TI Designs: TIDM-1020

SimpleLink™ CC3220 Wireless MCU-Based Thermostat Reference Design Supplement with Alexa Voice Control

Description

This TI design guide supplement provides a software reference that builds on the existing SimpleLink™ CC3220 Wireless MCU-Based Thermostat Reference Design with BLE Provisioning with an implementation of Amazon Web Services (AWS) Internet-of-Things (IoT) connectivity to the cloud. With the emergence of voice-controlled ecosystems, implementing voice control from the cloud is a key component of many smart thermostat systems. This design guide supplement provides details on how to configure Alexa Voice Services (AVS) cloud and the AWS environment needed for voice control using Alexa. This design guide supplement is for thermostat end-equipment developers, engineers, and system evaluators.

This supplement provides reference code to demonstrate integration of the CC3220SF based smart thermostat reference design with the Amazon cloud, as well as an AWS Lambda function demonstrating how to interface Alexa to the AWS IoT functionality using the CC3220SF device. This reference design also highlights the existing features of the smart thermostat including: low-power connection to the Internet and cloud, remote monitoring and control, and enhanced security capabilities like secure Over The Air (OTA) updates of the device and application firmware.

Resources

TIDM-1020	Design Folder
TIDUEC9	Design Guide
CC3220SF-LAUNCHXL	Product Folder
BOOSTXL-SENSORS	Product Folder
BOOSTXL-K350QVG-S1	Product Folder
SEEED STUDIO GROVE	Product Folder

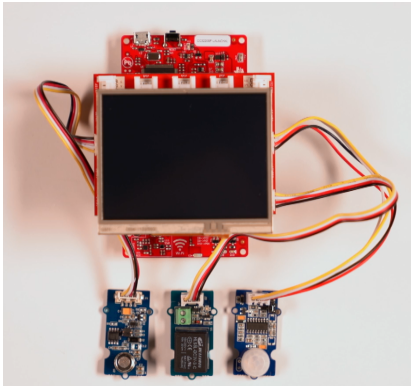
TI E2E™ Community — ASK Our E2E™ Experts

New Features

- Demonstrates connectivity to the AWS IoT cloud
- Interfaces with AVS and allows for direct voice control through Alexa

Applications

- Thermostat
- HVAC System Controller
- Boiler System
- Weather Station
- Wireless Environmental Sensor
- Air Quality and Gas Detection



An IMPORTANT NOTICE at the end of this TI reference design addresses authorized use, intellectual property matters and other important disclaimers and information.

1 System Description

This supplement builds on the TIDM-1020 foundation and enables the use of Amazon's Alexa Voice Services to remotely control the smart thermostat with voice. This is achieved by removing the IBM cloud connectivity used in the base TIDM-1020 design, and replacing it with AWS IoT functionality that allows for direct connectivity to Amazon's suite of cloud services. Alongside the AWS IoT interface, the AWS cloud and AVS configuration are also demonstrated to build a fully-functioning voice-controlled smart device.

For this AWS IoT variant of the smart thermostat design, the same hardware as the existing TIDM-1020 is used. Furthermore, most of the code of TIDM-1020 is reused, leveraging theSimpleLink™ SDK platform. This common SDK code can easily be used with other SimpleLink connectivity platforms as well as the MSP432™ platforms. The entire design, including the firmware and hardware components, can be reconfigured or modified to suit specific system requirements and allow for scalability, should additional connectivity needs arise in the future.

The full description of the smart-thermostat features as well as instructions implementing the thermostat using the CC3220 family of devices, are provided in the main SimpleLink™ CC3220 Wireless MCU-Based Thermostat Reference Design with BLE Provisioning design guide. The scope of this TI reference design supplement is to implement voice control using Amazon Alexa. As such, unchanged sections of the TI design guide will not be repeated in this supplement. Only modified sections will be included, and it can be assumed that for omitted sections the corresponding section of the main design guide is still applicable.

1.1 Key System Specifications

Table 1 lists the key system specification and features of the TIDM-1020.

Table 1. Key System Specifications

FEATURE	SPECIFICATION	ADDITIONAL DETAILS
Sensor integration	Temperature, humidity, air quality, atmospheric pressure	The TIDM-1020 demonstrates the integration of both analog and digital sensors. In this design, an air quality sensor is interfaced as an analog sensor.
Cloud connectivity	AWS IoT Cloud	Interface to the AWS cloud services
Provisioning	AP mode and SmartConfig™	Uses SimpleLink™ Wi-Fi® Starter Pro mobile application for Apple iOS or Android™ devices
HMI	HMI using a Kentec display and resistive touchscreen	—
Serial interface requirement	I^2C – all digital sensors SPI – interface to HMI ADC – analog sensors GPIOs – relay and LED control	This design can be used as a reference for the peripherals listed.
PIR sensor	Proximity-based display turn-on feature	—
Memory use – code	338 KB	Integrated development environment (IDE): Code Composer Studio™ (CCS) RTOS: TI-RTOS Compiler: TI v18.1.1.LTS

2 System Overview

This section is omitted as the content of this section is the same as that of the SimpleLink™ CC3220 Wireless MCU-Based Thermostat Reference Design with BLE Provisioning design guide.

3 Hardware, Software, Testing Requirements, and Test Results

3.1 Required Hardware

This section is omitted as the content of this section is the same as that of the SimpleLink™ CC3220 Wireless MCU-Based Thermostat Reference Design with BLE Provisioning design guide.

3.2 Getting Started Firmware

3.2.1 Required Software

- TIDM-1020 Software
- Code Composer Studio (CCS) Integrated Development Environment (IDE) (Arm® Compiler TI v18.1.1.LTS)
- SimpleLink™ Wi-Fi® CC3220 Software Development Kit (SDK) v2.20.00.10
- Bluetooth® Plugin for SimpleLink™ MCU SDK v1.40.00.42 (required for BLE provisioning)
- Sensor and Actuator Plug-ins for SimpleLink™ MCU SDKs v1.20.00.02
- SimpleLink™ SDK Plugin for Amazon Web Services v.2.00.00.09
- SimpleLink™ CC2640R2 SDK - Bluetooth® low energy (required for BLE provisioning)
- CCS UniFlash 4.3.1.1835 or newer
- SimpleLink SDK Explorer, mobile application for Apple iOS or Android devices (required for BLE provisioning)
- **or** SimpleLink™ Wi-Fi® Starter Pro, mobile application for Apple iOS or Android devices

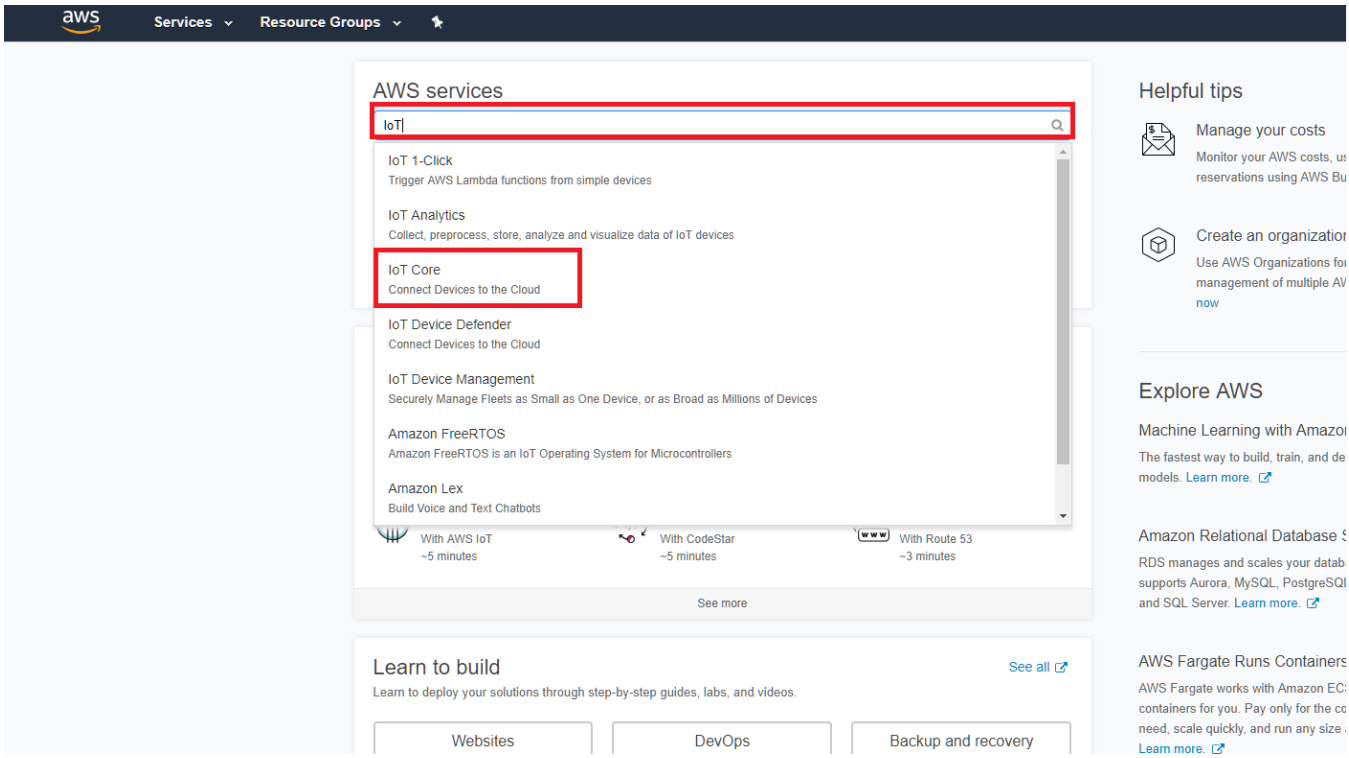
To import the software project into CCS, the required SDK and Plugin packages must be installed.

3.2.2 Opening and Configuring Amazon Web Services Account

3.2.2.1 *Configuring AWS IoT and Adding a Thing*

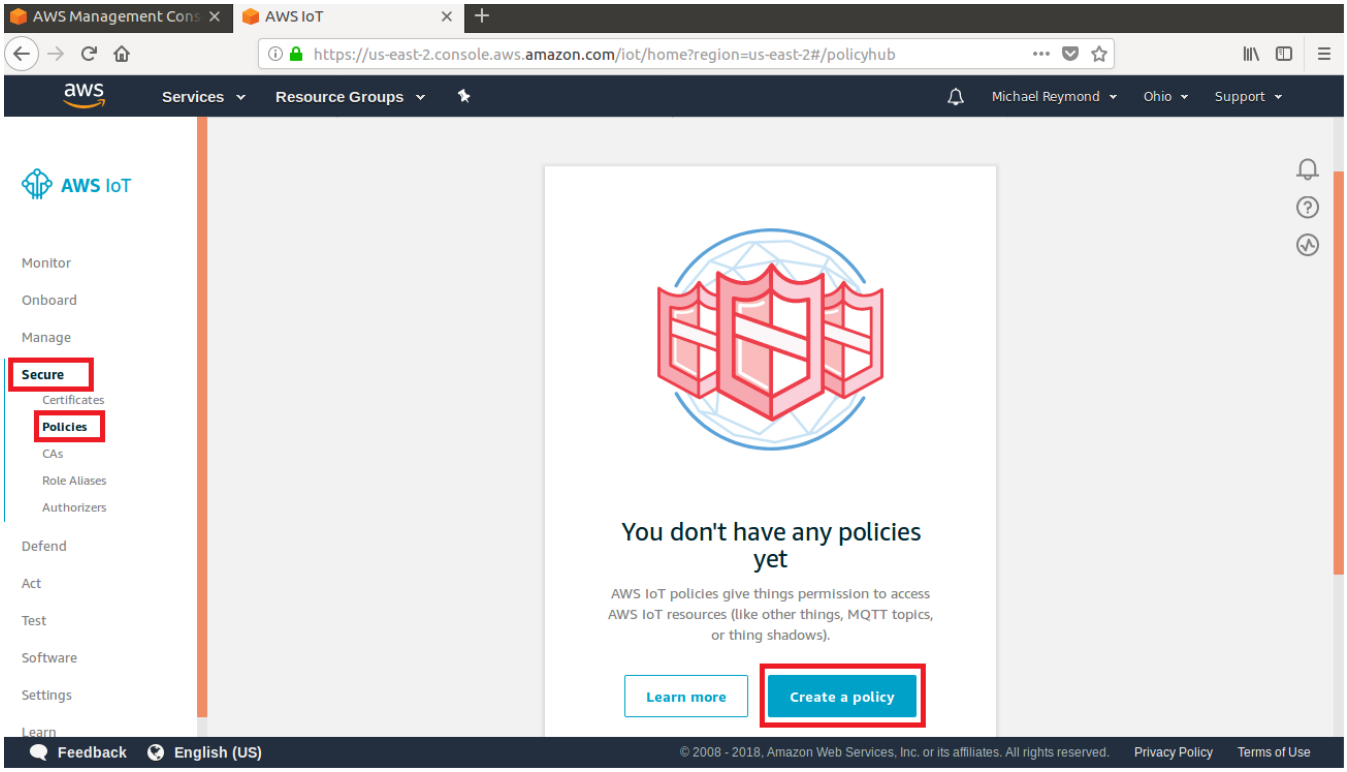
1. First, sign up for an AWS account at aws.amazon.com. The free 12 month trial account is sufficient for the purposes of this TI Design. Once you have successfully signed up, access the main developer console at console.aws.amazon.com.
2. Once at the console, access AWS IoT by typing in “IoT” into the search bar under “AWS services”, and then click on “IoT Core” in the drop-down menu that appears as shown in Figure 1.

Figure 1. Accessing AWS IoT Core



3. In the next welcome screen, click “Get started” to get to the main dashboard. Then, click on Secure, then on Policies. In there, click on the “Create a policy” button as shown in Figure 2.

Figure 2. Accessing AWS IoT Policies

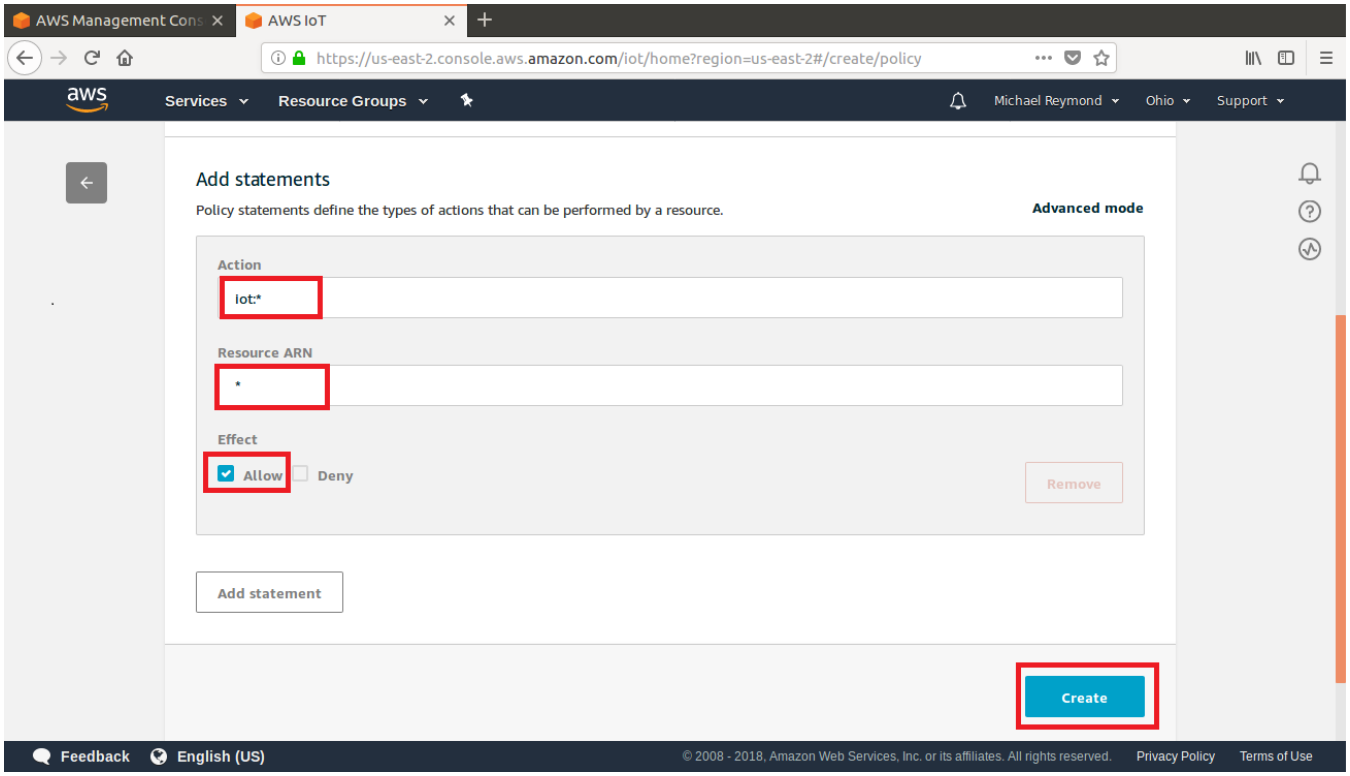


Type in a name for your policy, and then ensure that the following policy statements are configured:

- Action: iot:*
- Resource ARN: *
- Effect: Allow

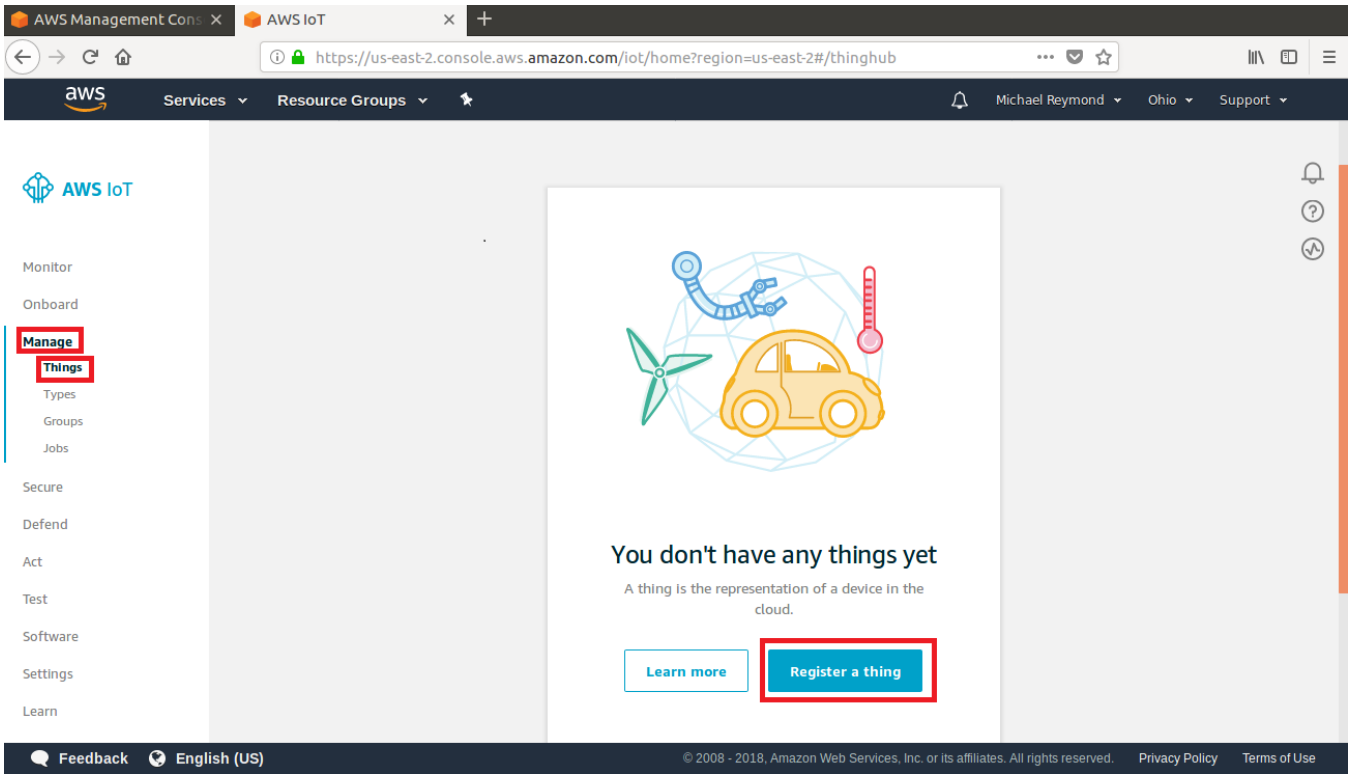
4. The policy settings should look like Figure 3. Then, click on "Create".

Figure 3. Creating the AWS IoT Policy



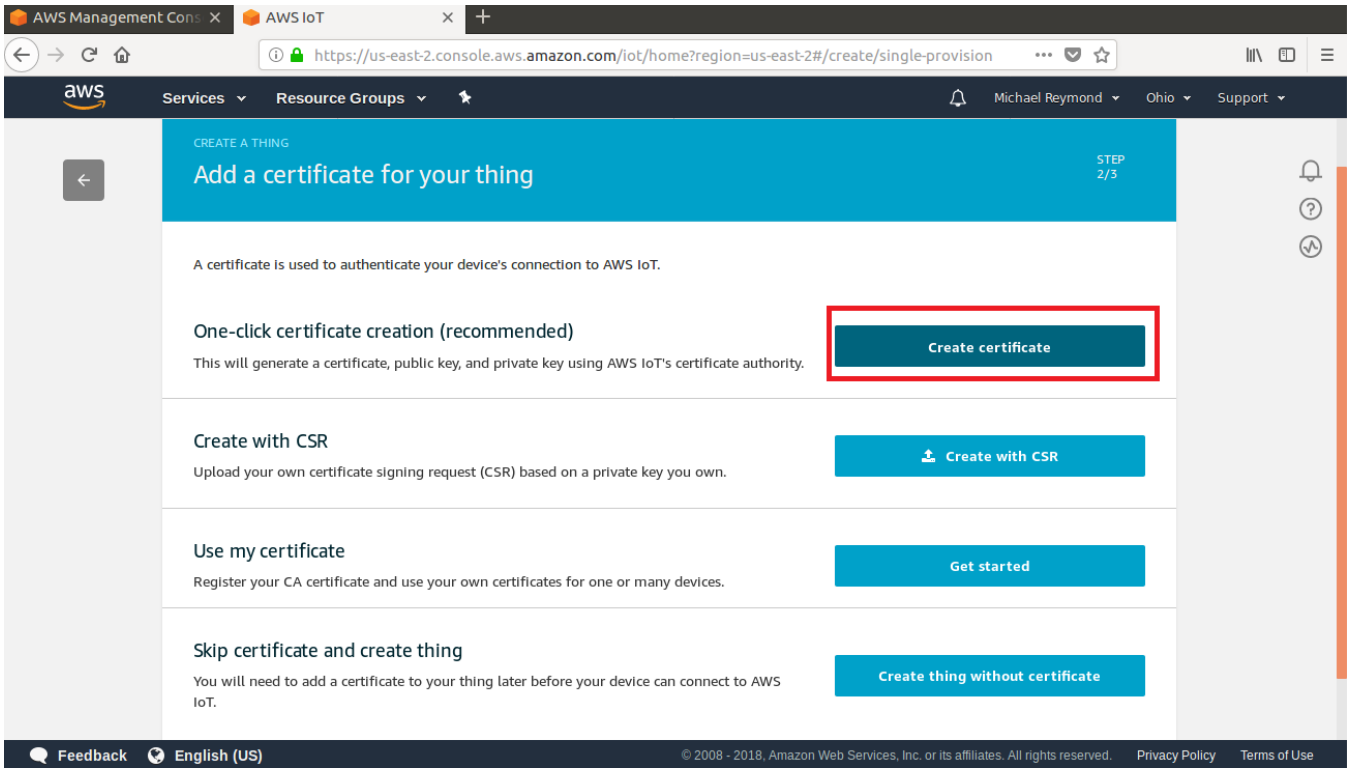
5. Next, click on “Manage” then “Things” to access the things hub. Click on “Show me later” to dismiss the intro message. Then, click on “Register a thing”, then “Create a single thing” to create your thermostat thing, as shown in Figure 4.

Figure 4. Creating an AWS IoT Thing



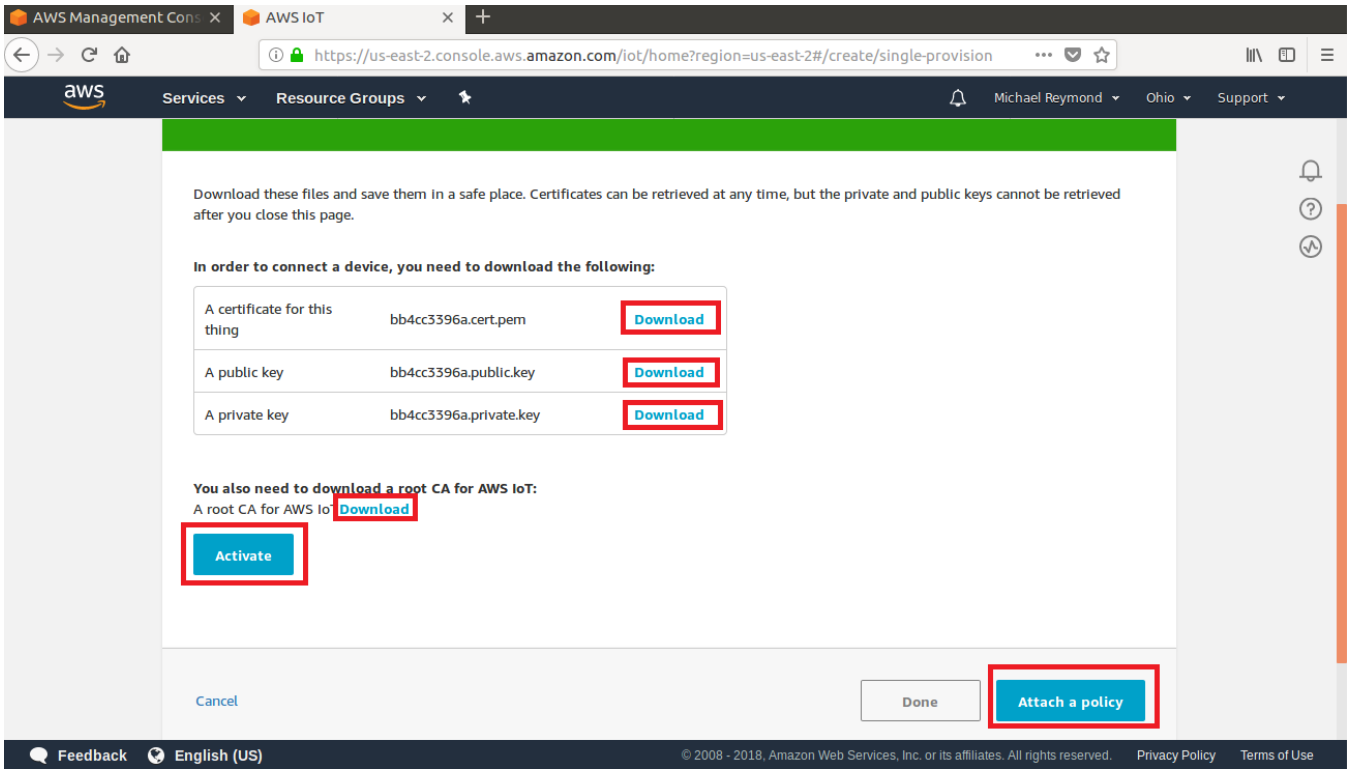
6. In the next menu (Figure 5), type in a name for your thing, and then click on next. Then, click on “Create Certificate” to create the certificate pair to be associated with your thing. Note down the name of your thing, as it will be needed in later setup steps.

Figure 5. Creating a One-Click Certificate



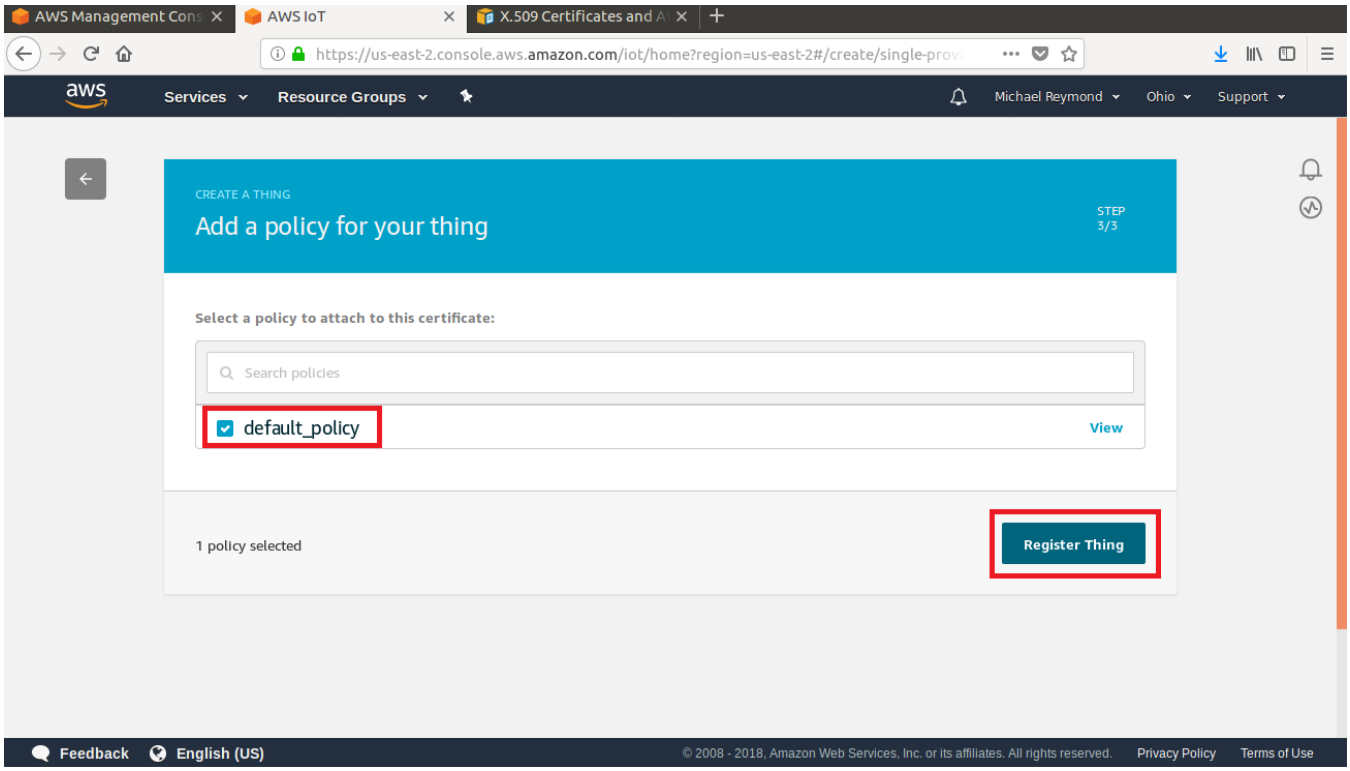
7. In the resulting screen (Figure 6), download the certificate file, the public and private key, as well as the root CA for AWS IoT (the Verisign Class 3 root CA certificate). Then, click on Activate, and then finally on “Attach a Policy”.

Figure 6. Downloading the Thing and Root CA Certificates



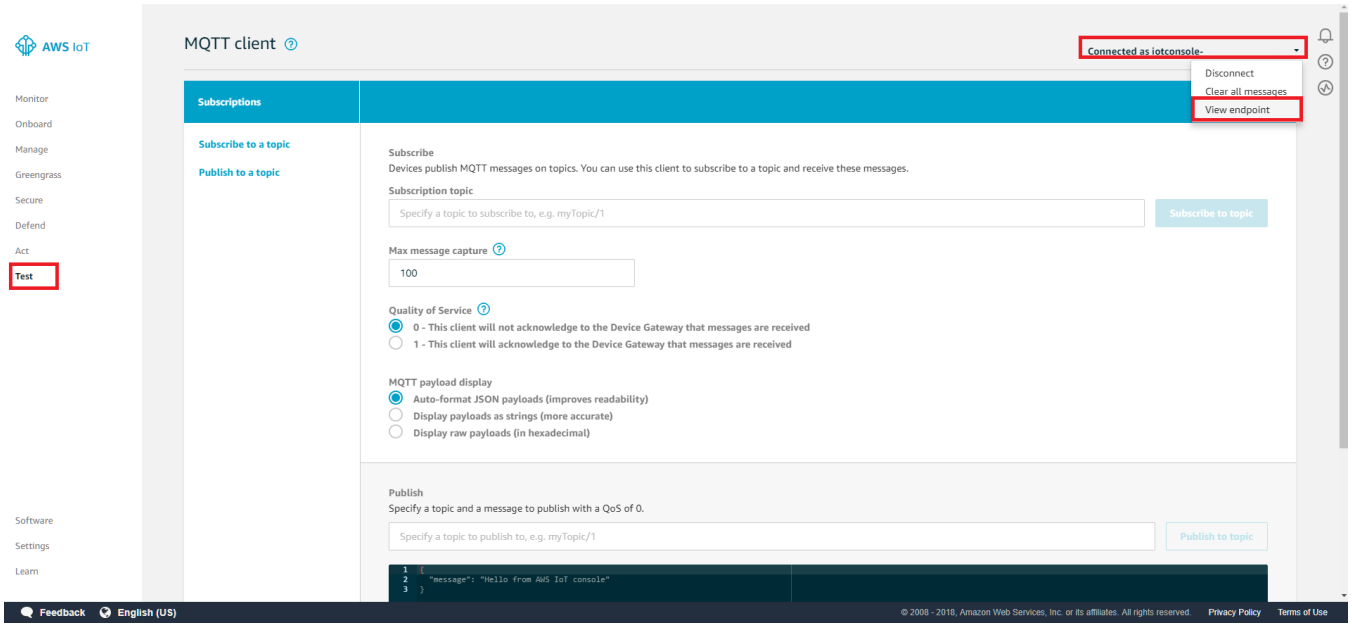
8. In the next screen (Figure 7), select the policy that you just created, and then click “Register Thing”. At this point, your thing should be fully setup on the cloud, ready for use a the thermostat thing on AWS IoT.

