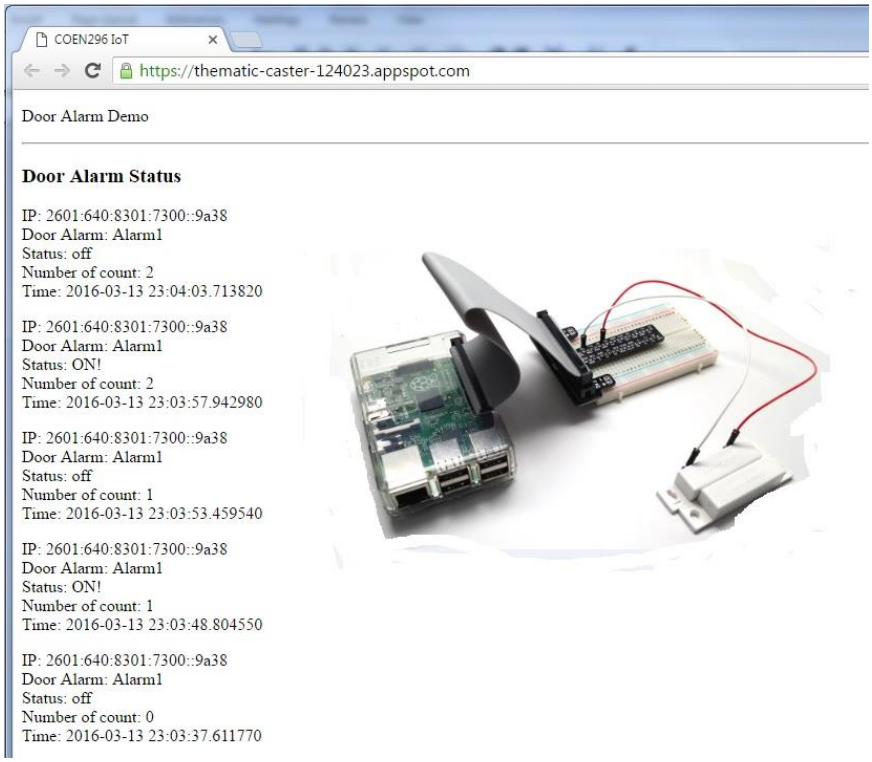
DOOR/WINDOW ALARM SYSTEM



3/19/2016

COEN296 - IOT

Santa Clara University

Final Survey Paper

Yi-Chiang Chang

Door/Window Alarm System

COEN296 - IOT

INTRODUCTION

This project was using Raspberry Pi and Google Cloud Platform to implement a Door/Window Alarm System that can record the date, time and total number of door opened after system power on. This system will also upload the information to google cloud platform that using can browse the history of these data.

Python was used for most of source code. But we also need to use one app.yaml for App Engine configuration and one HTML file for web framework.

