# 

A close-up of a sign

Description automatically generated

MSc Cloud and Network Security

A report submitted in partial fulfilment for the Cloud and Network Security MSc degree.

Author: Bilal Naseer (1902692)  
Lecturer: Francis Morrissey  
Submission Date: 16th May 2025

**Cloud Solutions and Implementation**

Assessment - 2

Implementation of Cloud solutions

**Abstract**

Oceanic Energy Solutions (OES) has developed a real-time cloud platform to compile and assess live data from ocean buoys using Amazon Web Services (AWS). These buoys improve offshore protection and decision-making by tracking critical variables, including temperature and wave height. The solution is provided on all fronts; the system must be scalable, dependable, safe, and reasonably priced. Combining services like AWS Lambda, API Gateway, DynamoDB, S3, Cognito, and SNS, the project runs on a serverless architecture. Built using Chart.js, an online dashboard allows registered users to monitor recent sensor data, get email warnings for severe circumstances, and access. Every layer is built with security in mind; strict IAM rules, encryption, and full CloudWatch and X-Ray monitoring abound. A custom AWS Virtual Private Cloud (VPC) added an extra layer of design security. Lambda functions run within private subnets and interface with DynamoDB, S3, and SNS via VPC endpoints, avoiding public internet access. This architecture conforms to AWS's Well-Architected Framework and Zero Trust guidelines. From a financial standpoint, AWS Budgets incorporated a zero-spending budget. This generates alarms right away once even $0.01 of use is found. It guarantees complete control of cloud expenses and helps the project emphasise environmentally friendly and economical design. This work describes the use of every AWS service and its general coherence. It also evaluates system performance, security, cost, and sustainability and takes significant decisions such as choosing serverless rather than traditional architecture. The final product is a simple, cloud-native fit for effective maintenance and scalability.

Contents

[Introduction 2](#_Toc196468116)

[1. Compute service using AWS Lambda functions 3](#_Toc196468117)

[2. Storage Services using DynamoDB and S3 3](#_Toc196468118)

[3. Networking and Content Delivery 3](#_Toc196468119)

[4. Security Considerations 3](#_Toc196468120)

[5. Sustainability and Cost Efficiency 3](#_Toc196468121)

[6. Critical Evaluation and Architectural Justification 3](#_Toc196468122)

[7. Conclusion and Recommendations 4](#_Toc196468123)

**List of Figures**

[Figure 1: All Lambda functions implemented for the OES solution. 5](#_Toc196468160)

[Figure 2: OES custom VPC overview. 6](#_Toc196468161)

[Figure 3: Custom IAM policy for SNS wave alerts. 7](#_Toc196468162)

[Figure 4: DynamoDB containing BuoyData table. 8](#_Toc196468163)

[Figure 5: Dashboard view of authorised system user. 9](#_Toc196468164)

[Figure 6: S3 bucket storing static webpage file and buoy sensor data backup logs. 10](#_Toc196468165)

[Figure 7: API Gateways for triggering store and retreive buoy data. 11](#_Toc196468166)

[Figure 8: AWS X-Ray trace map for a data retrieval request. 12](#_Toc196468167)

[Figure 9: CloudFront Distribution for the OES solution. 13](#_Toc196468168)

[Figure 10: IAM policies ensuring least-privilege access. 15](#_Toc196468169)

[Figure 11: IAM user with limited access to perform CI/CD tasks. 15](#_Toc196468170)

[Figure 12: Cognito user pool with registered OES users. 16](#_Toc196468171)

[Figure 13: Cognito supports JWT tokens using implicit grant for security. 16](#_Toc196468172)

[Figure 14: API Usage Plan. 17](#_Toc196468173)

[Figure 15: AWS Budgets monitoring setup. 19](#_Toc196468174)

# Introduction

Oceanic Energy Solutions (OES) is investigating how cloud computing may provide real-time ocean condition monitoring. Deployed in the water, buoys gather vital information comprising wave height and temperature. That information must be handled quickly, kept safely, and made available to authorised people. Dangerous situations must be sensed and then automatically set off alerts.

The system was developed on Amazon Web Services (AWS) to reach these targets with a serverless architecture. No servers were, therefore, manually controlled. Instead, data is stored and processed using scalable AWS products, including Lambda, API Gateway, DynamoDB, and S3. Cognito manages user authentication and SNS emails, CloudFront distributes online content, and CloudWatch/X-Ray tracks system performance and health.

The primary concern was security. Tight access policies and encryption are developed for a Virtual Private Cloud (VPC). Lambda functions today operate completely isolated from the public internet inside private subnets. Following the Zero Trust paradigm and matching with AWS's Well-Architected security and dependability rules, they link to DynamoDB, S3, and SNS, utilising VPC Gateway and Interface Endpoints.

Cost control was also considered. A Zero Spend Budget is built using AWS Budgets to set off an alert when the instant expenses surpass $0.01. This maintains the project in the free tier and helps prevent unanticipated expenses. It also promotes financial sustainability, which is critical for production-grade systems.

This work aims to walk through every service used, explain its purpose, and assess how it supports the general architecture. The category-based organisation covers computation, storage, identity, networking, security, monitoring, cost, and sustainability. Many design choices were informed by real-world instances, such as WEGoT's AWS IoT management of over 100,000 IoT sensors (aws, 2024a).

# Compute service using AWS Lambda functions

AWS Lambda assisted in developing the Oceanic Energy Solutions (OES) computing layer. This service was selected as it eliminates the requirement for server management and demand-based automated scaling (Wadia *et al*., 2019).