Figure 7. Attaching an AWS IoT Policy to the Thing



9. Before leaving AWS IoT, click on “Test”, then after your PC connects to AWS using the in-browser MQTT client (Figure 8), dismiss the green message box in the upper right. Then click on the “Connected as iotconsole-…” message, then on “view endpoint”. In the pane that pops up, copy the URL in the Endpoint text box, and note it for use later.

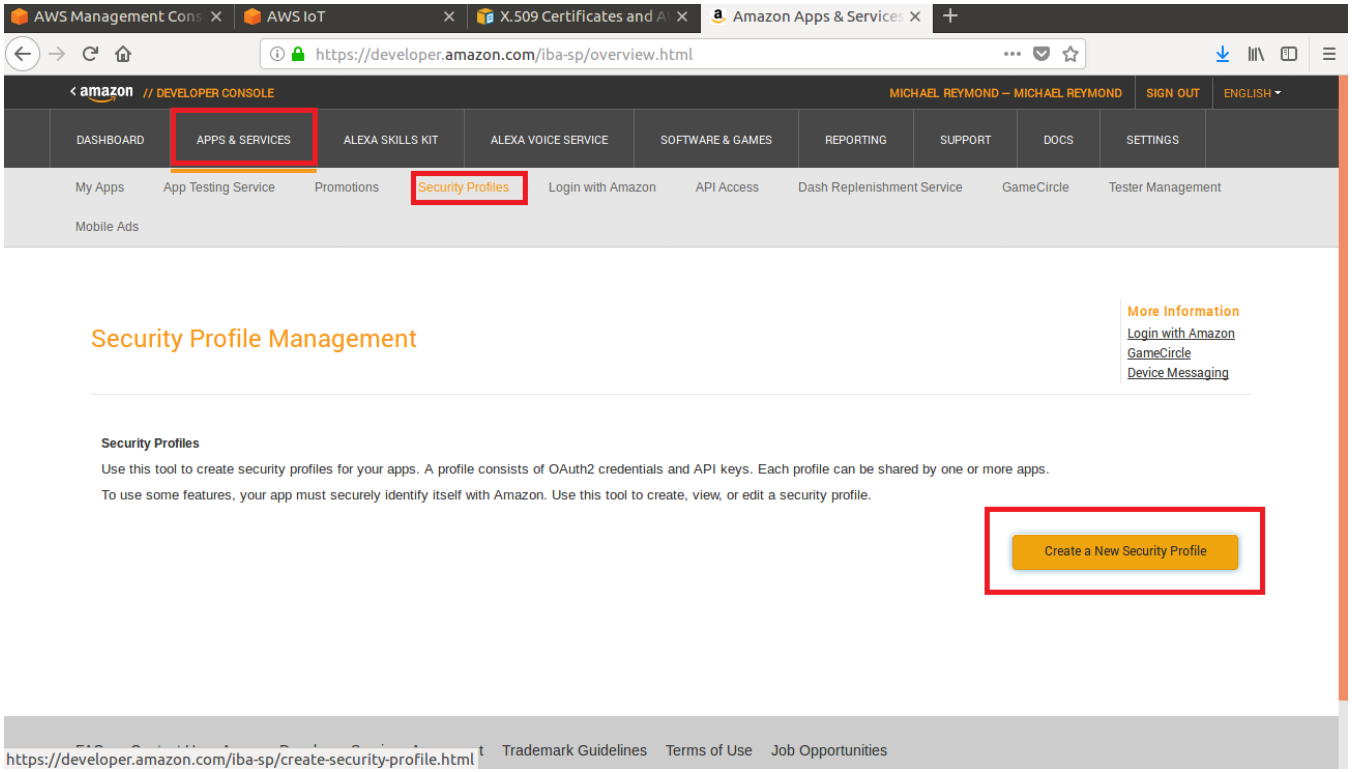
Figure 8. Accessing the Endpoint URL



3.2.2.2 Configuring Alexa Voice Services

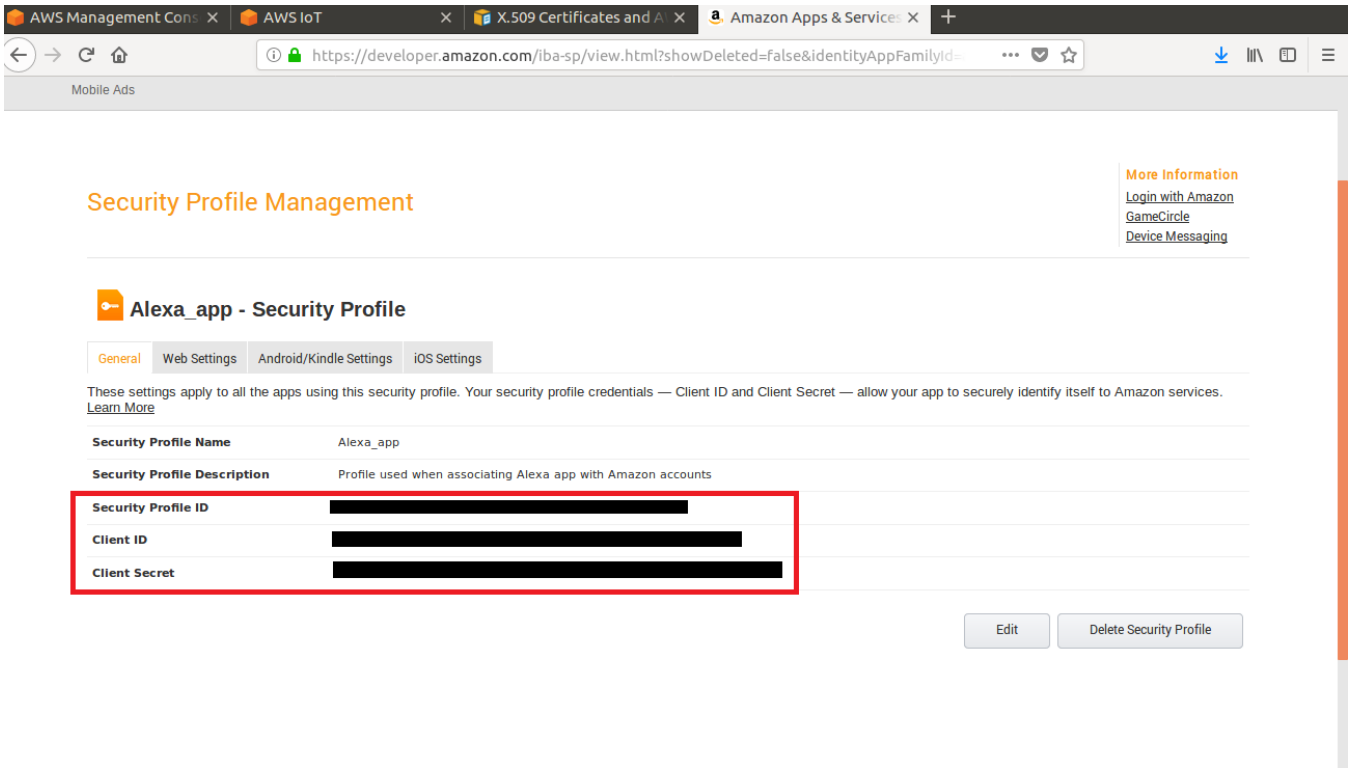
1. First, navigate to developer.amazon.com. If you do not have an account yet, sign up for an account. Once you have logged in, click on "Developer Console" in the upper right.
2. Next, navigate to the Security Profile menu by clicking on “APPS&SERVICES”, then “Security Profiles”. Before we can build and configure the Alexa skill, we need to first setup a security profile for your skill, so that the skill will be able to associate your Amazon account info when invoked. To do this, begin by clicking on “Create a New Security Profile” as shown in Figure 9.

Figure 9. Accessing Amazon Security Profiles



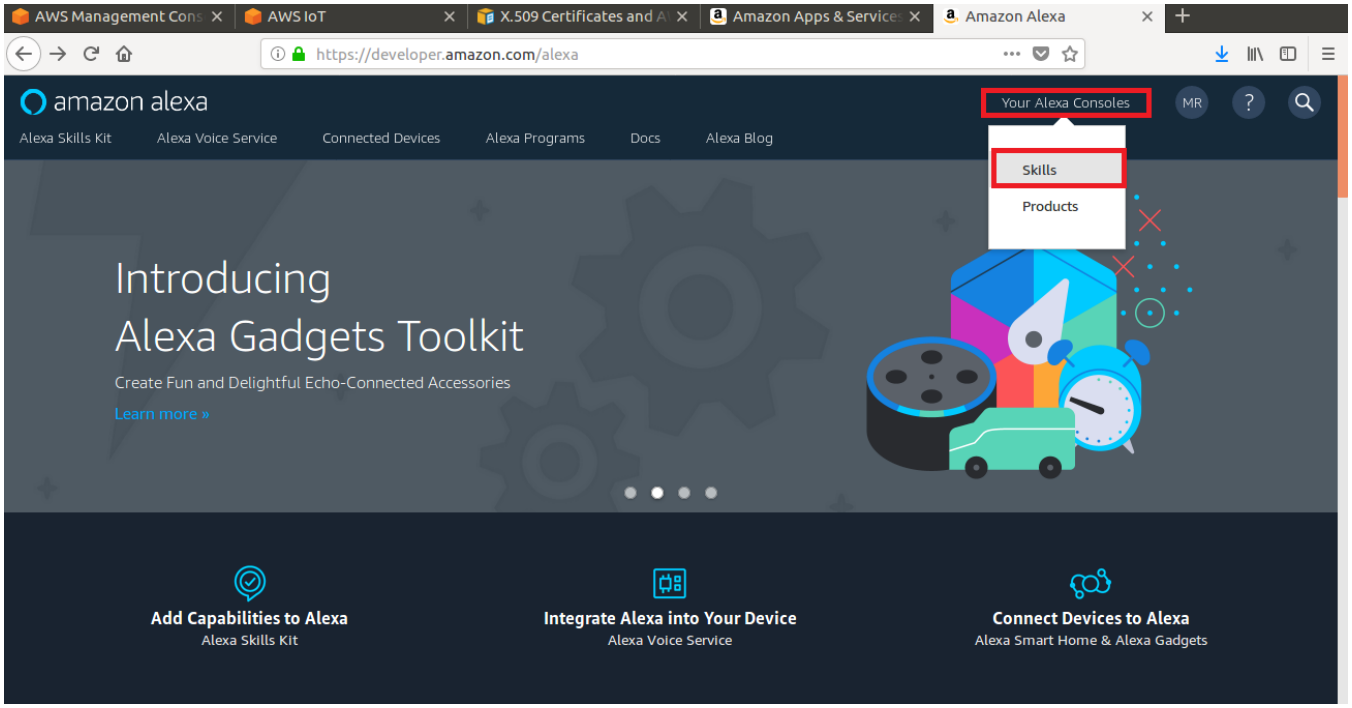
3. In the next screen type in a name and description for your new profile, then click on save.
4. After your security profile is created, copy down the security details, specifically the Security Profile ID, the Client ID, and the Client Secret, as shown in Figure 10. This info will be needed later when creating the Alexa skill. Keep a browser tab on this screen, as we will need to come back and do some more editing as part of the Alexa skill setup.

Figure 10. Security Profile Details



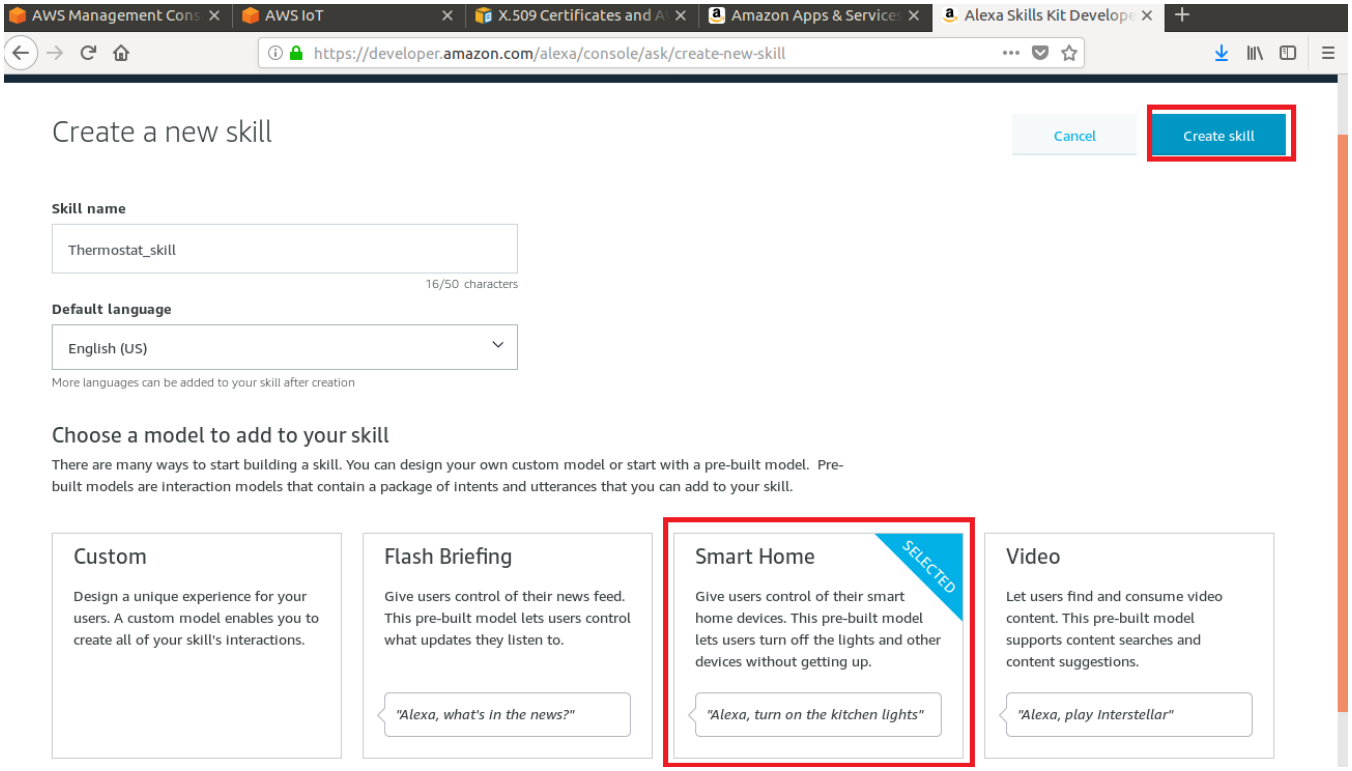
5. Go to developer.amazon.com/alexa. In the resulting screen (Figure 11), go to your Alexa console by hovering over “Your Alexa Consoles” in the upper right, then clicking on “Skills”.

Figure 11. Accessing the Alexa Developer Console



6. Next, click on “Create skill”. After that, type in a name for your skill, and be sure to select the “Smart Home” model before clicking “Create Skill” as shown in Figure 12.

