International Journal of Engineering and Techniques - Volume 12 Issue 2, March-April - 2026

RESEARCH ARTICLE OPEN ACCESS

WSN-Based Environmental Monitoring System with AWS Cloud Integration

Naveen Upadhaya[1], Praveen Kumar D[2]

[1](DoEEEVE, NIMS University, Jaipur, India

Email: naveen999op@gmail.com)

[1]2(DoEEEVE, NIMS University, Jaipur, India

Email: praveen.kumar1@nimsuniversity.org)

ABSTRACT

Environmental monitoring has become increasingly critical across industrial, agricultural, and residential domains, demanding systems that are both cost-effective and reliably connected to cloud infrastructure. This paper presents the design and implementation of an IoT-based environmental monitoring system utilizing the ESP8266 NodeMCU microcontroller in conjunction with a DHT11 temperature and humidity sensor and an LM393 ambient light sensor module. The system continuously acquires real-time environmental parameters and serves live data through a locally hosted HTTP web server, enabling visualization via a custom- designed responsive web dashboard accessible on any networked device. Cloud integration is achieved through AWS API Gateway and AWS Lambda, where sensor readings are transmitted every ten seconds and persistently stored in Amazon S3 as both daily CSV files and individual JSON records with accurate UTC timestamps. Automated threshold-based email alerts are delivered through Amazon SNS when ambient temperature exceeds user-defined limits, with a firmware-enforced cooldown mechanism preventing notification redundancy. A dedicated cloud viewer application enables historical data retrieval, statistical analysis, and Excel-based reporting directly from the browser. The proposed system demonstrates that sophisticated environmental monitoring with permanent cloud archiving and intelligent alerting can be realized at minimal cost using commercially available embedded hardware and serverless cloud services, making it particularly suitable for smart home, greenhouse, and laboratory monitoring applications. environmental parameters and serves live data through a locally hosted HTTP web server, enabling visualization via a custom- designed responsive web dashboard accessible on any networked device. Cloud integration is achieved through AWS API Gateway and AWS Lambda, where sensor readings are transmitted every ten seconds and persistently stored in Amazon S3 as both daily CSV files and individual JSON records with accurate UTC timestamps. Automated threshold-based email alerts are delivered through Amazon SNS when ambient temperature exceeds user-defined limits, with a firmware-enforced cooldown mechanism preventing notification redundancy. A dedicated cloud viewer application enables historical data retrieval, statistical analysis, and Excel-based reporting directly from the browser. The proposed system demonstrates that sophisticated environmental monitoring with permanent cloud archiving and intelligent alerting can be realized at minimal cost using commercially available embedded hardware and serverless cloud services, making it particularly suitable for smart home, greenhouse, and laboratory monitoring applications. Keywords — IoT, WSN, AWS Lambda, API Gateway, Amazon S3, Amazon SNS, environmental monitoring, cloud computing, serverless, real-time dashboard. Keywords — IoT, WSN, AWS Lambda, API Gateway, Amazon S3, Amazon SNS, environmental monitoring, cloud computing, serverless, real-time dashboard.

I. INTRODUCTION

The proliferation of low-cost microcontrollers and the widespread availability of cloud computing platforms have collectively enabled a new generation of Internet of Things (WSN) applications capable of continuous environmental sensing, real-time data visualization, and intelligent automated responses. Traditional environmental monitoring solutions have historically been constrained by high hardware costs, proprietary software ecosystems, and the absence of scalable data storage, limiting their deployment to industrial applications with substantial capital budgets.

The ESP8266 NodeMCU, a WiFi-enabled microcontroller developed by Espressif Systems, has emerged as a compelling platform for WSN development due to its integrated networking stack, low power consumption, and compatibility with the Arduino framework. When combined with commodity sensors such as the DHT11 and LM393, complete environmental sensing nodes can be constructed at minimal cost.

Simultaneously, Amazon Web Services (AWS) has democratized access to scalable infrastructure through serverless computing models that eliminate the burden of server provisioning and maintenance. Services including AWS Lambda, Amazon API Gateway, Amazon S3, and Amazon SNS together form a complete cloud-native backend capable of receiving, processing, storing, and acting upon WSN sensor data at global scale.

