Deploy an End-to-End IoT Application

February 2017

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Step 1: Set Up the Environment

Create an SSH Keypair

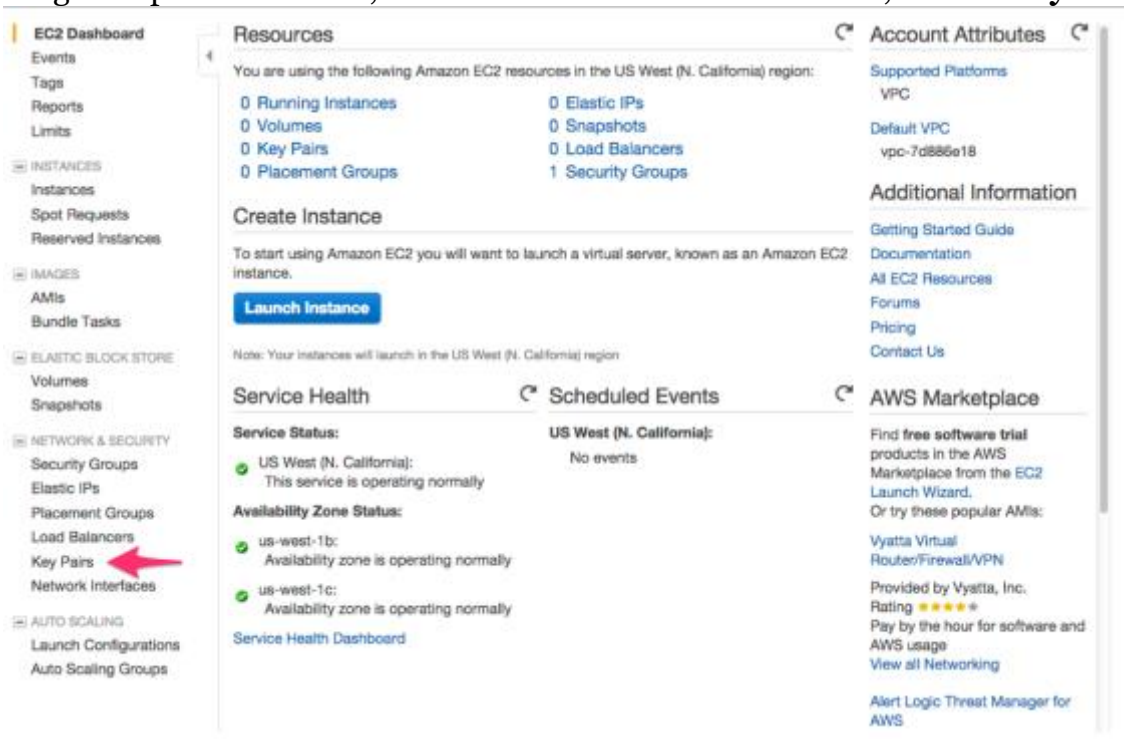
In this tutorial, an EC2 instance is used to simulate your IoT devices. Amazon EC2 uses public–key cryptography to encrypt and decrypt login information. Public–key cryptography uses a public key to encrypt a piece of data, such as a password, then the recipient uses the private key to decrypt the data. The public and private keys are known as a key pair.

To create your IoT environment, you will need to create an SSH keypair that will be used to access your device simulator EC2 instance. The following steps outline creating a unique SSH keypair to use in this lab.

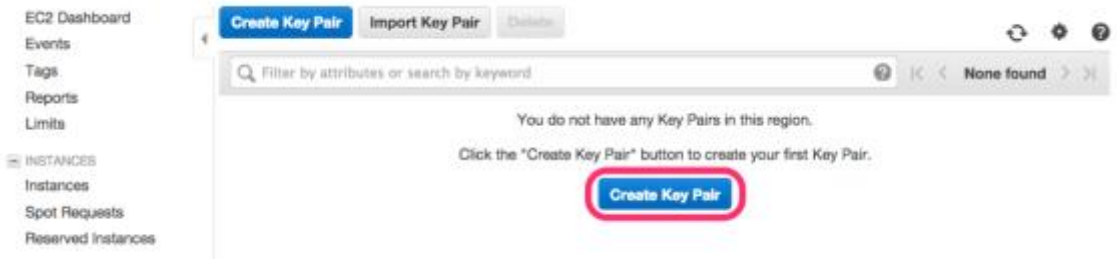
1. Sign into the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2.

2. In the upper-right corner of the AWS Management Console, confirm you are in the desired AWS region. Make sure to select a region that supports AWS IoT

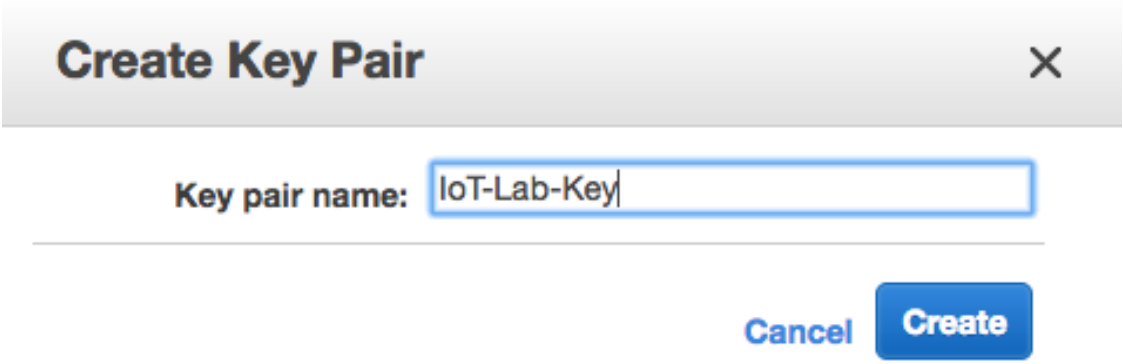
3. In the navigation pane on the left, under **NETWORK & SECURITY**, choose **Key Pairs**.



4. Choose **Create Key Pair**.



5. Enter a name for the new key pair in dialog box, and then choose **Create**.



The private key file is automatically downloaded by your browser. The base file name is the name you specified as the name of your key pair, and the file name extension is .pem. Save the private key file in a safe place.

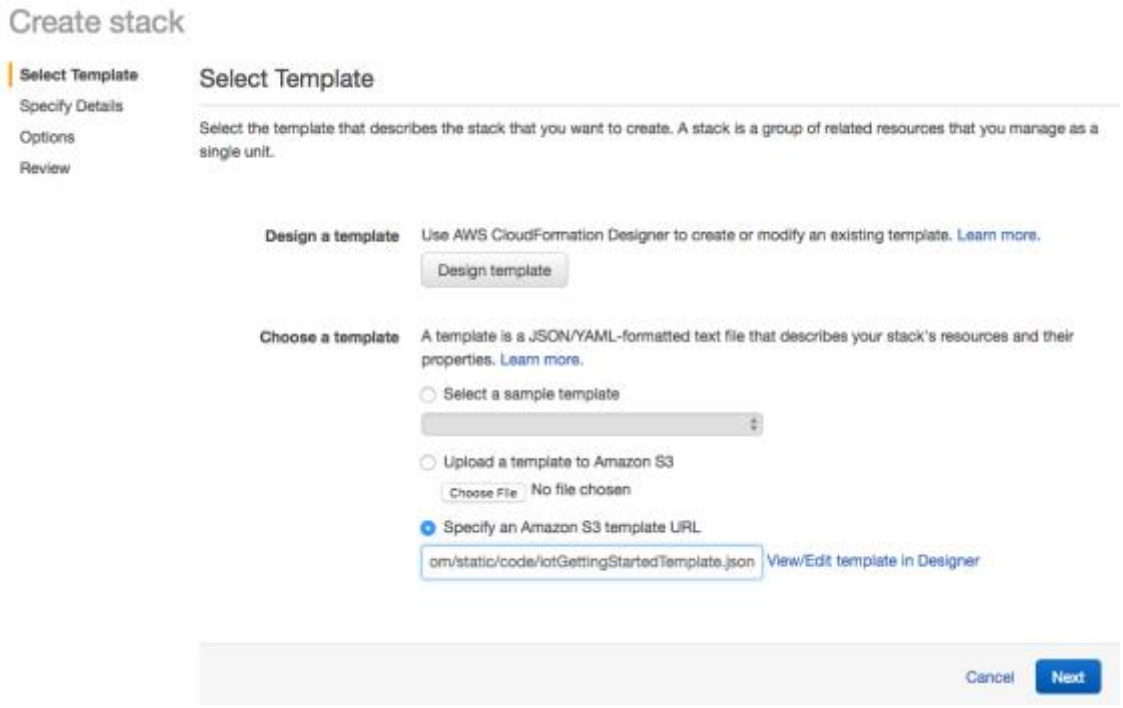
Important: This is the only chance for you to save the private key file. You'll need to provide the name of your key pair when you launch an instance and the corresponding private key each time you connect to the instance.

Deploy the AWS CloudFormation Template

AWS CloudFormation is a service that helps you model and set up your Amazon Web Services resources as code so that you can spend less time managing those resources and more time focusing on your applications that run in AWS. We have created a template (written in JSON) that defines the AWS resources that are needed for the sample IoT application. AWS CloudFormation then uses that template to provision and configure those resources for you. You don't need to individually create and configure AWS resources and figure out what's dependent on what; AWS CloudFormation handles all of that.

