/free/), but you can use the Pricing Calculator (https://cloud.google.com/products/calculator/) to generate a cost estimate based on your projected production usage.

Required hardware

- Raspberry Pi 3 Model B (https://www.raspberrypi.org/products/raspberry-pi-3-model-b/) or B+ (https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/). **Note:** the instructions should work on other Pi models as well, but this has not been verified
- Raspberry Pi case (optional but recommended)
- Raspberry Pi power supply, rated at 2.5A or more
- MicroSD card for Raspbian OS. Minimum 8GB card recommended

- Raspberry Pi Sense HAT (https://www.raspberrypi.org/products/sense-hat/) add-on board
- Arduino Uno Genuino (https://www.arduino.cc/en/Guide/BoardAnatomy) or 100% compatible microcontroller
- Two (recommended) analog sensors for the Arduino. In this tutorial we use a small solar panel (https://www.adafruit.com/product/700) and a small DC motor (https://www.sparkfun.com/products/11696) to generate analog voltage readings
- USB type A to type B cable (to interconnect the Pi and Arduino)
- USB keyboard
- Monitor with HDMI input
- HDMI cable

Create an IoT Core registry and device

First, execute all the steps in the IoT Core Quickstart (https://cloud.google.com/iot/docs/quickstart) *on your local development environment e.g laptop*. But *do not clean up* the environment. The rest of this tutorial will use the registry, device and device keys created with the quickstart.

Raspberry Pi installation and configuration

Raspbian OS installation and configuration

1. Follow the instructions here (https://www.raspberrypi.org/documentation/installation/installing-images/README.md) to download the latest Raspbian Lite OS image (https://www.raspberrypi.org/downloads/raspbian/), and write it on your MicroSD card. All the steps in this tutorial run in console mode on the Pi, or over SSH to the Pi.
2. Insert your Pi in the Pi case.
3. Plug your Sense HAT into the Pi’s 40-pin connector.
4. Insert the MicroSD card into the Pi card slot.
5. Connect the USB keyboard into the Pi.
6. Connect the Pi to your monitor with the HDMI cable.
7. Optionally, if you use Ethernet connectivity, connect your ethernet cable to the Pi.
8. Connect the Pi power supply to the wall socket and to the Pi, to boot up Raspbian OS.
9. Login to the Pi with the initial username: `pi` and password: `raspberry`
10. Execute raspi-config as the superuser with: `$ sudo raspi-config`
11. Change the pi user’s password by selecting: `1 Change User Password`
12. If you are using Wi-fi, select: `2 Network Options > N2 Wi-fi` to configure your Wi-fi country settings, SSID and passphrase.

13. To get most console apps to display text correctly, select: `4 Localisation options > I1 Change Locale` . Scroll down with the down arrow key, and deselect the default option: `en_GB.UTF-8 UTF-8` using the space bar.
14. Scroll down to `en_US.UTF-8 UTF-8` and select that option with the space bar.
15. Press Tabulator and select: `<Ok>`
16. Use the arrow keys in the `Configuring locales` window to select `en_US.UTF-8` from the list of choices as the default locale. Press Tabulator and select: `<Ok>`
17. To set your system timezone, select: `4 Localisation options > I2 Change Timezone` and use the menu options to find your geographic region and country. Setting the timezone is important because Google IoT Core requires client devices' system clocks to be within 10 minutes of real time. Raspbian uses global NTP time servers to synchronize the clock.
18. Optionally, select: `4 Localisation options > I3 Change Keyboard Layout` to match your layout. If you are not sure, select: `Generic 105-key (Intl) PC > English (US) > The default for the keyboard layout > No compose key` .
19. Select: `Interfacing Options > P2 SSH > Yes` to enable the SSH server on your Pi.
20. Select: `Interfacing Options > P5 I2C > Yes` to load the ARM I2C kernel module. The Raspberry Pi Sense HAT sensor module used in this tutorial uses I2C.
21. Select: `<Finish>` , to exit the raspi-config tool and accept to `Reboot` the Pi. If you are not prompted to reboot when you exit raspi-config, reboot the Pi by executing: `$ sudo reboot`
22. Log in as the `pi` user with your new custom password.
23. Verify that you have an IP address on your Pi by executing: `$ ifconfig -a` You should see a client IP from your local network segment in either the `wlan0` or `eth0` interface.
24. Verify that your DNS resolution and internet connection work, by executing: `$ ping www.google.com`
25. If you do not have an IP address, or DNS resolution or internet connectivity fail, do not proceed further until you have fixed your Pi's network configuration.
26. For Wi-fi connectivity only - check if your Wi-fi adapter has power saving enabled. We have to disable Wi-fi power saving, or it will give you sporadic network connectivity issues later on. Check the current state by executing: `$ iwconfig` Look for "Power Management" under the `wlan0` interface. If you see `Power Management:on` , then we have to disable it.
27. On Pi 3 model B/B+, disable `wlan0` Wi-fi power management by configuring `/etc/network/interfaces` . Execute: `$ sudo vi /etc/network/interfaces` or use the nano editor if you prefer. Add these lines to the end of the file:

```
allow-hotplug wlan0
iface wlan0 inet manual
    # disable wlan0 power saving
    post-up iw dev $IFACE set power_save off
```

