**AWS-IoT controlled LED Using Python and Raspberry Pi**

In recent years, we’ve seen that powering the best customer experiences and most reliable applications means leveraging cloud solutions like AWS IoT Core. For anyone getting started working with cloud services from a hardware background it can be an intimidating journey that requires learning a lot of new concepts and best practices so I wanted to share my experience and help bridge the gap between those skillsets.

In this guide, we’ll learn how to connect your Raspberry Pi Device to AWS IoT Core in order to communicate with it securely from the cloud. We’ll cover everything from the hardware and peripherals you need to using the AWS IoT Core console UI to control your Raspberry Pi and everything in between!

**Getting Started Materials**

Before you start, make sure you have all the materials we’re going to be using in this tutorial. you’ll need the following:

**Hardware:**

* A Raspberry Pi 4 or 3b+
* A microSD card with 8GB of space or more
* A power cable for the Pi
* An HDMI to micro HDMI cable
* A microSD card reader for your computer
* A USB keyboard
* A USB mouse
* A monitor with an HDMI port to display the Raspberry Pi’s Screen

**Software and Accounts:**

In addition to the hardware above, you’ll also need:

* An Amazon Web Services (AWS) Account.

After you create the account, make sure to save your login details somewhere as we’ll need them later.

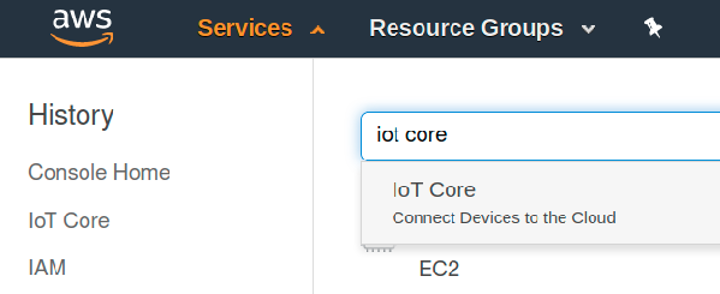
**Creating an IoT Thing in the AWS IoT**

**Thing Registry**

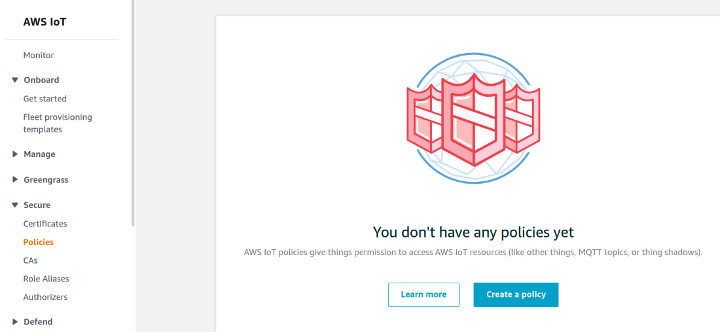
The next step for us is to register this IoT Device inside of AWS and get the credentials that will allow us to connect back and forth with the cloud.

Start by signing into your AWS Account while you’re on the Raspberry Pi. You can open up the Raspberry Pi OS web browser and go to aws.amazon.com before clicking “create account” or “sign in”. On the create an account page there should be an option to sign into an AWS Account where you can enter in your credentials.

Once you’ve signed in, then navigate to the “Services” dropdown and search for “IoT Core”:



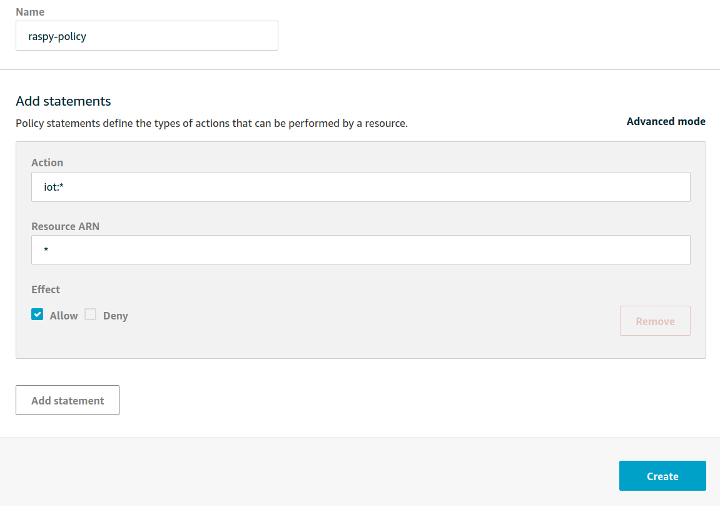
In the IoT Core console, go to Secure > Policies and click “Create a policy”:



This will allow us to create a policy that will give our IoT device permissions to act on things inside of the AWS cloud such as IoT Core MQTT Topics that allow us to send and receive data between the device and the cloud.

Let’s call our policy raspi-policy.

We’re going to give it some broad permissions that in a production environment that’s why, we’ll set the Action as “iot:\*” and the resource as “\*” along with selecting the “Allow” checkbox. Your screen should look like the photo below:

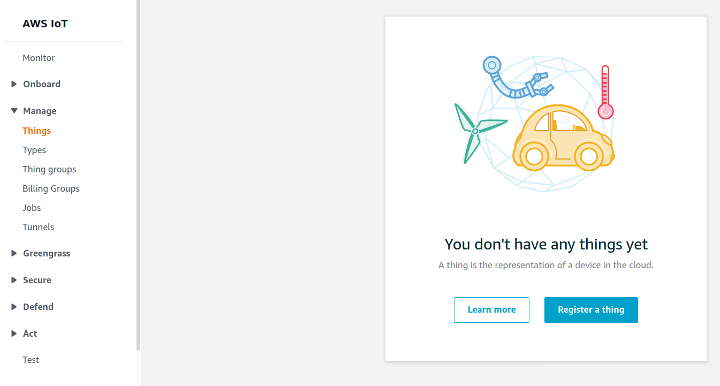


Let’s break this down a little:

* The Allow checkbox means that this policy will allow specific actions on AWS cloud resources. We could also set a policy that explicitly denies permissions to AWS but by default AWS denies all actions unless granted explicitly.
* The “iot:\*” action means that we are talking about ALL actions under the AWS IoT service umbrella. The asterisk indicates the “all” part here.
* The Resource ARN indicates what specific AWS resources we can perform these IoT actions on. In this case, we’re referring to any AWS resource.

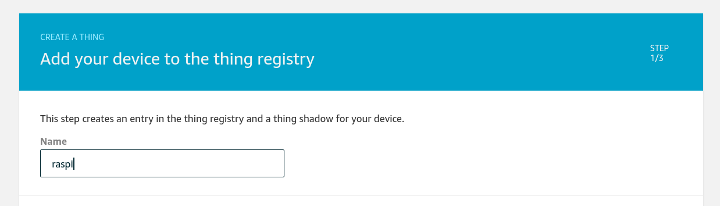
For our demo though we can leave these values and the allow checkbox as shown above and press “Create”.

Next, we’ll go to Manage > Things and press “Register a Thing”:



AWS IoT Core provides lots of great tooling for managing IoT “Things” that can help us sort things in to “Types” or “Groups” and use these settings to determine permissions these things might require and how we might want to interact with them. This can also be combined with some more appropriate permissions (as mentioned above) to help secure your IoT environments.

For right now though, we’ll use the easiest method of getting started and press “Create a Single Thing” and name our thing “raspi”:



We won’t bother with thing types, groups or attributes and we’ll click next.

**Securing Our Connection to AWS IoT**

On this next page we have the option to add a certificate for our thing. This is why I suggested you log into AWS while still on the IoT device – because for the device to speak to the cloud we’ll need the certificate on it locally.

At this point it might make sense to briefly summarize how public key cryptography works, how it applies to IoT devices and how that differs from typical web-based public key scenarios.

