Sony Sunny

Direct Edge MQTT → AWS IoT → Lambda → RDS (PostgreSQL)

By sonysunny

Table of Contents

[Direct Edge MQTT → AWS IoT → Lambda → RDS (PostgreSQL) 2](#_Toc218721253)

[Final Architecture 2](#_Toc218721254)

[PHASE 1 — Lock Contracts & Database Schema 3](#_Toc218721255)

[Goal 3](#_Toc218721256)

[Step 1.1 — Lock MQTT Topic 3](#_Toc218721257)

[Step 1.2 — Lock MQTT Payload 3](#_Toc218721258)

[Step 1.3 — Create PostgreSQL Schema 4](#_Toc218721259)

[PHASE 2 — Amazon RDS (PostgreSQL) 5](#_Toc218721260)

[Goal 5](#_Toc218721261)

[Step 2.1 — Create RDS Instance 5](#_Toc218721262)

[Step 2.2 — Open Port 5432 (Local Access) 5](#_Toc218721263)

[Step 2.3 — Connect From Laptop 6](#_Toc218721264)

[PHASE 3 — AWS IoT Core (MQTT Ingestion) 6](#_Toc218721265)

[Goal 6](#_Toc218721266)

[Step 3.1 — Create IoT Thing 6](#_Toc218721267)

[Step 3.2 — Create Certificate (Auto-generate) 7](#_Toc218721268)

[Step 3.3 — Attach IoT Policy 7](#_Toc218721269)

[Step 3.4 — Get IoT MQTT Endpoint 8](#_Toc218721270)

[Step 3.5 — Configure MQTTX 8](#_Toc218721271)

[Step 3.6 — Verify MQTT 9](#_Toc218721272)

[PHASE 4 — IoT Rule → Lambda → RDS 9](#_Toc218721273)

[Goal 9](#_Toc218721274)

[Step 4.1 — Create IoT Rule 9](#_Toc218721275)

[Step 4.2 — Create Lambda Function 10](#_Toc218721276)

[🔐 REQUIRED — RDS Security Group (Lambda → RDS) 11](#_Toc218721277)

[Step 4.3 — Identify Lambda Security Group 11](#_Toc218721278)

[Step 4.4 — Allow Lambda in RDS SG 12](#_Toc218721279)

[Final Validation 12](#_Toc218721280)

[PHASE 5 — Production Hardening & Scalable Architecture 12](#_Toc218721281)

[🎯 Phase 5 Goals 12](#_Toc218721282)

[Final Production Architecture (Phase 5) 13](#_Toc218721283)

[PHASE 5.1 — Make RDS Private (Security Hardening) 13](#_Toc218721284)

[Goal 13](#_Toc218721285)

[Step 5.1.1 — Disable Public Access 14](#_Toc218721286)

[Step 5.1.2 — Security Group Rules (Final) 14](#_Toc218721287)

[Expected Result 14](#_Toc218721288)

[PHASE 5.2 — Private Networking (NAT + Subnets) 14](#_Toc218721289)

[Goal 14](#_Toc218721290)

[Step 5.2.1 — Subnet Strategy 15](#_Toc218721291)

[Step 5.2.2 — Create NAT Gateway 15](#_Toc218721292)

[Step 5.2.3 — Route Table (Private Subnet) 15](#_Toc218721293)

[Cost Note 15](#_Toc218721294)

[PHASE 5.3 — Secrets Manager (Credential Hardening) 16](#_Toc218721295)

[Goal 16](#_Toc218721296)

[Step 5.3.1 — Store DB Credentials 16](#_Toc218721297)

[Step 5.3.2 — Update Lambda IAM Role 16](#_Toc218721298)

[Step 5.3.3 — Update Lambda Code (conceptual) 17](#_Toc218721299)

[PHASE 5.4 — Reliable Ingestion (SQS + DLQ) 17](#_Toc218721300)

[Goal 17](#_Toc218721301)

[Why SQS is REQUIRED in production 17](#_Toc218721302)

[Step 5.4.1 — Create SQS Queues 17](#_Toc218721303)

[Step 5.4.2 — Configure Redrive Policy 18](#_Toc218721304)

[Step 5.4.3 — Update IoT Rule Target 18](#_Toc218721305)

[Step 5.4.4 — Update Lambda Trigger 18](#_Toc218721306)

[Behavior 18](#_Toc218721307)

