Add-on work

To Visualize data obtained from sense hat into Open source dashboards (IoT perspective)

The task is to build our very own hyper-local weather dashboard, capturing the weather inside and outside our house over time.

So In order to send data over Internet and visualize it we can Use Wunderground Dashboard.

1. Wunderground Dashboard:

The Weather Underground (http://www.wunderground.com) provides real-time online weather information assisted by a network 140,000+ personal weather stations setup all over the world. Chances are, there is a Weather Underground personal weather station near you. The really cool aspect of Wunderground is that they have an API that we can use to retrieve the weather data (pretty much) anywhere in the world.

Temperature, dew point, wind gust, humidity, precipitation, pressure, UV index, ... all easily available for wherever we want, whenever we want. Let's learn how to use Wunderground's super-simple API.

Part 1: How to Use the Wunderground API

In order to use the Wunderground API, you first need your own API key. Getting an API key is quick and free.

- **Go to http://www.wunderground.com/weather/api/?MR=1.**
- **Click "Sign Up for FREE!".**

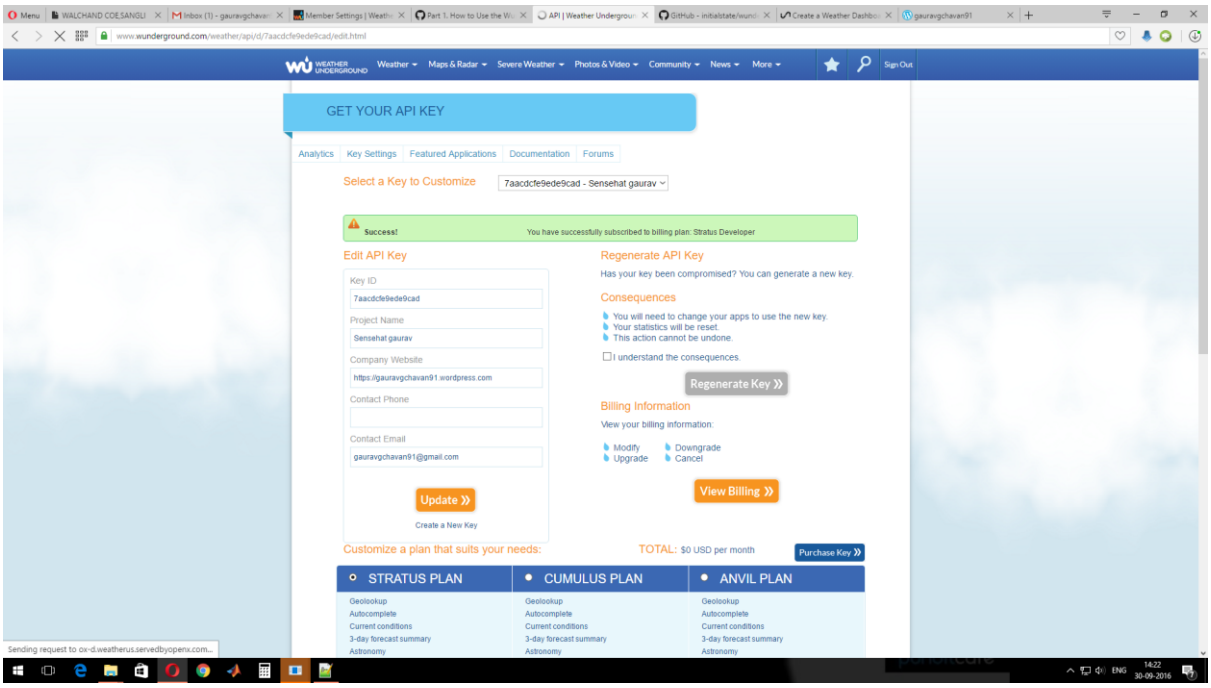

Fig.1 Wunderground account

- **Create an account and click the link sent to you in a validation email to activate your account**
- **Sign in**
- **Go to Pricing and select the free Stratus Plan (default selection). You get 500 API calls per day for $0. There is no credit card required to get the Developer level API.**

- **Click "Purchase Key"**
- **Fill out the form and submit it to get your API key**

Your key will look something like this: 0def10027afaebb7. Save it.

You can make an API to Wunderground by typing in a URL into your browser in the following format:

```
http://api.wunderground.com/api/YOUR_API_KEY/conditions/q/THE_DESIRED_STATE/THE_DESIRED_CITY.json
```

for example, to get the weather in San Francisco, CA:

- http://api.wunderground.com/api/0def10027afaebb7/conditions/q/CA/San_Francisco.json