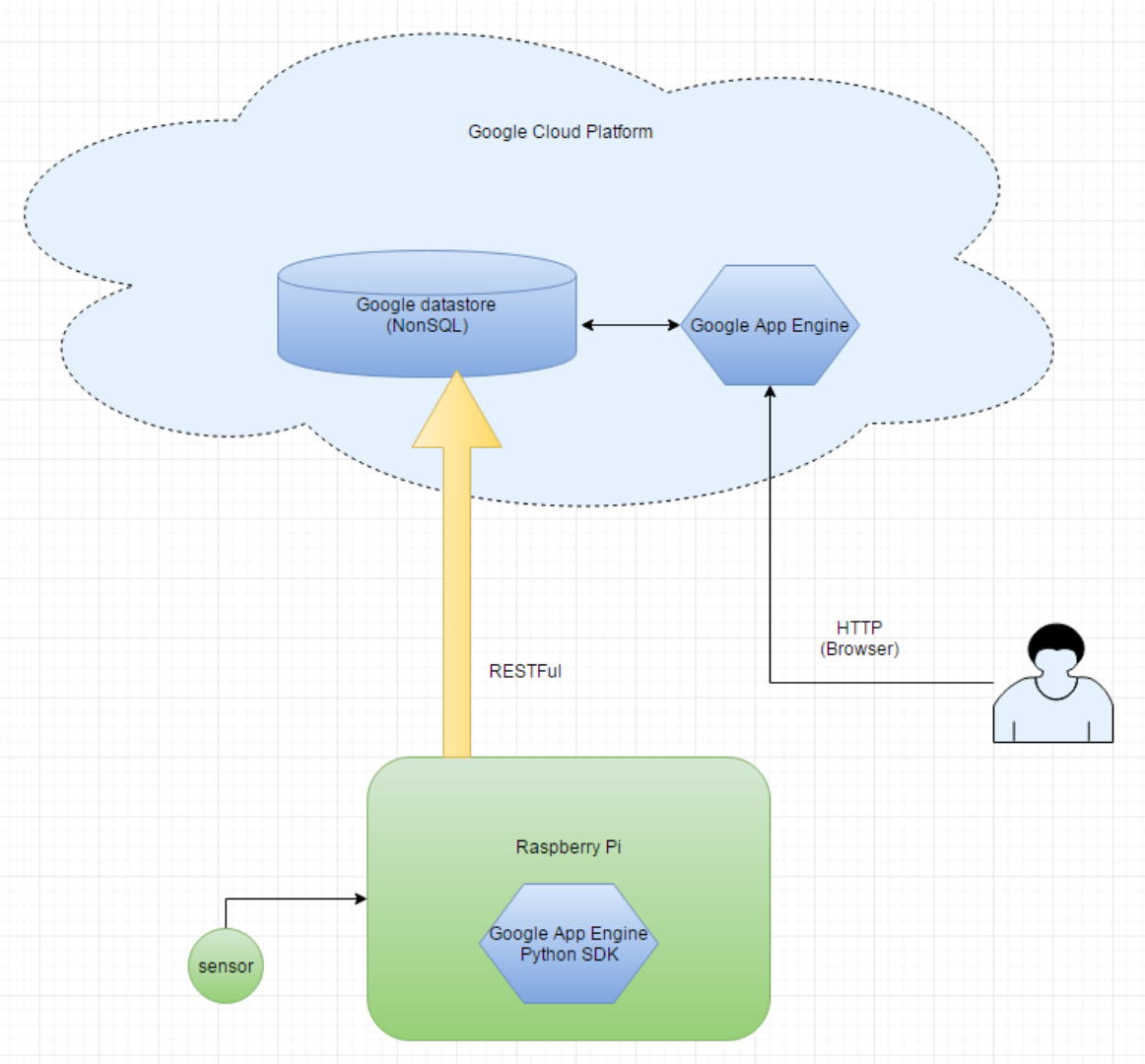
HARDWARE DEVICE REQUESTMENT

- ☐ Raspberry Pi 1 Model A+, B+ or Raspberry Pi 2
- ☐ 8+ GB MicroSD card
- ☐ Ethernet Cable or USB WiFi adapter
- ☐ MicroUSB cable or power supply to provide power for Raspberry Pi
- ☐ Normally Closed(N.C.) window/door contact magnetic switch
- ☐ HDMI cable and Monitor
- ☐ USB keyboard
- ☐ USB Mouse

Following items are optional

- ☐ 40 pins GPIO Ribbon cable with Breakout Board
- ☐ Breadboard
- ☐ 220-400Ω resistor
- ☐ LED

DOOR ALARM SYSTEM ARCHTECTURE



PROJECT IMPLEMENTATION

To implement this system, you need to follow few steps to setup this project.

1. Raspberry PI Real Time Operating System and working environment setup
2. Sensor and raspberry Pi connection
3. Google Cloud Platform setup
4. Source code implementation
5. Testing

All the source codes can be download from https://github.com/yi-chg2/COEN296_IoT_DoorAlarmGAE

STEP1: RASPBERRY PI RTOS SETUP

Install Rasbian

There are many online tutorials for RTOS setup for Raspberry PI. Here we are using the tutorial from Raspberry Pi official website to setup the NOOBS for installing Rasbian(O.S.)

https://www.raspberrypi.org/documentation/installation/

Here is the video showing the steps

https://www.raspberrypi.org/help/noobs-setup/

There is also other tutorial on https://learn.adafruit.com/category/learn-raspberry-pi

Setup Rasbian

Once you install the Rasbian and boot it first, please follow the instruction and finished some basic setup. You can also change the setup in the future if you want to change any configurations.

