aws IoT lambda Backend Documentation

Release 1.0

Khoi Trinh

Nov 04, 2018

Contents

CHAPTER 1

Introduction

This is the Sphinx documentation for the IoT lambda backend modules, for overview and other info, visit the github page of the project

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

CHAPTER 3

README

CHAPTER 4

AWS Lambda IoT backend

Code for lambda function as backend for iot devices, written in **python 3**

4.1 The IoT devices code

The code for the IoT devices like esp8266 can be found at aws iot device code

4.2 Sphinx docs

Lambda Handler Docs

4.3 How to deploy to aws lambda

Use the create_deployment script to get a lambda.zip then upload it to your lambda function console, before that makes sure you have created a lambda function with Alexa Smart Home as a trigger

4.4 How the system works

The job of the lambda handler is to take request from Alexa, extract the necessary information and then pass it onto the correct IoT devices and vice versa, thus, it carries a database of devices that is implemented, their capabilities as well as their endpoint ID

When the user utters a smart home command, the command is sent to the lambda hander in a python dictionary, the Master Handler class parses that and obtain the payload, the name and the namespace of the command, package it into a json string(using a class called the Translator), then look up the endpoint ID in the message to know which devices to forward the information to, if the device exists, the handler will know which mqtt server to publish to to send the request to the IoT devices. The master handler use a class called MqttManager to manage pub/sub activities

Once the message has been forwarded, the handler waits until it timeouts(8 seconds) or it receives a reply from the expected topic (ie the topic that the target IoT device publishes to)

Once it receives the message from the IoT(which will usually contain the name, namespace and result value of the command that was sent to it), the master handler sends the message to the Translator class, which will reformat and attach more information to the base message so that it fits Alexa Smarthome response guideline for the corresponding endpoint/command types, after that the message is validated using Amazon schema and then returned so that Alexa can tell users the result

4.5 How to add new IoT devices

To add new devices that can be supported by Alexa, you need to initialize a new awsProfile, use that to create an IotObject for that object and then add it to the IOT_OBJ_LIST in the iot_object.py file, for example:

```
# insidie iot_object.py

# consult aws smart home documentation to see how to generate these things
# https://developer.amazon.com/docs/smarthome/understand-the-smart-home-skill-api.html
deviceAwsProfile = {
    "endpointId": "endpoint-001",
    "manufacturerName": "your name",
    "friendlyName": "my computer",
    "description": "Controller that controls and monitor the desktop PC",
    "displayCategories": [
        "SWITCH"
    ],
    "cookie": {},
    "capabilities": [
        {
            "type": "AlexaInterface",
            "interface": "Alexa",
            "version": "3"
        },
        {
            "type": "AlexaInterface",
            "interface": "Alexa.PowerController",
            "version": "3",
            "properties": {
                "supported": [
                    {
                        "name": "powerState"
                    }
                ],
                "proactivelyReported": True,
                "retrievable": True
            }
        },
        {
            "type": "AlexaInterface",
            "interface": "Alexa.EndpointHealth",
            "version": "3",
            "properties": {
                "supported": [
                    {
                        "name": "connectivity"
                    }
```

(continues on next page)

(continued from previous page)

```
                ],
                "proactivelyReported": True,
                "retrievable": True
            }
        }
    ]
}

# create IoT object representation of the device
# note that pub and sub topic refers to topic that the device pub/sub not the lambda␣
↪handler's pub/sub
deviceName = IotObject("mqttPubChannel", "mqttSubChannel", deviceAwsProfile)

IOT_OBJ_LIST = [device1, device2, deviceName] # add the device to the list
```

4.6 Project strucutre

The python source files are in the main dir of the project, the files in the folders are usually dependencies

- certs/: carry public, private keys, access point ID, certificate, certificate of authority, basically sensitive stuffs used for authentication purpose, the folder contain these files:
  - accessPointID.txt: the access point ID of your device, in the aws IoT console, there is a Rest API endpoint, it will be in the format of accessPointID-ats.iot.us-east or accessPointID.iot.us-east, copy this ID to this file
  - certificate.pem.crt: certificate of your device, downloaded from console
  - private.pem.key: private key of your device only downloadable when first creating the aws iot thing
  - VeriSign-Class_3-Public-Primary-Certification-Authority-G5.pem: Certificate of Authority that can be downloaded from aws website
- docs/: contain Sphinx documentation for python modules
- src/: contain the source files of the handler
- lambda_main.py: this is the main lambda handler that AWS uses as entry point
- The rest of the folders: dependencies for either json schemas or for AWS python SDK

4.6.1 Python source files structure

- iot_object.py: defines a basic class that represents every IoT devices and also store the aws IoT profiles of every profile that can be discovered by Alexa
- lambda_master_handler.py: contain the MasterHandler class, which is responsible for coordinating between different modules such as the MqttManager and the Translator
- lambda_mqtt_manager.py: contain the MqttManager class, which is responsible for establishing and sub/pub with the AWS MQTT server, it also contains the callback function to be called every time a message arrived
- mqtt_constant.py: files used for storing settings of the mqtt connection as well as information necessary to make an mqtt connection
- translator.py: file contains the Translator class which is responsible for translating from Alexa message to IoT and vice versa

- utils.py: general utils files mainly used for parsing the json dictionary
- validation.py: the file was given by Amazon to validate the final response json to make sure it fits Alexa response schema

4.7 Shell script documentation

- create_deployment.sh: used for creating a zip file that can be uploaded to aws lambda to serve as the lambda function

```
./create_deployment.sh # create a file called lambda.zip
```

- install_dependencies.sh: install the necessary dependencies to the lambda handler including things like AWS python IoT SDK and the json parser tools

```
./install_dependencies.sh # install dependencies to the current dir
```