sumato_systems

AWS CLOUD DATA COLLECTING WITH

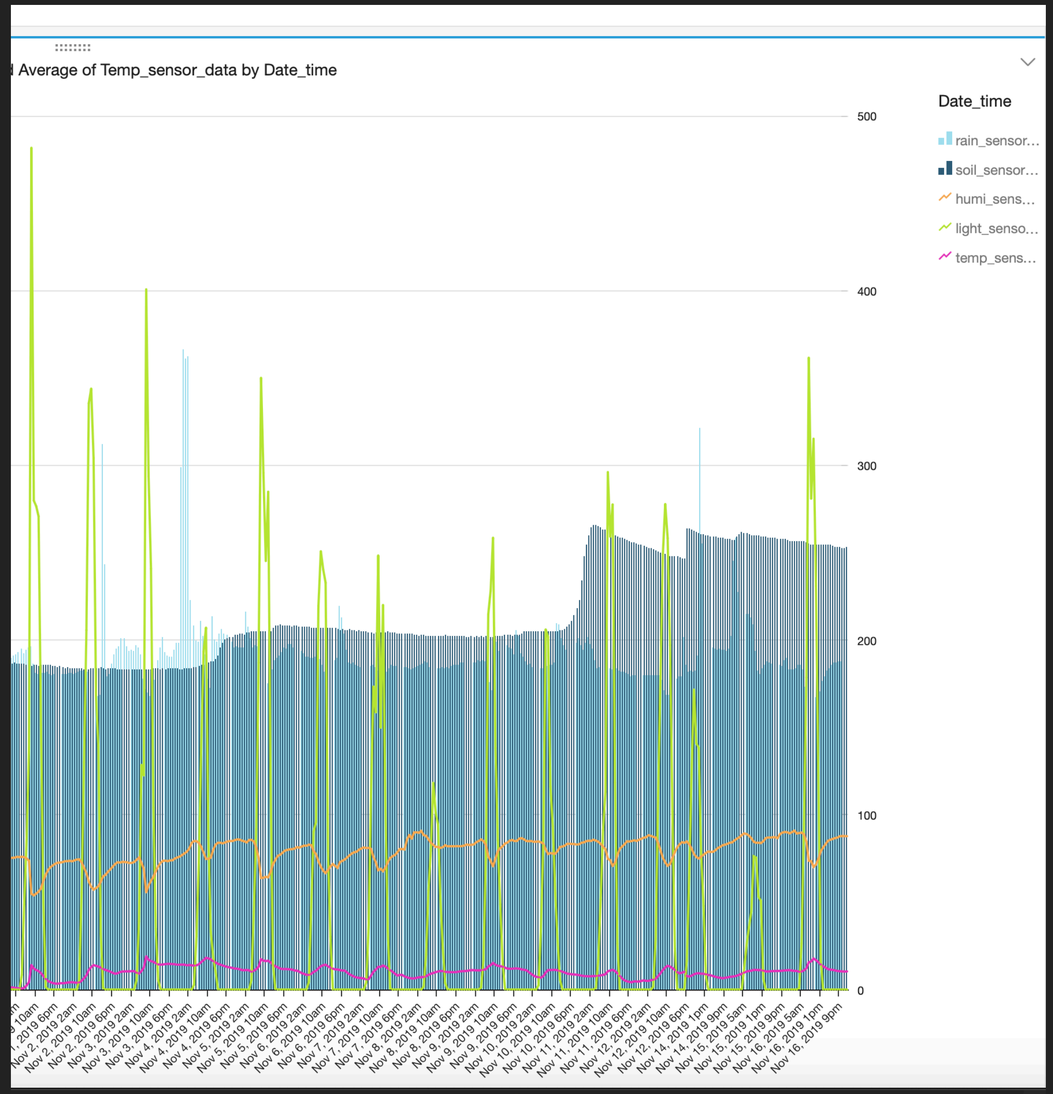
MICROPYTHON

MICHAŁ SUŁKIEWICZ

- sumatosystems.com
- IT consulting, mobile development, software engineering
- IoT, developer (iOS, Python), cloud services
- https://github.com/michalisul/AWSLogger

TODAY'S AGENDA

- Why microcontrollers (MCUs) over Linux boards
- Overview of the data collecting project
- Introduction to MicroPython language
- ESP32 hardware setup
- MicroPython application development
- Setting up AWS IoT Core shadow and Connections



OVERVIEW OF THE PROJECT

- Connecting sensors to ESP32 Feather board (analog sensors, touch sensor and I2C temperature)
- MicroPython firmware installation and initial setup
- Developing MicroPython code
- Configuring AWS IoT Core thing shadow service
- Thing Shadow service workflow
- Implementing thing shadow service communication in the code
- Configuring AWS DynamoDB connection
- Sending data do IoT Analytics service
- Visualising data in QuickSight service*

WHY MCU BOARDS OVER LINUX BOARDS

- No need to support whole OS (Linux)
- Built in ADC inputs
- Almost instant on
- Deep Sleep options
- Built in memory
- Smaller size

INTRO TO MICROPYTHON FOR MCUS

- Compact Python 3.4 interpreter written in C for RTOS-based filesystem
- Loads to the microcontroller (MCU) as a firmware and enables REPL (read-eval-print-loop) - a Python prompt
- REPL has history, tab completion, auto-indent and paste mode
- uPython enables MCU specific libraries to interact with GPIOs, filesystem, hardware modules and interrupts and timer handling
- Package installation via upip
- Cross-compiler available
- More on: http://docs.micropython.org/en/latest/

```
import os

# list root directory
print(os.listdir('/'))

# print current directory
print(os.getcwd())

# open and read a file from the SD card
with open('/sd/readme.txt') as f:
    print(f.read())
```

