# 

Table of Contents

[Final Architecture 2](#_Toc218722305)

[PHASE 1 — Lock Contracts & Database Schema 3](#_Toc218722306)

[Goal 3](#_Toc218722307)

[Step 1.1 — Lock MQTT Topic 3](#_Toc218722308)

[Step 1.2 — Lock MQTT Payload 3](#_Toc218722309)

[Step 1.3 — Create PostgreSQL Schema 4](#_Toc218722310)

[PHASE 2 — Amazon RDS (PostgreSQL) 5](#_Toc218722311)

[Goal 5](#_Toc218722312)

[Step 2.1 — Create RDS Instance 5](#_Toc218722313)

[Step 2.2 — Open Port 5432 (Local Access) 5](#_Toc218722314)

[Step 2.3 — Connect From Laptop 6](#_Toc218722315)

[PHASE 3 — AWS IoT Core (MQTT Ingestion) 7](#_Toc218722316)

[Goal 7](#_Toc218722317)

[Step 3.1 — Create IoT Thing 7](#_Toc218722318)

[Step 3.2 — Create Certificate (Auto-generate) 7](#_Toc218722319)

[Step 3.3 — Attach IoT Policy 7](#_Toc218722320)

[Step 3.4 — Get IoT MQTT Endpoint 8](#_Toc218722321)

[Step 3.5 — Configure MQTTX 9](#_Toc218722322)

[Step 3.6 — Verify MQTT 9](#_Toc218722323)

[PHASE 4 — IoT Rule → Lambda → RDS 10](#_Toc218722324)

[Goal 10](#_Toc218722325)

[Step 4.1 — Create IoT Rule 10](#_Toc218722326)

[Step 4.2 — Create Lambda Function 10](#_Toc218722327)

[🔐 REQUIRED — RDS Security Group (Lambda → RDS) 12](#_Toc218722328)

[**Step 4.3 — Identify Lambda Security Group** 12](#_Toc218722329)

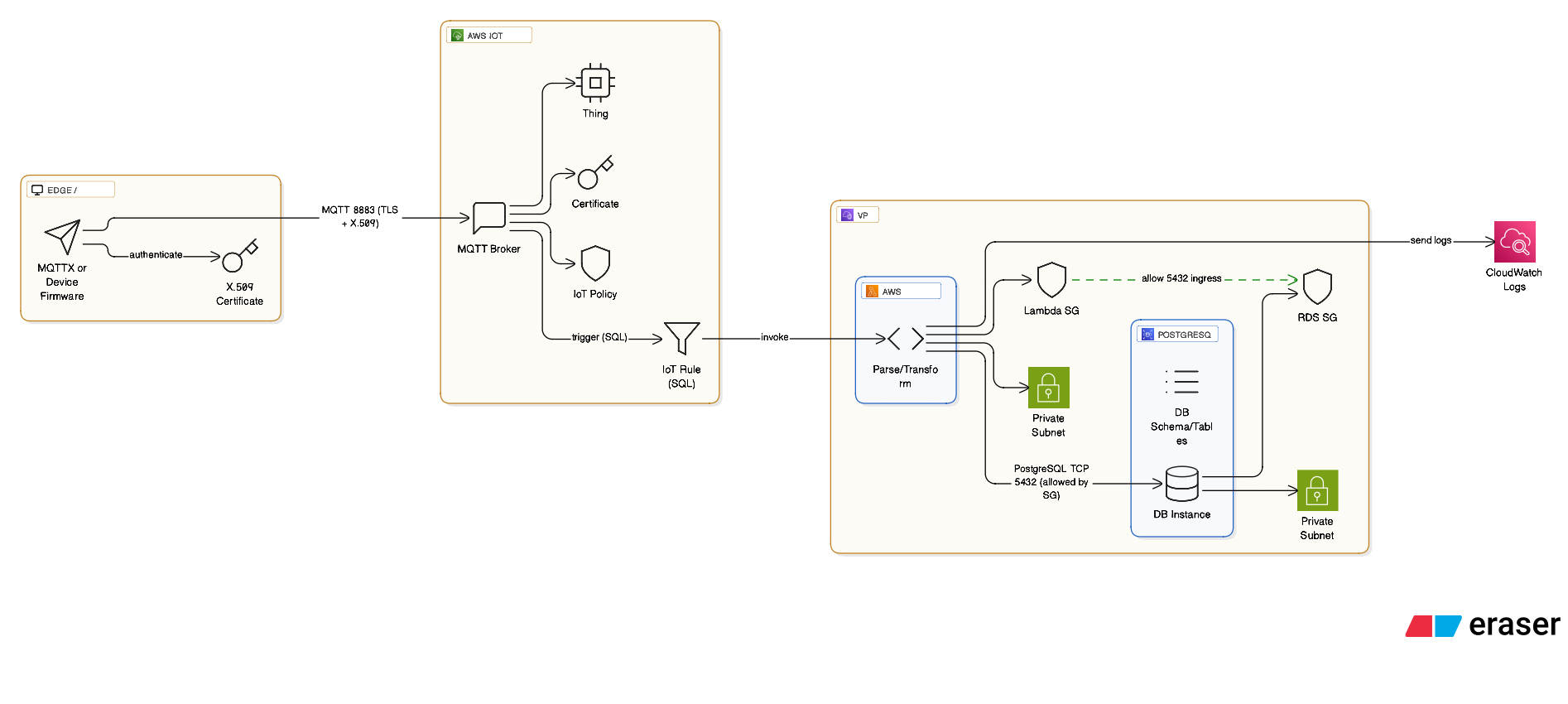
[**Step 4.4 — Allow Lambda in RDS SG** 12](#_Toc218722330)

[Final Validation 13](#_Toc218722331)

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

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

## **Final Architecture**



# **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;