# Deploy an End-to-End IoT Application

**SPL-168 - Version 1.1.15**

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

Corrections, feedback, or other questions? Contact us at *AWS Training and Certification*.

## Overview

In this lab, you will: connect virtual things using AWS IoT, publish messages and visualize real-time data using a serverless web application leveraging AWS Lambda, Amazon API Gateway and Amazon S3.

## LAB DESCRIPTION

By the end of this lab, you will be able to:

- Set up AWS IoT
- Create an IoT thing, policy, and certificate
- Run a device simulator on Amazon EC2
- Process and Visualize streaming data using a serverless app

## Lab Setup

The lab comes pre-configured with an EC2 instance that you use as an IoT device simulator. It also configures an API Gateway, a Lambda function and DynamoDB tables that will be used by you to set up a serverless visualization dashboard.

## Start lab

1. To launch the lab, at the top of the page, choose Start lab.

You must wait for the provisioned AWS services to be ready before you can continue.

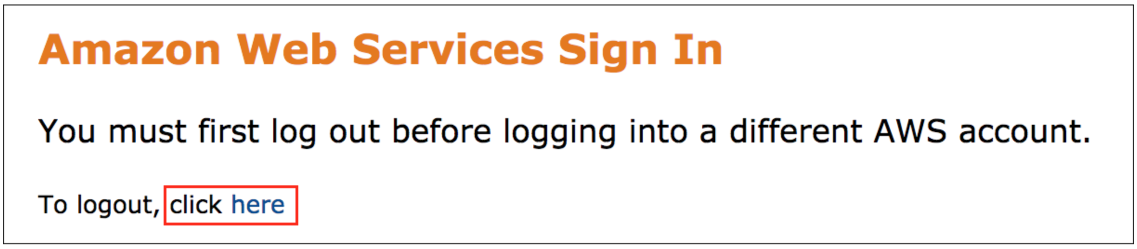
2. To open the lab, choose Open Console.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**Do not change the Region unless instructed.**

## COMMON SIGN-IN ERRORS

**Error: You must first sign out**



Amazon Web Services Sign In

You must first log out before logging into a different AWS account.

To logout, click here

If you see the message, **You must first log out before logging into a different AWS account:**

- Choose the **click here** link.
- Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.

- Choose **Open Console** again.

**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

- Add the lab domain name to your pop-up or script blocker's allow list or turn it off.
- Refresh the page and try again.

---

# ICON KEY

Various icons are used throughout this lab to call attention to certain aspects of the guide. The following list explains the purpose for each one:

- Specifies the command you must run.
- Specifies a sample output that you can use to verify the output of a command or edited file.
- Specifies important hints, tips, guidance, or advice.

---

# Set up AWS IoT

In this task, you create the resources needed in the AWS IoT console. There are 4 components that need to be created:

- **Thing** – A thing is a logical representation of a device stored in IoT's Registry. A thing supports attributes, as well as Device Shadows, which can be used to store device state & define desired state.
- **Policy** – You attach a policy to a certificate to dictate what the certificate (or rather, a Thing using that certificate) is entitled to do on AWS IoT.
- **Certificates** – Certificates are used for authentication. Things can communicate with AWS IoT via MQTT, MQTT over WebSockets or HTTPS. MQTT is a machine-to-machine pub-sub protocol well-suited for IoT use cases given its low overhead and low resource requirements. MQTT transmission to your AWS IoT gateway is encrypted using TLS and authenticated using certificates that you create.
- **Rule** – Rules leverages AWS IoT's Rules Engine to dictate how messages sent from Things to AWS IoT are handled. You configure rules that send data published to an MQTT topic to a variety of AWS Services.

# Task 1: Create an IoT Thing

3. In the **AWS Management Console**, to the right of **Services** menu, in the search bar, search for `IoT Core` and then choose **IoT Core** from the list.

If you see a message that says "Introducing the new AWS IoT console experience", you can dismiss the message by selecting the X to the right of the message.

4. In the navigation pane at the left of the **AWS IoT** page, under the **Manage** category, choose **All devices** -> **Things**.

There will not be any things listed in this section yet.

5. On the **Things** page, choose **Create things** .
6. On the **Create things** page, in the **Number of things to create** section, select **Create a single thing**.
7. Choose **Next** .
8. On the **Specify thing properties** page, in the **Thing properties** section:

- **Thing name:** `iotThing`
- Keep all other default settings

9. At the bottom of the page, choose **Next** .
10. On the **Configure device certificate -** ***optional*** page, choose **Skip creating a certificate at this time**.
11. Choose **Create thing** .

A message should display stating: *You successfully created thing iotThing*.

12. In the AWS IoT navigation pane, choose **Settings**.
13. In the **Device data endpoint** section, copy the **Endpoint** to your text editor.

It should look like this: ***a1xduq2dksdz70-ats.iot.us-west-2.amazonaws.com***. You need this endpoint to configure the device simulator later in the lab.

# Task 2: Review the IoT Policy

In this task, you review the IoT policy that was created for this lab.

14. In the AWS IoT navigation pane, choose **Security** > **Policies**.

15. Choose **iotPolicy**.

16. Under “Policy Document”, there is an IoT policy listed in this page. This policy was created in advance for this lab.

In this policy, only your specific Amazon Resource Name (arn) is given specific rights to publish, receive, and subscribe to the MQTT topic **device/*/rechargeAlert** and **device/*/devicePayload**. No other topics are allowed. Also, this policy limits the devices that can send messages to any IoT thing that is attached. **iot:Connection.Thing.IsAttached**. This means that if a thing cannot publish or subscribe unless it exists in the AWS IoT Core thing registry.