28. Reboot the Pi with: `$ sudo reboot` to apply the changes. Then log in again as the `pi` user.
29. Execute: `$ iwconfig` to ensure that Wi-fi power management is now: `Power Management:off` for `wlan0` .
30. Execute: `$ ping www.google.com` to check your internet connectivity.
31. Upgrade the Raspbian packages to latest versions with:

```
$ sudo apt-get update && sudo apt-get upgrade -y
```

Install Cloud SDK and dependencies for Pi Sense HAT

1. SSH to the Raspberry Pi with: `$ ssh pi@<your_Pi_IP>`
2. On the Pi, follow all the steps here (https://cloud.google.com/sdk/docs/#deb) to install and initialize Cloud SDK for Debian systems.
3. Check that Cloud SDK is installed and initialized with: `$ gcloud info` . Ensure that the `Account` and `Project` properties are set correctly.
4. Install Git, Python development packages and Sense HAT dependencies with:

```
$ sudo apt-get update && sudo apt-get install -y git virtualenv
python-dev sense-hat libffi-dev libssl-dev libopenjp2-7 libjpeg-dev
```

Clone the Pi client app and install its dependencies

1. Using the SSH connection to the Pi, clone the repo associated with the community tutorials:

```
$ git clone https://github.com/GoogleCloudPlatform/community.git
```

2. Change to the app's directory:

```
$ cd community/tutorials/ardu-pi-serial/part1
```

3. Create a Python virtual environment:

```
$ virtualenv venv
```

4. Activate the virtualenv:

```
$ source venv/bin/activate
```

5. Upgrade pip and setuptools:

```
(venv) $ pip install -U pip setuptools
```

6. Install the required Python modules:

```
(venv) $ pip install -r requirements.txt
```

7. Copy the private key file that you generated from IoT Core Quickstart (https://cloud.google.com/iot/docs/quickstart) guide to the app's current working directory on the Pi. You can use `scp` on Linux and Mac OS, to copy the file from your local machine to the Pi, with for example:

```
$ scp rsa_private.pem pi@<your_Pi_IP>:ardu-pi-serial/
```

   You can check your Pi's IP address by executing: `$ ifconfig` on the Pi.

Arduino Uno installation and configuration

Connect the sensors to the Arduino

1. Connect 2 analog sensors to the Arduino. See Figure 1 (/Users/markku/GoogleDrive/src/python/ardu-pi-serial-part-1/ardu-pi.png) to see how to wire the 2 sensors to the Arduino. In this example, we will read the analog voltage values of a DC motor, and a solar panel. *Note: you can replace the DC motor and solart panel with other analog sensors - but you* ***have to ensure that the sensor output does not exceed 5V*** *or it can damage the Arduino pin.*
2. Connect the first sensor (small DC motor) to the Arduino. Connect the black ground wire of the motor to a `GND` port, and the red (voltage input/output) wire to port `A0` on the Arduino.
3. Connect the second sensor (small solar panel) to the Arduino. Connect the black ground wire of the panel to a `GND` port, and the red (voltage output) wire to port `A1` on the Arduino.

