CHIP Sample Projects

Table of Contents

[Install dev/build dependencies 1](#_Toc444693463)

[Install NodeJS 2](#_Toc444693464)

[Run Sample code setup 2](#_Toc444693465)

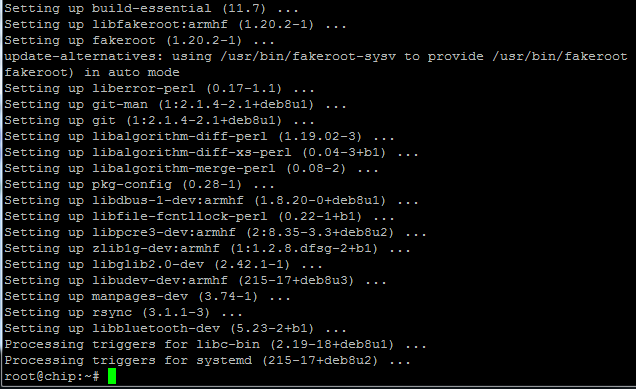
[Sample code 4](#_Toc444693466)

# 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.



# 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):



# 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.

Make or change directory to: /home/chip:

# cd /home/chip or

# mkdir /home/chip

Pull the setup and samples from the secure S3 repository:

# wget <http://apac-chip-challenge.s3.amazonaws.com/CHIP/GetCHIP.sh> | bash -

|  |
| --- |
| **Note:** If you have not correctly changed to the /home/chip direction the setup script for the samples will fail |

Mark GetCHIP.sh as executable,

# cd /home/chip

# sudo chmod 744 CHIPSetup.sh

And then run it. Running the setup will require to you provide a name for your CHIP. This name is used in the Central Registry to identify your CHIP as yours.

sudo ./GetCHIP.sh



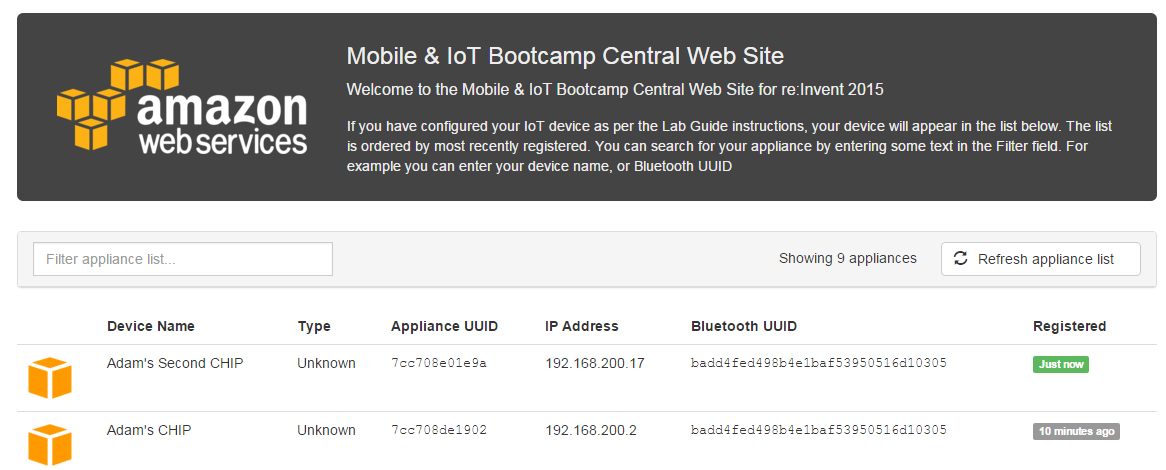
Setup takes a minute or two to pull in some NodeJS dependencies for the startup service. The script does the following:

* 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#](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.



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

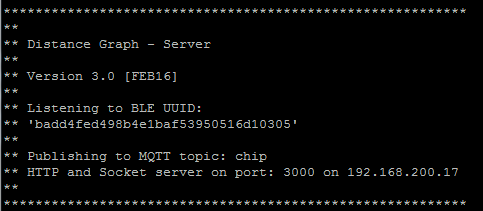
On one of the CHIPs, run

# sudo node server

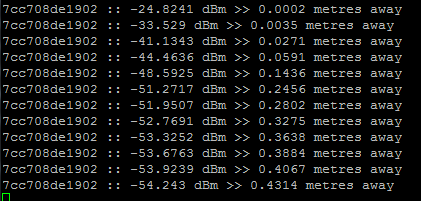
And on the other CHIP, run

# sudo node client

On the server CHIP, you will see this:

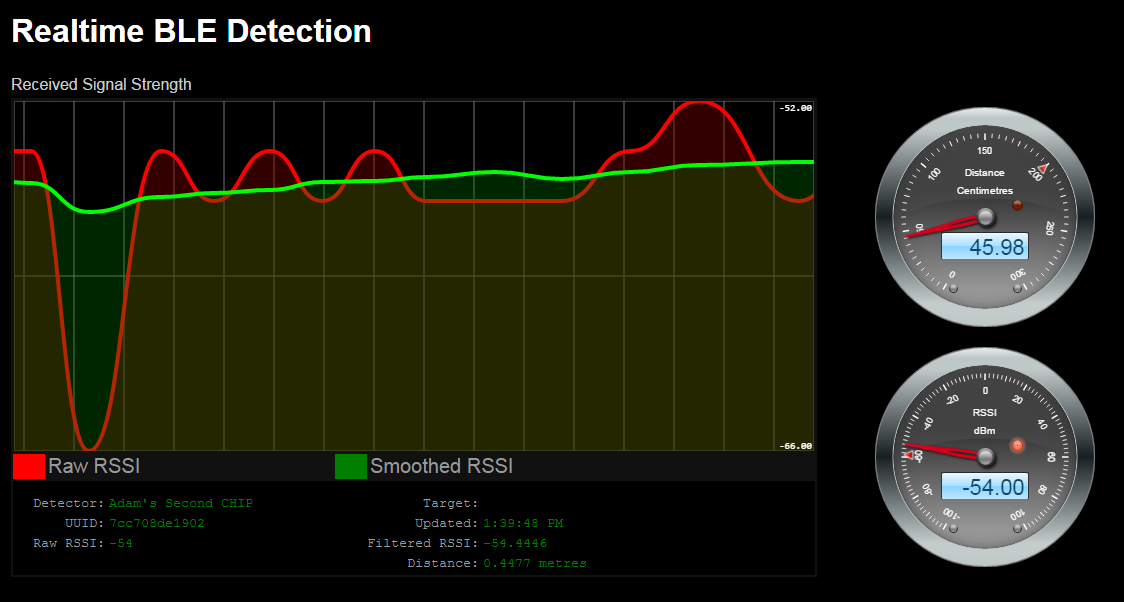


If you have the client running on the other CHIP, you will start to see advertisements received on the server:



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:



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.

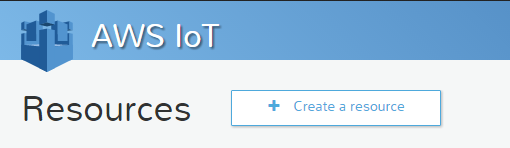
iot-mock-sensor

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.

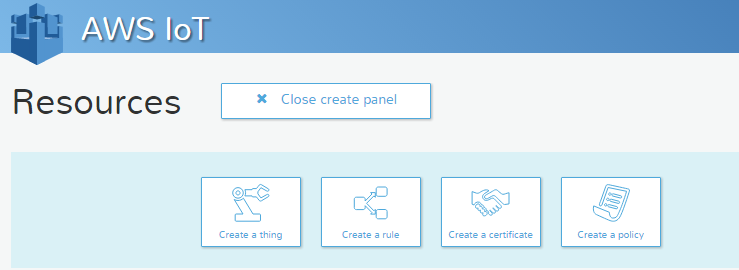
You need to set a few things up to use this demo. First, pull in the code and run:

sudo npm install

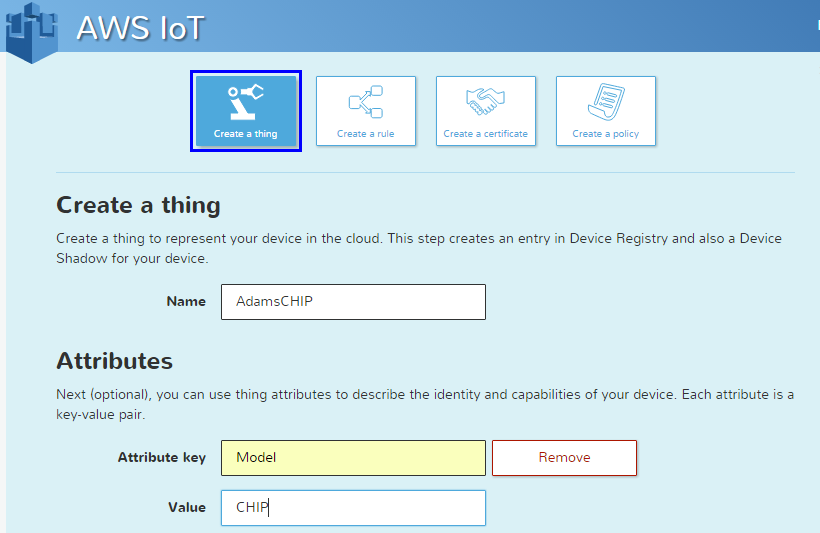
While that is happening, open your AWS console, and browse to AWS IoT. Click Create a Resource:



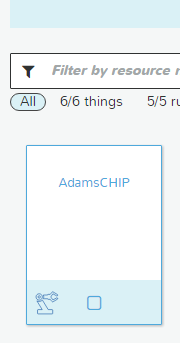
Then select Create a Thing



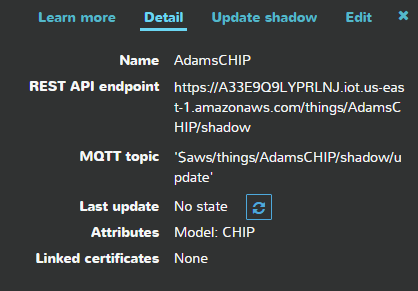
In the fields shown, give your new ‘thing’ a name (here, called AdamsCHIP, but you can use whatever name you like, as long as it matches the configuration we will do in the source code later)



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:



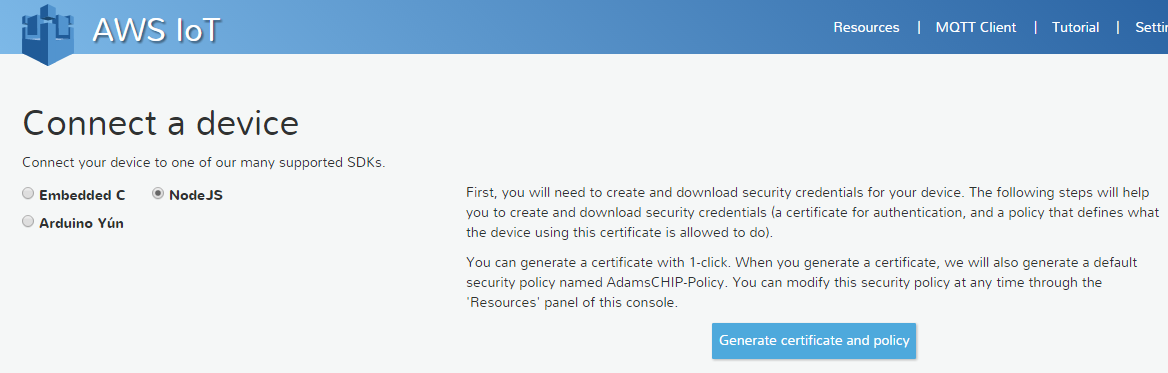
Click on the name of the thing, and view the information in the right-hand panel:



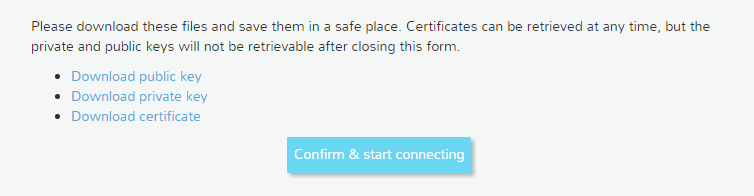
Note that you can see the model attribute you entered, as well as the fact that there is no current state – we haven’t connected a device and updated the shadow. To do that, we need to create a device and obtain the certificates to use on the CHIP. In the lower-right, click the Connect a Device option

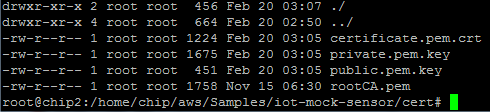


Choose NodeJS:



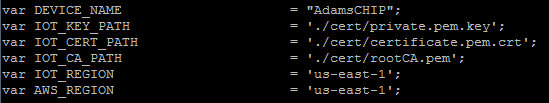
Then click Generate certificate and policy. The public key, private key and certificate will be generated and made available to download. You need to download each of these and store them on your machine for safe-keeping. You also need to copy the private key and certificate to the cert/ directory on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





