smart cities

MDPI

Article

A Cloud-Based Flood Monitoring and Alerting System

Loza Asmare [1], Abdullah Mahmood [1], Glen Mitchell [1], Kwadwo Tenkorang [2], Conor Todd [1], and Jonathan L. Goodall [1]

[1] Department of Engineering Systems and Environment, University of Virginia, Charlottesville, VA 22904; lda5fv@virginia.edu, aam2vp@virginia.edu, gm2rg@virginia.edu, cht6rh@virginia.edu, jlg7h@virginia.edu
[2] Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA 22904; knt4xx@virginia.edu
* Correspondence: jlg7h@virginia.edu

Abstract: Flood warning systems can help to mitigate potential damages and loss of life in flood prone regions. The objective of this research was to create a real-time and cloud-based flood warning system leveraging Internet of Things (IoT) sensors. The system incorporates scalable, autonomous, and inexpensive features that allow stakeholders to understand real-time conditions and issue alerts to residents. Built in Amazon Web Services (AWS), the system leverages serverless technology for data ingestion, a relational database for data management, and a graphical user interface (GUI) for data visualizations and alerts. An Application Programming Interface (API) allows for more advanced analysis in environments like Jupyter notebooks. The system can ingest data from IoT sensors deployed through The Things Network (TTN) as well as from flood prediction models to provide forecasting capabilities. A proof-of-concept demonstration of the system was built for a flood prone urban watershed in Albemarle County as a case study.

Keywords: Internet of Things, Environmental Monitoring, Environmental Modeling, LoRa, AWS, Data Management, Cloud Computing

Citation: Asmare, L.; Mahmood, A.; Mitchell, G.; Tenkorang, K.; Todd, C.; Goodall, J. L. Title. *Smart Cities* **2021**, *4*, Firstpage–Lastpage. https://doi.org/10.3390/xxxxx

Academic Editor: Firstname Lastname

Received: date
Accepted: date
Published: date

1. Introduction

Flooding is one of the most frequently occurring types of natural disaster in the United States, and climate change is causing extreme precipitation events to become more intense and frequent [1]. The effects of flooding causes $3.75 worth of damages annually in the U.S., and in Charlottesville, 1,565 houses will be at risk of flooding events within 30 years [2]. Predicting precise precipitation amounts is difficult in remote areas due to the high variability and complexity of storm systems [3], so ground watershed modeling and real time alert systems are imperative to mitigate flood damages.

The system built by the project team will provide stakeholders with the ability to monitor and understand real-time flood conditions using Internet of Things (IoT) sensing and cloud-based monitoring system. The alerting capability of the system can benefit community residents by forecasting flooded areas ahead of time and potentially reducing damage, loss of life, and inconvenience from floods. The approach and system designed in this project can be utilized by any communities that want to leverage IoT technology to improve flooding mitigation infrastructure.

2. Problem Statement and Objectives

With the increase of weather unpredictability and flooding, it is vital that communities launch flood mitigation initiatives for the safety and quality of life of their residents. In order to create a real-time sensing and alert system for the city of Charlottesville, environmental data needs to be collected from various locations around the city and uploaded into the cloud as quickly as possible. For preemptive flood management strategies, data

needs to be collected about existing infrastructure and land features to model stormwater flow and forecast future flood conditions.

The project team's goal was to lay a solid foundation to the backend and modeling of flooding events. Basic features of the monitoring system include data collection, visualization, alert creation, data-driven environmental forecasting, and physics-based stormwater flow simulation.

3. Methodology

3.1 System Architecture Overview

Currently, several IoT measurement sensors are deployed around Charlottesville, VA, collecting and reporting various types of environmental data to The Things Network. Through AWS Lambda, these measurements are then queried, transformed, and uploaded to different parts of the flood monitoring and alert system through cloud-based services. The original incoming sensor record data is stored separately for archival and backup purposes in an AWS S3 bucket while the transformed sensor data is entered into a MySQL database to allow for fast querying. The data is then queried for visualization, monitoring, and alerts through a graphical user interface (GUI) tool, Grafana. The data can also be queried by clients through an application programming interface (API) for use in client software systems for purposes such as analysis and modeling in Jupyter notebooks and SWMM software. A website was created to allow clients to access the GUIs, APIs, and manually modify sensor configurations. The system architecture diagram can be seen in Figure 1.

