Prototype to Development: A cloud based centralized IoT platform for healthcare monitoring

Parnika Kaushik[1] *Rubani Bhatia*[2] *Vivek Chauhan*[3]

University of Waterloo[1], *University of Waterloo*[2], *University of Waterloo*[3],

Canada[1] *Canada*[2] *Canada*[3]

Abstract— Engineers can use IoT-based healthcare monitoring to create more personalized methods for analyzing their health and more logical approaches to disease management. In general, one of the most difficult aspects of prototyping an IoT device is the steep learning curve for traditional simulation tools like OMNET++. Storage and computing capacity can also be a significant limiting factor when simulating multiple devices at the same time with such tools. Prototyping operational monitoring and/or streaming analytics capabilities on a single machine is time-consuming, inefficient, and costly in terms of computation. To address these issues, the authors' propose a centralized solution for engineers to simulate IoT devices, as well as enhanced features such as analytics and reporting services. Furthermore, it is a service that monitors and audits devices for failures and notify concerned personnel of such failures.

Index Terms— AWS, IoT, Cloud-native, Cloud Computing, Healthcare Monitoring

---

1 Introduction

A medical practitioner can develop more individualized methods for analyzing a patient's health status and developing illness management strategies with the aid of IoT-based healthcare monitoring. Caretakers may use the remotely collected data points of interest to inform their care of sick or elderly patients. Due to recent advancements in wearable technology, these non-intrusive techniques have become very popular, although they have not yet reached their full potential. The recent improvements have been focused on unifying sensors and the cloud, but they have not succeeded in incorporating more individualized approaches to monitoring since they have not been translated into an end-to-end system, which requires extensive reworking at several levels. Due to its greater learning curve, the simulation component alone becomes a bottleneck.

Utilizing the cloud for the storage, processing, and privacy of massive amounts of data has been the main emphasis of earlier work in this field. Designing cloud native solutions that can accommodate the full range of IoT devices has not received much attention. Cloud orchestration offers enormous incentive to design and research on constructing towards such solutions, from simulating virtual devices to solutions ready for deployment. The Internet of Things (IoT), the most recent innovation in the present ICT evolution, is exemplified by the vast array of powerful smart devices that have begun to appear online. It is a type of global, dynamic network architecture made up of intelligent gadgets and sensors with self-configurability [14].

Recent developments have already shown how cloud computing can be used to perform data creation, processing, and visualization tasks to meet IoT requirements. Cloud computing has grown in popularity and reliability over the last few years by providing scalable, virtualized computation and data storage. The concept of cloud computing has evolved over the years. It began as a single virtualized datacenter and has since expanded into a larger network of linked, interconnected datacenters.

This project aims to create a centralized, scalable solution that is easily accessible and extensible by cloud engineers, ML engineers, and business analysts alike. It enables an end user (such as an IoT engineer) to change the data collection attributes of the simulated healthcare IoT devices. It also acts as a consolidated repository for query-able messages. The entire deployment is carried out on the AWS platform. On an existing personal AWS account, various AWS services such as AWS IoT Core (which serves as the main backbone), AWS IoT Analytics

(which provides streaming analytics capability), and Amazon QuickSight (which serves as an analytics and reporting dashboard) are used. The health monitors devices are assumed to record human blood pressure, heart rate and body temperature readings and these device level attributes are fixed.

2 Literature Review

[1] Botta et al discuss open issues and challenges towards the integration of cloud and IoT and [2] Nastic et al. discusses the challenges associated with realizing such unified systems. There are numerous existing simulators for wireless and distributed systems but lack focused treatment for deploying IoT systems in Cloud. They are not able to correctly describe the analytical, processing and storage requirements while designing the solution. The ability to centrally register, activate, monitor health of devices, and virtually audit the devices are mostly missing and not envisioned while prototyping which leads to redesigning the entire solutions.

[3] OMNeT++ is a well-established discrete event simulation environment, an all-purpose tool for simulating communication networks and distributed systems. Other specialized IoT simulators exist, but their approach is limited because they focus on architectural layers [4]. Han et al. [5] developed DPWSim, a simulation toolkit, to aid in the development of service-oriented and event-driven IoT applications with secure web service capabilities. SimIOT [6] was built on top of the SimIC simulation framework [7]. It provides several methods for simulating data exchange between an IoT sensor and the cloud, but it can only compute activity models. Zeng et al. [8] discuss the evaluation of big data analytical solutions for IoT through cloud simulation of Map-Reduce models during prototyping stages. Their solution is limited to applications that use MapReduce Model. [9] discusses the Sensor-Cloud Service Life-Cycle Model in the context of healthcare monitoring and the challenges that arise because of varying QoS requirements. The layered architecture provides abstraction for managing virtual devices and emphasizes the importance of a supervised sensor management system.

[10] proposes an Android-based simulator for IoT devices. It emphasizes the importance of simulating virtual devices as closely as possible to real-world devices and uses device traces transmitted over the internet to simulate real-world sensor data. It discusses a gateway solution that can be used in conjunction with MobIoTSim [10] to provide a sophisticated semi-simulated environment for analyzing IoT-Fog-Cloud systems. [11] describes a cloud-based platform for monitoring, collecting, and analyzing data from smart devices. The authors highlight the heterogeneity of data collected from servers and propose a generic cloud interface responsible for mediating between the cloud and middleware/applications. The interface manages data formats, security, and communication between devices and the cloud. [11], [12] propose a holistic platform for IoT related data aggregation, management and processing in Cloud but focuses on handling data heterogeneity and interoperability.

In [13], the authors concentrate on integrating edge and cloud computing. It creates a business and data-driven IoT platform. The central concept of the unified IoT platform is to process, communicate, and store aggregate data using a centralized approach for managed microservices architecture. They were able to deploy devices, manage communications, and store data on their prototyped platform MainFlux thanks to the microservice architecture. Security is a critical aspect that is addressed in the architecture described in the paper through authorization, authentication, and rules mapping services to services/devices. Any IoT platform should support security at multiple levels, including device-application, application-communication, and application-application. MainFlux system solutions are domain specific, and the platform is a minimalistic, in it supports time-series analytics and primary authentication via JSON web tokens which may not work for resource constrained devices. It does not support custom simulations, but it does demonstrate service unification for IoT applications. The abundance of work done to bring IoT devices closer to the cloud, as well as the opportunities for building a centralized cloud native solution, has been a driving factor for the authors' project.

3 Background

3.1 IoT Based Healthcare Monitoring

Even so, it fails to translate simulation efforts to actual deployment because these specifications are to be translated in the cloud domain. the authors' goal has been to provide an end-to-end cloud solution that allows the authors to simulate, test, and deploy in the same environment without having to change many things. After that, the virtual devices could be mapped to real sensor nodes, and the system would perform similarly. To

overcome these challenges, the authors' project demonstrates a centralized solution for an engineer to simulate IoT devices and provides enhanced capabilities such as analytics and reporting services. Furthermore, monitoring and auditing devices for failures and notifying relevant personnel of such failures would be considered a service.

The decision to use one of the popular cloud providers that are publicly available and have been successfully used by many businesses reduces the effort required to learn a new platform. The authors' decision was influenced by the possibility of incorporating the authors' approach. Of the most popular public clouds, the authors chose AWS because of the authors' prior experience with it and the vast amount of documentation that is publicly available for it. Its suite of services for IoT applications is expanding and could be used to serve most business use cases. The following sections goes over the specifics of the AWS services and how they were used for this project.

