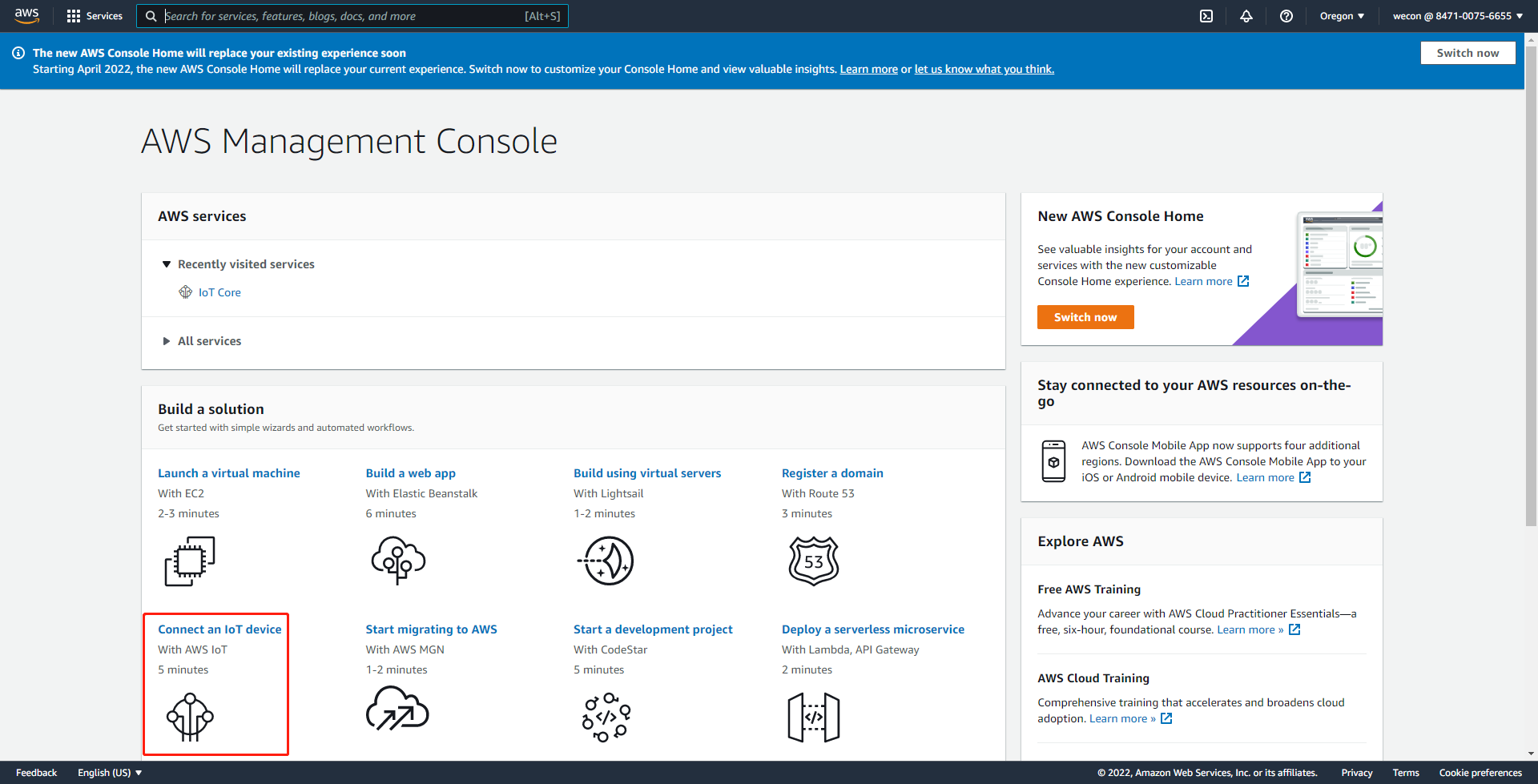
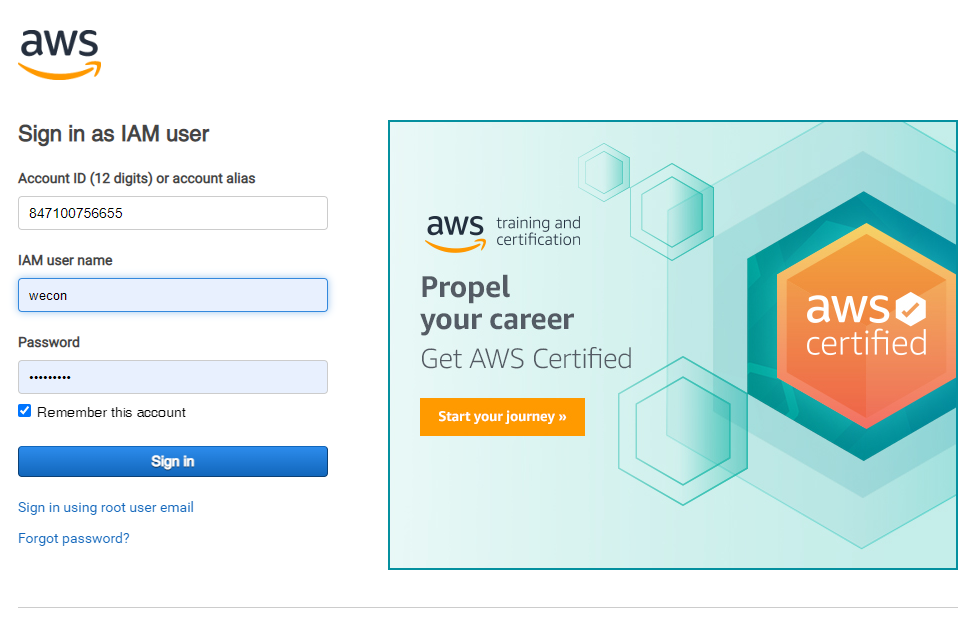