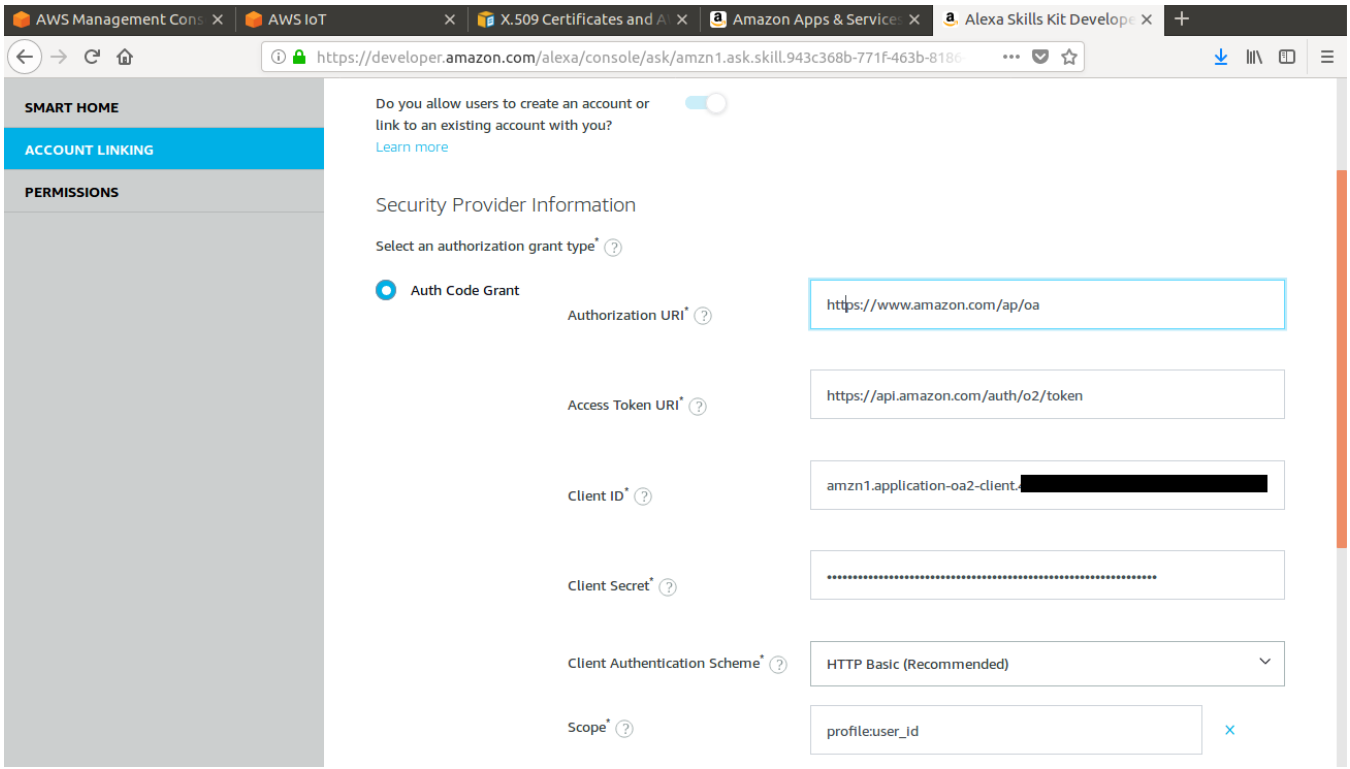
Figure 12. Creating the Alexa Skill



7. In the next screen (Figure 13), click on the “Account Linking” button on the left pane. This is where you configure your skill to connect to Amazon’s authentication service so that it can associate this skill with the correct Amazon account. Type in the following details:
   - Authorization URI: https://www.amazon.com/ap/oa
   - Access Token URI: https://api.amazon.com/auth/o2/token
   - Client ID: <Client ID from step 4>
   - Client Secret: <Client Secret from step 4>
   - Scope: profile:user_id

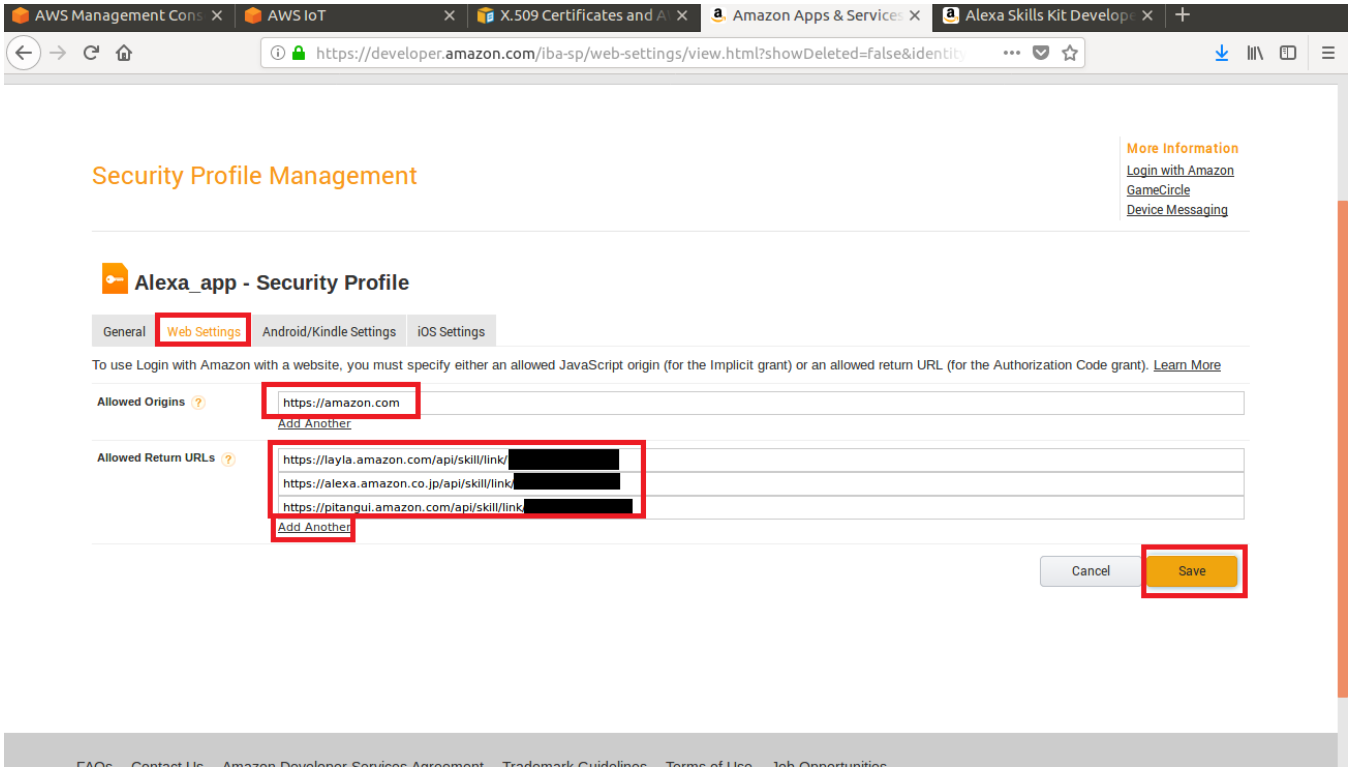
   Leave the rest of the fields as-is. Note that at the bottom of the form are a few redirect URLs. Copy these URLs down, as they will be needed to finish the configuration of the security profile. Then, click “Save” to confirm your account linking settings. Then, click on “Smart Home” on the left pane. You will see a configuration page with sections on Payload version, Smart Home service endpoint, and account linking (which has already been completed). Keep this tab open on this page, as we will need to link the AWS Lambda function to this skill later.

Figure 13. Filling the Account Linking Details



8. Go back to the tab with the security profile that you generated in step 4 (Figure 14). Click on the “Web Settings” tab, click on edit, then in “Allowed Origins” type in https://amazon.com and for the “Allowed Return URLs” paste in each redirect URL that you got in the previous step when setting up account linking for the Alexa Skill. Then click on Save.

Figure 14. Filling the Web Settings for the Security Profile



3.2.2.3 Configuring the AWS Lambda Function

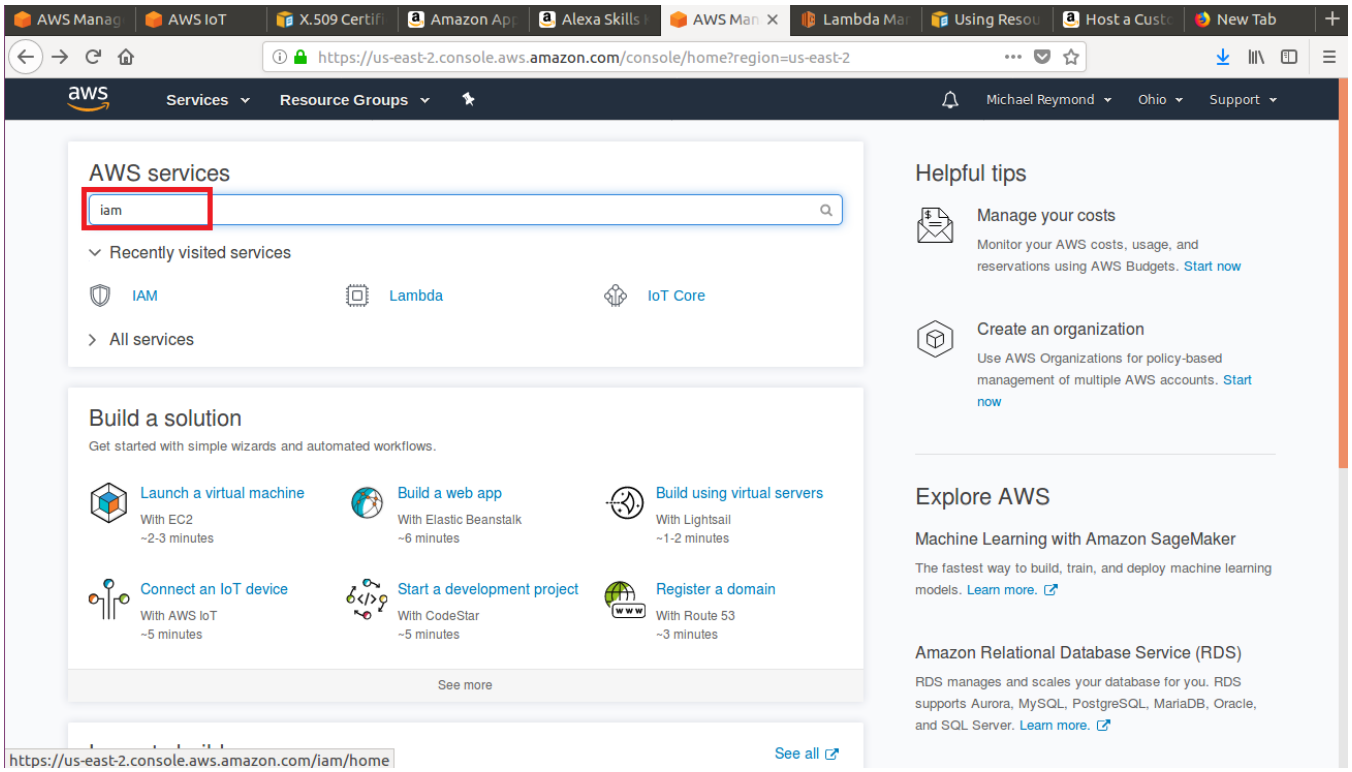
The steps in this section are largely adapted from Amazon's custom skill guide that can be found at:

https://developer.amazon.com/docs/custom-skills/host-a-custom-skill-as-an-aws-lambda-function.html

However, this walkthrough will show you how to interface the AWS IoT Thing to the Alexa Smart Home Skill created in the previous steps.

1. Go to console.aws.amazon.com. Then, type in “iam” in the search box (Figure 15) and click on the IAM service.

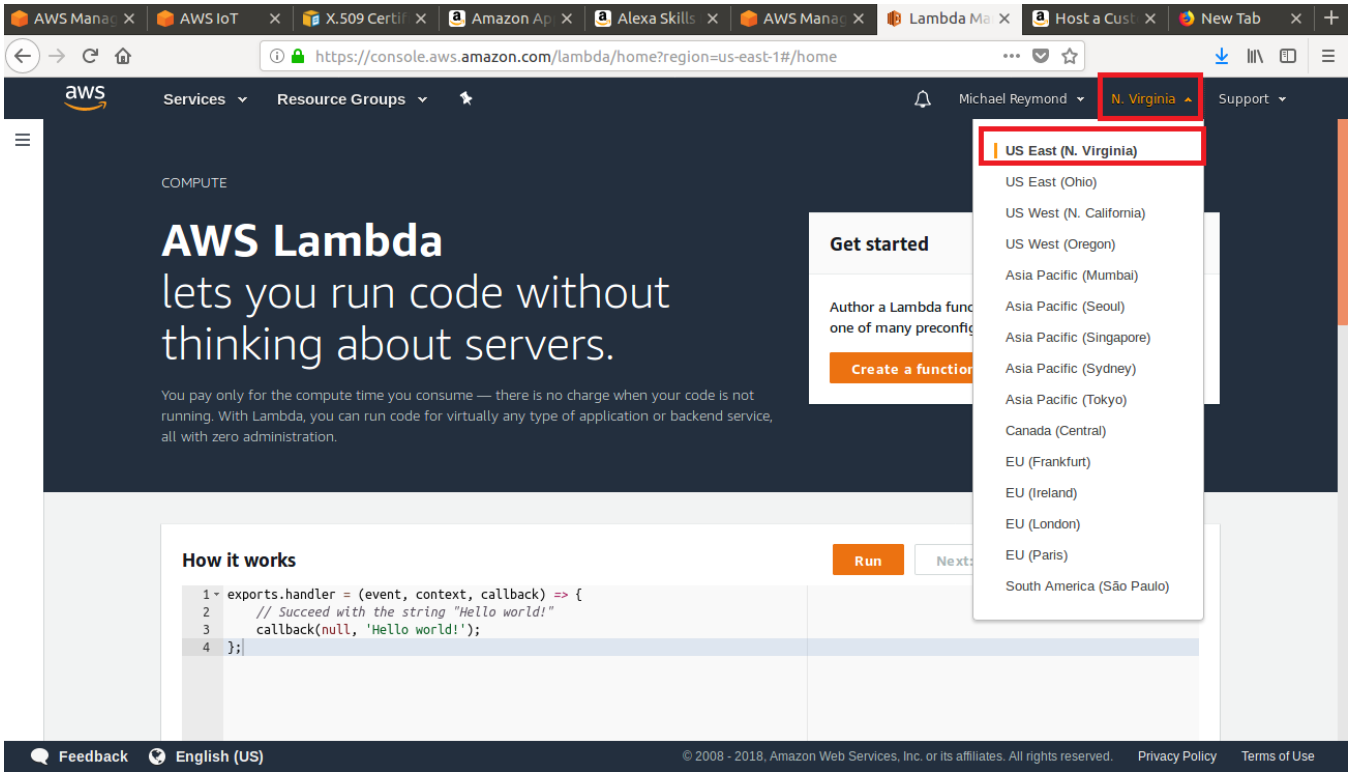
Figure 15. Accessing Amazon IAM



2. Ensure that the AWS region you are currently accessing corresponds to one of the regions that support Alexa Skill integration in AWS Lambda. The current list of supported regions are:
   - Asia Pacific (Tokyo)
   - EU (Ireland)
   - US East (N. Virginia)
   - US West (Oregon)

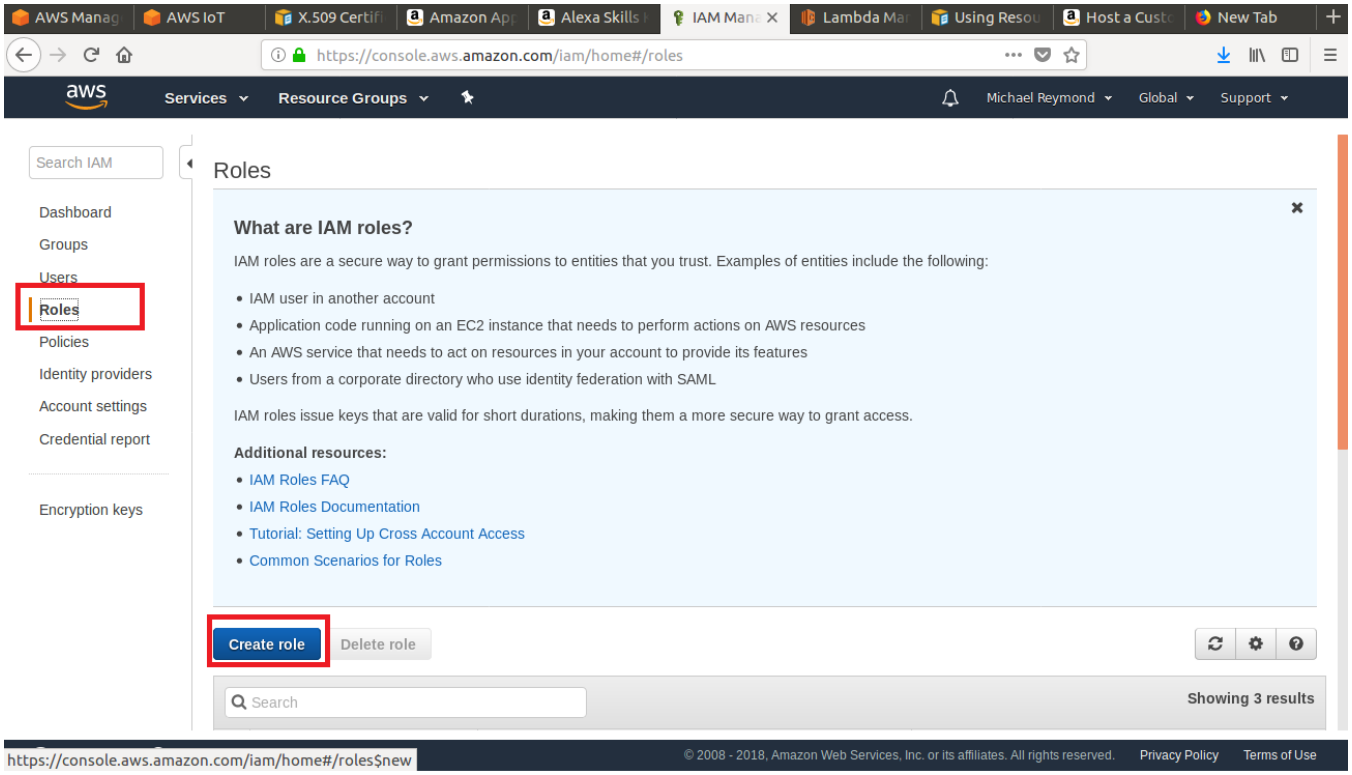
   Your region is displayed in the upper right of the dashboard (Figure 16). If your account is not set to one of the supported regions, you must change to one of the supported region by clicking on your current region, then selecting one of the regions above in the drop-down menu.