[PHASE 5.5 — Idempotency (Duplicate Protection) 19](#_Toc218721308)

[Problem 19](#_Toc218721309)

[Option A — Unique Constraint (Recommended) 19](#_Toc218721310)

[Option B — Message ID 19](#_Toc218721311)

[PHASE 5.6 — Cost Optimization 19](#_Toc218721312)

[Biggest Cost Risks 19](#_Toc218721313)

[Step 5.6.1 — Reduce NAT Usage (VPC Endpoints) 20](#_Toc218721314)

[Step 5.6.2 — Logging Discipline 20](#_Toc218721315)

[Step 5.6.3 — Right-size Resources 20](#_Toc218721316)

[PHASE 5.7 — Monitoring & Alarms 20](#_Toc218721317)

[Lambda Alarms 20](#_Toc218721318)

[RDS Alarms 20](#_Toc218721319)

[DLQ Alarm 21](#_Toc218721320)

[✅ Phase 5 Completion Checklist 21](#_Toc218721321)

[🏁 Final State 21](#_Toc218721322)

[PHASE 6 — Multi-Device Scaling (Fleet Readiness) 21](#_Toc218721323)

[🎯 Goal 21](#_Toc218721324)

[PHASE 6.1 — Topic & Identity Strategy (Fleet-safe) 22](#_Toc218721325)

[Final Topic Structure (already correct) 22](#_Toc218721326)

[Device Identity Rules 22](#_Toc218721327)

[PHASE 6.2 — IoT Policy (Least Privilege per Device) 22](#_Toc218721328)

[PHASE 6.3 — Database Scaling Basics 23](#_Toc218721329)

[Indexing (already done) 23](#_Toc218721330)

[Partitioning (optional at scale) 23](#_Toc218721331)

[PHASE 6.4 — Lambda Concurrency Control 24](#_Toc218721332)

[Protect RDS 24](#_Toc218721333)

[PHASE 6 Output 24](#_Toc218721334)

[PHASE 7 — Schema Evolution (Future-Proofing) 24](#_Toc218721335)

[🎯 Goal 24](#_Toc218721336)

[PHASE 7.1 — Versioned Payloads 25](#_Toc218721337)

[Add schema version (recommended) 25](#_Toc218721338)

[PHASE 7.2 — JSONB-First Strategy (Why you chose it) 25](#_Toc218721339)

[PHASE 7.3 — Breaking Change Strategy 25](#_Toc218721340)

[PHASE 7 Output 26](#_Toc218721341)

[PHASE 8 — Analytics & Dashboards 26](#_Toc218721342)

[🎯 Goal 26](#_Toc218721343)

[PHASE 8.1 — Read-Optimized Views 26](#_Toc218721344)

[PHASE 8.2 — Dashboard Options 27](#_Toc218721345)

[Option A — Amazon QuickSight 27](#_Toc218721346)

[Option B — Grafana 27](#_Toc218721347)

[Option C — Custom Dashboard 27](#_Toc218721348)

[PHASE 8.3 — Common Metrics 27](#_Toc218721349)

[PHASE 8 Output 28](#_Toc218721350)

[PHASE 9 — Cost-Per-Message Analysis 28](#_Toc218721351)

[🎯 Goal 28](#_Toc218721352)

[PHASE 9.1 — Cost Components (per message) 28](#_Toc218721353)

[PHASE 9.2 — Example Cost Estimate 29](#_Toc218721354)

[PHASE 9.3 — Cost Optimization Levers 29](#_Toc218721355)

[PHASE 9 Output 30](#_Toc218721356)

[FINAL STATE (Phases 1–9) 30](#_Toc218721357)

# Direct Edge MQTT → AWS IoT → Lambda → RDS (PostgreSQL)

**Hands-On Step-by-Step Guide (Practice Version)**

## Final Architecture

MQTTX / Edge Device

        |

        | MQTT over TLS (X.509)

        v

AWS IoT Core

        |

        | IoT Rule

        v

AWS Lambda (VPC)

        |

        v

Amazon RDS (PostgreSQL)

# PHASE 1 — Lock Contracts & Database Schema

## Goal

Freeze interfaces so AWS setup becomes mechanical.

