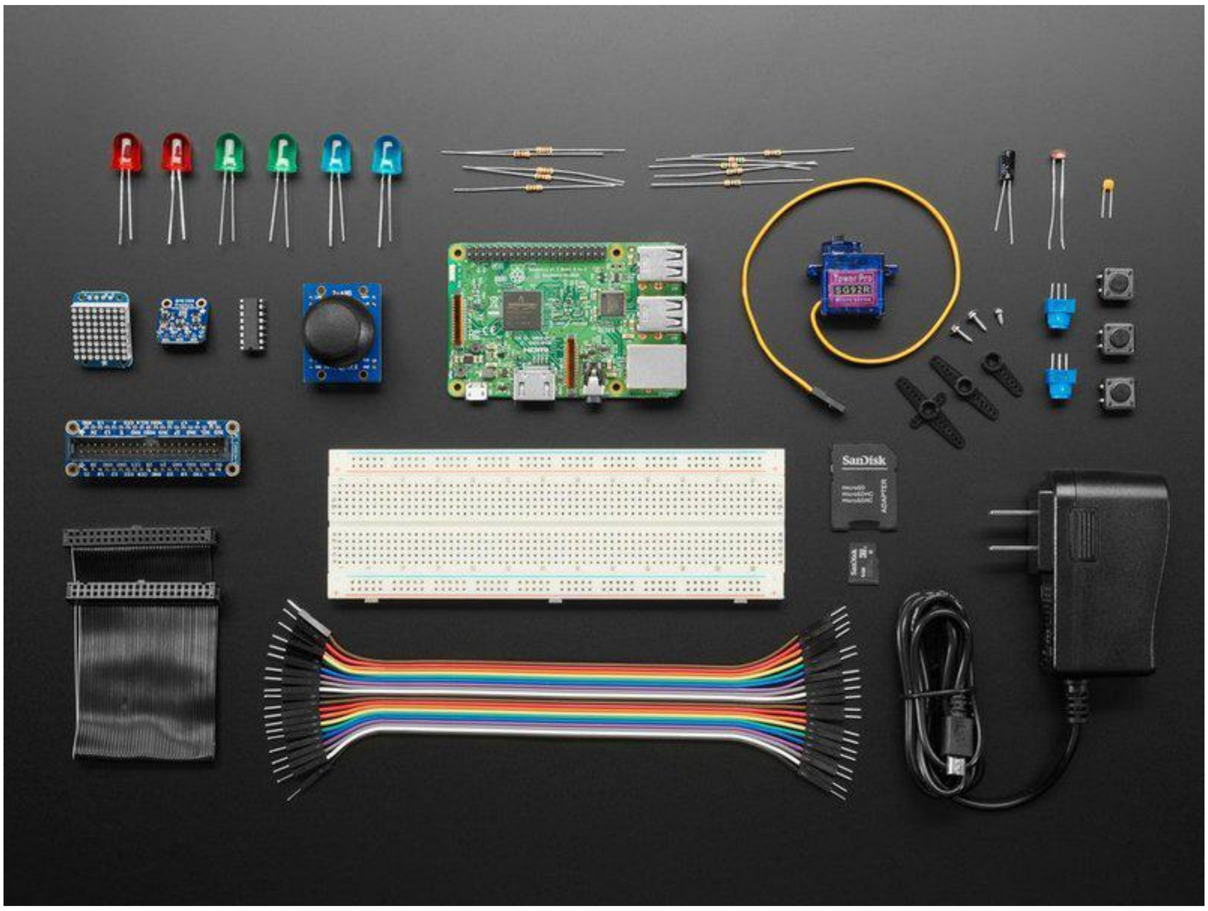
Arm-based IoT Kit for Cloud IoT Core - Getting Started

Created by Matthew DuPuy

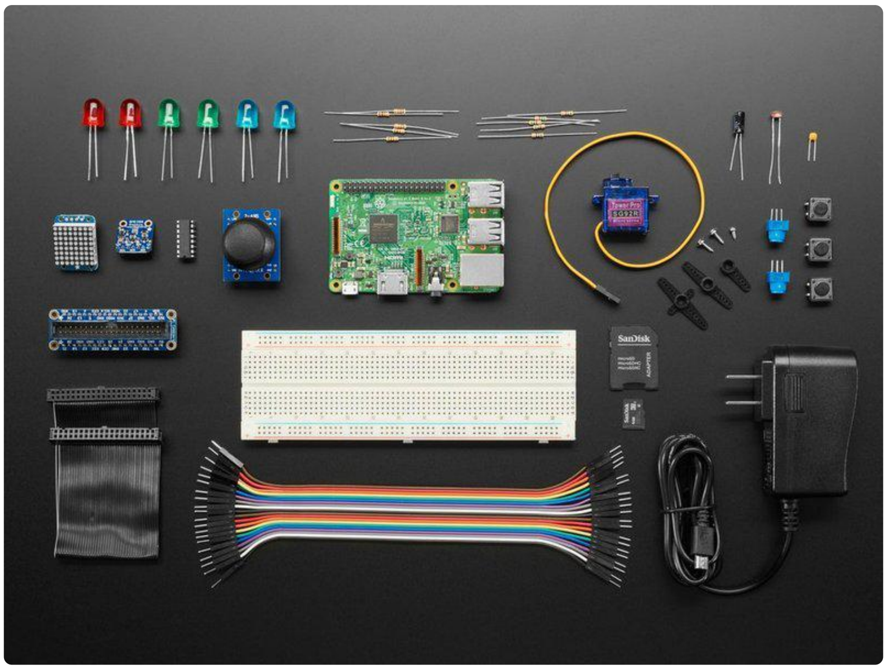


https://learn.adafruit.com/raspberry-pi-3-and-sensor-kit-for-google-cloud-iot-core

Last updated on 2022-12-01 03:00:54 PM EST

Table of Contents

Arm-based IoT Kit for Cloud IoT Core



Example projects and code on GitHub are supplied to support the Arm-based IoT Kit for Cloud IoT Core.

