Pollution Tracking and Monitoring System for Workshops
Final Report

Vasudevan Perumal -A20446003

Acknowledgment:

I acknowledge all works including figures, codes, and writings belong to me and/or persons who are referenced. I understand if any similarity in the code, comments, customized program behavior, report writings, and/or figures are found, both the helper (original work) and the requestor (duplicated/modified work) will be called for academic disciplinary action.

Sign: [illegible] 5/7/2020

Abstract:

Workshops, Machine Shops, and Model shops across our campus or in general might be plagued with various pollutants that might be hazardous to people with allergies or certain health conditions. Numerous pollutants may be over an undesirable high level during a particular task that takes place in a workshop. Carbon Monoxide levels may be high while using an acetylene arc which can cause a deleterious effect to people with breathing problems. While machining metal, the decibel of sound produced may be too high that people with sensitive hearing would like to avoid it. People with this type of health might want to be aware of these conditions while working at a workshop. We are proposing a project that helps monitor these conditions and provide a graphical graph for constant tracking and attempt to build a voice-based interface to interact with our system to learn about the level of certain pollutants in the workshop.

1. Introduction:

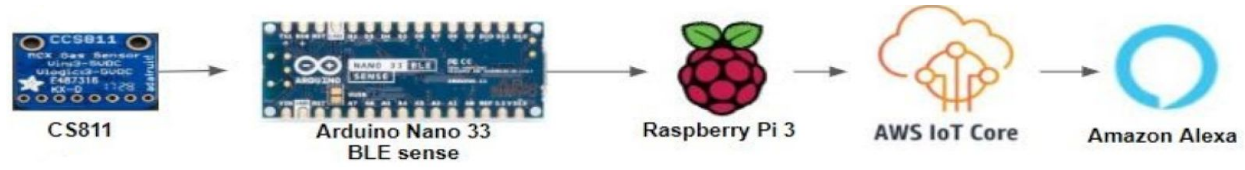
In general, all workshops, garages, makerspace, machine shops have an open floor layout. This means various different tools are used and different processes are done in the same environment. All the people from all parts of the workshop experience the same pollution even if they are not involved in that certain work. Thus people sensitive to certain types of pollution might want to avoid working in the workshop at those times. People with asthma or breathing problems might want to avoid the workshop during the use of acetylene arcs. The volatile organic compounds generated by aerosols may cause nausea and people sensitive to these conditions would want to refrain from using the workshop. People with sensitive hearing would want to avoid working during times when metals are machined, drilled, and smoothened as these processes cause high noise to be produced. With the emergence of cheap sensors and the concept of IoT systems being more widely adopted, it is now easy to create a system that can be used to monitor various parameters in such indoor locations. Monitoring these parameters creates an informative system that the user can interact with to be more aware of their surroundings.

Thus we are proposing a project that can monitor and track various pollutants in the workshop environment that the user can interact with to learn what pollutant is high and would they want to avoid coming to the workshop or not. Through this project, we try to understand how enterprise cloud-based IoT platforms can help in setting up the wireless sensor network and also try to implement the feature these types of platforms can provide like easy database logging, easy dataset creation for visual interpretation, easy pipelining of data to other services like data visualization software and user-interface services. This system will monitor the temperature, humidity, air quality, and noise level in the workshop environment through various sensing nodes. These sensing nodes will be connected to an IoT gateway. The IoT gateway collects these data points and does some data preprocessing after which the data points are sent to the cloud service. This cloud service allows us to process the data for visualization, storage, or analysis. The user can now visualize the data in the form of graphs or learn about the current sensor values through a voice-based interface.

2. Description:

Our project is a Level-6 IoT system that monitors certain pollutants and provides a voice-based interface to interact with our system to learn about the level of these certain pollutants in the workshop. We use the CCS811 air quality sensor to detect the Total Volatile Organic Compounds(TVOCs) levels and also detect the equivalent carbon dioxide(eCO2) levels. This sensor is interfaced using I2C. We also monitor the humidity and temperature to understand the working conditions. A microphone is used to measure the decibels for the system to learn if the sound level is above the advisable level. The sensors are connected to an Arduino type board called the Arduino Nano 33 BLE sense. This board will connect with the sensors and retrieve the data through analog signals and I2C protocols. This subsystem is connected via serial communication or BLE to a Raspberry Pi which will act as an IoT gateway to our cloud service.

For our project, we will use the AWS IoT core for orchestration management, data storage, and other features. We choose AWS IoT core as our cloud management service due to the security it provides and also the easy scalability it can provide for further expansion of the system if it is so necessary. The use of AWS IoT Core also allows us to create a graphical visualization of the data that we aggregate from the server. This visual data can also be used to analyze trends in pollution levels during certain durations or operations in the workshop. The cloud interface also enables users to create an Alexa based voice user interface such that the user can learn about the various pollution levels in the workspace. The main aim of our project is to show how modern cloud solutions like AWS IoT can be used to create an easily scalable and secure pollution monitoring system. A general system architecture can be seen below.



3. Hardware Used:

3.1 Arduino NANO 33 BLE Sense:

The sensing node and controller board used in this project is the Arduino NANO 33 BLE Sense. This all-in-one board has various sensors already integrated on the board like an accelerometer, temperature, humidity, a digital microphone, gesture, and pressure sensor. This board is based on the Nordic nRF52480 SoC which has an M4F microcontroller. This board also provides BLE connectivity using the NINA B306 module.

The USB port on the board has a direct connection to the native USB of the NINA B306 module. This allows us to use the NANO BLE sense as a USB peripheral device like a mouse, keyboard, or joystick. The NANO also has an analog to digital converter(ADC) that can change its resolution from 10-bit default to 12-bit ADC. The PWM feature on the board has an 8-bit resolution. For this project, the on-board sensors we are interfacing with are the HTS221 that is the relative humidity and temperature sensor and the MP34DT05 digital microphone to sense the noise levels(*not implemented as of this report)*. The HTS221 uses polymer dielectric planar capacitors that detect the relative humidity and temperature that are available through a serial interface. This sensor has a range of -40ºC to +120ºC with an accuracy of ±0.5ºC for temperature and range of 0 to 100% and accuracy of ±3.5% rH. The MP34DT05 is a digital MEMS microphone that provides an output that is coded in PDM(Pulse Density Modulation) format.

A more thorough description of the Arduino NANO BLE Sense 33 board can be found at NANO33BLESense.

3.2 Raspberry Pi 3 B+:

The Raspberry Pi 3 B+ acts as the IoT gateway in our system. This system is a Level-6 IoT system. The Raspberry Pi communicates with Arduino NANO and obtains the stream of sensor data points. It does certain data conversions and preprocessing such that it is human-readable before it sends it to the Cloud Service. Raspberry Pi is a single-board micro-computer. The model 3 B+ with Cortex-A53 SoC that is based on the ARMv8 64-bit architecture that runs at 1.4 GHz. It has 1GB LPDDR2 RAM. The Pi has also 2.4GHz and 5GHz IEEE 802.11 b/g/n/ac wireless LAN and Bluetooth 4.2 BLE these wireless technologies can be used to communicate with sensors, interconnect with other devices, or to a router for internet access. Raspberry Pi also has Gigabit Ethernet with PoE(Power over Ethernet) capability which enables easy deployment of the gateways to existing networks on the location. One of the features that entice the use of Raspberry Pi for IoT applications is the 40 GPIO headers. This enables easy interfacing with sensors through I2C, SPI, or other interfacing methods at a voltage level of 3.3V. 5V sensors have to be level shifted to support interfacing with the Pi. The schematics and product brief can be found at Raspberry Pi 3 Model B+**.**

We use the BLE module that is inbuilt on the Arduino NANO 33 and Raspberry for communication between the sensing node and the gateway. BLE is a low energy consuming version of Bluetooth based on the Bluetooth 4.0 specification. Unlike traditional Bluetooth, BLE presents its information like a community bulletin board. The information is structured in a hierarchy as services that are further subdivided into characteristics. In this system, the RaspberryPi is the Central Device which acts as clients that read and write data from the Peripheral devices that is the Arduino NANO that acts as the server. The peripheral will represent all the sensor parameters it can provide as a service. Each service in the peripheral has 16 bit UUID(Unique User ID). Each sensor parameter is represented as a characteristic that also has a 16 bit UUID each. Thus the temperature, humidity, noise level, and air quality are presented to the client as separate characteristics each with a UUID. The client or central device i.e the Raspberry Pi can operate in three modes. The read mode is where the client obtains data that is not changed frequently like version ID and configuration. In the write mode, the client can modify the value of the characteristic in the peripheral. Using the Notify mode the client can read a continuous stream of values from the peripheral.