# Task 3: Connect to Your EC2 IoT Device Simulator

In this task you connect to your EC2 IoT instance.

17. In the **AWS Management Console**, to the right of **Services** menu, in the search bar, search for `EC2` and then choose **EC2** from the list.

18. In the navigation pane at the left of the page, choose **Instances**.

19. Select **Ec2InstanceDeviceSimulator** and then, at the top-right of the page, choose Connect

20. On the **Connect to instance** page, choose the **Session Manager** tab, and then choose **Connect** .

A new browser tab opens with a console connection to the instance. A set of commands are run automatically when you connect to the instance that change to the user’s home directory and display the path of the working directory, similar to this:

```
cd HOME; pwd
sh-4.2$ cd HOME; pwd
/home/ec2-user
sh-4.2$
```

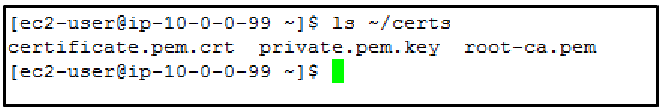
On a Windows-based computer, you might need to use **Ctrl + Shift + V** or open the context menu (right-click) to paste text into the console window.

# Task 4: Attach the Policy and Thing to the IoT Certificate

In this task, you associate the certificate from your EC2 IoT Simulator to your policy and thing. While it is possible to create the device certificates in the AWS Management Console, these have been created for you during the CloudFormation stack creation via a script on your EC2 device simulator instance.

21. In the terminal, enter:

```
ls ~/certs
```



```
[ec2-user@ip-10-0-0-99 ~]$ ls ~/certs
certificate.pem.crt  private.pem.key  root-ca.pem
[ec2-user@ip-10-0-0-99 ~]$
```

You should have 3 files in the directory:

- certificate.pem.crt
- private.pem.key
- root-ca.pem

22. In the AWS IoT navigation pane, choose **Security** > **Certificates**.

A certificate has been pre-created by the CloudFormation template in your lab. The certificate must be associated with the thing and the policy that you created earlier. If there isn’t a certificate listed, wait 30 seconds and then refresh the page.

23. Choose on the name of the listed certificate.

24. Select Actions

25. Choose **Attach policy**.

26. In Attach policies to certificate(s) section:

- Select the **iotPolicy**
- Choose **Attach policies** .

27. Select Actions

28. Choose **Attach to things**.

29. In Attach things to certificate(s) section

- Select the **iotThing**
- Choose **Attach to thing** .

# Task 5: Configure and Run the IoT Device Simulator

An example script is provided that will send messages containing current battery charge, simulated GPS location data, as well as other telemetry data. The AWS IoT Service will process these messages and send them to the appropriate AWS services based on the rule actions that you configure.

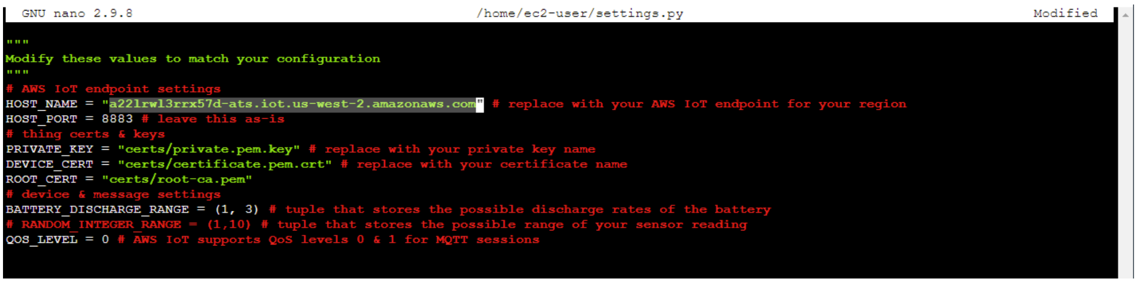
30. Use the arrow keys on your keyboard to navigate through the editor. If using a mouse, use the right mouse button to paste into the editor.

In the terminal window, edit the settings.py file by entering:

```
nano ~/settings.py
```

You are welcome to use any other text editor. This lab guide uses the nano editor.

31. Replace the **HOST_NAME** value with the Device data endpoint of your thing that you copied earlier to the clipboard. If you forgot to copy the endpoint, you can obtain it again by going to: **Settings** > **Device data endpoint**.



```
  GNU nano 2.9.8                    /home/ec2-user/settings.py                    Modified

"""
Modify these values to match your configuration
"""
# AWS IoT endpoint settings
HOST_NAME = "a22lrwl3rrx57d-ats.iot.us-west-2.amazonaws.com" # replace with your AWS IoT endpoint for your region
HOST_PORT = 8883 # leave this as-is
# thing certs & keys
PRIVATE_KEY = "certs/private.pem.key" # replace with your private key name
DEVICE_CERT = "certs/certificate.pem.crt" # replace with your certificate name
ROOT_CERT = "certs/root-ca.pem"
# device & message settings
BATTERY_DISCHARGE_RANGE = (1, 3) # tuple that stores the possible discharge rates of the battery
# RANDOM_INTEGER_RANGE = (1,10) # tuple that stores the possible range of your sensor reading
QOS_LEVEL = 0 # AWS IoT supports QoS levels 0 & 1 for MQTT sessions
```