This getting started guide assumes you have enabled Google Cloud IoT Core in your GCP concole. It will step you through setting up the gcloud SDK tools, getting your Raspberry Pi 3 connected to the internet and securely registered as a device with a Google Cloud project.

Parts included in kit:

- 1x Pi3
- 1x Power Supply
- 1x I2C Temp/Pressure/Humidity Sensor
- 1x 8-Channel 10-bit ADC with SPI
- 1x 8Gb SD with latest Raspian pre-loaded
- 1x Breadboard
- 1x Servo
- 1x Joystick (pluggable into breadboard)
- 1 x Small pixel screen
- 1x Photo cell- CdS photoresistor
- 1x Premium Male/Male Jumper Wires - 20 x 6" [150mm]
- 1x Assembled Pi Cobbler Plus - Breakout Cable - for Pi B+/A+/Pi 2/Pi 3

- 1x 8 Channel ADC
- 5x 10K 5% 1/4W Resistor
- 5x 560 ohm 5% 1/4W Resistor
- 1x Diffused 10mm Blue LED
- 1x Electrolytic Capacitor - 1.0uF
- 2x Diffused 10mm Red LED
- 2x Diffused 10mm Green LED
- 2x Diffused 10mm Blue LED
- 2x Breadboard Trim Potentiometer
- 3x 12mm Tactile Switches

Example projects and code are supplied to support the Arm-based IoT Kit for Cloud IoT Core (kit including RasPi3) from Adafruit

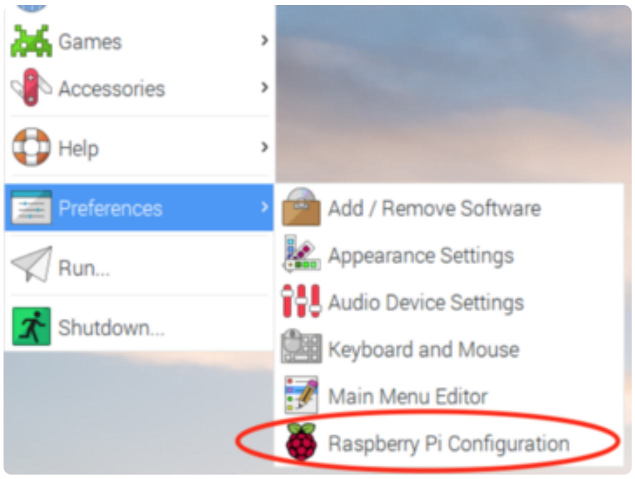
---

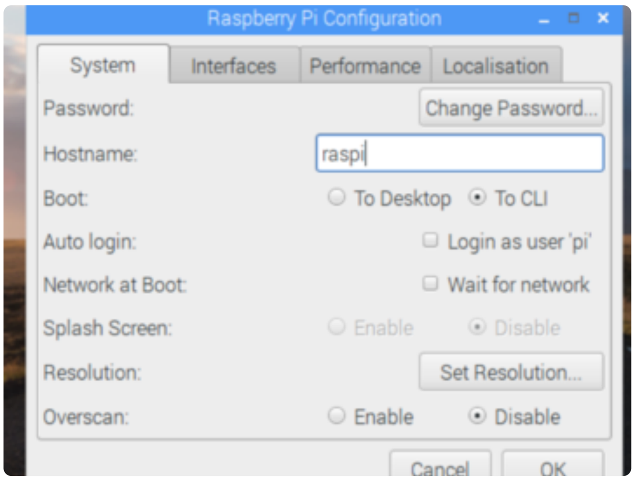
Getting Started

If you purchased the kit that includes the Raspberry Pi 3 Model B, this comes with a pre-formatted NOOBS microSD card. Simply inserting the card into the Pi and powering up the Pi with the included 5v micro USB power supply will boot the Pi and with no interaction, it will default to installing the Raspbian Linux distribution. This is what we want. There are many ways to get a Raspbian and the Pi set up for Google Cloud IoT Core functionality but this guide will focus on getting Raspbian on your WiFi network and headless with secure shell running, gcloud tools installed and IoT Core dependencies for Python installed. These steps will require an HDMI monitor, USB keyboard and mouse.

