

AWS IoT Resources

- Getting started with AWS IoT Core: <https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs.html>
- AWS IoT Interactive Tutorial: this tutorial illustrates the interworking of some of the key components, see Figure 1. <https://console.aws.amazon.com/iot/home?region=us-east-1#/tutorial?step=1>

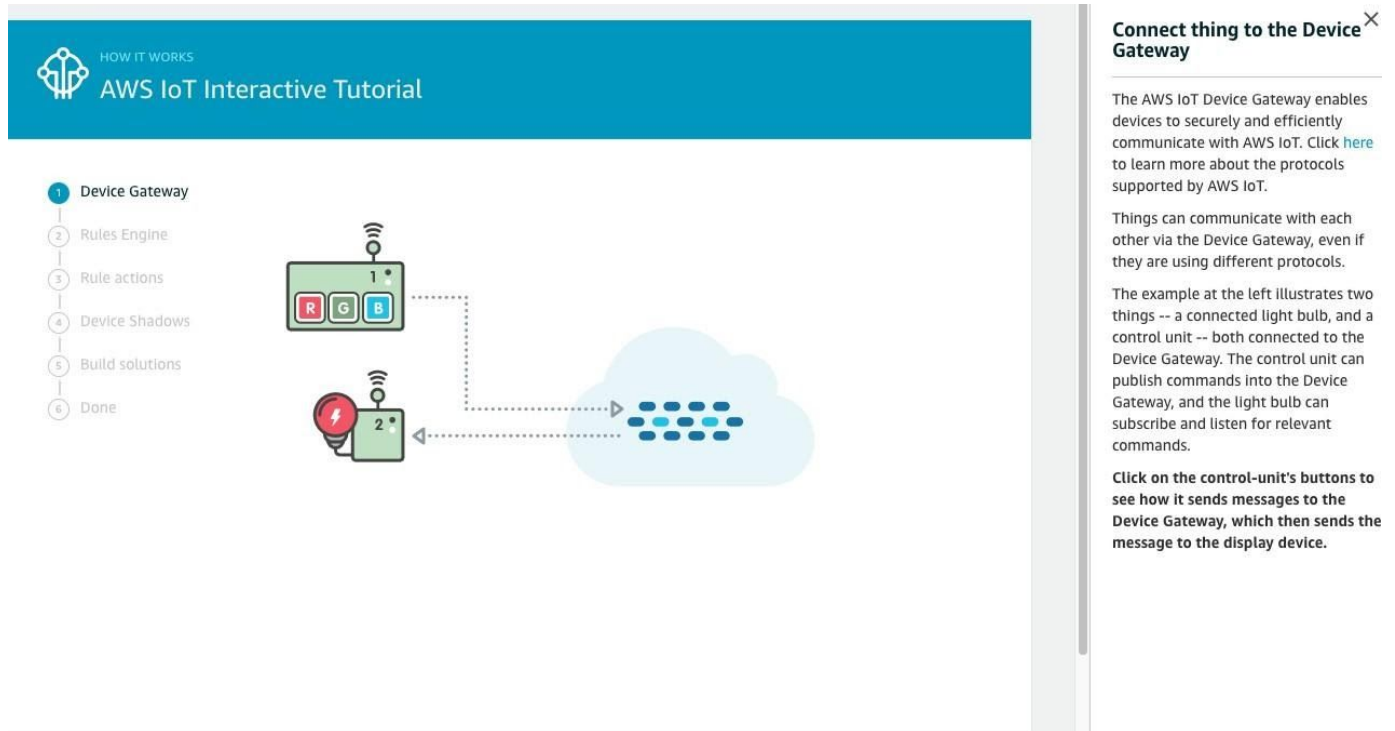


Figure 1: AWS IoT Interactive Tutorial

Create an AWS Account

Please follow the online instructions on the AWS website to create a free account (for the first 12 months):

Open <https://portal.aws.amazon.com/billing/signup>

Obtain AWS Account Security Credentials

Aher login to AWS, you will be taken to the "AWS Management Console". Please execute the following steps to create the security credentials that will be used later:

1. Under your username, click on "My Security Credential", as shown in Figure 2.
2. On the "Your Security Credentials" page, please click "Create New Access Key", as shown in Figure 3.
3. Follow the instruction to download the key file, as shown in Figure 4. A **rootkey.csv** file will be downloaded that contains "**access key ID**" and "**secret access key**". Make a note of those two keys for later usage.

