Great! We’ll set this up step by step. Let's begin with Step 1.

**✅ Step 1: Create a VPC**

We’ll create a new Virtual Private Cloud (VPC) that will be used by RDS and Lambda (if needed in a private setup). This is optional if you want to use default VPC, but for learning and control, we’ll create a new one.

**🔧 Steps to Create a VPC (via AWS Console):**

1. **Login** to your AWS Console.
2. Go to the **VPC Dashboard**:
   * Use the search bar at the top → type VPC → click on **VPC**.
3. On the left, click on **"Your VPCs"** → then click **"Create VPC"**.
4. In the **Create VPC** form:
   * **Name tag**: my-custom-vpc
   * **IPv4 CIDR block**: 10.0.0.0/16
   * Leave **IPv6** disabled for now.
   * **Tenancy**: Default
5. Click **Create VPC**.

Let me know once you’ve done this or if you face any issue — then we’ll proceed to **Step 2: Create Subnets**.

Awesome! ✅ VPC is ready.

**✅ Step 2: Create Public and Private Subnets**

We’ll now create **two subnets** inside your VPC:

* A **public subnet** (used for the NAT gateway and Lambda if needed).
* A **private subnet** (used for RDS to keep it secure).

**🔧 Steps to Create Subnets:**

1. Go back to the **VPC Dashboard**.
2. In the left menu, click **Subnets** → then click **Create Subnet**.

**📥 Public Subnet**

**Configure as follows:**

* **Name tag**: public-subnet-1
* **VPC ID**: Select my-custom-vpc (the one you created).
* **Availability Zone**: Choose one (e.g., ap-south-1a for Mumbai region).
* **IPv4 CIDR block**: 10.0.1.0/24

✅ Click **Create subnet**

**📥 Private Subnet**

Repeat the same process:

* **Name tag**: private-subnet-1
* **VPC ID**: Same as above.
* **Availability Zone**: (Choose same or different — for high availability, choose ap-south-1b)
* **IPv4 CIDR block**: 10.0.2.0/24

✅ Click **Create subnet**

After this, let me know once both subnets are created. Then we’ll move to **Step 3: Create Internet Gateway and Route Table for Public Access**.

Perfect! ✅ Subnets are ready.

**✅ Step 3: Create and Attach an Internet Gateway (for Public Access)**

The **Internet Gateway (IGW)** allows public internet access for resources in the **public subnet** (like NAT Gateway, or debugging from Lambda if needed).

**🔧 Steps to Create Internet Gateway:**

1. Go to the **VPC Dashboard**.
2. Click on **Internet Gateways** → then **Create Internet Gateway**.

**Fill in:**

* **Name tag**: my-igw
* ✅ Click **Create Internet Gateway**.

1. After creation, click on the new IGW → click **Actions → Attach to VPC**.
   * Select my-custom-vpc
   * ✅ Click **Attach Internet Gateway**

**✅ Step 4: Update the Route Table for Public Subnet**

We need to ensure that traffic from the public subnet knows how to reach the internet via the Internet Gateway.

**🔧 Create and Associate Route Table**

1. In the **VPC Dashboard**, go to **Route Tables** → Click **Create Route Table**

**Fill in:**

* **Name tag**: public-route-table
* **VPC**: my-custom-vpc
* ✅ Click **Create**

1. After creation:
   * Click on the route table → Go to **Routes** tab → Click **Edit routes**
   * Click **Add route**
     + **Destination**: 0.0.0.0/0
     + **Target**: Select the Internet Gateway (my-igw)
   * ✅ Click **Save changes**
2. Go to **Subnet associations** tab → Click **Edit subnet associations**
   * Select public-subnet-1
   * ✅ Save

Let me know once the route table is associated and the IGW is working. Then we’ll proceed to **Step 5: NAT Gateway + Private Route Table (optional for Lambda/RDS to access internet)** or skip it if not required.

Awesome! ✅ You now have:

* A VPC
* Public & Private subnets
* Internet Gateway
* Public subnet connected to the internet

**✅ Step 5 (Optional but Recommended): Create NAT Gateway & Private Route Table**

If:

* Your **RDS** or **Lambda (in private subnet)** needs **outbound internet access** (for downloading Python packages, etc.), then we need a **NAT Gateway**.
* If **you’ll place Lambda in public subnet**, this step is optional.

✅ **Let’s do it anyway** to follow best practices.

**🔧 Create Elastic IP for NAT Gateway**

1. In AWS Console, go to **EC2 Dashboard** → Left menu → **Elastic IPs**
2. Click **Allocate Elastic IP address**
   * Leave settings as default → click **Allocate**
3. Select the new EIP → Click **Actions → Associate Elastic IP**
   * We will do this when we create the NAT Gateway next.

