CHIP initial setup and samples

Connect CHIP to the wireless network

Now that you have console access to your CHIP via the serial connection, you will want to connect your CHIP to the Wifi network. You can do this using the nmcli tool.

The format of the command is:

```
nmcli device wifi connect '(your wifi network name/SSID)' password '(your wifi password)' ifname wlan0
```

Run this as root, or use sudo

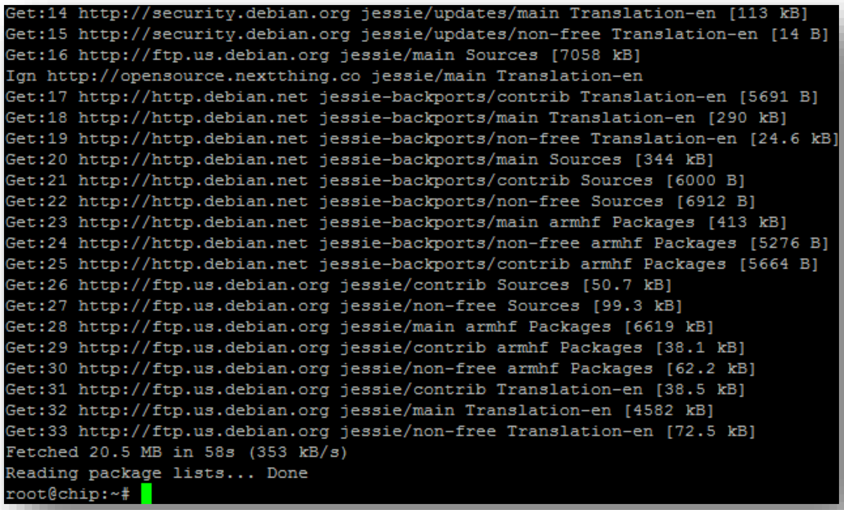
Confirm you are on the network – try a ping to 8.8.8.8

Update your package list

The world has moved on since your CHIP was imaged, so before we go ahead, we need to update the package list on the CHIP. Issuing the apt-get update command downloads the package lists from the repositories and "updates" them to get information on the newest versions of packages and their dependencies. Enter the following:

```
apt-get update
```

If all is well, you will see lots of activity as the package list is updated across the network:



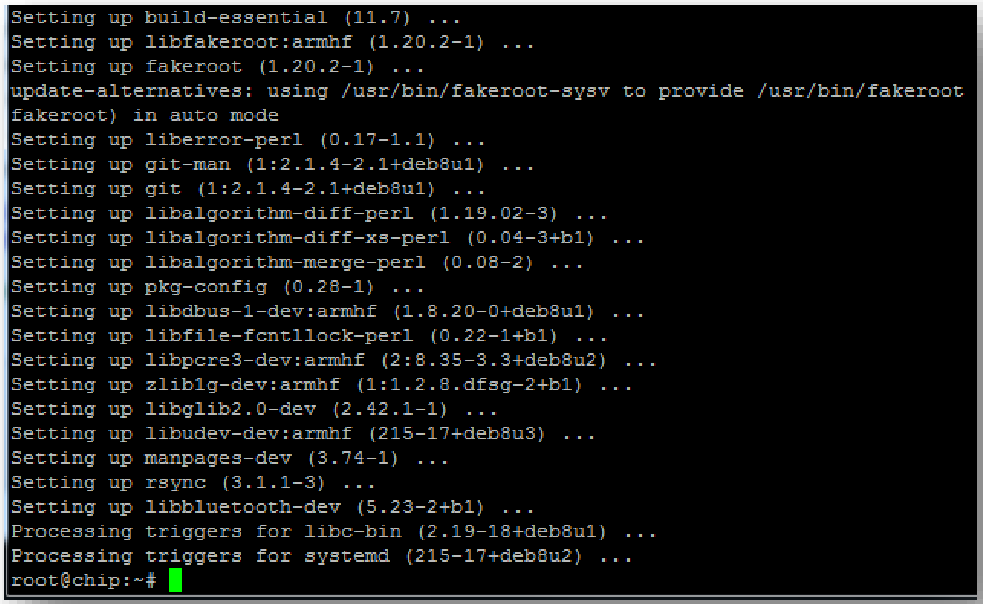
```
Get:14 http://security.debian.org jessie/updates/main Translation-en [113 kB]
Get:15 http://security.debian.org jessie/updates/non-free Translation-en [14 B]
Get:16 http://ftp.us.debian.org jessie/main Sources [7058 kB]
Ign http://opensource.nextthing.co jessie/main Translation-en
Get:17 http://http.debian.net jessie-backports/contrib Translation-en [5691 B]
Get:18 http://http.debian.net jessie-backports/main Translation-en [290 kB]
Get:19 http://http.debian.net jessie-backports/non-free Translation-en [24.6 kB]
Get:20 http://http.debian.net jessie-backports/main Sources [344 kB]
Get:21 http://http.debian.net jessie-backports/contrib Sources [6000 B]
Get:22 http://http.debian.net jessie-backports/non-free Sources [6912 B]
Get:23 http://http.debian.net jessie-backports/main armhf Packages [413 kB]
Get:24 http://http.debian.net jessie-backports/non-free armhf Packages [5276 B]
Get:25 http://http.debian.net jessie-backports/contrib armhf Packages [5664 B]
Get:26 http://ftp.us.debian.org jessie/contrib Sources [50.7 kB]
Get:27 http://ftp.us.debian.org jessie/non-free Sources [99.3 kB]
Get:28 http://ftp.us.debian.org jessie/main armhf Packages [6619 kB]
Get:29 http://ftp.us.debian.org jessie/contrib armhf Packages [38.1 kB]
Get:30 http://ftp.us.debian.org jessie/non-free armhf Packages [62.2 kB]
Get:31 http://ftp.us.debian.org jessie/contrib Translation-en [38.5 kB]
Get:32 http://ftp.us.debian.org jessie/main Translation-en [4582 kB]
Get:33 http://ftp.us.debian.org jessie/non-free Translation-en [72.5 kB]
Fetched 20.5 MB in 58s (353 kB/s)
Reading package lists... Done
root@chip:~#
```

Install dev/build dependencies

We will be developing some code on the CHIP, so we need some definitions, headers, build tools, etc, installed on the device. Issue:

```
      apt-get install -y libbluetooth-dev libudev-dev libdbus-1-dev
libglib2.0-dev build-essential git
```

This will take a few minutes for the relevant packages to be downloaded and built, then installed.



```
Setting up build-essential (11.7) ...
Setting up libfakeroot:armhf (1.20.2-1) ...
Setting up fakeroot (1.20.2-1) ...
update-alternatives: using /usr/bin/fakeroot-sysv to provide /usr/bin/fakeroot
fakeroot) in auto mode
Setting up liberror-perl (0.17-1.1) ...
Setting up git-man (1:2.1.4-2.1+deb8u1) ...
Setting up git (1:2.1.4-2.1+deb8u1) ...
Setting up libalgorithm-diff-perl (1.19.02-3) ...
Setting up libalgorithm-diff-xs-perl (0.04-3+b1) ...
Setting up libalgorithm-merge-perl (0.08-2) ...
Setting up pkg-config (0.28-1) ...
Setting up libdbus-1-dev:armhf (1.8.20-0+deb8u1) ...
Setting up libfile-fcntllock-perl (0.22-1+b1) ...
Setting up libpcre3-dev:armhf (2:8.35-3.3+deb8u2) ...
Setting up zlib1g-dev:armhf (1:1.2.8.dfsg-2+b1) ...
Setting up libglib2.0-dev (2.42.1-1) ...
Setting up libudev-dev:armhf (215-17+deb8u3) ...
Setting up manpages-dev (3.74-1) ...
Setting up rsync (3.1.1-3) ...
Setting up libbluetooth-dev (5.23-2+b1) ...
Processing triggers for libc-bin (2.19-18+deb8u1) ...
Processing triggers for systemd (215-17+deb8u2) ...
root@chip:~#
```

Install NodeJS

Some of the examples we will be providing as a starting point, are written in NodeJS. We need to install the NodeJS runtime on your CHIP to be able to run these examples. First, we need to install curl so we can pull down the NodeJS installer. Issue the following:

```
      apt-get install -y curl
```

This will pull down the curl package, and install it. It won't take too long.