MICROPYTHON FOR MCUS PROS AND CONS

- Pros:
  - Modern, well established, high level language we all know
  - Easy ports from Raspberry Pi
  - Native JSON format handling
  - Easy file handling
  - Fast development with many ready libraries and drivers
  - Easy to debug and catch errors
- Cons:
  - Still in beta with no support from a big company
  - Slower than C/C++, hardly realtime response
  - Not officially supported by solution providers
  - Hard to secure in commercial/mass production environment*
- http://micropython.org/unicorn/

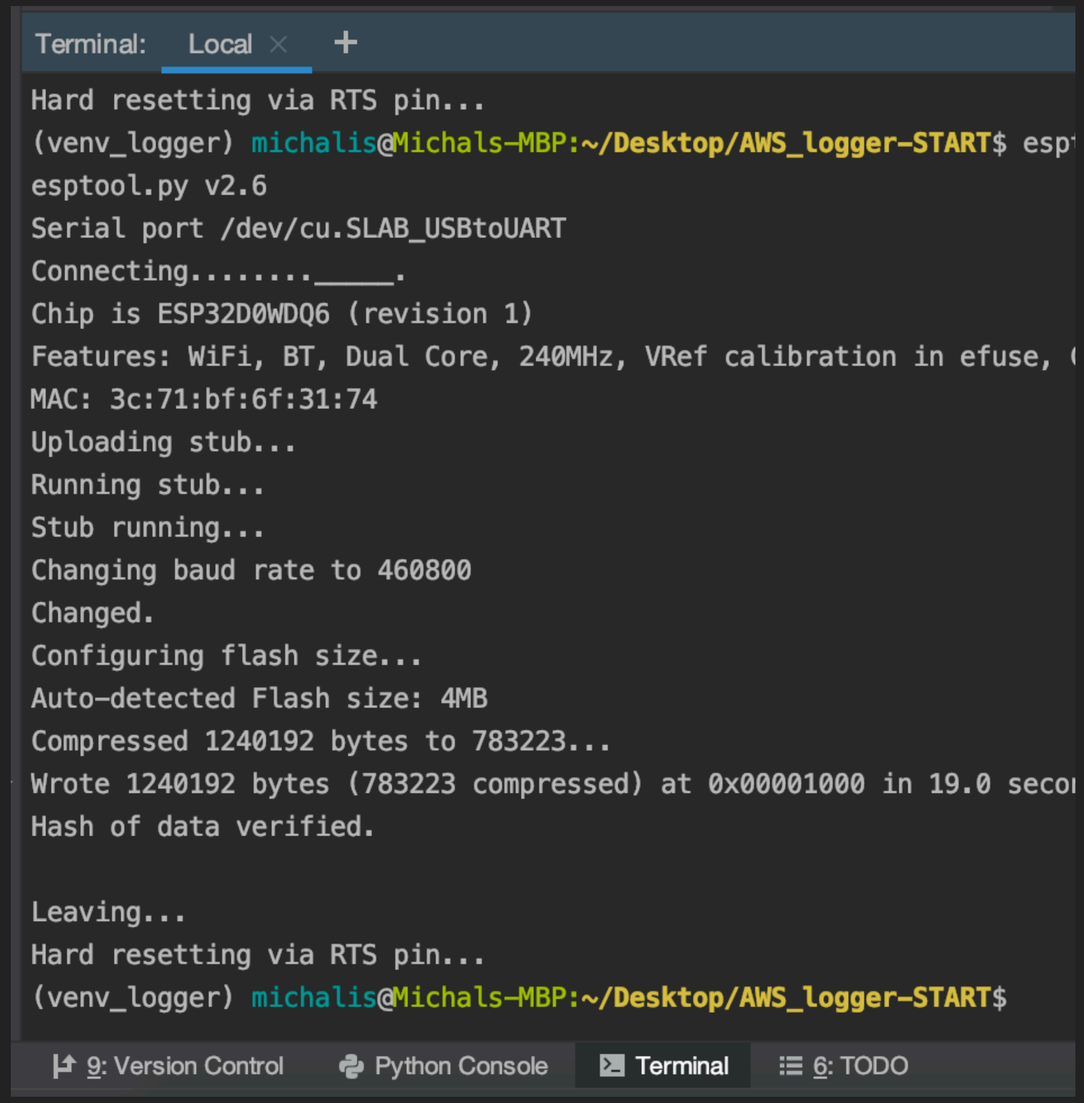
```
def connect(self, clean_session=True):
    self.sock = socket.socket()
    addr = socket.getaddrinfo(self.server, self.port)[0][-1]
    self.sock.connect(addr)
    if self.ssl:
        import ussl
        self.sock = ussl.wrap_socket(self.sock, **self.ssl_params)
    premsg = bytearray(b"\x10\0\0\0\0\0")
    msg = bytearray(b"\x04MQTT\x04\x02\0\0")

    sz = 10 + 2 + len(self.client_id)
    msg[6] = clean_session << 1
    if self.user is not None:
        sz += 2 + len(self.user) + 2 + len(self.pswd)
        msg[6] |= 0xC0
    if self.keepalive:
        assert self.keepalive < 65536
        msg[7] |= self.keepalive >> 8
        msg[8] |= self.keepalive & 0x00FF
    if self.lw_topic:
        sz += 2 + len(self.lw_topic) + 2 + len(self.lw_msg)
        msg[6] |= 0x4 | (self.lw_qos & 0x1) << 3 | (self.lw_qos & 0x2
        msg[6] |= self.lw_retain << 5

    i = 1
    while sz > 0x7f:
        premsg[i] = (sz & 0x7f) | 0x80
        sz >>= 7
```

