

# **NATIONAL UNIVERSITY OF SINGAPORE**



## **FACULTY OF ENGINEERING DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING**

**EE5111 Special Topics in Industrial Control and Automation  
Systems**

**Simulation using Jet Engine Data to implement an Internet of  
Things (IoT) Setup using Amazon Web Services (AWS)**

**By**

**Name: PENG JINGMING (A0195042J)**

**Submission Date: 21 Sept 2019**

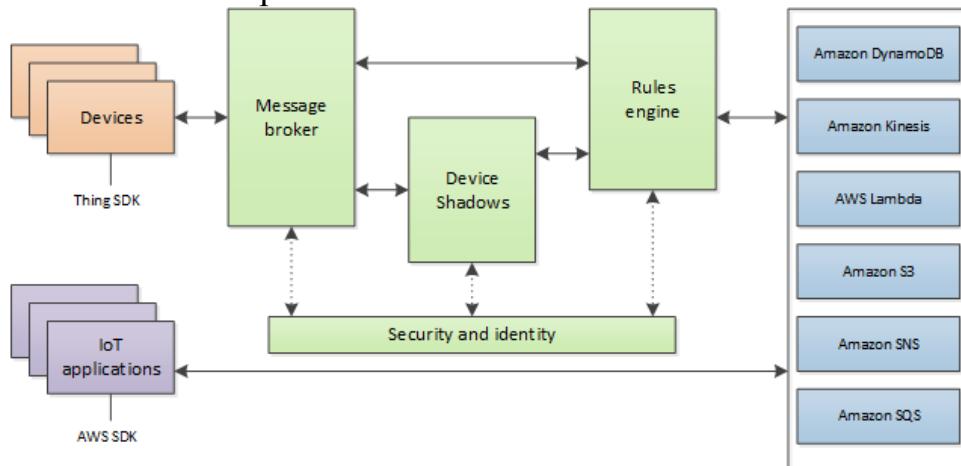
## **Outline**

1. Project Description
2. AWS IoT Concept
3. AWS IoT Realization
4. Visualization Concept
5. Visualization Realization
6. Task1 – Publish Engine Data into DynamoDB through laptop and do visualization
7. Task 2 – Publish rainfall data through raspberry pi to DynamoDB and do visualization
8. Task3 – Self-monitor raspberry pi's cpu temperature and establish email alert on abnormal high temperature (temperature > 47.00DegC)
9. Discussion
10. Appendixes

## 1. Project Description

- 1.1) To implement a simple IoT pipeline with AWS Cloud platform and visualize the data.
- 1.2) To write a guideline on how to set up the pipeline with pictures showing matrix No. inside.
- 1.3) 3 IoT tasks are discussed and covered within this report:
  - Based on a laptop - the endpoint device platform, to simulate 2 small IoT setups that record and push data from 2 jet engines.
  - To expand the idea, based on a Raspberry Pi 3 model B board - a different endpoint device platform, to simulate 1 IoT setup that pushes environmental data downloaded from Data.gov into AWS cloud.
  - Relying on the same Raspberry Pi platform, capture its self-monitoring real-time data (temperature) and push it up to IoT cloud, and send pre-defined email alert on any abnormality (overtemperature).

## 2. AWS IoT Concept



Within a general architecture of AWS IoT (shown above), it enables endpoint devices to connect to the AWS Cloud and lets applications in the cloud interact with these devices.

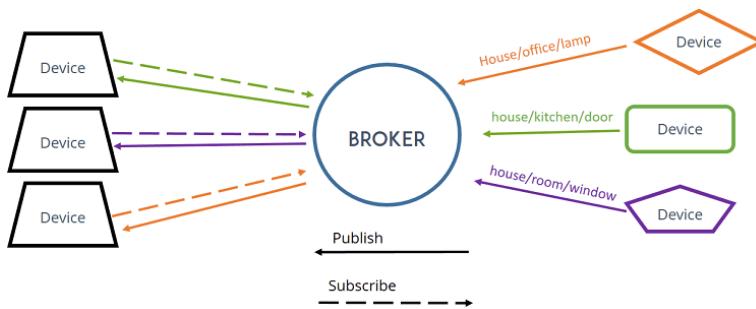
Main components:

2.1) Devices with policies - All devices that connect to AWS IoT need to have an entry in the registry, where information about a device and the certificates are stored. Within the registry, these devices are represented by “Things”, or the so-called “device simulator”. For example, the “Thing” can be your laptop or Raspberry Pi Board.

2.2) The AWS IoT policies, attached to the “Things”, are used to authorize devices to perform AWS IoT operations, such as subscribing or publishing to MQTT topics.



2.3) Message broker - It is responsible for sending all messages published on an MQTT topic to all clients subscribed to that topic.



Data exchange in between the device and cloud is through MQTT protocol. In a MQTT message broker, it uses topic (an UTF-8 string) to filter messages for each connected client.

Devices report their state by publishing messages, in JSON format, on MQTT topics.

Each MQTT topic can trace the device when it gets updated.

2.4) Device shadow – It's a JSON document that is used to store and retrieve current state information for a device. You can use the shadow to get and set the state of a device over

MQTT. Example shown below:

The screenshot shows the AWS IoT Device Shadow interface. On the left, a sidebar menu lists: Details, Security, Thing Groups, Billing Groups, **Shadow** (which is selected), Interact, Activity, Jobs, Violations, and Defender metrics. The main content area has two tabs: 'Shadow ARN' and 'Shadow Document'. The 'Shadow ARN' tab displays the ARN: arn:aws:iot:ap-southeast-1:121070346726:thing/A0195042J\_EngineFD003. The 'Shadow Document' tab shows a JSON document with the following content:

```
{  
  "reported": {  
    "id": "FD003_18",  
    "timestamp": "2019-09-09 04:00:41.049797",  
    "Metric_Number": "A0195042J",  
    "os1": "-0.0018",  
    "os2": "-0.0004",  
    "os3": "100.0",  
    "cycle": "308",  
    "os4": "0.0001",  
    "os5": "0.0001",  
    "os6": "0.0001",  
    "os7": "0.0001",  
    "os8": "0.0001",  
    "os9": "0.0001",  
    "os10": "0.0001",  
    "os11": "0.0001",  
    "os12": "0.0001",  
    "os13": "0.0001",  
    "os14": "0.0001",  
    "os15": "0.0001",  
    "os16": "0.0001",  
    "os17": "0.0001",  
    "os18": "0.0001",  
    "os19": "0.0001",  
    "os20": "0.0001",  
    "os21": "0.0001",  
    "os22": "0.0001",  
    "os23": "0.0001",  
    "os24": "0.0001",  
    "os25": "0.0001",  
    "os26": "0.0001",  
    "os27": "0.0001",  
    "os28": "0.0001",  
    "os29": "0.0001",  
    "os30": "0.0001",  
    "os31": "0.0001",  
    "os32": "0.0001",  
    "os33": "0.0001",  
    "os34": "0.0001",  
    "os35": "0.0001",  
    "os36": "0.0001",  
    "os37": "0.0001",  
    "os38": "0.0001",  
    "os39": "0.0001",  
    "os40": "0.0001",  
    "os41": "0.0001",  
    "os42": "0.0001",  
    "os43": "0.0001",  
    "os44": "0.0001",  
    "os45": "0.0001",  
    "os46": "0.0001",  
    "os47": "0.0001",  
    "os48": "0.0001",  
    "os49": "0.0001",  
    "os50": "0.0001",  
    "os51": "0.0001",  
    "os52": "0.0001",  
    "os53": "0.0001",  
    "os54": "0.0001",  
    "os55": "0.0001",  
    "os56": "0.0001",  
    "os57": "0.0001",  
    "os58": "0.0001",  
    "os59": "0.0001",  
    "os60": "0.0001",  
    "os61": "0.0001",  
    "os62": "0.0001",  
    "os63": "0.0001",  
    "os64": "0.0001",  
    "os65": "0.0001",  
    "os66": "0.0001",  
    "os67": "0.0001",  
    "os68": "0.0001",  
    "os69": "0.0001",  
    "os70": "0.0001",  
    "os71": "0.0001",  
    "os72": "0.0001",  
    "os73": "0.0001",  
    "os74": "0.0001",  
    "os75": "0.0001",  
    "os76": "0.0001",  
    "os77": "0.0001",  
    "os78": "0.0001",  
    "os79": "0.0001",  
    "os80": "0.0001",  
    "os81": "0.0001",  
    "os82": "0.0001",  
    "os83": "0.0001",  
    "os84": "0.0001",  
    "os85": "0.0001",  
    "os86": "0.0001",  
    "os87": "0.0001",  
    "os88": "0.0001",  
    "os89": "0.0001",  
    "os90": "0.0001",  
    "os91": "0.0001",  
    "os92": "0.0001",  
    "os93": "0.0001",  
    "os94": "0.0001",  
    "os95": "0.0001",  
    "os96": "0.0001",  
    "os97": "0.0001",  
    "os98": "0.0001",  
    "os99": "0.0001",  
    "os100": "0.0001",  
    "os101": "0.0001",  
    "os102": "0.0001",  
    "os103": "0.0001",  
    "os104": "0.0001",  
    "os105": "0.0001",  
    "os106": "0.0001",  
    "os107": "0.0001",  
    "os108": "0.0001",  
    "os109": "0.0001",  
    "os110": "0.0001",  
    "os111": "0.0001",  
    "os112": "0.0001",  
    "os113": "0.0001",  
    "os114": "0.0001",  
    "os115": "0.0001",  
    "os116": "0.0001",  
    "os117": "0.0001",  
    "os118": "0.0001",  
    "os119": "0.0001",  
    "os120": "0.0001",  
    "os121": "0.0001",  
    "os122": "0.0001",  
    "os123": "0.0001",  
    "os124": "0.0001",  
    "os125": "0.0001",  
    "os126": "0.0001",  
    "os127": "0.0001",  
    "os128": "0.0001",  
    "os129": "0.0001",  
    "os130": "0.0001",  
    "os131": "0.0001",  
    "os132": "0.0001",  
    "os133": "0.0001",  
    "os134": "0.0001",  
    "os135": "0.0001",  
    "os136": "0.0001",  
    "os137": "0.0001",  
    "os138": "0.0001",  
    "os139": "0.0001",  
    "os140": "0.0001",  
    "os141": "0.0001",  
    "os142": "0.0001",  
    "os143": "0.0001",  
    "os144": "0.0001",  
    "os145": "0.0001",  
    "os146": "0.0001",  
    "os147": "0.0001",  
    "os148": "0.0001",  
    "os149": "0.0001",  
    "os150": "0.0001",  
    "os151": "0.0001",  
    "os152": "0.0001",  
    "os153": "0.0001",  
    "os154": "0.0001",  
    "os155": "0.0001",  
    "os156": "0.0001",  
    "os157": "0.0001",  
    "os158": "0.0001",  
    "os159": "0.0001",  
    "os160": "0.0001",  
    "os161": "0.0001",  
    "os162": "0.0001",  
    "os163": "0.0001",  
    "os164": "0.0001",  
    "os165": "0.0001",  
    "os166": "0.0001",  
    "os167": "0.0001",  
    "os168": "0.0001",  
    "os169": "0.0001",  
    "os170": "0.0001",  
    "os171": "0.0001",  
    "os172": "0.0001",  
    "os173": "0.0001",  
    "os174": "0.0001",  
    "os175": "0.0001",  
    "os176": "0.0001",  
    "os177": "0.0001",  
    "os178": "0.0001",  
    "os179": "0.0001",  
    "os180": "0.0001",  
    "os181": "0.0001",  
    "os182": "0.0001",  
    "os183": "0.0001",  
    "os184": "0.0001",  
    "os185": "0.0001",  
    "os186": "0.0001",  
    "os187": "0.0001",  
    "os188": "0.0001",  
    "os189": "0.0001",  
    "os190": "0.0001",  
    "os191": "0.0001",  
    "os192": "0.0001",  
    "os193": "0.0001",  
    "os194": "0.0001",  
    "os195": "0.0001",  
    "os196": "0.0001",  
    "os197": "0.0001",  
    "os198": "0.0001",  
    "os199": "0.0001",  
    "os200": "0.0001",  
    "os201": "0.0001",  
    "os202": "0.0001",  
    "os203": "0.0001",  
    "os204": "0.0001",  
    "os205": "0.0001",  
    "os206": "0.0001",  
    "os207": "0.0001",  
    "os208": "0.0001",  
    "os209": "0.0001",  
    "os210": "0.0001",  
    "os211": "0.0001",  
    "os212": "0.0001",  
    "os213": "0.0001",  
    "os214": "0.0001",  
    "os215": "0.0001",  
    "os216": "0.0001",  
    "os217": "0.0001",  
    "os218": "0.0001",  
    "os219": "0.0001",  
    "os220": "0.0001",  
    "os221": "0.0001",  
    "os222": "0.0001",  
    "os223": "0.0001",  
    "os224": "0.0001",  
    "os225": "0.0001",  
    "os226": "0.0001",  
    "os227": "0.0001",  
    "os228": "0.0001",  
    "os229": "0.0001",  
    "os230": "0.0001",  
    "os231": "0.0001",  
    "os232": "0.0001",  
    "os233": "0.0001",  
    "os234": "0.0001",  
    "os235": "0.0001",  
    "os236": "0.0001",  
    "os237": "0.0001",  
    "os238": "0.0001",  
    "os239": "0.0001",  
    "os240": "0.0001",  
    "os241": "0.0001",  
    "os242": "0.0001",  
    "os243": "0.0001",  
    "os244": "0.0001",  
    "os245": "0.0001",  
    "os246": "0.0001",  
    "os247": "0.0001",  
    "os248": "0.0001",  
    "os249": "0.0001",  
    "os250": "0.0001",  
    "os251": "0.0001",  
    "os252": "0.0001",  
    "os253": "0.0001",  
    "os254": "0.0001",  
    "os255": "0.0001",  
    "os256": "0.0001",  
    "os257": "0.0001",  
    "os258": "0.0001",  
    "os259": "0.0001",  
    "os260": "0.0001",  
    "os261": "0.0001",  
    "os262": "0.0001",  
    "os263": "0.0001",  
    "os264": "0.0001",  
    "os265": "0.0001",  
    "os266": "0.0001",  
    "os267": "0.0001",  
    "os268": "0.0001",  
    "os269": "0.0001",  
    "os270": "0.0001",  
    "os271": "0.0001",  
    "os272": "0.0001",  
    "os273": "0.0001",  
    "os274": "0.0001",  
    "os275": "0.0001",  
    "os276": "0.0001",  
    "os277": "0.0001",  
    "os278": "0.0001",  
    "os279": "0.0001",  
    "os280": "0.0001",  
    "os281": "0.0001",  
    "os282": "0.0001",  
    "os283": "0.0001",  
    "os284": "0.0001",  
    "os285": "0.0001",  
    "os286": "0.0001",  
    "os287": "0.0001",  
    "os288": "0.0001",  
    "os289": "0.0001",  
    "os290": "0.0001",  
    "os291": "0.0001",  
    "os292": "0.0001",  
    "os293": "0.0001",  
    "os294": "0.0001",  
    "os295": "0.0001",  
    "os296": "0.0001",  
    "os297": "0.0001",  
    "os298": "0.0001",  
    "os299": "0.0001",  
    "os300": "0.0001",  
    "os301": "0.0001",  
    "os302": "0.0001",  
    "os303": "0.0001",  
    "os304": "0.0001",  
    "os305": "0.0001",  
    "os306": "0.0001",  
    "os307": "0.0001",  
    "os308": "0.0001",  
    "os309": "0.0001",  
    "os310": "0.0001",  
    "os311": "0.0001",  
    "os312": "0.0001",  
    "os313": "0.0001",  
    "os314": "0.0001",  
    "os315": "0.0001",  
    "os316": "0.0001",  
    "os317": "0.0001",  
    "os318": "0.0001",  
    "os319": "0.0001",  
    "os320": "0.0001",  
    "os321": "0.0001",  
    "os322": "0.0001",  
    "os323": "0.0001",  
    "os324": "0.0001",  
    "os325": "0.0001",  
    "os326": "0.0001",  
    "os327": "0.0001",  
    "os328": "0.0001",  
    "os329": "0.0001",  
    "os330": "0.0001",  
    "os331": "0.0001",  
    "os332": "0.0001",  
    "os333": "0.0001",  
    "os334": "0.0001",  
    "os335": "0.0001",  
    "os336": "0.0001",  
    "os337": "0.0001",  
    "os338": "0.0001",  
    "os339": "0.0001",  
    "os340": "0.0001",  
    "os341": "0.0001",  
    "os342": "0.0001",  
    "os343": "0.0001",  
    "os344": "0.0001",  
    "os345": "0.0001",  
    "os346": "0.0001",  
    "os347": "0.0001",  
    "os348": "0.0001",  
    "os349": "0.0001",  
    "os350": "0.0001",  
    "os351": "0.0001",  
    "os352": "0.0001",  
    "os353": "0.0001",  
    "os354": "0.0001",  
    "os355": "0.0001",  
    "os356": "0.0001",  
    "os357": "0.0001",  
    "os358": "0.0001",  
    "os359": "0.0001",  
    "os360": "0.0001",  
    "os361": "0.0001",  
    "os362": "0.0001",  
    "os363": "0.0001",  
    "os364": "0.0001",  
    "os365": "0.0001",  
    "os366": "0.0001",  
    "os367": "0.0001",  
    "os368": "0.0001",  
    "os369": "0.0001",  
    "os370": "0.0001",  
    "os371": "0.0001",  
    "os372": "0.0001",  
    "os373": "0.0001",  
    "os374": "0.0001",  
    "os375": "0.0001",  
    "os376": "0.0001",  
    "os377": "0.0001",  
    "os378": "0.0001",  
    "os379": "0.0001",  
    "os380": "0.0001",  
    "os381": "0.0001",  
    "os382": "0.0001",  
    "os383": "0.0001",  
    "os384": "0.0001",  
    "os385": "0.0001",  
    "os386": "0.0001",  
    "os387": "0.0001",  
    "os388": "0.0001",  
    "os389": "0.0001",  
    "os390": "0.0001",  
    "os391": "0.0001",  
    "os392": "0.0001",  
    "os393": "0.0001",  
    "os394": "0.0001",  
    "os395": "0.0001",  
    "os396": "0.0001",  
    "os397": "0.0001",  
    "os398": "0.0001",  
    "os399": "0.0001",  
    "os400": "0.0001",  
    "os401": "0.0001",  
    "os402": "0.0001",  
    "os403": "0.0001",  
    "os404": "0.0001",  
    "os405": "0.0001",  
    "os406": "0.0001",  
    "os407": "0.0001",  
    "os408": "0.0001",  
    "os409": "0.0001",  
    "os410": "0.0001",  
    "os411": "0.0001",  
    "os412": "0.0001",  
    "os413": "0.0001",  
    "os414": "0.0001",  
    "os415": "0.0001",  
    "os416": "0.0001",  
    "os417": "0.0001",  
    "os418": "0.0001",  
    "os419": "0.0001",  
    "os420": "0.0001",  
    "os421": "0.0001",  
    "os422": "0.0001",  
    "os423": "0.0001",  
    "os424": "0.0001",  
    "os425": "0.0001",  
    "os426": "0.0001",  
    "os427": "0.0001",  
    "os428": "0.0001",  
    "os429": "0.0001",  
    "os430": "0.0001",  
    "os431": "0.0001",  
    "os432": "0.0001",  
    "os433": "0.0001",  
    "os434": "0.0001",  
    "os435": "0.0001",  
    "os436": "0.0001",  
    "os437": "0.0001",  
    "os438": "0.0001",  
    "os439": "0.0001",  
    "os440": "0.0001",  
    "os441": "0.0001",  
    "os442": "0.0001",  
    "os443": "0.0001",  
    "os444": "0.0001",  
    "os445": "0.0001",  
    "os446": "0.0001",  
    "os447": "0.0001",  
    "os448": "0.0001",  
    "os449": "0.0001",  
    "os450": "0.0001",  
    "os451": "0.0001",  
    "os452": "0.0001",  
    "os453": "0.0001",  
    "os454": "0.0001",  
    "os455": "0.0001",  
    "os456": "0.0001",  
    "os457": "0.0001",  
    "os458": "0.0001",  
    "os459": "0.0001",  
    "os460": "0.0001",  
    "os461": "0.0001",  
    "os462": "0.0001",  
    "os463": "0.0001",  
    "os464": "0.0001",  
    "os465": "0.0001",  
    "os466": "0.0001",  
    "os467": "0.0001",  
    "os468": "0.0001",  
    "os469": "0.0001",  
    "os470": "0.0001",  
    "os471": "0.0001",  
    "os472": "0.0001",  
    "os473": "0.0001",  
    "os474": "0.0001",  
    "os475": "0.0001",  
    "os476": "0.0001",  
    "os477": "0.0001",  
    "os478": "0.0001",  
    "os479": "0.0001",  
    "os480": "0.0001",  
    "os481": "0.0001",  
    "os482": "0.0001",  
    "os483": "0.0001",  
    "os484": "0.0001",  
    "os485": "0.0001",  
    "os486": "0.0001",  
    "os487": "0.0001",  
    "os488": "0.0001",  
    "os489": "0.0001",  
    "os490": "0.0001",  
    "os491": "0.0001",  
    "os492": "0.0001",  
    "os493": "0.0001",  
    "os494": "0.0001",  
    "os495": "0.0001",  
    "os496": "0.0001",  
    "os497": "0.0001",  
    "os498": "0.0001",  
    "os499": "0.0001",  
    "os500": "0.0001",  
    "os501": "0.0001",  
    "os502": "0.0001",  
    "os503": "0.0001",  
    "os504": "0.0001",  
    "os505": "0.0001",  
    "os506": "0.0001",  
    "os507": "0.0001",  
    "os508": "0.0001",  
    "os509": "0.0001",  
    "os510": "0.0001",  
    "os511": "0.0001",  
    "os512": "0.0001",  
    "os513": "0.0001",  
    "os514": "0.0001",  
    "os515": "0.0001",  
    "os516": "0.0001",  
    "os517": "0.0001",  
    "os518": "0.0001",  
    "os519": "0.0001",  
    "os520": "0.0001",  
    "os521": "0.0001",  
    "os522": "0.0001",  
    "os523": "0.0001",  
    "os524": "0.0001",  
    "os525": "0.0001",  
    "os526": "0.0001",  
    "os527": "0.0001",  
    "os528": "0.0001",  
    "os529": "0.0001",  
    "os530": "0.0001",  
    "os531": "0.0001",  
    "os532": "0.0001",  
    "os533": "0.0001",  
    "os534": "0.0001",  
    "os535": "0.0001",  
    "os536": "0.0001",  
    "os537": "0.0001",  
    "os538": "0.0001",  
    "os539": "0.0001",  
    "os540": "0.0001",  
    "os541": "0.0001",  
    "os542": "0.0001",  
    "os543": "0.0001",  
    "os544": "0.0001",  
    "os545": "0.0001",  
    "os546": "0.0001",  
    "os547": "0.0001",  
    "os548": "0.0001",  
    "os549": "0.0001",  
    "os550": "0.0001",  
    "os551": "0.0001",  
    "os552": "0.0001",  
    "os553": "0.0001",  
    "os554": "0.0001",  
    "os555": "0.0001",  
    "os556": "0.0001",  
    "os557": "0.0001",  
    "os558": "0.0001",  
    "os559": "0.0001",  
    "os560": "0.0001",  
    "os561": "0.0001",  
    "os562": "0.0001",  
    "os563": "0.0001",  
    "os564": "0.0001",  
    "os565": "0.0001",  
    "os566": "0.0001",  
    "os567": "0.0001",  
    "os568": "0.0001",  
    "os569": "0.0001",  
    "os570": "0.0001",  
    "os571": "0.0001",  
    "os572": "0.0001",  
    "os573": "0.0001",  
    "os574": "0.0001",  
    "os575": "0.0001",  
    "os576": "0.0001",  
    "os577": "0.0001",  
    "os578": "0.0001",  
    "os579": "0.0001",  
    "os580": "0.0001",  
    "os581": "0.0001",  
    "os582": "0.0001",  
    "os583": "0.0001",  
    "os584": "0.0001",  
    "os585": "0.0001",  
    "os586": "0.0001",  
    "os587": "0.0001",  
    "os588": "0.0001",  
    "os589": "0.0001",  
    "os590": "0.0001",  
    "os591": "0.0001",  
    "os592": "0.0001",  
    "os593": "0.0001",  
    "os594": "0.0001",  
    "os595": "0.0001",  
    "os596": "0.0001",  
    "os597": "0.0001",  
    "os598": "0.0001",  
    "os599": "0.0001",  
    "os600": "0.0001",  
    "os601": "0.0001",  
    "os602": "0.0001",  
    "os603": "0.0001",  
    "os604": "0.0001",  
    "os605": "0.0001",  
    "os606": "0.0001",  
    "os607": "0.0001",  
    "os608": "0.0
```

message is received on a topic that matches the topic filter;

Rule actions allow you to take the information extracted from an MQTT message and send it to another AWS service. In this report, DynamoDB database (for storing the published data from device) and AWS SNS (for pushing email alert on abnormality) are engaged.

Rule query statement [Edit](#)

The source of the messages you want to process with this rule.

```
SELECT state.reported.* FROM  
'$aws/things/A0195042J_EngineFD003/shadow/update/accepted'
```

Using SQL version 2016-03-23

#### Actions

Actions are what happens when a rule is triggered. [Learn more](#)



Split message into multiple columns of a Dyna...  
A0195042J\_Engine

[Remove](#)

[Edit](#) ▾

#### Select an action

Select an action.



Insert a message into a DynamoDB table  
DYNAMODB



Split message into multiple columns of a database table (DynamoDBv2)  
DYNAMODBV2



Invoke a Lambda function passing the message data  
LAMBDA



Send a message as an SNS push notification  
SNS

2.6) Security and Identity - AWS IoT can generate a certificate (X.509) as the credential to secure communication between the device and AWS IoT. The certificate must be registered and activated with AWS IoT, and then copied onto your device. Hence it is 1-to-1 uniquely paired with the device or “Thing”.

The screenshot shows the 'Certificates' section of the AWS IAM console. At the top is a search bar with placeholder text 'Search certificates' and a magnifying glass icon. Below the search bar are three certificate cards, each with a three-dot menu icon and the word 'ACTIVE'. The first card has the ID '4d2dec1d7ba4ac4286...' and the second has 'ef8e4b716bd02ee675...'. The third card has 'ecf87423dad4688c65...'.

## 2.7) AWS services - DynamoDB, Lambda, SNS, S3, and etc.

Rules will be created to define the interactions:

f.1) Publish data into DynamoDB table.

f.2) Push email alert using SNS.

## 3. AWS IoT Realization

Have known the concept of AWS IoT, the rest is to implement it. I start with creating an AWS free tier account as well as the IAM role.

- 3.1) Log in to url below and create the free tier account first:

[https://portal.aws.amazon.com/billing/signup?refid=em\\_127222&redirect\\_url=https%3A%2F%2Faws.amazon.com%2Fregistration-confirmation#/start](https://portal.aws.amazon.com/billing/signup?refid=em_127222&redirect_url=https%3A%2F%2Faws.amazon.com%2Fregistration-confirmation#/start)

The screenshot shows the 'Create an AWS account' form. It includes fields for 'Email address', 'Password', 'Confirm password', and 'AWS account name'. There is also a 'Continue' button and a link 'Sign in to an existing AWS account'.

- 3.2) Within AWS management console

-> IAM under “security, identity, & compliance” -> Roles -> Create role -> Another AWS account -> Account ID -> next:Permission -> check policy name “Administrator Access” -> next:Tags -> skip “add tags” -> next:Review -> add “administrator” as Role name -> “create role”

Screenshot of the AWS Management Console showing the creation of a new IAM role.

