How to Bridge Mosquitto MQTT Broker to AWS IoT

The Internet of Things on AWS – Official Blog

How to Bridge Mosquitto MQTT Broker to AWS IoT

by Michael Garcia | on 18 AUG 2016 | in AWS IoT, MQTT | Permalink

You can connect securely millions of objects to AWS IoT using our AWS SDKs or the AWS IoT Device SDKs. In the context of industrial IoT, objects are usually connected to a gateway for multiple reasons: sensors can be very constrained and not able to directly connect to the cloud, sensors are only capable of using other protocols than MQTT or you might might need to perform analytics and processing locally on the gateway.

One feature of local MQTT broker is called 'Bridge' and will enable you to connect your local MQTT broker to AWS IoT so they can exchange MQTT messages. This will enable your objects to communicate in a bi-directional fashion with AWS IoT and benefit from the power of the AWS Cloud.

In this article we are going to explain use cases where this feature can be very useful and show you how to implement it.

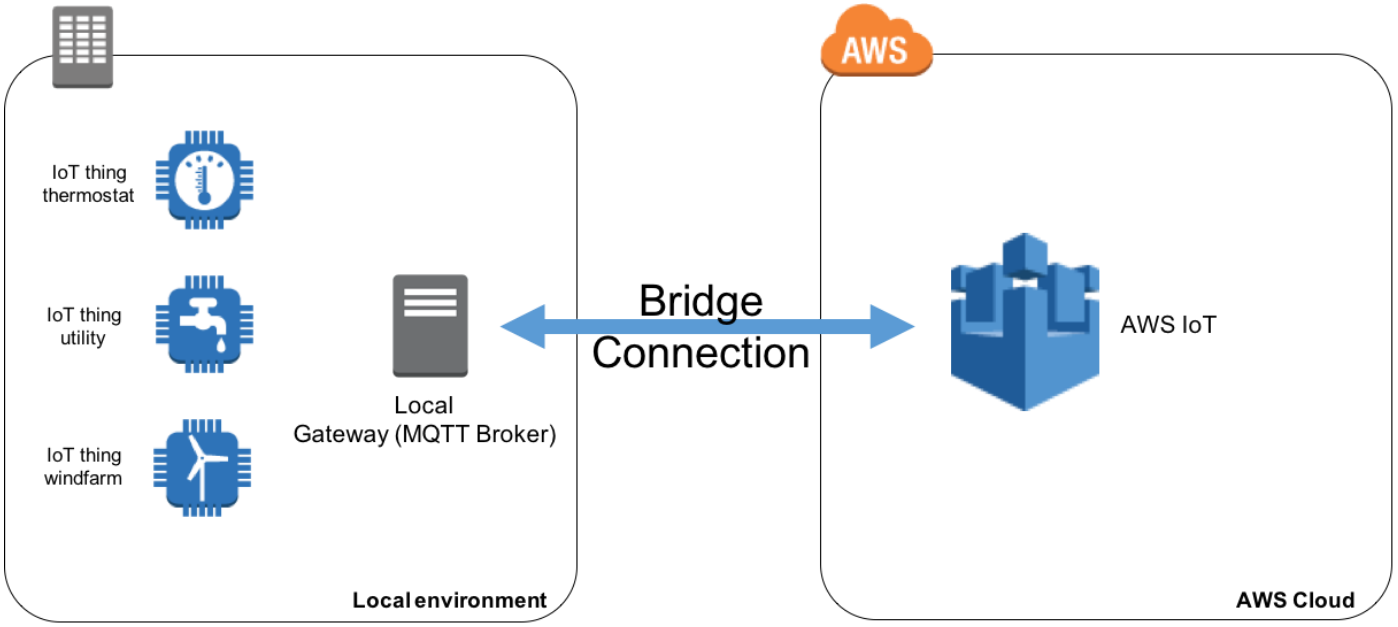
Why Bridge your MQTT Broker to AWS IoT

Security is paramount in IoT and the AWS IoT broker has a high level of security built-in to authenticate and authorize devices base on standards like TLS 1.2 with client certificates.

If you have legacy IoT deployments, you might already have objects connected to an MQTT broker using other authentication mechanism like username and passwords. Your MQTT broker can be very close to where your sensors are deployed (local MQTT broker) or in a remote location like the Cloud.

If you plan to upgrade your current security standards to match those of AWS IoT but want to benefit from the scalability and Rule Engine of AWS IoT today, you can bridge your legacy MQTT broker to AWS IoT. This represents an easy transient solution that you can deploy quickly without having to wait for your current system's upgrade. Scaling beyond a single broker is not in the scope of this post, we will focus on the bridging feature of Mosquitto MQTT Broker.

Open source MQTT broker like Mosquitto can be installed on many operating systems like Linux for example. For those wishing to deploy a local gateway quickly without developing extra code to send data to AWS IoT, installing Mosquitto on a local device can represent an attractive solution as well as you will benefit locally from Mosquitto boker's features (persist messages locally, log activity locally, …).



How to Install Mosquitto MQTT Broker

The first step will be to install Mosquitto broker on your device/virtual machine, you can go to Mosquitto download page for instructions.

Typically, you should install this on your local gateway. Mosquitto supports a wide range of platforms including many distributions of Linux. Therefore, you can run your local gateway on low powered devices as well as on a full-fledged server/virtual machine.

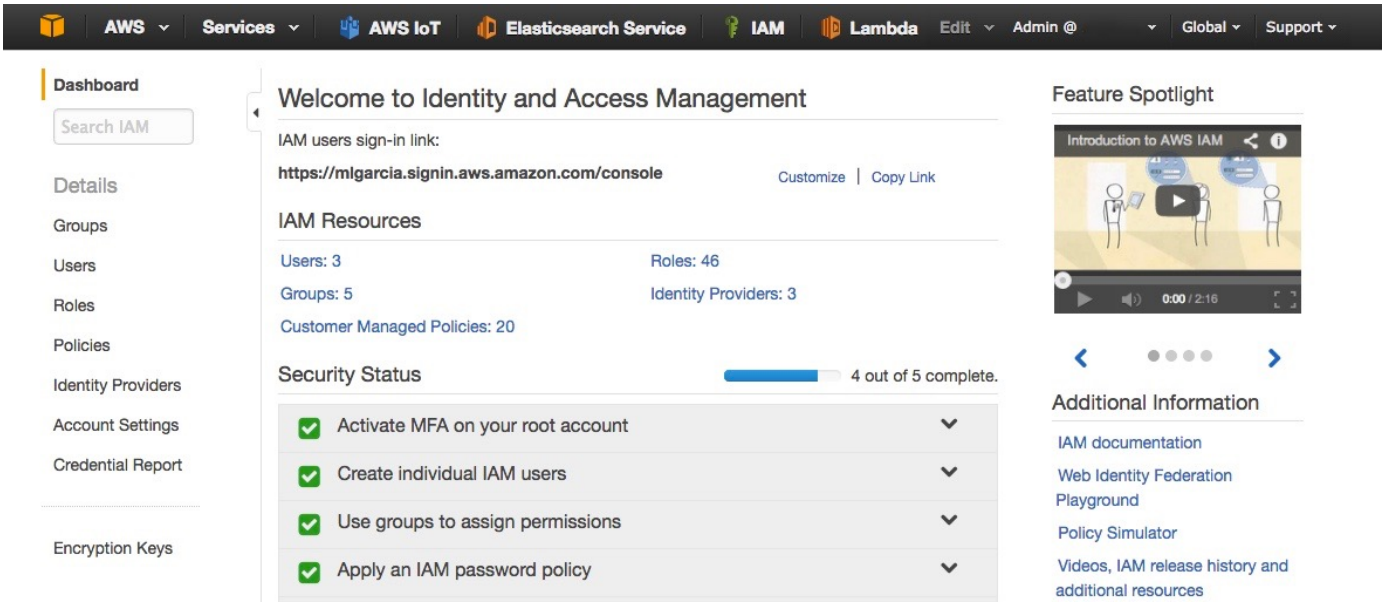
In our case we will install Mosquitto on an EC2 Amazon Linux instance which would be equivalent to having a local gateway running a Linux distribution.