or, to get the weather in sangli

http://api.wunderground.com/api/0def10027afaebb7/conditions/q/IN/Sangli.json

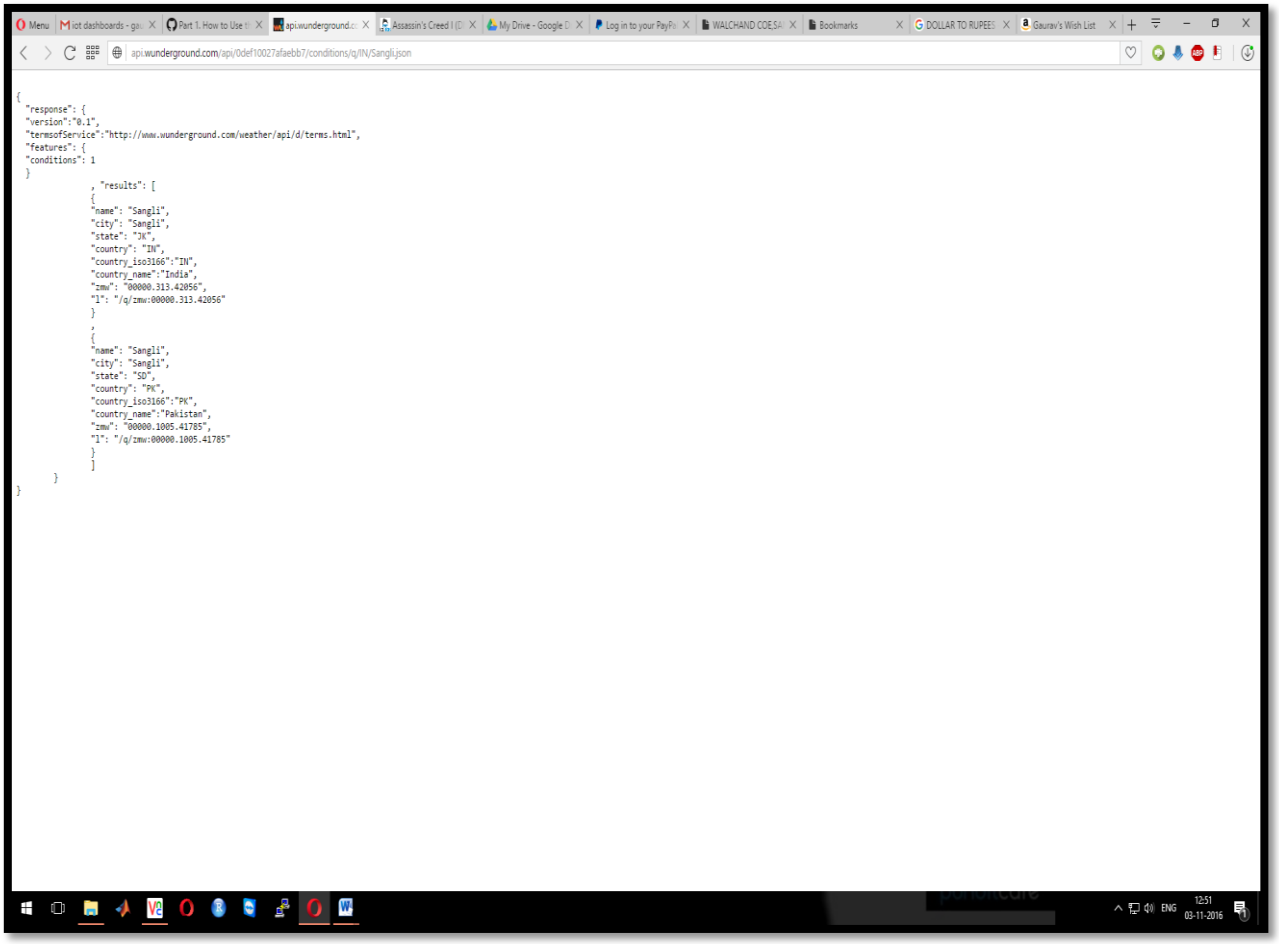

Fig.2 Sangli.json

We just need to make a script to parse it, then ship it to a web-based dashboard. First, let's setup the destination for our data -> Initial State.

Part two: Initial State

We want to stream all of our weather data to a cloud service and have that service turn our data into a nice dashboard that we can access from our laptop or mobile device. Our data needs a destination. We will use Initial State as that destination.

Step 1: Register for Initial State Account

Go to https://app.initialstate.com/#/register/ and create a new account.

Step 2: Install the ISStreamer

Install the Initial State Python module onto your Raspberry Pi:

At a command prompt (don't forget to SSH into your Pi first), run the following command:

```
$ cd /home/pi/
$ \curl -sSL https://get.initialstate.com/python -o - | sudo bash
```

After Step 2 you will see something similar to the following output to the screen:

```
pi@raspberrypi ~ $ \curl -sSL https://get.initialstate.com/python -o - | sudo bash
Password:
Beginning ISStreamer Python Easy Installation!
This may take a couple minutes to install, grab some coffee :)
But don't forget to come back, I'll have questions later!

Found easy_install: setuptools 1.1.6
Found pip: pip 1.5.6 from /Library/Python/2.7/site-packages/pip-1.5.6- py2.7.egg (python 2.7)
pip major version: 1
pip minor version: 5
ISStreamer found, updating...
Requirement already up-to-date: ISStreamer in /Library/Python/2.7/site-packages
Cleaning up...
Do you want automagically get an example script? [y/N]
```

When prompted to automatically get an example script, type y. This will create a test script that we can run to ensure that we can stream data to Initial State from our Pi. You will be prompted:

```
Where do you want to save the example? [default: ./is_example.py]:
```

You will be prompted for your username and password that you just created when you registered your Initial State account. Enter both and the installation will complete.

Go back to your Initial State account in your web browser. A new data bucket called "Python Stream Example" should have shown up on the left in your log shelf (you may have to refresh the page). Click on this bucket and then click on the Waves icon to view the test data.