3.2 AWS IoT Core (IoT Backbone)

AWS IoT Core is a managed cloud service that allows connected IoT devices to interact with cloud applications and other IoT devices in a secure manner. It can handle many devices and messages, as well as process and route those messages to AWS IoT endpoints and other devices. Even when they are not connected, devices can interact with each other. The AWS IoT Core connectivity services enable secure communication with IoT devices and ma age messages sent between them and AWS IoT. [5]

The AWS IoT Core connectivity services enable secure communication with IoT devices and manage messages sent between them and AWS IoT. AWS IoT Core Registry organizes the AWS cloud resources associated with each device. Each device can be registered and assigned up to three custom attributes. Certificate Authority (CA) and device certificates can be created and associated with each device to improve management and troubleshooting and security. [5] MQTT topics are designed by end-users

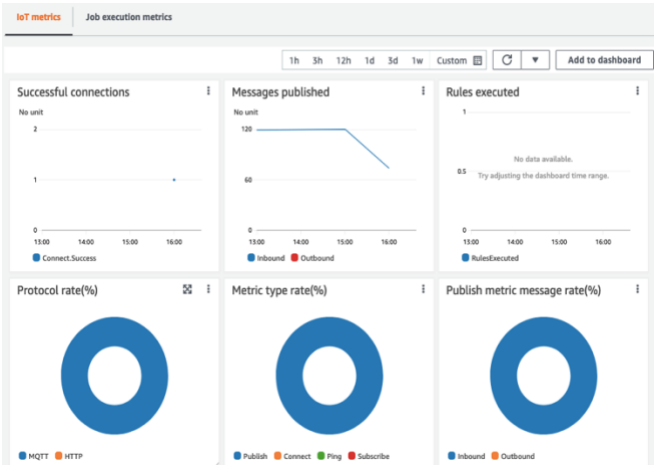


Figure 1: IoT device and other metrics viewable in IoT Core console

and the ATS (Amazon Trusted Security) endpoint is recommended for use when communicating to devices using the MQTT protocol.

Each AWS 'region' in each AWS account is unique and hence must be referenced correctly. The topic used for all devices is the same ('healthMonitors/readings/'). The details of the publishing of random data to the virtual devices created during this project are discussed in Section 4.4. IoT Core serves as the backbone for communicating with the virtual IoT devices, managing them centrally, pushing messages published to IoT device topics to IoT Analytics using IoT Core Rules. An IoT rule 'IoTAnalytics_hm_iot_analytics_channel' is the driver of pushing the messages to AWS IoT Analytics. In an SQL-like manner, the authors' selected all the attributes of the message payload to be pushed to the 'hm_iot_analytics_channel'. The AWS IoT rules engine forwards device data to other devices or other AWS services according to rules you define. It uses IAM roles to securely transfer data to its destination.

Its integrations with AWS IoT Analytics and QuickSight made it easier to setup the streaming data pipeline part of the project. Please refer to Figure 1 for a screenshot of the IoT device metrics visible in the IoT Core console.

3.3 AWS IAM (Identity and Access Management)

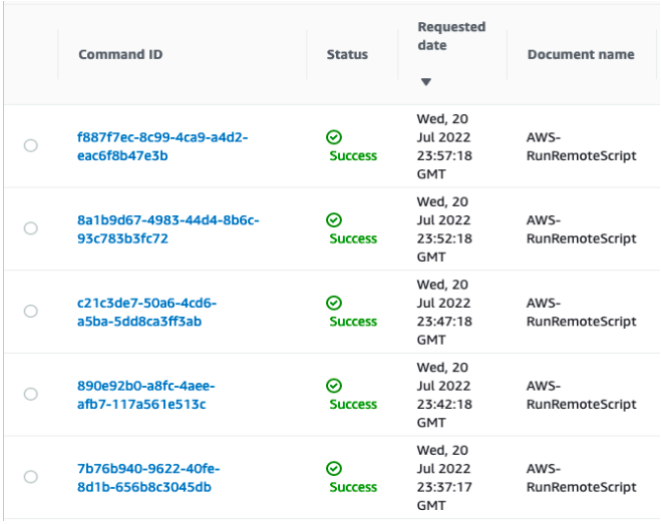
AWS Identity and Access Management (IAM) is an AWS service that allows an AWS account administrator to design the access permissions and abilities for several types of users who would need access to either the data in an AWS account or to the underlying AWS services themselves. Different IAM entities exist, namely IAM users, roles, groups, policies. The root account user grants an "almighty"-level access to an AWS account. It is because of this ability that AWS recommends not to use root account user security credentials like AWS Access Key ID and AWS Secret Access Key to assume any IAM role in the account. [3] The security credentials were not shared with anyone and were stored securely. Without the necessary permissions for all the IAM roles created, each process would run into authorization or access related errors.

3.4 AWS Systems Manager (Programming Enabler)

One can remotely and securely manage the configuration of managed nodes using Run Command, an AWS Systems Manager capability. Run Command is provided at no extra cost. [4] The Cloud9 instance is the managed node used by the authors. The authors used Run Command to create the customised IoT devices, validate the attributes' values and syntax, enable enhanced logging and auditing of the virtual IoT devices, and build the simulation cloud resources that will run on a user-defined schedule. AWSRunRemoteScript is a 'Document' used by AWS Systems Manager to run scripts stored remotely in either Github or AWS S3. AWS Systems Manager can be thought

of as a back-door entry into running scripts on any EC2 instance if the Systems Manager agent is installed on it and the IAM role associated with the EC2 instance has the necessary permissions.

Since the authors' store the virtual device creation script 'virtual_device_creator.sh' on the 'hm-iot-infra-bucket' S3 bucket, it was the most logical choice to choose this. Notifications of successful or failed invocations of each of the 'Document' runs are sent via email to the primary author's existing University of Waterloo email ID currently. Figure 2 provides the status of the outputs of each of the Bash Script runs to create and simulate the IoT devices.



Command ID	Status	Requested date	Document name
f887f7ec-8c99-4ca9-a4d2-eac6f8b47e3b	Success	Wed, 20 Jul 2022 23:57:18 GMT	AWS-RunRemoteScript
8a1b9d67-4983-44d4-8b6c-93c783b3fc72	Success	Wed, 20 Jul 2022 23:52:18 GMT	AWS-RunRemoteScript
c21c3de7-50a6-4cd6-a5ba-5dd8ca3ff3ab	Success	Wed, 20 Jul 2022 23:47:18 GMT	AWS-RunRemoteScript
890e92b0-a8fc-4aee-afb7-117a561e513c	Success	Wed, 20 Jul 2022 23:42:18 GMT	AWS-RunRemoteScript
7b76b940-9622-40fe-8d1b-656b8c3045db	Success	Wed, 20 Jul 2022 23:37:17 GMT	AWS-RunRemoteScript

Figure 2: AWS Systems Manager console - Successful simulation invocations

3.5 AWS S3 (Data/Scripting Repository)

Because Amazon S3 is an object storage service, data is stored in buckets as 'objects'. A file and any metadata that describes the file are considered objects. A bucket is a container that holds objects. To save data in Amazon S3, one must first create a bucket and specify a bucket name as well as an AWS Region. Then, as objects in Amazon S3, one must upload their data to that bucket. Each object has a key (or key name), which serves as the object's unique identifier within the bucket. [1]

For this project, four AWS S3 buckets have been created that serve different purposes. 'hm-iot-infra-bucket' was created as a storage bucket for storing the necessary programming scripts to ensure automatic device creation and simulations. It also serves as a repository to store the outputs of all the Bash scripts runs made using AWS Systems Manager service. It is the main storage repository for the purposes of device creation, simulation and setting up enhanced logging. Each of the buckets have encryption at rest enabled, thus keeping security in mind during all stages of development.

