MASTERs 2018

Lab Manual for 22073 SEC4

Zero Touch Secure Provisioning Kit for AWS IoT

Contents

Introduction

This manual gives a detailed walkthrough of the Zero-Touch Provisioning Kit for AWS IoT used as the hands-on portion of the MASTERs class.

Outside of the classroom, please check the appendix for the software and setup steps that this manual assumes. The classroom computers should already have all the required software installed. Additionally, temporary AWS account credentials will be distributed for use during the class.

AWS Account Setup

Please note, AWS changes the console website frequently and these instructions may not match the current website exactly.

1. Open command line to lab files

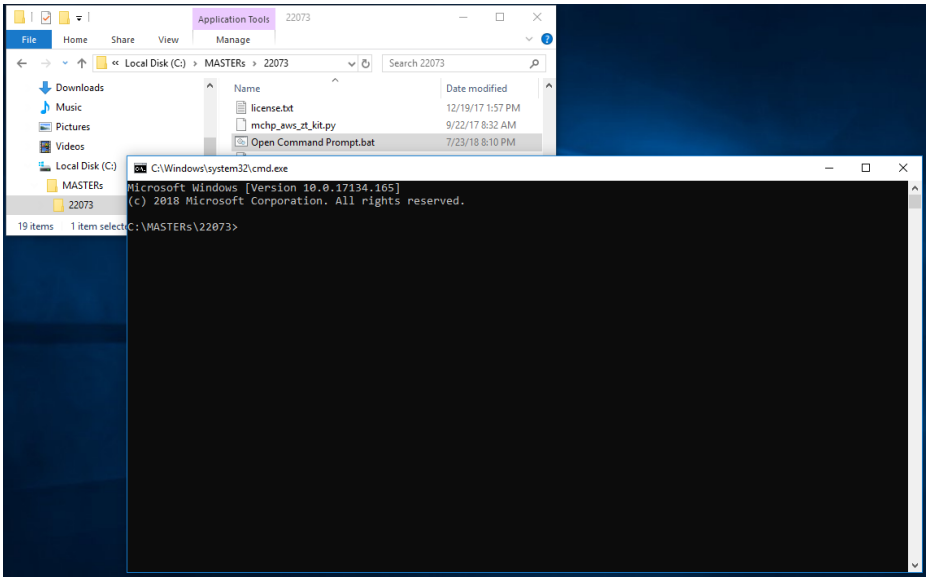
    Open **windows explorer** (Start -> Computer or keyboard shortcut win + E).

    Browse to **This PC -> Local Disk (C:) -> MASTERs -> 22073**

        If performing this lab outside the classroom, you will need to browse to the folder you saved the lab files to.

Run the "**Open Command Prompt.bat**" script.

You should get a command prompt that looks like this:



2. Obtain AWS Credentials

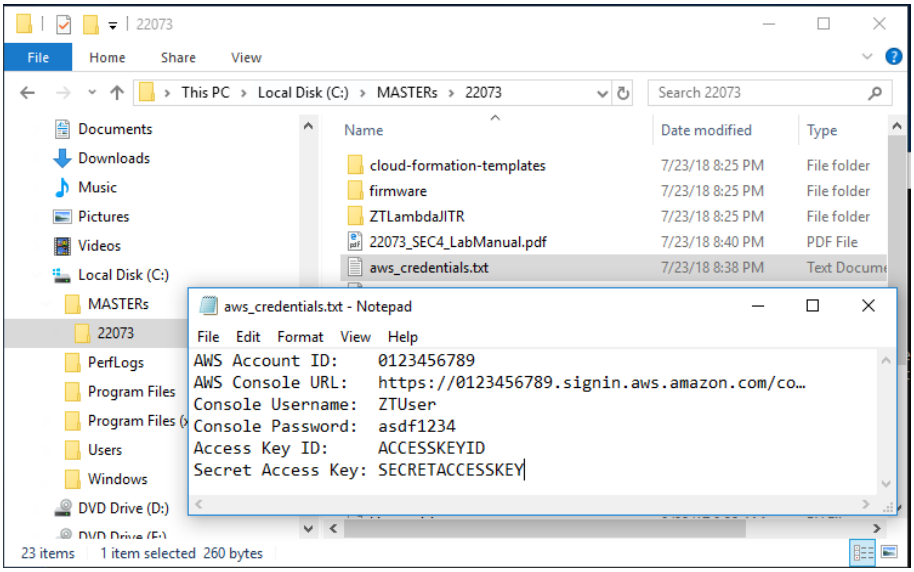
If running this lab *outside* the class, please refer to appendix A, step 3 for how to obtain your own credentials and skip to the next step.

In the class room, we will use the aws_get_credentials.py python script to get credentials to an AWS account for use during the class.

Please run the following command using the password given out during the class.

```
>python aws_get_credentials.py --password password
AWS Account ID:     0123456789
AWS Console URL:    https://0123456789.signin.aws.amazon.com/co…
Console Username:   ZTUser
Console Password:   asdf1234
Access Key ID:      ACCESSKEYID
Secret Access Key: SECRETACCESSKEY
```

These credentials are also saved to the aws_credentials.txt file for future reference.



3. Configure AWS Credentials

Before we can interact with AWS, we need to configure the tools with the appropriate AWS credentials. These credentials are composed of the **Access Key ID** and the **Secret Access Key** and were obtained in the previous step. Outside of the classroom, they will be generated when creating the IAM user for the lab as described in the appendix.

We will be using the AWS CLI (command line interface) to set the credentials. From the command prompt, run the following command:

```
aws configure
```

Enter your Access Key ID and Secret Access Key when prompted. You should copy and paste the credentials to avoid any typos. Pasting in the command prompt is performed by **right-clicking**.

We will use the us-west-2 region. Leave “Default output format” blank.

```
>aws configure
AWS Access Key ID [None]: ACCESSKEYID
AWS Secret Access Key [None]: SECRETACCESSKEY
Default region name [None]: us-west-2
Default output format [None]:
```

Once configured these setting will be used by both the AWS CLI and the python scripts.

More information can be found at the following links:

http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html

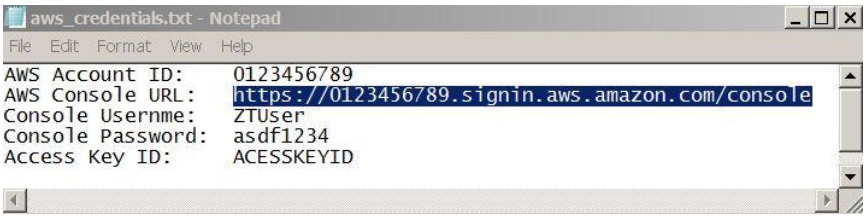
https://boto3.readthedocs.io/en/latest/guide/quickstart.html#configuration

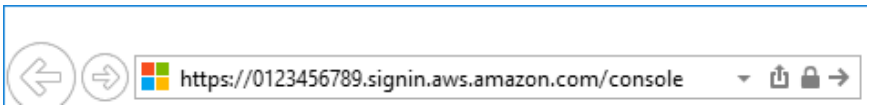
4. Log into the AWS console webpage

Open up a web browser (e.g. Internet Explorer)

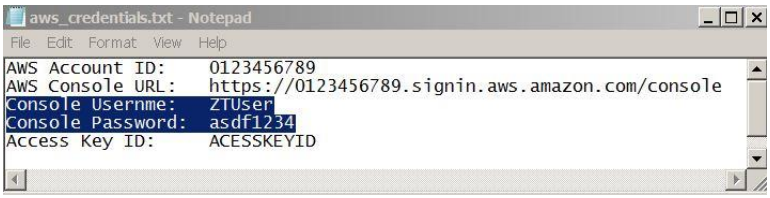
Go to the AWS Console URL as specified in step 1. The web address looks like:

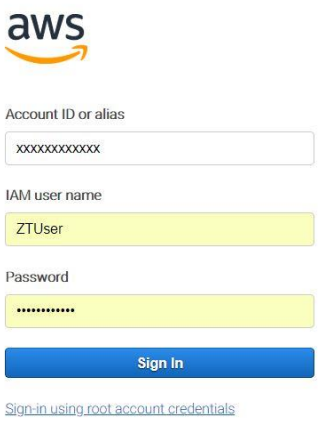
https://*xxxxxxxxxxxx*.signin.aws.amazon.com/console
where *xxxxxxxxxxxx* is the account ID



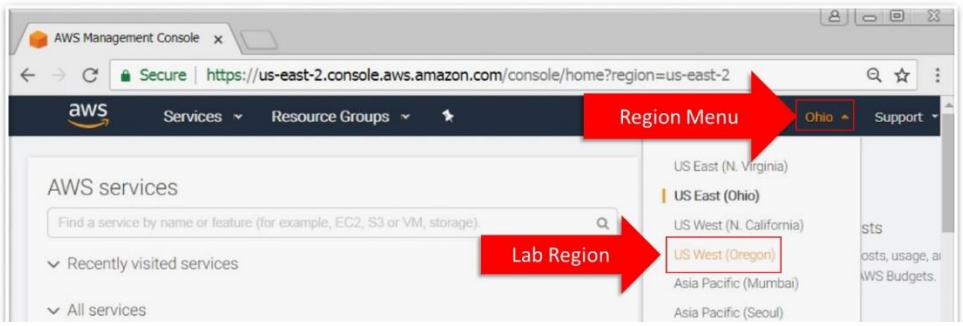


Log in to the AWS console using the Username and password also provided in step 1

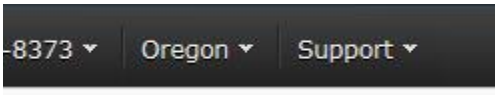




Once logged in, change your region to US West (Oregon), by selecting the region menu (upper-right, left of support menu).



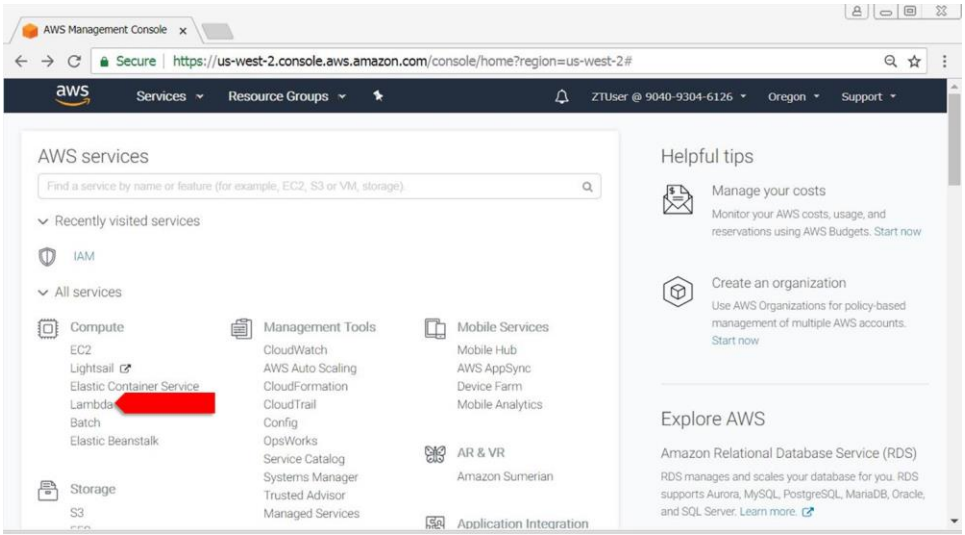
Properly selected, the region menu should now say **Oregon**.



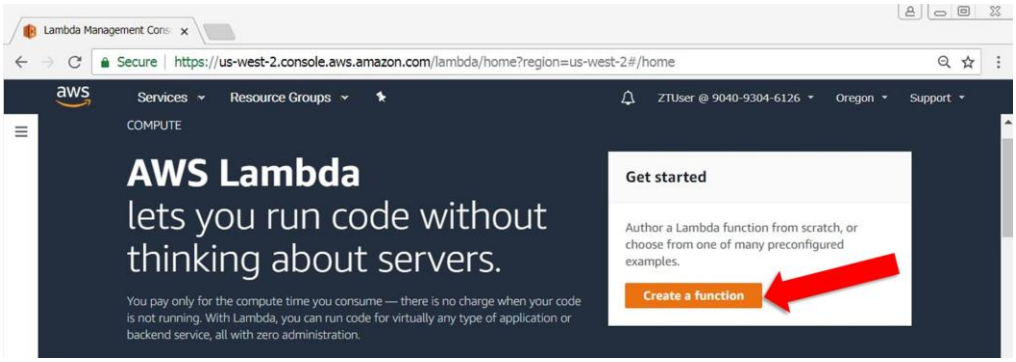
5. Create the Just In Time Registration (JITR) Lambda Function

The JITR lambda function is code that is called within AWS when a new device attempts to connect, but isn't registered yet. It's this function's responsibility to perform the actual registration of the device with AWS IoT.

a) Go to the **Lambda** service under **Compute** category of the **All services** menu.



