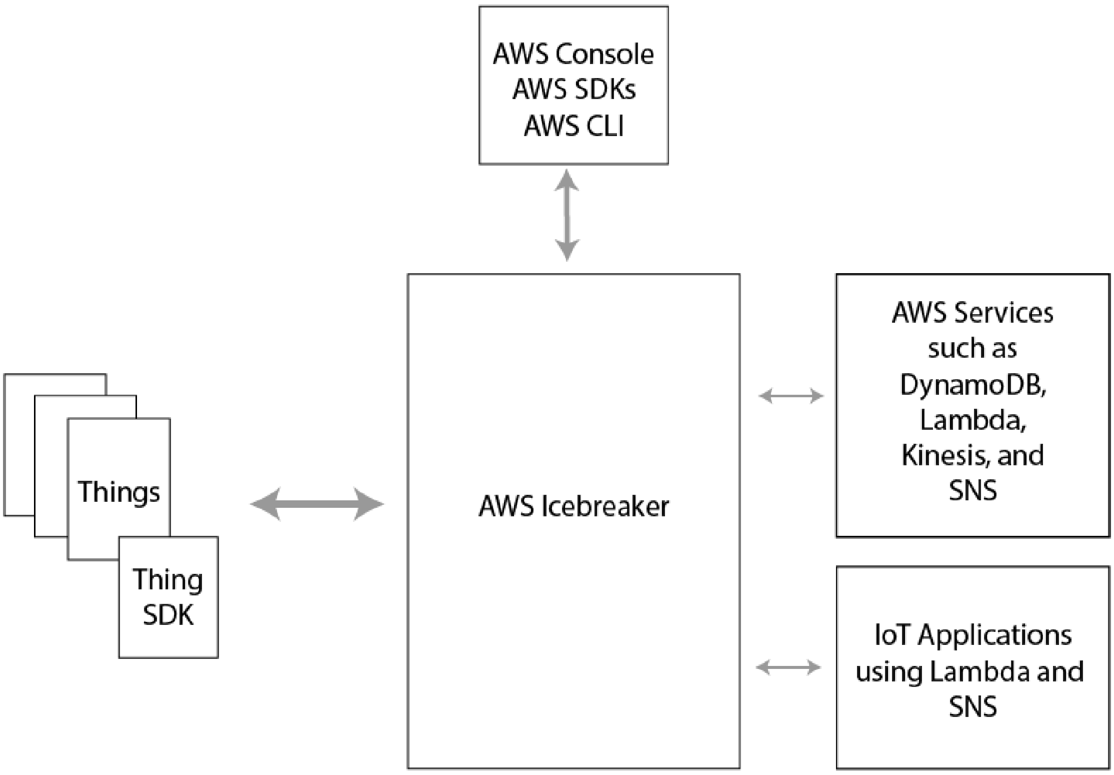
What Is AWS Icebreaker?

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:


AWS Console
AWS SDKs
AWS CLI
Things
Thing SDK
AWS Icebreaker
AWS Services such as DynamoDB, Lambda, Kinesis, and SNS
IoT Applications using Lambda and SNS

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 recieve MQTT messages from Icebreaker. They define what information your devices send to Icebreaker and how they respond to messages recieved 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 and existing certificate.

Rules and Integrations

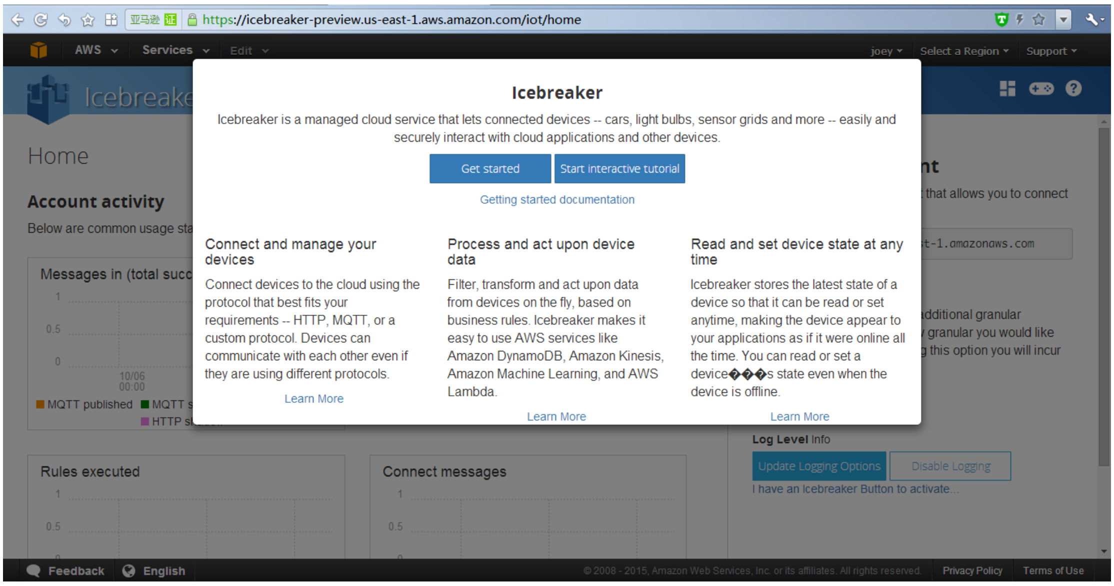
The rules and integrations section allows you to add a new rule and view your existing rules.