Additionally, three more buckets 'health-monitor-iot-analytics-datastore', 'health-monitor-iot-analytics-dataset' and 'health-monitor-iot-analytics-channel' were created using the AWS S3 console to serve different purposes for the streaming analytics pipeline setup. The naming convention that AWS requires for IoT Analytics pipeline is that an S3 bucket end with 'channel' only. 'health-monitor-iot-analytics-channel' is therefore a store for all messages that get pushed into the streaming analytics pipeline.

'health-monitor-iot-analytics-datastore' is used to store the outputs of the IoT Analytics Pipeline. The details of what the input and outputs of the IoT Analytics Pipeline are will be described in Section 3.6. 'health-monitor-iot-analytics-dataset' is used to store the SQL query set in AWS IoT Analytics Dataset section of the console.

3.6 AWS IoT Analytics (Streaming Analytics)

AWS IoT Analytics automates the process of analyzing data from IoT devices. Before storing IoT data in a time-series data store for analysis, AWS IoT Analytics filters, transforms, and enriches it. One can configure the service to collect only the data one requires from one's devices, process the data using mathematical transforms, and enrich the data with device-specific metadata such as device type and location before storing it. Then, one can run queries against one's data using the built-in SQL query engine, or they can perform more complex analytics and machine learning inference. Data visualization is also possible with AWS IoT Analytics thanks to integration with Amazon QuickSight. [2]

AWS IoT Analytics is completely integrated with AWS IoT Core, allowing it to receive messages from connected devices as they arrive. One can use the AWS IoT Analytics console to configure AWS IoT Analytics to receive messages from devices in various formats and frequencies via MQTT topic filters. AWS IoT Analytics validates the data and creates channels if it falls within the parameters one specifies. The channels are then routed to appropriate pipelines for message processing, transformation, and enrichment by the service. For faster retrieval and analysis, AWS IoT Analytics stores device data in an optimized time-series data store. AWS IoT Analytics stores the processed data as well as the raw ingested data so that one can process it later. AWS IoT Analytics includes a SQL query engine that allows one to run ad-hoc queries and receive results quickly. The service allows one to extract data from the data store using standard SQL queries. These queries can be reused even if the number of connected devices, the size of the device fleet, or the analytic requirements change. [2]

Before publishing data to a pipeline, a 'channel' collects data from a MQTT topic (here, 'healthMonitors/readings/') and archives the raw, unprocessed messages. Unprocessed messages are saved in an Amazon S3 bucket

called 'health-monitor-iot-analytics-channel'. A 'pipeline' takes messages from a channel and processes them before storing them in a data store. Pipeline activities perform transformations on the messages during the processing steps. Pipelines save the messages they process in a 'data store'.

A data store is a scalable and queryable repository of messages, not a database. A 'data set' is used to retrieve data from a 'data store'. One can use AWS IoT Analytics to create a SQL data set or a container data set. After creating a data set, the authors' use Amazon QuickSight to explore and gain insights into it. [2] For the purposes of the project, the 'hm_iot_analytics_channel' is the IoT Analytics channel created via the Amazon console manually. It uses the 'health-monitor-iot-analytics-channel' bucket to store the unprocessed messages as mentioned above under the 'pubsub/' prefix and uses the 'hm_iot_analytics_role' IAM role to perform the necessary actions. Without the necessary permissions, the channel would be unable to store the unprocessed messages in AWS S3 bucket. The channel was created manually using the Amazon console. Figure 3 shows the unprocessed JSON Gzip file of the messages publish to the MQTT Topic 'healthMonitors/readings/'

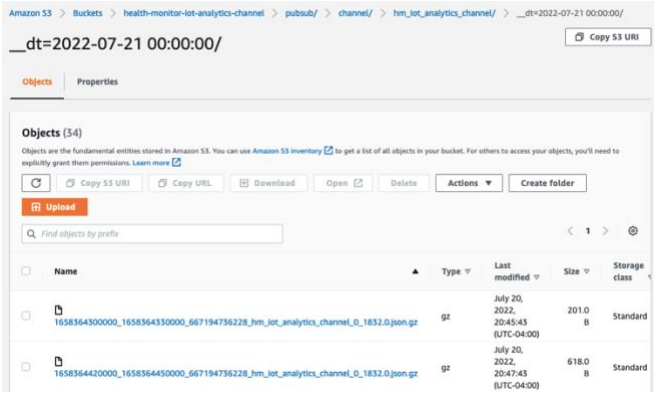


Figure 3: AWS S3 bucket hm_iot_analytics_channel with the unprocessed messages as JSON GZIP files

The RemoveAttributes activity was chosen in the Pipeline activity, which only selects specific attributes from each of the messages pushed from IoT Core to the IoT Analytics channel. The attributes selected by the 'hm_iot_analytics_pipeline' ingests streaming data from the 'hm_iot_analytics_channel' channel, only removes the 'vendorID' message attribute from the payload. This is also what is called a 'transformation' activity. All the other attributes are retained and then stored in the 'health-monitor-iot-analytics-datastore' S3 bucket.

An interesting quality of AWS IoT Analytics Pipelines is that they infer the attribute types from the messages published to the topic the channel ingests data from. The creation of the AWS IoT Analytics Pipeline was also done manually, through the Amazon console. Figure 4 shows the content for the Data Set created in the respective AWS console UI. The pipeline is streaming the necessary healthcare measurements along with the necessary attributes selected.

Result preview

c1249bf8-ae1a-4bec-be1d-47e9d4f06ca2.csv

bloodpressure	heartrate	bodytemperature	serialnumber	utctimestamp
182/100	139	42	19475	1658364140
148/83	69	35	15533	1658364439)
101/104	112	42	11020	1658364439)
126/123	168	37	12989	1658364439)
179/63	129	39	13978	1658364440)
94/108	90	42	19011	1658364440)
100/53	159	40	2240	1658364440)
115/71	67	36	30576	1658364440)
176/105	183	43	24973	1658364440)

Figure 4: The dataset created, after running SQL-like queries periodically on the 'hm_iot_analytics_datastore' S3 bucket through IoT Analytics

3.7 AWS Device Defender (IoT Device Security)

With the help of the security service AWS IoT Device Defender, you can audit your devices' settings, keep an eye on connected devices for unusual activity, and reduce security threats. IoT fleets can include a sizable number of diversely capable, durable, and dispersed geographically devices. An IoT device fleet setup is complicated and prone to error because of these factors. This also restricts the use of encryption and other forms of security on the devices themselves because devices frequently have limited computing, memory, and storage capacities. Additionally, devices frequently run software with known security flaws. Due to these issues, it is challenging to safeguard your device fleet and IoT fleets are appealing targets for hackers. AWS IoT Device Defender can audit device fleets to ensure they follow best practices for security and detect abnormal device behavior. [21]

An AWS IoT Device Defender audit examines account and device settings and policies to ensure security measures are in place and can assist in detecting any deviations from security best practices or access policies. [22] According to AWS's default levels, the authors chose the daily audit check, which checks for various CA, device level, and certificate level checks to run on all 'registered' devices. To receive the status of this check, the authors have also configured a notification to the primary author's University of Waterloo email ID. Refer to Figure 5 for the

different types of checks enabled for the daily audit checks.