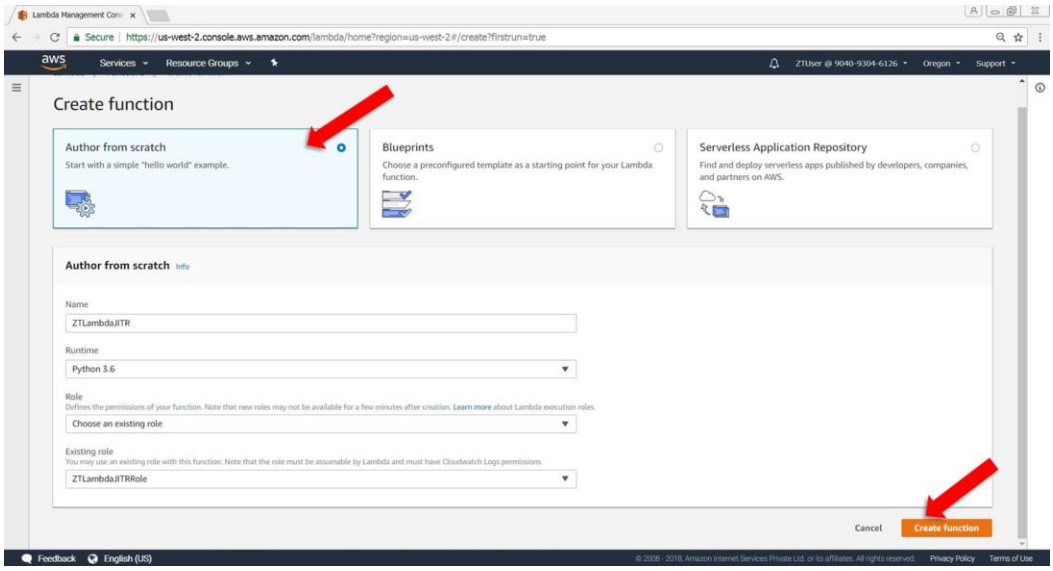
b) Click **Create a function**



c) Select **Author from scratch**

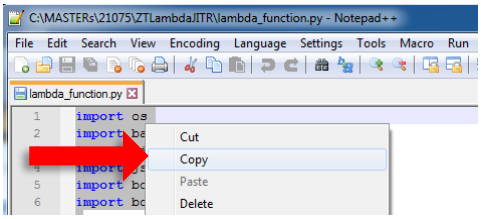
Name: **ZTLambdaJITR**
Runtime: **Python 3.6**
Role**:** **Choose an existing role**
Existing role**:** **ZTLambdaJITRRole**

And click **Create function**



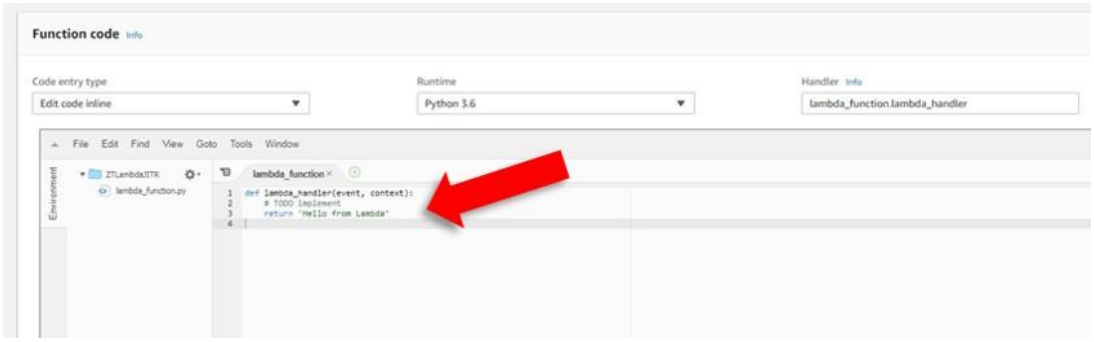
d) Switch to the file explorer and open the **C:\MASTERs\22073\ZTLambdaJITR\lambda_function.py** file in the Notepad++ editor by **right-clicking** on the file and selecting **Edit with Notepad++**

e) Select all the code in Notepad++ (Ctrl-A is a helpful shortcut) and copy



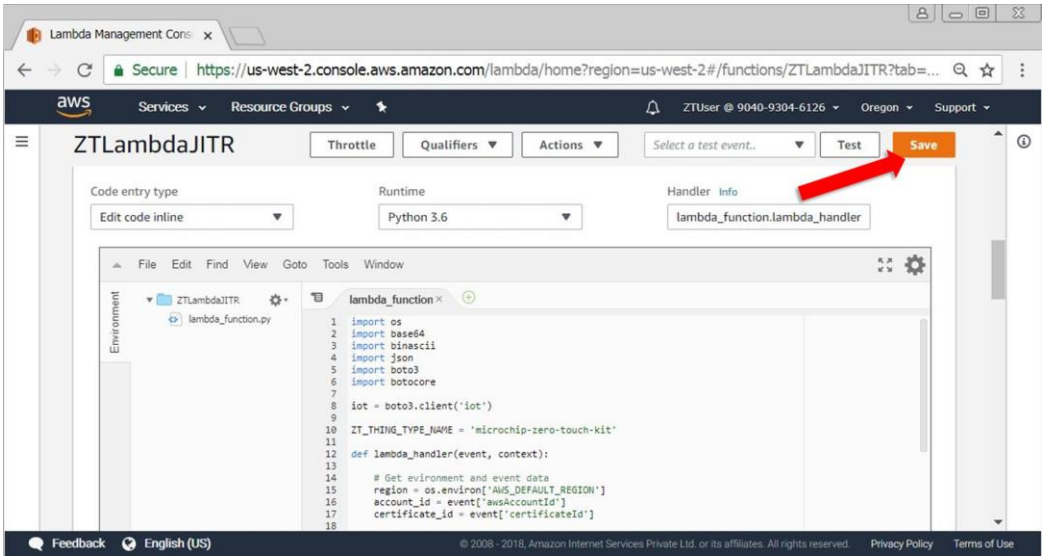
f) Switch back to the AWS console webpage. Make sure the following is selected
Code entry type: **Edit code inline**
Runtime: **Python 3.6**
Handler: l**ambda_function.lambda_handler**

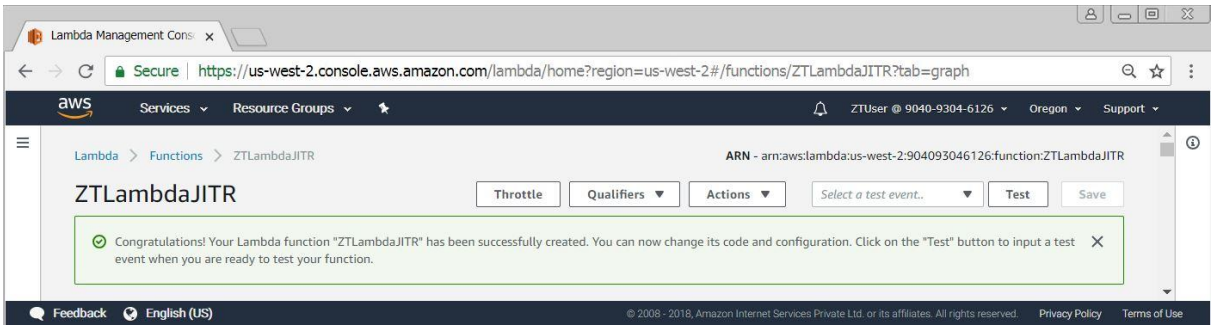
g) Delete the contents of the code entry area by selecting everything and hitting delete



h) Paste the new code from the **ZTLambdaJITR\lambda_function.py** file into the code entry area.

i) Click on **Save** to create the function

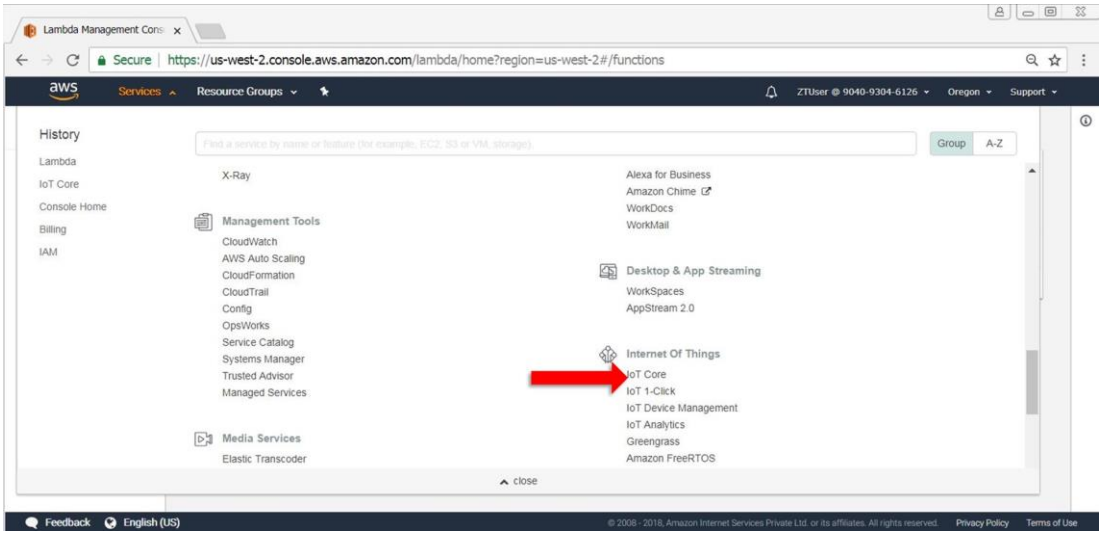




6. Create IoT Rules Engine Rule

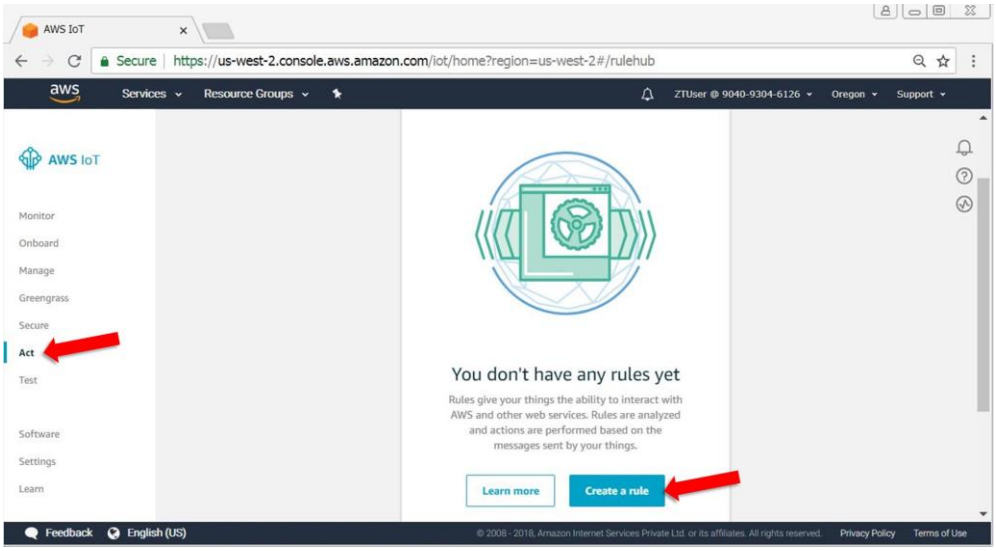
   While the lambda function performs the registration, it needs to be triggered by an event. The following instructions will create a rule that will run the lambda function when a device connects for the first time.

   a) Go to the **IoT Core** service under the **Services** menu and **Internet of Things** category



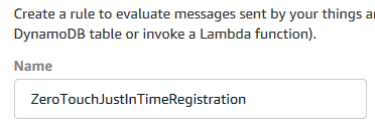
   b) Sometimes, the AWS IoT Console will show a getting started window, click the **Get started** button to dismiss the intro screen.

c) Click on **Act** and then click the **Create a rule** as shown in below figure

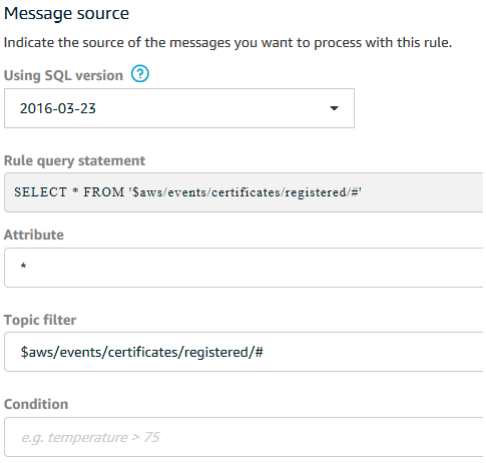


d) Fill out the following fields:

Name: **ZeroTouchJustInTimeRegistration**



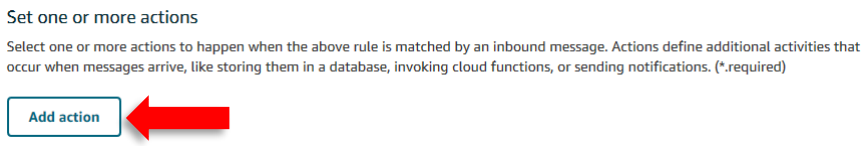
SQL version: **2016-03-23**
Attribute: *****
Topic filter: **$aws/events/certificates/registered/#**
Condition:



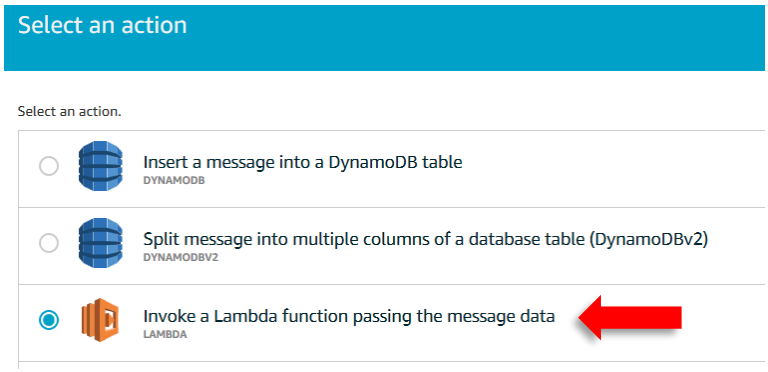
$aws/events/certificates/registered/# is a special administrative MQTT topic that AWS IoT will publish to when a device connects with a certificate that hasn't been seen before, but *has* been signed by a CA that was registered in the account.