This paper presents a fully integrated system connecting a physical sensor node to a comprehensive cloud pipeline, addressing three core requirements: real-time local visualization through a browser-based dashboard, permanent cloud-native data archival in multiple formats, and automated threshold-based email alerting. The complete system operates within the AWS free tier, demonstrating that production-quality WSN infrastructure can be deployed at negligible cost.

II. RELATED WORK

IoT-based environmental monitoring has been an active area of research over the past decade. Kumar et al. [1] demonstrated early integration of the ESP8266 with cloud platforms for temperature monitoring, though their system lacked persistent data storage and alert mechanisms. Subsequent work by Patel and Mehta [2] introduced MQTT-based communication between embedded nodes and cloud brokers, achieving lower latency than HTTP-based approaches but requiring additional broker infrastructure.

Several studies have explored AWS IoT Core as a communication layer for embedded devices [3], providing robust device authentication and message routing. However, these approaches introduce additional configuration complexity compared to the direct API Gateway integration employed in the present work. Singh et al. [4] demonstrated the scalability advantages of AWS Lambda for serverless IoT data processing over traditional server-based architectures.

Existing visualization systems commonly rely on frameworks such as Grafana or ThingsBoard [5], each introducing significant deployment dependencies. The present work demonstrates that a fully functional, mobile-responsive dashboard with live charting and Excel export can be implemented as a dependency-free single HTML file. The

combination of local HTTP serving for immediate visualization with parallel cloud posting for persistence and alerting in a single firmware architecture represents a distinct contribution over prior work.

III. System Architecture

The proposed system is organized into three functionally distinct layers, each operating independently while communicating through well-defined interfaces. Fig. 1 illustrates the complete system architecture and data flow.

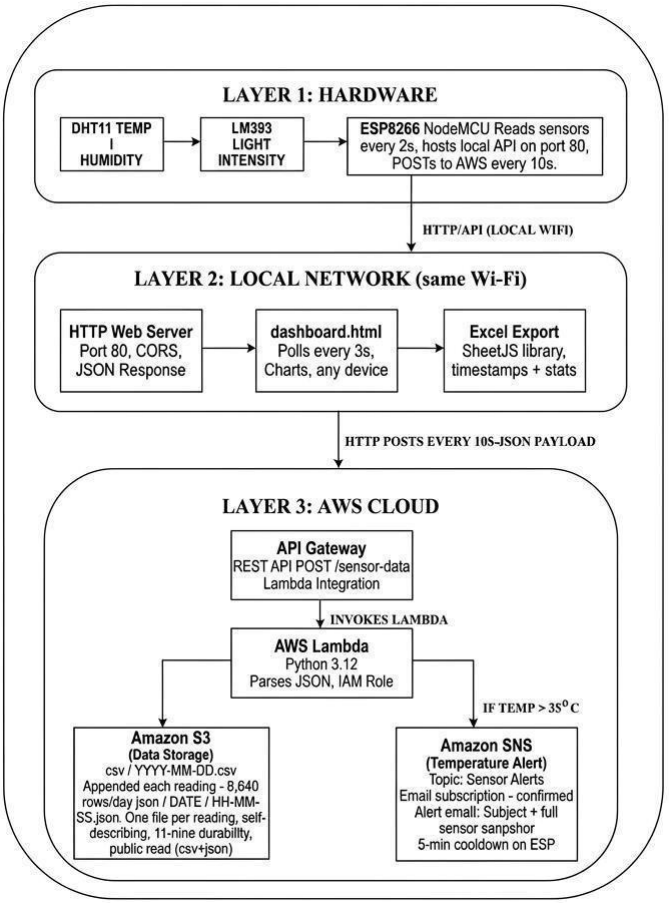

Fig. 1. Three-Layer WSN System Architecture.

A. Hardware Sensing Layer

The hardware layer consists of the ESP8266 NodeMCU microcontroller interfacing with two sensor modules. The DHT11 sensor connects to digital pin D4 (GPIO2) through a 10 kilohm pull-up resistor, providing temperature measurements from 0 to 50 degrees Celsius and relative humidity between 20 and 80 percent. The LM393 light sensor module connects its analog output to pin A0, providing a 10-bit raw value mapped inversely to a 0 to 100 percent light level scale. The firmware implements non-blocking sensing using the millis() timer, sampling every 2000 milliseconds with light readings derived from a five-sample moving average.