**Top Navigation:** Services ▾ Resource Groups ▾ History Console Home

**Search Bar:** Find a service by name or feature (for example, EC2, S3 or VM storage)

**Group Buttons:** Group A-Z

**Service Categories:**

- Compute:** EC2, Lightsail, ECS, ECR, Lambda, Batch, Elastic Beanstalk, Serverless Application Repository
- Storage:** S3, EFS, FSx
- Robotics:** AWS RoboMaker
- Blockchain:** Amazon Managed Blockchain
- Analytics:** Athena, EMR, CloudSearch, Elasticsearch Service, Kinesis, QuickSight, Data Pipeline, AWS Glue, AWS Lake Formation, MSK
- Business Applications:** Alexa for Business, Amazon Chime, WorkMail
- Management & Governance:** AWS Organizations, CloudWatch, AWS Auto Scaling
- Security, Identity, & Compliance:** IAM
- End User Computing:** WorkSpaces, AppStream 2.0, WorkDocs, WorkLink
- Internet Of Things:** IoT Core, Amazon FreeRTOS

**IAM Section:**

**Create role** (button) Delete role

Showing 8 results

Role name	Description	Trusted entities
A0195042J_EngineFD001...	AWS service: iot	
A0195042J_EngineFD005...	AWS service: iot	
A0195042J_FD001_Rule	AWS service: iot	
A0195042J_Rasp_Rule	AWS service: iot	
A0195042J_Rule	AWS service: iot	
AWSServiceRoleForSupport	Enables resource access for AWS to provide billing, admin...	AWS service: support (Service-Linked role)
AWSServiceRoleForTrustee...	Access for the AWS Trusted Advisor Service to help reduc...	AWS service: trustedadvisor (Service-Linked ...)
TestCac-role-5kgrny6o		AWS service: lambda

1 2 3 4

**Create role (Step 1 of 4): Select type of trusted entity**

Allows entities in other accounts to perform actions in this account. Learn more

**Account ID\***:

**Options**:

- Require external ID (Best practice when a third party will assume this role)
- Require MFA

**Create role (Step 2 of 4): Specify accounts that can use this role**

**Create role (Step 3 of 4): Attach permissions policies**

Choose one or more policies to attach to your new role.

**Create policy**

**Filter policies** ▾  Showing 567 results

Policy name	Used as	Description
<input checked="" type="checkbox"/> AdministratorAccess	Permissions policy (1)	Provides full access to AWS services ...
<input type="checkbox"/> AlexaForBusinessDeviceSetup	None	Provide device setup access to Alexa...
<input type="checkbox"/> AlexaForBusinessFullAccess	None	Grants full access to Alexa for Busines...
<input type="checkbox"/> AlexaForBusinessGatewayExecution	None	Provide gateway execution access to ...
<input type="checkbox"/> AlexaForBusinessNetworkProfileServicePolicy	None	This policy enables Alexa for Business...

1 2 3 4 Cancel Next: Permissions

**Create role (Step 4 of 4): Review**

Provide the required information below and review this role before you create it.

**Role name\***:

Use alphanumeric and '+-\_.' characters. Maximum 64 characters.

**Role description**:

**Trusted entities**: The account

**Policies**:  AdministratorAccess

\* Required Cancel Previous Create role

- 3.3) Go back to AWS management console, choose region “**Asia Pacific (Singapore)**”, linking to a unique endpoint **host name**, which is to be deployed later for establish MQTT communication. And then access to “**IoT Core**”:

The screenshot shows two separate views of the AWS Management Console.

**Top View (AWS Services Catalog):**

- Region: Singapore
- Custom Endpoint: `si1o9sgjgf5z0-ats.iot.ap-southeast-1.amazonaws.com`
- Logs: DISABLED

**Bottom View (AWS Services Catalog):**

- Region: Singapore
- Services listed include Compute, Robotics, Analytics, Business Applications, Storage, Management & Governance, Security, Identity, & Compliance, and Internet Of Things (IoT Core).

- 3.4) A device must be registered in the registry first, in order to grant the certificate, private key, and root CA certificate that are used to communicate with AWS IoT:
- On the **Welcome to the AWS IoT Console** page, in the navigation pane, choose **Manage**:

The screenshot shows the “Welcome to the AWS IoT Console” page. The navigation pane on the left has “Manage” highlighted.

**Welcome to the AWS IoT Console**

To get started, you can jump into the recommended starting points below, or explore other learning resources as needed.

**See how AWS IoT works**

Explore an interactive tutorial through the components of AWS IoT.

**Start the tutorial**

**Connect to AWS IoT**

Connect a device, a mobile or web app to AWS IoT in a few easy steps!

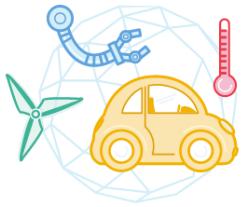
**View connection options**

**Explore documentation**

The AWS IoT documentation is a great resource for more details.

**Go to documentation**

- b. On the **You don't have any things yet** page, choose **Register a thing**:



**You don't have any things yet**

A thing is the representation of a device in the cloud.

[Learn more](#)

[Register a thing](#)

- c. On the Creating AWS IoT things page, choose Create a single thing:

**Creating AWS IoT things**

An IoT thing is a representation and record of your physical device in the cloud. Any physical device needs a thing record in order to work with AWS IoT. [Learn more](#).

**Register a single AWS IoT thing**

Create a thing in your registry

[Create a single thing](#)

**Bulk register many AWS IoT things**

Create things in your registry for a large number of devices already using AWS IoT, or register devices so they are ready to connect to AWS IoT.

[Create many things](#)

[Cancel](#)

[Create a single thing](#)

- d. On the **Create a thing** page, in the **Name** field, enter a name for your thing;

For task 1 - publish 2 different engines' data into AWS IoT, name the thing as

**A0195042J\_EngineFD001 & A0195042J\_EngineFD003.**

For task 2 – publish downloaded rainfall data into AWS IoT through Raspberry Pi board,

name the thing as **A0195042J\_Raspi**.

For task 3 – publish Raspberry Pi self-monitoring temperature data, name the thing as

**A0195042J\_Raspi\_Monitor.**

Choose **Next**.

CREATE A THING  
Add your device to the thing registry

This step creates an entry in the thing registry and a thing shadow for your device.

Name  
A0195042J\_Raspi\_Monitor

STEP  
1/3

- e. On the **Add a certificate for your thing** page, choose **Create certificate**. This generates an X.509 certificate and key pair:

CREATE A THING  
Add a certificate for your thing

A certificate is used to authenticate your device's connection to AWS IoT.

**One-click certificate creation (recommended)**  
This will generate a certificate, public key, and private key using AWS IoT's certificate authority.  
**Create certificate**

---

**Create with CSR**  
Upload your own certificate signing request (CSR) based on a private key you own.  
**Create with CSR**

---

**Use my certificate**  
Register your CA certificate and use your own certificates for one or many devices.  
**Get started**

---

**Skip certificate and create thing**  
You will need to add a certificate to your thing later before your device can connect to AWS IoT.  
**Create thing without certificate**

- f. On the **Certificate created!** page, download your public and private keys, certificate, and root certificate authority (CA):

- f.1) Choose **Download** for your certificate.  
f.2) Choose Download for your private key.  
f.3) Choose Download for the Amazon root CA. A new webpage is displayed.  
Choose **RSA 2048 bit key: Amazon Root CA 1**. This opens another webpage with the text of the root CA certificate. Copy this text and paste it into a file named **Amazon\_Root\_CA\_1.pem**.  
Keep these files to a different directory.

Choose **Activate** to activate the X.509 certificate, and then choose **Attach a policy**.

In order to connect a device, you need to download the following:

A certificate for this thing	a9589dcfe7.cert.pem	<b>Download</b>
A public key	a9589dcfe7.public.key	<b>Download</b>
A private key	a9589dcfe7.private.key	<b>Download</b>

You also need to download a root CA for AWS IoT:  
A root CA for AWS IoT [Download](#)

**Activate**

Cancel      Done      **Attach a policy**

#### Server Authentication

Server certificates allow your devices to verify that they're communicating with AWS IoT and not another server impersonating AWS IoT. Service certificates must be copied onto your device and referenced when devices connect to AWS IoT. For more information, see the [AWS IoT Device SDKs](#).

AWS IoT server certificates are signed by one of the following CA certificates:

##### VeriSign Endpoints (legacy)

- RSA 2048 bit key: [VeriSign Class 3 Public Primary G5 root CA certificate](#)

##### Amazon Trust Services Endpoints (preferred)

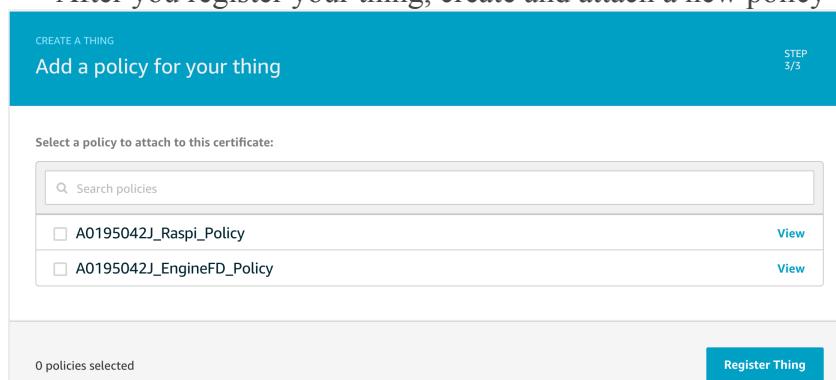
- RSA 2048 bit key: [Amazon Root CA 1](#).
- RSA 4096 bit key: Amazon Root CA 2 - Reserved for future use.
- ECC 256 bit key: [Amazon Root CA 3](#).
- ECC 384 bit key: Amazon Root CA 4 - Reserved for future use.

### 3.5) Add a policy to the thing, and attach this policy to the certificate.

This policy allows your device to perform all AWS IoT actions on all AWS IoT resources.

#### a. On the **Add a policy for your thing** page, choose **Register Thing**.

After you register your thing, create and attach a new policy to the certificate.



#### b. On the AWS IoT console, in the navigation pane, choose **Secure**, and then choose **Policies > Create**.

On the **Create a policy** page:

##### b.1) Enter a **Name** for the policy:

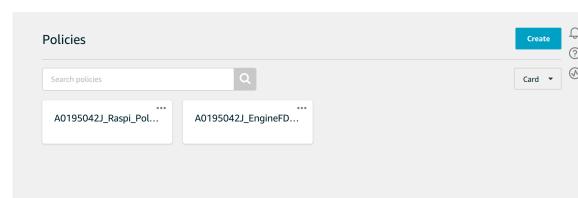
For task1 - **A0195042J\_EngineFD\_Policy**

For task2 - **A0195042J\_Raspi\_Policy**

For task3 - **A0195042J\_Raspi\_Monitor\_Policy**

##### b.2) For **Action**, enter **iot:\***. For **Resource ARN**, enter **\***.

##### b.3) Under **Effect**, choose **Allow**, and then choose **Create**.



Name  
A0195042J\_Raspi\_Monitor\_Policy

Add statements

Policy statements define the types of actions that can be performed by a resource.

Action  
iot:\*

Resource ARN  
\*

Effect  
 Allow  Deny

[Remove](#)

- c. Choose the previous created AWS IoT thing -> **Security** -> choose linked certificate. In the certificate detail page, choose **Actions**, and then choose **Attach policy**. Choose the policy created, and then choose **Attach**.

THING  
A0195042J\_Raspi\_Monitor  
NO TYPE

Actions ▾

Details Security Thing Groups Billing Groups Shadow Interact

Certificates [Create certificate](#) [View other options](#)

a9589dcfe7bba338318ccbbedaaec512e5c3d26227a9f65eaa8c230d9b1ccfac6 ACTIVE

Actions ▾

Details Policies Things Non-compliance

Certificate ARN  
A certificate Amazon Resource Name (ARN) uniquely identifies this certificate. [Learn more](#)  
arn:aws:iot:ap-southeast-1:121070346726:cert:a9589dcfe7bba338318

Details  
Issuer  
OU=Amazon Web Services O=Amazon.com Inc. L=Seattle ST=Washington C=US  
Subject  
CN=AWS IoT Certificate

Activate  
Deactivate  
Revoke  
Accept transfer  
Reject transfer  
Revoke transfer  
Start transfer  
Attach policy  
Attach thing  
Download  
Delete

Attach policies to certificate(s)

Policies will be attached to the following certificate(s):  
a9589dcfe7bba338318ccbbedaaec512e5c3d26227a9f65eaa8c230d9b1ccfac6

Choose one or more policies

<input type="checkbox"/>	Search policies
<input type="checkbox"/>	A0195042J_Raspi_Policy <a href="#">View</a>
<input type="checkbox"/>	A0195042J_EngineFD_Policy <a href="#">View</a>
<input checked="" type="checkbox"/>	A0195042J_Raspi_Monitor_Policy <a href="#">View</a>

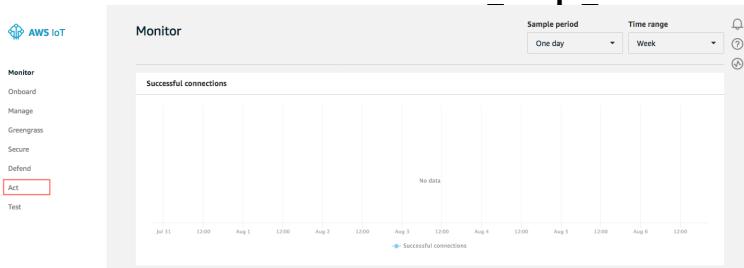
1 policy selected [Cancel](#) [Attach](#)

- 3.6) Configure and test the rules for publish data into DynamoDB table
- To publish the data from the device/Thing into DynamoDB table, the rule splits the data into columns of a DynamoDB table.
  - 1) In the AWS IoT console, in the left navigation pane, choose **Act**.
  - 2) On the **Act** page, choose **Create a rule**.
  - 3) On the **Create a rule** page, in the Name field, enter a name for your rule

For task1 - **A0195042J\_EngineFD001\_Rule &**

**A0195042J\_EngineFD003\_Rule**

For task2 - **A0195042J\_Raspi\_Rule**



#### You don't have any rules yet

Rules give your things the ability to interact with AWS and other web services. Rules are analyzed and actions are performed based on the messages sent by your things.

