Internet of Things Smart Farming

Taylor McKinney

Fall 2019

Abstract

Technology is seeping its way into every sector, including agriculture. Most high-tech solutions for agriculture are expensive and inaccessible to local farmers, as was pointed out to me by two farmers and full time professors I interned with the summer of 2019. My goal of this project is to explore popular technology that I was previously very unfamiliar with, including the Arduino, Amazon Web Services (AWS), and just embedded systems and hardware-based projects in general. The other main goal is to create a prototype of an Internet of Things (IoT) system that could one day be used for agricultural monitoring. The project uses an Arduino MKR Wifi 1010, inexpensive sensors, and the free version of many services available on AWS to send email alerts and visualize sensor data.

Keywords: Arduino, AWS, IoT, DHT11, MQTT

Table of Contents

Figures

Tables

Introduction

The rapid development of technology and the growing ease at which we can access technology has radicalized many industries, including one that surprises many: agriculture. Larger “factory farms” have made use of a technique dubbed “precision farming” or “smart farming”, which requires expensive high-tech equipment that is made to maximize profits and efficiency while minimizing labor and production costs. Some examples include machines that use GPS and the coordinates of the field to plant seeds in perfectly straight, perfectly spaced rows, software that allows the farmer to keep track of crop yield based on location on the farm as well as how much water, nutrients, and herbicide was applied to each location, and sensors that allow the farmer to remotely monitor their crops [1].

While these technologies are extremely useful for big farms, they’re often not feasible, affordable, or scalable for smaller, local farms. I first learned of these issues and realized the need for a smart farming solution for smaller farms while I interned for a small, local, sustainable farm, Heartwood Farms [2] during the summer of 2019. While the owners live on the farm, they both have full-time professor jobs, and find it difficult to monitor their crops when summer break ends. They also expressed a difficulty with greenhouse farming because of their busy schedules during the university semesters, as greenhouse farming can be very precarious and difficult to monitor remotely. Along with learning powerful technologies that are popular in the field, a major goal of this project is to develop a prototype of an Internet of Things system that Heartwood Farms might one day be able to use for agriculture monitoring.

An Internet of Things (IoT) system is comprised of several “smart” objects or devices that collect, monitor, or produce data of some sort that are connected through a gateway, and can communicate with each other, the internet or the cloud, other devices, etc [3]. It’s a general concept that has many options for implementation, including several open source platforms that handle the lower-level connections, security, and storage that allow you to implement the IoT concepts quickly and easily. Similar IoT systems for agriculture exist already, but my main focus for this project is making this technology accessible to farmers who have little to no technical background by utilizing the free Amazon Web Services (AWS) IoT platform and simple, affordable Arduino hardware. The model of Arduino I’m using, the MKR 1010 Wifi [4], retails for just under $35. The sensors are also cost less than three dollars each [5, 6], making the total cost for the system around $40.This low-cost, lightweight solution could potentially benefit farmers by allowing them to better monitor their crops, respond to undesirable conditions (ie dry soil, high greenhouse temperatures, etc) faster, and conserve time and resources by

knowing the exact state of their fields. This solution should also yield a higher crop profit, as it will be a tool for the farmer to decrease crop loss and waste.

The AWS IoT platform has a multitude of built-in features, including the framework to build a web application that interacts with the devices, compatibility with the MQTT communication protocol to communicate with the device, the ability to send email or text notifications, and more [7]. The application consists of a simple dashboard that allows the farmer to see the most recent data collected from several sensors placed in their fields, including data about the soil moisture, temperature, and humidity. Different plants require different conditions, but all vegetables require around 40%-80% soil moisture [8], so the application would use that as a default value to determine whether the crops need water or not. The application will also give the farmer alerts, a feature supported by AWS Simple Notification Service (SNS), if the soil moisture were to ever go below the 40% threshold, or if perhaps the temperature was dangerously high for the crops. This would be especially useful for greenhouse farming, where the optimal temperature range is 50°F-68°F [9] and the optimal relative humidity range is 50%-70% [9], which can be difficult to maintain and monitor remotely. The system would serve as a "black box" to the farmer, in that they will simply need to place the physical component of the system in the field, connect it to WiFi, and then be able to receive data at various time increments that is interpreted and displayed in an understandable fashion.

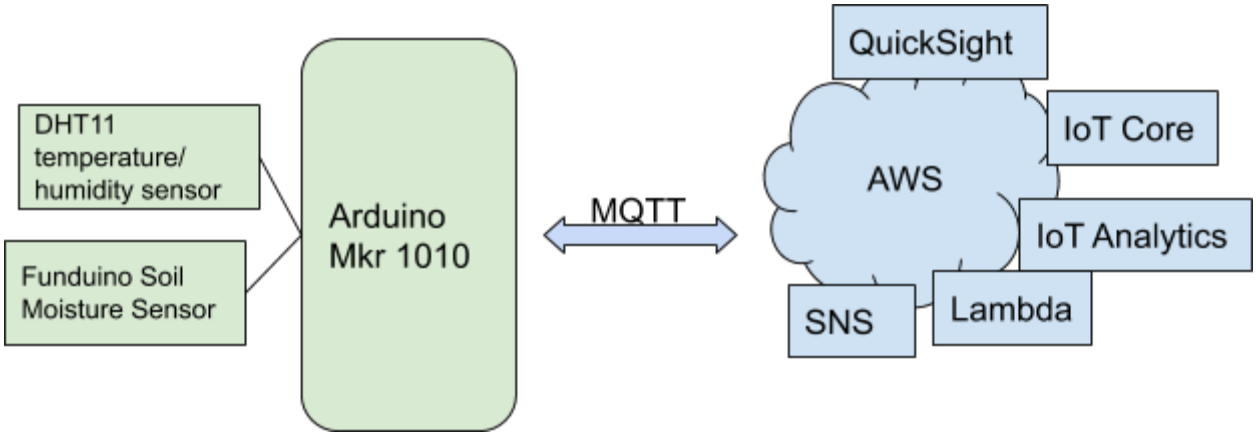


Figure 1 - Basic system block diagram
This is a simple representation of the main components of the system. The green blocks represent hardware and the blue blocks represent software. The Arduino board sends and receives data from AWS using the popular protocol MQTT, as detailed below in the Design section.

Design, Development, and Testing

Design

In order to complete the goals of this project, the IoT system contains both hardware and software components that are interconnected and working together. This section describes and defines the major technical components of the system that are shown above in Figure 1. The hardware components, the Arduino and the sensors, collect temperature, humidity, and soil moisture data and then send the data to the software components, MQTT and Amazon Web Services, in a process detailed below. The relevant Amazon Web Services descriptions and their use in the system are detailed below in Table 1.

