What's That Smell?

Detecting Air Quality with Python, Raspberry Pi, and Redis

Justin Castilla

Senior Developer Advocate @ Redis

justin@redis.com

(AP Photo/ Eric Risberg)

Bummer Introductory Stats for 2020 Wildfires in the United States West Coast:

- 10,274,679 acres of land burned
- 58,258 individual fires
- 176 acres average per fire
- 13,887 buildings destroyed
- Financial loss of 19.884 billion dollars
- 1,200 to 3,000 excess deaths from exposure to wildfire smoke

Bummer Introductory Stats for 2020 Wildfires in the United States West Coast:

- **We learned about fire tornadoes**

Wildfire Smoke - How does it affect us?

- Eye and respiratory tract irritation
- Reduced lung function
- Bronchitis
- Exacerbation of Asthma
- Exacerbation of Heart Failure
- Premature death

Wildfire Smoke - How we measure it

- PM 2.5: Particulate Matter 2.5 micrometers and smaller
- Small enough to pass through to the deepest part of the lungs and into the bloodstream
- AQI (Air Quality Index): a computed value based on PM 2.5 to convey health risks

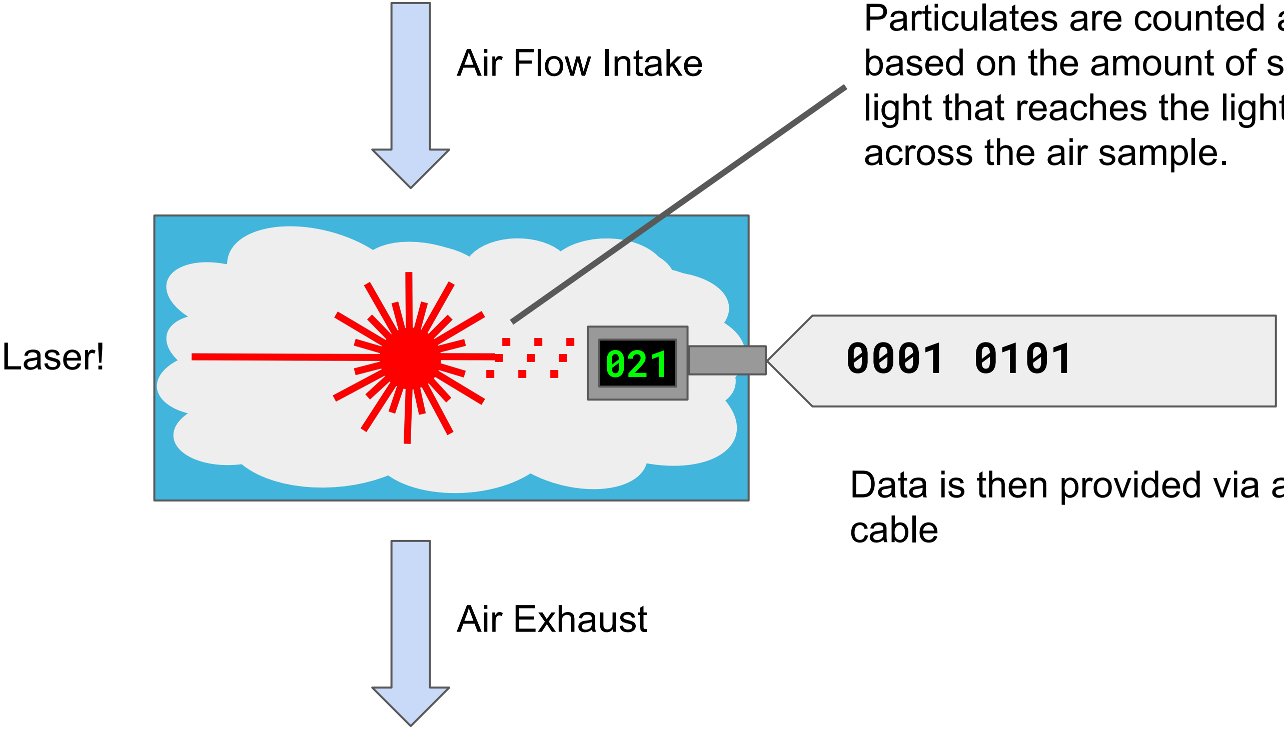
Wildfire Smoke - How we measure it

0 - 50	Good	Air quality is considered satisfactory, and air pollution poses little or no risk
51 - 100	Moderate	Air quality is acceptable; however, for some pollutants there may be a moderate health concern for a very small number of people who are unusually sensitive to air pollution.
101-150	Unhealthy for Sensitive Groups	Members of sensitive groups may experience health effects. The general public is not likely to be affected.
151-200	Unhealthy	Everyone may begin to experience health effects; members of sensitive groups may experience more serious health effects
201-300	Very Unhealthy	Health warnings of emergency conditions. The entire population is more likely to be affected.
300+	Hazardous	Health alert: everyone may experience more serious health effects

Wildfire Smoke - How do we measure it

Plantower PMS 5003 Particulate Matter Sensor

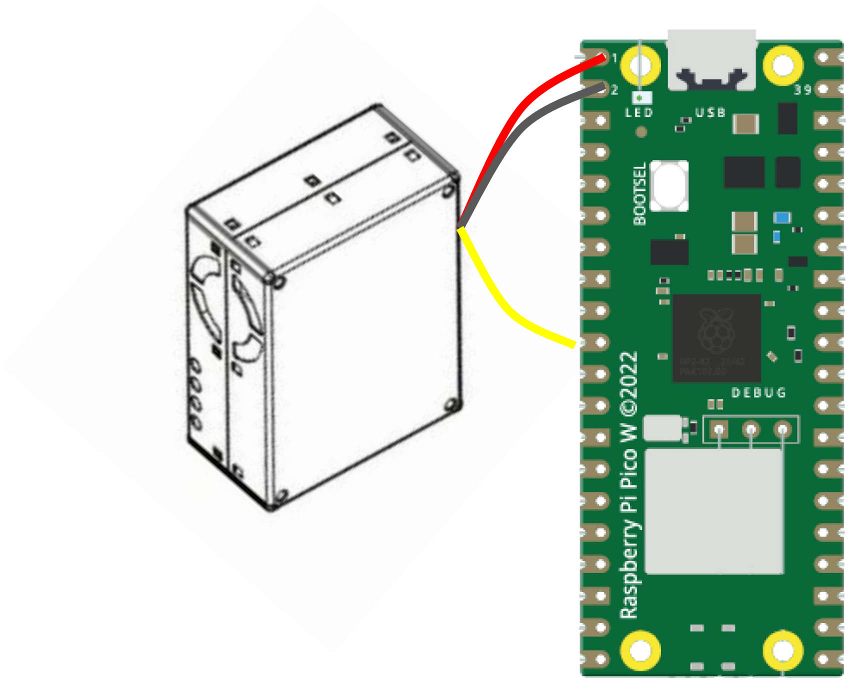
Wildfire Smoke - Plantower PMS5003 breakdown



Particulates are counted and sized based on the amount of scattered light that reaches the light sensor across the air sample.