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

On the CHIP, edit the config.js file

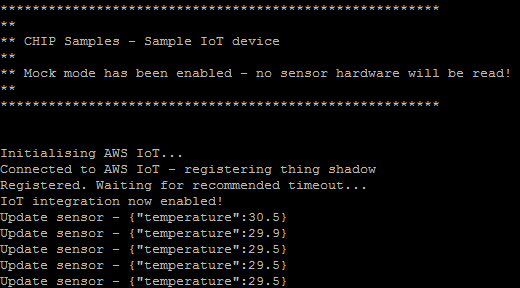


Ensure that the paths for the private key, certificate and root CA are correct, which they will be if you renamed the files you downloaded and removed the unique ID at the head of the filenames. If you prefer to keep the unique Id, update the config.js file accordingly.

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

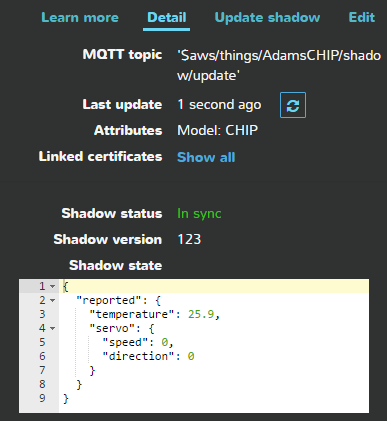
Now you’re ready to test it out! Run the following module:

sudo node device



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:

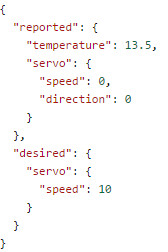


Also note that the reported state has a ‘servo’ section, as well as the temperature. The servo section is there to give an example of how we can command the CHIP to react to a change in metadata – in this case, we are pretending that there is a servo connected to the CHIP, and we can update the value 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!

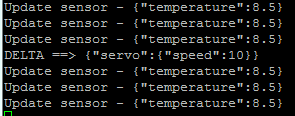
Click the Update Shadow link:



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:



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



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.

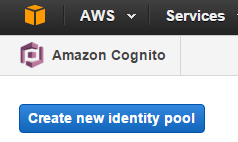
write-to-kinesis

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.

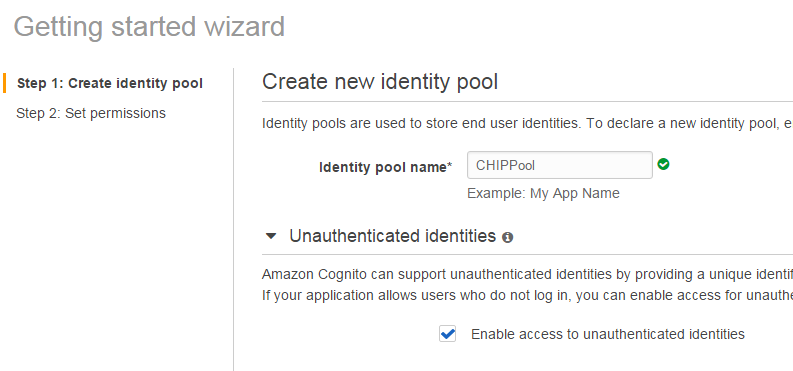
To set up, cd into the write-to-kinesis directory on the CHIP, and run:

sudo npm install

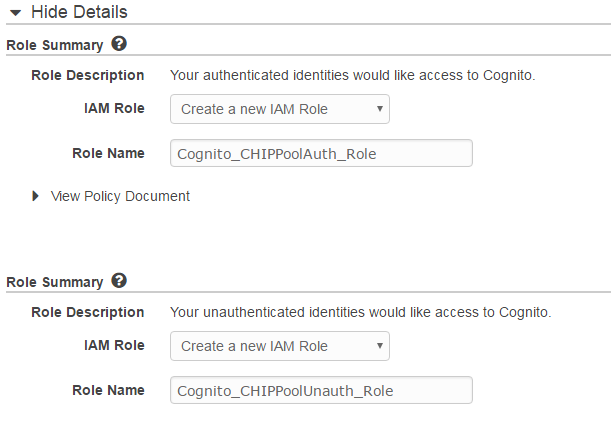
Now, you need to set up a Cognito Identity Pool. If you already have one, you can use that, but otherwise, go to the Cognito console and click Create new Identity Pool



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

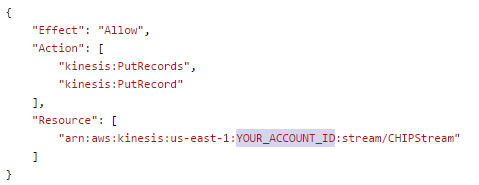


Click Create Pool. The wizard will suggest creating two new IAM roles. You can modify this if you like, but for now, we will just use the suggested role names and create new roles.