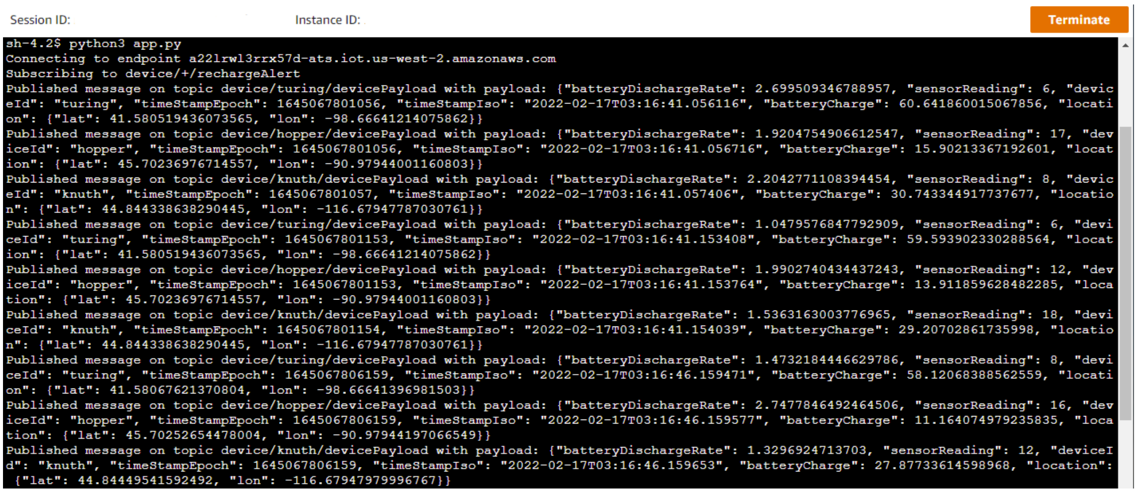
32. Save the settings.py file by:

- Pressing **Ctrl + X**
- Typing `y` and then pressing **Enter**

33. Start the device simulator by entering:

```
python3 app.py
```

34. IoT Core receives and displays data when the devices connect.



```
sh-4.2$ python3 app.py
Connecting to endpoint a221rwl3rrx57d-ats.iot.us-west-2.amazonaws.com
Subscribing to device/+/rechargeAlert
Published message on topic device/turing/devicePayload with payload: {"batteryDischargeRate": 2.699509346788957, "sensorReading": 6, "devic
eId": "turing", "timeStampEpoch": 1645067801056, "timeStampIso": "2022-02-17T03:16:41.056116", "batteryCharge": 60.641860015067856, "locati
on": {"lat": 41.580519436073565, "lon": -98.66641214075862}}
Published message on topic device/hopper/devicePayload with payload: {"batteryDischargeRate": 1.9204754906612547, "sensorReading": 17, "dev
iceId": "hopper", "timeStampEpoch": 1645067801056, "timeStampIso": "2022-02-17T03:16:41.056716", "batteryCharge": 15.90213367192601, "locat
ion": {"lat": 45.70236976714557, "lon": -90.97944001160803}}
Published message on topic device/knuth/devicePayload with payload: {"batteryDischargeRate": 2.2042771108394454, "sensorReading": 8, "devic
eId": "knuth", "timeStampEpoch": 1645067801057, "timeStampIso": "2022-02-17T03:16:41.057406", "batteryCharge": 30.743344917737677, "locatio
n": {"lat": 44.844338638290445, "lon": -116.67947787030761}}
Published message on topic device/turing/devicePayload with payload: {"batteryDischargeRate": 1.0479576847792909, "sensorReading": 6, "devi
ceId": "turing", "timeStampEpoch": 1645067801153, "timeStampIso": "2022-02-17T03:16:41.153408", "batteryCharge": 59.593902330288564, "locat
ion": {"lat": 41.580519436073565, "lon": -98.66641214075862}}
Published message on topic device/hopper/devicePayload with payload: {"batteryDischargeRate": 1.9902740434437243, "sensorReading": 12, "dev
iceId": "hopper", "timeStampEpoch": 1645067801153, "timeStampIso": "2022-02-17T03:16:41.153764", "batteryCharge": 13.911859628482285, "loca
tion": {"lat": 45.70236976714557, "lon": -90.97944001160803}}
Published message on topic device/knuth/devicePayload with payload: {"batteryDischargeRate": 1.5363163003776965, "sensorReading": 18, "devi
ceId": "knuth", "timeStampEpoch": 1645067801154, "timeStampIso": "2022-02-17T03:16:41.154039", "batteryCharge": 29.20702861735998, "locatio
n": {"lat": 44.844338638290445, "lon": -116.67947787030761}}
Published message on topic device/turing/devicePayload with payload: {"batteryDischargeRate": 1.4732184446629786, "sensorReading": 8, "devi
ceId": "turing", "timeStampEpoch": 1645067806159, "timeStampIso": "2022-02-17T03:16:46.159471", "batteryCharge": 58.12068388562559, "locati
on": {"lat": 41.58067621370804, "lon": -98.66641396981503}}
Published message on topic device/hopper/devicePayload with payload: {"batteryDischargeRate": 2.7477846492464506, "sensorReading": 16, "dev
iceId": "hopper", "timeStampEpoch": 1645067806159, "timeStampIso": "2022-02-17T03:16:46.159577", "batteryCharge": 11.164074979235835, "loca
tion": {"lat": 45.70252654478004, "lon": -90.97944197066549}}
Published message on topic device/knuth/devicePayload with payload: {"batteryDischargeRate": 1.3296924713703, "sensorReading": 12, "deviceI
d": "knuth", "timeStampEpoch": 1645067806159, "timeStampIso": "2022-02-17T03:16:46.159653", "batteryCharge": 27.87733614598968, "location":
 {"lat": 44.84449541592492, "lon": -116.67947979996767}}
```

# Task 6: Create an IoT Rule and Action

IoT Rule Actions give your devices the ability to interact with AWS services. Rules are analyzed and actions are performed based on the MQTT topic stream. The simulated IoT devices report current battery charge percentage which decreases over time. We will create a rule action that will monitor the reported battery charge and publish a message to a new topic when it is time to recharge. The device is subscribed to this topic and will “take action” to recharge.

