

# **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**

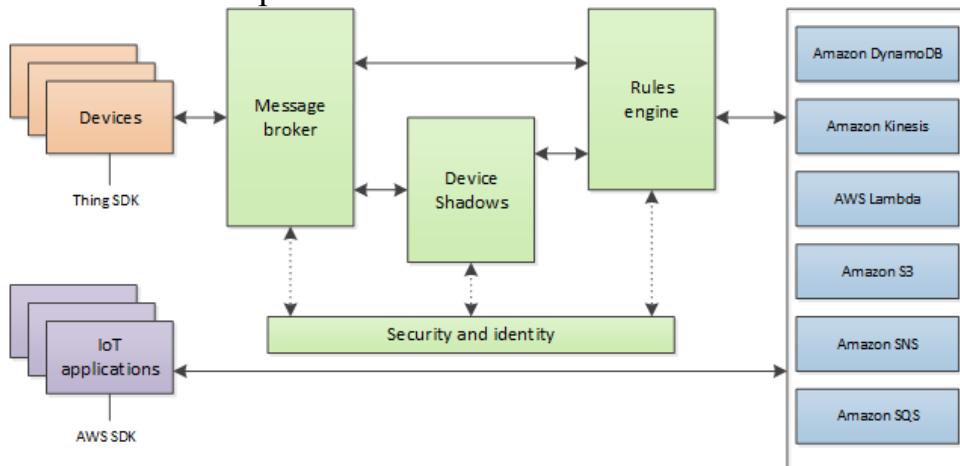
## Table of Contents

1. Project Description
2. AWS IoT Concept
3. AWS IoT Realization
4. Visualization Concept
5. Visualization Realization
6. Task1
7. Task2
8. Discuss
9. 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.

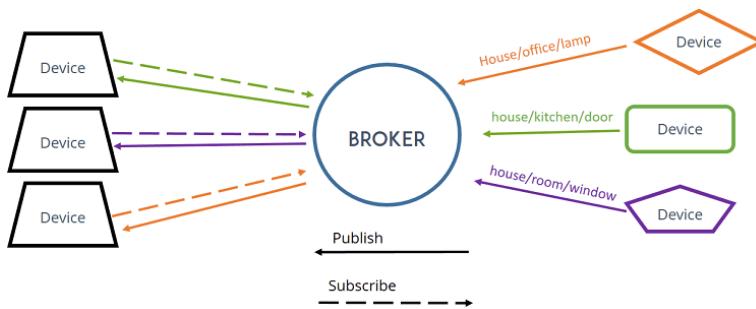
Main components:

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 lists navigation options: Details, Security, Thing Groups, Billing Groups, Shadow (which is selected and highlighted in blue), 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", }}
```

2.5) Rule engine – It's where rules are created. The rules define corresponding actions to perform based on the data in a message. When a rule matches a message, the rules engine triggers the action using the selected properties. Rules also contain an IAM role that grants AWS IoT permission to the AWS resources used to perform the action. An AWS IoT rule consists of an SQL SELECT statement, a topic filter, and a rule action:

The SQL SELECT statement allows you to extract data from an incoming MQTT message;

The topic filter specifies one or more MQTT topics. The rule is triggered when an MQTT

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 IAM 'Create role' wizard, Step 2: Select type of trusted entity.

The 'Create role' dialog is open, showing the 'Select type of trusted entity' step. A red arrow points to the 'AWS service' option under 'AWS service'. Below it, a note says: 'Allows entities in other accounts to perform actions in this account. Learn more'.

The 'Specify accounts that can use this role' step follows, with an 'Account ID\*' input field containing a redacted value. Under 'Options', there are two checkboxes: 'Require external ID' (unchecked) and 'Require MFA' (unchecked). A red arrow points to the 'Require external ID' checkbox.

The 'Create role' step is shown next, with a red arrow pointing to the 'AdministratorAccess' policy in the list. The 'Next: Permissions' button is highlighted with a red box.

Create role

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:**

Maximum 1000 characters. Use alphanumeric and '+-=.\_-' characters.

**Trusted entities:** The account ~~XXXXXXXXXX~~

**Policies:**  AdministratorAccess \*

\* Required

**Create role** (highlighted)

- 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 image contains two screenshots of the AWS Management Console.

The top screenshot shows the "Create role" wizard step 4, "Review". It displays the role name "Administrator", a blank role description, and a selected policy "AdministratorAccess". The "Create role" button is highlighted with a red arrow.

The bottom screenshot shows the AWS Services dashboard. The navigation bar indicates the user is in the "Services" section, specifically under "Resource Groups", and the region is set to "Singapore". A dropdown menu for regions is open, with "Asia Pacific (Singapore)" highlighted with a red arrow. Other regions listed include US East (N. Virginia), US East (Ohio), US West (N. California), US West (Oregon), Asia Pacific (Hong Kong), Asia Pacific (Mumbai), Asia Pacific (Seoul), and Asia Pacific (Sydney).

- 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:

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.

Manage

Monitor

Onboard

Greengrass

Secure

Defend

Act

Test

Software

Settings

Learn

See how AWS IoT works

Explore an interactive tutorial through the components of AWS IoT.

Start the tutorial

It takes 5 minutes

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.

This screenshot shows the first step of creating a new thing in the AWS IoT console. The title bar says 'CREATE A THING' and 'Add your device to the thing registry'. In the top right corner, it says 'STEP 1/3'. Below the title, there is a note: 'This step creates an entry in the thing registry and a thing shadow for your device.' A 'Name' input field contains the value 'A0195042J\_Raspi\_Monitor'. The background is light blue.

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

This screenshot shows the second step of creating a certificate. The title bar says 'CREATE A THING' and 'Add a certificate for your thing'. In the top right corner, it says 'STEP 2/3'. Below the title, there is a note: 'A certificate is used to authenticate your device's connection to AWS IoT.' There are four options: 1) 'One-click certificate creation (recommended)' which is highlighted with a red border; 2) 'Create with CSR'; 3) 'Use my certificate'; 4) 'Skip certificate and create thing'. Each option has a corresponding 'Create certificate' button. The 'Create certificate' button for the recommended option is also highlighted with a red border. The background is light blue.

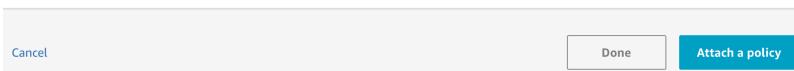
- f. On the **Certificate created!** page, download your public and private keys, certificate, and root certificate authority (CA):  
f.1) Choose **Download** for your certificate.  
f.2) Choose Download for your private key.  
f.3) Choose Download for the Amazon root CA. A new webpage is displayed. 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	<a href="#">Download</a>
A public key	a9589dcfe7.public.key	<a href="#">Download</a>
A private key	a9589dcfe7.private.key	<a href="#">Download</a>

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

[Activate](#)



#### 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.

CREATE A THING  
**Add a policy for your thing**

STEP  
3/3

Select a policy to attach to this certificate:

[Search policies](#)

[A0195042J\\_Raspi\\_Policy](#) [View](#)

[A0195042J\\_EngineFD\\_Policy](#) [View](#)

0 policies selected

[Register Thing](#)

#### 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:

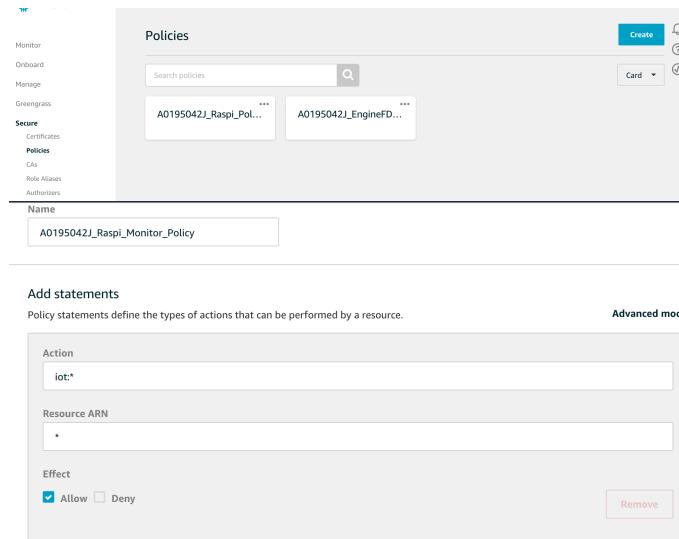
For task1 - **A0195042J\_EngineFD\_Policy**

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**.



- 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**.

## Attach policies to certificate(s)

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

a9589dcfe7bba338318ccbedaaec512e5c3d26227a9f65ea8c230d9b1ccfac6

Choose one or more policies

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>

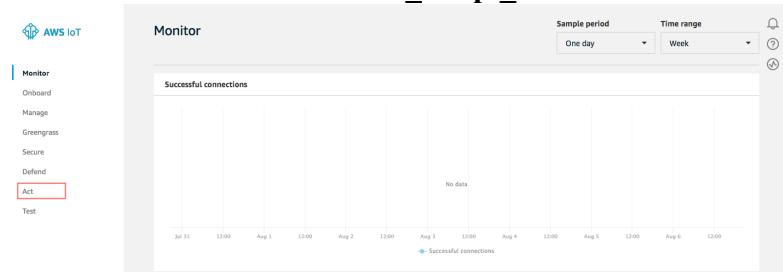
1 policy selected [Cancel](#) [Attach](#)

### 3.6) Configure and test the rules for publish data into DynamoDB table

- a. To publish the data from the device/Thing into DynamoDB table, the rule splits the data into columns of a DynamoDB table.
  - a.1) In the AWS IoT console, in the left navigation pane, choose **Act**.
  - a.2) On the **Act** page, choose **Create a rule**.
  - a.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](#)

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\_EngineFD001\_Rule

Description  
publish engine1 data and split it into columns in DynamoDB table

- b. 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_EngineFD001/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

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

Tutorial



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

String

Add sort key

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”:



### 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

A0195042J\_Engine

Create a new resource

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

A0195042J\_EngineFD001\_Rule

Update Role

Create Role

Select

Cancel

Add action

- 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:

**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
```

In general, they are:

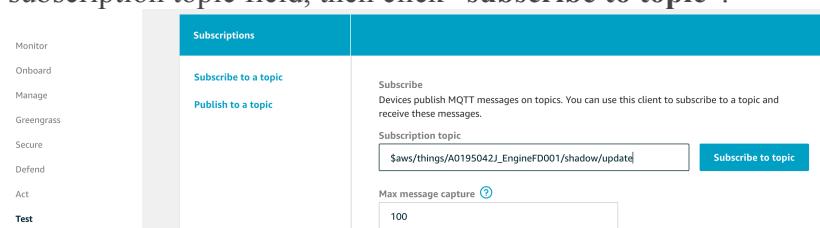
\$aws/things/A0195042J\_EngineFD001/shadow/update

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

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

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

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:

The screenshot shows the AWS IoT Device Shadows interface. On the left, there's a sidebar with 'Subscriptions' and a list of topics. One topic, '\$aws/things/A0195042J\_EngineFD001/shadow/update', is selected and highlighted in blue. On the right, there's a 'Publish' section with a text input field containing the topic '\$aws/things/A0195042J\_EngineFD001/shadow/update'. Below the input field is a code editor window showing the following JSON message:

```

1 {
2   "message": "Hello from AWS IoT console"
3 }

```

At the bottom right of the publish section is a blue 'Publish to topic' button.