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:



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"

]

}

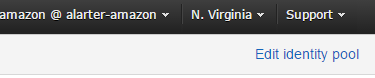
]

}

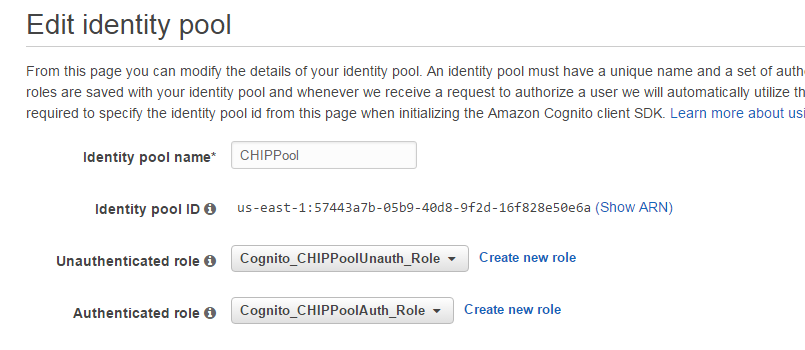
Click the Allow button



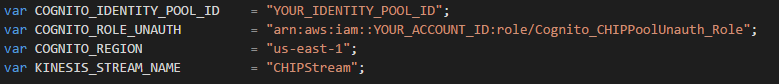
From the Cognito dashboard, top right, click the Edit identity pool button:



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



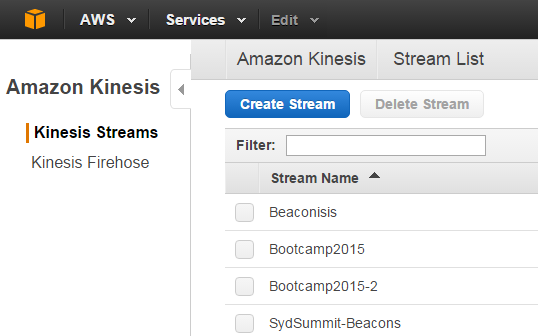
You will need to edit the index.js file with this (and other) details. Let’s do that now:



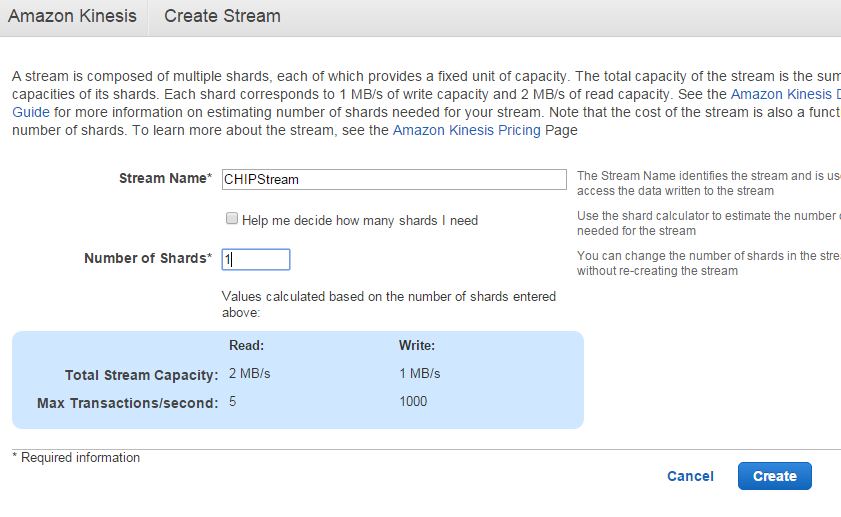
Add in the Cognito Identity Pool Id from above, into the COGNITO\_IDENTITY\_POOL\_ID constant definition.

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.



The stream will take a moment to create.



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:

