

Article

Development of a Microservice-Based Storm Sewer Simulation System with IoT Devices for Early Warning in Urban Areas

Shiu-Shin Lin ^{1,*}, Kai-Yang Zhu ¹, Xian-Hao Zhang ², Yi-Chuan Liu ¹ and Chen-Yu Wang ¹

¹ Department of Civil Engineering, Chung Yuan Christian University, Taoyuan 320314, Taiwan; patzhu10@gmail.com (K.-Y.Z.); 020501y682@gmail.com (Y.-C.L.); x890904x@gmail.com (C.-Y.W.)

² Yuhong Water Technology Co., Ltd., Suzhou 002271, China; hnrzxx@gmail.com

* Correspondence: linxx@cycu.edu.tw

Abstract: This study proposes an integrated approach to developing a Microservice, Cloud Computing, and Software as a Service (SaaS)-based Real-Time Storm Sewer Simulation System (MBSS). The MBSS combined the Storm Water Management Model (SWMM) microservice running on the EC2 Amazon Web Services (AWS) cloud platform and an Internet of Things (IoT) monitoring device to prevent disasters in smart cities. The Python language and Docker container were used to develop the MBSS and Web API of the SWMM microservice. The IoT comprised a pressure water level meter, an Arduino, and a Raspberry Pi. After laboratory channel testing, the simulated and IoT-monitored water levels under different flow rates indicate that the simulated water level in MBSS was such as that monitored by the IoT. These findings suggest that MBSS is feasible and can be further used as a reference for smart urban early warning systems. The MBSS can be applied in on-site stormwater sewers during heavy rain, with the goal of issuing early warnings and reducing disaster damage. The use case can be the process by which the SWMM model parameters will be optimized based on the water level data from IoT monitoring devices in stormwater sewer systems. The predicted rainfall will then be used by the SWMM microservices of MBSS to simulate the water levels at all manholes. The status of the water levels will finally be applied to early warning.



Citation: Lin, S.-S.; Zhu, K.-Y.; Zhang, X.-H.; Liu, Y.-C.; Wang, C.-Y.

Development of a Microservice-Based Storm Sewer Simulation System with IoT Devices for Early Warning in Urban Areas. *Smart Cities* **2023**, *6*, 3411–3426. <https://doi.org/10.3390/smartcities6060151>

Academic Editor: Pierluigi Siano

Received: 1 October 2023

Revised: 16 November 2023

Accepted: 22 November 2023

Published: 5 December 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: SWMM; cloud computing; microservice; IoT; Docker container; SaaS

1. Introduction

The frequency of extreme precipitation events in urban areas has increased with climate change, which inevitably affects the design capacity of existing storm sewer systems. Governments worldwide are paying more attention to establishing real-time urban flooding simulation and forecasting systems; the objective is to improve urban storm sewer flooding early warning mechanisms based on sustainable development to reduce flood losses in urban areas caused by such changes. Based on the non-engineering method of sustainable development, forecasting systems can be established in advance. These systems can plan for disaster prevention in real-time, give early warnings for evacuating people, and reduce losses. The flow rate and corresponding water level of storm sewers during rainstorms are key factors in achieving this goal. It is essential for adequate flooding warnings to use the model of storm sewers to forecast the water condition of storm sewers when rain occurs in the future.

The Storm Water Management Model (SWMM) developed by the U.S. Environmental Protection Agency (USEPA) has been widely used in simulating urban storm runoff and storm sewer water management [1–3]. In the past 10 years, relevant studies have proposed to optimize SWMM model parameters and integrate urban hydrological models to apply SWMM more effectively to solve hydrological problems in urban areas. For example, Lin et al. [4] used a genetic algorithm to select the hydrological parameters needed for the RUNOFF and EXTRAN modules in SWMM. It has reduced the time required for

parameter selection and adjustment through manual experience in the past. The SWMM was optimized by Sadler et al. [5] to be capable of parallel computing. Liao et al. [6] used the Open Modeling Interface (OpenMI) framework to integrate the RUNOFF and EXTRAN models in SWMM, effectively solving the hydrological data exchange problem during flood simulations.

In recent years, SWMM simulation alone has not solved the complex hydrological problems in urban areas. Researchers have since combined SWMM with other technologies for the secondary development of SWMM. Lin et al. [7] integrated Google Map technology to display SWMM simulation results on mobile apps, providing a real-time reference for urban flooding warnings. Warsta et al. [8] developed an automated sub-catchment generator tool for SWMM to automate tedious stages in the model construction process. Xiao et al. [9] and Zeng et al. [10] designed a SWMM-based web service framework. Riaño-Briceño et al. [11] developed a tool for real-time control of urban drainage systems based on Matlab and Python. Allende-Prieto et al. [12] combined GIS and SWMM and incorporated them into urban design.

The hydrological environment is becoming increasingly complex under climate change. Urban areas are affected by the heat island effect, and extreme, short-duration heavy rainfall often leads to more severe disasters. The short duration of such hefty rainfall poses more challenges regarding early warning for urban storm prevention, such as conducting immediate urban storm sewer management operations and issuing information to inform the public. Traditional simulation is based on a single machine. Since computing power in this form is inadequate, long simulation times are necessary, and the transmission of real-time messages is restricted, which is not conducive to real-time early warning mechanisms. Suppose the high computing power of cloud services is used for hydrological simulation. In that case, the simulation efficiency can be improved, and messages can be transmitted in real-time more effectively [13,14]. However, to enable the SWMM model to be used for real-time simulation and messaging on cloud computing platforms, the following must be addressed: (1) dependencies on the execution environment (such as an operating system and library version) and (2) problems in network messaging (such as an operating system, programming language, library, and communication protocol). The microservice concept and Docker container technology make solving problems (1) that have persisted easier and more convenient for deploying cloud programs. The technology is easy to develop and maintain. Wan et al. [14] used microservices and Docker containers to encapsulate programs, which minimized deployment and execution costs. Pérez et al. [15] deployed Docker in the AWS (Amazon Web Services) cloud, processing many images. The results indicate that AWS could still work stably with a high computing workload. The Web API uses the standard HTTP communication protocol for a Remote Procedure Call to provide a solution to the problem (2) consistency.

Additionally, IoT technology is becoming increasingly mature and is often used for monitoring. For example, Fang et al. [16] suggested an integrated approach to examining snowmelt floods. Early caution has centered on RS (Remote Sensing), GIS (Global Information Systems), IoT, and cloud services. Edmondson et al. [17] connected the sewage pipe network with IoT to monitor the condition of the sewage system in real-time. For urban news and early warning disaster prevention, apart from the real-time SWMM simulation on a cloud platform, if IoT can be combined with real-time monitoring of water levels and data transmission, it will help to grasp the water management characteristics of storm sewers instantly.