[Learn more](#)

[Create a rule](#)

- Scroll down to **Rule query statement**. Choose the latest version from the **Using SQL version** drop-down list (2016-03-23).

In the **Rule query statement** field,

For task1, enter:

```
SELECT state.reported.* FROM '$aws/things/A0195042J_EngineFD001/shadow/update/accepted'
SELECT state.reported.* FROM '$aws/things/A0195042J_EngineFD003/shadow/update/accepted'
```

For task 2, enter:

```
SELECT state.reported.* FROM '$aws/things/A0195042J_Raspi/shadow/update/accepted'
```

Rule query statement

Indicate the source of the messages you want to process with this rule.

Using SQL version

2016-03-23

Rule query statement

SELECT <Attribute> FROM <Topic Filter> WHERE <Condition>. For example: SELECT temperature FROM 'iot/topic' WHERE temperature > 50. To learn more, see [AWS IoT SQL Reference](#).

```
1 | SELECT state.reported.* FROM '$aws/things/A0195042J_EngineD0001/shadow/update/accepted'
```

c. In Set one or more actions, choose Add action

Set one or more actions

Select one or more actions to happen when the above rule is matched by an inbound message. Actions define additional activities that occur when messages arrive, like storing them in a database, invoking cloud functions, or sending notifications. (\*.required)

[Add action](#)

d. On the Select an action page, choose Split message into multiple columns of a DynamoDB table, configure action

Select an action

Select an action.

-  Insert a message into a DynamoDB table  
DYNAMODB
-  Split message into multiple columns of a DynamoDB table (DynamoDBv2)  
DYNAMODBV2
-  Send a message to a Lambda function  
LAMBDA

e. From the Configure Action page, select ‘Create a new resource’.

 Split message into multiple columns of a DynamoDB table (DynamoDBv2)

The DynamoDBv2 action allows you to write all or part of an MQTT message to a DynamoDB table. Each attribute in the payload is written to a separate column in the DynamoDB database. Messages processed by this action must be in the JSON format.

\*Table name

Choose a resource

[Create a new resource](#)

f. In the newly pop-out window, select “Create table”

DynamoDB

Dashboard

Tables

Backups

Reserved capacity

Preferences

DAX

Dashboard

Clusters

Create table

Amazon DynamoDB is a fully managed non-relational database service that provides fast and predictable performance with seamless scalability.

[Create table](#)

Recent alerts

No CloudWatch alarms have been triggered. [View all in CloudWatch](#)

Total capacity for Asia Pacific (Singapore)

g. Provide the table name, primary key, and sort key, then select “Create”  
 For task1, table name – **A0195042J\_Engine**  
 For task 2, table name – **A0195042J\_Raspi**  
 Primary key is common – **id**  
 Sort key is also common – **timestamp**

Create DynamoDB table

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name*	A0195042J_Engine	
Primary key*	Partition key	
	<input type="text" value="Id"/> String	
<input checked="" type="checkbox"/> Add sort key	<input type="text" value="timestamp"/> String	

Table settings

Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table is created.

h. Go back to previous window, from scroll-down list select just created table name - e.g. **A0195042J\_Engine**, and also select the rule created earlier - e.g. **A0195042J\_EngineFD001\_Rule**. Then select “**Add action**”:

The DynamoDBv2 action allows you to write all or part of an MQTT message to a DynamoDB table. Each attribute in the payload is written to a separate column in the DynamoDB database. Messages processed by this action must be in the JSON format.

\*Table name  
A0195042J\_Engine

Choose or create a role to grant AWS IoT access to perform this action.

A0195042J\_EngineFD001\_Rule

- i. In order for the shadow document to authenticate and receive the actual data sent from device by the python script, we need to first simulate the MQTT communication by configuring the shadow document in JSON format, then test it based on what we have established the IoT configuration - Thing attached with certificate, policy, and the defined rule action.  
 Devices publish MQTT messages on topics. Hence we use the AWS IoT MQTT client to subscribe to these topics to see these messages.

Take engine1 (thing) as an example, it is from this Shadow Document that the device stores its information payload to either publish or subscribe to a Topic. Topics are communication channels in which the devices connect and communicate with each other. For this verification, we make use of the following Topics as listed below:

In general, they are:

\$aws/things/A0195042J\_EngineFD001/shadow/update  
 \$aws/things/A0195042J\_EngineFD001/shadow/update/accepted  
 \$aws/things/A0195042J\_EngineFD001/shadow/get  
 \$aws/things/A0195042J\_EngineFD001/shadow/get/accepted

```

Update to this thing shadow
$aws/things/A0195042J_EngineFD001/shadow/update

Update to this thing shadow was accepted
$aws/things/A0195042J_EngineFD001/shadow/update/accepted

Update this thing shadow documents
$aws/things/A0195042J_EngineFD001/shadow/update/documents

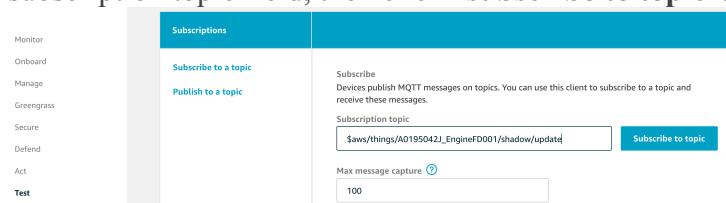
Update to this thing shadow was rejected
$aws/things/A0195042J_EngineFD001/shadow/update/rejected

Get this thing shadow
$aws/things/A0195042J_EngineFD001/shadow/get

Get this thing shadow accepted
$aws/things/A0195042J_EngineFD001/shadow/get/accepted

```

Go to Test, enter “\$aws/things/A0195042J\_EngineFD001/shadow/update” in the subscription topic field, then click “subscribe to topic”:



Do a similar “subscribe a topic” for the rest 3 topics:



Select topic of “\$aws/things/A0195042J\_EngineFD001/shadow/update”, edit JSON message as below, and click “publish to topic”:

```
{
  "state": {
    "desired": {
      "message": null
    },
    "reported": {
      "message": null,
      "id": "FD001_1",
      "timestamp": "timestamp",
      "Matric_Number": "A0195042J",
      "cycle": "1",
      "os1": "01",
      "os2": "02",
      "os3": "03"
    }
  }
}
```

In the topic “\$aws/things/A0195042J\_EngineFD001/shadow/update/accepted”, it is supposed to receive message like below for successful publishing:

The screenshot shows the AWS IoT Publish interface. A message is being sent to the topic `$aws/things/A0195042J_EngineFD001/shadow/update/accepted`. The message content is:

```

1 [
2   "message": "Hello from AWS IoT console"
3 ]

```

After publishing, the message is shown in the message list with the timestamp Sep 20, 2019 5:45:17 PM. The message content is:

```
{
  "state": {
    "desired": {
      "message": null
    },
    "reported": {
      "message": null,
      "id": "FD001_1",
      "timestamp": "timestamp",
      "Metric_Number": "A0195042J",
      "cycle": "1",
      "os1": "01",
      "os2": "02",
      "os3": "03"
    }
  },
  "metadata": {
    "desired": {
      "message": {
        "timestamp": 1568972717
      }
    },
    "reported": {
      "message": {
        "timestamp": 1568972717
      },
      "id": {
        "timestamp": 1568972717
      }
    }
  }
}
```

- 3.7) To receive email alert on abnormal reading, rule needs to be created to engage with AWS SNS service. In task3, rule is configured so that the abnormal temperature will be sent from raspberry pi board to an Amazon SNS topic.
- From the AWS SNS console, choose **Topics**, and then choose **Create topic**

The screenshot shows the AWS SNS Topics page. There is one topic listed:

Name	ARN
dynamodb	arn:aws:sns:southeast-1:121070346726:dynamodb

- Enter a name for the topic - **IoT\_SNS\_Email\_Alert**  
Enter a display name for the topic – **IoT\_SNS**  
Then choose **Create topic** and under subscriptions tab, choose **Create subscription**:

The screenshot shows the Create topic form. The Name field is filled with `IoT_SNS_Email_Alert`. The Display name - optional field is filled with `IoT_SNS`.

Topic IoT\_SNS\_Email\_Alert created successfully.  
You can create subscriptions and send messages to them from this topic.

Amazon SNS > Topics > IoT\_SNS\_Email\_Alert

**IoT\_SNS\_Email\_Alert**

**Details**

Name IoT_SNS_Email_Alert	Display name IoT_SNS
ARN arn:aws:sns:ap-southeast-1:121070346726:IoT_SNS_Email_Alert	Topic owner 121070346726

**Subscriptions** | Access policy | Delivery retry policy (HTTP/S) | Delivery status logging | Encryption | Tags

**Subscriptions (0)**

**Create subscription**

No subscriptions found

You don't have any subscriptions to this topic.

Create subscription

- c. On **Create subscription**, from the **Protocol** drop-down list, choose **Email**. In the **Endpoint** field, enter school email – **e0383707@u.nus.edu**, and then choose **Create subscription**:

Amazon SNS > Subscriptions > Create subscription

**Create subscription**

**Details**

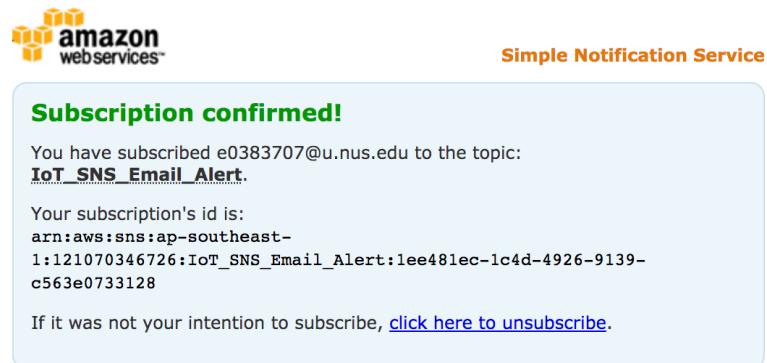
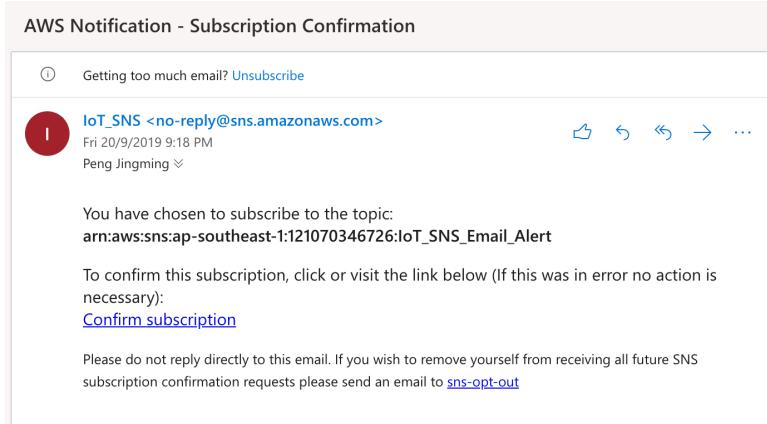
Topic ARN  
arn:aws:sns:ap-southeast-1:121070346726:IoT\_SNS\_Email\_Alert

Protocol  
The type of endpoint to subscribe  
Email

Endpoint  
An email address that can receive notifications from Amazon SNS.  
e0383707@u.nus.edu

After your subscription is created, you must confirm it. Info

After your subscription is created, you must confirm it. Amazon SNS doesn't send messages to an endpoint until the subscription is confirmed. Hence, access into the email and click the url link to confirm this subscription:



Subscriptions (1)					
	ID	Endpoint	Status	Protocol	Topic
<input checked="" type="radio"/>	1ee481ec-1c4d-4926-9139-c563e0733128	e0383707@u.nus.edu	Confirmed	EMAIL	IoT_SNS_Email_Alert

- d. Create corresponding rules to trigger this SNS action, enter a name for it - **A0195042J\_Raspi\_AbnTemp\_Rule**

Create a rule

Create a rule to evaluate messages sent by your things and specify what to do when a message is received (for example, write data to a DynamoDB table or invoke a Lambda function).

Name: A0195042J\_Raspi\_AbnTemp\_Rule

Description: To send email alert when device send abnormal temperature data

Under **Rule query statement**, enter the following AWS IoT SQL query statement:

```
SELECT state.reported.* FROM '$aws/things/A0195042J_Raspi_Monitor/shadow/update/accepted' WHERE state.reported.temp > 47.00
```

Rule query statement

Using SQL version

2016-03-23

Rule query statement

SELECT <Attribute> FROM <Topic Filter> WHERE <Condition>. For example: SELECT temperature FROM 'iot/topic' WHERE temperature > 50. To learn more, see [AWS IoT SQL Reference](#).

```

1  SELECT state.reported.* FROM '$aws/things/A0195042J_Raspi_Monitor/shadow/update/accepted'
2  WHERE state.reported.temp > 47.00
3

```

**Cancel** **Update**

On the **Select an action** page, choose **Send a message as an SNS push notification**:

Select an action

Select an action.

-  Insert a message into a DynamoDB table  
DYNAMODB
-  Split message into multiple columns of a DynamoDB table (DynamoDBv2)  
DYNAMODBV2
-  Send a message to a Lambda function  
LAMBDA
-  Send a message as an SNS push notification  
SNS

On the **Configure action** page, under **SNS target**, choose **Select** to expand the SNS topic. Then choose **Select** next to the Amazon SNS topic you created earlier.

Under **Message format**, choose **RAW**.

Choose **Create a new role**, give the name - **A0195042J\_Raspi\_AbnTemp\_Rule**:

Configure action

 Send a message as an SNS push notification

\*SNS target

IoT_SNS_Email_Alert	<b>Create</b>	<b>Clear</b>	<b>Select</b>
---------------------	---------------	--------------	---------------

Message format

RAW
-----

Choose or create a role to grant AWS IoT access to perform this action.