Now we have curl, we can download NodeJS and dependencies. Issue the following:

```
      curl -sL https://deb.nodesource.com/setup_5.x | bash -
```

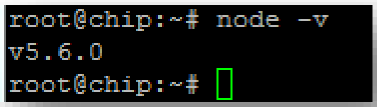
This step will take a few minutes. When it is done, we're ready to ask the package manager to install NodeJS. Issue the following:

```
      apt-get install -y nodejs
```

This won't take too long. Once it finishes, confirm the version of NodeJS you have installed, by issuing:

```
node -v
```

If all is well, you should see version v5.6.0 (or greater):



```
root@chip:~# node -v
v5.6.0
root@chip:~# 
```

Run sample code setup

We have provided some example code to allow you to play with the AWS IoT service, Bluetooth Low Energy and other features – just to get you started. Some of the samples assume some files are created that contain the Bluetooth MAC address, etc, so you need to run a quick setup process to get you started. The setup and samples are in the code repository:

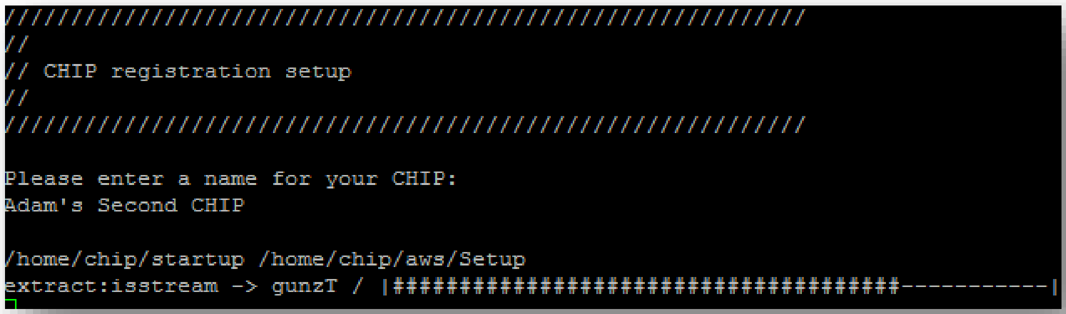
```
https://code.amazon.com/packages/Apac-chip-challenge/trees/mainline
```

Pull in the Setup folder into /home/chip on your CHIP, and mark CHIPSetup.sh as executable, and then run it. You will need to provide a name for your CHIP. This name is used in the Central Registry to identify your CHIP as yours:

```
cd /home/chip

sudo chmod 744 CHIPSetup.sh

sudo ./CHIPSetup.sh
```



```
/////////////////////////////////////////////////////////////////
//
// CHIP registration setup
//
/////////////////////////////////////////////////////////////////

Please enter a name for your CHIP:
Adam's Second CHIP

/home/chip/startup /home/chip/aws/Setup
extract:isstream -> gunzT / |#########################################-----------|
```

Setup takes a minute or two to pull in some NodeJS dependencies for the startup service. The script does the following:

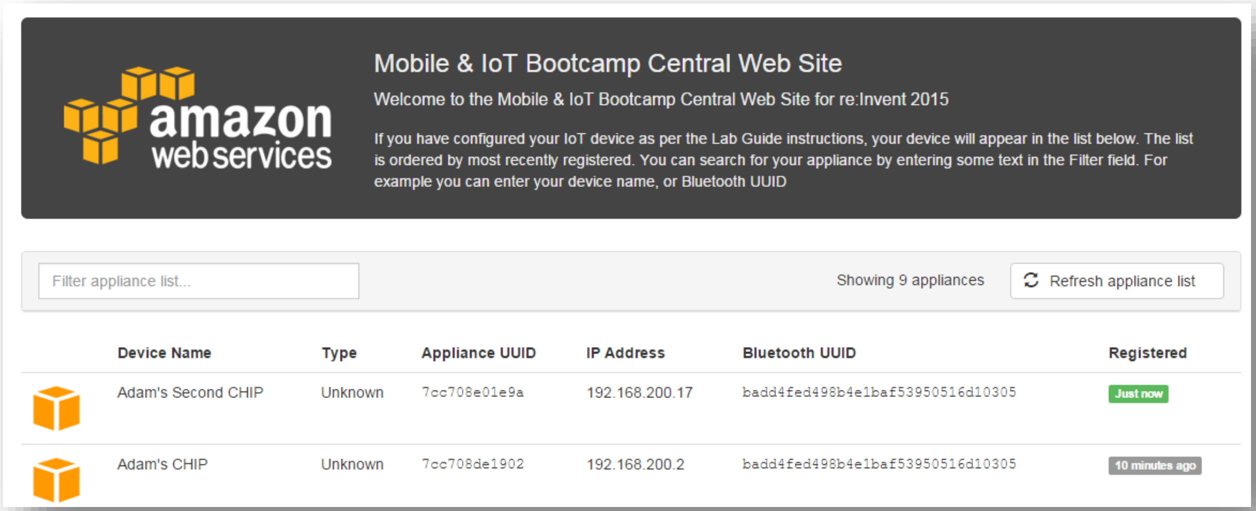
- Moves startup files to the /home/chip/startup directory, and installs the NodeJS dependencies
- Writes the Bluetooth MAC address of your CHIP to /etc/bluetooth_address. We do this so other sample scripts can determine the MAC address easily. If you are writing your own code, you can also access the MAC address of the CHIP from this location
- Writes the Bluetooth UUID (used for BLE samples) to /eetc/bluetooth_uuid. We do this so other sample scripts can access it.

- Asks for the name of your CHIP, and writes that to /etc/chip_name – again, so the name can be accessed by other scripts
- Finally, it installs a startup service that will register your CHIP with the central repository, using the MAC address, BLE UUID and your CHIP's name.

After the script runs to completion, browse to:

> http://bootcamp2015-mobile-iot.s3.amazonaws.com/bootcamp-central-web-site/index.html#

We have hijacked the central repository used for the Mobile & IoT bootcamp at re:Invent 2015 as our repository.



Mobile & IoT Bootcamp Central Web Site

Welcome to the Mobile & IoT Bootcamp Central Web Site for re:Invent 2015

If you have configured your IoT device as per the Lab Guide instructions, your device will appear in the list below. The list is ordered by most recently registered. You can search for your appliance by entering some text in the Filter field. For example you can enter your device name, or Bluetooth UUID

Device Name	Type	Appliance UUID	IP Address	Bluetooth UUID	Registered
Adam's Second CHIP	Unknown	7cc708e01e9a	192.168.200.17	badd4fed498b4e1baf53950516d10305	Just now
Adam's CHIP	Unknown	7cc708de1902	192.168.200.2	badd4fed498b4e1baf53950516d10305	10 minutes ago

Note that the device name is the name you entered. The IP address of the device is also listed, making it easier to connect to your CHIP when you boot it – DHCP won't always give you the same address. It also shows you how long ago the device was registered.

Sample code

In the code repository, we have a few projects to help you get started.

Please note that the software provided is *not* to be released publicly. Please contact the relevant author(s) for more information.

distance_graph

This sample requires 2 CHIPs, so either borrow one from a colleague, or work together.

The sample makes use of the Bluetooth Low Energy features of the CHIP, to perform proximity detection. To set up, cd into the distance_graph directory on both CHIPs, and run:

```
sudo npm install
```

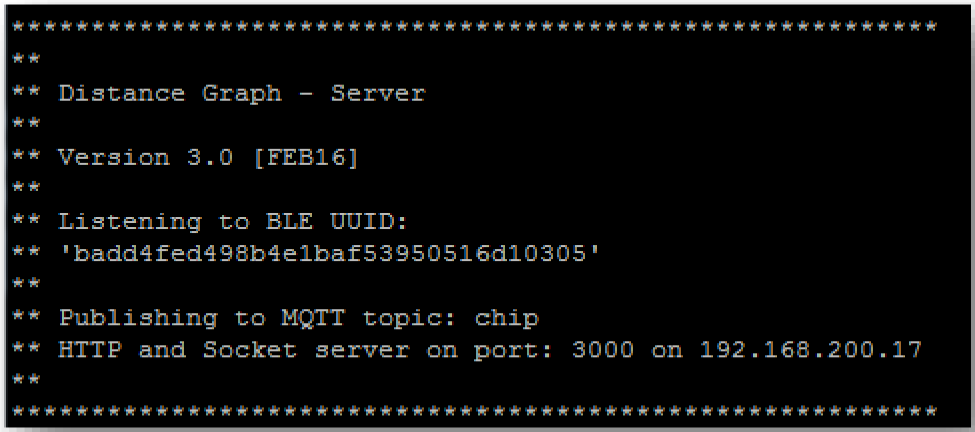
On one of the CHIPs, run

```
sudo node server
```

And on the other CHIP, run

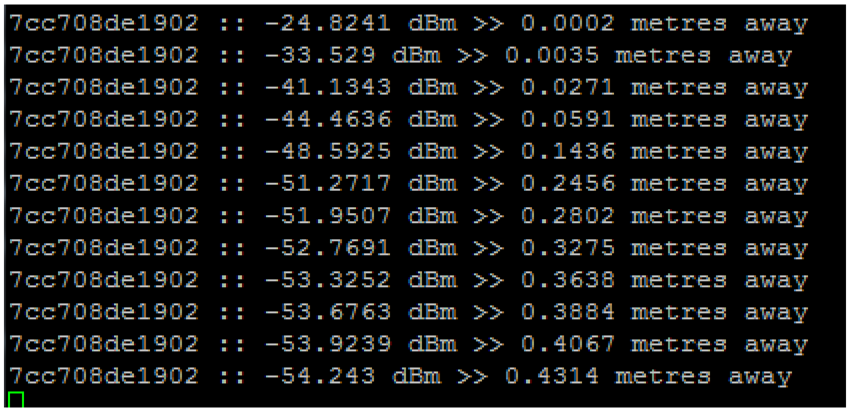
```
sudo node client
```

On the server CHIP, you will see this:



```
************************************************************
**
** Distance Graph - Server
**
** Version 3.0 [FEB16]
**
** Listening to BLE UUID:
** 'badd4fed498b4e1baf53950516d10305'
**
** Publishing to MQTT topic: chip
** HTTP and Socket server on port: 3000 on 192.168.200.17
**
************************************************************
```