Figure 16. Selecting an AWS Region



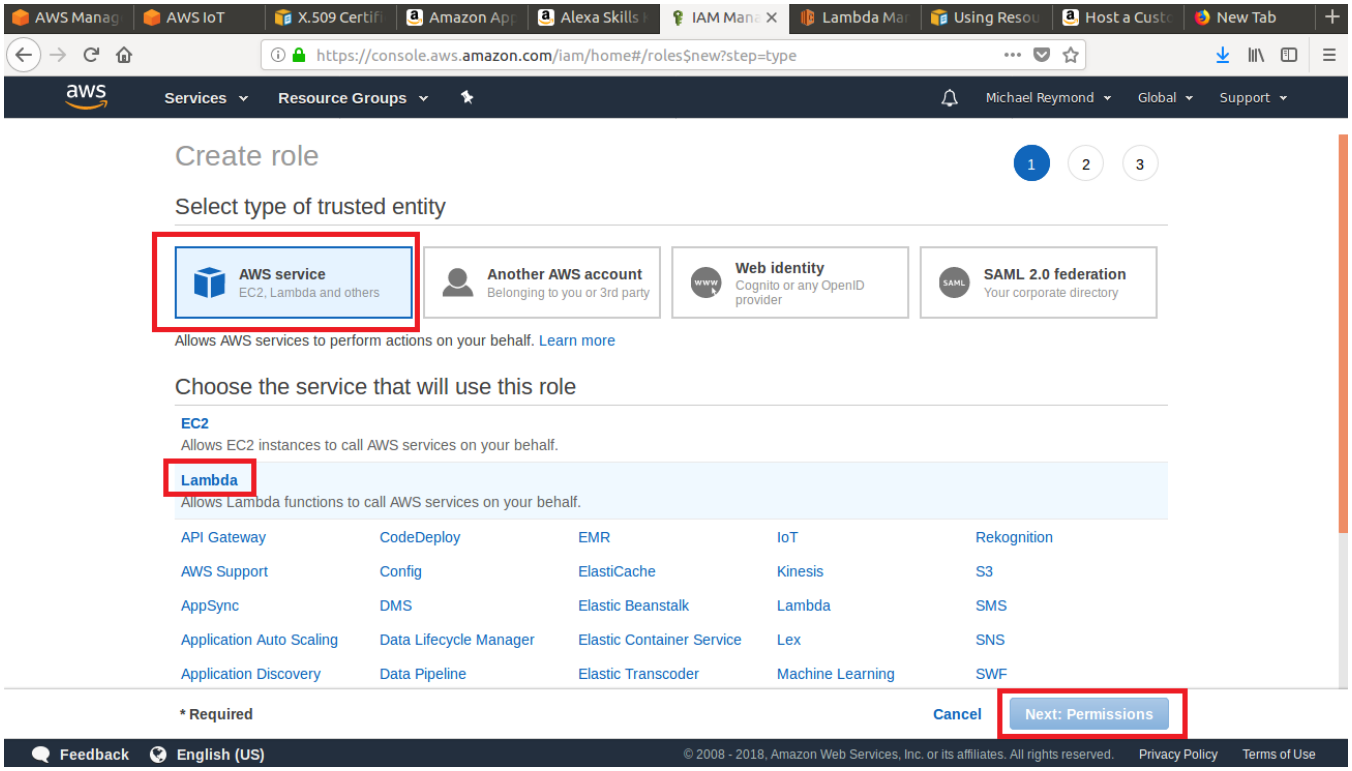
3. Then, click on “Roles” in the left pane (Figure 17), and then click on “Create Role”.

Figure 17. Creating an IAM Role



4. In the next menu (Figure 18), select “AWS service” as the type of trusted entity, then select “Lambda” and click on the Next:Permissions button.

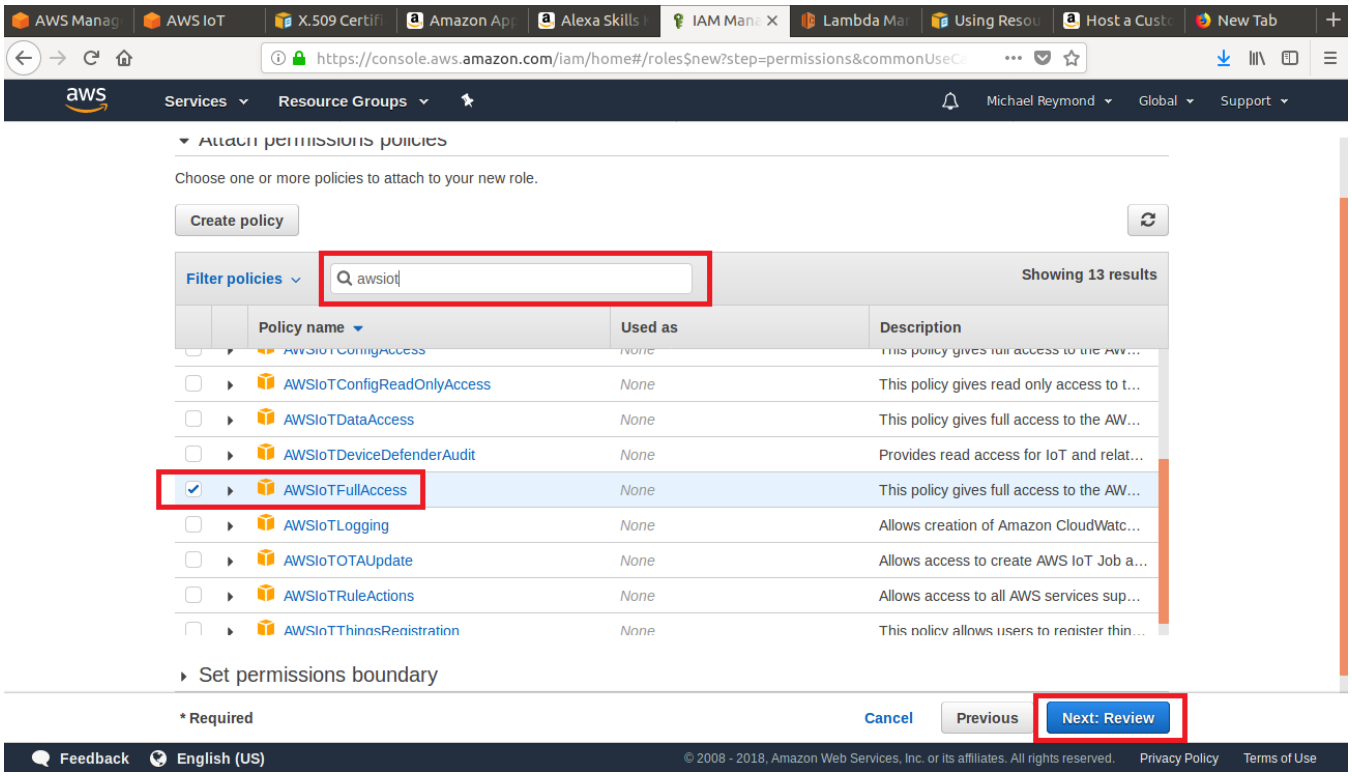
Figure 18. Selecting AWS Service as a Trusted Entity



5. Next, you will need to add the correct permissions to your IAM role (Figure 19). The policies that are needed are the following:
   - AWSIoTFullAccess
   - CloudWatchFullAccess
   - CloudWatchLogsFullAccess

   To add policies, enter the policy names into the search box, and then tick the policy in the resulting policy list to add it to the role.

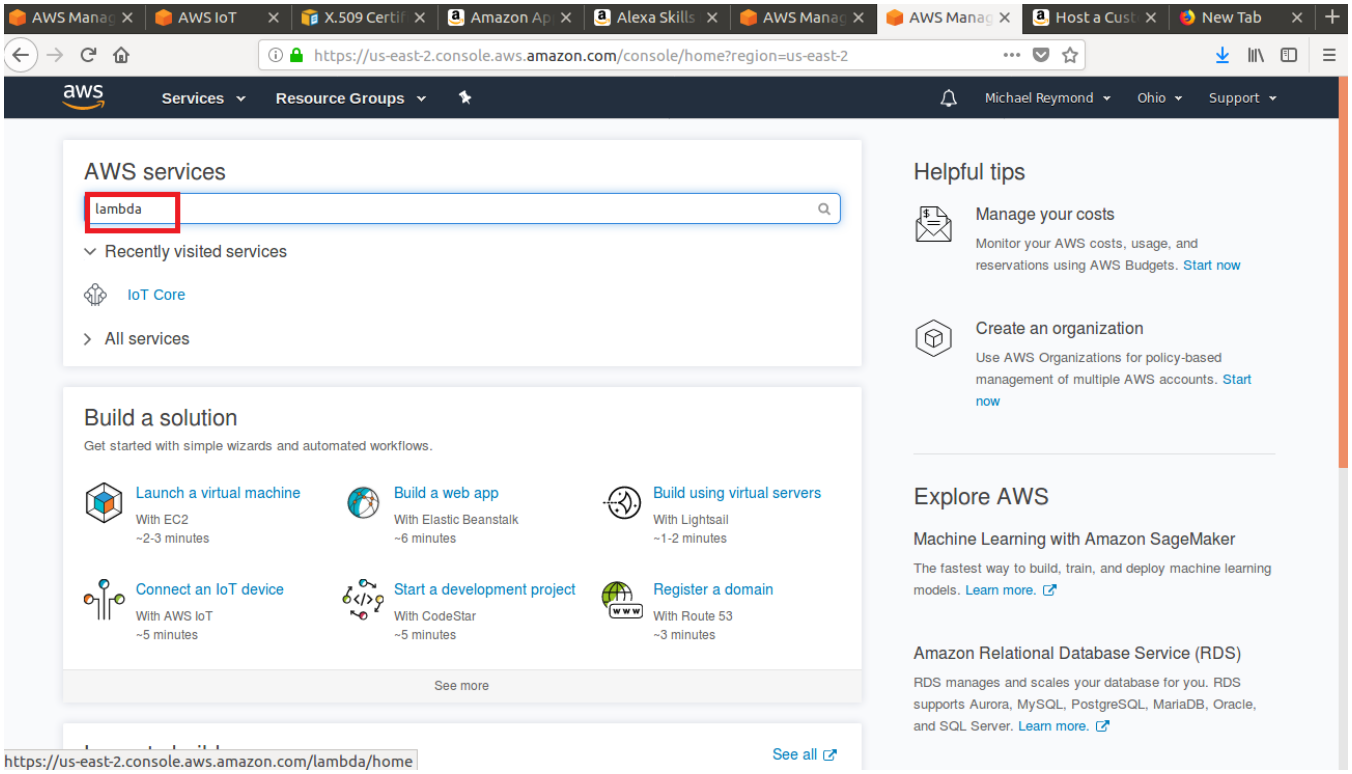
Figure 19. Assigning Policies to the IAM Role



6. Once all of the needed policies have been added, click on “Next:Review” in the lower right to continue. Double-check to see that the needed policies have been added, and then type in a name for your role and click “Create role”.

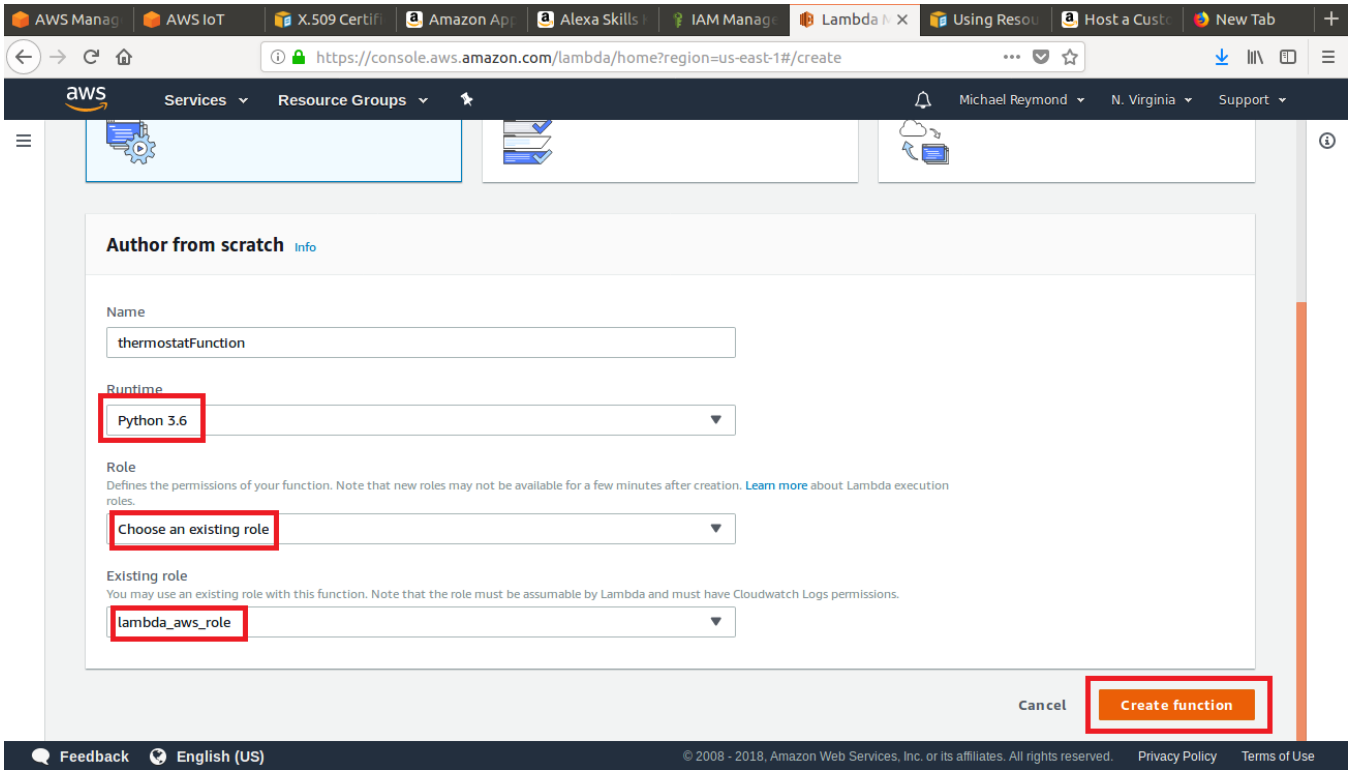
7. Go to console.aws.amazon.com. Then, type in “lambda” in the search box (Figure 20) and click on the Lambda service to access the Lambda dashboard.

Figure 20. Accessing AWS Lambda



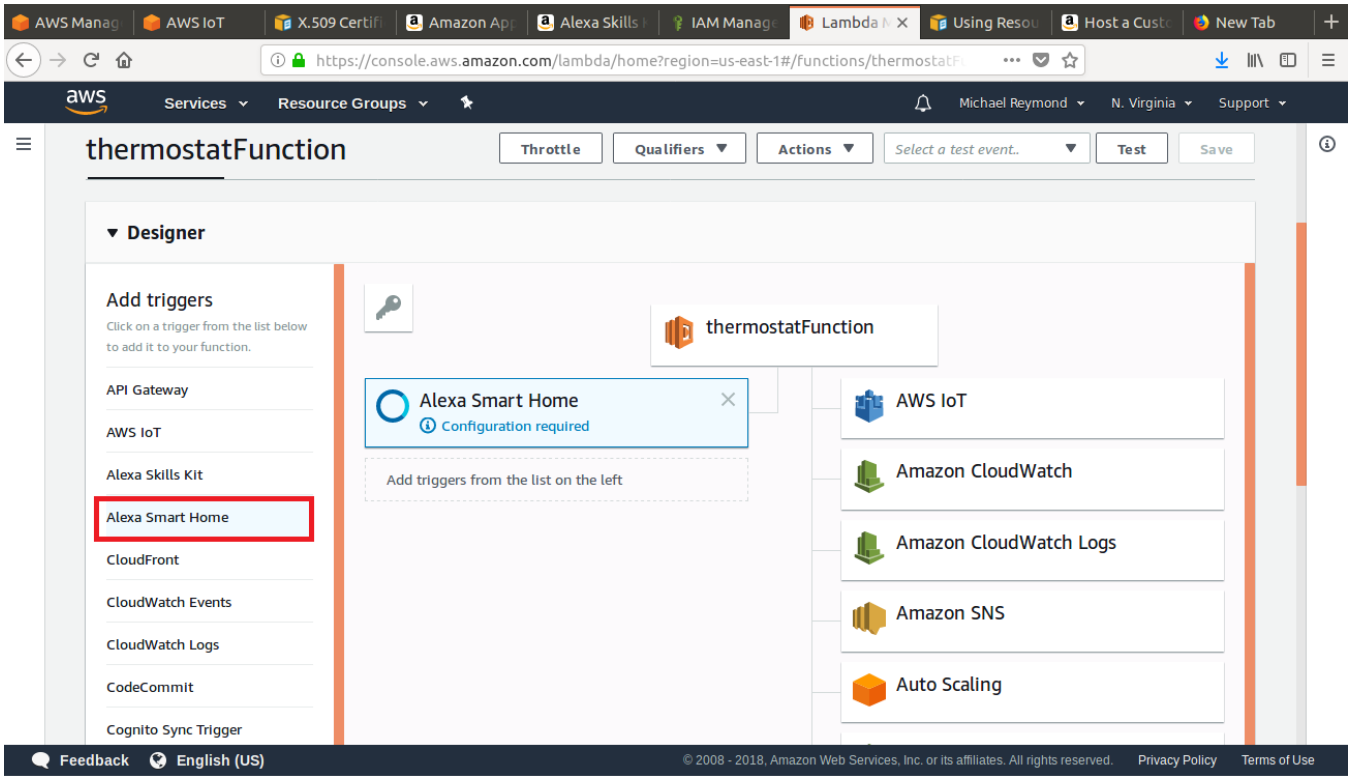
8. Next, click on “Create function”. In the following prompts (Figure 21), select “Author from scratch”, and then input the following for the data fields:
   - Name: <your choice>
   - Runtime: Python 3.6
   - Role: Choose an existing role
   - Role name: <the role you made in step 5>