The # at the end indicates we want to trigger this rule for any CA registered with the account.

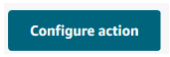
e) Click **Add action**



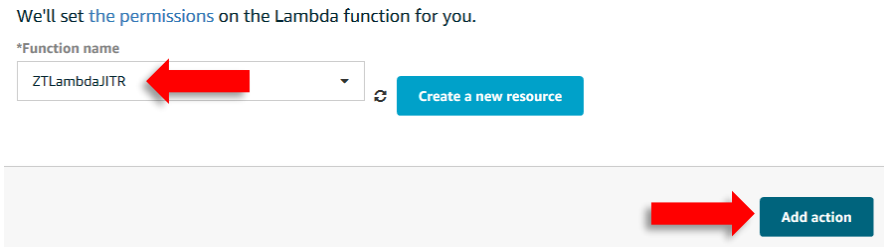
f) Select **Invoke a Lambda function passing the message data**



g) Click **Configure action** (back at bottom of page)

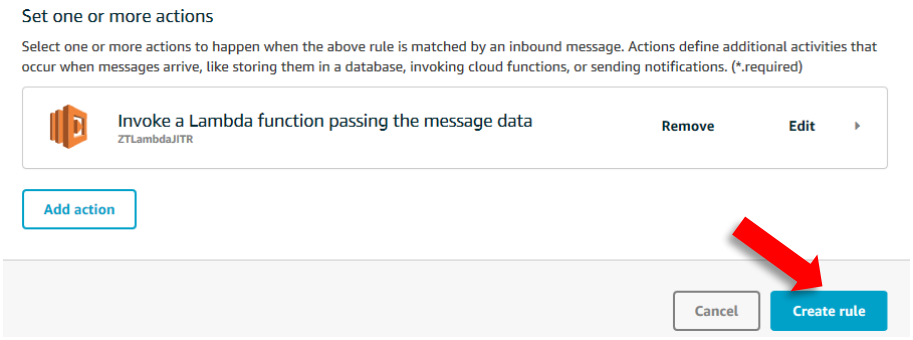


h) Select the **ZTLambdaJITR** function and click **Add action**



Now that this configured, this rule will trigger our registration lambda function when a new device is seen.

i) Finish by clicking **Create rule**



Certificate Authority Setup

1. Open command line to lab files

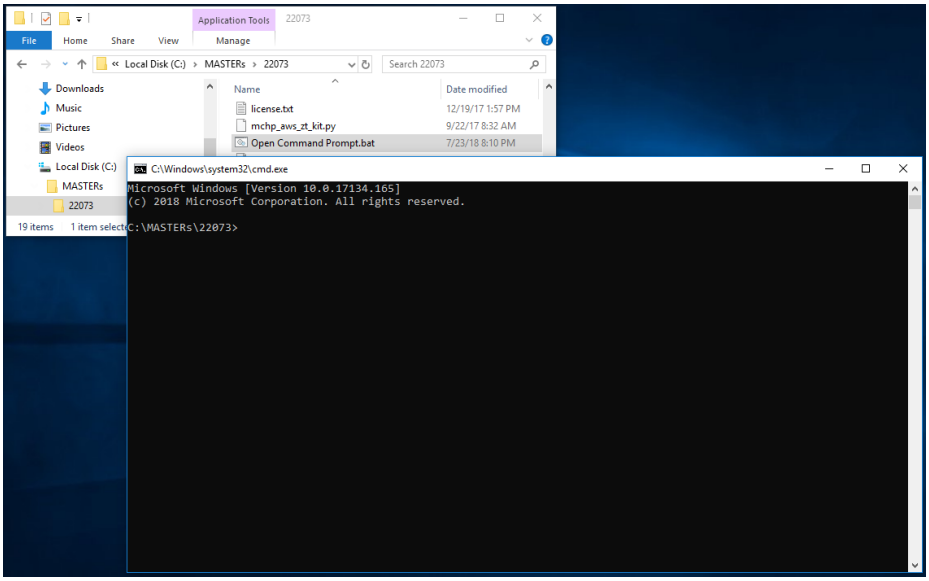
   Open **windows explorer** (Start -> Computer or keyboard shortcut win + E).

   Browse to **This PC -> Local Disk (C:) -> MASTERs -> 22073**

   > If performing this lab outside the classroom, you will need to browse to the folder you saved the lab files to.

   Run the "**Open Command Prompt.bat**" script.

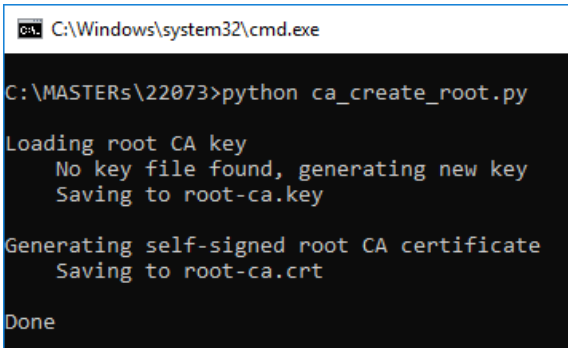
   You should get a command prompt that looks like this:



2. Create the root key and certificate

   The root certificate authority serves as a single authority over an IoT ecosystem.

   Run `python` **`ca_create_root.py`**



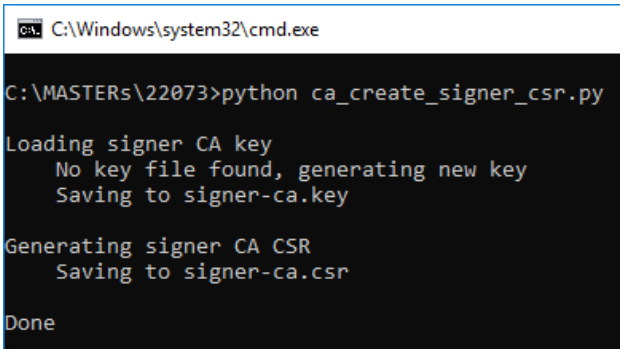
This script will create the root key, **root-ca.key**, and the root certificate **root-ca.crt**. Because this is the root CA, its certificate is signed by its own key.

If the root-ca.key file already exists, the script will use that existing key and generate a new certificate.

3. Create the signer key and certificate signing request

   The signer certificate authority is responsible for directly signing the device certificates. Signer creation is split into 2 steps, the first is generating its key and a certificate signing request.

   Run `python` **`ca_create_signer_csr.py`**



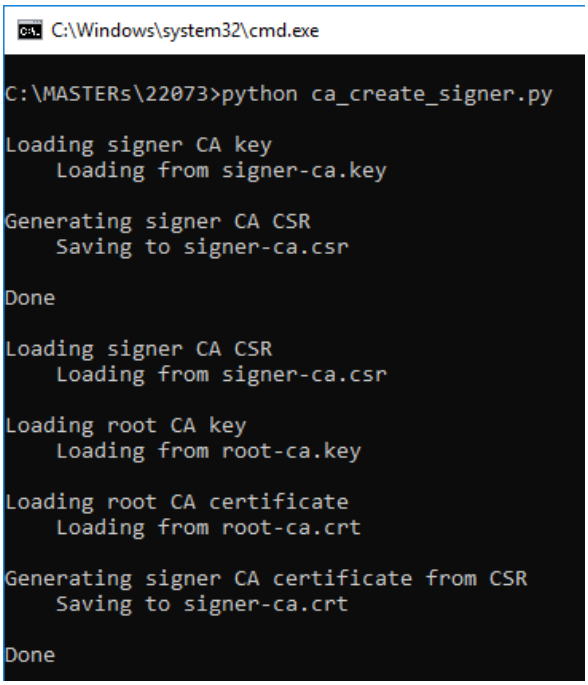
   This script will create the signer key, **signer-ca.key**, and its CSR, **signer-ca.csr**.

   If the signer-ca.key file already exists, the script will use that existing key and generate a new CSR.

4. Create the signer certificate

   The root CA is now used with the signer CSR to create a signer certificate. While this could technically be done in a single script in the lab, it's split out to represent the split in responsibilities between the authority and subject in PKI systems.

   Run `python` **`ca_create_signer.py`**



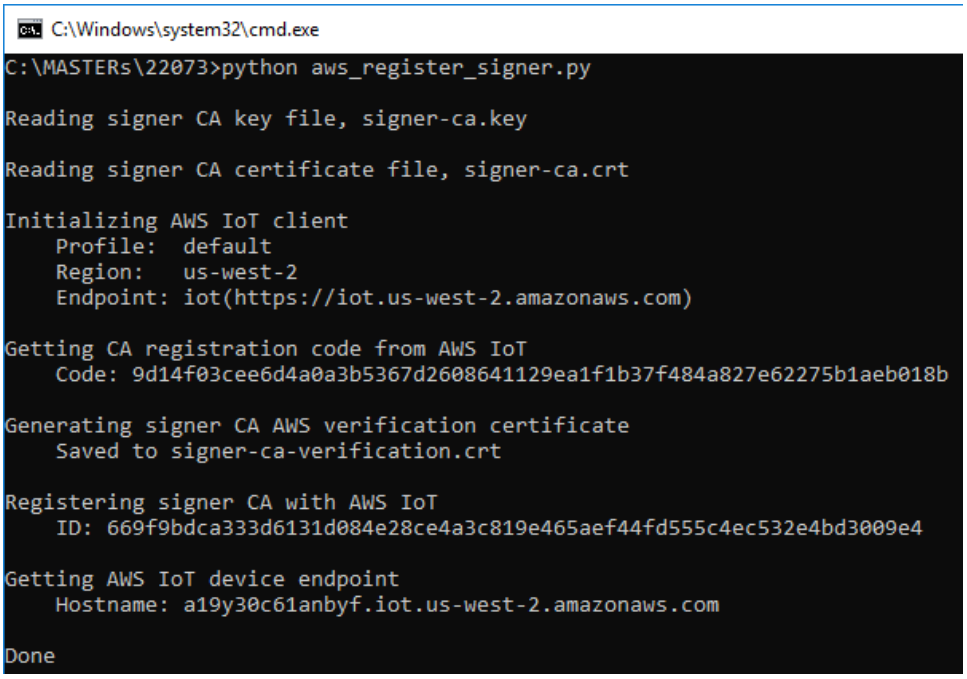
   This script will create the signer certificate, **signer-ca.crt**.

5. Register the signer with AWS IoT

   The final step in setting up the certificate chain is to register the signer with AWS IoT. As a security feature, AWS IoT requires one prove they have access to the CA private key before registering that CA. This involves the following steps:

   - Request a verification code from AWS IoT
   - Create a verification certificate around that verification code
   - Sign the verification certificate with the CA (signer)
   - Supply both the CA (signer) certificate and verification certificate when registering

Run `python` **`aws_register_signer.py`**



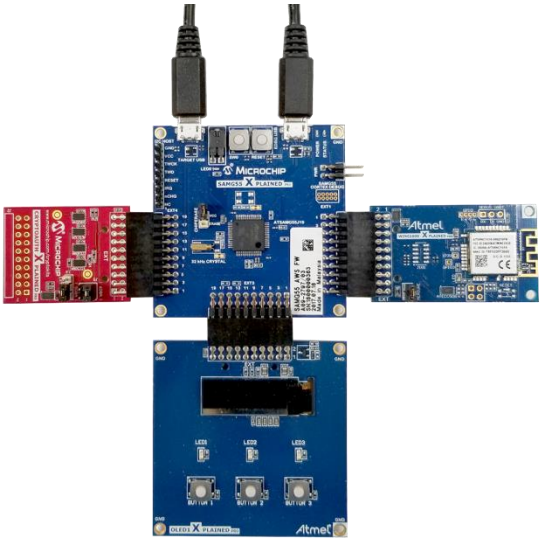
This script will perform the above steps and save the verification certificate to **signer-ca-verification.crt**. This file is not required by any other step, but is saved for reference.

Provision the Device

1. Assemble and plug in the kit

   The SAMG55 Xplained Pro forms the central hub, while the other boards plug in:

   EXT1: WINC1500 Xplained Pro
   EXT3: OLED1 Xplained Pro
   EXT4: CryptoAuth Xplained Pro



2. Plug in the board to the PC from the **TARGET USB** port on the SAMG55 board.
3. Connect a second USB cable, connect the EDBG USB port to the PC as well.