1. Open LXTerminal
2. $ sudo raspi-config
3. Update Expand_Filesystem to using whole SD card free space
4. Update Internationalization Options
5. Select Advanced Options
6. Enable SSH, SPI, I2C, Serial (optional)

I highly recommend you to change the Internationalization Options to setup the language, time zone and keyboard. The default setting are based on U.K. in Rasbian and there are few keyboard special characters are different to the U.S. keyboard layout.

Connect to Network

Please follow the link below to setup the network connection.

https://learn.adafruit.com/adafruits-raspberry-pi-lesson-3-network-setup

Update package and other software

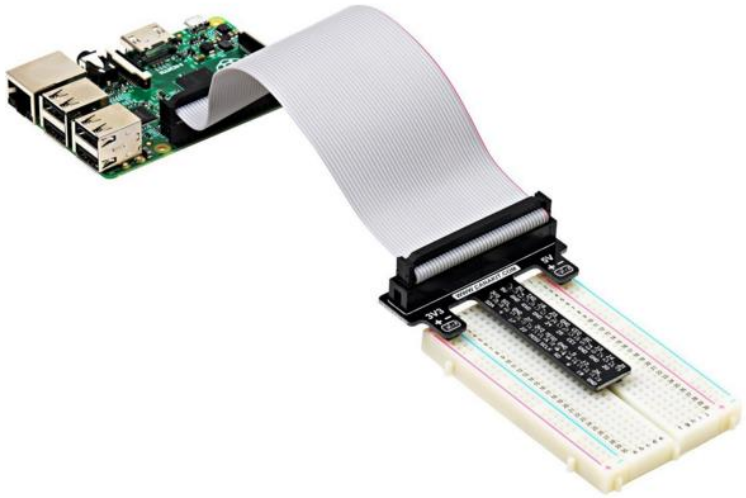
1. Open LXTermial
2. $ sudo apt-get update
3. $ sudo apt-get install git
4. $ sudo apt-get install python-dev
5. $ sudo apt-get install python-rpi.gpio

There are few other settings for I2C, SPI, SSH, GUI remote control etc. You can find the tutorials at https://learn.adafruit.com/series/learn-raspberry-pi. We will skip these steps because this project is not necessarily need to use them.

STEP2: SENSOR AND RASPBERRY PI CONNECTION

Connect Raspberry Pi with Breadboard (optional)

To make the work easier, we are using the 40 pins GPIO Ribbon cable with Breakout Board and stack the breakout board on top of breadboard.



This step can skip if you are comfortable to connect the sensor, resistor and LED directly to Raspberry PI and here is the pin mapping on the Paspberry Pi website http://pinout.xyz/