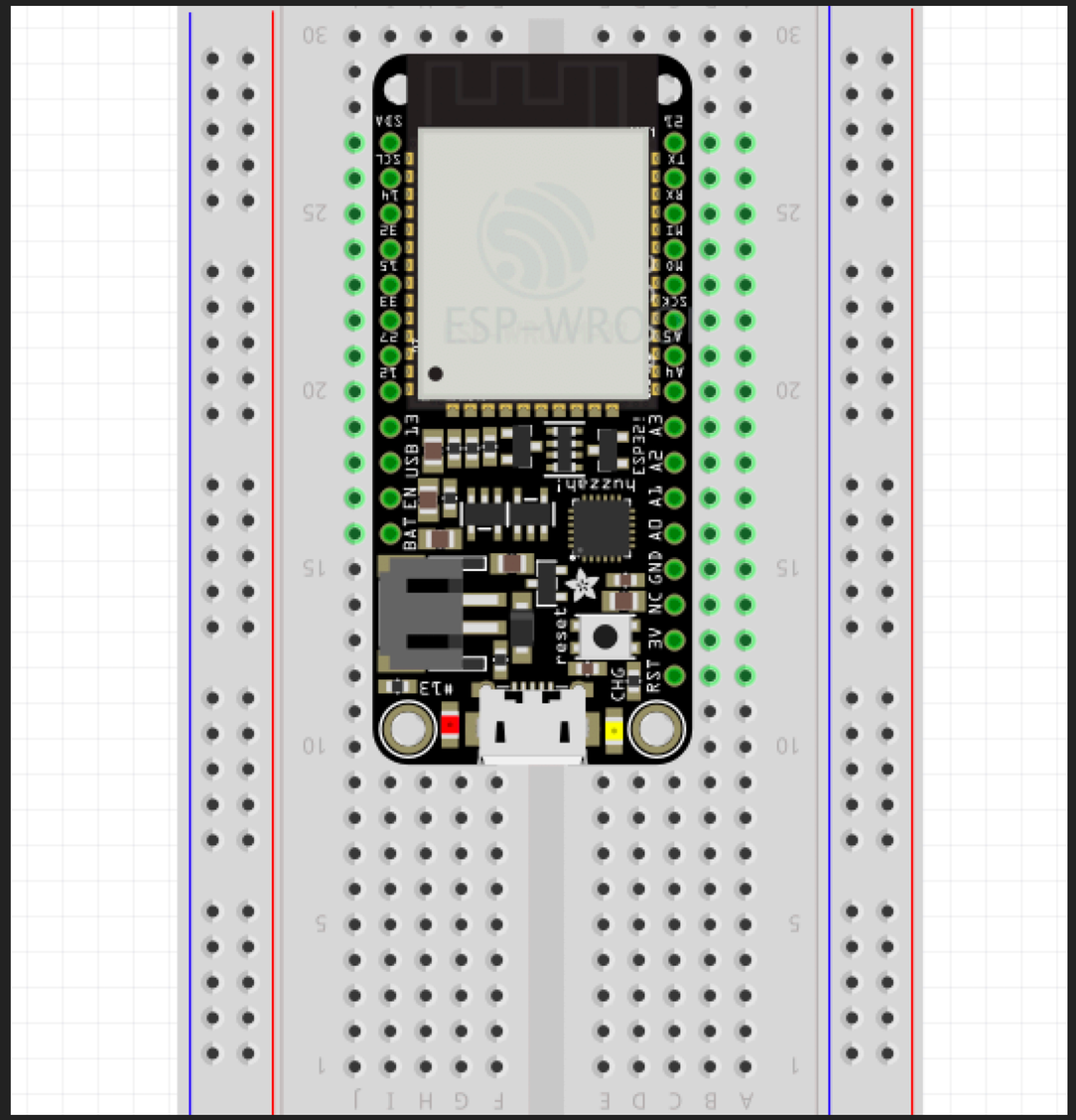
ESP32 ADAFRUIT FEATHER BOARD SETUP 1

- PyCharm IDE setup:
  - Preferences > Plugins > MicroPython Install
  - New Project with Python 3.5+ venv
  - Preferences > Languages & Frameworks > MicroPython > Enable MicroPython support > ESP8266
  - New Python File > main.py > Install dependencies
  - Prepare app structure (.gitIgnore, SECRET/), src/)
- MicroPython firmware installation:
  - pip3 install esptool
  - Download latest uPython firmware
  - Connect the ESP32 board
  - esptool.py –chip esp32 -p /dev/??? erase_flash
  - esptool.py –chip esp32 -p /dev/??? –baud 480800 write_flash -z 0x1000 ~/Downloads/???.bin