   Debugging information is exposed via a com port available through the EDBG connection.

   To see the debugging information we will need to connect to the COM port using a terminal program.

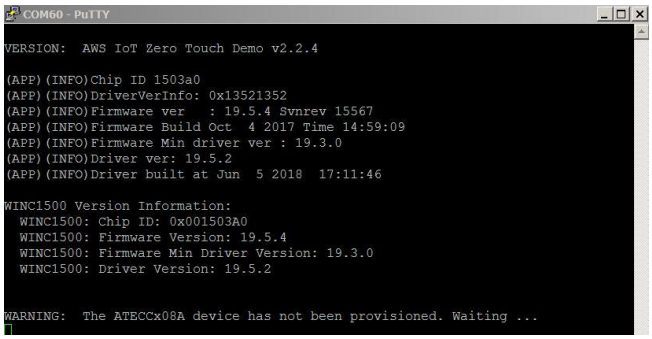
   If using PuTTY:

   To find the right com port number, open **device manager**, expand **ports** and look for the port labeled **EDBG Virtual Comport (COM*x*)**, where x is the number you're looking for.

   Next, to see the board status, open PuTTY and enter the following:

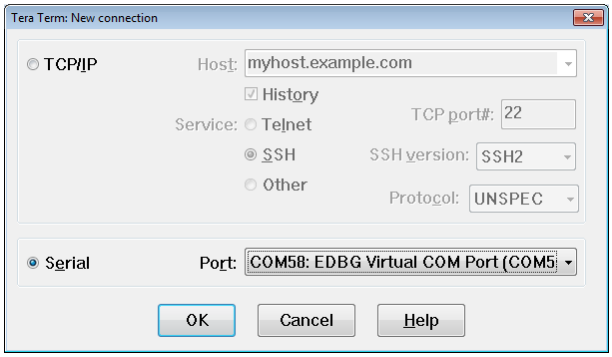
   Connection type: **Serial**
   Serial line: **COM*x*** – where x is the number from the previous step
   Speed: **115200**

   Click **Open** and you should see a window with status messages. If nothing appears, try pressing the RESET button on the SAMG55 board.

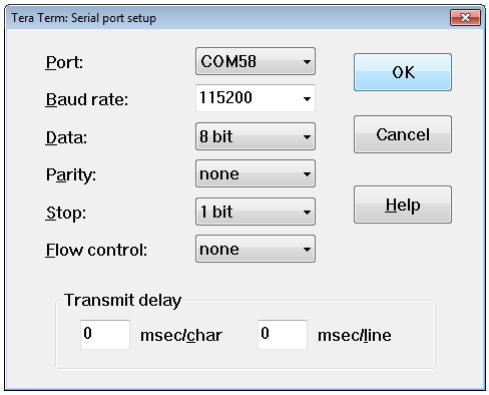


If using Tera Term:

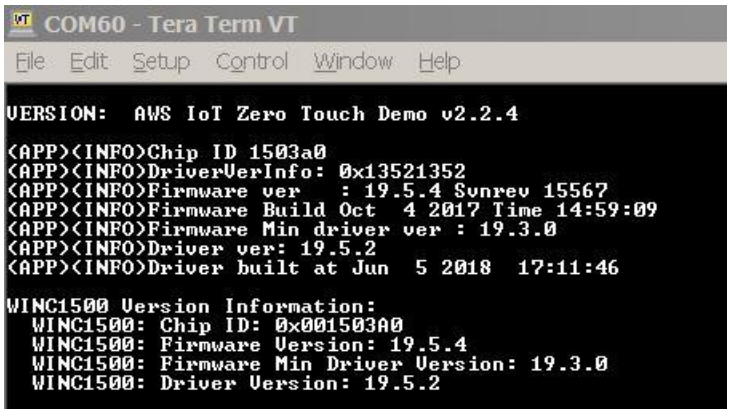
Open **Tera Term**, select **Serial**, select the **EDBG Virtual COM Port** (actual COM number may be different), and click **OK**:



Got the **Setup** menu and select **Serial**. Change the **Baud rate to 115200**, click **OK**:



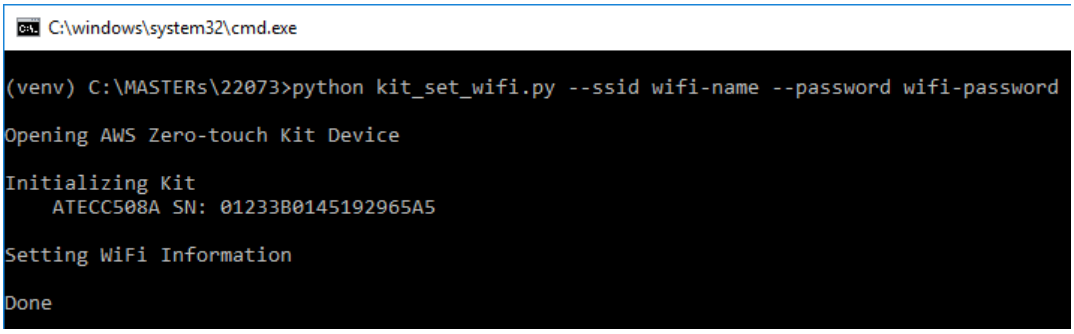
You should see a window with status messages. If nothing appears, try pressing the RESET button on the SAMG55 board.



4. Set the WiFi credentials

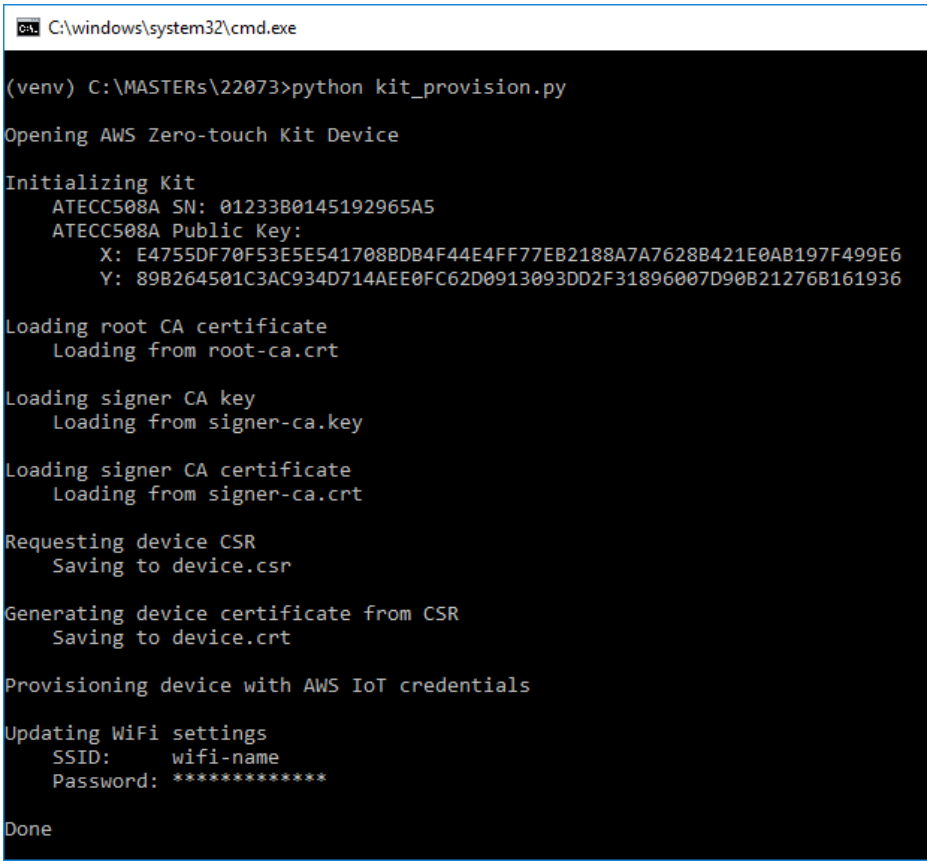
Run `python` **`kit_set_wifi.py --ssid`** *`wifi-name`* **`--password`** *`wifi-password`*

Where wifi-name and wifi-password are the WiFi credentials supplied in the class. Outside the class, one will need a wifi network with internet access. Ports 123 and 8883 will need to be open for time server access and secure MQTT access respectively.



5. Provision the device

Run `python` **`kit_provision.py`**



This will request a CSR from the device.

> The CSR will use the key pair stored in slot 0 of the ATECC608A. The ATECC608A is a secure container for the private key. The key internally generated with its secure RNG and the ATECC608A provides no mechanism for reading out a private key.

This key provides a secure identity for the IoT device that can't be copied, either intentionally, by an attacker, or through a software bug.

Create a device certificate using the CSR and signer CA.

Send the device certificate, signer certificate, and AWS connection information to the board.

These certificates and the AWS connection information is all stored on the ATECC608A:

Slot 8 – AWS Connection Information (including wifi credentials)

Slot 10 – Device compressed certificate

Slot 11 – Signer public key

Slot 12 – Signer compressed certificate

Slot 14 – Signer certificate serial number and full validity dates

Once the board has been successfully provisioned, LED0 on the SAMG55 Xplained Pro board should blink 5 times. Additionally, if watching the debug output from the EDBG virtual com port in PuTTY, one should see the following message:

```
Attempting to connect to AWS IoT ...
  SSID:     SCH-I545D1C4
  Password: idit852!
WINC1500 WIFI: Connected to the WIFI access point.
WINC1500 WIFI: Device IP Address: 192.168.97.192
WINC1500 WIFI: DNS lookup:
  Host:       a22gwrh37tk4qi.iot.us-west-2.amazonaws.com
  IP Address: 52.36.212.103
(APP)(INFO)Socket 0 session ID = 1
WINC1500 WIFI: Device Time:       2018/06/11 10:58:39
SUCCESS:  AWS Zero Touch Demo: Connected to AWS IoT.


SUCCESS:  Subscribed to the MQTT update topic subscription:
SUCCESS:  $aws/things/46bd2db50fae58e3c662769a19a3ba2f718d8fbc/sh
adow/update/delta
```

Note, it should take the board at least two attempts to successfully connect after being provisioned. On the first attempt, AWS IoT will disconnect it, because the device certificate isn't registered yet. However, this should kick off the device registration lambda function (ZTLambdaJITR) in AWS to perform the actual registration. The board's second attempt to connect should succeed assuming the registration process has completed by then.

Note that all asymmetric math (authentication and key agreement) used during the TLS handshake is routed through the ATECC608A from the WINC1500. The WINC1500 has a callback system that sends requests for ECC crypto operations to the MCU, the MCU then sends these requests to the ATECC608A, and returns the results back to the WINC1500.

The board uses AWS IoT's shadow system topics to inform AWS of state changes (button presses) and to learn of requested state changes (LED status).

Device Shadows - https://docs.aws.amazon.com/iot/latest/developerguide/iot-device-shadows.html

Device Shadow Topics - https://docs.aws.amazon.com/iot/latest/developerguide/device-shadow-mqtt.html

The board subscribes to the $aws/things/thingName/shadow/update/delta topic, which will send out messages whenever the reported device state differs from the desired device state. The board receives LED state updates through this topic.

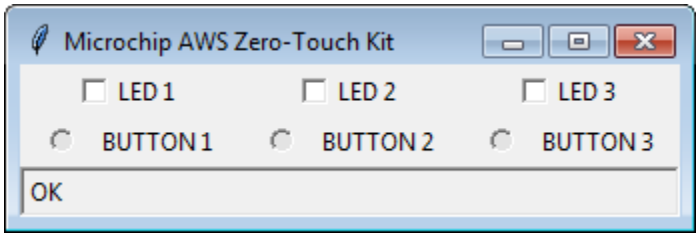
The board separately publishes to the $aws/things/thingName/shadow/update topic to inform AWS of button state changes.

AWS IoT Interaction

Now that the board has been provisioned, we will pass some simple messages back and forth to toggle the LEDs and show button state.