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 Device Shadows interface. On the left, there's a sidebar with 'Subscriptions' and a list of topics. One topic, '\$aws/things/A0195042J\_EngineFD001/shadow/update/accepted', is selected and highlighted in blue. On the right, there's a 'Publish' section with a text input field containing the topic '\$aws/things/A0195042J\_EngineFD001/shadow/update/accepted'. Below the input field is a code editor window showing the same JSON message as before:

```

1 {
2   "message": "Hello from AWS IoT console"
3 }

```

At the bottom right of the publish section is a blue 'Publish to topic' button.

Below the publish section, there's a log table with one entry:

\$aws/things/A0195042J_EngineFD001/shadow...	Sep 20, 2019 5:45:17 PM +0800	Exp... Hi...
{ "state": { "desired": { "message": null }, } }		

```

    "reported": {
      "message": null,
      "id": "FD001_1",
      "timestamp": "timestamp",
      "Matriic_Number": "A0195042J",
      "cycle": "1",
      "os1": "01",
      "os2": "02",
      "os3": "03"
    },
    "metadata": {
      "desired": {
        "message": {
          "timestamp": 1568972717
        }
      },
      "reported": {
        "message": {
          "timestamp": 1568972717
        },
        "id": {
          "timestamp": 1568972717
        },
        "os": {
          "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. On the left, there's a navigation sidebar with links for Dashboard, Topics (which is selected and highlighted in orange), Subscriptions, Mobile (with Push notifications and Text messaging (SMS)), and AWS Lambda. The main content area has a breadcrumb trail: Amazon SNS > Topics. It displays a table titled 'Topics (1)' with one entry: 'Name' (dynamodb) and 'ARN' (arn:aws:sns:ap-southeast-1:121070346726:dynamodb). There are buttons for Edit, Delete, Publish message, and Create topic.

- 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' dialog and the 'IoT\_SNS\_Email\_Alert' topic details page.

**Create topic Dialog:**

- Details:**
  - Name:** IoT\_SNS\_Email\_Alert
  - Display name - optional:** IoT\_SNS
- A success message at the bottom: **Topic IoT\_SNS\_Email\_Alert created successfully.** You can create subscriptions and send messages to them from this topic.

**IoT\_SNS\_Email\_Alert Topic Details:**

Details	
<b>Name:</b> IoT_SNS_Email_Alert	<b>Display name:</b> IoT_SNS
<b>ARN:</b> arn:aws:sns:ap-southeast-1:121070346726:IoT_SNS_Email_Alert	<b>Topic owner:</b> 121070346726

The screenshot shows the AWS SNS Subscriptions page. At the top, there are tabs for Subscriptions, Access policy, Delivery retry policy (HTTP/S), Delivery status logging, and Encryption. Below the tabs is a 'Tags' section. The main area is titled 'Subscriptions (0)' and contains a table with the following columns: ID, Endpoint, Status, and Protocol. The table displays the message 'No subscriptions found'. Below the table, a note says 'You don't have any subscriptions to this topic.' and a 'Create subscription' button is visible.

- 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**:

The screenshot shows the 'Create subscription' dialog box. It has a 'Details' tab. Under 'Topic ARN', the value is 'arn:aws:sns:ap-southeast-1:121070346726'. Under 'Protocol', the dropdown is set to 'Email'. Under 'Endpoint', the value is 'e0383707@u.nus.edu'. At the bottom of the dialog, there is a note: 'After your subscription is created, you must confirm it.' with a link to '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:

The screenshot shows an email from 'IoT\_SNS <no-reply@sns.amazonaws.com>' received on 'Fri 20/9/2019 9:18 PM' from 'Peng Jingming'. The subject is 'AWS Notification - Subscription Confirmation'. The message body starts with 'You have chosen to subscribe to the topic: arn:aws:sns:ap-southeast-1:121070346726:IoT\_SNS\_Email\_Alert'. It then instructs the recipient to 'To confirm this subscription, click or visit the link below (if this was in error no action is necessary):' followed by a blue 'Confirm subscription' link. At the bottom, there is a note: 'Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)'.

**Subscription confirmed!**

You have subscribed e0383707@u.nus.edu to the topic:  
**IOT\_SNS\_Email\_Alert**.

Your subscription's id is:  
arn:aws:sns:ap-southeast-1:121070346726:IoT\_SNS\_Email\_Alert:1ee481ec-1c4d-4926-9139-c563e0733128

If it was not your intention to subscribe, [click here to unsubscribe](#).

ID	Endpoint	Status	Protocol	Topic
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 > 50
```