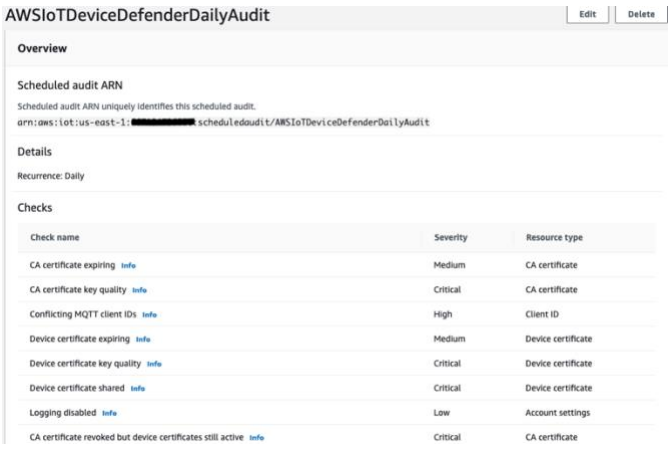


Figure 5: AWS IoT Device Defender Daily Audit Checks details

Refer to Figure 16 in Results section for the sample notification received from AWS after the daily audit checks runs on all registered devices. Refer also to Figure 10 in the Results section for a sample 'failed' audit check. If there is any mismatch between the keys or certificates can't be authenticated or other rules such as expiry dates etc. are not met, then device owners are notified, and device are set as unactive.

3.8 Amazon QuickSight (Analytics Dashboard)

Amazon QuickSight connects one's data to the cloud. It is a business intelligence tool that aggregates data from various sources into a single data dashboard [23]. It's an incredibly handy tool for tying the data streaming pipeline with a reporting and dashboarding capability because of how easily it integrates with IoT Analytics Data Store. The authors chose this service due to these factors and the results of the analytics performed on the data stored in the S3 based data store ('health-monitor-iot-analytics-datastore') are depicted in the Results section. It allowed convenient and easily comprehensible visualization of the simulated healthcare data published over time to the MQTT topic 'healthMonitors/readings/'.

4 Virtual Device Creation in AWS

Simulating virtual IoT devices is not a new concept. Several tools are available for running simulations on such virtualizations. However, the authors would like to concentrate on a cloud-native approach to performing this task. As opposed to worrying about the specifics of cloud-based implementations, the goal is to make the work of IoT engineers simpler and give them more time to focus on essential IoT tasks. The authors' project aims to develop a solution that not only creates virtual IoT devices programmatically, but also runs simulations at user-specified intervals and durations. The current solution does not allow them to edit device attributes, but they can change the initial values via the 'simulation_details.json' file. Users are only minimally exposed to the cloud when they can edit and upload the 'simulation_details.json' JSON file to a specific S3 bucket (the 'hm-iot-infra-bucket'). Bash, Python and AWS CloudFormation programming scripts are used to fully automate the creation, security, logging, auditing, and simulation processes. The user is not aware of the underlying infrastructure, which provides the necessary level of abstraction.

4.1 Orchestration Details

To create virtual IoT devices on the AWS cloud, a Bash script called 'virtual_device_creator.sh' was written. This script is stored in the 'virtual_device_creator' folder of an S3 bucket 'hm-iot-infra-bucket' and is called when the end user (IoT engineer) updates and uploads a 'simulation_details.json' file to the S3 bucket 'hm-iot-infra-bucket'. This 'simulation_details.json' file enables users to easily update device attribute values and edit simulation details such as the interval at which data should be published and how long the simulation should run. In addition, the simulation details are parsed and checked for syntax and range values. The bash script is executed on a Cloud9 EC2 instance using the Systems Manager Run Command command.

The upload action triggers a Lambda function 'RunVirtualIotCommand', previously created via Lambda console, which has exactly one relevant command utilizing AWS' SDK for Python (boto3), calls the Systems Manager Run Command command and calls on the 'virtual_device_creator.sh' file. The Cloud9 instance ID for the specified Cloud9 EC2 instance is essentially the 'target' where this Bash script runs. The Cloud9 instance is already configured appropriately to successfully utilize Systems Manager capabilities.

The 'virtual_device_creator.sh' file, references inside it, a 'runSimulation.json' AWS CloudFormation script, also developed by the authors, to create the necessary resources to perform the random data simulation on the virtual IoT devices on AWS. This file is stored under 'device_simulator' folder and is referenced using the AWS CLI command 'create-stack'[23]. It creates this stack, which consists of specific cloud resources to run simulations on the user-specified interval in the 'simulation_details.json' file. It currently sleeps for the user-specified 'duration' in the 'simulation_details.json' file and then deleted the stack created to stop the simulations. This not only stops the simulation at the appropriate time, but also saves on cloud-related costs (since the simulations are the primary source of cloud computational costs in this entire setup).

The AWS CloudFormation setup to create the required simulations uses AWS Lambda, AWS Events Rule and AWS Systems Manager, AWS Simple Notification Service to achieve the required functionality. The 'virtual_device_creator.sh' Bash script specifies the interval to the AWS Events Rule by parsing the 'simulation_details.json' 'interval' field which sets up the proper trigger for the AWS Lambda function SimulatorTriggerLambda. when creating the device simulation stack. This Lambda function uses a singular AWS Systems Manager Run Command command to call 'pub_sub.sh' file store under 'device_simulator' folder of the 'hm-iot-infra-bucket', so that it can run locally on the Cloud9 EC2 instance and notify users of its successful/failed attempts to publish to the MQTT Topic 'healthMonitors/readings/'.

Refer to Figure 14 in the Results section for the AWS architecture diagram for the detailed diagrammatic representation of the cloud orchestration details.

4.2 Creation and Registering Virtual IoT Devices

The 'virtual_device_creator.sh' Bash script serves as the primary driver to create and register IoT devices in AWS cloud. AWS recognizes all virtual devices as 'things'. All things require policies that will allow any interaction with the device, regardless of the choice of the communication client used. Without this policy attached, any communication or authentication attempt to a device will fail on access denied or authorization issues. Things associated with 'thing types' can have up 50 attributes. Hence a 'thing type' 'HealthMonitor', 'AllThingsPolicy' policy and 'thing group' 'HealthMonitors' are created first. The 'things' or virtual devices are then created and set to active, basis of the number of devices specified in the 'simulation_details.json' file ('numDevices' parameter) using specific AWS CLI commands. The 'things' are then added to the 'HealthMonitors' group and the necessary device level certificates and public-private keypairs are generated programmatically. These are currently stored locally on the Cloud9 EC2 instance. Although it would be best to keep these in a secret management tool, that is outside the scope of the current project. After creating the devices, they must be registered before they can be used to generate data for simulation. The newly created device is first added to the 'things' group, and its certificate and public-private keys are stored locally on the Cloud9 EC2 instance. These certificates and keys add an extra layer of security and are checked for rotation or security vulnerabilities on a regular basis as part of the daily audit checks described in Section 3.7. All 'things' are deactivated by default, and the device is created, secured, and registered once the appropriate certificate ID is associated with its corresponding thing and the policy is attached. The simulated device(s) can be verified on user AWS IoT Things console and their metadata reflects the attribute values as discussed earlier.