Hardware Components

Arduino MKR WiFi 1010—an open-source electronics platform. [4] I specifically chose to use the MKR 1010 because it's WiFi enabled, which allows data to be sent to and from the Arduino wirelessly, which was an attractive feature for my project as the board will need to be part of the physical component that goes in the field. In order to communicate via WiFi, I used the following open source Arduino Libraries: *WifiNINA* [15], *ArduinoBearSSL* [16], *ArduinoECCX08* [17], and *ArduinoMqttClient* [18]. These libraries all work together to provide crucial methods for connecting the board to WiFi and then connecting it to AWS. I also utilized the *EduIntro* library [19] to read temperature and humidity data from the DHT11 sensor.

Sensors—Two sensors are connected to the Arduino to gather moisture, temperature, and humidity data. For temperature and humidity, I'm using the DHT11 [5] and for soil moisture I'm using the Funduino Moisture Sensor [6]. Both sensors have three pins each: power (VCC/V), ground (GND/G), and data signal (SIG/S). The power and the ground pins of both sensors are connected to the VCC (3.3V) and GND ports on the Arduino, respectively. Both sensors are able to use 5V power as well if used with a different board, but the MKR doesn't support and could be damaged by voltage higher than 3.3V. The DHT11's data pin is connected to any Digital pin on the Arduino, but the Soil Moisture Sensor's data pin is connected to the first Analog pin (A0). Digital data only has two values, HIGH and LOW, so that's why the DHT11 requires a library to read data from it, whereas the Soil Moisture sensor produces Analog data, which can be any range of numbers, that can be read straight from the device.

Software Components

MQ Telemetry Transfer (MQTT)—a lightweight messaging protocol. [11] MQTT is the essential link between the Arduino and AWS and is how they are able to send data back and forth. MQTT was specifically designed with the goal of minimizing network bandwidth and device resource requirements while still maintaining reliability. This makes it perfect for an IoT system, because it allows multiple devices or services to communicate without needing perfectly compatible APIs. MQTT works by having user-defined 'topics' to which you can either publish or subscribe to. Messages, also called payloads, are typically formatted in JavaScript Object Notation (JSON), which is just to ensure various endpoints can interpret and understand the data. A JSON object consists of unordered key/value pairs, or attributes, that are formatted in specific ways. [12] Below is an example of what a payload might look like in this system.

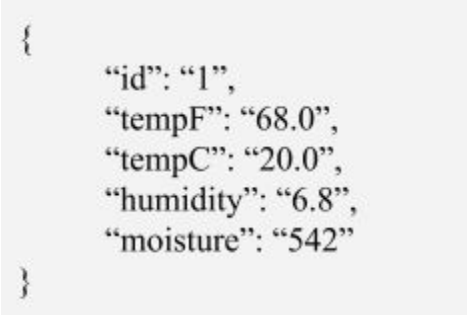


Figure 2 - Example MQTT Payload
This is an example with arbitrary data of the format that MQTT payloads are sent and received in this system. The curly brackets, quotations, commas, and line breaks are all part of the JSON format.

Amazon Web Services—a cloud-based platform that has many powerful free features, allowing me to connect devices with different protocols and have them communicate through the device gateway. It also has security measures in place, has database storage, and supports mobile app development that is integrated with the IoT system. [7] The specific services utilized in this project are detailed in the table below.

Table 1- Amazon Web Services Used

Service	Description	Use in System
Internet of Things (IoT) Core	An IoT platform that is the driving service of this system. It consists of a "shadow", a virtual representation of the	All of the communication with the Arduino is done by IoT Core, and then the data is processed and sent to the

	Arduino, which the Arduino connects to using a remote broker, which is basically a link to that specific Amazon service. IoT Core also has built-in MQTT publish/subscribe functionality, as well as Iot Rules that are used to send data to other services to be interpreted.	appropriate endpoints. IoT Rules use a SQL query to select a certain subset of data that you want to trigger each action. For example, the system has an IoT Rule that is triggered whenever the temperature attribute of the MQTT payload is above 68°F that sends a signal to Lambda to send out an email notification using SNS.
Simple Notification Service (SNS)	A notification service with a very explanatory title. The service offers email and SMS notifications, but I chose to use the email option. The service works by having a 'topic' to which an email can subscribe to. An automated email is sent to confirm subscription to the topic, and once the confirmation link is clicked by the recipient, they will begin receiving emails every time a notification is published to the topic.	The SNS topic for this system is called SmartFarmAlert. To subscribe the farmer to the topic, the system administrator must create a new subscription to that topic using their email. Once the farmer clicks the confirmation link, they will receive emails anytime the conditions are outside the desirable thresholds. The control logic for this is handled by Lambda; SNS is solely responsible for sending out the email and providing a dashboard for the administrator to manage the subscriptions to the topic.
Lambda	AWS Lambda is a serverless architecture that allows the user to run code without worrying about servers, storage space, etc. The user creates a Lambda function that has a trigger, code to be run, associated AWS services, and more that is ready to be run as soon as it is triggered.	The Lambda function for this system takes as input the MQTT payload from IoT Core and runs Python code that checks the moisture, humidity, and temperature data for undesirable thresholds and then publishes a specified message to the smartFarmAlert SNS topic, which then sends the email to the farmer.
IoT Analytics (Channel and Pipeline components)	IoT Analytics as a whole is an analysis service that allows	The Channel for this system (smartFarm_channel) listens to

	you to clean up, enrich, and contextualize IoT data. The Channel component is how the service knows what MQTT topic to listen to and the Pipeline component is where the user is able to modify the data however necessary.	the same topic as IoT Core, "arduino/outgoing". The Pipeline (smartFarm_pipeline) for this system adds datetime data to each incoming MQTT payload.
IoT Analytics (Datastore and Dataset components)	The IoT Analytics service also has components called the Datastore and Dataset, which store the data after it has gone through the Pipeline. The Datastore is not quite a database, but it is a "scalable and queryable repository" [7]. The Dataset is the result of a given query on the Datastore.	The Datastore for the system (smartFarm_datastore) is where all of the raw sensor and datetime data is stored; the Dataset (smartFarm_dataset) is the result of "SELECT * FROM smartFarm_datastore". The Dataset is then used by QuickSight for visualization purposes, while the Datastore maintains the historical data.
QuickSight	A powerful and dynamic visualization and analytic tool that allows the user to quickly represent data in various formats using ad-hoc queries. It also allows the user to create interactive dashboards and business insights that can be shared with multiple users.	QuickSight is the main way the farmer interacts with the data. There is a dashboard that visualizes the sensor data collected and processed by the system that the farmer can easily access by signing on to the AWS Console. There, they can see the data displayed in a line graph, and are able to manipulate the data and change what is displayed however they would like.

Development

The main features I planned on implementing to create a prototype of a plant monitoring system are listed in Table 2. When I first decided which features I would focus on, I divided them into the categories essential, desired, and stretch; I did this to focus my attention first on the parts of the project that were essential to get a working prototype, and then expand to the features that would enhance the

usability of the system. This section then thoroughly details the steps of development and reasoning behind key design decisions organized by feature.

