

Demonstration & Setup of Virtualised IoT Devices

(Smart hospital network IoT deployment)

Table of Contents

| | |
|---|-----------|
| Introduction..... | 1 |
| 1. Preparing the virtualised Cloud9 & Docker environments | 2 |
| I. Set up the AWS Cloud9..... | 2 |
| II. Set up Docker Containers | 3 |
| 2. Preparing AWS IoT Core, registering Things | 4 |
| I. Create the IoT Policy | 4 |
| II. Register IoT Things | 4 |
| 3. Preparing IoT devices with AWS IoT SDK..... | 6 |
| I. Prepare the Cloud9 IoT environment..... | 6 |
| II. Prepare the Docker IoT environment..... | 7 |
| 4. Preparing Python Scripts, Device Shadow, and MQTT messages | 8 |
| I. Prepare the Python Scripts | 8 |
| II. Configure AWS Things Device Shadows..... | 8 |
| III. Test the MQTT messaging..... | 10 |
| 5. Preparing AWS IoT Analytics, ingesting IoT data | 12 |
| I. IoT Analytics quick setup | 12 |
| II. Running the IoT Analytics..... | 13 |
| 6. Implemented Security: AWS IoT Device Defender, Auditing, and Monitoring | 15 |
| I. IoT Device Defender setup | 15 |
| II. SNS notifications setup..... | 17 |
| 7. The Demonstration breakdown & script | 19 |
| I. Demonstration breakdown | 19 |
| II. Demonstration script..... | 20 |
| References..... | 21 |

Introduction

For this capstone project, a portion of the hospital's smart IoT network design is to be demonstrated as a limited proof-of-concept exercise.

After weeks of research and hands-on live testing was carried out, it was determined that Amazon Web Services would be the main technology used to carry out the IoT demonstration task. The experience gained during this process has provided the knowledge to design and deploy a smart IoT network using the AWS IoT Core suite.

For a more in-depth look at this project's exploration of AWS IoT Core, refer to the document "Group1-Research-AWS-Handling-of-Virtualised-IoT-Devices.docx". The feasibility of implementing aspects of Analytics and AI into the demonstration plan was also researched, please refer to the document "Group1-Research-AWS-Analytics-and-AI.docx".

To quickly summarise the demonstration plan, the smart handling and integration of IoT devices into the hospital network is to be simulated via the AWS IoT Core. Medical IoTs considered on premises of the hospital will be hosted on AWS Cloud9 instances, whilst medical IoTs acting as external devices are to be hosted by Docker installed on a student's personal computer and mobile device. Medical devices such as a pacemaker, glucose monitor, and temperature reader, will be used to demonstrate smart aspects of IoT data handling by the hospital network, such as IoT security, monitoring, notifications, and use of analytics.

This document describes in full the demonstration plan, step-by-step setup instructions, custom IoT scripts, and technologies utilised.

Sections 1 to 5 document the preparation and installation instructions of the virtualised IoT demonstration network.

Section 6 documents the cybersecurity aspects of the IoT setup.

Section 7 breaks down the architecture of the demonstration plan, along with an improv script for the live demonstration.

1. Preparing the virtualised Cloud9 & Docker environments

To achieve the simulation of smartly managed IoT devices, two sources of virtualised environments are utilised. Both are running Ubuntu Linux. This is important as Linux allows the easy installation of the required Python3 language and AWS IoT SDK.

- I. **AWS Cloud9 environment.** Simulating on-premises IoTs, of the internal hospital network. This Cloud9 instance is an EC2 server hosted on the same cloud as the main hospital network demonstration.
- II. **Docker Container environment.** Simulating external IoTs, at the edge of, or remotely located from the hospital. Docker is installed on a student's home computer, and mobile device.

I. Set up the AWS Cloud9

The first step is to launch the AWS CloudFormation template, this automates several setup steps of the Cloud9 environment.

Log in to AWS Cloud using an IAM account with a high-level role such as PowerUserAccess, or with the root user, then launch this instructive setup link:

<https://catalog.us-east-1.prod.workshops.aws/workshops/6d30487a-48e1-4631-b6bc-5602582800b5/en-US/chapter2-lets-begin/10-step1>

Follow the setup instructions and create the CloudFormation stack, which contains the Cloud9 environment. During this process, name this instance as *Medical-IoT-Instance* and name the workspace as *iot-folder*.

Launch the Cloud9 IDE, which allows you to write, run, and debug a virtualised IoT, see Figure 1 below. Word of caution, this deploys a S3 server which is charged per hour the Cloud9 IDE is opened for use.

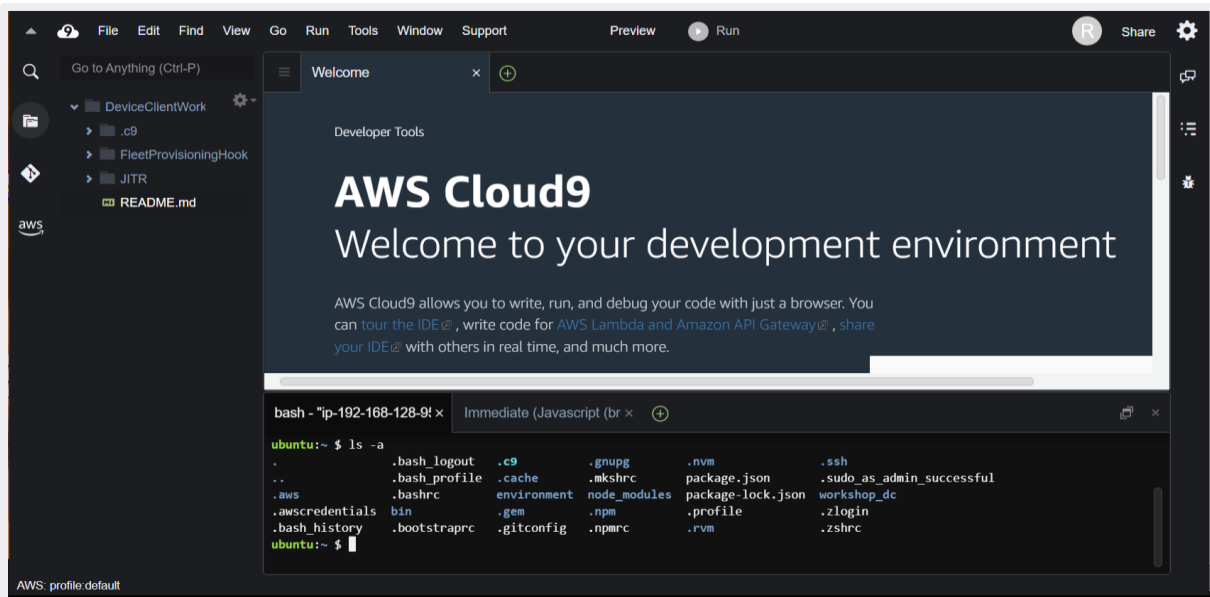


Figure 1. AWS Cloud9 IDE environment, hosting a virtualised IoT.

II. Set up Docker Containers

The first step is to download and install Docker on a personal device, from <https://www.docker.com/>.

Docker is an open-source platform that can easily create virtualised containers. For this demonstration Docker allows the rapid deployment of Linux environments that will host simulated IoT device, deployable from our personal computers or portable devices.

Launch the Docker program and then also open the host computer's Windows, or Linux command line interface (CLI), such as PowerShell or Terminal. As the IoTs are just virtual representation of physical devices, only a headless deployment of the Linux container is necessary.