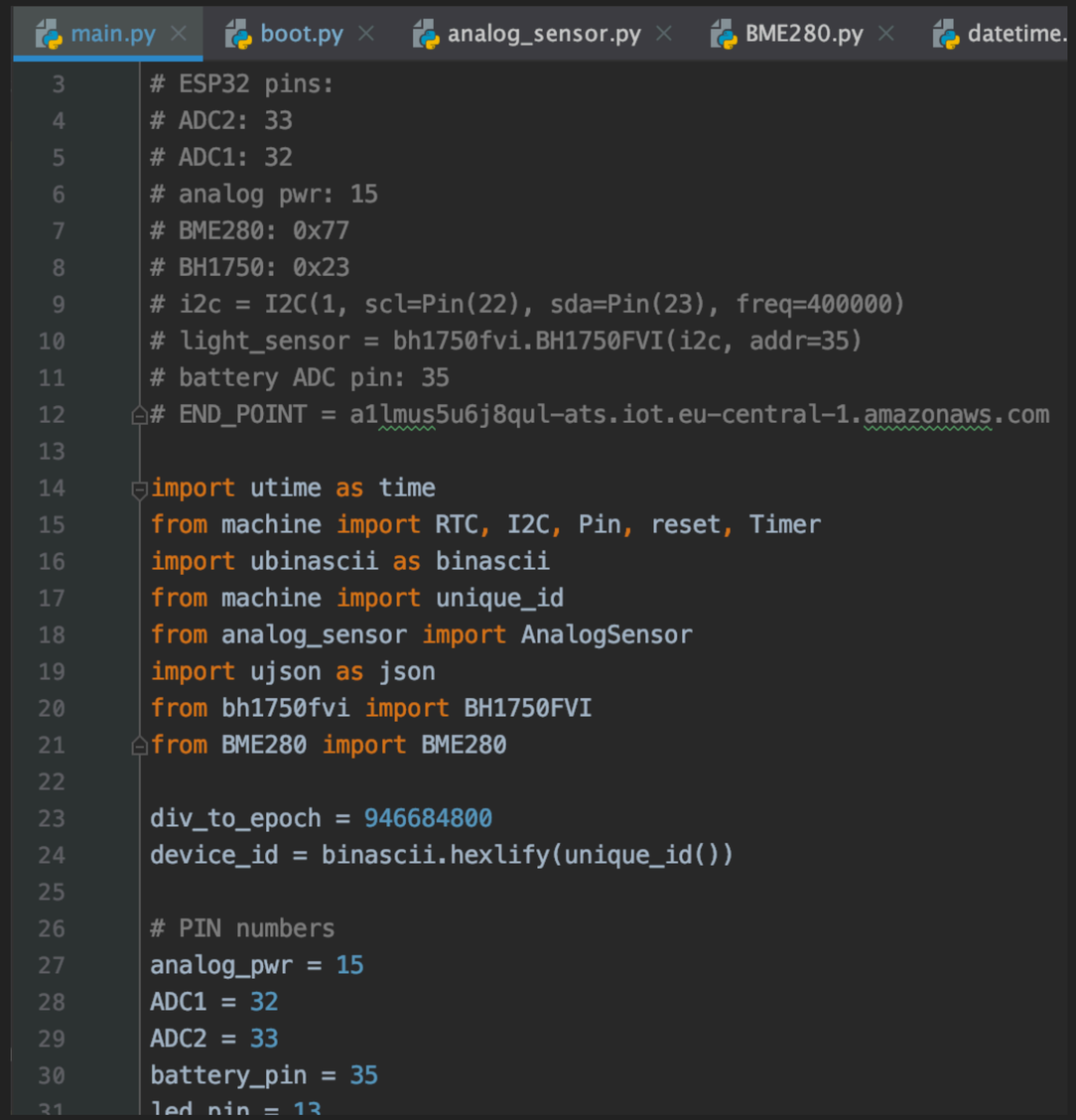
ESP32 ADAFRUIT FEATHER BOARD SETUP 2

- REPL test
  - ampy -p /dev/tty.SLAB_USBtoUART ls
  - >>> import os
  - >>> os.uname()
  - (screen /dev/tty.SLAB_USBtoUART 115200)
- Hardware connections:
  - ADC pin: 32
  - Battery pin: 35
  - Touch pin: 14
  - Built in LED: 13



MICROPYTHON APP STRUCTURE

- boot.py and main.py
- External drivers and libraries:
  - BME280 (temp, pres, humi)*
  - MQTT Client
- App custom module:
  - Analog sensors
  - https://github.com/michalisul/AWSLoggerModules
- Unit testing**



AWS IOT CORE THING SHADOW CONCEPT

- Server representation of the device state
- MQTT or RESTful API communication
- JSON structure
- Desired and reported device states
- Based on predefined topics:
  - $aws/things/***MyDevice***/shadow/***Command***
- https://docs.aws.amazon.com/iot/latest/developerguide/device-shadow-data-flow.html


```
{
  "messageNumber": 1,
  "payload": {
    "state": {
      "reported": {
        "color": "red"
      }
    },
    "metadata": {
      "reported": {
        "color": {
          "timestamp": 1469564492
        }
      }
    },
    "version": 1,
    "timestamp": 1469564571
  },
  "qos": 0,
  "timestamp": 1469564571533,
  "topic": "$aws/things/myLightBulb/shadow/g
}
```

THING SHADOW SERVICE WORKFLOW

- When device (i.e. light bulb) comes online sends message to topic:
  - $aws/things/myLightBulb/shadow/update
- If the update is accepted, Shadow service responds to topic:
  - $aws/things/myLightBulb/shadow/update/accepted
- If not, Shadow service responds to topic:
  - $aws/things/myLightBulb/shadow/update/rejected
- If user changes thing state to “green” in application, or dashboard, changing the bulb colour, the app publishes message with *“desired”* key to server to topic:
  - $aws/things/myLightBulb/shadow/update
- Shadow state is:
  - { “desired”: { “color”: “green” }, “reported”: { “color”: “red”}}
- Device gets notification on desired state to topic:
  - $aws/things/myLightBulb/shadow/update/delta
- If successfully changes its state sends new reported state to:
  - $aws/things/myLightBulb/shadow/update