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



Icebreaker

Icebreaker is a managed cloud service that lets connected devices -- cars, light bulbs, sensor grids and more -- easily and securely interact with cloud applications and other devices.

Get started | Start interactive tutorial

Getting started documentation

Connect and manage your devices

Connect devices to the cloud using the protocol that best fits your requirements -- HTTP, MQTT, or a custom protocol. Devices can communicate with each other even if they are using different protocols.

Learn More

Process and act upon device data

Filter, transform and act upon data from devices on the fly, based on business rules. Icebreaker makes it easy to use AWS services like Amazon DynamoDB, Amazon Kinesis, Amazon Machine Learning, and AWS Lambda.

Learn More

Read and set device state at any time

Icebreaker stores the latest state of a device so that it can be read or set anytime, making the device appear to your applications as if it were online all the time. You can read or set a device���s state even when the device is offline.

Learn More

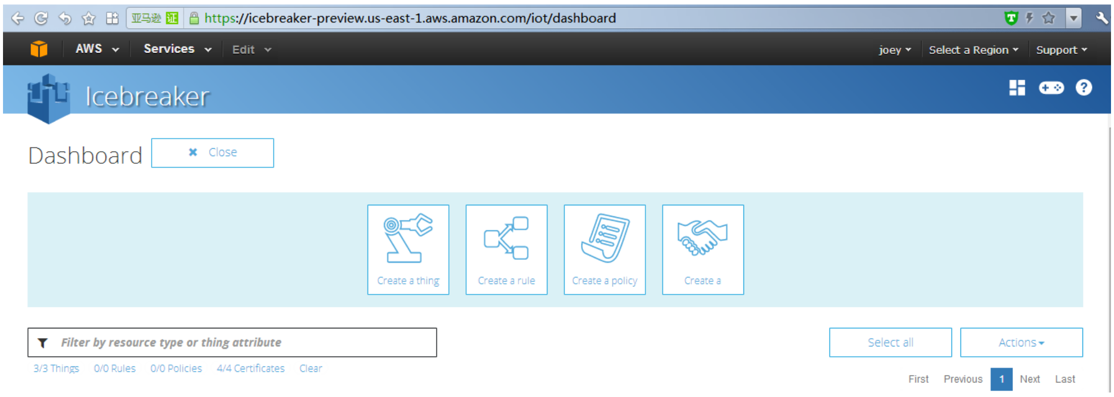
Below are common usage sta

I have an Icebreaker Button to activate...