## Step 1.1 — Lock MQTT Topic

devices/{device\_id}/telemetry

Example:

devices/esp32\_001/telemetry

## Step 1.2 — Lock MQTT Payload

{

  "device\_id": "esp32\_001",

  "timestamp": "2026-01-03T10:15:30Z",

  "metrics": {

    "temperature": 26.4,

    "pressure": 101.2,

    "humidity": 58.1

  }

}

## Step 1.3 — Create PostgreSQL Schema

CREATE TABLE devices (

    id SERIAL PRIMARY KEY,

    device\_id VARCHAR(64) UNIQUE NOT NULL,

    device\_type VARCHAR(50),

    created\_at TIMESTAMP DEFAULT NOW()

);

CREATE TABLE telemetry (

    id BIGSERIAL PRIMARY KEY,

    device\_id VARCHAR(64) NOT NULL,

    ts TIMESTAMP NOT NULL,

    metrics JSONB NOT NULL,

    received\_at TIMESTAMP DEFAULT NOW()

);

CREATE INDEX idx\_telemetry\_device\_ts

ON telemetry (device\_id, ts);

✅ Phase 1 Output

* Schema ready
* JSONB metrics supported

# PHASE 2 — Amazon RDS (PostgreSQL)

## Goal

Create a low-cost DB and prove it works **before MQTT**.

## Step 2.1 — Create RDS Instance

**Region**

us-east-1

**Settings**

* Engine: PostgreSQL 15.x
* Template: Free tier
* DB identifier: iot-telemetry-db
* DB name: iotdb
* Username: postgres
* Instance: db.t3.micro
* Storage: 20 GB
* Public access: **Yes (temporary)**

Security group: **Create new**rds-postgres-sg

* Monitoring: **Database Insights – Standard**
* Performance Insights: **Enabled (7 days)**
* Deletion protection: ❌ Off

Create database → wait until **Status = Available**

## Step 2.2 — Open Port 5432 (Local Access)

Go to:

EC2 → Security Groups → rds-postgres-sg → Inbound rules

Add rule:

* Type: PostgreSQL
* Port: 5432
* Source: **My IP**

## Step 2.3 — Connect From Laptop

psql -h <RDS-ENDPOINT> -U postgres -d iotdb

Run schema from Phase 1.

✅ Phase 2 Output

* DB reachable
* Tables created
* Manual inserts possible

# PHASE 3 — AWS IoT Core (MQTT Ingestion)

## Goal

Verify MQTT → AWS **without Lambda or DB**.

## Step 3.1 — Create IoT Thing

Thing name: esp32\_001

## Step 3.2 — Create Certificate (Auto-generate)

Download:

* certificate.pem.crt
* private.pem.key
* AmazonRootCA1.pem

Store in:

iot-certs/esp32\_001/

## Step 3.3 — Attach IoT Policy

Policy name:

esp32\_mqtt\_policy

{

  "Version": "2012-10-17",

  "Statement": [

    {

      "Effect": "Allow",

      "Action": [

        "iot:Connect",

        "iot:Publish",

        "iot:Subscribe",

        "iot:Receive"

      ],

      "Resource": "\*"

    }

  ]

}

Certificate status: **ACTIVE**

## Step 3.4 — Get IoT MQTT Endpoint

AWS IoT Core → **Connect → Connect one device → MQTT**

Copy:

xxxxxxxx-ats.iot.us-east-1.amazonaws.com

📌 Not an ARN. Ping will not work.

## Step 3.5 — Configure MQTTX

**Connection**

* Protocol: mqtts
* Host: IoT endpoint
* Port: 8883
* Client ID: esp32\_001
* Username/Password: empty

**TLS**

* CA: AmazonRootCA1.pem
* Client cert: device cert
* Private key: device key

Connect → Status must be **Connected**

## Step 3.6 — Verify MQTT

Subscribe:

devices/+/telemetry

Publish:

devices/esp32\_001/telemetry

Payload = locked JSON.

Verify in:

* MQTTX
* AWS IoT → MQTT Test Client

✅ Phase 3 Output

* MQTT ingestion verified
* Certificates + policy confirmed

# PHASE 4 — IoT Rule → Lambda → RDS

## Goal

Persist telemetry into PostgreSQL.