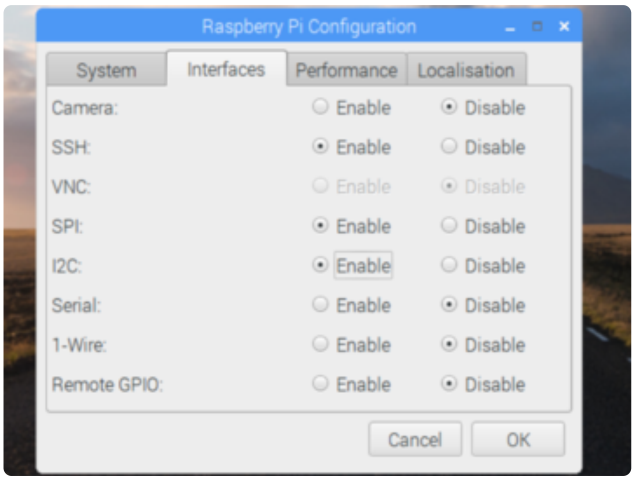
Network and firmware updates

1. Hook up an HDMI monitor, USB keyboard and mouse (plug in an Ethernet cable if you do not intend to use WiFi) then power up your Pi. Once booted, use the WiFi menu in the upper right hand corner of the screen (it should appear with two red 'x's on boot) to connect to the SSID of the wireless network you wish to use. This assumes your network has a DHCP service running on it. If your network has corporate security features, please use another guide appropriate to the type of security required [most require creative use of the wpa_supplicant command and configuration in /etc].

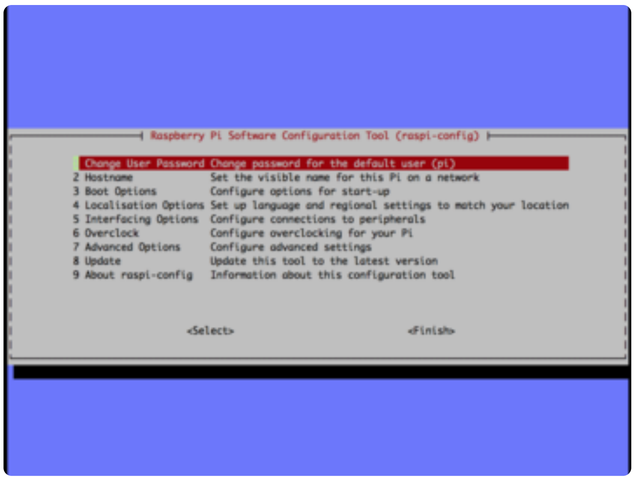




Password:
Hostname: raspi
Boot: To Desktop [✓ To CLI]
Auto login: Login as user 'pi'
Network at Boot: Wait for network
Splash Screen: Enable [✓ Disable]
Resolution:
Overscan: Enable [✓ Disable]



Camera: Enable [✓ Disable]
SSH: [✓ Enable] Disable
VNC: Enable [✓ Disable]
SPI: [✓ Enable] Disable
I2C: [✓ Enable] Disable
Serial: Enable [✓ Disable]
1-Wire: Enable [✓ Disable]
Remote GPIO: Enable [✓ Disable]



```
Raspberry Pi Software Configuration Tool (raspi-config)

 Change User Password Change password for the default user (pi)
 2 Hostname             Set the visible name for this Pi on a network
 3 Boot Options         Configure options for start-up
 4 Localisation Options Set up language and regional settings to match your location
 5 Interfacing Options  Configure connections to peripherals
 6 Overclock            Configure overclocking for your Pi
 7 Advanced Options     Configure advanced settings
 8 Update               Update this tool to the latest version
 9 About raspi-config   Information about this configuration tool

              <Select>                       <Finish>
```

2. Use the Raspberry menu to access Preferences->Raspberry Pi Configuration. Under the system tab you can change the hostname to whatever you like and set Boot to CLI (not Desktop); this is optional. Under the Interfaces tab enable "ssh" if you intend to use the Pi without a keyboard and monitor going forward. Enable SPI and I2C while you're there. Under the Localisation tab, set up your Locale, Time Zone and Keyboard preferences. A reboot is required after this. All of these options are also available with the raspi-config command in a terminal shell.

3. Once rebooted and connected to a network we can secure shell into our Pi remotely or use the command line directly to update our Linux distro and Raspberry Pi 3 firmware. The default uersname is "pi", default password is "raspberry ". To get the Pi's IP, use the command "ifconfig" or nmap your subnet for new ssh services. However you connect, update your Pi with the following commands and change your pi's default password with the "passwd" command if you so choose.

Get root access for updates

```
sudo -s
```

This step can take a while due to the number of packages installed by default on the Pi, feel free to uninstall the wolfram-engine, browsers, office applications, etc. at your discretion before running the updates

```
apt update && apt upgrade && apt dist-upgrade
```

Update the pi firmware (most likely requires a reboot after completion)

```
rpi-update && reboot
```

note: you can change most boot, bus and, interface options with a curses interface as well using *sudo raspi-config* i.e. enabling the i2c interface

---

Enabling Cloud IoT Core AP, installing the Google Cloud SDK and registering your first device

Before you proceed please ensure you are logged into Google via your browser with the same userid and password you used with gcloud init on your development machine.

The Google Cloud SDK can be installed on another host machine or the Pi itself. These steps will get the gcloud command installed on the Pi but it can just as easily be done on any machine that you do your development on.