**🔧 Create NAT Gateway**

1. Go to **VPC Dashboard** → **NAT Gateways** → Click **Create NAT Gateway**

**Fill in:**

* **Name**: my-nat-gateway
* **Subnet**: public-subnet-1
* **Elastic IP**: Select the one you just created
* ✅ Click **Create NAT Gateway**

🕐 Wait until the NAT gateway becomes **Available** (status will change from "Pending").

**🔧 Create Private Route Table (for Private Subnet to use NAT)**

1. Go to **Route Tables** → Click **Create Route Table**

**Fill in:**

* **Name tag**: private-route-table
* **VPC**: my-custom-vpc
* ✅ Click **Create**

1. Click on your new route table:
   * Go to **Routes** tab → **Edit routes**
   * **Add route**:
     + **Destination**: 0.0.0.0/0
     + **Target**: Select the **NAT Gateway** (my-nat-gateway)
   * ✅ Save changes
2. Go to **Subnet associations** → **Edit subnet associations**
   * Select private-subnet-1
   * ✅ Save

✅ Once done, confirm:

* NAT Gateway is active.
* Private subnet route table is set to use the NAT.

Let me know when you’re done — next we’ll set up **IAM roles & permissions**.

Perfect — skipping NAT Gateway is totally fine for now ✅  
(Just remember: if you place Lambda or RDS in a **private subnet**, they won’t be able to access the internet unless NAT is added.)

**✅ Step 6: Setup IAM Roles and Permissions**

We now need to:

1. Create an **IAM Role** for **Lambda** to access **S3 and RDS**
2. Optionally, create an **IAM User** (not required if you're using root or a role-based access)

**🔧 Step 6.1: Create IAM Role for Lambda**

This role lets Lambda:

* Read from S3 (CSV file)
* Connect to RDS (we'll allow basic access)

**👣 Create Role:**

1. Go to **IAM Dashboard** → Click **Roles** → **Create role**
2. **Trusted entity type**: Select **AWS service**
3. **Use case**: Select **Lambda** → Click **Next**

**👣 Attach Permissions Policies:**

Search and select these policies:

* ✅ AmazonS3ReadOnlyAccess
* ✅ AmazonRDSFullAccess *(for demo; we'll later restrict it if needed)*
* (Optional: add CloudWatchLogsFullAccess if you want Lambda logs to go to CloudWatch)

Click **Next**

**👣 Name the Role:**

* **Role name**: lambda-s3-rds-role
* ✅ Create Role

Once done, let me know.

Next step will be:  
📦 **Create S3 bucket & upload CSV**  
🏗️ Then we’ll move to RDS and Lambda setup.

Great! ✅ IAM Role for Lambda is now ready.

**✅ Step 7: Create S3 Bucket and Upload Your CSV**

We’ll now create the S3 bucket where your CSV file will be stored. Lambda will later read this CSV to insert data into RDS.

**🔧 Steps to Create S3 Bucket:**

1. Go to **S3 Dashboard**
2. Click **Create bucket**

**Fill in:**

* **Bucket name**: Must be globally unique. Example: pranoy-lambda-csv-bucket
* **Region**: Same as your VPC (e.g., ap-south-1 for Mumbai)

**Important Settings:**

* **Uncheck "Block all public access"** — keep it checked (we don’t want it public).
* Leave everything else as default.
* ✅ Click **Create bucket**

**📤 Upload CSV File**

1. Click on your new bucket
2. Click **Upload**
   * Upload your CSV file (e.g., data.csv)
   * ✅ Click **Upload**

Let me know once:

* S3 bucket is created
* CSV file is uploaded

Then we’ll proceed to **Step 8: Create Amazon RDS (PostgreSQL or MySQL)**.

Perfect! ✅ S3 is ready and CSV is uploaded.

**✅ Step 8: Create Amazon RDS (PostgreSQL or MySQL)**

We’ll now create an RDS database where the Lambda function will insert data from your CSV.

Let’s go with **PostgreSQL**, since it has easier Python library support (psycopg2), and is widely used.

**🔧 Steps to Create RDS PostgreSQL (Free Tier Eligible)**

1. Go to **RDS Dashboard**
2. Click **Create database**

**Step-by-Step Configuration:**

* **Choose a database creation method**: Standard Create
* **Engine options**:
  + Engine type: ✅ **PostgreSQL**
  + Version: Select the default Free Tier–eligible version (like 15.4-R1 or similar)
* **Templates**: ✅ **Free Tier**

**Settings:**

* **DB instance identifier**: mydb-instance
* **Master username**: postgres
* **Password**: Choose a strong password (e.g., Postgres@12345)
* ✅ Save the password somewhere — you’ll need it in Lambda.