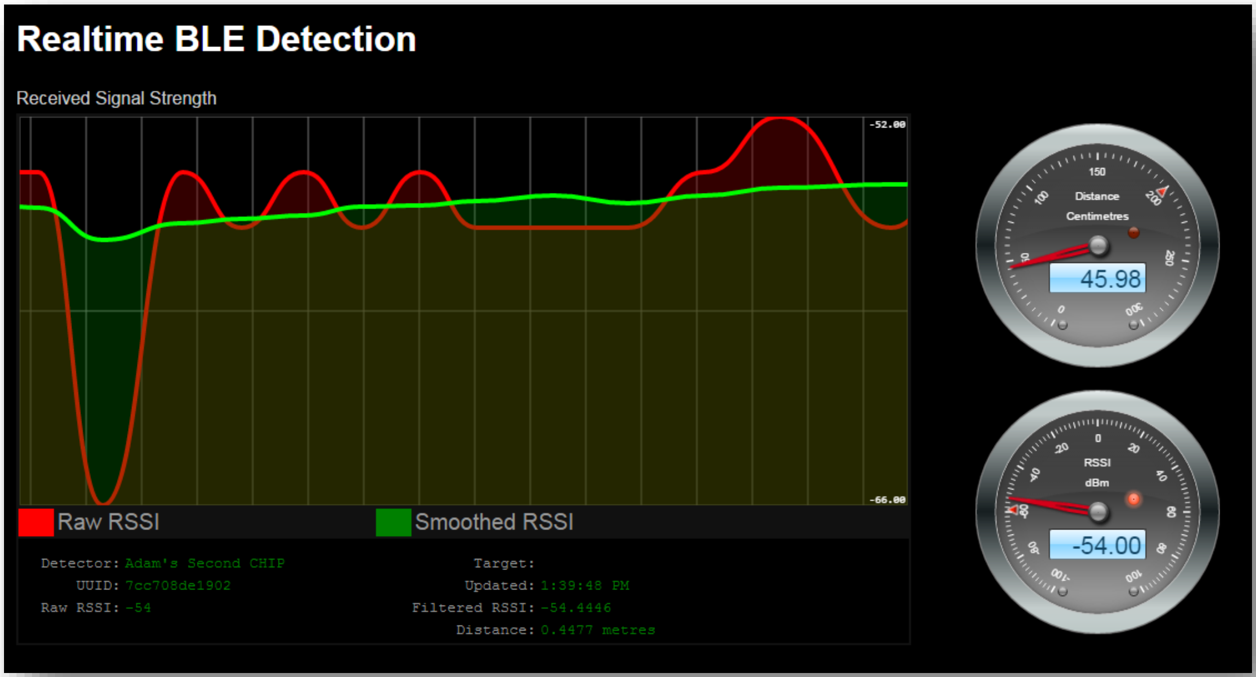
If you have the client running on the other CHIP, you will start to see advertisements received on the server:



```
7cc708de1902 :: -24.8241 dBm >> 0.0002 metres away
7cc708de1902 :: -33.529 dBm >> 0.0035 metres away
7cc708de1902 :: -41.1343 dBm >> 0.0271 metres away
7cc708de1902 :: -44.4636 dBm >> 0.0591 metres away
7cc708de1902 :: -48.5925 dBm >> 0.1436 metres away
7cc708de1902 :: -51.2717 dBm >> 0.2456 metres away
7cc708de1902 :: -51.9507 dBm >> 0.2802 metres away
7cc708de1902 :: -52.7691 dBm >> 0.3275 metres away
7cc708de1902 :: -53.3252 dBm >> 0.3638 metres away
7cc708de1902 :: -53.6763 dBm >> 0.3884 metres away
7cc708de1902 :: -53.9239 dBm >> 0.4067 metres away
7cc708de1902 :: -54.243 dBm >> 0.4314 metres away
```

The code is calculating the distance based on the signal strength, and applying a smoothing filter to iron out the interference (using a Kalman filter).

If you open a web browser, and connect to http://IP_ADDRESS:3000 you will see the graphical representation of the received signal strength between the client CHIP and the server CHIP:



```
Detector: Adam's Second CHIP                  Target:
    UUID: 7cc708de1902                      Updated: 1:39:48 PM
Raw RSSI: -54                          Filtered RSSI: -54.4446
                                            Distance: 0.4477 metres
```

The red line is the raw Received Signal Strength without filtering. The green lie is the smoothed result. The gauges show the distance and current signal strength.

The browser uses websockets to connect directly to the web server and socket server on the server CHIP. If you move the devices apart, you will see the distance gauge update.

You can use this sample as a basis for a proximity detection application.

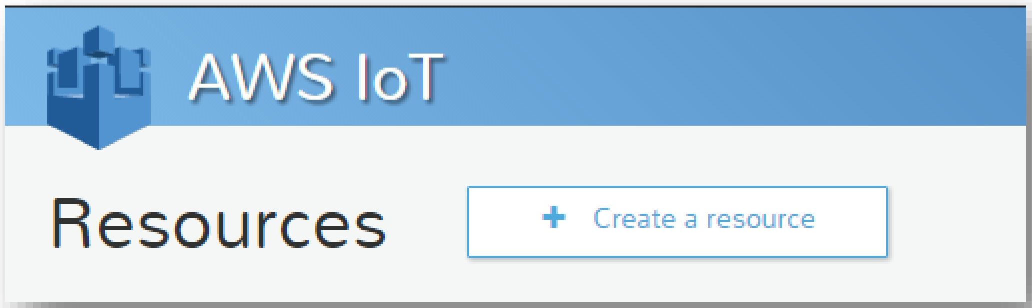
iot-mock-sensor

This sample shows how you can update a device shadow in AWS IoT. It simulates updates from a temperature sensor, using a Mock Implementation. When you have your own sensors, you can turn the mock off, and instead read the values from a real hardware sensor.

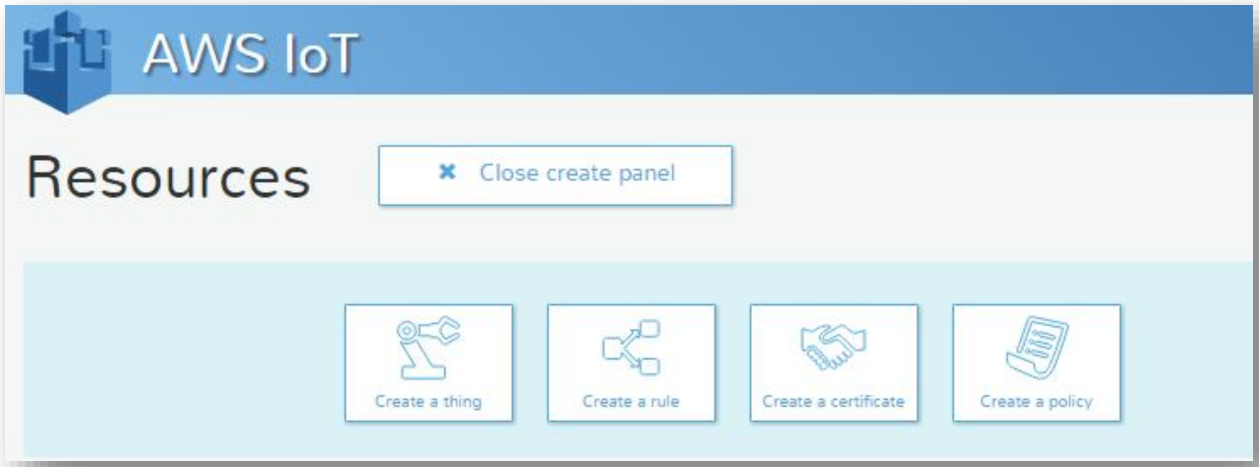
You need to set a few things up to use this demo. First, pull in the code and run:

```
sudo npm install
```