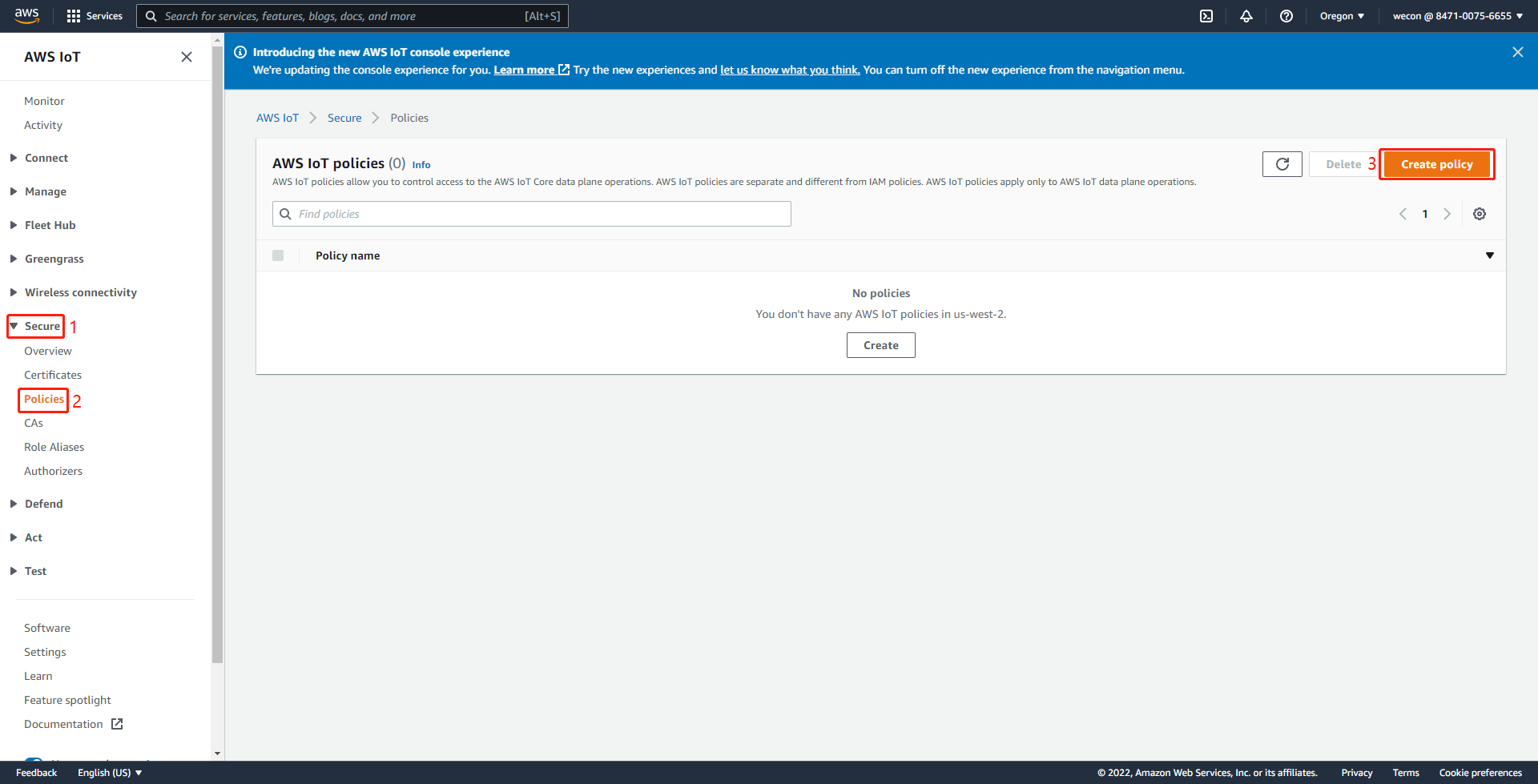
V-Box OpenCloud Mode Connect to AWS IoT

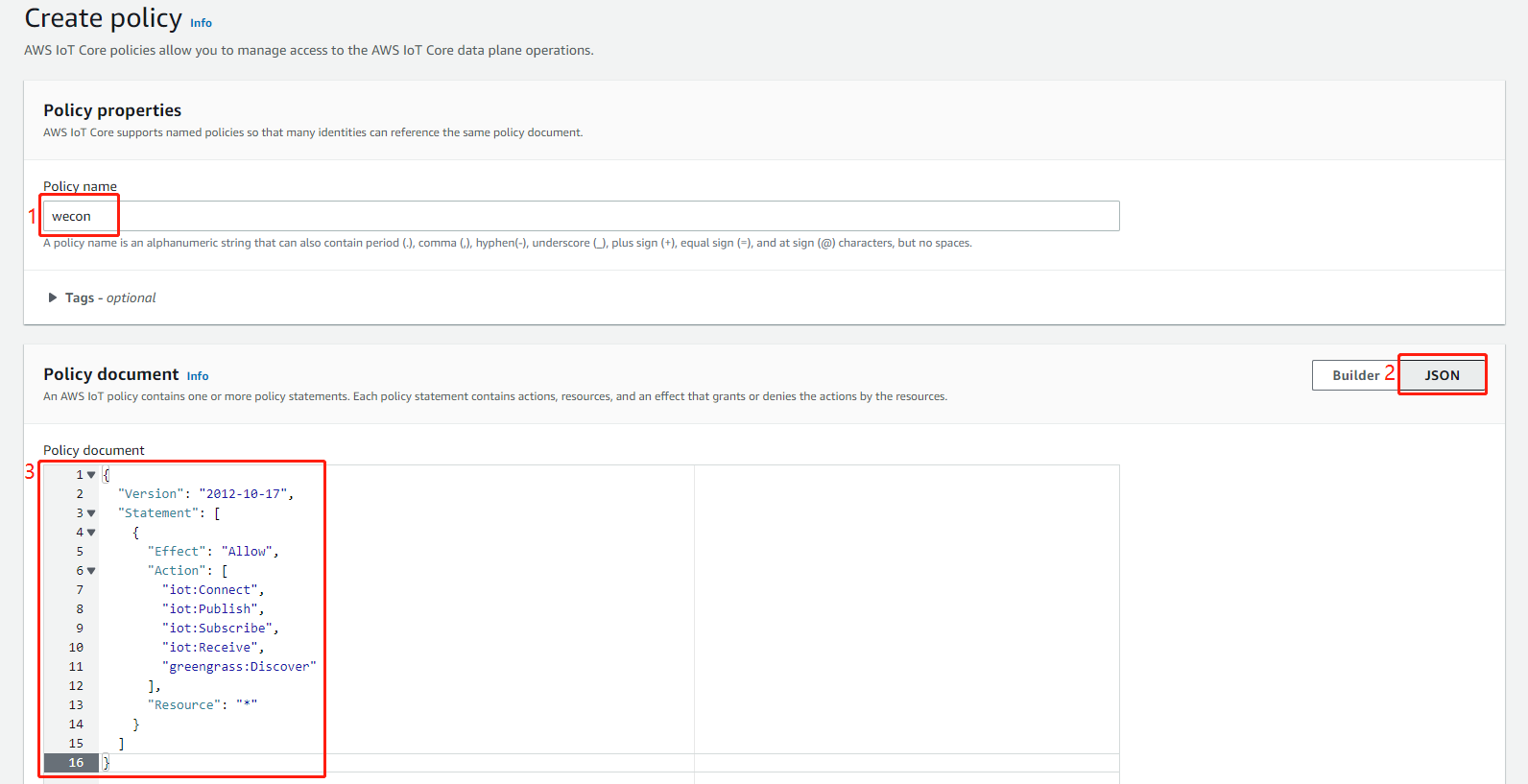
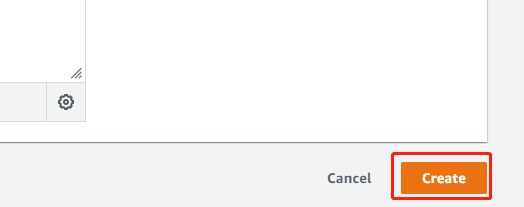
# Log in AWS

Login aws account and click“Connect an IoT device”

# Create policy

Click “Secure”--->“Policies”--->“Create policy”--->Click “Create”

Name the policy--->Click “JSON”--->Copy the following content--->Click “Create”



{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"iot:Connect",

"iot:Publish",

"iot:Subscribe",

"iot:Receive",

"greengrass:Discover"

],

"Resource": "\*"

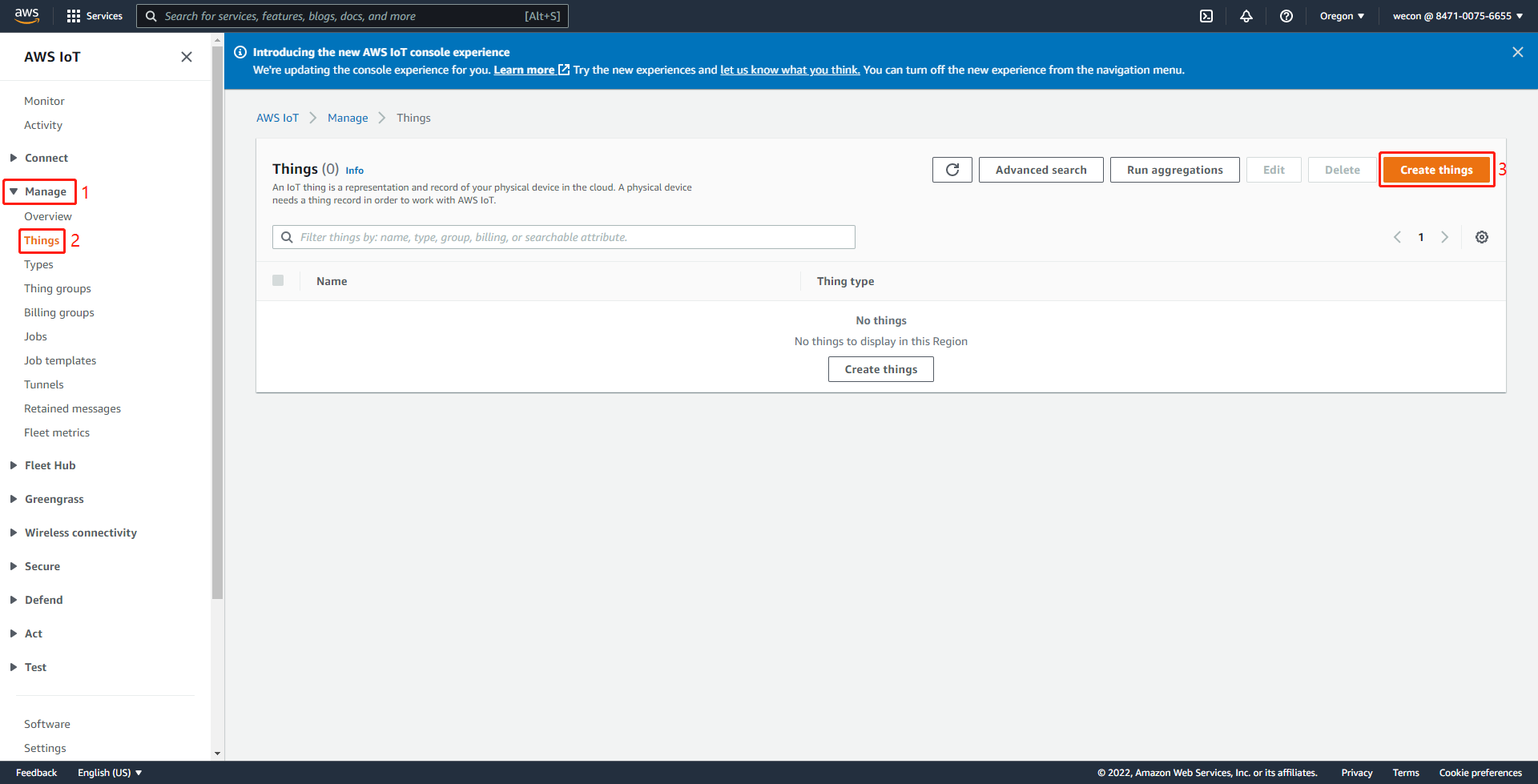
}

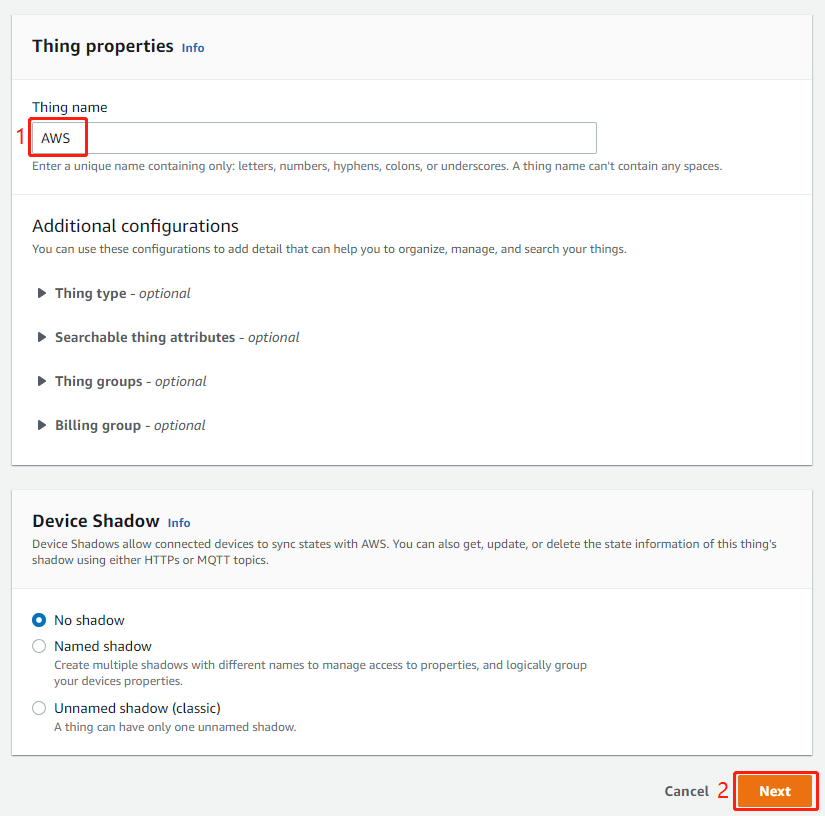
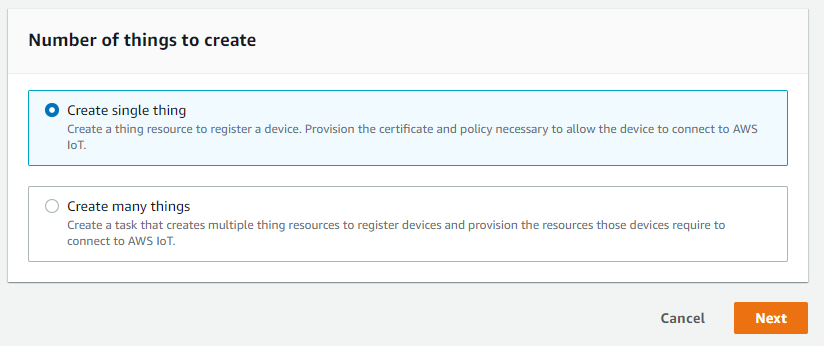
]

}

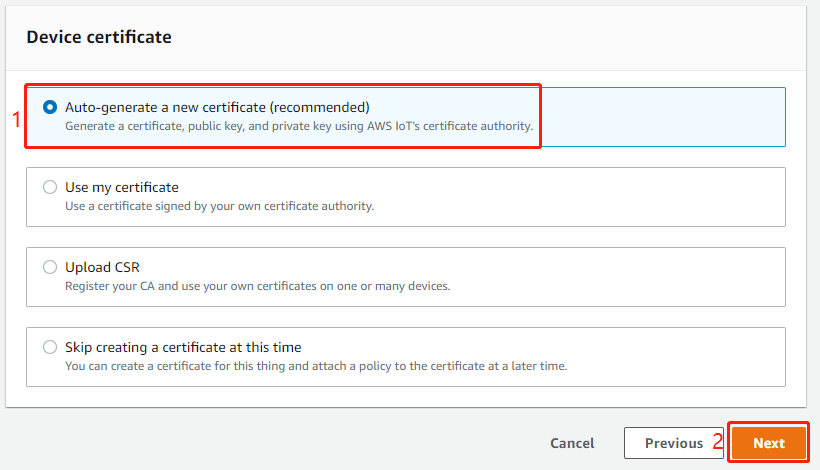
# Create things

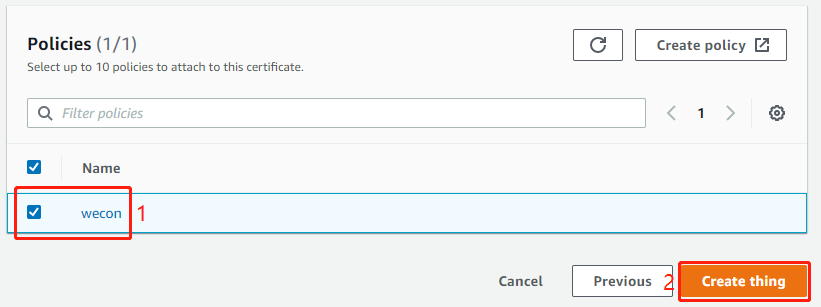
Click “Manage”--->“Things”--->“Create things”--->“Create single thing”

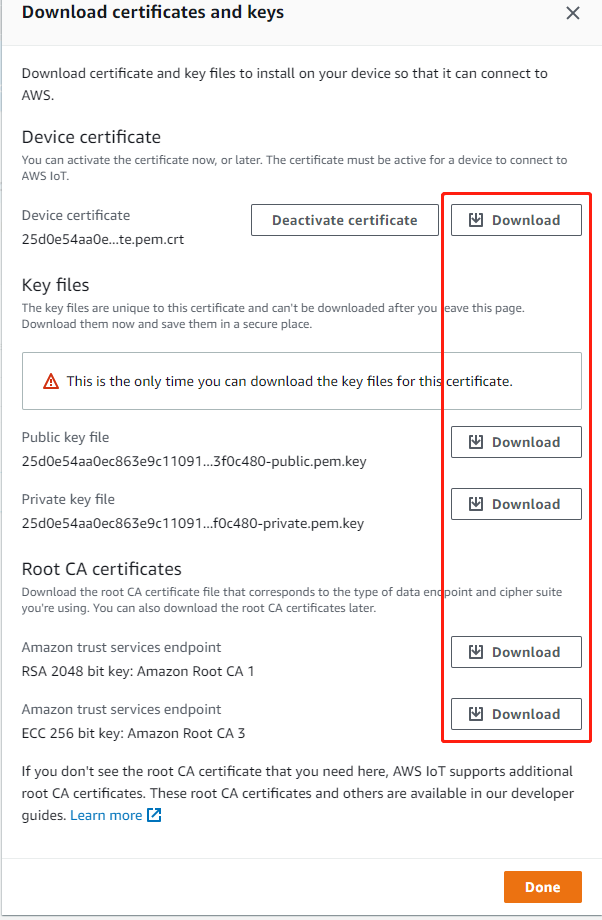


Name the thing--->Click “Next”

Select the way to create certificate

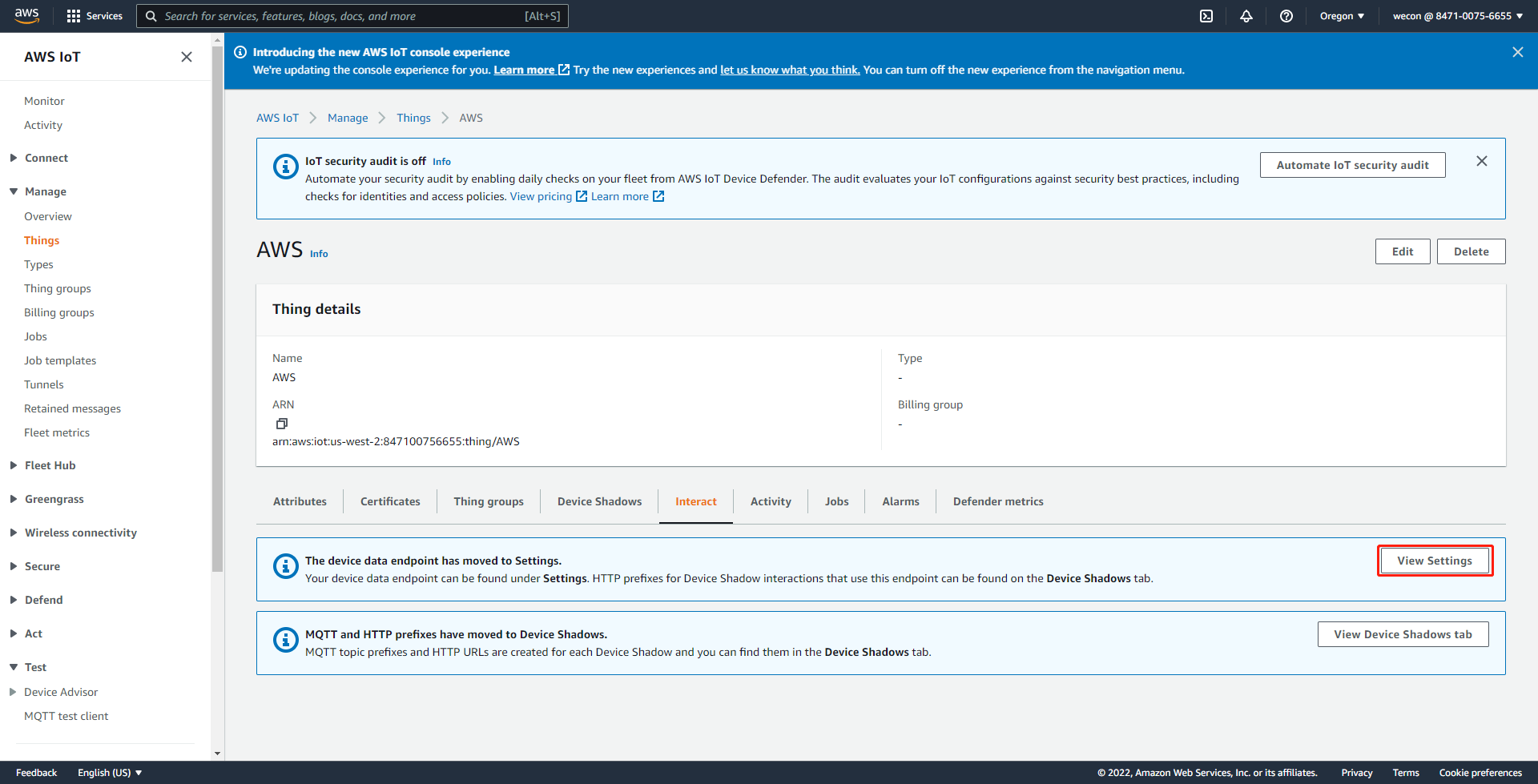
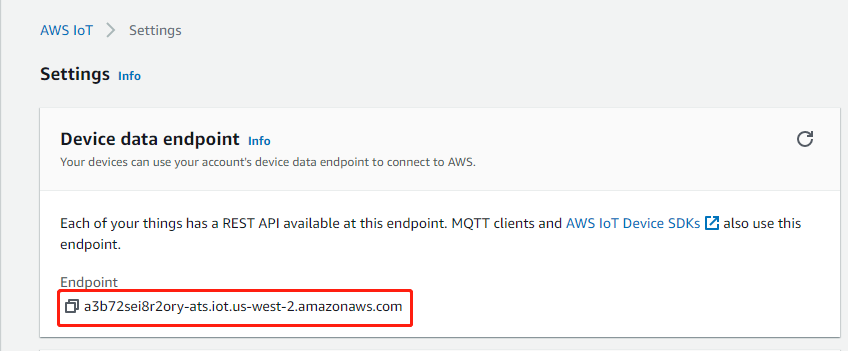
Select policy