Table 2 - Features Organized by Importance

Feature	Brief Description	Class	Current Status
Gather and transmit data from sensors	Design a circuit using an Arduino, DHT11 sensor, and soil moisture sensor that can gather temperature, humidity, and soil moisture data. The Arduino code will then send the data in JSON format over MQTT to AWS.	Essential	Complete
Send alerts based on predetermined thresholds	Based on the generic soil moisture and temperature thresholds mentioned above, the system would send alerts using AWS SNS and Lambda, as described in the Design section.	Essential	Complete
Store historical data	Develop a simple database to store the data from the sensors, including temperature in Fahrenheit and Celsius, percent humidity, percent soil moisture, and datetime information.	Essential	Complete
Dashboard with simple visualization	Using AWS QuickSight, create a very simple visualization (most logically a line graph) of the data produced by the sensors.	Essential	Complete
Web application with more detailed visualization	Write a web application to allow for more detailed visualization, as well as the ability to use different queries of the data being displayed.	Desired	Incomplete
Flexibility in thresholds	For now, I'm going to have pre-set conditions that send off a notification, but a stretch goal I have is to allow the user to put in different conditions that they'd like to watch for, to allow for the variance in favorable conditions for	Stretch	Incomplete

	various plants.		
Package system in weather-proof container	Create a container that can safely house the Arduino and sensors on the farm in a way that allows them to collect accurate data while being protected from the weather. The goal of this project is to create a prototype of a system that could be used for agriculture to explore the technologies involved, so complete practicality may be forfeited to allow more time with AWS.	Stretch	Incomplete

In addition to completing the desired features, one of the main goals of this project was to explore embedded systems and Amazon Web Services, as I previously had little to no experience with those powerful and widely used technologies. Because of this, I started development with the hardware. In order to build my circuit, I used a solderless breadboard, as this would allow me to move the components around as much as needed to connect them all. One of the main reasons I chose to use the Arduino for the hardware side of the system is because of the online community and extensive documentation; there are hundreds of tutorials that walk you through how to connect and code various projects, and all of the code is open source.

Gather and transmit data from sensors

I first found one of these tutorials for connecting the DHT11 temperature and humidity sensor on circuitbasics.com [13]. I connected the VCC (+) pin on the sensor to the VCC (3.3V) pin on the arduino using a red wire, the ground (-) pin on the sensor to the GND pin on the arduino using a black wire, and the SIG pin on the sensor to the D7 (digital) pin on the arduino using a blue wire. Below is a schematic from the website that represents the circuit at this point of development, with the difference being the schematic shows an Arduino Uno instead of an Arduino MKR; there are some differences in the two Arduino boards, but the pins and connections are the same.

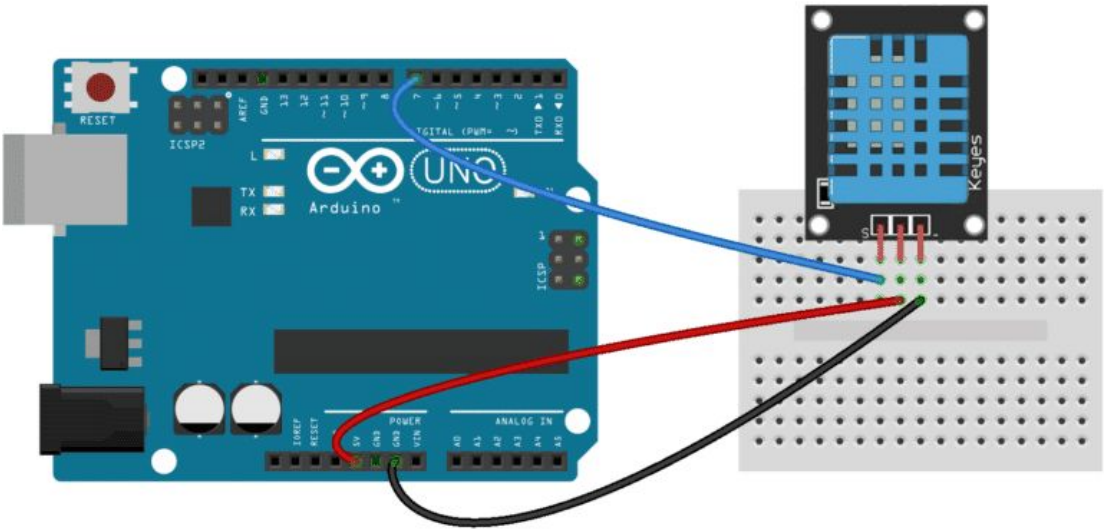


Figure 3 - DHT11 Circuit
A schematic of the circuit in the first stage of development borrowed from circuitbasics.com [13].

After connecting the circuit, running, and testing the open source code provided in the tutorial, I moved on to connect the soil moisture sensor. I was able to find a project tutorial to do this on the arduino.cc website itself [14] and made slight modifications due to the fact that I had already connected one sensor. The Arduino MKR has a 5V pin, but using devices with a voltage greater than 3.3V can damage the board, so I designed the circuit such that they were both connected to the VCC (3.3V) pin to avoid potential damage. To do this I connected additional red and black wires to the ones connected to the DHT11 sensor and also connected them to the corresponding pins on the soil moisture sensor so that they could share the same power source. I then used a yellow wire to attach the SIG (signal) pin on the soil moisture sensor to the A0 (Analog) pin on the Arduino. The code provided in this tutorial utilized the D7 pin on the Arduino to facilitate the reading of the values from the sensor; when D7 is HIGH, the data is read. Because of this, I simply moved the DHT11 sensor to the D6 pin, as the only requirement is that it's plugged into a digital pin. Below is a diagram borrowed from the Arduino website that shows the connections needed for the circuit that reads soil moisture data from the sensor. While I chose to connect the DHT11 sensor first, this was an arbitrary decision and it wouldn't affect the system if I started with the circuit depicted below and then added the DHT11. The most important thing at this point of development is ultimately that each pin on the sensors is connected to the correct pin on the Arduino, regardless of how complicated the circuit is.

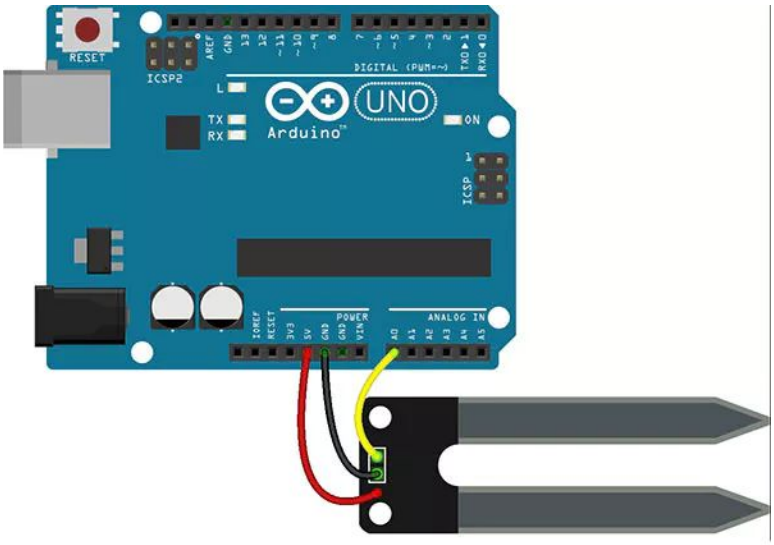


Figure 4- Soil Moisture Sensor Circuit
This is a schematic showing the connections necessary to gather soil moisture data from the sensor. This is borrowed from the arduino.cc website [14] and depicts an Arduino Uno, whereas in my project I used an MKR-- the connections are the same for both boards.