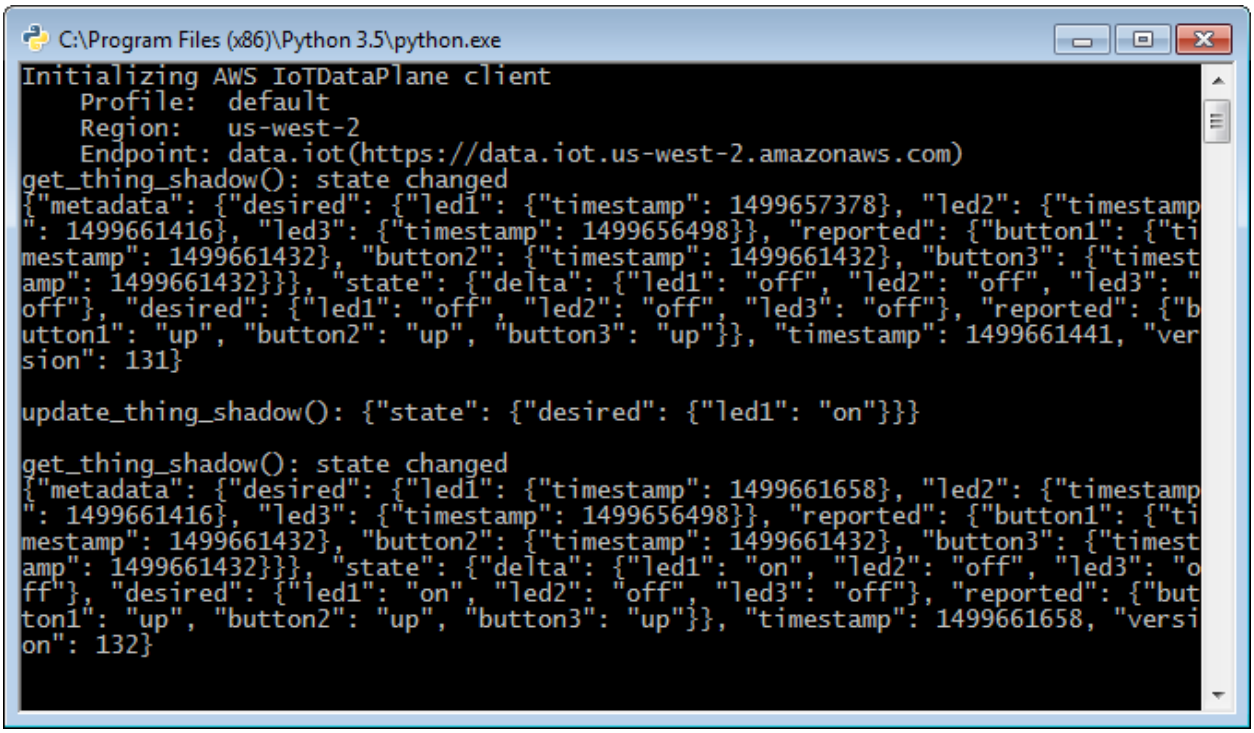
Run `python` **`aws_interact_gui.py`**

After successfully connecting the AWS from the PC side, it will create a simple interface for interacting with the board:

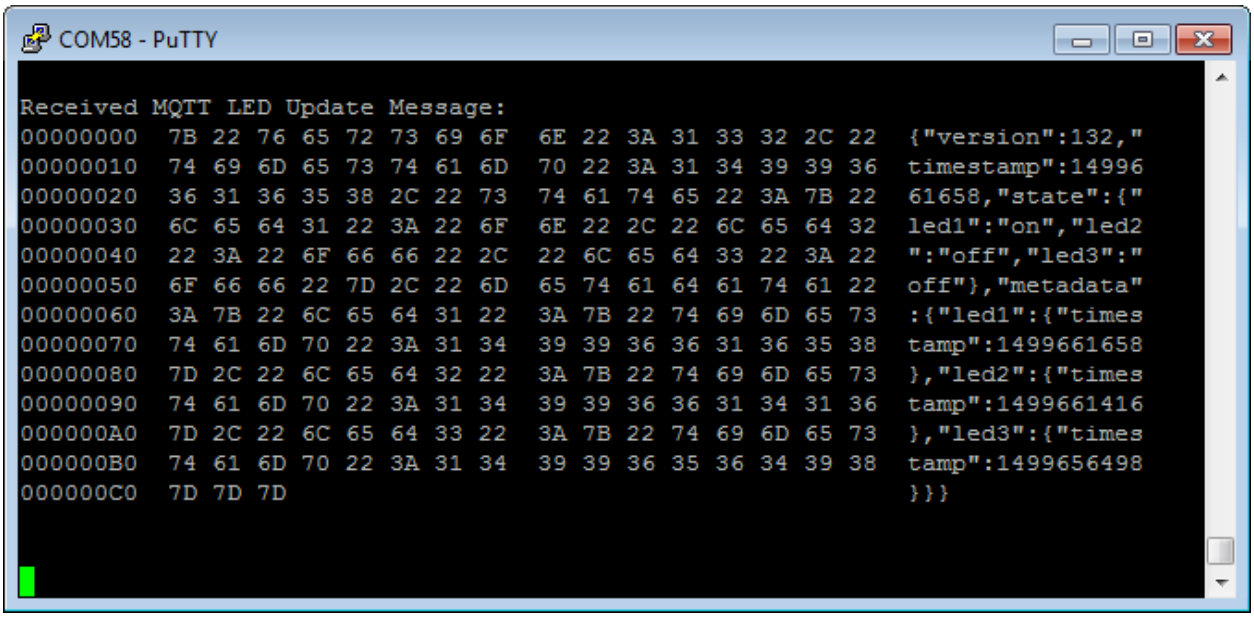


Selecting any of the LED check boxes will turn on or off the LEDs on the OLED1 Xplained Pro board. Likewise, pressing the buttons on the board will light up the indicators in the interface, showing their current state.

The script console window will show the messages being passed back and forth:



Likewise, the debug output from the EDBG virtual com port in PuTTY/TeraTerm will showing the corresponding messages on the device side:



Appendix A: Software Installation and AWS Account Setup

The Masters Conference laptops have all the software installations and configurations needed to run the lab for this class. If you want to perform this lab on your own laptop when Masters is over:

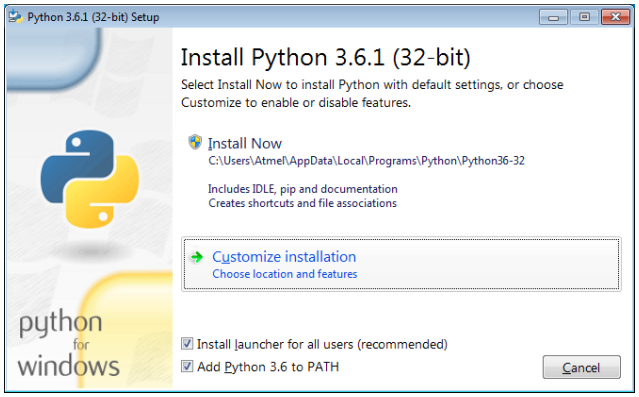
1. Install the software
2. Create your own AWS account
3. Sign into your AWS Console to manage user access and permissions

1. Install the Software:

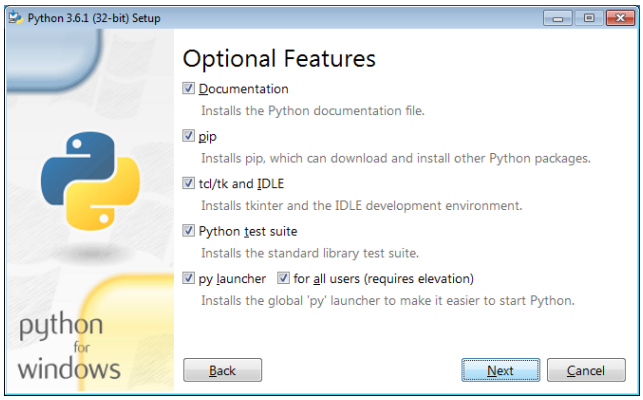
a. AWS CLI — https://aws.amazon.com/cli/
AWS Command Line Interface from the Windows command prompt

b. PuTTY — https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html
Terminal emulator (could use Tera Term or CoolTerm instead)

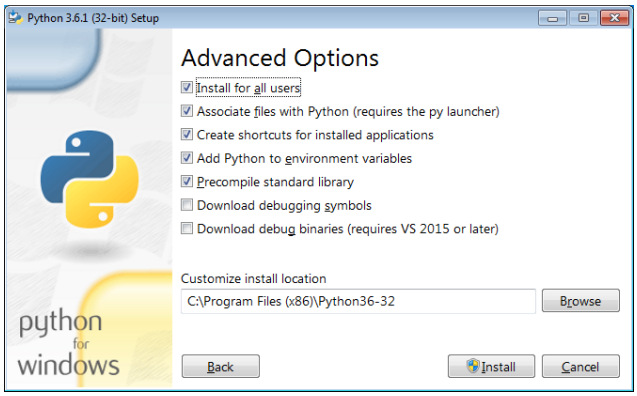
c. Python 3 — https://www.python.org/downloads/
An interpreted programming language used to interact with the hardware and AWS
Check **Add Python 3.x to PATH**



Choose **Customize installation** and make sure everything is selected



Click **Next**, then select **Install for all users** and **Precompile standard library**



Click **Install**

d. Python Packages

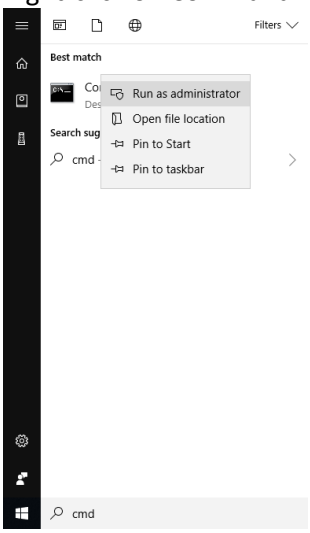
Found here `22073\requirements.txt`
This file is used by the python package manager (pip) to install all the required packages.

Please note, if you see the following error while trying to perform this step, please see the next step for install the C++ Build Tools, then repeat this one. Note the URL in the error message is no longer valid.

```
building 'hid' extension
    error: Microsoft Visual C++ 14.0 is required.
Get it with "Microsoft Visual C++ Build Tools":
http://landinghub.visualstudio.com/visual-cpp-
build-tools
```

These packages will be installed from an administrative command prompt.

- Open the start menu (bottom left window) and search for cmd
- Right-click on Command Prompt and select "*Run as **Administrator***"



From the command prompt that opens, navigate to the directory where the requirements.txt file is, and run the following command:

```
pip install -r requirements.txt
```

Alternatively, the packages can be installed to just the current user account, which doesn't require Admin access.

```
pip install -r requirements.txt --user
```

e. C++ Build Tools http://go.microsoft.com/fwlink/?LinkId=691126
Microsoft Visual C++ Build Tools 14.0 may be required for the hidapi python package. This is a big download/install and it is not needed if the previous step completed without error.

The following software is not required, but it can be useful:

OpenSSL https://wiki.openssl.org/index.php/Binaries
Standard software for working with certificates and keys.

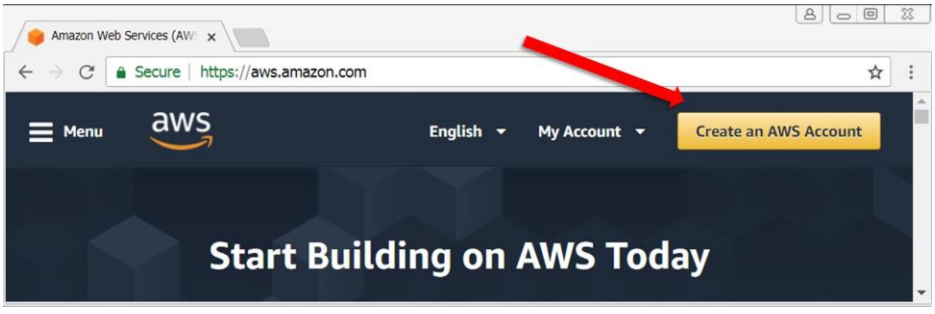
Notepad++ https://notepad-plus-plus.org/
Text editor with good syntax highlighting for a variety of file types.

ASN.1 Editor https://www.codeproject.com/Articles/4910/ASN-Editor
Tool for inspecting and editing ASN.1 data including X.509 certificates.

2. Create Your Own AWS Account

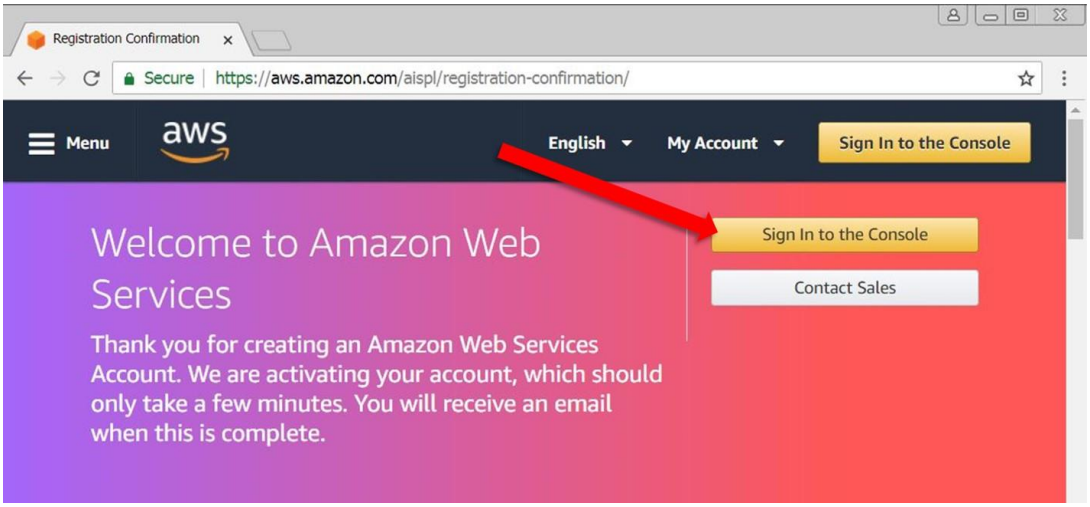
This lab's primary purpose is to demonstrate secure authentication with the AWS IoT managed cloud platform. Therefore, an AWS account is required to run the lab. If you don't have your own AWS account, go to this link to create one:

https://aws.amazon.com/

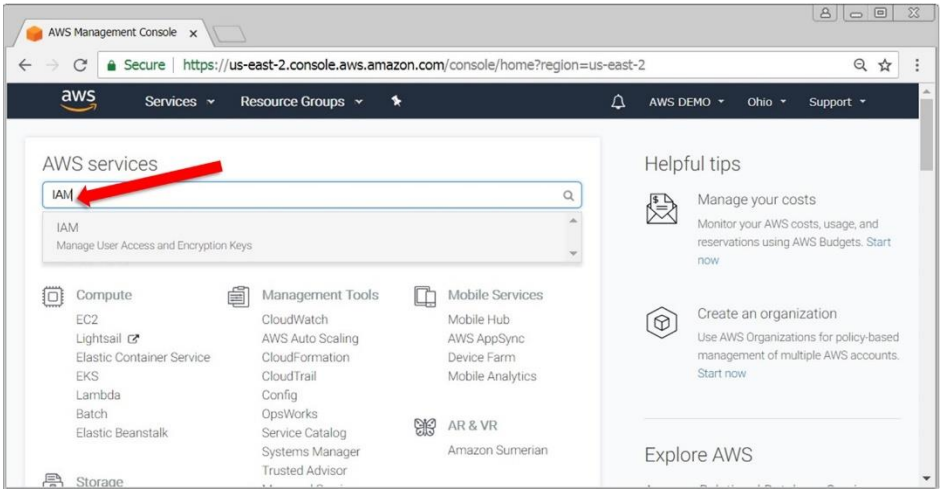


3. Sign into the AWS Console to Manage User Access and Permissions

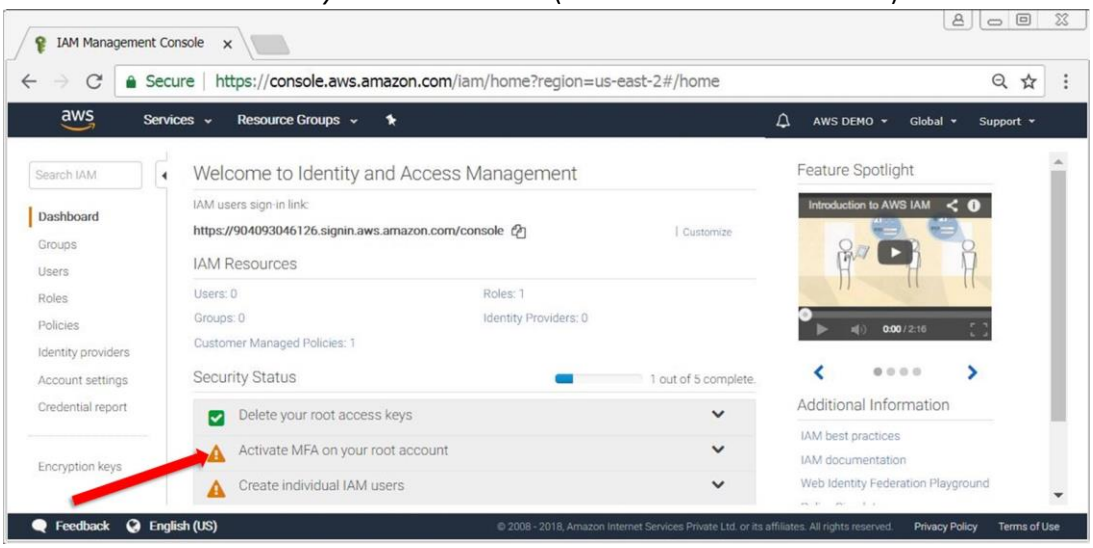
Once your account is created, you can sign into your AWS Console to access the IAM (Identity and Access Management) Service.



a. Search "***IAM***" and select IAM Manage User Access and Encryption Keys.

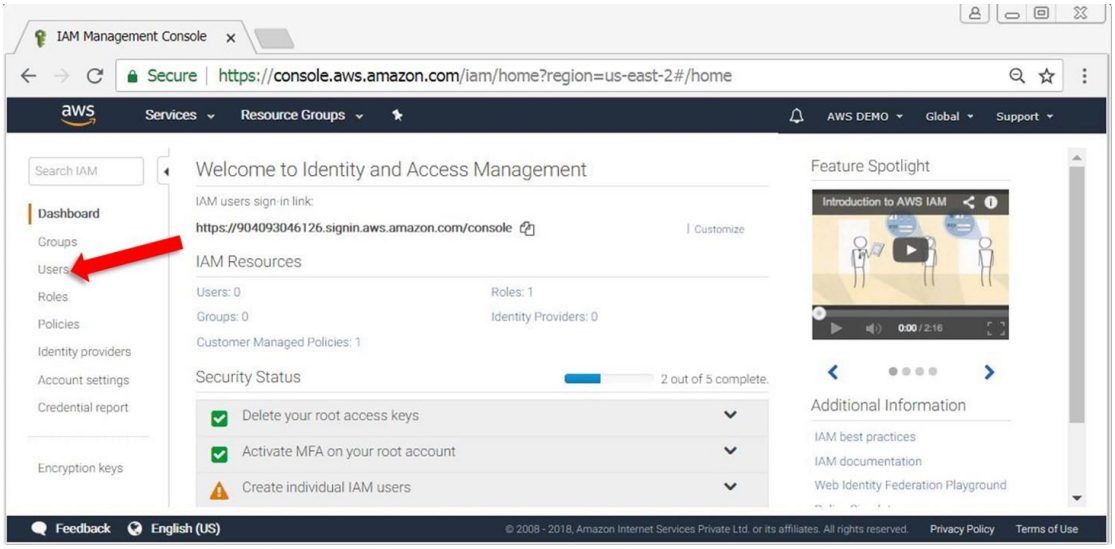


b. Click on **"*Activate MFA* *on your root account*"** (Multi-factor Authentication).



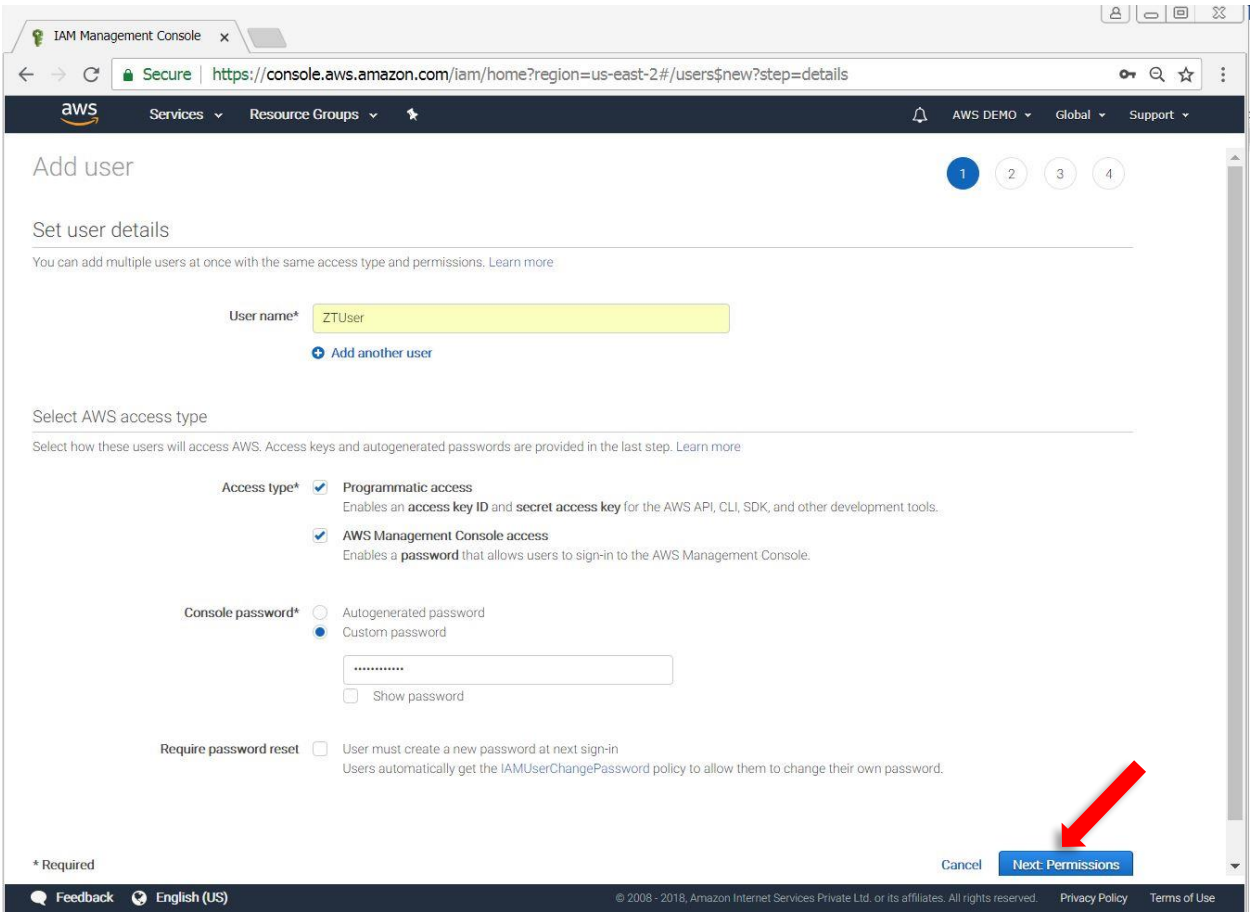
- This is an important step to better secure your root account against attackers. Anyone logging in not only needs to know the password, but also a constantly changing code generated by an MFA device.
- AWS recommends a number of MFA device options at the following link: https://aws.amazon.com/iam/details/mfa/
- The quickest solution is a **virtual MFA** device running on a phone. These apps provide the ability to scan the QR code AWS will generate to setup the MFA device.

c. Create a new user and group in your account (you can have many users and groups). You will run the lab as this user.
In the IAM Service window, click on "***Users***" then *"**Add user***".

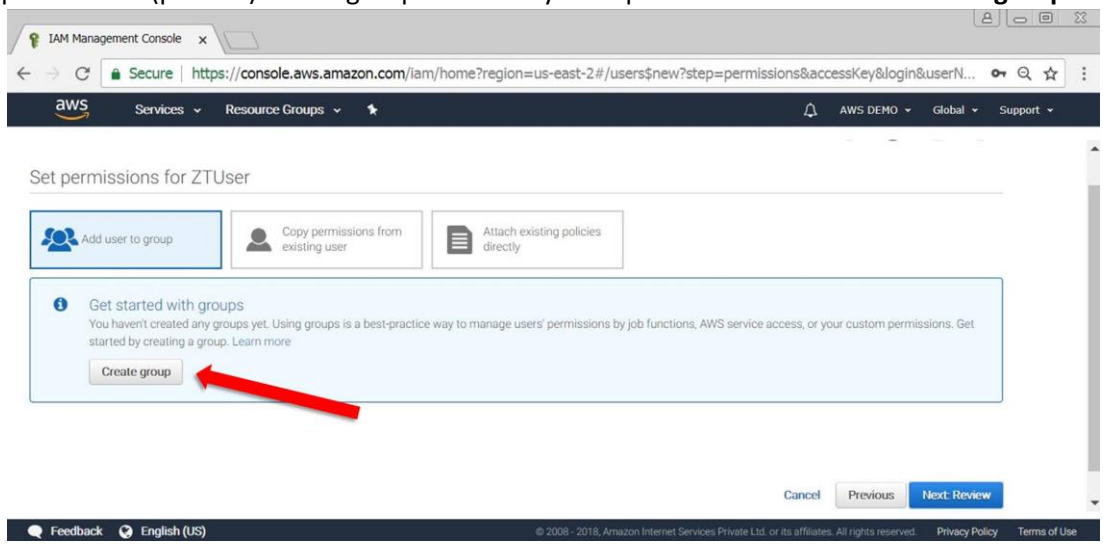


1. User name: **ZTUser**
2. Enable **Programmatic** access and **AWS Management Console** access
3. Select and enter a **Custom password**. Record this for logging in to the console later.

