# Direct Edge MQTT Ingestion → AWS RDS (PostgreSQL)

## Step-by-Step Guide (Phases 1 to 4 based on our chats)

## What we built

**MQTTX / Edge** → **AWS IoT Core** → **IoT Rule (SQL filter)** → **Lambda** → **RDS PostgreSQL**

# Phase 1 — Lock the contracts + DB schema (foundation)

## Step 1. Lock 3 things (we did this first)

1. MQTT topic structure
2. MQTT JSON payload
3. RDS schema

### Topic (locked)

devices/{device\_id}/telemetry  
Example: devices/esp32\_001/telemetry

### Payload (locked)

{

  "device\_id": "esp32\_001",

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

  "metrics": {

    "temperature": 26.4,

    "pressure": 101.2,

    "humidity": 58.1

  }

}

## Step 2. Create RDS schema exactly like you wrote

Run these in PostgreSQL:

### Devices table (optional but recommended)

CREATE TABLE devices (

    id SERIAL PRIMARY KEY,

    device\_id VARCHAR(64) UNIQUE NOT NULL,

    device\_type VARCHAR(50),

    created\_at TIMESTAMP DEFAULT NOW()

);

### Telemetry table (main)

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()

);

### Index (important)

CREATE INDEX idx\_telemetry\_device\_ts

ON telemetry (device\_id, ts);

✅ Output of Phase 1:

* DB has telemetry table ready.
* You can query metrics using JSONB.

# PHASE 2 — AWS RDS (PostgreSQL) SETUP

🎯 Goal: Create a **safe, low-cost PostgreSQL database** that Lambda can write to later.

We’ll do this in **small controlled steps**.

## Step 2.1 — Choose Region (Important)

At the **top-right of AWS Console**, select:

### Recommended

👉 **us-east-1 (N. Virginia)**

Why:

* Best AWS IoT + Lambda support
* Most tutorials match this region
* Slightly cheaper

⚠️ Once chosen, **do not change region mid-project**

## Step 2.2 — Open RDS Service

1. Go to **AWS Console**
2. Search for **RDS**
3. Click **Amazon RDS**
4. Click **Create database**

## Step 2.3 — Database Creation Options

### 1️⃣ Choose creation method

Select:

Standard create

### 2️⃣ Engine options

Choose:

PostgreSQL

Version:

PostgreSQL 15.x (default is fine)

### 3️⃣ Templates (IMPORTANT for cost)

Choose:

Free tier

This automatically selects safe defaults.

## Step 2.4 — DB Instance Settings

### DB instance identifier

iot-telemetry-db

### Credentials

Choose:

Self managed

Set:

Master username: postgres

Master password: <strong password>

📌 Save this password somewhere safe  
(We’ll later move it to Secrets Manager)

## Step 2.5 — Instance Configuration

For Free Tier / Low Cost:

DB instance class: db.t3.micro

Storage type: General Purpose (gp2 or gp3)

Allocated storage: 20 GB

Enable storage autoscaling: ❌ (disable)

## Step 2.6 — Connectivity (VERY IMPORTANT)

### Virtual Private Cloud (VPC)

Keep:

Default VPC

### Public access

For now (learning phase):

Yes

⚠️ This is **TEMPORARY**Later we’ll make it private and connect via Lambda VPC.

### VPC security group

Choose:

Create new

Security group name:

rds-postgres-sg

### Database port

5432

## Step 2.7 — Database Authentication

Keep default:

Password authentication

## Step 2.8 — Additional Configuration

### Initial database name

iotdb

### Backup

Enable automated backups: ❌

### Monitoring

Enable Enhanced Monitoring: ❌

### Deletion protection

❌ Disable (IMPORTANT for learning)

## Step 2.9 — Create Database

Click:  
👉 **Create database**

⏳ Creation time:

* 5–10 minutes

Status will move from:

Creating → Available

## ✅ Checkpoint (STOP HERE)

Once the DB status becomes **Available**, reply with:

1. ✅ DB instance name: iot-telemetry-db
2. ✅ Endpoint (looks like xxxx.rds.amazonaws.com)
3. ✅ Region
4. ❓ Any error you see (if any)

## What we will do NEXT (Phase 2 continuation)

### Phase 2.10

* Connect to RDS from your laptop
* Create tables (devices, telemetry)
* Verify inserts manually

Only **after DB is proven working**, we move to:  
👉 **Phase 3 — AWS IoT Core (MQTT ingestion)**

Take your time — tell me when the DB is **Available**.

is this access denied an issue now?

