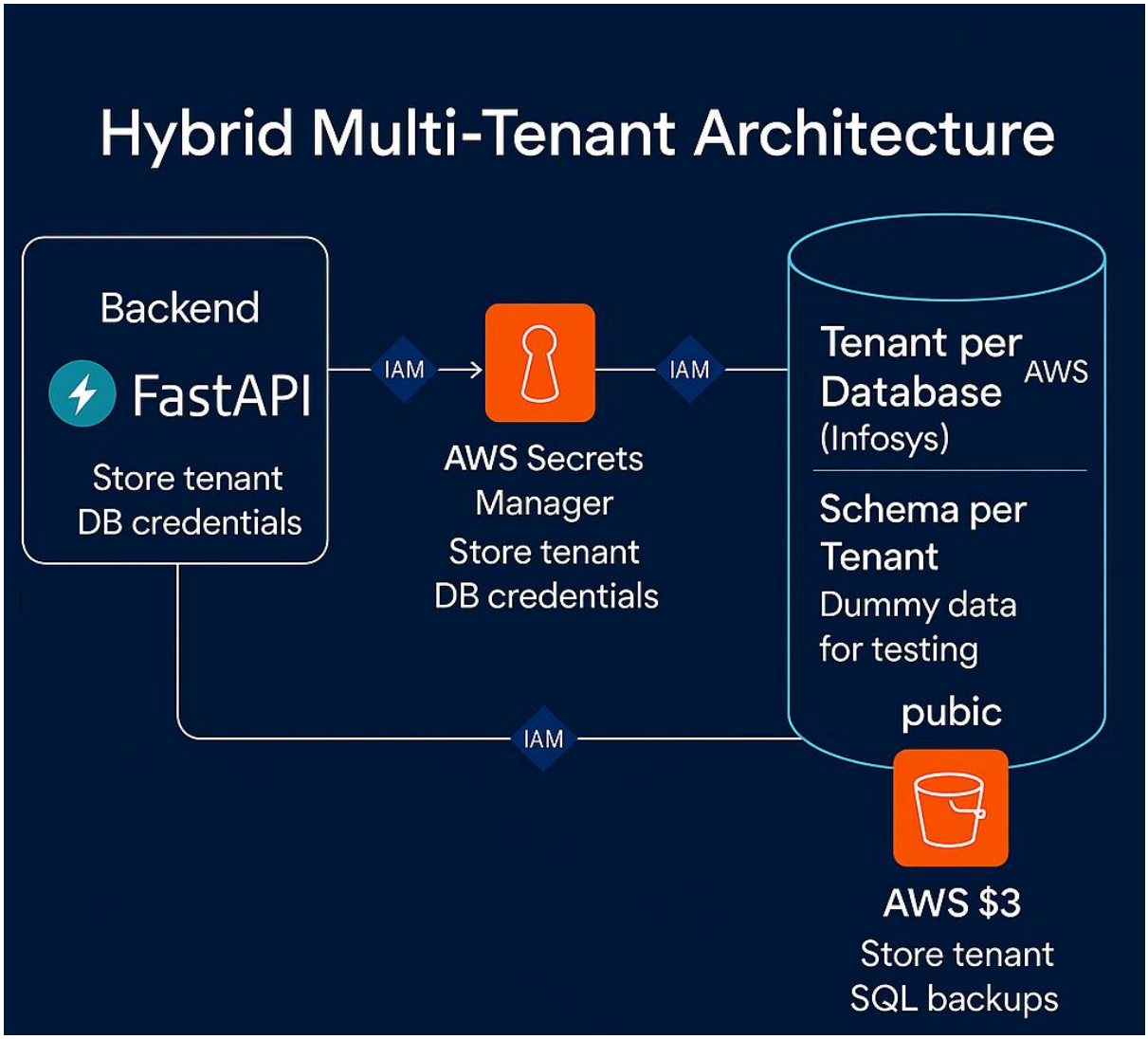
Hybrid Multi-Tenant Architecture (Database-Per-Tenant + Schema per Tenant)



Overview:

This architecture involves deploying a multi-tenant application using a single AWS EC2 instance (VM) that hosts a PostgreSQL database. Each tenant has a dedicated schema within the same PostgreSQL database.