Connect Door Sensor, LED and Resistor on breadboard

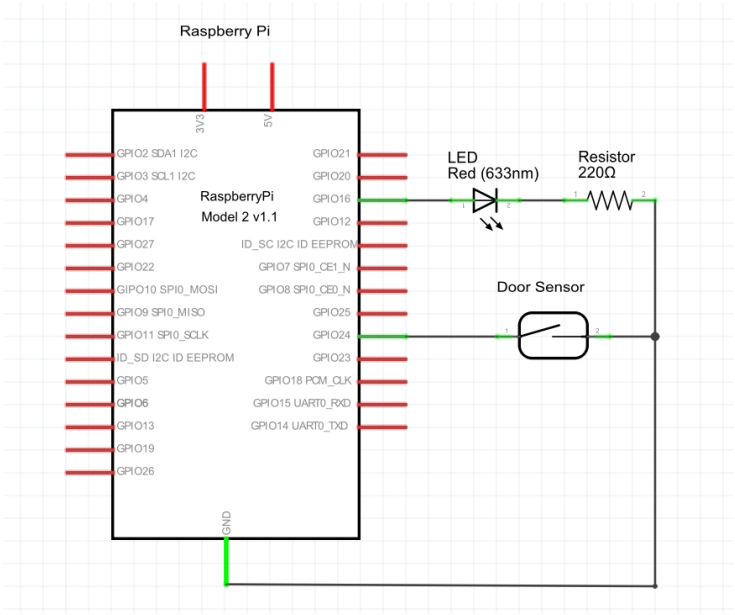
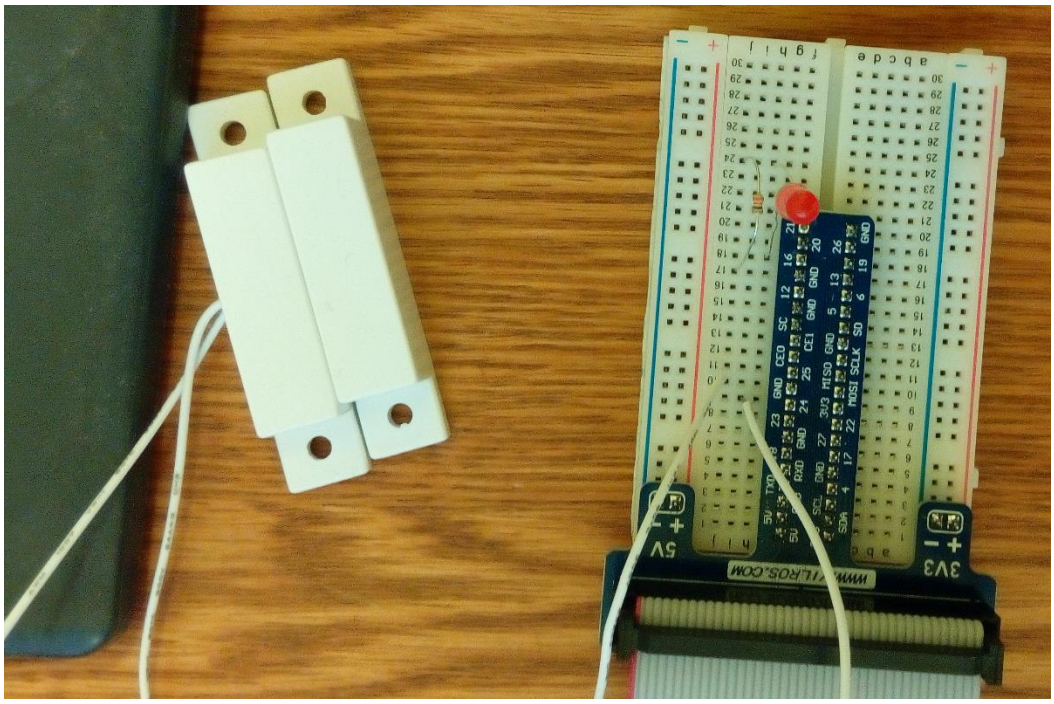


FIGURE 1SCEMATIC OF THE DEVICE

LED and Resistor: connects the long pin of LED to GPIO pin16 and short pin connect to one end of the resistor. Also the other end of resistor connects to ground (GND).

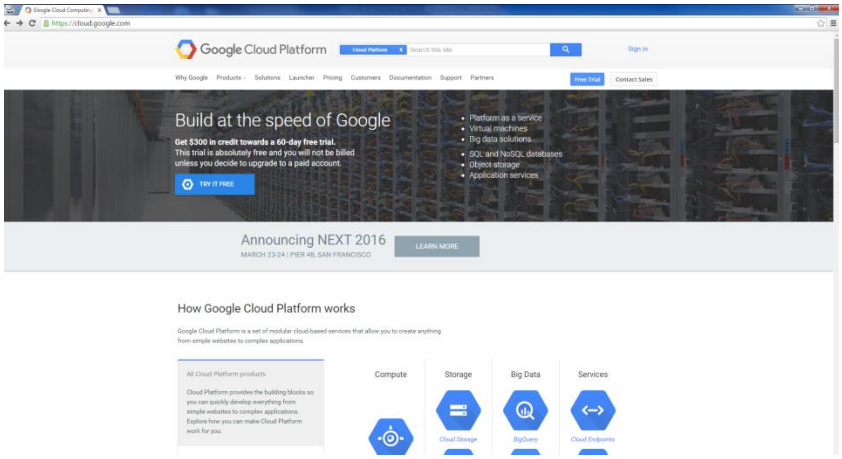
Door sensor: One end connects to GPIO pin 24 and the other end connect to GND



STEP3: GOOGLE CLOUD PLATFORM SETUP

Setup google account and activate free trial

Before using google cloud platform, we need to setup a google account and activate the Free Trial for this platform (https://cloud.google.com/). Google currently provides 60-day free trial with $300 credit for user to try it out! Click the free trial button and follow the instruction. Google will ask for billing information but don't worry about it if you didn't upgrade your account to enable the billing.