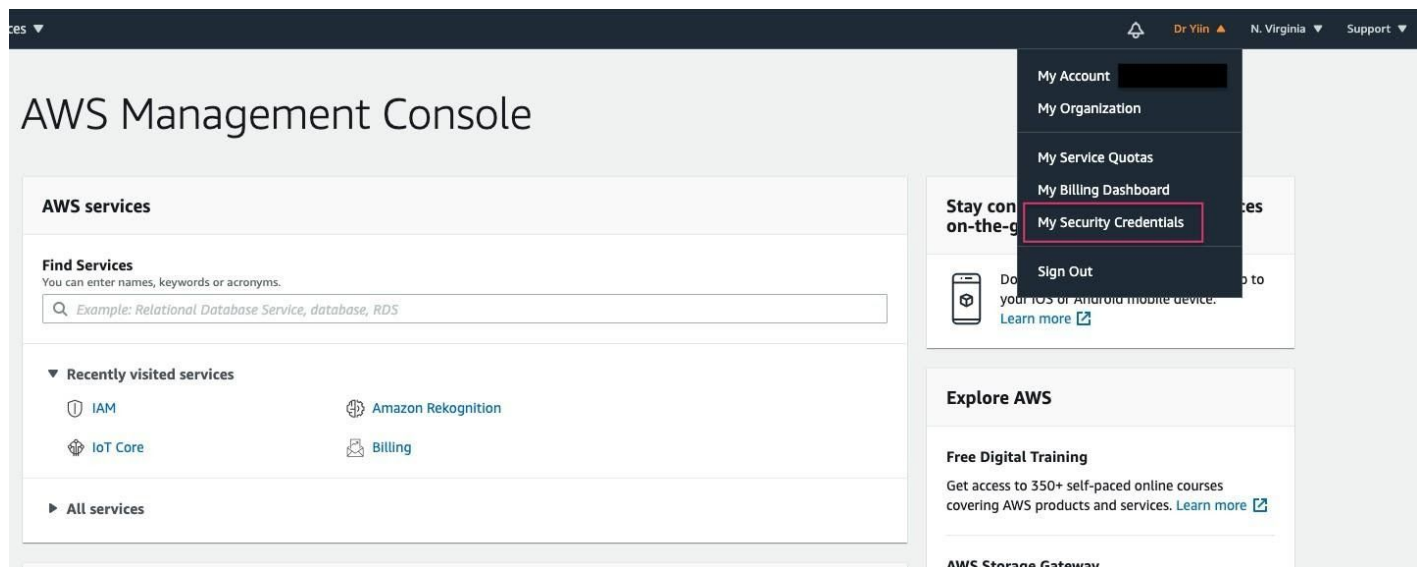
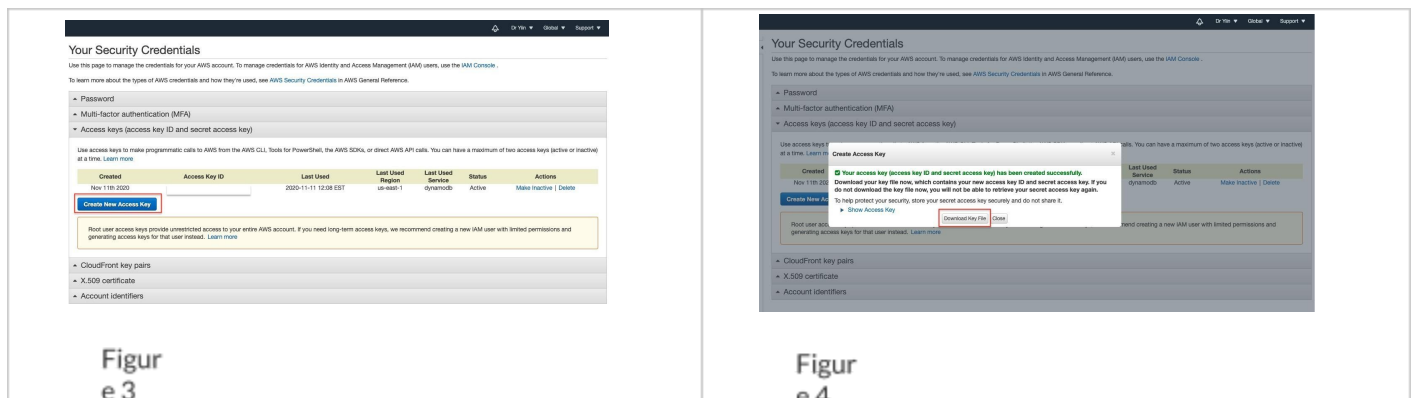