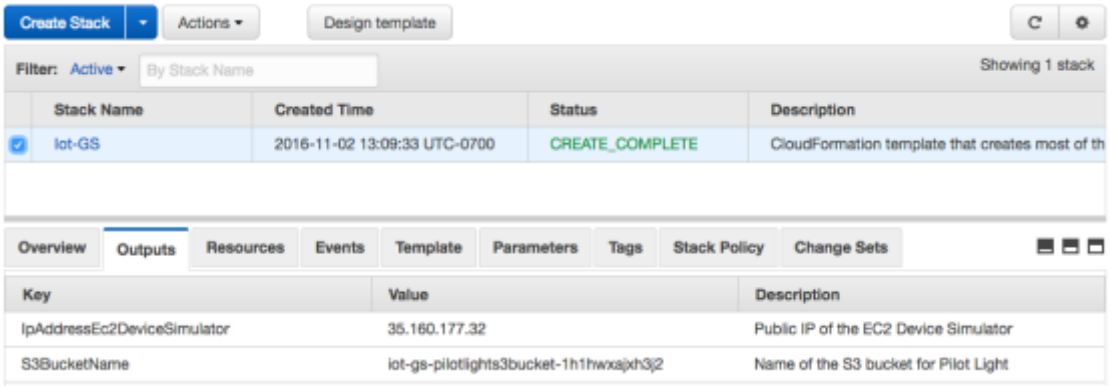
1. Sign in to the AWS Management Console
2. If this is a new AWS CloudFormation account, click **Create New Stack**. Otherwise, click **Create Stack**.

3. In the Template section, select **Specify an Amazon S3 Template URL** to type or paste the following URL for the IoT Getting Started template: https://s3.amazonaws.com/awsprojects-code/iotGettingStartedTemplate.json



4. Click **Next**.
5. In the **Stack name** field, enter a friendly name for the IoT stack. A shorter name here will improve readability in future modules (e.g. IoTGS).
6. In the **KeyName** field, select the keypair you created earlier. This will "key" your EC2 instance with the appropriate public key.
7. On the Options page, leave all defaults and click **Next**.
8. On the Review screen, confirm the configuration, check the box that says **I acknowledge that AWS CloudFormation might create IAM resources**, and click **Create**.
9. The environment can take a few minutes to provision completely. You can refresh periodically to monitor the creation of the stack. When AWS CloudFormation is finished creating the stack, the status will show *CREATE_COMPLETE*.
10. Select the check box beside your stack and then click on the **Outputs** tab below.

11. Note the **IpAddressEc2DeviceSimulator** Value. This is the public IP address of your IoT Device Simulator EC2 instance.



Confirmation: Connecting to your Instance

We will now confirm that we have access to the EC2 instance that will be simulating the IoT devices. Follow the instructions for your operating system.

Mac or Linux (OpenSSH)

By default, both Mac OS X and Linux operating systems ship with an SSH client that you can use to connect to your EC2 Linux instances. To use the SSH client with the key you created, a few steps are required.

1. Use the following command to set the permissions of your private key file so that only you can read it. Replace *IoT-GettingStarted-Key.pem* with the name of your SSH key pair.

   ```
   $ chmod 400 IoT-GettingStarted-Key.pem
   ```

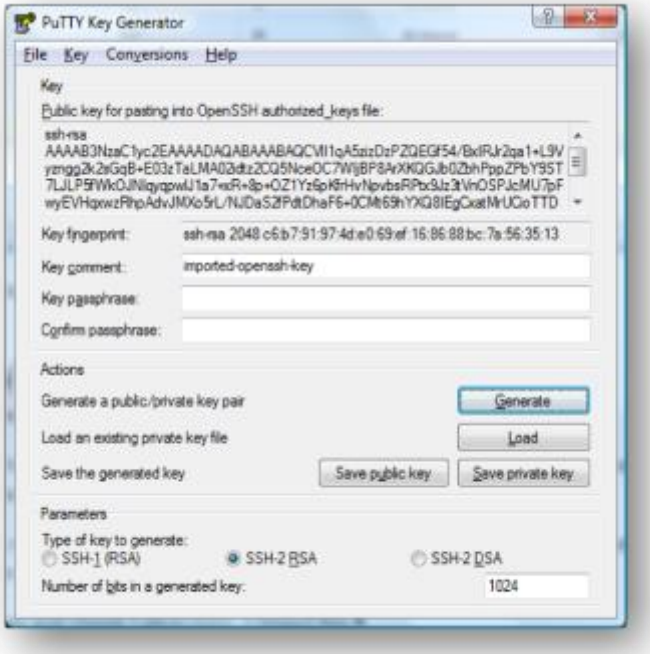
2. Use your private key when connecting to the instance. You will reference your private key file and the default user name which is ec2-user. The format of the ssh client is as follows: `$ ssh -i IoT-GettingStarted-Key.pem ec2-user@<IP Address of EC2 Host>`

3. Type *Yes* to accept the fingerprint. You should now be connected to your instance.

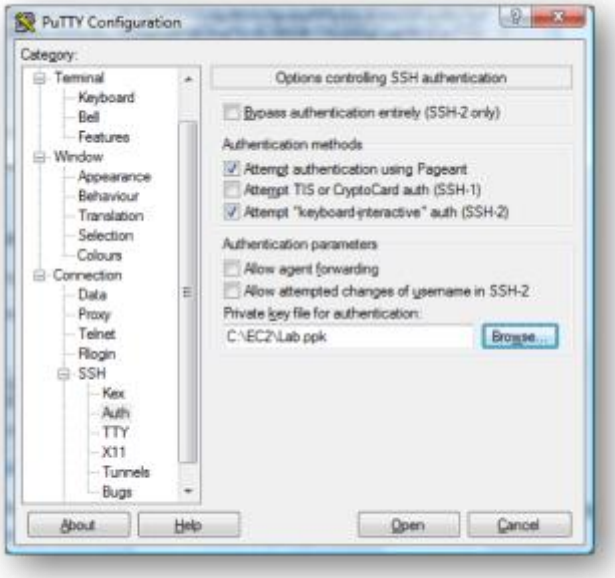
Windows (PuTTY)

This is a Windows-only step, because other operating systems have SSH built in. Download and install PuTTY. The single word “putty” in Google will return a list of download sites. Be certain that you install both PuTTY and PuTTYGen

1. Launch PuTTYGen and choose **Conversions** -> **Import Key**. Browse for the downloaded pem file (e.g., IoT-GettingStarted-Key.pem) and import the key. The result will look similar to this:

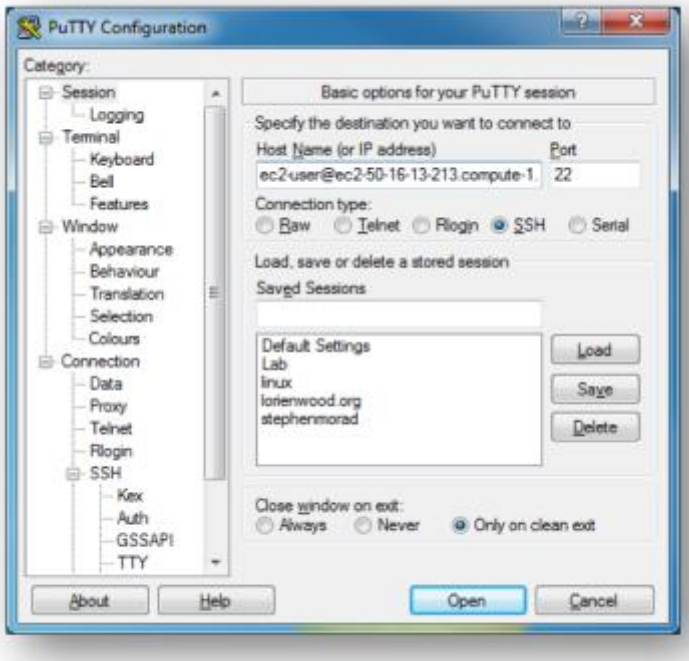


2. Save the key as the same file name with a .ppk extension. Click **File** -> **Save as Private Key**. Ignore the dialog that asks if you want to do this without a passphrase.
3. Close PuTTYGen.
4. Open PuTTY.
5. On the left menu expand **Connection** -> **SSH** and select the **Auth** sub-menu. Click **Browse** and select your PPK file from the previous step.

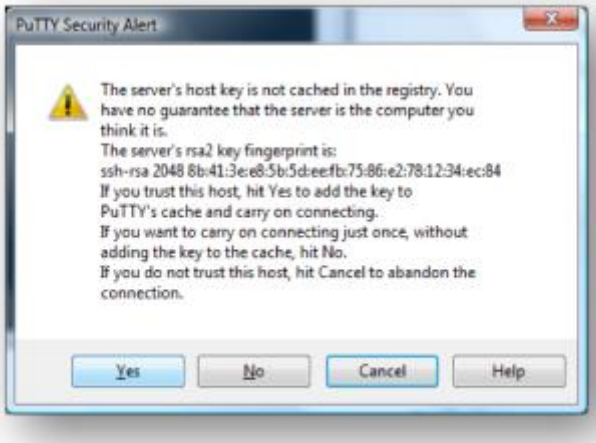


6. Select **Connection** and configure the *keepalive* to 60. This will prevent your SSH session from timing out.

7. Select **Session** on the left. In the **Host Name** box, enter *ec2-user@* followed by the IP address of your Simulator EC2 instance. (e.g. *ec2-user@ 50.17.175.10*).



8. Click **Yes** to confirm the fingerprint.



Note: The SSH fingerprint will eventually show up in the System Log and you can take that and compare it to protect against a man in the middle attack.

