Secure Internet of Things(IoT) Network system

Chandana NS
MS in Information Systems with Computing
Dublin Business School

CONTENTS

LIST OF FIGURES

Secure Internet of Things(IoT) Network system

***Abstract*—The primary objective of this project is to implement an end to end Internet of Things(IoT) network system architecture, which is highly scalable and secure. Using Amazon Web Services(AWS), an IoT thing was registered and an Elastic Compute Cloud(EC2) instance was created. Secure Shell(SSH) was used to access the EC2 instance. Secure Copy Protocol/Secure File Transfer Protocol(SCP/SFTP) was used to transfer files that are necessary to subscribe to a topic on the EC2 instance. Temperature data from two devices were extracted and published through Message Queuing Telemetry Transport(MQTT) on the AWS cloud server. The EC2 instance was the subscriber and subscribed to the same topic to which the devices published the data. The AWS IoT acts as a broker and receives the data. The data was stored and visualized on a dashboard that is accessible using a Public Domain or IP Address with a specific port number, that was deployed using Nodered. The report summarizes, how one or more devices communicate on a network and periodically send relevant data to a centralized server on the cloud and displays the data for visualization and monitoring on a remote device.**

I. Introduction

IoT, Internet of things also called M2M (Machine to Machine) is a network of several computers or devices that has access to the internet without requiring a human interface. AWS IoT provides device software, control services, and data services(Amazon Web Services, Inc., 2020). The programs in the AWS Cloud can communicate over the internet with the devices connected and eventually can control them remotely. Each device communicates its status by publishing messages on the MQTT (Message Queuing Telemetry Transport) topics in JSON (JavaScript Object Notation) format. Every MQTT topic has a categorized name which helps in the device's identification whose status has changed. Every time a message is published on the topic, it is sent to the message broker(Hcc-embedded.com, 2020). It is the duty of the message-broker to send all messages published on the MQTT topic to all devices subscribed/registered to that MQTT topic. Before starting any communication every device needs to get its certificate verified by the AWS IoT. The communication between a device/equipment and AWS IoT-Platform is secured by X.509 certificates. The end-user needs to create rules, and each rule has its definitions to perform actions according to the message published. They also define IAM roles are also inside the rules which grant permissions to IoT. To retrieve the information about the state of any device, it uses a Shadow.

II. Problem Definition and Architecture

The aim is to implement an end to end IoT network system architecture, by developing a dashboard that monitors temperature over time by retrieving data on the cloud server from one or more devices through the network which is highly scalable and secure. The temperature data extracted from two devices is pushed to the AWS IoT cloud service and later retrieved on an EC2 server to visualize data on a monitor.

A thing is registered on AWS, the functional block diagram(Fig 1), indicates a set of two devices that publish data to AWS IoT that acts as the broker. AWS EC2 acts like a subscriber and subscribes to the same topic on AWS IoT. This data is stored on the EC2 server and retrieved by a Web-application that is hosted on the same server and accessible on a remote device for monitoring.

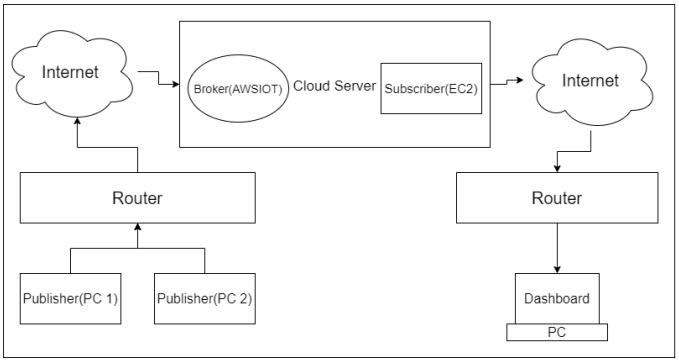

Fig. 1. Block Diagram

III. Background

The architecture consists of one or more devices that are on a network are allowed to send data to a centralized server on the cloud, in-turn the cloud retrieves the data and displays it on a monitor to make a visual representation of the data. An example is obtaining a histogram chart of temperature vs time. Since the devices are all connected either directly or indirectly to the cloud it requires the devices and the communication to be secure.