If you reach the limit for your free trial account, the google will just stop the service and network traffic for you to send/receive more data.

Google App Engine

In this project, we need to use a very useful tool App Engine (https://cloud.google.com/appengine/docs) to create a Web based application and also o maintain and scale the traffic and data storage.

Google provides a very simple HelloWorld example for this tool using python, JAVA, PHP and GO for standard environment development. In order to easier integrating with the code for catching sensor data later on Raspberry Pi, we choose the Python for this project to implement.

Once you activate google app engine, google will assign a project ID for you. You can find this project-ID as showing in the following figure and use this ID or create a new Project ID for future implementation.

Here is the link to the example of the "HELLO, WORLD!" using Python(https://cloud.google.com/appengine/docs/python/).

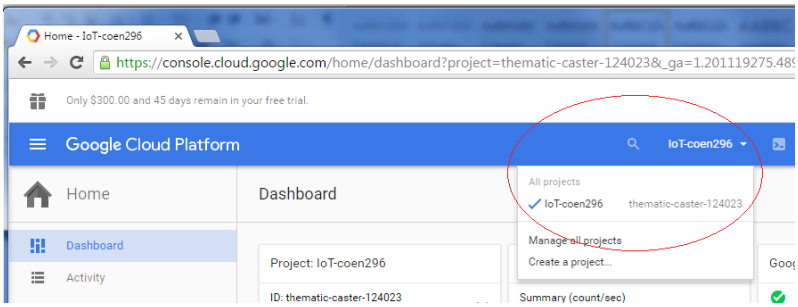


FIGURE 2 PROJECT ID

- Download Google App Engine SDK for Python

1. Open web browser in Raspberry Pi
2. Go to this link and download the SDK for Linux
   https://cloud.google.com/appengine/downloads#Google_App_Engine_SDK_for_Python
3. $ unzip google_appengine_<version>.zip -d [destination folder]
4. $ export PATH=/path/to/google_app_engine:$PATH

Adding the path can make it easier without providing the path to the python app when we need to use some python program to update and launch the application.

STEP4: SOURCE CODES IMPLEMENTATION

- app.yaml

```
application: thematic-caster-124023
version: 1
runtime: python27
api_version: 1
threadsafe: true

handlers:
- url: /.*
  script: door_alarm_gae.app

libraries:
- name: webapp2
  version: latest
- name: jinja2
  version: latest
```

This code runs in the python27 runtime environment, API version 1. Additional runtime environments and languages may be supported in the future.

This application is threadsafe so the same instance can handle several simultaneous requests. Threadsafe is an advanced feature and may result in erratic behavior if your application is not specifically designed to be threadsafe.

Every request to a URL whose path matches the regular expression /.* (all URLs) should be handled by the app object in the helloworld module.

Declare that you are using webapp2 by adding this libraries section to your app.yaml

- door_alarm_gae.py

```
import webapp2, jinja2, os, json, urllib, urllib2
from models import Log

JINJA_ENVIRONMENT = jinja2.Environment(
                loader = jinja2.FileSystemLoader(os.path.dirname(__file__)),
                extensions = ['jinja2.ext.autoescape'],
                autoescape = True)

class DataHandler(webapp2.RequestHandler):
  def post(self):
      # init models object
      custom_index_log = Log()
      custom_index_log.ip_address = self.request.remote_addr
      custom_index_log.alarm_id = self.request.get("SENSOR_ID")
      custom_index_log.alarm_status = self.request.get("SENSOR_STATUS")
      custom_index_log.alarm_count = self.request.get("SENSOR_COUNT")
      custom_index_log.put()
      self.response.headers['Content-Type'] = 'text/plain'
      self.response.write('return successful')
      # store data door alarm

# request handler
class MainPage(webapp2.RequestHandler):
   def get(self):
      # set template
      template = JINJA_ENVIRONMENT.get_template('index.html')
      # get ip addresses of visitors
      ip_logs = Log.query().order(-Log.access_time)
      # values to pass to front-end
      template_values = { 'title' : 'COEN296 IoT',
                          'greeting' : 'Door Alarm Demo',
                          'ip_logs' : ip_logs}
      # dispatch template and values to front-end
      self.response.write(template.render(template_values))
# WSGI setting
app = webapp2.WSGIApplication([
   ('/', MainPage),
   ('/upload_data', DataHandler),
], debug=True)
```