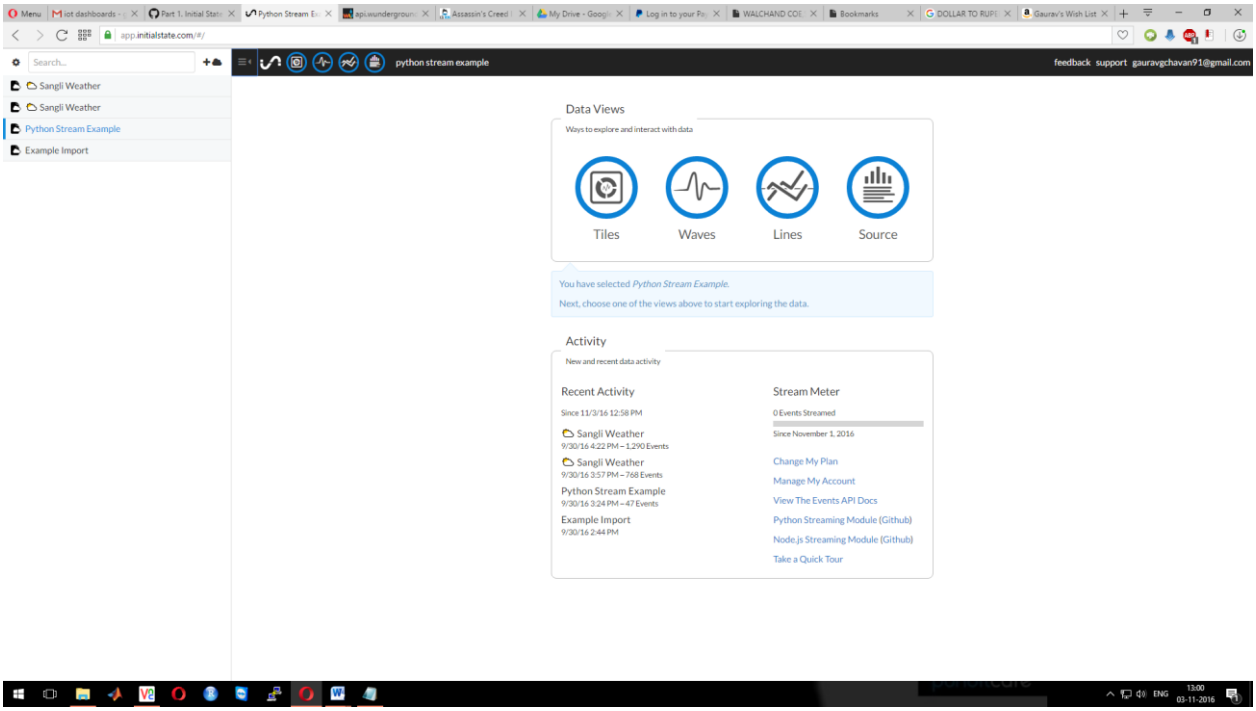

Fig.3. Wunderground Profile

We are ready to start using the Wunderground API to create a weather dashboard and capture the weather history for wherever we choose. To do this, we are going to use the Python script: https://github.com/InitialState/wunderground-sensehat/blob/master/wunderground.py. This script simply calls the Wunderground API using your API key and retrieves the weather information on a specified time interval. It also streams that data to your Initial State account, which will allow you to create a Wunderground dashboard.

You can either copy this script to your Pi, or access it through the Github repository that we cloned earlier. You can do this by changing into your wunderground-sensehat directory by typing:

```
$ cd wunderground-sensehat
```

From here, you'll be able to access the python file that we'll run to create our weather dashboard.

Before you run it, you need to set your desired parameters and insert your keys. Nano into the wunderground.py file by typing:

```
$ nano wunderground.py
```