Data is then provided via a serial cable

The Raspberry Pi Pico W

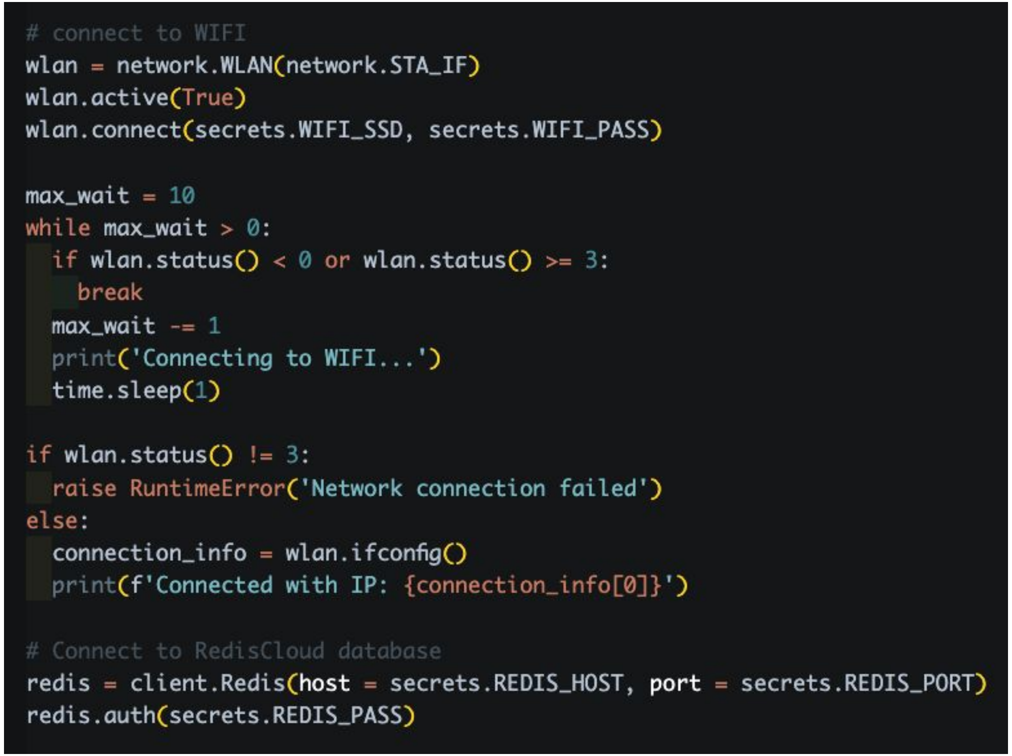


- Capable of running Micropython
- Wireless capabilities
- Dual-core ARM processor,
- 264 kB of SRAM
- 2MB of on-board flash memory
- Only $6.00 (USD)

Redis

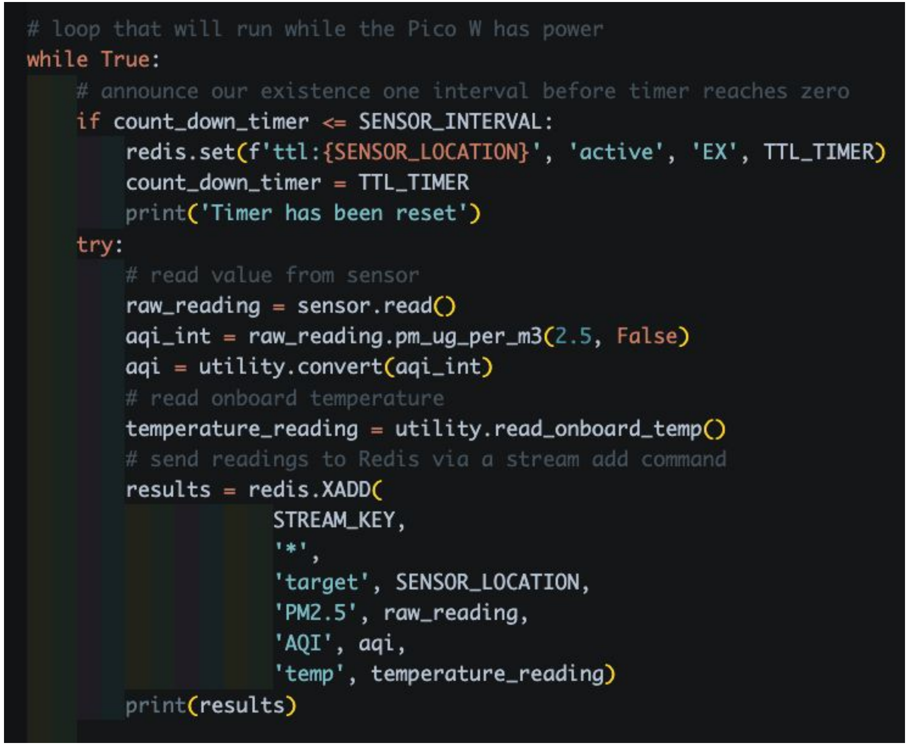
- NoSQL Database
- Runs on RAM, not on hard drives
- Exists on all major cloud providers
- Stores key/value pairs
    - Strings/Numbers
    - Lists/Sets/Sorted Sets
    - TimeSeries
    - JSON / Query
    - Streams