Rule query statement

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

Using SQL version

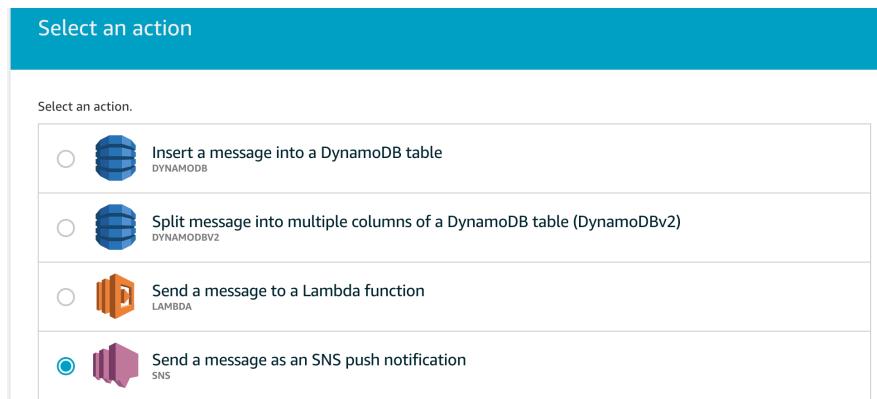
2016-03-23

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 * FROM '$aws/things/ A0195042J_Raspi_Monitor/shadow/update/accepted' WHERE state.reported.temp > 50
```

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



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 **JSON**.

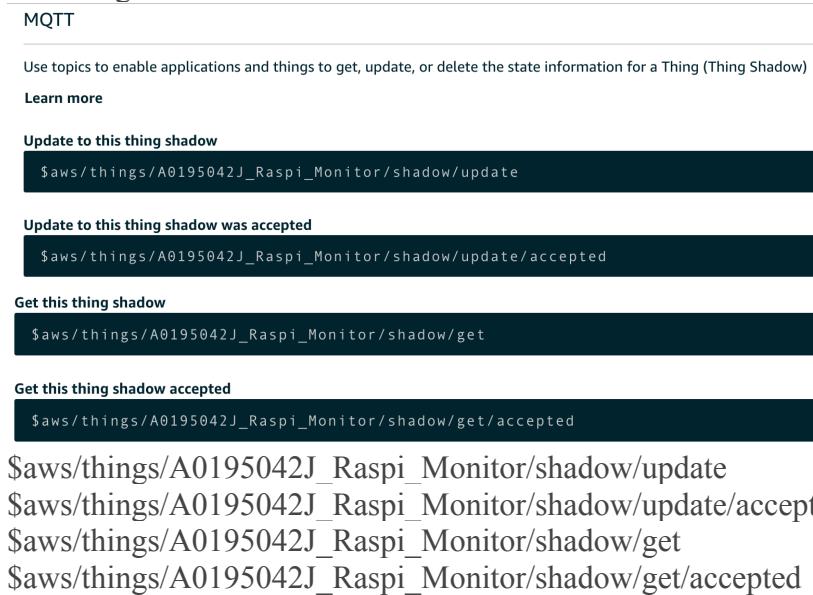
Choose **Create a new role**, give the name - **A0195042J\_Raspi\_AbnTemp\_Rule**:

\*SNS target  
IoT\_SNS\_Email\_Alert      Create      Clear      Select

Message format  
JSON

Choose or create a role to grant AWS IoT access to perform this action.  
A0195042J\_Raspi\_AbnTemp\_Rule      Policy Attached      Create Role      Select

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



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": {
      "message": null,
      "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:

```
$aws/things/A0195042J_Raspi_Monitor/shad... Sep 20, 2019 11:47:11 PM +0800 Exp... Hi...
{
  "state": {
    "desired": {
      "message": null
    },
    "reported": {
      "message": null,
      "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:

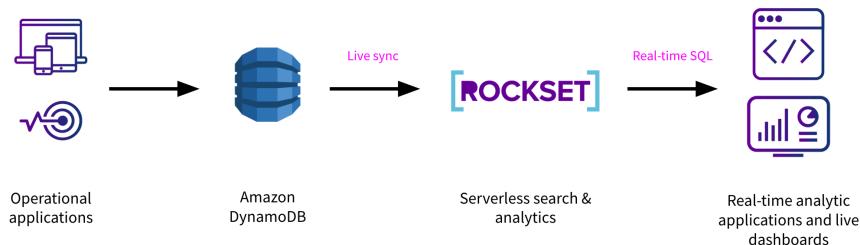
	Task1		Task2
Thing	A0195042J_EngineFD001	A0195042J_EngineFD003	A0195042J_Raspi
X.509 Certificate	ecf87423da-certificate.pem.crt	ef8e4b716b-certificate.pem.crt	4d2dec1d7b-certificate.pem.crt
Private Key	ecf87423da-private.pem.key	ef8e4b716b-private.pem.key	4d2dec1d7b-private.pem.key
Public Key	ecf87423da-public.pem.key	ef8e4b716b-public.pem.key	4d2dec1d7b-public.pem.key
root CA	AmazonRootCA1.pem	AmazonRootCA1.pem	AmazonRootCA1.pem
Policy	A0195042J_EngineFD_Policy	A0195042J_EngineFD_Policy	A0195042J_Raspi_Policy
Rule	A0195042J_EngineFD001_Rule	A0195042J_EngineFD003_Rule	A0195042J_Raspi_Rule
Table	A0195042J_Engine	A0195042J_Engine	A0195042J_Raspi

## 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 is:  
For task 1 - **A0195042J\_Engine**;  
For task 2 - **A0195042J\_Raspi**;

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:Scan",
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator",
        "dynamodb:DescribeTable"
      ],
      "Resource": [
        "arn:aws:dynamodb:*:*:table/A0195042J_Engine",
        "arn:aws:dynamodb:*:*:table/A0195042J_Engine/stream/*",
        "arn:aws:dynamodb:*:*:table/A0195042J_Raspi",
        "arn:aws:dynamodb:*:*:table/A0195042J_Raspi/stream/*"
      ]
    }
  ]
}
```

Policies > Rockset\_AWS

**Summary**

**Policy ARN** arn:aws:iam::121070346726:policy/Rockset\_AWS

**Description** Rockset can read-only from DynamoDB

**Permissions** **Policy usage** **Policy versions** **Access Advisor**

**Policy summary** **{ JSON** **Edit policy** **?**