This code defines the application structure and two handlers that publishes application (class MainPage) and dynamically updates local sensor data (class DataHandler) to the App Engine

DataHandler class also catch dynamical data from "models.py" and it also return success message when the data upload properly.

MainPage class not only define the main structure of the web page but also by loading the "index.html" that defines the GUI of the webpage.

WSGIApplication instance that routes incoming requests to handlers based on the URL. The application itself is represented by a webapp2.WSGIApplication instance. The parameter debug=true passed to its constructor

tells webapp2 to print stack traces to the browser output if a handler encounters an error or raises an uncaught exception.

- models.py

```
from google.appengine.ext import ndb

class Log(ndb.Model):
    access_time = ndb.DateTimeProperty(auto_now_add=True)
    ip_address = ndb.StringProperty()
    alarm_id = ndb.StringProperty()
    alarm_status = ndb.StringProperty()
    alarm_count = ndb.StringProperty()
```

This file defines the dynamic data that can be update by event or status. The ndb property type can refer to https://cloud.google.com/appengine/docs/python/ndb/properties.

- index.html

```
<!DOCTYPE html>
{% autoescape true %}
<head>
<meta http-equiv="Content-Type" content="text/html" charset="utf-8">
<meta name="author" content="Yi-Chiang Chang">
<meta name="Demo" content="Door Alarm Demo">

<title>{{ title | safe }}</title>
</head>
<html>
   <body>
      <p>{{ greeting | safe }}</p>

      <hr/>
      <!-- ip logs -->
      <h3>Door Alarm Status</h3>
      {% if ip_logs %}
         {% for ip_log in ip_logs %}
            <p>
            IP: <span>{{ip_log.ip_address | safe}}</span>
            <br/>
                  Door Alarm: <span>{{ip_log.alarm_id | safe}}</span>
                  <br/>
                  Status: <span>{{ip_log.alarm_status | safe}}</span>
            <br/>
            Number of count: <span>{{ip_log.alarm_count | safe}}</span>
                  <br/>
                  Time: <span>{{ip_log.access_time | safe}}</span>
            </p>
         {% endfor %}
      {% else %}
         <p>No Logs</p>
      {% endif %}
   </body>
</html>
{% endautoescape %}
```

{{ip_log.<Variable> | safe}} is dynamical data that map to “models.py” and | safe ctelling the web framework this variable should keep the original symbol.

- door_alarm.py

```
import time, urllib, urllib2
import RPi.GPIO as io
import os

io.setmode(io.BCM)

door_pin1 = 24
led_1 = 16

io.setup(door_pin1, io.IN, pull_up_down=io.PUD_UP)
io.setup(led_1, io.OUT)
var1 = "ON!"
count = 0

def blink_led():
    io.output(led_1, True)
    time.sleep(0.3)
    io.output(led_1, False)
    time.sleep(0.3)

while True:
    if io.input(door_pin1):
        print("LOCAL PRINT : DOOR ALARM ON!!!")
        var1 = "ON!"
        count = count +1
        query_args = {"SENSOR_ID":"Alarm1","SENSOR_STATUS":var1,"SENSOR_COUNT":count}
        encoded_args = urllib.urlencode(query_args)
        url = 'https://thematic-caster-124023.appspot.com/upload_data'
        print urllib2.urlopen(url, encoded_args).read()
        while io.input(door_pin1):
            blink_led()
    else:
        io.output(led_1, False)
        if var1 == "ON!":
            print("LOCAL PRINT : DOOR ALARM OFF")
            var1 = "off"
            query_args = {"SENSOR_ID":"Alarm1","SENSOR_STATUS":var1,"SENSOR_COUNT":count}
            encoded_args = urllib.urlencode(query_args)
            url = 'https://thematic-caster-124023.appspot.com/upload_data'
            print urllib2.urlopen(url, encoded_args).read()


    time.sleep(0.5)
```