Code: wunderground.py

```
import urllib2
import json
import os
import glob
import time
from ISStreamer.Streamer import Streamer

# --------- User Settings ---------
STATE = "IN"
CITY = "Sangli"
WUNDERGROUND_API_KEY = "PLACE YOUR WUNDERGROUND API KEY HERE"
BUCKET_NAME = ":partly_sunny: " + CITY + " Weather"
BUCKET_KEY = "wu1"
ACCESS_KEY = "PLACE YOUR INITIAL STATE ACCESS KEY HERE"
MINUTES_BETWEEN_READS = 15
METRIC_UNITS = False
# ---------------------------------
def isFloat(string):
    try:
       float(string)
        return True
    except ValueError:
        return False

def get_conditions():
api_conditions_url = "http://api.wunderground.com/api/" + WUNDERGROUND_API_KEY + "/conditions/q/" + STATE + "/" + CITY + ".json"
try:
f = urllib2.urlopen(api_conditions_url)
```

```
except:
return []
json_conditions = f.read()
f.close()
return json.loads(json_conditions)

def get_astronomy():
api_astronomy_url = "http://api.wunderground.com/api/" + WUNDERGROUND_API_KEY + "/astronomy/q/" + STATE + "/" + CITY + ".json"
try:
f = urllib2.urlopen(api_astronomy_url)
except:
return []
json_astronomy = f.read()
f.close()
return json.loads(json_astronomy)

def is_night(astronomy):
sunrise_hour = int(astronomy['moon_phase']['sunrise']['hour'])
sunrise_min  = int(astronomy['moon_phase']['sunrise']['minute'])
sunset_hour  = int(astronomy['moon_phase']['sunset']['hour'])
sunset_min   = int(astronomy['moon_phase']['sunset']['minute'])
current_hour = int(astronomy['moon_phase']['current_time']['hour'])
current_min  = int(astronomy['moon_phase']['current_time']['minute'])
if ( (current_hour < sunrise_hour) or
(current_hour > sunset_hour) or
 ((current_hour == sunrise_hour) and
(current_min < sunrise_min)) or
((current_hour == sunset_hour) and
(current_min > sunset_min)) ):
return True
return False
```

```
        def moon_icon(moon_phase):
        icon = {
        "New Moon"        : ":new_moon:",
        "Waxing Crescent" : ":waxing_crescent_moon:",
        "First Quarter"   : ":first_quarter_moon:",
        "Waxing Gibbous"  : ":waxing_gibbous_moon:",
        "Full Moon"       : ":full_moon:",
        "Full"            : ":full_moon:",
        "Waning Gibbous"  : ":waning_gibbous_moon:",
        "Last Quarter"    : ":last_quarter_moon:",
        "Waning Crescent" : ":waning_crescent_moon:",
}
return icon.get(moon_phase,":crescent_moon:")

def weather_icon(weather_conditions):
icon = {
"clear"           : ":sun_with_face:",
"cloudy"          : ":cloud:",
“flurries"        : ":snowflake:",
"fog"             : ":foggy:",
"hazy"            : ":foggy:",
"mostlycloudy"    : ":cloud:",
"mostlysunny"     : ":sun_with_face:",
"partlycloudy"    : ":partly_sunny:",
"partlysunny"     : ":partly_sunny:",
"sleet"           : ":sweat_drops: :snowflake:",
"rain"            : ":umbrella:",
"snow"            : ":snowflake:",
"sunny"           : ":sun_with_face:",
```

```
"tstorms"          : ":zap: :umbrella:",
unknown"           : ":sun_with_face:",
}
                    return icon.get(weather_conditions,":sun_with_face:")

        def weather_status_icon (conditions, astronomy):
                    moon_phase = astronomy['moon_phase']['phaseofMoon']
                    weather_conditions = conditions['current_observation']['icon']
                    icon = weather_icon(weather_conditions)
                    if is_night(astronomy):
                                if ((icon == ":sunny:") or
                                    (icon == ":partly_sunny:") or
                                    (icon == ":sun_with_face:")):
                                            return moon_icon(moon_phase)
                    return icon

        def wind_dir_icon (conditions, astronomy):
                    icon = {
                                "East"     : ":arrow_right:",
                                "ENE"      : ":arrow_upper_right:",
                                "ESE"      : ":arrow_lower_right:",
                                "NE"       : ":arrow_upper_right:",
                                "NNE"      : ":arrow_upper_right:",
                                "NNW"      : ":arrow_upper_left:",
                                "North"    : ":arrow_up:",
                                "NW"       : ":arrow_upper_left:",
                                "SE"       : ":arrow_lower_right:",
                                "South"    : ":arrow_down:",
                                "SSE"      : ":arrow_lower_right:",
                                "SSW"      : ":arrow_lower_left:",
                                "SW"       : ":arrow_lower_left:",
```

```
                "Variable" : ":arrows_counterclockwise:",
                "West"     : ":arrow_left:",
                "WNW"      : ":arrow_upper_left:",
                "WSW"      : ":arrow_lower_left:",
        }
        return icon.get(conditions['current_observation']['wind_dir'],":crescent_moon:")

    conditions = get_conditions()
    astronomy = get_astronomy()
    if ('current_observation' not in conditions) or ('moon_phase' not in astronomy):
print "Error! Wunderground API call failed, check your STATE and CITY and make sure your Wunderground API key is valid!"
if 'error' in conditions['response']:
print "Error Type: " + conditions['response']['error']['type']
print "Error Description: " + conditions['response']['error']['description']
exit()
else:
streamer = Streamer(bucket_name=BUCKET_NAME, bucket_key=BUCKET_KEY, access_key=ACCESS_KEY)
streamer.log(":house: Location",conditions['current_observation']['display_location']['full'])
while True:
conditions = get_conditions()
astronomy = get_astronomy()
if ('current_observation' not in conditions) or ('moon_phase' not in astronomy):
print "Error! Wunderground API call failed. Skipping a reading then continuing ..."
else:
humidity_pct = conditions['current_observation']['relative_humidity']
humidity = humidity_pct.replace("%","")

# Stream valid conditions to Initial State
streamer.log(":clock3: Updated Time",astronomy['moon_phase']['current_time']['hour'] + ":" +astronomy['moon_phase']['current_time']['minute'])
streamer.log(":cloud: " + CITY + " Weather Conditions",weather_status_icon(conditions, astronomy))
```

```
streamer.log(":crescent_moon: Moon Phase",moon_icon(astronomy['moon_phase']['phaseofMoon']))

streamer.log(":dash: " + CITY + " Wind Direction",wind_dir_icon(conditions, astronomy))

if (METRIC_UNITS):

if isFloat(conditions['current_observation']['temp_c']):

streamer.log(CITY + " Temperature(C)",conditions['current_observation']['temp_c'])


if isFloat(conditions['current_observation']['dewpoint_c']):

streamer.log(CITY + " Dewpoint(C)",conditions['current_observation']['dewpoint_c'])


if isFloat(conditions['current_observation']['wind_kph']):

streamer.log(":dash: " + CITY + " Wind Speed(KPH)",conditions['current_observation']['wind_kph'])


if isFloat(conditions['current_observation']['wind_gust_kph']):streamer.log(":dash: " + CITY + " Wind
Gust(KPH)",conditions['current_observation']['wind_gust_kph'])


if isFloat(conditions['current_observation']['pressure_mb']):

streamer.log(CITY + " Pressure(mb)",conditions['current_observation']['pressure_mb'])


if isFloat(conditions['current_observation']['precip_1hr_metric']):

streamer.log(":umbrella: " + CITY + " Precip 1 Hour(mm)",conditions['current_observation']['precip_1hr_metric'])


if isFloat(conditions['current_observation']['precip_today_metric']):

streamer.log(":umbrella: " + CITY + " Precip Today(mm)",conditions['current_observation']['precip_today_metric'])

else:


if isFloat(conditions['current_observation']['temp_f']):

streamer.log(CITY + " Temperature(F)",conditions['current_observation']['temp_f'])


if isFloat(conditions['current_observation']['dewpoint_f']):

streamer.log(CITY + " Dewpoint(F)",conditions['current_observation']['dewpoint_f'])


if isFloat(conditions['current_observation']['wind_mph']):
```

```
streamer.log(":dash: " + CITY + " Wind Speed(MPH)",conditions['current_observation']['wind_mph'])

if isFloat(conditions['current_observation']['wind_gust_mph']):
streamer.log(":dash: " + CITY + " Wind Gust(MPH)",conditions['current_observation']['wind_gust_mph'])

if isFloat(conditions['current_observation']['pressure_in']):
streamer.log(CITY + " Pressure(IN)",conditions['current_observation']['pressure_in'])

if isFloat(conditions['current_observation']['precip_1hr_in']):
streamer.log(":umbrella: " + CITY + " Precip 1 Hour(IN)",conditions['current_observation']['precip_1hr_in'])

if isFloat(conditions['current_observation']['precip_today_in']):
streamer.log(":umbrella: " + CITY + " Precip Today(IN)",conditions['current_observation']['precip_today_in'])

if isFloat(conditions['current_observation']['solarradiation']):
streamer.log(":sunny: " + CITY + " Solar Radiation (watt/m^2)",conditions['current_observation']['solarradiation'])

if isFloat(humidity):
streamer.log(":droplet: " + CITY + " Humidity(%)",humidity)
if isFloat(conditions['current_observation']['UV']):
streamer.log(":sunny: " + CITY + " UV Index:",conditions['current_observation']['UV'])streamer.flush()
time.sleep(60*MINUTES_BETWEEN_READS)
```