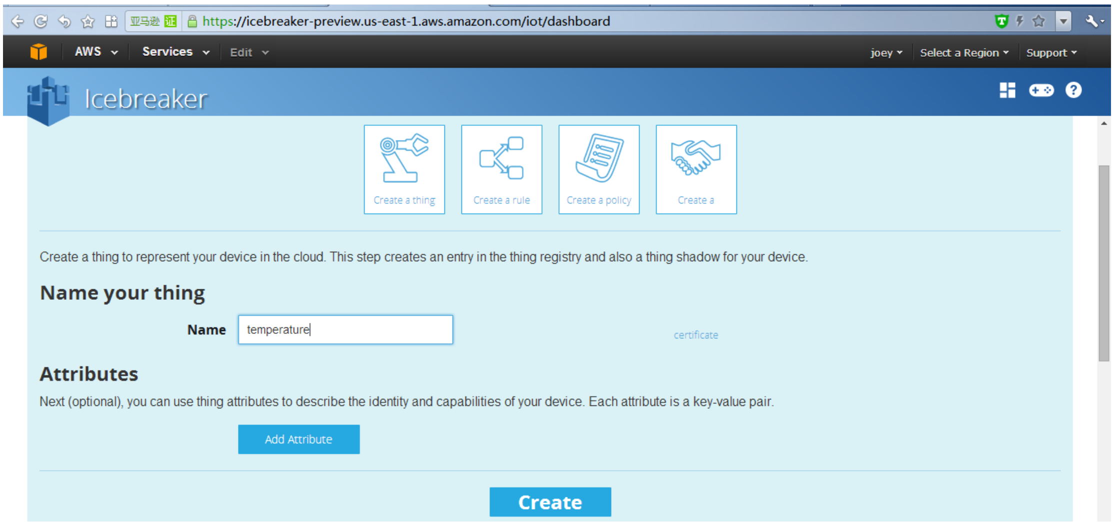
© 2008 - 2015, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Create a Thing in the Thing Registry

Go the Icebreaker console ,click the Dashboard.



Create a thing, e,g: temperature.



Create a thing to represent your device in the cloud. This step creates an entry in the thing registry and also a thing shadow for your device.

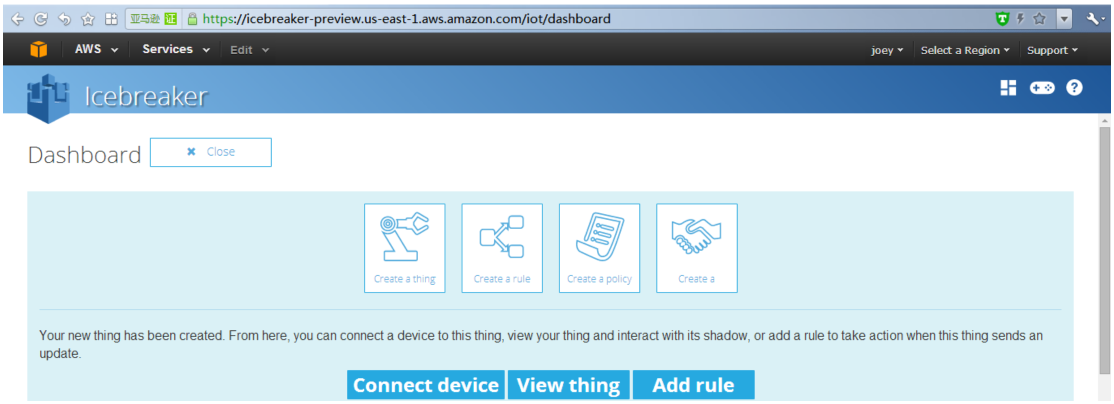
Name your thing

Name: temperature

Attributes

Next (optional), you can use thing attributes to describe the identity and capabilities of your device. Each attribute is a key-value pair.

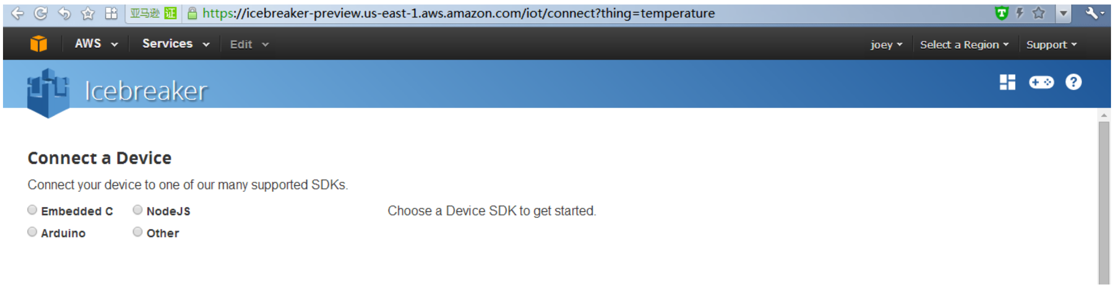
The web page will appear “Connect device” button, Click it.



Your new thing has been created. From here, you can connect a device to this thing, view your thing and interact with its shadow, or add a rule to take action when this thing sends an update.