**Securing Websites with Public Key Cryptography**

If you have a website that you want to allow your users to connect securely to you can request a certificate for your website from a Issuing Certificate Authority or CA. This CA goes through a very stringent process to securely create a public and private key and store the private key securely. There is then another layer called a Root CA which signs the issuing certificate authority’s certificate.

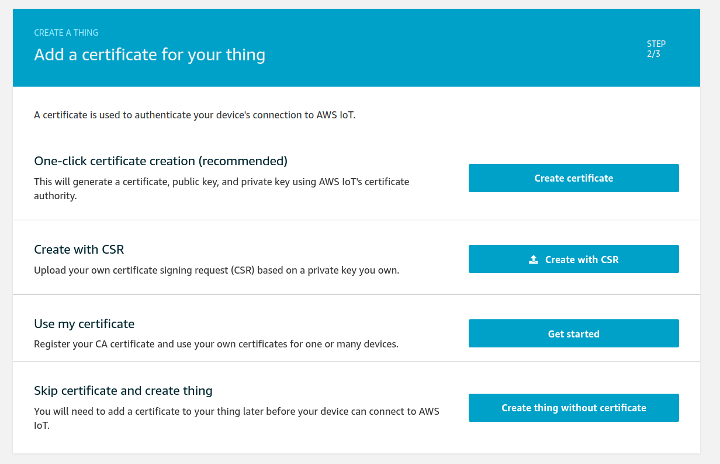
These Root CAs then work with hardware and software vendors like web browsers and consumer electronics like laptop manufacturers in order to bundle public keys in with the devices they sell. This allows a person to visit your website, review the website certificate you provide, see who signed it, review that CA certificate and check it against the Root CA certificate that was already bundled on their machine. After confirming the Issuing Certificate Authority’s certificate is good it will then use that certificate to verify yours and then establish a connection with your site using that information.

While we’re ignoring a few other layers of public key infrastructure (PKI) around how revoking certificates works and newer changes to PKI related to public listings of issued certificates, this is essentially how securing website works on the web. But what happens when you are making the devices yourself?

**Securing IoT Devices with Public Key Cryptography**

Let me start by saying that there are other mechanisms for trying to implement security on IoT devices. Access keys or tokens can be used on embedded on devices to perform some of the same operations. However, these methods rely on symmetric encryption and are considered less secure because the key must be protected on both the cloud and the device. Additionally, when using access keys or tokens like this they are more likely to be abused and stored in plaintext or hardcoded in software which makes them more accessible to attackers. For these and other reasons, a public key cryptography approach is preferred

When connecting our device to AWS IoT we can use public key cryptography to do so. Within that, we have a few options for generating certificates:

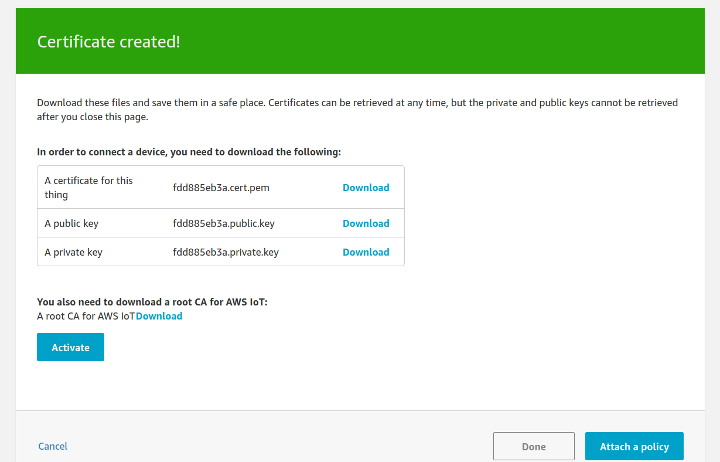


Let’s see how these options work:

* **One-click certificate creation (recommended) –**AWS generates a public/private key that we’ll be using for our device. It then takes the public key and makes a Certificate Signing Request or CSR that it signs with its own CA Private Key to generate a certificate we can use on our device. This all happens in the background and then AWS allows us to download the public/private key and the certificate it created for us so we can authenticate our device later on.We also download a Root CA certificate from AWS that we use to authenticate our connection with AWS.
* **Create with Certificate Signing Request**– In this model, we have our own public and private key already, either on our laptop or maybe provisioned already on our IoT device. We then use our keys to create a certificate signing request on our end before sending that CSR to AWS. AWS will then send us back a certificate which we can use in combination with our private key on the device. But like the first option, this requires us to have permissions to call AWS APIs for every new certificate we want to generate.
* **Use my Certificate –**In this option, we can register our own certificate authority certificate so that we can sign one or many kinds of devices. Then, we can use that certificate to sign new certificate requests without having to make a request to AWS. This is the most common approach when working on production services where we are creating a large number of devices where we want to provision a unique certificate for each.

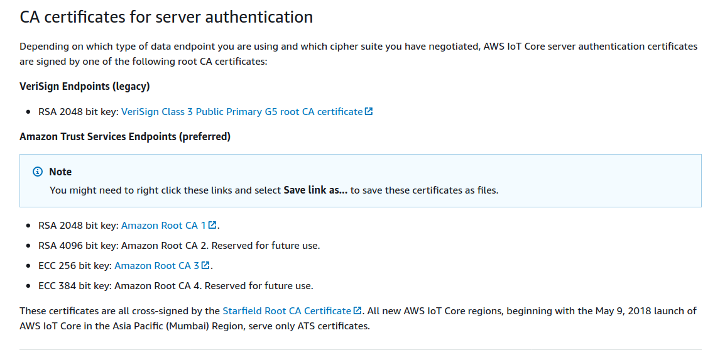
For right now, given that we’re only provisioning one device and that we’re not deploying this device into the wild beyond our home networks

After we press this, we should download all the created certificates to our Raspberry Pi device:



Download all three of these files  **and then make sure to press “Activate”**. Because AWS has already used the public key to help generate a certificate for us we will mainly be using the certificate and the private key to authenticate and sign our requests to AWS.

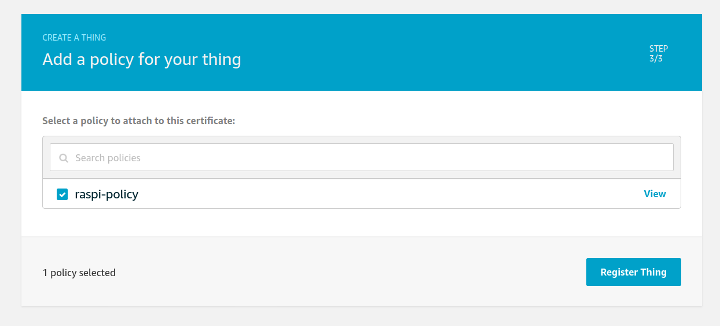
Also, on this page we need to open the link for the root CA download which is right above the “Activate/Deactivate” button:



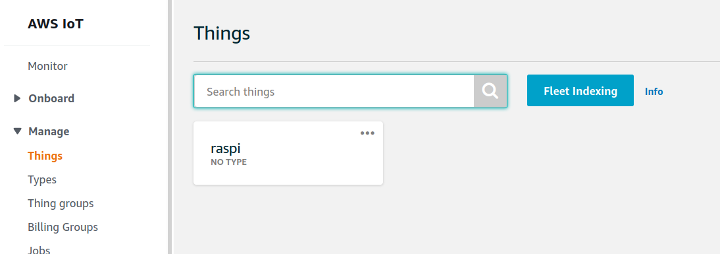
From this page, download the Amazon Root CA 1. Right click the link and click “Save link as” and save it to the Raspberry Pi. This certificate will allow us to trust AWS IoT itself.

Once you’ve downloaded the Root CA certificate along with the three other certificates, you should have four total files on your Raspberry Pi. Go back to the “Certificate created!” page and make sure to press “Activate” if you haven’t already.