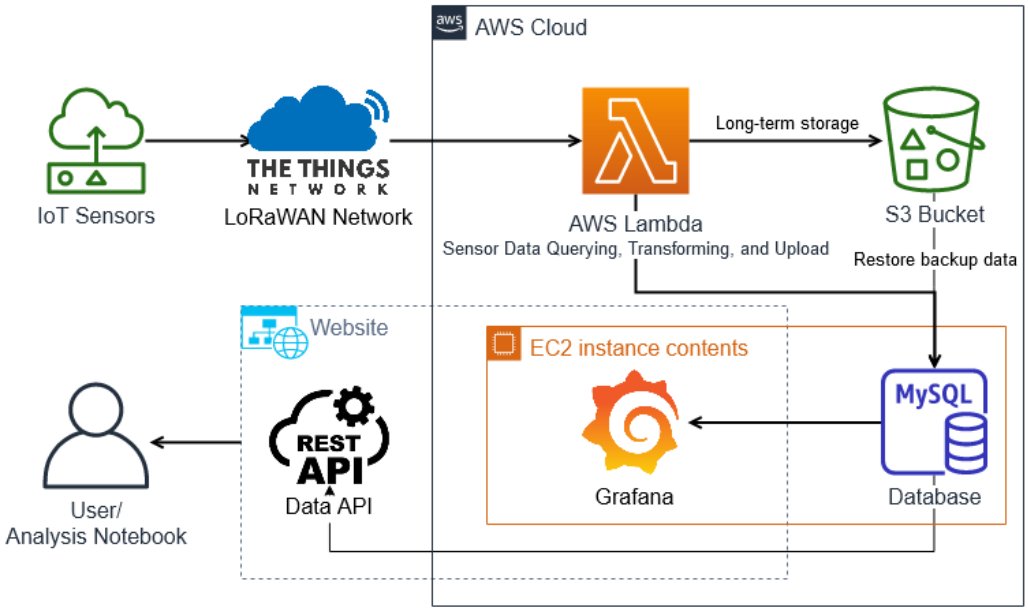

Figure 1. System architecture diagram.

3.2 Design Requirements

This cloud-based system requires several different components to operate. First, the deployed IoT sensors must maintain connection with The Things Network (TTN) in order to report measured environmental data. The data must then be processed and stored in different parts of the system. To make the system scalable and low-maintenance, services and tools used must be hosted through a single cloud service provider. In this system's case, it must be hosted by provisioning services through Amazon Web Services (AWS). Next, in order for this system to be sustainable and used based on different client cost constraints, it must operate at minimum cost and have efficient resource consumption.

The system must also be intuitive to deploy, use, maintain, and modify by both technical and non-technical clients.

4. Results and Discussion

4.1 System Components

4.1.1 Data Access and Loading

The system utilizes commercial sensors as opposed to creating a new sensor in order to place the focus on the data gathering, storage, and analysis system rather than the sensor hardware and software alone. One of the primary reasons for this decision was to enable the team to focus on developing a more robust software system that can operate with any sensor hardware compatible with The Things Network (TTN). The TTN platform sets up infrastructure for communication to and from the sensors. In addition to the high durability and reliability of the commercial sensors, they are also built to communicate over LoRaWAN®, a low power, wide area network. The sensors were connected to TTN to enable cost effective interfacing and to utilize the platform's free seven day storage via TTN's data integration service. To connect data from TTN to the Amazon Web Service (AWS) stack, an HTTP API was written to retrieve data for a particular sensor and place it in long term storage.

The bulk of the cloud platform is built using AWS tools. For regions impacted by flooding, high availability of the computing backend is imperative due to the need for quick analysis of incoming weather and environmental data. AWS offers high availability, which includes regional failovers in case a data center is taken offline. Deploying and redeploying resources on AWS can also be quickly automated using AWS CloudFormation, a tool used to provision specified resources (such as Lambda, EC2, RDS, etc.) through a provided script in the GitHub repository. The code written for the backend of the cloud-based system can be found at https://github.com/uva-hydroinformatics/FloodWarningSystems_20-21.

AWS provides a serverless computing platform, Lambda. The underlying infrastructure of Lambda is maintained by AWS, which means the client only has to worry about choosing the correct runtime environment to deploy their code. Using Lambda, the sensors are queried for collected data payloads at specified intervals. The payload is then parsed and the data is transformed to only include information pertinent to the project. The original data payload is first uploaded to S3 for long-term storage and then the transformed data is uploaded into the MySQL database. After the Lambda finishes uploading the transformed data, it automatically shuts off, allowing the client to pay only for the computing time and memory resources used rather than provisioning a continuously running machine (e.g. EC2). Lambda was chosen for this project due to ease of scalability with future added devices, monitoring, high availability, and resource efficiency. For instance, if a new device is added to the network, the existing Lambda can quickly start communicating with the device. Should multiple devices need to report data in overlapping intervals, the same Lambda function can run in parallel of up to 1,000 instances if needed.

The Lambda function for this project requires modification from the default settings. It is important to note that the current code running in the Lambda function requires a Lambda layer. Lambda layers are user-provided libraries that AWS itself does not provide and maintain. For this project, a Lambda layer was created for the runtime environment of Python 3.7 including the libraries of pandas, sshtunnel, PyMySQL, and Requests. Pandas is used to quickly transform and manipulate data for users when querying from the database. Sshtunnel and PyMySQL are used to maintain a connection to the MySQL database. The Requests library is used to send HTTP requests to The Things Network. Other configurations for the Lambda function include setting the allocated memory to 160 MB (determined by AWS Cost Optimizer), timeout limit of 3 minutes, and being triggered to run once every hour.

AWS Simple Storage Solution (S3) is a cost-effective way to store data for an extended period of time. Data collected by the measurement sensors are uploaded as individual readings in S3 for long-term storage. These readings can be used to repopulate the database in case of a database failure or migration and can be done using the library created for this project. The library also allows the user to download a copy of the readings from S3 to a local machine. All readings in S3 are currently stored as the AWS Standard tier for regular access for this project.