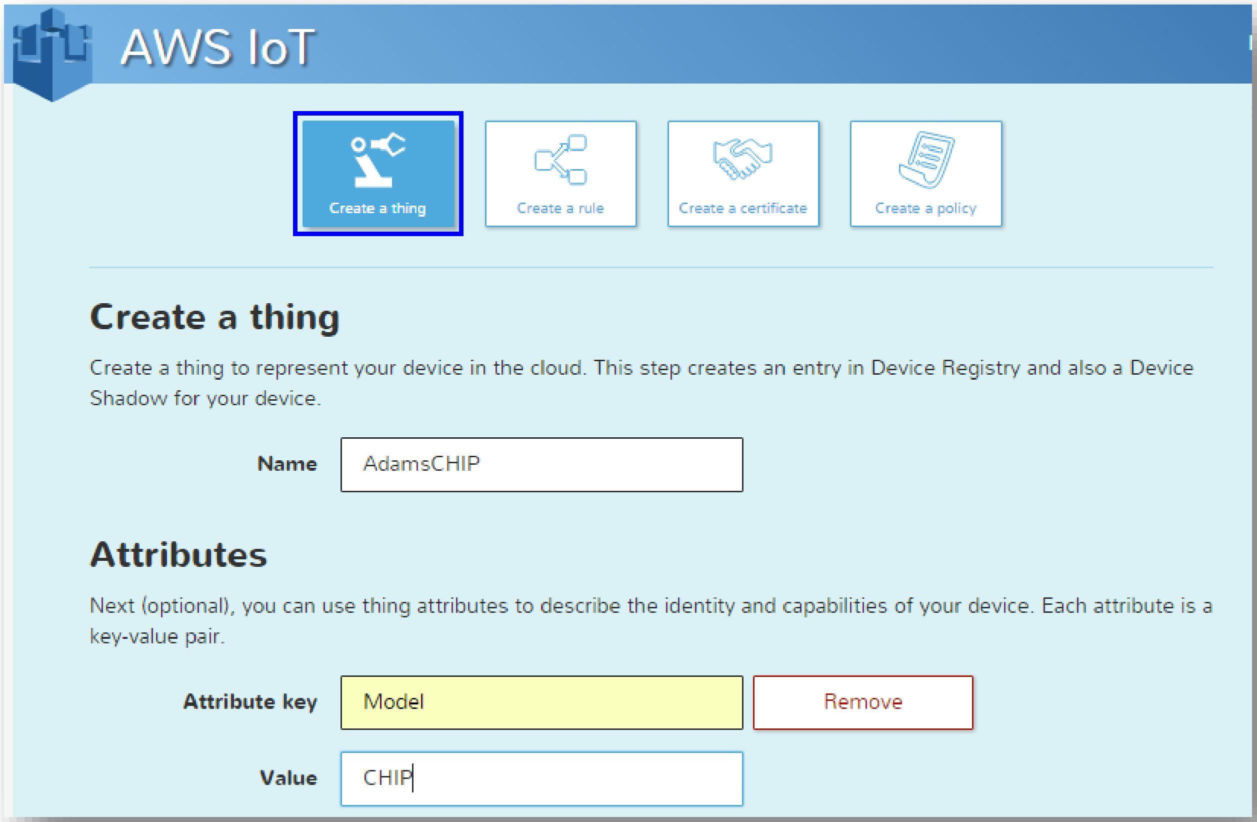
While that is happening, open your AWS console, and browse to AWS IoT. Click Create a Resource:



Then select Create a Thing



In the fields shown, give your new ‘thing’ a name (here, called AdamsCHIP, but you can use whatever name you like, as long as it matches the configuration we will do in the source code later)



Create a thing

Create a thing to represent your device in the cloud. This step creates an entry in Device Registry and also a Device Shadow for your device.

Name: AdamsCHIP

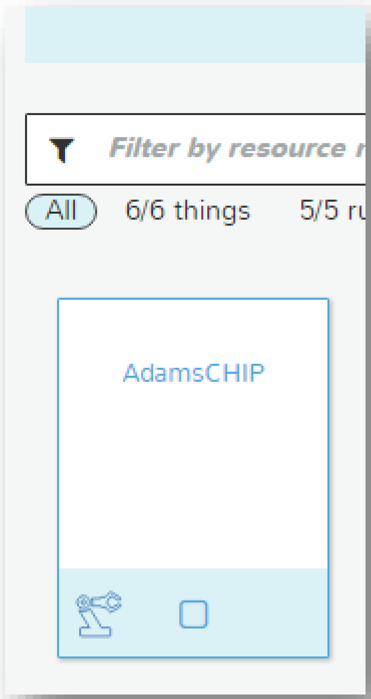
Attributes

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

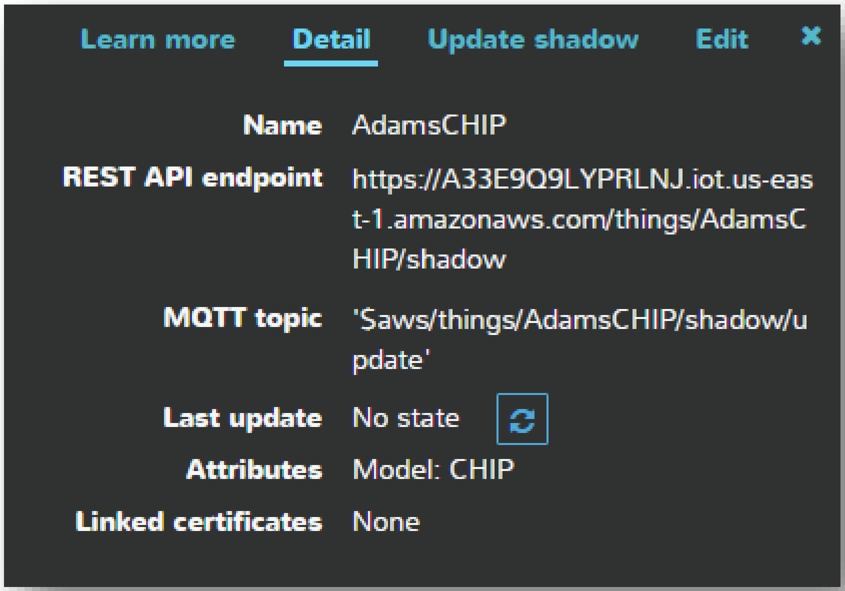
Attribute key: Model

Value: CHIP

You can also add an attribute just for demonstration purposes. Add the Model attribute, and give it the value CHIP. Scroll down and click Create. You will now see the new thing in the list:

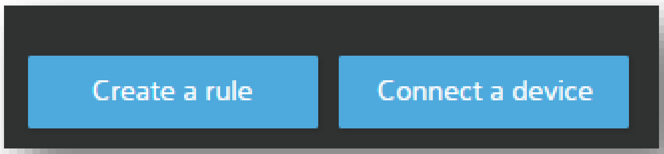


Click on the name of the thing, and view the information in the right-hand panel:

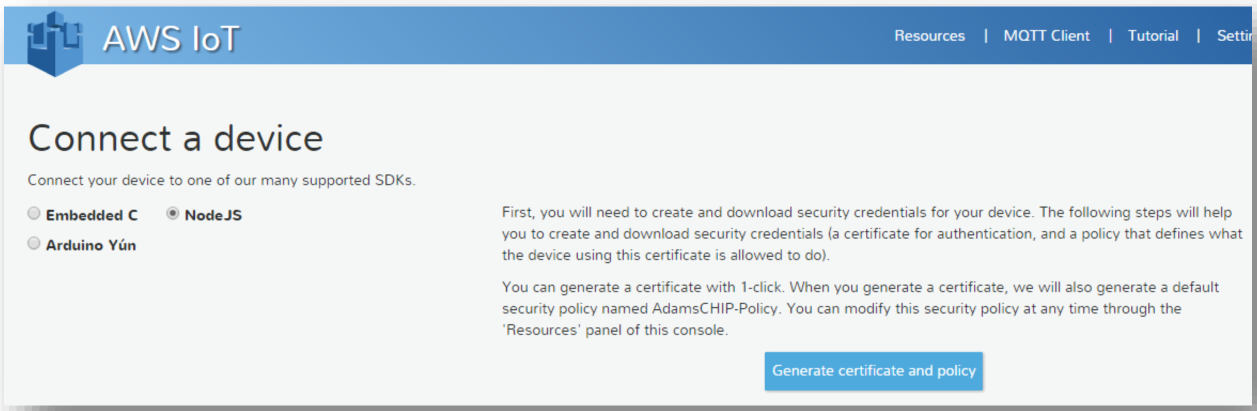


Name	AdamsCHIP
REST API endpoint	https://A33E9Q9LYPRLNJ.iot.us-east-1.amazonaws.com/things/AdamsCHIP/shadow
MQTT topic	'$aws/things/AdamsCHIP/shadow/update'
Last update	No state
Attributes	Model: CHIP
Linked certificates	None

Note that you can see the model attribute you entered, as well as the fact that there is no current state – we haven't connected a device and updated the shadow. To do that, we need to create a device and obtain the certificates to use on the CHIP. In the lower-right, click the Connect a Device option



Choose NodeJS:



Connect a device

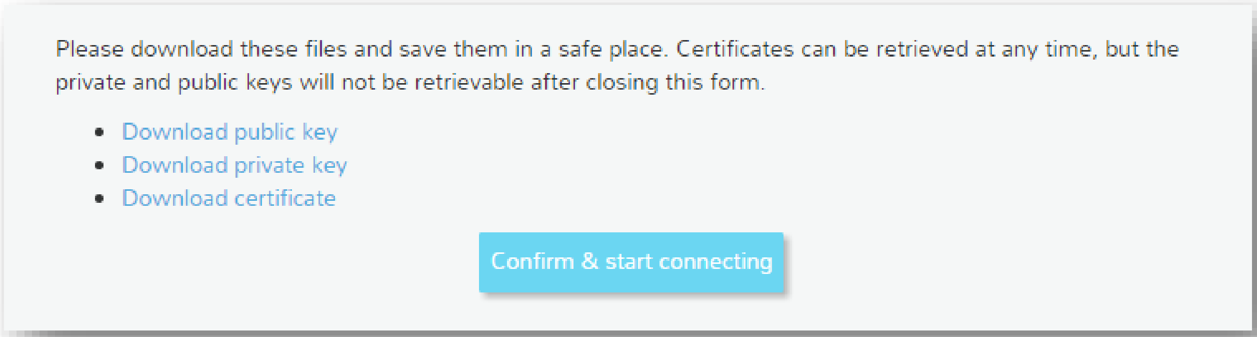
Connect your device to one of our many supported SDKs.