The Software Used and Technologies Implemented:

A. Cloud Computing

Cloud Computing is a model for providing on-demand basis access to a network that includes various network resources like storage, web-servers, databases, mail servers that can be activated and deactivated depending on the requirement for any application. This helps in reducing cost and increasing the efficiency of the entire system.

1) AWS IoT: IoT core service provided by AWS facilitates devices that connect to the internet to send data through popular protocols like HTTP and MQTT. AWS IoT core service is highly scalable because it can be used to send trillion messages and provides the ability to connect billions of edge devices. There are other services that can be integrated with the IoT core service. Some examples are adding rules like storing data on Database, integrating with AWS Lambda function, enabling notification service based on the data received and processed on IoT core. There are also useful applications like Kibana to which IoT core can be integrated that allows users to visualize data in real time.

2) AWS EC2: Amazon Elastic Compute Cloud is a network service that caters to a resizable and secure compute module in the cloud (Amazon EC2). It is designed to make web-scale cloud computing easier for developers. Amazon EC2's simple web service interface allows us to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment. (Amazon Web Services, Inc., 2020)

B. Secure Shell (SSH)

The Secure Shell protocol(SSH) runs on TCP port 22. This protocol allows a client to establish a secure connection with the server. SSH is a secure protocol for the reason that it allows only trusted users to login using authentication and transfer data that is encrypted. Authentication can be through methods such as basic username-password based and SSH keys. Terminal applications like putty on windows and terminal on MAC is used to SSH into a server.

C. Secure Copy Protocol (SFTP/SCP)

SFTP (SSH File Transfer Protocol) is a secure file transfer protocol. It runs over the SSH protocol. It supports the full security and authentication functionality of SSH. SFTP provides all the functionality offered by these protocols, but more securely and more reliably, with easier configuration. SFTP also protects against password sniffing and man-in-the-middle attacks. It protects the integrity of the data using encryption and cryptographic hash functions and authenticates both the server and the user.

D. Web Services

Web services based on the service-oriented architecture framework provide a suitable technical foundation for making business processes accessible within enterprises and across enterprises. But to appropriately support dynamic business processes and their management, more is needed, namely, the ability to prescribe how Web services are used to implement activities within a business process, how business processes are represented as Web services, and also which business partners perform what parts of the actual business process(Leymann, Roller and Schmidt, 2002).

E. Message Queuing Telemetry Transport (MQTT)

Message Queue Telemetry Transport (MQTT)(IoT Agenda, 2020) protocol is a publisher-subscriber protocol model with a central broker known both by the publisher and subscriber for filtering all incoming messages and distributing them accordingly. MQTT is applicable only for constrained devices with low power, low consumption capacity, memory, and low bandwidth. MQTT supports multi-cast communication and can provide communication to any remote device. The advantage of using MQTT is that it reduces network traffic by reducing the overhead of transport and protocol exchange, it reduces the bandwidth requirement and guarantees delivery of the packet. It also notifies the user if any abnormal conditions have occurred. It uses three methods to achieve the quality of service (QoS). QoS1: The publisher sends the data to the broker and does not wait for an acknowledgment. If the sent data is not received it is not re-transmitted again. Data is lost. QoS2: The publisher sends the data to the broker and waits for an acknowledgment. If the acknowledgment does not arrive within a predefined time, the data is re-transmitted. This profile improves the reliability and increases the overhead. QoS3: This Scenario requires an additional protocol to make sure the message is delivered only once. It supports mobile applications and different devices. Since it has a high sampling rate and high latency it is not applicable for real-time application.