3.3 CCS811 Air Quality Sensor:

CCS811 is the only externally connected sensor in this project; all other sensors are available on-board the NANO 33. This sensor(*not yet implemented as of the report)* is used to obtain air quality in the workshop. This is measured by detecting the eCO_2 (equivalent calculated carbon-dioxide))and TVOCs(Total Volatile Organic Compounds) levels. The Arduino NANO interfaces with this Sensor using the I2C protocol. I2C is a two-wire protocol with a master and slave hierarchy. Here the Arduino NANO acts as the master and the CCS811 sensor is the slave. This sensor from AMS is intended for indoor air quality monitoring, this is why we thought this sensor would be suitable for our use-case. This sensor breakout board consists of a hot-plate metal oxide (MOX) sensor, It also contains an in-built microcontroller that controls the power supplied to the plate and converts the analog values from the sensor to an I2C reading to interface with.

The sensor can measure eCO_2 concentration from a range of 400 to 8192 ppm(parts per million). It can detect TVOCs from a range of 0 to 1187 ppb (parts per billion). TVOCs that are included in the detection are Alcohols, Aldehydes, Ketones, and Organic Acids. This sensor can operate in both the 3.3V logic level and 5V logic level making it versatile as it can interface a variety of microcontrollers. The datasheet of the sensor containing further details can be found at CCS811.

4. **Software Platforms:**
    - **Arduino IDE and Arduino libraries**
    - **C++**
    - **Python**
    - **Linux**
    - **AWS IoT Core**
    - **AWS IoT SDK**
    - **AWS IoT Analytics**
    - **AWS DynamoDB**
    - **AWS QuickSight**
    - **AWS Lambda**
    - **Alexa**

The system starts with the Arduino NANO microcontroller. The Arduino NANO is programmed using the Arduino IDE and uses a USB COM serial interface to program the controller. The programming language used here is an altered flavor of C++ that is tailored for Arduino development. The libraries used in the Arduino ecosystem are majorly written in C++ and Arduino IDE is built to support C++ out of the box. Thus, we chose C++ to program a microcontroller. It is also possible to use a version of Python called MicroPython to program out micro-controller but this requires a workaround and extra steps to be taken for the bootloader to load the program.

The Raspberry Pi runs on a Debian based Linux OS called Raspbian. We use the Python programming language for architecting our gateway functionalities for data aggregation and processing. AWS IoT SDK is used to create the pipeline to process and transfer data to the AWS Cloud services. Python programming language was used to construct our gateway functions and interface with Cloud services because of it's easy to prototype nature and quicker development and deployment time. Both Raspberry Pi and AWS IoT cloud service have SDKs in NodeJS, Java, and C++ but due to the benefits mentioned before and familiarity with the Python platform, Python was the programming language of choice.

AWS IoT Core is a cloud service that provides secure, bi-directional communication between devices like sensors, micro-controllers, and actuators through Internet hosted services. AWS IoT core allows us to collect the telemetry data from multiple nodes and use them for analysis and storage. This service also allows us to create a data pipeline that can be used to make applications that the end-user can interface with.

AWS IoT Core consists of various components. It has a Device Registry that keeps track of all the nodes that are connected to the cloud. The Message Broker is used to agglomerate the MQTT telemetry data in a publish and subscribe fashion and pipeline the data to other cloud services or other nodes. The end-device and IoT message broker publish to and subscribe to respectively to a topic. This topic allows for the logical segmentation of payload streams sent from each end-device. MQTT or Message Query Transport Telemetry protocol is a lightweight protocol used for IoT applications as the data payload is small and does not require large data buffers and IoT applications desire a protocol that can provide low latency which MQTT protocol does. All nodes connected to AWS IoT core are authenticated using X.509 certificates that are based on the X.509 public key infrastructure. Trusted entities called Certificate Authorities(CA) issue these certificates. This enables the authentication and confidentiality of the data and mitigates against Man-in-the-Middle(MitM) attacks.

AWS IoT Core has a feature called Rules Engine which can filter specific telemetry parameters through SQL like syntax and send this data to other cloud services. In this project, we send our telemetry data to be stored in AWS DynamoDB which is a NoSQL database. The data is also sent to AWS IoT Analytics and AWS QuickSight which are two cloud services that pipeline the telemetry data to create a dataset. This dataset can be transformed and applied to machine learning models in AWS IoT Analytics or we could create graphs and visualize the data using AWS QuickSight.

Our project also implements the Voice-based user interface that is powered by Amazon's Alexa. We use Alexa's speech-to-text engine to convert the invocation that we speak to programmatic input for or backend logic and response. The response or program that interfaces with the database to provide the sensor data as a VUI is hosted on a server-less runtime service called AWS Lambda. AWS Lambda enables us to run code without needing the infrastructure or virtualization of a dedicated server. We have to upload our program and all the required libraries and dependencies to AWS Lambda service. Whenever service is triggered by our Alexa invocation, a container is created with the required dependencies and is executed after which the container is disbanded and deleted. In Alexa's development environment we create our own Skill that is similar to an App on other services. This Skill enables us to create custom invocation queries and trigger custom Lambda endpoints.

5. Related Work:

We found a similar paper [1] that investigates an IoT-based Indoor Air quality monitoring system The system architecture proposed in this paper differs from the architecture that we have used in our system. In [1] the sensing node and IoT-gateway are housed in a single unit, unlike our system where the sensing node and gateway are two separate entities and they intercommunicate using BLE. Our system uses WiFi to connect to the cloud service but this system uses LTE to connect to cloud services. The microcontroller used in their system is the STM 32 F407IG which is based on the ARM 32-bit Cortex-M4 similar to our system. The microcontroller is connected to an LTE module for wireless communication. The sensors used in this system are a laser dust sensor that senses particulate matter PM2.5 and PM10, VOC, Carbon monoxide and dioxide sensor, temperature, and humidity sensor. The system we have proposed also senses all these parameters except particulate matter through indirect methods.

The major way the system proposed in [1] and our system differs is in system architecture and backend processing. In [1] the sensing node i.e is the perception layer and IoT gateway i.e is the network layer is housed in a single unit. In our system, the sensing node consisting of the sensors and IoT gateway is decoupled. We believe our system is more cost-efficient and easy for scalability. The IoT gateway can support multiple sensing nodes, thus if we need to increase the sensor density in an environment only the number of sensing nodes has to be increased. In [1] the entire unit has to be multiplied to increase the sensor density. This may not be the most cost-efficient method for scaling the network. Also in [1], each node has to use the same wireless medium for communication with the cloud service, this will cause increased network congestions and collisions. In our system, a proper queuing scheme can be implemented to mitigate these issues.

The system in [1] uses the AWS cloud as an IaaS(Infrastructure as a Service), where they implement their own server and MySQL database. Our system uses AWS cloud both as PaaS(Platform as a Service) and SaaS(Software as a Service). In the methodology [1] uses scalability of resources has to be configured by the operator but in our implementation, the dynamic scalability of resources is handled by the intelligent engines that the service provides.

6. Results:

We have successfully been able to implement our system architecture that consists of a decoupled sensing node connected to the IoT gateway it's communication to the backend Cloud services that provide us with data storage, telemetry, visualization, and other features. In Fig 2.0 we can see a picture of our system, the Arduino NANO 33BLE Sense can be seen on the red breadboard and the IoT gateway Raspberry Pi 3 B+ can also be seen. Both these devices are powered by micro-USB. In this system, we intend wired powering as this system is located in an indoor environment, the sensing node, and the IoT gateway can be powered using existing power outlets.