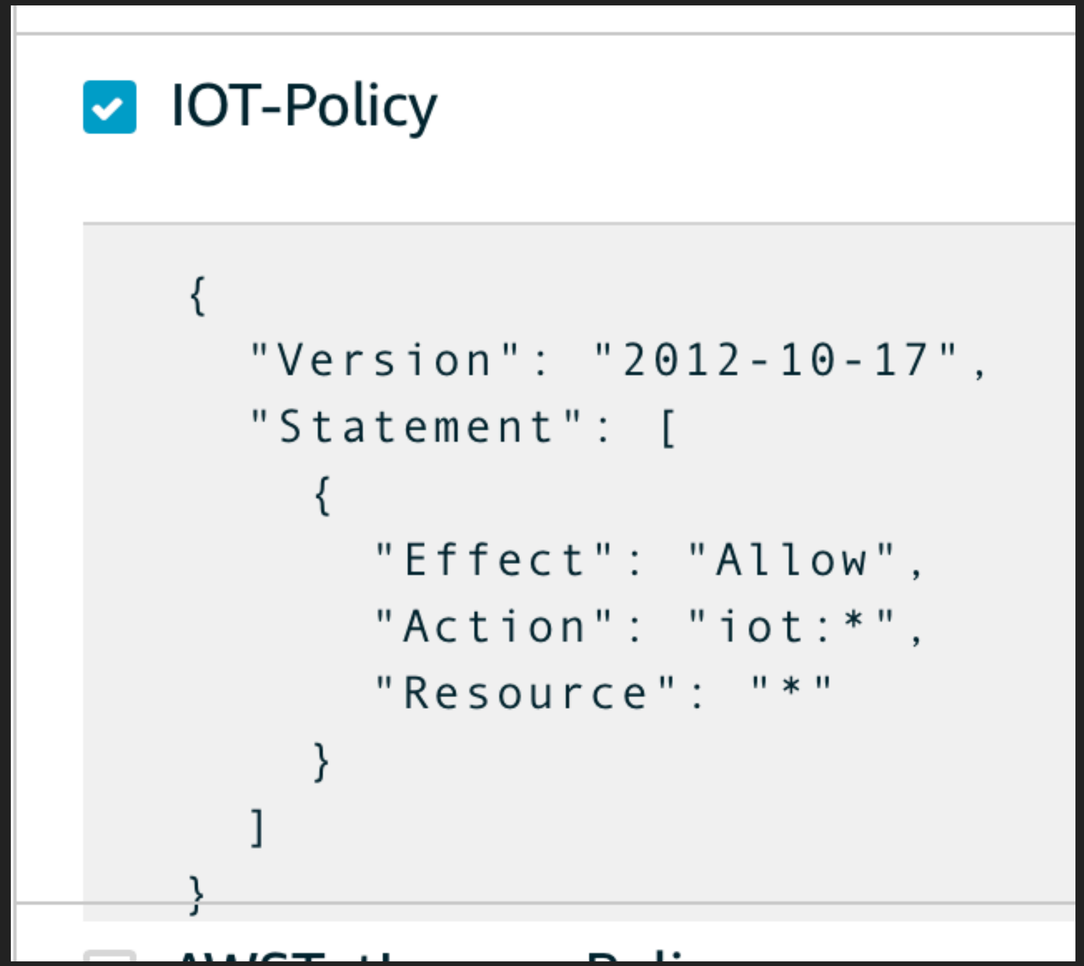
`$aws/things/myLightBulb/shadow/update` topic:

```
{
    "state": {
        "reported": {
            "color": "red"
        }
    }
}
```

```
{
    "state": {
        "desired": {
            "color": "green"
        }
    }
}
```