IV. Technical Description of the Proof of Concept

During my initial days of research, I always wanted to implement an end to end system using AWS IoT in a way it works enabling devices to connect to the AWS Cloud and lets applications in the cloud interact with the devices connected to the internet. Common IoT applications either collect and process data from devices or enable users to control or monitor a device remotely. The initial walk in the system process is to register a thing (Fig 2. Register Thing) named device1 on AWS IoT so that it is simple to distinguish between various devices on the cloud. The account has been secured by using certificates, this is performed through adding a certificate for the thing (Fig 3. Create Certificate), it has to be created in-order to authenticate device connection to AWS IoT. To connect to the device, A Certificate for the thing, A Public Key, A Private key, and A Root CA: RSA 2048 bit key has been downloaded and activated (Fig 4. Download Certificates). An AWS IoT policy (Fig 5. Create Policy) has been created under the secure menu of AWS IoT to define a set of authorized actions, In this case, I named it as 'allowall' with action type as 'iot:*' that indicates no constraint on the things (Fig 6. Create Policy action). This policy has to be attached to the thing (Fig 7. Attach Policy) which completes the AWS IoT setup on the cloud server. To confirm this thing, a topic has been subscribed and published under the Test tab (Fig 8.Test AWS thing).

In the next step, to publish a topic from any device like a PC, a script has been written using the AWSIoTMQTTClient package (GitHub, 2016). In the script MQTTClient instance(NS, 2020) has been created and an endpoint has been configured (Endpoint is available in AWS IoT, setting) with standard

MQTT port number 8883. The corresponding certificates are attached to the configureCredentials method in the script. The MQTT client has been connected with a topic, and the JSON has been published and then the client has to be disconnected. Buy running the script(Fig 18.Python script), the topic published can be seen on the AWS IoT test tab (Fig 9.Test script).

Similarly, to subscribe to a topic, an Ubuntu EC2 instance has been launched(Fig 10.Ubuntu EC2 instance), a new key pair has been created to connect to the instance securely before launching an instance(Fig 11.Key pair). An elastic IP has been associated with the instance to make the instance IP static(Fig 12.Allocate elastic IP). To access an instance, an Open SSH client, Terminus is opened, in the host tab new host is added with an appropriate IP address, name, primary key, and private key has been associated(Fig 13. Connect Host). Localhost and EC2 instances were connected using SFTP, the certificates and script file for subscribing were transferred from localhost to Ubuntu instance over SFTP terminus(Fig 14.SFTP). Python and packages have been installed in the Ubuntu instance, and the instance can also be accessed through the terminal (Fig 16. Terminal). Inside the EC2 instance, the same topic has been subscribed. On the local machines, the topic has been published and saved to a text file. The resulted payload can also be seen on the AWS IoT test.

The data from the text file has been rendered on a dashboard using the Nodered platform((Nodered.org, 2020)). Nodered has been installed to the EC2 instance(Fig 19.Setup nodered) and the inbound rules have been defined under security groups (Fig 20.Inbound rules). The nodered architecture of the dashboard has been installed and initialized by adding an inject node. The output payload has been connected to the file storage node, this node has been used to read the content of the file as a string. The route messages were split based on device1 or device2 data. The final gauge node is joined after stripping and splitting the data value as required. The dashboard has been deployed and published values from the devices based on time. The complete architecture can be viewed in Fig 21.Nodered Architecture. The visualization of the data can be accessed on the run time using AWS DNS or IP with default nodered port 1880(Fig 22. Dashboard).

A. Project Testing and Evaluation

An End to end IoT system has been implemented using MQTT to publish and subscribe to topics on AWS IoT. To test the setup, a python script was run on an IDE. The function of the script was to connect to the AWS IoT broker on MQTT port 1883 using certificates. Once a connection was established random temperature data was generated and published. To mimic an additional device another instance of the IDE was used and the same script was run with a different client-Id and a device name. AWS IoT Test window was used to check if the data was getting published to the topic. A subscriber script was run on EC2 instance by accessing the instance through the terminal. The IP address was used to access the dashboard on a browser and the published data was visible on the dashboard in the form of a chart. The frequency of publishing was set to 5 seconds, i.e. data from each device was published every 5 seconds and was instantaneously available for viewing on the dashboard. Some challenges that were observed while building the project are as follows: Each publisher must have a unique client Id that makes scalability a challenge. Although this challenge can be overcome by having an automated production system that assigns a unique client-id to each device. Also, certain organizations might not want their data to be sent to a third-party server like AWS and might request for an on-premise solution. In this case, managing these servers becomes a challenge as the service is not hosted on the cloud and shall not be accessible from outside the network.