4. Unselect **Require password reset**
5. Click **Next: Permissions**

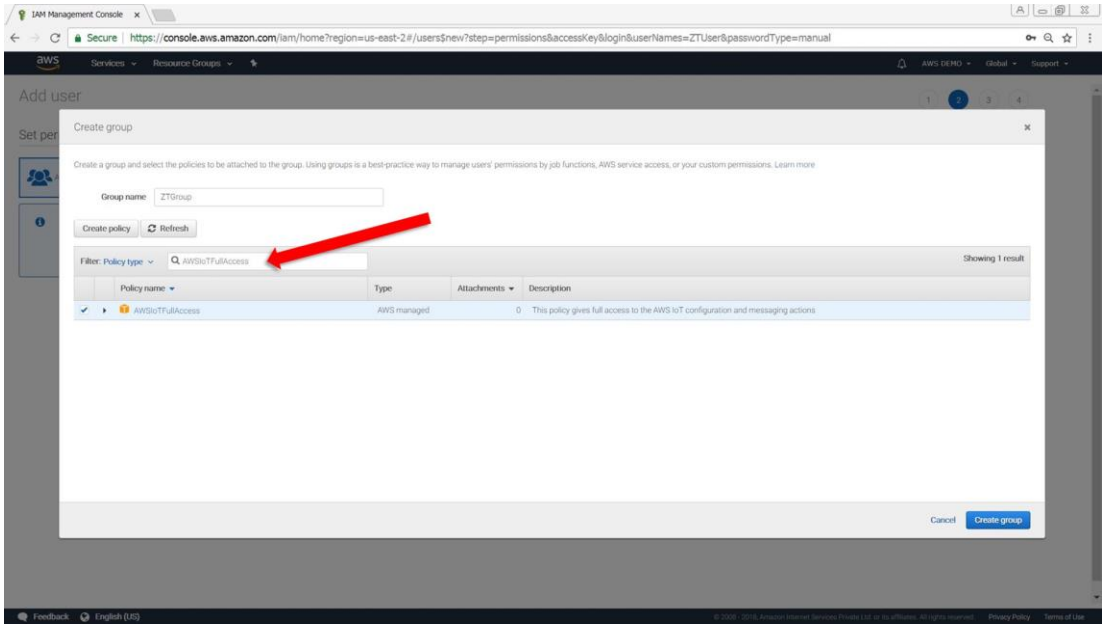


6. Create a new group and assign the user you just created to that group. You will also assign permissions (policies) to this group to enable you to perform the lab. Click "**Create group**"

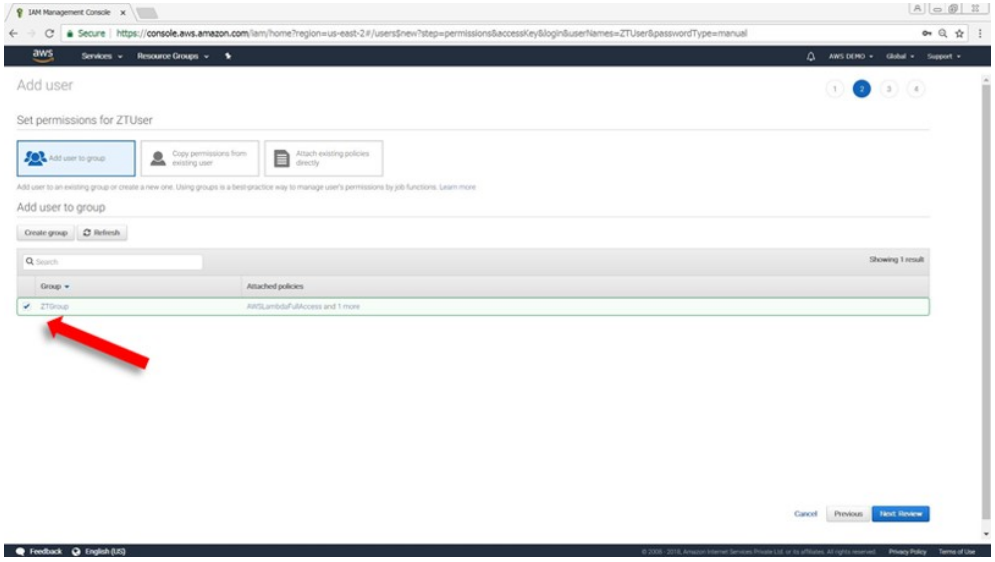


i. Group name: **ZTGroup**
ii. Search **AWSIoTFullAccess** and attach policy (select it)
iii. Search **AWSLambdaFullAccess** and attach policy (select it)

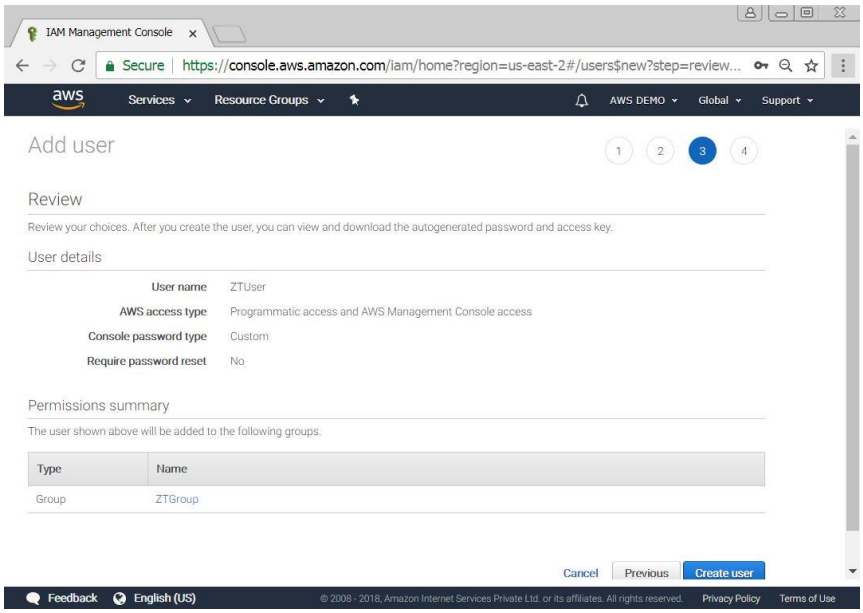
and Click **create group**



7. Select the new **ZTGroup** to add the new user to it and Click **Next: Review**,



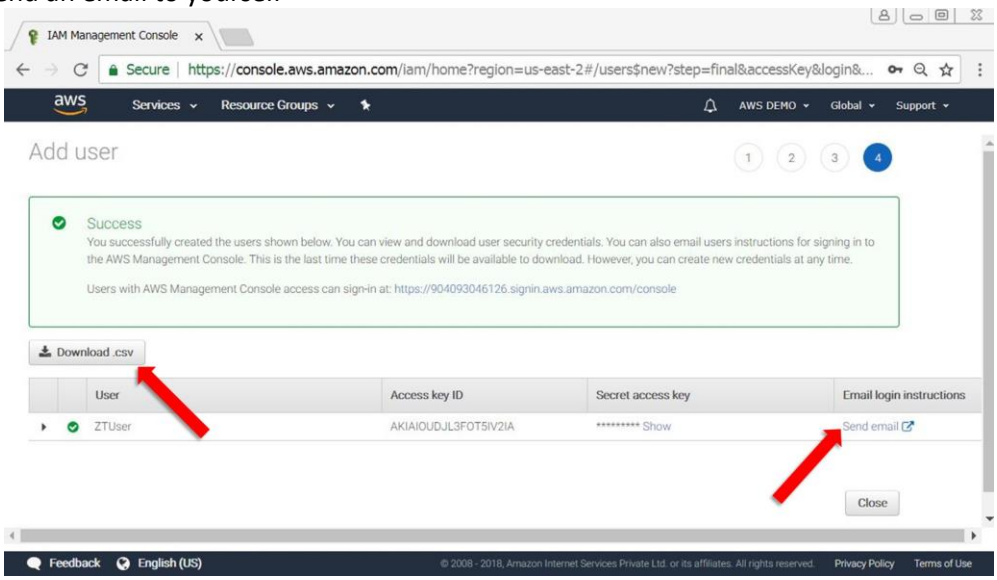
8. Then select the **Create User**



9. When you are done creating this user, AWS will create a personal sign-in URL and access keys for the user. To access your AWS account as this user, you will need this URL and keys.

   The keys and URL are quite long, so you will want to copy and paste them from somewhere.
   - Example URL: https://*xxxxxxxxxxxx*.signin.aws.amazon.com/console
     where *xxxxxxxxxxxx* is the account ID
   - Example Access key ID: AKIAJLX6XEDTYTWCF57B
   - Example Secret access key: T6rp/lFT4kjJFvW+RwhMVNiW46Qfo7EBF56H7jsb

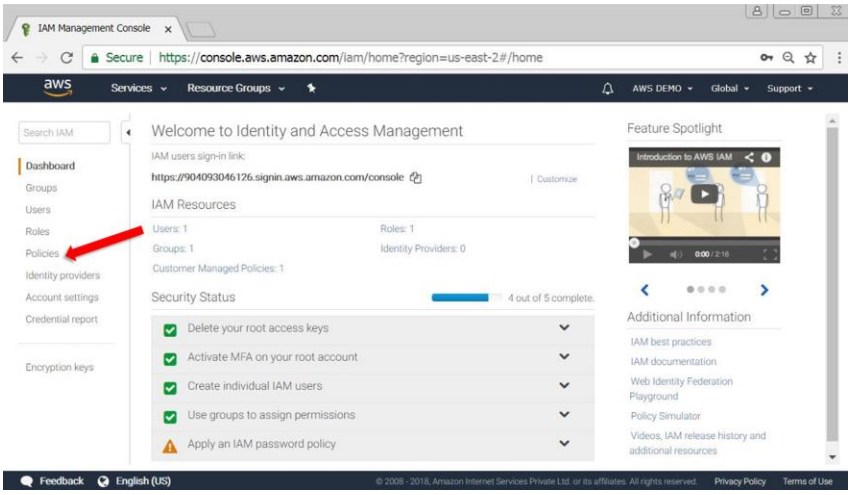
   There are two ways to get easy access to these security credentials:
   - Download a csv file
   - Send an email to yourself

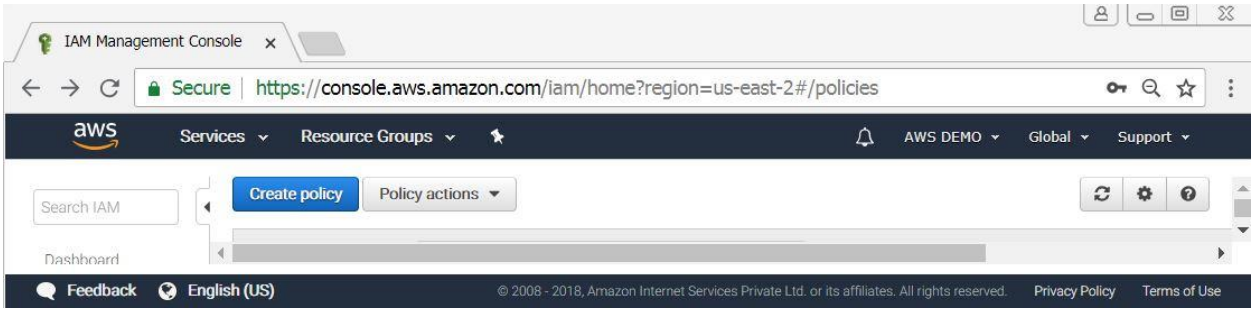


d. Create the JITR Lambda Function Policy

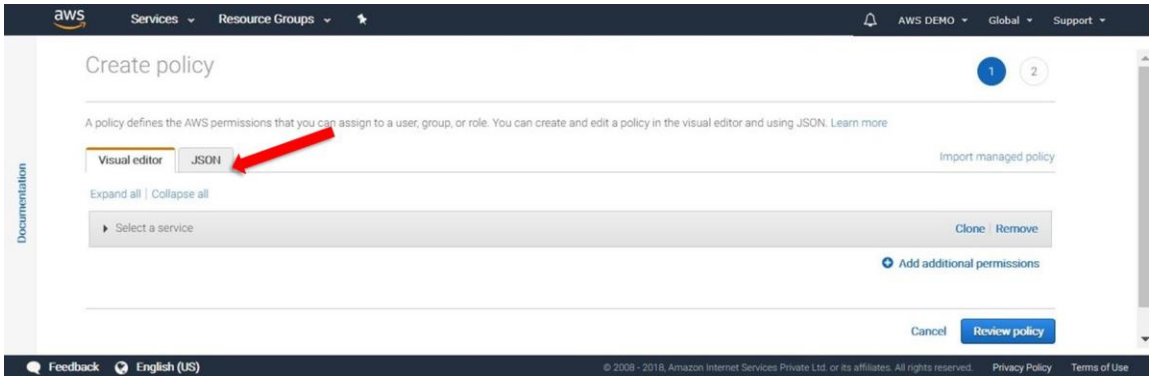
The JITR (Just in Time Registration) process is controlled by a lambda function. However, before the lambda function can be created we need to define a new policy and role describing the permissions it has within AWS.