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.


https://icebreaker-preview.us-east-1.aws.amazon.com/iot/connect?thing=temperature
AWS
Services
Edit
joey
Select a Region
Support
Icebreaker

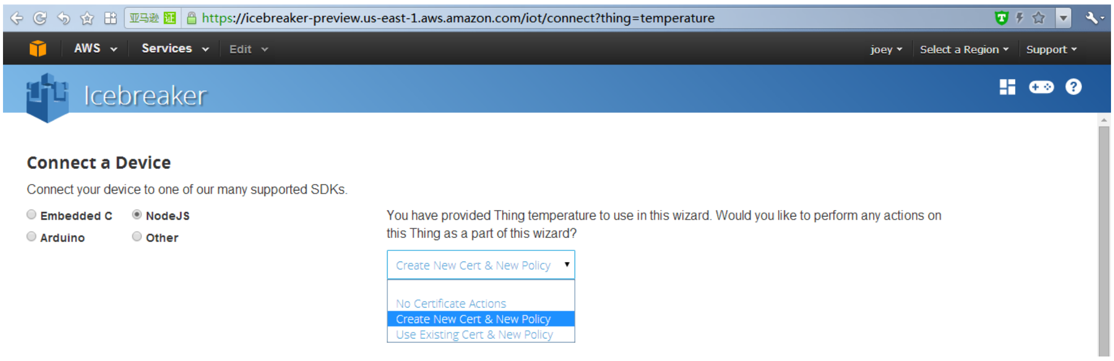
Connect a Device

Connect your device to one of our many supported SDKs.

○ Embedded C ○ NodeJS
○ Arduino ○ Other

Choose a Device SDK to get started.

Create new cert and new policy.


https://icebreaker-preview.us-east-1.aws.amazon.com/iot/connect?thing=temperature
AWS
Services
Edit
joey
Select a Region
Support
Icebreaker
No Certificate Actions
Create New Cert & New Policy
Use Existing Cert & New Policy

Connect a Device

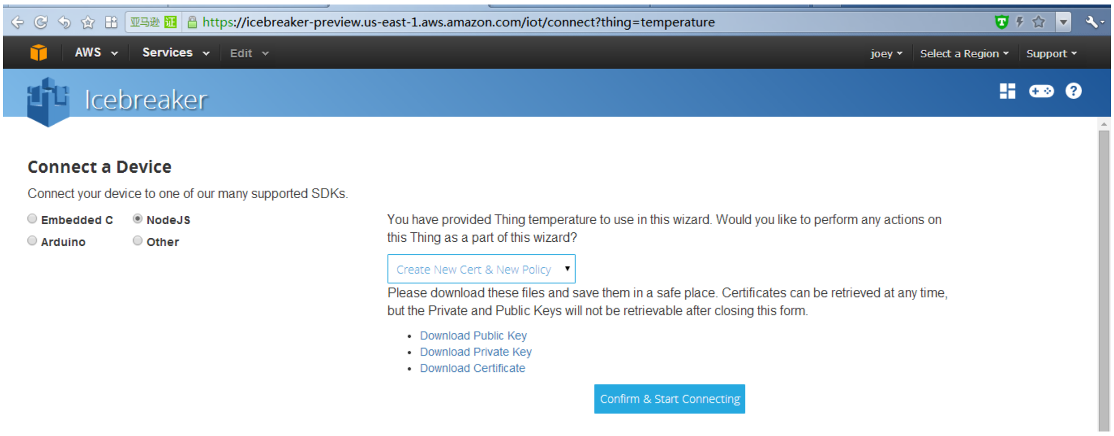
Connect your device to one of our many supported SDKs.

○ Embedded C ● NodeJS
○ Arduino ○ Other

You have provided Thing temperature to use in this wizard. Would you like to perform any actions on this Thing as a part of this wizard?

Create New Cert & New Policy

Download the three files. Then power on your board.


https://icebreaker-preview.us-east-1.aws.amazon.com/iot/connect?thing=temperature
AWS
Services
Edit
joey
Select a Region
Support
Icebreaker
Confirm & Start Connecting

Connect a Device

Connect your device to one of our many supported SDKs.