The 'virtual_device_creator.sh' Bash script also performs syntax and value checks on the 'simulation_details.json' file and errors on invalid or empty values. Each run of 'virtual_device_creator.sh' notifies the primary author of a successful or failure event via email. It also sets vendor ID, vendor name and serial numbers of each of the devices as arbitrary fixed values prior to calling the create-stack command mentioned previously. The assumption is that during a simulation, the vendor IDs and vendor names are non-alterable values and thus on each publish activity to the MQTT Topic 'healthMonitors/readings/', they should stay constant, while other health monitor readings should be simulated and randomized in each interval. The syntax and value range checks are performed on the basis of common knowledge of the format of human blood pressure, heart rate and body temperature measurements. Approximate lowest and highest values of each of these measurements was based on usual internet searches.

5 Random Data Simulation on AWS

The simulation of data to the virtual devices thus created requires appropriate data i.e. in the necessary format and matching the ranges assumed. The process uses Mosquitto MQTT client. The Mosquitto_pub command helped the authors publish the necessary data to the MQTT Topic 'healthMonitors/readings/' [24].

5.1 Generating and Publish Data To MQTT Topic

The following constraints are put on the user when editing the 'simulation_details.json' file: 'interval' is specified in minutes and must be between 2 and 60 minutes only; 'duration' must be greater than equal to 'interval', be specified in seconds and cannot exceed 3600 seconds; 'numDevices' cannot be greater than 50 devices; 'iotRoleArn' should not be edited in any form; 'UTCtimestamp', 'serialNumber' must be kept empty; 'bloodPressure' measurements must be in the standard "diastolic/systolic" mmHg format (e.g. 120/80); 'bodyTemperature' is specified in Celsius and must be between 35-43 degrees; 'heartRate' to be between 25-200. Any of these violations will results in errors and begin neither the device creation nor the device simulation parts of the project.

Simple Bash commands that specify the ranges of numbers for blood pressure, including the diastolic pressure range of 80–200, the systolic pressure range of 50–140, the body temperature range of 35–43, and the heart rate range of 25–200. These ranges were chosen based on a typical internet search and are based on the observed minimum and maximum values. The 'mosquito_pub' command was used to establish a connection to the MQTT ATS endpoint for AWS in the 'us-east-1' region for the individual AWS account once the JSON message payload had been entirely finalised, combining the various values as mentioned above, along with vendor and timestamp information. TLS version 1.2 was used to secure data transmission in transit. To ensure proper connection with the respective devices, the 'mosquito_pub' command specified the CA certificates, device certificates, and private keypair created and stored locally on the Cloud9 EC2 instance mentioned in Section 4.2. Refer to Figure 8 in the Results section for the results of the connection details when subscribed to the MQTT Topic 'healthMonitors/readings/'. Figure 9 in the Results section depicts the messages published to the MQTT Topic 'healthMonitors/readings/' for one sample interval.

6 Monitoring and Logging

All devices have enhanced device logging enabled via an AWS CLI command in the 'virtual device creator.sh' Bash script. All device-related activity logs are sent to AWS CloudWatch. AWS Cloudwatch is an AWS-provided standard monitoring and logging tool. All Lambda function executions are also logged in CloudWatch logs via IAM roles. All output and error logs generated by the AWS Systems Manager Run Command command are saved in the SSMOutputs/ folder of the 'hm-iot-infra-bucket'. Logs and outputs are also secure at rest since all S3 buckets have encryption at rest enabled. Thus, issue logging has been meticulously examined at each stage.

7 Analytics and Reporting

AWS IoT Analytics can query the data to answer analytical questions. AWS IoT Analytics Data Sets query the 'hm_iot_analytics_datastore' data store on a thirty-minute schedule. It runs a simple SQL-like, e.g. SELECT * FROM 'hm_iot_analytics_datastore' and creates the data set similar to a relational database. The contents of the dataset can be viewed in a familiar table format. The query can be altered to select fewer columns from the message payload so that the overall size of the dataset is reduced. For the scope of this project however, the authors have chosen to maintain the standard SELECT * query. Figure 6 shows the AWS IoT Analytics Datasets console with the query details selected by the authors. Figure 7 shows the AWS IoT Analytics Datasets console with the query cron schedule details selected by the authors.

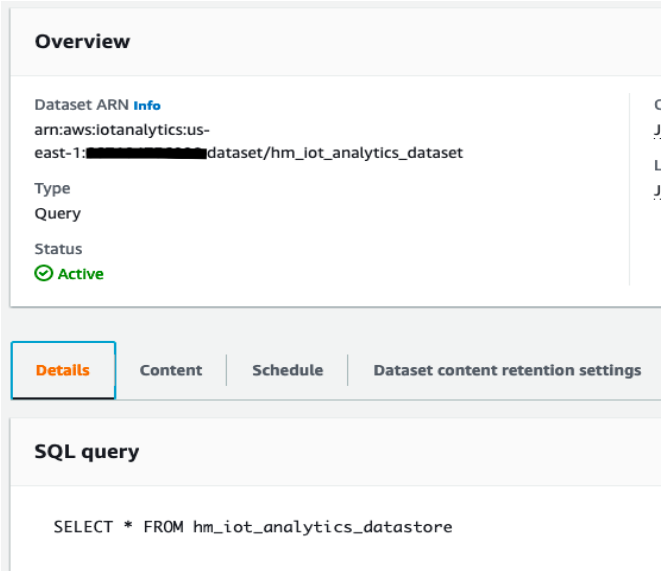


Figure 6: SQL-like query run on 'hm_iot_analytics_datastore'

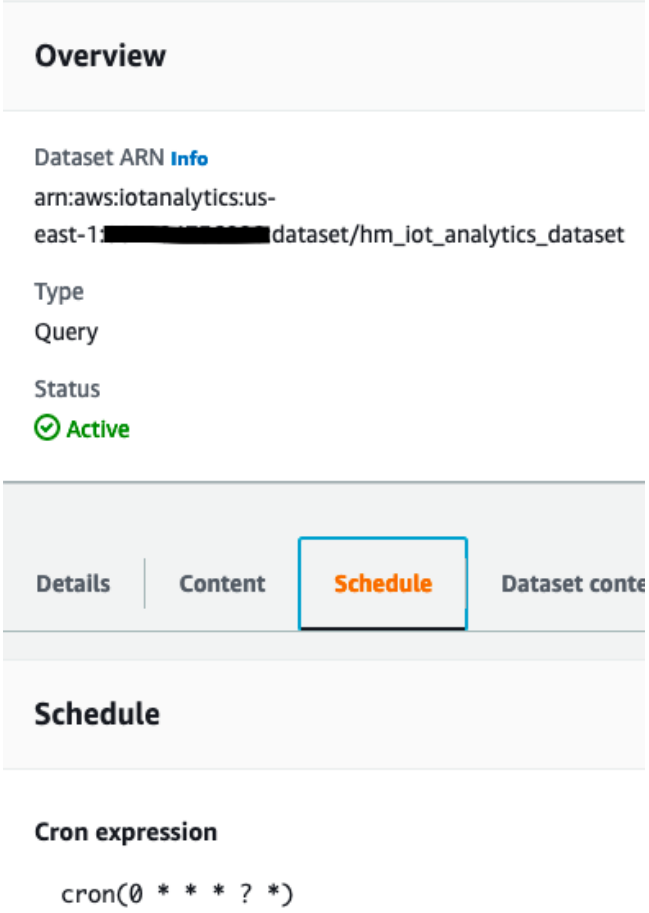


Figure 7: Cron schedule details for 'hm_iot_analytics_dataset'