Click on "**Policies**" then "**Create policy**"



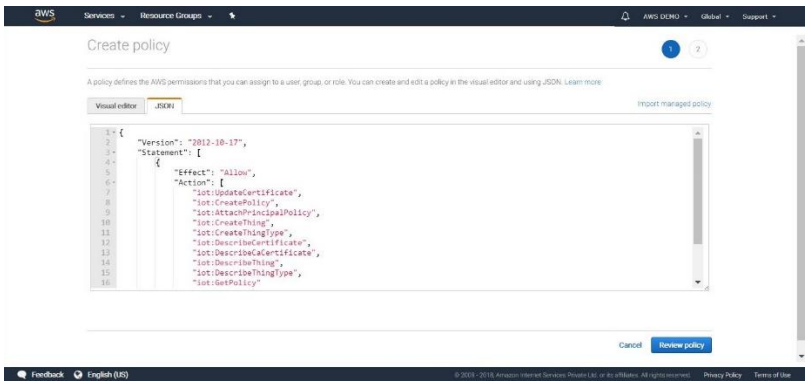


1. Select **JSON**



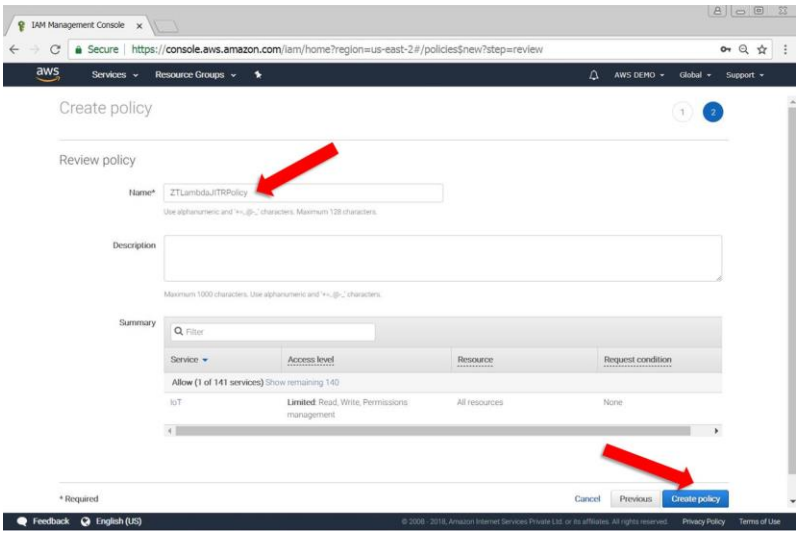
2. Copy the below text and place it as shown in below figure and click **Review policy**

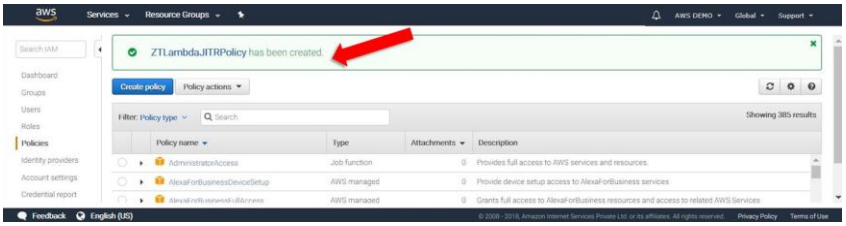
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:UpdateCertificate",
                "iot:CreatePolicy",
                "iot:AttachPrincipalPolicy",
                "iot:CreateThing",
                "iot:CreateThingType",
                "iot:DescribeCertificate",
                "iot:DescribeCaCertificate",
                "iot:DescribeThing",
                "iot:DescribeThingType",
                "iot:GetPolicy"
            ],
            "Resource": "*"
        }
    ]
}
```



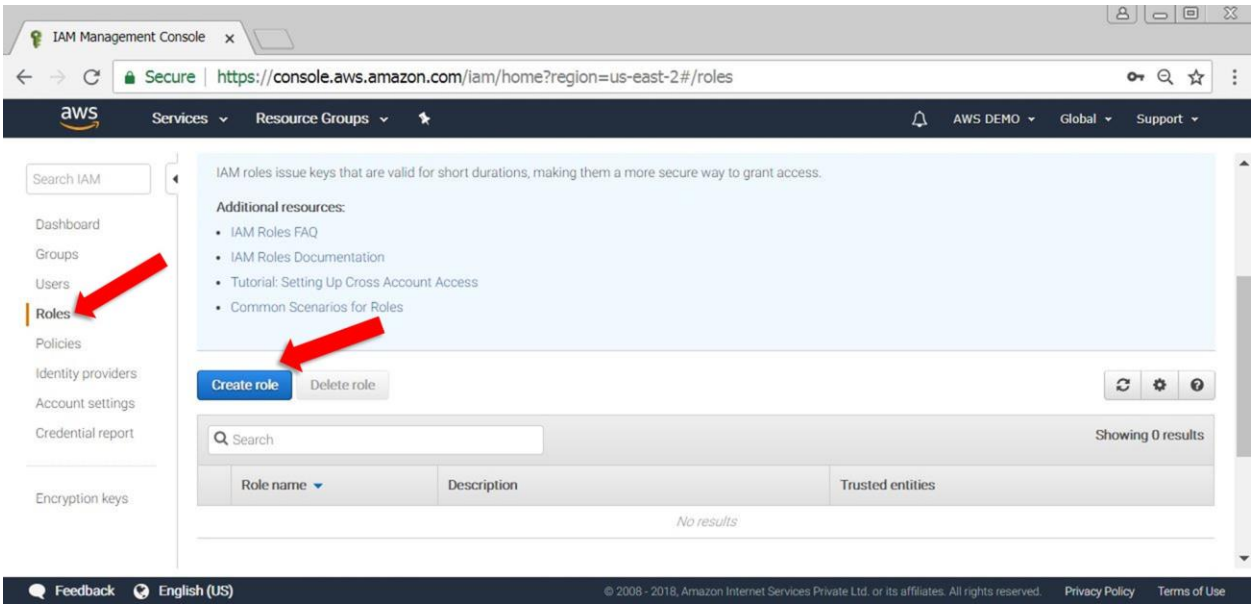
3. Name for the Policy
   Name: **ZTLambdaJITRPolicy**

   And click **Create Policy**

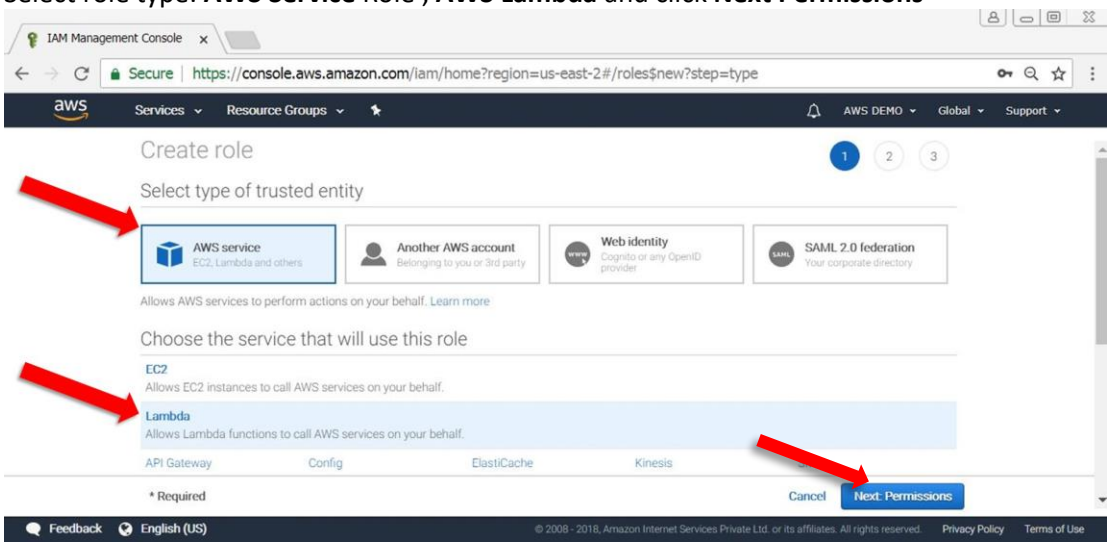




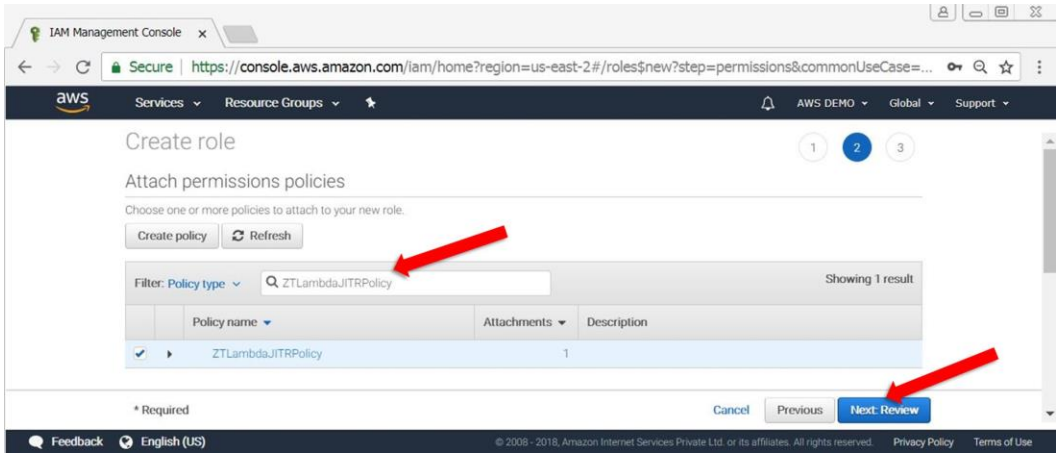
e. Create the JITR Lambda Function Role. Click "***Roles***" then "***Create role***"

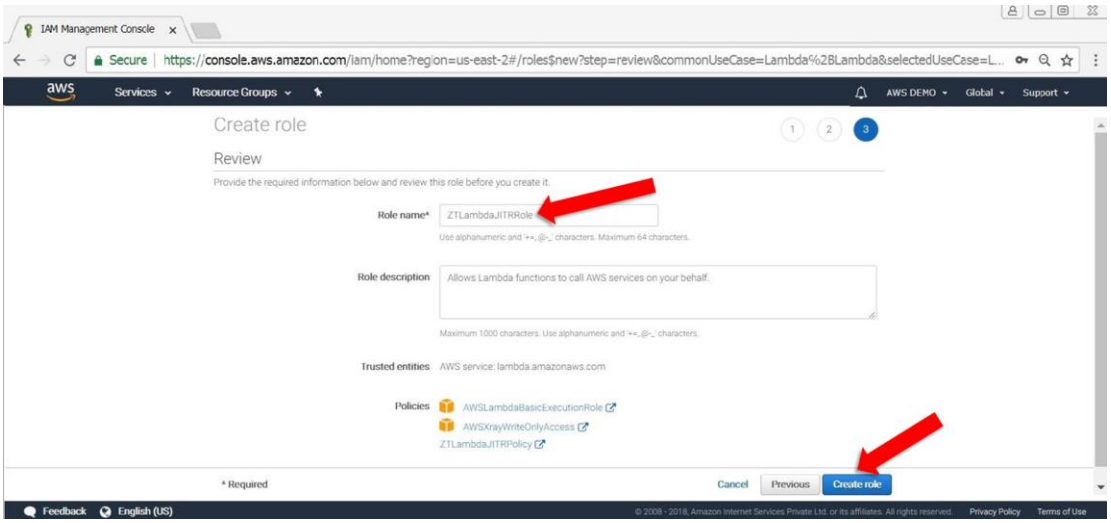


1. Select role type: **AWS Service** Role , **AWS Lambda** and click **Next Permissions**



2. Attach the following policies:
   Type the following in the policy type as shown in below figure and select it.
   - **AWSLambdaBasicExecutionRole**
   - **AWSXrayWriteOnlyAccess**
   - **ZTLambdaJITRPolicy**
3. Click **Next Review**
4. Role Name: **ZTLambdaJITRRole**
5. Click **Create role**

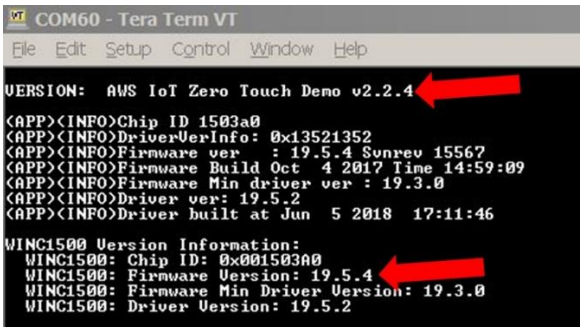




Appendix B: Update Development Board Firmware

here's a possibility you many need to update older firmware on the SAMG55 Xplained Pro and/or the WINC1500 Xplained Pro board(s). When you connect the SAMG55 Xplained Pro board to your computer via a terminal emulator (e.g.: PuTTY), the firmware version numbers will be displayed. If the firmware version is older than these, you will need to update the firmware:

- AWS IoT Zero Touch Demo (SAMG55 Xplained Pro): **v2.2.4**
- WINC1500: Firmware Version (WINC1500 Xplained Pro): **v19.5.4**



If a firmware update is needed, you will need to install the Atmel Studio 7 IDE:
Atmel Studio 7 http://www.microchip.com/development-tools/atmel-studio-7

1. WINC1500 Xplained Pro Firmware:

To update the WINC1500 firmware:

a. Open Atmel Studio 7 and select **New Example Project…** from the Start Page
b. In the ASF Example Dialog, enter **WINC1500 Firmware** into the search bar
c. Select the **WINC1500 Firmware Update Project (v19.5.4) - SAMG55 Xplained Pro** project and click OK to create it
d. Plug the WINC1500 Xplained Pro into EXT1 of the SAMG55 Xplained Pro
e. Plug the SAMG55 Xplained Pro into the computer via the EDBG USB Port

f. In Atmel Studio, under the Solution Explorer, expand the **src** folder
g. **Right-click** on samg55_xplained_pro_firmware_update.bat and select **Open File Location**
h. From the explorer window that opens, run the **samg55_xplained_pro_firmware_update.bat** script to update the WINC1500 firmware.

2. SAMG55 Xplained Pro Firmware:

To update the SAMG55 firmware:

a. Open Atmel Studio 7 and open the zero touch firmware solution, AWS_IoT_Zero_Touch_SAMG55.atsln
b. Plug the SAMG55 Xplained Pro into the computer via the EDBG USB Port
c. Within Atmel Studio, using the **Debug -> Start Without Debugging** menu option to rebuild and load the firmware onto the board.