## Step 4.1 — Create IoT Rule

SQL:

SELECT \*

FROM 'devices/#'

WHERE topic(2) = 'telemetry'

Action:

* Invoke Lambda

## Step 4.2 — Create Lambda Function

**Environment Variables**

DB\_HOST

DB\_NAME

DB\_USER

DB\_PASS

DB\_PORT=5432

**Lambda Code**

import os, json, socket

import pg8000.native

socket.setdefaulttimeout(5)

def lambda\_handler(event, context):

    conn = pg8000.native.Connection(

        user=os.environ["DB\_USER"],

        password=os.environ["DB\_PASS"],

        host=os.environ["DB\_HOST"],

        port=int(os.environ["DB\_PORT"]),

        database=os.environ["DB\_NAME"]

    )

    payload = event

    if "payload" in payload:

        payload = json.loads(payload["payload"])

    conn.run(

        "INSERT INTO telemetry (device\_id, ts, metrics) "

        "VALUES (:d, :t::timestamptz, :m::jsonb)",

        d=payload["device\_id"],

        t=payload["timestamp"],

        m=json.dumps(payload["metrics"])

    )

    return {"ok": True}

## 🔐 REQUIRED — RDS Security Group (Lambda → RDS)

### Step 4.3 — Identify Lambda Security Group

Lambda → Configuration → VPC

Copy SG ID.

### Step 4.4 — Allow Lambda in RDS SG

EC2 → Security Groups → rds-postgres-sg → Inbound rules

Add:

* Type: PostgreSQL
* Port: 5432
* Source: **Lambda Security Group ID**

❌ Do NOT use 0.0.0.0/0

## Final Validation

1. Publish MQTT message
2. IoT Rule triggers Lambda
3. Lambda inserts row
4. Verify:

SELECT \* FROM telemetry ORDER BY ts DESC;

# PHASE 5 — Production Hardening & Scalable Architecture

**Do Phase 5 only after Phase 1–4 are fully working end-to-end.**This phase upgrades the system from *learning-grade* to *production-grade*.

## 🎯 Phase 5 Goals

* Secure database (no public access)
* Controlled outbound networking
* Reliable ingestion with retries
* Failure isolation (DLQ)
* Predictable AWS costs
* Clear operational visibility

## Final Production Architecture (Phase 5)

MQTT Device / MQTTX

        |

        | mTLS MQTT (8883)

        v

AWS IoT Core

        |

        | IoT Rule

        v

SQS Queue (buffer + retry)

        |

        v

AWS Lambda (Private Subnet)

        |

        v

Amazon RDS (Private Subnet)

# PHASE 5.1 — Make RDS Private (Security Hardening)

### Goal

Remove all public exposure of the database.

### Step 5.1.1 — Disable Public Access

RDS → Databases → iot-telemetry-db → Modify

Set:

* **Publicly accessible** → ❌ No

Apply immediately (or during maintenance window).

### Step 5.1.2 — Security Group Rules (Final)

RDS Security Group (rds-postgres-sg) inbound rules:

|  |  |  |
| --- | --- | --- |
| **Type** | **Port** | **Source** |
| PostgreSQL | 5432 | **Lambda Security Group** |

❌ Remove:

* My IP
* 0.0.0.0/0

### Expected Result

* Database reachable **only** from Lambda
* No direct laptop access
* Zero public attack surface

# PHASE 5.2 — Private Networking (NAT + Subnets)

### Goal

Allow Lambda outbound access **without public IPs**.

### Step 5.2.1 — Subnet Strategy

|  |  |
| --- | --- |
| **Subnet Type** | **Purpose** |
| Public Subnet | NAT Gateway |
| Private Subnet | Lambda + RDS |

RDS and Lambda **must be in private subnets**.

### Step 5.2.2 — Create NAT Gateway

VPC → NAT Gateways → Create NAT Gateway

* Subnet: **Public subnet**
* Elastic IP: Allocate new

### Step 5.2.3 — Route Table (Private Subnet)

Private subnet route table:

0.0.0.0/0 → NAT Gateway

This allows Lambda to:

* Reach AWS services
* Pull secrets
* Write logs

### Cost Note

NAT Gateway is **one of the biggest AWS cost drivers**.  
We reduce usage in Phase 5.6.

# PHASE 5.3 — Secrets Manager (Credential Hardening)

### Goal

Remove DB credentials from Lambda environment variables.

### Step 5.3.1 — Store DB Credentials

Secrets Manager → Store new secret

Secret type:

* Credentials for RDS

Store:

username

password

host

port

dbname

Secret name:

prod/rds/iotdb

### Step 5.3.2 — Update Lambda IAM Role

Attach policy:

SecretsManagerReadWrite (or least-privilege custom policy)

### Step 5.3.3 — Update Lambda Code (conceptual)

Replace env vars with Secrets Manager fetch.  
(No code change needed for practice — but required in production.)