If you are not planning on using an Amazon EC2 Instance you can skip to the section “How to configure the bridge to AWS IoT”

Launching and Configuring the EC2 Instance

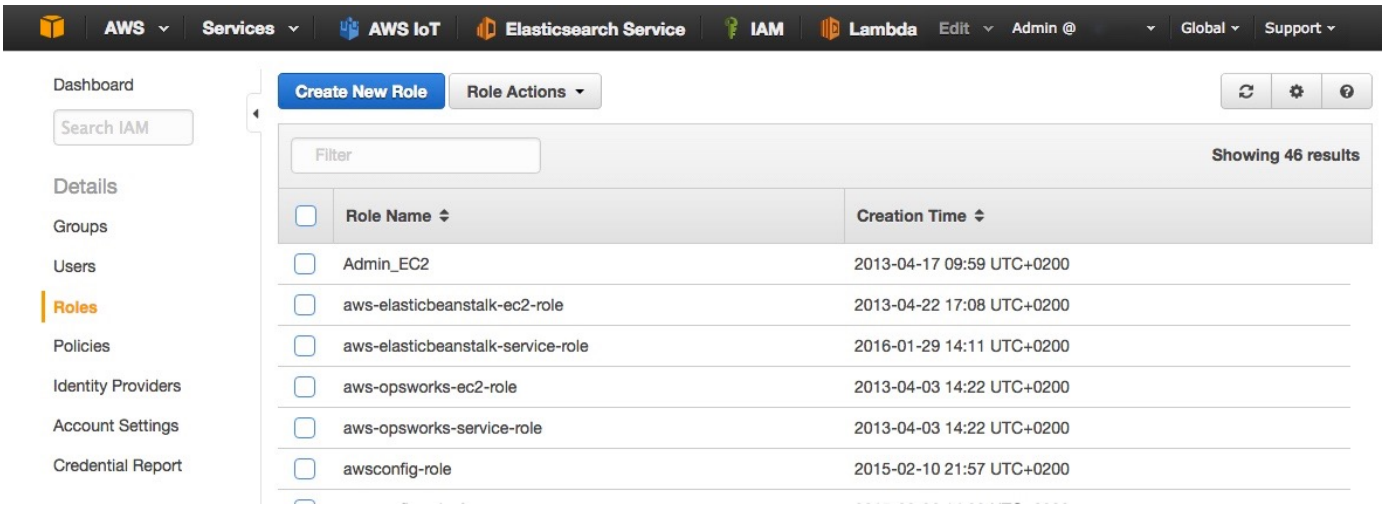
Before launching an EC2 Amazon Linux instance to host the Mosquitto broker, we are going to create an IAM Role so we’ll be able to use the CLI on the instance to create keys and certificate in AWS IoT for the bridge.