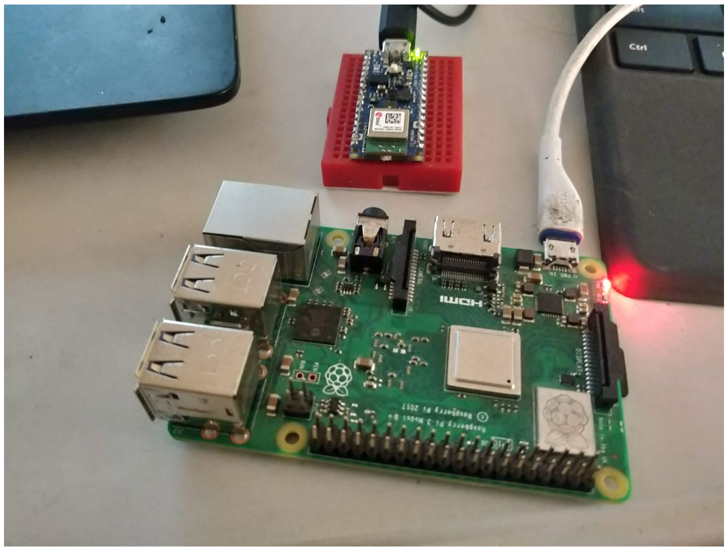
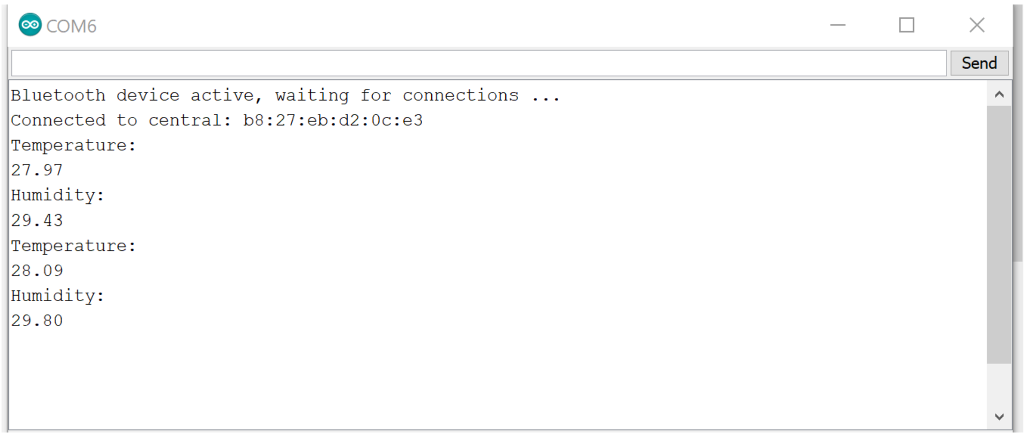


Fig 2.0 System picture with sensing node and IoT gateway

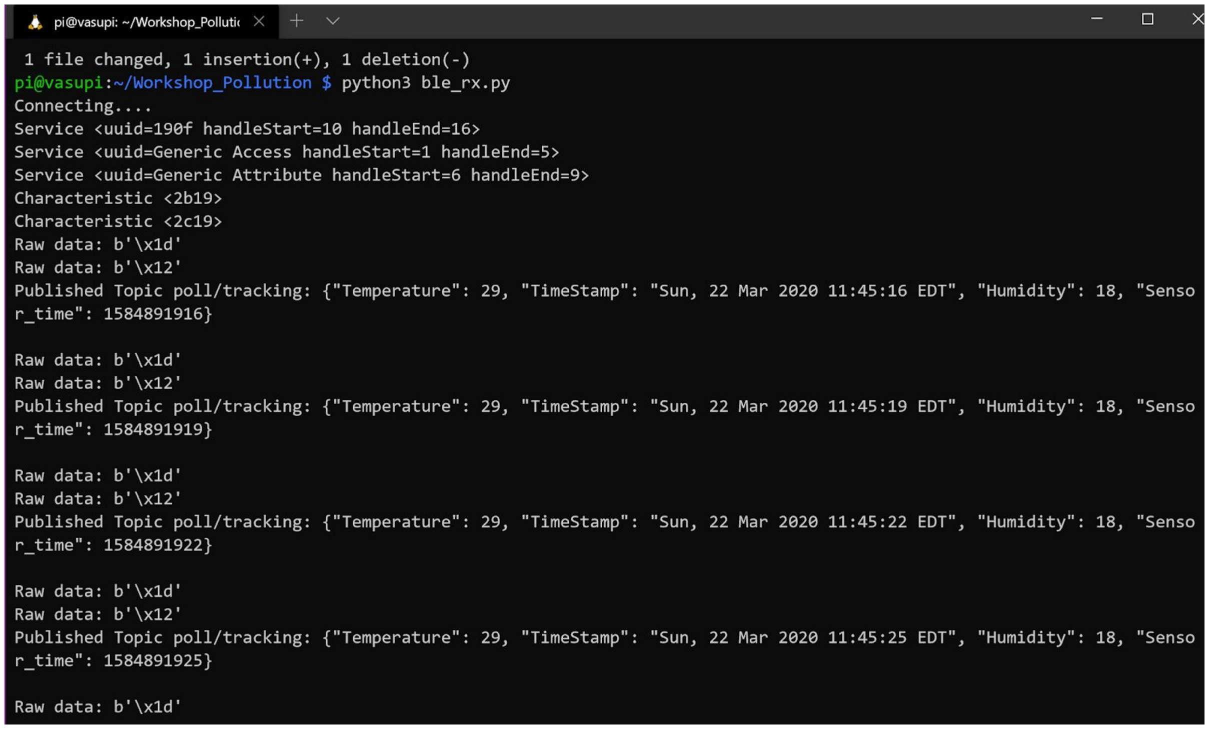
We have interfaced with the HT221 Temperature and Humidity Sensor that is built-in to the Arduino NANO 33 BLE Sense board. A communication channel has been established between the Arduino and the Raspberry Pi through BLE. The temperature and humidity are represented as two different characteristics with respective UUIDs under a BLE service. In Fig 3.0 we can observe the data values from the sensors on the Serial Monitor of the Arduino NANO, we can also see the MAC address of the central master i.e the IoT gateway which it is connected to. This also for easy correlation when a large number of sensing nodes are deployed.



```
Bluetooth device active, waiting for connections ...
Connected to central: b8:27:eb:d2:0c:e3
Temperature:
27.97
Humidity:
29.43
Temperature:
28.09
Humidity:
29.80
```

Fig 3.0: BLE connection established, temperature and humidity data interfaces, and output to the serial monitor.

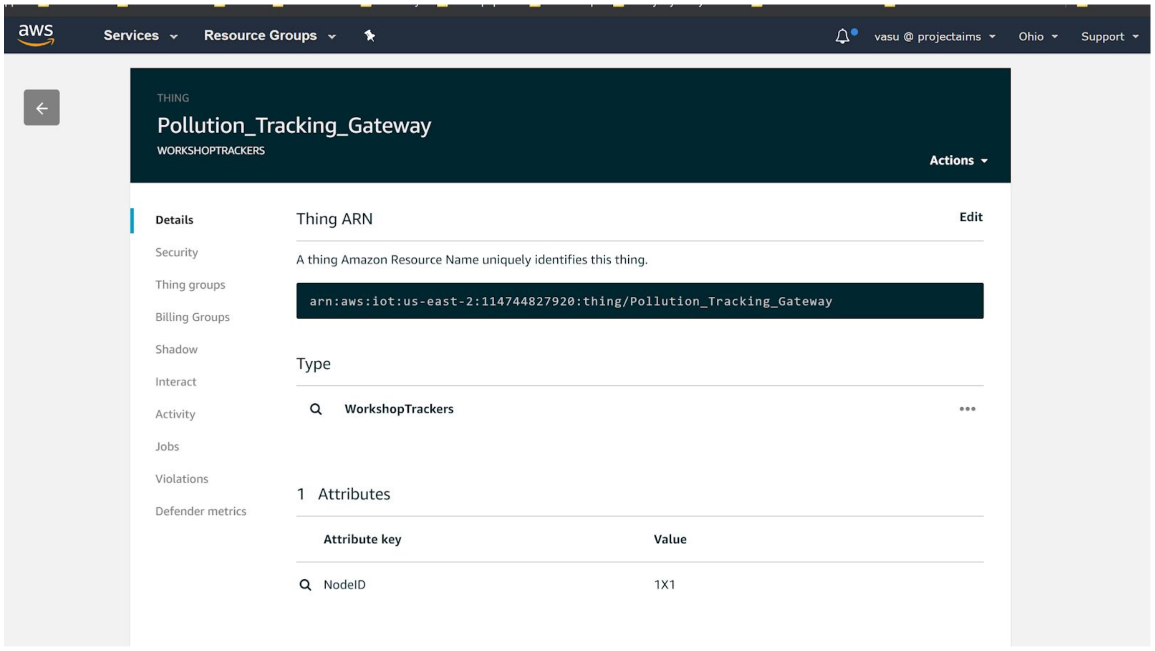
BLE transfers data in hexadecimal format. The Raspberry Pi collects these data and does appropriate conversions and processing such that it is human-readable. This can be seen in Fig 4.0. The Raspberry Pi also adds a timestamp depicting the data and time the gateway received the data and a sensor time depicting the time the sensing node obtained the data this is in seconds format calculated from the Epoch. Fig 4.0 also shows us the UUID of the characteristics sent over through BLE and the topic the payload is published to in the AWS IoT Message Broker which is poll/tracking.



```
 1 file changed, 1 insertion(+), 1 deletion(-)
pi@vasupi:~/Workshop_Pollution $ python3 ble_rx.py
Connecting....
Service <uuid=190f handleStart=10 handleEnd=16>
Service <uuid=Generic Access handleStart=1 handleEnd=5>
Service <uuid=Generic Attribute handleStart=6 handleEnd=9>
Characteristic <2b19>
Characteristic <2c19>
Raw data: b'\x1d'
Raw data: b'\x12'
Published Topic poll/tracking: {"Temperature": 29, "TimeStamp": "Sun, 22 Mar 2020 11:45:16 EDT", "Humidity": 18, "Senso
r_time": 1584891916}

Raw data: b'\x1d'
Raw data: b'\x12'
Published Topic poll/tracking: {"Temperature": 29, "TimeStamp": "Sun, 22 Mar 2020 11:45:19 EDT", "Humidity": 18, "Senso
r_time": 1584891919}

Raw data: b'\x1d'
Raw data: b'\x12'
Published Topic poll/tracking: {"Temperature": 29, "TimeStamp": "Sun, 22 Mar 2020 11:45:22 EDT", "Humidity": 18, "Senso
r_time": 1584891922}

Raw data: b'\x1d'
Raw data: b'\x12'
Published Topic poll/tracking: {"Temperature": 29, "TimeStamp": "Sun, 22 Mar 2020 11:45:25 EDT", "Humidity": 18, "Senso
r_time": 1584891925}

Raw data: b'\x1d'
```