Next, I combined the code needed to read data from both sensors and continuously polled for data in short time increments to test the accuracy of the sensors, a process which is detailed further in the testing section below. The code I used to read sensor data can be found in the Appendix [A1] and the final circuit of the Arduino and sensors can also be found in the Appendix [A4].

I then moved on to establish a connection between the circuit and Amazon Web Services. I found a tutorial on the arduino.cc website titled Securely Connecting an Arduino MKR WiFi 1010 to AWS IoT Core [10] and followed the directions closely. To start this process, I had to use sample code from the ArduinoECCX08 Library [17] to generate a Certificate Signing Request (CSR) that AWS IoT requires to authenticate and securely connect the board to MQTT. I then used the CSR to create a "Thing" in AWS IoT Core, which acts as a virtual representation of the Arduino board. The tutorial then described the process of creating security policies for the Thing, as well as how to locate the Certificate and MQTT endpoint generated by AWS for the Thing, which are used by the Arduino code to connect the board to AWS through MQTT. I generated the code needed to do this using the *WifiNINA* [15], *ArduinoBearSSL* [16], and *ArduinoECCX08* [17] libraries and example open source code associated with them. This code can be found in the Appendix [A2].

After establishing a secure connection between the Arduino and AWS, I used the *ArduinoMqttClient* library to pass messages between them over the MQTT protocol. The code for the helper function that publishes a message to the MQTT topic can be found in the Appendix [A3]. MQTT

allows for either endpoint, AWS IoT Core or the Arduino, to publish or subscribe, but this system only really requires the Arduino to publish the sensor data to a topic and for AWS to subscribe to the topic and receive said data. I combined all of the code snippets into one file that connects the Arduino to WiFi, then connects it to MQTT, then connects it to AWS, reads sensor data in 30 second intervals, and then formats and publishes a payload to the MQTT topic "arduino/outgoing". The complete Arduino code for the system can be found in my Smart Farm repository on GitHub [20].

Send alerts based on predetermined thresholds

Once I was able to get the payloads into IoT Core in JSON format, the focus of development shifted to using Amazon Web Services to process, store, react to, and visualize the data in meaningful ways to the farmer. For all of these actions, I used the tutorials and documentation on the AWS website [7]. I initially found the documentation to be difficult to navigate, but once familiar with the organization I found it robust and extremely helpful in quickly understanding how each of the services worked and in figuring out which services I needed to use.

The feature I considered most important and thus began with was the notification of undesirable plant conditions. To do this, I needed an AWS Simple Notification Service (SNS) topic, which I named SmartFarmAlert, and a subscription to it. I chose to use email notifications, mainly because the intended user preferred that method, but AWS SNS offers many options for notifications including Text Messaging (SMS), desktop notifications from an application, and notifications to a web server via HTTP. No matter the form of notification, a confirmation of the subscription from the endpoint or user is required. In the case of email, AWS sends a confirmation email to the given email address with a link that the person must click in order to receive any messages that are published to the topic. As soon as they click the link, they will receive any future messages published to the topic to which they are now subscribed. Each message has a link to unsubscribe that is automated by AWS so they may cancel their subscription at any time.

The next design challenge I faced was the triggering of the notifications. SNS facilitates the notification subscription and delivery, but I had to find another AWS resource to perform the logic based on the sensor data. I first tried to implement this using IoT Rules, which allow you to run SQL queries on the incoming payloads to determine what data triggers certain actions from other services, but I didn't like the rigidity of that method. I found there was no easy way to customize the message and fine tune the thresholds using this method, as that required creating a different IoT Rule for each query, and the entire payload was just sent as an email, which isn't very intuitive to the human eye as we saw above in Figure 2. I then decided to use Lambda, which allowed me to run simple Python code that decides what message

to send based off the incoming values, and is able to compare multiple values at once. Figure 5 shows the flow for the Lambda function I created for this system, notificationControl, which is triggered by AWS IoT and has a destination of Amazon SNS. I created an IoT Rule, named sendToLambda, that would send all payloads to notificationControl. The actual Python 3.8 code that's run to handle the "event", or in this case the incoming JSON payload, can be found in the Appendix [A5]. It contains the logic that checks the moisture, temperature, and humidity values and sends relevant and easily understandable messages in an email to anyone subscribed to the SmartFarmAlert SNS Topic.

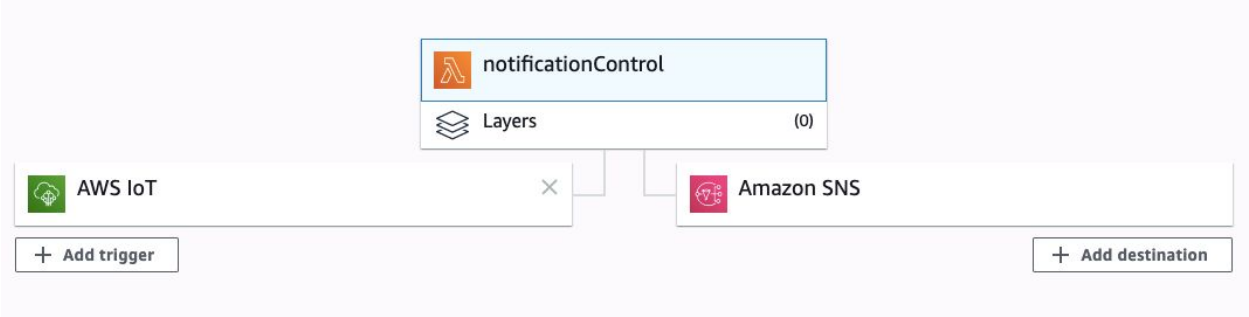


Figure 5- Lambda Function Flow
The Lambda function that controls when email notifications are sent is called notificationControl. This is a screenshot of the flow designer tab, where you are able to add various triggers and destinations for the function.

Store historical data

The next essential feature I wanted to implement was the ability to store historical data. The purpose of this is so that one day in the future the farmer may be able to compare data trends over time and track the impact of any changes they make in crop management on the measured properties. In early stages of development, I used AWS DynamoDB, a NoSQL database service, to store the data, but after exploring the steps necessary to visualize the data, I realized that storage is built in to IoT Analytics, thus making the DynamoDB database redundant. IoT Analytics has four components that allow the system administrator to process, manipulate, and enrich the IoT data. The components—the pipeline, channel, datastore, and dataset—are described in the Design section above, and at this point mostly just pass the payload through and add datetime data. The smartFarm_datastore is where the raw data from the payloads are stored, and the smartFarm_dataset then uses the query "SELECT * from smartFarm_datastore" to generate organized data for AWS QuickSight to visualize. Luckily, I had high control over the format of the IoT data so this step is mostly just a bridge between IoT Core and visualization software that allows for more business analytics, dynamic data modification, and computation necessary for this project.