Figure 2: AWS Security Credentials



Choose AWS Regional Server

On "AWS Management Console", choose the AWS regional server. It is suggested to pick "US East (N. Virginia): us-east-1, as shown in Figure 5.

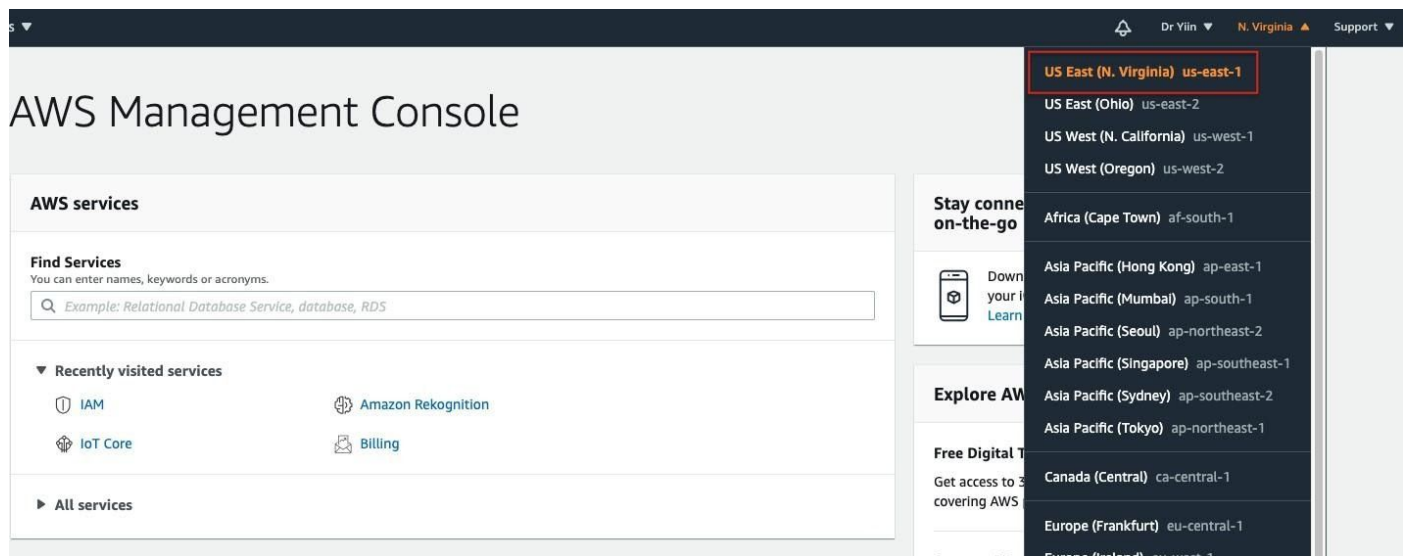


Figure 5: Selecting AWS Regional Server

Python Build Scripts

All example codes used in this assignment is delivered in a ZIP file (aws_iot.zip), along with this document. After unzipping, you will find the source code directory as follows:

```
bootstrap.sh
cert
├── root-CA.crt
├── create_thing.py
├── create_topic_rule.py
├── debug
│   ├── create_cloudwatch_logging.py
│   ├── destroy_cloudwatch_logging.py
│   └── json
│       ├── iot-policy-logging.json
│       ├── iot-policy-passrole.json
│       ├── iot-role-trust.json
│       └── settings.json
├── demo
│   └── RpiSenseHatDemo.py
├── demo.sh
├── destroy_everything.py
├── destroy.sh
├── json
│   ├── iot-policy-dynamodb.json
│   ├── iot-policy-passrole.json
│   ├── iot-role-trust.json
│   ├── iot-shadow-policy.json
│   ├── iot-thing-policy.json
│   ├── iot-topic-rule-dynamodb.json
│   ├── iot-topic-rule-republish.json
│   └── settings.json
├── shadow_listener.sh
├── shadow.sh
└── start.sh
```

Install AWS Command Line Interface (AWS CLI)

AWS CLI provides command-line access to the functions provided by the AWS IoT services. In this assignment, we utilize this unified tool for several of the Python example codes. Install the package

```
sudo apt-get install awscli
```

Run "AWS Configure"

For general use, the `aws configure` command is the fastest way to set up your AWS CLI installation. Four pieces of information are needed, as shown below.

1. Note that the Access Key ID and the Secret Access Key are stored in the rootkey.csv file that we generated earlier.
2. The default output format is set as json.

```
$ aws configure
AWS Access Key ID [None]: Your Access Key ID
AWS Secret Access Key [None]: Your Secret Access Key
Default region name [None]: us-east-1
Default output format [None]: json
```

Section 1: Create an AWS IoT Thing

To create a "thing" in AWS IoT, there are three steps involved which we went through manually in the class.

- Step 1: Register the device (in this assignment, it is called RpiSenseHat).
- Step 2: Create and save AWS IoT certificate, public key, and private key. Download the root CA as well.
- Step 3: Create and attach a policy to the thing key/certificate.

1. In Raspberry Pi, open a "Terminal" then go to the unzipped source folder. Type the following command to create a thing:

```
python create_thing.py
```