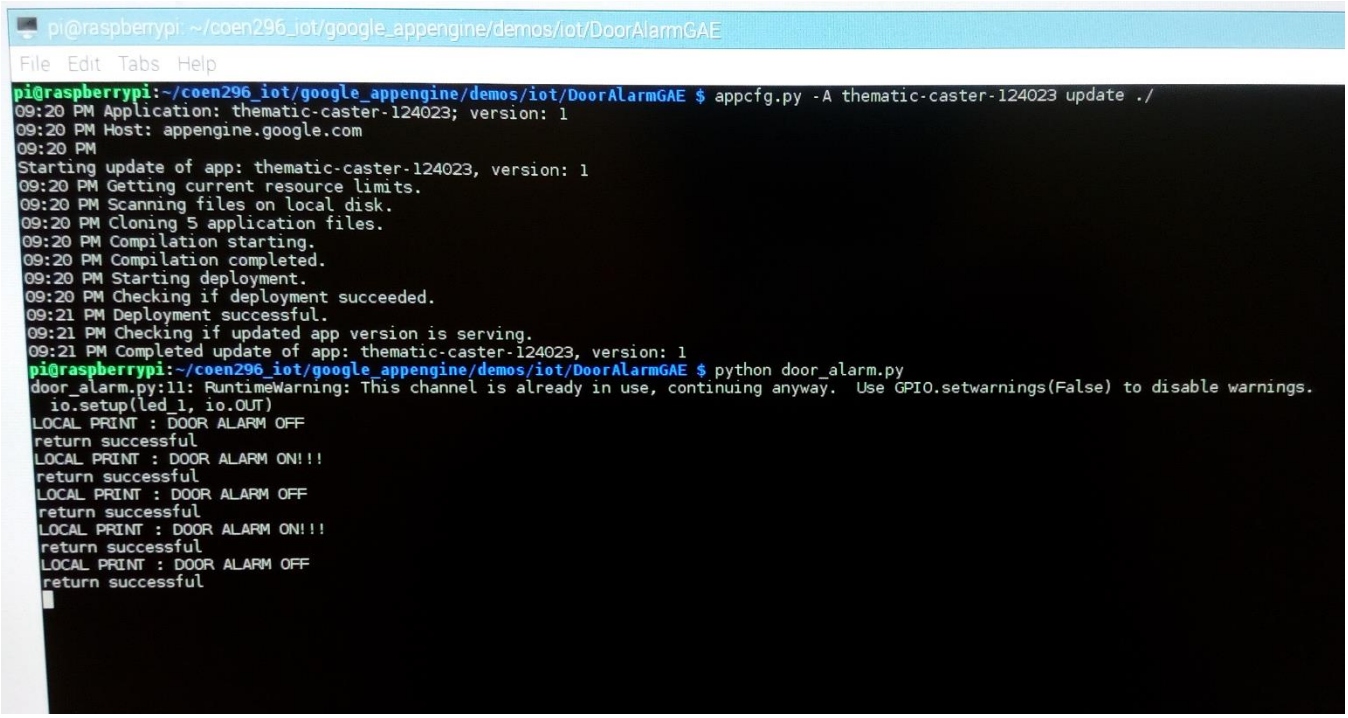
Door sensor is normal closed model and the GPIO24 is set to pull up logic. When sensor is close, the GND signal is stronger so the GPIO24 is under low voltage. But when sensor open, the GND signal cannot pass to GPIO24 and this will change the PGIO24 to pull up and become voltage high.

The count is the counter for how many times the door opened after system power on and query_args pass the dynamical data that map to models.py.