Embedded C [✓ NodeJS]
Arduino Yún

First, you will need to create and download security credentials for your device. The following steps will help you to create and download security credentials (a certificate for authentication, and a policy that defines what the device using this certificate is allowed to do).

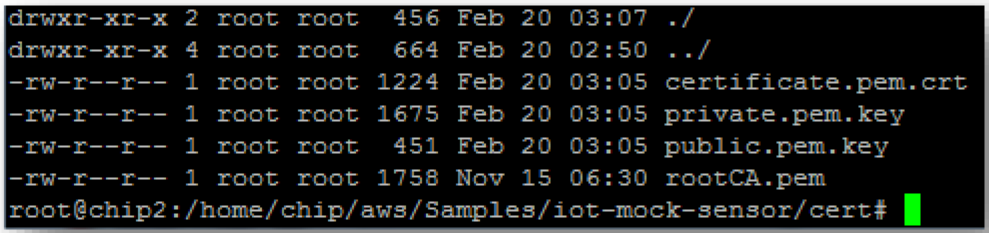
You can generate a certificate with 1-click. When you generate a certificate, we will also generate a default security policy named AdamsCHIP-Policy. You can modify this security policy at any time through the 'Resources' panel of this console.

Then click Generate certificate and policy. The public key, private key and certificate will be generated and made available to download. You need to download each of these and store them on your machine for safe-keeping. You also need to copy the private key and certificate to the cert/ directory on the CHIP.



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.

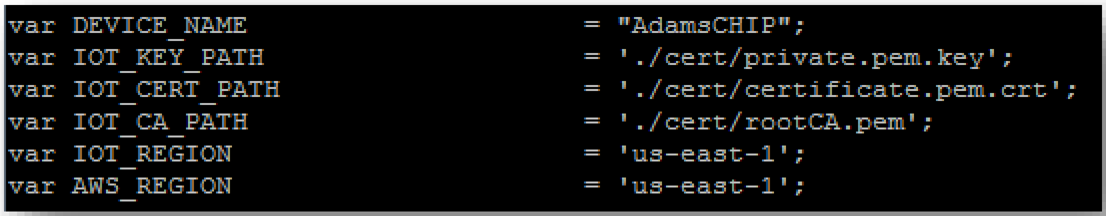
- Download public key
- Download private key
- Download certificate



```
drwxr-xr-x 2 root root  456 Feb 20 03:07 ./
drwxr-xr-x 4 root root  664 Feb 20 02:50 ../
-rw-r--r-- 1 root root 1224 Feb 20 03:05 certificate.pem.crt
-rw-r--r-- 1 root root 1675 Feb 20 03:05 private.pem.key
-rw-r--r-- 1 root root  451 Feb 20 03:05 public.pem.key
-rw-r--r-- 1 root root 1758 Nov 15 06:30 rootCA.pem
root@chip2:/home/chip/aws/Samples/iot-mock-sensor/cert#
```

The cert/ directory will already contain the rootCA.pem file – leave this file as it is.

On the CHIP, edit the config.js file



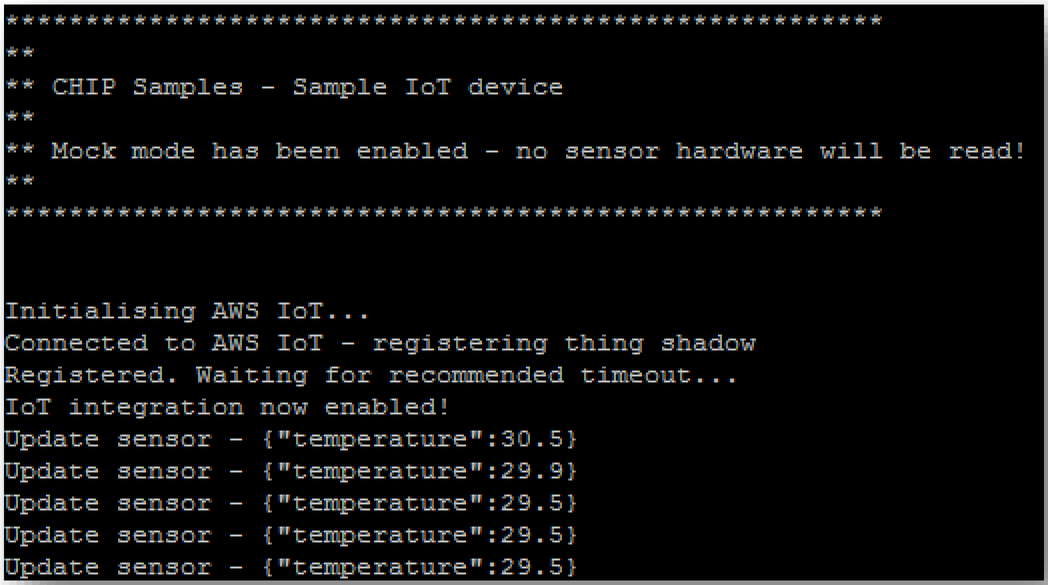
```
var DEVICE_NAME                         = "AdamsCHIP";
var IOT_KEY_PATH                        = './cert/private.pem.key';
var IOT_CERT_PATH                       = './cert/certificate.pem.crt';
var IOT_CA_PATH                         = './cert/rootCA.pem';
var IOT_REGION                          = 'us-east-1';
var AWS_REGION                          = 'us-east-1';
```

Ensure that the paths for the private key, certificate and root CA are correct, which they will be if you renamed the files you downloaded and removed the unique ID at the head of the filenames. If you prefer to keep the unique Id, update the config.js file accordingly.

Also note that the DEVICE_NAME constant needs to match the name of the thing you created in the previous step.

Now you're ready to test it out! Run the following module:

```
sudo node device
```

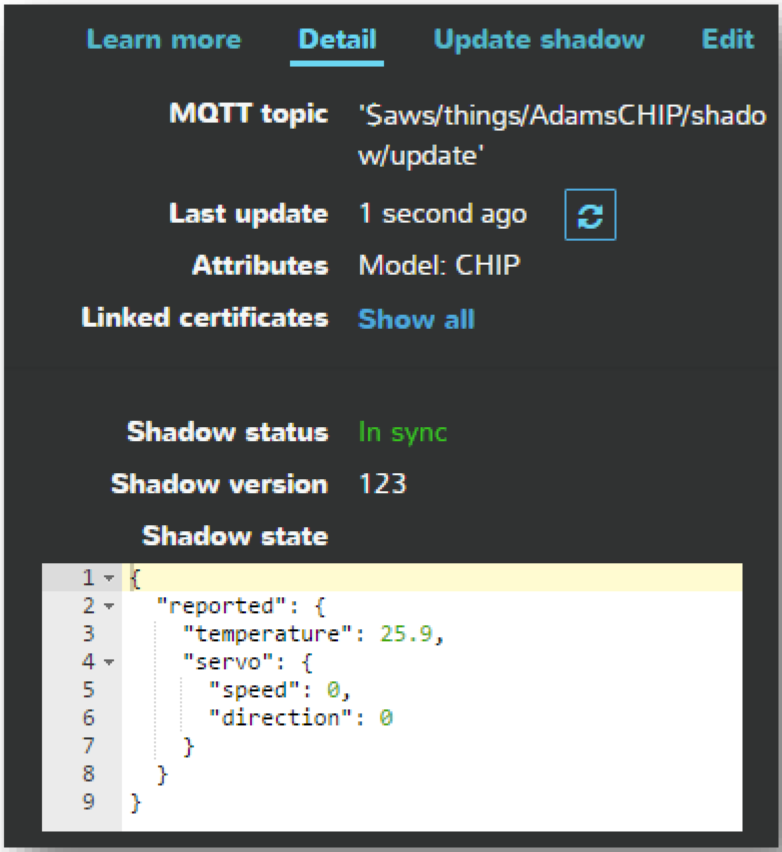


```
********************************************************
**
** CHIP Samples - Sample IoT device
**
** Mock mode has been enabled - no sensor hardware will be read!
**
********************************************************


Initialising AWS IoT...
Connected to AWS IoT - registering thing shadow
Registered. Waiting for recommended timeout...
IoT integration now enabled!
Update sensor - {"temperature":30.5}
Update sensor - {"temperature":29.9}
Update sensor - {"temperature":29.5}
Update sensor - {"temperature":29.5}
Update sensor - {"temperature":29.5}
```

The code will connect to the IoT service, and then start receiving pretend temperature updates from the mock implementation. Once per second, an update will be sent from the CHIP to the IoT service.