Fig 4.0: Raspberry Pi collects data through BLE connection and publish the data to AWS IoT core with a timestamp

In the AWS IoT Core console, we create a Thing which represents our IoT gateway and sensing node pair. This Thing is authenticated and authorized by using an X.509 certificate. The Thing that is our Raspberry Pi can now publish the data to the Message Broker that is subscribed to the topic poll/tracking. The Rules Engine has also been established to pipeline the data to the DynamoDB database and AWS IoT Analytics. In Fig 5.0 we can see the dashboard of the Thing, the Thing can be uniquely identified by the Amazon provided ARN(Amazon Resource Number) or the Administrator configured NodeID which is 1X1 for this Thing. Other details like the certificates, keys, shadow state, activity time can also be monitored here.



Thing ARN

A thing Amazon Resource Name uniquely identifies this thing.

`arn:aws:iot:us-east-2:114744827920:thing/Pollution_Tracking_Gateway`

Type

WorkshopTrackers

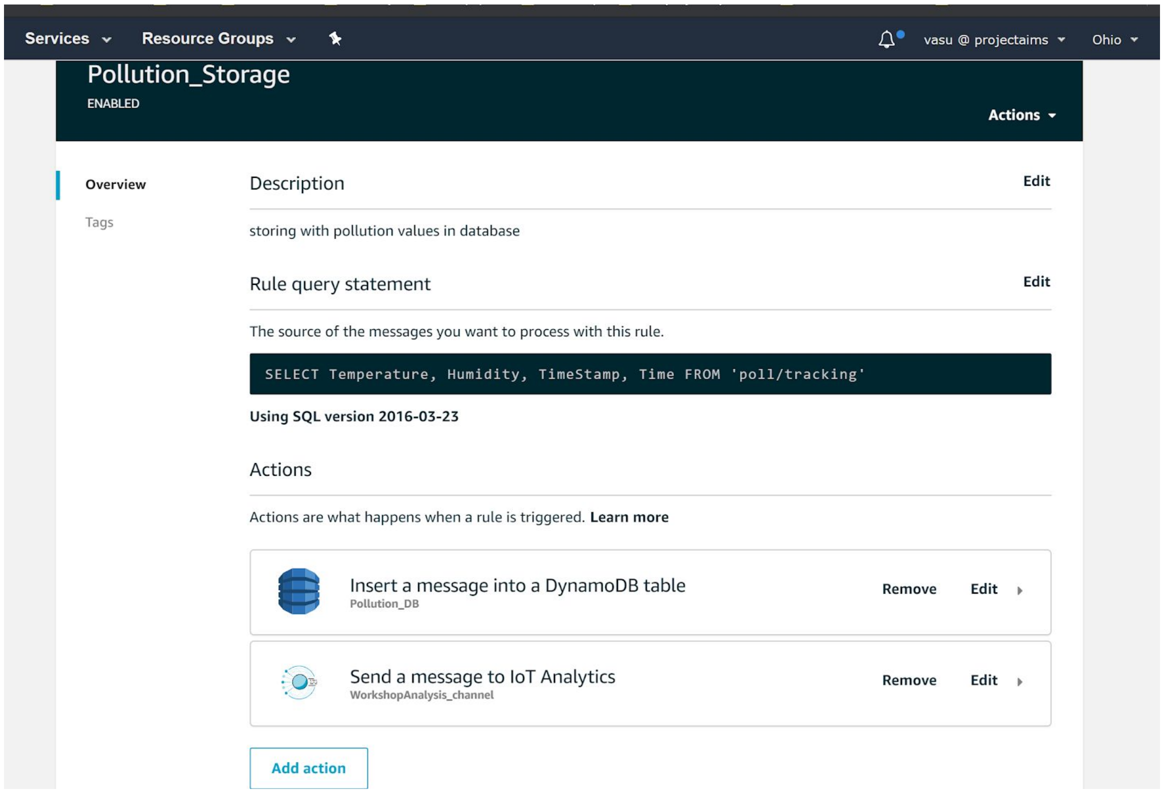
1 Attributes

Attribute key	Value
NodeID	1X1

Fig 5.0: Thing Created with unique NodeID

Fig 6.0 depicts the establishment of the Rules Engine which triggers other AWS services through SQL queries like syntax. The Rule Query statement shows that filters the payload, to obtain the appropriate data. The data is now pipelined to be stored in the DynamoDB Database and pipelined to

AWS QuickSight through AWS IoT Analytics to visualize the data. This implementation can be seen in the Actions field.



Pollution_Storage
ENABLED

Description

storing with pollution values in database

Rule query statement

The source of the messages you want to process with this rule.

```
SELECT Temperature, Humidity, TimeStamp, Time FROM 'poll/tracking'
```

Using SQL version 2016-03-23

Actions

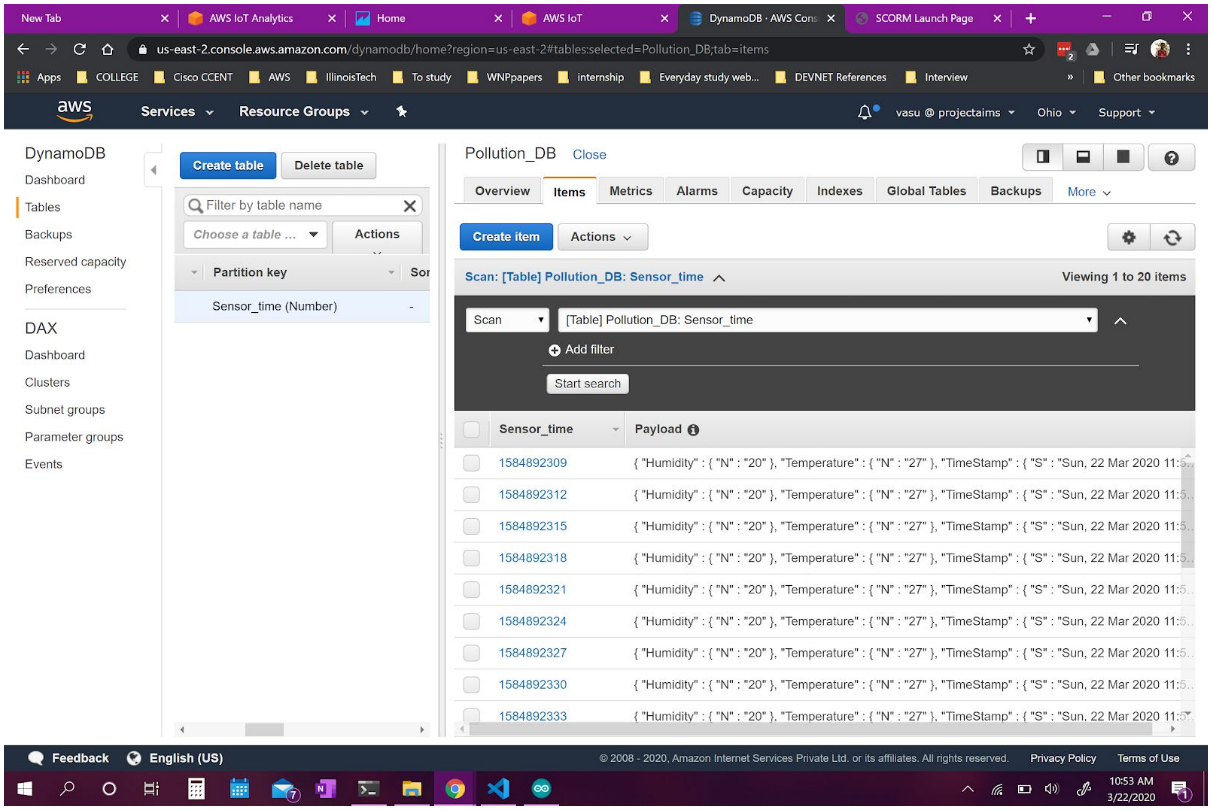
Actions are what happens when a rule is triggered. Learn more

Insert a message into a DynamoDB table
Pollution_DB

Send a message to IoT Analytics
WorkshopAnalysis_channel

Fig 6.0: Rules Engines establishment to store data in the database and create a dataset using IoT Analytics.