**DB Instance Size (Free Tier):**

* **DB instance class**: db.t3.micro (free tier)
* **Storage**: 20 GB (default is fine)

**Connectivity:**

* **VPC**: Select your custom VPC (my-custom-vpc)
* **Subnet group**: Leave default or auto-created
* **Public access**: ✅ **Yes** *(for now, so you can test and connect from your local or Lambda)*
* **VPC security group**: Create new or choose existing

**Additional Config:**

* Leave all other settings default (no backup, no monitoring, etc.)
* ✅ Click **Create database**

⏱️ It may take 5–10 minutes to become "Available".

Once created:

* Go to **Connectivity & Security** section
* Note down:
  + **Endpoint**
  + **Port** (default: 5432)

Let me know once your RDS instance is available and you can see the endpoint. Then we’ll move to:  
✅ **Step 9: Connect RDS to Lambda + Write the Code**.

Excellent observation ✅ — you're absolutely right.

Using **PostgreSQL (psycopg2)** in AWS Lambda **requires a Lambda Layer** or a container image **because it's a compiled binary**, and cannot be simply imported like in local Python.

So, to **keep things simple and faster**, yes — let's go with **MySQL** instead for this project.

**✅ Why MySQL is Easier for Lambda:**

* Python's pymysql package is **pure Python**, so you can directly include it in your Lambda zip — **no Lambda Layer needed**.
* No need to build or deploy custom binaries or container images.
* Clean and quick for Free Tier prototyping.

**✅ What To Do Now:**

1. **Delete the PostgreSQL RDS instance** (if already created) — no worries, it’s free tier, and we’re early.
2. **Create new MySQL RDS instance** with these settings:

**🔧 RDS MySQL Setup**

* **Engine**: ✅ MySQL
* **Version**: 8.0 (latest Free Tier–eligible)
* **Template**: Free Tier

**DB Settings:**

* **DB identifier**: mysql-instance
* **Username**: admin
* **Password**: Admin@12345 (choose your own)
* **DB Instance class**: db.t3.micro
* **Storage**: 20 GB (default)

**Connectivity:**

* **VPC**: my-custom-vpc
* **Public access**: ✅ Yes
* **VPC security group**: Create new (or allow inbound port 3306)
  + We’ll configure the security group after creation if needed.

✅ Create the instance and wait until status = **Available**

Once done:

* Share the **endpoint**, and confirm the **username/password**
* I’ll help you create:
  + ✅ The table schema in MySQL for your CSV
  + ✅ The Lambda function to read from S3 and insert to RDS

Let me know when your MySQL instance is ready.

Perfect — let's configure **connectivity** for your **MySQL RDS instance** so that:

* AWS Lambda can connect to it
* (Optional) you can connect from your local machine using tools like DBeaver or MySQL Workbench

**✅ Step 1: Modify Security Group to Allow MySQL (Port 3306)**

1. Go to the **EC2 Dashboard** → **Security Groups** (left menu)
2. Find the **Security Group** attached to your **RDS instance**:
   * You can find this from the **Connectivity & Security** tab in your RDS instance

**🔧 Edit Inbound Rules:**

1. Select the correct Security Group → Click **Edit inbound rules**
2. Add the following rule:

| **Type** | **Protocol** | **Port Range** | **Source** |
| --- | --- | --- | --- |
| MySQL/Aurora | TCP | 3306 | **Your IP** (or 0.0.0.0/0 for testing) |

🔒 **Important Note**: If you use 0.0.0.0/0, it makes your DB publicly accessible — use **only temporarily for testing**. Use **"My IP"** for safer access.

1. ✅ Click **Save rules**

**✅ Step 2: Confirm RDS Public Access Setting**

* Go to **RDS Dashboard** → Your **MySQL instance**
* In the **Connectivity & security** section:
  + **Publicly accessible** should be **Yes**
  + If it's **No**, you need to modify the DB and set it to Yes (this may require a reboot)

**✅ Step 3: (Optional) Test the Connection from Your Local**

You can test it now using:

* **MySQL Workbench**, **DBeaver**, or terminal:

mysql -h <your-endpoint> -P 3306 -u admin -p

* Enter password when prompted.

Let me know once this is done ✅  
Next, I’ll help you:

1. Design the **MySQL table** for your CSV
2. Write the **Lambda function** to read from S3 and insert into RDS.

Great questions — let me walk you through both:

**✅ Step 2: Confirm "Publicly Accessible" Setting for RDS**

1. Go to **RDS Dashboard** in AWS Console.
2. Click on your **MySQL instance** (name like mysql-instance).
3. Scroll down to the **"Connectivity & security"** section.