Yes — **“AccessDenied” is an issue right now**, because it means AWS is blocking you from doing the next step (RDS).

Most of the time it’s **one of these 4 causes**, and you can fix it quickly:

## 1) You’re logged in as the IAM user, but it has no permissions

Even if you created sony-admin, it might not have the **AdministratorAccess** policy attached.

**Fix**

1. Sign in as **root** (just for this fix)
2. Go to **IAM → Users → sony-admin → Permissions**
3. Confirm you see **AdministratorAccess**
   * If not: **Add permissions → Attach policies directly → AdministratorAccess → Add permissions**

Then log out root → log back in as sony-admin.

## 2) You created the IAM user, but you’re still logged in as root or a different user

Sometimes you think you’re using sony-admin, but you’re actually in another session.

**Check**Top-right in AWS console → click your profile name → it should show something like:

* **User: sony-admin** (good)  
  or
* **Root** (not for daily work, but root should not be denied)

If it shows a different user, switch.

## 3) Your AWS account is still in “activation / restricted” state

New accounts sometimes block some services for a short time.

**Check**

* Can you open **Billing** page?
* Do you see any banner like “account verification pending”?

If yes, wait until AWS confirms activation via email.

## 4) You’re inside an AWS Organization with restrictions (SCP)

Less common for personal accounts, but if this is a company-managed AWS account, an Organization policy can deny RDS.

**Hint**The error message often says something like:

* “explicit deny in a service control policy”

If you see that wording, only the org admin can fix it.

# Fastest way to diagnose (do this now)

Please paste **the exact AccessDenied message text** (just the text).  
Or tell me **where** you got it:

* Opening RDS service?
* Clicking “Create database”?
* Another page?

With that, I’ll tell you the exact fix in one shot.

Good news 👍  
**This screenshot does NOT show an Access Denied error.**What you’re seeing is **normal AWS console behavior**, and there’s no permission problem *yet*.

Let me explain exactly what’s happening and what to click next.

## What this screen actually is

You searched for **“amazon RDS”** in the AWS search bar.

AWS shows **multiple related services**, not just one:

* ❌ **Database Migration Service** → NOT what we want
* ✅ **Aurora and RDS (Managed Relational Database Service)** → THIS is correct
* ❌ **AWS Private Certificate Authority** → not related

So there is **no access denial** here.

## ✅ What you must click (important)

Click **👉 “Aurora and RDS”**(the one that says *Managed Relational Database Service*)

This opens the **actual RDS console**.

## What an AccessDenied error would look like (for reference)

You would see messages like:

* “You are not authorized to perform rds:CreateDBInstance”
* “AccessDeniedException”
* Red banner with permission error

👉 **You have NONE of that right now** ✅

## Next expected screen (after clicking Aurora and RDS)

You should see:

* **Amazon RDS dashboard**
* Buttons like:
  + “Create database”
  + “Databases”
  + “Snapshots”

If you see that → permissions are **100% fine**.

## If you *do* get Access Denied after this

Then it would mean:

* sony-admin does not have AdministratorAccess
* Or you’re accidentally logged in as a restricted user

But **based on this screenshot**, everything is OK.

## ✅ Do this now

1. Click **Aurora and RDS**
2. Tell me:
   * Do you see **“Create database”** button?
   * Or do you see an **AccessDenied** message?

Once confirmed, we continue **Phase 2 → RDS creation** step-by-step.  
You’re on the right path — this is normal AWS UI confusion, not an error.

Access denied issue has been resolved now I am in RDS dashboard

Excellent 🎉  
That’s a **big milestone** — AccessDenied issues are usually the hardest AWS hurdle.

Now we are **cleanly inside Phase 2** and can move forward with confidence.

# PHASE 2 — RDS PostgreSQL (Hands-On Creation)

🎯 **Goal:** Create a **minimal-cost PostgreSQL RDS** instance that we will use for MQTT ingestion.

We’ll go **slow + safe**, no skipped steps.

## STEP 2.1 — Click “Create database”

On the **RDS dashboard**, click:  
👉 **Create database**

## STEP 2.2 — Database creation method

Select:

Standard create

Click **Next** / scroll down.

## STEP 2.3 — Engine options

Choose:

PostgreSQL

Version:

PostgreSQL 15.x (default)

## STEP 2.4 — Templates (VERY IMPORTANT)

Select:

Free tier

This keeps cost low and config safe.