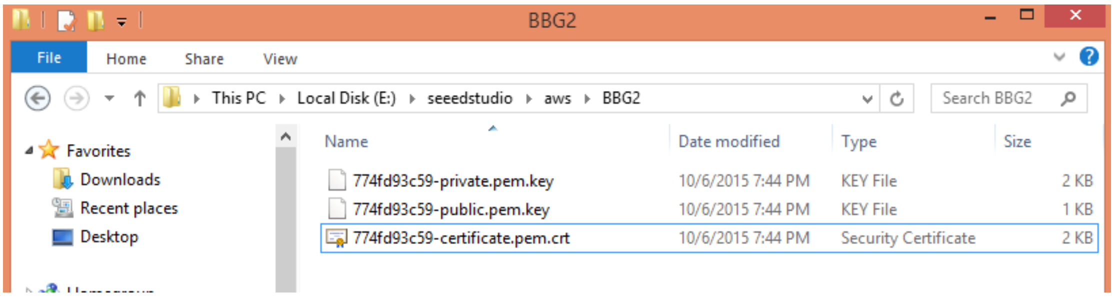
○ Embedded C ● NodeJS
○ Arduino ○ Other

You have provided Thing temperature to use in this wizard. Would you like to perform any actions on this Thing as a part of this wizard?

Create New Cert & New Policy

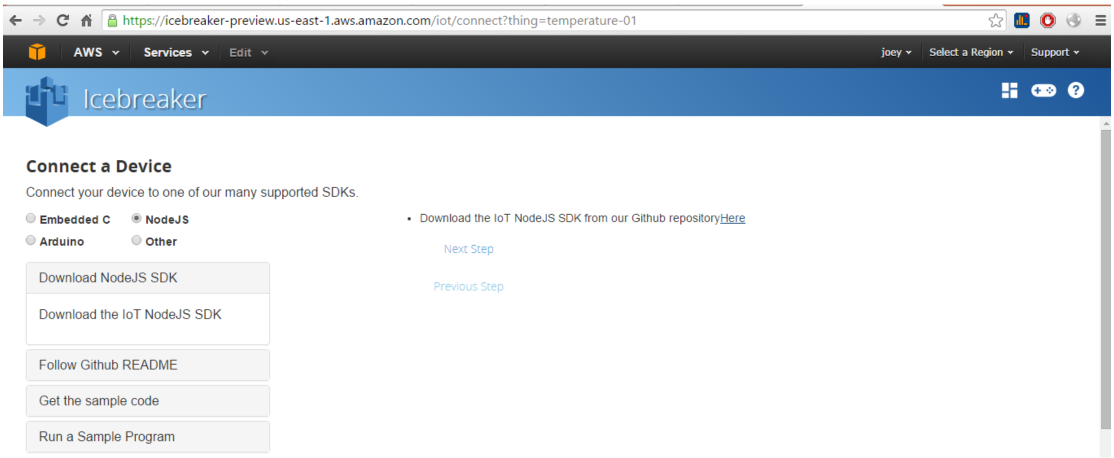
Please download these files and save them in a safe place. Certificates can be retrieved at any time, but the Private and Public Keys will not be retrievable after closing this form.

- Download Public Key
- Download Private Key
- Download Certificate



Name	Date modified	Type	Size
774fd93c59-private.pem.key	10/6/2015 7:44 PM	KEY File	2 KB
774fd93c59-public.pem.key	10/6/2015 7:44 PM	KEY File	1 KB
774fd93c59-certificate.pem.crt	10/6/2015 7:44 PM	Security Certificate	2 KB

Amazon provides the github maintain the code. Next page is the latest code introduction. We also provide on-board code to test the icebreaker.


https://icebreaker-preview.us-east-1.aws.amazon.com/iot/connect?thing=temperature-01
AWS
Services
Edit
joey
Select a Region
Support
Icebreaker