Fig 7.0 depicts the console of the DynamoDB database where the data is categorized using the unique sensor times. This console can be used to filter data according to certain limits or ranges. We can also add redundant backups to the database.



Sensor_time	Payload
1584892309	{ "Humidity" : { "N" : "20" }, "Temperature" : { "N" : "27" }, "TimeStamp" : { "S" : "Sun, 22 Mar 2020 11:5
1584892312	{ "Humidity" : { "N" : "20" }, "Temperature" : { "N" : "27" }, "TimeStamp" : { "S" : "Sun, 22 Mar 2020 11:5
1584892315	{ "Humidity" : { "N" : "20" }, "Temperature" : { "N" : "27" }, "TimeStamp" : { "S" : "Sun, 22 Mar 2020 11:5
1584892318	{ "Humidity" : { "N" : "20" }, "Temperature" : { "N" : "27" }, "TimeStamp" : { "S" : "Sun, 22 Mar 2020 11:5
1584892321	{ "Humidity" : { "N" : "20" }, "Temperature" : { "N" : "27" }, "TimeStamp" : { "S" : "Sun, 22 Mar 2020 11:5
1584892324	{ "Humidity" : { "N" : "20" }, "Temperature" : { "N" : "27" }, "TimeStamp" : { "S" : "Sun, 22 Mar 2020 11:5
1584892327	{ "Humidity" : { "N" : "20" }, "Temperature" : { "N" : "27" }, "TimeStamp" : { "S" : "Sun, 22 Mar 2020 11:5
1584892330	{ "Humidity" : { "N" : "20" }, "Temperature" : { "N" : "27" }, "TimeStamp" : { "S" : "Sun, 22 Mar 2020 11:5
1584892333	{ "Humidity" : { "N" : "20" }, "Temperature" : { "N" : "27" }, "TimeStamp" : { "S" : "Sun, 22 Mar 2020 11:5

Fig 7.0: Data stored in the DynamoDB table.

Fig 8.0 and 9.0 depict the AWS Quicksight console which is a cloud-based tool to provide data visualization. AWS Quicksight obtains a dataset from the AWS IoT Analytics stream and uses that data set to plot an Average Temperature vs Time Graph with a legend of Humidity and a Humidity vs Time Graph with a legend of Average temperature. Thus we can see the correlation between temperature and humidity and vice versa.

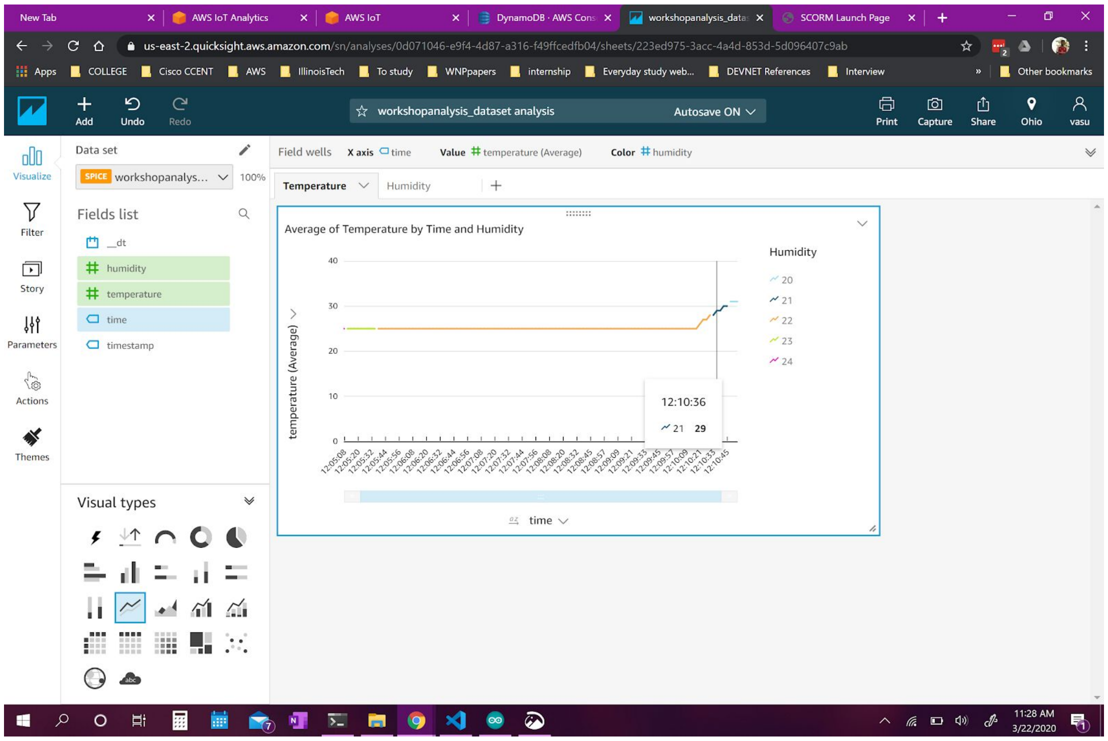


Fig 8.0: QuickSight used to create Average Temperature vs Time Graph with a legend of Humidity.

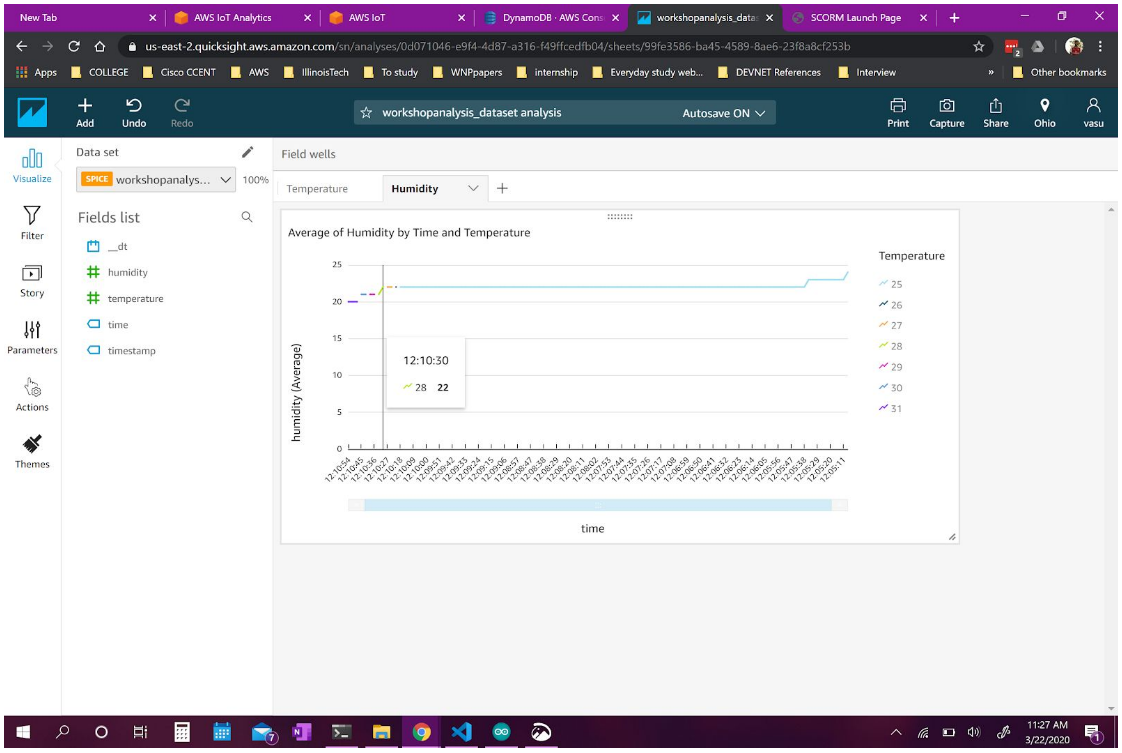
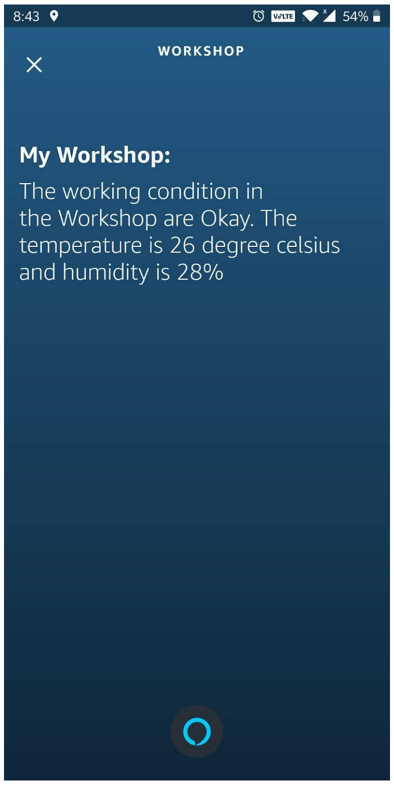


Fig 9.0: Quicksight user to create Average Humidity vs Time Graph with a legend of Temperature.