Figure 21. Creating the AWS Lambda Function



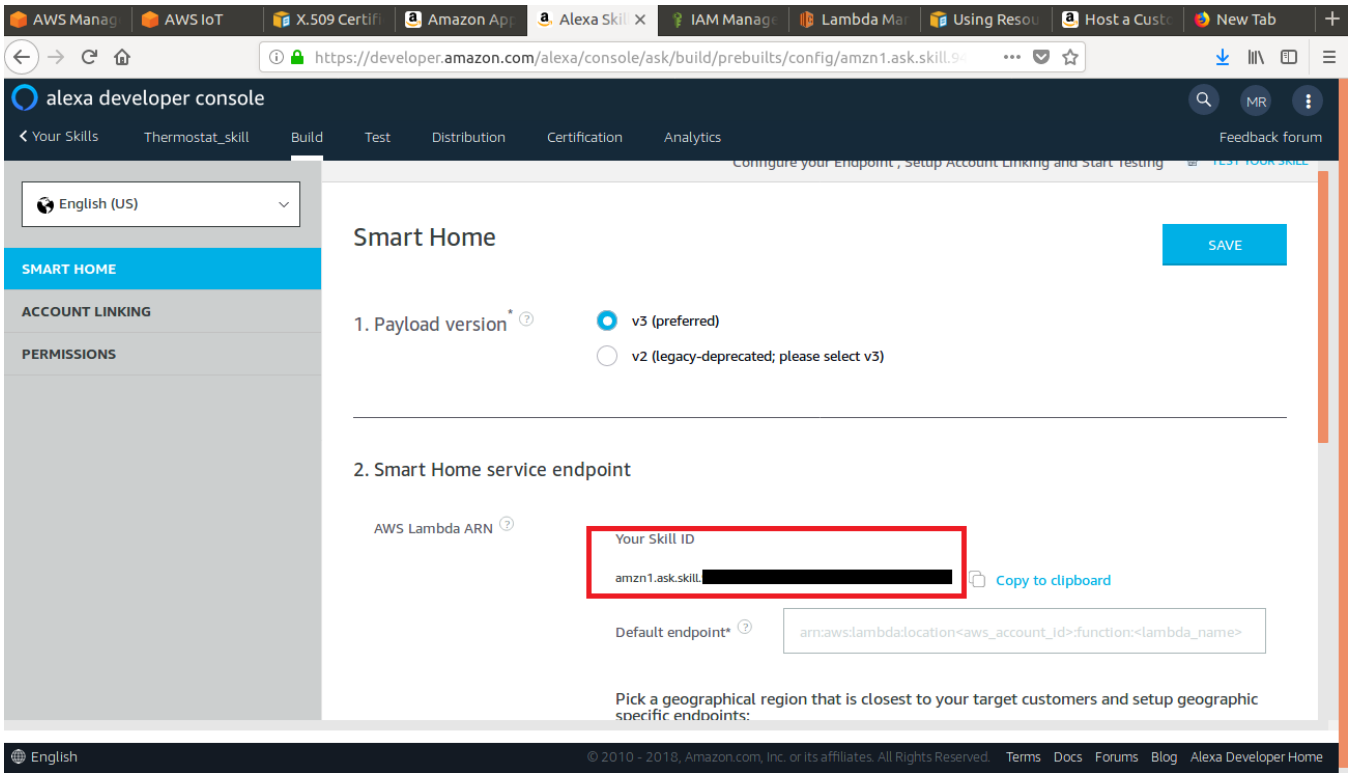
9. After creating the Lambda function, you will be brought to the function's configuration page (Figure 22). We will need to first add the Alexa integration. To do this, click on "Alexa Smart Home" on the left pane to add it to the function as a trigger.

Figure 22. Adding Alexa Home Integration to Lambda Function



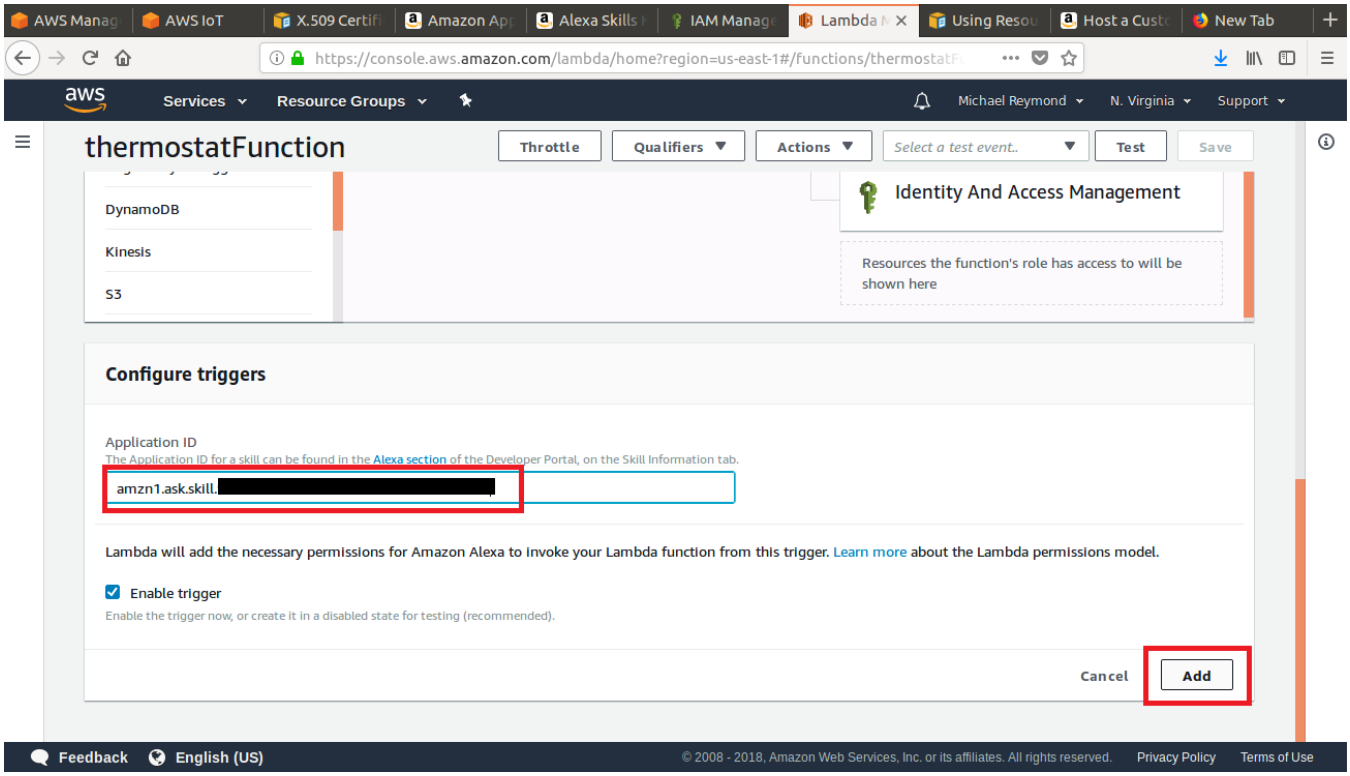
10. The Configure triggers menu for the Alexa Smart Home trigger will appear automatically, with the "Application ID" field blank. In the Alexa Smart Home skill configuration page that was accessed in Section 3.2.2.2, step 7, copy the skill ID (Figure 23).

Figure 23. Copying the Skill ID



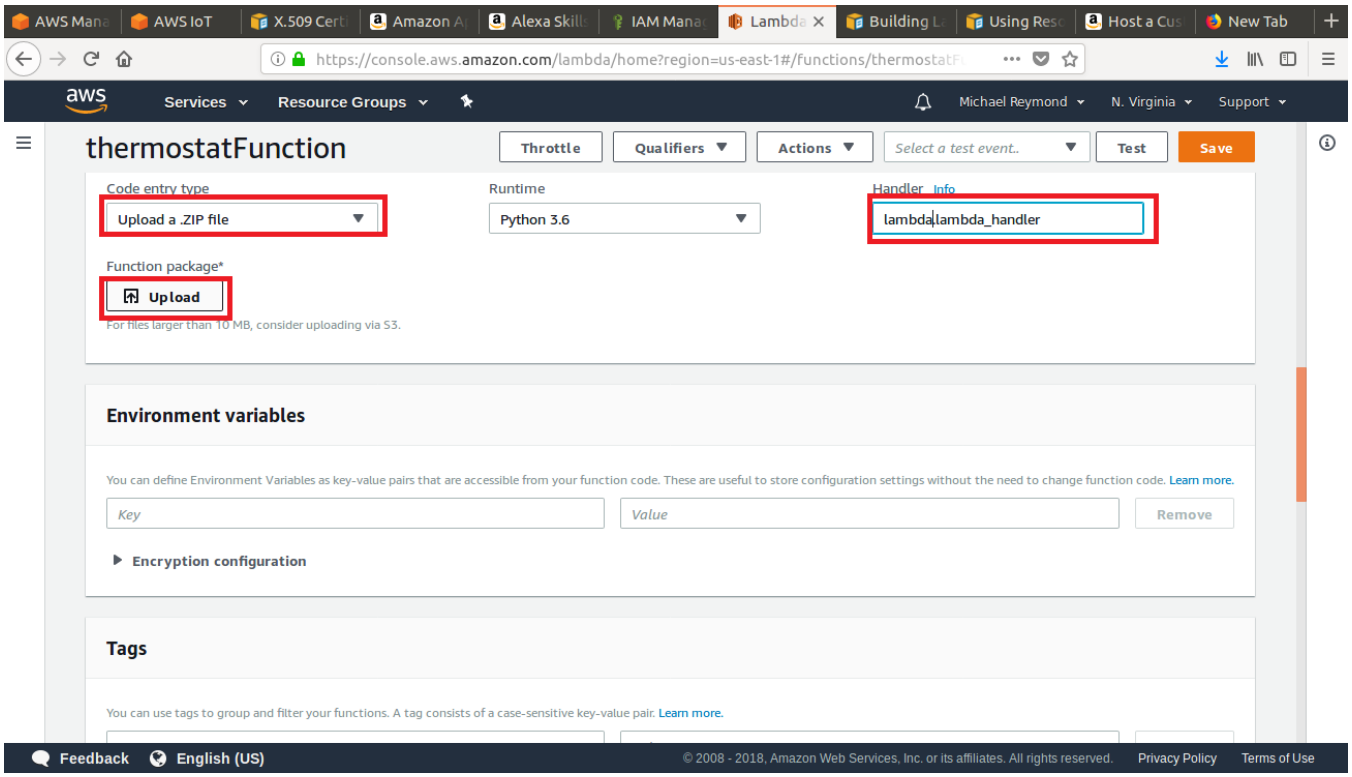
11. Then, paste the skill ID into the Application ID field in the Lambda configuration (Figure 24). Click on "Add" to finish configuring the trigger.

Figure 24. Pasting the Skill ID into Lambda Function



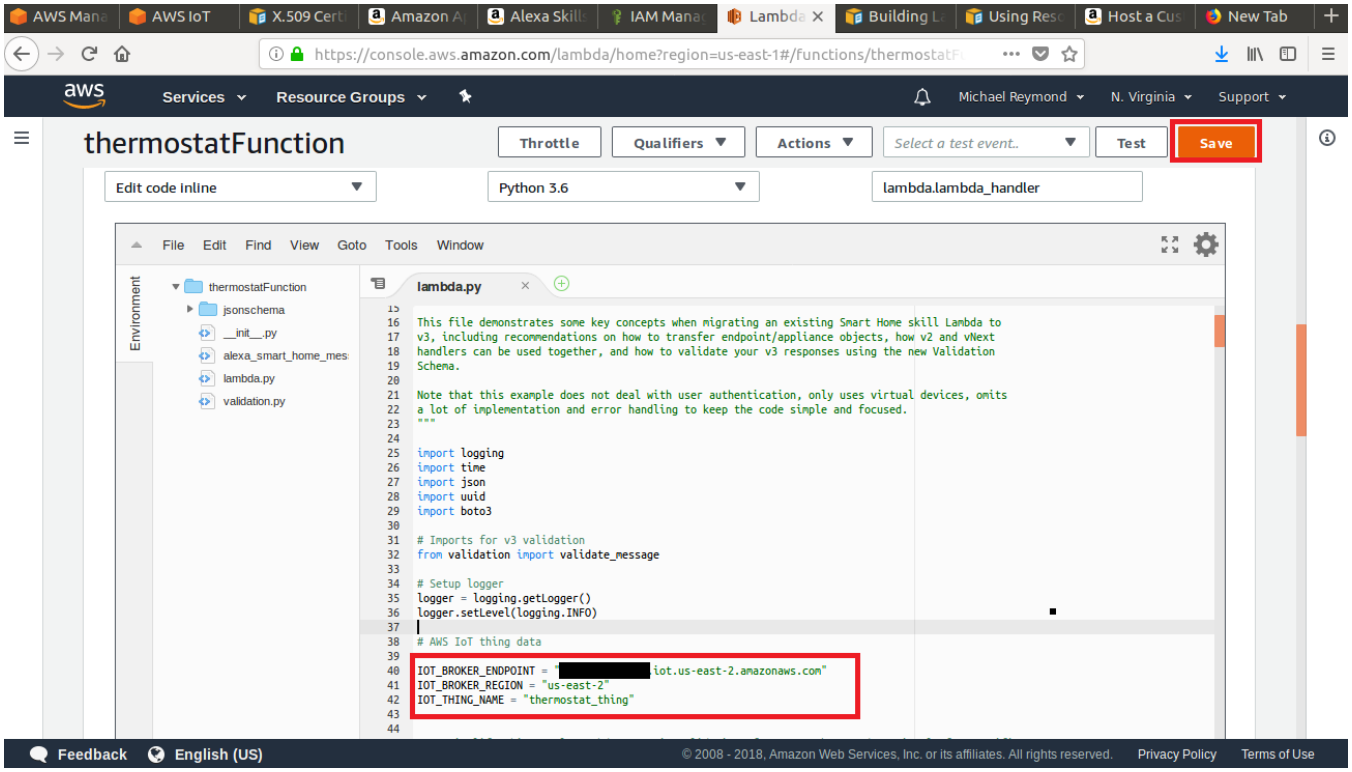
12. Next, we need to configure what the function actually does once it is invoked by the Alexa Smart Home skill. Click on the main Lambda function block labeled with the name of your function (Figure 25), and you should see the full function configuration appear below. In the Function code section, change the Handler to “lambda.lambda_handler”. Then, click on the Code entry type drop-down menu, and select “Upload a .ZIP file”. Navigate to the thermostatControl.zip file provided in the code distribution, then click on the Save button on the upper right. This should upload the zip file to AWS Lambda and extract it as for code execution.

Figure 25. Uploading the Lambda Code



13. Once the code is done uploading, open lambda.py in the editor (Figure 26). In order for the skill to properly link to the AWS thing created, you will need to modify some AWS IoT defines. They are IOT_BROKER_ENDPOINT, IOT_BROKER_REGION, and IOT_THING_NAME. The endpoint URL is the URL obtained from the AWS IoT test console in Section 3.2.2.1, step 9. The IOT_BROKER_REGION is a string describing the region that was used when creating the thermostat thing and can be copied from the broker endpoint URL. Specifically, broker endpoints are formatted as "xxxxx.iot.<region string>.amazonaws.com", and the <region string> part of the URL should be copied directly into IOT_BROKER_REGION. Finally, IOT_THING_NAME should be the name of the thing that was created in Section 3.2.2.1. Once the necessary changes are made, click on “Save” in the upper right.

Figure 26. Editing the IoT Details



14. Finally, copy the ARN of the Lambda function displayed on the upper right (Figure 27), above the "Save" button, and paste that into the "Default endpoint" section of the Alexa Smart Home service endpoint configuration (Figure 28). Then click "Save" on the Smart Home configuration.