As shown from Figure 6 below, the script creates **RpiSenseHat**, then creates a certificate, a public key, and a private key (**all saved in the /cert directory**). It then creates two policies, RpiSenseHatPolicy and RpiSenseHatShadowPolicy, for the thing and its shadow, respectively. Finally, the two policies are attached to the key/certificate.

```

pi@Rulai:~/aws_iot $ python create_thing.py
Creating AWS IoT Thing: RpiSenseHat
--> Created ARN: arn:aws:iot:us-east-1:639322701643:thing/RpiSenseHat

Creating AWS IoT Certificate/Public Key/Private Key: certFile: cert/RpiSenseHat.cert.pem,
rt/RpiSenseHat.public.key, privateKeyFile: cert/RpiSenseHat.private.key
--> Created ARN: arn:aws:iot:us-east-1:639322701643:cert/246849e614dfcf6d3c7d483984a26f2a0
8463cfc1dd0567, Id: 246849e614dfcf6d3c7d483984a26f2a0c5f012f2cf3fd8d438463cfc1dd0567

Creating AWS IoT thing policy: RpiSenseHatThingPolicy
--> Created ARN: arn:aws:iot:us-east-1:639322701643:policy/RpiSenseHatThingPolicy

Creating AWS IoT shadow policy: RpiSenseHatShadowPolicy
--> Created ARN: arn:aws:iot:us-east-1:639322701643:policy/RpiSenseHatShadowPolicy