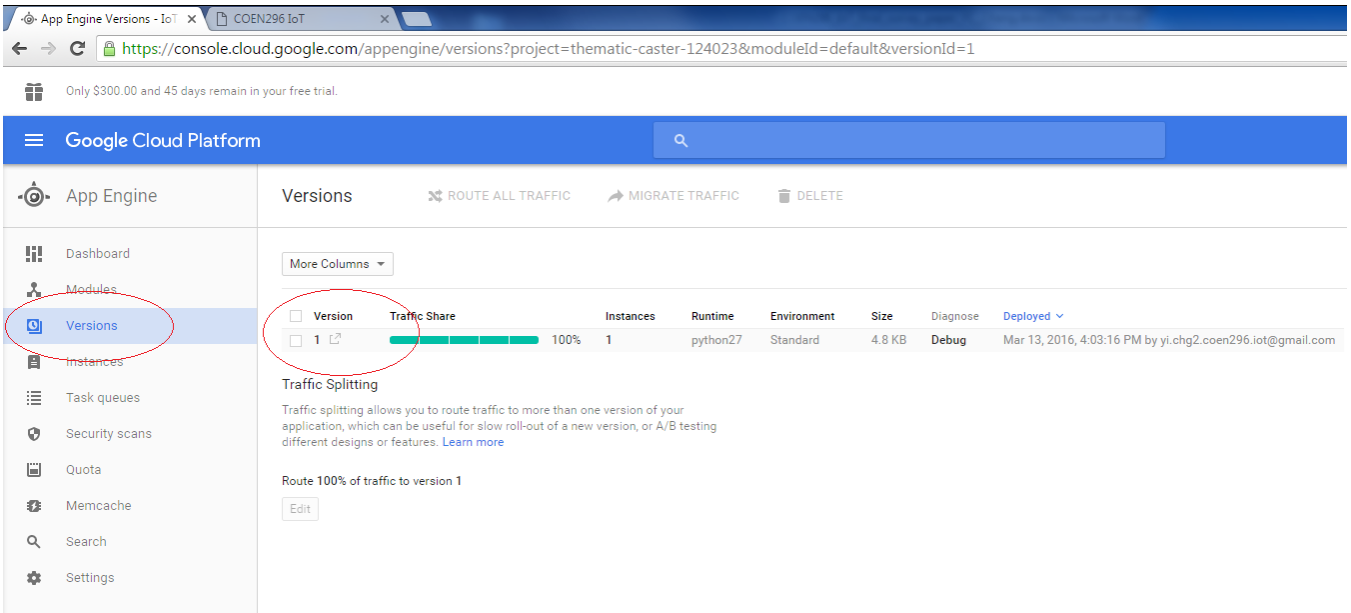
STEP5: TESTING

- Upload your application

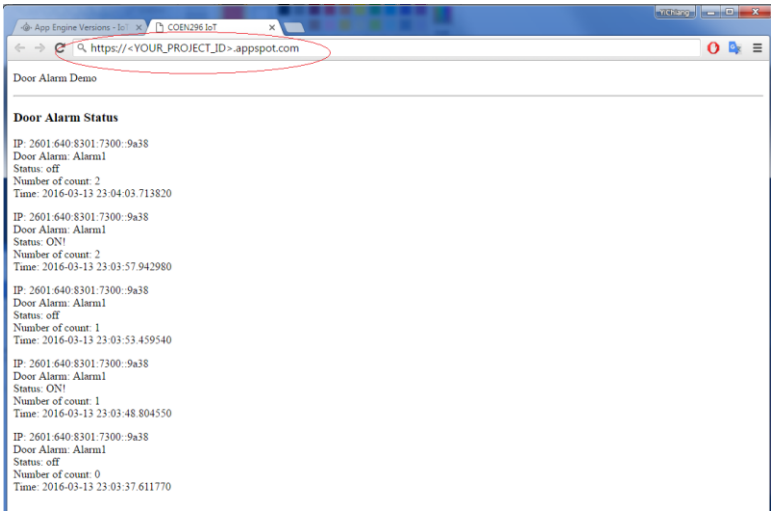
1. Open LXTerminal
2. $ python appcfg.py –A <YOUR_PROJECT_ID> update ./
3. Check your app engine console and see if there is any error and try to debug it
4. $ python door_alarm.py



5. Check your app engine and open the versions under App Engine



6. After click your application version, your web application will open on the other windows



7. Congratulations! You can share your web address to other user to check the Alarm now.