1. Go to the AWS Web Console and access the IAM service (Fig. 1)

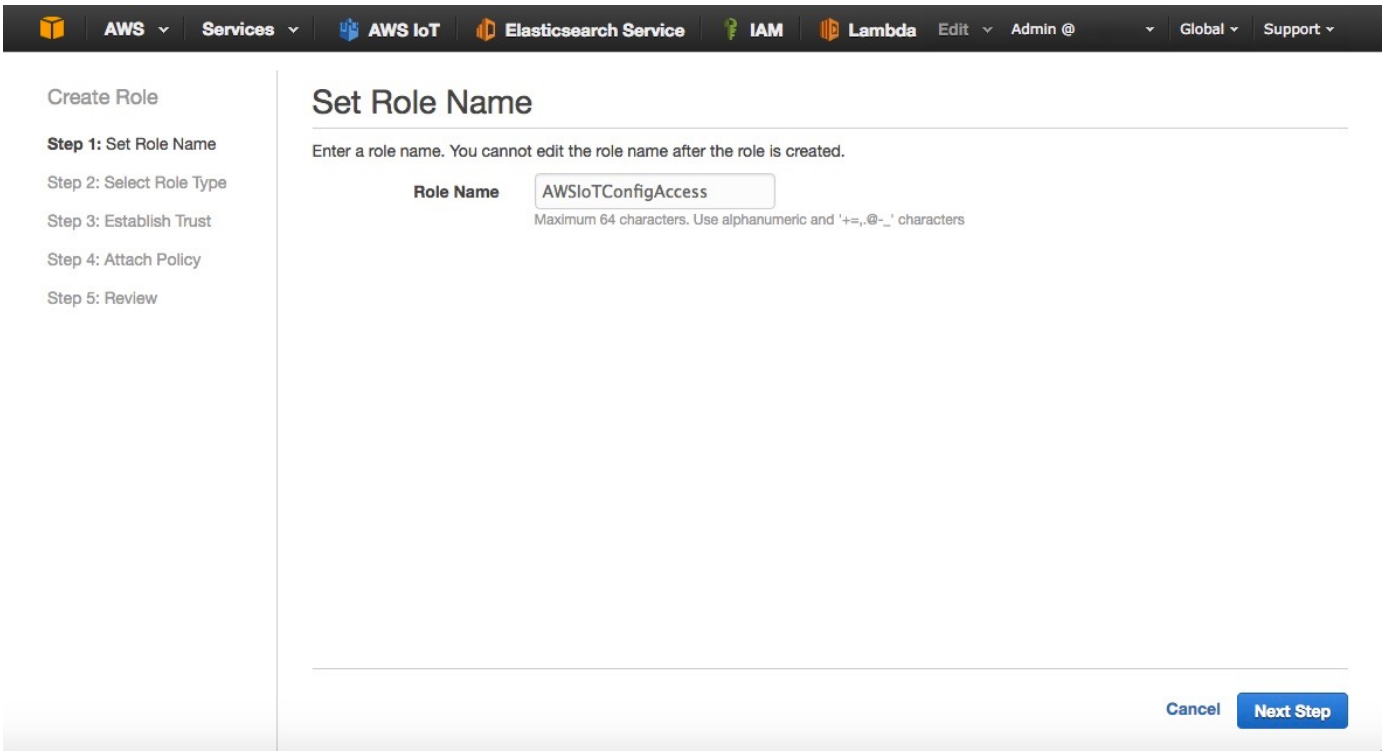


2. Click on Roles

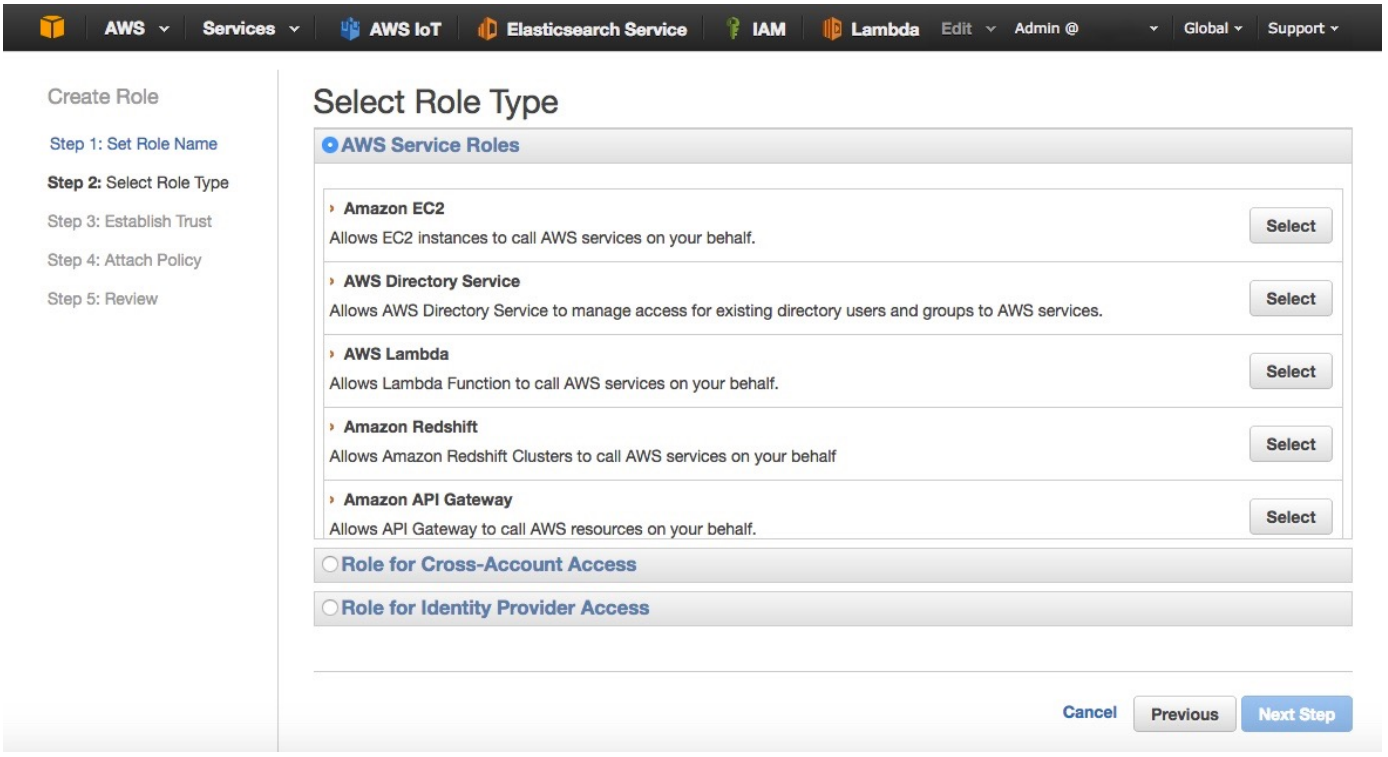
3. Click on Create New Role (Fig. 2)



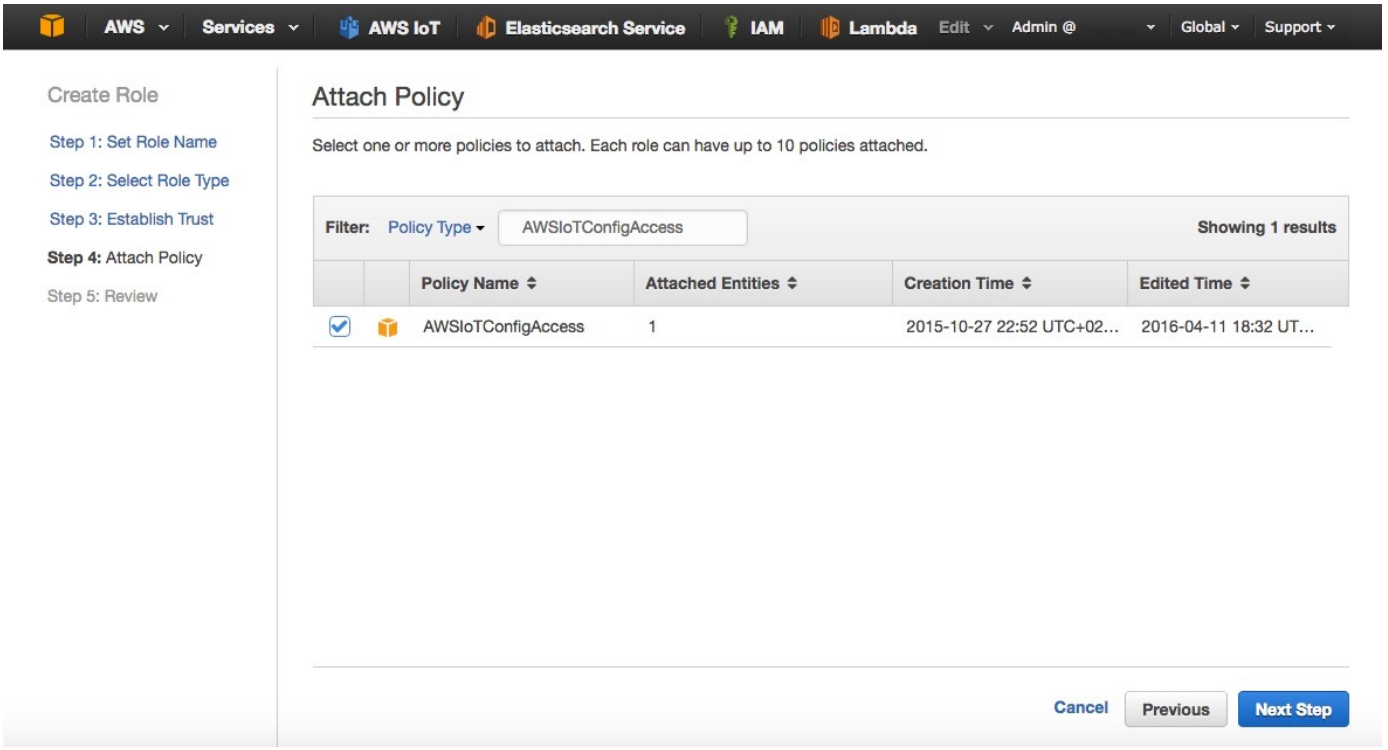
4. Name the role AWSIoTConfigAccess (Fig. 3)