## STEP 2.5 — Settings

### DB instance identifier

iot-telemetry-db

### Credentials

Select:

Self managed

Set:

Master username: postgres

Master password: <your strong password>

⚠️ Save this password somewhere secure.

## STEP 2.6 — Instance configuration

Ensure these values:

DB instance class: db.t3.micro

Storage: 20 GB

Enable storage autoscaling: ❌ OFF

## STEP 2.7 — Connectivity (IMPORTANT)

### Virtual Private Cloud (VPC)

Default VPC

### Public access

Yes (for now)

(We will secure this later.)

### VPC security group

Create new

Security group name: rds-postgres-sg

### Port

5432

## STEP 2.8 — Additional configuration

Expand **Additional configuration**:

Initial database name: iotdb

Enable automated backups: ❌ OFF

Enable Enhanced monitoring: ❌ OFF

Deletion protection: ❌ OFF

## STEP 2.9 — Create database

Scroll to bottom → click:  
👉 **Create database**

## ⏳ Wait state

Creation takes **5–10 minutes**.

Status:

Creating → Available

## ✅ STOP HERE & CONFIRM

Once status becomes **Available**, reply with:

1. ✅ DB instance identifier
2. ✅ Endpoint (something like iot-telemetry-db.xxxxxx.us-east-1.rds.amazonaws.com)
3. 🌍 Region
4. ❓ Any warning messages you see

## What comes NEXT (do not jump ahead yet)

Next we will:

* Connect from your laptop
* Create tables
* Insert test data

Only after DB is **100% verified**, we move to:  
👉 **Phase 3 — AWS IoT Core (MQTT ingestion)**

You’re doing this the *right* way. Take your time — tell me when the DB is **Available**.

where to enter port?

Great question — this is an **important AWS nuance**, and you’re not missing anything 👍  
Short answer: **you don’t enter the port on this screen**.

Let me explain clearly.

## Where the PostgreSQL port actually lives

### 1️⃣ RDS itself already has a port

For PostgreSQL, AWS **automatically sets the port to 5432**.

You normally **do NOT need to type it manually** unless you want a custom port (we don’t).

You’ll see the port later in:

* RDS → Databases → your DB → **Connectivity & security**

It will show:  
Port: 5432

## 2️⃣ The port is controlled by the Security Group (THIS is the key)

What really matters is:  
👉 **Security Group inbound rules**

Right now, you’re only **naming the security group** (rds-postgres-sg).  
The **port rule is added after DB creation**.

AWS does this intentionally to avoid exposing ports by accident.

## What you should do RIGHT NOW on this screen

✔️ Leave everything exactly as you have it:

* ✅ Create new security group
* ✅ Name: rds-postgres-sg
* ✅ Availability Zone: No preference
* ❌ Do NOT check “Create an RDS Proxy”
* ❌ Do NOT worry about port here

Then **continue creating the database**.

## Where you WILL open the port (next step)

After DB status becomes **Available**:

### Step A — Open the security group

1. Go to **EC2 → Security Groups**

Select:  
rds-postgres-sg

### Step B — Add inbound rule

* Type: PostgreSQL
* Protocol: TCP
* Port: 5432
* Source:
  + For learning (temporary): **My IP**
  + Later (production): Lambda security group

Click **Save rules**

## Why AWS does it this way (important concept)

* RDS = database engine + network interface
* Security Group = firewall
* Port exposure belongs to **firewall**, not DB config

This separation is core to AWS networking.

## ✅ Summary (so you’re confident)

* ❌ You did NOT miss the port field
* ❌ No mistake on your screen
* ✅ PostgreSQL defaults to port 5432
* ✅ Port is opened later via **Security Group rules**

## What to do next (immediate)

👉 Continue DB creation until status = **Available**

When it’s ready, tell me:

* DB status
* I’ll guide you **exactly** through opening port 5432 safely

You’re doing everything correctly — this is a classic AWS learning moment.

what to do for monitoring?

Great that you paused here 👍 — **this screen affects cost**, so it’s exactly the right place to ask.

## Short answer (what to choose)

For **this learning + ingestion project**, do this:

### ✅ Select

* **Database Insights – Standard**
* **Enable Performance Insights** ✅
* **Retention period: 7 days**
* **AWS KMS key: (default) aws/rds**

### ❌ Do NOT select

* Database Insights – Advanced
* Any longer retention
* Custom KMS key

Then **continue**.