```
11
12
13 ]
14
15
16
17
18
19
  "Resource": [
    "arn:aws:dynamodb:*:*:table/A0195042J_Engine",
    "arn:aws:dynamodb:*:*:table/A0195042J_Engine/stream/*",
    "arn:aws:dynamodb:*:*:table/A0195042J_Raspi",
    "arn:aws:dynamodb:*:*:table/A0195042J_Raspi/stream/*"
```

- d. Save this newly created policy:

**Identity and Access Management (IAM)**

**AWS Account (121070346726)**

- Dashboard
- Groups
- Users
- Roles
- Policies**
- Identity providers
- Account settings
- Credential report

**Create policy** **Policy actions** **?**

**Filter policies** Q: Rock

Policy name	Type	Used as	Description
Rockset_AWS	Customer managed	Permissions policy (1)	Rockset can read-only from DynamoDB

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

- Create a new user by navigating to **Users** and clicking **Add User - Administrator**
- 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 in progress, specifically Step 2: Set permissions. A red oval highlights the 'Attach existing policies directly' button, which is highlighted in blue. Below it is a table listing various AWS managed policies. At the bottom of the table are 'Cancel', 'Previous', and 'Next: Tags' buttons.

Policy name	Type	Used as	Description
AlexaForBusinessDeviceSetup...	AWS managed	None	Provide device setup access to AlexaForBusiness...
AlexaForBusinessFullAccess...	AWS managed	None	Grants full access to AlexaForBusiness resour...
AlexaForBusinessGateway...	AWS managed	None	Provide gateway execution access to Alexa...
AlexaForBusinessReadOnly...	AWS managed	None	Provide read only access to AlexaForBusine...
AmazonAPIGatewayInvoke...	AWS managed	None	Provides full access to create/edit/delete A...
AmazonAPIGatewayInvoke...	AWS managed	None	Provides full access to invoke APIs in Amaz...

- 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' section for the newly created user. It displays the 'Access key ID' (AKIARYMC24HTPX5F6XH2) and 'Secret access key' (which is partially obscured). A note at the bottom encourages frequent key rotation.

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

The screenshot shows the Rockset Console interface. On the left is a sidebar with navigation links: Overview, Collections, Query, Catalog, Manage (with sub-links Integrations, Users, API Keys, Billing), and a search bar. The main area is titled 'New Collection'. It has a sub-header 'Select your Data Source' with a note: 'The data source you choose will be used to load data into your Rockset collection. Don't see the source you're looking for? Tell us about it.' Below this is a search bar and a grid of icons representing different data sources: Amazon DynamoDB, Amazon Kinesis, Amazon Redshift, Amazon S3, Apache Kafka, File Upload (Beta), Google Cloud Storage, and Sample Datasets.

b. Choose integration - Rock\_AWS and then Start

This screenshot continues the 'New Collection' process. The 'Integrations' section is shown, listing 'Rockset\_AWS' as the selected integration, created by 'kevinming2012@gmail.com' on 'Sep 8'. Below this is a 'Create Integration' button. At the bottom are 'Back' and 'Start' buttons, with 'Start' being highlighted.

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

The 'Create Collection' screen is shown. For task1, the collection name is 'A0195042J\_DynamoDB\_Engine' and the table name is 'A0195042J\_Engine'. For task2, the collection name is 'A0195042J\_DynamoDB\_Raspi' and the table name is 'A0195042J\_Raspi'. The 'AWS Region' is set to 'ap-southeast-1'. The 'Source Preview' table shows sample data from the DynamoDB table, including columns 'DaqDate', 'Metric\_No', 'RainFall', and '\_meta'. At the bottom are 'Back' and 'Create' buttons, and the 'Create' button is highlighted.

Below this, the 'Collections' list shows two collections: 'A0195042J\_DynamoDB\_Engine' and 'A0195042J\_DynamoDB\_Raspi', both in a 'Ready' status. A 'Create Collection +' button is also visible.

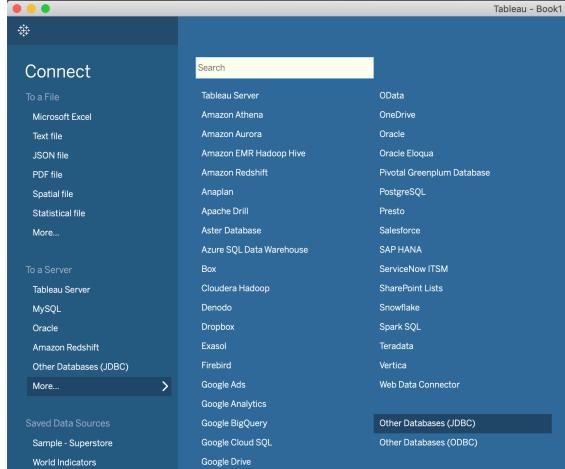
5.4) Set up Tableau Desktop and configure it for use with Rockset.  
Tableau Desktop uses JDBC driver to perform operations.  
JDBC support is available in Tableau Desktop version 2019.1 (or later) and Rockset Java Client version 0.5.11 (or later).

- 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):



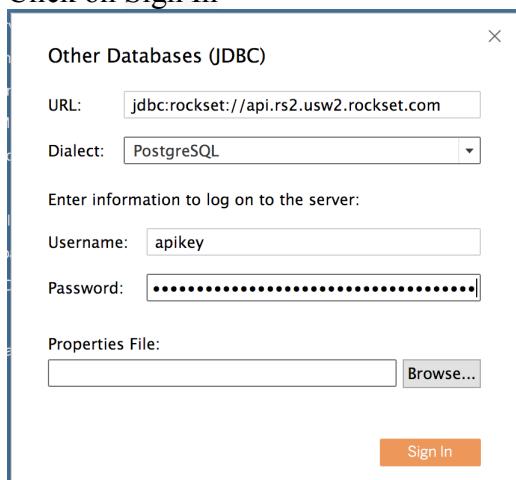
- Configure the connection as follows:

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:** `*****`
- Properties File:**  [Browse...](#)
- Sign In** button

DynamoDB Table - A0195042J\_Engine inside Tableau

Tableau - Book1

Connections

- jdbc:rockset://w2.rockset.com
- Other Databases (0 items)

Add

AO195042J\_DynamoDB\_Engine (commons)

Connection

- Live
- Extract

Filters

Add

Database

rockset

Schema

commons

Table

\_events

AO195042J\_-oDB\_Engine

AO195042J\_-moDB\_Rsp

New Custom SQL

AO195042J\_DynamoDB\_Engine

Sort Fields Data source order

Show aliases Show hidden fields 1,000 ↴ rows

| No.                       |
|---------------------------|---------------------------|---------------------------|---------------------------|---------------------------|---------------------------|---------------------------|---------------------------|---------------------------|---------------------------|---------------------------|---------------------------|
| AO195042J_DynamoDB_Eng... |
_Event Time	_Id	Matric Number	_Meta	Cycle	Id	Message	Os1	Os2			
2019-09-08T20:10...	1339f9ff-47bf-497...	A0195042J	194	F0003_15	null	0.0019	-0.0000				
2019-09-08T20:10...	e0cd3784-49d1-4wf8...	A0195042J	118	F0001_13	null	-0.0022	0.0002				
2019-09-08T20:10...	1339f9ff-47bf-497...	A0195042J	159	F0001_11	null	-0.0037	-0.0003				
2019-09-08T20:10...	1339f9ff-47bf-497...	A0195042J	163	F0001_4	null	-0.0041	-0.0002				
2019-09-08T20:10...	bff4c43-08af-400...	A0195042J	146	F0001_B	null	0.0001	-0.0004				
2019-09-08T20:10...	21659e6f-4472-412...	A0195042J	148	F0003_15	null	0.0012	0.0001				