9. You should now be connected to your instance.

Step 2: Set Up AWS IoT

AWS IoT Overview

AWS IoT consists of the following components:

- **Message Broker** — Provides a secure mechanism for things and AWS IoT applications to publish and receive messages from each other. You can use either the MQTT protocol directly or MQTT over WebSockets to publish and subscribe. You can use the HTTP REST interface to publish.
- **Rules Engine** — Provides message processing and integration with other AWS services. You can use a SQL-based language to select data from message payloads, process the data, and send the data to other services, such as Amazon S3, Amazon DynamoDB, and AWS Lambda. You can also use the message broker to republish messages to other subscribers.
- **Thing Registry** — Sometimes referred to as the Device Registry. Organizes the resources associated with each thing. You register your things and associate up to three custom attributes with each thing. You can also associate certificates and MQTT client IDs with each thing to improve your ability to manage and troubleshoot your things.
- **Thing Shadows Service** — Provides persistent representations of your things in the AWS cloud. You can publish updated state information to a thing shadow, and your thing can synchronize its state when it connects. Your things can also publish their current state to a thing shadow for use by applications or devices.
- **Thing Shadow** — Sometimes referred to as a device shadow. A JSON document used to store and retrieve current state information for a thing (device, app, and so on).
- **Device Gateway** — Enables devices to securely and efficiently communicate with AWS IoT. Security and Identity service—Provides shared responsibility for security in the AWS cloud. Your things must keep their credentials safe in order to send data securely to the message broker. The message broker and rules engine use AWS security features to send data securely to devices or other AWS services.

Create the AWS IoT Resources

Now you will create the resources needed in the AWS IoT console. There are 4 components that will need to be created:

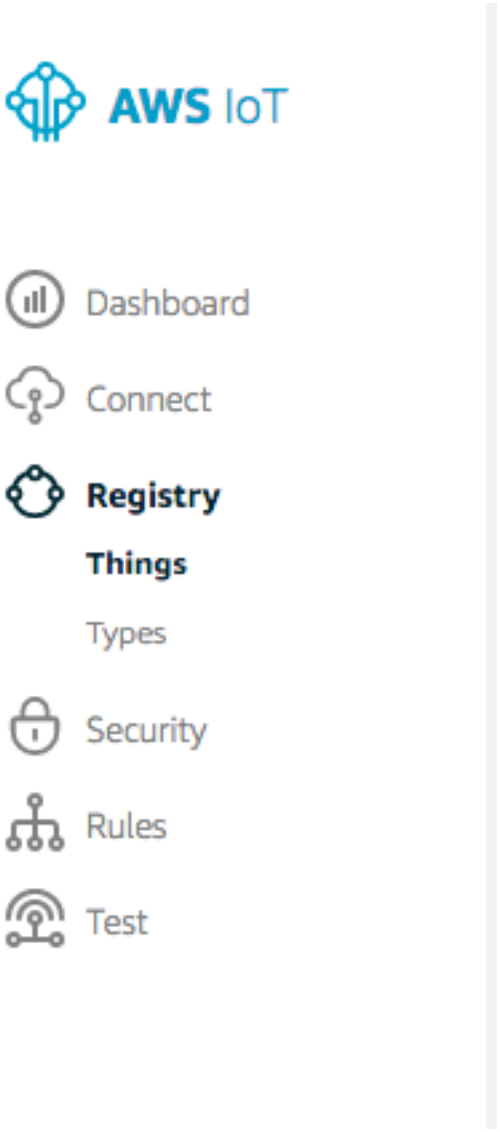
- **Thing** – A logical representation of a device stored in IoT's Registry. Supports attributes, as well as Device Shadows, which can be used to store device state & define desired state.
- **Policy** – Attached to Certificates to dictate was that Certificate (or rather, a Thing using that certificate) is entitled to do on AWS IoT.
- **Certificate** – 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 certs you will create.

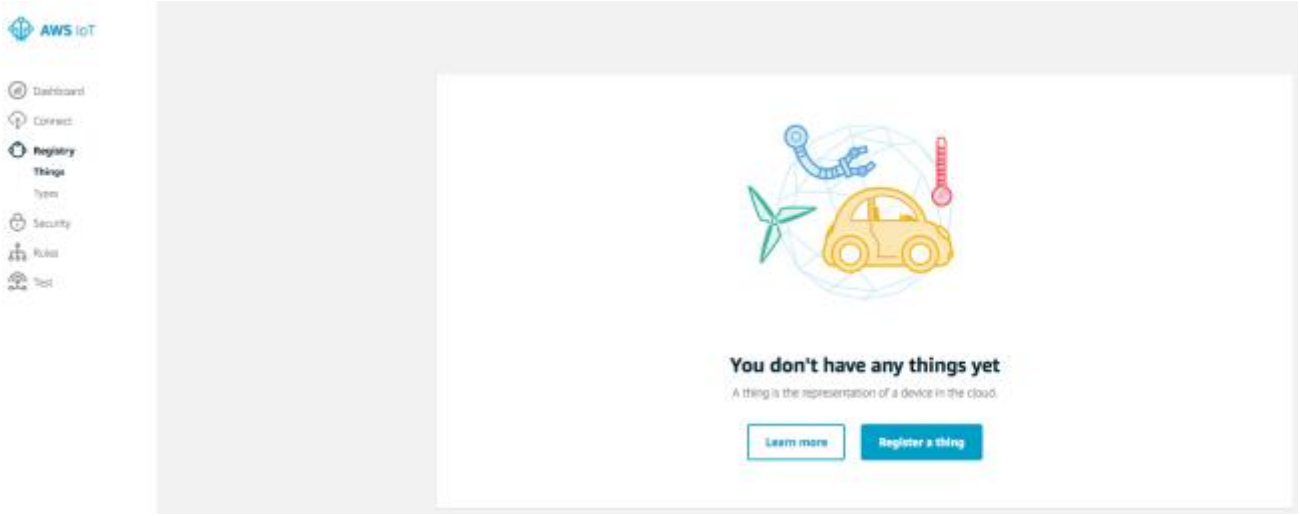
- **Rule** – Leverages AWS IoT's Rules Engine to dictate how messages sent from Things to AWS IoT are handled. You will configure rules that send data published to an MQTT topic to a variety of AWS Services.

Create an IoT Thing

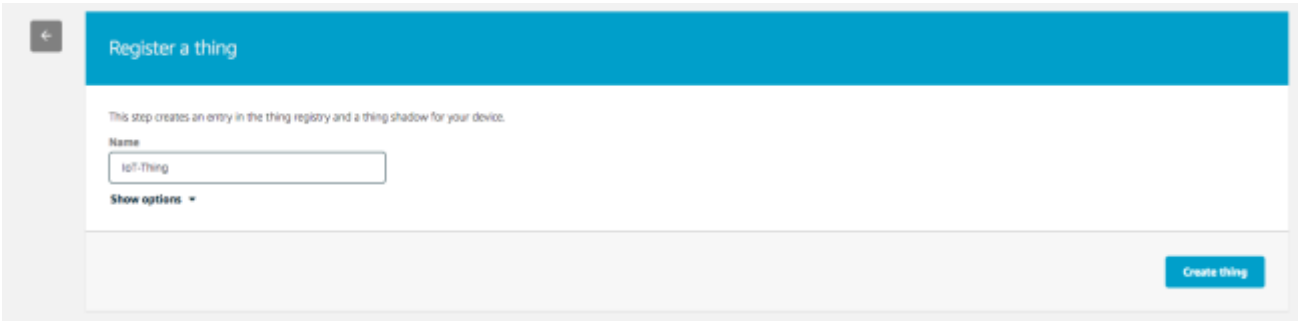
1. Sign in to the AWS IoT console.
2. On the left side of the console, click on **Registry**, then click **Things.**



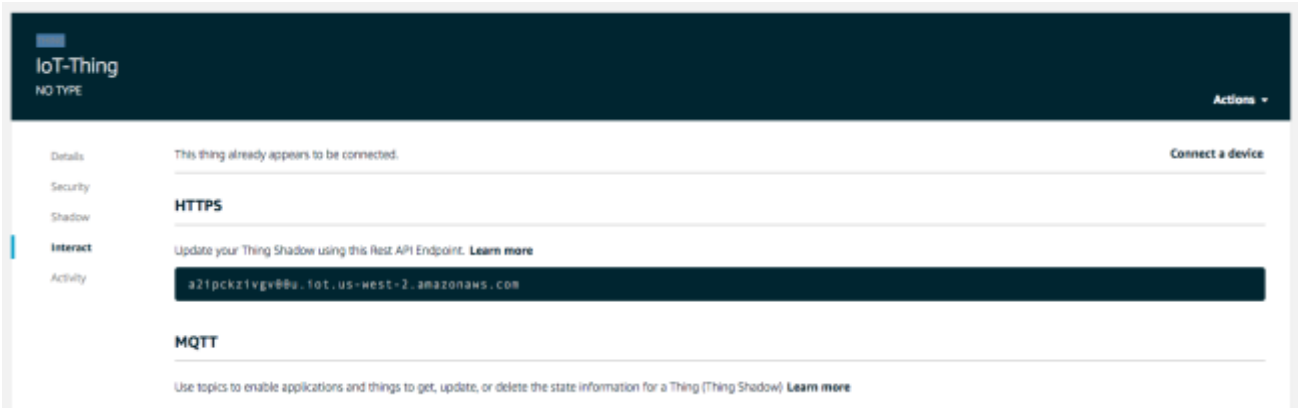
3. If you have never used the service before, then click **Register a thing**. Otherwise, **Create** will be in the top right corner