V. Conclusion

To conclude, AWS IoT architecture helps to summarize, how one or more devices communicate on a network over the cloud services, and periodically send relevant data to a centralized server and displays the data that can be visualized and monitored on a remote device.

References

[1] Amazon Web Services, Inc. (2020). Amazon Web Services (AWS) - Cloud Computing Services. [online] Available at: https://aws.amazon.com[Accessed 6 Mar. 2020].

[2] Amazon Web Services, Inc. (2020). AWS IoT Core FAQs - Amazon Web Services. [online] Available at: https://www.amazonaws.cn/en/iot-core/faqs/ [Accessed 7 Mar. 2020].

[3] Docs.termius.com. (2020). Connecting to a Host. [online] Available at: https://docs.termius.com/termius-handbook/connecting-to-a-host [Accessed 7 Mar. 2020].

[4] GitHub. (2016). AWS IoT SDK PYTHON AWS IoT MQTTClient. [online] Available at: https://github.com/aws/aws-iot-device-sdk-python/ [Accessed 26 Feb. 2020].

[5] Hcc-embedded.com. (2020). MQTT Primer. [online] Available at: https://www.hcc-embedded.com/uploads/2019-05/HCC-MQTT-Primer-v1_00.pdf [Accessed 7 Mar. 2020].

[6] Leymann, F., Roller, D. and Schmidt, M. (2002). Web services and business process management. IBM Systems Journal, 41(2), pp.198-211.

[7] Mell, P. and Grance, T. (n.d.). [online] Faculty.winthrop.edu. Available at: http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf [Accessed 6 Mar. 2020].

[8] Nodered.org. (2020). Node-RED. [online] Available at:https://nodered.org/ [Accessed 6 Mar. 2020].

[9] NS, C. (2020). AWS IOT publisher from a device. [online] Available at: https://github.com/ChandanaNS/Networks [Accessed 6 Mar. 2020].

[10] S. Saritha and V. Sarasvathi, "A study on application layer protocols used in IoT," 2017 International Conference on Circuits, Controls, and Communications (CCUBE), Bangalore, 2017, pp. 155-159.

[11] Ssh.com. (2020). SSH (Secure Shell) Home Page. [online] Available at: https://www.ssh.com/ssh [Accessed 6 Mar. 2020].