Attach thing to certificate...
--> Done

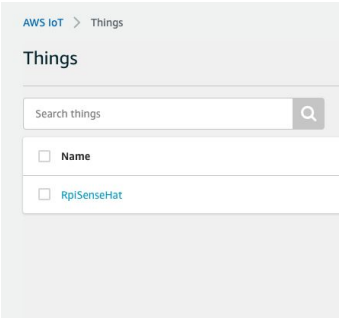
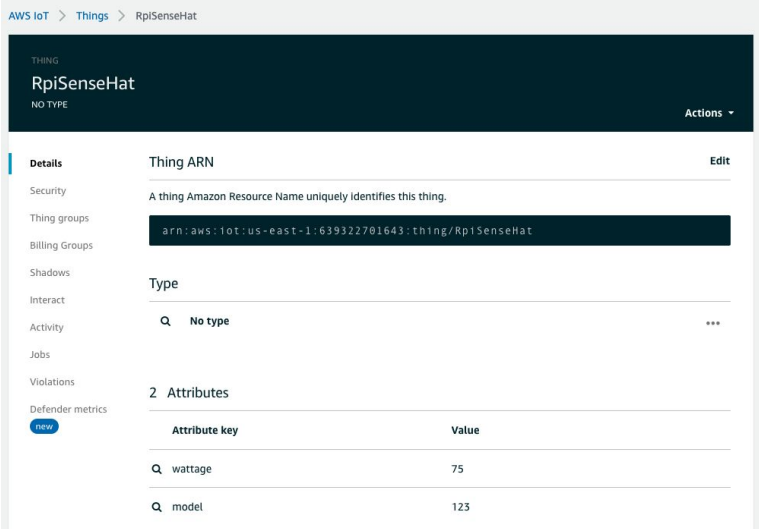
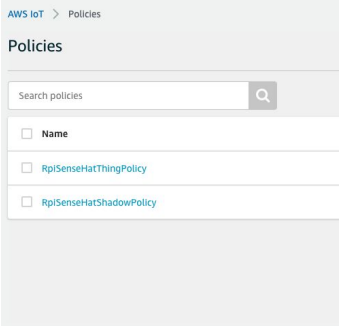
Attach IoT thing policy to certificate...
--> Done

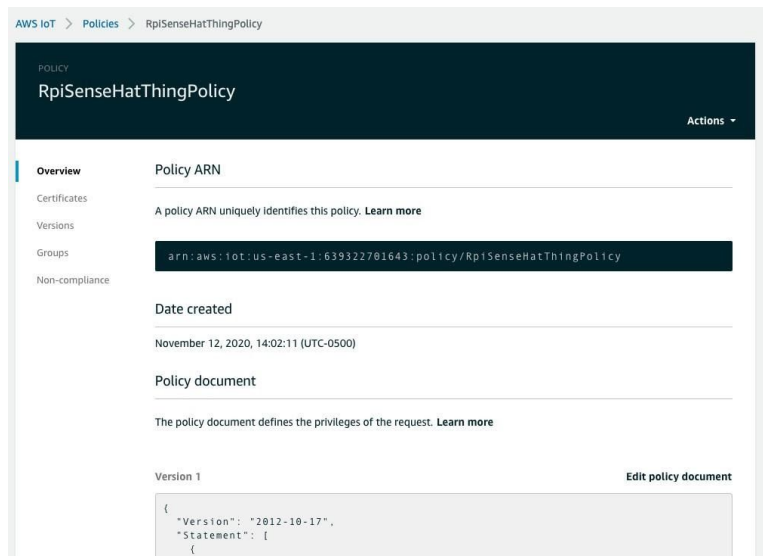
Attach IoT shadow policy to certificate...
--> Done

```

Figure 6: Running Script to Create an AWS IoT Thing.

2. Aher running the script, you can check back to your AWS IoT Console and confirm that a new thing and its corresponding attributes are indeed created, as illustrated in the following figures.

| | |
|---|---|
|  |  |
|  | |



Section 2: Create Topic Rules:

AWS IoT employs the **rules engine** to give devices the ability to interact with AWS services. Rules are analyzed and actions are performed based on the MQTT topic stream. You can use rules to support tasks such as writing device data to an Amazon DynamoDB database or sending message data to an AWS IoT Analytics channel. Before AWS IoT can take action based on the rule, you must grant it permission to access your AWS resources on your behalf.