Compile the Arduino Sketch

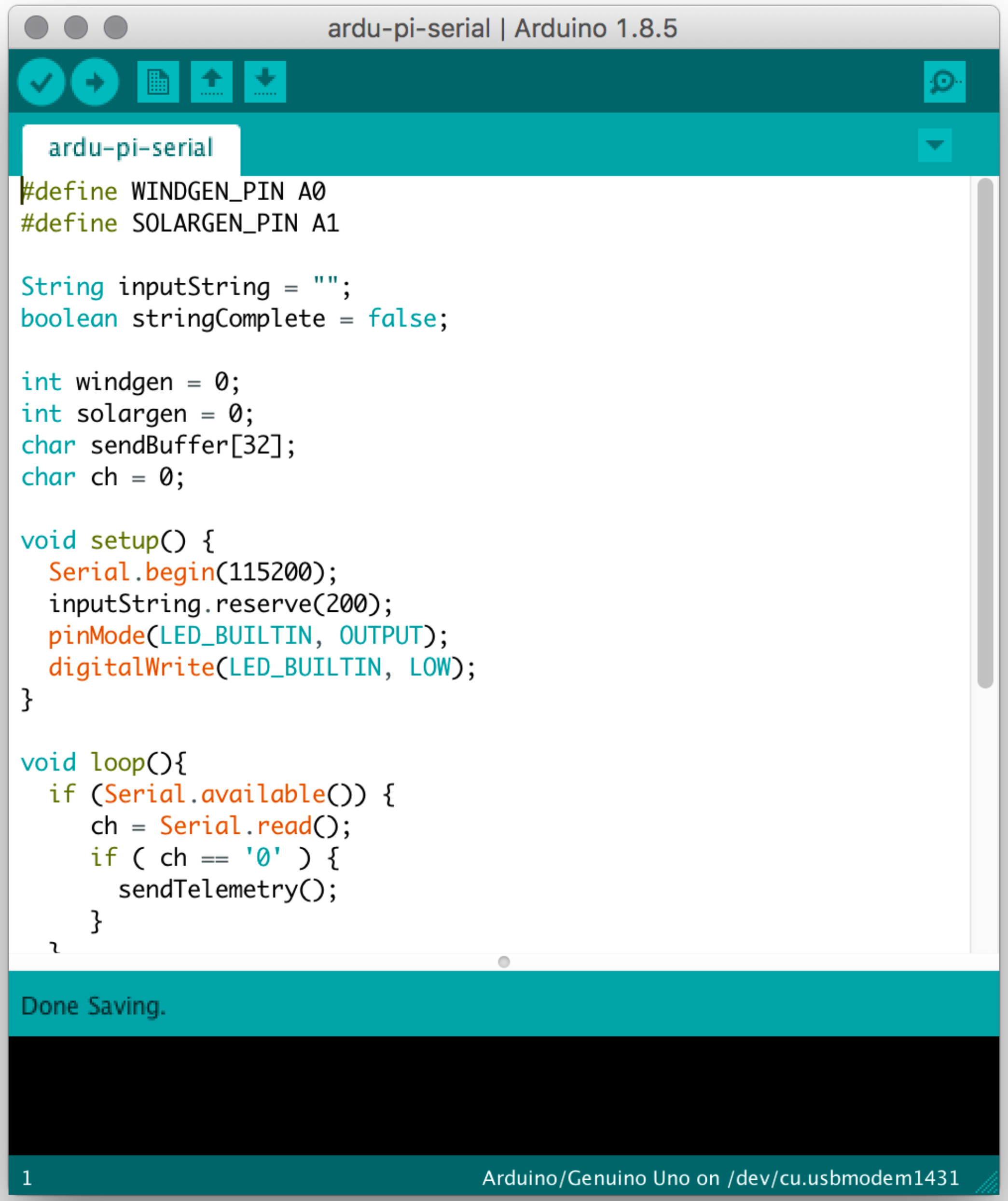
1. Connect your Arduino to your local development machine with a USB cable. Note that the Arduino Uno board used as the example in this tutorial, has a USB type B connector on the board. The USB cable will provide both power, and serial data connectivity to the Arduino from your machine.

