**CSCI 5411 Milestone 2 Project Report**

**Name:** Vaibhavkumar Patel  
**Banner ID:** B01006432  
**Project Title:** Smart Facility Environmental Monitoring Dashboard

1. **Final Architecture and System Design**

**A diagram of a computer

AI-generated content may be incorrect.**

The Smart Facility Environmental Monitoring Dashboard is a cloud-based IoT solution built on AWS to monitor temperature, humidity, CO₂, and AQI data. Simulated sensor data is ingested using MQTT, processed in real time, stored in both raw and structured formats, and visualized via a Grafana dashboard. The system is designed for secure, scalable, and cost-effective operation using services allowed in the AWS Learner Lab.

**2. Service Selection and Configuration**

|  |  |  |
| --- | --- | --- |
| **AWS Service** | **Function** | **Justification** |
| AWS IoT Core | MQTT-based IoT data ingestion | Provides native MQTT broker and device message routing. Enables secure, scalable communication between IoT simulator and backend infrastructure. |
| AWS Lambda | Data transformation, routing | Event-driven compute service perfect for processing real-time IoT streams without provisioning infrastructure. Ensures elasticity and low cost. |
| Amazon S3 | Archive raw sensor data | Durable object storage with lifecycle management. Ideal for storing time-series sensor payloads with auto-deletion to optimize costs. |
| Amazon DynamoDB | Store processed data | Fully managed NoSQL service supporting real-time reads/writes for dashboard use. On-demand capacity mode scales with incoming data rate. |
| Amazon SNS | Alert developers | Lightweight pub/sub service used to notify admins of system faults or data threshold breaches. Ensures proactive intervention. |
| Amazon CloudWatch | Logging, alarms, metrics | Enables end-to-end observability by collecting Lambda logs, defining alarms, and tracking custom system metrics. |
| Amazon VPC/Subnets | Network isolation | Ensures all services operate within isolated environments, supporting secure routing and inter-service communication across private subnets. |
| VPC Endpoints | Private AWS service access | Allows Lambda and EC2 to access S3 and DynamoDB securely over AWS backbone without traversing the public internet. |

Removed:

1. AWS kinesis : Not available in AWS free tier account.
2. GRAFANA: Integrating the Grafana based custom dashboard with the one click automation file using the terraform was more time consuming. For this simple use case, I replaced it with the CloudWatch based dashboard for the insights from the data.

**3. Requirements Fulfillment**

**Functional:**

Simulate IoT sensors producing temperature, humidity, AQI, CO₂.

A screenshot of a computer

AI-generated content may be incorrect.

• Ingest sensor data via MQTT through AWS IoT Core.

A screenshot of a computer

AI-generated content may be incorrect.

• Use AWS Lambda to process, enrich, and route data.

A screenshot of a computer

AI-generated content may be incorrect.

Store raw sensor data in S3 with a 34-hour lifecycle deletion policy.

A screenshot of a computer

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

AI-generated content may be incorrect.

Store processed sensor data in DynamoDB for efficient dashboard querying.

A screenshot of a computer

AI-generated content may be incorrect.

• Generate SNS alerts on critical AWS resource events.

A screenshot of a computer

AI-generated content may be incorrect.

Visualize trends and real-time conditions

A screenshot of a computer

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

AI-generated content may be incorrect.

Monitor application health and data pipeline via AWS CloudWatch dashboards

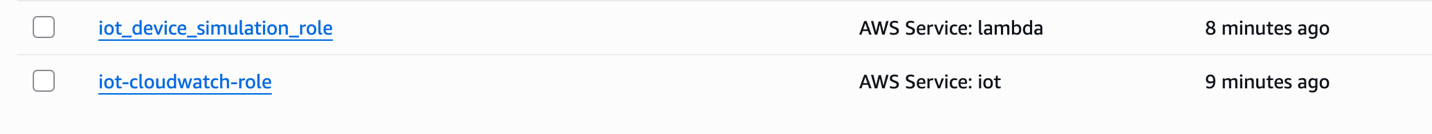
and alarms.

A screenshot of a computer

AI-generated content may be incorrect.