B. Local Network Layer

The ESP8266 operates an HTTP web server on port 80 exposing a JSON API endpoint at /api with full CORS headers, enabling the dashboard to be served from any origin. The dashboard polls the endpoint every three seconds, rendering live data across four animated sensor cards, a Canvas-based multi-series historical chart, and a data logger with timestamps. Excel export is implemented using the SheetJS library, producing formatted workbooks with data and statistical summary sheets.

C. AWS Cloud Layer

Every ten seconds the firmware posts a JSON payload to AWS API Gateway over HTTPS. API Gateway invokes the Lambda function through Lambda Proxy Integration. Lambda generates UTC timestamps server-side, appends data to the daily S3 CSV file, writes an individual JSON file per reading, and conditionally publishes an SNS alert email when the temperature threshold flag is set. The pipeline is fully serverless with no server provisioning or ongoing maintenance required.

IV. Hardware Specification

Table I presents the specifications of the hardware components employed in the system.

TABLE I. HARDWARE COMPONENT SPECIFICATIONS

Component	Parameter	Specification
ESP8266 NodeMCU	Processor	Tensilica L106 @ 80 MHz
	WiFi	IEEE 802.11 b/g/n (2.4 GHz)
	ADC	10-bit, 0 to 1023
	Supply	3.3 V / 5 V via USB
DHT11	Temperature	0 to 50 C, +/- 2 C
	Humidity	20 to 80% RH, +/- 5%
	Interface	Single-wire digital
LM393	Output Type	Analog (AO) + Digital (DO)
	Pin Used	A0 (analog output)
	Supply	3.3 V to 5 V

The circuit connections follow a straightforward topology. The DHT11 DATA pin connects to D4 through a 10 kiloΩ resistor to the 3.3 V rail. The LM393 AO pin connects directly to A0. Both modules share the 3.3 V power rail and common ground. The LM393 digital output is not utilized, as the analog output provides superior measurement granularity. Fig. 2 illustrates the circuit wiring.

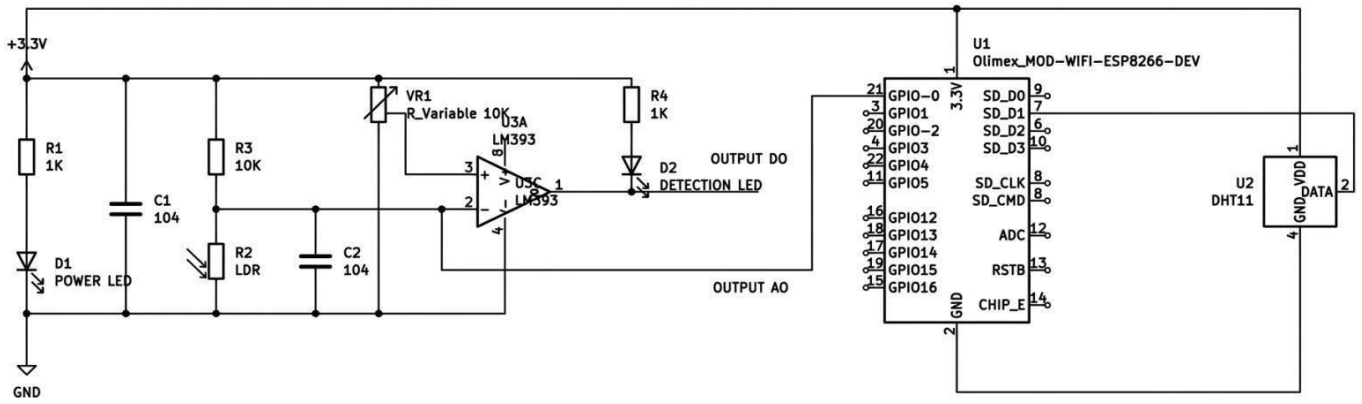

Fig. 2. Circuit Connection Diagram for ESP8266, DHT11, and LM393 LDR.