A0195042J_Raspi_AbnTemp_Rule	<b>Create Role</b>	<b>Select</b>
------------------------------	--------------------	---------------

- e. Follow the similar steps in 1.11) to test the rule, this time the MQTT topics are the following:

\$aws/things/A0195042J\_Raspi\_Monitor/shadow/update  
 \$aws/things/A0195042J\_Raspi\_Monitor/shadow/update/accepted  
 \$aws/things/A0195042J\_Raspi\_Monitor/shadow/get  
 \$aws/things/A0195042J\_Raspi\_Monitor/shadow/get/accepted

## MQTT

Use topics to enable applications and things to get, update, or delete the state information for a Thing (Thing Shadow)

[Learn more](#)

**Update to this thing shadow**

```
$aws/things/A0195042J_Raspi_Monitor/shadow/update
```

**Update to this thing shadow was accepted**

```
$aws/things/A0195042J_Raspi_Monitor/shadow/update/accepted
```

**Get this thing shadow**

```
$aws/things/A0195042J_Raspi_Monitor/shadow/get
```

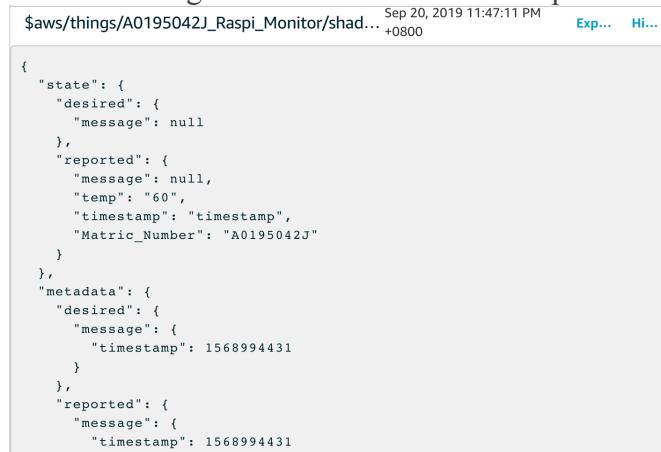
**Get this thing shadow accepted**

```
$aws/things/A0195042J_Raspi_Monitor/shadow/get/accepted
```

Select topic of “\$aws/things/A0195042J\_Raspi\_Monitor/shadow/update”, edit JSON message as below, and click “publish to topic”:

```
{  
  "state": {  
    "desired": {  
      "message": null  
    },  
    "reported": {  
      "temp": "60",  
      "timestamp": "timestamp",  
      "Matric_Number": "A0195042J"  
    }  
  }  
}
```

In the topic “\$aws/things/A0195042J\_Raspi\_Monitor/shadow/update/accepted”, it is supposed to receive message like below for successful publishing:



The screenshot shows a CloudWatch Metrics log entry. The log ID is \$aws/things/A0195042J\_Raspi\_Monitor/shadow...+0800. The timestamp is Sep 20, 2019 11:47:11 PM. The log message is a JSON object representing the published MQTT message:

```
{  
  "state": {  
    "desired": {  
      "message": null  
    },  
    "reported": {  
      "temp": "60",  
      "timestamp": "timestamp",  
      "Matric_Number": "A0195042J"  
    }  
  },  
  "metadata": {  
    "desired": {  
      "message": {  
        "timestamp": 1568994431  
      }  
    },  
    "reported": {  
      "message": {  
        "timestamp": 1568994431  
      }  
    }  
  }  
}
```

- 3.8) In a short summary, for the 3 tasks, what have been created for the Things, certificates, keys, policies, rules, tables:

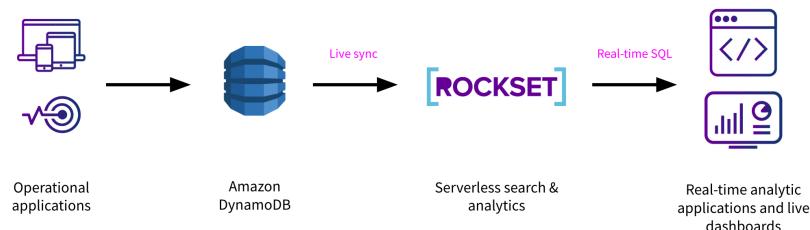
Task1		
Thing	A0195042J_EngineFD001	A0195042J_EngineFD003
X.509 Certificate	ecf87423da-certificate.pem.crt	ef8e4b716b-certificate.pem.crt
Private Key	ecf87423da-private.pem.key	ef8e4b716b-private.pem.key
Public Key	ecf87423da-public.pem.key	ef8e4b716b-public.pem.key
root CA	AmazonRootCA1.pem	AmazonRootCA1.pem
Policy	A0195042J_EngineFD_Policy	A0195042J_EngineFD_Policy
Rule	A0195042J_EngineFD001_Rule	A0195042J_EngineFD003_Rule
table	A0195042J_Engine	A0195042J_Engine
Task2		
Thing	A0195042J_Raspi	
X.509 Certificate	4d2dec1d7b-certificate.pem.crt	
Private Key	4d2dec1d7b-private.pem.key	
Public Key	4d2dec1d7b-public.pem.key	
root CA	AmazonRootCA1.pem	
Policy	A0195042J_Raspi_Policy	
Rule	A0195042J_Raspi_Rule	
table	A0195042J_Raspi	
Task3		
Thing	A0195042J_Raspi_Monitor	
X.509 Certificate	a9589dcfe7-certificate.pem.crt	
Private Key	a9589dcfe7-private.pem.key	
Public Key	a9589dcfe7-public.pem.key	
root CA	AmazonRootCA1.pem	
Policy	A0195042J_Raspi_Monitor_Policy	
Rule	A0195042J_Raspi_AbnTemp_Rule	
table	N.A.	

## 4. Visualization Concept

There're various ways of visualizing data that were transferred into the DynamoDB table:

- 4.1) By python – directly call matplotlib library to plot out the data;
- 4.2) By API gateway – create the API gateway to retrieve and present data in url;
- 4.3) By BI tools – establish connection between DynamoDB and BI tool, and visualize data inside BI tool.

In this report, I am demonstrating how to connect DynamoDB with Tableau through Rockset JDBC driver, and visualize the data inside Tableau:



## 5. Visualization Realization

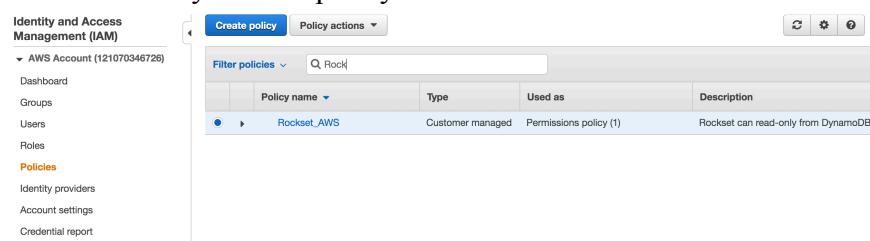
### 5.1) Configure AWS IAM policy

- a. Navigate to the IAM Service in the AWS Management Console.
- b. Set up a new policy by navigating to **Policies** and clicking **Create policy – Rockset\_AWS**
- c. Set up read-only access to your DynamoDB table, switch to the “JSON” tab and paste the policy shown below:

Table name are:

For task 1 - **A0195042J\_Engine**;  
For task 2 - **A0195042J\_Raspi**;

- d. Save this newly created policy:



5.2) Grant Rockset permissions to access your AWS resource using AWS Access Keys.

- a. Create a new user by navigating to **Users** and clicking **Add User - Administrator**
  - b. Enter a user name and check the **Programmatic access** option. Click to continue.
  - c. Choose **Attach existing policies directly** then select the policy you created in 5.1)

The screenshot shows the 'Add user' wizard step 2: Set permissions. It includes options like 'Add user to group', 'Copy permissions from existing user', and the highlighted 'Attach existing policies directly'. Below is a table of 422 policies, with one row selected:

	Policy name	Type	Used as	Description
<input type="checkbox"/>	AlexaForBusinessD...	AWS managed	None	Provide device setup access to AlexaForBu...
<input type="checkbox"/>	AlexaForBusinessF...	AWS managed	None	Grants full access to AlexaForBusiness reso...
<input type="checkbox"/>	AlexaForBusinessG...	AWS managed	None	Provide gateway execution access to Alexa...
<input type="checkbox"/>	AlexaForBusinessR...	AWS managed	None	Provide read only access to AlexaForBusine...
<input type="checkbox"/>	AmazonAPIGatewa...	AWS managed	None	Provides full access to create/edit/delete A...
<input type="checkbox"/>	AmazonAPIGatewa...	AWS managed	None	Provides full access to invoke APIs in Amaz...

The screenshot shows the 'Summary' page for the user 'Administrator'. It displays the User ARN (arn:aws:iam::121070346726:user/Administrator), Path (/), and Creation time (2019-08-28 00:48 UTC+0800). The 'Permissions' tab is selected, showing that two policies are applied: 'Rockset\_AWS' (Attached directly) and 'Managed policy'.

- d. When the new user is successfully created you should see the **Access key ID** and **Secret access key** displayed on the screen.

The screenshot shows the 'Sign-in credentials' and 'Access keys' sections for the user 'Administrator'. In the 'Sign-in credentials' section, it shows the console sign-in link (https://121070346726.signin.aws.amazon.com/console), console password status (Enabled), assigned MFA device (Not assigned), and signing certificates (None). In the 'Access keys' section, it shows a table of access keys:

Access key ID	Created	Last used	Status
AKIARYMC24HTPX5F6XH2	2019-09-08 00:52 UTC+0800	2019-09-21 01:32 UTC+0800 with ...	Active   Make inactive <span style="color:red;">X</span>

5.3) Create a collection backed by Amazon DynamoDB, Rockset scans the DynamoDB tables to continuously ingest and then subsequently uses the stream to update collections as new objects are added to the DynamoDB table. The sync latency is no more than 5 seconds under regular load.

In the Rockset Console, create a collection from Workspace > Collections > Create Collection:

a. Choose Amazon DynamoDB as the new data source

b. Choose integration - Rock\_AWS and then Start

c. Choose Collection name, DynamoDB table name, and AWS Region, then **create**  
For task1:

Collection name - A0195042J\_DynamoDB\_Engine  
DynamoDB table name - A0195042J\_Engine

For task2:

Collection name - A0195042J\_DynamoDB\_Raspi  
DynamoDB table name - A0195042J\_Raspi

Name	Workspace	Source Type	Last Updated	Created By	Size	Status
A0195042J_DynamoDB_Engine	commons	Amazon DynamoDB	Sep 20	kevinming2012@...	3.5 MB	Ready
A0195042J_DynamoDB_Raspi	commons	Amazon DynamoDB	2:15 am	kevinming2012@...	30.3 KB	Ready
_events	commons		2:15 am		58.5 KB	Ready

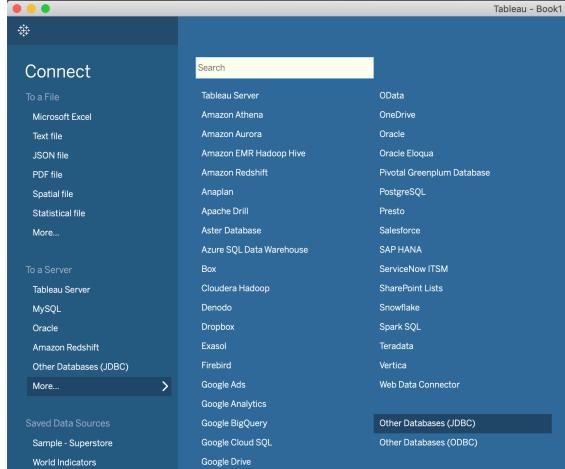
5.4) Set up Tableau Desktop and configure it for use with Rockset.  
Tableau Desktop uses JDBC driver to perform operations.  
JDBC support is available in Tableau Desktop version 2019.1 (or later) and Rockset Java Client version 0.5.11 (or later).

- Download the jar (rockset-java-\$version.jar) from <https://oss.sonatype.org/#nexus-search;quick~rockset>
- Place the Rockset Java Client jar in the following folder:  
Mac: ~/Library/Tableau/Drivers
- Create an API Key using the Rockset Console under Manage > API Keys.



Name	Key	Created
A0195042J_tableau	s0o6*****K8HP <a href="#">Copy</a>	Sep 7

- In Tableau, navigate to Connect To a Server and select Other Databases (JDBC):



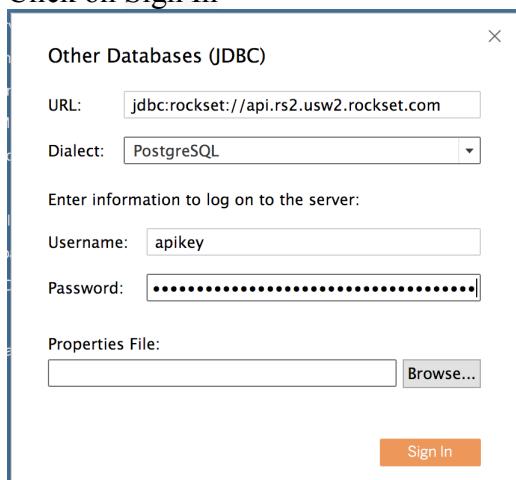
- Configure the connection as follows:

Use `jdbc:rockset://api.rs2.usw2.rockset.com` for URL

Select PostgreSQL as dialect.

Enter apikey as Username and Rockset API Key as the password.