A screenshot of a computer

AI-generated content may be incorrect.

Figure 1: All Lambda functions implemented for the OES solution.

Figure 1 overviews all the lambda functions (storeBuoyData, getBuoyData, AutoSubscribeToSNS) implemented for the OES solution. Lambda executes tiny amounts of code, processing a user's data or dashboard update request and returning a response whenever it receives it. This increases the system's general cost-effectiveness and efficiency, particularly in erratic traffic cases (Lemine *et al*, 2024). The functions were developed using zip packages and written in Python 3.13. Lambda is strong as it only runs as needed and is idle otherwise. This fits very nicely with AWS's drive for more energy-efficient workloads and sustainable design goals (aws, 2019a; Mathew *et al.*, 2021). The Lambda functions were housed within a Virtual Private Cloud (VPC) to increase security.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2: OES custom VPC overview.

Figure 2 shows the whole design of the OES virtual network. Every service was linked to private subnets, and VPC Gateway Endpoints were utilised rather than public internet access to DynamoDB and S3. Thus, all traffic stays inside the AWS network. For SNS, an interface endpoint was also developed to enable the safe publication of notifications. This configuration conforms to AWS security best practices and supports the Zero Trust paradigm (aws, 2023b; aws, 2025a). Several inline policies were created for relevant IAM roles.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 3: Custom IAM policy for SNS wave alerts.

Figure 3 displays a custom IAM policy allowing safe SNS publication to the BuoyAlerts topic for critical wave height warnings only to registered members. Every operation executes under an IAM role using the least privilege concept. For instance, storeBuoyData cannot access any other AWS services; its sole role is to write to DynamoDB and publish to SNS. Its role is to process incoming sensor data and publish alerts via SNS. AWS's security recommendations call for this fine-grained authorisation mechanism (aws, 2024c). Using Amazon CloudWatch, logs and measurements for every Lambda function were gathered. AWS X-Ray was turned on from API Gateway to Lambda to DynamoDB to track enquiries and see how long each step took. This facilitated performance adjustment and troubleshooting. Throughout testing, cold starts were uncommon (Vahidinia *et al.*, 2020). Python launches fast; hence, the user-facing dashboard has no latency. During peak tests, the system processed arriving and departing queries in tens of milliseconds.

In summary, AWS Lambda gave the backend of the OES platform a lightweight and safe approach. It operated seamlessly with other AWS services, answered fast to real-time data, and did not call for server maintenance. All this was accomplished with low energy use and expenses (Chapin and Roberts, 2020).

# Storage Services using DynamoDB and S3

Amazon DynamoDB and Amazon S3 are used in Oceanic Energy Solutions (OES) storage architecture. Every service has a distinct use. DynamoDB manages live buoy data; S3 keeps the website's front end and raw sensor backups. OES decided to use Amazon DynamoDB to store real-time sensor data. This NoSQL database supports fast, scalable writes and reads, which is advantageous for time-series data, such as regular temperature and wave height updates from buoys (aws, 2016; Dhingra and Mackay, 2024). BuoyData was developed with a buoy\_id as the partition key and timestamp as the sort key.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4: DynamoDB containing BuoyData table.

Figure 4 displays “BuoyData” table with real-time entries of buoy\_id, timestamps, wave height, and temperature. The table maintains simplicity and scalability by running in on-demand capacity mode. It just charges for reads and writes and scales automatically depending on use. Not knowing how much traffic will arrive helps to keep performance consistent and expenses low (aws, n.d.). Additionally, DynamoDB was selected for its zero-maintenance approach—no hardware administration or patching is required. AWS manages everything behind it. To guarantee it is always secure and accessible, the data is kept highly durable and copied throughout several Availability Zones. Using keys controlled by AWS Key Management Service (KMS), all data in DynamoDB is encrypted at rest by default, helping fulfil security and compliance requirements (aws, n.d.). Using Amazon S3, Simple Storage Service, had two key objectives. First, it hosts the OES online dashboard's static frontend, comprising index.html, CSS, JavaScript, and Chart.js code.

A screenshot of a computer monitor

AI-generated content may be incorrect.

Figure 5: Dashboard view of authorised system user.

Figure 5 shows the dashboard (index.html) of the authorised system user, who monitors the wave height and temperature over time. The site is accessed worldwide using Amazon CloudFront, which sits in front of S3 to provide fast and secure service. S3 also saves raw JSON files, including buoy information. This serves as a backup and could facilitate upcoming reporting or data analysis.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6: S3 bucket storing static webpage file and buoy sensor data backup logs.

Figure 6 shows that every sensor reading transmitted to the storeBuoyData Lambda is stored in S3 in a JSON file. The files have buoy ID and timestamp, say "buoy-901\_2025-04-22T08:11:01.json". These backups fit archiving, compliance, or offline processing. DynamoDB is still the primary data source for real-time usage. S3 was selected because it is reasonably priced, secure, and robust. It promises 99.999999999% durability (Sequeira, 2019); hence, the data is protected even if many discs fail. By default, all newly uploaded files in S3 also use AES-256 encryption (aws, 2023a). This ensures strict control of access to the S3 bucket as it is not openly available. CloudFront instead grabs files safely using an origin access identity. This guarantees that only the web frontend may access the data; users cannot circumvent the mechanism and open the S3 URLS immediately. DynamoDB and S3 offer versatile and safe storage layers. While S3 provides robust, long-term storage for static files and backups, DynamoDB gives quick access to organised sensor data for the current dashboard usage. Both services are perfect fits for a serverless system like the OES platform. They scale automatically, are controlled, and provide robust encryption by default.