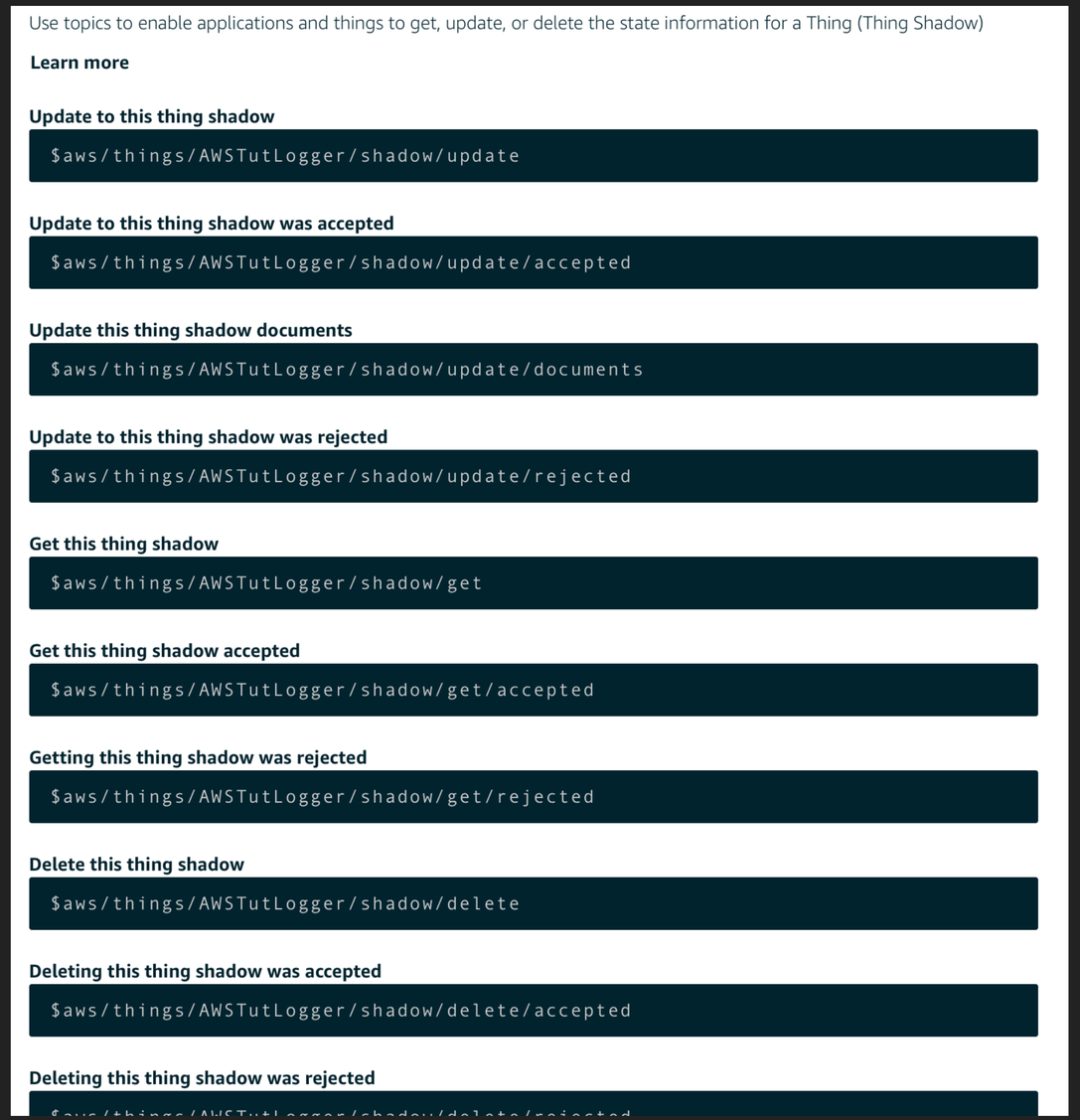
SETTING UP AWS IOT CORE SHADOW

- Choose IoT Core from your AWS Management Console
- Secure > Policies > *Create*
- Click Manage > Things > *Create/Register a thing*
- Click *Create a single thing*
- Give the device (thing) a name and click *Next*
- Click *Create certificate*
- Download all keys and certificate
- Click *Activate*
- Click *Attach a policy*
- Click *Register a thing*
- Activate Certificate in Secure > Certificates



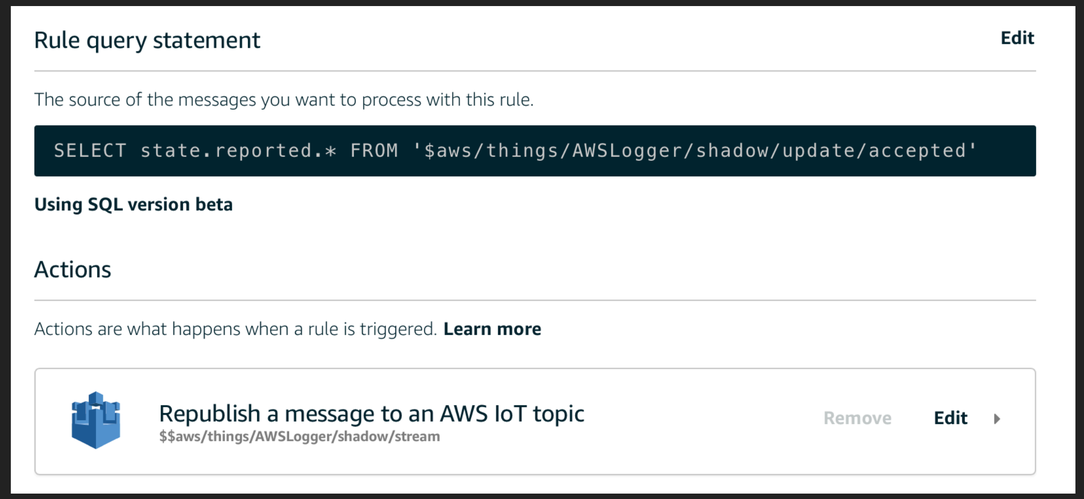
IMPLEMENTING SHADOW COMMUNICATION

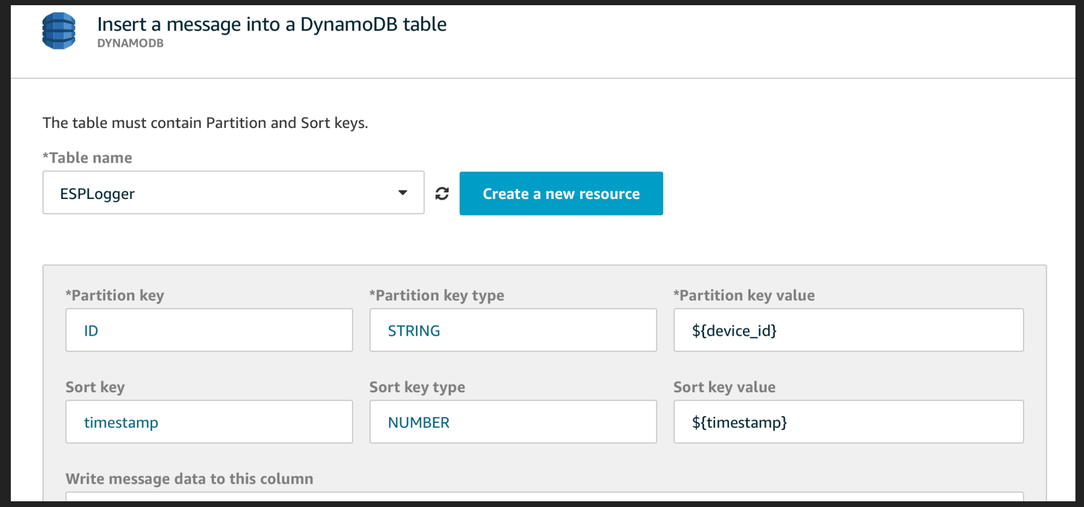
- Copy keys into the board
- Find the endpoint (IoT Core main console > Settings)
- Edit MCU code to securely connect to AWS IoT server
- Update MQTT topics
- Check if thing shadow gets updated