Since SWMM simulation, cloud computing, and IoT are rarely applied to urban rainstorm disaster prevention and early warning systems, this study developed the real-time storm sewer simulation system (MBSS). It integrates the water management core of urban storm sewer simulation analysis based on the SWMM model, IoT, and front-end app. This study used a Docker container to encapsulate the water management core based on a microservice architecture. It further developed microservices combining the water management core (called SWMM microservice), drawing, and database. The microservices

were deployed on the AWS cloud platform, and a Web API was provided to front-end developers. The front-end app was developed by calling a Web API. Users can use the app to simulate SWMM anytime and anywhere and upload the measurement data of the on-site measurement equipment to the cloud through IoT technology for comparison and analysis with the SWMM simulation results. This study includes an assessment of the feasibility of artificial channels in the laboratory, which can be used as a reference for subsequent disaster prevention and early warning-related applications. In practical applications, real-time water level monitoring data from on-site IoT devices in urban stormwater sewer systems is employed to optimize the SWMM model parameters. The SWMM microservice is then applied to simulate and predict the water levels at sewer manholes, which are important for early warning based on the forecasted rainfall. The MBSS with SWMM microservices and IoT monitoring can be applied to practical applications for effective early disaster warnings and loss reduction.

2. Methodology

2.1. Cloud Computing Service Platform

The cloud computing service used in this study was Platform as a Service (PaaS), which provides infrastructure services, including hardware and operating systems. PaaS is between Infrastructure as a Service (IaaS) and Software as a Service (SaaS) and can be upgraded and expanded in the future. When using PaaS, operating systems, underlying infrastructure, middleware services, and other complex software and hardware are left to the vendors, while developers only need to address software deployment and implementation.

Currently, the mainstream cloud computing service platforms are Microsoft Azure, Google Cloud Platform (GCP), and Amazon AWS. Amazon Elastic Compute Cloud (Amazon EC2) is a web service developed by Amazon to provide developers with secure and reliable cloud computing services. Amazon EC2 encompasses Linux, Windows, and MacOS operating environments, as well as a variety of processors and storage capacities to meet different computing needs. It can optimize the virtualization system, manage and allocate hardware, improve server performance, availability, and security, and reduce operating costs. Meanwhile, the 400 Gbps-enhanced Ethernet network used by AWS can maintain high-speed transmission, reduce network fluctuation, reduce network delay, and respond to user requests in high I/O instances when mass access and data exchange occur. When conducting a real-time simulation of SWMM over a large area, EC2, with its excellent performance, allows the simulation to be completed quickly; high-speed network connections can instantly return simulation results to the front end, ensuring that the information is sent back in time.

Amazon EC2 has considerable service capabilities and data centers worldwide. EC2 can cluster containers using Amazon Elastic Container Service (Amazon ECS) without additional installation or maintenance. It provides a good foundation for the functional extension of the system. When new functions are developed and multiple containers are available, ECS will be used for container management. Therefore, Amazon EC2 was selected as the cloud computing service platform for this study.

2.2. Hydrological Simulation Kernel

This study used SWMM, which is a hydrologic model developed by the United States Environmental Protection Agency for urban rainfall-runoff and water quality simulation [8]. This model primarily simulates single events or long-term continuous events. SWMM applies to a variety of hydrologic environments. It can achieve good simulation effects for urban or non-urban areas and small or large watershed regions and is widely used in urban drainage system design. SWMM was first developed in 1971 and has undergone several significant upgrades to the present version, SWMM5.1. The SWMM surface runoff module (SWMM-RUNOFF) calculates the runoff formed by rainfall. The runoff is used as input to the storm sewer module (SWMM-EXTRAN) to calculate the water level and discharge of manholes and pipelines.

SWMM-RUNOFF simulates runoff and pollutant production from rainfall in a catchment. SWMM-EXTRAN uses numerical methods to solve the dynamic wave model of one-dimensional Saint-Venant Equations, as shown in (1) and (2). Equation (1) is the equation of mass conservation, and Equation (2) is the momentum conservation equation of water transfer.

$$\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = 0 \quad (1)$$

$$\frac{\partial Q}{\partial t} + \frac{\partial(Q^2/A)}{\partial x} + gA \frac{\partial H}{\partial t} + gAS_f = 0 \quad (2)$$

where,

x : Storm sewer length (m)

t : Time (s)

A : Channel cross-sectional area (m²)

Q : Flow (cms)

H : Hydraulic head (m)

S_f : Friction slope

g : Acceleration of gravity (m/s²).

SWMM5.1 provided the source code and binary execution installation file. The SWMM source program was rewritten in the C language. It can be compiled and executed on Windows and Linux operating systems, while the binary execution installation file is based on the Windows system as the execution platform. Additionally, it is a Graphical User Interface (GUI) application, which does not meet the needs of the MBSS developed in this study. Therefore, secondary development was necessary for the SWMM to enhance and expand its functions. Its secondary development includes Python, C++, Matlab, and other commonly used languages. Python, with its advantages of simplicity, efficiency, and low learning threshold, provides a suitable development environment for the secondary development of SWMM and provides more expansion functions.

OpenSWMM is an open-source computing engine module based on EPA SWMM 5.1.012. Developers worldwide can secondary-develop SWMM through OpenSWMM. PySWMM, developed by McDonnell et al. [18] in the Python language, is an example of its secondary development. PySWMM can execute the SWMM model, set different parameters, and generate custom simulation results. For instance, it can set rainfall time series, pollutants, canal, flow, and other parameters. The output and report files are automatically generated, which are combined with various Python modules for further analysis and visualization of the simulation results. Thus, it has exceptionally high scalability. This study used the PySWMM module as the basic module for developing the hydrologic kernel.

2.3. Docker Container

Docker was first released as an open-source platform in 2013 under the name dotCloud. It is an advanced container engine based on LXC (Linux container) kernel virtualization technology and written in the GO language. Docker uses a client/server architecture. As shown in Figure 1, the Docker client uses a remote API to call and execute the Docker. Docker Daemon receives requests from users and performs corresponding operations according to those requests. Images are read-only templates for creating a new Docker container. A graph is used for image management and storage. Users can use local GraphDB or a cloud-based Docker HUB to obtain Docker images. Finally, the Docker image is executed to complete the creation of the Docker container.

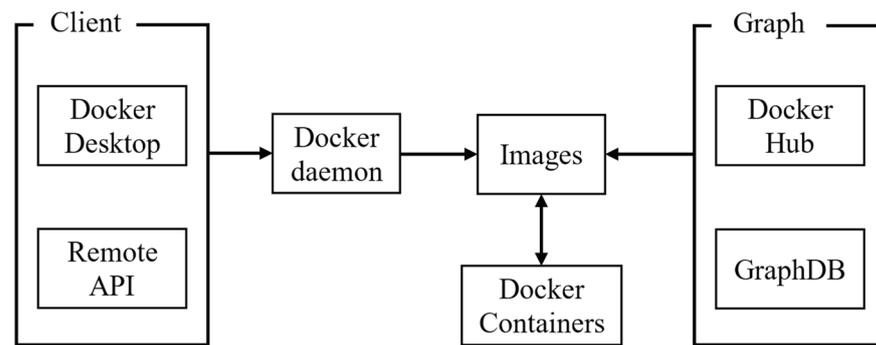


Figure 1. Architecture of Docker.

A Docker provides operating system virtualization, while traditional virtual machines provide virtualization from the underlying hardware. Therefore, the Docker does not need to encapsulate the entire host, thus saving a lot of space. Since a Docker is based on LXC, it has the advantage of being a lightweight virtual machine and is superior in startup speed, memory occupation, and other aspects. A Docker usually takes up only a few dozen megabytes of space. Since it occupies only a small space, it allows multiple containers to be executed on one machine simultaneously. The same system kernel can be shared among containers, thus reducing memory usage and improving execution efficiency.