**Non-Functional:**

* **Security**: IAM roles, VPC isolation, Policies with least privileges



* **Scalability**: Serverless design, managed services.
* **Performance**: Low-latency dashboard (<2s).
* **Reliability**: CloudWatch alarms, Central logging for the logs, SNS based alerts for the critical failures in the system.
* **Cost**: on-demand usage, Event-driven architecture.
* **Disaster Recovery**: Stateless processing + Terraform redeployment.

**4. Monitoring and Logging**

* Configured the CloudWatch based logs from the lambda functions.
* Also, exported custom critical matrices to get the pin point in the system failure.
* Crafted a System health dashboard to see the events at one central place
* Alarms notify via SNS on errors or abnormal conditions.
* Centralized logs improve observability and traceability.

**5. Security Controls**

* IAM roles per service, least privilege applied.
* TLS used for MQTT and HTTPS traffic.
* S3 and DynamoDB encrypted at rest.
* Grafana IP-restricted; future support for Cognito considered.

**6. Cost Optimization**

| **Component** | **Monthly Est.** | **Strategy** |
| --- | --- | --- |
| EC2 (Grafana) | $10–15 | t3.small, minimal uptime |
| Lambda | <$1 | Serverless, under free tier |
| S3 | <$0.25 | Lifecycle deletes after 34 hours |
| DynamoDB | <$2 | On-demand pricing |
| Monitoring | <$2 | Controlled metrics, minimal logs |
| **Total** | **~$20** | Well under Learner Lab budget |

**7. Lessons Learned & Future Scope**

**Challenges:**

* Learner Lab restrictions on IAM/Cognito.
* Persistent storage configuration for Grafana.

**Future Improvements:**

* Use Cognito for secure dashboard access.
* Integrate Amazon Timestream for long-term trends.
* ML-based anomaly detection with SageMaker or Lookout.

**8. Infrastructure as Code (Terraform)**

* IaC used to provision all components.
* Modules defined for IoT, Lambda, S3, DynamoDB, EC2, VPC.
* One-step deployment via terraform apply.
* Screenshots and TF files provided in appendix.

**9. AWS Well-Architected Framework Alignment**

**Operational Excellence:**  
The application is built with observability and operability in mind. CloudWatch aggregates logs and performance metrics from all Lambda functions. Alerts are configured via CloudWatch alarms and delivered through SNS, notifying developers about failures or threshold breaches. IaC using Terraform promotes consistent, repeatable deployments and facilitates rapid disaster recovery. The modular deployment structure allows operational improvements without service downtime.

**Security:**  
Security was incorporated from the initial design. All services operate within a VPC, segregated across public and private subnets. Sensitive services like Lambda, S3, and DynamoDB reside in private subnets. IAM roles enforce the principle of least privilege across all components. Communication between services is encrypted in transit (MQTT over TLS and HTTPS) and at rest (S3 and DynamoDB encryption). VPC endpoints are used for S3 and DynamoDB, eliminating the need for public access.

**Reliability:**  
The application achieves reliability through decoupling, redundancy, and failover awareness. IoT Core buffers incoming messages, while Lambda processes them asynchronously, allowing retry on failure. DynamoDB and S3 provide durable, highly available storage. CloudWatch alarms are configured to detect failures early, and alerts help in quick remediation. Resources can be re-deployed reliably using Terraform scripts.

**Performance Efficiency:**  
Performance is ensured using lightweight, event-driven Lambda functions optimized for sub-second execution. IoT Core efficiently routes MQTT traffic with minimal latency. DynamoDB, configured with on-demand capacity, supports high-speed lookups for dashboard queries. CloudWatch dashboards monitor processing latency and throughput. Unnecessary computation is avoided by filtering and enriching data close to ingestion points.

**Cost Optimization:**  
Serverless services (Lambda, DynamoDB) minimize costs during idle periods. S3 lifecycle policies automatically delete raw data after 34 hours, reducing long-term storage expenses. Logs and metrics are scoped to avoid excessive CloudWatch charges. On-demand billing models ensure spending aligns with usage.

**Sustainability:**  
By adopting serverless and managed services, the architecture minimizes energy consumption and carbon footprint. Stateless Lambda functions and DynamoDB reduce the need for overprovisioned resources. S3 storage is kept efficient using TTL via lifecycle policies.