Figure 27. Copying the ARN from Lambda

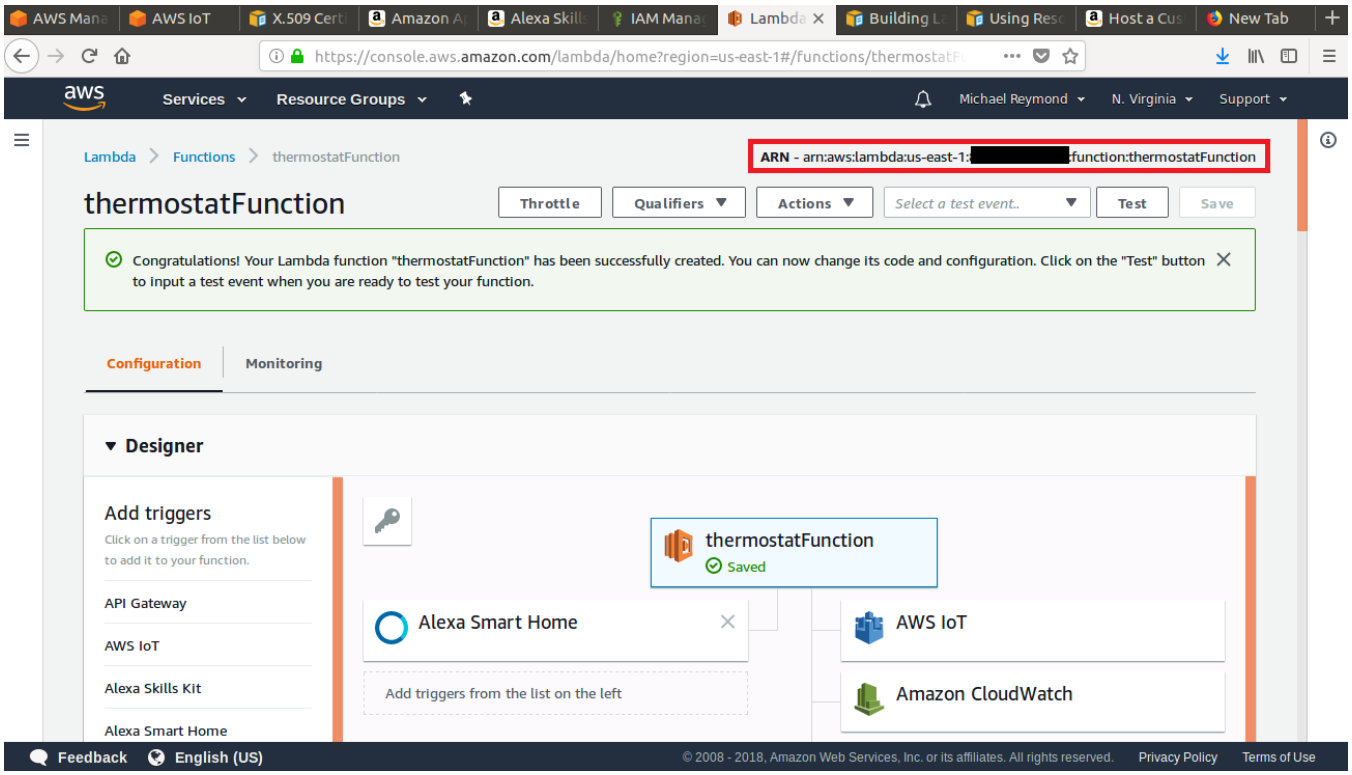
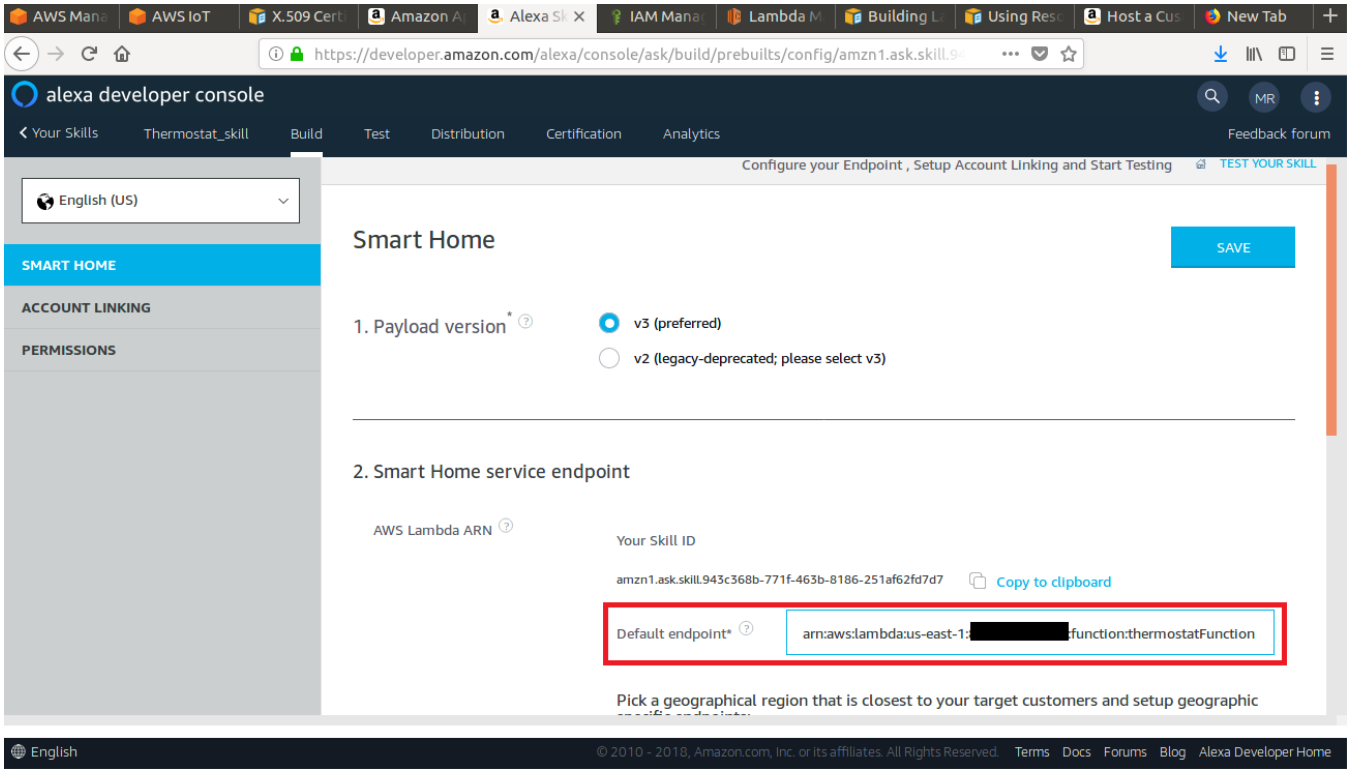


Figure 28. Pasting the ARN into Alexa



15. At this point, the cloud side Alexa Smart Home skill setup is complete. However, we still need to enable the skill for your Amazon account. This will require you to download the Amazon Alexa app from the appstore/play store. Download the Alexa app from your platform's store.
16. Once you have downloaded the Alexa app, open it and sign in with your Amazon developer account. Then, open the menu, and tap on “Skills & Games” to access the skills store. At the store (Figure 29), click on the “Your skills” link in the upper right, then tap on “Dev Skill” in the center. You should see your thermostat skill that you created here.Tap on that to select it, and then hit “Enable”.
17. You will be redirected to an Amazon authentication page, login with the same account you have been using in Amazon Developer and provide its password. You should see a successfully linked message, similar to the one in Figure 30, if you have setup the account linking of your skill correctly. Close the page by pressing on the x in the upper left. And tap on discover device in the prompt that appears. Alexa should discover the thermostat you have setup, if the Lambda skill has been configured correctly. With account linking and device discovery complete, the skill has been enabled and you can speak directly into the app to invoke your skill as you would any Alexa request.

Figure 29. Selecting the Alexa Smart Home Skill

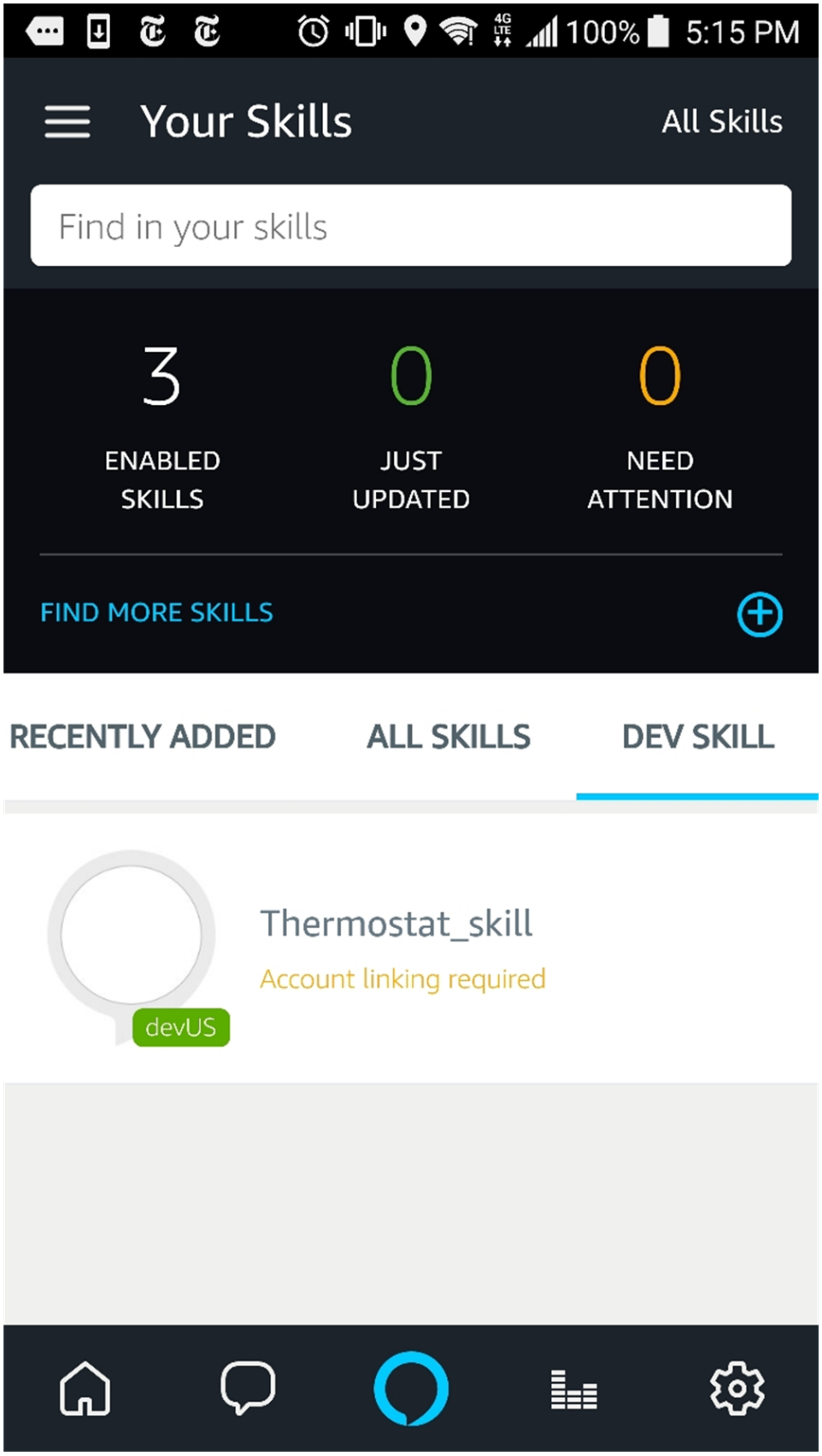
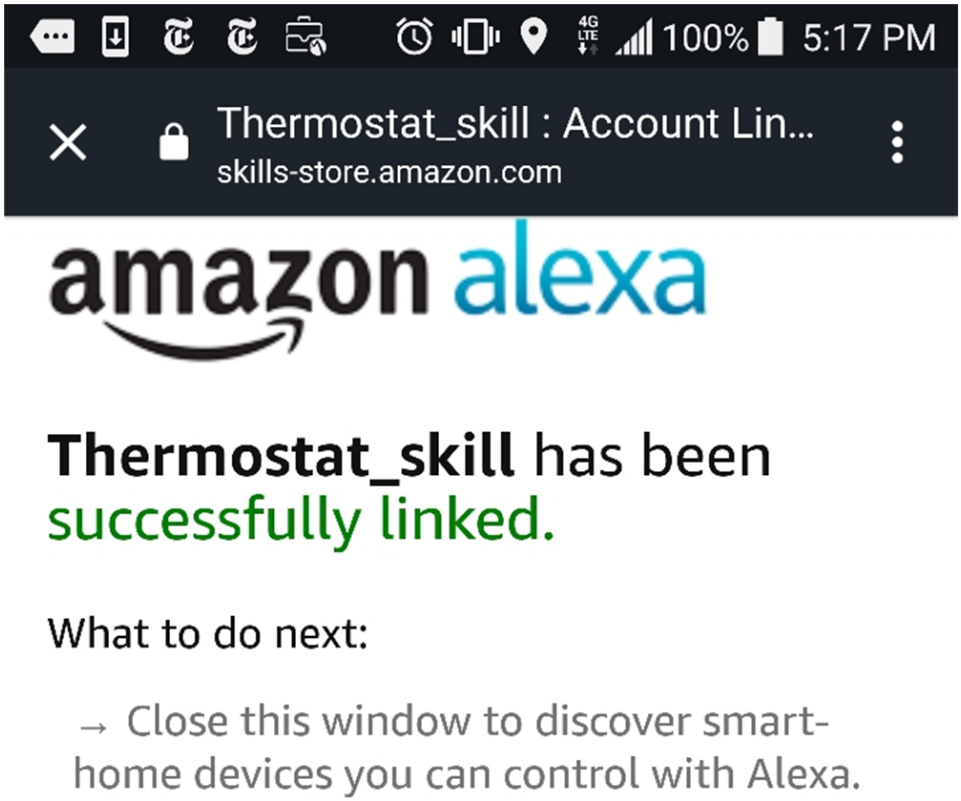
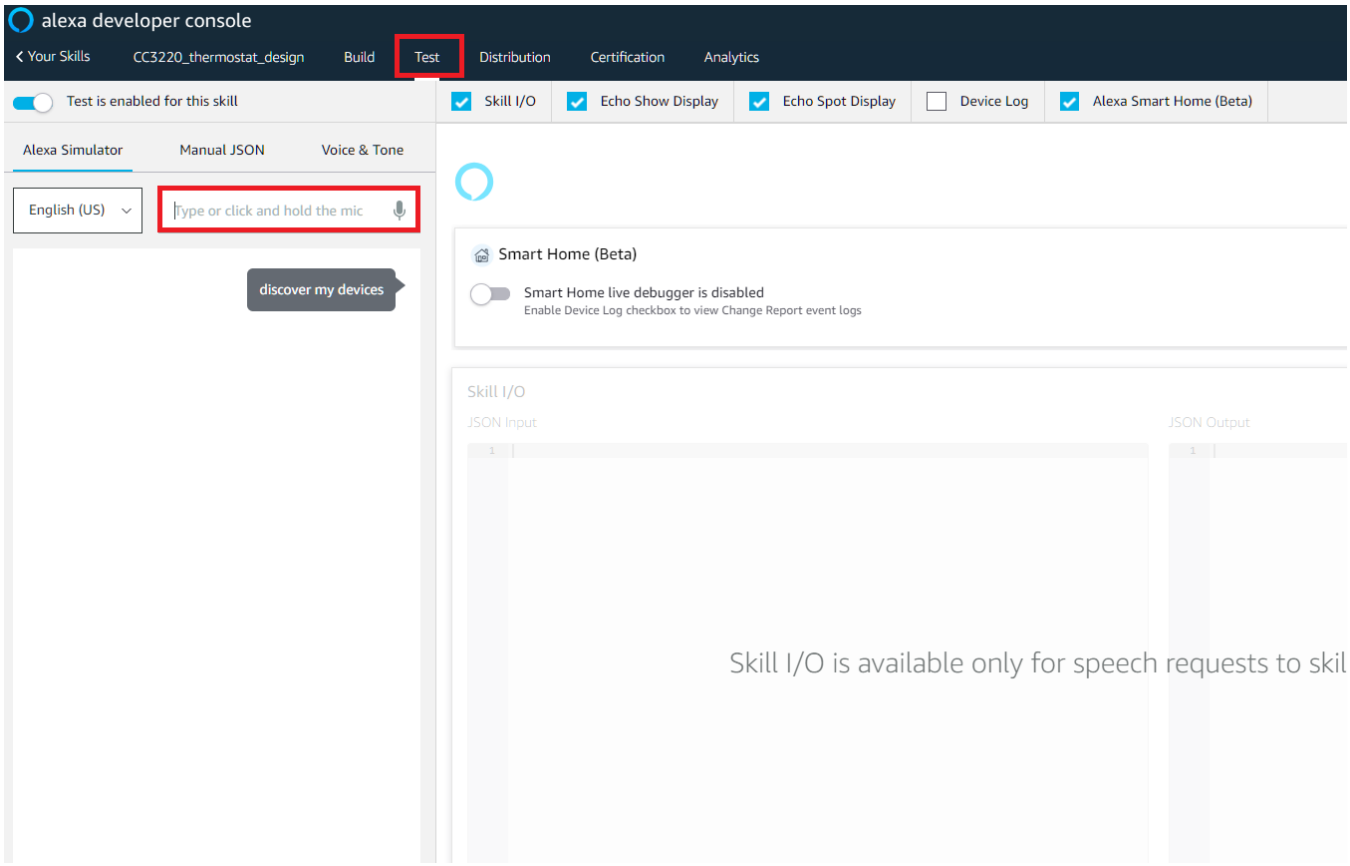


Figure 30. Successful Account Linking Result Screen



18. If device discovery fails in the app, try performing discovery through the Alexa test console. To access the console, go to the configuration page of your Alexa skill. This is the page that you had open in Section 3.2.2.2, step 7. Next, click on 'Test' in the upper menu bar (Figure 31). You should see a console-like screen. This console can be used to test any Alexa interaction, including the thermostat skill that we just created. To initiate device discovery, type in "discover my devices" and then press Enter. You should have the thermostat device appear in your Alexa app linked to your account within a minute.