# Networking and Content Delivery

The OES platform uses Amazon CloudFront and Amazon API Gateway to control data transfers between users and services safely. These programs expedite online content delivery and answer backend enquiries. They, taken together, guarantee that the system is scalable, responsive, and secure. The OES platform's backend opens via the Amazon API Gateway like a front door. Users provide HTTP enquiries, which it verifies and forwards to the appropriate AWS Lambda functions. Two primary API endpoints—one to access data (GET /getData) and one to enter fresh readings (POST /sendData)—were developed for this project (Awad, 2025).

A screenshot of a computer

AI-generated content may be incorrect.

Figure 7: API Gateways for triggering store and retreive buoy data.

Figure 7 displays the API gateways that trigger lambda functions using GET and POST methods. HTTPS locks all traffic passing the API Gateway. Therefore, sensitive data, including user tokens and sensor data, is better protected. Using a Cognito authoriser, the API was additionally locked; hence, only logged-in Amazon Cognito users may access secured routes. Before meeting the demand, API Gateway verifies the user's JWT (JSON Web Token). This offers a high layer of security straight at the edge (aws, 2019b; Amplify, 2024). Additionally, CORS, or cross-origin resource sharing, was made possible. This lets the front end, housed on another domain, safely submit JavaScript queries to the API (Panwar, 2024). AWS X-Ray was turned on for the API to track performance and debug problems.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8: AWS X-Ray trace map for a data retrieval request.

Figure 8 shows the AWS X-Ray trace map when a “getbuoydata” request is made. This lets one follow every demand from the user to API Gateway, Lambda, and beyond. Should something go wrong, it reveals reaction times, status codes, and mistakes. Amazon CloudFront is a content delivery network (CDN) (Aggarwal, 2018). It loads faster for users as it sits in front of static files on S3, including index.html, JavaScript, pictures, and styles.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 9: CloudFront Distribution for the OES solution.

Figure 9 shows the CloudFront distribution used to serve the frontend securely. By storing files at AWS edge sites, CloudFront enables users to grab material from the closest geographic server rather than waiting for a far-off S3 bucket (AntStack, 2019). CloudFront increases security as well. Only CloudFront, using an origin access identity (OAI), may access the S3 bucket; it is kept private (aws, n.d.). This stops anyone from circumventing the website and downloading material straight from the bucket. All material is sent over HTTPS using a bespoke SSL certificate kept under AWS Certificate Manager (Soni, 2021). This guarantees that data is encrypted in transit and helps guard consumers from man-in-the-middle attacks. Future improvements like WAF (Web Application Firewall) or AWS Shield can assist in screening dangerous enquiries and thwarting DDoS attacks. They also support CloudFront—additionally, caching results in fewer backend loads. Most stationary files do not change much. Hence, CloudFront can swiftly provide them from the cache without traversing the origin every time. Time-to-live (TTL) values were changed to strike a balance between long-term storage (for assets like logos or libraries) and quick updates (for files like index.html).

CloudFront and API Gateway, taken together, provide a strong networking configuration. CloudFront effectively and securely provides the frontend; API Gateway manages and safeguards the backend API. Both offerings follow cloud security best practices, provide encryption, and scale automatically. Whether users are simply looking at the dashboard or are contributing fresh data, this guarantees a safe and seamless connection with the OES platform.

# Security Considerations

Throughout the OES cloud platform's development, security dominated all else. The system manages user information and environmental data; hence, it must be constructed with extraordinary precautions (Elghoul *et al.*, 2023). Security was implemented at various tiers, from access control to encryption, identity management, and monitoring. The system controls rights using AWS Identity and Access Management (IAM). Every Lambda function has an IAM purpose all by itself. Following the least privilege concept, these roles allow each to carry out the required tasks, nothing more. The storeBuoyData feature, for instance, only writes messages to SNS and the DynamoDB table. It lowers risk if something goes wrong, as it cannot access unrelated resources or read from other services. IAM policies were written carefully to define what each role can do (aws, 2024b; Singh *et al.*, 2023).

A screenshot of a computer

AI-generated content may be incorrect.

Figure 10: IAM policies ensuring least-privilege access.

Figure 10 highlights the IAM policies applied to the storeBuoyData function, providing least-privilege access to DynamoDB, S3, SNS, and logging services. In this work, even the CI/CD workflow set up using GitHub Actions uses an IAM account with limited rights. This includes uploading files to S3 and invalidating the CloudFront cache. This keeps sensitive AWS environment components hidden during automation.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 11: IAM user with limited access to perform CI/CD tasks.

Figure 11 shows the IAM role “github\_deployer”, which gives limited permission to invalidate the CloudFront cache, write to S3, update Lambda functions, and access DynamoDB. User sign-up and login using Amazon Cognito helped to regulate who could use the system.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 12: Cognito user pool with registered OES users.

Figure 12 displays two registered users who can access the OES monitoring dashboard. Cognito oversees user information, including names, emails, and hashed passwords. It also emails verification notes during registration (aws, 2025b). Users log in and get JWT tokens, which they incorporate into backend queries.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 13: Cognito supports JWT tokens using implicit grant for security.