1. Create a Cloud Platform project and enable the Cloud IoT Core API using these "Before you begin ()" directions.

2. Install the latest Google Cloud Tools () with the included directions. In Linux some of the additions require "sudo gcloud" to be used so you'll need to authorize your root account with sudo in addition to your 'pi' account so instructions from here will diverge from those included here (). Simply follow the directions below instead if you are installing gcloud on the Pi rather than another host machine. SSHing into your Pi (headless) is strongly advised in order facilitate authentication of your accounts with your normal desktop browser using copy/paste.

```
sudo gcloud components repositories add https://
storage.googleapis.com/cloud-iot-gcloud/components-json
```

3. Create shell variables with your specific project name from step 1 as well as region, registry, device, subscription and event names. Fill in your project ID from step 1, the rest can remain as is below and used in your .profile or .bashrc. i.e.

```
project=my-project-name-1234
region=us-central1
registry=example-registry
device=my-rs256-device
device2=my-es256-device
mysub=my-sub
events=events
```

4. Create a new registry using the gcloud command.

```
gcloud iot registries create $registry \
--project=$project \
--region=$region \
--pubsub-topic=projects/$project/topics/$events
```

5. Create a public/private key pair for your device and create a new device in your project and registry. Or, stretch goal, register one programmatically with [these code samples](https://cloud.google.com/iot/docs/device_manager_samples).

```
openssl req -x509 -newkey rsa:2048 -keyout rsa_private.pem -nodes -
out rsa_cert.pem
```

```
gcloud iot devices create $device \
--project=$project \
--region=$region \
--registry=$registry \
--public-key path=rsa_cert.pem,type=rs256

openssl ecparam -genkey -name prime256v1 -noout -out ec_private.pem
openssl ec -in ec_private.pem -pubout -out ec_public.pem

gcloud iot devices create $device2 \
--project=$project \
--region=$region \
--registry=$registry \
--public-key path=ec_public.pem,type=es256
```

6. Create a new pubsub subscription to an event

```
gcloud pubsub subscriptions create projects/$project/subscriptions/
$mysub --topic=$events
```

7. Download the CA root certificates from pki.google.com into the same directory as the example script you want to use:

```
wget https://pki.google.com/roots.pem
```

Dependencies

Our initial examples for this kit will focus on Python but it is entirely possible to use Ruby, Java, C and other languages to work with Google Cloud IoT. Dependencies include a JSON Web Token and MQTT library as well as a SSL/TLS library like OpenSSL. You'll need the following to run any of the examples included in this repository.

```
sudo -s
apt install build-essential libssl-dev libffi-dev python-dev
pip install pyjwt paho-mqtt cryptography
pip install --upgrade google-api-python-client
pip install --upgrade google-cloud-core
pip install --upgrade google-cloud-pubsub
exit
```

Hello World - Temperature example

See CPUTemp example's code to verify your device can communicate with your gcloud project.

CPUTemp Example

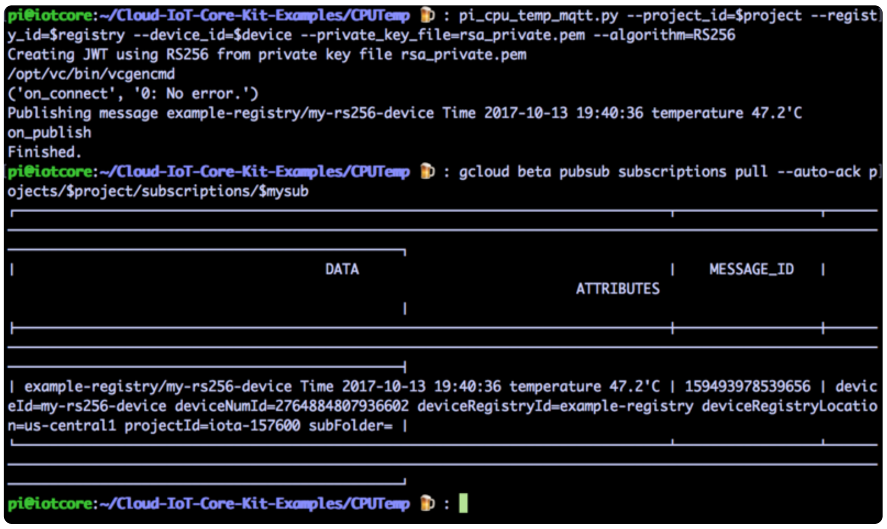
This example is the our "Hello World" for our Raspberry Pi 3 setup. This should verify that you are able to send JWT encoded messages with MQTT to your Google Cloud project registery topic

On your Pi export or set $project $registry and $device varialbes to your own and run:

```
pi_cpu_temp_mqtt.py --project_id=$project --registry_id=$registry --device_id=$device --private_key_file=rsa_private.pem --algorithm=RS256
```

gcloud command to fetch CPU temperature:

```
gcloud pubsub subscriptions pull --auto-ack projects/$project/subscriptions/$mysub
```



```
pi@iotcore:~/Cloud-IoT-Core-Kit-Examples/CPUTemp 🍺 : pi_cpu_temp_mqtt.py --project_id=$project --regist
y_id=$registry --device_id=$device --private_key_file=rsa_private.pem --algorithm=RS256
Creating JWT using RS256 from private key file rsa_private.pem
/opt/vc/bin/vcgencmd
('on_connect', '0: No error.')
Publishing message example-registry/my-rs256-device Time 2017-10-13 19:40:36 temperature 47.2'C
on_publish
Finished.
pi@iotcore:~/Cloud-IoT-Core-Kit-Examples/CPUTemp 🍺 : gcloud beta pubsub subscriptions pull --auto-ack p
ojects/$project/subscriptions/$mysub
┌──────────────────────────────────────────────────────────────────────────────┬──────────────┬──────
─────────────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────────────┐
|                                 DATA                                         |  MESSAGE_ID  |
                                                              ATTRIBUTES
                                        |
├──────────────────────────────────────────────────────────────────────────────┼──────────────┼──────
─────────────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────────────┤
| example-registry/my-rs256-device Time 2017-10-13 19:40:36 temperature 47.2'C | 159493978539656 | devic
eId=my-rs256-device deviceNumId=2764884807936602 deviceRegistryId=example-registry deviceRegistryLocatio
n=us-central1 projectId=iota-157600 subFolder= |
└──────────────────────────────────────────────────────────────────────────────┴──────────────┴──────
─────────────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────────────┘
pi@iotcore:~/Cloud-IoT-Core-Kit-Examples/CPUTemp 🍺 :
```

Find more samples and documentation at the Google Cloud Platform IoT site.