35. In the **AWS IoT Console**, in the navigation pane at the left of the page, under the **Message Routing** category, choose **Rules**.

36. Choose **Create rule** .

37. In the **Specify rule properties** section, configure:

- Under **Rule properties**, for **Rule name**, enter `gsRecharge` .

38. Choose **Next** .

39. In the **Configure SQL statement** section, in the **SQL statement** code block, delete the existing text and replace it with the following:

```
SELECT * FROM 'device/+/devicePayload' WHERE batteryCharge <=0
```

40. Choose **Next** .

41. In **Attach rule actions** page, under **Rule actions** section, for **Action 1** form, configure:

- **Action 1:** Select `Republish to AWS IoT topic` option in the dropdown.
- **Topic:** `device/${topic(2)}/rechargeAlert`
- **IAM role:** (Choose the **Select** button and then choose) `AwsIotRepublishRole`

42. Choose **Next** .

43. Then choose **Create** .

Once complete, a message appears stating that your rule was successfully created.

# Task 7: View Device Messages with the AWS IoT MQTT Client

Devices publish MQTT messages on topics. You can use the AWS IoT MQTT client to subscribe to these topics to receive the content of these messages. You now use the AWS IoT MQTT client to confirm that the IoT messages are being sent back and forth between the devices and the AWS IoT Device Gateway.

44. In the Navigation pane, Choose **MQTT test client**.

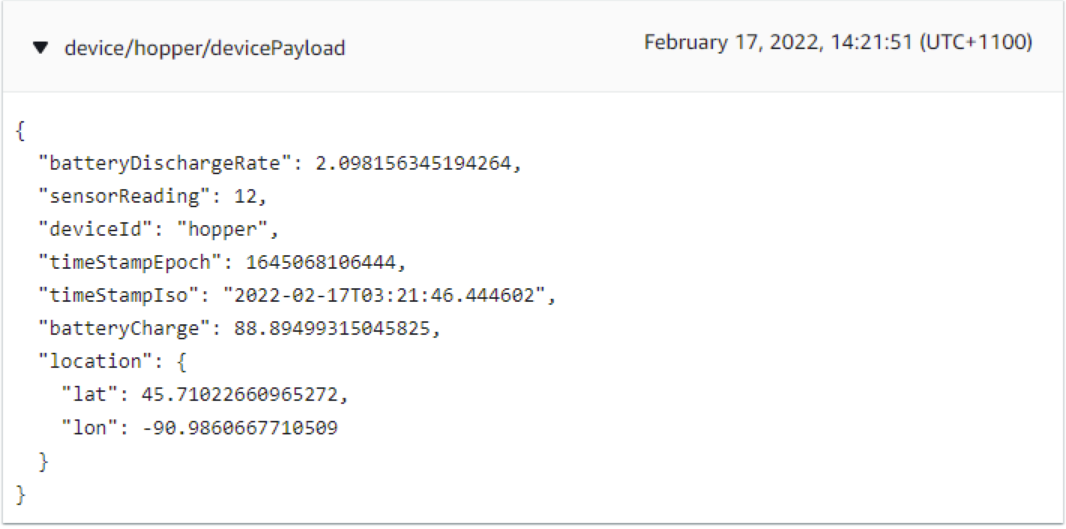
45. Under **Subscribe to a topic** section, enter

`#` as Topic filter.

This is a wild card symbol.

46. Select **Subscribe** .

If the devices are successfully configured, the MQTT messages that are received display on the page. Next, you confirm that the Recharge Rule is configured correctly.



▼ device/hopper/devicePayload February 17, 2022, 14:21:51 (UTC+1100)

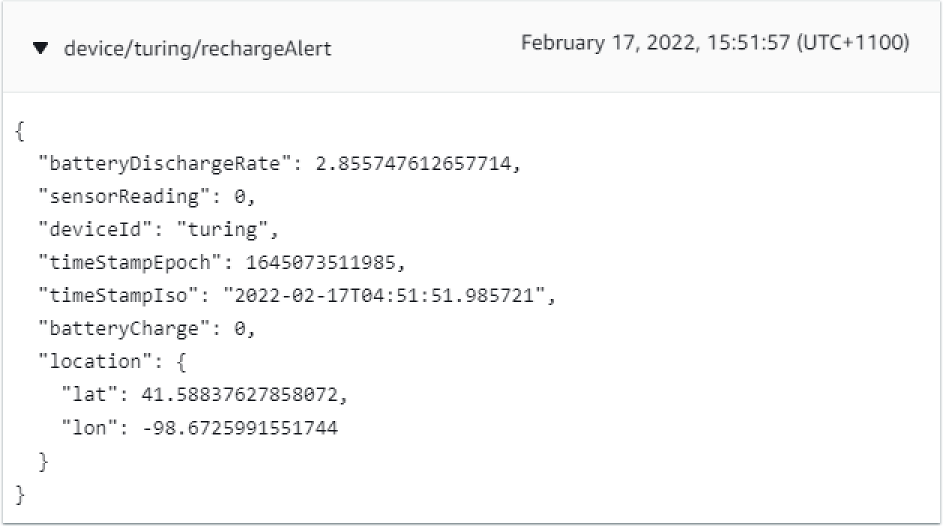
```
{
  "batteryDischargeRate": 2.098156345194264,
  "sensorReading": 12,
  "deviceId": "hopper",
  "timeStampEpoch": 1645068106444,
  "timeStampIso": "2022-02-17T03:21:46.444602",
  "batteryCharge": 88.89499315045825,
  "location": {
    "lat": 45.71022660965272,
    "lon": -90.9860667710509
  }
}
```

47. Delete the **#** symbol in the **Subscriptions** field.
48. Under **Subscribe to a topic** section, enter

   `device/+/rechargeAlert` as Topic filter.