Figure 13 displays the Amazon Cognito client page for OES, which provides secure authentication using JWT tokens via the implicit Grant flow. By verifying these tokens, API Gateway adds a robust authorising layer at the network edge and guarantees that only authorised users may access secured paths (Gontrum, 2019). Cognito backs account recovery, safe password practices, and multi-factor authentication (MFA). It complies with essential standards, including HIPAA, PCI DSS, SOC, and ISO 27001 (aws, n.d.).

A screenshot of a computer

AI-generated content may be incorrect.

Figure 14: API Usage Plan.

Figure 14 shows the API usage plan implemented as a security layer using an API key for the OES solution. API keys and usage plans were initially configured to regulate access control and throttling. However, the "API Key Required" option in API Gateway was disabled due to the adoption of Amazon Cognito for JWT-based user authentication. This is because it cannot be combined with Cognito authorisations, rendering JWT the primary mechanism for secure, authenticated access.

All the data in the system is encrypted both at rest and in transit.

**• At rest:** AWS-managed KMS keys let DynamoDB and S3 automatically encrypt all data. S3 Default now sets this default for all new objects (aws, 2023a). Without further configuration, DynamoDB also uses built-in encryption (aws, n.d.).

**• In transit:** Everything operates via HTTPS. CloudFront delivers the page under an SSL certificate from the AWS Certificate Manager. Additionally, API Gateway enforces HTTPS for every call. Using encrypted channels via the AWS SDK, even internal AWS services, such as Lambda and DynamoDB, communicate.

This end-to-end encryption facilitates compliance with data protection rules and helps guard private information. AWS CloudWatch and AWS X-Ray were used to track system activity.

• Lambda function runs, including input, output, and errors, are captured in CloudWatch Logs. Debugging and verifying if the data was accurately stored depends on this.

• CloudWatch Metrics offers notifications on function performance, together with charts.

• X-ray tracks end-to-end requests, enabling Lambda, API Gateway, and other services to identify failures and bottlenecks.

X-ray, for instance, made it clear how long every system component responded, and CloudWatch verified whether alert messages were delivered effectively. Additionally, AWS Cloudtrail was turned on to monitor account-level activities. Useful for audits or security investigations, it records events such as Cognito user sign-ups or IAM modifications (Routavaara, 2020).

Overall, the OES platform's security is tiered and thoroughly considered. While Cognito oversees who may access each system component, IAM regulates what each can do. All data is encrypted, and monitoring tools track events in real-time. These characteristics guarantee safety, compliance, and simplicity of maintenance even as the system grows.

# Sustainability and Cost Efficiency

The OES cloud platform was meant to be ecologically friendly, reasonably affordable, and technically robust. Using AWS's serverless and managed services helps the system function when needed and consume fewer resources generally, therefore helping to lower cloud fees and energy waste. Maintaining the sustainability of the platform depended on its being built with serverless architecture (Patros *et al*., 2021; Akour and Alenezi, 2025). Services that are not running continuously include AWS Lambda, DynamoDB, and S3. They only turn on when work is to be done, such as handling fresh data or customer enquiries. This prevents the energy loss caused by idle computers. AWS (2021) claims they remove unneeded infrastructure, and serverless architectures are perfect for maximising resource efficiency. Lambda, for instance, costs for the precise time a function runs. Without data processing, no energy or cost is used (aws, 2024b, Bagai, 2024). AWS states that its data centres are up to 3.6 times more energy-efficient than usual on-site infrastructure and is dedicated to running 100% renewable energy by 2025 (aws, n.d.). This implies that a system such as OES has a smaller carbon footprint than running actual servers, the more cloud services it consumes. Maintaining the AWS Free Tier was a significant objective of this project. Using AWS Budgets, a zero-spend budget was created to help with management.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 15: AWS Budgets monitoring setup.

Figure 15 shows a budget set at $1.00 to track AWS usage. Even for a single Lambda invocation or S3 request, it was set to generate an email alert when expenses were above $0.01. This basic alarm system guaranteed total control over resource use and helped to prevent any unanticipated costs during testing. One of the main foundations of the AWS Well-Architected Framework is cost-aware architecture, which is also reflected here (Adamson, 2023). Most services, including Lambda, API Gateway, DynamoDB, and CloudFront, have reasonable free levels, such as:

* DynamoDB provides up to 25 GB of free storage.
* Lambda enables 1 million free requests per month.
* CloudFront allows 50 GB of data to be transported, and 2 million free requests are made annually (Hava, 2022).

The OES platform has traffic levels and limited data amounts within these constraints. This makes it perfect for early-stage real-world projects as well as student-level deployments. S3 lifecycle regulations help the system also promote future cost optimisation (aws, 2022). These can erase old buoy data after a specified period or automatically migrate it to less expensive storage classes like S3 Glacier. Time to Live (TTL), which eliminates outdated entries without human cleaning, is another supported capability of DynamoDB (Tsubouchi *et al.,* 2019). These characteristics assist in maintaining a clean, efficient, and ecologically friendly system by saving money and lowering storage overhead.

The OES cloud design shows that excellent performance does not have to be accompanied by significant environmental or financial costs. AWS's built-in budget controls and serverless capabilities help the system maintain low energy usage and cheap expenditures. Though it will not waste resources during calm times, it is meant to scale as needed (Lavoie *et al*, 2019). The platform is a good illustration of current, responsible cloud architecture as it mixes efficiency and sustainability.