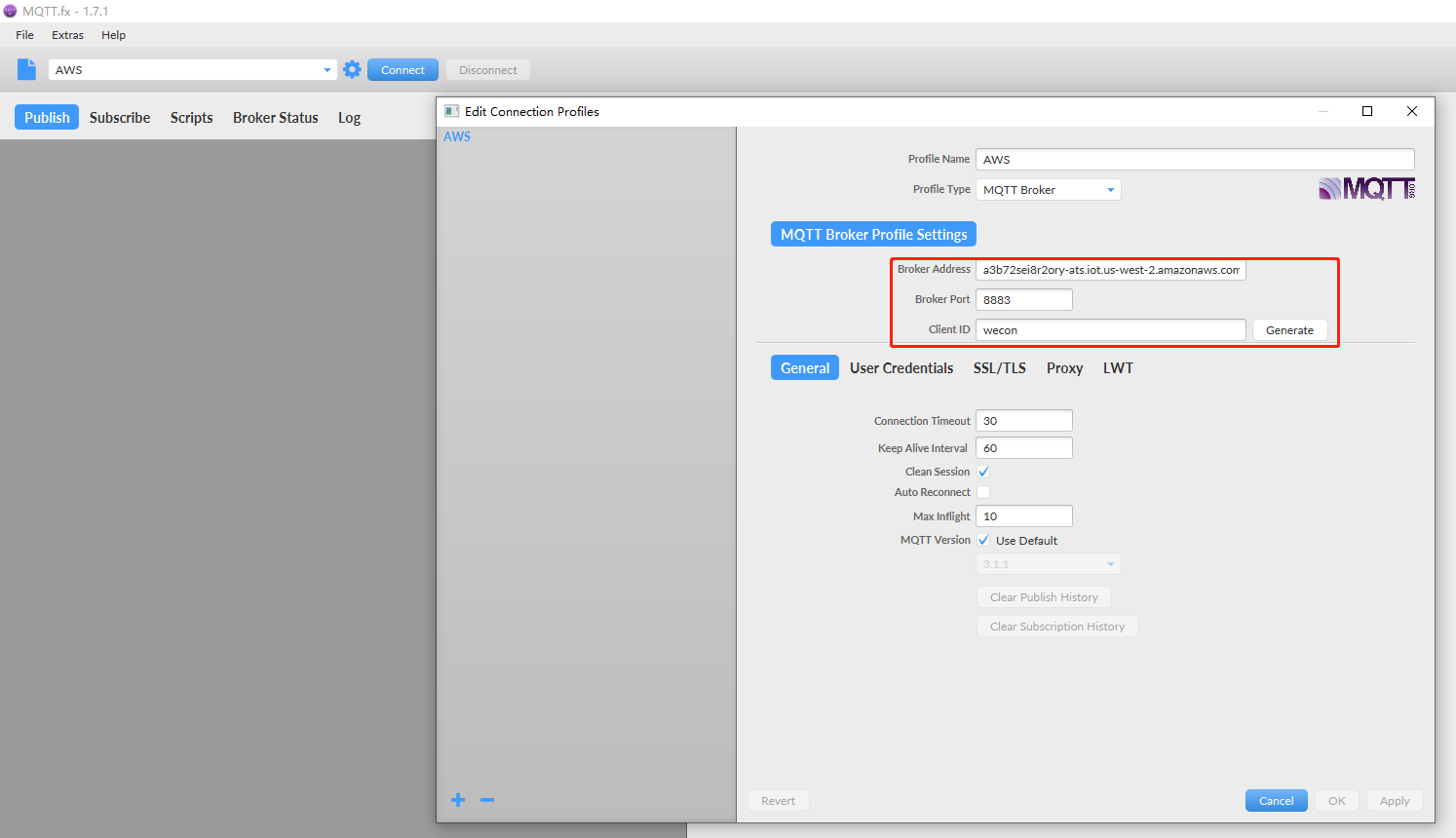




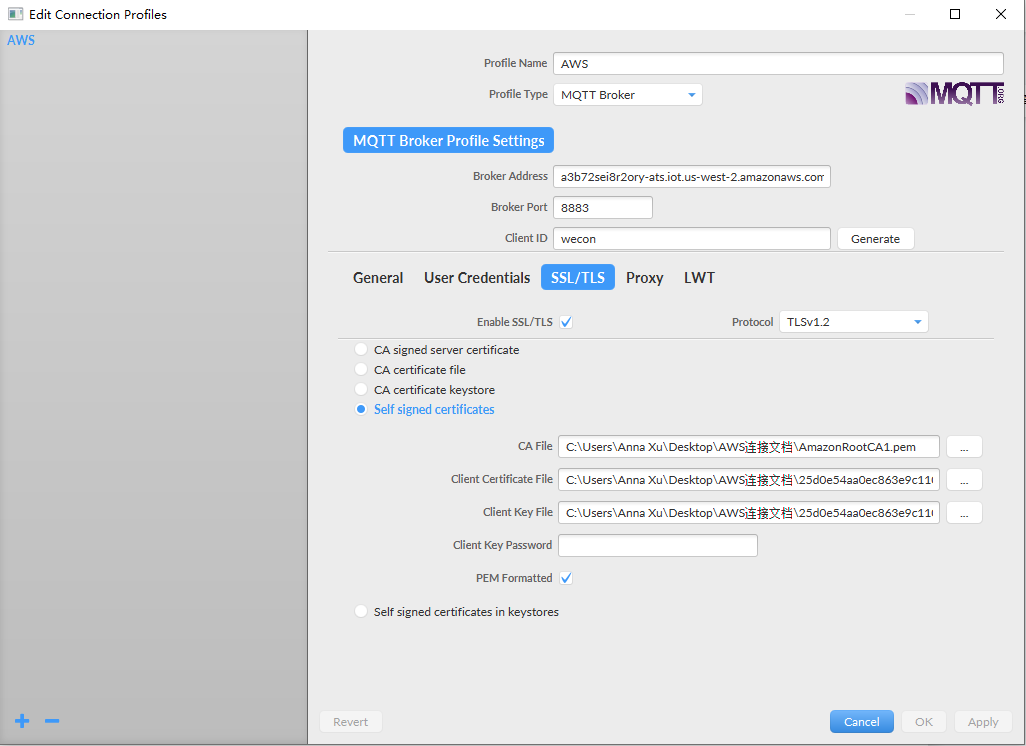
# Test with MQTT.fx tool

Click “View Setting” to get the “Broker Adress”

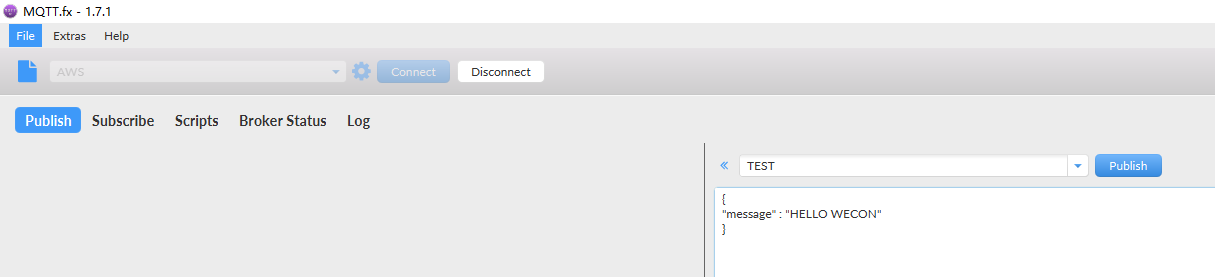
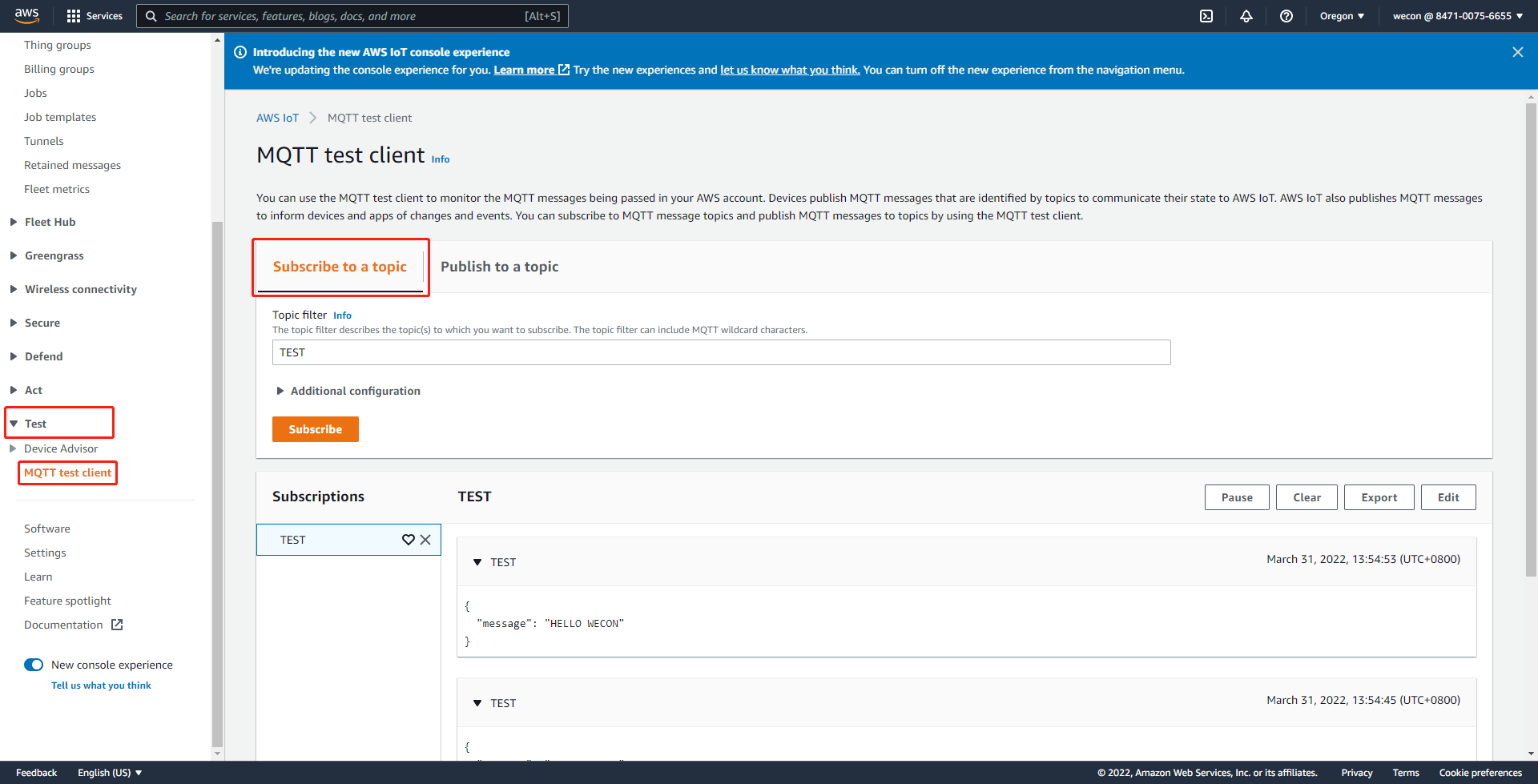


Create one connection in MQTT.fx tool, set broker port as 8883.