Then you can press the “Attach a policy” button. On this page, you can select the checkbox next to the “raspi-policy” we created earlier and then press “Register thing”:



Now we should have the certificates we need on our device to have it start talking to AWS and it should show up in the Things portion of IoT Core:



Next, we need to download some dependencies and write some code!

**Installing Dependencies on Our**

**Raspberry Pi**

In this tutorial, we’ll be using Python 3.

Fortunately for us, Raspberry Pi OS should come with Python 3 installed on it already. If you’re working with another operating system on the Raspberry Pi, you will want to install Python 3 with at least version 3.5 or later.

Raspberry Pi OS also comes with git installed so if you’re using a different operating system, you’ll want to install that too.

The next step will be to install the AWS IoT SDK for Python. Before we do this, we’ll need to gather some dependencies that are not included on our Raspberry Pi OS. We’ll do this by running the following commands and press Y when prompted to add the dependencies to our system:

sudo apt-get update

sudo apt-get install cmake

sudo apt-get install libssl-dev

After we run these commands, we should be able to use pip3 to install the AWS IoT SDK:

pip3 install awsiotsdk

This installation process might take a few minutes but when it’s finished, we can get the code we need for this tutorial. To do this, run:

git clone <https://github.com/Witekio/aws-iot-core-raspberry-pi-demo.git>

**Syncing Our Raspberry Pi with an AWS IoT Shadow**

Now that we have the code, we’ll need both code files in the same directory as our certificates.

cd raspberry-pi-aws-iot

mv ~/Downloads/<your-certificate-name>-certificate.pem.crt .

mv ~/Downloads/<your-certificate-name>-private.pem.key  .

mv ~/Downloads/AmazonRootCA1.pem .

Replace “<your-certificate-name>” with the actual first part of your certificate names.

At this point, we might also want to make sure we change the permissions on the downloaded files to make them only readable to the current user. We can do this by running the following commands in the same directory as the certificates are now:

chmod 400 <your-certificate-name>-certificate.pem.crt

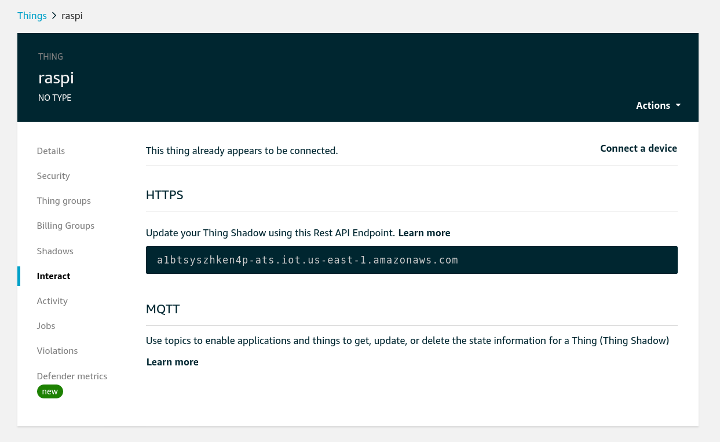
chmod 400 <your-certificate-name>-private.pem.key

chmod 400 AmazonRootCA1.pem

Remember to replace the part of the commands with <these> in them.

In a production scenario, we probably wouldn’t even provision these files in this way. Instead, we’d have a process for storing sensitive information like this inside something like a Trusted Platform Module or TPM. This allows us to be more confident that the certificates can’t be stolen off the device and used in some other system. While devices might still be compromised, with a TMP it becomes much harder to steal the keys and use them elsewhere without us noticing.

The last thing we’ll need to get before syncing our device to the cloud is the endpoint URL for our Thing. In order to find the endpoint URL you can click into the details of your new AWS IoT Thing and look under the “Interact” section. You’ll notice in this section there is an HTTPS URL and an MQTT section:



This is because AWS IoT supports connections using multiple protocols including HTTPS, MQTT and MQTT over WebSocket.