# Critical Evaluation and Architectural Justification

The OES platform was created based on real-time requirements, security considerations, and keeping costs and maintenance low. Even if the ultimate answer satisfied all main goals, evaluating why these decisions were made, what other choices were considered, and where development is needed is crucial.

A computer screen shot of a computer

AI-generated content may be incorrect.

Figure 16: Architecture Diagram of the OES solution.

Figure 16 shows an architectural diagram that exhibits the entire cloud-based data flow of the OES platform. It starts (from left) with sensor data recorded from sea buoys via API Gateway into AWS Lambda functions maintained within a secure VPC. The system records raw logs in S3, analyses and stores buoy data in DynamoDB and generates SNS notifications when wave heights surpass safe limits (3.0). The dashboard is accessible to a Cognito-authenticated user via CloudFront, which hosts the static front end on S3. In contrast, GitHub CI/CD guarantees automated frontend deployment, while JWT tokens guarantee secure API access (Amplify, 2024). CloudWatch + IAM enforces observability and security best practices, and AWS Budgets tracks expenses. Among the most crucial choices the project made was deciding on a serverless design. Because they do not call for on-premises server setup, patching, or scalability, services such as AWS Lambda, DynamoDB, and API Gateway were selected. This lets the system concentrate on the real functionality: data intake, processing, and visualisation, without infrastructure concerns. By comparison, conventional choices like Amazon EC2 would have needed monitoring, scaling decisions, and continual upgrades. Without clearly benefiting from this use case, the complexity and expense would have increased without payback. Ideal for workloads that vary over time, Lambda also scales automatically and costs just for what is needed (Patterson, 2019). Lambda's cold start delay, a little gap when a function executes following inactivity, is the only obvious drawback. This delay seemed reasonable for our dashboard-style program, where most calls are read-only and non-urgent. Performance tests revealed that even cold starts finish in a few hundred milliseconds.

An alternative to API Gateway for data intake may have been AWS IoT Core. IoT Core uses MQTT and other protocols expressly for device connectivity (Slootjes, 2022). However, since the project runs buoy data over HTTP, utilising API Gateway and Lambda was easier and more appropriate for the present phase. IoT Core might still be added should actual buoys be placed at sea. Regarding data storage, one additional consideration is DynamoDB (NoSQL) and Amazon RDS (relational database). DynamoDB's flexibility, speed, and zero-maintenance architecture fit time-series sensor values better than RDS, even if it might provide more sophisticated searches or joins (QA, 2023). Since the data of every buoy is independent, relational tools like multi-table joins or foreign keys were not needed. Chart.js, a lightweight JavaScript charting tool (w3schools, n.d.), was used throughout the dashboard building. For a basic, responsive UI, this performed nicely. Extra capabilities may have come from alternatives like Amazon QuickSight or more sophisticated systems like React. Still, they would have become more complicated for a small-scale undertaking and might not be reasonably priced. Chart.js, vanilla HTML, and JavaScript keep the system small and straightforward. These tools may be modified or enhanced to allow more sophisticated user interactions or analytics as the project develops (Leon, 2025). Generally, the security design performed effectively. Cognito managed logins consistently, IAM roles were precisely specified, and data was encrypted at all levels. Including multi-factor authentication (MFA) would increase user security even more. Using AWS WAF (Web Application Firewall) with CloudFront will also help to stop typical threats like SQL injection or cross-site scripting. Additionally, successful cost containment was achieved. Most services stayed in the AWS Free Tier, and the Zero Spend Budget helped keep consumption under control. Setting up CloudWatch alarms for unexpected activities or imposing use limitations in API Gateway might help enhance financial control.

The present system is housed in one AWS region; hence, some risk is involved if that region fails. Resilience might be enhanced by additions to DynamoDB Global Tables, multi-region S3 replication, or Route 53 failover routing. These would enable the platform to keep operating even in cases where a whole area becomes inaccessible. Also enhanced might be real-time updates. Users should currently refresh the dashboard to view fresh data. Live updates made possible by WebSocket or AWS AppSync convey data to consumers as it comes, negating the need for refreshment. Should analytics take the front stage, data kept in DynamoDB might be transferred to S3 and examined using AWS Glue or Amazon Athena. This would facilitate trend analysis or large-scale data mining without interfering with live performance. All things considered, the OES architecture selected matched the situation exactly. It is cost-effective, scalable, safe, and performance-wise efficient. While AWS APIS helped tie things together with little setup, serverless services made development faster and easier. Though there is always room for improvement, the present solution offers a solid basis for further development and creativity.

# Conclusion and Recommendations

The Oceanic Energy Solutions (OES) platform shows how contemporary cloud computing can efficiently solve real-time problems in a scalable, ecologically friendly, and safe manner. Amazon Web Services (AWS) constructed a serverless architecture without running any physical infrastructure to gather ocean sensor data, send automatic alarms, and show it all on an interactive dashboard. Every element of the solution—security, efficiency, minimal running costs—was painstakingly chosen and set to support data input and monitoring. Lambda, API Gateway, DynamoDB, S3, and Cognito satisfied the speed and privacy needs. Network isolation, encryption, and identity management were the best security mechanisms. VPC settings and budget monitoring were taken to improve reliability and control. The platform must stay cheap and light, qualifying it for academic demonstrations and later manufacturing applications. The flexibility of the system allows it to grow and meet evolving needs, such as including extra sensors or using sophisticated analytics. Ultimately, the OES project demonstrates how effectively well-designed cloud solutions can deliver actual value in real time, supporting technological objectives and more general environmental obligations. Given its solid basis, the platform is positioned to develop going forward and significantly support the renewable energy industry.