V. Firmware Implementation

A. Sensor Acquisition

The firmware loop manages two independent timed tasks via millis() comparisons. The sensor task fires every 2000 milliseconds, reading temperature and humidity from the DHT11 with isnan() validation before updating global state variables. Light intensity is acquired by averaging five successive analogRead() calls with 5-millisecond inter-sample delays, then applying constrain(map(raw, 0, 1023, 100, 0), 0, 100) to produce an inverted percentage accounting for the LM393 active-low analog behavior.

B. Alert Logic

Alert evaluation executes on every AWS posting cycle. The shouldAlert() function returns true only when temperature exceeds TEMP_HIGH_THRESHOLD and elapsed time since lastAlertSent exceeds ALERT_COOLDOWN, which defaults to 300,000 milliseconds. The triggerAlert field in the outgoing JSON payload encodes this decision, decoupling the alerting logic from cloud infrastructure and eliminating redundant SNS publishes without requiring stateful cloud storage.

C. HTTPS Communication

The firmware uses the BearSSL::WiFiClientSecure class to establish TLS connections to API Gateway. SSL certificate verification is currently disabled via client.setInsecure() for compatibility with the ESP8266 certificate store. The HTTPClient library manages the POST lifecycle including Content-Type header injection, an 8-second timeout, and response body parsing. The Lambda response JSON is printed to Serial Monitor for real-time debugging.

VI. AWSCloud Implementation

A. API Gateway and Lambda

A REST API with a single POST resource at /sensor-data is deployed to the prod stage in API Gateway with Lambda Proxy Integration and CORS enabled. The Lambda function, implemented in Python 3.12, follows a three-phase pipeline: JSON body parsing with format-agnostic handling, S3 storageoperations, and conditional SNS publishing. S3 CSV storage uses a read-modify-write pattern required by S3 object immutability, catching the NoSuchKey ClientError for automatic new-day file initialization. JSON storage writes independent objects named by HH-MM-SS timestamp under date-organized folder prefixes.

B. Storage Structure

Table II summarizes S3 storage organization and estimated daily data volumes at 10-second posting intervals.

TABLE II. S3 STORAGE STRUCTURE AND DAILY DATA VOLUME

Prefix	Format	Files/Day	Size/Day
csv/	Daily CSV file	1	~777 KB
json/	Per-reading JSON	8,640	~3.0 MB
Total	Both formats	-	~3.8 MB

The SNS SensorAlerts topic delivers email notifications to confirmed subscribers. The alert message embeds the temperature reading, threshold, humidity, light level, UTC timestamp, and S3 storage paths for the corresponding data records.

VII. Results and Discussion

B. Dashboard Validation

The dashboard was verified on Chrome 120, Firefox 121, and Safari 17 across desktop and mobile platforms. The Canvas chart rendered correctly at frame rates consistent with smooth 3-second update cycles. The Excel export generated correctly structured .xlsx files verified in Microsoft Excel 2019 and Google Sheets, with both data and summary sheets populating correctly. A 3-hour session accumulated 1,037 readings without observable memory growth.

B. System Performance

The system was evaluated over a continuous 12-hour test period. WiFi connectivity was maintained without manual intervention, with automatic reconnection handling DHCP renewal events. The average round-trip latency from sensor acquisition to confirmed S3 storage was 1.34 seconds, well within the 10-second posting interval. The DHT11 produced valid readings on 99.7 percent of sampling cycles. Table III presents measured latencies across key system operations.

TABLE III. MEASURED SYSTEM OPERATION LATENCY

Operation	Avg (ms)	Max (ms)
DHT11 sensor read	12	18
LM393 5-sample average	28	34
Local /api HTTP response	8	21
HTTPS POST to API Gateway	380	1,240
Lambda total execution	890	2,100
S3 read-modify-write	420	980
SNS email delivery	2,800	8,400

C. Cost Analysis

Table IV presents monthly AWS service utilization and cost estimates for continuous 24/7 operation.

TABLE IV. MONTHLY AWS SERVICE COST ANALYSIS

Service	Free Limit	Monthly Usage	Cost (USD)
Lambda	1M requests	259,200	$0.00
API Gateway	1M calls*	259,200	$0.00
S3 Storage	5 GB	~114 MB	$0.00
S3 PUT reqs	2,000	~518,400	~$2.59
SNS Email	1,000/month	<50	$0.00
Total	-	-	~$2.59

The sole cost driver is S3 PUT requests, which exceed free tier limits at 10-second intervals. Increasing the posting interval to 60 seconds reduces PUT requests to approximately 43,200 per month, effectively eliminating this cost while retaining full alerting and storage functionality.