1. Before creating a rule, you must create an **IAM role (IAM = Identity and Access Management)** with an attached **policy** that gives permission to the required AWS resources. The rule engine assumes this role when executing a rule. For details of this operation, please review the information here: **Granting AWS IoT the required access** (<https://docs.aws.amazon.com/iot/latest/developerguide/iot-create-role.html>)
2. Rules engine also needs to have the ability to pass the IAM role to the AWS services. This is achieved by creating a policy that grants the **iam:PassRole** permission and attaching it to the IAM role. More details can be found here: **Pass role permissions** (<https://docs.aws.amazon.com/iot/latest/developerguide/pass-role.html>)

An AWS IoT rule consists of the following key components:

- **Rule name**
- **SQL statement:** A simplified SQL syntax to filter MQTT topic messages and route the data elsewhere
- **Actions:** The actions that AWS IoT performs when executing the rule

Examples of some topic rules can be found here: **Creating an AWS IoT rule** (<https://docs.aws.amazon.com/iot/latest/developerguide/iot-create-rule.html>)

Two topic rules are created in this assignment: **RpiSenseHatRuleDynamoDB** and **RpiSenseHatRuleRepublish**. The former reads the incoming data from RpiSenseHat and writes them to a DynamoDB table, while the later directs the data to the device shadow.

1. In Raspberry Pi, open a "Terminal" then go to the unzipped source folder. Type the following command to create rules and topics:

```
python create_topic_rule.py
```

As shown from Figure 7 below, the script creates an IAM role, **RpiSenseHatRole**, and a policy, **RpiSenseHatPolicyDb**, to access DynamoDB. It also creates another policy, **RpiSenseHatPolicyPassRole** to grant permission for pass role. Finally, the two topic rules are created.

```

pi@Rulai:~/aws_iot $ python create_topic_rule.py
Creating AWS DynamoDB Table: RpiSenseHatDbTable
--> Created ARN: arn:aws:dynamodb:us-east-1:639322701643:table/RpiSenseHatDbTable

Creating AWS IAM Role: RpiSenseHatRole
--> Created ARN: arn:aws:iam::639322701643:role/RpiSenseHatRole

Creating AWS IAM policy to access DynamoDB: RpiSenseHatPolicyDb
--> Created ARN: arn:aws:iam::639322701643:policy/RpiSenseHatPolicyDb

Attach IAM Role & Policy with DynamoDB access...
--> Done

Creating AWS IAM policy for passRole: RpiSenseHatPolicyPassRole
--> Created ARN: arn:aws:iam::639322701643:policy/RpiSenseHatPolicyPassRole

Attach IAM Role & Policy with passRole...
--> Done

Attach IAM Role & Policy with AWSIoTFullAccess..
--> Done

Creating AWS IoT topic rule for DynamoDB: RpiSenseHatRuleDynamoDb
--> Done

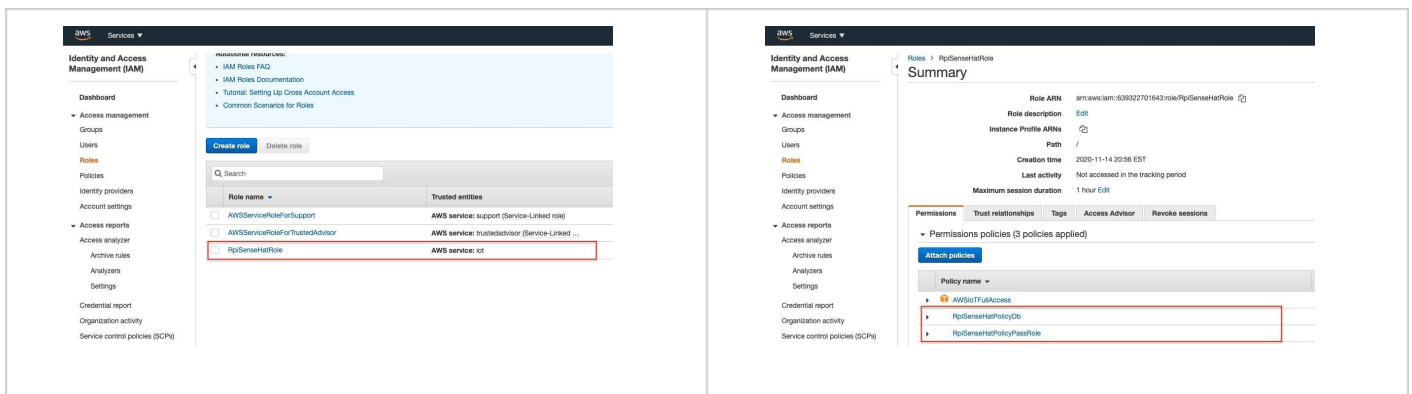
Creating AWS IoT topic rule for republish: RpiSenseHatRuleRepublish
--> Done