Architecture Type: Hybrid Multi-Tenant SaaS

- ✓ **Single PostgreSQL instance**
- ✓ **Separate database per tenant** (full DB isolation)
- ✓ **Within each DB: schema-per-tenant isolation**

✓ **Through IAM we can Assign Specific roles and User for to connect AWS CLI**

✓ **Backups per tenant uploaded to AWS S3**

✓ **Each tenant's credentials stored securely in AWS Secrets Manager**

➔Architecture Overview

Prerequisites:

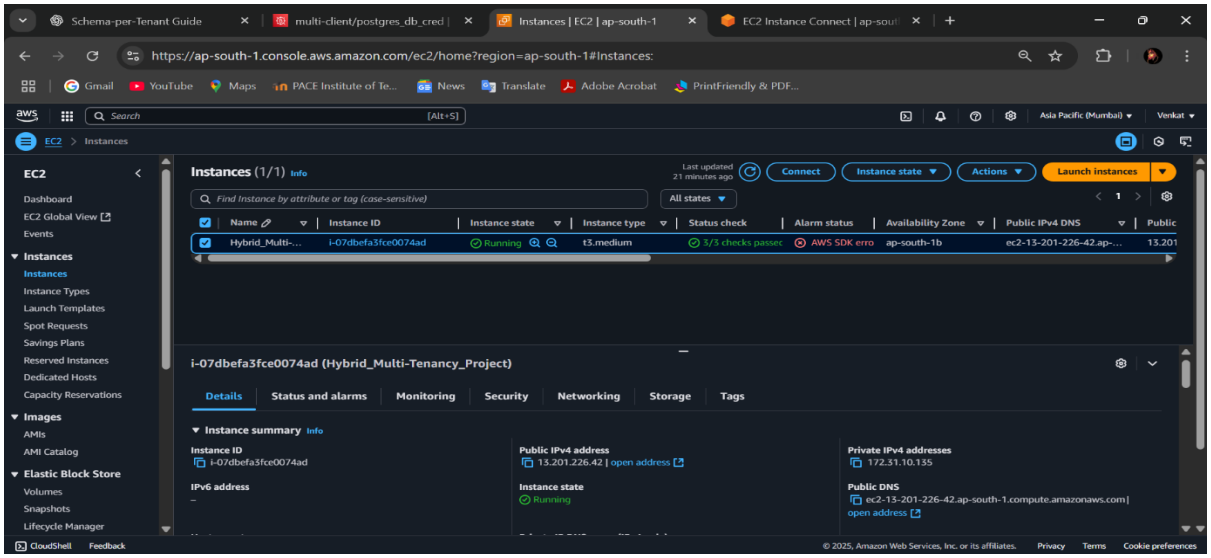
- **AWS EC2 instance (Ubuntu)**
- **PostgreSQL installed**
- **IAM (User & Assign Roles)**
- **Secrets securely stored in AWS Secrets Manager**
- **Backups stored in Amazon S3**
- **FastAPI as backend (on same EC2)**