Connect a Device

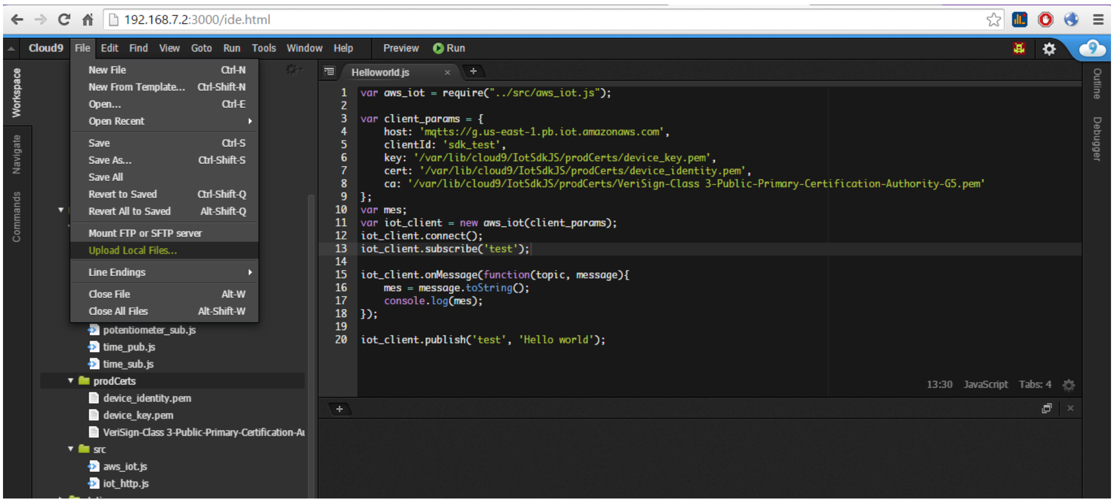
Connect your device to one of our many supported SDKs.

○ Embedded C ● NodeJS
○ Arduino ○ Other

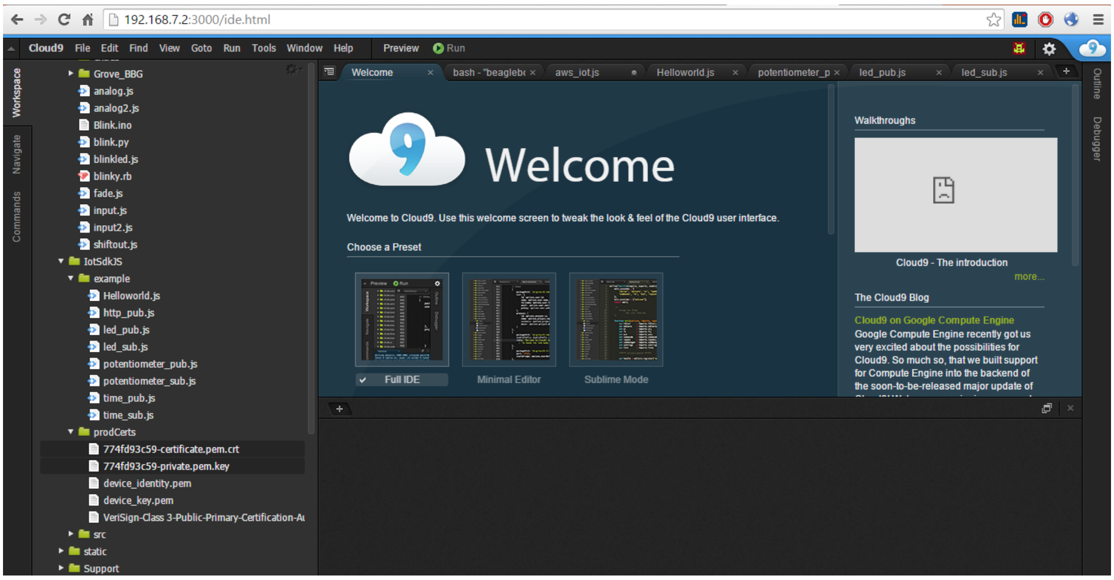
- Download the IoT NodeJS SDK from our Github repositoryHere

AWS MQTT publish and subscribe

Upload xxxxxx-private.pem.key and xxxxxxx-certificate.pem.crt to prodCerts folder.



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

var client_params = {
    host: 'mqtts://g.us-east-1.pb.iot.amazonaws.com',
    clientId: 'sdk_test',
    key: '/var/lib/cloud9/IotSdkJS/prodCerts/device_key.pem',
    cert: '/var/lib/cloud9/IotSdkJS/prodCerts/device_identity.pem',
    ca: '/var/lib/cloud9/IotSdkJS/prodCerts/VeriSign-Class 3-Public-Primary-Certification-Authority-G5.pem'
};
var mes;
var iot_client = new aws_iot(client_params);
iot_client.connect();
iot_client.subscribe('test');

iot_client.onMessage(function(topic, message){
    mes = message.toString();
    console.log(mes);
});

iot_client.publish('test', 'Hello world');
```



Welcome to Cloud9. Use this welcome screen to tweak the look & feel of the Cloud9 user interface.

Google Compute Engine recently got us very excited about the possibilities for Cloud9. So much so, that we built support for Compute Engine into the backend of the soon-to-be-released major update of

Rename 774fd93c59-private.pem.key as device_identity.pem.

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