7.1 Amazon QuickSight Setup

Even though Amazon QuickSight is a BI service provided by AWS, a user must still sign up for it using an email ID. This email ID serves as the equivalent of a login password used to connect to any other BI tool. The primary author's email address was used for this implementation. A user must also grant Quicksight access to the account's Amazon S3 buckets and AWS IoT Analytics when signing up. QuickSight will not be able to access either resource unless this critical step is completed. By checking the appropriate checkboxes, the required IAM Roles and permissions are updated in the background, allowing appropriate access. However, if a user overlooks this step, they can correct the error by editing the QuickSight Security Settings. Refer to Figure 12 in the Results section for a chart depicting the Average body temperature by serial number of the IoT device, combined with the blood pressure values obtained from the data set. The average heart rate recorded from simulated values is visualized in Figure 11 and average body temperature recorded from the simulated data values kept in the data set are visualised in Figure 13. Any business or data science analyst can focus solely on visualisations and data inference by using a variety of drag-and-drop graphics that are incredibly convenient, quick, and efficient.

7.2 Dashboard for Analytics Pipeline Activity

As an additional activity, the authors created a monitoring dashboard in CloudWatch, that monitors the transformation activity described in Section 3.6. Figure 15 shows the pipeline activity execution errors on the CloudWatch dashboard.

8 Results

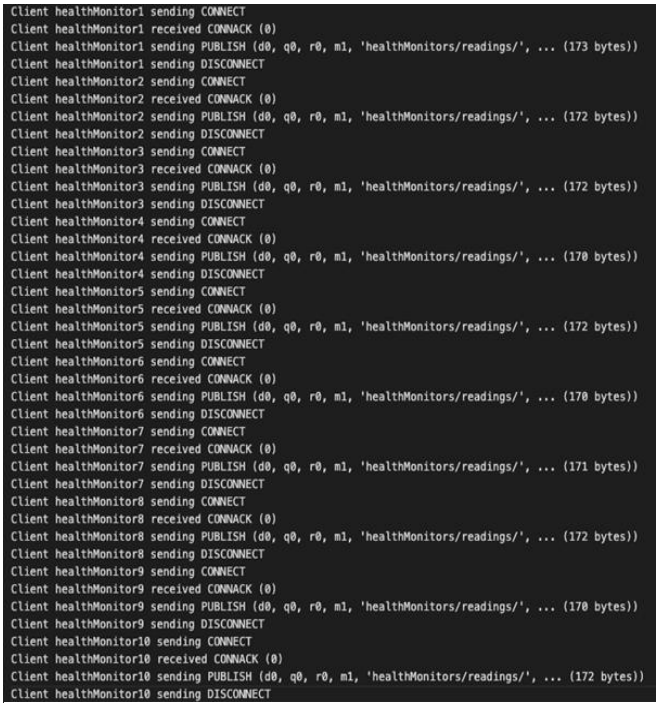


Figure 8: Connection status when subscribed to MQTT Topic 'healthMonitors/readings/' – messages published to the MQTT topic

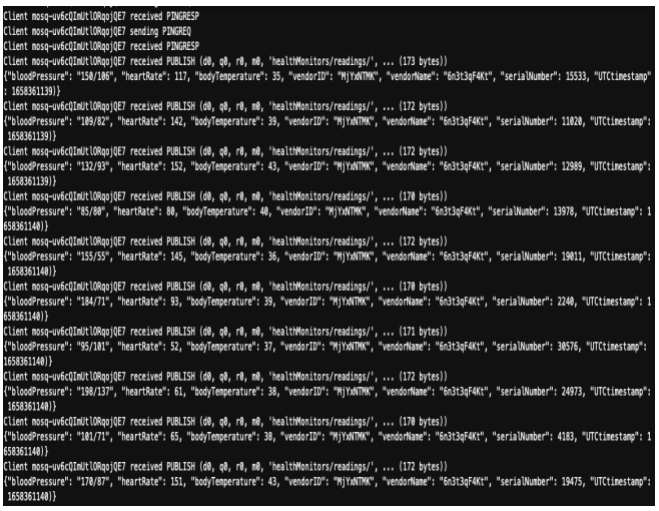


Figure 9: Random Data being published to the MQTT Topic for a sample interval

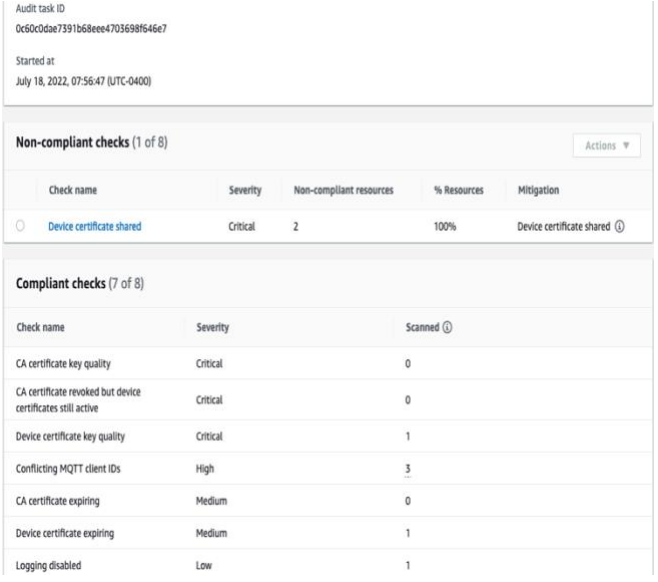


Figure 10: Sample of a 'Failed' daily Device Defender Audit check. Device key quality is a 'Critical' security flaw.