APPENDIX A
CODES, SNIPPETS, AND SCREENSHOTS

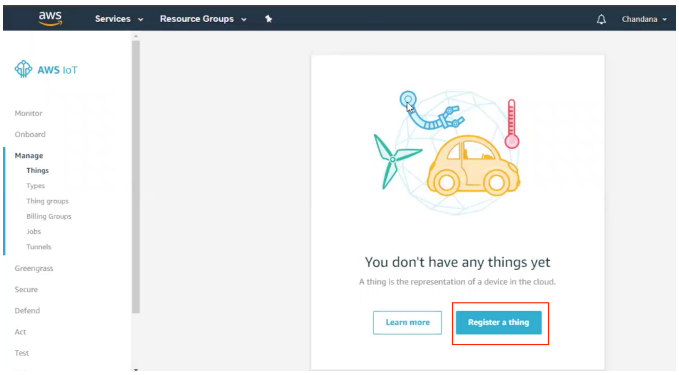

Fig. 2. Register a thing on AWS IoT

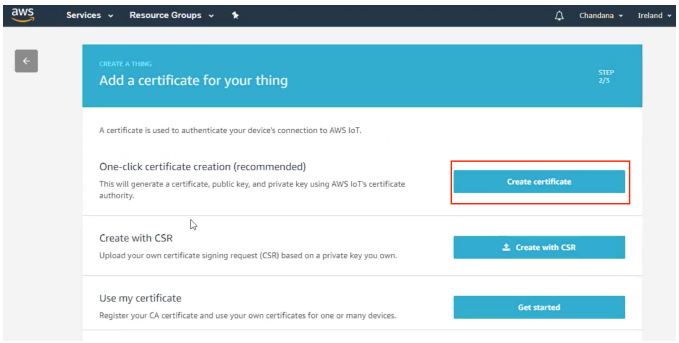

Fig. 3. Create a certificate for the thing

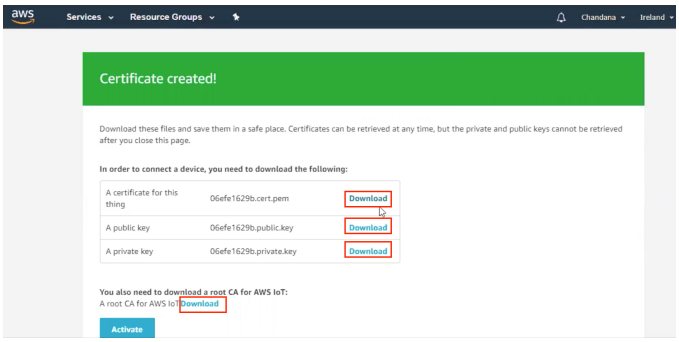

Fig. 4. Download created certificates

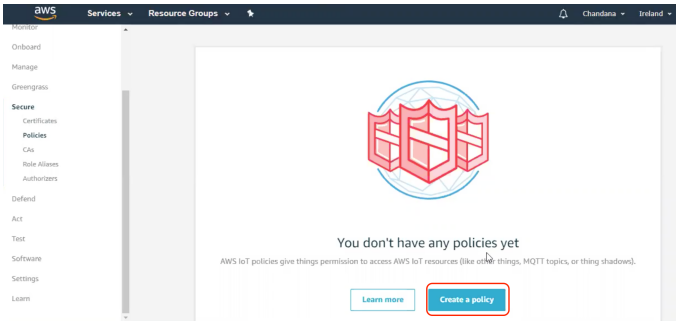

Fig. 5. Creating a policy for the certificate