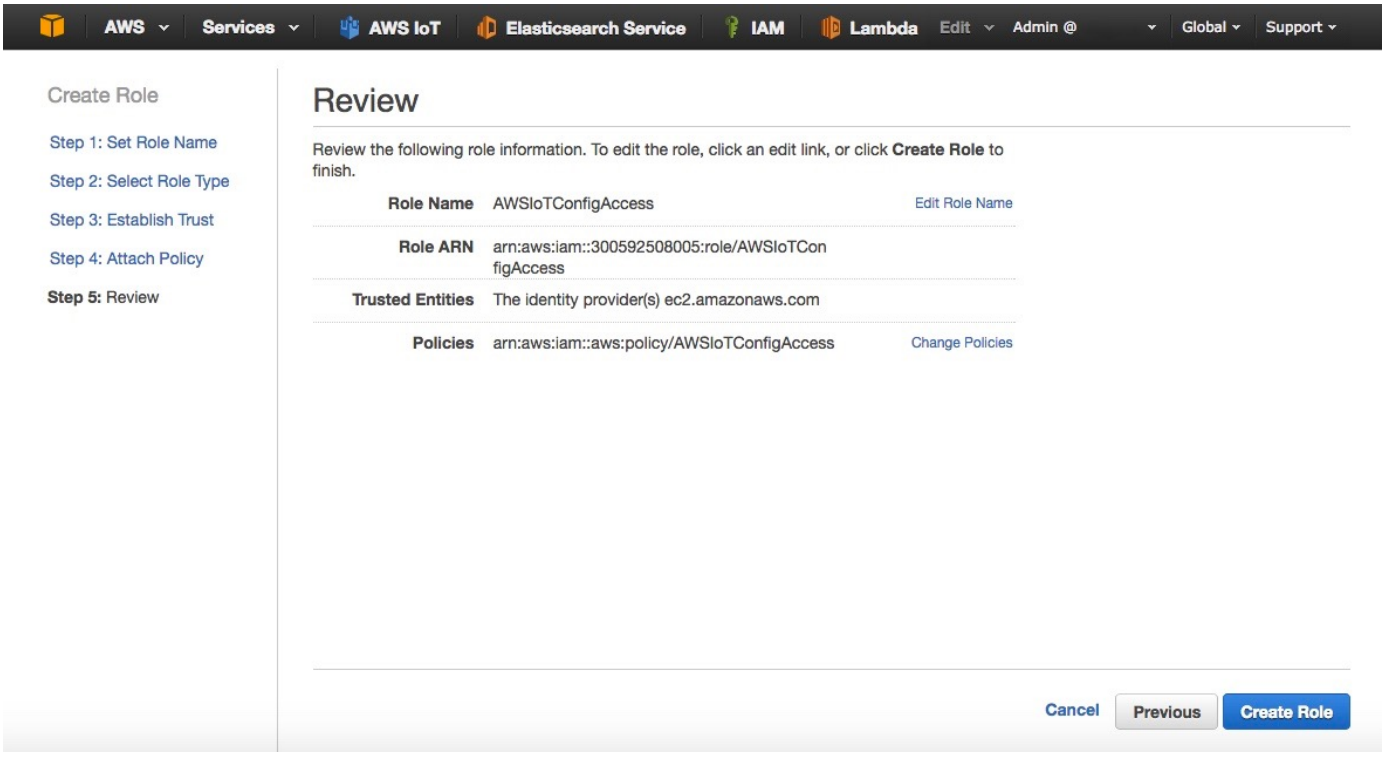
5. Click Next Step
6. Select Amazon EC2 (Fig. 4)



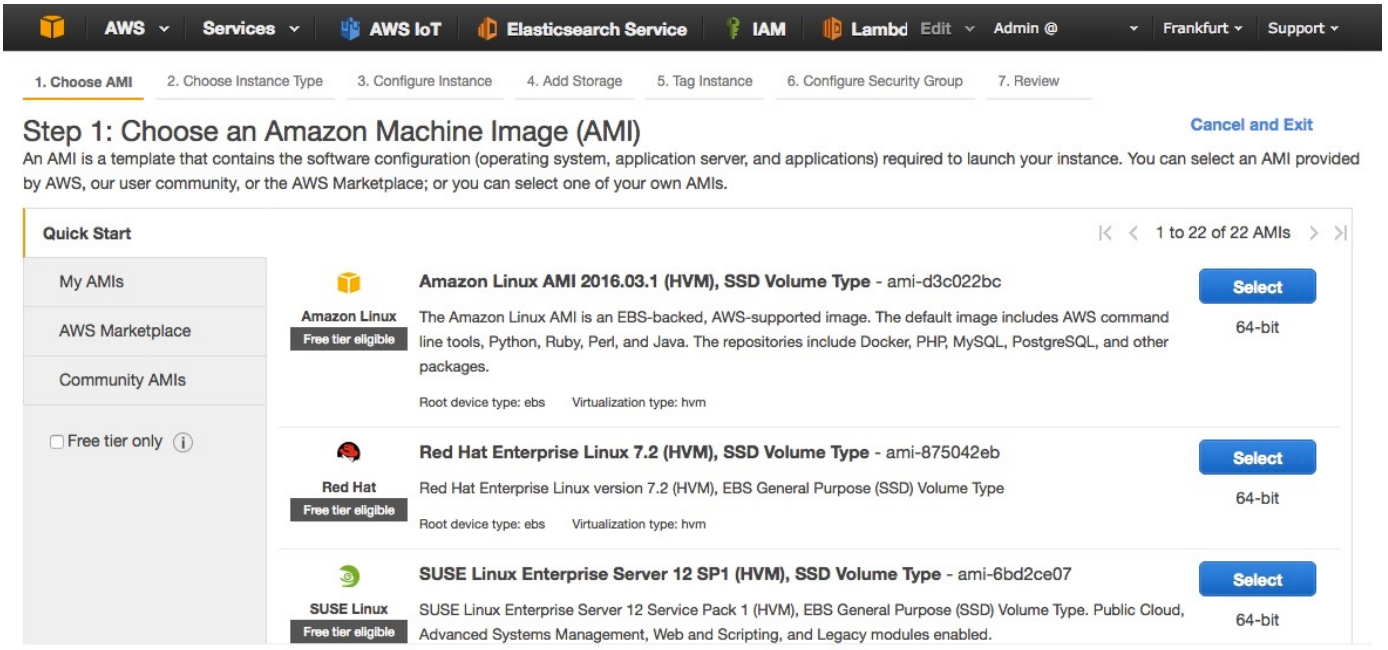
7. Filter with the value AWSIoTConfigAccess (Fig. 5)