To host MySQL 5.7 and Grafana, an Amazon Elastic Compute Cloud (Amazon EC2) instance was provisioned. Amazon EC2 allows for a continuous computing platform on the cloud, which allows access to the database and Grafana when needed. The project uses a t3.micro, which fits the needs of this capstone project by keeping costs to a minimum while still maintaining reliable performance for the relatively low number of sensors currently in the system. A larger instance could be used for an expanded network of devices for more data storage or for a use case requiring quicker response times. For this project, MySQL and Grafana were hosted on the same EC2 instance. This was done for simplicity and was sufficient for the proof-of-concept implementation. For a production level implementation, it may be worthwhile to provision an EC2 solely for Grafana and a second EC2 to serve the MySQL database. Or, it may be worthwhile to use the AWS RDS service for their database and scale the RDS database based on client requirements for maintainability and access speed. RDS comes at a higher cost, but provides built in scalability as data volumes and users grow.

4.1.2 Relational Database Design and Implementation

In order to improve the scalability of this system, an entity relationship diagram (ERD) was created to normalize the sensor readings as shown in Figure 2. The ERD is centered around the values entity, which stores the value of individual data points along with the time of data collection. The devices entity stores the unique identifier, type, and the current battery of the device the data was collected from. Similarly, the locations entity contains data on the latitude and longitude for each location that data is collected from, along with a unique identifier for each location. For each value in the values table, the variables entity stores the data points' unique variable name and a description of the variable. The values entity has a one-to-many relationship with the three other entities, meaning that each value data point can only have one device, variable, and location, while the remaining entities can have many values for each data point in their tables. This ERD was developed by advancing an approach by previous related research [4]. This design of the database allows for easy further advancement and change as additional devices and variables can be easily incorporated.

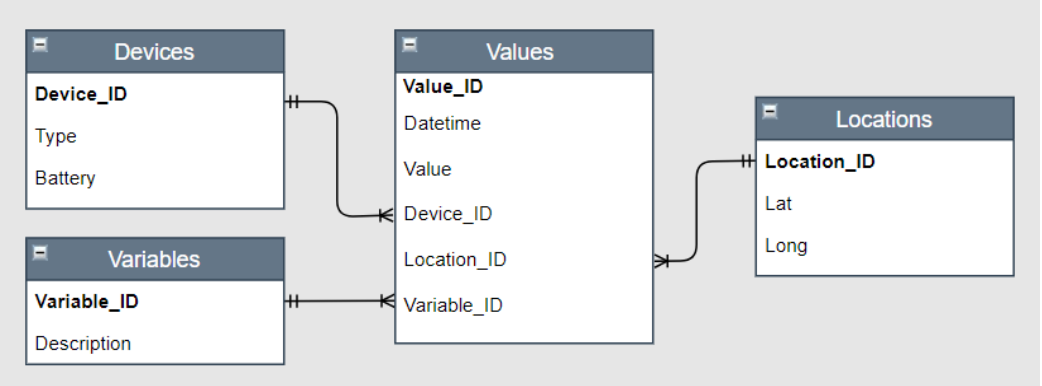

Figure 2. Entity relationship diagram for database design.

4.1.3 Graphical User Interface

This system allows users to visualize and monitor real-time data through Grafana, an open-source analytics platform for querying, visualizing, and alerting on data metrics. Grafana was selected as the program to visualize incoming data due to its dynamic dashboards, built-in alerting capabilities, and its specialization in time series data.

Grafana was deployed on the cloud through Bitnami, a library of packaged applications that allows for easy deployment of a pre-configured Grafana stack on AWS. A connection was then made to the MySQL database in Grafana to access the data for visualization. Dashboards of each monitoring station were created to display relevant information for clients. Figure 3 depicts a decision support dashboard of the water depth monitoring station. This dashboard includes a graph of the water depth over time, statistics on the water depth values for the set time range, a water level gauge of the current depth, a map of the sensor station location, and a gauge of the sensor battery level. The water depth graph and gauge allow stakeholders to view the current and past water levels in relation to a threshold of 0.4m to signify flooding. Grafana's built-in alert system has the ability to send alert notifications if the incoming data triggers a set alert rule. The water depth dashboard has alert rules set which send a notification on Slack if the 0.4m threshold is met.

