What Is AWS Icebreaker?

AWS Icebreaker is a service that enables secure, bi-directional communication between internet-connected things (sensors, actuators, devices, applications, etc.) and the cloud over MQTT and HTTP. You can think of Icebreaker as a message processing engine. It receives messages from internet connected "things" and processes those messages. This includes recording, transforming, augmenting, or routing messages to AWS, other web services and applications. Manufacturers, application developers, and enterprises can use Icebreaker to extend the onboard capabilities of physical products by using the cloud to execute logic, communicate with other products/services, and process telemetry data. End users can control their physical devices from smart phone apps.

The following diagram illustrates a high-level view of the Icebreaker service:

![Diagram of AWS Icebreaker service](image)

You can interact with Icebreaker in a number of ways:

- The Icebreaker Console allows you to configure AWS Icebreaker services within a graphical environment
The Icebreaker Command Line Interface (CLI) allows you to configure AWS Icebreaker services from the command line. The Icebreaker SDKs allow you to write applications on top of Icebreaker. The Icebreaker Thing SDK allows you to write applications in C that run on internet-connected things.

Things are any clients such as micro controllers, sensors, actuators, mobile devices, or applications that use Icebreaker to connect to the AWS cloud. The Thing SDK makes it simple to write code running on Internet connected things to communicate with the Icebreaker service.

There are essentially three types of client applications that interact with Icebreaker:

- Embedded applications running on Internet connected devices
- Companion applications running on mobile devices or on the web.
- Server applications

Embedded applications are written in C with the Icebreaker thing SDK. They enable your device to send MQTT messages to and receive MQTT messages from Icebreaker. They define what information your devices send to Icebreaker and how they respond to messages received from Icebreaker.

Companion applications are written with the Icebreaker SDKs. These applications allow you to remotely control your devices.

Server applications query Icebreaker for information about your things and process and display the information. A device dashboard showing all active devices is an example of a server application.

Authentication is provided by X509 certificates or AWS Cognito Identities. Authorization is provided by Icebreaker roles and IAM roles.

**Getting Started with AWS Icebreaker**

There are three ways to interact with the Icebreaker service:

- Using the Icebreaker Console
- Using the Icebreaker CLI
• Using the Icebreaker SDKs

The following sections will describe using the icebreaker console in more detail. If you want to use icebreaker CLI, you can refer to AWS-Icebreaker-User-Guide.pdf

**Using the Icebreaker Console**

The Icebreaker console can be found at: Icebreaker Console. The console is divided up into three sections:

• Certificates
• Rules and Integrations
• Access and Policies

These sections are selectable by clicking on the appropriate icon in the upper left hand corner of the console.

**Certificates**

The certificates section allows you to submit a certificate signing request to generate a new certificate. It also allows you to activate, transfer, deactivate, or revoke an existing certificate.

**Rules and Integrations**

The rules and integrations section allows you to add a new rule and view your existing rules.

**Access and Policies**

The access and policies section allows you to add new Icebreaker policies and view existing Icebreaker policies.

**Signe to Icebreaker console**

If you didn’t have AWS account, you need go to the http://aws.amazon.com/ and register an account
Create a Thing in the Thing Registry

Go the Icebreaker console, click the Dashboard.

Create a thing, e.g: temperature.

The web page will appear “Connect device” button, Click it.
Choose which SDK you want to use. If you use Beaglebone Green, we recommend select NodeJS.

If you use Seeeduino cloud, we recommend select Arduino.

Create new cert and new policy.

Download the three files. Then power on your board.
Amazon provides the github maintain the code. Next page is the latest code introduction. We also provide on-board code to test the icebreaker.

**AWS MQTT publish and subscribe**

Upload xxxxx-private.pem.key and xxxxxxx-certificate.pem.crt to prodCerts folder.
Rename 774fd93c59-private.pem.key as device_identity.pem.

Rename 774fd93c59-certificate.pem.crt as device_key.pem.

If you use mqtt protocol, you need modify the port to 8883.

```
    /var/lib/cloud9/IotSdkJS/src/aws_iot.js
    this.client_params.port = checkParams(client_params.port, 8883);
```

If you use http protocol, you need modify the port to 443. and default port is 443.

Run the Helloworld.js example. You can see publishes and subscribe successfully.

```
    e.g Grove temperature sensor.

    Connect the Grove temperature sensor to BBG, Modify the time_pub.js file.
```
var aws_iot = require("./src/aws_iot.js");
var net = require('net');
var exec = require('child_process').exec;
var HOST = '127.0.0.1';
var PORT = 7000;
var temperature = 25;
var client_params = {
    host: 'mqtts://g.us-east-1.pb.iot.amazonaws.com',
    clientId: 'sdk_pub2'
};

// Create a server instance, and chain the listen function to it
net.createServer(function(socket) {
    console.log('CONNECTED: ' + socket.remoteAddress + ':' + socket.remotePort);
    // Add a 'data' event handler to this instance of socket
    socket.on('data', function(data) {
        //console.log('DATA ' + socket.remoteAddress + ':' + data);
        temperature = data;
        socket.write('This is your request: "' + data + '"
        });
    // Add a 'close' event handler to this instance of socket
    socket.on('close', function(data) {
        console.log('Socket connection closed... ');
    });
}).listen(PORT, HOST);

iot_client = new aws_iot(client_params);
iot_client.connect();
exec('python Grove_Starter_Kit_for_BBG/Python-App.py', function(error,stdout,stderr){
    if(stdout.length >1){

```python
import socket
import grove_temperature_sensor

if __name__ == '__main__':
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('127.0.0.1', 7000))

    while True:
        temperature = grove_temperature_sensor.read_temperature('v1.2')
        client.sendall(str(temperature))

        data = client.recv(1024)
        print data
        client.close()

        print 'Received', repr(data)
```
Modify the `time_sub.js` file.

```javascript
var aws_iot = require("./src/aws_iot.js");

var client_params = {
    host: 'mqtts://g.us-east-1.pb.iot.amazonaws.com',
    clientId: 'sdk_sub2'
};

var iot_client = new aws_iot(client_params);

//iot_client.subscribe(['topic/a', 'topic/b']);

iot_client.subscribe('topic/test');

iot_client.onMessage(function(topic, message){
    console.log(topic.toString() + ' ' + message.toString());
});
```

First, run the `time_pub.js` to publish temperature data to AWS.

Second, run the `time_sub.js` to subscribe data from the AWS.
debugger listening on port 35434
Connected to mqtt://us-east-1.pb.iot.amazonaws.com
Successfully sending subscribe
topic/test 25.69
topic/test 25.69
topic/test 25.69
topic/test 25.69