4. Provide a name for the Thing, and click **Create thing**.

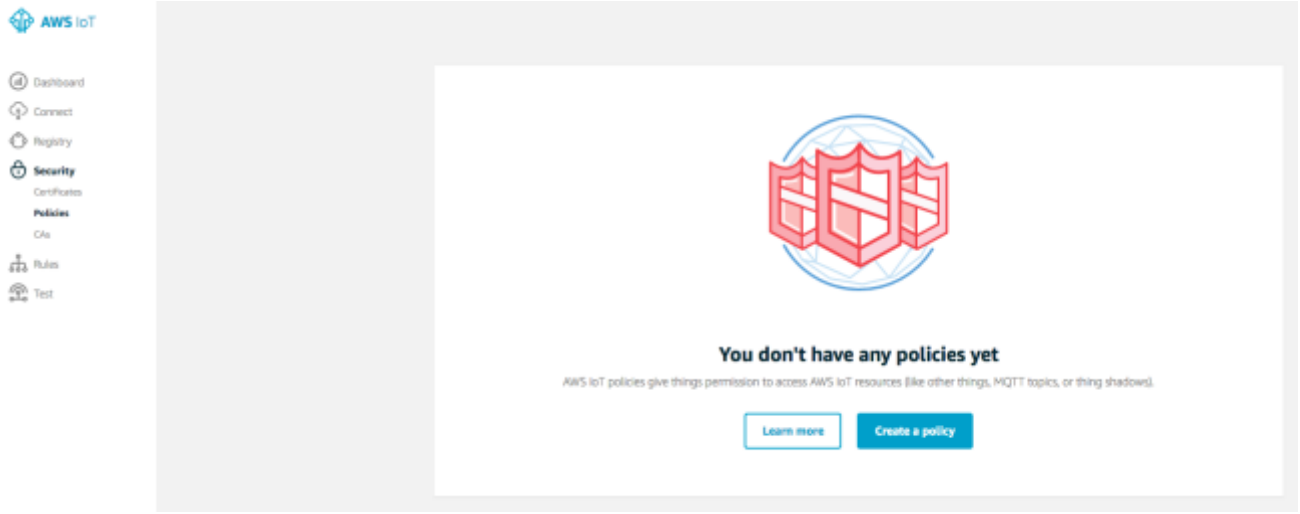


5. On the Thing Detail page, click on **Interact** in the left side menu. Capture the Rest API Endpoint (e.g. **a2ipckzivgv00u.iot.us-west-2.amazonaws.com)** listed under HTTPS. You will need this host name to configure the Device Simulator.

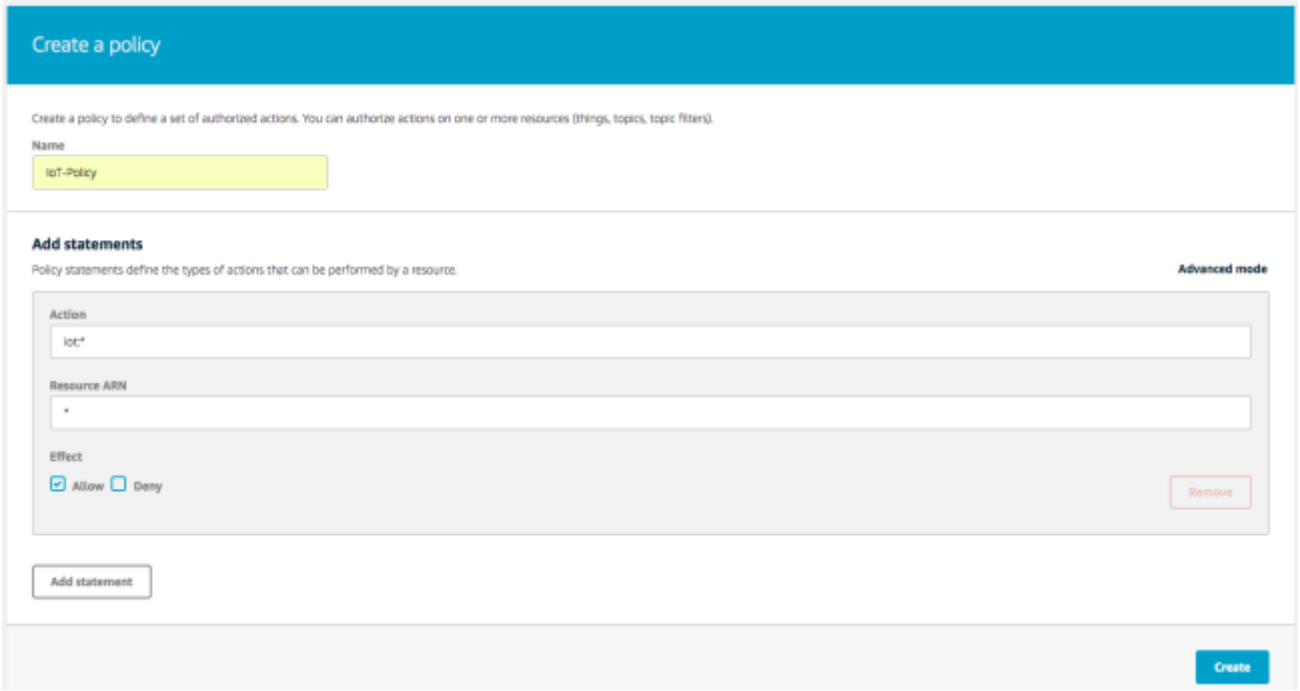


Create an IoT Policy

1. From the AWS IoT Console, select **Security**, and then **Policies**. If you have never used the service before, then click on **Create a Policy.** If you have never, used the service before, then click. If a previous policy exists, then you will click **Create** on the top right.



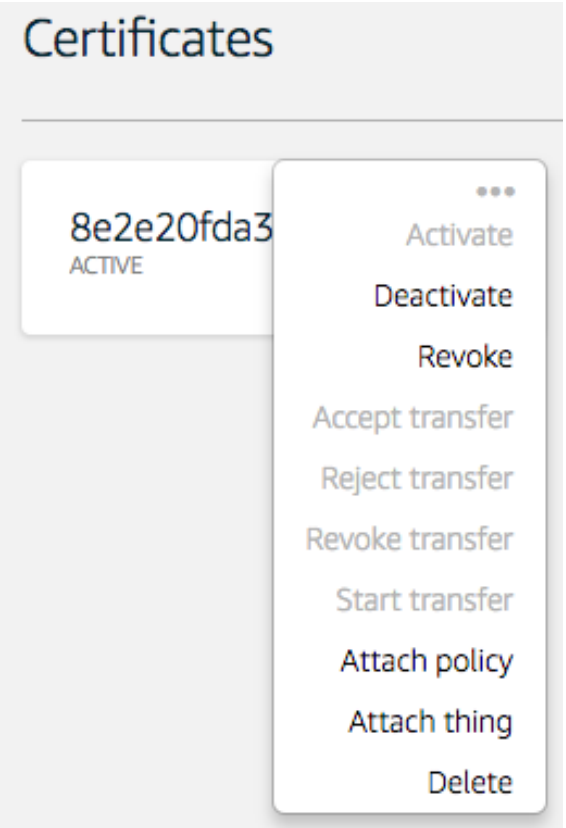
2. Give the Policy a **Name**.
3. Replace the **Action** with *iot:**
4. For the **Resource ARN**, replace the statement with *.
5. The **Create** button should turn blue. Click it to complete the policy creation.



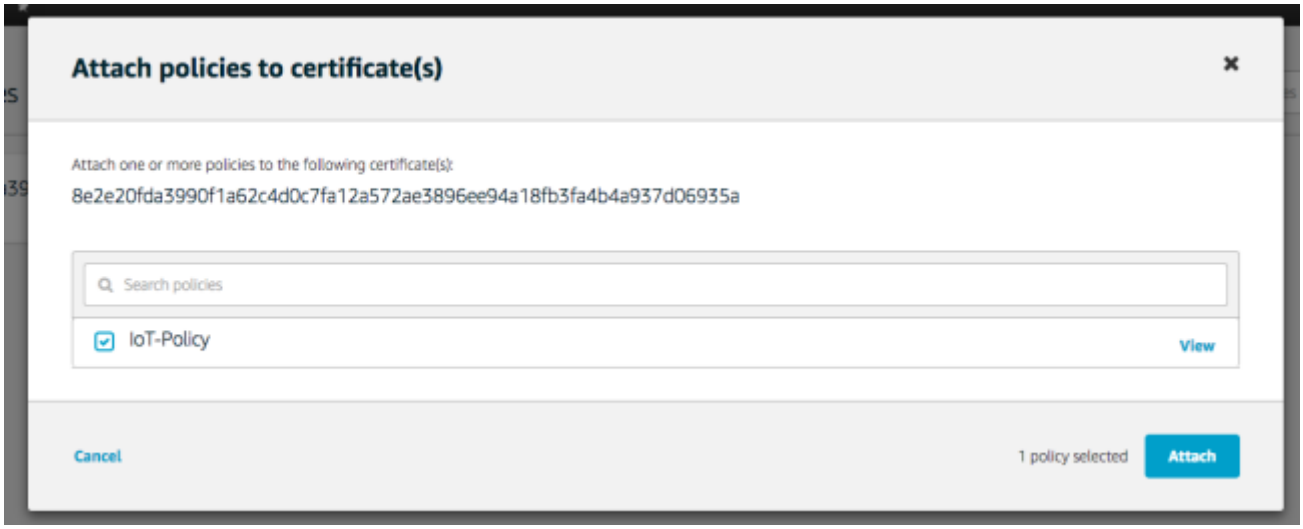
Create an IoT Certificate

While it is possible to create the device certificates in the AWS Management Console, we have created these during the CloudFormation stack creation via a script that runs on your EC2 device simulator instance.