Although the system is functional and stable, there are certain areas where development may strengthen it even more:

**1.    Add Multi-Region Redundancy:** Everything flows in one area right now. Investigating multi-region deployment, such as copying DynamoDB using Global Tables or maintaining a backup S3 bucket with cross-region replication, would help boost availability.

**2.    Implement Real-Time Dashboards:** The dashboard currently updates manually. Without restarting the website, using WebSockets, AppSync, or AWS IoT Core might provide live updates, pushing fresh data to consumers as it comes (serverless, 2024).

**3.    Improve Security with WAF and MFA:** Although the current security is good, AWS WAF in front of CloudFront and API Gateway to stop typical attacks like SQL injection or brute force efforts might improve it for Cognito users (Nithya and Apinayaprethi, 2023; Servifyspheresolutions, 2024).

**4.    Add Logging for Alerts:** Although alert emails are verified via the AWS interface, SNS delivery logs might provide overtime tracking of email success or failure. For audit needs, these logs could be sent to S3 or CloudWatch.

**5.    Build DevOps Pipelines for Backend Code**: The front end deploys GitHub Actions. A future enhancement would be using infrastructure-as-code technologies like AWS SAM or CDK to automate Lambdas, API Gateway, and other backend resources.

**6.    Explore Analytics and Reporting:** Exporting DynamoDB measurements to S3 data lakes and querying them with Athena will provide deeper insights, such as long-term wave trends or seasonal temperature fluctuations as data develops. These realisations could help with energy planning for activities near oceans.

**References**

Adamson, C. (2023). *Using AWS Budgets to Manage Cloud Costs - Christopher Adamson - Medium*. [online] Medium. Available at: <https://medium.com/@christopheradamson253/using-aws-budgets-to-manage-cloud-costs-b15339d8ece0> [Accessed 23 April 2025].

Aggarwal, M. (2018). *AWS MasterClass*. [online] Packt Publishing. Available at: <https://learning-oreilly-com.ezproxy.bolton.ac.uk/course/aws-masterclass-storage/9781788992930/> [Accessed 21 April 2025].

Akour, M. and Alenezi, M. (2025). Reducing Environmental Impact with Sustainable Serverless Computing. *Sustainability*, [online] 17(7), pp.2999–2999. doi: <https://doi.org/10.3390/su17072999> [Accessed 21 April 2025].

Amplify (2024). *Tokens and credentials - AWS Amplify Gen 2 Documentation*. [online] Amplify.aws. Available at: <https://docs.amplify.aws/react/build-a-backend/auth/concepts/tokens-and-credentials/> [Accessed 22 April 2025].

AntStack (2019). *What Is AWS Cloudfront? Here’s Everything You Need to Know | AntStack - Full-Stack Serverless Company*. [online] Antstack.com. Available at: <https://www.antstack.com/guides/amazon-cloudfront-full-guide/> [Accessed 23 April 2025].

Awad, J.W. (2025). *AWS API Gateway (REST API) Deep Dive | By Joud W. Awad | Medium*. [online] Medium. Available at: <https://medium.com/@joudwawad/aws-api-gateway-deep-dive-rest-apis-5ae16a326b3a> [Accessed 22 April 2025].

aws (2016). *Analyze a Time Series in Real Time with AWS Lambda, Amazon Kinesis and Amazon DynamoDB Streams | Amazon Web Services*. [online] Amazon Web Services. Available at: <https://aws.amazon.com/blogs/big-data/analyze-a-time-series-in-real-time-with-aws-lambda-amazon-kinesis-and-amazon-dynamodb-streams/> [Accessed 21 April 2025].

aws (2019a). *AWS Lambda – Serverless Compute*. [online] Amazon Web Services, Inc. Available at: <https://aws.amazon.com/lambda/> [Accessed 20 April 2025].

aws (2023a). *Amazon S3 now automatically encrypts all new objects*. [online] Amazon Web Services, Inc. Available at: <https://aws.amazon.com/about-aws/whats-new/2023/01/amazon-s3-automatically-encrypts-new-objects/> [Accessed 21 April 2025].

aws (2024a). *AWS IoT Solutions: Customer Success Stories - AWS*. [online] Amazon Web Services, Inc. Available at: <https://aws.amazon.com/awstv/watch/57581b62142/> [Accessed 23 April 2025].

aws (2024b). *AWS Lambda – Pricing*. [online] Amazon Web Services, Inc. Available at: <https://aws.amazon.com/lambda/pricing/> [Accessed 24 April 2025].

aws (2025a). *AWS Prescriptive Guidance Embracing Zero Trust: A strategy for secure and agile business transformation*. [online] *docs.aws.amazon.com*. aws. Available at: <https://docs.aws.amazon.com/pdfs/prescriptive-guidance/latest/strategy-zero-trust-architecture/strategy-zero-trust-architecture.pdf> [Accessed 24 April 2025].

aws (n.d.). *DynamoDB Encryption at Rest - Amazon DynamoDB*. [online] docs.aws.amazon.com. Available at: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/EncryptionAtRest.html> [Accessed 22 April 2025].

aws (n.d.). *DynamoDB throughput capacity - Amazon DynamoDB*. [online] docs.aws.amazon.com. Available at: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/capacity-mode.html> [Accessed 21 April 2025].

aws (n.d.). *Energy Transition | Amazon Web Services*. [online] Amazon Web Services, Inc. Available at: <https://aws.amazon.com/energy/transition1/> [Accessed 22 April 2025].