**Look for**:

* **Publicly accessible**: This should say **Yes**

**❌ If It Says "No" (and you want to change it):**

1. Click the **"Modify"** button at the top of the DB instance page.
2. Scroll down to the **Connectivity** section.
3. Under **Public access**, choose **"Yes"**
4. Scroll to the bottom → Tick ✅ **Apply immediately**
5. Click **Continue** → then **Modify DB instance**

🕐 It may take a few minutes to apply the change. Once done, refresh the page and check that "Publicly accessible" is now **Yes**.

**👤 Where to See Username and Password?**

Unfortunately, **AWS does not show the password again after creation** (for security reasons). But here’s what you can do:

**✅ To check the master username:**

1. Go to your RDS instance page
2. Scroll to **"Configuration"**
3. Look for **Master username** — e.g., admin

**❌ For the password:**

* If you **don’t remember** it, you’ll need to **reset** it:
  + Click **Modify**
  + Scroll to the **Settings** section
  + Enter a **new master password**
  + ✅ Apply immediately
  + Click **Modify DB instance**

Let me know if you want to reset it, and I can guide you through.

Once this is confirmed, let me know:

* ✅ Publicly accessible = Yes?
* ✅ Username known?
* ✅ Password available (or reset)?

Then I’ll help you:  
📊 Create table based on your CSV  
📦 Write the Lambda function to automate the insert.

Perfect, now let’s build the Lambda function! 🛠️  
This function will:

1. Be triggered manually (for now)
2. Fetch the CSV file from your S3 bucket
3. Parse the rows
4. Insert each row into your MySQL employees table in RDS

**✅ Step 2: Create Lambda Function (Python 3.12)**

**🔧 Step-by-Step Setup in AWS Console**

1. Go to **Lambda** → Click **Create function**
2. **Choose Author from scratch**

| **Field** | **Value** |
| --- | --- |
| Function name | s3-to-mysql-loader |
| Runtime | Python 3.12 |
| Execution role | Use existing role |
| Existing role | lambda-s3-rds-role (we created earlier) |

✅ Click **Create Function**

**📦 Step 3: Add pymysql Library (required for MySQL access)**

You can’t use pip install pymysql directly in Lambda console, so do this **locally**:

1. Create a folder called lambda\_package
2. Inside it, create a file named lambda\_function.py (we'll paste code below)
3. Install pymysql in that folder:

cd lambda\_package

pip install pymysql -t .

1. After install, zip everything:

zip -r lambda\_function.zip .

1. Go to AWS Lambda → Your function → Upload the ZIP file

**🧠 Step 4: Lambda Function Code (lambda\_function.py)**

Replace this with your actual:

* host: your RDS endpoint
* user, password: your RDS credentials
* database: demodb
* bucket and file\_key: your CSV S3 details

import boto3

import pymysql

import csv

import os

from io import StringIO

# RDS connection info

rds\_host = "your-rds-endpoint.rds.amazonaws.com"

rds\_user = "admin"

rds\_password = "YourPassword"

rds\_db = "demodb"

# S3 info

bucket = "your-s3-bucket-name"

file\_key = "your-file.csv" # match exact S3 filename

def lambda\_handler(event, context):

# Connect to S3

s3 = boto3.client('s3')

response = s3.get\_object(Bucket=bucket, Key=file\_key)

content = response['Body'].read().decode('utf-8')

csv\_data = csv.reader(StringIO(content))

next(csv\_data) # Skip header

# Connect to RDS MySQL

connection = pymysql.connect(

host=rds\_host,

user=rds\_user,

password=rds\_password,

database=rds\_db,

cursorclass=pymysql.cursors.DictCursor

)

inserted = 0

try:

with connection.cursor() as cursor:

for row in csv\_data:

id, name, department = row

sql = "INSERT INTO employees (id, name, department) VALUES (%s, %s, %s)"

cursor.execute(sql, (id, name, department))

inserted += 1

connection.commit()

finally:

connection.close()

return {

'statusCode': 200,

'body': f'Successfully inserted {inserted} records into RDS.'

}

**🔐 Environment & Permissions Notes**

* Make sure your Lambda **timeout** is set to 1–2 mins
* Memory: 256MB is enough
* VPC Settings: If RDS is in a private subnet, **add Lambda to the same VPC and subnet**, and configure security groups (you can do this in Lambda → Configuration → VPC)

✅ Once the ZIP is uploaded and the function is configured:

* Click **Test** (create a dummy test event)
* Check **CloudWatch logs** for output or any errors

Let me know:

* Once you've zipped and uploaded the Lambda function
* Any errors you face
* Ready to test it live ✅