1. SSH into the EC2 Instance.
2. Type *ls ~/certs* to view the certificates were created. You should have 3 files in the directory
   - certificate.pem.crt
   - private.pem.key
   - root-ca.pem
3. In the AWS IoT Console window, click on **Security** and **Certificates**. You should see your certificate. Confirm that it says **ACTIVE**.
4. Now the certificate must be associated with Thing and Policy that were created previously. Click on Options (**...**) on the top right of the certificate and click on **Attach policy**.



5. Select the policy you just created and click **Attach**.



6. Repeat the process, selecting **Attach a thing**. Select the Thing you created earlier.

Configure and Run the 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 to the appropriate AWS services based on the rule actions that you will configure throughout the workshop modules.

1. SSH into the EC2 instance.

2. Open the file *settings.py* in the editor of your choice. We will be using nano in this example.
   ```
   $ nano ~/settings.py
   ```

3. Replace the **HOST_NAME** with the host name REST API Endpoint of your Thing (e.g., *a2ipckzivgvoou.iot.us-east-1.amazonaws.com*).

4. Save the file. In nano, press **CTRL-X**, Type **Y** to save changes, and press **enter** to save the file as *settings.py*.

5. Start the device simulator.

   ```
   $ nohup python app.py &
   ```

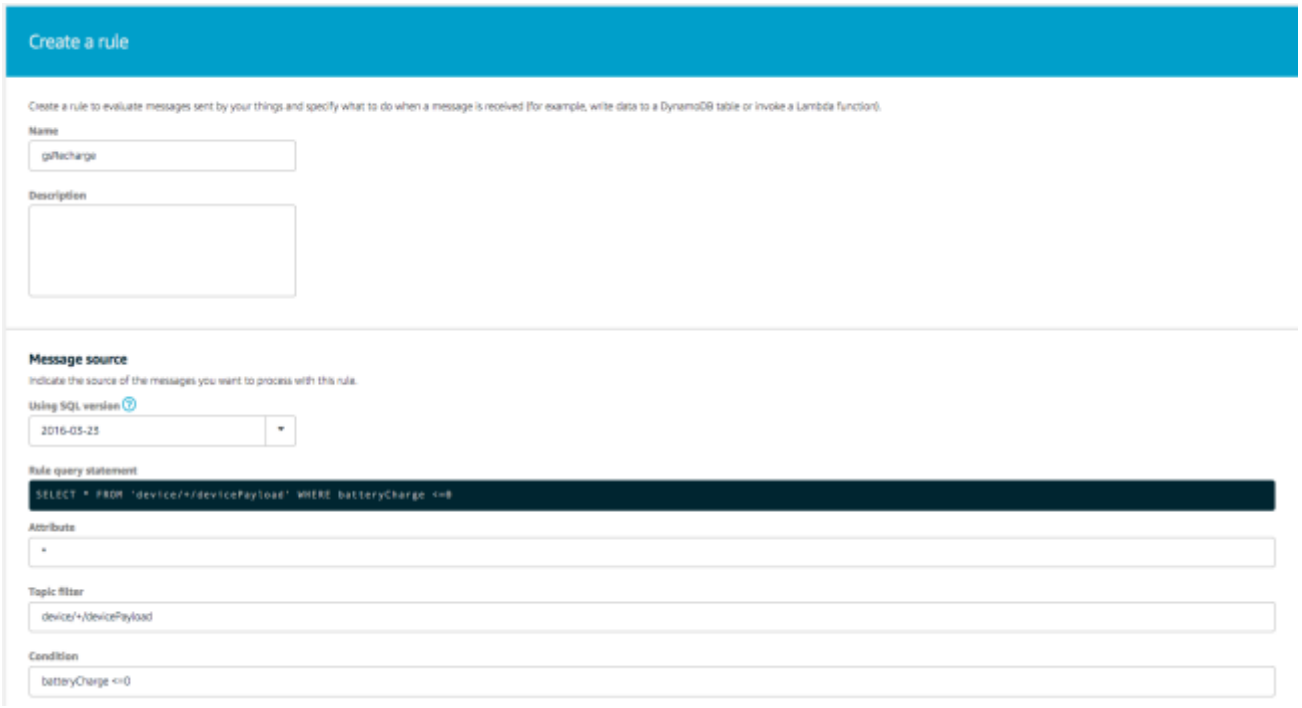
Create an IoT Rule and Action

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

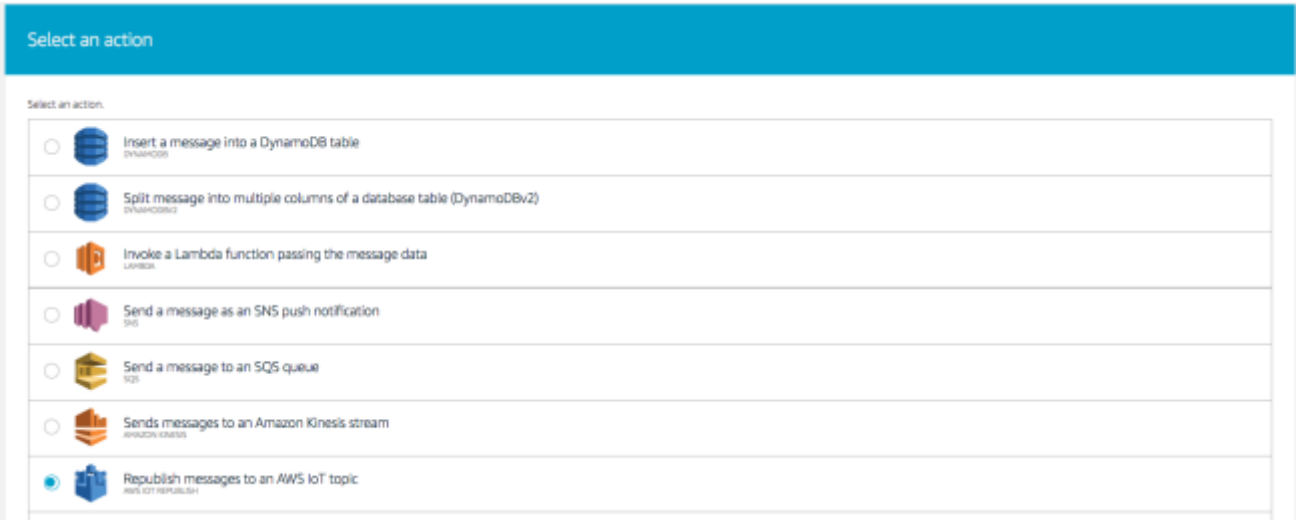
a rule action that will monitor the reported battery charge and publish a message to a new topic when it is time to recharge. The device is subscribed to this topic and will "take action" to recharge.

1. In the AWS IoT console, click on **Rules** on the left and then click **Create a Rule**.
2. Configure the rule as follows:

Field	Value
Name	gsRecharge
Description	*leave blank*
Attribute	*
Topic Filter	device/+/devicePayload
Condition	batteryCharge <=0



3. Click **Add action** and select **Republish messages to an AWS IoT topic**. Click on **Configure action**.

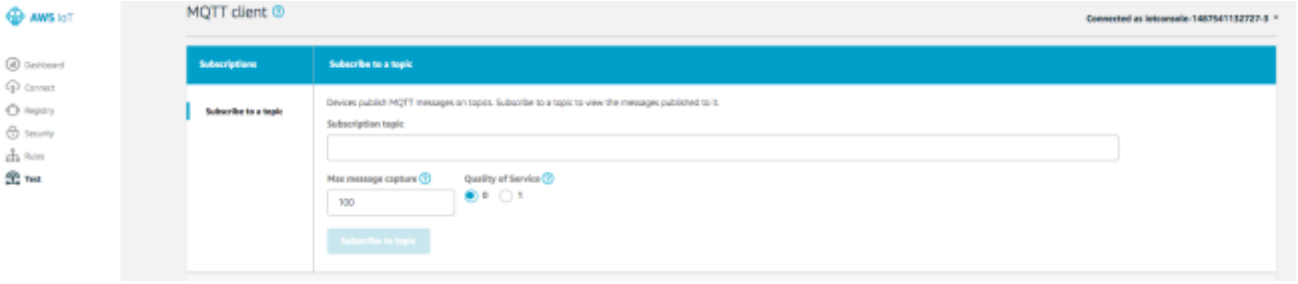


4. In the **Topic** dialog box, type *device/${topic(2)}/rechargeAlert*.
5. Click the **IAM role name** dropdown box and select the role that begins with the stack name you configured followed by **AwsIotRepublishRole**.
6. Click **Add action**.
7. Click **Create rule.**

Confirmation: View Device Messages with the AWS IoT MQTT Client

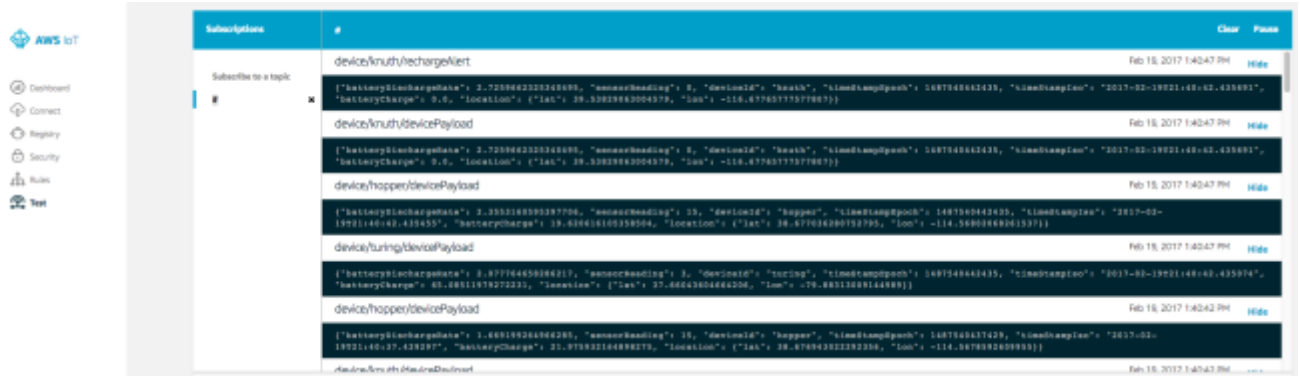
Devices publish MQTT messages on topics. You can use the AWS IoT MQTT client to subscribe to these topics to see the content of these messages. We will now use the AWS IoT MQTT client to confirm that the IoT messages are being sent back and forth between the devices and the AWS IoT Device Gateway.