Finally in Fig 10 we can see the output of our Voice-based interface. Our Alexa Skill called the Workshop can be invoked by asking Alexa this query "Alexa, ask My Workshop, How is the Workshop?" or by asking "Alexa, ask My Workshop, Workshop conditions". These queries will execute the backend program that can access the database of sensor data to formulate a response. Fig 11 we can see the hosting of this program in AWS Lambda.

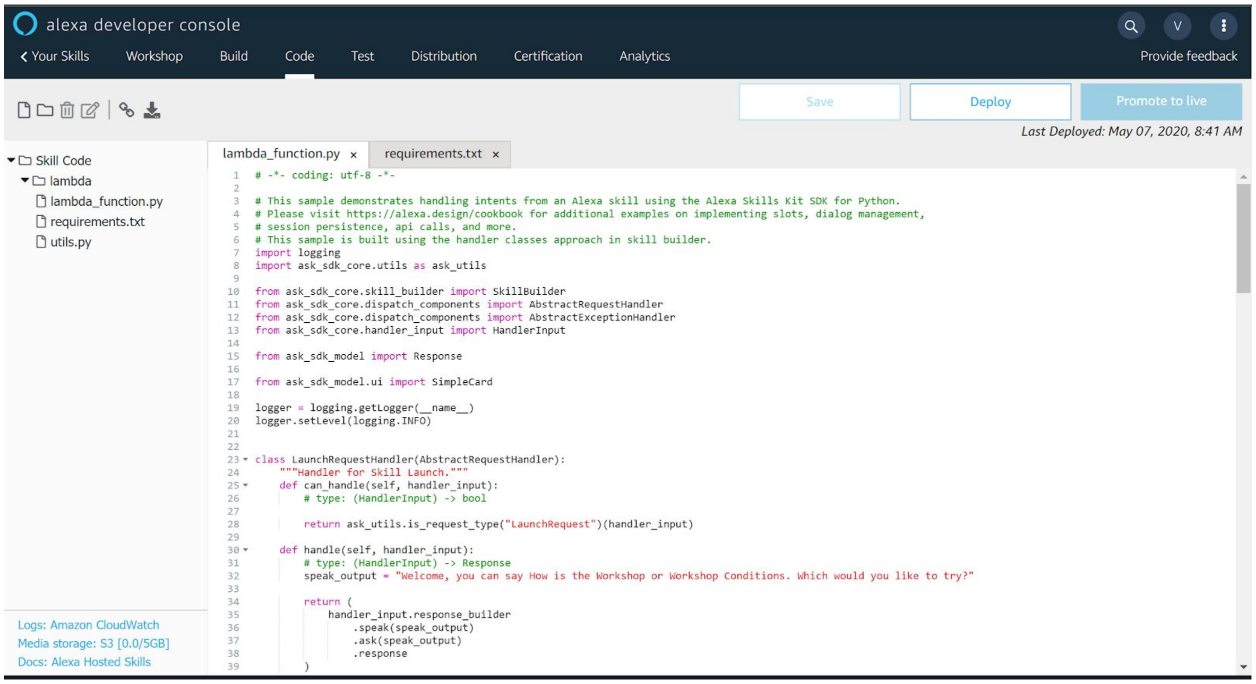


WORKSHOP

My Workshop:

The working condition in the Workshop are Okay. The temperature is 26 degree celsius and humidity is 28%

Fig 10. VUI output with sensor values.



```python
# -*- coding: utf-8 -*-

# This sample demonstrates handling intents from an Alexa skill using the Alexa Skills Kit SDK for Python.
# Please visit https://alexa.design/cookbook for additional examples on implementing slots, dialog management,
# session persistence, api calls, and more.
# This sample is built using the handler classes approach in skill builder.
import logging
import ask_sdk_core.utils as ask_utils

from ask_sdk_core.skill_builder import SkillBuilder
from ask_sdk_core.dispatch_components import AbstractRequestHandler
from ask_sdk_core.dispatch_components import AbstractExceptionHandler
from ask_sdk_core.handler_input import HandlerInput

from ask_sdk_model import Response

from ask_sdk_model.ui import SimpleCard

logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)


class LaunchRequestHandler(AbstractRequestHandler):
    """Handler for Skill Launch."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool

        return ask_utils.is_request_type("LaunchRequest")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        speak_output = "Welcome, you can say How is the Workshop or Workshop Conditions. Which would you like to try?"

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(speak_output)
                .response
        )
```

Fig 11. AWS Lambda hosting the backend logic of our My Workshop skill.

7. Discussion

There were certain deliverables that we promised in the initial proposal that could not be implemented as of this final report. We proposed a data parameter of sound in decibels that can depict the sound pollution in the workshop. This parameter was to be obtained using the MP34DT05 digital microphone built in the Arduino NANO. But as of implementation the microphone data could only be obtained as PDM data points and it was not possible to convert to decibel data points for human-readable understanding. We could not also obtain the CCS811 sensor which is a major component of our project due to the circumstances caused by the pandemic. During our initial proposal we only proposed a voice based user interface but to add some features we used AWS IoT Analytics and AWS Quicksight to also visualize the data we have obtained.

Question 1: Can the system sustain fire, rain, or any other natural phenomenon? Explain with technical reasons?

Our system in our initial design or deployment does not take into consideration sustenance against natural phenomena like fire, rain, etc. But through our research, we have found certain technical specifications or additional accessories that will protect our system. We found that the Raspberry Pi is qualified to operate in a temperature range of -40ºC and 85ºC. This operation range also applies to our sensing node, the Arduino NANO 33. These characteristics are suitable for normal operation in a workshop but not for extreme cases like a fire which is a high probability in a workshop. For this, we have discovered after-market housings that we can accessorize to our existing system. Weatherproof Enclosure for Raspberry Pi from Sixfab[2] provides IP65 rated housing. An IP65 rating means that it is dust-tight i.e there is no ingress of dust possible and protects against jets of water from all directions allowing near-perfect water-resistance. This housing enables the gateway to be weatherproof to harsh conditions like fire, rain, and also resilient to dust which is abundant in workshop environments or in case of a natural disaster. But when investigating methods to protect the sensing node, run into a problem. For proper gas humidity monitoring, unoccluded exposure is required for the sensor. This is not possible if we attach a weatherproof housing to Arduino NANO 33.

Questions 2: How would you correlate temperature, humidity, and air quality into a single 'meaningful' information?

Temperature, humidity, and air quality data as separate entities help us understand the working conditions in a workshop, model shop, or garage. Through research made in [3], we come to find out that for people in indoor environments there is a direct correlation between the temperature and humidity and the perception of air quality in the environment. It is observed that at constant pollution level when there is an increase in temperature and humidity, denizens in the environment perceive that there is a decrease in air quality[3]. It is found that in a constant pollution level the accessibility of air quality is linearly correlated with the enthalpy of the air[3]. In a workshop environment, it is quite common that the temperature of the environment will always be higher than the general room temperature of 25^{o}C. This means that workers might perceive that the working conditions are not safe even though the pollution level in the environment is kept below the acceptable threshold. Our system can provide accurate readings of the pollution level informing the workers that the environment is safe to work, discounting their doubts.

Question 3: Explain TVOC levels and describe how these levels would represent air quality.