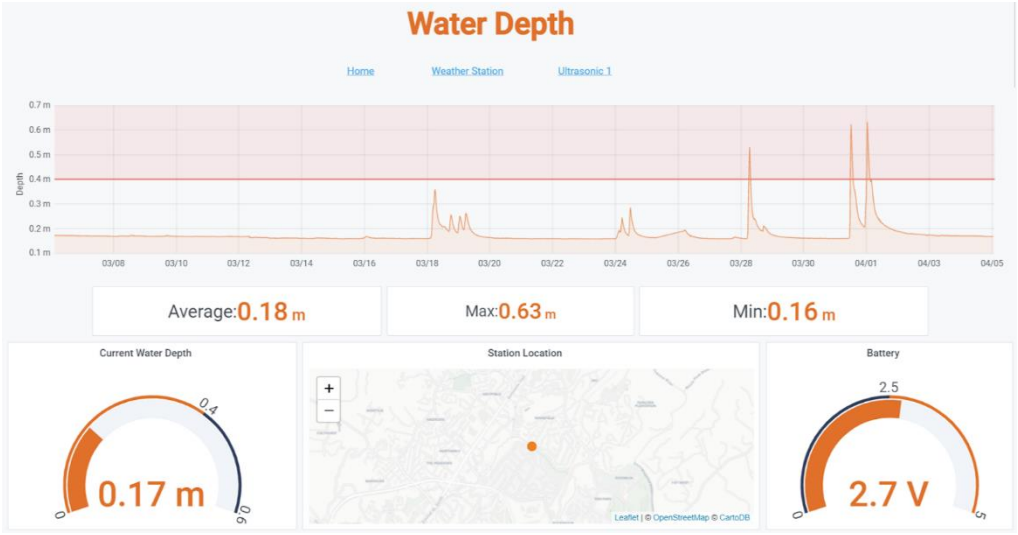

Figure 3. Grafana decision support dashboard of water depth monitoring station.

4.1.4 Application Programming Interface

For users, the website serves as a key interface to quickly perform important functions such as quickly creating and downloading datasets for sensors, adding or removing sensors from the system, manually configuring sensor measurement frequencies, and providing access to Grafana for visualizing, editing, and creating alerts through the dashboards. Access to Grafana is done by redirecting a page on the Heroku website to the endpoint of the EC2 hosting Grafana. The website developed for this technical project used the python FastAPI framework and is currently hosted using Heroku.

To download datasets from the website, OpenAPI is used for users to query devices connected to the network. To download data, the user will be required to input the device ID of the sensor they need data from and need to specify either a “last” or “start_date” parameter (“end_date” optional if using “start_date”). Using the “last” parameter, users can retrieve data collected by the sensors from the time of querying to the time specified. Using the “start_date” parameter, users can specify the beginning of the time range of the dataset to download. By default, if there is no “end_date” parameter specified as the end of the time range of the dataset, the time of querying will be used. From the OpenAPI page, the user will be able to quickly download datasets for the sensor of interest.

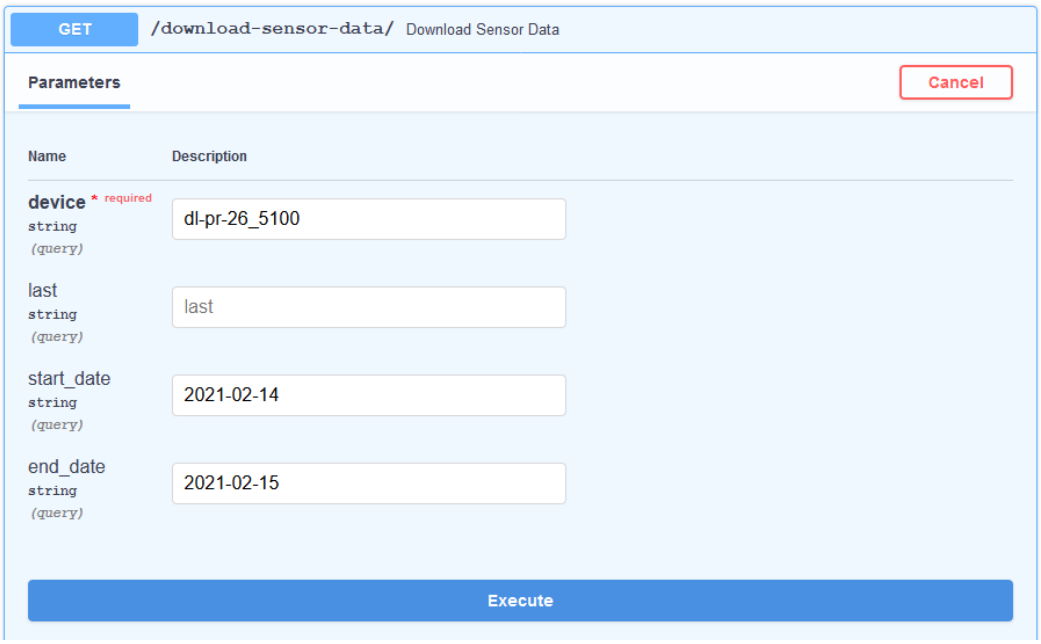

Figure 4. Example of parameters for download API.