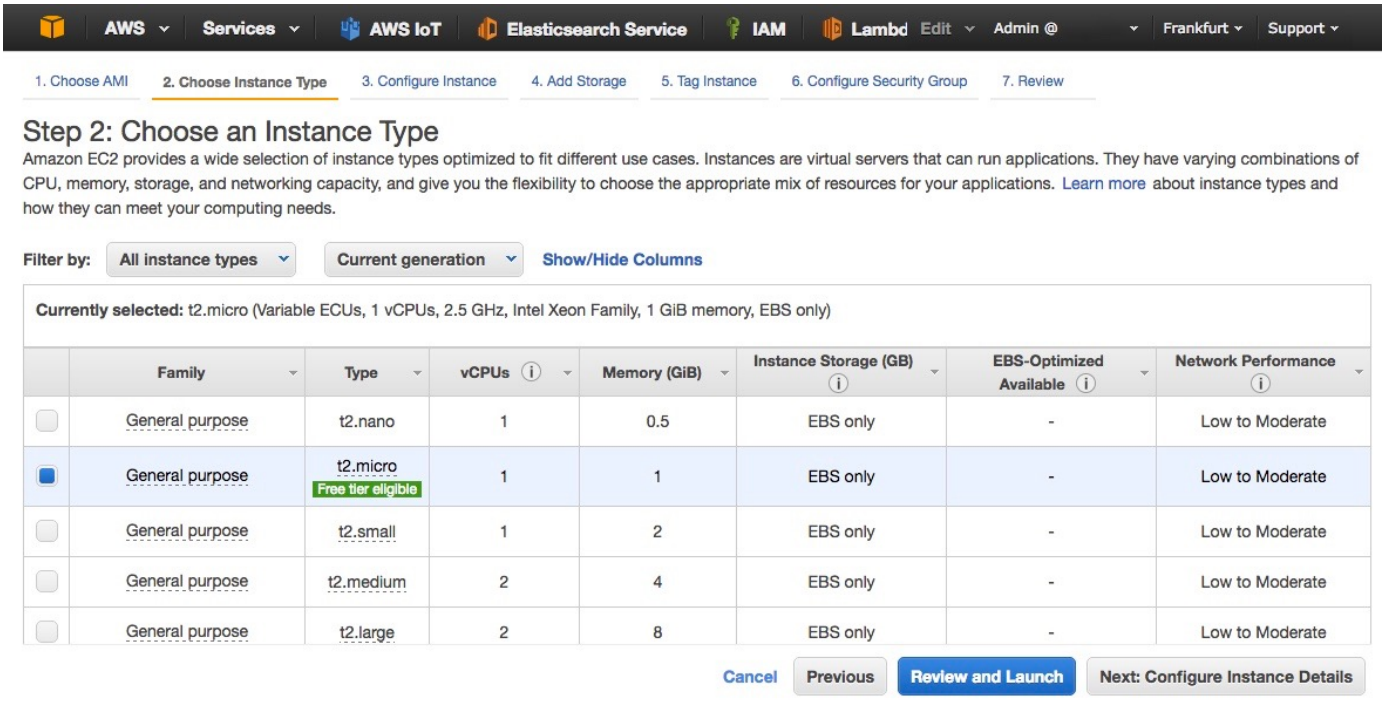
8. Select the policy AWSIoTConfigAccess and click on Next Step
9. Review the Role and click on Create Role (Fig. 6)



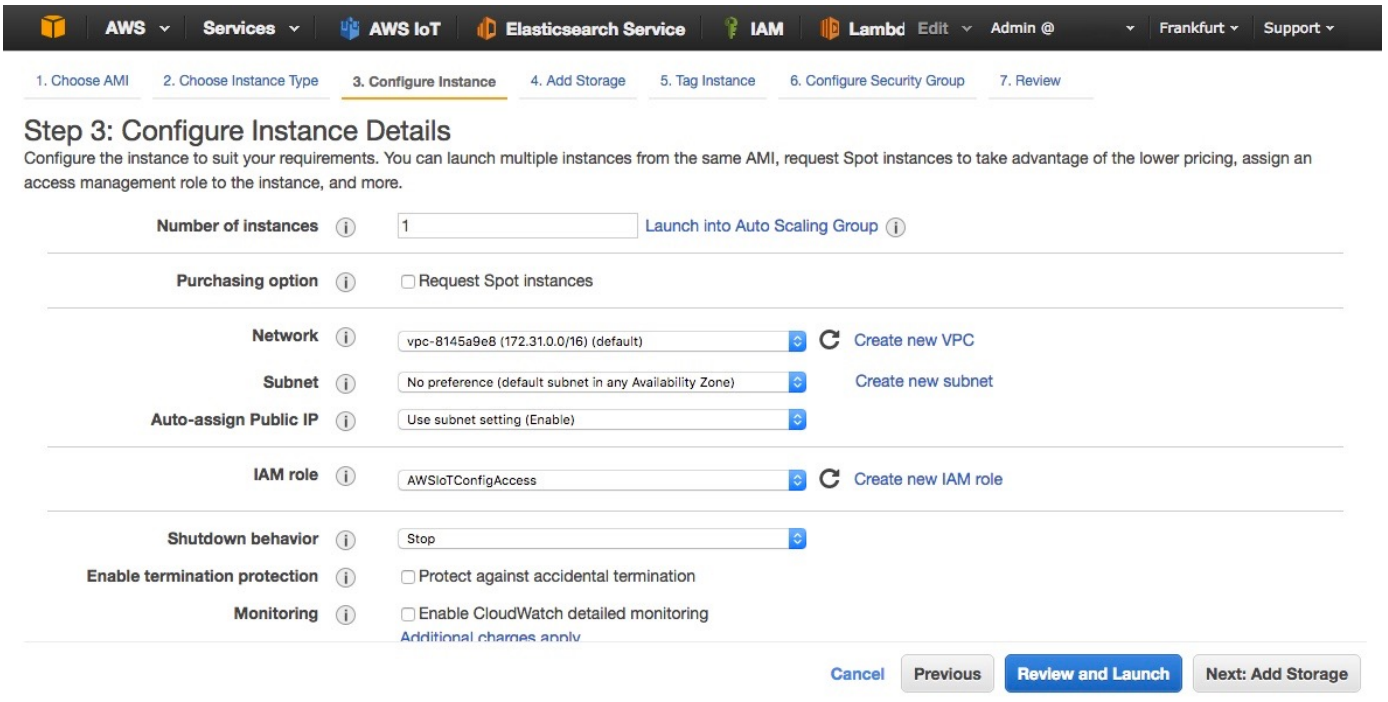
10. Now that the Role has been created you can go to Amazon EC2. Choose a region, preferably where AWS IoT is available, in this article I am using Frankfurt.
11. Click on Launch Instance.
12. Select Amazon Linux AMI 2016.03.1 (Fig. 7)