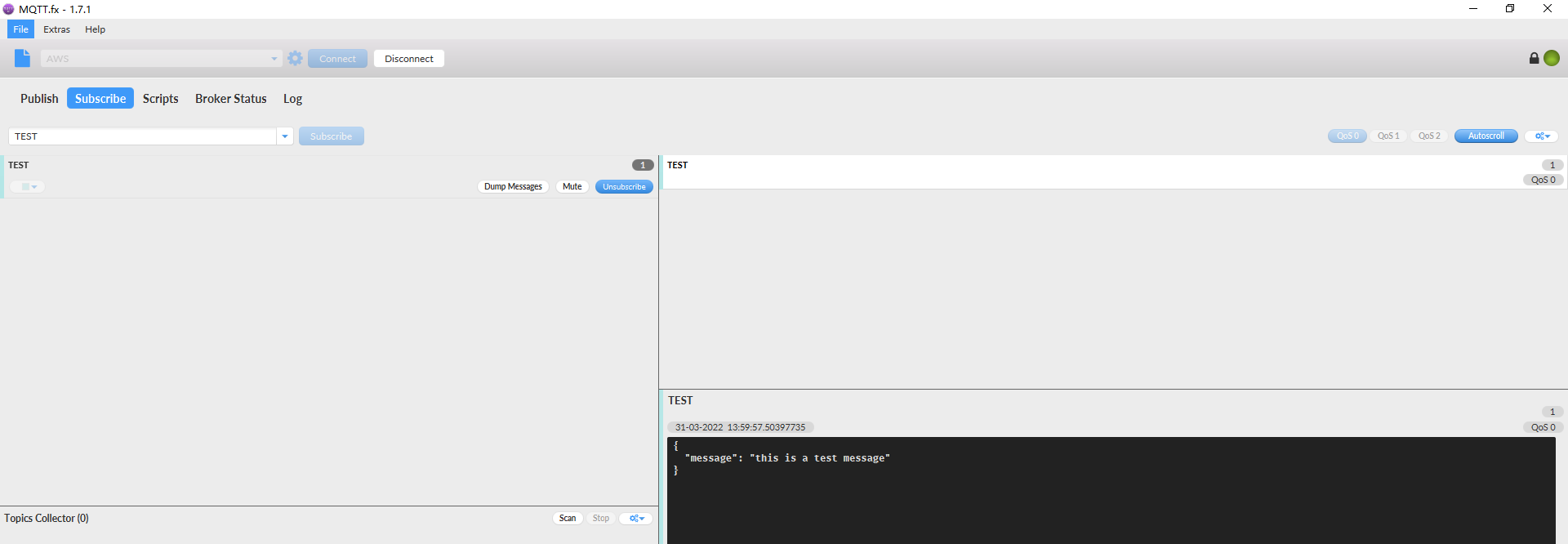
Upload the CA File, Client Certificate File, Client Key File



Publish message to topic “TEST”

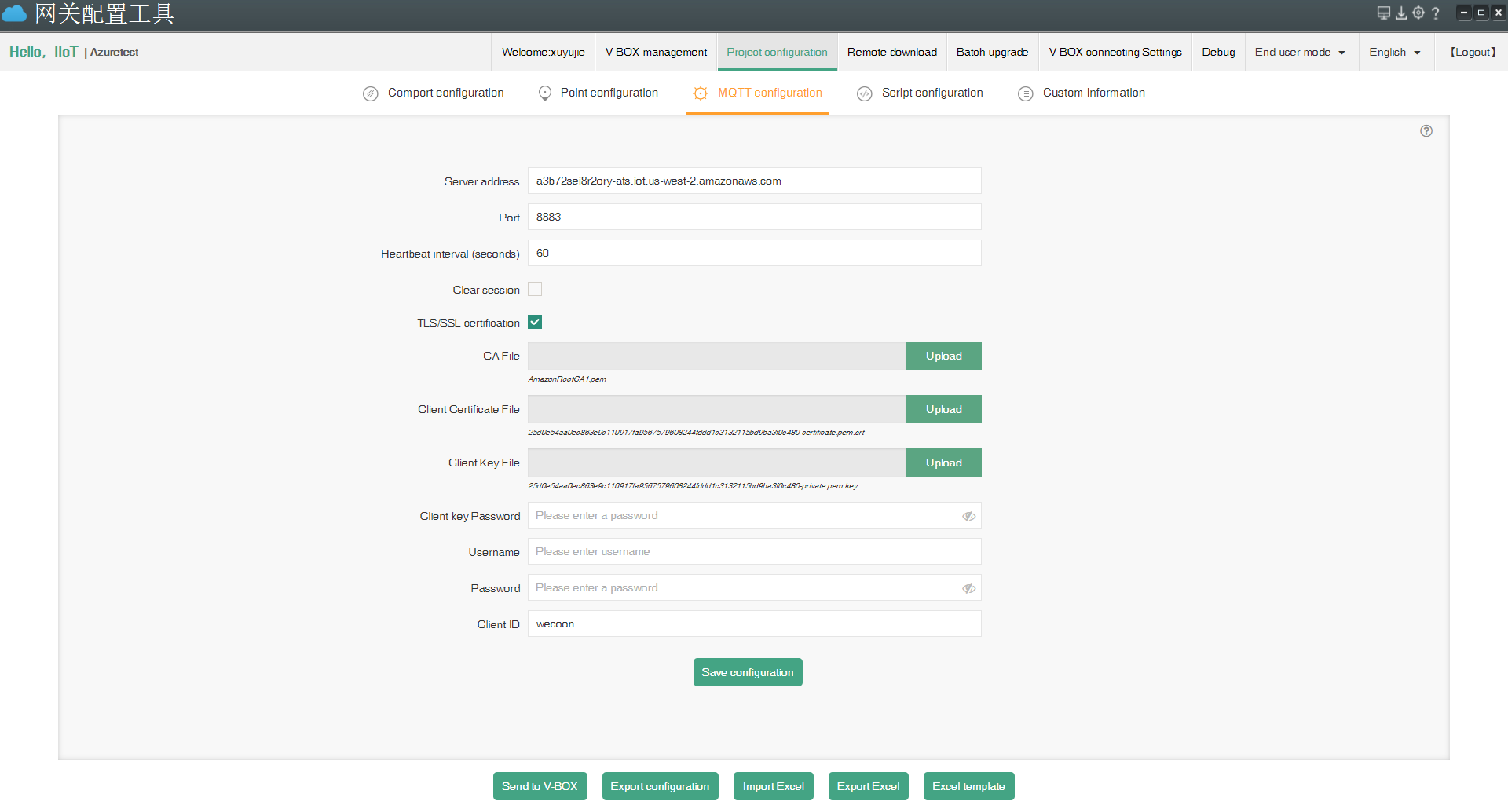
Click”Test”--->”MQTT test client”--->”Subscrible to a topic”, to get message publish from MQTT.fx tool.