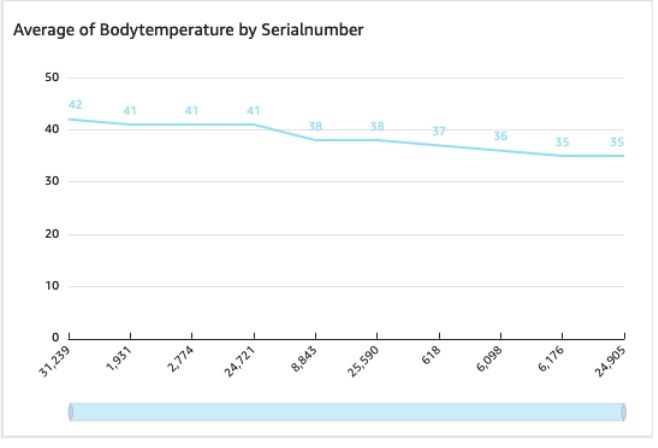


Figure 11: Sample QuickSight Line Chart visualization of average body temperature reported per device (denoted by randomized serial numbers

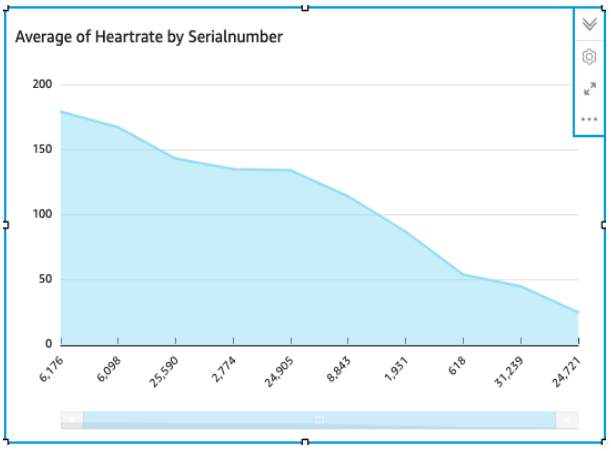


Figure 12: Area chart of the average heart rate from the simulated data per device (serial number unique to device)

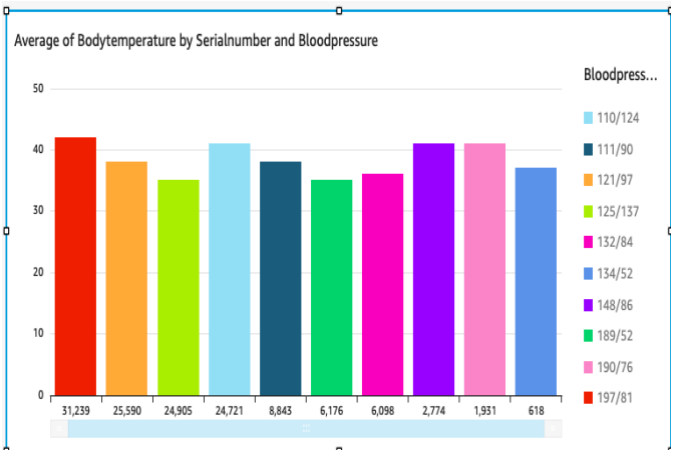


Figure 13: Combined bar chart of average body temperature reported per serial number, grouped by blood pressure values

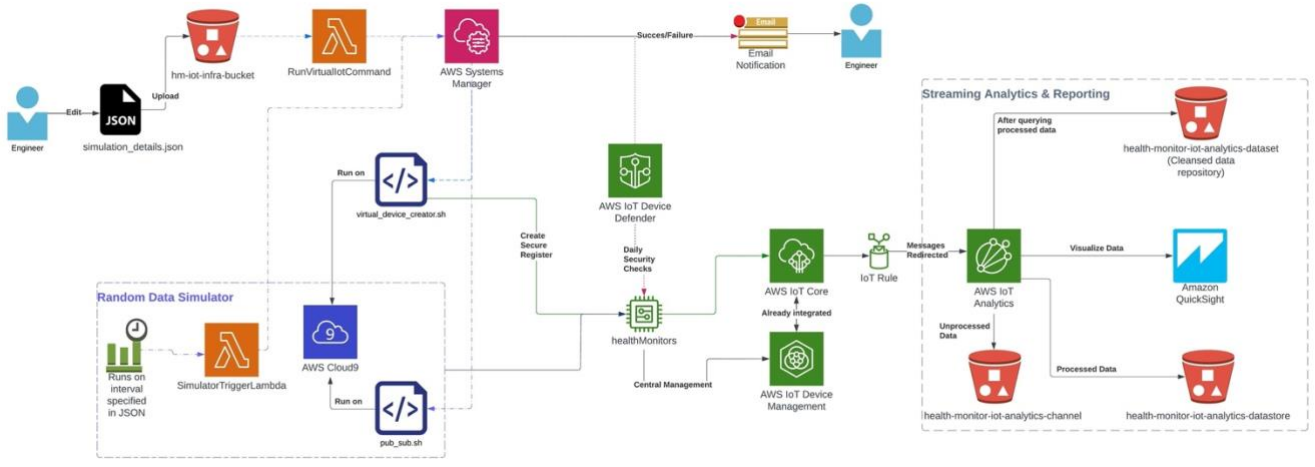


Figure 14: AWS Architecture Diagram explaining all AWS services used, flow of information, triggers, and sections of the unified IoT platform hosted natively on AWS

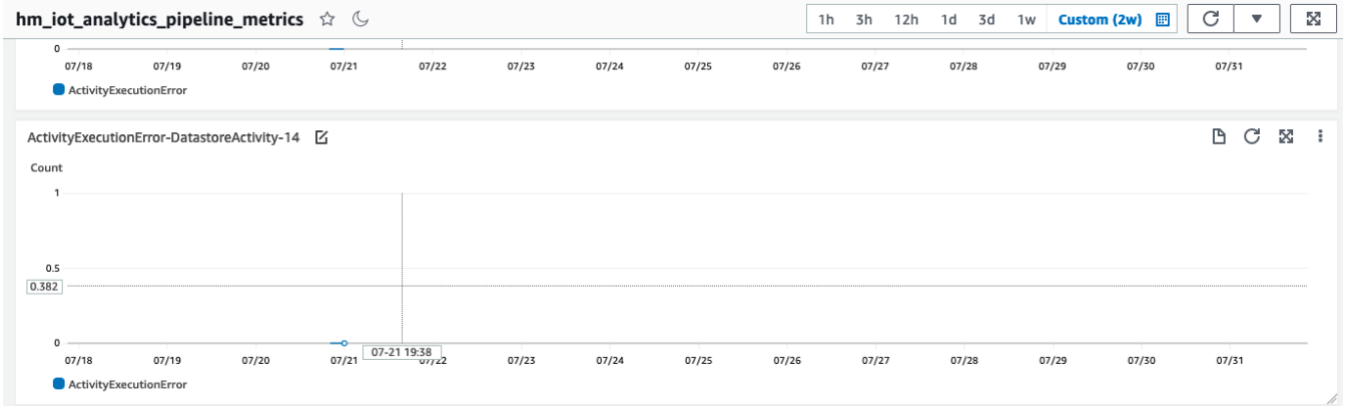


Figure 15: AWS IoT Analytics Pipeline activity execution metrics visualized in custom CloudWatch Dashboard.

Figure 16: Sample notification of the daily audit check

9 Conclusions

The creation, registration, and maintenance of centralized, automated IoT devices have been accomplished by the authors. Additionally, the outcomes demonstrate that random data simulations were successfully executed without the need for manual assistance. The entire solution, from the prototyping to the deployment phases, is cloud-native. It offers real-time dashboard and reporting features that are coupled with streaming analytics capabilities. The writers carefully analyzed security during development, as well as ensuring effective recording and monitoring at all levels, at practically every stage of the process. Along with this, the writers added email notifications for daily audit checks and programmatic runs.