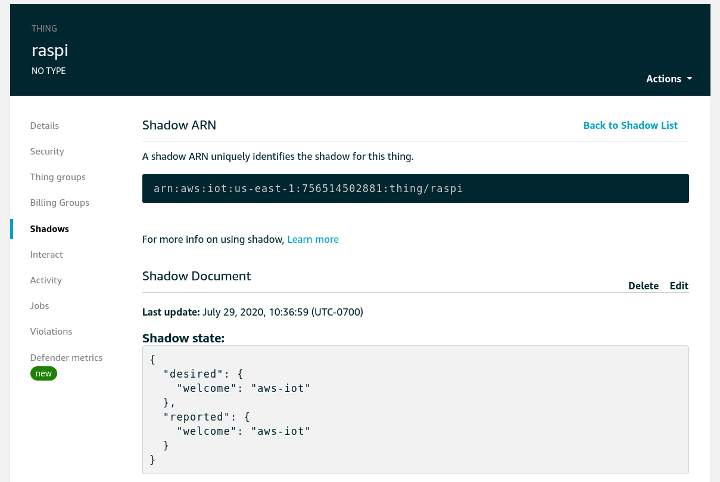
On this page, copy the endpoint value under HTTPS down and use it, along with the names of the certificates you have in the present directory, to replace the “<your-certificate-name>” and “<endpoint>” sections of this command:

python3 shadow.py --endpoint <endpoint>-ats.iot.us-east-1.amazonaws.com \  
--cert <your-certificate-name>-certificate.pem.crt \  
--key <your-certificate-name>-private.pem.key \  
--root-ca AmazonRootCA1.pem \  
--thing-name raspi \  
--shadow-property light

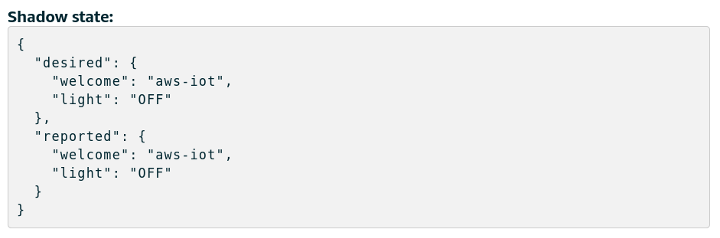
After you update it, it would look something like this:

python3 shadow.py --endpoint a1btsyszhken4p-ats.iot.us-east-1.amazonaws.com \  
--cert fdd885eb3a-certificate.pem.crt \  
--key fdd885eb3a -private.pem.key \  
--root-ca AmazonRootCA1.pem \  
--thing-name raspi \  
--shadow-property light

Before you run this command, go over to the “Shadows” section of your “raspi” thing and click “Classic Shadow”. You should see something like this:



Now, run the command above and see what happens on your device. You should see an output that describes the process of syncing with AWS IoT and then a prompt that allows you to update the state of the device from the device itself. Type in “OFF” and press enter. Then see what happens to the IoT Code Shadow state:



Now, click the “Edit” option in AWS IoT Core and update your shadow by editing the “Desired” property of “light” to “ON”. You should see the device report the new updated state in the terminal!

All of this means we’ve successfully connected our device to AWS IoT Core. In the background what’s happening here is that our “shadow” here has a desired and reported value. When we update the desired value the script running on the device polls an MQTT topic provided by AWS IoT Core to see if anything has changed. If it sees a new “desired” value then it knows it needs to update the state on the device. This triggers the script on the device to update the “reported” value in the MQTT shadow for the device. In the long run, this will allow us to act on changes to the desired or reported state of a device and manage connections between the cloud and devices more effectively, especially in situations when network access cannot be maintained – something common for many IoT devices.

Now we need to set up our hardware and have the cloud control an LED on our device!