You need to set the desired state and city. You also have to insert your Wunderground API key and your Initial State account access key or your data isn't going to go anywhere. The MINUTES_BETWEEN_READS parameter will set how often your script will poll the Wunderground API for weather information. 15 minutes provides a nice interval long-term. For the sake of short-term testing, you can set this to 0.5 minutes. The METRIC_UNITS option allows you to choose between Fahrenheit/Celsius, MPH/KPH, etc.

Once you have your parameters set, you are ready to run your script:

```
$ python wunderground.py
```

If you are ssh'ing into your Pi and want to leave this script running uninterrupted for a long time, you can use the nohup command (no hang-up) as follows:

```
$ nohup python wunderground.py &
```

This script is going to do a bit more than just read the weather data and send it to Initial State. This script is going to take advantage of the emoji support built into Initial State's tools to make the dashboard a bit more sexy. You can see the logic used to take the weather status from the current_observation -> icon status and convert it to an emoji token in the weather_icon function. Something similar happens for the moon phase in the moon_icon function and wind direction in the wind_dir_icon function.

Go to your Initial State account and look at your data. The screenshot of the dashboard above was taken after 9 days of data collection running at 15 minute intervals. You can edit your dashboard by changing chart types with the cog icon in the top right corner of each tile and by right-clicking on a tile to resize/move it around.

Part 3. A Sense HAT Dashboard

The Sense HAT is an add-on board for the Raspberry Pi that is full of sensors, LEDs, and a tiny joystick. It costs $30 and you can get it from places like http://www.mcmelectronics.com and http://www.element14.com. What is so great about this little add-on is that it is incredibly easy to install and use thanks to a fantastic Python library that you can quickly install. For this project, we will use the temperature, humidity, and barometric pressure sensors on the Sense HAT. Let's set it up.

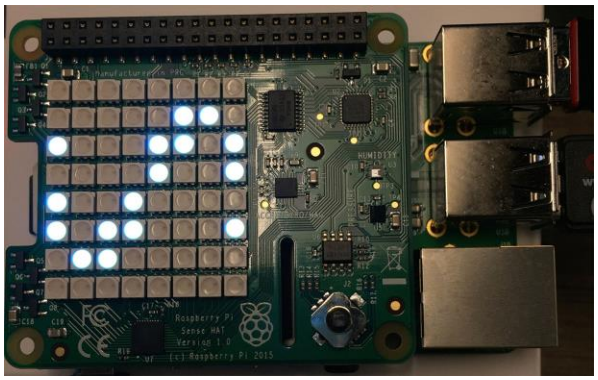

Fig.4 SenseHat

Let's test our Sense HAT to make sure everything is working. We will use the script located at https://github.com/InitialState/wunderground-sensehat/blob/master/sensehat.py. You can copy this script to a file on your Pi or access it from our Github repository that we cloned earlier. Change into your wunderground-sensehat directory and then nano into your sensehat.py file by typing:

```
$ nano sensehat.py
```

Notice on the first line that we are importing the SenseHat library into the script. Before you run this script, we need to setup our user parameters.

```
# --------- User Settings ---------
CITY = "Nashville"
BUCKET_NAME = ":partly_sunny: " + CITY + " Weather"
BUCKET_KEY = "sensehat"
ACCESS_KEY = "Your_Access_Key"
SENSOR_LOCATION_NAME = "Office"
MINUTES_BETWEEN_SENSEHAT_READS = 0.1
# ---------------------------------
```

Specifically, you need to set your ACCESS_KEY to your Initial State account access key. Notice how easy it is to read data from the Sense HAT in a single line of Python (e.g. sense.get_temperature()).

At a command prompt on your Pi, run the script:

```
$ sudo python sensehat.py
```

Go to your Initial State account and view the new data bucket created by the Sense HAT.

We are ready to put it all together and create our hyper-local weather dashboard.