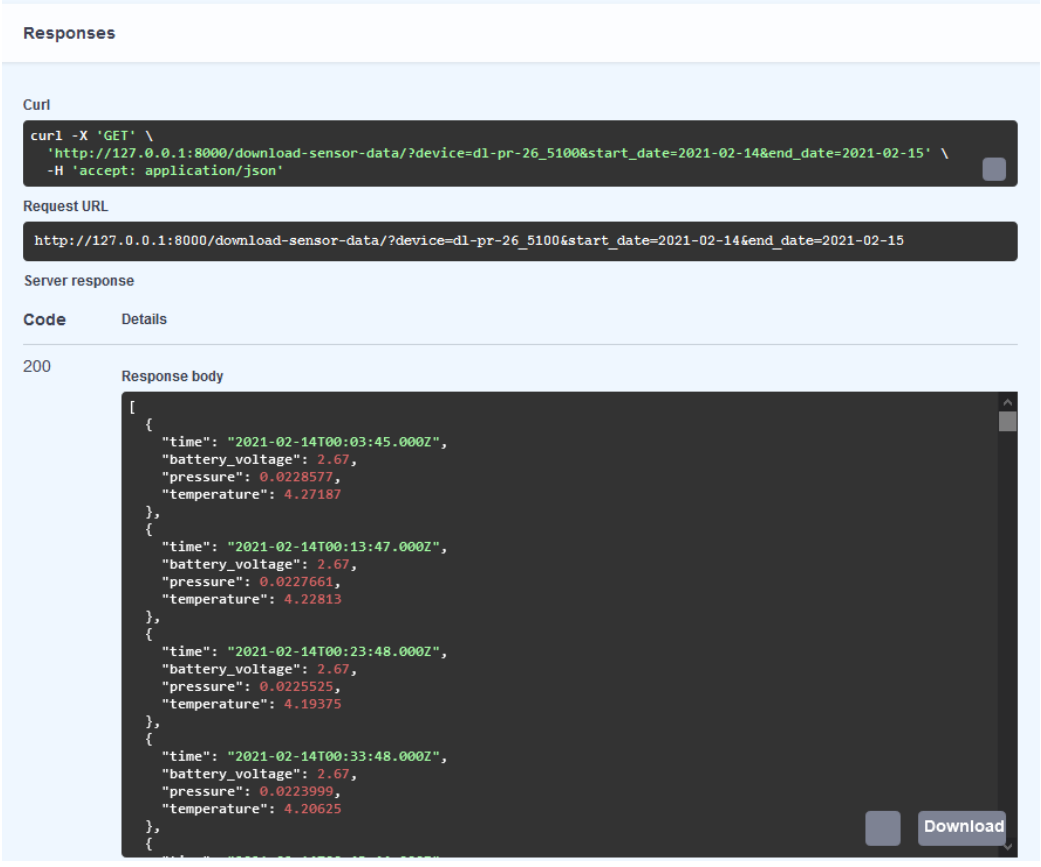

Figure 5. Results from API using example parameters.

Users are also able to configure new or existing sensors in the system, as seen in Figure 6. To add a new sensor to the system, that sensor type needs to first be added to the system along with the measured variables and their descriptions for the Variables table, if it does not already exist for that sensor type. This can be done with the add-sensor-type API by providing a list of variables and their descriptions in a JSON format.

Individual sensors can also be added and removed from the system by providing information about their device ID, sensor type, and coordinates of its location. To manually edit sensor measurement frequency, the user provides the device ID and the specified measurement frequency in terms of seconds. Decent Lab sensors can have a frequency limit of up to one measurement per minute.

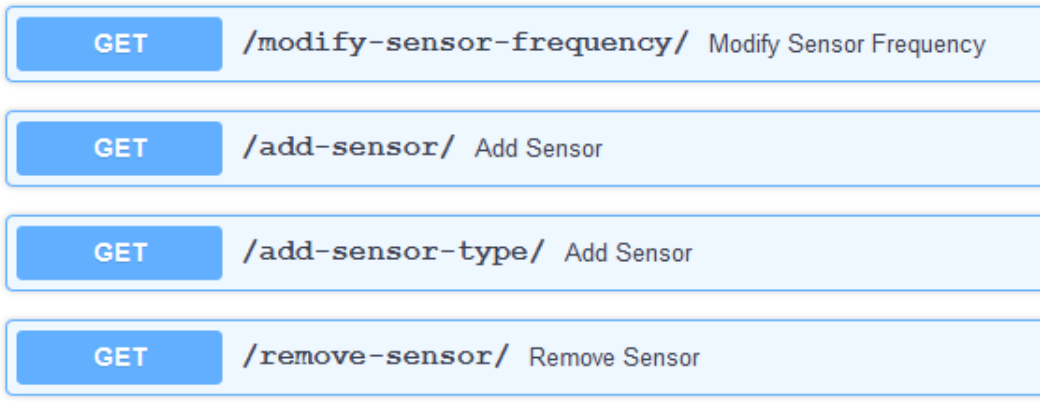

Figure 6. Other available API queries.

4.1.5 Client Software Applications

In order to give clients the ability to perform their own research and analysis on data that the sensors record without technical knowledge about the system architecture, a live visualization interface with customizable parameters was developed in a Google Colab notebook. The rationale behind choosing Colab as the software to develop this interface on include the fact that Colab employs python as it's language, which is consistent with the other coding elements of this project, the ease of access and sharing that Google Drive consists of, and that Colab notebooks execute code on GPUs and TPUs on Google's cloud servers, saving the need to download a python IDE or use personal machine CPU power. From the raw data format that the S3 download provides, it is run through a JSON combiner function that is later converted into a pandas dataframe. Through the use of these dataframes, the readings of data between sensors are combined on the nearest timestamp, which allows for overlaid visualizations of fields that were recorded at the same time on different sensors. These visualizations were developed through the matplotlib that is integrated with pandas.