Figure 31. Discovering Devices Using the Alexa Developer Console



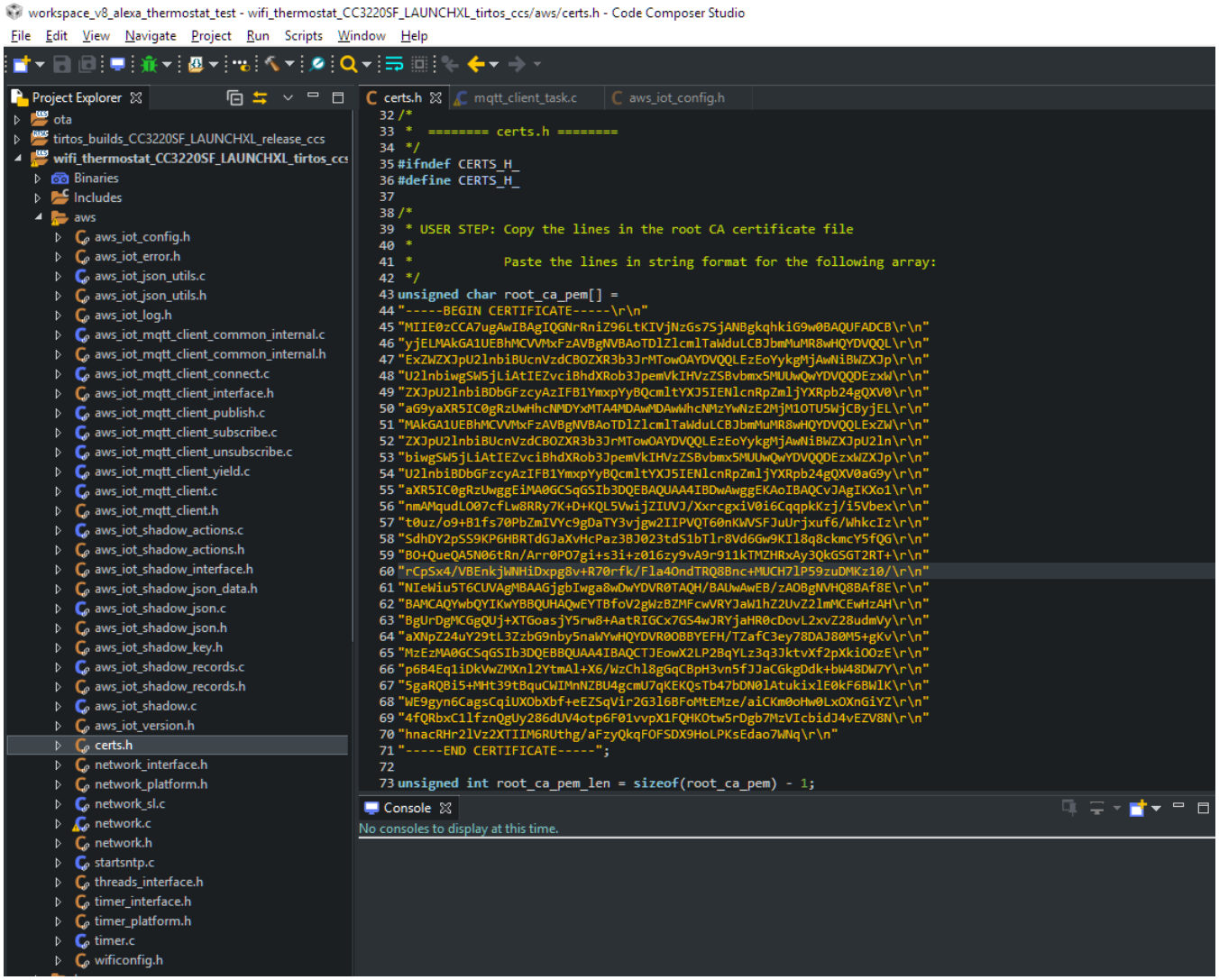
3.2.3 Setting Up Cloud Application

The cloud application setup is not applicable with the AWS + Alexa configuration option, as the cloud-based control is demonstrated through the use of voice commands through Alexa Voice Services.

3.2.4 Configuring Wi-Fi® Thermostat Project

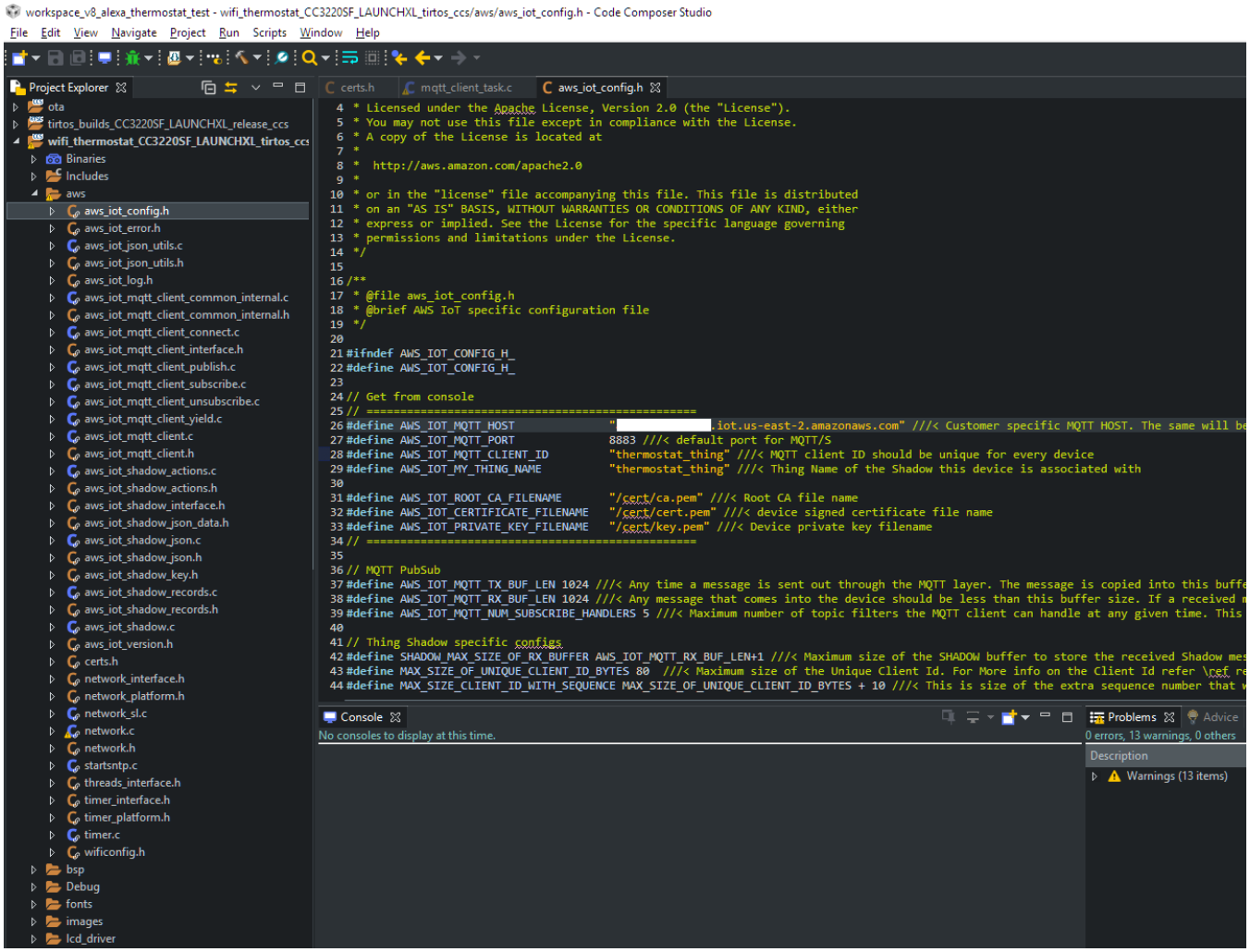
1. Open CCS and select Project → Import CCS Projects.
2. Browse to the *wifi_thermostat* folder in the design software and select the project (if this returns an error, ensure the SAIL, AWS IoT, and BLE plugins are installed).
3. In wifi_thermostat_app.h, change DEMO_CITY and DEMO_TIME_ZONE to the desired city and time zone for weather forecast and clock display. The city must be a string containing "City, Country" (do not include a U.S. state in this string). For the full list of supported time-zone definitions in the CC3220 SDK utilities library, see source/ti/net/utils/clock_sync.h.
4. Next, take the certificates linked to the AWS thing created in section 3.2.2.1 and copy them into aws/certs.h. The files needed are client certificate (xxxxxx-certificate.pem.crt), the client private key (xxxxxx-private.pem.key), as well as the AWS IoT root CA certificate. To do this, open the certificate files in a text editor such as notepad++. Then, copy each line starting from "-----BEGIN CERTIFICATE-----" with a “\r\n” break at the end of each line, except for the last line with "-----END CERTIFICATE-----". As an example, the resulting character array for the AWS IoT root CA certificate should look as shown in Figure 32.

Figure 32. Copying the Root CA Certificate



5. Open aws/aws_iot_config.h. The AWS_IOT_MQTT_HOST needs to be defined as the endpoint as provided in the AWS thing test menu, obtained in Section 3.2.2.1, step 9. The AWS_IOT_MY_THING_NAME needs to be defined as the Thing name created in Section 3.2.2.1. Once those edits are complete, the aws/aws_iot_config.h file should look like Figure 33.

Figure 33. Editing the AWS IoT Config Files



6. The design by default is configured for BLE Provisioning. Users can recompile the application to use AP Provisioning or SmartConfig by removing the BLE module. This can be done by changing the BLE_PROVISIONING define in wifi_thermostat_app.h to 0. If AP Provisioning is used, the dummy-rootca-cert-key (/simplelink_cc32xx_sdk_2_10_00_04/tools/cc32xx_tools/certificateplayground/dummy-root-ca-cert-key) must also be added as a user file to the serial flash. For detailed instructions on adding a user file to the SimpleLink™ Wi-Fi® external serial flash, see the UniFlash ImageCreator Basics SimpleLink Academy module.
7. To demonstrate OTA functionality, import and configure the OTA library from the CC3220 SDK. For detailed instructions on preparing and loading an OTA image, see the Wi-Fi OTA SimpleLink Academy Lab.
8. Rebuild the Wi-Fi® thermostat project. Run a debug session in Code Composer Studio™ or load the MCU image to the serial flash with UniFlash ImageCreator. For detailed instructions on programming an image to the SimpleLink™ Wi-Fi® external serial flash, see the UniFlash ImageCreator Basics SimpleLink Academy module.

3.2.5 Build simple_np Application and Flash CC2640R2F

This section is omitted as the content of this section is the same as that of the SimpleLink™ CC3220 Wireless MCU-Based Thermostat Reference Design with BLE Provisioning design guide.

3.2.6 Running Wi-Fi® Thermostat Demo

1. The first screen on start-up is the calibration screen. Carefully tap the boxes onscreen to calibrate the resistive-touch functionality.
2. When calibrated, the application stays on the calibration data page until the CC3220 device is connected. If the device cannot connect to a known profile before the timeout, the device enables AP and SmartConfig Provisioning. For detailed instructions on how to use the Wi-Fi® Starter Pro mobile app to provision the CC3220, see the SimpleLink™ SDK Explorer section of the BLE Wi-Fi® Provisioning README.
   - If the application has been configured for AP Provisioning and SmartConfig, this is enabled instead of BLE provisioning. For detailed instructions on how to use the Wi-Fi® Stater Pro mobile app to provision the CC3220 device, see the Wi-Fi Provisioning SimpleLink Academy Lab.
3. After successfully provisioning, the main thermostat screen appears. From here, you can interact with the thermostat locally through use of the touchscreen, or through the use of Alexa commands. These commands can be invoked through the phone app, or through an Amazon Echo or other Alexa-enabled device. A non-exhaustive list of commands include:
   - *"Set the thermostat to 20 degrees"*
   - *"Make the thermostat cooler by 2 degrees"*
   - *"What is the thermostat temperature?"*

3.2.7 Implementation

3.2.7.1 Program Flow

This section is omitted as the content of this section is the same as that of the SimpleLink™ CC3220 Wireless MCU-Based Thermostat Reference Design with BLE Provisioning design guide.

3.2.7.2 Provisioning Device

This section is omitted as the content of this section is the same as that of the SimpleLink™ CC3220 Wireless MCU-Based Thermostat Reference Design with BLE Provisioning design guide.

3.2.7.3 Secure OTA

OTA shares *ProvOtaTask* and *gProvOtaTransitionTable* with the provisioning implementation. When the provisioning or network connection is successful, *ProvOtaTask* waits in a pending-OTA command state for a trigger from the user cloud application through the MQTT broker.

The OTA implementation for this design is based on the Cloud OTA example from the CC3220 SDK. For further details on the SDK example or the OTA library, see the Wi-Fi OTA SimpleLink Academy Lab. One key difference between the instructions described in the SimpleLink Academy lab and the Alexa thermostat code is that to trigger the OTA update, the *OTA* JSON variable should be set to 1 on the device shadow through the AWS IoT cloud interface.

3.2.7.4 Secure Cloud Connectivity

This design leverages the AWS IoT SDK from the SimpleLink SDK Plugin for Amazon Web Services to communicate with the cloud. Within the SDK, this demo uses the AWS IoT Shadow APIs over MQTT for cloud connectivity. In *mqtt_client_task.c*, the CC3220 connects securely to the AWS IoT platform with TLS and certificate authentication. The demo connects to the AWS IoT Device Shadow service and syncs the *thermostatMode*, *targetSetpoint*, *temperature*, *pressure*, *airQuality*, *humidity*, *setPoint*, and *otaUpdate* JSON variables. The AWS IoT client publishes an update to the device shadow stored on the cloud whenever one of these values changes. If the MCU is in LPDS, the cloud application must send an update to wake the MCU to collect new data. Otherwise, the thermostat is configured to wake up the MCU periodically, to read sensor data and sync updates before returning to LPDS. This interval is defined by *SENSOR_SLEEP_MS* and is set to thirty minutes by default. The thermostat also monitors the device shadow for changes made to it on the cloud side, and updates the thermostat locally to match the desired shadow state to match the cloud.

3.2.7.5 Sensor Interface and Measurements

This section is omitted as the content of this section is the same as that of the SimpleLink™ CC3220 Wireless MCU-Based Thermostat Reference Design with BLE Provisioning design guide.

3.2.7.6 Power Management

This section is omitted as the content of this section is the same as that of the SimpleLink™ CC3220 Wireless MCU-Based Thermostat Reference Design with BLE Provisioning design guide.

3.3 Average Current Measurement for Thermostat Use Cases for TIDM-1020

This section is omitted as the content of this section is the same as that of the SimpleLink™ CC3220 Wireless MCU-Based Thermostat Reference Design with BLE Provisioning design guide.

4 Design Files

The TIDM-1020 is based on existing LaunchPads and BoosterPacks. To download the schematics, bill of materials, PCB layout recommendations, Gerber files, assembly drawings, and software files, see the design files.

- SimpleLink™ Wi-Fi® CC3220SF Wireless Microcontroller LaunchPad™ Development Kit
- Sensors BoosterPack Plug-In Module
- Kentec QVGA Display BoosterPack
- Grove Base BoosterPack for Texas Instruments LaunchPad
- Grove – PIR Motion Sensor
- Grove – Air quality sensor v1.3
- Grove – SPDT Relay

5 Related Documentation

5.1 Product Pages

- SimpleLink™ Wi-Fi® Main Page
- CC3220 SimpleLink Product Page
- TI E2E Support Community
- SimpleLink™ MCU Platform

5.2 Application Notes

- Designing Thermostats With CC3220 SimpleLink™ Single-Chip Wi-Fi® MCU System-on-Chip
- CC3120, CC3220 SimpleLink™ Wi-Fi® Internet-on-a chip™ Solution Device Provisioning
- SimpleLink™ CC3120, CC3220 Wi-Fi® Internet-on-a chip™ Solution Built-In Security Features
- SimpleLink™ CC3120, CC3220 Wi-Fi® Internet-on-a chip™ Networking Subsystem Power Management
- CC3x20 SimpleLink™ Wi-Fi® and Internet-of-Things Over the Air Update

5.3 Software

- SimpleLink™ CC3220 SDK
- TI Resource Explorer
- SimpleLink Academy: Provisioning
- SimpleLink Academy: Cloud OTA
- SimpleLink Academy: MQTT Client Server
- SDK Code Example: Sensor Interface

5.4 Blogs

- What Are You Sensing? Pros and Cons of Four Temperature Sensor Types
- Strengthening Wi-Fi Security at the Hardware Level

5.5 Videos

Thermostat Video

5.6 Trademarks

E2E, SimpleLink, MSP432, SmartConfig, Code Composer Studio, Internet-on-a chip are trademarks of Texas Instruments.
Arm is a registered trademark of Arm Limited.
Bluetooth is a registered trademark of Bluetooth SIG Inc..
Android is a trademark of Google, LLC.
Wi-Fi is a registered trademark of Wi-Fi Alliance.
All other trademarks are the property of their respective owners.

6 Terminology

OTA— Over the Air update: update device firmware over a wireless connection

IDE— Integrated development environment: software tool for firmware development

HMI— Human machine interface

TCP— Transmission control protocol

CDN— Content delivery network

AP— Access point

PIR— Passive infrared

SPI— Serial peripheral interface

I^2C— Inter-integrated circuit

ADC— Analog-to-digital converter

7 About the Authors

MICHAEL REYMOND is an applications engineer at Texas Instruments, where he is responsible for supporting customers designing Wi-Fi®-enabled systems. Michael earned his Bachelors of Science in Computer Engineering from the Georgia Institute of Technology.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265