2019-09-08T20:10...	9350207-5645-093...	A0195042J	154	F0001_13	null	-0.0003	-0.0003				
2019-09-08T20:10...	6bd5cb0-638b-265...	A0195042J	35	F0004_A	null	0.0002	0.0004				
2019-09-08T20:10...	6bd5cb0-638b-265...	A0195042J	191	F0003_15	null	-0.0026	-0.0001				

Go to Worksheet X

## DynamoDB Table - A0195042J\_Raspi inside Tableau

The screenshot shows the Rockset interface with a query results page. The top navigation bar includes 'Connections' (Add), 'Live' (Extract), and 'Filters' (Add). The main area displays a table titled 'A0195042J\_DynamoDB\_Rarsi (commons)'. The table has columns: 'Abc', 'Meta', 'Id', and 'Timestamp'. The data rows show various entries corresponding to the schema. A secondary table at the bottom lists 'events' from 'A0195042J...oDB\_Engine' and 'A0195042J...moDB\_Rarsi'. A 'New Custom SQL' button is also visible.

## 6. Task 1

## 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:

train_F0001.txt	
1	-0.0007 -0.0004 189.0 518.67 641.82 1589.79 1488.69 14.62 21.61 554.36 2388.06 9846.19 13.47 47.52 61.66 2388.02 8138.62 8.4195 8.93 392 2388 180.0 39.06 23.4198
2	0.0019 -0.0004 189.0 518.67 642.15 1591.87 1434.14 14.62 21.61 553.75 2388.04 9844.07 13.47 47.52 62.28 2388.07 8133.49 8.4318 0.03 392 2388 180.0 39.06 23.4236
3	-0.0043 -0.0003 189.0 518.67 642.35 1589.79 1484.06 14.62 21.61 554.28 2388.05 9852.94 13.47 47.52 62.42 2388.03 8133.23 14.0783 0.03 392 2388 180.0 38.95 23.3442
4	-0.0019 -0.0004 189.0 518.67 642.45 1589.79 1484.06 14.62 21.61 554.28 2388.05 9852.94 13.47 47.52 62.42 2388.03 8133.23 14.0783 0.03 392 2388 180.0 38.95 23.3442
5	-0.0019 -0.0004 189.0 518.67 642.47 1586.22 1446.22 14.62 21.61 554.00 2388.05 9849.61 13.47 47.52 61.29 2388.04 8133.80 8.4294 0.03 392 2388 180.0 39.38 23.4044
6	-0.0043 -0.0003 189.0 518.67 642.47 1584.47 1398.37 14.62 21.61 554.67 2388.02 9849.68 13.47 47.51 62.18 2388.03 8132.85 8.4188 0.03 391 2388 180.0 38.95 23.3669
7	-0.0019 -0.0004 189.0 518.67 642.47 1584.47 1398.37 14.62 21.61 554.67 2388.02 9849.68 13.47 47.51 62.18 2388.03 8132.85 8.4188 0.03 391 2388 180.0 38.95 23.3669
8	-0.0034 -0.0003 189.0 518.67 642.56 1586.92 1446.97 14.62 21.61 553.85 2388.06 9846.48 13.47 47.52 62.42 2388.03 8131.87 8.4076 0.03 391 2388 180.0 38.97 23.2186
9	0.0008 -0.0001 189.0 518.67 642.62 1598.79 1494.98 14.62 21.61 553.69 2388.05 9846.41 13.47 47.52 62.19 2388.05 8131.87 6.6972 0.03 392 2388 180.0 39.05 23.4866
10	-0.0007 -0.0004 189.0 518.67 642.62 1598.79 1494.98 14.62 21.61 553.69 2388.05 9846.41 13.47 47.52 62.19 2388.05 8131.87 6.6972 0.03 392 2388 180.0 39.05 23.4866
11	-0.0019 -0.0003 189.0 518.67 642.68 1581.75 1498.64 14.62 21.61 554.54 2388.05 9849.61 13.47 47.51 62.18 2388.01 8114.48 8.4340 0.03 392 2388 180.0 38.94 23.4737
12	0.0016 -0.0002 189.0 518.67 642.68 1581.75 1498.64 14.62 21.61 554.54 2388.09 9849.67 13.47 47.51 62.18 2388.06 8134.25 3.3930 0.03 392 2388 180.0 38.99 23.3660
13	-0.0019 -0.0003 189.0 518.67 642.68 1581.75 1498.64 14.62 21.61 554.54 2388.09 9849.67 13.47 47.51 62.18 2388.06 8134.25 3.3930 0.03 392 2388 180.0 38.99 23.3660
14	-0.0009 -0.0003 189.0 518.67 642.68 1592.95 1499.39 14.62 21.61 554.54 2388.09 9849.67 13.47 47.51 62.18 2388.06 8134.25 3.3930 0.03 392 2388 180.0 38.99 23.3660
15	-0.0018 -0.0003 189.0 518.67 642.68 1593.25 1499.39 14.62 21.61 554.54 2388.09 9849.67 13.47 47.51 62.18 2388.06 8134.25 3.3930 0.03 392 2388 180.0 38.99 23.3660
16	-0.0019 -0.0003 189.0 518.67 642.68 1593.25 1499.39 14.62 21.61 554.54 2388.09 9849.67 13.47 47.51 62.18 2388.06 8134.25 3.3930 0.03 392 2388 180.0 38.99 23.3660
17	0.0002 -0.0002 189.0 518.67 642.69 1594.99 1499.39 14.62 21.61 554.55 2388.09 9854.92 13.47 47.52 62.18 2388.06 8137.27 8.4542 0.03 392 2388 180.0 38.81 23.3331
18	-0.0003 -0.0001 189.0 518.67 642.69 1594.99 1499.39 14.62 21.61 554.55 2388.09 9854.95 13.47 47.52 62.18 2388.07 8137.27 8.4542 0.03 392 2388 180.0 38.81 23.3331
19	-0.0003 -0.0001 189.0 518.67 642.69 1594.99 1499.39 14.62 21.61 554.55 2388.09 9854.95 13.47 47.52 62.18 2388.07 8137.27 8.4542 0.03 392 2388 180.0 38.81 23.3331
20	-0.0037 0.0001 189.0 518.67 643.04 1581.11 1495.23 14.62 21.61 554.81 2388.05 9845.98 13.47 47.52 62.22 2388.02 8129.71 8.4210 0.03 392 2388 180.0 39.03 23.4226
21	-0.0012 -0.0001 189.0 518.67 643.37 1586.07 1498.13 14.62 21.61 554.88 2388.11 9845.98 1.13 47.52 62.19 2388.06 8132.04 8.4084 0.03 392 2388 180.0 39.09 23.3101
22	0.0002 -0.0001 189.0 518.67 642.77 1582.95 1498.17 14.62 21.61 554.88 2388.04 9861.11 1.13 47.52 62.19 2388.06 8135.44 8.4384 0.03 392 2388 180.0 39.07 23.3197

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 (including MQTT library)

▼	AWSIoTPythonSDK			
►	__pycache__			
►	core			
►	exception			
?	__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
a9589dcf7-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	Keynote
4d2dec1d7b-public.pem.key		10 Sep 2019, 11:47 PM	451 bytes	Keynote
a9589dcf7-private.pem.key		Yesterday, 1:40 AM	2 KB	Keynote
a9589dcf7-public.pem.key		Yesterday, 1:40 AM	451 bytes	Keynote
ecf87423da-private.pem.key		29 Aug 2019, 1:10 AM	2 KB	Keynote
ecf87423da-public.pem.key		29 Aug 2019, 1:10 AM	451 bytes	Keynote
ef8e4b716b-private.pem.key		30 Aug 2019, 12:56 PM	2 KB	Keynote
ef8e4b716b-public.pem.key		30 Aug 2019, 12:55 PM	451 bytes	Keynote
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	printa...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  with open('train_FD001.txt','r') as infile_1:
34  with open('FD001_out.txt','a') as outfile_1:
35      # Due to process time issue, only read the 1st 5000 lines of data
36      for line in infile_1.readlines()[0:5000]:
37          outfile_1.write(line)
38
39  with open('train_FD003.txt','r') as infile_2:
40  with open('FD003_out.txt','a') as outfile_2:
41      # Due to process time issue, only read the 1st 5000 lines of data
42      for line in infile_2.readlines()[0:5000]:
43          outfile_2.write(line)