4.2 Discussion of Alternative System Components and Potential System Enhancements

4.2.1 Alternative for AWS System Architecture

The first two versions of the databases created for the project hosted the MySQL database using Amazon Aurora and then Amazon Relational Database Service (RDS). For the project needs, Aurora and RDS costs presented a constraint, which is currently why a single EC2 is hosting both MySQL and Grafana. The current method provides the lowest cost of $100/year while still maintaining moderate performance for the current sensors deployed in the system. However, the database hosted this way may require maintenance or service by the user, along with providing no regional failover. In the event an AWS region experiences an outage, regional failover allows a copy of the database hosted in a separate region to quickly take over operations. In the case a client requires minimal maintenance, it is recommended to provision a t3.micro EC2 instance to host the website and Grafana while also provisioning a db.t3.micro EC2 instance through RDS to host the MySQL database. This configuration would cost $260/year for more reliable database performance. Should the client require seamless regional failover and high database performance, it is recommended to provision a t3.small instance using Aurora. This configuration costs $860/year. The latter two configurations include the $100/year cost for hosting

the website and Grafana on t3.micro instance. Memory storage calculations and associated costs with S3 and the database configurations are provided in the following tables (1-6):

Table 1. S3 storage costs calculations for the first year.

Number of Devices		Q1	Q2	Q3	Q4	Total
1	Storage (GB)	0.013679266	0.034198165	0.054717064	0.075235963	0.177830458
	Cost (USD)	$0.0723	$0.0728	$0.0729	$0.0731	$0.2911
4	Storage (GB)	0.054717064	0.13679266	0.218868256	0.300943851	0.711321831
	Cost (USD)	$0.2893	$0.2911	$0.2918	$0.2924	$1.1646
25	Storage (GB)	0.341981649	0.854954123	1.367926598	1.880899072	4.445761442
	Cost (USD)	$1.8079	$1.8197	$1.8236	$1.8275	$7.2787
50	Storage (GB)	0.683963299	1.709908247	2.735853195	3.761798143	8.891522884
	Cost (USD)	$3.6157	$3.6393	$3.6472	$3.6551	$14.5573

Table 2. Database storage (MB) requirements over time per device type (4800 readings per month)

Device	1 month	1 year	5 months
Atmospheric	7.131958008	85.58349609	427.9174805
Pressure	1.501464844	18.01757813	90.08789063
Ultrasonic	1.501464844	18.01757813	90.08789063
Current Config (CC)	11.63635254	139.6362305	698.1811523
Average (CC)	2.909088135	34.90905762	174.5452881

Table 3. Database storage (GB) requirements over time based on number of devices in current configuration

Number of Devices	1 month	1 year	5 months
1	0.002840906	0.034090877	0.170454383
5	0.014204532	0.170454383	0.852271914
25	0.07102266	0.852271914	4.261359572
50	0.142045319	1.704543829	8.522719145
100	0.284090638	3.409087658	17.04543829

Table 4. Database cost on single EC2 instance (t3.micro)

Number of Devices	Storage Requirement (GB)	Cost/Month (USD)	Cost/Year (USD)
1	5	$8.09	$97.10
4	5	$8.09	$97.10
25	5	$8.09	$97.10
50	10	$8.59	$103.10
100	20	$9.59	$115.10
Every 5 devices	1	$0.10	$1.20

Table 5. Database cost on separate RDS EC2 instance (db.t3.micro)

Number of Devices	Storage Requirement (GB)	Cost/Month (USD)	Cost/Year (USD)
1	5	$12.99	$155.82
4	5	$12.99	$155.82
25	5	$12.99	$155.82
50	10	$13.56	$162.72
100	20	$14.71	$176.52
Every 5 devices	1	$0.12	$1.38

Table 6. Database cost on Aurora (t3.small)[1]

Number of Devices	Storage Requirement (GB)	Cost/Month (USD)	Cost/Year (USD)
1	5	$18.60	$736.92
4	5	$18.60	$736.92
25	5	$18.60	$736.92
50	10	$24.60	$742.92
100	20	$42.96	$761.28
Every 5 devices[2]	1	$0.11	$1.31

[1] After 50 devices, IOPS needs to have the input increased to handle the average measurement writing load to S3. The cost also includes running 2 EC2 instances by default for regional failover.
[2] This cost is based of extra storage costs

The above calculations were done based on the current Decent Lab sensor configuration of the system and a projected 5-year use. The default measurement frequency for the system is 1 measurement every 10 minutes, averaging 4380 readings per month. To account for temporary measurement frequency increases during storm events, calculations instead used a figure of 4800 readings per month. Every write request to S3 costs $0.000005, coming out to $0.024 per month. Sensors currently in use are 1 eleven parameter weather station (DL-ATM41), 1 pressure/liquid level and temperature sensor (DL-PR26), and 2 ultrasonic distance/level sensors (DL-MBX). The average reading size of these 4 sensors is 510 bytes, with the weather station containing more measurements per reading than the other 2 sensor types. It is important to note that, when in the database, the weather station requires almost 5 times as much storage capacity as either of the other 2 sensors. Since the current system is based on these 4 sensors, AWS storage configurations may need to be readjusted based on the chosen sensors for the client's system.