Useful code

Python driver for BME280 Temp/Pressure/Humidity Sensor
Adafruit Python GPIO library
Servo control with wiringpi
Servo control with Tkinter
Small Pixel Screen (8x8 Backpack) driver
Adafruit MCP3008 ADC Library (Setup Guide)
Adafruit Fritzing Library

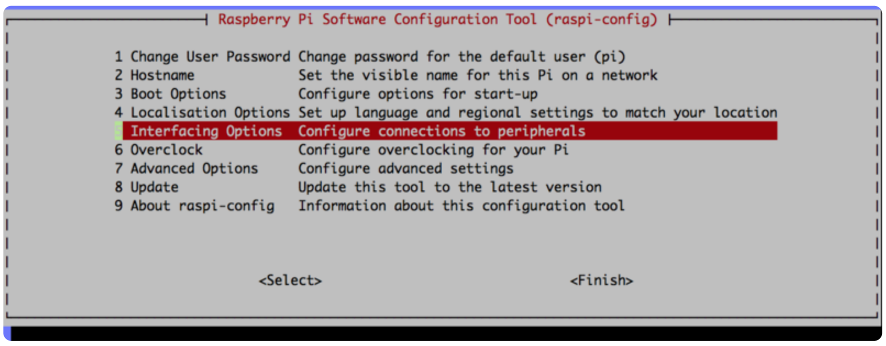
Pubsub thermostat example

This example will use the kit's temperature/pressure/humidity sensor to monitor temperature and control a fan in a complete IoT system with both a server and device component. The devices in this system (your Cloud IoT Core kit(s) in this case) publish temperature data on their pubsub registry feeds and individual device IDs. A server python application, which you can run from any machine you like, consumes the telemetry data from your Cloud Pub/Sub topic and events. The server then decides whether to turn on or off the individual devices' fans via a Cloud IoT Core configuration update.

This example requires i2c to be enabled in order to read the temperature sensor included with this kit. If you haven't already enabled i2c during your initial setup, please run

```
sudo raspi-config
```

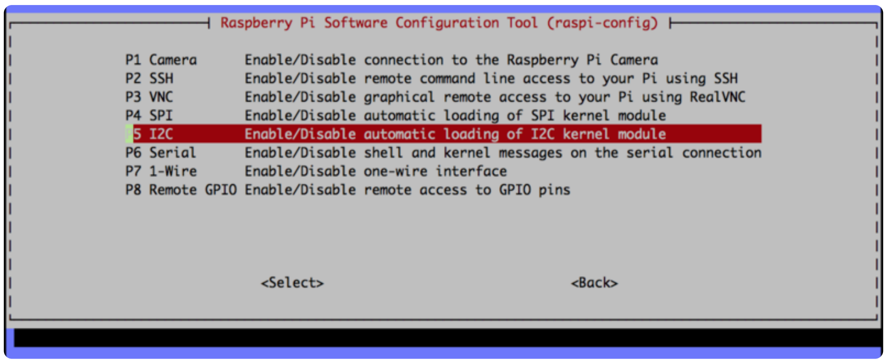
Go to Interfacing Options->I2C and enable.



```
Raspberry Pi Software Configuration Tool (raspi-config)

1 Change User Password Change password for the default user (pi)
2 Hostname             Set the visible name for this Pi on a network
3 Boot Options         Configure options for start-up
4 Localisation Options Set up language and regional settings to match your location
5 Interfacing Options  Configure connections to peripherals
6 Overclock            Configure overclocking for your Pi
7 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool

                 <Select>                    <Finish>
```