1. Test Docker is running, in the CLI run the command:
 - ❖ `$ docker ps`
2. Then pull down the Ubuntu image for Docker:
 - ❖ `$ docker pull ubuntu`
3. Next, create two Docker containers using the following commands, when done the new containers should appear in the Docker container program window (Figure 2):
 - ❖ `$ docker run -td --name pacemaker-iot ubuntu`
 - ❖ `$ docker run -td --name diabetes-iot ubuntu`

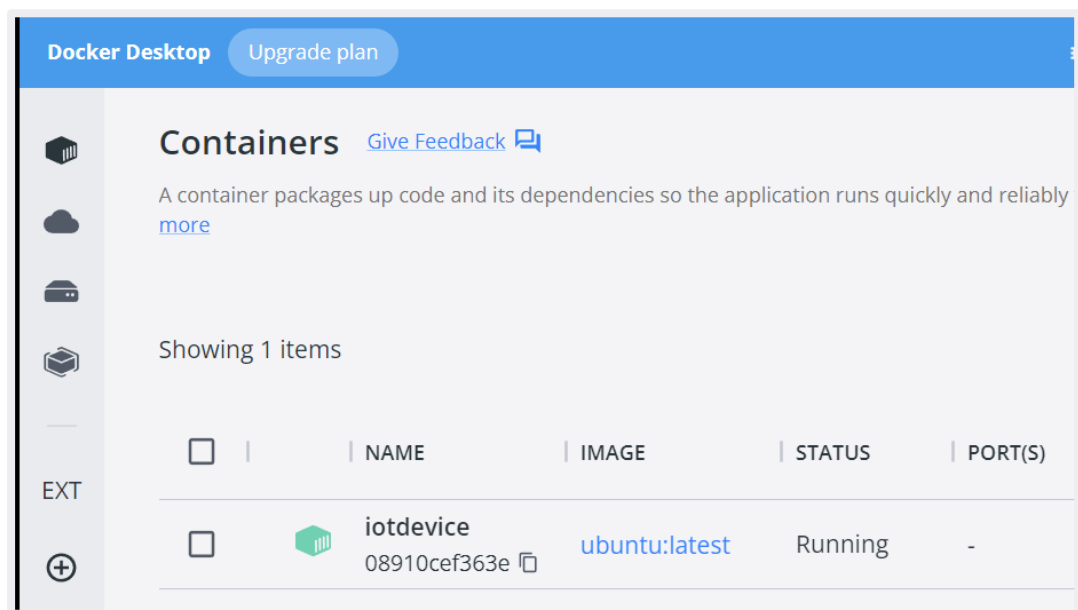


Figure 2. Docker can rapidly launch Linux containers, with which to host our virtualised IoTs.

4. Now open two new and separate CLI terminals, for logging into each newly made container with bash. Linux and the file structure for each IoT environment is now accessible:
 - ❖ `$ docker exec -it pacemaker-iot /bin/bash`
 - ❖ `$ docker exec -it diabetes-iot /bin/bash`

Three IoT environments are thus created: Medical-IoT, Pacemaker-IoT, Diabetes-IoT.

2. Preparing AWS IoT Core, registering Things

The AWS IoT Core will be set up to manage the IoT devices, using the IoT Device Manager. This allows our IoTs to communicate to the Amazon Cloud via a message broker server called the MQTT. Essentially the message broker updates the status of the IoT registered in the AWS IoT Core, with every message it sends.

First, we create a “Thing” that represents an AWS-recognizable “identity” for each IoT registered wanting to communicate with AWS IoT Core. These *Things* provide the policies, security configuration, certificates, encryption keys and rules for communicating with each virtualised IoT device.

IMPORTANT! For the entirety of this document’s execution, and the live demonstration, make sure the correct region is selected: **Asia Pacific (Sydney) ap-southeast-2**. This is always selectable in the top right corner of the AWS browser interface (Figure 3).



Figure 3. AWS console bar. Selected region should always be Sydney.

I. Create the IoT Policy

First we are required to set up a policy that allows communication with IoT devices:

1. From the AWS Dashboard, navigate to the “IoT Core” page. In the sidebar, navigate to *Manage > Security > Policies*.
2. Click on the “*Create policy*” button.
3. For the Policy name enter “*Hospital-IoT-Policy*”.
4. For the Policy document, click the “JSON” tab and replace with the following string:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
```

5. Click “Create” and the new policy will be listed. This policy allows ingestion of all IoT formatted JSON messages.

II. Register IoT Things

Then for each virtualised device set up in the previous stage, register a paired Thing:

1. From the AWS IoT Core page, in the sidebar, navigate to *Manage > All Devices > Things*.
2. Click on the “*Create Things*” button.
3. Choose “*Create single thing*”. Next.
4. For each Thing, set the name to match an IoT device (Medical-IoT, Pacemaker-IoT, Diabetes-IoT). Next.
5. Choose “Auto-generate a new certificate”. Next.
6. Select the “Hospital-IoT-Policy” just created previously, then click “Create Thing”.

7. Important! In the proceeding “Download certificates and keys” section, download the following files for each IoT, and store them in a safe place for use in the next stages:
 - a. Device Certificate ending in *pem.crt*
 - b. Public key file ending in *public.pem.key*
 - c. Private key file ending in *private.pem.key*
 - d. RSA 2048 bit key: *Amazon Root CA 1*
8. Finally, click “Done” and the new IoT Thing is created. Figure 4 shows the demonstration Things registered:

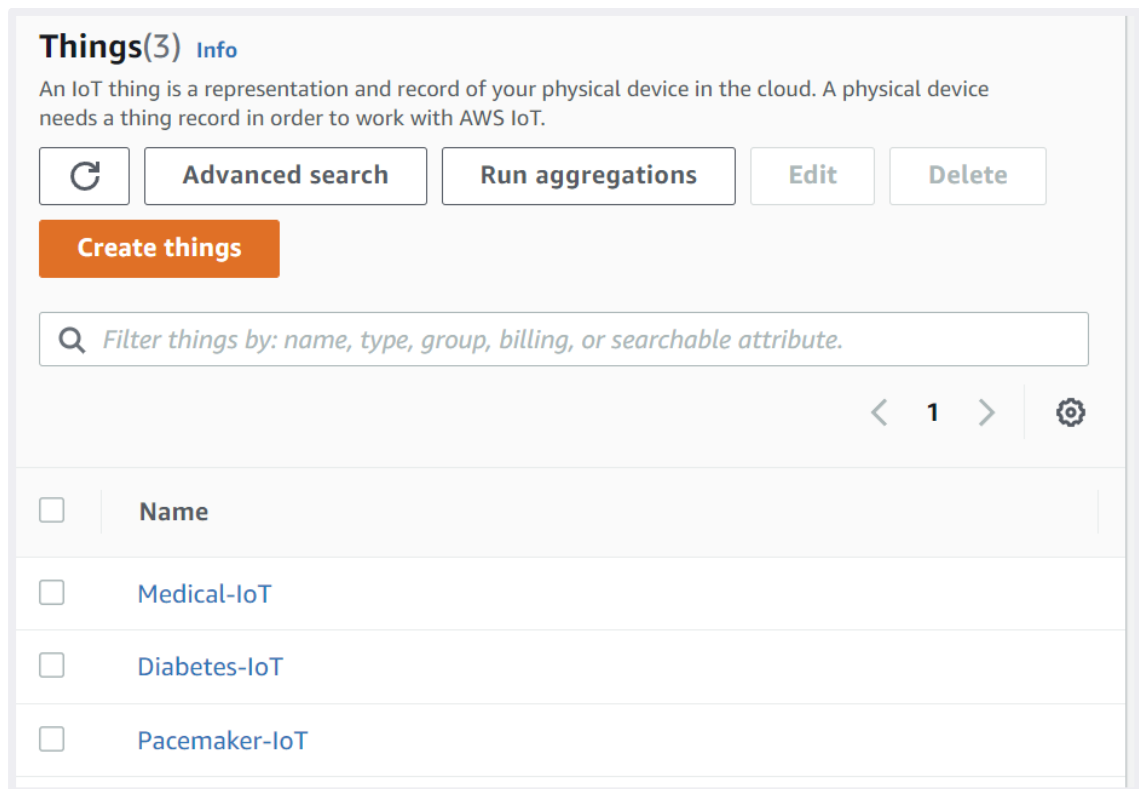


Figure 4. The IoTs are registered on the AWS cloud as Things.

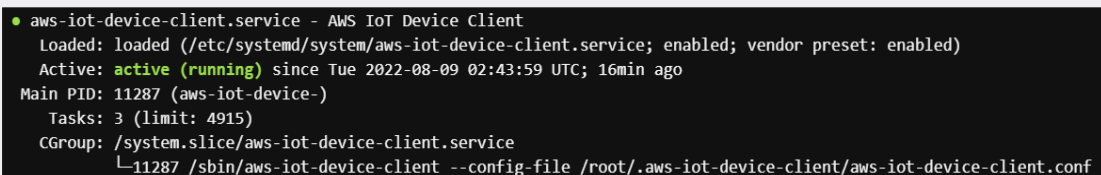
3. Preparing IoT devices with AWS IoT SDK

All the virtualised IoT device environments prepared earlier require the AWS IoT SDK, python3, encryption keys and certificates, to securely communicate with the AWS IoT Core environment in the cloud.

I. Prepare the Cloud9 IoT environment

Back in the AWS dashboard, launch the Cloud9 instance created earlier named *Medical-IoT-Instance*. Here a virtualised Medical-IoT environment is prepared. In the Cloud9 IDE terminal and directory structure, execute the following:

1. Create a directory to store the Medical-IoT certificates and keys and set permissions.
 - ❖ `$ mkdir /home/ubuntu/iot-folder/certs`
 - ❖ `$ chmod 700 /home/ubuntu/iot-folder/certs`
2. In the Cloud9 IDE left-side navigation pane, navigate to the folder `/home/ubuntu/iot-folder/certs`. In the top menu bar, click on *File - > Upload Local Files*, and upload all 4 certificates downloaded earlier for the Medical-IoT device.
3. Set security permissions and create a file to temporarily store MQTT messages.
 - ❖ `$ chmod 600 /home/ubuntu/iot-folder/certs/*-private.pem.key`
 - ❖ `$ chmod 644 /home/ubuntu/iot-folder/certs/*-certificate.pem.crt`
 - ❖ `$ chmod 644 /home/ubuntu/iot-folder/certs/AmazonRootCA1.pem`
 - ❖ `$ touch /home/ubuntu/iot-folder/subfile.txt`
 - ❖ `$ chmod 600 /home/ubuntu/iot-folder/subfile.txt`
4. Install the AWS IoT Device Client. If successful, the process should end with the IoT Device Client up and running, see Figure 5 below.
 - ❖ `$ cd /home/ubuntu/iot-folder`
 - ❖ `$ chmod ugo+x build_execute.sh`
 - ❖ `$./build_execute.sh`