Dashboard with simple visualization

The final essential feature I developed was the visualization of the data in a meaningful way to the farmer. I had originally wanted to create a web application myself to display the data, but my lack of experience and remaining time caused me to use resources through AWS to visualize the data. AWS QuickSight is a sandbox-style visualization tool that allows the system administrator to create various dashboards, graphs, and views, as well as maintain which AWS accounts have which permissions to each. I used this to create a simple dashboard with two line graphs: Figure 6a shows the graph displaying percent humidity and percent soil moisture and Figure 6b shows the graph displaying temperature in degrees Fahrenheit. The farmer would then have to create an Amazon Web Services account to view this dashboard with the two graphs, but in future work I'd like to explore exporting the visualization to a web application. This would also allow me to better format the x-axis, as the best way I could display the data at this point is by unique id of the payload, as the datetime data attached by the IoT Analytics is always the exact same time and therefore doesn't help visualization purposes. The data displayed below was gathered at thirty second increments, so there isn't much variance in values, but the graphs serve as proof of concept for visualizing the data.

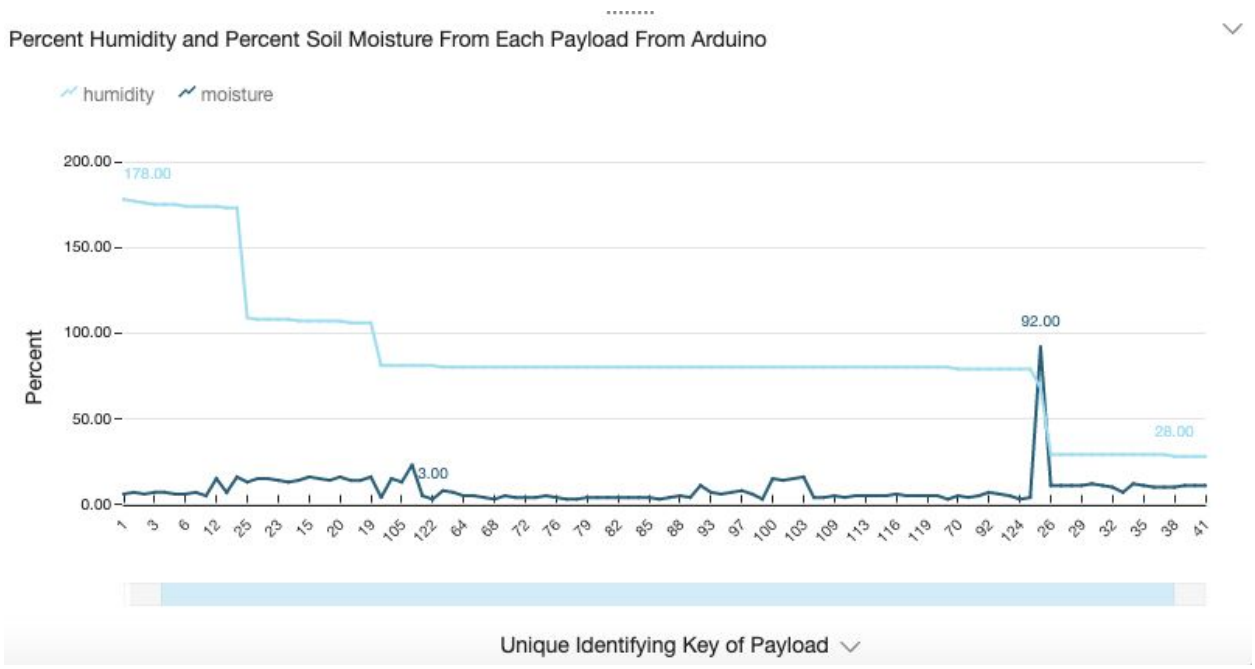


Figure 6a- Line Graphs Generated by AWS QuickSight
This graph shows the percent humidity and percent soil moisture of each incoming payload from the Arduino. The minimum and maximum values of each property are labeled on the graph for ease of reading.

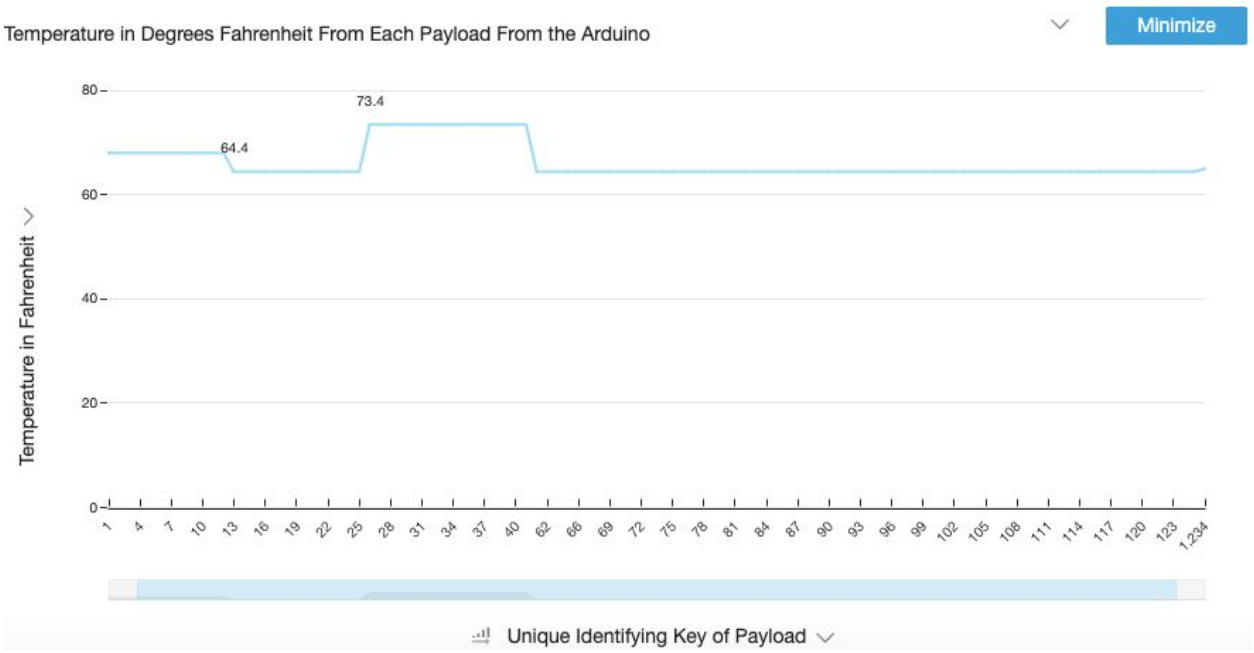


Figure 6b- Line Graphs Generated by AWS QuickSight
This graph shows the temperature in degrees Fahrenheit from each incoming payload from the Arduino. The range is automatically determined by QuickSight, starting at 0, with auto-scaling axes.