Docker images encapsulate the program code and all its execution environments, including the system environment, system settings, and software library. The program can be easily migrated through images on multiple platforms with stable execution. Presently, Docker is being applied more widely, especially in cloud services. Docker improves server efficiency and reduces server execution costs with its security, lightweight, and portable features. Therefore, this study used a Docker container to encapsulate the CHMS.

2.4. Web API

The Representational State Transfer (REST) protocol, proposed by Fielding [19], is a standard architecture for web services and APIs that defines a set of constraints and principles. In constructing microservices, REST can unify the way each API service is written, making it easier for developers to maintain and manage. In REST rules, each URI represents a resource. The interaction between the front and back ends is stateless between requests, i.e., the server does not store any state about the front-end session. Therefore, every request sent from the front end to the server must contain the information the server needs to complete the request. Flask-RESTful is a module of Flask, and Resources are the basis of this module. The module is constructed in Flask Pluggable Views. It enables users to build microservices in Flask using a RESTful API that maps multiple HTTP modes into the same category. The front end calls the API through GET, POST, PUT, and DELETE. Specifically, GET means obtaining resources, POST means inserting new resources, PUT means updating resources, and DELETE means deleting resources.

Flask has many advantages over other frameworks in microservice construction. REST is used to compile services, facilitating subsequent upgrades and expansion of services. Therefore, Flask-RESTful was used in the API development of the MBSS in this study.

2.5. IoT Technology for Water Level Gauges

2.5.1. Arduino

Arduino is an open-source electronics platform based on easy-to-use hardware (the Arduino board, as shown in Figure 2a) and software (the Arduino IDE). Users can connect various modules through the Arduino board, including environmental modules such as temperature and humidity sensors, gas and smoke sensors, infrared sensors, and communication modules such as the GMS module, Bluetooth, and Wi-Fi. Programs are written in the IDE, compiled into binaries, and created on the Arduino board. Arduino IDE can run on most mainstream operating systems, including Windows, macOS, and Linux. The

programming language used by the IDE is based on the programming language and can be extended through C++ libraries. Arduino is open source, both in terms of hardware and software. Due to its high scalability and flexibility, Arduino is becoming more widely used. In the present study, Arduino was responsible for receiving the output signal of the pressure water level meter, converting the voltage into the water level depth through a program written by the Arduino IDE, and recording the real-time water level.

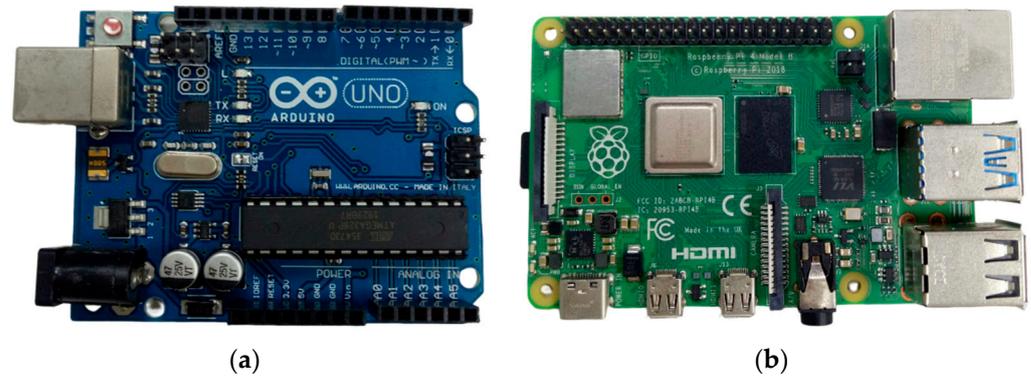


Figure 2. Arduino electronic board (a) and Raspberry Pi (b).

2.5.2. Raspberry Pi

The Raspberry Pi is a tiny computer developed by the UK Raspberry Pi Foundation, as shown in Figure 2b. Raspberry Pi is based on ARM architecture, uses an SD card as the storage medium, and has input and output serial ports such as a USB, Ethernet port, and HDMI. It can execute various Linux systems, including the official customized Raspberry Pi OS, Arch Linux ARM, Ubuntu MATE, etc. The Raspberry Pi supports Java, Python, C, Perl, and other programming languages, and the GPU supports OpenGL. As a credit-card-sized computer, the Raspberry Pi is widely used in the development process of IoT for its rich application scenarios and portability.

This study used the Raspberry Pi as a tool for water-level data processing and uploading. After the Raspberry Pi was connected to the Arduino, the water level data were read, sorted, and saved with Python. Next, the Web API uploaded the data to the CHMS. Although Arduino can transmit data directly to the cloud through the Wi-Fi module, because Raspberry Pi can preprocess data and then develop Edge Computing, Arduino was combined with Raspberry Pi for IoT development.

3. System Structure

Figure 3 shows the MBSS architecture proposed in this study. It is divided into a Cloud hydrological microservice (CHMS or SWMM microservice), a Front-end APP, and IoT devices. The CHMS architecture was built on the Cloud computing platform.

CHMS includes the water management simulation kernel, plotting, and database functions. The Web API was constructed with the Flask framework, Python packages, and RESTful format and was finally encapsulated by the Docker container. CHMS was deployed to the AWS EC2 cloud computing platform in Docker container mode. The IoT device comprised a Raspberry Pi, Arduino, and a pressure water level meter and sent the water level data from the site to the cloud service through a Web API. The front-end APP comprised a Web APP and a mobile app, which provided users with two ways to conduct real-time SWMM simulation. They could view the simulation results through the web or a mobile phone and monitor the water level with the in-situ IoT.