```
● aws-iot-device-client.service - AWS IoT Device Client
Loaded: loaded (/etc/systemd/system/aws-iot-device-client.service; enabled; vendor preset: enabled)
Active: active (running) since Tue 2022-08-09 02:43:59 UTC; 16min ago
Main PID: 11287 (aws-iot-device-)
Tasks: 3 (limit: 4915)
CGroup: /system.slice/aws-iot-device-client.service
└─11287 /sbin/aws-iot-device-client --config-file /root/.aws-iot-device-client/aws-iot-device-client.conf
```

Figure 5. AWS Device Client running on the Medical-IoT Cloud9 instance.

5. In the Cloud9 bash terminal, install Python3, Pip3, and AWS IoT SDK. Note that python may already be installed.
 - ❖ `$ apt-get install python3`
 - ❖ `$ apt-get install pip3`
 - ❖ `$ pip3 install AWSIoTPythonSDK`(Note: the AWSIoTPythonSDK may already be installed)

II. Prepare the Docker IoT environment

On the PC, laptop or personal device where Docker is set up, make sure Docker and the IoT containers are running. Note that `sudo` may be required for some commands. Here the *Diabetes-IoT* and *Pacemaker-IoT* environments are prepared.

1. Using two separate PowerShell CLI terminals, portal into each IoT instance with `bash`. (skip this step if still logged in from previous stage).
 - ❖ `$ docker exec -it pacemaker-iot /bin/bash`
 - ❖ `$ docker exec -it diabetes-iot /bin/bash`
2. For each IoT, run the following commands to install Python3, Pip3, and AWS IoT SDK. (Note that python may already be installed).
 - ❖ `$ apt-get update`
 - ❖ `$ apt-get install python3`
 - ❖ `$ apt-get install pip`
 - ❖ `$ pip3 install AWSIoTPythonSDK`
3. Next create a directory on each respective IoT image, to store certs, keys, and python script:
 - ❖ `$ mkdir /home/pacemaker`
 - ❖ `$ mkdir /home/diabetes`
4. Open a third PowerShell or CLI window. For each IoT, change to the directory on your computer where the respective certificates and keys are stored. For example:
 - ❖ `$ cd c:\capstone-project\demonstration\IoTs\pacemaker\certs-and-keys`
5. Then copy over the certificates and keys into each respective IoT docker instance, using the format `$ docker cp .\filename pacemaker-iot:/home/pacemaker`. For example, copy commands for the Pacemaker-IoT:
 - ❖ `$ docker cp .\0804dc1e2492faa91e762a9d41e51-certificate.pem.crt pacemaker-iot:/home/pacemaker`
 - ❖ `$ docker cp .\0804dc1e24970a91e762a9d41e51-private.pem.key pacemaker-iot:/home/pacemaker`
 - ❖ `$ docker cp .\0804dc1e24970f8a3fb39d391e762a9d41e51-public.pem.key pacemaker-iot:/home/pacemaker`
 - ❖ `$ docker cp .\AmazonRootCA1.pem pacemaker-iot:/home/pacemaker`
6. On the CLI or Powershell of each Docker IoT instance, check the files copied over successfully. Example:
 - ❖ `$ ls /home/pacemaker`

4. Preparing Python Scripts, Device Shadow, and MQTT messages

The intended goal of this demonstration is to send and receive messages in real-time from virtualised IoT devices, simulating real-world IoT network traffic. For this we use the AWS provided Message Queuing Telemetry Transport (MQTT) broker, which is a publish-subscribe format network communication protocol.

For our purposes we are using the MQTT to simulate messages between hospital IoT devices and the AWS cloud, to then process this data with analytical displays.

I. Prepare the Python Scripts

A simple python program is created to run on each IoT, to simulate the JSON formatted messages that the MQTT broker handles. The python scripts are configured to use *AWSIoTPythonSDK* and *AWSIoTMQTTShadowClient*. These libraries facilitate the implementation and handling of the certificates and keys required for secure connection and communication with the IoT's registered Thing on the AWS cloud.

Now copy over the python scripts to each respective virtualised IoT environment:

1. **For the AWS hosted IoT:** In the AWS Cloud9 IDE window, upload the *Iot-medical-program.py* to the cert folder created earlier */home/ubuntu/iot-folder/certs*. In the top menu bar, click on *File - > Upload Local Files*.
2. **For the Docker hosted IoTs:** In the 3rd PowerShell or CLI window previously open, change directory to the python script folder on computer. Then copy over the python script for each respective IoT device:
 - ❖ `$ docker cp .\Iot-pacemaker-program.py pacemaker-iot:/home/pacemaker`
 - ❖ `$ docker cp .\Iot-diabetes-program.py diabetes-iot:/home/diabetes`

Table 1 catalogues the python scripts created for this capstone project.