Pi Pico W Code - Connecting to Wifi and Redis



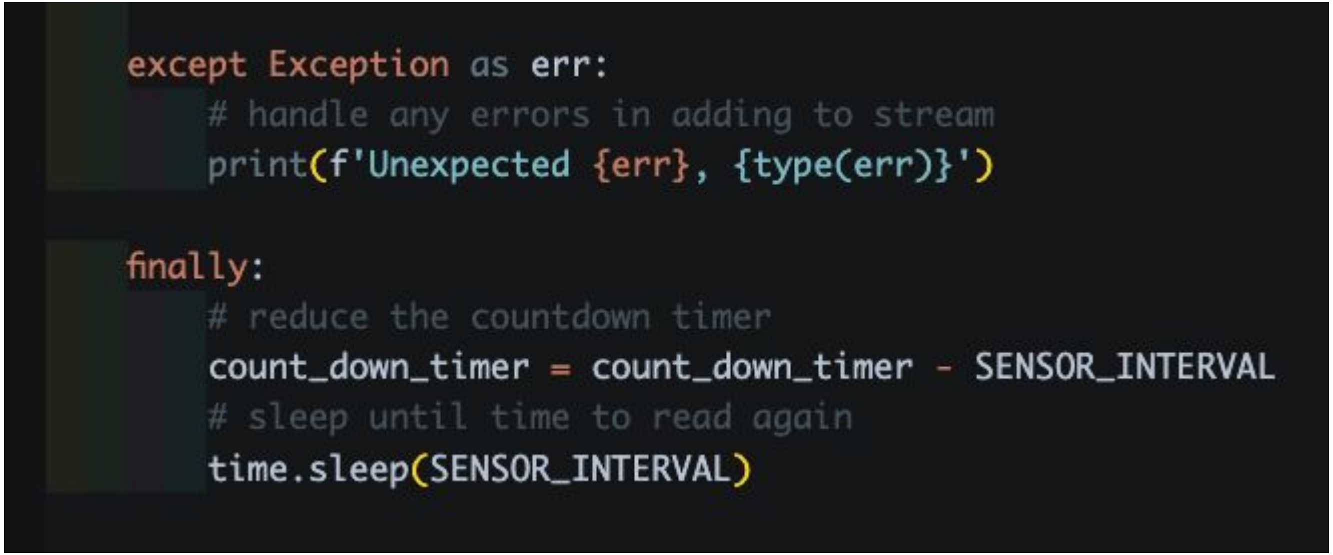
```
# connect to WIFI
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(secrets.WIFI_SSD, secrets.WIFI_PASS)

max_wait = 10
while max_wait > 0:
    if wlan.status() < 0 or wlan.status() >= 3:
        break
    max_wait -= 1
    print('Connecting to WIFI...')
    time.sleep(1)

if wlan.status() != 3:
    raise RuntimeError('Network connection failed')
else:
    connection_info = wlan.ifconfig()
    print(f'Connected with IP: {connection_info[0]}')

# Connect to RedisCloud database
redis = client.Redis(host = secrets.REDIS_HOST, port = secrets.REDIS_PORT)
redis.auth(secrets.REDIS_PASS)
```

Pi Pico W Code - Reading the sensor and saying Hi!



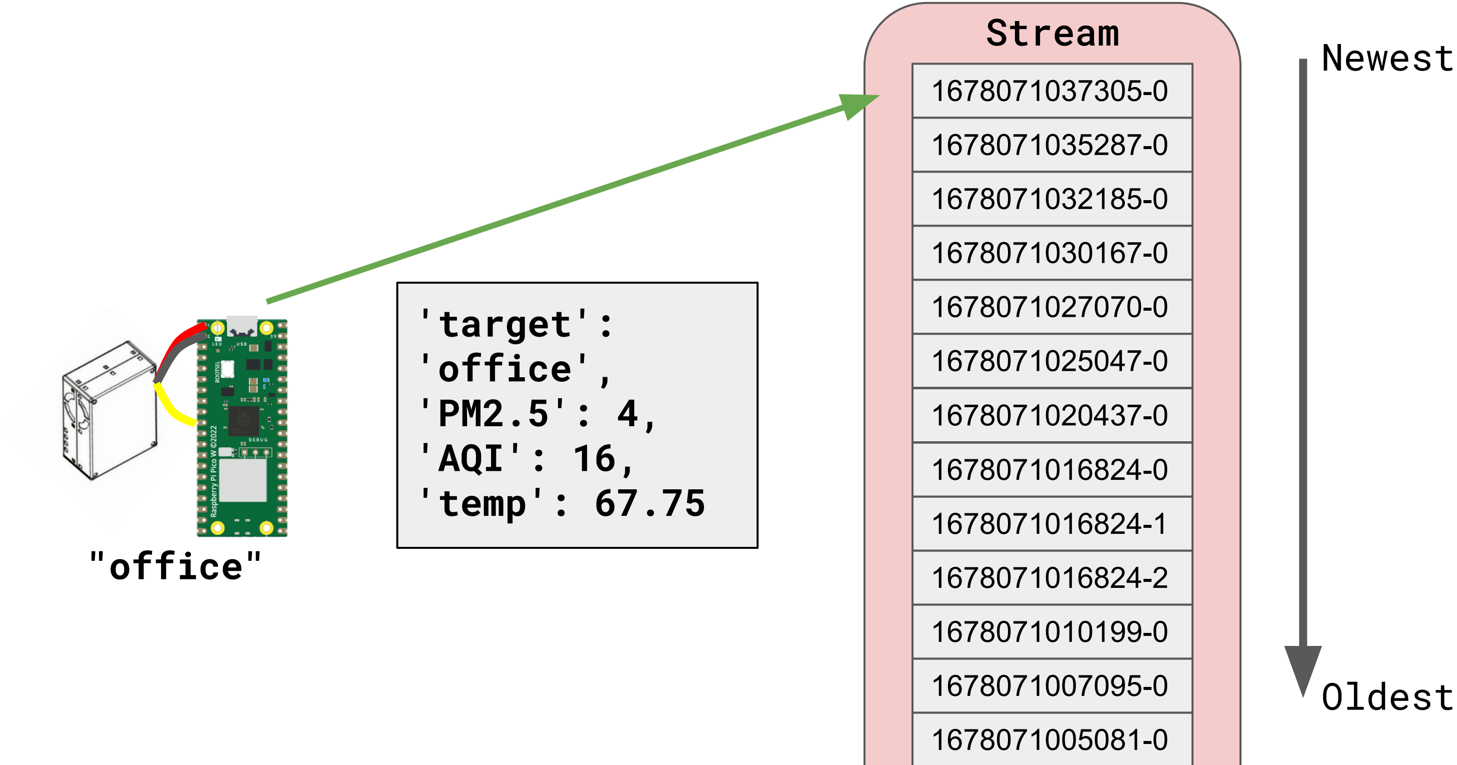
```
# loop that will run while the Pico W has power
while True:
    # announce our existence one interval before timer reaches zero
    if count_down_timer <= SENSOR_INTERVAL:
        redis.set(f'ttl:{SENSOR_LOCATION}', 'active', 'EX', TTL_TIMER)
        count_down_timer = TTL_TIMER
        print('Timer has been reset')
    try:
        # read value from sensor
        raw_reading = sensor.read()
        aqi_int = raw_reading.pm_ug_per_m3(2.5, False)
        aqi = utility.convert(aqi_int)
        # read onboard temperature
        temperature_reading = utility.read_onboard_temp()
        # send readings to Redis via a stream add command
        results = redis.XADD(
                    STREAM_KEY,
                    '*',
                    'target', SENSOR_LOCATION,
                    'PM2.5', raw_reading,
                    'AQI', aqi,
                    'temp', temperature_reading)
        print(results)
```

Pi Pico W Code - Update timer for next iteration



```
    except Exception as err:
        # handle any errors in adding to stream
        print(f'Unexpected {err}, {type(err)}')

    finally:
        # reduce the countdown timer
        count_down_timer = count_down_timer - SENSOR_INTERVAL
        # sleep until time to read again
        time.sleep(SENSOR_INTERVAL)
```

Overview - What's going on?