49. Select **Subscribe** .
50. Wait for messages to be displayed.

This may take 2-5 minutes before the messages appear.



▼ device/turing/rechargeAlert February 17, 2022, 15:51:57 (UTC+1100)

```
{
  "batteryDischargeRate": 2.855747612657714,
  "sensorReading": 0,
  "deviceId": "turing",
  "timeStampEpoch": 1645073511985,
  "timeStampIso": "2022-02-17T04:51:51.985721",
  "batteryCharge": 0,
  "location": {
    "lat": 41.58837627858072,
    "lon": -98.6725991551744
  }
}
```

51. In the left navigation pane, choose **Monitor**.

The **Messages published** count should increase. You may need to refresh the page.

52. Take a moment to review the rest of the Dashboard.

# Task 8: Process and Visualize Streaming Data

In this task, you save time-series data from the devices to Amazon DynamoDB tables. You create a rule with two actions, to query the incoming messages and capture the payload. The first rule will write time series data from devices to a DynamoDB table called IoTDynamoTimeSeriesTable. The second rule will write the latest received messages to a DynamoDB table called IoTDynamoDeviceStatusTable. The actual DynamoDB table names will be prefixed by *qls*-….

53. In the **AWS IoT Console**, in the navigation pane at the left of the page, under the **Message Routing** category, choose **Rules**.
54. Choose **Create rule** .
55. In the **Specify rule properties** section, configure:

- Under **Rule properties**, for **Rule name**, enter `IoTToDynamo` .

56. Choose **Next** .
57. In the **Configure SQL statement** section, in the **SQL statement** code block, delete the existing text and replace it with the following:

```
SELECT * FROM 'device/+/devicePayload'
```

58. Choose **Next** .
59. In **Attach rule actions** page, under **Rule actions** section, for **Action 1** form, configure:

- **Action 1:** Select `Insert a message into a DynamoDB table` option in the dropdown.
- **Table name:** `GSDynamoTimeSeries`
- **Partition key:** `deviceId`
- **Partition key value:** `${topic(2)}`
- **Sort key:** `payloadTimestamp`
- **Sort key value:** `${timeStampEpoch}`
- **IAM role:** (Choose the **Select** button and then choose) `AwsIotToDynamoRole`

This action will write the payload to DynamoDB table using the timestamp as a range key value.

Next, you create a table of connected devices using the same IoT rule that reports the last reported value from the devices. You create an additional action to accomplish this.

- Under **Rule actions**, choose **Add rule action** .

60. For **Action 2** form, configure:

- **Action 2:** Select `Insert a message into a DynamoDB table` option in the dropdown.
- **Table name:** `GSDynamoDeviceStatus`
- **Partition key:** `deviceId`
- **Partition key value:** `${topic(2)}`
- **IAM role:** (Choose the **Select** button and then choose) `AwsIotToDynamoRole`

61. Choose **Next** .
62. Choose **Create** .

The rule and actions are now configured. In the next step you enable the APIs that read the DynamoDB table and return devices data in an API GET method.

# Task 9: Test the APIs

In this task you test that the API works, you use a command line to read the data via HTTP. The API definition and the backing AWS Lambda function that support the API were configured for you during the lab setup. In the next section, you "hook" the APIs into a dashboard to visualize the data in the website. The CloudFormation template has already configured a production stage called **prod**.

63. In the **AWS Management Console**, to the right of **Services** menu, in the search bar, search for

    ```
    API Gateway
    ```

    and then choose **API Gateway** from the list.

64. Under the name column, select the available **API name**.

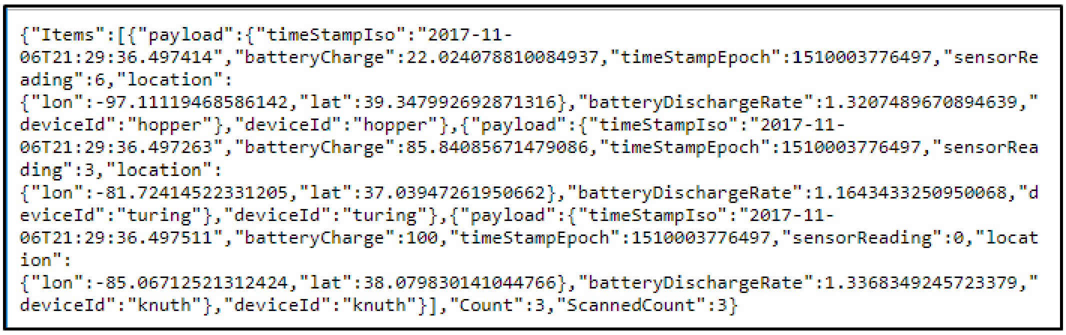
65. In the left navigation menu, choose **Stages**.