| Script | Role | Output |
|----------------------------|----------------------|---|
| iot-pacemaker-program.py | Pacemaker | A heartbeat rate, from a randomly generated safe range (80-110), is sent as a message every 15 seconds. |
| iot-diabetes-program.py | Glucose Monitor | A blood sugar reading, from a randomly generated range, is sent as a message every 60 seconds. |
| iot-medical-program.py | Hospital Bed Monitor | A body temperature reading, from a randomly generated range, is sent as a message every 5 seconds. |
| pacemaker-alarm-trigger.py | Pacemaker | Simulates an SNS alert, to send out warning notifications that the patient's heart rate is very high. |

Table 1. Python scripts to run the IoT message simulation. For further insight, the AWS documentation contains examples of similar python code: https://docs.aws.amazon.com/code-samples/latest/catalog/code-catalog-python-example_code-iot.html

II. Configure AWS Things Device Shadows

Back in the AWS dashboard, navigate to the previously registered Things.

1. From the AWS IoT Core page, in the sidebar, navigate to *Manage > All devices > Things*.
2. One by one, open each Thing registered in the AWS IoT Core, and set up its Device Shadow state. A JSON formatted string allows AWS tools, such as SNS notifications, Analytics, and

MQTT subscription client, to ingest the virtualised IoT communications and meaningfully process the data. Then navigate to the bottom of each Thing's settings page and select the *Device Shadows* tab, then click *Create Shadow* (Figure 6):

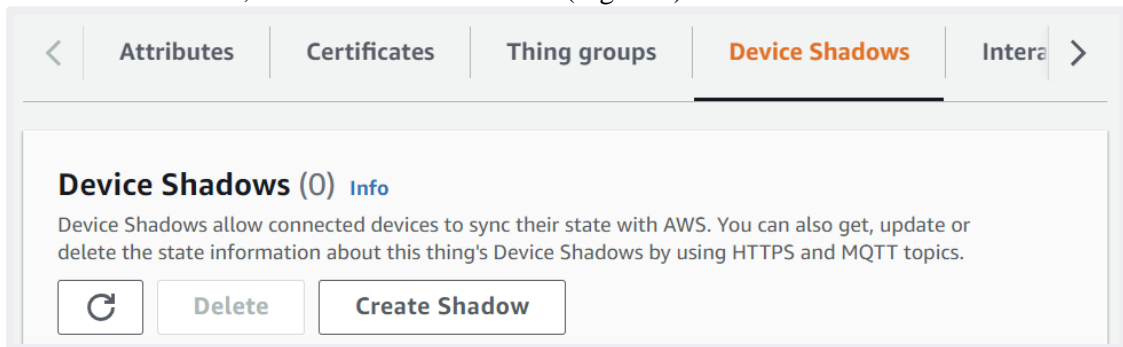


Figure 6. Create Shadow state for each IoT Thing.

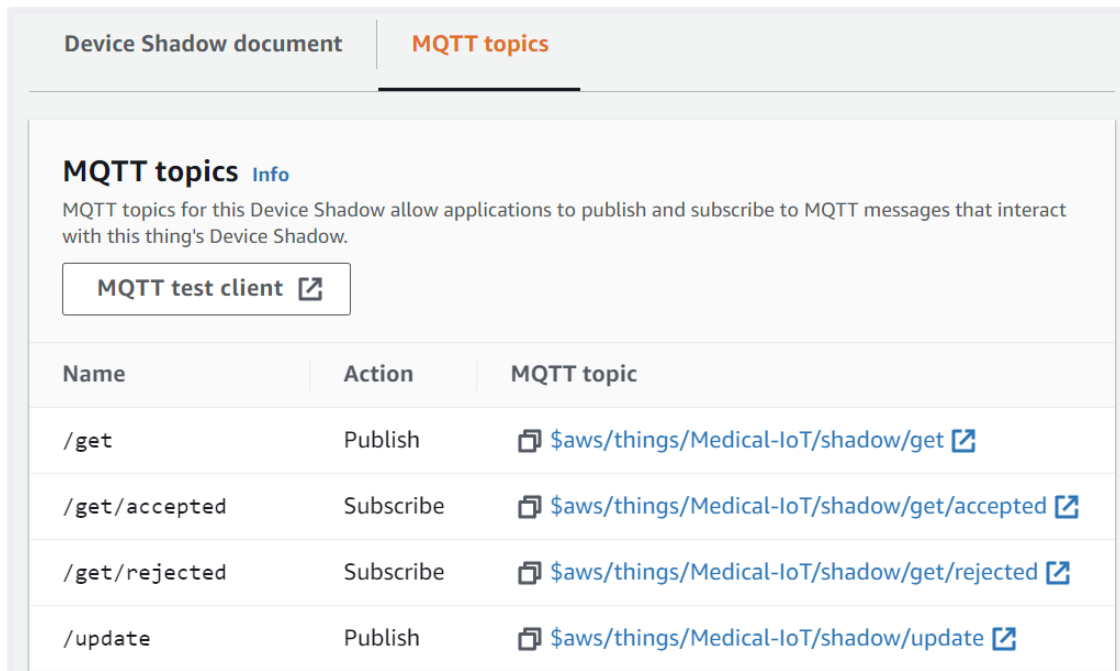
- For each of our IoT Things, choose the option *Unnamed (classic) Shadow*, select *Create*. There will now be a new Device Shadow listed, open it and choose *Edit* for the *Device Shadow document* option, then paste in the respective JSON string listed below in Table 2, and then click *Update*. Do not close these windows, MQTT testing is done in the following section.

| IoT Thing | Shadow State JSON |
|---------------|---|
| Pacemaker-IoT | <pre>{ "state": { "desired": { "welcome": "pacemaker-iot" }, "reported": { "welcome": "pacemaker-iot", "heartbeat": 0 } } }</pre> |
| Diabetes-IoT | <pre>{ "state": { "desired": { "welcome": "diabetes-monitor-iot" }, "reported": { "welcome": "diabetes-monitor-iot", "glucoselevel": 0 } } }</pre> |
| Medical-IoT | <pre>{ "state": { "desired": { "welcome": "temperature-monitor-iot" }, "reported": { "welcome": "temperature-monitor-iot", "temperature": 0 } } }</pre> |

Table 2. JSON formatted Device Shadow state.

III. Test the MQTT messaging

Each Thing's Shadow Device state has now been configured. While still in the Classic Shadow configuration page, click on the *MQTT topics* tab (Figure 7). Each IoT thing is now subscribed to several message channels, such as */update* and */get*.

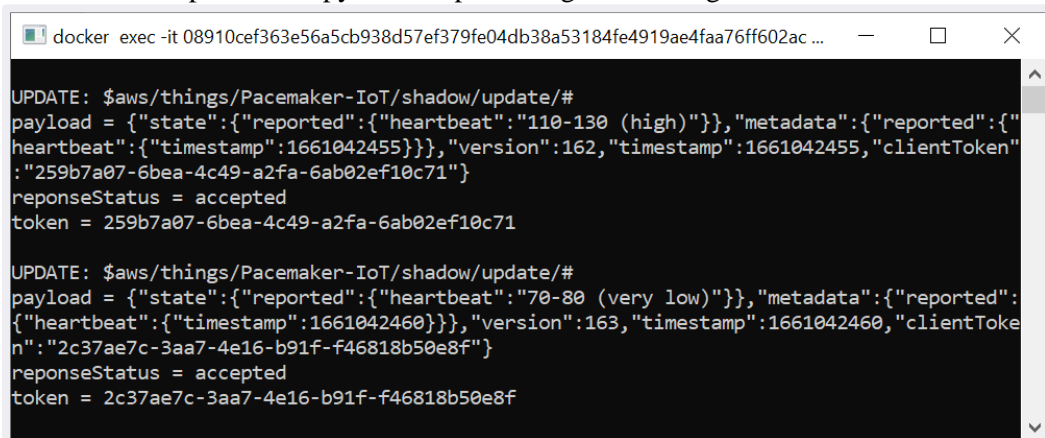


| Device Shadow document | | |
|--|-----------|--|
| MQTT topics | | |
| MQTT topics Info | | |
| MQTT topics for this Device Shadow allow applications to publish and subscribe to MQTT messages that interact with this thing's Device Shadow. | | |
| MQTT test client | | |
| Name | Action | MQTT topic |
| /get | Publish | \$aws/things/Medical-IoT/shadow/get |
| /get/accepted | Subscribe | \$aws/things/Medical-IoT/shadow/get/accepted |
| /get/rejected | Subscribe | \$aws/things/Medical-IoT/shadow/get/rejected |
| /update | Publish | \$aws/things/Medical-IoT/shadow/update |

Figure 7. Device Shadow automatically subscribes to certain MQTT message channels.

To test if the virtualised IoTs are communicating correctly through the AWS MQTT client:

1. One by one for each IoT, launch the MQTT test client located in the MQTT topics page (see previous Figure 7).
2. In both the Docker container's CLI, and the Cloud9 IDE bash terminal, navigate to the directory where the python scripts are located. For now, this should be */home/ubuntu/iot-folder/certs*.
3. Then run the python scripts, for example by using the command:
❖ `$ python3 iot-pacemaker-program.py`
4. The program will execute, securely connecting to the AWS IoT Core, using the asymmetrical key pair and certificate, and send a JSON formatted message through the MQTT broker. Figure 8 illustrates the pacemaker python script sending live messages.