# PHASE 5.4 — Reliable Ingestion (SQS + DLQ)

### Goal

Never lose MQTT messages during failures or spikes.

## Why SQS is REQUIRED in production

* Lambda retries are limited
* DB outages cause message loss
* Traffic spikes overwhelm RDS

SQS solves all three.

### Step 5.4.1 — Create SQS Queues

Create **two queues**:

1️⃣ Main Queue  
Name:

telemetry-ingest-queue

2️⃣ Dead Letter Queue  
Name:

telemetry-ingest-dlq

### Step 5.4.2 — Configure Redrive Policy

On main queue:

* DLQ: telemetry-ingest-dlq
* Max receive count: **5**

### Step 5.4.3 — Update IoT Rule Target

Change IoT Rule action:

❌ Invoke Lambda  
✅ Send message to **SQS queue**

Message body:

* Entire MQTT payload (JSON)

### Step 5.4.4 — Update Lambda Trigger

Configure Lambda trigger:

* Source: SQS
* Batch size: 5–10
* Visibility timeout: > Lambda timeout

### Behavior

* Lambda fails → message retried
* Repeated failures → message moves to DLQ
* No data loss

# PHASE 5.5 — Idempotency (Duplicate Protection)

### Problem

Retries can insert duplicate rows.

### Option A — Unique Constraint (Recommended)

ALTER TABLE telemetry

ADD CONSTRAINT uniq\_device\_ts UNIQUE (device\_id, ts);

Then change insert:

INSERT INTO telemetry (...)

ON CONFLICT DO NOTHING;

### Option B — Message ID

Add message\_id to payload and enforce uniqueness.

# PHASE 5.6 — Cost Optimization

### Biggest Cost Risks

* NAT Gateway traffic
* RDS running 24/7
* Excessive logging

### Step 5.6.1 — Reduce NAT Usage (VPC Endpoints)

Create VPC endpoints for:

* CloudWatch Logs
* Secrets Manager
* KMS
* SQS

This keeps traffic inside AWS network.

### Step 5.6.2 — Logging Discipline

Set:

* IoT logs → ERROR only
* Lambda log retention → 7–14 days

### Step 5.6.3 — Right-size Resources

* RDS: db.t3.micro or t4g.micro
* Storage: gp3
* Scale **only after measuring**

# PHASE 5.7 — Monitoring & Alarms

### Lambda Alarms

* Errors > 0
* Throttles > 0
* Duration near timeout

### RDS Alarms

* CPU > 70%
* Free storage < 20%
* DB connections spike

### DLQ Alarm

* Messages in DLQ > 0 → immediate investigation

## ✅ Phase 5 Completion Checklist

* ✅ RDS is private
* ✅ Lambda runs in private subnet
* ✅ NAT Gateway configured
* ✅ Secrets Manager used
* ✅ SQS buffering enabled
* ✅ DLQ configured
* ✅ Duplicate protection in DB
* ✅ Logging + alarms active

## 🏁 Final State

You now have:

* Secure IoT ingestion
* Zero public DB exposure
* No message loss
* Controlled cost
* Production-grade AWS architecture

# PHASE 6 — Multi-Device Scaling (Fleet Readiness)

## 🎯 Goal

Scale from **1 device → thousands of devices** safely and predictably.

## PHASE 6.1 — Topic & Identity Strategy (Fleet-safe)

### Final Topic Structure (already correct)

devices/{device\_id}/telemetry

### Device Identity Rules

* **One Thing per device**
* **One certificate per device**
* **One unique Client ID per device**
* Never reuse certificates across devices

Example:

|  |  |  |
| --- | --- | --- |
| **Device** | **Thing** | **Client ID** |
| ESP32 #1 | esp32\_001 | esp32\_001 |
| ESP32 #2 | esp32\_002 | esp32\_002 |
| Gateway | gw\_01 | gw\_01 |

## PHASE 6.2 — IoT Policy (Least Privilege per Device)

Replace wildcard policy with **device-scoped policy**:

{

  "Version": "2012-10-17",

  "Statement": [

    {

      "Effect": "Allow",

      "Action": "iot:Connect",

      "Resource": "arn:aws:iot:us-east-1:\*:client/${iot:Connection.Thing.ThingName}"

    },

    {

      "Effect": "Allow",

      "Action": [

        "iot:Publish"