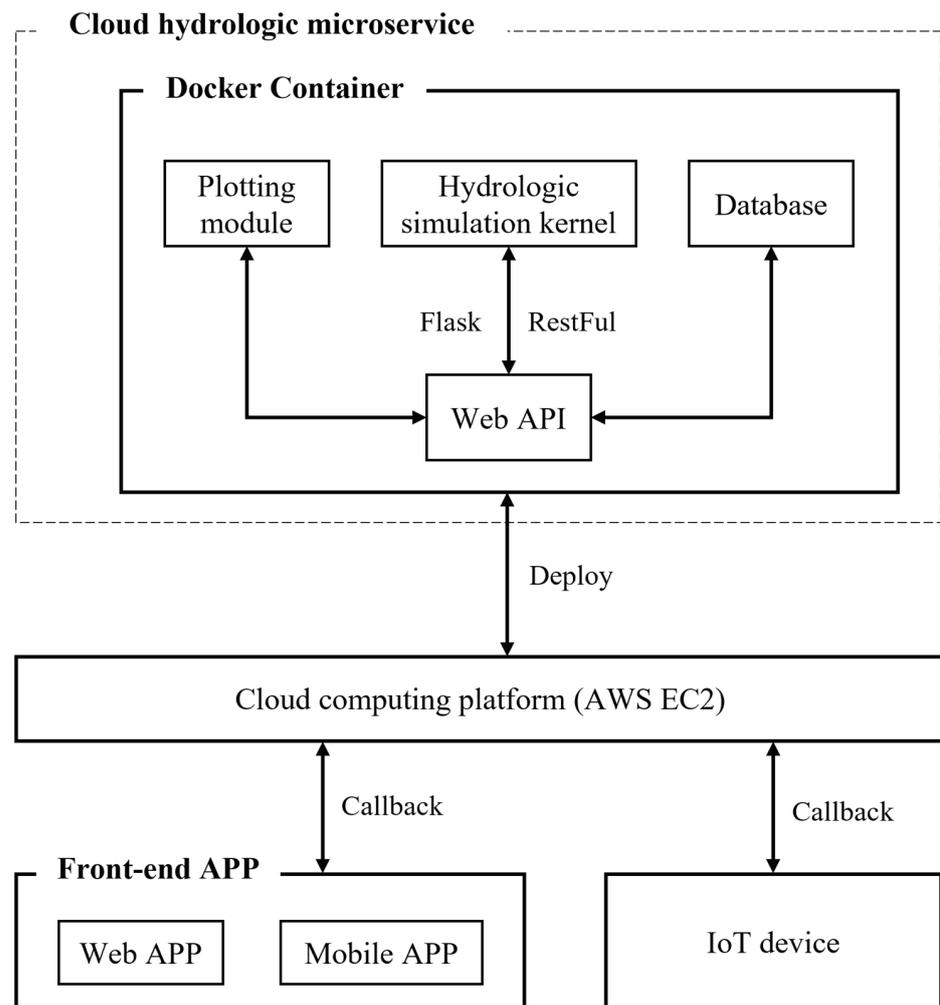


Figure 3. Architecture of a Microservice- and Cloud-Based Real-Time Storm Sewer Simulation System (MBSS).

4. System Implementation

4.1. Cloud Computing Platform Deployment

Cloud computing platform deployment can be divided into AWS EC2 and Docker deployment.

(a) AWS EC2 configuration

In this study, Amazon Linux 2 AMI 64-bit (x86) was used as the image of the virtual machine. The Linux 4.14 kernel and an Intel Xeon high-frequency CPU were used to establish the virtual machine environment for executing Docker. The corresponding port used by the API was opened in the security option to ensure normal communication between the front and back ends.

(b) Docker deployment

In Docker deployment, we started the EC2 service, entered the server terminal, logged in, and started the Docker service. Next, we input instructions to download the image from Docker Hub to the server. After downloading and verifying the image, we input instructions to start and execute the Docker image. After these steps, the Docker could be deployed to the cloud and completed successfully.

4.2. Construction of a Docker Container

Developers can bundle software and environment packages into a lightweight, portable container through Docker. This study used Docker container virtualization technology to encapsulate the storm water simulation kernel. The PySWMM water management core basic module, Matplotlib module, SQLite module, and SWMM execution operating environment (including the operating system and dependent library) were encapsulated into the container. The PySWMM module mainly executed SWMM for hydrological simulation; the Matplotlib module further analyzed and plotted the PySWMM output results; and the SQLite module wrote simulation results as a database.

There were several necessary steps for constructing a Docker container: (1) First, the execution environment for the Docker image was established; (2) construction instructions using a Docker file were written. For example, we specified the execution environment of the image, the path when the Docker was deployed to the server, executed commands when the images were constructed, and executed automatically when the Docker was executed; (3) Docker packing instructions were executed in the terminal console to construct a Docker image; (4) We uploaded to the Docker hub cloud after construction and then deployed it on the AWS server.

4.3. Development of Web APIs

In this study, the Web API of CHMS was developed using the Python language Flask_Restful module. There are mainly five types of Web APIs, as shown in Figure 4:

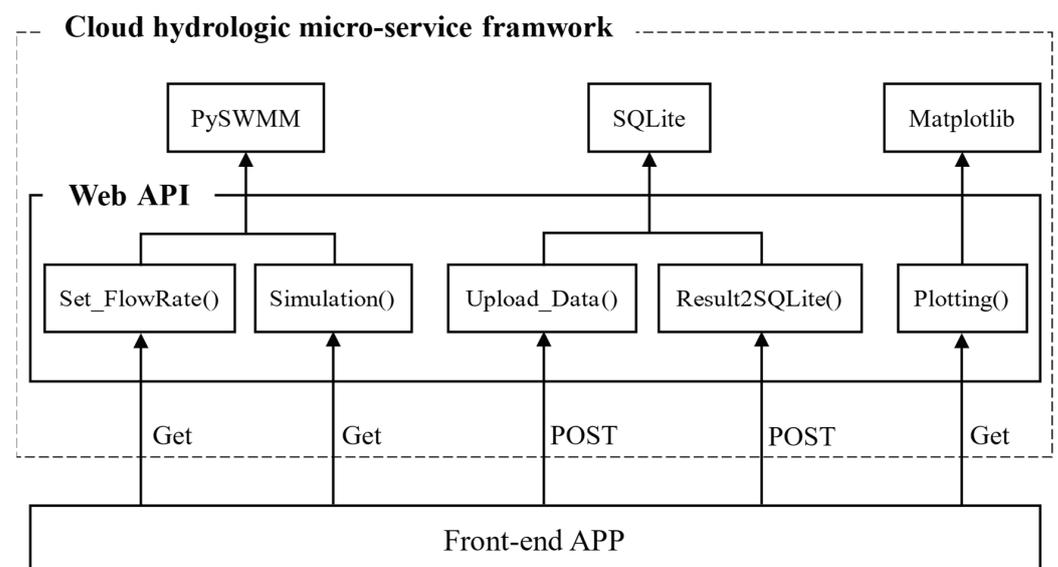


Figure 4. Architecture of the CHMS API.

1. Upload_Data(): The SWMM input file in INP format and the water level gauge measurement data were uploaded and saved to the cloud server as Excel files by POST. Upload_Data() comprises two parts. The two parts were responsible for uploading the SWMM input file and the Excel document, respectively.
2. Set_FlowRate(): The SWMM simulation parameters were defined, such as node flow, by calling the PySWMM parameter setting module in POST mode.
3. Simulation(): The simulation request in GET mode was sent and simulated after the service received the request.
4. Result2SQLite(): The simulation results were transformed in the SQLite database based on the node IDs in GET mode. According to the user's request, the SQL instruction sent the node water level data back to the front end in JSON format.

- Plotting(): Based on the simulation results and the measured data of the water level gauge in GET mode, we used the Matplotlib module to plot a pair of line graphs and sent them back to the front end in PNG format.

The class diagram of the Web API is shown in Figure 5. The class of API () contained the above five types of API. API established the service and URL correspondence through api.add_resource(). PySWMM, SQLite, and Malplotlib modules process simulation, database reading and writing, and plotting steps, respectively.