Testing

Each feature of the system required specific testing methods, as are described in this section. Testing the hardware components was a matter of ensuring the sensors were generating reliable data and responding appropriately to changing environment conditions. The software components generally required a more involved testing process, which was a matter of ensuring that all possible conditions are appropriately handled. I generated many arbitrary MQTT payloads in JSON format with specific attributes to ensure the IoT data was being processed properly by the interworking Amazon Web Services.

Gather and transmit data from sensors

This feature involved a lot of incremental development, described above, and also extensive testing. I thoroughly tested the DHT11 circuit by moving the sensor to different environments, covering it with my hand, etc. to ensure the data was being updated and changed appropriately. The soil moisture sensor simply outputs analog data, and required some experimentation to determine what range of analog values corresponded to what range of percent soil moisture. I submerged the sensor completely in water and noted the range of values it output, then gathered the output for dry air, dry soil, and a mixture of half soil and half water. By doing this I was able to determine that fully saturated soil gave analog values

around 700, and about half saturated soil gave analog values around 300-400. This process was imperfect and subjective, so it does present an opportunity for some loss of precision when it comes to percent soil moisture, but through this calibration I determined that I would send a notification whenever the analog soil moisture value was below 300, as I reasoned this was close enough to the 40% lower threshold.

The second half of testing for this feature involved testing the code that connected the Arduino to WiFi and then to AWS. I learned through trial and error that the Arduino wouldn't connect to WPA2 networks, but was able to connect to WPA networks. Even though the system only required that the Arduino could publish a payload, I published and subscribed payloads from both the Arduino and AWS to ensure they could communicate in both directions for the sake of thorough testing and potential future feedback features.

Send alerts based on predetermined thresholds

To test that my Lambda function was triggered by the proper conditions and sent the appropriate message in the email, I used the built in testing features on AWS Lambda. I first subscribed my own email address to the SmartFarmAlert topic and confirmed the subscription to see the messages being published to the SNS topic by AWS Lambda. Figure 7 shows an example of a test configuration—which is a sample MQTT payload with a "tempF" value that is outside of the recommended temperature range for optimal plant growth—and the email notification that was sent in response to the test payload.

Figure 7a- Test Configuration for Lambda Function and Corresponding Notification
This is the example MQTT payload I created to test that the system would recognize the temperature in Fahrenheit was outside of the recommended range of values and send an email that indicated that.

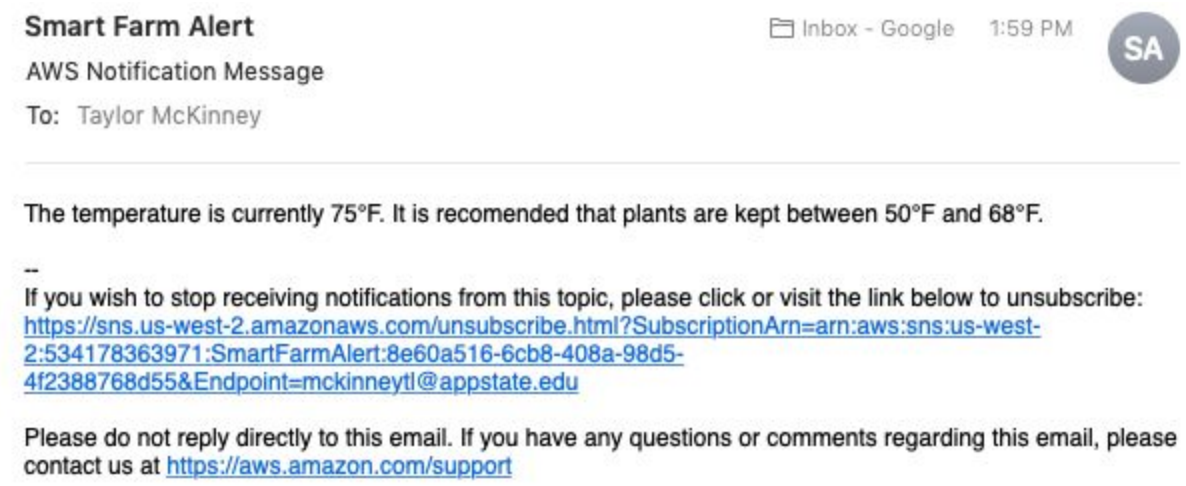
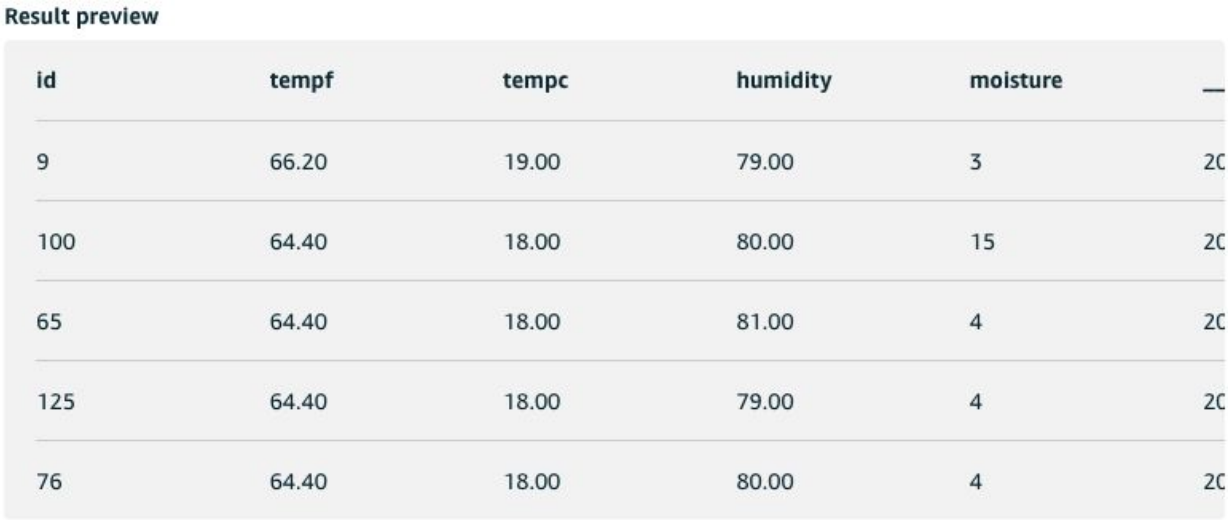


Figure 7b- Test Configuration for Lambda Function and Corresponding Notification
This is the email that was sent by the Lambda function when tested with the above payload.