If you view the thing in the IoT console, you will notice that it now has state, which matches the latest update from the CHIP:



MQTT topic '$aws/things/AdamsCHIP/shadow/update'

Last update 1 second ago

Attributes Model: CHIP

Linked certificates Show all

Shadow status In sync

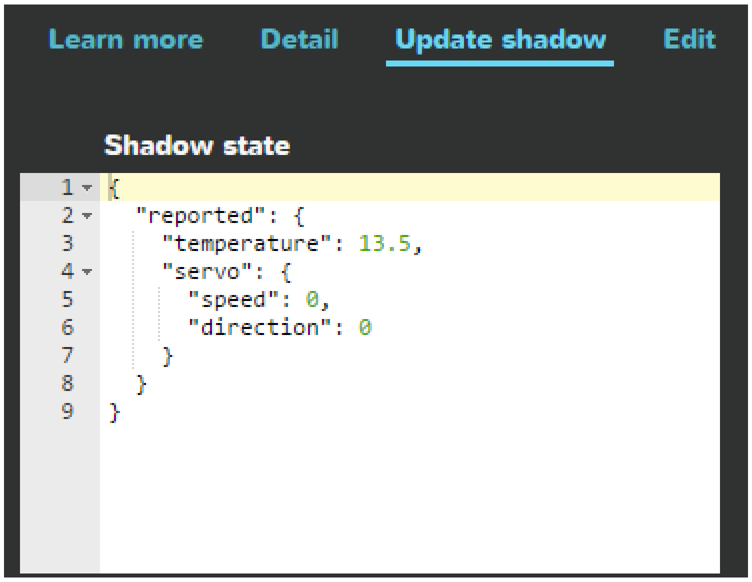
Shadow version 123

Shadow state

```
{
  "reported": {
    "temperature": 25.9,
    "servo": {
      "speed": 0,
      "direction": 0
    }
  }
}
```

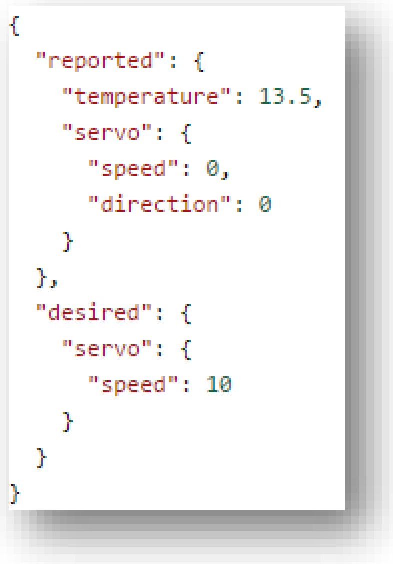
Also note that the reported state has a ‘servo’ section, as well as the temperature. The servo section is there to give an example of how we can command the CHIP to react to a change in metadata – in this case, we are pretending that there is a servo connected to the CHIP, and we can update the value of the servo’s speed and direction in the thing shadow, to cause the servo to move in the real world. Of course, we don’t have a servo, so we will just echo the intention on the CHIP’s console, and leave it as an exercise for the reader to implement this!

Click the Update Shadow link:



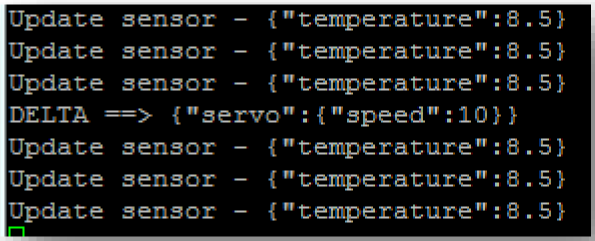
```
{
  "reported": {
    "temperature": 13.5,
    "servo": {
      "speed": 0,
      "direction": 0
    }
  }
}
```

Don’t alter the reported state. Instead, update the JSON to add a ‘desired’ state and set the servo’s speed to 10. It will look like this:



```
{
  "reported": {
    "temperature": 13.5,
    "servo": {
      "speed": 0,
      "direction": 0
    }
  },
  "desired": {
    "servo": {
      "speed": 10
    }
  }
}
```

Press the Update Shadow button, and a moment later, notice the CHIP responds:



```
Update sensor - {"temperature":8.5}
Update sensor - {"temperature":8.5}
Update sensor - {"temperature":8.5}
DELTA ==> {"servo":{"speed":10}}
Update sensor - {"temperature":8.5}
Update sensor - {"temperature":8.5}
Update sensor - {"temperature":8.5}
```

The sample code detects and validates the desired state change, and prints out the details. Since we have no servo, the CHIP can't do anything more, but if you had a servo (or any other attachment you can control the state of) you could implement a function to update it in the real world.

This sample shows you how to get bi-directional metadata in and out of the CHIP using the AWS IoT service.

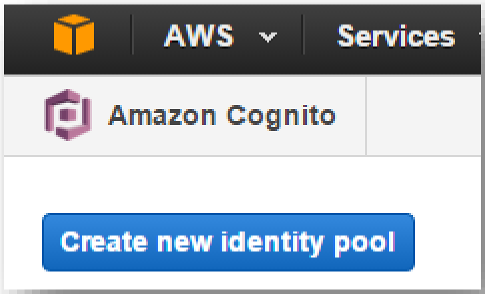
write-to-kinesis

The sample demonstrates how to use the AWS SDK for NodeJS to write a payload to Kinesis. It uses Cognito for authorization to access a Kinesis stream.

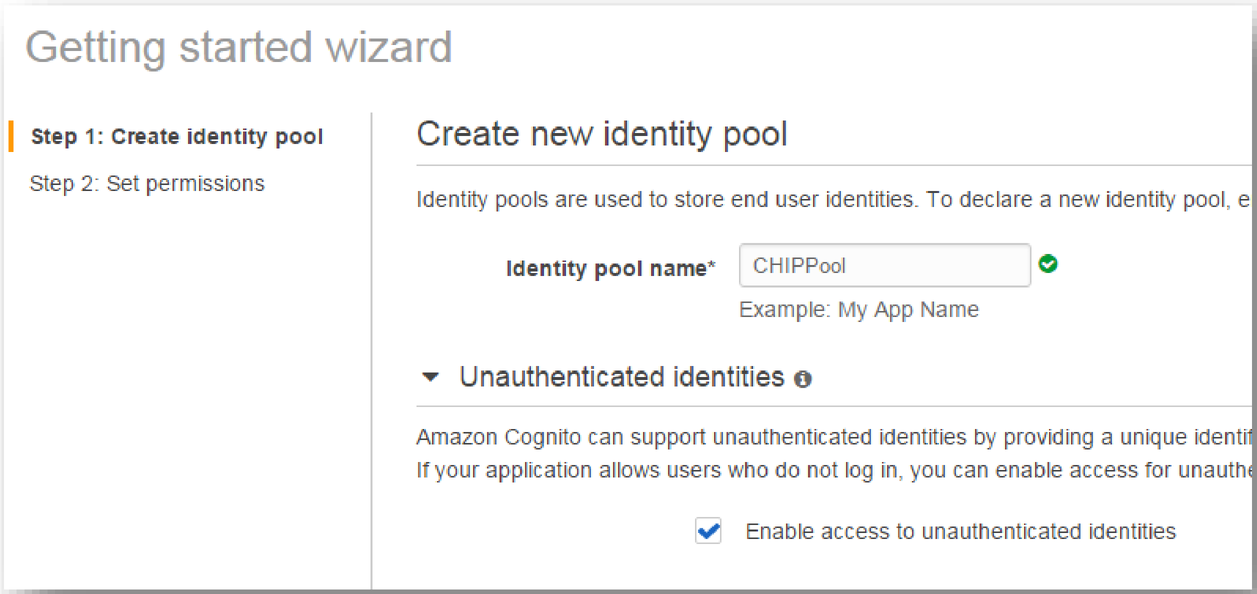
To set up, cd into the write-to-kinesis directory on the CHIP, and run:

```
sudo npm install
```

Now, you need to set up a Cognito Identity Pool. If you already have one, you can use that, but otherwise, go to the Cognito console and click Create new Identity Pool



Call your pool CHIPPool (or whatever you want!) and make sure you click Enable access to unauthenticated identities. We won't be using authenticated identities for this sample, but you are welcome to implement this in your code if you do not want to use unauth


Getting started wizard

Step 1: Create identity pool

Step 2: Set permissions

Create new identity pool

Identity pools are used to store end user identities. To declare a new identity pool, e