```

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

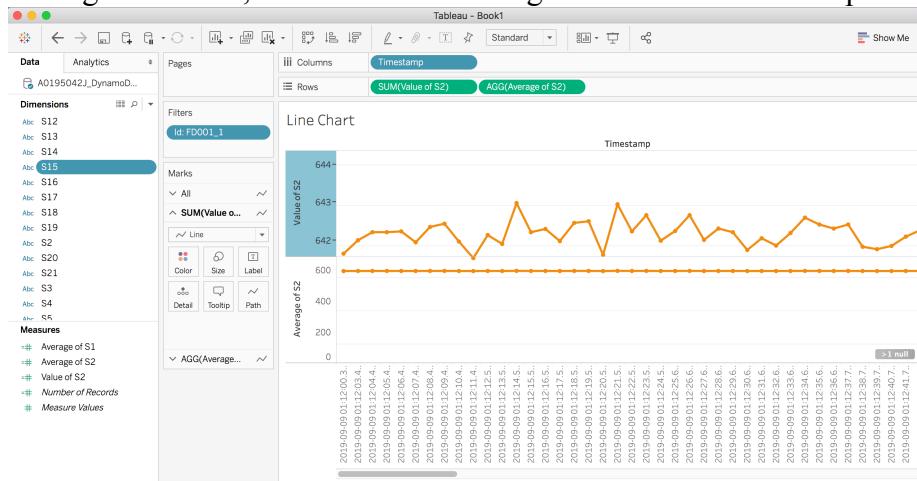
The screenshot shows the AWS DynamoDB console for the table **A0195042J\_Engine**. The table structure is as follows:

	<b>id</b>	<b>timestamp</b>	<b>Metric_Number</b>	<b>cycle</b>	<b>os1</b>	<b>os2</b>
<input type="checkbox"/>	FD001_1	2019-09-09 01:12:00.380299	A0195042J	1	-0.0007	-0.0004
<input type="checkbox"/>	FD001_1	2019-09-09 01:12:03.434963	A0195042J	2	0.0019	-0.0003
<input type="checkbox"/>	FD001_1	2019-09-09 01:12:04.438422	A0195042J	3	-0.0043	0.0003