Click on Sign In



The dialog shows the following fields:

- URL:** `jdbc:rockset://api.rs2.usw2.rockset.com`
- Dialect:** PostgreSQL
- Enter information to log on to the server:**
  - Username:** apikey
  - Password:** (redacted)
- Properties File:** (empty input field with a browse button)
- Sign In** button

DynamoDB Table - A0195042J\_Engine inside Tableau

Tableau - Book1

Connections Add

Connections

jdbc:rockset://...-o2.rockset.com  
Other Databases (0 items)

Database rockset

Schema commons

Table \_events

A0195042J\_DynamoDB\_Engine A0195042J\_L\_moDB\_Resp New Custom SQL

Go to Worksheet

⋮ Sort fields Data source order ▾

Show aliases Show hidden fields 1,000 ↴ rows

Connection Live Extract Filters Add

⋮

Panel A0195042J\_DynamoDB\_Engine

No.	No.	No.	No.	No.	No.	No.	No.	No.	No.	No.	No.
A0195042J_DynamoDB_Eng...	A0195042J_DynamoDB_Eng...	A0195042J_DynamoDB_Eng...	A0195042J_DynamoDB_Eng...	A0195042J_D...							
_EventTime	_EventTime	_EventTime	_EventTime	_Meta	_Meta	_Meta	_Meta	_Id	_Message	_Os1	_Os2
2019-09-08T20:10:1...	19339m10s-347bf-49f7-...	A0195042J	194	F0003_19	null	0.0019	-0.0000				
2019-09-08T20:10:1...	e0d3784-49d1-4a#8	A0195042J	118	F0001_13	null	-0.0022	0.0002				
2019-09-08T20:10:1...	1939f08e-4647-47b5-...	A0195042J	159	F0001_11	null	-0.0037	-0.0003				
2019-09-08T20:10:1...	15f49efc-3c9a-76cc-...	A0195042J	163	F0001_4	null	-0.0041	-0.0002				
2019-09-08T20:10:2...	bff4ca3-0fa1-4000-...	A0195042J	146	F0001_B	null	0.0001	-0.0004				
2019-09-08T20:10:2...	26596fc-647f-412a-...	A0195042J	148	F0003_15	null	0.0012	0.0001				
2019-09-08T20:10:1...	93502c07-9605-093...	A0195042J	154	F0001_13	null	-0.0003	-0.0003				
2019-09-08T20:11:0...	6ab5c9a0-e368-265...	A0195042J	35	F0003_4	null	0.0002	0.0004				
2019-09-08T20:10:1...	10a0a0a0-456a-47ed-...	A0195042J	191	F0003_19	null	-0.0026	-0.0001				

## DynamoDB Table - A0195042J\_Raspi inside Tableau

Connections [Add](#)

Connections [Add](#)

jdbc:rockset://.w2.rockset.com  
Other databases (100)

Database [rockset](#)

Schemas [commons](#)

Table [events](#)

[A0195042J...oDB\\_Engine](#)

[A0195042J...moDB\\_Raspi](#)

[New Custom SQL](#)

A0195042J\_DynamoDB\_Raspi (commons)

Connection [Live](#) [Extract](#) Filters [Add](#)

A0195042J\_DynamoDB\_Raspi

Sort fields	Data source order	Show aliases	Show hidden fields	216	row
Abc	Abc	Abc	Abc	Abc	Abc
A0195042J_Dynamo...	A0195042J_Dynamo...	A0195042J_Dynamo...	A0195042J_DynamoDB_Raspi	A0195042J_DynamoDB_Raspi	A0195042J_DynamoDB_Raspi
Matric No	Daq Date	Rain Fall	Event Time	Id	_Meta
A0195042J	2014-04	110	2019-09-20T18:15:0...	0x830372-d3b-5e7c...	388
A0195042J	2010-11	278.6	2019-09-20T18:15:0...	9f4743d8-383a-4e9f...	347
A0195042J	2014-06	71.4	2019-09-20T18:15:0...	0ddaf1fe-1491-6fdb...	390
A0195042J	2003-12	273	2019-09-20T18:15:0...	c10022de-2117-45f6...	264
A0195042J	2010-03	238	2019-09-20T18:15:0...	f87b73bd-d500-10af...	339
A0195042J	2017-08	84.2	2019-09-20T18:15:0...	0ad38c31-99c6-fe46...	428
A0195042J	2002-02	50.8	2019-09-20T18:15:0...	406f7b30-69fa-45fc...	242
A0195042J	2018-02	14.8	2019-09-20T18:15:0...	92ae7f90-2675-e086...	434
A0195042J	2002-07	233.7	2019-09-20T18:15:0...	3e0a095b-1985-7a8...	247

## 6. Task 1 – Publish Engine Data into DynamoDB through laptop and do visualization

### 6.1) Background

We are given jet engine data in four different text files which correspond to each of the jet engines in a propulsion system (with space delimiters to separate the fields) which are shown as follows:

I am using engine1 and engine3 data for this task

There is already a total of 26 sets of data per line, which is given in the following format:

- a. ID
  - b. OS1, OS2, OS3
  - c. Sensors 1 to 22

Required to add more field as columns in DynamoDB table:

- a. Cycle
- b. Timestamp
- c. Matric\_Number

## 6.2) Configure the endpoint device – laptop

- a. Download AWS\_IoT\_Python\_SDK and install into the laptop – Mac OS

Within the terminal:

### Install brew by running:

```
/usr/bin/ruby -e "$(curl -fsSL
```

```
https://raw.githubusercontent.com/Homebrew/install/master/install"
```

### Install openssl and python3:

```
brew update
```

```
brew install openssl
```

```
brew install python@3
```

### Check version:

```
python3 --version
```

```
import ssl
```

```
print(ssl.OPENSSL_VERSION)
```

```
exit()
```

**Note that, the OpenSSL version need to be 1.0.1 or later; Python version need to be Python 3.3+.**

### Install the AWS IoT Device SDK for Python by running:

```
cd ~
```

```
git clone https://github.com/aws/aws-iot-device-sdk-python.git
```

```
cd aws-iot-device-sdk-python
```

```
python setup.py install
```

- b. Put all related files into a common directory

The files consist of:

#### b.1 Downloaded AWS IoT device SDK (consists of MQTT library)

└─ AWSIoTPythonSDK	30 Aug 2019, 9:16 AM	--	Folder
└─ _pycache_	30 Aug 2019, 9:30 AM	--	Folder
└─ core	30 Aug 2019, 9:30 AM	--	Folder
└─ exception	30 Aug 2019, 9:30 AM	--	Folder
└─ __init__.py	30 Aug 2019, 9:16 AM	24 bytes	Python Source
└─ MQTTLib.py	30 Aug 2019, 9:16 AM	61 KB	Python Source

#### b.2 Engine1 and engine3 data

└─ train_FD001.txt	20 Aug 2019, 11:27 PM	3.5 MB	Plain Text
└─ train_FD003.txt	20 Aug 2019, 11:27 PM	4.2 MB	Plain Text

#### b.3 Thing's certificate, key, CA

#### b.4 Script to publish data – A0195042J\_PublishCode\_EngineData.py

└─ AWSIoTPythonSDK	30 Aug 2019, 9:16 AM	--	Folder
└─ 4d2dec1d7b-certificate.pem.crt	10 Sep 2019, 11:45 PM	1 KB	certificate
└─ a9589dcf67-certificate.pem.crt	Yesterday, 1:40 AM	1 KB	certificate
└─ ecf87423da-certificate.pem.crt	29 Aug 2019, 1:09 AM	1 KB	certificate
└─ ef8e4b716b-certificate.pem.crt	30 Aug 2019, 12:55 PM	1 KB	certificate
└─ accessKeys.csv	8 Sep 2019, 12:53 AM	96 bytes	Comm...t (.csv)
└─ rainfall-monthly-total.csv	28 Aug 2019, 6:02 PM	7 KB	Comm...t (.csv)
└─ 4d2dec1d7b-private.pem.key	10 Sep 2019, 11:48 PM	2 KB	Keypote
└─ 4d2dec1d7b-public.pem.key	10 Sep 2019, 11:47 PM	451 bytes	Keypote
└─ a9589dcf67-private.pem.key	Yesterday, 1:40 AM	2 KB	Keypote
└─ a9589dcf67-public.pem.key	Yesterday, 1:40 AM	451 bytes	Keypote
└─ ecf87423da-private.pem.key	29 Aug 2019, 1:10 AM	2 KB	Keypote
└─ ecf87423da-public.pem.key	29 Aug 2019, 1:10 AM	451 bytes	Keypote
└─ ef8e4b716b-private.pem.key	30 Aug 2019, 12:56 PM	2 KB	Keypote
└─ ef8e4b716b-public.pem.key	30 Aug 2019, 12:55 PM	451 bytes	Keypote
└─ FD001.out.txt	9 Sep 2019, 1:12 AM	850 KB	Plain Text
└─ FD003.out.txt	9 Sep 2019, 1:12 AM	850 KB	Plain Text
└─ train_FD001.txt	20 Aug 2019, 11:27 PM	3.5 MB	Plain Text
└─ train_FD003.txt	20 Aug 2019, 11:27 PM	4.2 MB	Plain Text
└─ AmazonRootCA1.pem	29 Aug 2019, 1:11 AM	1 KB	print...archive
└─ A0195042J_PublishCode_EngineData.py	9 Sep 2019, 1:06 AM	7 KB	Python Source

### 6.3) Run the script.

Due to process time issue, I only upload the first 5000 lines of data from engine1 and engine3 txt file. So the total size of DynamoDB table - **A0195042J\_Engine**, will be 10000 lines.

```
33  v with open('train_FD001.txt','r') as infile_1:  
34  v   with open('FD001_out.txt','a') as outfile_1:  
35    # Due to process time issue, only read the 1st 5000 lines of data  
36  v     for line in infile_1.readlines()[0:5000]:  
37    |     outfile_1.write(line)  
38  
39  v with open('train_FD003.txt','r') as infile_2:  
40  v   with open('FD003_out.txt','a') as outfile_2:  
41    # Due to process time issue, only read the 1st 5000 lines of data  
42  v     for line in infile_2.readlines()[0:5000]:  
43    |     outfile_2.write([line])
```

### 6.4) Verify DynamoDB table - **A0195042J\_Engine**

The screenshot shows the AWS DynamoDB console with two tables displayed:

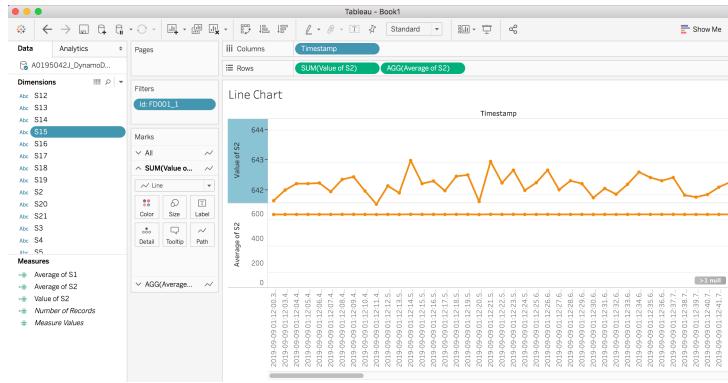
- A0195042J\_Engine**:
  - Items tab selected.
  - Scan results: Viewing 1 to 100 items.
  - Filter: id = FD001\_1.
  - Data table:

	id	timestamp	Metric_Number	cycle	os1	os2
1	FD001_1	2019-09-09 01:12:00.380299	A0195042J	1	-0.0007	-0.0001
2	FD001_1	2019-09-09 01:12:03.434963	A0195042J	2	0.0019	-0.0001
3	FD001_1	2019-09-09 01:12:04.438422	A0195042J	3	-0.0043	0.0003
- FD003\_Engine**:
  - Items tab selected.
  - Scan results: Viewing 1 to 100 items.
  - Filter: id = FD003\_1.
  - Data table:

	id	timestamp	Metric_Number	cycle	os1	os2
1	FD003_1	2019-09-09 02:36:20.197349	A0195042J	1	-0.0005	0.0004
2	FD003_1	2019-09-09 02:36:23.259513	A0195042J	2	0.0008	-0.0001
3	FD003_1	2019-09-09 02:36:24.272663	A0195042J	3	-0.0014	-0.0002

### 6.5) Visualization inside tableau

Taking the FD001, sensor 2 data – average vs distinct as an example



## 7. Task 2 – Publish rainfall data through raspberry pi to DynamoDB and do visualization

### 7.1) Background

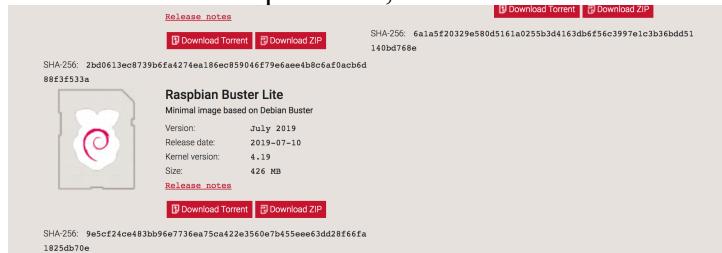
Prepare dataset by downloading the rain fall data - **rainfall-monthly-total.csv** from <https://data.gov.sg/>

### 7.2) Configure endpoint device – raspberry pi

Hardware preparation:

- raspberry pi model 3 B
- microSD card, 8GB
- Keyboard
- Monitor with HDMI cable
- WiFi Dongle

Download OS - Raspbian lite, and flash into microSD card:



- Insert microSD card onto raspberry pi, power up and login using default credential:  
username: pi  
password: raspberry

#### g. Set up WiFi connection

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
network={

    ssid="WiFi name"
    psk="WiFi Password"
}
```

See details in link below:

<https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md>

#### h. Auto-mount USB flash onto raspbian

Follow steps in link below:

<https://www.raspberrypi-spy.co.uk/2014/05/how-to-mount-a-usb-flash-disk-on-the-raspberry-pi/>

Note that, the USB flash is the median to store all the file (script, certificate, key, etc.) and to transfer into raspberry pi.

- i. Download Python3 and Pip into raspberry pi

Follow steps in link below:

<https://gist.github.com/SeppPenner/6a5a30ebc8f79936fa136c524417761d>

## Check the versions:

```

Started System Logging Service
[ OK ] Started triggerhappy global hotkey daemon
[ OK ] Started udevd -热插拔设备管理器卡状态。
[ OK ] Started rug-tools.services
[ OK ] Reached target Sound Card
[ OK ] Started Avahi - DNS+SD Stack
[ OK ] Started NetworkManager
[ OK ] Started NetworkManager-wifi
[ OK ] Started NetworkManager-wired
[ OK ] Started NetworkManager-ppp
[ OK ] Started dphys-swappfile - set up, mount/unmount, and delete a swap file.
[ OK ] Started NetworkManager-dispatcher - start/stop cgroup governor (unless shift key is pressed).
[ OK ] Reached target All Interfaces
[ OK ] Reached target Network
Starting Permit User Sessions
Starting NetworkManager-wifi service...
Starting NetworkManager-wired service...
Starting NetworkManager-ppp service...
Starting NetworkManager-dispatcher service...
Starting daily apt download activities...
May 16 11:49:46 RPI-ARMUZ apt[114]: Starting apt-daily-upgrade...
[ OK ] Started Permit User Sessions
[ OK ] Started NetworkManager-wifi
[ OK ] Started NetworkManager-wired
[ OK ] Started NetworkManager-ppp
[ OK ] Reached target Login Prompts
[ OK ] Started OpenSSH Secure Shell server
[ OK ] Started Daily apt download activities

Raspbian GNU/Linux 10 raspberrypi ttym
pi@raspberrypi: ~ python3.7 --version
Python 3.7.3
pi@raspberrypi: ~ pip3.7 --version
pip 19.2.3 from /usr/local/lib/python3.7/site-packages/pip (python 3.7)
pi@raspberrypi: ~
```

- j. Download AWS\_IoT\_Device\_Python\_SDK into raspberry pi

Inside raspberry, follow steps (unix-like system) in link below:

<https://docs.aws.amazon.com/greengrass/latest/developerguide/IoT-SDK.html>

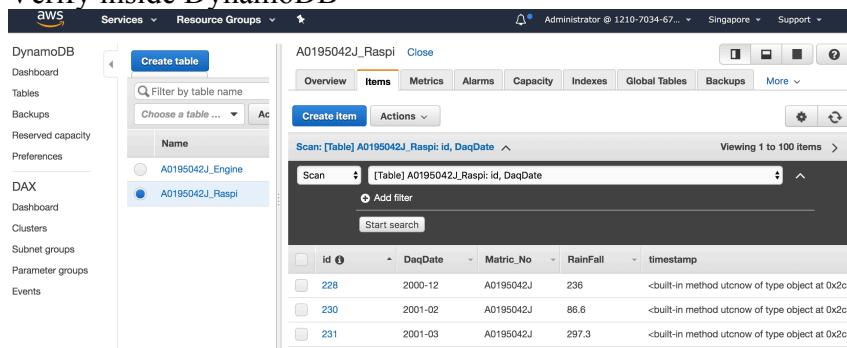
The SDK will be created into a folder named “aws-iot-device-sdk-python” under the root directory.

- k. In order to get sufficient permission, create “deviceSDK” directory with both read and write access under raspberry root directory. Move the whole folder “aws-iot-device-sdk-python” to this “deviceSDK” folder
  - l. Copy the downloaded rainfall dataset, unique certificate, key, CA, and script - **A0195042J\_PublishCode\_RaspiData.py**, into folder (EE5111) within USB flash, connect the USB flash to raspberry, and copy all these files into the “deviceSDK” folder by command:  
`cp -a /mnt/usb_flash/EE5111 /deviceSDK/aws_iot_device_sdk_python`

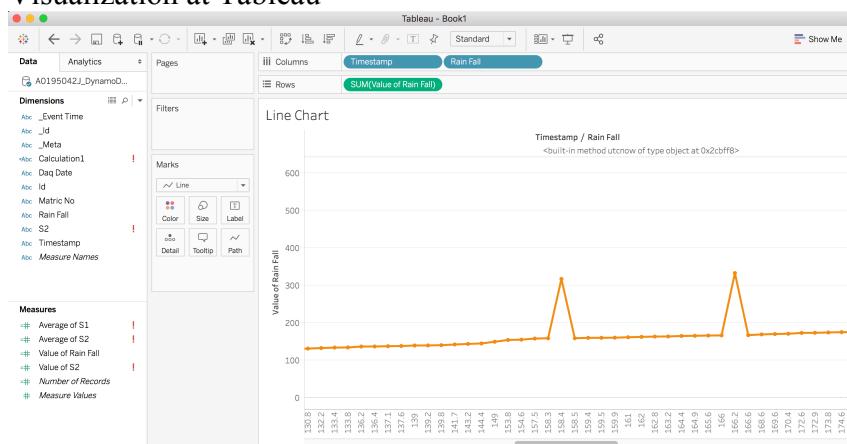
```
cp -a /mnt/usb /flash/EE3111 ./deviceSDK/aws-iot-device-sdk-python
```

- m. Run the script - **A0195042J\_PublishCode\_RaspiData.py**, and start publishing the rain fall data into DynamoDB table by command:  
cd /deviceSDK/aws-iot-device-sdk-python  
python3 A0195042J\_PublishCode\_RaspiData.py

### 7.3) Verify inside DynamoDB



#### 7.4) Visualization at Tableau



## 8. Task3 – Self-monitor raspberry pi's cpu temperature and establish email alert on abnormal high temperature (temperature > 47.00DegC)

- 8.1) Similar to 7.2)-l, copy the script – A0195042J\_RaspiTemp\_Monitor.py and certificate, keys into folder /deviceSDK/aws-iot-device-sdk-python

```

Starting rs-tools.service...
Starting Systemd Sound Card Service...
Starting Login Services...
OK [  Started Sound Card Service (restore and store)
OK [  Started Sound Card State
OK [  Started Daily Cleanup of Temporary Directories...
OK [  Started Target Timers...
OK [  Started Target Network...
OK [  Started rs-tools...
OK [  Started Sound Restore Sound Card State...
OK [  Started Systemd Sound Card...
OK [  Started Target Sound Card...
OK [  Reached target Sound Card...
OK [  Started libnsls-DNS-SD Stack...
OK [  Started NetworkManager...
OK [  Started dhcpc-service - set up, mount/unmount, and delete a swap file...
OK [  Started gpioscrape - set up, mount/unmount, and delete a swap file...
OK [  Started raspberrypi...
OK [  Started triggerhappy global hotkey daemon...
OK [  Started triggerhappy...
OK [  Started Permit User Sessions...
OK [  Started Gettng on ttyp0...
OK [  Reached target Network...
OK [  Started OpenSSH Secure Shell server...
OK [  Started target Local Compatibility...
Starting raspi-temp...
IP address is 192.168.0.114
OK [  Started target Local Compatibility...
OK [  Started Permit User Sessions...
OK [  Started Gettng on ttyp0...
OK [  Reached target Network...
OK [  Started OpenSSH Secure Shell server...
raspberrypi login: pi
Password:
Last login: Sat Sep 21 10:56:17 +08 2019 on ttys0
Linux raspberrypi 4.19.66+ #1253 SMP Thu Aug 15 11:49:46 BST 2019 armv7l
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*-copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY; to the extent
permitted by applicable law.

You are currently logged in as the default password for the "pi" user has not been changed.
This is a security risk - please login as the "pi" user and type "passwd" to set a new password.

pi@raspberrypi: ~ cp -r /root/flash/ES111..._aws-iot-device-sdk-python
pi@raspberrypi: ~ mv aws-iot-device-sdk-python/RaspData.py awsboot01G1.py
pi@raspberrypi: ~ mv aws-iot-device-sdk-python/4225e1d79-certificate.pem.crt aws0195042j_PublishCode_RaspiData.py
pi@raspberrypi: ~ mv aws-iot-device-sdk-python/CHNGELOG.read NOTICE.txt
pi@raspberrypi: ~ mv aws-iot-device-sdk-python/4225e1d79-private.pem.key aws0195042j_RaspiTemp_Monitor.py
pi@raspberrypi: ~ mv aws-iot-device-sdk-python/4225e1d79-public.pem.key
pi@raspberrypi: ~ cd /aws-iot-device-sdk-python
pi@raspberrypi: ~ python3 A0195042J_PublishCode_RaspiData.py

```

- 8.2) Install library – gpiozero by command:  
sudo apt install python3-gpiozero

- 8.3) Direct to the folder and Run the script by command:  
cd /deviceSDK/aws-iot-device-sdk-python  
python3 A0195042J\_PublishCode\_RaspiData.py

```

[  OK ] Reached target Timers.
[  OK ] Started triggerhappy global hotkey daemon...
[  OK ] Started triggerhappy...
[  OK ] Started Sound Restore Sound Card State...
[  OK ] Started Sound Card...
[  OK ] Started Systemd Sound Card...
[  OK ] Started Target Sound Card...
[  OK ] Reached target Sound Card...
[  OK ] Started libnsls-DNS-SD Stack...
[  OK ] Started NetworkManager...
[  OK ] Started dhcpc-service - set up, mount/unmount, and delete a swap file...
[  OK ] Started gpioscrape - set up, mount/unmount, and delete a swap file...
[  OK ] Started raspberrypi...
[  OK ] Started triggerhappy global hotkey daemon...
[  OK ] Started triggerhappy...
[  OK ] Started Permit User Sessions...
[  OK ] Started Gettng on ttyp0...
[  OK ] Reached target Network...
[  OK ] Started OpenSSH Secure Shell server...
OK [  Started target Local Compatibility...
Starting raspi-temp...
IP address is 192.168.0.114
OK [  Started target Local Compatibility...
OK [  Started Permit User Sessions...
OK [  Started Gettng on ttyp0...
OK [  Reached target Network...
OK [  Started OpenSSH Secure Shell server...
raspberrypi login: pi
Password:
Last login: Sat Sep 21 10:56:17 +08 2019 on ttys0
Linux raspberrypi 4.19.66+ #1253 SMP Thu Aug 15 11:49:46 BST 2019 armv7l
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*-copyright.

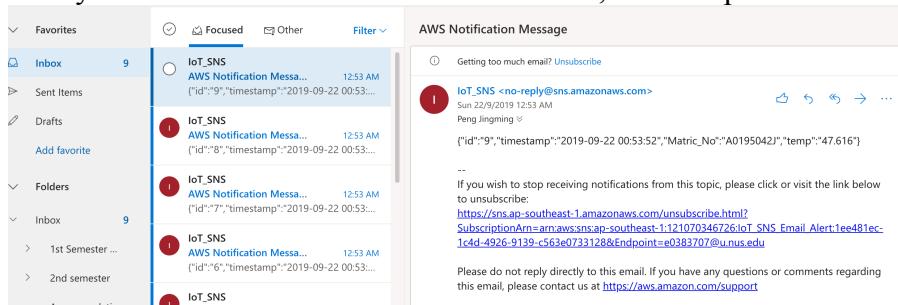
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY; to the extent
permitted by applicable law.

You are currently logged in as the default password for the "pi" user has not been changed.
This is a security risk - please login as the "pi" user and type "passwd" to set a new password.

pi@raspberrypi: ~ cp -r /root/flash/ES111..._aws-iot-device-sdk-python
pi@raspberrypi: ~ mv aws-iot-device-sdk-python/RaspData.py awsboot01G1.py
pi@raspberrypi: ~ mv aws-iot-device-sdk-python/4225e1d79-certificate.pem.crt aws0195042j_PublishCode_RaspiData.py
pi@raspberrypi: ~ mv aws-iot-device-sdk-python/CHNGELOG.read NOTICE.txt
pi@raspberrypi: ~ mv aws-iot-device-sdk-python/4225e1d79-private.pem.key aws0195042j_RaspiTemp_Monitor.py
pi@raspberrypi: ~ mv aws-iot-device-sdk-python/4225e1d79-public.pem.key
pi@raspberrypi: ~ cd /aws-iot-device-sdk-python
pi@raspberrypi: ~ python3 A0195042j_RaspiTemp_Monitor.py

```

- 8.4) Verify email alert inside school email account, screencapture as follow:



## 9. Discussion

The targeted AWS IoT functions (publish in and query out data from DynamoDB and establish SNS email alert) are realized by these 3 tasks.

There're more AWS services that could help to enhance user's IoT experience, and it's good worth of explore.

Beyond AWS platform, there're alternative IoT solution from other providers, such as Oracle, google, etc. Users are never limited within 1 solo IoT solution. However, cost, easiness, and effectiveness need to be carefully judged before deploying these technologies.

Another domain is about visualization. In this report, I am realizing it from tableau. AWS itself has similar solution, like QuickSight. To deploy this, no need to set up Rockset JDBC driver but have to go through S3 bucket. However, since it's not free, eventually I didn't choose it.

Tableau, as a powerful BI software, provides various ways of data visualization. And in this report, for illustration purpose, I only use very basic line-chart plot to show partial data visualization.

## 10. Appendixes:

### Task1 Script – Publish Engine Data into DynamoDB table

```
11. from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTShadowClient
12. import random, time, datetime
13.
14. # The unique hostname (can be found in setting) AWS IoT generates for this device.
15. HOST_NAME = "a1loa9sgjgf5z0-ats.iot.ap-southeast-1.amazonaws.com"
16.
17. # The relative path to the correct root CA file for AWS IoT,
18. ROOT_CA = "AmazonRootCA1.pem"
19.
20. # A random programmatic shadow client ID for both Things (engine1 and engine3)
21. SHADOW_CLIENT_1 = "A0195042J_EngineFD001"
22. SHADOW_CLIENT_2 = "A0195042J_EngineFD003"
23.
24. # The relative path to your private key file that AWS IoT generates for this
   device
25. PRIVATE_KEY_1 = "ecf87423da-private.pem.key"
26. PRIVATE_KEY_2 = "ef8e4b716b-private.pem.key"
27.
28. # The relative path to your certificate file that AWS IoT generates for this
   device
29. CERT_FILE_1 = "ecf87423da-certificate.pem.crt"
30. CERT_FILE_2 = "ef8e4b716b-certificate.pem.crt"
31.
32. # A programmatic shadow handler name prefix for both Things (engine1 and engine3)
33. SHADOW_HANDLER_1 = "A0195042J_EngineFD001"
34. SHADOW_HANDLER_2 = "A0195042J_EngineFD003"
35.
36. # ****
37. # Main script runs from here onwards.
38. # To stop running this script, press Ctrl+C.
39. # ****
40.
41. with open('train_FD001.txt','r') as infile_1:
42.     with open('FD001_out.txt','a') as outfile_1:
43.         # Due to process time issue, only read the 1st 5000 lines of data
44.         for line in infile_1.readlines()[0:5000]:
45.             outfile_1.write(line)
46.
47. with open('train_FD003.txt','r') as infile_2:
48.     with open('FD003_out.txt','a') as outfile_2:
49.         # Due to process time issue, only read the 1st 5000 lines of data
50.         for line in infile_2.readlines()[0:5000]:
51.             outfile_2.write(line)
52.
53. # Get the Data labels, later will be used as header of table inside DynamoDB
```