Part 4. Hyper Local Weather Dashboard

The final step in this project is simply combining our Wunderground script and our Sense HAT script into a single Python script. We will be using https://github.com/InitialState/wunderground-sensehat/blob/master/sensehat_wunderground.py for this last step. Copy this file to your Pi or access it via the Github repository that we cloned earlier in this tutorial. Change into your wunderground-sensehat directory, and then nano into the sensehat_wunderground.py file by typing:

```
$ nano sensehat_wunderground.py
```

Modify the user section near the top of the file:

```
# --------- User Settings ---------
STATE = "CA"
CITY = "San_Francisco"
SENSOR_LOCATION_NAME = "Office"
WUNDERGROUND_API_KEY = "PLACE YOUR WUNDERGROUND API KEY HERE"
BUCKET_NAME = ":partly_sunny: " + CITY + " Weather"
BUCKET_KEY = "shwu1"
ACCESS_KEY = "PLACE YOUR INITIAL STATE ACCESS KEY HERE"
MINUTES_BETWEEN_READS = 15
# ---------------------------------
```

Make sure you put your Wunderground API key, Initial State account access key, and desired city/state in this section. Specify the name of the location that your Sense HAT will be collecting environmental data in the SENSOR_LOCATION_NAME variable.

Run the script on your Pi:

```
$ sudo python sensehat_wunderground.py
```

If you are ssh'ing into your Pi and want this script to run uninterrupted for a long time, run the script using the nohup (no hang-up) command:

```
$ nohup sudo python sensehat_wunderground.py &
```

After a couple of days, it is interesting to compare the temperature changes in your room versus the temperature changes outside. Same with humidity. If you want to add more sensors to the same dashboard, simply send the data to the same BUCKET_KEY specified in the User Settings (along with the same ACCESS_KEY). These additional sensors can be on any device and located anywhere and still send data to the same bucket. For example, you could have 10 different temperature sensors connected to 10 different types of single-board computers (Pi, Arduino, BeagleBone, Edison) and have them all streaming data into your one hyper-local weather dashboard