Store historical data and Visualize data

Each AWS IoT Analytics component has built-in monitoring windows that display all connections, so the only way I could really test that the Channel and Pipeline worked was by checking that a new connection appeared whenever a new payload was published to the "arduino/outgoing" MQTT topic. The Datastore and Dataset were tested by simply ensuring that the payloads were then displayed in the result preview window of the Dataset tab of IoT Analytics as seen below in Figure 8. Testing the visualization feature simply required looking at the generated graphs and ensuring they represented the data in the correct format whenever the Dataset is updated.



id	tempf	tempc	humidity	moisture	__
9	66.20	19.00	79.00	3	20
100	64.40	18.00	80.00	15	20
65	64.40	18.00	81.00	4	20
125	64.40	18.00	79.00	4	20
76	64.40	18.00	80.00	4	20

Figure 8- Screenshot of Result Preview Window of the IoT Analytics Dataset Tab
This is a screenshot of the result preview of the dataset being sent to QuickSight for visualization at one point of testing with arbitrary data.

Results

The final results of the completed system in terms of how the system accomplished the goals originally set for each system-level feature, including the emphasis on low cost and ease of use of the final system. For more information of bonus features I was not able to complete, see the Future Work section below.

Table 3- Results by Feature

Feature	Result
Gather and transmit data from the sensors	The system is fully able to gather temperature, humidity, and soil moisture data from the Arduino circuit and transmit that data to AWS over the MQTT protocol. Because I have not addressed the actual storage on the farm, I haven't tested that the sensors can still collect data while not connected to the computer, because at this point it's both the source of the code and the power source. Other than the caveat of impracticality, the circuit works as a proof of concept. The entire circuit was purchased for under $40, and was the only cost incurred by the entire system.
Store historical data	The system loads the raw JSON data from every incoming payload to the MQTT topic "arduino/outgoing" into the IoT Analytics Datastore named smartFarm_datastore. That raw data is then processed, including converting the analog soil moisture data into a percentage, and stored in the IoT Analytics Dataset, smartFarm_dataset. Every single payload that I published to the topic throughout the testing process was stored in those locations, regardless of repeat data, missing attributes, etc. The system does not currently exceed the AWS free tier maximum usage, but there is a possibility in the future that the Datastore and Dataset may grow to a size that exceeds the storage allotted, which would then incur variable costs.
Send alerts based on predetermined thresholds	Using the AWS Lambda function notificationControl, I was able to implement this feature fully. Whenever a payload comes through IoT Core in the "arduino/outgoing" topic, the function is triggered. The Python 3.8 code then performs the logic to check the 'tempF', 'humidity', and 'moisture' attributes for undesirable growing conditions, and then sends an email notification with a short description of the problem to any email address with a confirmed subscription to the SNS topic called SmartFarmAlert. The final solution does require a small amount of setup due to the subscription and confirmation process, but once completed, the email notifications

	will continue to be automatically delivered to the farmer's email every time the function triggers a response from SNS. This solution also allows for multiple email addresses to receive the notifications, and the farmer or system administrator can cancel the subscription at any time.
Dashboard with simple visualization	All sensor data is processed by IoT Analytics and then visualized with AWS QuickSight. The visualization is composed of two simple line graphs, one showing the percent humidity and percent soil moisture and the other showing temperature in Fahrenheit for each payload that is published to the "arduino/outgoing" topic. The visualization is simple and clean, but could use fine tuning in future work. One disadvantage of the current solution is that in order to see the visualization, the farmer must create an AWS account and the system administrator must grant their account access to the QuickSight dashboard. Then the farmer must sign on to the AWS Online Console, navigate to QuickSight, and open the Smart Farm Dashboard, which is intended more for the developer than the end user. Regardless, the data is visualized in a clear and concise way, and QuickSight offers resources that could be used to export the graphs to a web application in the future.
Web application with more detailed visualization	Bonus feature that was not completed due to time constraints.
Flexibility in thresholds	Bonus feature that was not completed due to time constraints.
Package system in weather-proof container	Bonus feature that was not completed due to time constraints.

Conclusions and Future Work

The inspiration for this capstone project originally came about during my summer internship at Heartwood Farms, a sustainable farm local to Boone co-owned by professors of Appalachian State University and Wilkes Community College. They were expressing their interest in smart farming—a recent trend of using technology to reduce labor and increase precision in agriculture—but found that the current solutions were all either too expensive, complicated, or inaccessible for their small farm. This was the driving force behind this project, and their preferences and convenience was considered at each point of development of the system. The final result of this iteration of the project is a prototype of a system that may one day be used by Heartwood Farms for agriculture monitoring, but still requires future work to do so.

The initial goals of the project were to explore technologies that are increasingly popular in the industry, Arduino and Amazon Web Services, while developing this agriculture monitoring prototype. The final system consists of two sensors attached to an Arduino MKR WiFi 1010 that communicate with various Amazon Web Services through the MQTT Protocol. At predetermined time increments the hardware components of the system, the sensors and the Arduino, automatically gather temperature, humidity, and soil moisture data from the "crops", formats the data into JSON format, publishes the JSON payload to the "arduino/outgoing" MQTT topic. The software components, MQTT and all AWS resources, receive the incoming payload, trigger Python code that sends specified email notifications of undesirable growing conditions, store the raw data, process and query data to create a Dataset, and finally, use the Dataset to visualize the data into two simple line graphs in a unified dashboard.

Reflecting on the completed project in regards to the initial project goals, I consider the system to be a working prototype of a system that may one day be used for agriculture monitoring. There are still practicality issues that must be addressed in future work before the system can be classified as a completed product. I was able to accomplish all of the goals I prioritized as essential in the early stages of development, but did not get to any of the desired or stretch goals I had originally hoped to complete. A large factor in why I was unable to complete these stretch goals was my inexperience with the technologies used in the system prior to this project. The entire time I was following tutorials I could find online and troubleshooting problems that arose, I was learning by trial and error. Because of the large risk area of my knowledge gap, I adjusted the scope of the project early on from creating a fully functioning and fleshed out final product to simply creating a prototype of an IoT crop monitoring system.

This allowed me to set practicality issues aside once the Arduino circuit was functioning properly with the breadboard implementation in favor of focusing my attention on Amazon Web Services and

expanding my knowledge of the software services they offer. I initially intended to return to the practical issues of storing and powering the circuit on the farm, but ultimately ran out of time to revisit that feature in this iteration of the project. In the future, I plan on designing a water and weather resistant container that can store the circuit and a power source for the circuit. An initial idea is to use PVC Pipe or a similar material, but full development of that will require more consideration beyond the constraints of the semester.

Another feature that I plan to expand on in the future is the development of a fully fleshed out user interface. The current solution requires more action on the farmer's part to view the visualization than I would prefer, and I'd like to create a web application that is easier for them to access. This web application would also allow the farmer to better interact with the system by allowing them to change the thresholds and conditions that trigger alerts as well as query the data in various ways, and also make the visualization more interesting to the eye. AWS QuickSight is used now to generate the graphs and serve as proof of concept that the system can visualize the data for the farmer, but may be replaced by some other software component in the future depending on the design of the web application.