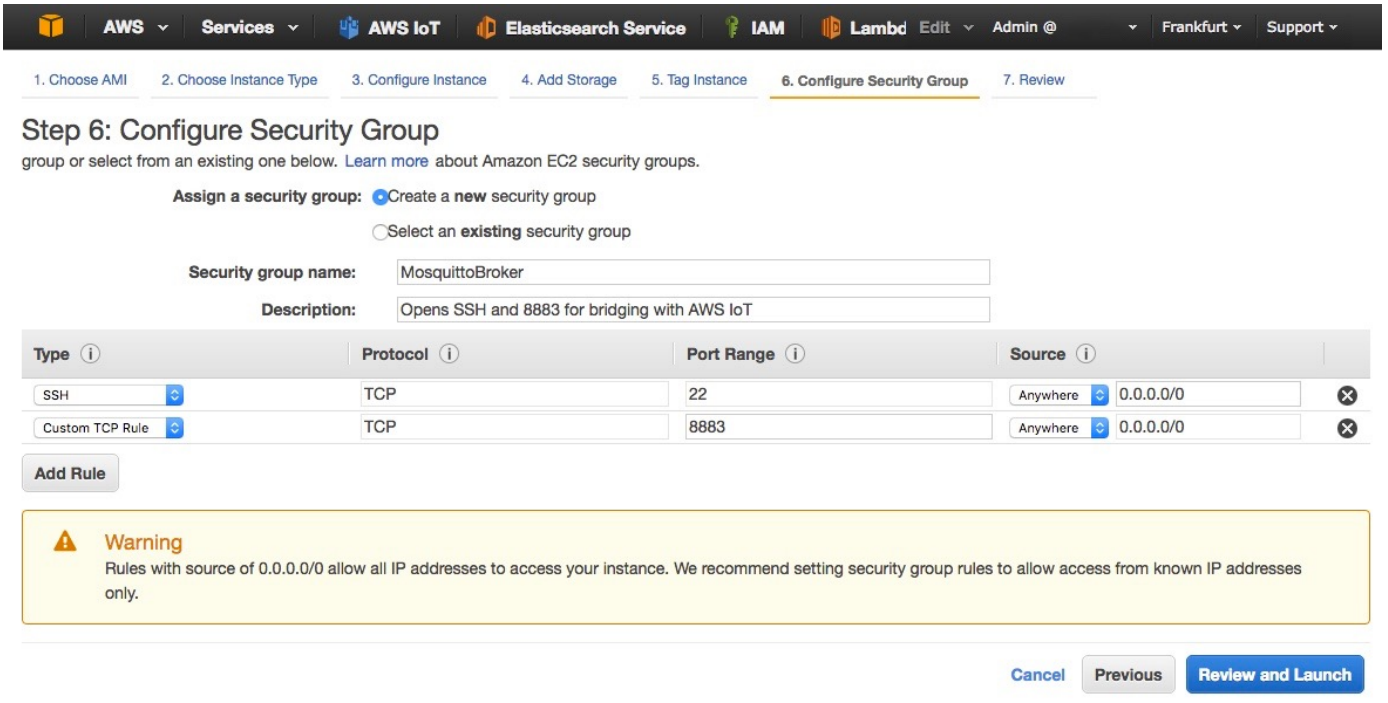
13. Select the t2.micro instance type (Fig. 8)



14. Click on Next: Configure Instance Details
15. In the IAM Role, select AWSIoTConfigAccess (Fig. 9)



16. Leave default parameters as shown in the picture and click on Next: Add Storage
17. Leave everything as is and click on Next: Tag Instance
18. Give a name to your instance ‘MosquittoBroker’
19. Click on Next: Configure Security Groups
20. Create a new security group (Fig. 10)



21. Review and launch the EC2 instance
22. Follow instructions to connect to the EC2 instance once it is running.
23. Once logged in type the following commands:

```
#Update the list of repositories with one containing Mosquitto
sudo wget http://download.opensuse.org/repositories/home:/oojah:/mqtt/CentOS_Cent
OS-7/home:oojah:mqtt.repo -O /etc/yum.repos.d/mqtt.repo
#Install Mosquitto broker and Mosquitto command line tools
sudo yum install mosquitto mosquitto-clients
```

How to Configure the Bridge to AWS IoT

Now that we have installed Mosquitto onto our EC2 instance (or local gateway), we will need to configure the bridge so that the Mosquitto broker can create a connection to AWS IoT. We will first use the AWS CLI to create the necessary resources on AWS IoT side.

Enter the following commands in your terminal:

```
#Configure the CLI with your region, leave access/private keys blank
aws configure

#Create an IAM policy for the bridge
aws iot create-policy --policy-name bridge --policy-document '{"Version": "2012-1
0-17","Statement": [{"Effect": "Allow","Action": "iot:*","Resource": "*"}]}'

#Place yourself in Mosquitto directory
#And create certificates and keys, note the certificate ARN
cd /etc/mosquitto/certs/
sudo aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfil
e cert.crt --private-key-outfile private.key --public-key-outfile public.key –reg
ion eu-central-1

#List the certificate and copy the ARN in the form of
# arn:aws:iot:eu-central-1:0123456789:cert/xyzxyz
```

```
aws iot list-certificates

#Attach the policy to your certificate
aws iot attach-principal-policy --policy-name bridge --principal <ARN_OF_CERTIFIC
ATE>

#Add read permissions to private key and client cert
sudo chmod 644 private.key
sudo chmod 644 cert.crt

#Download root CA certificate
sudo wget https://www.symantec.com/content/en/us/enterprise/verisign/roots/VeriSi
gn-Class%203-Public-Primary-Certification-Authority-G5.pem -O rootCA.pem
```

We now have a client certificate for our bridge, this certificate is associated with an IAM policy that will give all permissions to the bridge (this policy must be restricted for your usage). The bridge will have everything it needs to connect, we just need to edit the configuration file with our specific parameters for Mosquitto.

```
#Create the configuration file
sudo nano /etc/mosquitto/conf.d/bridge.conf
```