1. In the AWS IoT console, click on **Test**.



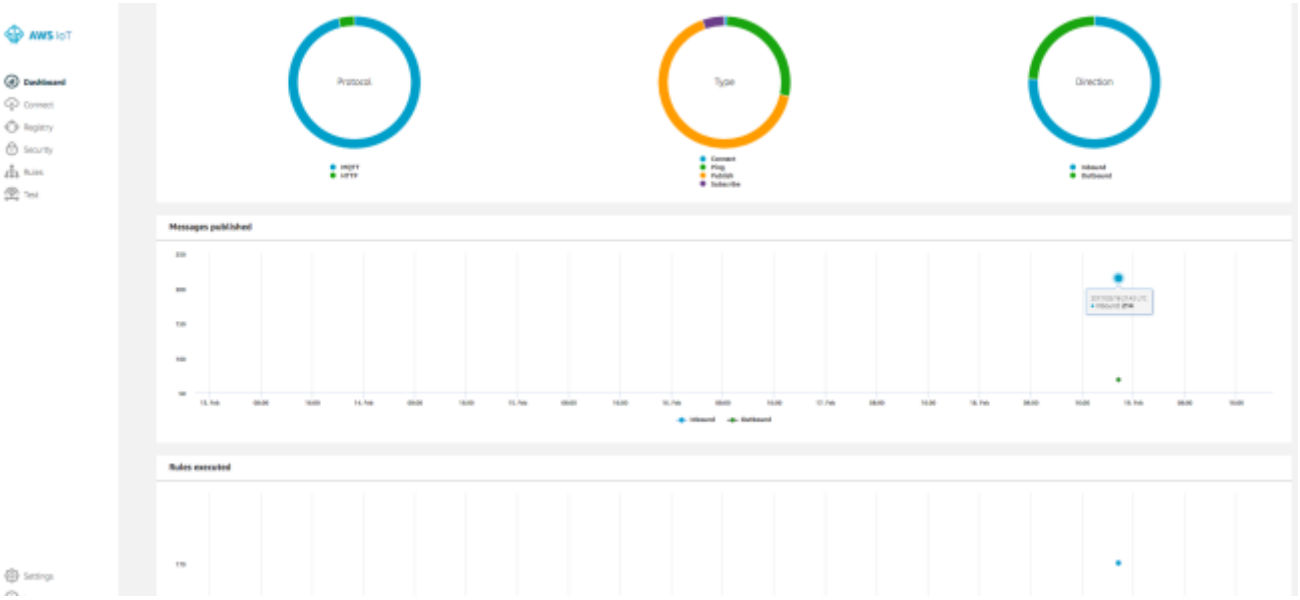
2. In the **Subscription topic** box, type the wildcard character # and click **Subscribe to topic**.
3. Click on the # symbol on the left pane under **Subscriptions.**

4. If the devices are successfully configured, you will see MQTT messages scrolling on the as pictured below.



5. Now we will confirm that our Recharge Rule is configured correctly. Click on **Subscribe to a topic**. In the **Subscription topic** box, type *device/+/rechargeAlert* and click **Subscribe**.
6. Confirm that you are seeing messages. This may take up to 2 minutes after the Rule Action is created.

7. Click on **Dashboard** on the left of the AWS IoT Console. You should see the counts of **Messages published** increasing. Take a moment to review the rest of the Dashboard.



Step 3: Process and Visualize Streaming Data

Dashboard Overview

In this module, we will be persisting time-series data from our devices in **Amazon DynamoDB** and then visualize that data with a real-time dashboard powered by a serverless API built with **Amazon API Gateway** and **AWS Lambda**.

Amazon DynamoDB: Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale. It is a fully managed cloud database and supports both document and key-value store models. You will create a set of AWS IoT Rule Actions to write device messages to your DynamoDB tables.

Amazon API Gateway: Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. For this module, your device data API has been created for you by CloudFormation, but you will have an opportunity to interact with your API configuration.

AWS Lambda: AWS Lambda lets you run code without provisioning or managing servers. With Lambda, you can run code for virtually any type of application or backend service - all with zero administration. Just upload your code and Lambda takes care of everything required to run and scale your code with high availability. For this module, the Lambda function has already been created for you and integrated with API Gateway, but in Module 5 you will create a Lambda function of your own.

Note on architecture: In this section you'll be building a dashboard that renders messages from your devices by pulling data from an Amazon DynamoDB table via a serverless API. An alternative pattern for this would include subscribing the dashboard to MQTT topics via WebSockets. This tutorial uses Amazon DynamoDB to illustrate both AWS IoT **and** serverless architectures.

In this section, you will create additional IoT rule actions to send device data to Amazon DynamoDB. There are 3 IoT devices in our setup, the devices are named: turing, hopper, and knuth. We will also set up a static website using Amazon S3 that will serve as a real-time dashboard allowing visualization of the device payload data. Each device sends the following JSON Payload to the AWS IoT Gateway every 5 seconds:


```
{
"Items": [
{
    "payload":
    {
        "timeStampIso": "2016-09-10T22:35:06.732142",
        "batteryDischargeRate": 1.8513595745796365,
        "location": {
            "lon": 99.13799750347447,
```

```
            "lat": 41.79078293335809
        },
        "timeStampEpoch": 1473546906732,
        "numVal": 5,
        "deviceId": "Hopper",
        "batteryCharge": 96.29728085084074
    },
    "deviceId": "Hopper"
},
{
    "payload":
    {
        "timeStampIso": "2016-09-10T22:35:06.732247",
        "batteryDischargeRate": 1.5474383816808281,
        "location": {
            "lon": 114.60811541199776,
            "lat": 41.15078293335809
        },
        "timeStampEpoch": 1473546906732,
        "numVal": 10,
        "deviceId": "Turing",
        "batteryCharge": 10.248573862511861
    },
    "deviceId": "Turing"
},
{
    "payload":
    {
        "timeStampIso": "2016-09-10T22:35:06.732010",
        "batteryDischargeRate": 1.5634519980188246,
        "location": {
            "lon": 95.74692230611322,
            "lat": 46.82078293335809
        },
        "timeStampEpoch": 1473546906732,
        "numVal": 6,
        "deviceId": "Knuth",
        "batteryCharge": 90.61928801188708
    },
    "deviceId": "Knuth"
}
],
"Count": 3,
"ScannedCount": 3
}
```

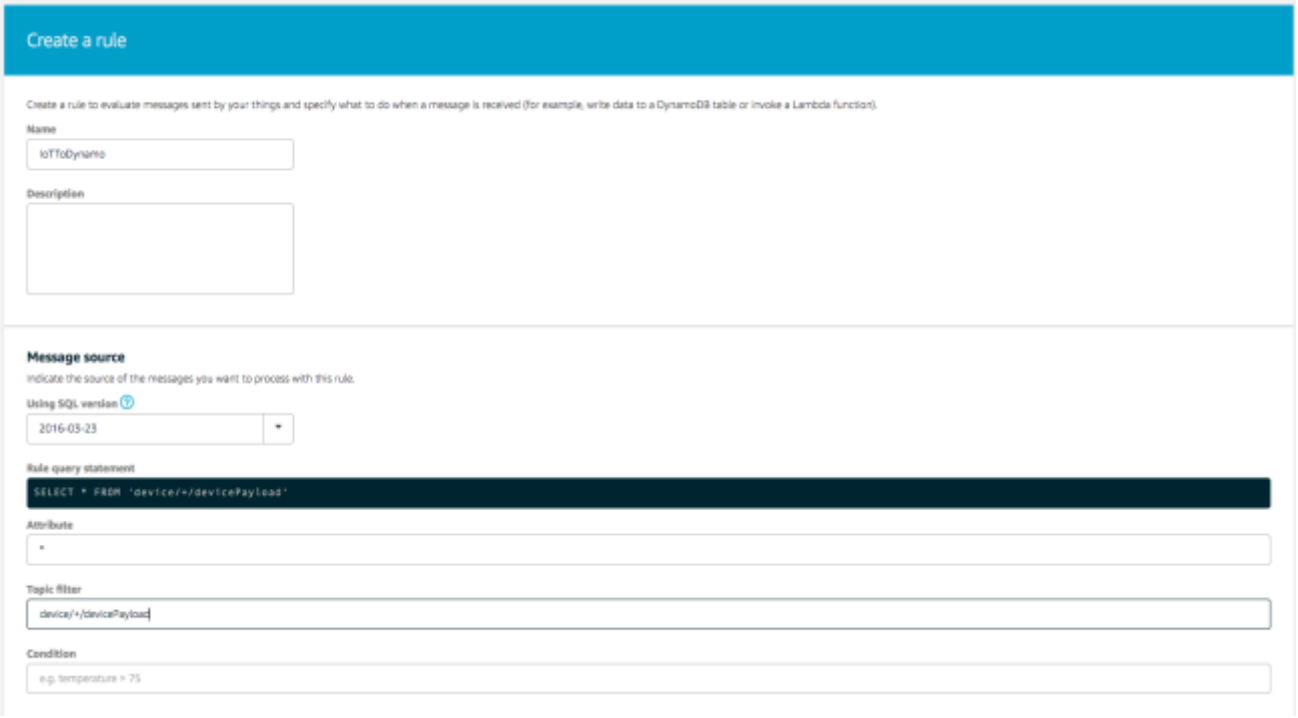
Create the IoT Rules and Actions

We will create a rule with two actions, to query the incoming messages and capture the payload section. The first rule will write time series data from devices to DynamoDB table called **IoTDynamoTimeSeriesTable**. The second rule will write the latest received messages to a DynamoDB table called **IoTDynamoDeviceStatusTable**.