"office"

```
'target':
'office',
'PM2.5': 4,
'AQI': 16,
'temp': 67.75
```

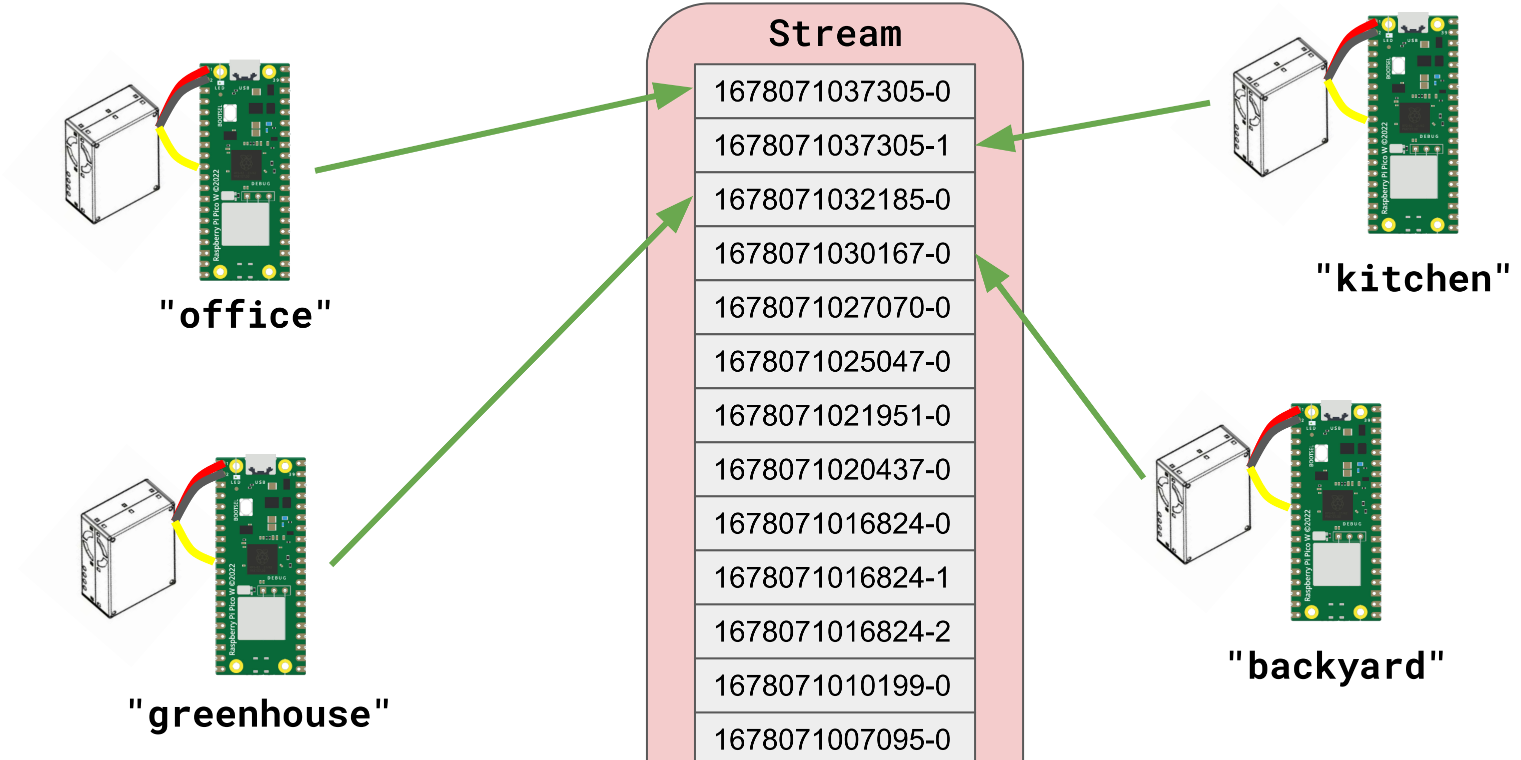
Stream

1678071037305-0
1678071035287-0
1678071032185-0
1678071030167-0
1678071027070-0
1678071025047-0
1678071020437-0
1678071016824-0
1678071016824-1
1678071016824-2
1678071010199-0
1678071007095-0
1678071005081-0

Newest

Oldest

Producers - Let's scale out!



Stream

1678071037305-0
1678071037305-1
1678071032185-0
1678071030167-0
1678071027070-0
1678071025047-0
1678071021951-0
1678071020437-0
1678071016824-0
1678071016824-1
1678071016824-2
1678071010199-0
1678071007095-0

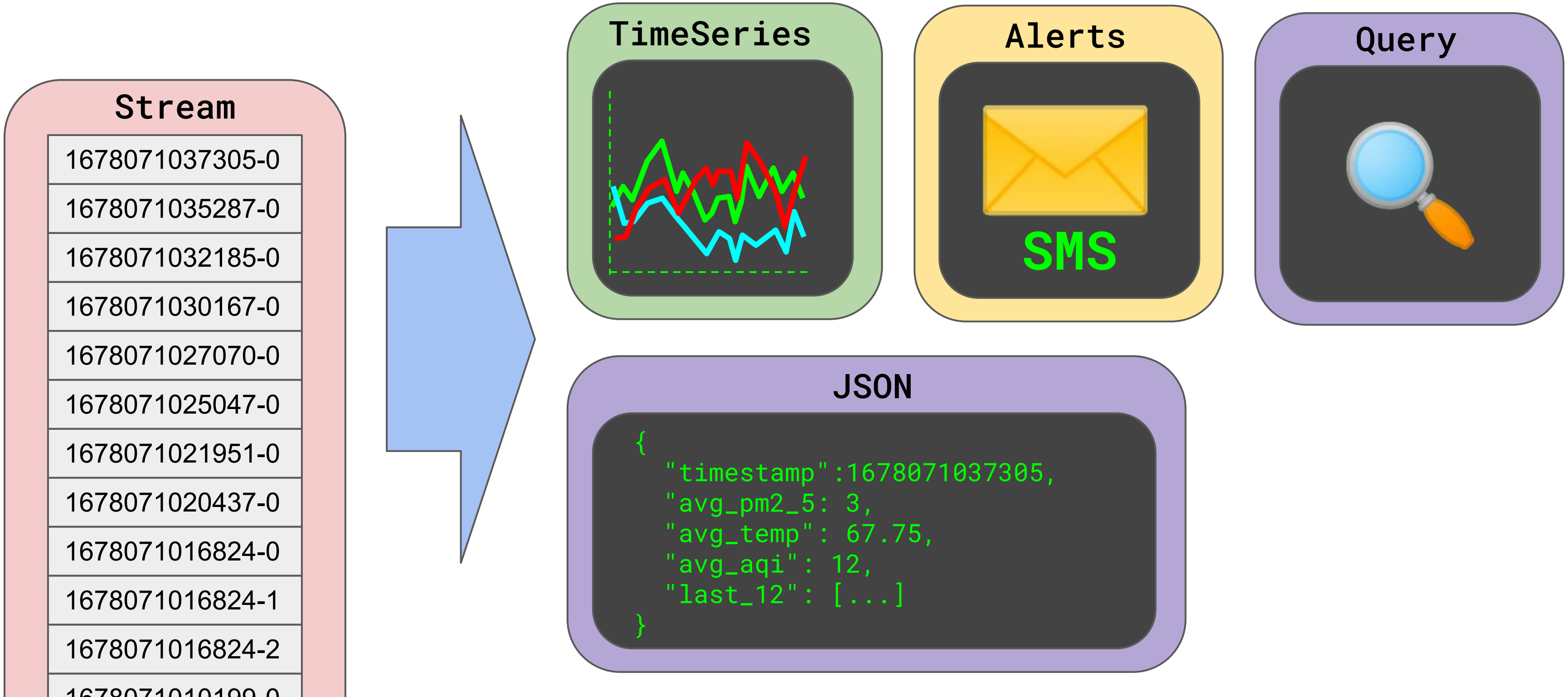
"office"