```
docker exec -it 08910cef363e56a5cb938d57ef379fe04db38a53184fe4919ae4faa76ff602ac ...  
  
UPDATE: $aws/things/Pacemaker-IoT/shadow/update/#  
payload = {"state":{"reported":{"heartbeat":"110-130 (high)"}, "metadata":{"reported":{"heartbeat":{"timestamp":1661042455}}}, "version":162, "timestamp":1661042455, "clientToken": "259b7a07-6bea-4c49-a2fa-6ab02ef10c71"}  
reponseStatus = accepted  
token = 259b7a07-6bea-4c49-a2fa-6ab02ef10c71  
  
UPDATE: $aws/things/Pacemaker-IoT/shadow/update/#  
payload = {"state":{"reported":{"heartbeat":"70-80 (very low)"}, "metadata":{"reported":{"heartbeat":{"timestamp":1661042460}}}, "version":163, "timestamp":1661042460, "clientToken": "2c37ae7c-3aa7-4e16-b91f-f46818b50e8f"}  
reponseStatus = accepted  
token = 2c37ae7c-3aa7-4e16-b91f-f46818b50e8f
```

Figure 8. The python script sends heartbeat updates to the AWS IoT Core.

5. As the IoT Things are now configured with a shadow state, the virtualised IoT messages are now received by the MQTT client in the AWS IoT cloud (Figure 9).

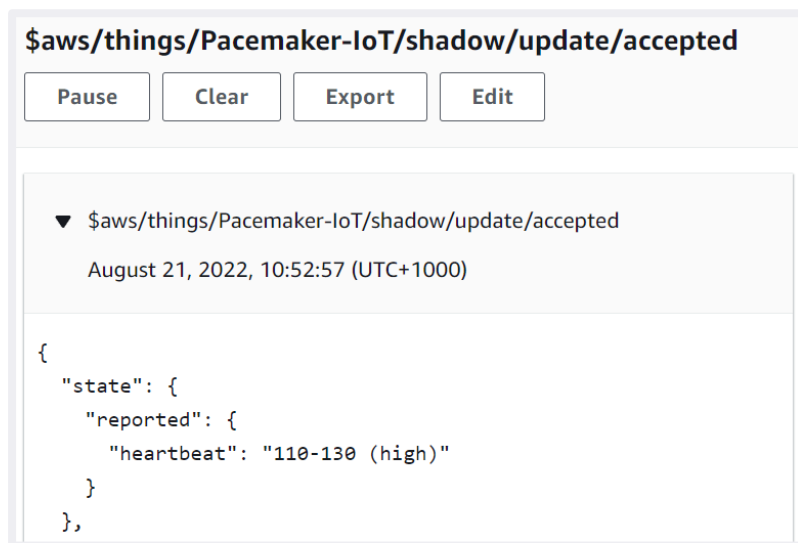


Figure 9. The Pacemaker-IoT Thing receives IoT state updates.

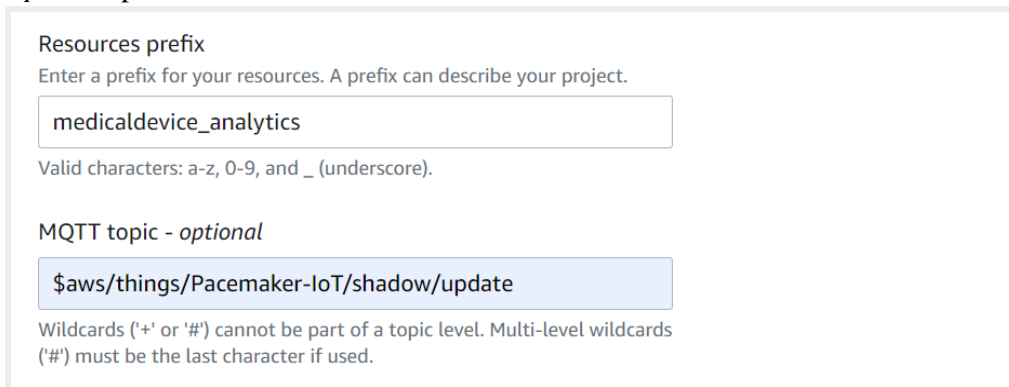
5. Preparing AWS IoT Analytics, ingesting IoT data

Analytics are a very effective tool in processing, understanding, interpreting, and communicating significant patterns and insights drawn from data. AWS has a suite of tools that takes advantage of their integrated IoT Analytics technology. For this demonstration, we only make simple use of IoT Analytics to display a chart that illustrates the data ingested from each registered IoT.

I. IoT Analytics quick setup

The following is simplified setup instructions for the IoT Things. If required, detailed instructions are available at the AWS Dojo analytics workshop (aws-dojo.com, n.d.).

1. Create an analytics resource for each IoT, in the Amazon console. Search for, or navigate to *IoT Analytics*. On the main Analytics page, launch the quick setup titled *Get started with AWS IoT Analytics* (Figure 10). For each IoT being set up with analytics, enter their respective */update* topic, created earlier.



Resources prefix
Enter a prefix for your resources. A prefix can describe your project.

medicaldevice_analytics


Valid characters: a-z, 0-9, and _ (underscore).

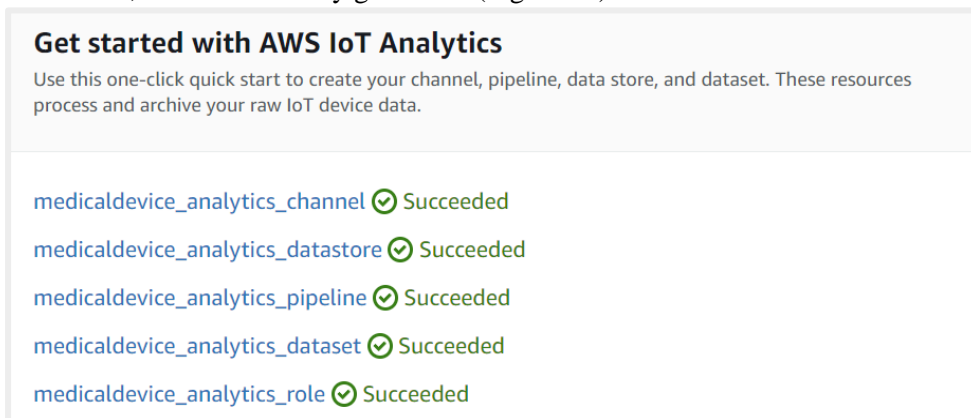
MQTT topic - optional

\$aws/things/Pacemaker-IoT/shadow/update

Wildcards ('+' or '#') cannot be part of a topic level. Multi-level wildcards ('#') must be the last character if used.


Figure 10. AWS IoT Analytics provide a quick setup function for existing IoT Things.


2. Next create a rule using a simple SQL query, that will send all the data captured by the IoT's shadow updates, to the new analytics channel *medicaldevice_analytics*. This query collects all message data at this point, to be refined at a later stage:
 `SELECT * FROM 'aws/things/Pacemaker-IoT/shadow/update'`
3. Next add an *Action* to this rule, choose 'Send a message to IoT Analytics'. This action will occur when the rule is triggered, and in this case all messages.
4. Finalise the Analytics quick setup. The channel, pipeline, datastore and dataset for the ingested IoT device, are automatically generated (Figure 11).





Get started with AWS IoT Analytics

Use this one-click quick start to create your channel, pipeline, data store, and dataset. These resources process and archive your raw IoT device data.

medicaldevice_analytics_channel  Succeeded

medicaldevice_analytics_datastore  Succeeded

medicaldevice_analytics_pipeline  Succeeded

medicaldevice_analytics_dataset  Succeeded


medicaldevice_analytics_role  Succeeded

Figure 11. The IoT Analytics quick setup creates the resources required.

- Next it is required to set up AWS SageMaker Notebook to associate the AWS IoT Analytics dataset that we just created with AWS Jupyter Notebook tool. On Amazon, search and browse to SageMaker cloud. On the main page, Choose Quick setup, then assign an amazon role from your Amazon account that has either PowerUserAccess or root permissions. Proceed to set up the Notebook, naming the instance such as *medicaldevice-instance* or *pacemaker-instance* (see Figure 12). Caution! When creating the Notebook, a medium tier 2 AWS cloud instance is generated and is charged at an hourly rate.

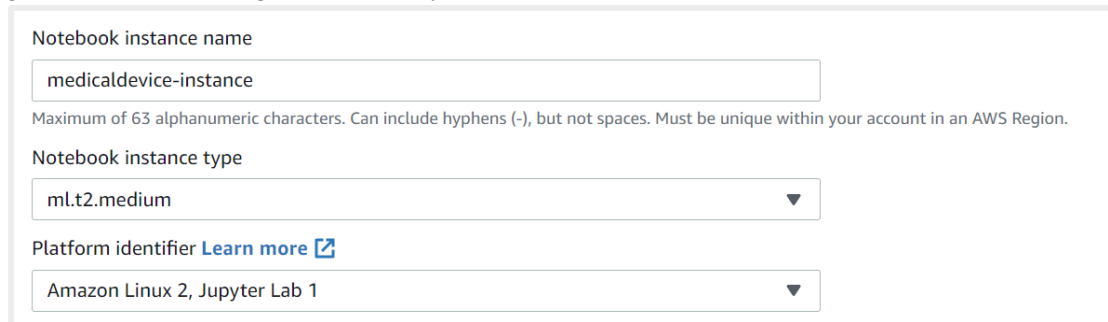


Figure 12. Setting up AWS SageMaker Notebook.

- Now that the Notebook service is running, the new notebook instance must be configured to use the dataset of ingested IoT messages from the MQTT topics, previously set up (Figure 13).

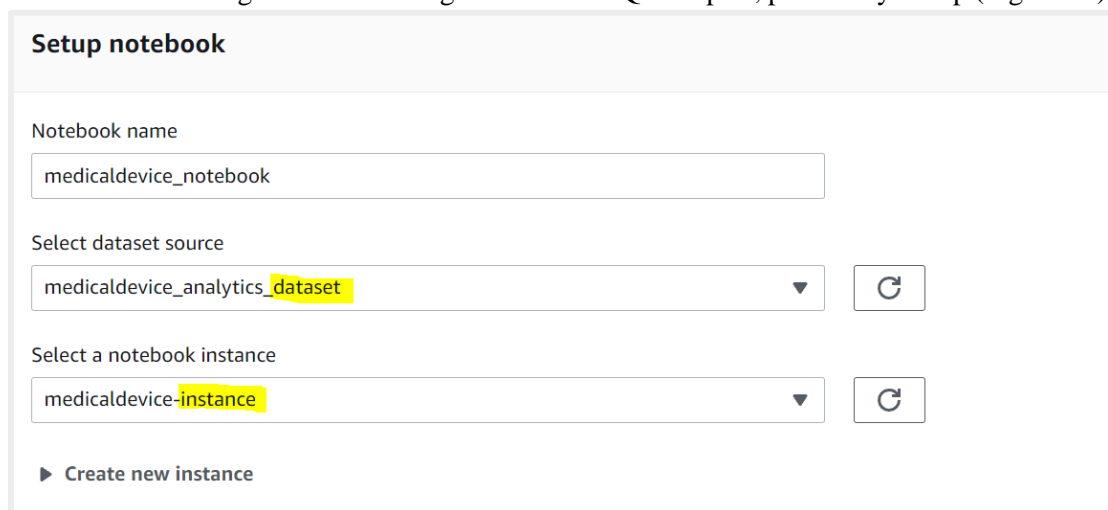


Figure 13. Create a notebook instance to pipe the dataset which contains the IoT's MQTT ingested messages.

II. Running the IoT Analytics

In order to run the IoT Analytics and output the resulting data in a meaningful way, simulated IoT traffic must be executed from our virtualised IoT devices on Docker and Cloud9, thus AWS Jupyter Notebook needs to be launched to execute Pandas chart plotting scripts.