CONFIGURING DYNAMO DB CONNECTION

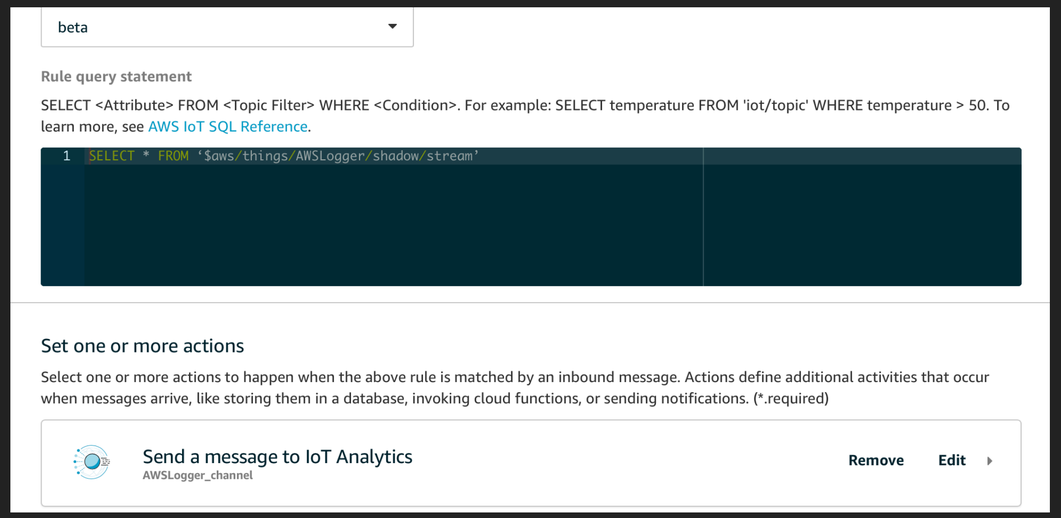
- Setting a Rule to trim data from shadow update
  - Act > Rules > *Create*
  - SELECT state.reported.* FROM '$aws/things/AWSLogger/shadow/update/accepted'
  - Republish a message to **$**$aws/things/AWSLogger/shadow/stream
  - Create Role
  - Click *Add Action*
  - Click *Create Rule*
- Setting a rule to add trimmed message to DynamoDB
  - Act > Rules > *Create*
  - SELECT * FROM '$aws/things/AWSLogger/shadow/stream'
  - Insert Message into DynamoDB table
  - Click *Create New Resource* and finish creating new DDB table
  - Back in the rule creator and fill Partition key and Sort key with corresponding keys from the message (i.e. $(device_id) and $(timestamp))
  - Click *Add Action*
  - Click *Create Rule*