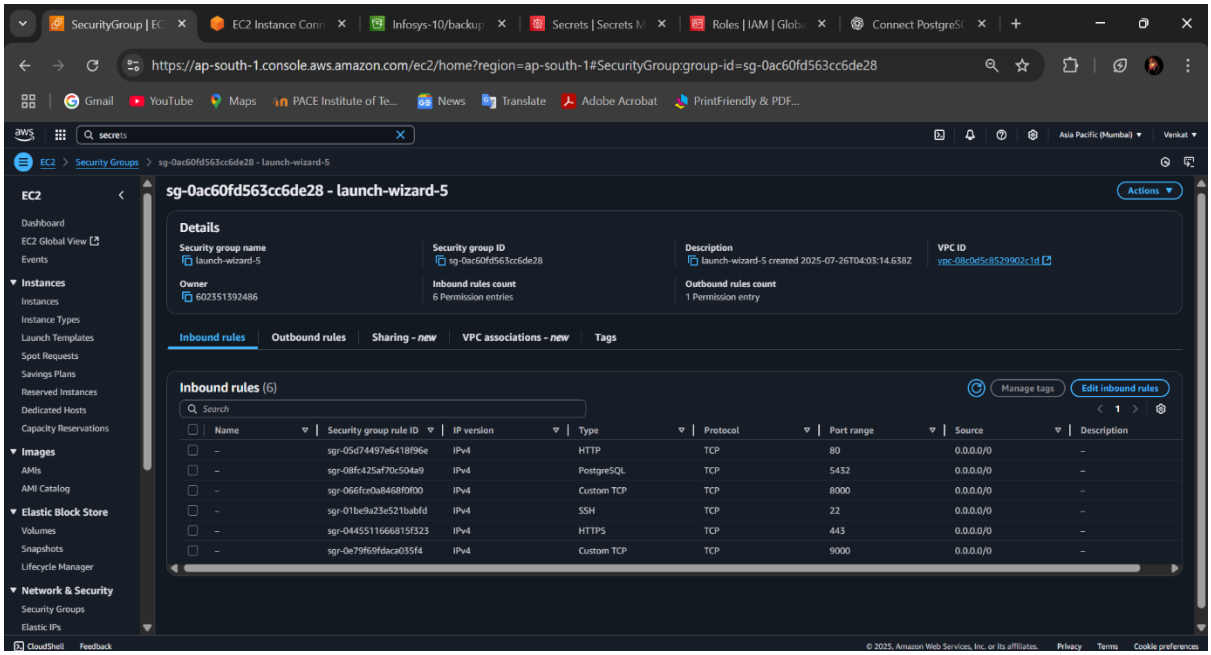
Step-by-Step Setup Guide

➔Step 1: Launch an EC2 Instance

1. Go to **EC2 Dashboard** → Launch Instance
2. Choose **Ubuntu 22.04 LTS**
3. Instance Type: t3.medium or higher (2 vCPU, 4GB RAM)
4. Configure: Security Groups Enable
    - Enable inbound port 22 (SSH), 5432 (PostgreSQL), 80/443 (backend), 8000 (FastAPI)

   Add tags: Name=multi-tenant-db-vm
5. Launch with SSH Key pair (pem or ppk files)





➔IAM:

Step-by-Step: Create IAM Role and Attach to EC2

Step 1: Go to IAM in AWS Console

1. Visit https://console.aws.amazon.com/iam/
2. In the left panel, click **Roles**
3. Click **Create role**

Step 2: Select Use Case

1. **Trusted entity type**: AWS service
2. **Use case**: Choose **EC2**
3. Click **Next**

Step 3: Attach Permissions Policies

In the "Add permissions" section, attach the following two policies:

- **AmazonS3FullAccess**
- **SecretsManagerReadWrite**

You can search and select them using the search bar.

Then click **Next**.

Step 4: Add Role Name and Tags

1. **Role name**: multi-tenant-db
2. (Optional) Tags:
    - Key: project, Value: multi-tenant
    - Key: owner, Value: your-name

Click **Create role**

Step 5: Attach Role to EC2 Instance

Now go to your EC2 dashboard:

1. Select the running EC2 instance
2. Click **Actions → Security → Modify IAM Role**
3. Choose the role you created: multi-tenant-db
4. Click **Update IAM Role**

Your EC2 instance now has permissions to:

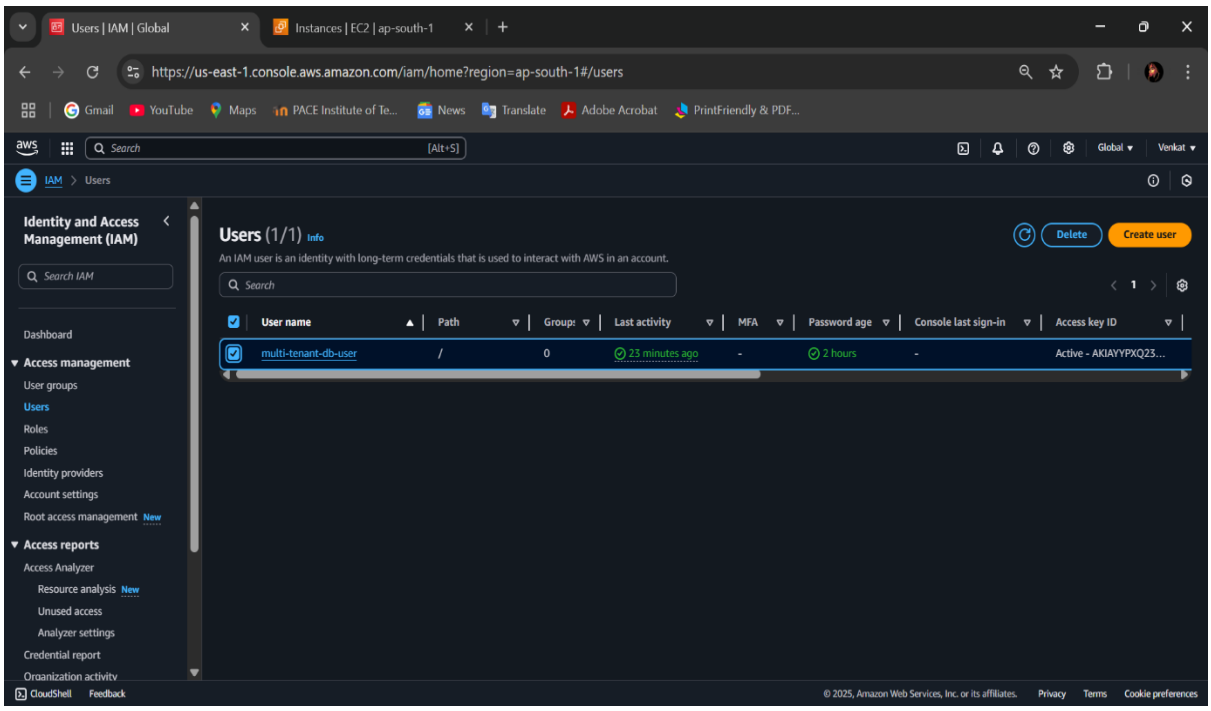
- Upload/download backups from **S3**
- Fetch/store DB credentials using **Secrets Manager**