```

Figure 7: Running the Script for Creating Topic Rules.

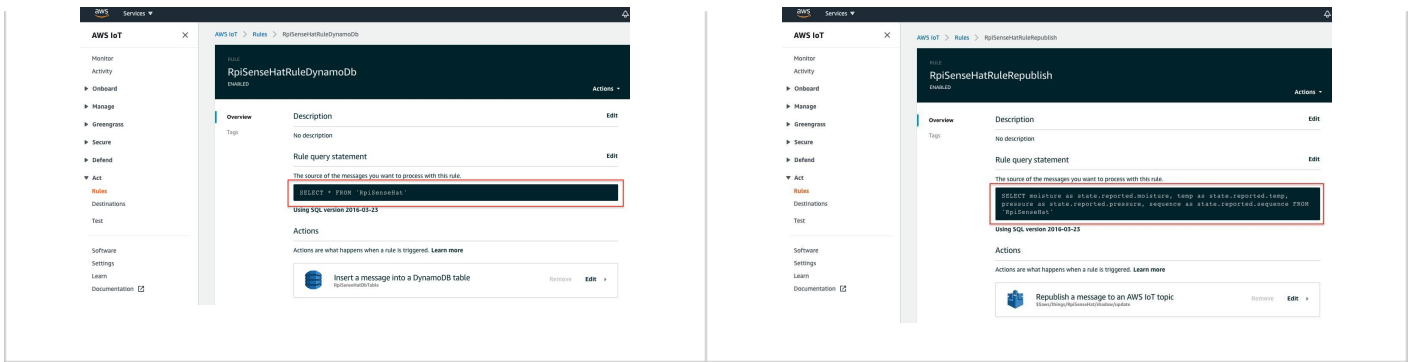
2. You can visit the AWS Identity and Access Management (IAM) dashboard and click the "Roles" on the left side of the panel. The new IAM roles should be created.

Click the IAM role and, on the Policies page, you will find the two policies attached to the role, as shown in the figure below.



3. You can check back to the AWS IoT Console, click on the "Rules" tab and see the two topic rules created for access DynamoDB service and Republish service, as shown in the figure below. Please make a note of the SQL statement for each rule.