VOCs or Volatile Organic Compounds are a combination of gases that are released from a myriad of toxins and chemicals found in everyday products. VOCs generally include Formaldehyde, toluene, acetone, ethanol, propane, butane, etc[4]. These VOCs can be found in paint, glues, aerosol sprays, disinfectants, etc which are ubiquitous in a workshop environment[4]. TVOC is a measure of the concentration of such gases. There are short effects like headaches, skin irritation, and coughing and long term effects like asthma and cardiovascular disease when exposed to VOCs. It is considered that if the concentration is between 0-250 ppb it means the VOC contents are low. If the concentration is between 250 to 2000 ppb and this concentration persists for a month there are sources that emit VOC and have to be regulated. If the concentration is greater than 2000 ppb serious action has to be taken for ventilation[5]. Indoor air is often 5 times more polluted by these pollutants than outdoor environments. Thus, the concentration of TVOC in an indoor environment directly correlates with the air quality in the location

8. Conclusion:

Our goal through this project was to understand the easy-of-use, deployment method, service integration, and scalability feature of enterprise-level cloud based IoT backend service. We have successfully implemented a scalable and easily deployable Pollution tracking and monitoring system with use of AWS Cloud services. The use of AWS Cloud service allowed us to easily integrate features like databases, dataset creation and visualization with very quick time-to-market or time-to-deployment. If we would have used traditional methods like creating our own server, or SQL database, scalability and time-to-deployment would have been drastically affected. Our system is one of the very few pollution monitoring systems that decouples the sensing node and the IoT gateway which connects to the backbone network. We see that this type of architecture allows for flexible scaling. This project also implements a novel method to interact with the pollution monitoring system through a Voice-based interface that is powered by the Alexa speech-to-text engine.

9. Future Work:

In future implementations of our system we would like to actuation of HVACs if the sensors measure pollutants that are above a certain threshold and also like to implement a machine learning algorithm that can correlate the use of certain machinery to certain pollutants to better understand the source of the pollution.

Question 1: How would you scale up your system by 1000 times (coverage)? Do you have to change the communication protocol? Will devices interfere with each other?

One of the main advantages of our system is easy scalability due to the fact we use AWS IoT cloud service as our backend which supports dynamic and plug-n-play scalability. AWS IoT cloud service all the services that we use on AWS is advertised to support up to a billion devices. Our system decouples the sensing node and the IoT gateway. Thus to get the entire picture of scaling the system 1000 times, we must both take into the consideration of scaling the number of sensing nodes and gateways. The sensing nodes and the gateway are connected using BLE. It is found that Raspberry Pi 3 B+ can connect up to 8 devices with one device as the master. This means 7 peripheral devices can be connected to each gateway. BLE uses the L2CAP(Logical Link Control and Adaptation) protocol which will take care of the interference or collision that will occur due to traffic caused by the simultaneous transfer of 7 sensing nodes. Thus to support 1001 sensing nodes the system would require 143 IoT gateways. The scalability and deployment of these gateways are taken care of by the AWS IoT Device Registry where the deployment and scalability of gateways are as easy a writing a script that sends the required authorization and credential, and software OTA(Over-The-Air) to the newly deployed IoT gateways. The communication between the gateway and the cloud service is through MQTT protocol that runs in a pub-sub model with builtin load balancers called the Elastic Load balancer in AWS. The gateways connect to an AP using WiFi 802.11ac. 802.11ac uses the CSMA/CA protocol to avoid collisions and OFDMA to avoid interference between multiple devices.

Question 2: Name and explain 3 industries/types of businesses that can be benefitted by adapting your system?

The system we proposed and implemented is meant for workshops, garages, and model-shops. These environments are likely to be indoor environments. Thus through research, we observe that this system can also be retrofitted to other indoor environments like offices. It is reported by the EPA(Environmental Protection Agency) that indoor environments like offices experience a phenomenon called SBS(Sick Building Syndrome). The occupants might experience symptoms like headaches, eyes, nose, or throat irritation, dry or itchy skin, nausea due[6]. SBS is understood to be caused due to inadequate ventilation, chemical contaminants from indoor sources like cleaning products and air fresheners which introduce VOCs, and biological contaminants like mold, pollen, and bacteria. Our system architecture can be easily retrofitted to office buildings where a couple of sensing nodes and a gateway can be deployed to each floor. The system can inform the denizens of the office that the air quality is below a certain threshold and ventilation has to be increased or inform them to step out for a certain time to mitigate prolonged exposure.

Through research and a proposed system in [7], we find that another industry that can use our system is the construction industry where construction workers are in constant exposure to particulate matter, aerosols, high temperature, and VOCs through the various construction materials they use like cement, asbestos, and paint. Construction sites are micro-environments that are subject to both indoor and outdoor pollutants. The sensors in our system have to be calibrated to set different thresholds of pollutants for indoor and outdoor environments. The mere information of the existing pollutants in construction sites can inform the workers to take appropriate breaks to mitigate prolonged exposure.

Factories are another industry that can retrofit our system. Factories tend to have a similar open floor layout and working condition as compared to workshops but on a larger scale. We have observed that our system is highly scalable through our IoT cloud service. Thus our system can support and perform at the same efficiency as in a workshop environment.

Question 3: Name and explain the 3 additional features you can add to your system other than your original plan, and how would you achieve them?

As per our final submission, our system does not have a visual user interface. The system provides a graphical visualization of the various parameters that we sense. We can add a web-based application that represents the layout of the workshop with the location of each machine and the depict a localized data value for each parameter near the machinery for more extensive knowledge on the operation of which machinery causes a decrease in air quality or increase in temperature in the workshop. Both Google and Apple provide Javascript-based SDKs and APIs called Google Maps API(https://www.google.com/maps/about/partners/indoormaps/) and MapKit JS (https://developer.apple.com/videos/play/wwdc2019/245/) to develop our indoor map allowing us to depict the floor plan of the workshop. Because we use AWS cloud service we can create a pipeline to publish our sensor data through HTTPS messages directly to the web interface and correlate the data to certain coordinates according to each sensing node to properly depict the spread-out sensor data concentration across the workshop floor plan.

Another addition we could add to the system is adding a particulate matter sensor that senses PM2.5 and PM10. This additional data parameter can increase the accuracy of our estimate of the air quality. Particulate matter is also the main cause of asthma, thus this additional data point can be valuable to denizens with asthma in that environment. The PM 2.5 sensor from Adafruit (https://learn.adafruit.com/pm25-air-quality-sensor) can be used here to measure particulate matter per 0.1L air. PM1.0, PM2.5, and PM10. are measured in both standard and environmental units. This sensor will be connected to our sensing node, the Arduino NANO 33 BLE Sense, through UART input with a baud rate of 9600. This sensor parameter has to be added as a characteristic to be sent over via BLE and added to the MQTT payload as a JSON object for interpretation by the AWS IoT service.

Another feature we would like to add to the system is the actuation of ventilation passages like windows through a stepper motor. If the VOCs in the workshop increase over a certain threshold we can automate opening a window for certain durations. This can be implemented by attaching a heavy gauge stepper motor to the hinge of the window, which is then connected to an Arduino MKR1000 microcontroller that is connected to the IoT gateway using WiFi which is an inbuilt module in this board. The gateway sends the command and control to the microcontroller for it to actuate the windows of the workshop.

Question 4: How would you reduce power consumption without introducing new devices? Explain how your method will be reducing your current system design's power consumption?

One of the methods to reduce the power consumption of IoT end-devices implementing a duty cycle restriction on the devices. This means that the device will be in off-state or sleep mode for certain periods of time, thus saving power. The end-device will change to the awake mode and transfer data only when it is necessary. In our system, the sensing nodes send data every second which may be deemed unnecessary. We can place a duty cycle restriction on the operation time of the sensing node allowing it to send data only during certain intervals during the day. This may affect the real-time nature of the application but provides better efficiency. We can also program the sensing node to send data to the gateway only when there is a change in the parameter value and not for every data point, this drastically reduces the number of transfers that occur and will also save power. Longer the end-device is in sleep mode or not transferring data more the battery can be saved. If the application does not require frequent or real-time updating or can operate with only deltas in the data, applying duty cycle restriction or synchronizing an off-period can reduce the power consumption.

Question 5: If one or more devices fail in your system, how would your system behave? What is the failure safety feature of your system?

The two points of failure in our system is in the sensing node or in the IoTgateway. In the fully implemented system of our project, multiple sensing nodes are deployed across the open floor layout of the workshop. If one the sensing nodes fail there would be observable decrease in coverage of the workshop floor space. This means certain data points from certain locations in the workshop would be lost providing inaccurate measure of understanding the working conditions of the workshop. To mitigate this redundant sensing node might have to be placed. Another way to mitigate loss of data points by sensing node failure is by increasing the sensing range of nearby nodes. This can be done by calibration of the sensors by sometimes lowering voltage thresholds. This solution has not been tested. Another point of failure in the system is when the IoT gateway fails. This means the communication channel between the sensing node and gateway is broken or from gateway to the backend cloud service is broken. In our system we know that the Raspberry pi can connect to 7 sensing nodes simultaneously. If the communication channel between the sensing node and gateway is broken it means that failure of one gateway would result in loss of data from 7 sensing nodes. Arduino NANO has 1MB of program memory, this can be altered to provide data logging capabilities. If the failure of the channel is temporary then the data can be logged on-board the NANO and then queued to offload the data after the connection is re-established. This would mitigate the loss of data-points. If the communication channel between the gateway and cloud service is broken, there are various methods to mitigate the loss of data points. One is on-board storage on the gateway to then be queued to be published to the AWS Message broker. AWS IoT Core provides a feature when connectivity is lost called Device Shadow. The Device Shadow keeps a persistent log of the last known state of the end-device before it lost connection. When connection is re-established then the Device Shadow is updated to the latest state thus keeping consistency for frontend applications.