```

54. sensor_name = ['s'+ str(i) for i in range(1,22)]
55. dataLabels = ['id', 'timestamp', 'Matric_Number', 'cycle', 'os1', 'os2', 'os3'] +
   sensor_name
56.
57. Matric_Number = 'A0195042J'
58.
59. for i in range(0,len(dataLabels)):
60.     dataLabels[i] = '\"' + dataLabels[i] + '\"'
61.
62. process_1 = open("FD001_out.txt",'r')
63. process_2 = open("FD003_out.txt",'r')
64.
65. dataString_1 = []
66. dataString_2 = []
67. modifiedData_1 = []
68. modifiedData_2 = []
69.
70. head = '{"state":{"reported":{'
71. tail = '}}}'
72.
73. # Automatically called whenever the shadow is updated.
74. def myShadowUpdateCallback_1(payload, responseStatus, token):
75.     print()
76.     print('UPDATE: $aws/things/' + SHADOW_HANDLER_1 +
77.           '/shadow/update/#')
78.     print("payload = " + payload)
79.     print("responseStatus = " + responseStatus)
80.     print("token = " + token)
81.
82. # Create, configure, and connect a shadow client.
83. myShadowClient_1 = AWSIoTMQTTShadowClient(SHADOW_CLIENT_1)
84. myShadowClient_1.configureEndpoint(HOST_NAME, 8883)
85. myShadowClient_1.configureCredentials(ROOT_CA, PRIVATE_KEY_1,
86. CERT_FILE_1)
87. myShadowClient_1.configureConnectDisconnectTimeout(10)
88. myShadowClient_1.configureMQTTOperationTimeout(5)
89. myShadowClient_1.connect()
90.
91. # Create a programmatic representation of the shadow.
92. myDeviceShadow_1 = myShadowClient_1.createShadowHandlerWithName(
93. SHADOW_HANDLER_1, True)
94.
95. for x in process_1.readlines():
96.     newData_1 = x.split(" ")
97.     modifiedData_1 = []
98.     modifiedData_1.append(str('FD001_' + newData_1[0]))
99.     modifiedData_1.append(str(datetime.datetime.utcnow()))
100.    modifiedData_1.append(Matric_Number)

```

```

101.         for j in range(2,len(sensor_name)):
102.             modifiedData_1.append(newData_1[j])
103.
104.             ColumnLabels = []
105.             ColumnLabels.append(str(dataLabels[0] + ':'))
106.             ColumnLabels.append(str('"' + modifiedData_1[0] + '"', ''))
107.             ColumnLabels.append(str(dataLabels[1] + ':'))
108.             ColumnLabels.append(str('"' + str(datetime.datetime.now()) + '", ''))
109.             ColumnLabels.append(str(dataLabels[2] + ':'))
110.             ColumnLabels.append(str('"' + Matric_Number + '"', ''))
111.
112.
113.             for i in range(3,len(dataLabels)):
114.                 ColumnLabels.append(str(dataLabels[i] + ':'))
115.                 ColumnLabels.append(str('"' + newData_1[i-2] + '"', ''))
116.
117.                 string = ''.join(ColumnLabels)
118.                 string = string[:-1]
119.
120.                 data = []
121.                 data.append(head)
122.                 data.append(string)
123.                 data.append(tail)
124.                 data.append('\n')
125.                 dataString_1 = ''.join(data)
126.                 print(dataString_1)
127.
128.                 myDeviceShadow_1.shadowUpdate(dataString_1,myShadowUpdateCallback_1, 5)
129.                 # read the data each 1s
130.                 time.sleep(1)
131.
132.             # duplication for the second thing (engineFD003)
133.             # Automatically called whenever the shadow is updated.
134.             def myShadowUpdateCallback_2(payload, responseStatus, token):
135.                 print()
136.                 print('UPDATE: $aws/things/' + SHADOW_HANDLER_2 +
137.                     '/shadow/update/#')
138.                 print("payload = " + payload)
139.                 print("responseStatus = " + responseStatus)
140.                 print("token = " + token)
141.
142.             # Create, configure, and connect a shadow client.
143.             myShadowClient_2 = AWSIoTMQTTShadowClient(SHADOW_CLIENT_2)
144.             myShadowClient_2.configureEndpoint(HOST_NAME, 8883)
145.             myShadowClient_2.configureCredentials(ROOT_CA, PRIVATE_KEY_2,
146.                     CERT_FILE_2)
147.             myShadowClient_2.configureConnectDisconnectTimeout(10)
148.             myShadowClient_2.configureMQTTOperationTimeout(5)

```

```

149. myShadowClient_2.connect()
150.
151. # Create a programmatic representation of the shadow.
152. myDeviceShadow_2 = myShadowClient_2.createShadowHandlerWithName(
153.     SHADOW_HANDLER_2, True)
154.
155. for y in process_2.readlines():
156.     newData_2 = y.split(" ")
157.     modifiedData_2 = []
158.     modifiedData_2.append(str('FD003_' + newData_2[0]))
159.     modifiedData_2.append(str(datetime.datetime.utcnow()))
160.     modifiedData_2.append(Metric_Number)
161.     for k in range(2,len(sensor_name)):
162.         modifiedData_2.append(newData_2[k])
163.
164.     ColumnLabels = []
165.     ColumnLabels.append(str(dataLabels[0] + ':'))
166.     ColumnLabels.append(str('' + modifiedData_2[0] + '', ''))
167.     ColumnLabels.append(str(dataLabels[1] + ':'))
168.     ColumnLabels.append(str('' + str(datetime.datetime.now()) + "", ''))
169.     ColumnLabels.append(str(dataLabels[2] + ':'))
170.     ColumnLabels.append(str('' + Metric_Number + "", ''))
171.
172.     for l in range(3,len(dataLabels)):
173.         ColumnLabels.append(str(dataLabels[l] + ':'))
174.         ColumnLabels.append(str('' + newData_2[l-2] + "", ''))
175.
176.     string = ''.join(ColumnLabels)
177.     string = string[:-1]
178.
179.     data = []
180.     data.append(head)
181.     data.append(string)
182.     data.append(tail)
183.     data.append('\n')
184.     dataString_2 = ''.join(data)
185.     print(dataString_2)
186.
187.     myDeviceShadow_2.shadowUpdate(dataString_2,myShadowUpdateCallback_2, 5)
188.     # read the data each 1s
189.     time.sleep(1)

```

## Task2 Script – Publish Raspi rainfall data into DynamoDB table

```
190. from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTShadowClient
191. from datetime import datetime
192. import csv
193. import os
194. import json
195.
196. # A random programmatic shadow client ID.
197. SHADOW_CLIENT = "A0195042J_Raspi"
198.
199. # The unique hostname that &IoT; generated for
200. # this device.
201. HOST_NAME = "a1loa9sgjgf5z0-ats.iot.ap-southeast-1.amazonaws.com"
202.
203. # The relative path to the correct root CA file for &IoT;;,
204. # which you have already saved onto this device.
205. ROOT_CA = "AmazonRootCA1.pem"
206.
207. # The relative path to your private key file that
208. # &IoT; generated for this device, which you
209. # have already saved onto this device.
210. PRIVATE_KEY = "4d2dec1d7b-private.pem.key"
211.
212. # The relative path to your certificate file that
213. # &IoT; generated for this device, which you
214. # have already saved onto this device.
215. CERT_FILE = "4d2dec1d7b-certificate.pem.crt"
216.
217. # A programmatic shadow handler name prefix.
218. SHADOW_HANDLER = "A0195042J_Raspi"
219.
220. # Automatically called whenever the shadow is updated.
221. def myShadowUpdateCallback(payload, responseStatus, token):
222.     print()
223.     print('UPDATE: aws/things/' + SHADOW_HANDLER +
224.           '/shadow/update/#')
225.     print("payload = " + payload)
226.     print("responseStatus = " + responseStatus)
227.     print("token = " + token)
228.
229. # Create, configure, and connect a shadow client.
230. myShadowClient = AWSIoTMQTTShadowClient(SHADOW_CLIENT)
231. myShadowClient.configureEndpoint(HOST_NAME, 8883)
232. myShadowClient.configureCredentials(ROOT_CA, PRIVATE_KEY,
233.                                     CERT_FILE)
234. myShadowClient.configureConnectDisconnectTimeout(10)
235. myShadowClient.configureMQTTOperationTimeout(5)
```

```

236. myShadowClient.connect()
237.
238. # Create a programmatic representation of the shadow.
239. myDeviceShadow = myShadowClient.createShadowHandlerWithName(
240.     SHADOW_HANDLER, True)
241.
242. #object of CSV file
243. f = 'rainfall-monthly-total.csv'
244.
245. #Initialize weather data with 2 keyword: 'daq_time' & 'rain'
246. WeatherData = {'daq_time':[], 'rain':[]}
247.
248. #open rainfall monthly total csv file from directory (same directory where this script is
249. stored)
250. with open(os.path.join(os.getcwd(),f)) as csvfile:
251.     readCSV = csv.reader(csvfile,delimiter=',')
252.     #skip header
253.     next(readCSV)
254.
255.     for row in readCSV:
256.         WeatherData['daq_time'].append(row[0])
257.         WeatherData['rain'].append(row[1])
258.
259. #length of data
260. Length = len(WeatherData['daq_time'])
261. #number of digit in Length
262. Num = len(str(abs(Length)))
263.
264. #previous time
265. pt = datetime.now()
266. #hold time = 1s
267. delay = 1.0
268.
269. Matric_Number = 'A0195042J'
270.
271. for i in range(int(Length/2),Length):
272.     #current time
273.     ct = datetime.now()
274.     #timestamp
275.     timestamp = str(datetime.utcnow())
276.     id = str(i+1).zfill(Num)
277.     temp = {
278.         "state": {
279.             "reported": {
280.                 "id": id,
281.                 "timestamp": timestamp,
282.                 "Matric_No": Matric_Number,
283.                 "DaqDate": str(WeatherData['daq_time'][i]),

```

```

283.                 "RainFall": str(WeatherData['rain'][i])
284.             }
285.         }
286.     }
287.
288.     #create Json string
289.     msg = json.dumps(temp)
290.
291.     #publish Json message to AWS IoT device shadow
292.     myDeviceShadow.shadowUpdate(msg,myShadowUpdateCallback,5)
293.     #wait for 5s
294.
295.     while ((ct-pt).total_seconds() <= delay):
296.         #update current time
297.         ct = datetime.now()
298.         #update previous time
299.         pt = ct
300.

```

### Task3 script – raspberry pi self-monitoring temperature data for email alert

```

from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTShadowClient
from gpiozero import CPUTemperature
from datetime import datetime
from time import sleep, strftime, time
import os
import csv
import json

# A random programmatic shadow client ID.
SHADOW_CLIENT = "A0195042J_Raspi_Monitor"

# The unique hostname that &IoT; generated for
# this device.
HOST_NAME = "a1loa9sgjgf5z0-ats.iot.ap-southeast-1.amazonaws.com"

# The relative path to the correct root CA file for &IoT;;
# which you have already saved onto this device.
ROOT_CA = "AmazonRootCA1.pem"

# The relative path to your private key file that
# &IoT; generated for this device, which you
# have already saved onto this device.
PRIVATE_KEY = "a9589dcfe7-private.pem.key"

# The relative path to your certificate file that
# &IoT; generated for this device, which you
# have already saved onto this device.

```

```

CERT_FILE = "a9589dcfe7-certificate.pem.crt"

# A programmatic shadow handler name prefix.
SHADOW_HANDLER = "A0195042J_Raspi_Monitor"

# Automatically called whenever the shadow is updated.
def myShadowUpdateCallback(payload, responseStatus, token):
    print()
    print('UPDATE: $aws/things/' + SHADOW_HANDLER +
          '/shadow/update/#')
    print("payload = " + payload)
    print("responseStatus = " + responseStatus)
    print("token = " + token)

# Create, configure, and connect a shadow client.
myShadowClient = AWSIoTMQTTShadowClient(SHADOW_CLIENT)
myShadowClient.configureEndpoint(HOST_NAME, 8883)
myShadowClient.configureCredentials(ROOT_CA, PRIVATE_KEY,
                                   CERT_FILE)
myShadowClient.configureConnectDisconnectTimeout(10)
myShadowClient.configureMQTTOperationTimeout(5)
myShadowClient.connect()

# Create a programmatic representation of the shadow.
myDeviceShadow = myShadowClient.createShadowHandlerWithName(
    SHADOW_HANDLER, True)

# create object - cpu
cpu = CPUTemperature()

# initialize count
count = 0
# metric number
Metric_Number = 'A0195042J'

#create the log csv file
#with open('/deviceSDK/aws-iot-device-sdk-python/Raspi_Temp.csv','a') as log:
with open('Raspi_Temp.csv','a') as log:
    while count < 20:
        count = count + 1

        temp = cpu.temperature

        now = datetime.now()
        date_time = now.strftime('%Y-%m-%d %H:%M:%S')
        log.write('{0},{1}\n'.format(date_time,str(temp)))
        sleep(1)

```

```
msg = {
    "state": {
        "reported": {
            "id": str(count),
            "timestamp": date_time,
            "Matric_No": Matric_Number,
            "temp": str(temp)
        }
    }
}
myDeviceShadow.shadowUpdate(json.dumps(msg),myShadowUpdateCallback,5)
# rest for 5s before next data collection
sleep(5)
```