Code: sensehat-wunderground.py

```
import os
import urllib2
import json
import glob
import time
import RPi.GPIO as io
from ISStreamer.Streamer import Streamer
from sense_hat import SenseHat

# --------- User Settings ---------
STATE = "CA"
CITY = "San_Francisco"
SENSOR_LOCATION_NAME = "Office"
WUNDERGROUND_API_KEY = "PLACE YOUR WUNDERGROUND API KEY HERE"
BUCKET_NAME = ":partly_sunny: " + CITY + " Weather"
BUCKET_KEY = "shwu1"
ACCESS_KEY = "PLACE YOUR INITIAL STATE ACCESS KEY HERE"
MINUTES_BETWEEN_READS = 15
METRIC_UNITS = False
# ---------------------------------


def isFloat(string):
    try:
        float(string)
        return True
    except ValueError:
        return False
```

```
        def get_conditions():

api_conditions_url = "http://api.wunderground.com/api/" + WUNDERGROUND_API_KEY +
"/conditions/q/" + STATE + "/" + CITY + ".json"

                try:

                        f = urllib2.urlopen(api_conditions_url)

                except:

                        print "Failed to get conditions"

                        return []

                json_conditions = f.read()

                f.close()

                return json.loads(json_conditions)


        def get_astronomy():

api_astronomy_url = "http://api.wunderground.com/api/" + WUNDERGROUND_API_KEY +
"/astronomy/q/" + STATE + "/" + CITY + ".json"

                try:

                        f = urllib2.urlopen(api_astronomy_url)

                except:

                        print "Failed to get astronomy"

                        return []

                json_astronomy = f.read()

                f.close()

                return json.loads(json_astronomy)


        def is_night(astronomy):

                sunrise_hour = int(astronomy['moon_phase']['sunrise']['hour'])

                sunrise_min  = int(astronomy['moon_phase']['sunrise']['minute'])

                sunset_hour  = int(astronomy['moon_phase']['sunset']['hour'])
```

```
        sunset_min   = int(astronomy['moon_phase']['sunset']['minute'])
        current_hour = int(astronomy['moon_phase']['current_time']['hour'])
        current_min  = int(astronomy['moon_phase']['current_time']['minute'])
        if ( (current_hour < sunrise_hour) or
            (current_hour > sunset_hour) or
            ((current_hour == sunrise_hour) and
             (current_min < sunrise_min)) or
            ((current_hour == sunset_hour) and
             (current_min > sunset_min)) ):
                return True
        return False


def moon_icon(moon_phase):
        icon = {
                "New Moon"       : ":new_moon:",
                "Waxing Crescent" : ":waxing_crescent_moon:",
                "First Quarter"   : ":first_quarter_moon:",
                "Waxing Gibbous"  : ":waxing_gibbous_moon:",
                "Full Moon"      : ":full_moon:",
                "Full"           : ":full_moon:",
                "Waning Gibbous"  : ":waning_gibbous_moon:",
                "Last Quarter"    : ":last_quarter_moon:",
                "Waning Crescent" : ":waning_crescent_moon:",
        }
        return icon.get(moon_phase,":crescent_moon:")


def weather_icon(weather_conditions):
        icon = {
```

```
            "clear"           : ":sun_with_face:",
            "cloudy"          : ":cloud:",
            "flurries"        : ":snowflake:",
            "fog"             : ":foggy:",
            "hazy"            : ":foggy:",
            "mostlycloudy"    : ":cloud:",
            "mostlysunny"     : ":sun_with_face:",
            "partlycloudy"    : ":partly_sunny:",
            "partlysunny"     : ":partly_sunny:",
            "sleet"           : ":sweat_drops: :snowflake:",
            "rain"            : ":umbrella:",
            "snow"            : ":snowflake:",
            "sunny"           : ":sun_with_face:",
            "tstorms"         : ":zap: :umbrella:",
            "unknown"         : ":sun_with_face:",
        }
        return icon.get(weather_conditions,":sun_with_face:")


def weather_status_icon(conditions, astronomy):
        moon_phase = astronomy['moon_phase']['phaseofMoon']
        weather_conditions = conditions['current_observation']['icon']
        icon = weather_icon(weather_conditions)
        if is_night(astronomy):
            if ((icon == ":sunny:") or
               (icon == ":partly_sunny:") or
               (icon == ":sun_with_face:")):
                    return moon_icon(moon_phase)
        return icon
```

```
def wind_dir_icon(conditions, astronomy):
        icon = {
                "East"     : ":arrow_right:",
                "ENE"      : ":arrow_upper_right:",
                "ESE"      : ":arrow_lower_right:",
                "NE"       : ":arrow_upper_right:",
                "NNE"      : ":arrow_upper_right:",
                "NNW"      : ":arrow_upper_left:",
                "North"    : ":arrow_up:",
                "NW"       : ":arrow_upper_left:",
                "SE"       : ":arrow_lower_right:",
                "South"    : ":arrow_down:",
                "SSE"      : ":arrow_lower_right:",
                "SSW"      : ":arrow_lower_left:",
                "SW"       : ":arrow_lower_left:",
                "Variable" : ":arrows_counterclockwise:",
                "West"     : ":arrow_left:",
                "WNW"      : ":arrow_upper_left:",
                "WSW"      : ":arrow_lower_left:",
        }
        return icon.get(conditions['current_observation']['wind_dir'],":crescent_moon:")


def main():
        sense = SenseHat()
        conditions = get_conditions()
        astronomy = get_astronomy()
        if ('current_observation' not in conditions) or ('moon_phase' not in astronomy):
```

```
                print "Error! Wunderground API call failed, check your STATE and CITY and make sure your Wunderground API key is valid!"

                if 'error' in conditions['response']:

                        print "Error Type: " + conditions['response']['error']['type']

                        print "Error Description: " + conditions['response']['error']['description']

                        exit()

        else:

                streamer = Streamer(bucket_name=BUCKET_NAME, bucket_key=BUCKET_KEY, access_key=ACCESS_KEY)

                streamer.log(":house: Location",conditions['current_observation']['display_location']['full'])

        while True:

# ---------- Sense Hat ----------

                # Read the sensors

                temp_c = sense.get_temperature()

                humidity = sense.get_humidity()

                pressure_mb = sense.get_pressure()


                # Format the data

                temp_f = temp_c * 9.0 / 5.0 + 32.0

                temp_f = float("{0:.2f}".format(temp_f))

                temp_c = float("{0:.2f}".format(temp_c))

                humidity = float("{0:.2f}".format(humidity))

                pressure_in = 0.0295301*(pressure_mb)

                pressure_in = float("{0:.2f}".format(pressure_in))

                pressure_mb = float("{0:.2f}".format(pressure_mb))
```

```
        # Print and stream
if (METRIC_UNITS):
print SENSOR_LOCATION_NAME + " Temperature(C): " + str(temp_c)
print SENSOR_LOCATION_NAME + " Pressure(mb): " + str(pressure_mb)
streamer.log(":sunny: " + SENSOR_LOCATION_NAME + " Temperature(C)", temp_c)
streamer.log(":cloud: " + SENSOR_LOCATION_NAME + " Pressure (mb)", pressure_mb)
else:
print SENSOR_LOCATION_NAME + " Temperature(F): " + str(temp_f)
print SENSOR_LOCATION_NAME + " Pressure(IN): " + str(pressure_in)
streamer.log(":sunny: " + SENSOR_LOCATION_NAME + " Temperature(F)", temp_f)
streamer.log(":cloud: " + SENSOR_LOCATION_NAME + " Pressure (IN)", pressure_in)
print SENSOR_LOCATION_NAME + " Humidity(%): " + str(humidity)
streamer.log(":sweat_drops: " + SENSOR_LOCATION_NAME + " Humidity(%)", humidity)

# ---------- Wunderground ----------
conditions = get_conditions()
astronomy = get_astronomy()
if ('current_observation' not in conditions) or ('moon_phase' not in astronomy):
print "Error! Wunderground API call failed. Skipping a reading then continuing ..."
else:
humidity_pct = conditions['current_observation']['relative_humidity']
humidity = humidity_pct.replace("%","")
```

```
# Stream valid conditions to Initial State

streamer.log(":cloud: " + CITY + " Weather Conditions",weather_status_icon(conditions, astronomy))

streamer.log(":crescent_moon: Moon Phase",moon_icon(astronomy['moon_phase']['phaseofMoon']))

streamer.log(":dash: " + CITY + " Wind Direction",wind_dir_icon(conditions, astronomy))

if (METRIC_UNITS):

if isFloat(conditions['current_observation']['temp_c']): streamer.log(CITY + " Temperature(C)",conditions['current_observation']['temp_c'])

if isFloat(conditions['current_observation']['dewpoint_c']):

streamer.log(CITY + " Dewpoint(C)",conditions['current_observation']['dewpoint_c'])

if isFloat(conditions['current_observation']['wind_kph']):

streamer.log(":dash: " + CITY + " Wind Speed(KPH)",conditions['current_observation']['wind_kph'])

if isFloat(conditions['current_observation']['wind_gust_kph']):

streamer.log(":dash: " + CITY + " Wind Gust(KPH)",conditions['current_observation']['wind_gust_kph'])

if isFloat(conditions['current_observation']['pressure_mb']):

streamer.log(CITY + " Pressure(mb)",conditions['current_observation']['pressure_mb'])

if isFloat(conditions['current_observation']['precip_1hr_metric']):

streamer.log(":umbrella: " + CITY + " Precip 1 Hour(mm)",conditions['current_observation']['precip_1hr_metric'])

                                        if isFloat(conditions['current_observation']['precip_today_metric']):

streamer.log(":umbrella: " + CITY + " Precip Today(mm)",conditions['current_observation']['precip_today_metric'])

else:

if isFloat(conditions['current_observation']['temp_f']):

streamer.log(CITY + " Temperature(F)",conditions['current_observation']['temp_f'])

if isFloat(conditions['current_observation']['dewpoint_f']):

streamer.log(CITY + " Dewpoint(F)",conditions['current_observation']['dewpoint_f'])
```

```
if isFloat(conditions['current_observation']['wind_mph']):

streamer.log(":dash: " + CITY + " Wind
Speed(MPH)",conditions['current_observation']['wind_mph'])

if isFloat(conditions['current_observation']['wind_gust_mph']):

streamer.log(":dash: " + CITY + " Wind
Gust(MPH)",conditions['current_observation']['wind_gust_mph'])

if isFloat(conditions['current_observation']['pressure_in']):

streamer.log(CITY + " Pressure(IN)",conditions['current_observation']['pressure_in'])

if isFloat(conditions['current_observation']['precip_1hr_in']):

streamer.log(":umbrella: " + CITY + " Precip 1
Hour(IN)",conditions['current_observation']['precip_1hr_in'])

if isFloat(conditions['current_observation']['precip_today_in']):

streamer.log(":umbrella: " + CITY + " Precip
Today(IN)",conditions['current_observation']['precip_today_in'])

if isFloat(conditions['current_observation']['solarradiation']):

streamer.log(":sunny: " + CITY + " Solar Radiation
(watt/m^2)",conditions['current_observation']['solarradiation'])

if isFloat(humidity):

streamer.log(":droplet: " + CITY + " Humidity(%)",humidity)

if isFloat(conditions['current_observation']['UV']):

streamer.log(":sunny: " + CITY + " UV Index:",conditions['current_observation']['UV'])

streamer.flush()

time.sleep(60*MINUTES_BETWEEN_READS)

if __name__ == "__main__":

  main()
```