- Just as was done to test the MQTT messages, using the Docker container's CLI and the Cloud9 IDE bash terminal, run the IoT python scripts. This will update the MQTT channel that the IoT devices are subscribed to, and thus will be piped into the *medicaldevice_analytics_dataset* just created. Modify this command to suit each IoT:
 - ❖ `$ python3 iot-pacemaker-program.py`

- Next, each dataset is required to be “Run” in order to process the data captured, and then filter it according to the SQL query supplied earlier (Figure 14).

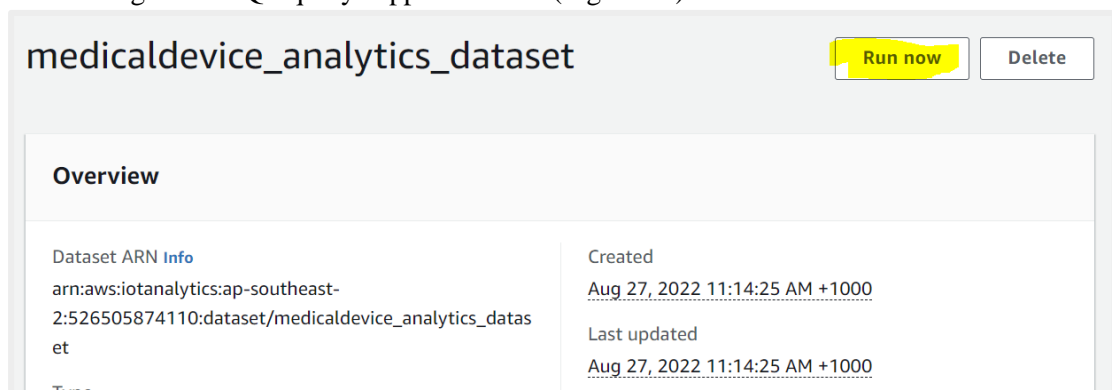


Figure 14. Datasets need to be processed before using, this can be automated.

- In the Amazon SageMaker main-page side panel, navigate to *Notebook > Notebook instances*. Here our running Notebook instances are found. Click on the Action ‘*Open Jupyter*’. The Jupyter Notebook lab opens in a new tab. If a popup appears with a ‘Kernel not found’ error - select the *conda_mxnnet_p36*, or similar, as the kernel and apply the *Set Kernel* button.
- Finally, the analytics are ready to be output as simple charts, using data from the IoT datasets. In the Jupyter Notebook cells, run the code in the following sequence (listed in Table 3). Adjust the resource and dataset names respective to each IoT being set up. Note some code may be pre-generated, or new cells may be required to run each step. This setup will be pre-configured for the demonstration, only requiring the dataset import to be refreshed.

| Step | Code to run in Jupyter | Purpose |
|------|---|---|
| 1 | <pre>import boto3 client = boto3.client('medicaldevice_analytics')</pre> | Creates a local boto3 analytics client. |
| 2 | <pre>dataset = "medicaldevice_analytics_dataset" dataset_url = client.get_dataset_content (datasetName = dataset) ['entries'][0]['dataURI']</pre> | Retrieves the dataset. |
| 3 | <pre>dataset_url</pre> | Prints the value of the retrieved dataset. |
| 4 | <pre>import pandas as pd import matplotlib.pyplot as plt df = pd.read_csv(dataset_url, header=0) df</pre> | Imports pandas and the matplotlib needed for displaying the data. Then prints the dataset contents. |
| 5 | <pre>df.plot(kind='bar',y='heartbeat', rot=0) plt.show()</pre> | Outputs the IoT data in a bar chart format. In this case, the heartbeat metric captured over time by the pacemaker IoT. |

Table 3. Pandas script to output IoT datasets into bar graph charts.

6. Implemented Security: AWS IoT Device Defender, Auditing, and Monitoring

IoT security is implemented in the hospital network design and demonstration portion of the cybersecurity project. IoT Device Defender is the AWS IoT security suite that secures IoT devices, with tools to identify and respond to security issues using auditing mechanisms, live monitoring, alarms, and customised alerts.

Defender compares cloud-side metrics, and device-side metrics, looking for anomalies, and expected value ranges. Rules detection behaviour helps detect when a device is operating outside its expected behaviour and is further combined with Security Profiles to monitor and analyse the devices.

All IoT device handling and communications are secured using:

- AWS IoT Defender
- Live IoT message and state Monitoring
- Triggers, Alerts, SNS notifications
- Behavioural Analytics, device auditing.

Encryption/cryptographic technology used for IoT device communication:

- AWS IoT facilitates the creating and registering of X.509 certificates, to securely authenticate all the medical IoT devices integrated into the AWS cloud. A private and public key pair is also generated for each registered IoT, allowing asymmetric cryptography. RSA and SHA algorithms are supported, up to 512 bit keys. Full certificate support is available, including expiration and reissuing management, and Certificate Authority handling.
- The Transport Layer Security (TLS) 1.2 protocol is used along with the X.509 certificates, to facilitate and secure the HTTPS communications of the IoT devices.

I. IoT Device Defender setup

For the demonstration, we set up some simple security auditing and monitoring of our virtualised IoT devices, using the AWS IoT Device Defender tools.

Here is the flow of how Defender works, and will be set up in the demonstration (Figure 15):



Figure 15. Defender utilises behaviour metrics and rules to create security profiles for monitoring IoT communications.

First a security profile is created, using rule-based behaviour metrics. These rules are command types of metrics used in IoT monitoring. However, for this demonstration the metrics are set with low tolerances, for the purpose of quickly triggering alarms through the IoT Defender monitoring suite.

1. In the AWS IoT sidebar control panel, navigate to *Manage > Security > Detect > Security Profiles*, and then choose *Create Rule-based Security Profile*.
2. Add the behaviour metrics and triggers rules prescribed in Table 4:

| Behaviour Metric | Datapoints required to trigger alarm, if the condition is false | Datapoints required to clear alarm |