Note: The actual DynamoDB table names will be prefixed by the name you chose for your CloudFormation stack, e.g *IoTGS-DynamoTimeSeriesTable*.

1. In the AWS IoT console, click on **Rules** on the left and then click **Create**.
2. Enter the following parameters. This rule includes a query statement that will capture the payload section from the incoming messages.

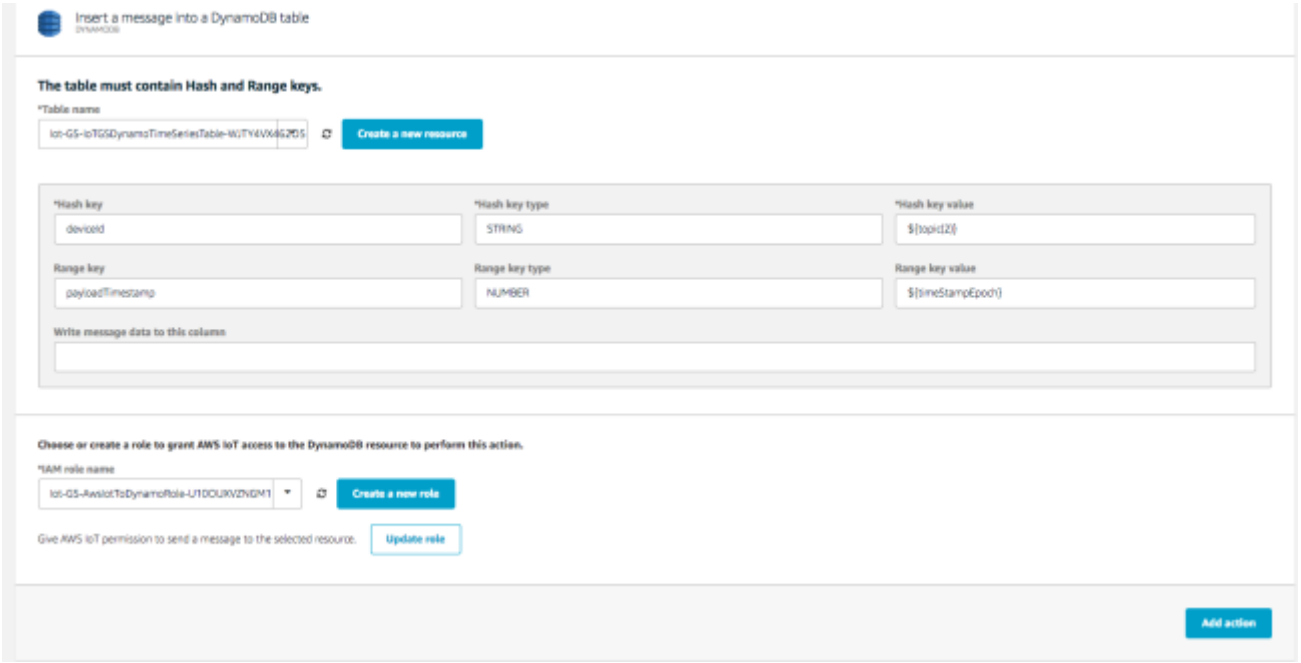
Field	Value
Name	IoTToDynamo
Description	*Leave blank*
Attribute	*
Topic filter	device/+/devicePayload



3. Click **Add action.**
4. Select **Insert a message into a DynamoDB table** and then click **Configure action**.
5. Click the **Table name** field and select the table whose name contains **TimeSeriesTable**.

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

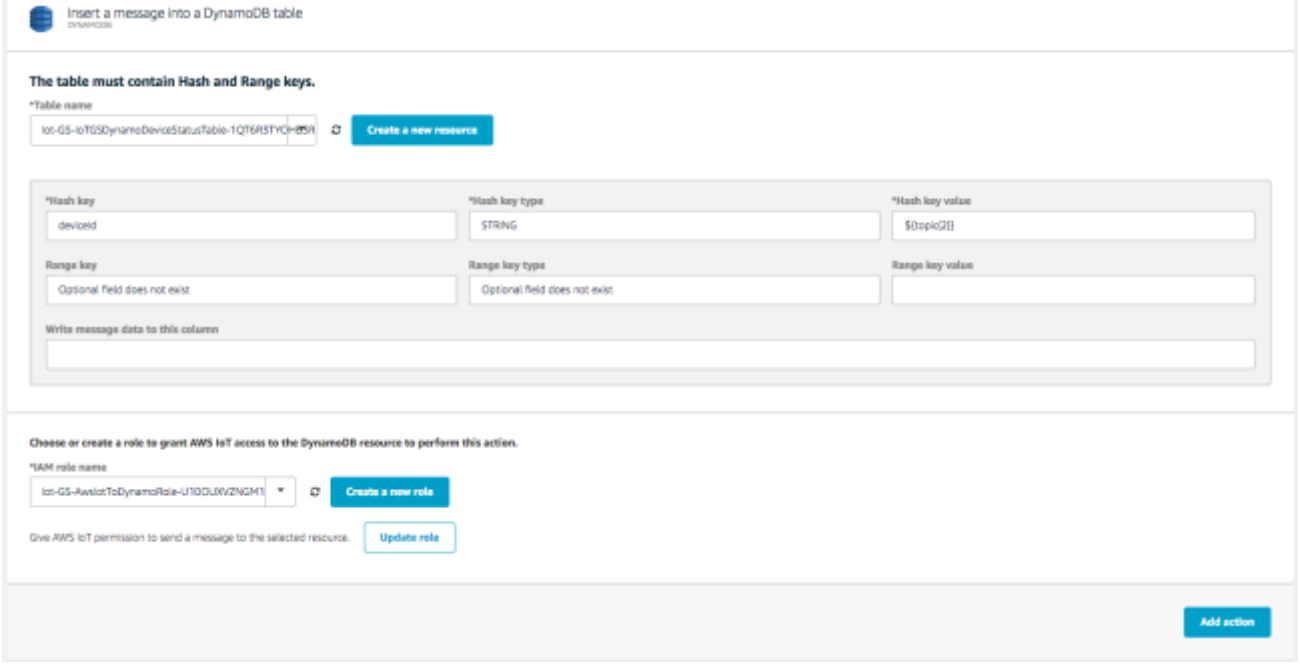
Field	Value
Hash key value	${topic(2)}
Range key value	${timeStampEpoch}
Role name	<stack-name>-AwsIotToDynamoRole-<random-number>



7. Click **Add action**.

8. We will also be creating a table of connected devices using this same IoT rule that reports the last reported value from the devices. We will create an additional action to accomplish this. Click **Add action** and repeat the process with the following values.

Field	Value
Table name	<stack-name>-IoTDynamoDeviceStatusTable
Hash key value	${topic(2)}
Range key value	*Leave empty*
Role name	<stack-name>-AwsIotToDynamoRole-<random-number>



9. Click **Add action**.

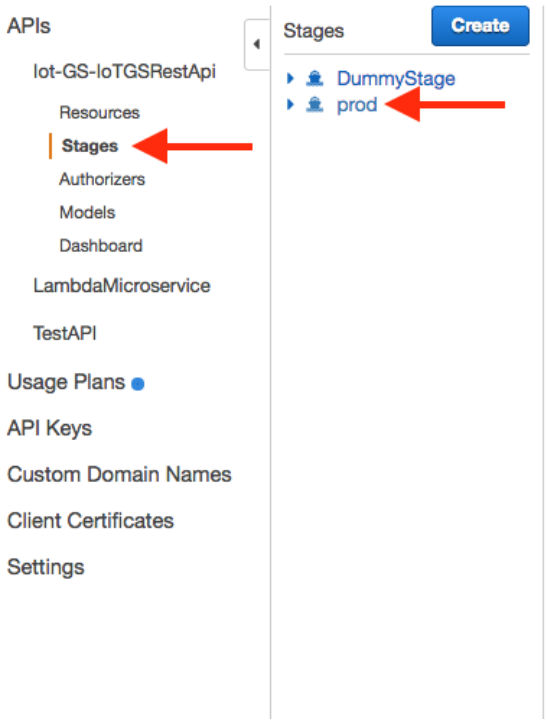
10. Click **Create rule.**

Rule and actions are now configured; in the next step you'll enable the APIs that read the DynamoDB table and return devices data in an API GET method.