66. Under **Stages**, choose **prod**.

67. Review the current API configuration.

68. Choose the **Invoke URL** link.

Device data appears in JSON format.



```
{"Items":[{"payload":{"timeStampIso":"2017-11-
06T21:29:36.497414","batteryCharge":22.024078810084937,"timeStampEpoch":1510003776497,"sensorRe
ading":6,"location":
{"lon":-97.11119468586142,"lat":39.347992692871316},"batteryDischargeRate":1.3207489670894639,"
deviceId":"hopper"},"deviceId":"hopper"},{"payload":{"timeStampIso":"2017-11-
06T21:29:36.497263","batteryCharge":85.84085671479086,"timeStampEpoch":1510003776497,"sensorRea
ding":3,"location":
{"lon":-81.72414522331205,"lat":37.03947261950662},"batteryDischargeRate":1.1643433250950068,"d
eviceId":"turing"},"deviceId":"turing"},{"payload":{"timeStampIso":"2017-11-
06T21:29:36.497511","batteryCharge":100,"timeStampEpoch":1510003776497,"sensorReading":0,"locat
ion":
{"lon":-85.06712521312424,"lat":38.079830141044766},"batteryDischargeRate":1.3368349245723379,"
deviceId":"knuth"},"deviceId":"knuth"}],"Count":3,"ScannedCount":3}
```

69. Refresh the page every few seconds and watch the **timeStampIso** value change.

70. Copy the URL endpoint to your text editor.

You need it in the next section.

# Task 10: Deploy the Real-Time Dashboard

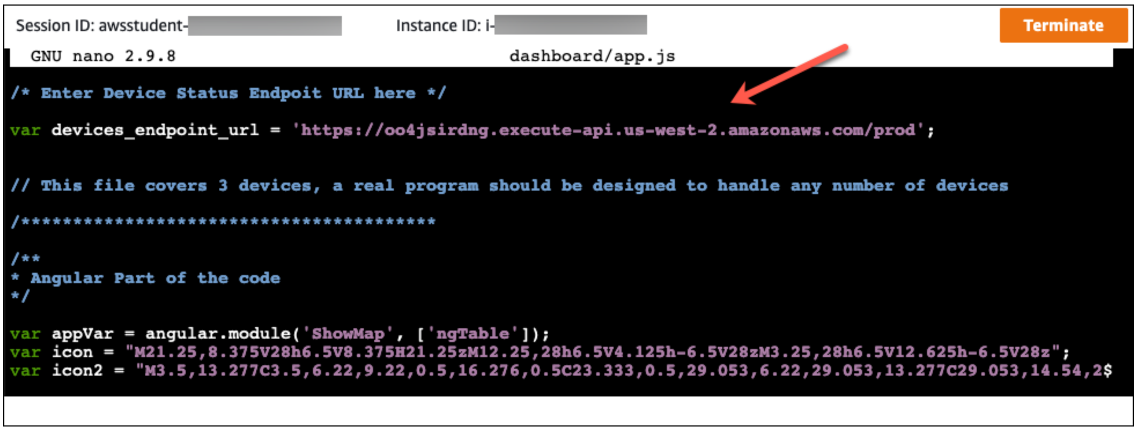
In this task, you visualize device data in a dashboard. The dashboard will place the devices in a map based on Geolocation (lon,lat), Battery Charge and Battery Discharge Rate will be displayed in a line chart. First, you download the dashboard code and update the API endpoint. Last, you setup a static website on S3 to host your dashboard.

71. Open the browser tab with the console connection to the EC2 instance. If the device simulator is still running, press CTRL-C to return to the command prompt.

72. Edit app.js by entering

    ```
    nano ~/dashboard/app.js
    ```

73. At the top of the file, set the **devices_endpoint_url** value to the API endpoint you copied to your text editor.



Session ID: awsstudent-[redacted]    Instance ID: i-[redacted]

```
 GNU nano 2.9.8                    dashboard/app.js

/* Enter Device Status Endpoit URL here */

var devices_endpoint_url = 'https://oo4jsirdng.execute-api.us-west-2.amazonaws.com/prod';


// This file covers 3 devices, a real program should be designed to handle any number of devices

/*****************************************

/**
* Angular Part of the code
*/

var appVar = angular.module('ShowMap', ['ngTable']);
var icon = "M21.25,8.375V28h6.5V8.375H21.25zM12.25,28h6.5V4.125h-6.5V28zM3.25,28h6.5V12.625h-6.5V28z";
var icon2 = "M3.5,13.277C3.5,6.22,9.22,0.5,16.276,0.5C23.333,0.5,29.053,6.22,29.053,13.277C29.053,14.54,2$
```

74. Save the app.js file by:

- Pressing **Ctrl + X**
- Typing

  ```
  y
  ```

  and then pressing **Enter**

**Host a Static Website on Amazon S3**

In this section, you configure a static website on an S3 bucket. To host your static website, you configure an Amazon S3 bucket for website hosting and then upload your website content to the bucket. The website is then available at the region-specific website endpoint of the bucket.

75. In the **AWS Management Console**, to the right of **Services** menu, in the search bar, search for `S3` and then choose **S3** from the list.

During the lab setup, a bucket was pre-created to hold the Dashboard. The bucket name should contain **iotgss3bucket** in the middle of the name.