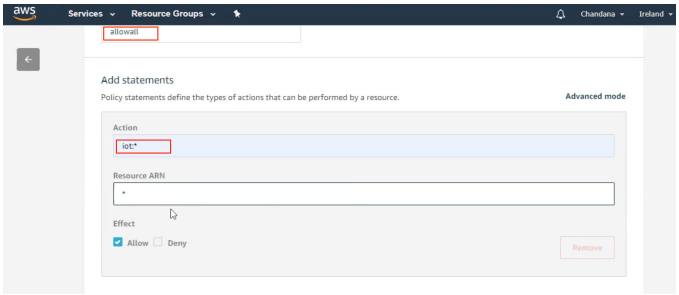

Fig. 6. Creating a policy action

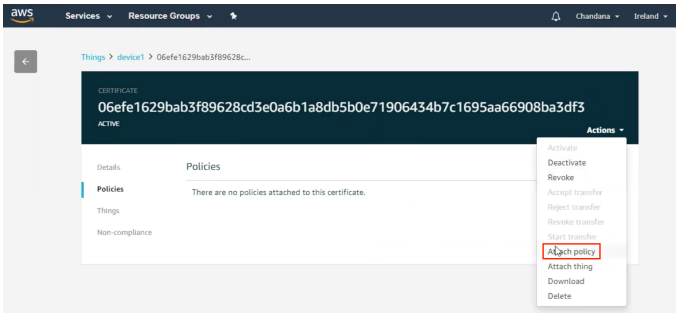

Fig. 7. Attach policy

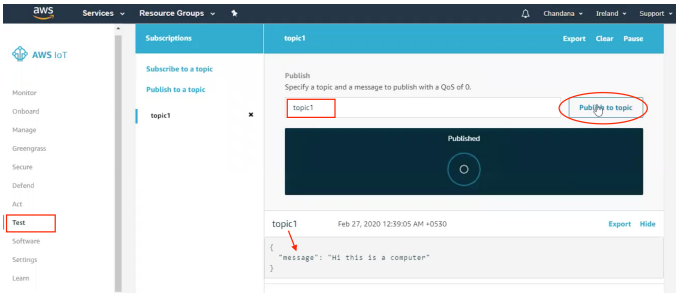

Fig. 8. Test an AWS IoT thing

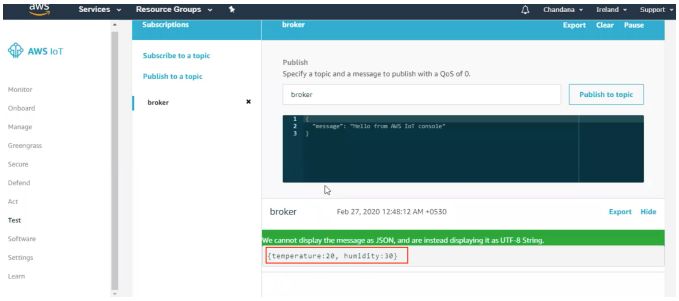

Fig. 9. Test an AWS IoT thing using script

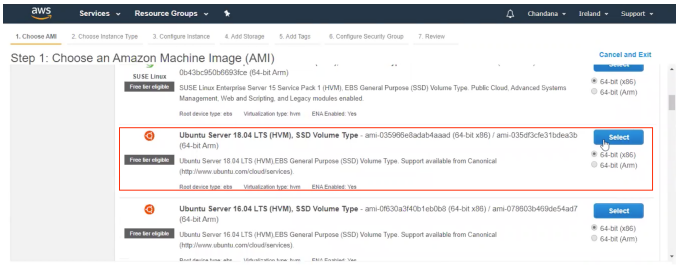

Fig. 10. AWS EC2 instance Creation

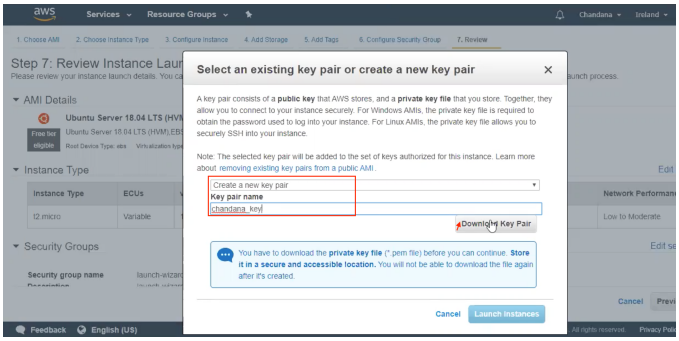