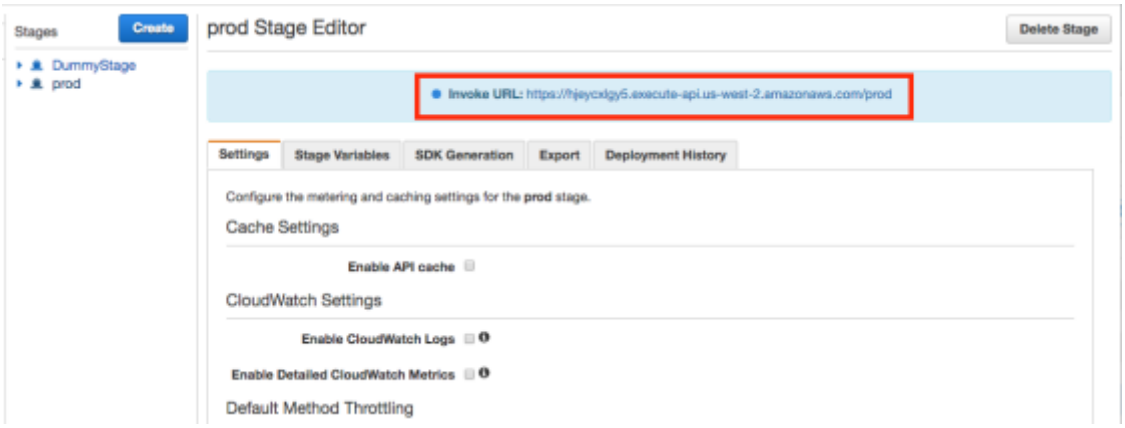
Test the APIs

In this section you will test that the API works, you will 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 when you provisioned the CloudFormation template. In the next section, you will "hook" the APIs into a dashboard to visualize the data in the website.

1. In Amazon API Gateway, a stage defines the path through which an API deployment is accessible. The CloudFormation template already configured a production stage called 'prod'. In the API Gateway console click **Stages** and then **prod**, and review the current API configuration.



2. Click on the link next to Invoke URL:



You should see devices data in a JSON format, refresh the page every few seconds and notice that data changes

3. Save the URL endpoint, you will need it in the next section.

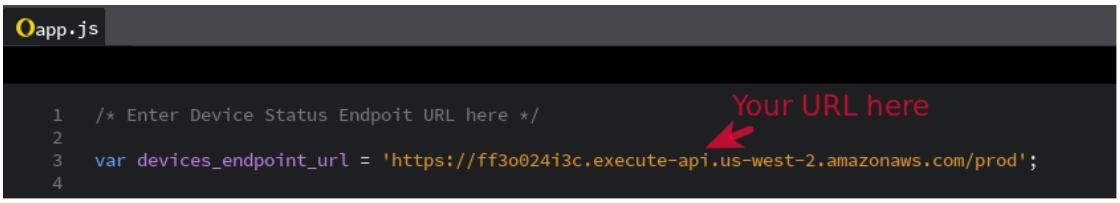
Deploy the Real-Time Dashboard

In this section, you will 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 will download the dashboard code and update API endpoint, and then you will setup a static website on S3 to host your dashboard.

1. SSH to the EC2 instance
2. Open app.js for editing in nano. $ nano ~/dashboard/app.js.

At the top of the file set the **devices_endpoint_url** to the API endpoint you'd created in the previous stage.



3. Save the file. In nano, press **CTRL-X**, Type **Y** to save changes, and press **enter** to save the file.

Host a Static Website on Amazon S3

In this step, you will configure a static website on 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.

1. From the AWS console, select **Services** and then **S3**.
2. The CloudFormation template already created a bucket to hold the Dashboard, the bucket name is: **<CloudFormation Template Name>-iotgss3bucket-<Random Number>**. Click on the bucket name, then click on **Properties**.
3. Select **Permissions** and **Add bucket policy**.

4. Paste the section below in the Bucket Policy Editor. Replace <*bucket-name*> with the name of your S3 bucket with the The following policy enables anyone to read the bucket (execute GET HTTP command):

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

5. Click **Save**.

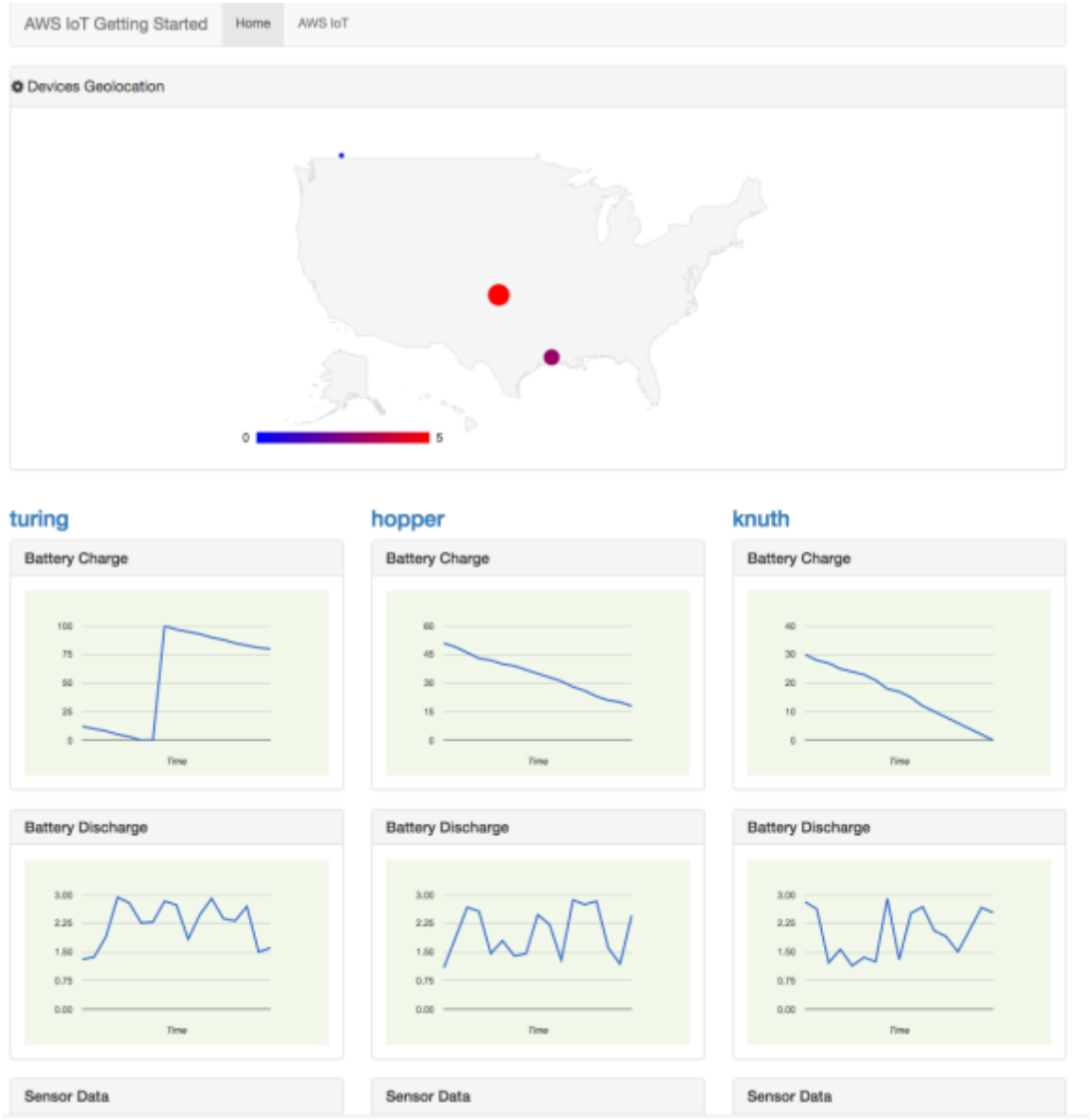
6. Now you can enable static website hosting on the bucket, select the **Static Website Hosting** and check the **Enable website hosting** radio box. In the **Index Document** field enter *index.html* and click **Save**.

7. Your dashboard will be available on the bucket's **Endpoint**. Save this endpoint - you will use it shortly.

8. Copy the dashboard code to the S3 bucket. In a terminal window, make sure you are in the Dashboard directory. Copy the content of the directory to the S3 bucket, you will use the AWS CLI for this. type the following command:

```
aws s3 sync ~/dashboard s3://<bucket-name>
```

9. In a browser, paste the S3 bucket endpoint to access your dashboard.


AWS IoT Getting Started
Home
AWS IoT
Devices Geolocation
0
5
turing
hopper
knuth
Battery Charge
Battery Charge
Battery Charge
Battery Discharge
Battery Discharge
Battery Discharge
Sensor Data
Sensor Data
Sensor Data

Step 4: Clean Up the Environment

We will now delete all of the AWS resources that were used during this session.

Clean up IOT Resources

1. Sign in to the AWS IoT console.
2. Click on **Rules**.
3. For each rule, Click **...** and select **Delete**.
4. In the confirmation window, click **Yes, continue with delete**.
5. Click on **Security** and then **Policies**
6. Click **...** and select **Delete**.
7. In the confirmation window, click **Yes, continue with delete**.
8. Under **Security**, click **Certificates**.
9. Click **...** and select **Delete**.
10. In the confirmation window, click **Yes, continue with delete**.
11. Under **Registry** click **Things**.
12. Click **...** and select **Delete**.
13. In the confirmation window, click **Yes, continue with delete**.

Clean up the S3 bucket

1. Open the AWS S3 Console.
2. Right-click on the IoT bucket that we have been using and click **Empty Bucket**.
3. You will need to type the name of bucket into the confirmation window. You can cut and paste for convenience. Click **Empty Bucket**.

Delete the CloudFormation Stack

1. Open the AWS CloudFormation Console.
2. Check the box next to your IoT stack.
3. Under **Actions**, click **Delete stack** and confirm. This may take up to 5 minutes to complete.

Additional Resources

To learn more about the concepts introduced in this tutorial, refer to the following resources:

- AWS IoT Documentation

- AWS IoT Developer Resources
- Deep Dive on AWS IoT
- Anomaly detection with AWS IoT and Lambda
- Building a Hardware Solution for Constrained Devices