```
root@beaglebone:/var/lib/cloud9/IotSdkJS/prodCerts# mv 774fd93c59-private.pem.key device_key.pem
root@beaglebone:/var/lib/cloud9/IotSdkJS/prodCerts# mv 774fd93c59-certificate.pem.crt device_identity.pem
root@beaglebone:/var/lib/cloud9/IotSdkJS/prodCerts# ls
VeriSign-Class 3-Public-Primary-Certification-Authority-G5.pem  device_identity.pem  device_key.pem
```

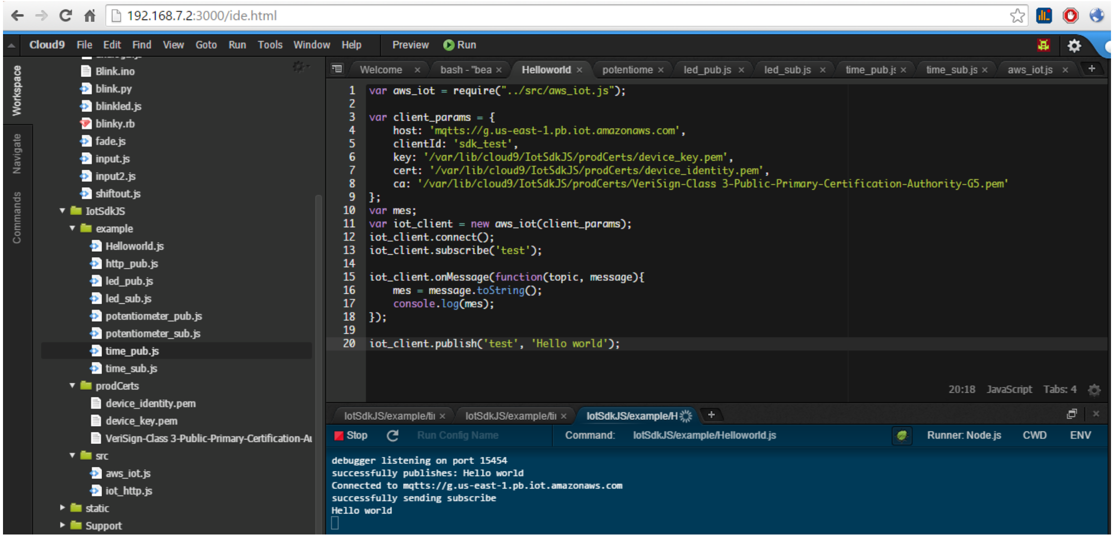
If you use mqtt protocol, you need modify the port to 8883.

/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.



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

var client_params = {
    host: 'mqtts://g.us-east-1.pb.iot.amazonaws.com',
    clientId: 'sdk_test',
    key: '/var/lib/cloud9/IotSdkJS/prodCerts/device_key.pem',
    cert: '/var/lib/cloud9/IotSdkJS/prodCerts/device_identity.pem',
    ca: '/var/lib/cloud9/IotSdkJS/prodCerts/VeriSign-Class 3-Public-Primary-Certification-Authority-G5.pem'
};
var mes;
var iot_client = new aws_iot(client_params);
iot_client.connect();
iot_client.subscribe('test');

iot_client.onMessage(function(topic, message){
    mes = message.toString();
    console.log(mes);
});

iot_client.publish('test', 'Hello world');
```

```
debugger listening on port 15454
successfully publishes: Hello world
Connected to mqtts://g.us-east-1.pb.iot.amazonaws.com
successfully sending subscribe
Hello world
```