Edit the following by replacing the value address with your own AWS IoT endpoint. You can use the AWS CLI to find it with ‘aws iot describe-endpoint’ as mentioned below. Then copy the content and paste it in the nano editor, finally save the file.

```
#Copy paste the following in the nano editor:
# ================================================================
# Bridges to AWS IOT
# ================================================================

# AWS IoT endpoint, use AWS CLI 'aws iot describe-endpoint'
connection awsiot
address XXXXXXXXXX.iot.eu-central-1.amazonaws.com:8883

# Specifying which topics are bridged
topic awsiot_to_localgateway in 1
topic localgateway_to_awsiot out 1
topic both_directions both 1

# Setting protocol version explicitly
bridge_protocol_version mqttv311
bridge_insecure false

# Bridge connection name and MQTT client Id,
# enabling the connection automatically when the broker starts.
cleansession true
clientid bridgeawsiot
start_type automatic
notifications false
log_type all
```

```
# ================================================================
# Certificate based SSL/TLS support
# ----------------------------------------------------------------
#Path to the rootCA
bridge_cafile /etc/mosquitto/certs/rootCA.pem

# Path to the PEM encoded client certificate
bridge_certfile /etc/mosquitto/certs/cert.crt

# Path to the PEM encoded client private key
bridge_keyfile /etc/mosquitto/certs/private.key
```

Now we can start the Mosquitto broker with this new configuration:

```
#Starts Mosquitto in the background
sudo mosquitto -c /etc/mosquitto/conf.d/bridge.conf –d
#Enable Mosquitto to run at startup automatically
sudo chkconfig --level 345 scriptname on
```

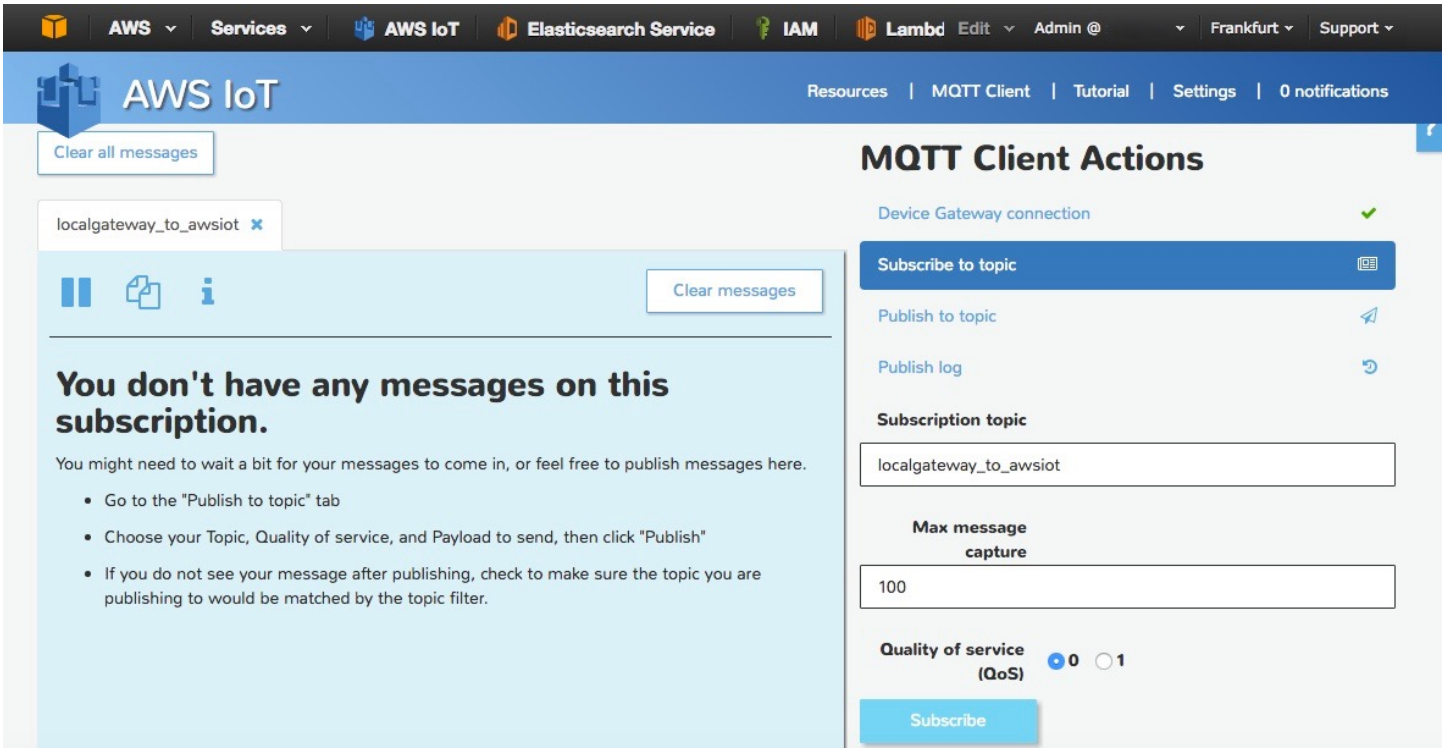
Making Sure Everything is Working

The broker has now started and has already connected to AWS IoT in the background. In our configuration we have bridged 3 topics:

- awsiot_to_localgateway: any message received by AWS IoT from this topic will be forwarded to the local gateway
- localgateway_to_awsiot: any message received by the local gateway will be forwarded to AWS IoT
- both_directions: any message received on this topic by one broker will be forwarded to the other broker

We will check that the topic localgateway_to_awsiot is working, feel free to check the whole configuration.

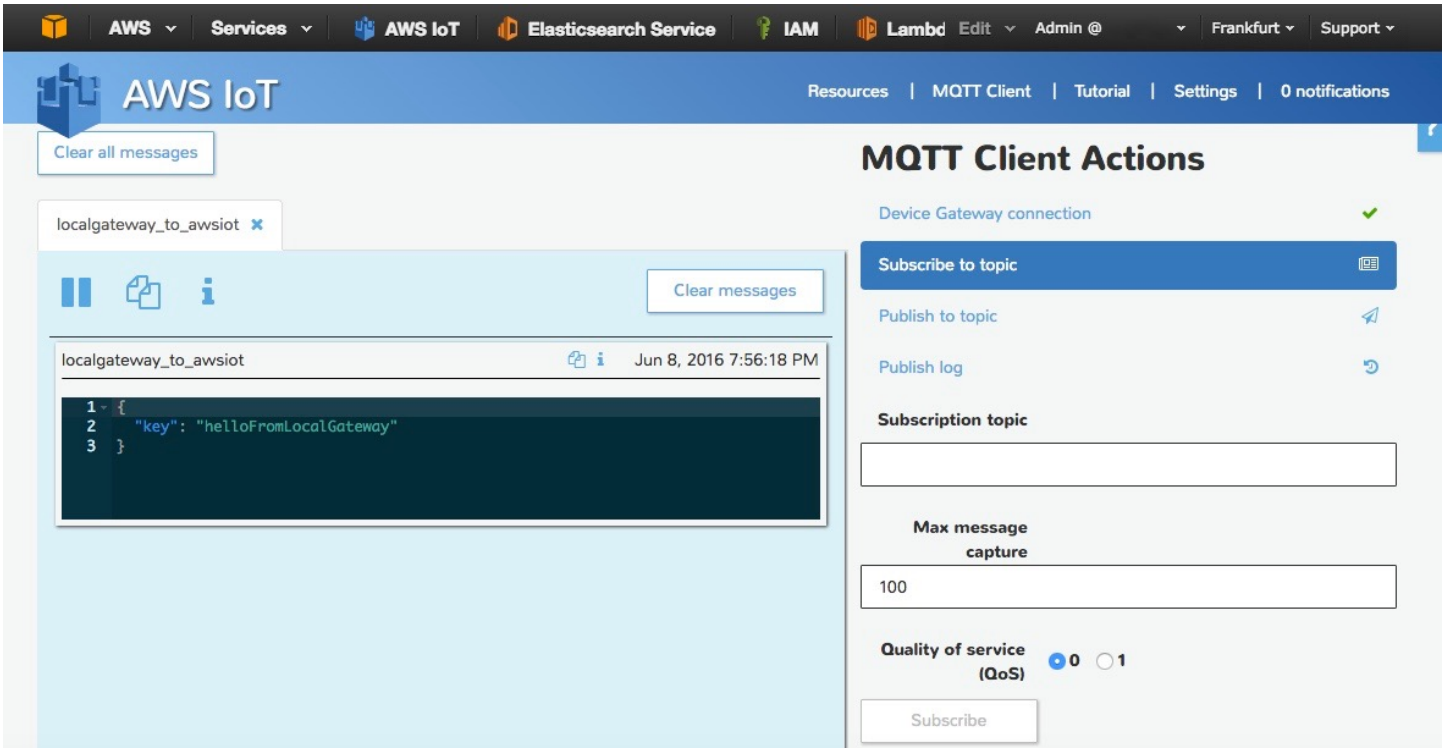
- Go to the AWS IoT Console and click on MQTT Client
- Click on Generate Client Id and Connect
- Click on Subscribe to topic and enter localgateway_to_awsiot, click on Subscribe (Fig. 11)/>