10 Future Scope

The platform can be extended in the future to deploy ML/AI models on the cloud. Because the current application does not support editing device attributes, a possible extension could be to modify the existing Bash scripts to work with any number and type of device attributes. Creating a UI/console-based simulation portal would allow for even more simplification in editing device attributes and simulation details. Furthermore, using infrastructure as code tools like Terraform, the entire code base could be extended to be deployed automatically even on the cloud. The project lays a solid foundation for different avenues to explore and enhance as the use case may be.

11 Acknowledgment

The chance to investigate the intersection of the IoT and cloud worlds was made possible, in part, by Dr. Otman Basir, Professor at the University of Waterloo. We commend Amazon Web Services for its well-documented resources, interactive workshops, tools, and integrations that helped this project be successfully realised. In conclusion, the authors would like to acknowledge the University of Waterloo for its ongoing support and encouragement to its students.

References

[1] Botta, A., De Donato, IoT., Persico, V., Pescape, A.: On the Integration Of Cloud Computing And Internet Of Things. In: The 2nd International Conference on Future Internet Of Things And Cloud (Ficloud-2014) (2014)

[2] Nastic, S., Sehic, S., Le, D., Truong, IoT., Dustdar, S.: Provisioning Software-Defined Iot Cloud Systems The 2nd International Conference on Future Internet Of Things And Cloud (Ficloud-2014) (August 2014)

[3] Varga, A., Hornig, R.: An Overview of The Omnet++ Simulation Environment. In: Proceedings of the 1st International Conference On Simulation Tools And Techniques For Communications Networks And Systems & Workshops (Simutools '08) (2008)

[4] Chernyshev, M., Baig, Z., Bello, O., Zeadally, S.: Internet of Things (Iot): Research, Simulators, And Testbeds. IEEE Internet Of Things Journal. https://doi.org/10.1109/jiot. 2017.2786639 (2017)

[5] Han, S.N., Lee, G.M., Crespi, N., Luong, N.V., Heo, K., Brut, M., Gatellier, P.: Dpwsim: A Simulation Toolkit for Iot Applications Using Devices Profile for Web Services. In: Proc. Of Ieee World Forum on Internet of Things (Wfiot), Vol. 6-8, Pp. 544–547 (2014)

[6] Sotiriadis, S., Bessis, N., Asimakopoulou, E., Mustafee, N.: Tow Simulating The Internet Of Things. In: 2014 28th International Conference on Advanced Information Networking and Applications Workshops (Waina). Ieee, Pp. 444–448 (2014)

[7] Sotiriadis, S., Bessis, N., Antonopoulos, N., Anjum, A.: Simic: Designing A New Inter-Cloud Simulation Platform for Integrating Large-Scale Resource Management. In: 2013 IEEE 27th International Conference on Advanced Information Networking And Applications (Aina). IEEE, Pp. 90–97 (2013)

[8] Zeng, X., Garg, S.K., Strazdins, P., Jayaraman, P.P., Georgakopoulos, D., Ranjan, R.: Iotsim: A Simulatorfor Analysing Iot Applications. J. Syst. Archit. 72, 93–107 (2017)

[9] Alamri A, Ansari Ws, Hassan Mm, Hossain Ms, Alelaiwi A, Hossain Ma. A Survey on Sensor-Cloud: Architecture, Applications, And Approaches. International Journal of Distributed Sensor Networks. February 2013. Doi:10.1155/2013/917923

[10] Kertész, Attila & Pflanzner, Tamas & Gyimothy, Tibor. (2019). A Mobile Iot Device Simulator for Iot-Fog-Cloud Systems. Journal Of Grid Computing. 17. 10.1007/S10723-018-9468-9.

[11] Vincent C. Emeakaroha, Neil Cafferkey, Philip Healy, John P. Morrison, A Cloud-Based Iot Data Gathering and Processing Platform. 2015 3rd International Conference on Future Internet of Things And Cloud (Ficloud) Doi:10.1109/Ficloud.2015.53

[12] M. Moreno, J. Santa, M. Zamora, And A. Skarmeta, "A Holistic Iot-Basedmanagement Platform for Smart Environments," In Communications (Icc), 2014 Ieee International Conference On, June 2014, Pp. 3823–3828.

[13] D. Mijić and E. Varga, "Unified Iot Platform Architecture Platforms as Major Iot Building Blocks," 2018 International Conference on Computing And Network Communications (Coconet), 2018, Pp. 6-13, Doi: 10.1109/Coconet.2018.8476881

[14] Sundmaeker, IoT., Guillemin, P., Friess, P., Woelffle, S.:Vision And Challenges For Realising The Internet Of Things Cerp Iot – Cluster Of European Research Projects On The Internet Of Things Cn: Kk-31-10-323-En-C (2010)

[15] https://docs.aws.amazon.com/amazons3/latest/userguide/s3-userguide.pdf, Amazon Simple Storage Service, User Guide

[16] https://docs.aws.amazon.com/iotanalytics/latest/userguide/welcome.html, AWS IoT Analytics, User Guide

[17] https://docs.aws.amazon.com/iam/latest/userguide/id_root-user.html, AWS Account Root User

[18] https://docs.aws.amazon.com/systems-manager/latest/userguide/execute-remote-commands.html, AWS Systems Manager Run Command

[19] https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html, How AWS IoT Works

[20] usha rani, Shola & Ignatious, Antony & Hari, Bhava & Balavishnu, V. (2017). Iot Patient Health Monitoring System. Indian Journal of Public Health Research & Development. 8. 1329. 10.5958/0976-5506.2017.00519.8.

[21] https://docs.aws.amazon.com/iot/latest/developerguide/device-defender.html, AWS IoT Device Defender, Developer Guide

[22] https://docs.aws.amazon.com/quicksight/latest/user/welcome.html, Amazon QuickSight, Welcome Guide

[23] https://awscli.amazonaws.com/v2/documentation/api/latest/reference/cloudformation/create-stack.html, create-stack AWS CLI v.2

[24] https://mosquitto.org/man/mosquitto_pub-1.html, Mosquitto_pub manual page

[1]Parnika Kaushik, B.Tech (Electrical & Electronics Engineering) 2016, Manipal Institute Of Technology. Parnika is currently a M.Eng (Electrical And Computer Engineering) student at the University Of Waterloo, Canada. she is a multiple award-winning cloud engineer and was employed with Deloitte Consulting India Pvt. Ltd from March 2017 – July 2021 as Python programmer and cloud engineer, with specialty in AWS platform and infrastructure automation, cloud security and DevOps engineering. Her current interests include data engineering and big data processing.

[2]**Rubani Bhatia** is currently a M.Eng (Electrical And Computer Engineering) student at the University Of Waterloo, Canada. Rubani is former Risk Analyst at KPMG, with an avid interest in introducing the concepts of security throughout development cycles. Rubani is looking to expand her interests in the field of IoT and cloud.

[3]**Vivek Chauhan** is currently a M.Eng (Electrical And Computer Engineering) student at the University Of Waterloo, Canada. He currently interns at UbiLab, Waterloo, in the data engineering domain, focused on IoT based zero-effort technology. He is a skilled Data Engineer with over 2.5 years of rich experience working for ZS Associates and Airtel in India. His current interests include big data processing, dimensional modelling and creating DWHBI solutions.