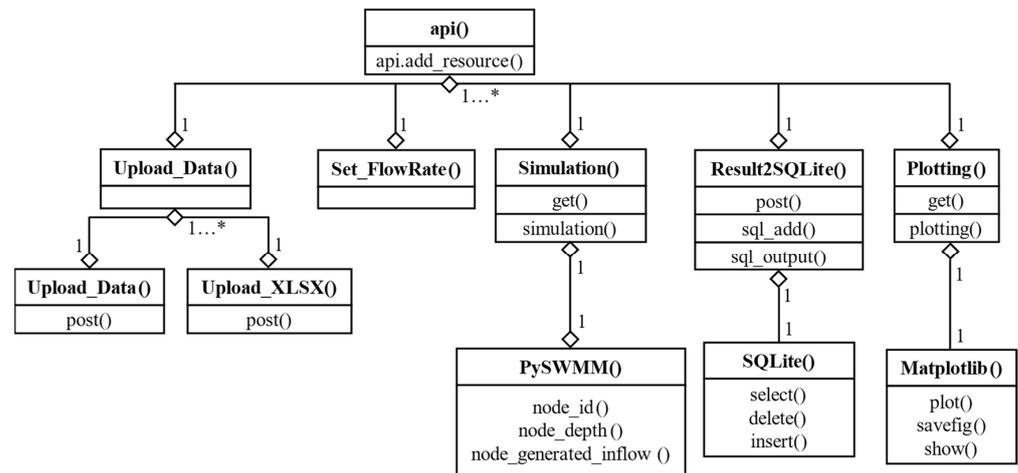


Figure 5. A class diagram of a web API.

4.4. Real-Time Water Level Monitoring

In this study, the IoT device was used to monitor water levels. In the laboratory, a pressure water level meter was used to convert the current of the liquid level sensor into voltage through the electric flow voltage module. Arduino received the voltage signal from the water level meter and converted it into the corresponding water level. Finally, the Raspberry Pi was connected to read water level data and call the API to upload the data to the cloud. The operation procedure of the IoT device is shown in Figure 6.

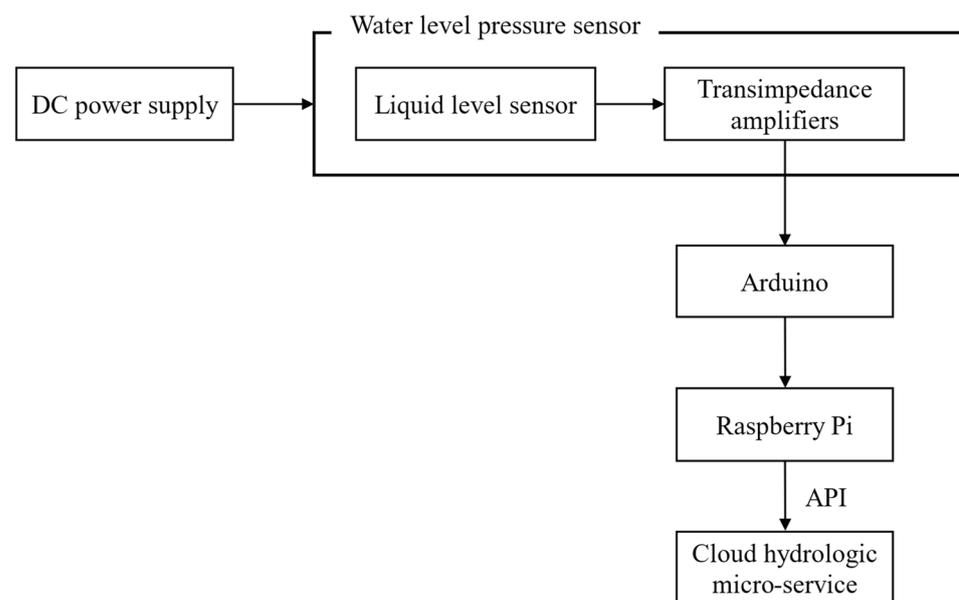


Figure 6. Process of real-time water level monitoring.

The Raspberry Pi is connected to Arduino through a USB. It used a serial module to read the serial port signal and set the serial port baud rate to 9600, consistent with Arduino, using a Python script. The serial module reads serial port data line by line. The water level data and time were recorded and imported into the xldr tool to write the results into Excel forms. Then, it called Upload_Data() to upload the water level file to the cloud.

4.5. Development of a Front-End App

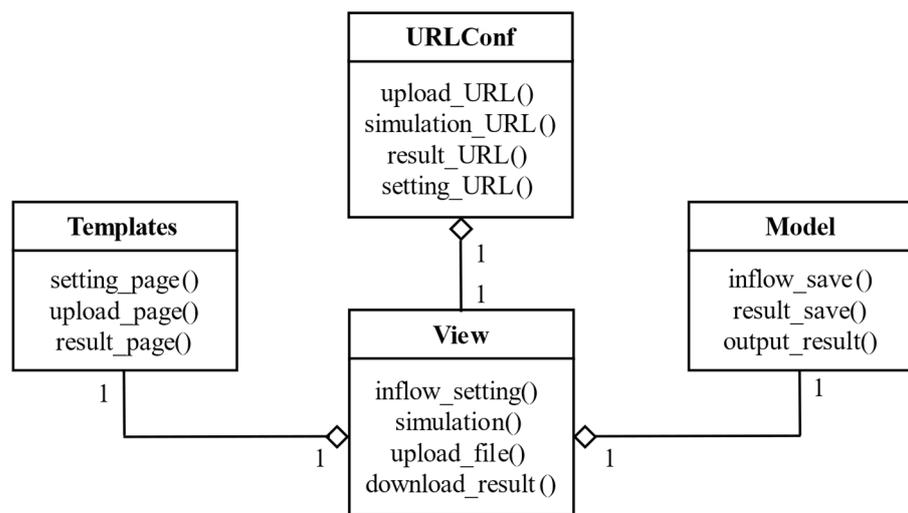
4.5.1. Development of a Web App

This study used Django as the framework for the development of web apps. Django, developed with MVC (Model View Controller) Design patterns, is a commonly used Web framework in Python. The user sends a request through the browser, and the view executes the corresponding function based on the request, such as inflow setting (inflow_setting()), file uploading (upload_file()), simulation (simulation()), and result in download (download_result()). The viewer accesses the database through the model. The template renders the web page according to the model's output data and sends the result back to the view for the user.

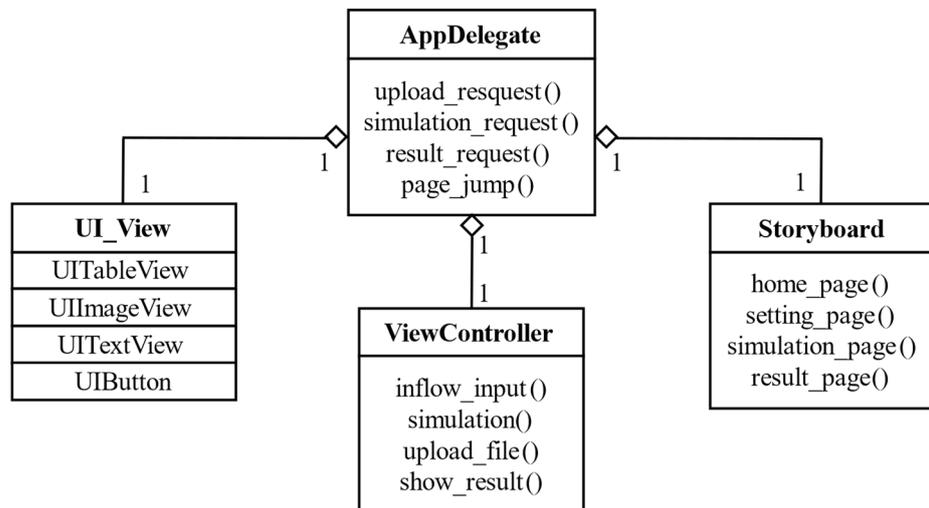
4.5.2. Development of the Mobile App

The Swift language was used in this study for the development of mobile apps. Swift is a programming language created by Apple in 2014 to provide developers with a powerful language to develop programs on iOS and macOS platforms. Swift combines the best features of Objective-C and C and is more compatible with the Cocoa framework and Objective-C. Swift compiles code using the LLVM compiler, which provides extremely fast compilation speed while ensuring consistent execution. Swift has a more concise syntax than the Objective-C language widely used in iOS and macOS. Mobile apps use SwiftUI to compile the UI interface, and SwiftUI's declarative syntax dramatically simplifies the programming difficulty. The app uses Alamofire, an API tool commonly used in Swift development, as the API call module.