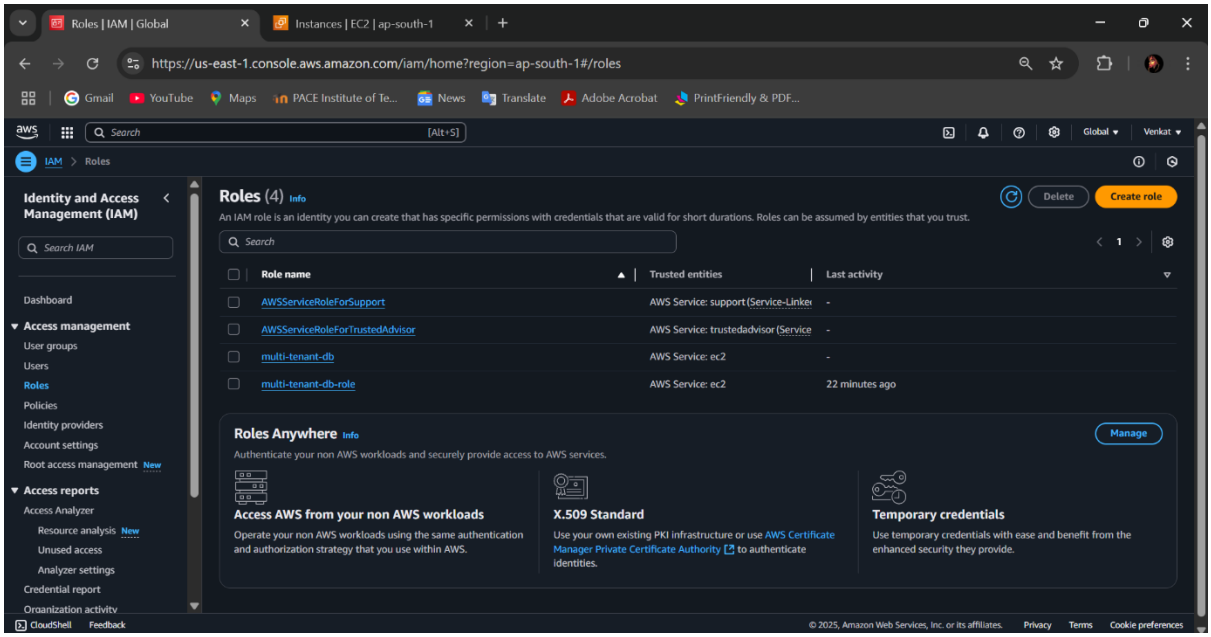
Test Role from EC2

Now back on your EC2 terminal, run:

- `aws sts get-caller-identity`
- If IAM role is correctly attached, you'll get an output like:

```
{
    "UserId": "AROA************",
    "Account": "123456789012",
    "Arn": "arn:aws:sts::123456789012:assumed-
role/EC2MultiTenantDBRole/i-xxxxxxxxx"
}
```





⇨ Create IAM User for to connect **AWS CLI** (Generates ACCESS_KEY & SECRET_ACCESS_KEY)

Confirm AWS CLI & PostgreSQL

⇨ Ensure these are available in your system **(in PATH)**:

psql – PostgreSQL client
pg_dump – for backups
aws – AWS CLI (configured with aws configure)

⇨ You can check with:

```
psql --version
pg_dump --version
aws --version
```

➔Install PostgreSQL on the VM

```
SSH into the instance:
sudo apt update && sudo apt upgrade -y
sudo apt install postgresql postgresql-contrib -y
Secure PostgreSQL:
sudo passwd postgres
sudo -i -u postgres
psql
Create a root DB user:
CREATE USER root_admin WITH SUPERUSER PASSWORD 'your-strong-password';
```

➔Configure PostgreSQL for Remote Access

⇨ Edit config files:

```
sudo vi /etc/postgresql/17/main/postgresql.conf
```

- Change listen_addresses = '*' (To Support ALL or you can ADD PUBLIC-IP for specific Support).

⇨ Then:

sudo vi /etc/postgresql/17/main/pg_hba.conf

- Add: In the Last line

host all all 0.0.0.0/0 md5

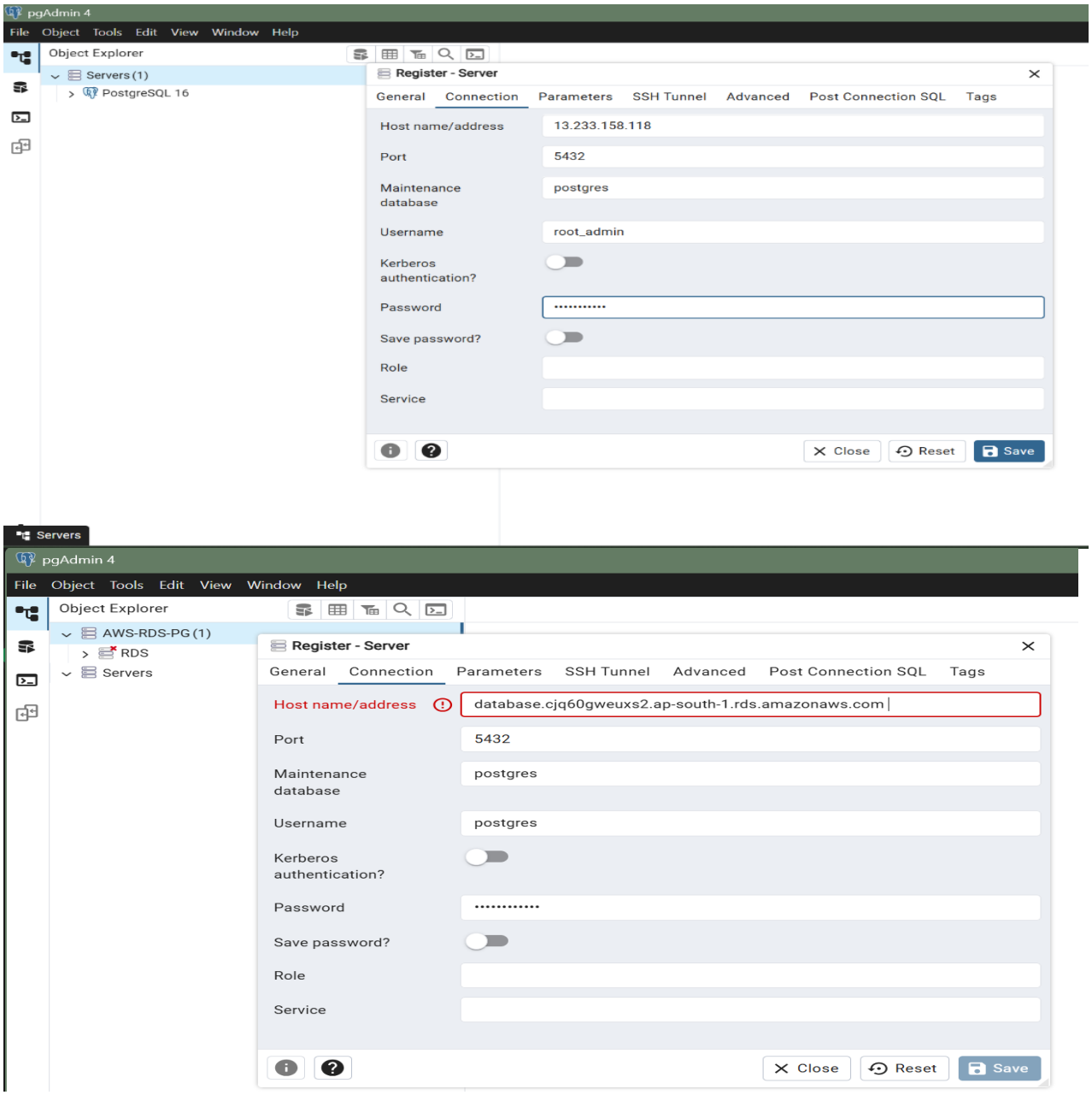
You can replace 0.0.0.0/0 with a specific IP or CIDR for **tighter security** (e.g., 203.0.113.15/32)