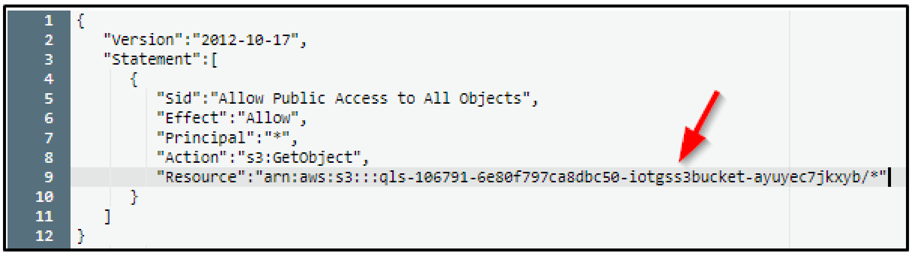
76. Choose the **iotgss3bucket**.

77. Update your **iotgss3bucket** policy by:

- Choose the **Permissions** tab
- Under **Bucket policy**, choose the **Edit** button
- Copy and paste the following code snippet into the Bucket Policy Editor:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"Allow Public Access to All Objects",
         "Effect":"Allow",
         "Principal":"*",
         "Action":"s3:GetObject",
         "Resource":"arn:aws:s3:::BUCKET/*"
      }
   ]
}
```

78. Replace **BUCKET** in the bottom line, with the name of **your S3 bucket**. For your convenience, the bucket name is located to the left of these instructions. You need to copy the value of **S3BucketName**.



```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"Allow Public Access to All Objects",
         "Effect":"Allow",
         "Principal":"*",
         "Action":"s3:GetObject",
         "Resource":"arn:aws:s3:::qls-106791-6e80f797ca8dbc50-iotgss3bucket-ayuyec7jkxyb/*"
      }
   ]
}
```

This policy enables anyone to read the bucket (run a GET HTTP command).

- Choose **Save changes** .

79. To enable static website hosting on your **iotgss3bucket**:

- Choose the **Properties** tab
- Under **Static website hosting**, choose the **Edit** button.
- Select **Enable**
- In the **Index document** field, enter `index.html` .
- In the **Error document** field, enter `index.html` . For this lab, there is no error handling.
- Choose **Save changes** .

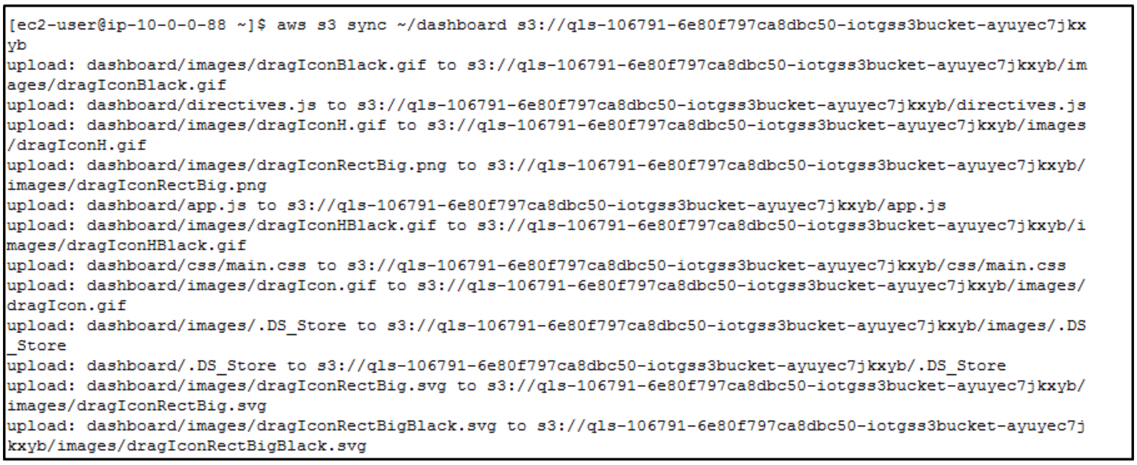
80. Under **Static website hosting**, copy the **Bucket website endpoint** URL to your text editor.

Your dashboard will be available at the bucket's endpoint.

81. Copy the dashboard code to the S3 bucket by configuring the following:

- Copy the following command into your terminal window: `aws s3 sync ~/dashboard s3://BUCKET`
- Replace **BUCKET** with the value of **S3BucketName** shown to the left of these instructions
- Run the command by pressing **Enter**

This will copy the content of your local directory into the S3 bucket.