Identity pool name* CHIPPool

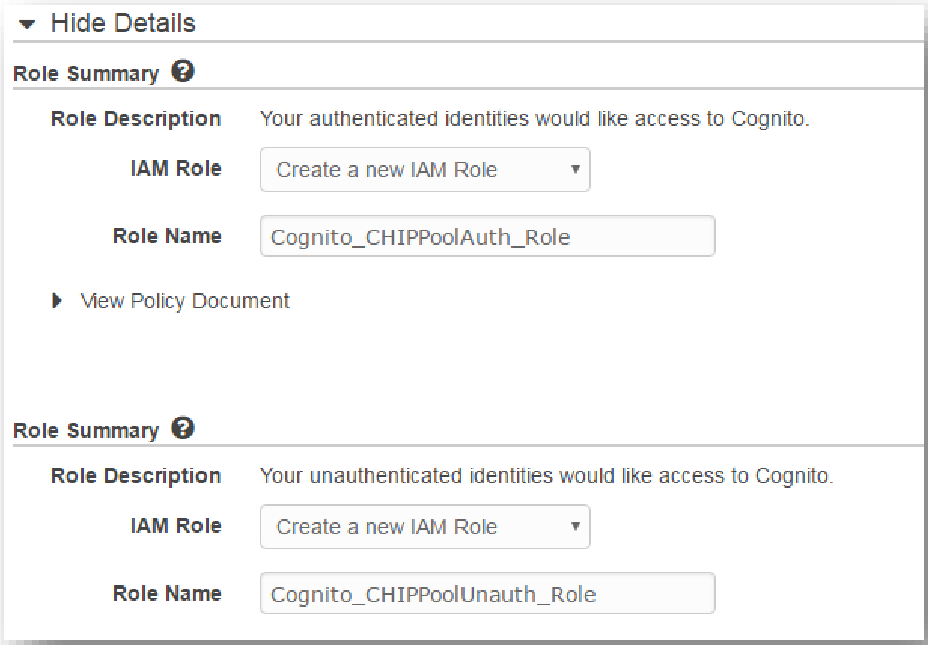
Example: My App Name

Unauthenticated identities

Amazon Cognito can support unauthenticated identities by providing a unique identi
If your application allows users who do not log in, you can enable access for unauthe

✓ Enable access to unauthenticated identities

Click Create Pool. The wizard will suggest creating two new IAM roles. You can modify this if you like, but for now, we will just use the suggested role names and create new roles.


Hide Details

Role Summary

Role Description: Your authenticated identities would like access to Cognito.

IAM Role: Create a new IAM Role

Role Name: Cognito_CHIPPoolAuth_Role

View Policy Document

Role Summary

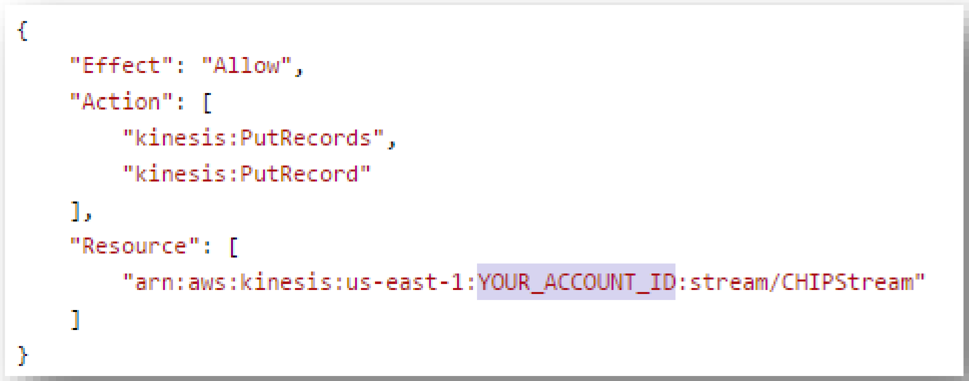
Role Description: Your unauthenticated identities would like access to Cognito.

IAM Role: Create a new IAM Role

Role Name: Cognito_CHIPPoolUnauth_Role

Before you create the roles, scroll down to the UnAuth role, and click the Policy Document twister to reveal the document. Click the Edit link. We will add permission for this role to write to the Kinesis stream we will create in the next step.

Add the following to the policy document:

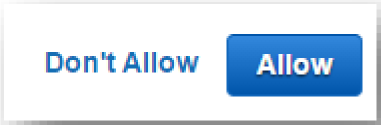


```
{
    "Effect": "Allow",
    "Action": [
        "kinesis:PutRecords",
        "kinesis:PutRecord"
    ],
    "Resource": [
        "arn:aws:kinesis:us-east-1:YOUR_ACCOUNT_ID:stream/CHIPStream"
    ]
}
```

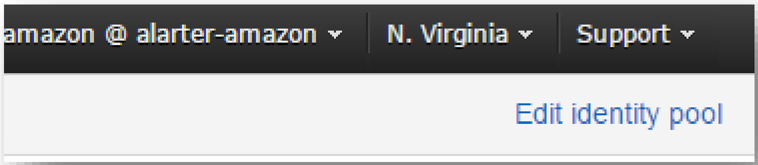
Be sure to put your account ID in place of the placeholder. The complete policy will be like this:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "mobileanalytics:PutEvents",
                "cognito-sync:*"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords",
                "kinesis:PutRecord"
            ],
            "Resource": [
                "arn:aws:kinesis:us-east-1:YOUR_ACCOUNT_ID:stream/CHIPStream"
            ]
        }
    ]
}
```

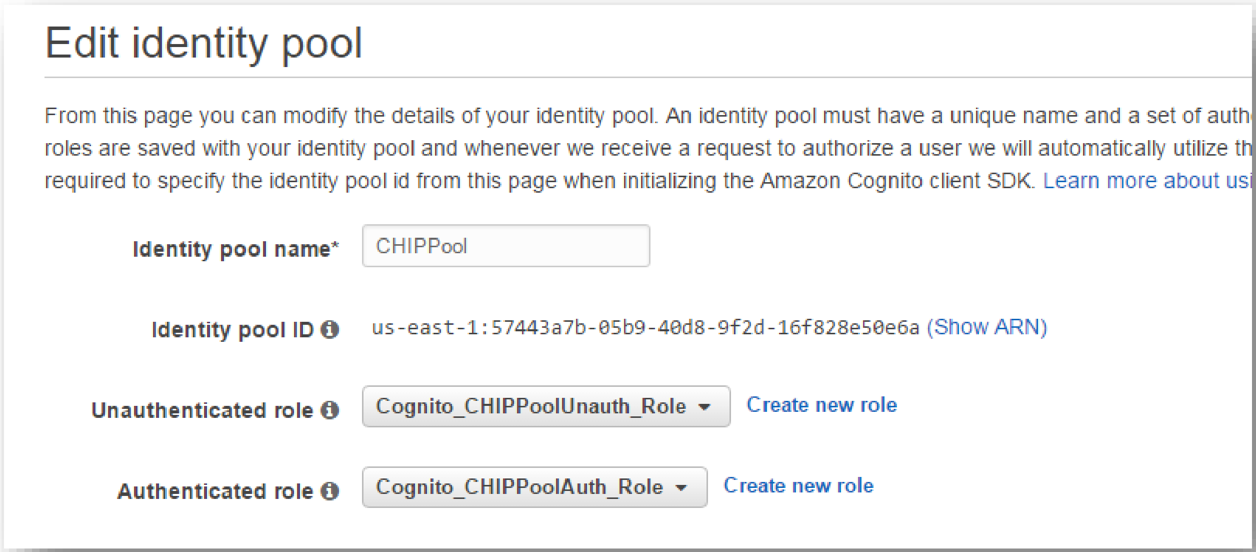
Click the Allow button



From the Cognito dashboard, top right, click the Edit identity pool button:



From the Edit identity pool page, node the Identity Pool ID:

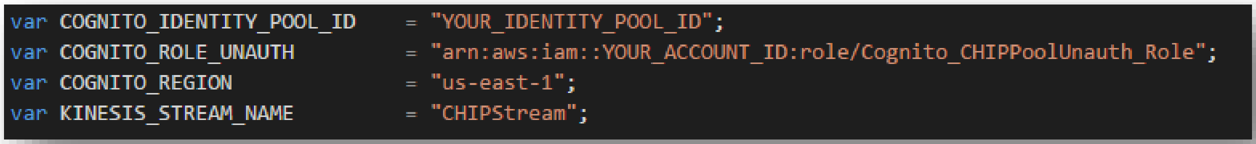


Edit identity pool

From this page you can modify the details of your identity pool. An identity pool must have a unique name and a set of auth
roles are saved with your identity pool and whenever we receive a request to authorize a user we will automatically utilize th
required to specify the identity pool id from this page when initializing the Amazon Cognito client SDK. Learn more about usi

Identity pool name*	CHIPPool
Identity pool ID	us-east-1:57443a7b-05b9-40d8-9f2d-16f828e50e6a (Show ARN)
Unauthenticated role	Cognito_CHIPPoolUnauth_Role Create new role
Authenticated role	Cognito_CHIPPoolAuth_Role Create new role

You will need to edit the index.js file with this (and other) details. Let's do that now:

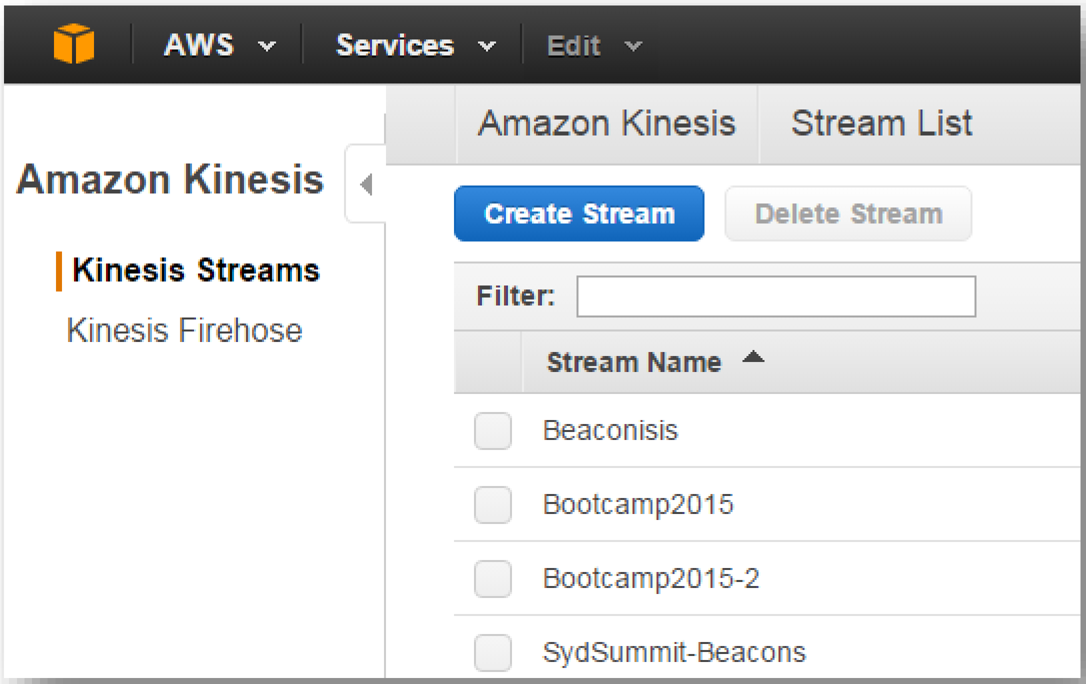