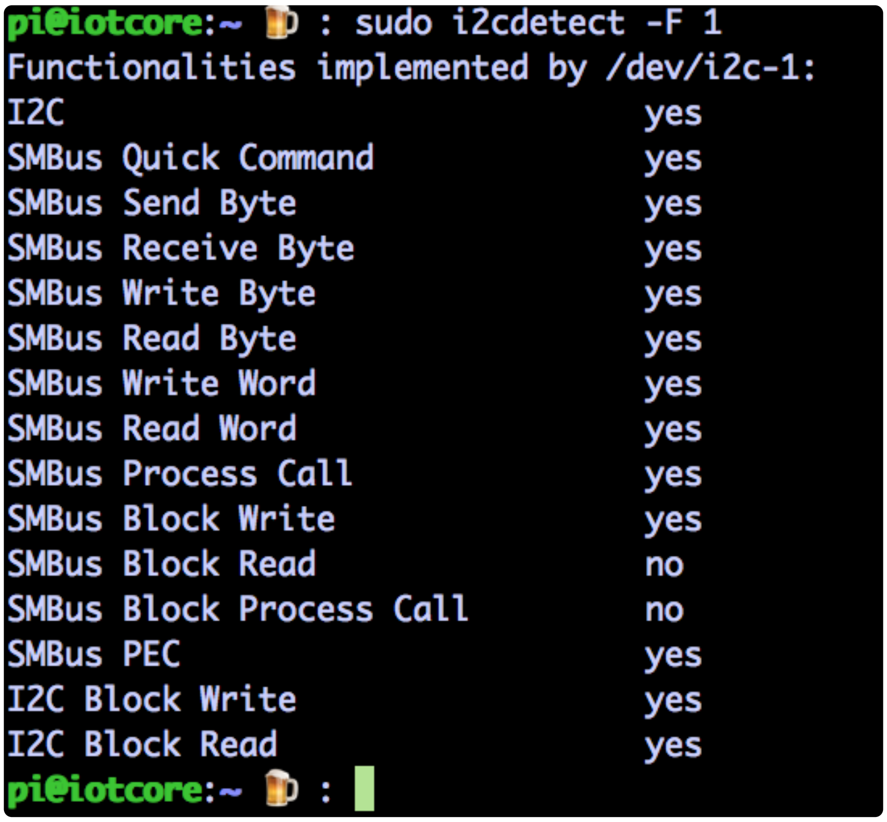
```
Raspberry Pi Software Configuration Tool (raspi-config)

P1 Camera      Enable/Disable connection to the Raspberry Pi Camera
P2 SSH         Enable/Disable remote command line access to your Pi using SSH
P3 VNC         Enable/Disable graphical remote access to your Pi using RealVNC
P4 SPI         Enable/Disable automatic loading of SPI kernel module
P5 I2C         Enable/Disable automatic loading of I2C kernel module
P6 Serial      Enable/Disable shell and kernel messages on the serial connection
P7 1-Wire      Enable/Disable one-wire interface
P8 Remote GPIO Enable/Disable remote access to GPIO pins

               <Select>                        <Back>
```

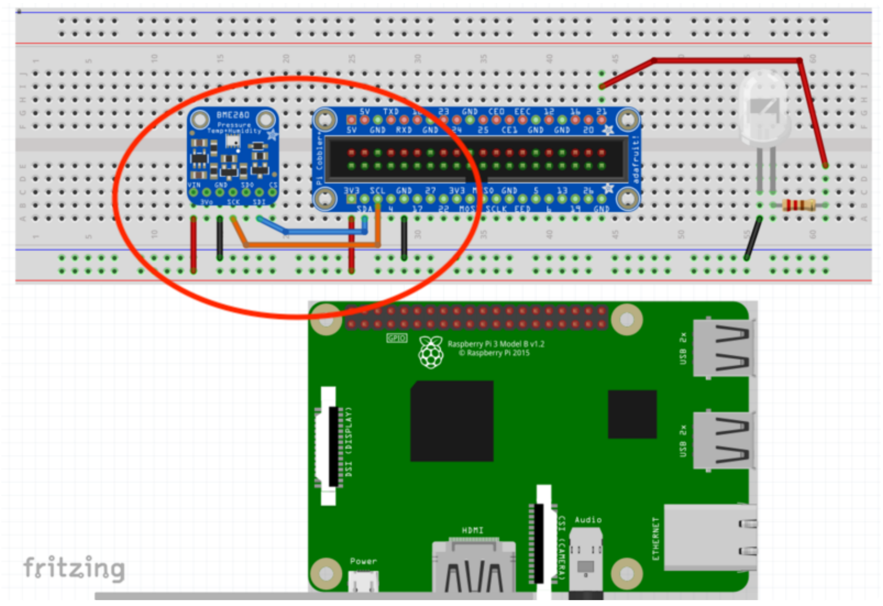
Exit out of raspi-config and run:

```
sudo i2cdetect -F 1
```



```
pi@iotcore:~ : sudo i2cdetect -F 1
Functionalities implemented by /dev/i2c-1:
I2C                              yes
SMBus Quick Command              yes
SMBus Send Byte                  yes
SMBus Receive Byte               yes
SMBus Write Byte                 yes
SMBus Read Byte                  yes
SMBus Write Word                 yes
SMBus Read Word                  yes
SMBus Process Call               yes
SMBus Block Write                yes
SMBus Block Read                 no
SMBus Block Process Call         no
SMBus PEC                        yes
I2C Block Write                  yes
I2C Block Read                   yes
pi@iotcore:~ :
```

Connect the RasPi Cobbler board to your breadboard and the 40 pin cable to your Pi 3 as pictured here. The keyed end in the cobbler is obvious, the white striped end of the cable and 90° angle of the cable coming off the RasPi (which is not keyed) are useful visual queues. Connect the Temp/Pressure/Humidity Sensor to the breadboard and connect the 3.3v and ground pins to the cobbler. Then connnect the i2c clock and data pins:

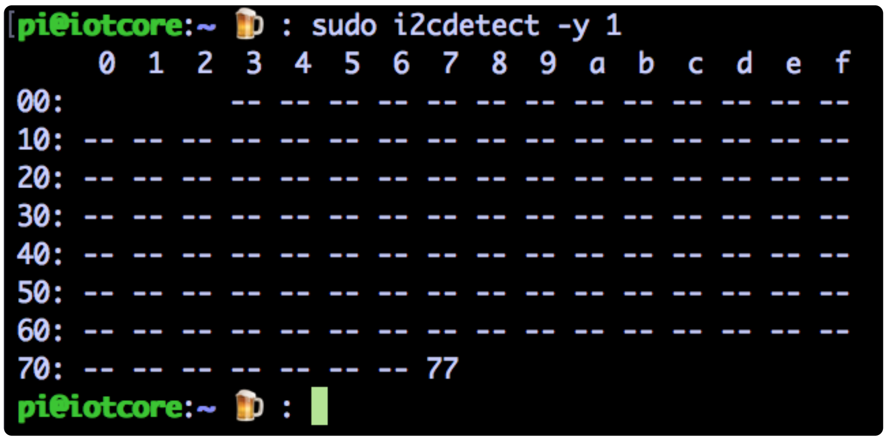


On the Pi Cobbler SDA is data pin and SCL is clock pin. On the BME280 sensor SDI is the data pin and SCK is the clock pin.