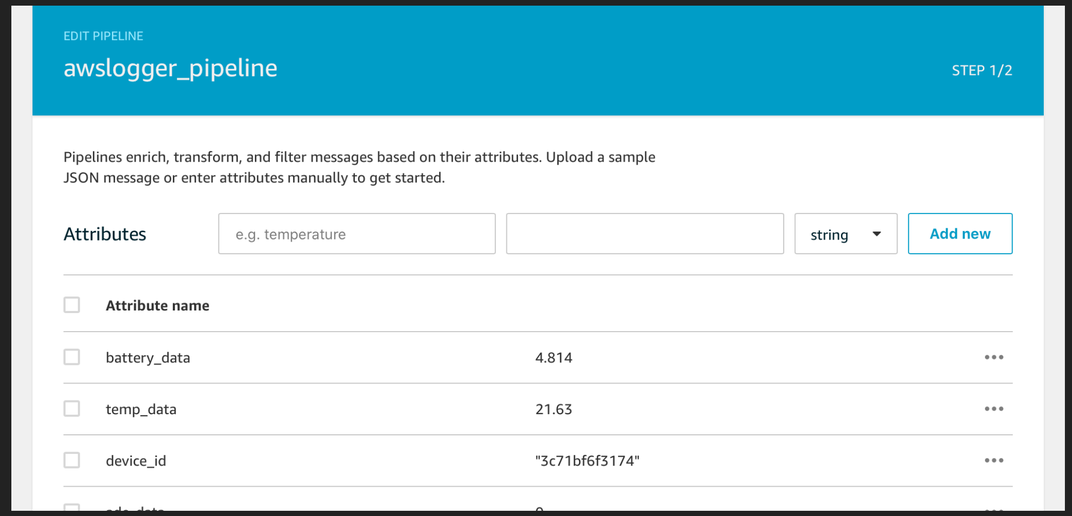
CONFIGURING IOT ANALYTICS DATA STREAM

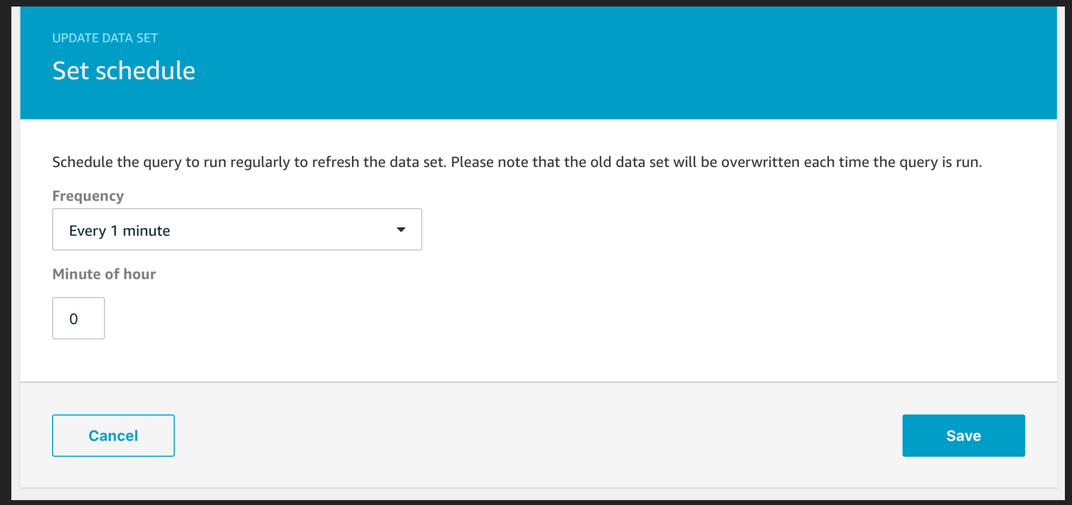
- Setting a Rule to send data to IoT
  - Act > Rules > *Create*
  - SELECT * FROM '$aws/things/AWSLogger/shadow/stream'
  - Send a message to IoT Analytics
    - Select option *Quick create IoT Analytics resources*
  - Click *Add Action*
  - Click *Create Rule*



CONFIGURING IOT ANALYTICS SERVICES

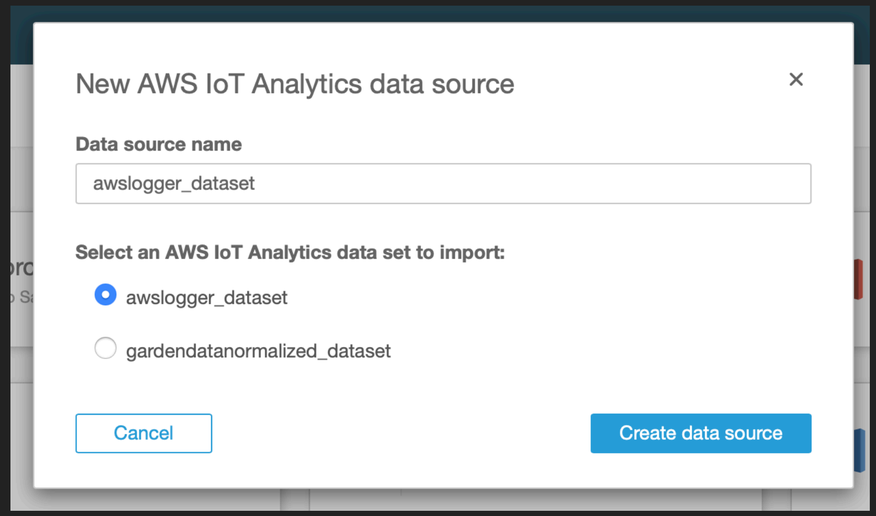
- AWS Management Console > IoT Analytics
  - Pipelines > awslogger_pipeline > Activities
  - Data sets > awslogger_dataset > Schedule
    - i.e . Every 1 minute > *Save*
  - Data sets > awslogger_dataset > Data set content retention settings

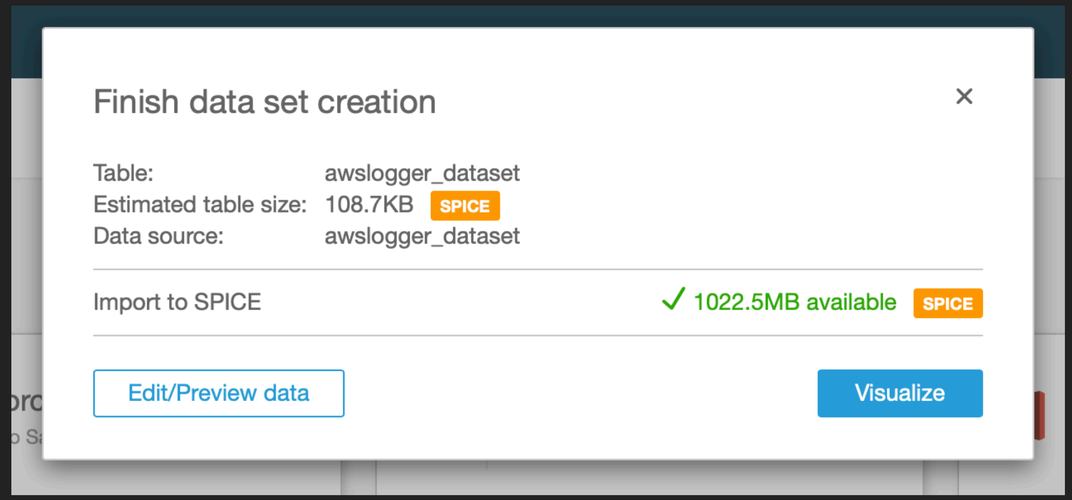




CONFIGURING QUICKSIGHT DATA VISUALISATION

- AWS Management Console > Analytics > QuickSight
  - *New analysis* > *New Data set* > AWS IoT Analytics
  - New AWS IoT Analytics data source > aws_logger_dataset > *Create data source*
  - Finish data set creation > *Visualize*





THANK YOU FOR YOUR ATTENTION.
ANY QUESTIONS ?

MICHALSULKIEWICZ@SUMATOSYSTEMS.COM

Michał Sułkiewicz