## Why this is the right choice (important context)

### Database Insights – Standard

* Gives you **basic visibility** (CPU, DB load, waits)
* Enough to debug:
  + Slow inserts
  + Lambda connection issues
* **Minimal extra cost** (often $0–very low)

### Database Insights – Advanced ❌

* Fleet-level monitoring
* Long-term retention
* **Unnecessary for a single DB**
* **Extra cost** → not needed now

## Performance Insights — should you enable it?

**Yes**, but **standard only**.

Why:

* When we connect Lambda later, you’ll be able to see:
  + Connection spikes
  + Insert bottlenecks
* Helps learning how real DBs behave under ingestion load

Retention:

* **7 days is perfect**
* Longer = unnecessary cost

## Encryption (KMS)

* Default key aws/rds is:
  + Secure
  + Free
  + AWS-managed
* Custom keys = extra management + cost

## What happens if you disable monitoring completely?

Nothing breaks — but:

* Debugging becomes blind
* Hard to learn DB behavior

So **Standard monitoring is the best learning tradeoff**.

## Final checklist for this screen

Before moving on, verify:

* 🔵 Database Insights – Standard (selected)
* ✅ Performance Insights enabled
* ⏱ Retention = 7 days
* 🔐 KMS = default aws/rds

Then scroll down and continue DB creation.

## Next step (after DB becomes Available)

We will:

1. Open port **5432** in security group
2. Connect from your laptop
3. Create tables

You’re doing everything **exactly like a real cloud engineer** would. Keep going 👍

ok now I have created a database in aws RDS

Perfect 🎉  
Nice work — **Phase 2.1 (RDS creation) is DONE**.

Now we move into **Phase 2.2: Make the database usable**.  
This has **3 clear steps**, and we’ll do them one by one.

# PHASE 2.2 — Prepare RDS for Use

## STEP 1 — Verify DB details (2 minutes)

Go to **RDS → Databases → click your DB instance**

Confirm these values (just check, don’t change):

* **Status**: Available
* **Engine**: PostgreSQL
* **DB name**: iotdb
* **Port**: 5432

**Endpoint**: looks like  
iot-telemetry-db.xxxxx.us-east-1.rds.amazonaws.com

📌 Copy the **Endpoint** — we’ll need it.

## STEP 2 — Open port 5432 (Firewall rule)

Right now the DB exists, but **nothing can connect to it yet**.

### 2.1 Open Security Group

1. Go to **EC2 → Security Groups**

Click:  
rds-postgres-sg

### 2.2 Add inbound rule

Click **Edit inbound rules → Add rule**

Set:

Type: PostgreSQL

Protocol: TCP

Port range: 5432

Source: My IP

💡 This allows **only your laptop** to connect (safe for learning).

Click **Save rules**.

## STEP 3 — Connect to the database (local machine)

Tell me what you’re using:

* macOS / Windows / Linux?

### Option A — Using psql (recommended)

Example:

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

You’ll be prompted for the password you set during creation.

### Option B — Using a GUI

* pgAdmin
* DBeaver
* TablePlus

(All work fine.)

## STEP 4 — Create tables (Already Created in phase 1)

Run this SQL:

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);

If this runs without errors → 🎯 **DB is ready for ingestion**.

## ✅ Checkpoint (reply with this)

Please tell me:

1. ✅ Were you able to connect to RDS?
2. ✅ Tables created successfully?
3. 🖥 OS you’re using (Mac / Windows / Linux)

Once confirmed, we move to the **fun part** 🚀

## Step 3. MQTTX connection settings

In MQTTX:

* Broker: AWS IoT endpoint (TLS)
* Port: 8883
* TLS certs: device certificate + private key + root CA
* Client ID: any unique id (ex: mqttx-sony)

✅ Verification (IoT side)  
Use AWS IoT Core **MQTT Test Client**:

* Subscribe to devices/#
* Publish from MQTTX to devices/esp32\_001/telemetry
* You must see the payload appear.

✅ Output of Phase 2:

* MQTTX → IoT Core is working.

# Phase 3 — IoT Rule + Lambda ingestion (routing + insert)

This is where you had the key real-world issue:

“When I publish using MQTTX it is not inserted into the database while Lambda console test works”

That means: **Lambda→DB was fine**, but **IoT Rule wasn’t matching / triggering**.

## Step 1. Create IoT Rule with the correct SQL (this was the fix)

You ended up using:

SELECT device\_id, timestamp, metrics

FROM 'devices/#'

WHERE topic(2) = 'telemetry'

### Why it works (important)

For topic: devices/esp32\_001/telemetry

* topic(0) = devices
* topic(1) = esp32\_001
* topic(2) = telemetry ✅

So the rule triggers only when the 3rd token is telemetry.