A0195042J_Engine		Close				
Overview		Items	Metrics	Alarms	Capacity	Indexes
Create item		Actions				
<b>Scan: [Table] A0195042J_Engine: id, timestamp</b>						
Filter	id	String	=	FD003_1		X
	+ Add filter					
Start search						
	id	timestamp	Matric_Number	cycle	os1	os2
FD003_1	2019-09-09 02:36:20.197349	A0195042J	1	-0.0005	0.0004	
FD003_1	2019-09-09 02:36:23.259513	A0195042J	2	0.0008	-0.0002	
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

### 7.1) Background

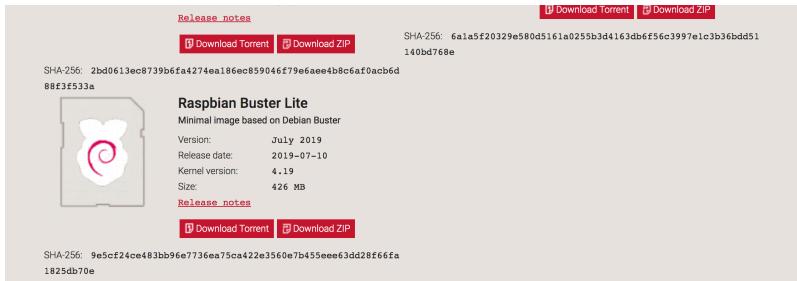
Prepare dataset by downloading the rain fall data - **rainfall-monthly-total.csv** from <https://data.gov.sg/>

### 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 Raspbian lite, and flash into microSD card:



- f. 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:

```
[ OK ] Started System Logging Service.
[ OK ] Started triggerhappy global hotkey daemon.
[ OK ] Started restore Sound Card State.
[ OK ] Started raspotify.
[ OK ] Reached target Sound Card.
[ OK ] Started avahi nMSS-DNS-SD Stack.
[ OK ] Started Login Service.
[ OK ] Started WPA supplicant.
[ OK ] Started dhclient - set up, mount/unmount, and delete a swap file.
[ OK ] Started LSB: Switch to ondemand cpu governor (unless shift key is pressed).
[ OK ] Started hwpcd on all interfaces.
[ OK ] Started target User Accounts.
    Starting Permit User Sessions...
    Starting OpenBSD Secure Shell server...
    Starting /etc/rc.local Compatibility...
    Starting Daily apt download activities...
My IP address is 192.168.0.114
[ OK ] Started Permit User Sessions.
[ OK ] Started /etc/rc.local Compatibility.
[ OK ] Started Getty on ttys0.
[ OK ] Started Serial Getty on ttym0.
[ OK ] Reached target Login Prompts.
[ OK ] Started OpenBSD Secure Shell server.
[ OK ] Started Daily apt download activities.

Raspbian GNU/Linux 10 raspberrypi ttys1
pi@raspberrypi:~$ pip3.7 --version
Python 3.7.4
pi@raspberrypi:~$ pip3.7 --version
pip 19.2.3 from /usr/local/lib/python3.7/site-packages/pip (python 3.7)
pi@raspberrypi:~$
```

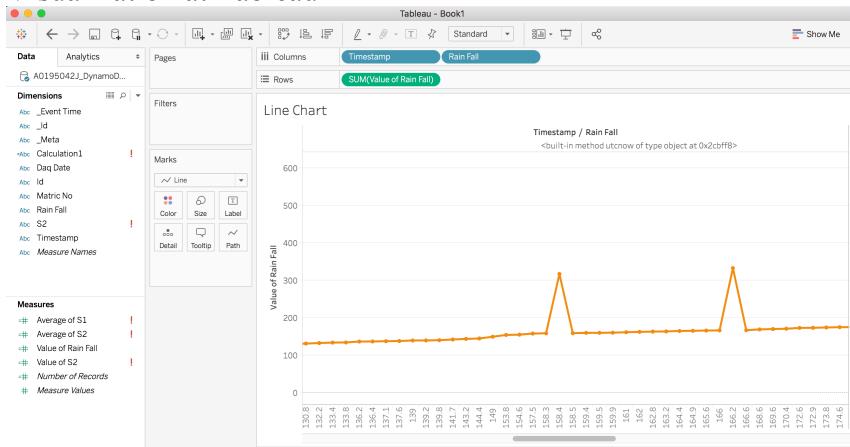
- j. Download AWS\_IoT\_Device\_Python\_SDK into raspberry pi  
Inside raspberry, follow steps (unix-like system) in link below:  
<https://docs.aws.amazon.com/greengrass/latest/developerguide/IoT-SDK.html>  
The SDK will be created into a folder named “aws-iot-device-sdk-python” under the root directory.
  - 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:

- m. Run the script - **A0195042J\_PublishCode\_RaspiData.py**, and start publishing the rain fall data into DynamoDB table:

### 7.3) Verify inside DynamoDB

id	DaqDate	Metric_No	RainFall	timestamp
228	2000-12	A0195042J	236	<built-in method utcnow of type object at 0x2cb
230	2001-02	A0195042J	86.6	<built-in method utcnow of type object at 0x2cb
231	2001-03	A0195042J	297.3	<built-in method utcnow of type object at 0x2cb

### 7.4) Visualization at Tableau



## 8. Discussion

For task3, only have time to finish the AWS IoT console setup.

Script is similar to task2's – the cpu temperature is collected using commands:

`vcgencmd measure_temp`

which calls from:

`from gpiozero import CPUTemperature`

## 9. Appendixes:

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

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

```

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

```

```

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

```

```

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

```

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

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

```

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

```

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