"kitchen"

"greenhouse"

"backyard"

Consumers - Making the data work



Stream

1678071037305-0
1678071035287-0
1678071032185-0
1678071030167-0
1678071027070-0
1678071025047-0
1678071021951-0
1678071020437-0
1678071016824-0
1678071016824-1
1678071016824-2
1678071010199-0

TimeSeries

Alerts

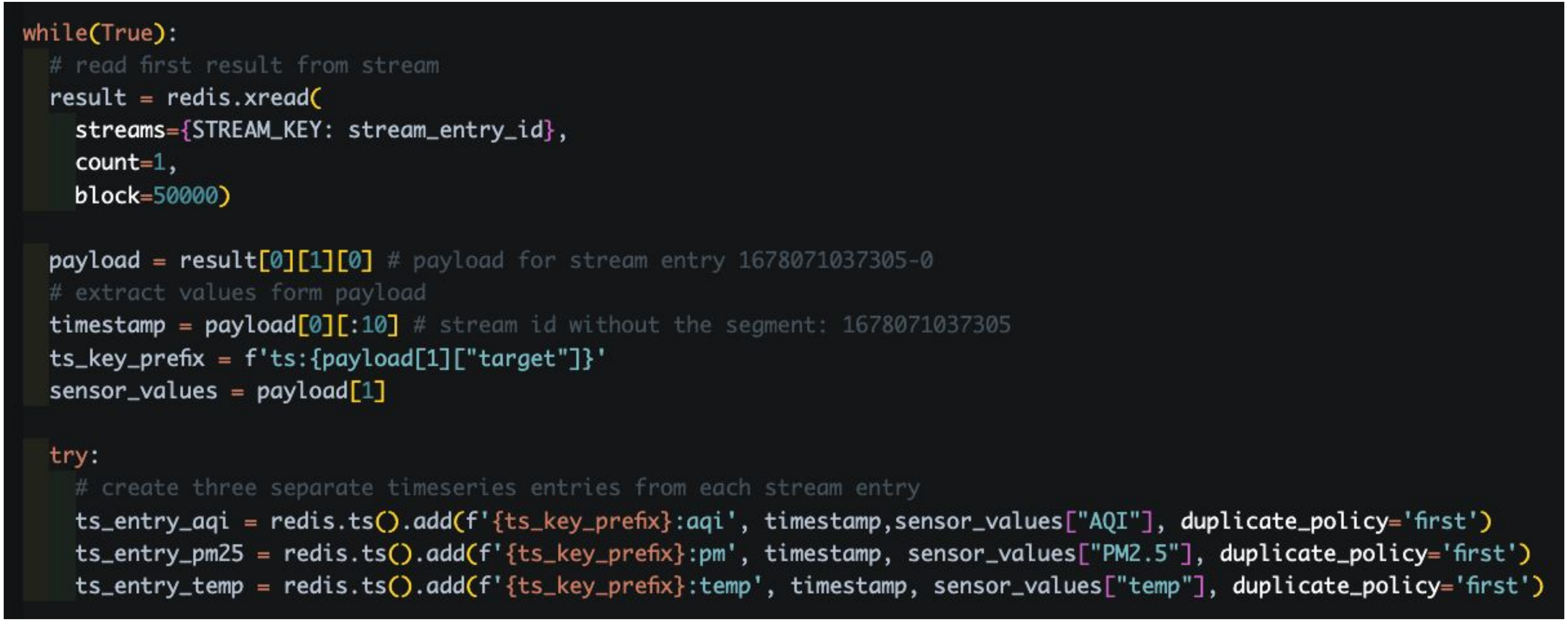
SMS

Query

JSON

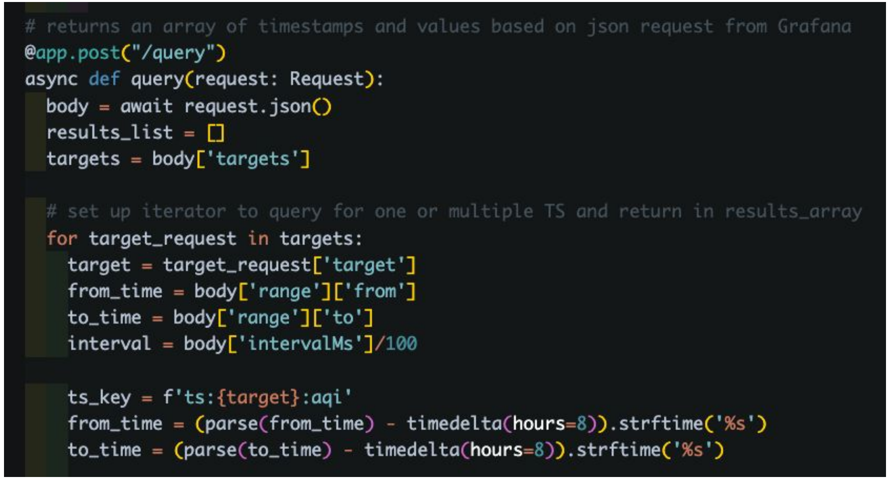
```
{
  "timestamp":1678071037305,
  "avg_pm2_5: 3,
  "avg_temp": 67.75,
  "avg_aqi": 12,
  "last_12": [...]
}
```

