Yes, Bilal — you're absolutely right ✅ and you've done a **huge amount of work** step-by-step across the AWS ecosystem. Let's now summarize everything clearly and in **sorted phase-wise order**, with each step explained — what it is, what we did, and **why** it's important.

**🌊 OES Cloud Solution – Full Implementation Summary (So Far)**

**✅ PHASE 1: DynamoDB Setup (Database)**

**🔹 What:**

We created a **DynamoDB table** to store structured buoy sensor data.

**🔧 How:**

* Table name: BuoyData
* Partition key: buoy\_id (String)
* Sort key: timestamp (String)

**🎯 Why:**

* Store time-series data from buoys
* Allows fast, scalable access to temperature and wave height history
* AWS Free Tier covers most of your usage

**✅ PHASE 2: S3 Bucket & Static Hosting**

**🔹 What:**

* Created an **S3 bucket**: oes-buoy-storage-bilal
* Enabled **static website hosting**
* Also used S3 to **store raw JSON data logs** from buoys

**🔧 Used for:**

* Hosting index.html dashboard
* Storing incoming buoy data logs from Lambda

**🎯 Why:**

* S3 is durable, cheap, and scalable
* Makes it easy to view logs and host frontend without a web server

**✅ PHASE 3: IAM Roles & Policies**

**🔹 What:**

* Created an IAM **Role for Lambda**: lambda-oes-role
* Attached least-privilege **inline policy** for:
  + dynamodb:PutItem, Scan
  + s3:PutObject
  + sns:Publish
  + logs:\*

**🎯 Why:**

* Securely grants Lambda access to only what it needs
* Helps pass AWS best practice of **"least privilege"**

**✅ PHASE 4: Lambda Functions**

**🔹 Function #1: storeBuoyData**

* Accepts POST data from buoys
* Saves it to DynamoDB
* Logs full data to S3
* Sends alert to SNS (if wave > 3.0)

**🔹 Function #2: getData**

* Returns all saved data from DynamoDB (GET)

**🎯 Why:**

* Core of your backend logic
* Completely serverless and auto-scalable

**✅ PHASE 5: API Gateway Integration**

**🔹 Created REST API:**

* Resource: /sendData → Method: POST → Calls storeBuoyData
* Resource: /getData → Method: GET → Calls getData

**🎯 Why:**

* Allows HTTP access to your Lambda functions
* Makes your backend accessible to frontend or 3rd-party sensors

**✅ PHASE 6: Frontend Dashboard**

**🔹 Hosted on S3:**

* index.html dashboard
* Fetches data from /getData API
* Uses **Chart.js** to show:
  + 📈 Wave height over time
  + 🌡️ Temperature over time
  + ⚠️ Real-time alerts (if wave height exceeds 3.0m)
  + Refresh button + smooth visual flow

**🎯 Why:**

* Simple, clean, and interactive monitoring interface
* Helps decision makers or operators monitor live buoy conditions

**✅ PHASE 7: SNS Email Alert (Wave Threshold Notification)**

**🔹 What:**

* Created an **SNS Topic**: BuoyAlerts
* Subscribed your **email**: bilal.naseer2k18@gmail.com
* Lambda sends alert if wave\_height > 3.0

**🎯 Why:**

* Helps stakeholders take immediate action
* Sends real-time critical notifications via email

**✅ PHASE 8: API Key Security**

**🔹 What:**

* Enabled x-api-key security for both /sendData and /getData
* Created:
  + API Key: OESFrontendKey
  + Usage Plan: OESUsagePlan
  + Attached API + Stage (dev) to plan
  + Bound the API Key to that plan

**🎯 Why:**

* Prevents unauthorized use of your APIs
* Adds quota and rate limit protection
* Ensures only your frontend or approved clients access your backend

**✅ PHASE 9: CloudWatch Monitoring (Partially Done)**

**🔹 What we’ve done:**

* Enabled **CloudWatch Logs** for Lambda
* Logs record every invocation, error, and print/debug output

**🔹 What’s Next (Optional for Phase E):**

* Enable **X-Ray tracing** for Lambda + API Gateway
* Create a **CloudWatch Dashboard** to visualize usage, latency, and errors
* Add CloudWatch **Alarms** (optional) if latency spikes or error rates rise

PHASE-10: GITHUB & CI/CD Pipeline

**✅ Summary Table**

| **AWS Service** | **Purpose** |
| --- | --- |
| DynamoDB | Store buoy sensor readings (structured) |
| Lambda | Handle backend processing logic (store and fetch data) |
| API Gateway | Expose HTTP endpoints for sensors and frontend |
| IAM | Securely connect services with strict permissions |
| S3 | Store raw logs + host frontend website |
| SNS | Send real-time alerts when wave height exceeds threshold |
| API Key + Usage Plan | Restrict API access + throttle usage |
| CloudWatch Logs | View backend logs for debugging and monitoring |
| Chart.js | Visualize real-time buoy data in the frontend |

**✅ What You've Built**

You now have a **fully working, secure, scalable, and serverless IoT dashboard** solution using only AWS Free Tier services. Amazing work, Bilal!

**🔜 Ready for the next Phase?**

Would you like to now:

1. ✅ Finish Phase E: Add **CloudWatch Dashboards + X-Ray**
2. 🧾 Start writing your **Assignment Report / Architecture Diagram**
3. 🌐 Add **Cognito Authentication** (for advanced login protection)
4. 📦 Add features like **offline logging**, **PDF exports**, or **data filtering**

Let me know what you'd like to tackle next!