      ],

      "Resource": "arn:aws:iot:us-east-1:\*:topic/devices/${iot:Connection.Thing.ThingName}/telemetry"

    }

  ]

}

✅ Device can publish **only its own data**

## PHASE 6.3 — Database Scaling Basics

### Indexing (already done)

CREATE INDEX idx\_telemetry\_device\_ts

ON telemetry (device\_id, ts);

### Partitioning (optional at scale)

Partition telemetry by **time** (monthly):

PARTITION BY RANGE (ts);

## PHASE 6.4 — Lambda Concurrency Control

### Protect RDS

Lambda → Configuration → Concurrency  
Set:

Reserved concurrency = 5–10

Prevents DB overload during spikes.

## PHASE 6 Output

* Fleet-safe authentication
* No cross-device data leakage
* Controlled DB pressure
* Ready for 100s–1000s devices

# PHASE 7 — Schema Evolution (Future-Proofing)

## 🎯 Goal

Change payloads **without breaking ingestion or analytics**.

## PHASE 7.1 — Versioned Payloads

### Add schema version (recommended)

{

  "schema\_version": "1.0",

  "device\_id": "esp32\_001",

  "timestamp": "2026-01-03T15:20:00Z",

  "metrics": {

    "temperature": 26.7

  }

}

## PHASE 7.2 — JSONB-First Strategy (Why you chose it)

* No ALTER TABLE for new sensors
* Backward compatible
* Queryable later

Example query:

SELECT

  metrics->>'temperature' AS temp

FROM telemetry

WHERE device\_id = 'esp32\_001';

## PHASE 7.3 — Breaking Change Strategy

If payload changes drastically:

* Increment schema\_version
* Handle logic in Lambda
* Keep DB unchanged

## PHASE 7 Output

* Schema evolves safely
* No downtime
* No DB migrations for new sensors

# PHASE 8 — Analytics & Dashboards

## 🎯 Goal

Turn telemetry into **insights**.

## PHASE 8.1 — Read-Optimized Views

Create SQL views for analytics:

CREATE VIEW telemetry\_latest AS

SELECT DISTINCT ON (device\_id)

device\_id, ts, metrics

FROM telemetry

ORDER BY device\_id, ts DESC;

## PHASE 8.2 — Dashboard Options

### Option A — Amazon QuickSight

* Native AWS integration
* Reads from RDS
* Cost-effective for AWS-only stack

### Option B — Grafana

* PostgreSQL datasource
* Real-time charts
* Popular for IoT/SCADA

### Option C — Custom Dashboard

* Backend: API Gateway + Lambda
* Frontend: React / Next.js
* Best for productized platforms

## PHASE 8.3 — Common Metrics

* Temperature trends
* Device uptime
* Message frequency
* Last-seen timestamp

Example:

SELECT

  device\_id,

  MAX(ts) AS last\_seen

FROM telemetry

GROUP BY device\_id;

## PHASE 8 Output

* Real-time visibility
* Device health monitoring
* Business-ready analytics

# PHASE 9 — Cost-Per-Message Analysis

## 🎯 Goal

Know exactly **how much each message costs**.

## PHASE 9.1 — Cost Components (per message)

|  |  |
| --- | --- |
| **Service** | **Cost Driver** |
| AWS IoT Core | Messages |
| SQS | Requests |
| Lambda | Invocations + duration |
| RDS | Instance + storage |
| NAT | Data processed |

## PHASE 9.2 — Example Cost Estimate

Assume:

* 1 device
* 1 message / second
* ~2.6M messages / month

Approx monthly:

|  |  |
| --- | --- |
| **Component** | **Cost** |
| IoT Core | ~$2–3 |
| Lambda | <$1 |
| SQS | <$0.50 |
| RDS (micro) | ~$15 |
| Logs | ~$1 |

➡️ **~$20–25 / month**

## PHASE 9.3 — Cost Optimization Levers

* Batch SQS → Lambda
* Reduce IoT logs
* Partition DB
* Use VPC endpoints (reduce NAT)
* Archive old telemetry to S3

## PHASE 9 Output

* Predictable AWS bill
* Cost scales linearly
* No surprise charges

# FINAL STATE (Phases 1–9)

You now have:

* Fleet-scale IoT ingestion
* Secure networking
* Reliable retries
* Evolving schemas
* Analytics pipelines
* Cost visibility

This is **real production IoT architecture**, not a demo.