Consumers - Creating a TimeSeries

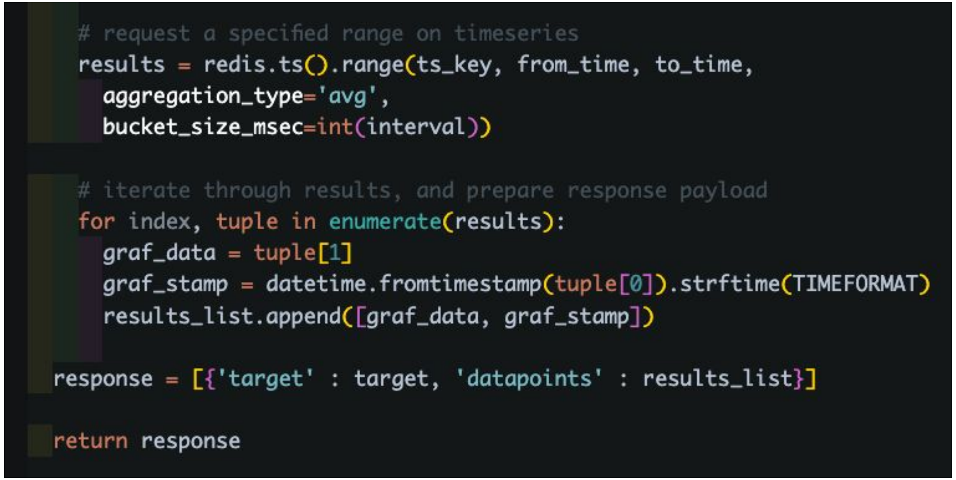


```
while(True):
  # read first result from stream
  result = redis.xread(
    streams={STREAM_KEY: stream_entry_id},
    count=1,
    block=50000)

  payload = result[0][1][0] # payload for stream entry 1678071037305-0
  # extract values form payload
  timestamp = payload[0][:10] # stream id without the segment: 1678071037305
  ts_key_prefix = f'ts:{payload[1]["target"]}'
  sensor_values = payload[1]

  try:
    # create three separate timeseries entries from each stream entry
    ts_entry_aqi = redis.ts().add(f'{ts_key_prefix}:aqi', timestamp,sensor_values["AQI"], duplicate_policy='first')
    ts_entry_pm25 = redis.ts().add(f'{ts_key_prefix}:pm', timestamp, sensor_values["PM2.5"], duplicate_policy='first')
    ts_entry_temp = redis.ts().add(f'{ts_key_prefix}:temp', timestamp, sensor_values["temp"], duplicate_policy='first')
```

Consumers - Sending the data to Grafana

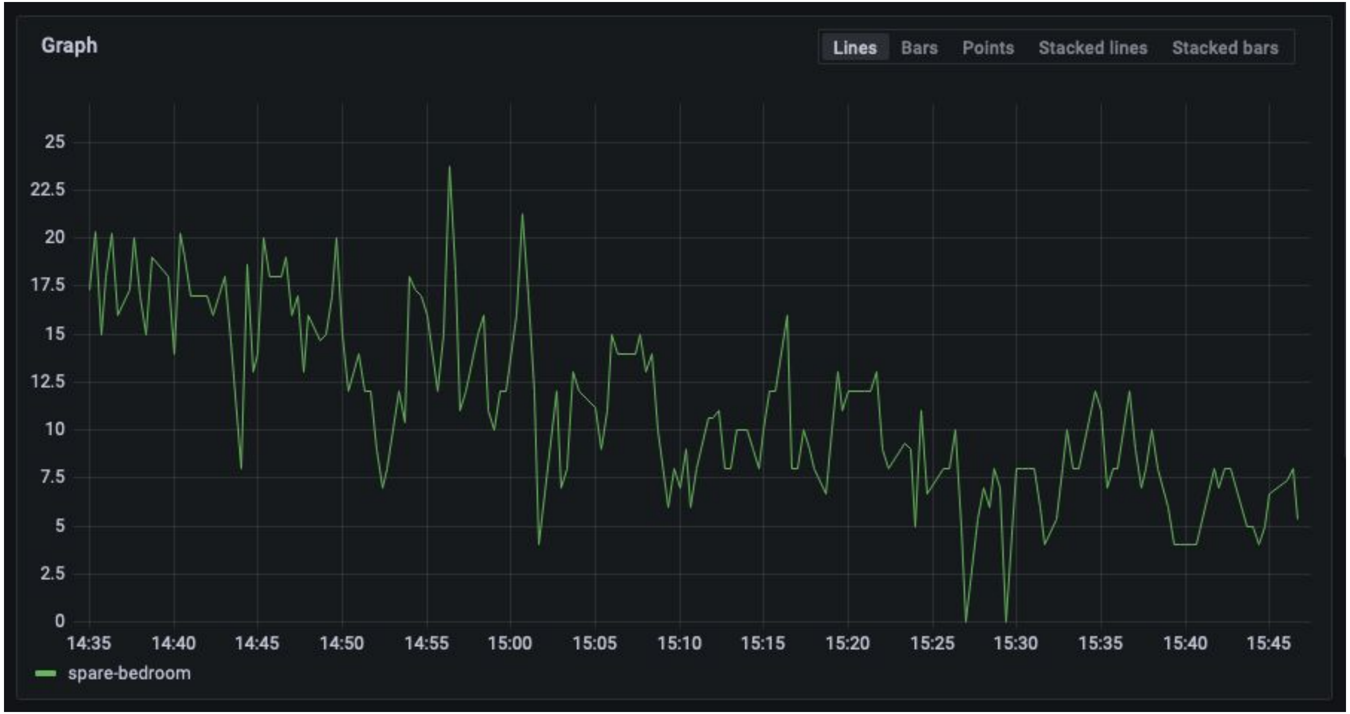


```
# returns an array of timestamps and values based on json request from Grafana
@app.post("/query")
async def query(request: Request):
    body = await request.json()
    results_list = []
    targets = body['targets']

    # set up iterator to query for one or multiple TS and return in results_array
    for target_request in targets:
        target = target_request['target']
        from_time = body['range']['from']
        to_time = body['range']['to']
        interval = body['intervalMs']/100

        ts_key = f'ts:{target}:aqi'
        from_time = (parse(from_time) - timedelta(hours=8)).strftime('%s')
        to_time = (parse(to_time) - timedelta(hours=8)).strftime('%s')
```