Mobile app development with the Swift language also adopts MVC architecture. The Storyboard is the Model; the ViewController is the Controller, which contains the inflow, performs the simulation, uploads the files, displays the simulation results, etc. The different types of UI interfaces in the APP are the subcategories of the View, including tables, pictures, text, and buttons. AppDelegate processes the communication between ViewController, Storyboard, and UI_View according to the user requests, such as simulation requests and page jumps. The class diagrams of web APPs and mobile APPs are shown in Figure 7.



(a)



(b)

Figure 7. Class Diagram of web APP (a) and mobile APP (b).

4.6. MBSS Operation Process

Figure 8 shows the sequence diagram of the MBSS. The IoT device uploaded the measured water level data using Measure_Data(). The middleware received the data, and the controller saved the data to the database. Simultaneously, the controller requested the INP format input file of SWMM from the user, and the user called “upload to API” to upload the input file. After receiving the input file, the Controller executes Simulation(), which is used to call the hydrologic simulation core for the SWMM simulation. Measure_Data_Select() obtained water level measurements from the database and compared them with simulated water levels. After the simulation analysis was completed, Output_Save() stored the results in the database and Receive_Output() returned the results to the front end for display to the user.

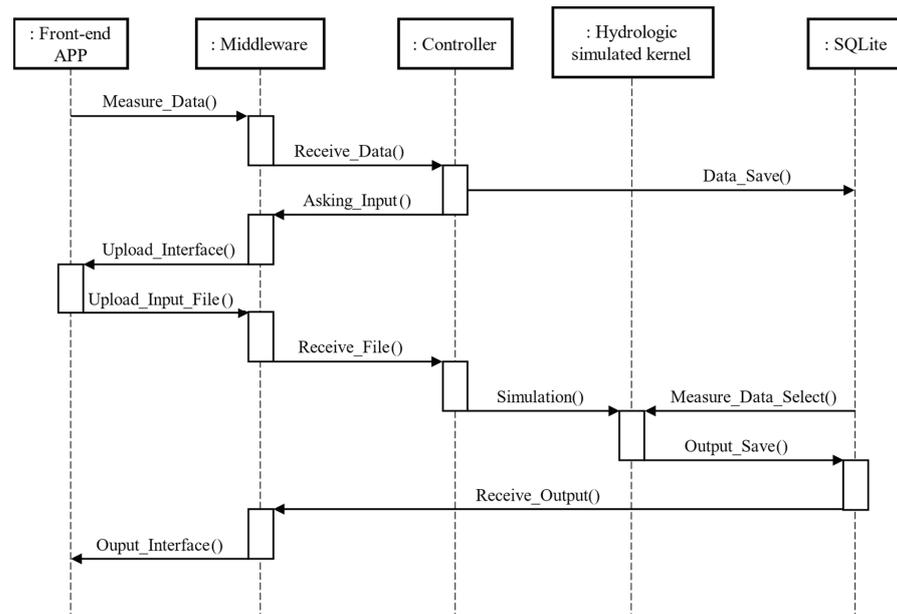


Figure 8. Sequence diagram of MBSS.

5. Results

In this study, the laboratory artificial channel (Figure 9a) was taken as the case for simulating the storm sewer flow. The total length of the artificial channel is 20 m, the section width is 0.5 m, and the section depth is 0.5 m. The bottom of the channel is made of cement, and the two sides are smooth glass. Manning’s n of the channel was set as 0.01, according to the technical specifications of soil and water conservation and relevant specifications. The channel was divided into four parts: three manhole nodes (Nodes 1–3) and one water outlet (Node 4), as shown in Figure 9b.



(a)

Figure 9. Cont.

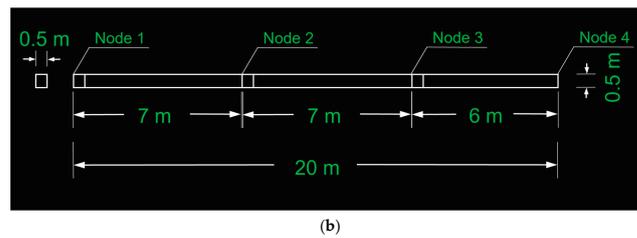


Figure 9. The photograph (a) and the Autocad drawing (b) of the artificial canal.

The pressure water level meter was calibrated before the simulation. The calibration process is as follows: (1) The pump was started and waited for it to run for 15 min to stabilize the water level; (2) The pressure water level meter was placed and needle water level gauge; (3) Manning’s roughness (Manning’s n) for the channel; (4) Different flow rates ($Q = 0.01\sim 0.05$ cms) were adjusted and water level data were read and compared. The pressure gauge was calibrated by adjusting the Arduino code.

After calibration, different flow rates ($Q = 0.01\sim 0.05$ cms) were set, respectively. Meanwhile, Arduino was set to record the measurement data of the pressure water level meter every 5 min. SWMM of MBSS was also simulated with the same flow. The flow rate of 0.01 cms was taken as an example. The changes in the measured water level and simulated water level of Nodes 1 to 4 over time are shown in Figure 10. The findings indicate that the simulated water level had a significant error with the measured water level at the beginning and end, and the water level error from the 4th minute to the 55th minute was within 0.025 m. Figure 11 shows the distribution diagram of the measured and simulated water levels when the flow rates ranged from 0.01 to 0.05 cm. The findings indicate that the maximum error was about 0.025 m under the five flow conditions.