|------------------------|---|------------------------------------|
| Authorization failures | Less than 2 | 5 Minutes |
| Connection attempts | Greater than 3 | 5 Minutes |
| Disconnects | Greater than 3 | 5 Minutes |
| Message size (bytes) | 128 | condition always exists |
| Messages received | Greater than 5 | 5 Minutes |
| Messages sent | Greater than 5 | 5 Minutes |

Table 4. Security behaviour metrics required for monitoring and auditing IoT Devices.

3. Our demonstration security profile and a Rule-based anomaly Detection profile is created, to allow monitoring of IoT behavioural metrics (Figure 16):

| Security Profile | Threshold type | Behaviors | Metrics retained | Target |
|------------------------------|----------------|-----------|------------------|-------------------|
| Remote-IoT-Device-Monitoring | Rule-based | 7 | - | Remote-Access-IoT |

Figure 16. Behaviour metrics are used to monitor IoT communications.

4. Navigate to *Manage > Things*, then open each medical IoT device to activate the monitoring. At the top of each Thing settings, there is now a prompt. Click on the *Automate IoT Security Audit* button (see Figure 17).

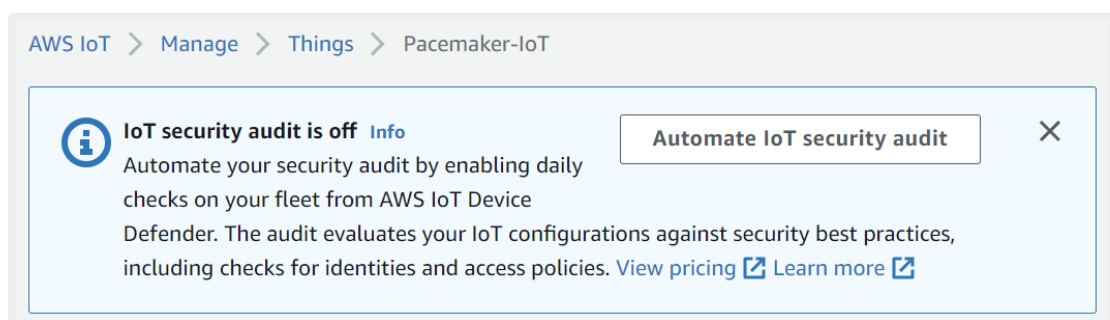
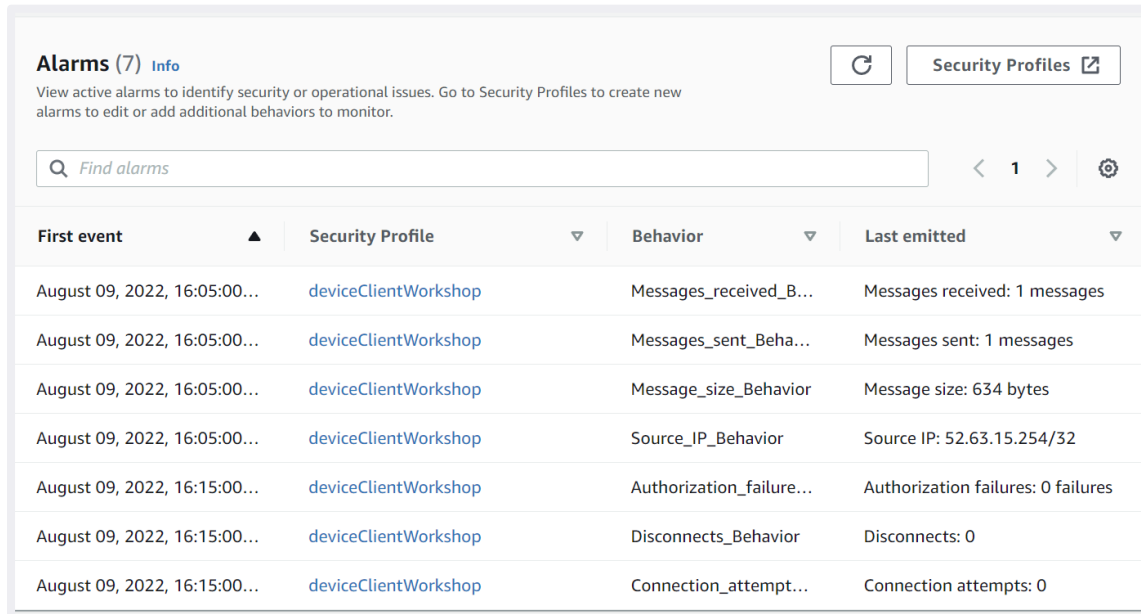


Figure 17. The security profile defines the monitoring rules. This can be turned off/on at any time.

For the demonstration, sensitive behaviour metrics are now configured to easily trigger alarms. For example, a misconfiguration in the connection protocol will cause an `Authorisation_failure` alert that sets off an alarm. Figure 18 illustrates the results of triggering several IoT Defender monitor alarms.



| First event | Security Profile | Behavior | Last emitted |
|------------------------------|----------------------|--------------------------|------------------------------------|
| August 09, 2022, 16:05:00... | deviceClientWorkshop | Messages_received_B... | Messages received: 1 messages |
| August 09, 2022, 16:05:00... | deviceClientWorkshop | Messages_sent_Beha... | Messages sent: 1 messages |
| August 09, 2022, 16:05:00... | deviceClientWorkshop | Message_size_Behavior | Message size: 634 bytes |
| August 09, 2022, 16:05:00... | deviceClientWorkshop | Source_IP_Behavior | Source IP: 52.63.15.254/32 |
| August 09, 2022, 16:15:00... | deviceClientWorkshop | Authorization_failure... | Authorization failures: 0 failures |
| August 09, 2022, 16:15:00... | deviceClientWorkshop | Disconnects_Behavior | Disconnects: 0 |
| August 09, 2022, 16:15:00... | deviceClientWorkshop | Connection_attempt... | Connection attempts: 0 |

Figure 18. Alarms trigger on the conditions set. Security or medical personnel can be notified.

II. SNS notifications setup

For the demonstration we create two SNS notification alerts, one to send an email, and one to send an SMS text, if the Pacemaker-IoT device reports a heartbeat rate over 140 BPM.

1. In the Amazon main console, search for and navigate to *Simple Notification Service*. In the side bar menu of Amazon SNS, navigate to *Topics*. Click *Create topic*.
2. Choose *Standard* type, *PacemakerAlert* as Name, and finalise the topic creation.
3. The new Topic is automatically opened, it now needs to be subscribed. At the bottom of the Topic page, click on *Create subscription*.
4. For the Topic ARN detail, use the correct AWS cloud instance zone, in this case the Sydney one `arn:aws:sns:ap-southeast-2:526505874110:PacemakerAlert`. For the protocol detail, select *Email*. For the endpoint enter the email address with which you desire to receive the alerts. Finalise the subscription creation, and then open your designated email to confirm the Topic subscription just created.
5. Next swap back to the AWS IoT dashboard to add a rule that defines what triggers the SNS email. Navigate to *Manage > Message > Routing Rules > Create Rule*.
6. Name the rule *PacemakerMonitorEmail*, then click *Next*. Add the following SQL statement query, which is the rule that defines when the SNS alert is triggered (Figure 19):

| Pacemaker heartbeat monitoring SQL statement |
|---|
| <pre>SELECT * FROM 'Saws/things/Pacemaker-IoT/shadow/update/accepted' WHERE state.reported.heartbeat > 140</pre> |

Figure 19. SQL statement used to define condition for triggering the SNS email alert. This can also be used for SMS and System Alert types.

7. In the *Next* settings, for the Rule actions, choose *Simple Notification Service (SNS)*. In the SNS topic, select the `arn:aws:sns:ap-southeast-2:526505874110:PacemakerAlert` created just before. Add a new role *Pacemaker-Monitor*, and then finalise the rule creation. This will automatically enable the rule and the SNS email event (Figure 20).



Figure 19. SNS monitoring is not enabled, this can be deactivated/activated at will.

8. Finally, switch to the CLI for the Docker container that hosts the Pacemaker-IoT, and run the `pacemaker-alarm-trigger.py` script provided. This script is configured to send a single message that trigger the SNS notification events. A single heartbeat rate of 141 BPM will be sent.