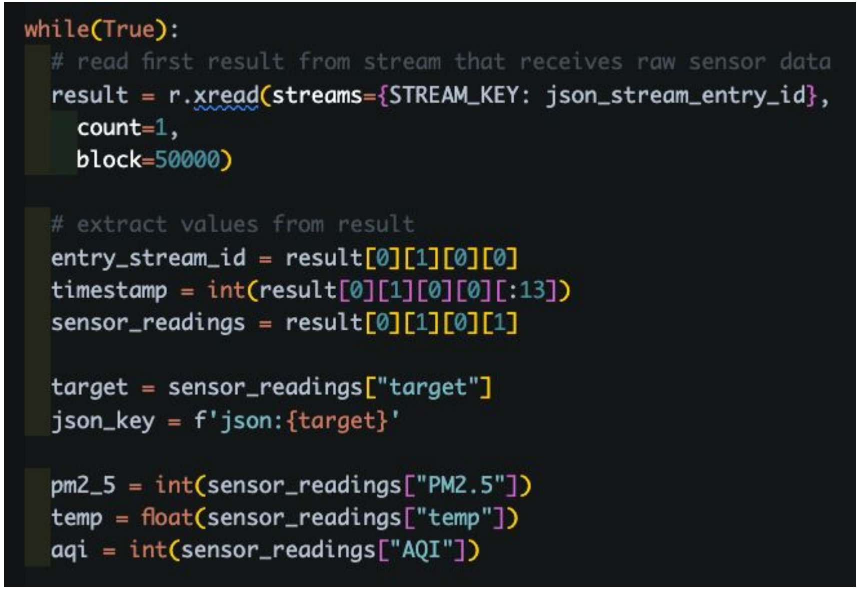


```
        # request a specified range on timeseries
        results = redis.ts().range(ts_key, from_time, to_time,
            aggregation_type='avg',
            bucket_size_msec=int(interval))

        # iterate through results, and prepare response payload
        for index, tuple in enumerate(results):
            graf_data = tuple[1]
            graf_stamp = datetime.fromtimestamp(tuple[0]).strftime(TIMEFORMAT)
            results_list.append([graf_data, graf_stamp])

    response = [{'target' : target, 'datapoints' : results_list}]

    return response
```

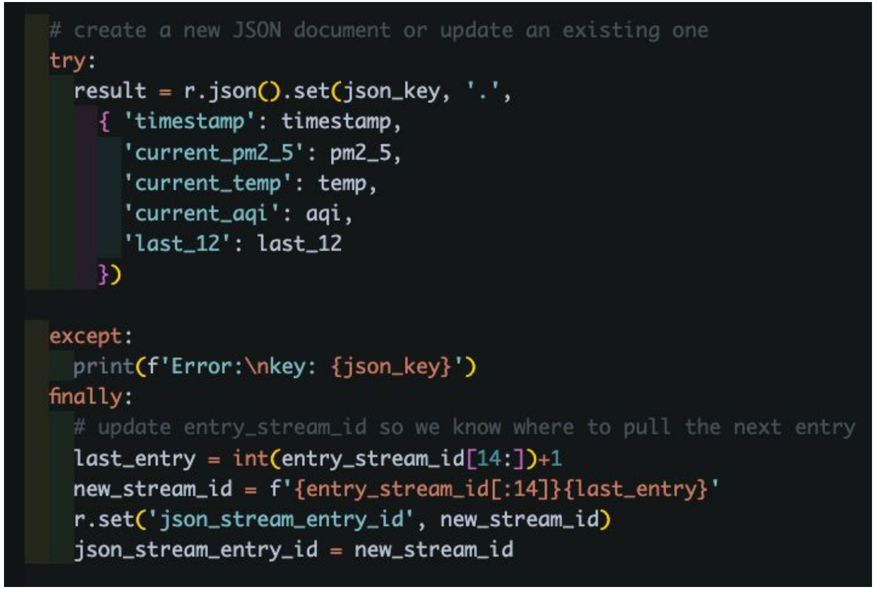
Consumers - Viewing a TimeSeries in Grafana



Consumers - Creating/Updating JSON documents

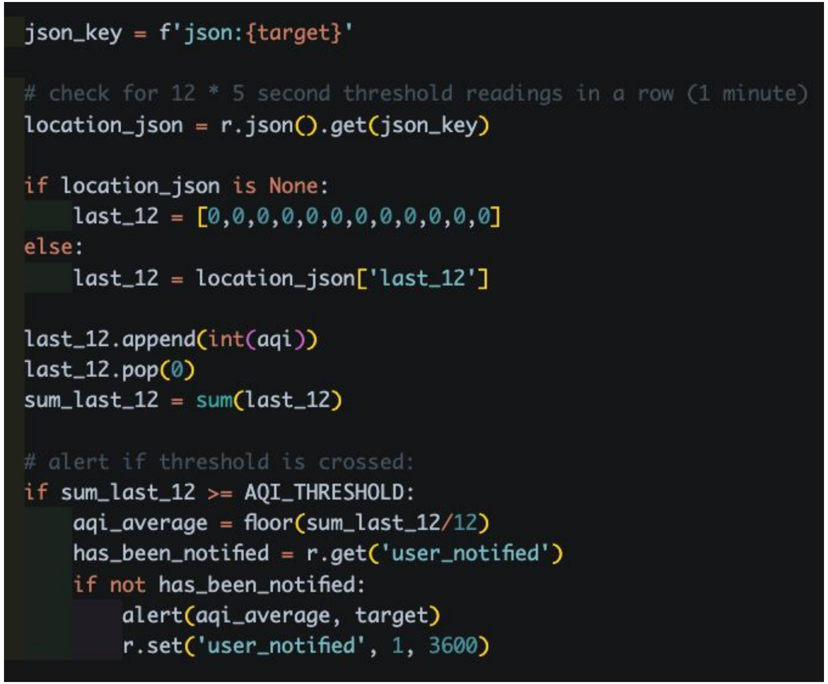


```
while(True):
  # read first result from stream that receives raw sensor data
  result = r.xread(streams={STREAM_KEY: json_stream_entry_id},
    count=1,
    block=50000)

  # extract values from result
  entry_stream_id = result[0][1][0][0]
  timestamp = int(result[0][1][0][0][:13])
  sensor_readings = result[0][1][0][1]

  target = sensor_readings["target"]
  json_key = f'json:{target}'

  pm2_5 = int(sensor_readings["PM2.5"])
  temp = float(sensor_readings["temp"])
  aqi = int(sensor_readings["AQI"])
```