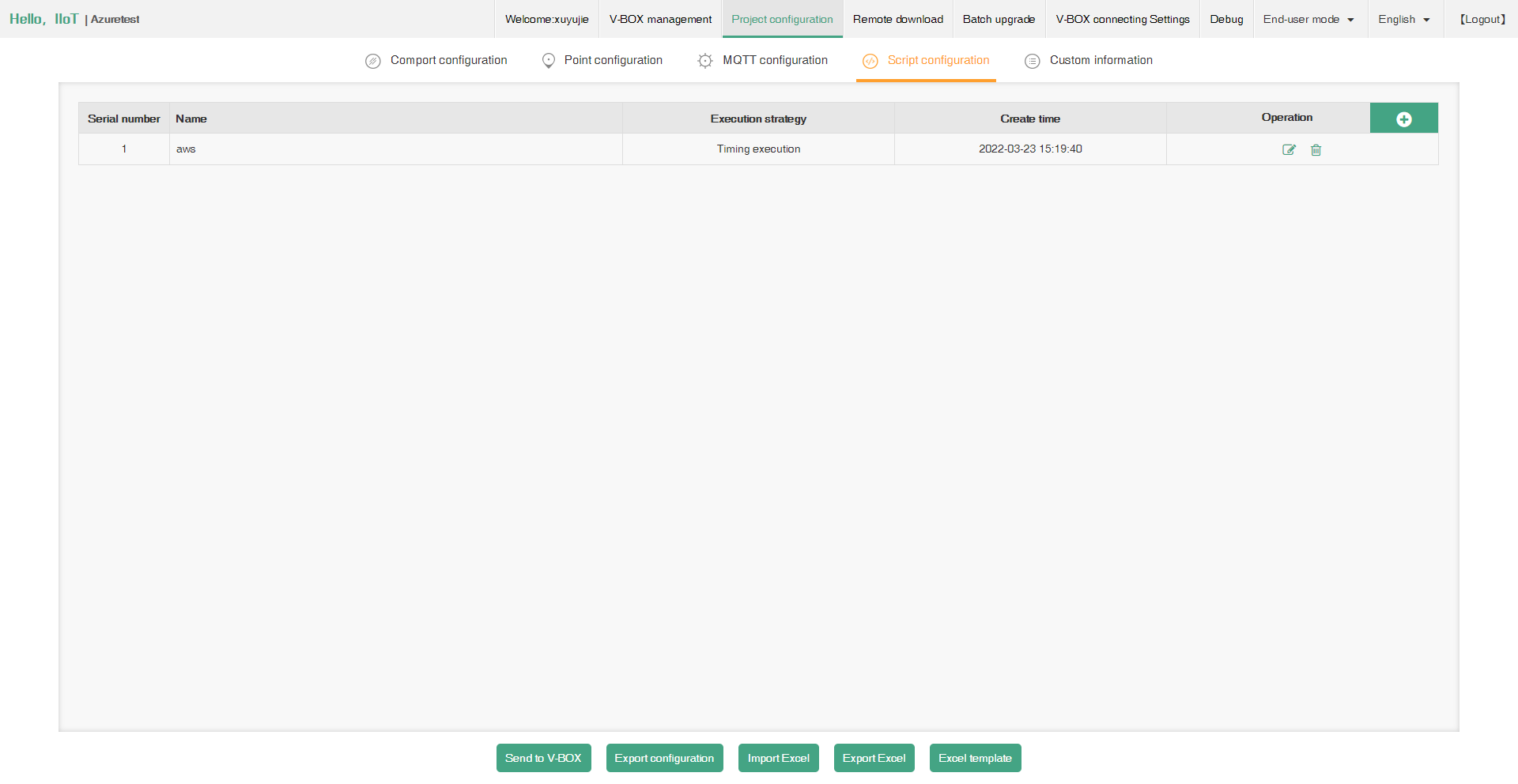
And we can also send message form AWS platform to MQTT.fx tool.



# Configurate in CloudTool

Copy the same setting in MQTT.fx to MQTT configuration

 Add a lua script and copy the lua demo into it.



sprint = print

--Cloud mode interface to obtain the MQTT information configured by the cloud platform: (5 returns, namely the server address, client ID, connection table, last word table, certificate table)

local MQTT\_URL, MQTT\_CLIENTID, MQTT\_CFG, MQTT\_LWT, MQTT\_CART = mqtt.setup\_cfg()

--publish to topics

local pub\_RE\_TOPIC = string.format('TEST')

--Subscribe topics

local Subscribe\_RE\_TOPIC1 = string.format('TEST')

--variable

local last\_time = 0

--Timing main function

function aws.main()

sprint(os.date("%Y-%m-%d %H:%M %S", os.time()) .. " aws.main start")

if g\_mq then

if g\_mq:isconnected() then

send\_Data()

else

if os.time() - last\_time > 5 then

last\_time = os.time()

mymqtt\_connect()

end

end

else

mymqtt\_init()

end

sprint(os.date("%Y-%m-%d %H:%M %S", os.time()) .. " aws.main end")

end

-- Initialize MQTT

function mymqtt\_init()

sprint(string.format("mqtt init mqtt\_url:%s mqtt\_clientid:%s", MQTT\_URL, MQTT\_CLIENTID))

g\_mq, err = mqtt.create(MQTT\_URL, MQTT\_CLIENTID) -- Create the object and declare it as a global variable

if g\_mq then

g\_mq:on("message", mymqtt\_msg\_callback) -- Register to receive message callbacks

sprint("mqtt init success")

else

sprint("mqtt init failed:", err)

end

end

-- Connect to MQTT server

function mymqtt\_connect()

sprint("mqtt connecting...")

local stat, err = g\_mq:connect(MQTT\_CFG,MQTT\_LWT, MQTT\_CART)

if stat == nil then

sprint("mqtt connect failed:", err)

return

else

sprint("mqtt connected")

end

g\_mq:subscribe(TEST, 0)

end

-- Receive MQTT message callback function

function mymqtt\_msg\_callback(topic, msg)

print("topic:",topic)

print("revdata:",msg)

local revData = json.decode(msg)

print (revData)

if topic == Subscribe\_RE\_TOPIC1 then --Process topic information subscribed from the cloud

if string.match(topic,Subscribe\_RE\_TOPIC1) then

--if revData ~= nil then

for k,v in pairs (revData) do

print("printing revdata after kv here")

print (k,v)

end

print ("current state is",fanstate)

--end

end

end

end

--Get real-time data

function getData()

local jdata = {}

local addr = bns\_get\_alldata()

print(json.encode(addr))

for i,v in pairs(addr) do

if v[2] == 1 then

jdata[v[3]] = v[4]

end

end

return jdata

end

--send data

function send\_Data()

local pub\_data =

{

123

}

sprint(json.encode(pub\_data))

print("..........",pub\_RE\_TOPIC)

return g\_mq:publish(pub\_RE\_TOPIC, json.encode(pub\_data), 0, 0)

end

Get message in AWS