9. This will result in SNS email notifications as seen in previous testing (Figure 21).

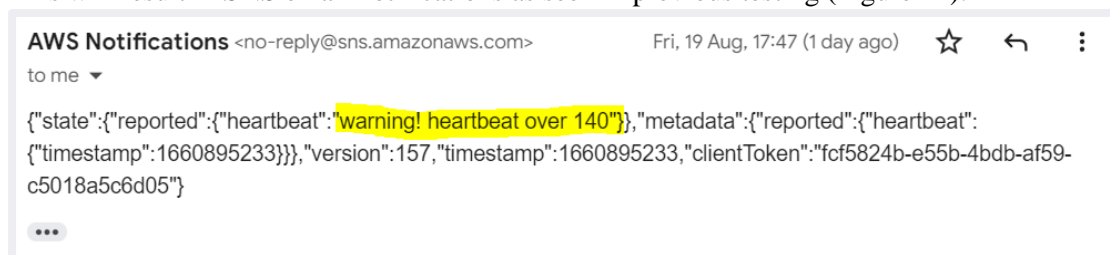


Figure 21. Example of an email received via an SNS notification alert event.

7. The Demonstration breakdown & script

The task was undertaken of setting up and running virtualised IoT devices on Amazon Cloud9 and computer hardware at home, such as a laptop and mobile phone.

Figure 22 is an overview of the IoT architecture and technology setup for the demonstration.

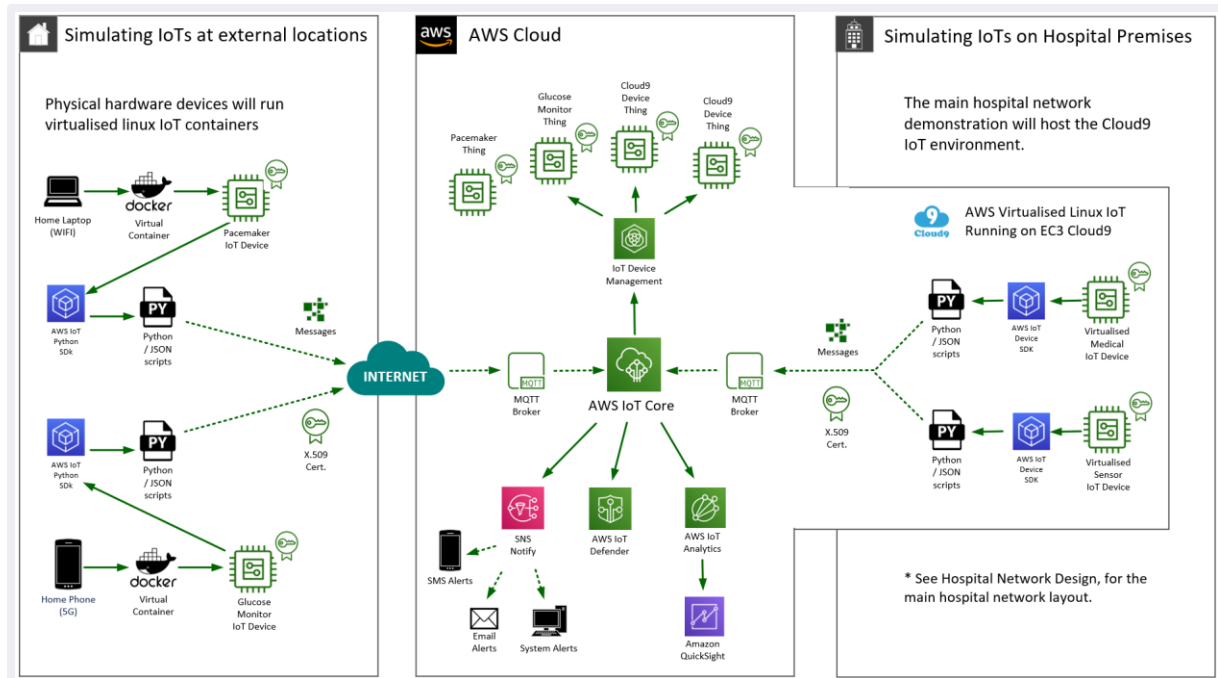


Figure 22. Design plans of the technology architecture set up for the smart IoT demonstration.

I. Demonstration breakdown

The AWS IoT Core will host all registered medical IoT devices and handle their communication through the MQTT broker. Additionally, AWS Defender, SNS, and Analytics will secure and intelligently process IoT data.

External medical devices. Externally simulated IoT devices (refer to the left panel in Figure 22), acting as ‘outpatient wearable’ and ‘emergency callout’ devices, will be hosted on a student’s laptop and mobile device, using locally hosted Docker containers running Ubuntu Linux environments. These images are configured with Python3, Pip3, and most importantly the AWS IoT Python SDK. Each virtual IoT also contains X.509 Certificates and Keys, and a customised Python3 script (refer to Section 4, Table 1) containing the communication protocols and JSON formatted messages required to securely communicate with the Amazon cloud. The pacemaker IoT configurations will be shown in the demonstration.

On hospital premises medical devices. Internally simulated IoT devices will be hosted on the main network hospital AWS demonstration setup, acting as ‘on-premises’ devices, running as AWS Cloud9 instances (refer to the right panel in Figure 22). As with the external IoTs, these on-premises IoTs will also be secured with X.509 Certificates and Keys, security protocols, and AWS IoT SDK. The medical device setup will also be briefly shown in the demonstration.

AWS IoT Analytics. Analytics are implemented in the IoT demonstration (refer to the middle panel in Figure 22), simulating smart data handling of medical devices and other IoT sensors. Amazon Notebook and SageMaker tools are used to set up the ingesting, storing, processing, and outputting of the

virtualised IoT data messages. The analytics data flow is shown in Figure 23, illustrating the key components in use. This will be demonstrated, with a bar chart showing the pacemaker IoT's readings.

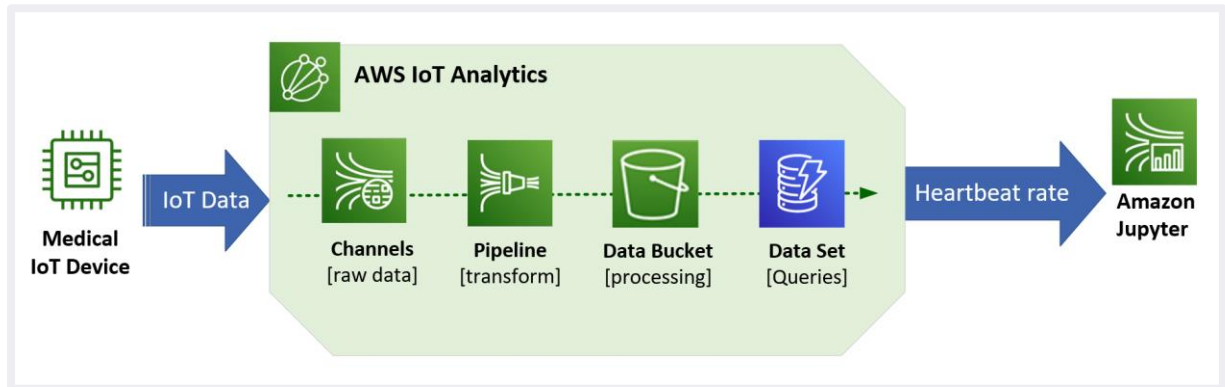


Figure 23. How the data flows from the IoT device to a graphical output in Amazon Jupyter, using the Analytics suite.

AWS SNS. Notifications are configured for IoT devices, such as a high heart rate from the Pacemaker-IoT will trigger an SMS text, an email, or a system message to be sent to the patient or the appropriate medical staff. A demonstration of this occurring will be shown.

AWS IoT Defender. Additionally, all devices are protected using the Defender security suite, and a demonstration of some live monitoring, and/or alarm tripping will be shown.

II. Demonstration script

For the live demonstration, James will introduce the core hospital network setup in the Amazon cloud, George will then transition to the smart deployment and handling of medical IoT devices, and Wayne will finish up by demonstrating the medical application he is developing.

Basic script outline for demonstration:

I. James demonstrates AWS hospital network.

II. George demonstrates AWS integrated IoT devices.

1. Intro (why use AWS IoT Core, benefits)
2. Phase 1 (quick overview of the actual IoTs setup)
3. Phase 2 (live observation of IoT data handling, analytics)
4. Phase 3 (live observation of security aspects)
5. Conclusion (challenges overcome, and how a real hospital could benefit using AWS IoT Core)

III. Wayne demonstrates medical application.

References

aws.amazon.com. (2021). Build a proof-of-concept IoT solution in under 3 hours with the AWS IoT Device Client | The Internet of Things on AWS – Official Blog. [online] Available at: <https://aws.amazon.com/blogs/iot/build-a-proof-of-concept-iot-solution-in-under-3-hours-with-the-aws-iot-device-client/> [Accessed 3 Sep. 2022].

aws-dojo.com. (n.d.). AWS Dojo - Free Workshops, Exercises and Tutorials for Amazon Web Services. [online] Available at: <https://aws-dojo.com/workshoplists/workshoplist35/> [Accessed 5 Sep. 2022].

docs.aws.amazon.com. (n.d.). Python Code Samples for AWS IoT - AWS Code Sample. [online] Available at: https://docs.aws.amazon.com/code-samples/latest/catalog/code-catalog-python-example_code-iot.html [Accessed 3 Sep. 2022].

docs.aws.amazon.com. (n.d.). Simulating Device Shadow service communications - AWS IoT Core. [online] Available at: <https://docs.aws.amazon.com/iot/latest/developerguide/using-device-shadows.html> [Accessed 3 Sep. 2022].