```
var COGNITO_IDENTITY_POOL_ID     = "YOUR_IDENTITY_POOL_ID";
var COGNITO_ROLE_UNAUTH          = "arn:aws:iam::YOUR_ACCOUNT_ID:role/Cognito_CHIPPoolUnauth_Role";
var COGNITO_REGION               = "us-east-1";
var KINESIS_STREAM_NAME          = "CHIPStream";
```

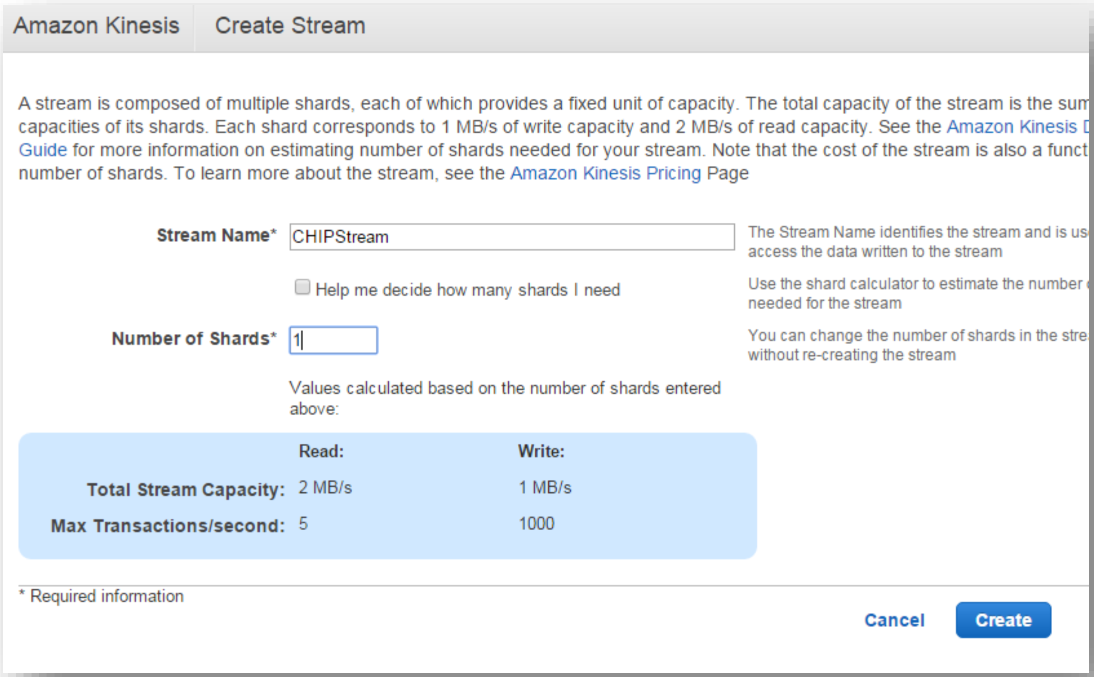
Add in the Cognito Identity Pool Id from above, into the COGNITO_IDENTITY_POOL_ID constant definition.

Also add the ARN of the UnAuthenticated IAM role that the Wizard created for you. If you didn't change the name, it will be *Cognito_CHIPPoolUnauth_Role* and you can use the template shown above, just add in your account Id

Now we need to create the Kinesis stream we will write to. Go to the Kinesis console, and click Create Stream:



Give the stream the name CHIPStream and provision 1 shard. Click Create.



A stream is composed of multiple shards, each of which provides a fixed unit of capacity. The total capacity of the stream is the sum capacities of its shards. Each shard corresponds to 1 MB/s of write capacity and 2 MB/s of read capacity. See the Amazon Kinesis D Guide for more information on estimating number of shards needed for your stream. Note that the cost of the stream is also a funct number of shards. To learn more about the stream, see the Amazon Kinesis Pricing Page

Stream Name* CHIPStream — The Stream Name identifies the stream and is us access the data written to the stream

☐ Help me decide how many shards I need — Use the shard calculator to estimate the number needed for the stream

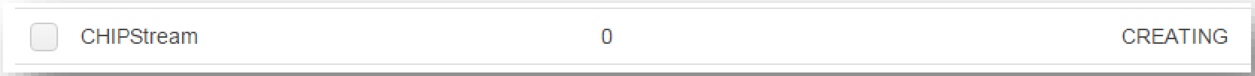
Number of Shards* 1 — You can change the number of shards in the stre without re-creating the stream

Values calculated based on the number of shards entered above:

	Read:	Write:
Total Stream Capacity:	2 MB/s	1 MB/s
Max Transactions/second:	5	1000

* Required information

The stream will take a moment to create.

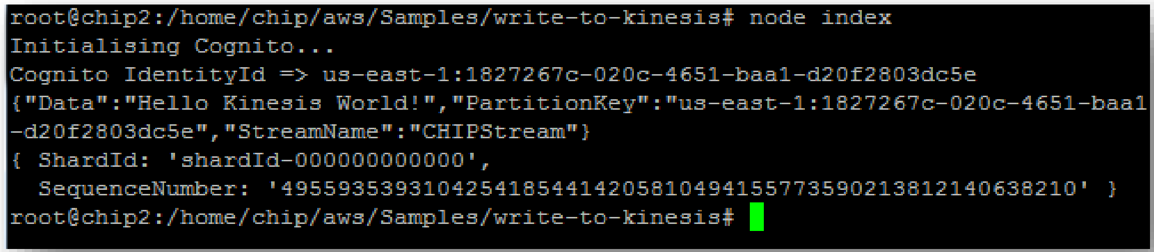


☐	CHIPStream	0	CREATING

Back on the CHIP, you’re ready to test once the stream is created. Run the sample app by issuing:

```
node index
```

If all is well, you will see a successful put to Kinesis, after a successful initialization and assume role via Cognito:



```
root@chip2:/home/chip/aws/Samples/write-to-kinesis# node index
Initialising Cognito...
Cognito IdentityId => us-east-1:1827267c-020c-4651-baa1-d20f2803dc5e
{"Data":"Hello Kinesis World!","PartitionKey":"us-east-1:1827267c-020c-4651-baa1
-d20f2803dc5e","StreamName":"CHIPStream"}
{ ShardId: 'shardId-000000000000',
  SequenceNumber: '49559353931042541854414205810494155773590213812140638210' }
root@chip2:/home/chip/aws/Samples/write-to-kinesis# 
```