References

1. What is Precision Agriculture?
2. Heartwood Farms ltd
3. What is internet of things (IoT)?
4. Arduino MKR 1010
5. DHT11 on eBay
6. Funduino Moisture Sensor on Amazon
7. AWS Documentation
8. Soil Moisture Guide for Flowers, Plants, and Vegetables
9. Greenhouse Climate Measurements Ensure Optimal Plant Growth
10. Securely Connecting an Arduino MKR WiFi 1010 to AWS IoT Core
11. MQTT FAQ
12. Introducing JSON
13. How to Set Up the DHT11 Humidity Sensor
14. Complete Guide to Use Soil Moisture Sensor
15. WiFiNINA Library
16. ArduinoBearSSL
17. ArduinoECCX08
18. ArduinoMqttClient
19. EduIntro Library
20. Smart Farm Driver on GitHub

Appendix

A1. Arduino Code Necessary to Read Sensor Data

```
#include <EduIntro.h>

DHT11 dht11(D6);
float tempF;
float tempC;
float humidity;

int id = 0;
int moisture = 0;
int soilPin = A0;
int soilPower = 7;



void setup() {
  pinMode(soilPower, OUTPUT);
  digitalWrite(soilPower, LOW);
}
```

```
void loop() {
 dht11.update();
 tempC = dht11.readCelsius();
 tempF = dht11.readFahrenheit();
 humidity = dht11.readHumidity();
 moisture = readSoil();
delay(5000);//wait 5 seconds to get new values
}

int readSoil()
{
   digitalWrite(soilPower, HIGH);//turn D7 "On"
   delay(10);//wait 10 milliseconds
   moisture = analogRead(soilPin);//Read the SIG
value from sensor
   digitalWrite(soilPower, LOW);//turn D7 "Off"
   return moisture;//send current moisture value
}
```

A2. Arduino Code Necessary to Connect WiFi and MQTT

```
#include <WiFiNINA.h>
#include <ArduinoMqttClient.h>
#include <ArduinoECCX08.h>
#include <ArduinoBearSSL.h>

const char ssid[]        = "asu-visitor";
const char pass[]        = "";
const char broker[]      = [omitted];
const char certificate[] = [omitted];
WiFiClient    wifiClient;
BearSSLClient sslClient(wifiClient);
MqttClient    mqttClient(sslClient);

void setup() {
  Serial.begin(115200);
  while (!Serial);
  if (!ECCX08.begin()) {
    Serial.println("No ECCX08 present!");
    while (1);
  }
  ArduinoBearSSL.onGetTime(getTime);
  sslClient.setEccSlot(0, certificate);
}
```

```
void loop() {
  if (WiFi.status() != WL_CONNECTED)
    {   connectWiFi();  }
  if (!mqttClient.connected())
  {   connectMQTT(); }
  mqttClient.poll();
}

void connectWiFi() {
  Serial.print("Attempting to connect to SSID: ");
  Serial.print(ssid);
while (WiFi.begin(ssid, pass) !=
WL_CONNECTED) {  // failed, retry
    Serial.print(".");
    delay(5000);
  }
  Serial.println("You're connected to the network");
}

unsigned long getTime()
{
      return WiFi.getTime();
```

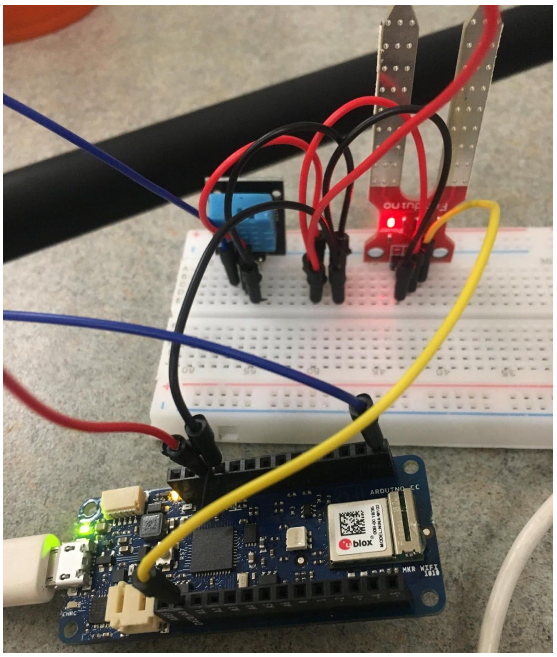
```
}

void connectMQTT()
{
 Serial.print("Attempting to MQTT broker: ");
 Serial.print(broker);
 while (!mqttClient.connect(broker, 8883)) {
  // failed, retry
  Serial.print(".");
  delay(5000);
 }
 Serial.println();
 Serial.println("You're connected to the MQTT broker");
 Serial.println();
 mqttClient.subscribe("arduino/incoming");
}
```

A3. Arduino Code to Publish to MQTT Topic

```
void publishMessage() {
 Serial.println("Publishing message");
 id++;
 mqttClient.beginMessage("arduino/outgoing");
 mqttClient.print("{\"id\": \"");
 mqttClient.print(id);
 mqttClient.print(" \", \n ");
 mqttClient.print("\"tempF\": \"");
 mqttClient.print(tempF);
 mqttClient.print( "\", \n");
 mqttClient.print("\"tempC\": \"");
 mqttClient.print(tempC);
 mqttClient.print( "\", \n");
 mqttClient.print("\"humidity\": \"");
 mqttClient.print(humidity);
 mqttClient.print( "\", \n");
 mqttClient.print("\"moisture\": \"");
 mqttClient.print(moisture);
 mqttClient.print("\"}");
 mqttClient.endMessage();
}
```

A4. Final Arduino Circuit



A5. Python Code for Sending Notifications Based on Thresholds

lambda_function

```python
import json
import boto3

def lambda_handler(event, context):
    moist = event.get("moisture")
    hum = event.get("humidity")
    tempF = event.get("tempF")
    sns = boto3.client('sns')
    if int(moist) < 300 :
        response = sns.publish(
        TopicArn='arn:aws:sns:us-west-2:534178363971:SmartFarmAlert',
        Message='Your plants are thirsty! It is recomended that you water them as soon as possible!'
      )
    elif (int(tempF) < 50 or int(tempF) > 68):
        response = sns.publish(
        TopicArn='arn:aws:sns:us-west-2:534178363971:SmartFarmAlert',
        Message='The temperature is currently ' + tempF
           + '°F. It is recomended that plants are kept between 50°F and 68°F.'
        )
    elif (int(hum) > 70 or int(hum) < 50):
        response = sns.publish(
        TopicArn='arn:aws:sns:us-west-2:534178363971:SmartFarmAlert',
        Message='The relative humidity is currently ' + hum
            + '%. It is recomended that plants are kept between 50% and 70% RH.',
        )

    else:
        response = ''

    return response

```

1:1 Python Spaces: 4