Now that we have subscribed to this topic on AWS IoT side you can publish an MQTT message from your terminal (so from the local gateway) to see if it gets forwarded.

```
#Publish a message to the topic
mosquitto_pub -h localhost -p 1883 -q 1 -d -t localgateway_to_awsiot  -i clientid
1 -m "{\"key\": \"helloFromLocalGateway\"}"
```

You should now get this message on your screen, delivered by AWS IoT thanks to the bridge.



If you are done testing with an Amazon EC2 Instance you can do this with your own local/remote MQTT broker!

Next Steps

The bridge between your local broker and AWS IoT is up and running, you might want to fine tune some parameters of the bridge connection. Please consult the Bridge section of the official Mosquitto documentation if you need additional details.

Now that your data is flowing through AWS IoT you can create new IoT applications using other AWS Services for Machine Learning,Analytics, Real-Time Dashboarding and much more so do not hesitate to read our blog, documentation and additional developer resources.

How to bridge between Mosquitto MQTT and AWS IoT MQTT Broker (to publish openHAB Smarthome data to the Cloud)

Jens Ihnow's Blog

montag, 26. oktober 2015

HOW TO BRIDGE BETWEEN MOSQUITTO MQTT AND AWS IOT MQTT BROKER (TO PUBLISH OPENHAB SMARTHOME DATA TO THE CLOUD)

AWS IoT recently launched in beta for everyone so I started poking around with the AWS IoT Button and the MQTT Broker. At some point I got the idea to connect my existing openHAB Smarthome control system to AWS IoT, to persist the sensor data of my house and/or integrate with the AWS IoT Button. Unfortunately the openHAB MQTT bindings do not support TLS (following the docs) and on the other hand I already had a running Mosquitto MQTT broker. Therefore the idea of bridging messages from one to another broker was born.

This is a short tutorial to setup a bridge between a Mosquitto broker and the AWS IoT broker. There is no need to have openHAB running at all, this is just my use case which I will explain in another blog a bit more. Let's focus on setting up the bridge now:

For any connection to the AWS IoT Broker you need a client, what basically means a set of certificates. The whole process is explained in detail on AWS IoT Quickstart.

First I'm going to create a new device which is used by the bridge to connect to the broker. You should give it self explaining name:

```
#aws iot create-thing --thing-name "
```

Save the thingArn, you need it later.
You can list all of your devices like this to verify is is created:

```
aws iot list-things
{
"things": [
{
"attributes": {},
"thingName": "mosquittobroker"
},
...
]
}
```

Next step is to create the certificates and described on AWS IoT secure communication

I'm going to list the commands here, but you should really understand what is going on because you have to deal with the certificates later.

```
# create and activate a certificate:
aws iot create-keys-and-certificate --set-as-active --output text
```

Save the private and public key as mosquittobroker-private-key.pem and mosquittobroker-public-key.pem

Saving the certificats is easier, because AWS provided ac command for it:

```
aws iot describe-certificate --certificate-id xxx --output text --query
certificateDescription.certificatePem > mosquittobroker-cert.pem
```

Now your certificate needs a IoT policy (NOT IAM!) which you need to create first.
The policy should have the following setup, save it as policy.json:

```
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "Stmt1445601185404",
"Action": [
"iot:Connect",
"iot:Publish",
"iot:Receive",
"iot:Subscribe"
],
"Effect": "Allow",
"Resource": "*"
}
]
```

```
}
```

To create the policy use the command:

```
aws iot create-policy --policy-name "Mosquittobroker" --policy-document
file://./policy.json
```

To attach the policy to the certificate execute the command:

```
# aws iot attach-principal-policy --principal "certificate-arn" --policy-name
"PubSubToAnyTopic"
aws iot attach-principal-policy --principal "arn:aws:iot:us-east-
1:xxx:cert/xxx" --policy-name "Mosquittobroker"
```

To attach the certificate to your thing execute the command:

```
aws iot attach-thing-principal --thing-name "Mosquittobroker" --principal
"arn:aws:iot:us-east-1:xxx:cert/xxx"
```

Verify everything is correct with the following command:

```
mosquitto_sub --cafile ./iot-root-ca.pem --cert ./mosquittobroker-cert.pem --
key ./mosquittobroker-private-key.pem -h "
```

Depending on your system configuration you might need to use --insecure.

If you would like to find out your AWS IoT endpoint, execute:

```
aws iot describe-endpoint
```

The iot-root-ca.pem nees to be downloaded fromhttps://www.symantec.com/content/en/us/enterprise/verisign/roots/VeriSign-Class%203-Public-Primary-Certification-Authority-G5.pem, as described in the documentation.

If the connection is successful everything is ready to go ahead with setting up the bride on your Mosquitto broker. You need at least version 1.4 to support TLS and the MQTT v3.1.1.

On Debian you just need to place a configuration file in /etc/mosquitto/conf.d/ named like *.conf. This file should contain the following settings:

```
# Connection name
connection awsiot

# Host and port of endpoint (your AWS IoT endpoint
address
```

You need to copy the pem files to the correct folder and make them readable for the Mosquitto user.

You need to configure your own topic patterns, see