References:

[1] J. Jo, B. Jo, J. Kim, S. Kim, and W. Han, "Development of an IoT-Based indoor air quality monitoring platform," J. Sensors, vol. 2020, pp. 13–15, 2020, doi: 10.1155/2020/8749764.

This reference introduced us to the concept of a pollution monitoring system in indoor environments. We deemed that the system proposed in this paper is not friendly to scalability. Thus we took inspiration from the system and proposed architecture that decoupled the sensors and communication gateway.

[2] https://sixfab.com/product/raspberry-pi-ip65-weatherproof-iot-project-enclosure/

The specification and solution to weatherproofing our system was obtained from this reference. Sixfab fabricates custom enclosures for a variety of development boards

[3] L. Fang, "Impact of temperature and humidity on the perception of indoor air quality," Indoor Air, vol. 8, no. 2, pp. 80–90, 1998, doi: 10.1111/j.1600-0668.1998.t01-2-00003.x.

This reference allows use to correlate the relationship between temperature , humidity and air quality for better understanding how the denizens of the environment where the system is located can understand the working conditions

[4] https://www.epa.gov/indoor-air-quality-iaq/technical-overview-volatile-organic-compounds

Teaches us about the sources of VOCs and the gases that constitute a VOC.

[5] https://www.airthings.com/what-is-tvoc

Teaches us about the levels of VOCs that are acceptable in a environment.

[6] https://www.epa.gov/sites/production/files/2014-08/documents/sick_building_factsheet.pdf

This reference from the EPA teaches us about the Sick Building Syndrome which is a phenomenon in indoor environments.

[7]M. S. Wong, E. Mok, T. Wang, and Z. Yong, "Development of an Integrated Micro-Environmental Monitoring System for Construction Sites," *in Proceedings of the International Conference on Geographies of Health and Living in Cities: Making Cities Healthy for All, Healthy Cities 2016, Hong Kong, 2016, pp. 207–214*

This reference informs us about a pollution monitoring system in micro-environments like Construction sites.

[8] https://developer.amazon.com/en-US/alexa/alexa-skills-kit

Reference and tutorials to build the VUI using Alexa. This website provides documentation and sample codes to implement an Alexa Skill.

Appendix A

Code:

For Sensing Node i.e Arduino NANO 33 BLE Sense:

```
#include <ArduinoBLE.h>
#include <Arduino_HTS221.h>
// BLE battery characteristics
BLEService pollutionService("190f");
//BLE Pollution Characteristics
BLEUnsignedCharCharacteristic temperatureLevelChar("2b19", BLERead | BLENotify);
BLEUnsignedCharCharacteristic  humidityLevelChar("2c19",  BLERead  |  BLENotify);//  standard  16-bit  characteristic
UUID
// remote clients will be able to get notifications if this characteristic changes
void setup() {
    Serial.begin(9600);
    while(!Serial);
    if (!BLE.begin()) {
        Serial.println("Staring BLE Failed!");
        while(1);
    }
    if (!HTS.begin()) {
    Serial.println("Failed to initialize humidity temperature sensor!");
    while (1);
    }
    /* Set a local name for the BLE device
     This name will appear in advertising packets
     and can be used by remote devices to identify this BLE device
     The name can be changed but maybe be truncated based on space left in the advertisement packet
    */
    BLE.setLocalName("PollutionMonitor");
    BLE.setAdvertisedService(pollutionService);
    pollutionService.addCharacteristic(temperatureLevelChar);
    pollutionService.addCharacteristic(humidityLevelChar);
    BLE.addService(pollutionService);
    /* Start advertising BLE.  It will start continuously transmitting BLE
     advertising packets and will be visible to remote BLE central devices
     until it receives a new connection */
    // start advertising
    BLE.advertise();
    Serial.println("Bluetooth device active, waiting for connections ... ");
}
void loop() {
    // wait for a BLE central
    BLEDevice central = BLE.central();
    if(central) {
        Serial.print("Connected to central: ");
        // print the central's BT address:
        Serial.println(central.address());
        // turn on the LED to indicate the connection
        digitalWrite(LED_BUILTIN, HIGH);
        //while central is connected:
        while (central.connected()) {
                float temperature = HTS.readTemperature();
                float humidity = HTS.readHumidity();

                temperatureLevelChar.writeValue(temperature);
                Serial.println("Temperature:");
                Serial.println(temperature);
                humidityLevelChar.writeValue(humidity);
                Serial.println("Humidity:");
                Serial.println(humidity);
                delay(1000);
        }
        digitalWrite(LED_BUILTIN, LOW);
        Serial.print("Disconnected from central: ");
        Serial.println(central.address());
    }
}
```

For IoT Gateway i.e the Raspberry Pi 3 B+:

```python
from bluepy import btle
import json
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
import time
from time import sleep

#authentication information and topic name
rootCAPath = "root-CA.crt"
certificatePath = "c5ffe0127c-certificate.pem.crt"
privateKeyPath = "c5ffe0127c-private.pem.key"
topic = "poll/tracking"

myAWSIoTMQTTClient = AWSIoTMQTTClient("myClientID")
myAWSIoTMQTTClient.configureEndpoint("a1jgcb96hr49vu-ats.iot.us-east-2.amazonaws.com", 8883) #endpoint of aws iot
core service
myAWSIoTMQTTClient.configureCredentials(rootCAPath, privateKeyPath, certificatePath)

# AWSIoTMQTTClient connection configuration
myAWSIoTMQTTClient.configureAutoReconnectBackoffTime(1, 32, 20)
myAWSIoTMQTTClient.configureOfflinePublishQueueing(-1)  # Infinite offline Publish queueing
myAWSIoTMQTTClient.configureDrainingFrequency(2)  # Draining: 2 Hz
myAWSIoTMQTTClient.configureConnectDisconnectTimeout(10)  # 10 sec
myAWSIoTMQTTClient.configureMQTTOperationTimeout(5)  # 5 sec

myAWSIoTMQTTClient.connect()

print("Connecting....")
dev = btle.Peripheral("E2:B4:50:FA:1D:D9") # peripheral device MAC ADDRESS
for svc in dev.services:
    print(str(svc))

pollution_sensor = btle.UUID("190f")
pollution_service = dev.getServiceByUUID(pollution_sensor)
for ch in pollution_service.getCharacteristics():
    print(str(ch))
temperature_uuid = btle.UUID("2b19")
humidity_uuid = btle.UUID("2c19")
temp_value = pollution_service.getCharacteristics(temperature_uuid)[0]
humidity_value  =  pollution_service.getCharacteristics(humidity_uuid)[0]  #  getting  sensor  data  from  the  BLE
characteristics

# function to process the data into proper formatting
def converter(data):
    data = str(data)
    print("Raw data: {}".format(data))
    data = data.strip('b')
    data = data.strip("'")
    data = data.strip('\\r\\n')
    data = data.strip("x")
    data = int(data, 16)

    return data

# Read sensor
if __name__ == "__main__":
    while True:
        temp = temp_value.read()
        humidity = humidity_value.read()
        temp_int = converter(temp)
        humidity_int = converter(humidity)
        milli = int(round(time.time()))
        current_time = time.strftime("%a, %d %b %Y %H:%M:%S %Z", time.localtime(milli)) # adding timestamp
        only_time = time.strftime("%H:%M:%S", time.localtime(milli))
           sensor_data = {"Sensor_time": milli, "Temperature": temp_int, "Humidity": humidity_int, "TimeStamp":
current_time, "Time": only_time}
        sensor_json = json.dumps(sensor_data)
          myAWSIoTMQTTClient.publish(topic, sensor_json, 1) # publishing json to poll/tracking topic in AWS Iot
message broker
        print('Published Topic %s: %s \n' % (topic, sensor_json))
        sleep(1)
```

Appendix B

Bill of Materials:

- CCS811 Sensor: $19.95
  - https://www.adafruit.com/product/3566
- Arduino NANO 33 BLE Sense: $31.10
  - https://store.arduino.cc/usa/nano-33-ble-sense
- Raspberry Pi 3 Model B+: $35.00
  - https://www.adafruit.com/product/3775?src=raspberrypi
- AWS IoT Core: Free Tier Version
- AWS Developer Suite: Free Tier Version