Fig. 11. Create new key pair

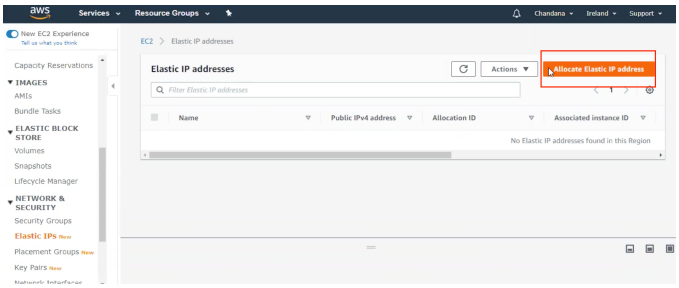

Fig. 12. Allocate elastic IP for EC2 instance

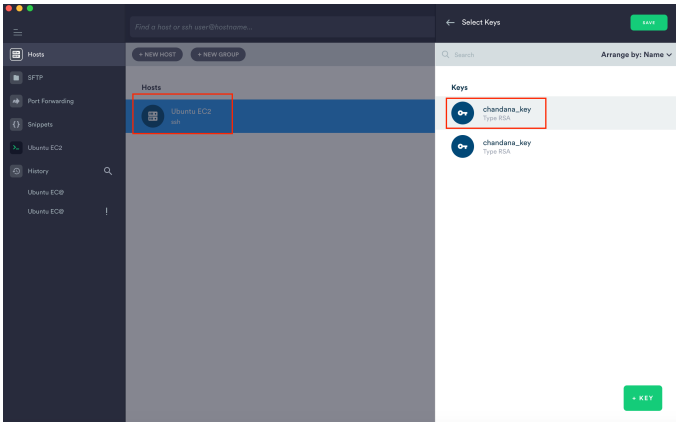

Fig. 13. Connect to the Host server

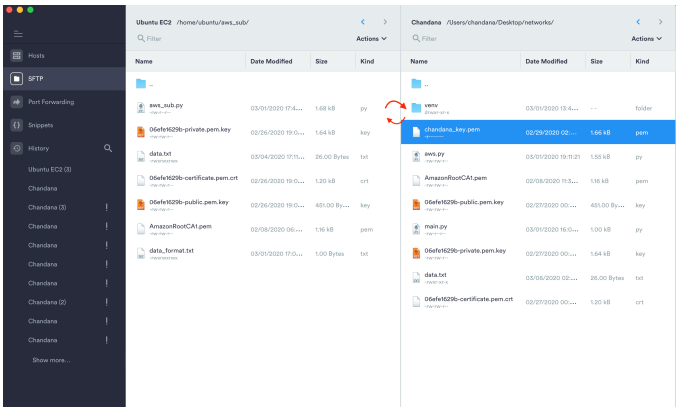

Fig. 14. Terminus SFTP

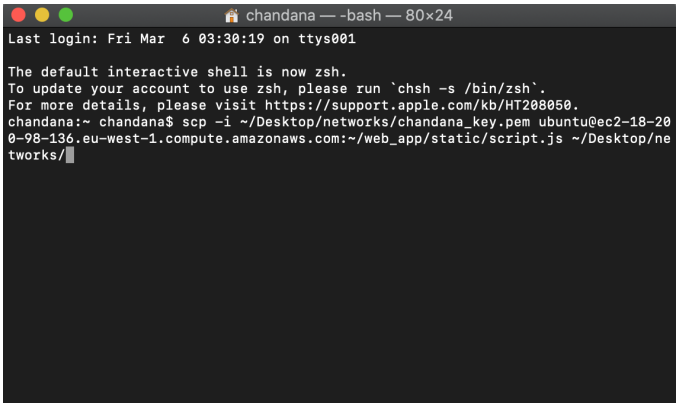

Fig. 15. Terminal SFTP

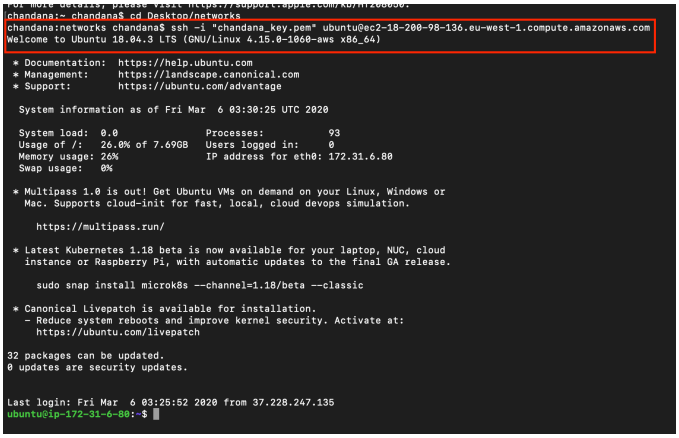

Fig. 16. Host connection using terminal