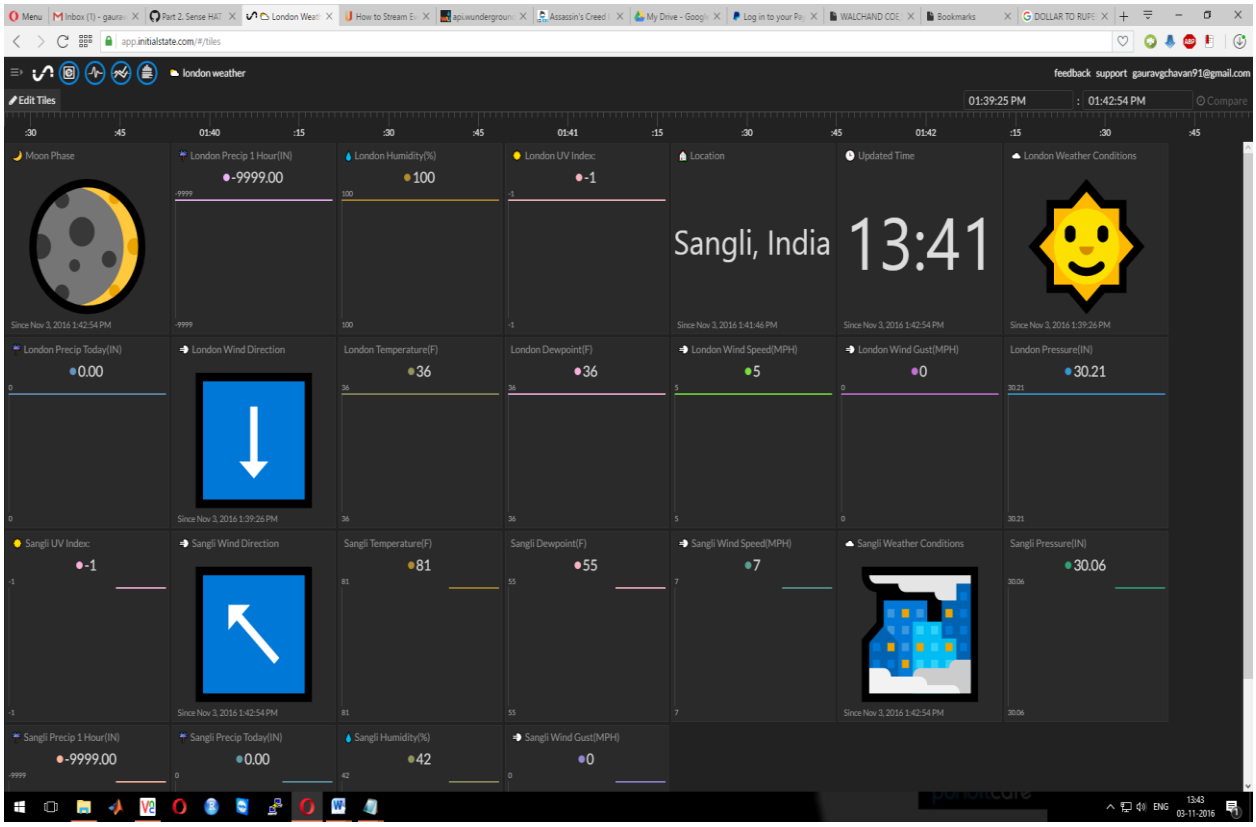

Fig.5. wunderground Dashobard for Sensehat Data