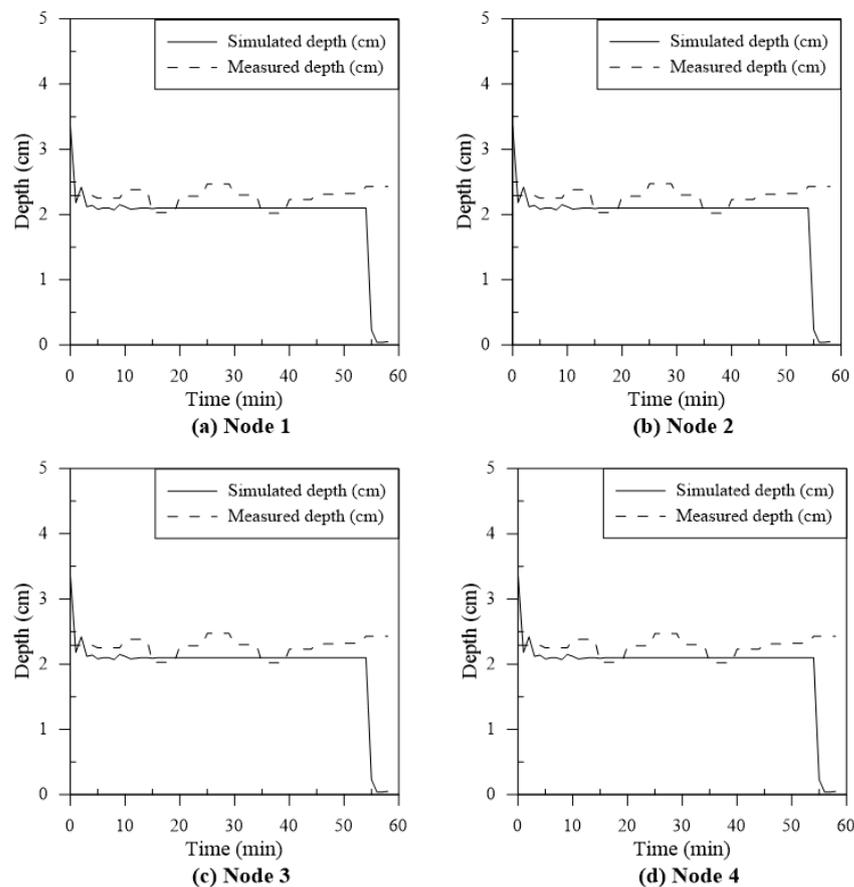


Figure 10. The relationship between the simulated water level and the measured water level from Node 1 to Node 4 at a flow rate of 0.01 CMS (m^3/s) over time.

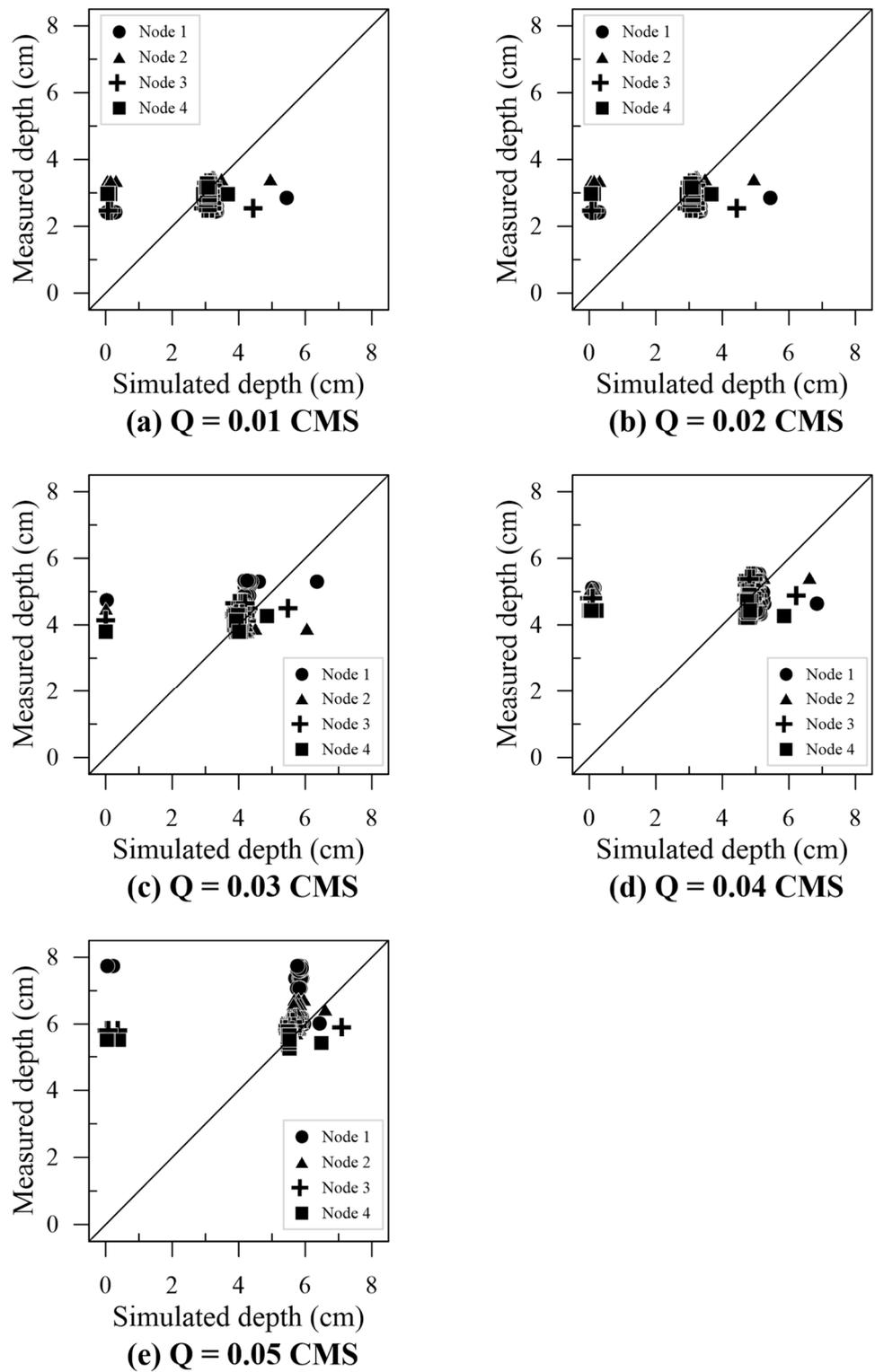


Figure 11. The scatter diagram shows the simulated and measured water levels from Node 1 to Node 4 at a flow rate of 0.01 CMS to 0.05 CMS.

Although the pressure water level meter had been calibrated, because of the laboratory pump’s power limitation, the channel’s simulated flow rate was low. Under the conditions of low water level and low water pressure, the pressure water level meter will still have some errors in water level measurement. If the experiment is conducted under the conditions of a high flow rate and a high water level at the site, the error will be further reduced.

Please refer to the Supplement Material for the system presentation. The MBSS system demonstration was shown in the Supplementary Material section at the end of this paper.

6. Conclusions

This study combined the SWMM model, cloud computing, and IoT technology to develop the MBSS, a front-end Web APP, and an iOS mobile app. With CHMS of MBSS as the cloud microservice, an IoT water level meter monitored the water level in real-time. First, the water management simulation kernel was developed with the SWMM model, and the Web API was written with the Flask framework. Subsequently, the API was encapsulated and deployed to the AWS EC2 cloud server with a Docker container. The artificial channel in the laboratory was taken as an example. Real-time water level measurement was conducted through the pressure water level meter. The IoT device was comprised of an Arduino and a Raspberry Pi. The data were uploaded to the cloud server. The web APP and iOS mobile APP were simulated, and the following conclusions were drawn after simulation analysis:

1. The water level simulated by SWMM Microservice was compared with that measured by a needle water level gauge under different flow conditions. The error (+1.1 cm, −0.45 cm) of the water level simulated by SWMM Microservice was smaller. This indicates that SWMM Microservices could accurately simulate the water level of the artificial channel.
2. The IoT device was used to measure the water level of the artificial channel in the laboratory. The error between the device and the needle water level gauge was small (+0.46 cm, −0.26 cm). Therefore, using the IoT device as an in-situ water level measuring device is feasible.
3. MBSS provided a Web API for the front end to conduct a real-time SWMM simulation. The simulation results were compared with the IoT device-measured data displayed through a mobile app to achieve a real-time and convenient simulation.
4. MBSS was constructed through microservices. The simulation services were split and implemented independently so that developers could maintain and upgrade them independently. Then, it was encapsulated by a Docker, which saved space occupied by deployment to the server and improved execution efficiency and follow-up maintenance.

7. Suggestions and Limitations

The MBSS system developed in this study integrates the SWMM model as a microservice, IoT technology, and cloud computing services. This system has been successfully tested in artificial channels analogizing stormwater sewer pipelines. For future applications, the proposed MBSS system can be implemented together with the operation of a real-world urban stormwater sewer system. An example of an operational process may include the following steps: (1) utilizing real-time water level measurements from the IoT devices in the stormwater sewer systems, such that the SWMM model parameters can be dynamically adjusted and optimized; (2) during the period of heavy rain, simulating the water levels at each manhole and discharge in each conduct with the SWMM microservice of the MBSS; (3) based upon the simulation results, issuing the early warnings via a front-end app. This operational process could effectively reduce flood-related losses, contributing to the goal of smart city stormwater disaster prevention.

Supplementary Materials: The following supporting information can be downloaded at: <https://www.mdpi.com/article/10.3390/smartcities6060151/s1>.

Author Contributions: Conceptualization, S.-S.L.; Methodology, S.-S.L.; Software, X.-H.Z.; Validation, K.-Y.Z., Y.-C.L. and C.-Y.W.; Investigation, K.-Y.Z. and X.-H.Z.; Writing—original draft, S.-S.L. All authors have read and agreed to the published version of the manuscript.

Funding: Project No. 111-2625-M-033-001 under the Taiwan National Science and Technology Council, Taiwan.

Data Availability Statement: The data presented in this study are available in the article itself.

Conflicts of Interest: Author Xian-Hao Zhang was employed by the company Yuhong Water Technology Co., Ltd. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Cipolla, S.S.; Maglionico, M.; Stojkov, I. A long-term hydrological modelling of an extensive green roof by means of SWMM. *Ecol. Eng.* **2016**, *95*, 876–887. [[CrossRef](#)]
2. Ophiyandri, T.; Istijono, B.; Hidayat, B. Identification of Drainage Systems Capacity Using EPA-SWMM 5.1 Version Modeling in Gunung Panglun of Padang City. *Geomate J.* **2020**, *18*, 16–22. [[CrossRef](#)]
3. Huber, W.C.; Dickinson, R.E.; Barnwell Jr, T.O.; Branch, A. *Storm Water Management Model; Version 4*; Environmental Protection Agency: Washington, DC, USA, 1988.
4. Lin, W.L.; Kan, P.S.; Lin, S.S. Integrating web GIS technology and an ANFIS model to develop a warning system of storm sewer water stage. *J. Technol.* **2014**, *29*, 105–114.
5. Sadler, J.M.; Goodall, J.L.; Behl, M.; Morsy, M.M.; Culver, T.B.; Bowes, B.D. Leveraging open source software and parallel computing for model predictive control of urban drainage systems using EPA-SWMM5. *Environ. Model. Softw.* **2019**, *120*, 104484. [[CrossRef](#)]
6. Liao, Y.P.; Lin, S.S.; Chou, H.S. Integration of urban runoff and storm sewer models using the OpenMI framework. *J. Hydroinformatics* **2012**, *14*, 884–901. [[CrossRef](#)]
7. Lin, S.S.; Chai, W.S.; Hsieh, S.H.; Liao, Y.P. Integrating real-time storm sewer simulation and web GIS into urban sewer warning system. *Chin. Inst. Civ. Hydraul. Eng.* **2009**, *36*, 080–086.
8. Warsta, L.; Niemi, T.J.; Taka, M.; Krebs, G.; Haahti, K.; Koivusalo, H.; Kokkonen, T. Development and application of an automated subcatchment generator for SWMM using open data. *Urban Water J.* **2017**, *14*, 954–963. [[CrossRef](#)]
9. Xiao, D.; Chen, M.; Lu, Y.; Yue, S.; Hou, T. Research on the construction method of the service-oriented web-SWMM system. *ISPRS Int. J. Geo-Inf.* **2019**, *8*, 268. [[CrossRef](#)]
10. Zeng, Z.; Yuan, X.; Liang, J.; Li, Y. Designing and implementing an SWMM-based web service framework to provide decision support for real-time urban stormwater management. *Environ. Model. Softw.* **2021**, *135*, 104887. [[CrossRef](#)]
11. Riaño-Briceño, G.; Barreiro-Gomez, J.; Ramirez-Jaime, A.; Quijano, N.; Ocampo-Martínez, C. MatSWMM—an open-source toolbox for designing real-time control of urban drainage systems. *Environ. Model. Softw.* **2016**, *83*, 143–154. [[CrossRef](#)]
12. Allende-Prieto, C.; Méndez-Fernández, B.I.; Sañudo-Fontaneda, L.A.; Charlesworth, S.M. Development of a geospatial data-based methodology for stormwater management in urban areas using freely-available software. *Int. J. Environ. Res. Public Health* **2018**, *15*, 1703. [[CrossRef](#)] [[PubMed](#)]
13. Euzébio, T.A.; Ramirez, M.A.; Reinecke, S.F.; Hampel, U. Energy Price as an Input to Fuzzy Wastewater Level Control in Pump Storage Operation. In *IEEE Access*; IEEE: Piscataway, NJ, USA, 2023.
14. Junior, Ê.L.; da Silva, M.T.; Euzébio, T.A. Avoiding Buffer Tank Overflow in an Iron Ore Dewatering System with Integrated Control System. *Sustainability* **2022**, *14*, 9347. [[CrossRef](#)]
15. Pérez, A.; Moltó, G.; Caballer, M.; Calatrava, A. Serverless computing for container-based architectures. *Future Gener. Comput. Syst.* **2018**, *83*, 50–59. [[CrossRef](#)]
16. Fang, S.; Xu, L.; Zhu, Y.; Liu, Y.; Liu, Z.; Pei, H.; Yan, J.; Zhang, H. An integrated information system for snowmelt flood early-warning based on internet of things. *Inf. Syst. Front.* **2015**, *17*, 321–335. [[CrossRef](#)]
17. Edmondson, V.; Cerny, M.; Lim, M.; Gledson, B.; Lockley, S.; Woodward, J. A smart sewer asset information model to enable an ‘Internet of Things’ for operational wastewater management. *Autom. Constr.* **2018**, *91*, 193–205. [[CrossRef](#)]
18. McDonnell, B.E.; Ratliff, K.; Tryby, M.E.; Wu, J.J.X.; Mullapudi, A. PySWMM: The python interface to stormwater management model (SWMM). *J. Open Source Softw.* **2020**, *5*, 2292. [[CrossRef](#)] [[PubMed](#)]
19. Fielding, R.T. *Architectural Styles and the Design of Network-Based Software Architectures*; University of California: Irvine, CA, USA, 2000.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.