Overall yearly system costs can also be lowered by configuring S3 and EC2 instance provisioning and by using the AWS Cost Optimizer tool. For S3, if the backup data will not be frequently accessed, it is recommended to change the access tiers of the data. For this project, the data is stored under Standard tier, which costs [PROVIDE COST/GB HERE]. In future iterations of the system, it is recommended to use Intelligent-Tiering [COST/GB HERE], Standard-Infrequent Access [COST/GB HERE], One Zone-Infrequent Access [COST/GB HERE], or even Glacier tiers [COST/GB HERE]. For the Infrequently Accessed and Glacier tiers, there is a retrieval fee for every gigabyte retrieved. Infrequently Accessed will allow for millisecond latency to the user when requesting data, whereas with Glacier it can take minutes or hours. Deleting data from non-standard S3 tiers before their minimum storage durations will charge the client for the respective minimum storage durations. Infrequently Accessed and Glacier tiers also have a minimum capacity charge per object, so it is recommended to combine individual readings into larger datasets (i.e. monthly readings per sensor) to store as one file in these tiers. To reduce costs of EC2 instance provisioning (including for RDS and Aurora), AWS allows for reserving instances in 1 and 3 year increments instead of using on-demand instances, bringing costs down by up to 38%. The costs calculated in this paper are using the current configuration of the system which uses on-demand EC2 instances.

One of the original goals of the project was to create the backend using only AWS. However, there were some limitations when creating the sensor data download Application Programming Interface (API). Originally, the sensor data download API was created using AWS API Gateway and AWS Lambda. Through an endpoint provided by API Gateway, a client request gets passed to the Lambda to retrieve datasets from the database, which would be transformed for use before being sent back to the client. The first limitation the project ran into is the 30 second timeout on API Gateway requests. After the retrieval code was changed to run faster, there was a second limitation through Lambda, which has a payload limit of 6 MB. For large datasets (e.g. 1 month of data from the weather sensor), the Lambda is not able to send the client their requested dataset. Due to

these technical limitations, the project team moved forward with using FastAPI on the Heroku-hosted website to create an API to fit project needs. The project team also used Heroku to deploy the website over hosting on an EC2 due to cost constraints and ease of deployment. In the future, it would be recommended to host the website on the same EC2 instance as Grafana so that the cloud backend can all be deployed using CloudFormation.

Security for a future implementation of this cloud-based system is also a major concern for clients. When deploying the cloud-based system, an AWS organization will need to be created by the client with trusted users to manage AWS Identity and Access Management (IAM) roles and policies. Instead of using an AWS organization, the students in the project used separate AWS accounts to create and manage Lambda, S3, and the database based on who focused on that part of the system. Due to this, the Lambda function currently requires the use of the sshtunnel python library and an EC2 instance key provided by another AWS account to communicate with the database. By creating a consolidated AWS organization, sshtunnel and the EC2 instance keys will not be needed by correctly configuring IAM roles and policies attached to the different components of the system, allowing the services to talk to each other without user-implemented SSH protocols. Besides making the overall system more secure, Lambda and API resource usage may also improve by not having to establish an SSH connection every time sensor readings need to be uploaded to or downloaded from the MySQL database. Proper security groups will also need to be configured for intended user access through the use of AWS Virtual Private Cloud (VPC).

Deploying a website was also necessary in order for users to download sensor data. Originally, the download data API utilized AWS Lambda and AWS API Gateway, as both are serverless architectures provided and maintained by AWS. The original API developed using AWS worked, however, there were limitations. The first limitation is that API Gateway has a timeout on requests that take 30 seconds or longer. The second limitation is that Lambda has a maximum payload size of 6 MB. When downloading large datasets, requests may take longer than 30 seconds or have a payload size larger than 6 MB. By deploying a website and using the FastAPI framework to create an API endpoint, these limitations no longer apply since the new API does not use Lambda or API Gateway.

4.2.2 Alternatives for Graphical User Interface

Providing stakeholders access to easily understandable information in a clear and efficient manner is paramount when working with large amounts of time series data. In this project, three data visualization platforms, Grafana, QuickSight, and SageMaker, were compared to find the best tool to effectively communicate information. Based on cost, visualization, analysis, and alerting capabilities, QuickSight was initially determined as the platform that met these metrics. However, after creating a QuickSight account and working with the platform, it was discovered that it does not support embedding visualizations in websites without assigning each user with permissions to view. It was thus determined that QuickSight was not a suitable tool as it did not meet the needs of the project. After conducting more research on data visualization platforms, it was decided that Grafana would be the best tool for this project due to its ability to easily share and embed visualizations. Grafana allows for the creation of snapshots of dashboards which can then be used to share interactive dashboards publicly through snapshot links. Additionally, Grafana is designed for time series data and allows for alerts to be sent out through many alert notifiers such as text message, email, and Slack.

4.2.3 Opportunities for Forecasting and Advanced Analytics

The creation of this system would allow for the production of real time forecasting models in the areas of interest. Generating time series forecasts on this data was hindered by the non-normality of the data. Developing the models over a longer period of time, incorporating each season could eliminate the randomness of precipitation effects and

create accurate forecasts. One potential data interest that could benefit the forecasting models would be smaller sampling intervals, as the wider variation of water depth between collection intervals cannot be explained by statistical algorithms.

5. Conclusion

Eventually, simulations within SWMM can be run using real time flow data instead of periodically measured flow. This narrows the precision of the model and will provide observations of how the model responds to normal, pre-storm flow conditions. Fine tuning the model's parameters to match real time flow will improve the predictability for large storm events. The plan view of the SWMM display with subcatchment and conduit connections is shown in Figure 7.

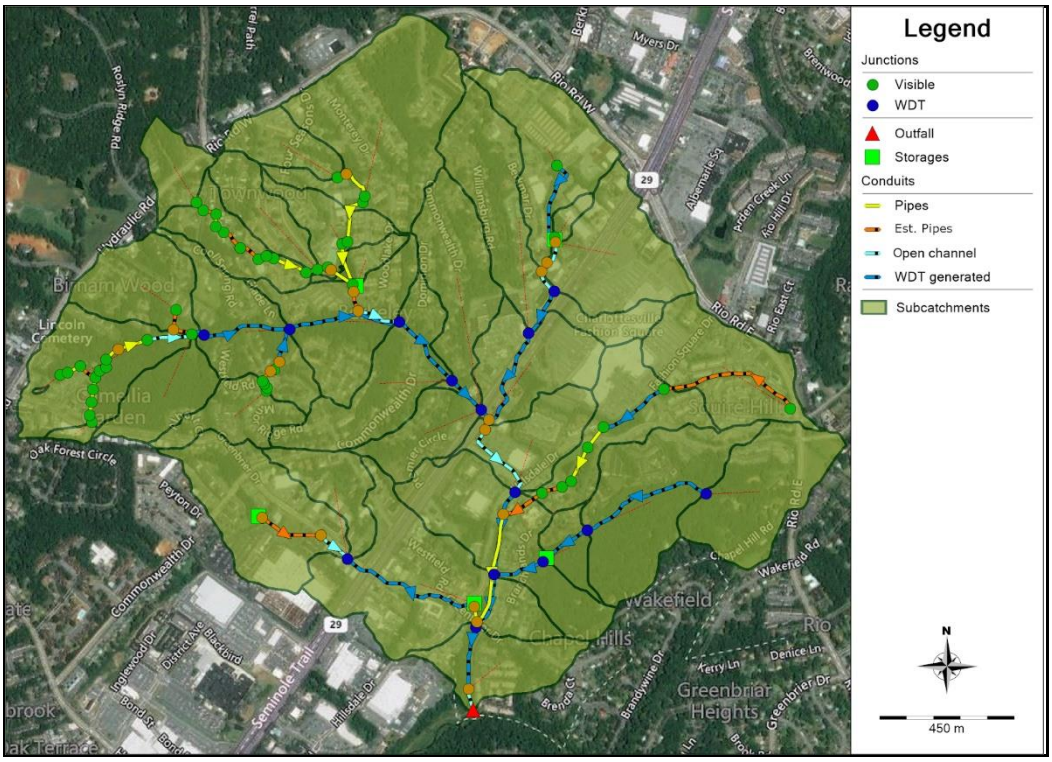

Figure 7. SWMM Subcatchment/Links Plan View.

To tune a model that is accurate based on each subcatchment, measurements will need to be taken at critical flow points and sortage basin locations, similar to how depth data is collected now at the outfall point. A pressure sensor records flow depth at the outfall of the watershed, and can be matched with the model's calculations for flow for the same measured storm event to evaluate the model's performance. A model is currently in the process of being built and calibrated. To calibrate the model, storage basin depth measurements will be used to tune parameter adjustments on an individual subcatchment basis.

Additional research can examine the potential of using an AWS machine learning platform (e.g. SageMaker) to perform a SWMM simulation on an automatic or when called for basis. Automated simulations could be leveraged to improve the accuracy of the flow model by iteratively adjusting subcatchment and conduit parameters through the use of cloud computing.

Acknowledgments: The team would like to recognize Ruchir Shah and Jacob Nelson for their knowledge and assistance on this project.

References

1. Floods. (2019, November 8). World Health Organization. https://www.who.int/health-topics/floods#tab=tab_1
2. NOAA and US National Weather Service, "Economic damage caused by floods and flash floods in the U.S. from 1995 to 2019," Statista, 15-Jan-2021. [Online]. Available: https://www.statista.com/statistics/237420/economic-damage-caused-by-floods-and-flash-floods-in-the-us/. [Accessed: 07-Apr-2021].
3. C.-Y. Hou, "Why are flash floods so hard to predict?," TheHill, 25-Dec-2019. [Online]. Available: https://thehill.com/changing-america/resilience/natural-disasters/474157-why-are-flash-floods-so-hard-to-predict. [Accessed: 07-Apr-2021].
4. Smart Cities Solutions for More Flood Resilient Communities | IEEE Conference Publication | IEEE Xplore. (n.d.). Retrieved April 12, 2021, from https://ieeexplore.ieee.org/document/8735625