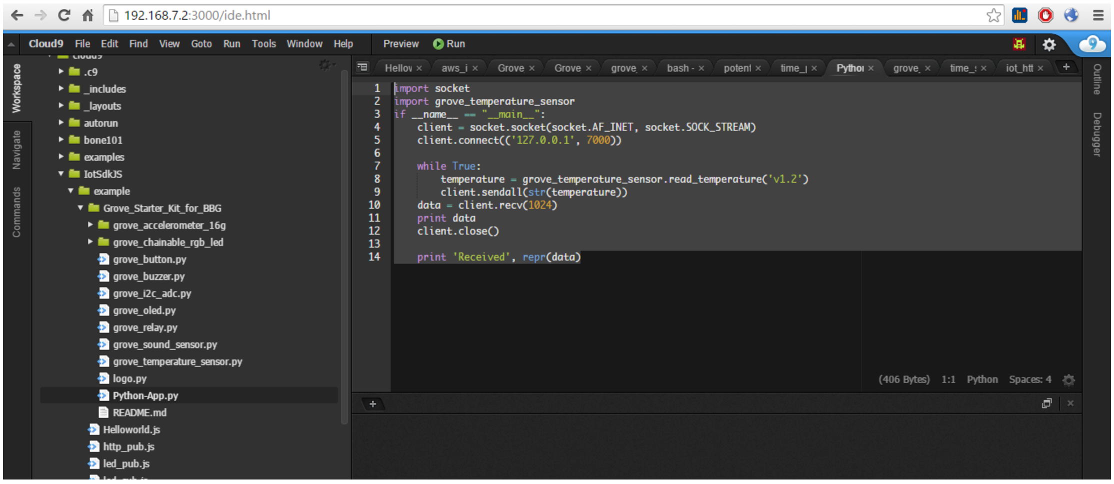
e.g Grove temperature sensor.

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){
        console.log('you offer args:',stdout);
    } else {
        console.log('you don\'t offer args');
    }
    if(error) {
        console.info('stderr : '+stderr);
    }
});


setInterval(function(){
    iot_client.publish('topic/test',temperature);
}, 2000);
```

Download https://github.com/Seeed-Studio/Grove_Starter_Kit_for_BBG to IotSdkJS fold. Create a python file named Python-App.py.

```
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)
```



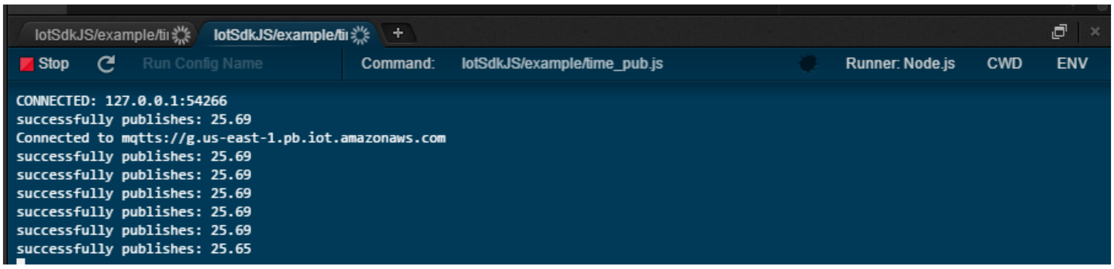
```
import socket
import grove_temperature_sensor
if __name__ == "__main__":
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('127.0.0.1', 7000))

    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.

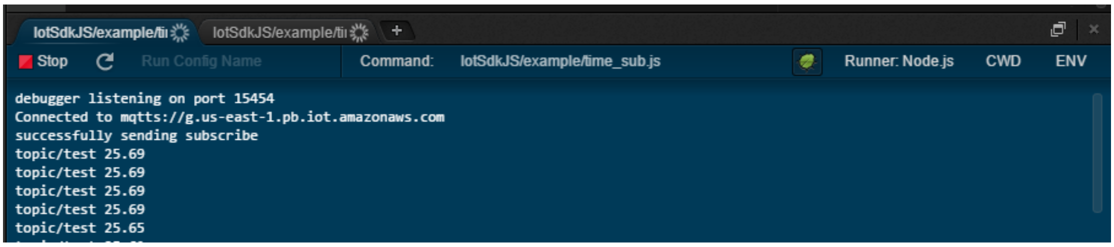
```
var aws_iot = require("../src/aws_iot.js");
var client_params = {
        host: 'mqtts://g.us-east-1.pb.iot.amazonaws.com',
        clientId: 'sdk_sub2'
};

var iot_client = new aws_iot(client_params);
iot_client.connect();
//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.



```
CONNECTED: 127.0.0.1:54266
successfully publishes: 25.69
Connected to mqtts://g.us-east-1.pb.iot.amazonaws.com
successfully publishes: 25.69
successfully publishes: 25.69
successfully publishes: 25.69
successfully publishes: 25.69
successfully publishes: 25.69
successfully publishes: 25.65
```



```
debugger listening on port 15454
Connected to mqtts://g.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
topic/test 25.65
```