2. Launch the Arduino IDE software on your local machine.
3. Go to the `Arduino menu bar > Tools > Board` and select `Arduino/Genuino Uno`.
4. Check the `Tools > Port` options and select the port on your computer that has `Arduino/Genuino Uno` visible and connected.
5. Select `File > New` to open a new Sketch.
6. Delete all the text in the new Sketch template.
7. Copy and paste this code into your new Sketch:

```
#define WINDGEN_PIN A0
#define SOLARGEN_PIN A1

String inputString = "";
boolean stringComplete = false;

int windgen = 0;
int solargen = 0;
char sendBuffer[32];
char ch = 0;

void setup() {
  Serial.begin(115200);
  inputString.reserve(200);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
}

void loop(){
  if (Serial.available()) {
     ch = Serial.read();
     if ( ch == '0' ) {
       sendTelemetry();
     }
  }
}

void sendTelemetry() {
  digitalWrite(LED_BUILTIN, HIGH);
  memset(sendBuffer, 0, sizeof(sendBuffer));
  int windgen = analogRead(WINDGEN_PIN);
  int solargen = analogRead(SOLARGEN_PIN);
  sprintf(sendBuffer, "S s:%d w:%d", solargen, windgen);
  Serial.println(sendBuffer);
  digitalWrite(LED_BUILTIN, LOW);
}
```

1. Select `File > Save As...` and save your sketch with a suitable name such as `ardu-pi-serial`

Figure 3. *Arduino IDE and Sketch*


ardu-pi-serial | Arduino 1.8.5
ardu-pi-serial

```
#define WINDGEN_PIN A0
#define SOLARGEN_PIN A1

String inputString = "";
boolean stringComplete = false;

int windgen = 0;
int solargen = 0;
char sendBuffer[32];
char ch = 0;

void setup() {
  Serial.begin(115200);
  inputString.reserve(200);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
}

void loop(){
  if (Serial.available()) {
     ch = Serial.read();
     if ( ch == '0' ) {
       sendTelemetry();
     }
```

```
Done Saving.
```

1
Arduino/Genuino Uno on /dev/cu.usbmodem1431

Test reading Arduino sensors over serial connection from your machine

The Arduino Sketch client has the following functionality:

- Check if the serial connection has data available to read
- If the received character (command) is 0:
  - Read the voltage values of pins A0 and A1
  - Print the sensor values to the serial connection

Perform the following steps to deploy and test the Arduino sketch:

1. Select `Sketch > Verify/Compile` from the Arduino IDE menu. If the code is successfully verified, the Arduino IDE should display “Done compiling”. You should also see output similar to the following:

   ```
   Sketch uses 4708 bytes (14%) of program storage space. Maximum is
   32256 bytes.
   Global variables use 249 bytes (12%) of dynamic memory, leaving 1799
   bytes for local variables. Maximum is 2048 bytes.
   ```

2. Select `Sketch > Upload` to transfer the compiled Sketch to your Arduino over the USB serial connection. You should see the Arduino’s LEDs blink rapidly while the binary is transferred over.
3. Select `Tools > Serial Monitor` to open the Serial Monitor window.
4. Ensure that the baud rate is set to “115200 baud”.
5. Test requesting the current sensor values from the Arduino, by typing `0` (zero) in the Serial Monitor’s input window, and pressing `<enter>`. If the Sketch and serial connection works, you should see a response similar to: `S s:136 w:43` in the Serial Monitor.