## Step 2. Rule Action → Invoke Lambda

In IoT Rule action:

* Choose your ingestion Lambda
* Allow permission when AWS asks

## Step 3. Lambda logic (what yours must do)

Lambda receives event (from IoT Rule) then:

1. Read device\_id, timestamp, metrics
2. Convert timestamp → insert into ts
3. Insert into telemetry

Insert SQL (conceptually):

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

VALUES ($1, $2, $3);

✅ Verification (this is exactly what you did)

* Lambda console “Test” worked → data inserted ✅  
  So Lambda code + DB credentials were correct.

✅ Output of Phase 3:

* IoT Rule triggers Lambda
* Lambda inserts rows

# Phase 4 — Networking + DB access + troubleshooting (the hard part you solved)

This phase is exactly based on your errors:

### Error you got

psql -h <rds-endpoint> -U postgres -d iotdb -p 5432

psql: error: connection ... failed: Operation timed out

This means: **network path to DB is blocked** (security group / subnet / routing).

You also ran:

curl ifconfig.me

curl: (28) Failed to connect ...

That indicates your current network environment could not reach that site at that moment (hotel Wi-Fi / ISP restriction / firewall). Not the main AWS issue, but it affected “find my public IP” troubleshooting.

## Step 1. Understand the two ways to access RDS

### Option A (Private DB — recommended)

* RDS is private
* Only Lambda (in same VPC) can access it
* Your laptop cannot directly psql unless you use:
  + Bastion host, or
  + SSM Session Manager jump box, or
  + VPN into VPC

### Option B (Public DB — dev only)

* RDS is public
* Add your laptop public IP /32 to inbound
* Then psql works directly

In your case, you were switching between:

* “Laptop IP worked earlier”
* then “now it has VPC IP”  
  Meaning you changed networking (subnets/VPC config) and direct access behavior changed.

## Step 2. The Security Group rule that mattered (your question)

You asked:

“are you sure changes should be done in rds-postgres-sg because in the lambda …”

Yes — **the DB is the server**, so inbound must be opened on **RDS SG**.

### Correct SG setup (what you ended up needing)

**On rds-postgres-sg inbound:**

1. Allow Postgres 5432 from **Lambda security group**

* Source: lambda-ingest-sg (SG-to-SG)

1. (Optional) Allow Postgres 5432 from **your laptop public IP**

* Source: x.x.x.x/32

That’s why later you said:

“working. both VPC and local IP.”

✅ That means your RDS SG had both:

* Lambda SG access (for pipeline)
* Your laptop IP access (for verification)

## Step 3. Why MQTTX didn’t insert but Lambda test did

This was resolved by **IoT Rule SQL** and **topic matching**.

Fix was:

* Keep policy as you shared (connect/publish/subscribe/receive)
* Update IoT Rule SQL to:

SELECT device\_id, timestamp, metrics

FROM 'devices/#' WHERE topic(2)='telemetry'

Now MQTTX publishes to devices/<id>/telemetry → Rule triggers → Lambda inserts.

## Step 4. Final verification steps (end-to-end)

Do these in order every time.

### A) Verify MQTT reaches IoT Core

AWS IoT Test Client:

* Subscribe: devices/#
* Publish from MQTTX
* Confirm message appears

### B) Verify IoT Rule is firing

IoT Core → Message Routing → Rules → your rule

* Check metrics / monitoring
* “Rule triggered / actions invoked” should increment

### C) Verify Lambda invoked

CloudWatch Logs for Lambda:

* New log stream appears
* No DB connection errors

### D) Verify rows in Postgres

Run:

SELECT device\_id, ts, metrics, received\_at

FROM telemetry

ORDER BY received\_at DESC

LIMIT 20;

# The exact “working configuration” summary (what you ended with)

✅ Topic: devices/{device\_id}/telemetry  
✅ IoT Policy: allow connect/publish/subscribe/receive  
✅ IoT Rule SQL:

SELECT device\_id, timestamp, metrics

FROM 'devices/#'

WHERE topic(2) = 'telemetry'

✅ RDS SG inbound:

* 5432 from Lambda SG
* 5432 from laptop public IP (optional for direct psql)  
  ✅ Verified:
* Lambda console test inserts ✅
* MQTTX publish inserts ✅
* psql works when inbound allows laptop IP ✅

If you paste your **Lambda event sample** (the payload structure you see in Lambda from IoT Rule), I’ll also write the **exact Lambda “event parsing → insert mapping” section** (so your document has zero gaps).