Verify i2c is enabled.

```
sudo i2cdetect -y 1
```

Will display a grid showing what address any devices are using on the i2C bus.



```
[pi@iotcore:~ 🍺 : sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- 77
pi@iotcore:~ 🍺 :
```

You can dump more information about any of the addresses shown with:

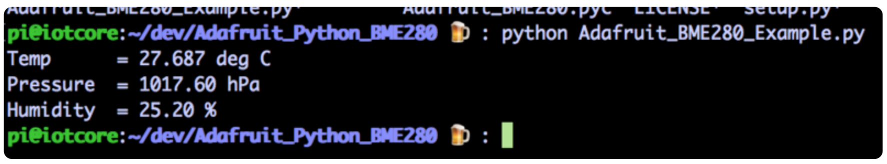
```
sudo i2cdump -y 1 0x77    <--- hex number shown from previous command
```

Install the AdafruitPythonGPIO and AdafruitPythonBME280 abstraction librabies

```
sudo apt-get install build-essential python-pip python-dev python-smbus git
cd ~ && mkdir dev
cd dev
git clone https://github.com/adafruit/Adafruit_Python_GPIO.git
cd Adafruit_Python_GPIO
sudo python setup.py install
cd ..
git clone https://github.com/adafruit/Adafruit_Python_BME280.git
cd Adafruit_Python_BME280
sudo python setup.py install
```

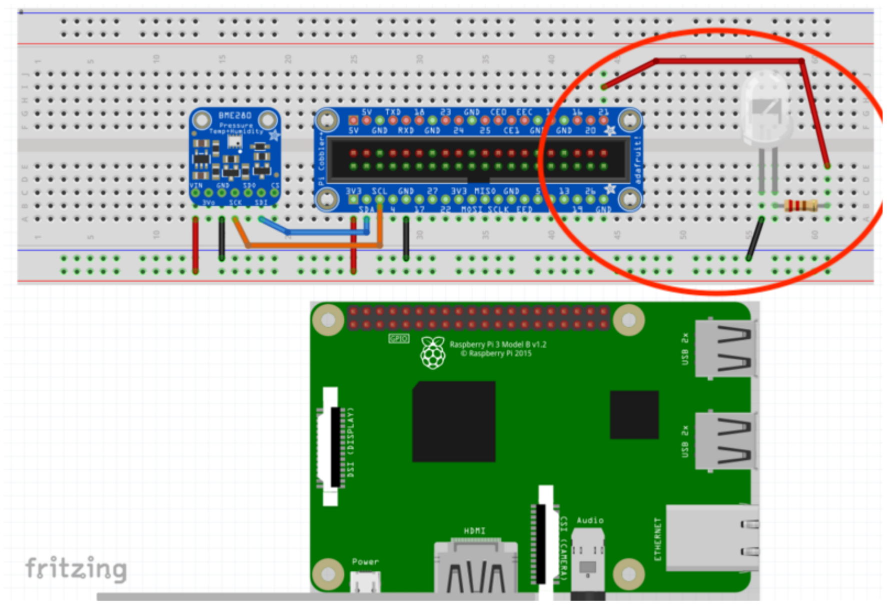
If you wish to sanity check your i2c wiring and sensor further:

```
python Adafruit_BME280_Example.py
```



```
pi@iotcore:~/dev/Adafruit_Python_BME280 🍺 : python Adafruit_BME280_Example.py
Temp      = 27.687 deg C
Pressure  = 1017.60 hPa
Humidity  = 25.20 %
pi@iotcore:~/dev/Adafruit_Python_BME280 🍺 :
```

Now connect an LED to GIPO 21 and one of the GND pins with a resistor in series on your breadboard. i.e Pin 21 on your Cobbler -> the long pin of the blue LED -> resistor -> GND rail or pin row:



The included 560 Ohm and 10K Ohm resistors will both protect the circuit, the latter make the LED dim. You can sanity check your wiring with python using the following commands one by one:

```
python
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(21, GPIO.OUT)
GPIO.output(21, GPIO.HIGH)
GPIO.output(21, GPIO.LOW)
quit()
```

Using "GPIO.output(21, GPIO.HIGH)" and "GPIO.output(21, GPIO.LOW)" should toggle your LED on an off. Or sanity check from bash using "gpio -g blink 21"

You'll also need the Python pub/sub library and APIs

```
sudo pip install --upgrade google-cloud-pubsub
sudo pip install google-api-python-client google-auth-httplib2 google-auth google-cloud
```

Create an API key and service account named api-tester and make a service_account.json file (steps 1 and 2 in the link) and put it in this example's directory (scp or rsync over ssh are easy ways to move files to your ssh connected Pi if you've downloaded the json file on a host machine).

Make sure you're authenticated. If you haven't already associated a gcloud project_id with this project, you'll be asked to do so. Use the project you created in the top level readme of this code base.