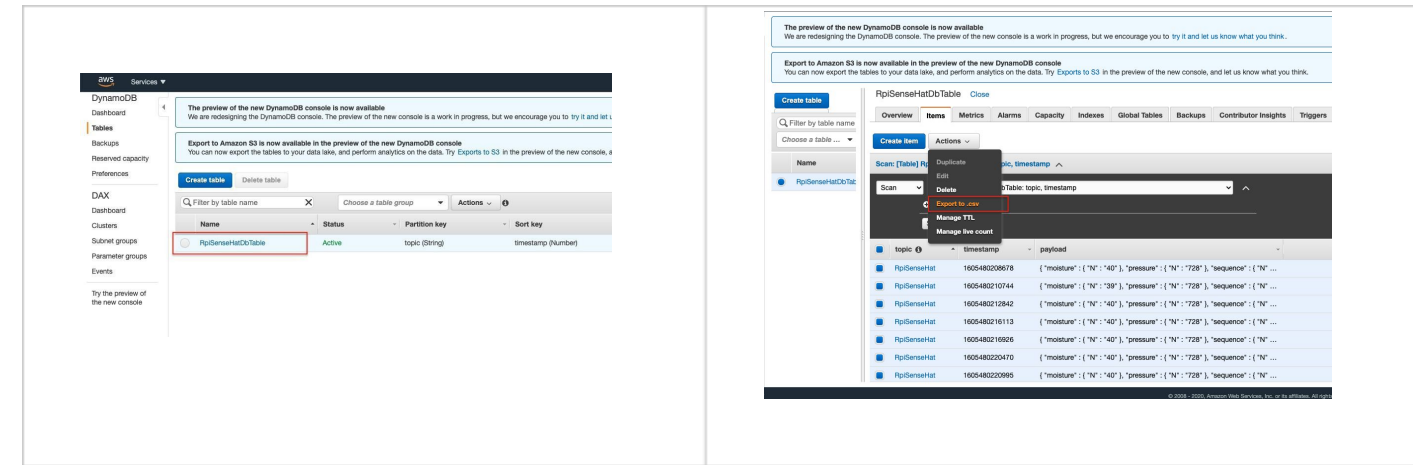


Section 3: Upload and Read Back SenseHAT Data

The last script is a demo code that publishes the SenseHAT environmental sensor data to AWS IoT, through the MQTT messaging protocol. The script also sends back the data for local display by subscribing to the topic, RpiSenseHat. At the same time, the sensor data stream is written to the DynamoDB table and is republished in the topic, RpiSenseHat/shadow.

1. Attach the SenseHat board to Raspberry Pi. You can also use the SenseHat emulator for the purpose of this demo.
2. Open a "Terminal" in Raspberry Pi, then go to the unzipped source code folder. Type `./demo.sh` to run the demo code. The SenseHat datasets (temp, pressure, and humidity) are sent to AWS IoT every two seconds and are published to the topic, **RpiSenseHat**.
3. Aher a 5-second countdown on the Pi LED, you will start to see loop-backed SenseHat data shown on the Terminal, through the MQTT subscription.
4. To see how data are recorded in the DynamoDB, go to the AWS DynamoDB service page. You will see the table generated by the demo code, **RpiSenseHatDbTable**, and the SenseHat data stored in the database. You can also use the Action button to export the data as .csv file.

* In demo.sh, the script will also first check whether the AWSIoTPython SDK is installed.



5. You can also observe, from the AWS IoT Console, the real-time data update in the shadow device of RpiSenseHat, as a result of the republishing rule.

aws

Services

AWS IoT

Monitor

Activity

Onboard

Manage

Things

Types

Thing groups

Billing groups

Jobs

Tunnels

Greengrass

Secure

Defend

Act

Test

Software

Settings

Learn

Documentation

AWS IoT > Things > RpiSenseHat > Classic Shadow

THING

RpiSenseHat

NO TYPE

Actions

Details

Security

Thing groups

Billing Groups

Shadows

Interact

Activity

Jobs

Violations

Defender metrics

Shadow ARN

Back to Shadow List

A shadow ARN uniquely identifies the shadow for this thing.

arn:aws:iot:us-east-1:639322701643:thing/RpiSenseHat

For more info on using shadow, [Learn more](#)

Shadow Document

Delete Edit

Last update: November 15, 2020, 23:31:27 (UTC-0500)

Shadow state:

```
{  "desired": {    "moisture": 38  },  "reported": {    "moisture": 50,    "temp": 33,    "pressure": 1012,    "sequence": 92  }}
```