Here, `S` indicates sensor output, `s:136` is the current value of the solar panel (analog pin `A1`) and `w:43` is the current value of the wind generator / DC motor (analog pin `A0`). This is how the Python client app on the Pi will request the sensor readings from the Arduino later on. Command `0` is an example of a simple protocol.

Different inputs (e.g `1`) could be used to command the Arduino to perform a different action, such as turning a servo motor. In this tutorial, the Arduino Sketch implements a single command `0`, which reads the voltage values of analog pins `A0` and `A1`.

- Unplug the USB cable from your local machine, to disconnect and power down the Arduino. The Arduino will now run the sketch automatically, when ever it is connected to a power source.

Connect the Arduino to the Pi and run the IoT Core client app

The Pi Python client has the following functionality:

- Sign a JWT token with the device private key
- Create and connect an IoT Core client
- Create a Sense HAT sensor board client
- Open the serial connection to the Arduino
- Read the Sense HAT environmental sensors
- Request the analog sensor readings from Arduino over the serial connection

- Publish the sensor readings as a JSON document once a second
- Re-generate and sign a new JWT token after it expires

1. SSH to the Raspberry Pi with:

   ```
   $ ssh pi@<your_Pi_IP>
   ```

2. Connect the Arduino to the Pi with a USB cable. This will power on the Arduino, and create a USB serial port on the Pi, for serial communications.
3. On the Pi, check that you now have a serial port connected to the Pi. Execute:

   ```
   $ file /dev/ttyACM0
   ```

   The output should be similar to:

   ```
   /dev/ttyACM0: character special (166/0)
   ```

4. With your virtualenv active, run the client app on the Pi, by executing:

   ```
   (venv) $ python ardu-pi-serial.py  \
       --project_id <your_project_id> \
       --registry_id my-registry \
       --device_id my-device \
       --private_key_file rsa_private.pem \
       --algorithm RS256 \
       --cloud_region us-central1 \
       --ca_certs roots.pem \
       --message_type event \
       --device_type pi
   ```

   If everything works, you should see output similar to this:

   ```
   Creating JWT using RS256 from private key file rsa_private.pem
   Creating and flushing serial port. Rebooting Arduino..
   ('gcp_on_connect', '0: No error.')
   Received message '' on topic '/devices/my-device/config' with Qos 1
   Sleeping 3s..
   Received from Arduino: S s:89 w:42
   Publishing message: {"temperature": 37.32, "timestamp": "2018-07-
   25T08:29:59+08:00", "clientid": "raspberrypi", "humidity": 37.98,
   "pressure": 1016.98, "windgen": 0.21, "solargen": 0.43}
   gcp_on_publish
   ```

5. On your local development machine, create a subscription for your IoT Core registry's telemetry topic, by executing:

```
$ gcloud pubsub subscriptions create my-device-events-sub --
topic=my-device-events
```

6. Verify that your telemetry data is streaming to Cloud Pub/Sub, by executing:

   ```
   $ gcloud pubsub subscriptions pull my-device-events-sub --limit 100
   --auto-ack
   ```

   You should see message payloads similar to:

   ```
   {"temperature": 36.62, "timestamp": "2018-07-25T08:42:10+08:00",
   "clientid": "raspberrypi", "humidity": 38.44, "pressure": 1017.05,
   "windgen": 0.1, "solargen": 1.2}   │ 151338809303544 │ deviceId=my-
   device
   ```

   Now your Arduino and Pi are streaming telemetry data through Cloud IoT Core to Pub/Sub. You should see the sensor readings change if you expose the solar panel to more, or less light. And the wind generator (DC motor) will start generating power, as you turn the rotors.

Next

In part 2 of this tutorial, you will use Cloud Dataflow with a Python streaming pipeline, to process this data stream, store it in Cloud BigQuery, explore the data with a Cloud Datalab Jupyter Notebook, and visualize it using Cloud Data Studio. Stay tuned for part 2.