aws (n.d.). *Features | Amazon Cognito | Amazon Web Services (AWS)*. [online] Amazon Web Services, Inc. Available at: <https://aws.amazon.com/cognito/features/> [Accessed 23 April 2025].

aws (2019b). *What Is Amazon API Gateway? - Amazon API Gateway*. [online] Amazon.com. Available at: <https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html> [Accessed 23 April 2025].

aws (2021). *Optimizing your AWS Infrastructure for Sustainability, Part I: Compute | AWS Architecture Blog*. [online] aws.amazon.com. Available at: <https://aws.amazon.com/blogs/architecture/optimizing-your-aws-infrastructure-for-sustainability-part-i-compute/> [Accessed 24 April 2025].

aws (2022). *Optimize storage costs with new Amazon S3 Lifecycle filters and actions | AWS Storage Blog*. [online] aws.amazon.com. Available at: <https://aws.amazon.com/blogs/storage/optimize-storage-costs-with-new-amazon-s3-lifecycle-filters-and-actions/> [Accessed 23 April 2025].

aws (n.d.). *Restricting access to Amazon S3 content by using an origin access identity (OAI) - Amazon CloudFront*. [online] docs.aws.amazon.com. Available at: <https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/private-content-restricting-access-to-s3.html> [Accessed 24 April 2025].

aws (2023b). *What Is Amazon VPC? - Amazon Virtual Private Cloud*. [online] Amazon.com. Available at: <https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html> [Accessed 23 April 2025].

aws (2024c). *Security Best Practices in IAM - AWS Identity and Access Management*. [online] docs.aws.amazon.com. Available at: <https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html> [Accessed 22 April 2025].

aws (2025b). *What Is Amazon Cognito? - Amazon Cognito*. [online] docs.aws.amazon.com. Available at: <https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html> [Accessed 23 April 2025].

Bagai, R. (2024). Comparative Analysis of AWS Model Deployment Services. *arXiv.org*, [online] 72(5). doi: <https://doi.org/10.14445/22312803/IJCTT-V72I5P113> [Accessed 22 April 2025].

Chapin, J. and Roberts, M. (2020). *Programming AWS Lambda*. first ed. [online] *Google Books*. USA: O’Reily. Available at: <https://books.google.co.uk/books?hl=en&lr=&id=HdPXDwAAQBAJ&oi=fnd&pg=PR4&dq=Programming+AWS+Lambda++John+Chapin> [Accessed 21 April 2025].

Dhingra, A. and Mackay, M. (2024). *Amazon DynamoDB - The Definitive Guide*. [online] Packt Publishing Ltd. Available at: <https://learning-oreilly-com.ezproxy.bolton.ac.uk/library/view/amazon-dynamodb/9781803246895/> [Accessed 22 April 2025].

Elghoul, M.K., Bahgat, S.F., Hussein, A.S. and Hamad, S. (2023). Management of medical record data with multi-level security on Amazon Web Services. *SN Applied Sciences*, [online] 5(11). doi: <https://doi.org/10.1007/s42452-023-05502-9> [Accessed 25 April 2025].

Gontrum, J. (2019). *JWT Authentication with FastAPI and AWS Cognito - Johannes Gontrum - Medium*. [online] Medium. Available at: <https://gntrm.medium.com/jwt-authentication-with-fastapi-and-aws-cognito-1333f7f2729e> [Accessed 23 April 2025].

Hava, T. (2022). *Amazon AWS Free Tiers and Service List*. [online] Hava.io. Available at: <https://www.hava.io/blog/aws-free-tiers-and-services> [Accessed 21 April 2025].

Lavoie, S., Garant, A. and Petrillo, F. (2019). *Serverless architecture efficiency: an exploratory study*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1901.03984v1> [Accessed 21 April 2025].

Lemine, M., Sad, B.C. and Shuwail, A. (2024). *Comprehensive Review of Performance Optimization Strategies for Serverless Applications on AWS Lambda*. [online] arXiv.org. Available at: <https://arxiv.org/abs/2407.10397v1> [Accessed 22 April 2025].

Leon (2025). *Building Interactive Dashboards with Chart.js: Visualizing Data Using the Chart.js Library - Savannah Software Solutions*. [online] Savannah Software Solutions. Available at: <https://savannahsoftwaresolutions.co.ke/building-interactive-dashboards-with-chart-js-visualizing-data-using-the-chart-js-library/> [Accessed 23 April 2025].

Mathew, A., Andrikopoulos, V. and Blaauw, F.J. (2021). Exploring the cost and performance benefits of AWS step functions using a data processing pipeline. *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing*, [online] pp. 7–16. Doi: <https://doi.org/10.1145/3468737.3494084> [Accessed 24 April 2025].

Nithya, S. and Apinayaprethi, K.N. (2023). Content delivery network optimization & analysis using Amazon Web Services. In: *2023 2nd International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA)*. [online] 2023 2nd International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA). New York, USA: IEEE, pp. 1–6. Available at: <https://ieeexplore-ieee-org.ezproxy.bolton.ac.uk/abstract/document/10199819> [Accessed 24 April 2025].

Panwar, V. (2024). *Setting Up CORS and Integration on AWS API Gateway Using CloudFormation*. [online] dzone.com. Available at: <https://dzone.com/articles/setting-up-cors-and-integration-on-aws-api-gateway> [Accessed 23 April 2025].

Patros, P., Spillner, J., Papadopoulos, A.V., Varghese, B., Rana, O., Dustdar, S. and Dustdar, S. (2021). Toward Sustainable Serverless Computing. *IEEE Internet Computing*, [online] 25(6), pp.42–50. doi: <https://doi.org/10.1109/mic.2021.3093105> [Accessed 22 April 2025].

Patterson, S. (2019). *Learn AWS Serverless Computing*. 1st ed. [online] *Google Books*. packt. Available at: <https://books.google.co.uk/books?hl=en&lr=&id=hiLHDwAAQBAJ&oi=fnd&pg=PP1&dq=Benefits+of+AWS+Lambda&ots=ysoc51fzaZ&sig=UrJdbgU345-wkG-igmAV9REDvi4&redir_esc=y#v=onepage&q=Benefits%20of%20AWS%20Lambda&f=false> [Accessed 22 April 2025].

QA (2023). *Amazon RDS vs DynamoDB: 12 Differences You Should Know*. [online] Qa.com. Available at: <https://www.qa.com/resources/blog/amazon-rds-vs-dynamodb-12-differences/> [Accessed 24 April 2025].

Routavaara, I. (2020). *Security monitoring in AWS public cloud*. [online] *Security monitoring in AWS public cloud*. theseus. Available at: <https://www.theseus.fi/bitstream/handle/10024/341640/Opinnaytetyo_Routavaara_Ilkka.pdf?sequence=2> [Accessed 23 April 2025].

Sequeira, A. (2019). *AWS certified solutions architect : associate (SAA-CO1) cert guide*. [online] Indianapolis: Pearson It Certification. Available at: <https://www.google.co.uk/books/edition/AWS_Certified_Solutions_Architect_associ/6bL2wQEACAAJ?hl=en> [Accessed 23 April 2025].

serverless (2024). *AWS AppSync - The Ultimate Guide*. [online] Serverless.com. Available at: <https://www.serverless.com/guides/aws-appsync> [Accessed 25 April 2025].

Servifyspheresolutions (2024). *What is AWS WAF? - Servifyspheresolutions - Medium*. [online] Medium. Available at: <https://medium.com/@servifyspheresolutions/what-is-aws-waf-56de2b242281> [Accessed 25 April 2025].

Singh, C., Thakkar, R. and Warraich, J. (2023). IAM Identity Access Management—Importance in Maintaining Security Systems within Organizations. *European Journal of Engineering and Technology Research*, [online] 8(4), pp.30–38. doi: <https://doi.org/10.24018/ejeng.2023.8.4.3074> [Accessed 23 April 2025].

Slootjes, R. (2022). *API Gateway Websockets vs IoT Core*. [online] DEV Community. Available at: <https://dev.to/slootjes/api-gateway-websockets-vs-iot-core-1me5> [Accessed 25 April 2025].

Soni, A. (2021). *Understanding and using Amazon CloudFront CDN*. [online] Mindful Engineering. Available at: <https://medium.com/mindful-engineering/today-we-will-learn-about-cloudfront-690bf3a8819a> [Accessed 24 April 2025].

Tsubouchi, Y., Wakisaka, A., Hamada, K., Matsuki, M., Abe, H. and Matsumoto, R. (2019). HeteroTSDB: An Extensible Time Series Database for Automatically Tiering on Heterogeneous Key-Value Stores. *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, [online] pp.264–269. doi: <https://doi.org/10.1109/compsac.2019.00046> [Accessed 23 April 2025].

Vahidinia, P., Farahani, B. and Aliee, F.S. (2020). Cold Start in Serverless Computing: Current Trends and Mitigation Strategies. In: *IEEE*. [online] 2020 International Conference on Omni-layer Intelligent Systems (COINS). Barcelona, Spain : IEEE. Available at: <https://ieeexplore-ieee-org.ezproxy.bolton.ac.uk/abstract/document/9191377> [Accessed 22 April 2025].

w3schools (n.d.). *Chart.js*. [online] www.w3schools.com. Available at: <https://www.w3schools.com/ai/ai_chartjs.as> [Accessed 25 April 2025].

Wadia, Y.R., Udell, R., Chan, L. and Gupta, U. (2019). *Implementing AWS : design, build, and manage your infrastructure : leverage AWS features to build highly secure, fault-tolerant, and scalable cloud environments*. 1st ed. [online] Birmingham, Uk: Packt Publishing. Available at: <https://learning-oreilly-com.ezproxy.bolton.ac.uk/library/view/implementing-aws-design/9781788835770/> [Accessed 21 April 2025].

**Appendices**

**Appendix A - VPC**

A screenshot of a computer

AI-generated content may be incorrect.A white background with text

AI-generated content may be incorrect.

**Appendix B – Lambda**

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

**Appendix C – IAM Roles, permissions, and policies**

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

**Appendix D – SNS**

A screenshot of a computer

AI-generated content may be incorrect.

**Appendix E – Budget**

A screenshot of a computer

AI-generated content may be incorrect.

**Appendix F – API Gateway & key**

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect. **Appendix G – CloudFront**

A screenshot of a computer

AI-generated content may be incorrect.

**Appendix H – Cognito**

A screenshot of a computer

AI-generated content may be incorrect.

**Appendix I – CloudWatch**

A screenshot of a computer

AI-generated content may be incorrect.

**Appendix J -GitHub Action CI/CD**

A screenshot of a computer

AI-generated content may be incorrect.