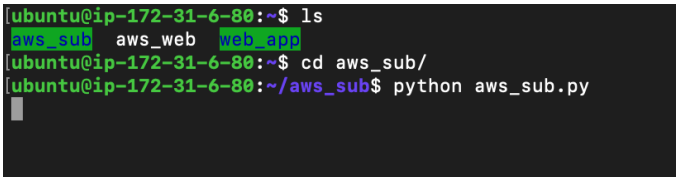

Fig. 17. Running Subscriber script

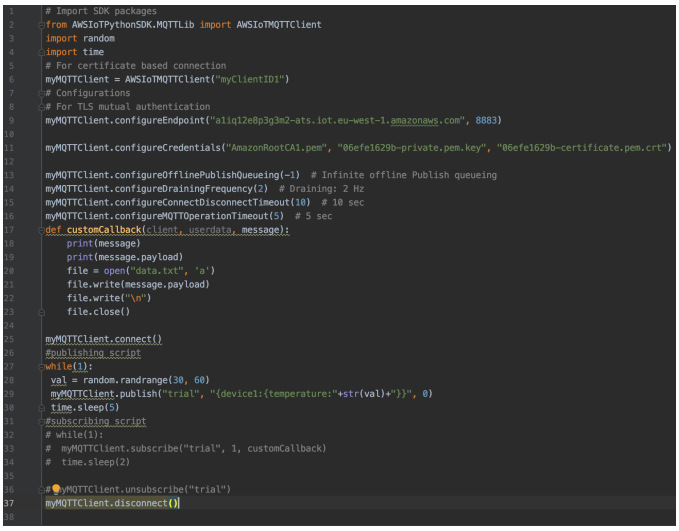

Fig. 18. Python script

Fig. 19. Nodered set-up script

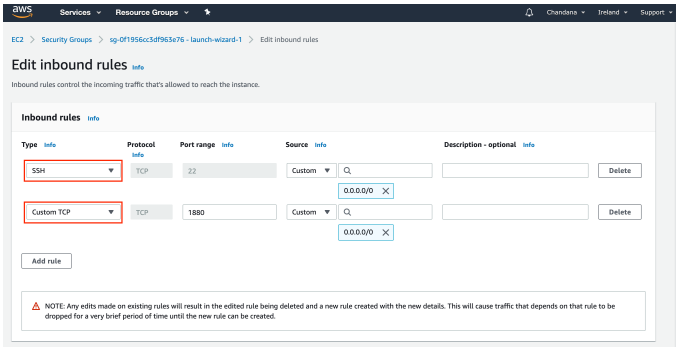

Fig. 20. EC2 inbound rules

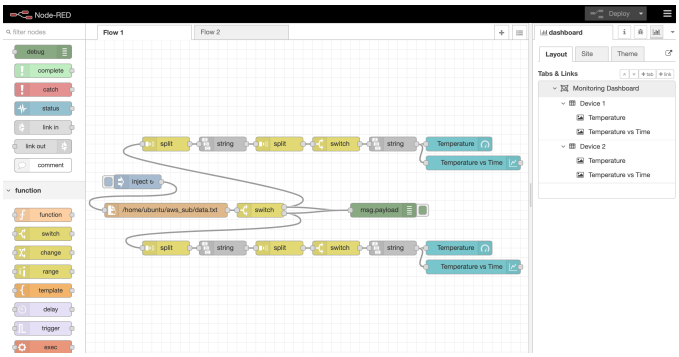

Fig. 21. Nodered Architecture

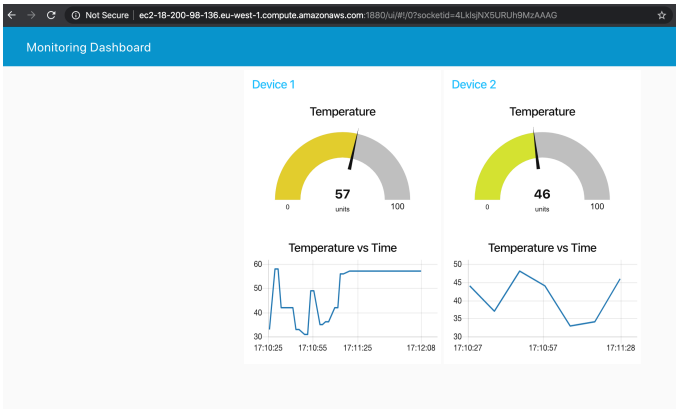

Fig. 22. Dashboard