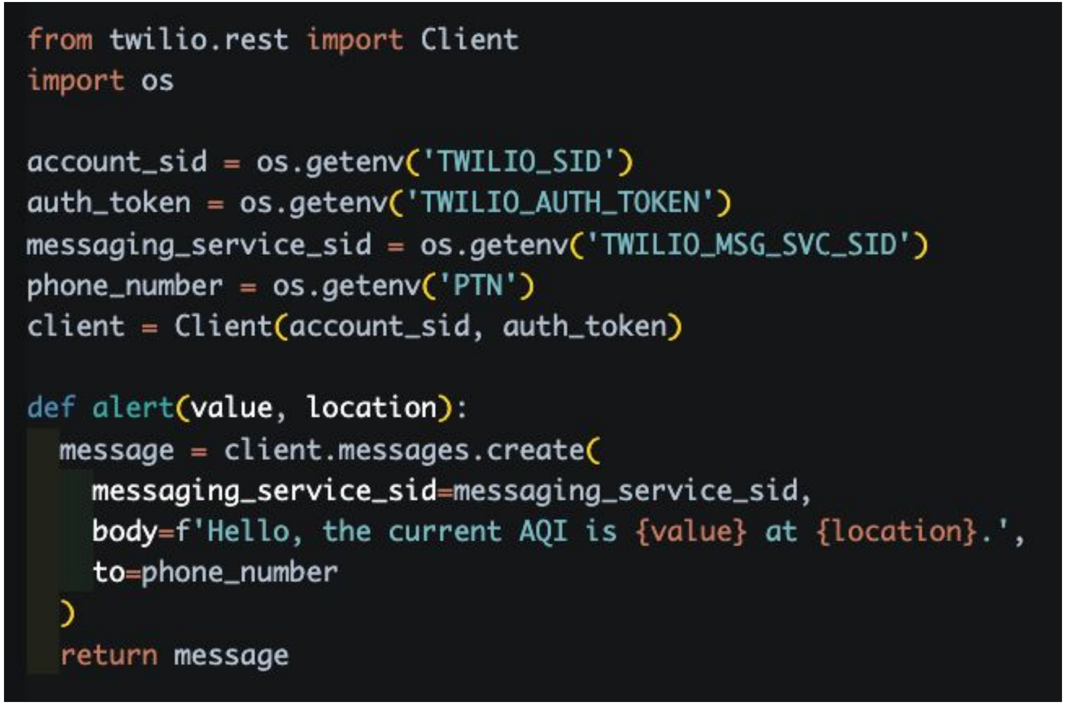
```
# create a new JSON document or update an existing one
try:
  result = r.json().set(json_key, '.',
    { 'timestamp': timestamp,
      'current_pm2_5': pm2_5,
      'current_temp': temp,
      'current_aqi': aqi,
      'last_12': last_12
    })

except:
  print(f'Error:\nkey: {json_key}')
finally:
  # update entry_stream_id so we know where to pull the next entry
  last_entry = int(entry_stream_id[14:])+1
  new_stream_id = f'{entry_stream_id[:14]}{last_entry}'
  r.set('json_stream_entry_id', new_stream_id)
  json_stream_entry_id = new_stream_id
```

Bonus SMS notifications!



```
json_key = f'json:{target}'

# check for 12 * 5 second threshold readings in a row (1 minute)
location_json = r.json().get(json_key)

if location_json is None:
    last_12 = [0,0,0,0,0,0,0,0,0,0,0,0]
else:
    last_12 = location_json['last_12']

last_12.append(int(aqi))
last_12.pop(0)
sum_last_12 = sum(last_12)

# alert if threshold is crossed:
if sum_last_12 >= AQI_THRESHOLD:
    aqi_average = floor(sum_last_12/12)
    has_been_notified = r.get('user_notified')
    if not has_been_notified:
        alert(aqi_average, target)
        r.set('user_notified', 1, 3600)
```

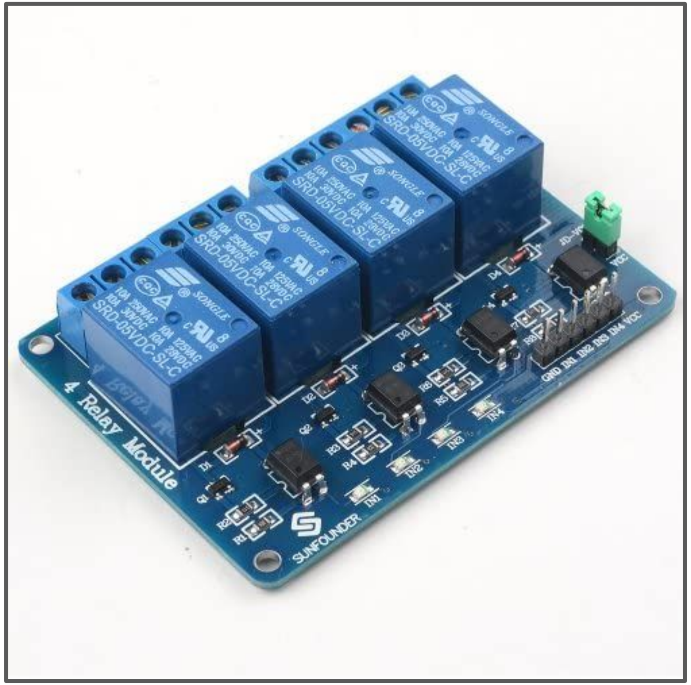
Bonus SMS notifications!



```python
from twilio.rest import Client
import os

account_sid = os.getenv('TWILIO_SID')
auth_token = os.getenv('TWILIO_AUTH_TOKEN')
messaging_service_sid = os.getenv('TWILIO_MSG_SVC_SID')
phone_number = os.getenv('PTN')
client = Client(account_sid, auth_token)

def alert(value, location):
  message = client.messages.create(
    messaging_service_sid=messaging_service_sid,
    body=f'Hello, the current AQI is {value} at {location}.',
    to=phone_number
  )
  return message
```

What else can you do with this data?

- Send a signal to a pico w web server and trigger an electric relay to activate a fan, air purifier, window opener, or HVAC system.
- Share outdoor locations with crowdsourced AQI maps, such as PurpleAir.
- Send notifications to Alexa to alert rooms of high AQI values
- Email notifications
- Create a heat map of a building of changing AQI values



Learn more about this project

Github repository:

- Pico W code
- Consumer services code
- API code
- Instructions on assembling your own unit
- .STL files for printing the box at home
- Data sources of statistics

https://github.com/redis-developer/redis-aqi-monitor.git

Learn more about Redis

Redis:
https://redis.com

Redis University:
https://university.redis.com

Youtube:
https://youtube.com/redis

Discord
https://discord.gg/redis

What's That Smell?

Detecting Air Quality with Python, Raspberry Pi, and Redis

Justin Castilla

Senior Developer Advocate @ Redis

justin@redis.com

(AP Photo/ Eric Risberg)