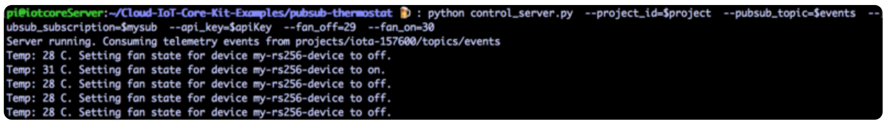
```
gcloud auth application-default login
```

Change to the directory you've cloned this example to. i.e. "cd ~/Cloud-IoT-Core-Kit-Examples/pubsub-thermostat"
Our control server can run on any host machine, including the RasPi. The "--fanoff" and "--fanon" arguments are the integer temperatures in °C that will turn on the "fan" LED i.e. when a devices is over 23°C and when it will turn the fan back off i.e. when a device is under 22°C. See optional argument options like "--serviceaccountjson=directory/location" in the code.

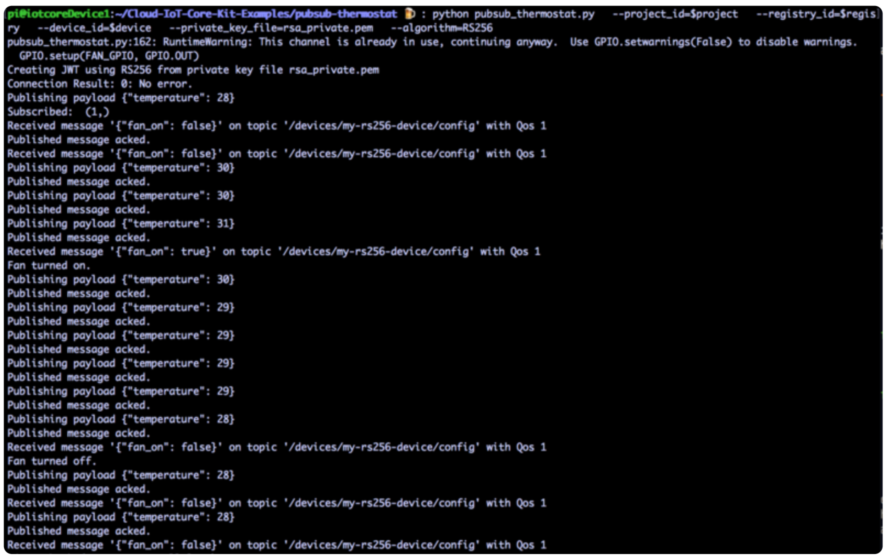
```
python control_server.py \
 --project_id=$project \
 --pubsub_topic=$events \
 --pubsub_subscription=$mysub \
 --api_key=$apiKey \
 --fan_off=22 \
 --fan_on=23
```

The client will run on one or many RasPi Cloud IoT kits with unique device ids:

```
python pubsub_thermostat.py \
  --project_id=$project \
  --registry_id=$registry \
  --device_id=$device \
  --private_key_file=rsa_private.pem \
  --algorithm=RS256
```



```
pi@iotcoreServer:~/Cloud-IoT-Core-Kit-Examples/pubsub-thermostat 🍺 : python control_server.py  --project_id=$project  --pubsub_topic=$events  --
ubsub_subscription=$mysub  --api_key=$apiKey  --fan_off=29  --fan_on=30
Server running. Consuming telemetry events from projects/iota-157600/topics/events
Temp: 28 C. Setting fan state for device my-rs256-device to off.
Temp: 31 C. Setting fan state for device my-rs256-device to on.
Temp: 28 C. Setting fan state for device my-rs256-device to off.
Temp: 28 C. Setting fan state for device my-rs256-device to off.
Temp: 28 C. Setting fan state for device my-rs256-device to off.
```



```
pi@iotcoreDevice1:~/Cloud-IoT-Core-Kit-Examples/pubsub-thermostat 🍺 : python pubsub_thermostat.py   --project_id=$project   --registry_id=$regis
ry   --device_id=$device   --private_key_file=rsa_private.pem   --algorithm=RS256
pubsub_thermostat.py:162: RuntimeWarning: This channel is already in use, continuing anyway.  Use GPIO.setwarnings(False) to disable warnings.
  GPIO.setup(FAN_GPIO, GPIO.OUT)
Creating JWT using RS256 from private key file rsa_private.pem
Connection Result: 0: No error.
Publishing payload {"temperature": 28}
Subscribed:  (1,)
Received message '{"fan_on": false}' on topic '/devices/my-rs256-device/config' with Qos 1
Published message acked.
Received message '{"fan_on": false}' on topic '/devices/my-rs256-device/config' with Qos 1
Publishing payload {"temperature": 30}
Published message acked.
Publishing payload {"temperature": 30}
Published message acked.
Publishing payload {"temperature": 31}
Published message acked.
Received message '{"fan_on": true}' on topic '/devices/my-rs256-device/config' with Qos 1
Fan turned on.
Publishing payload {"temperature": 30}
Published message acked.
Publishing payload {"temperature": 29}
Published message acked.
Publishing payload {"temperature": 29}
Published message acked.
Publishing payload {"temperature": 29}
Published message acked.
Publishing payload {"temperature": 29}
Published message acked.
Publishing payload {"temperature": 28}
Published message acked.
Received message '{"fan_on": false}' on topic '/devices/my-rs256-device/config' with Qos 1
Fan turned off.
Publishing payload {"temperature": 28}
Published message acked.
Received message '{"fan_on": false}' on topic '/devices/my-rs256-device/config' with Qos 1
Publishing payload {"temperature": 28}
Published message acked.
Received message '{"fan_on": false}' on topic '/devices/my-rs256-device/config' with Qos 1
```