```
[ec2-user@ip-10-0-0-88 ~]$ aws s3 sync ~/dashboard s3://qls-106791-6e80f797ca8dbc50-iotgss3bucket-ayuyec7jkx
yb
upload: dashboard/images/dragIconBlack.gif to s3://qls-106791-6e80f797ca8dbc50-iotgss3bucket-ayuyec7jkxyb/im
ages/dragIconBlack.gif
upload: dashboard/directives.js to s3://qls-106791-6e80f797ca8dbc50-iotgss3bucket-ayuyec7jkxyb/directives.js
upload: dashboard/images/dragIconH.gif to s3://qls-106791-6e80f797ca8dbc50-iotgss3bucket-ayuyec7jkxyb/images
/dragIconH.gif
upload: dashboard/images/dragIconRectBig.png to s3://qls-106791-6e80f797ca8dbc50-iotgss3bucket-ayuyec7jkxyb/
images/dragIconRectBig.png
upload: dashboard/app.js to s3://qls-106791-6e80f797ca8dbc50-iotgss3bucket-ayuyec7jkxyb/app.js
upload: dashboard/images/dragIconHBlack.gif to s3://qls-106791-6e80f797ca8dbc50-iotgss3bucket-ayuyec7jkxyb/i
mages/dragIconHBlack.gif
upload: dashboard/css/main.css to s3://qls-106791-6e80f797ca8dbc50-iotgss3bucket-ayuyec7jkxyb/css/main.css
upload: dashboard/images/dragIcon.gif to s3://qls-106791-6e80f797ca8dbc50-iotgss3bucket-ayuyec7jkxyb/images/
dragIcon.gif
upload: dashboard/images/.DS_Store to s3://qls-106791-6e80f797ca8dbc50-iotgss3bucket-ayuyec7jkxyb/images/.DS
_Store
upload: dashboard/.DS_Store to s3://qls-106791-6e80f797ca8dbc50-iotgss3bucket-ayuyec7jkxyb/.DS_Store
upload: dashboard/images/dragIconRectBig.svg to s3://qls-106791-6e80f797ca8dbc50-iotgss3bucket-ayuyec7jkxyb/
images/dragIconRectBig.svg
upload: dashboard/images/dragIconRectBigBlack.svg to s3://qls-106791-6e80f797ca8dbc50-iotgss3bucket-ayuyec7j
kxyb/images/dragIconRectBigBlack.svg
```

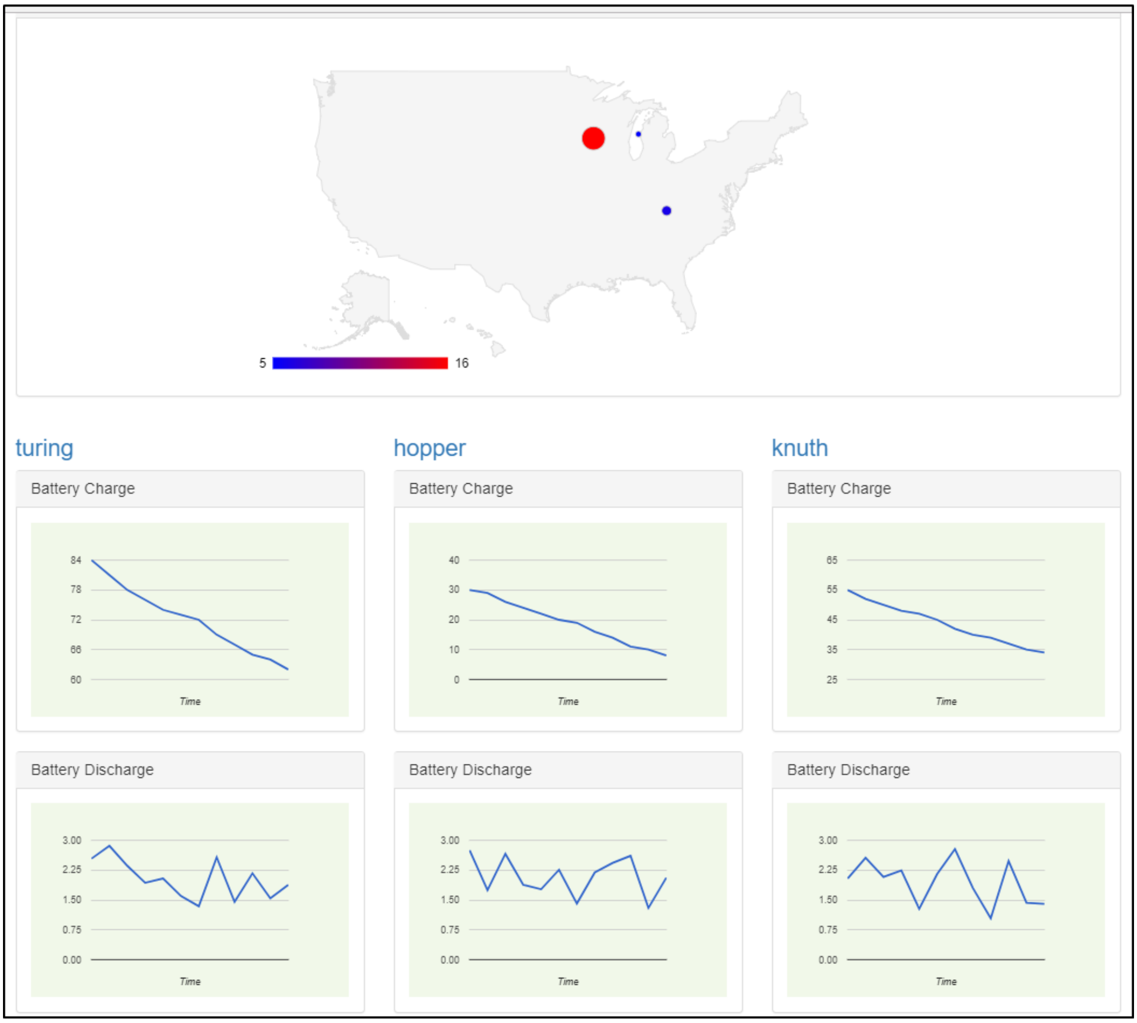
82. Restart the device simulator by entering:

```
python3 app.py
```

83. In a browser:

- Paste the S3 Bucket website endpoint URL that your copied earlier.
- Press **Enter**

You can visualize the devices and its related parameters using the dashboard.



# End lab

Follow these steps to close the console and end your lab.

84. Return to the **AWS Management Console**.
85. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
86. Choose End lab and then confirm that you want to end your lab.

# Conclusion

Congratulations! You now have successfully:

- Set up AWS IoT
- Ran a device simulator on EC2
- Processed and Visualized streaming data using a serverless app

# Additional Resources

For more information about AWS Training and Certification, see *https://aws.amazon.com/training/*.

*Your feedback is welcome and appreciated.*
If you would like to share any feedback, suggestions, or corrections, please provide the details in our *AWS Training and Certification Contact Form*.