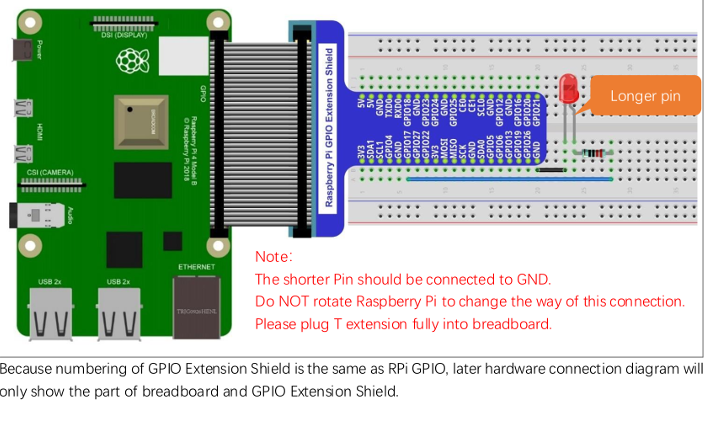
**Hardware Configuration and GPIO Testing**

Specifically, I’ll be using:

1. A breadboard
2. A Raspberry Pi GPIO Extension Board
3. A 40-pin extension cable
4. One red LED
5. One 10K Ohm resistor
6. Two jumper wires

For the hardware portion of this tutorial, we’ll be remotely controlling one of the Raspberry PI’s GPIO pins in order to provide power for an LED when the “light” property of the device is set to “ON”. For right now, shutdown and unplug your Raspberry Pi so you can setup the breadboard and the LED.

You can turn the Raspberry Pi back on after you have the breadboard configured as shown in the tutorial photo on page 55 here:



At this point, we can test the hardware configuration using some of the code we got from GitHub earlier. Change directories into the GitHub repo you cloned and make sure you see the “raspi.py” file.

With the terminal in the same directory as “raspi.py”, type “python3” and press enter in order to get into the python3 interpreter. Then, run the following commands (without the >>>) one at a time and wait for a second or two between each to look at your LED:

>>> from raspi import \*  
>>> setup()  
>>> turn\_on()  
>>> turn\_off()  
>>> destroy()

How did it go? Did you see the LED turn on and off when you ran those corresponding commands? If not, you might have a different hardware configuration than I do. This code assumes that you’ve followed the guide above and used GPIO 17 in order to power your LED. If you’re using some other GPIO then you will need to either change the script or update your hardware configuration to make it work. Make sure to leave the device off if you need to change some of the wiring!

If this works for you, we can proceed to the next step – doing this all from the cloud!

**Controlling Our Raspberry Pi’s GPIO from the Cloud**

Now that we have our hardware setup, our Raspberry Pi dependencies installed, and our device can connect to AWS IoT Core we should be able to control the LED from the cloud!

The first step is to slightly modify the shadow.py file. Start by opening it up in the text editor on the Raspberry Pi. The easiest way to do this from the terminal is by running xdg-open shadow.py

Then, scroll down a little in the file and change the line that says “HARDWARD\_CONFIGURED = False” to this:

“HARDWARD\_CONFIGURED = True”

Make sure to use a capital “T” for “True”! Then save the file and go find that command you ran earlier that looks something like this except with your certificate names:

python3 shadow.py --endpoint a1btsyszhken4p-ats.iot.us-east-1.amazonaws.com \  
--cert fdd885eb3a-certificate.pem.crt \  
--key fdd885eb3a -private.pem.key \  
--root-ca AmazonRootCA1.pem \  
--thing-name raspi \  
--shadow-property light

Run this command again. Now when it prompts you for the “light” property enter “ON” or “OFF” from the Raspberry Pi terminal and you should see the LED turn on and off!

But for the best part, try updating the Shadow inside of the AWS IoT Console and change the desired state for the light property to “ON” or “OFF” and watch it be reflected in your LED!

Nice work! By now you’ve managed to sync the state of your Raspberry Pi between the device itself and AWS IoT Core. You’ve also used Python to control the GPIO pins on your Raspberry Pi and used it to enable the controlling of those pins from AWS.

From here, you should feel comfortable doing more projects that combine AWS IoT with hardware connected to your device!