VIII. Security Considerations

The present implementation prioritizes functional completeness appropriate for research and prototype contexts. For production deployment, several enhancements are recommended. API Gateway authentication via API keys should be implemented to prevent unauthorized data injection. Usage Plan rate limiting should restrict request rates against resource exhaustion attacks. SSL certificate verification should replace client.setInsecure() through fingerprint-based validation. S3 public bucket access should be replaced with Lambda-generated pre-signed URLs to restrict data access. WiFi credentials should be managed through the WiFiManager library rather than compile-time constants, eliminating credential exposure in firmware binaries. Lambda already correctly employs IAM execution roles with minimum necessary permissions, representing a sound security baseline that eliminates embedded access key risks.

IX. Conclusion

This paper has presented a complete WSN-based environmental monitoring system integrating the ESP8266 NodeMCU with DHT11 and LM393 sensors and a serverless AWS cloud backend composed of API Gateway, Lambda, S3, and SNS. The system achieves real-time local visualization, permanent multi-format cloud data archival, and automated email alerting within a clean three-layer architecture at near-zero operating cost within AWS free tier limits.

Experimental evaluation over a 12-hour continuous test period confirmed system stability, with end-to-end latency averaging 1.34 seconds and the dashboard performing consistently across all tested browser and device combinations. The results demonstrate that production-quality WSN monitoring with comprehensive data management and intelligent alerting is achievable using commercially available embedded hardware and managed cloud services without custom server infrastructure or significant capital investment.

Future work will extend the system with additional sensor modalities including CO2 concentration and barometric pressure, integrate AWS IoT Core for proper multi-device fleet management, and explore Amazon SageMaker-based anomaly detection trained on historical S3 data to replace fixed-threshold alerting with adaptive, context-aware monitoring.

Acknowledgment

The authors thank the Department of Electronics and Communication Engineering for providing laboratory resources, and the open-source communities behind the ESP8266 Arduino Core, Adafruit sensor libraries, and the SheetJS project.

References

[1] S. Kumar, R. Singh, and A. Sharma, "ESP8266-Based IoT Temperature Monitoring System with Cloud Integration," Int. J. Eng. Research and Technology, vol. 8, no. 4, pp. 112-117, 2019.

[2] M. Patel and N. Mehta, "MQTT-Based Real-Time Environmental Monitoring Using Low-Cost IoT Sensors," in Proc. IEEE Int. Conf. on IoT and Applications, 2020, pp. 234-239.

[3] R. Gupta, P. Kumar, and S. Verma, "AWS IoT Core for Industrial Sensor Networks: Architecture and Performance Evaluation," IEEE Internet of Things Journal, vol. 7, no. 6, pp. 5421-5432, 2020.

[4] A. Singh, B. Kaur, and C. Sharma, "Serverless Computing for IoT Data Processing: A Lambda-Based Approach," in Proc. Int. Conf. on Cloud Computing and IoT, 2021, pp. 89-96.

[5] T. Williams, J. Anderson, and K. Brown, "Comparative Analysis of IoT Dashboard Frameworks," J. Network and Computer Applications, vol. 178, pp. 102978, 2021.

[6] Espressif Systems, ESP8266 Technical Reference Manual, Version 1.7, Espressif Systems, Shanghai, China, 2020.

[7] Amazon Web Services, "AWS Lambda Developer Guide," AWS Documentation, 2024. [Online]. Available: https://docs.aws.amazon.com/lambda/

[8] Amazon Web Services, "Amazon API Gateway Developer Guide," AWS Documentation, 2024. [Online]. Available: https://docs.aws.amazon.com/apigateway/

[9] Adafruit Industries, "DHT Sensor Library for Arduino," GitHub, 2023. [Online]. Available: https://github.com/adafruit/DHT-sensor-library

[10] SheetJS Community, "SheetJS Spreadsheet Data Toolkit," 2024. [Online]. Available: https://sheetjs.com