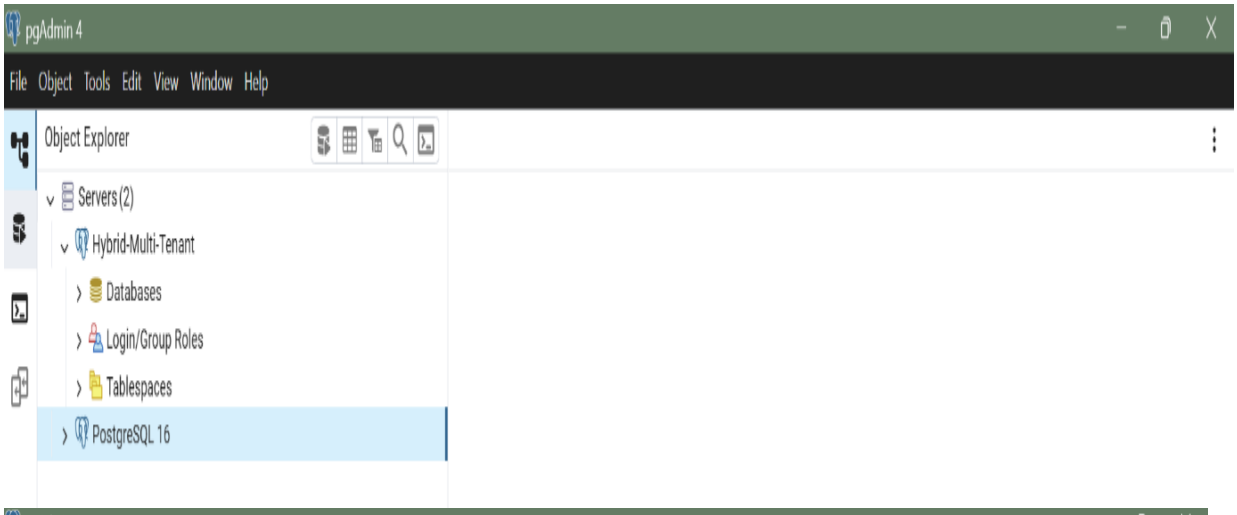
⇨ Restart PostgreSQL:

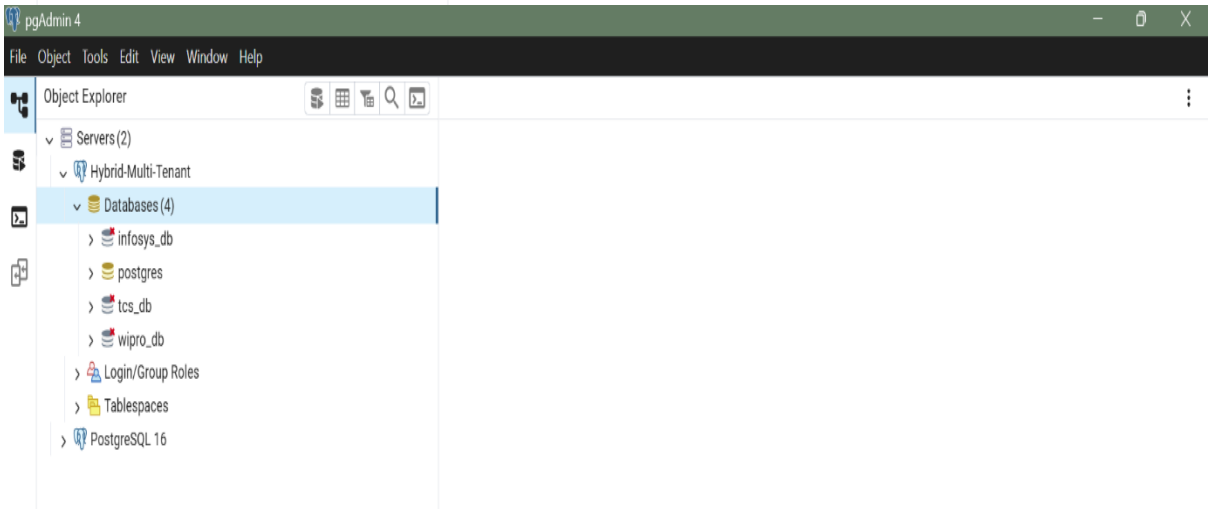
sudo systemctl restart postgresql

sudo systemctl status postgresql (Active Running)

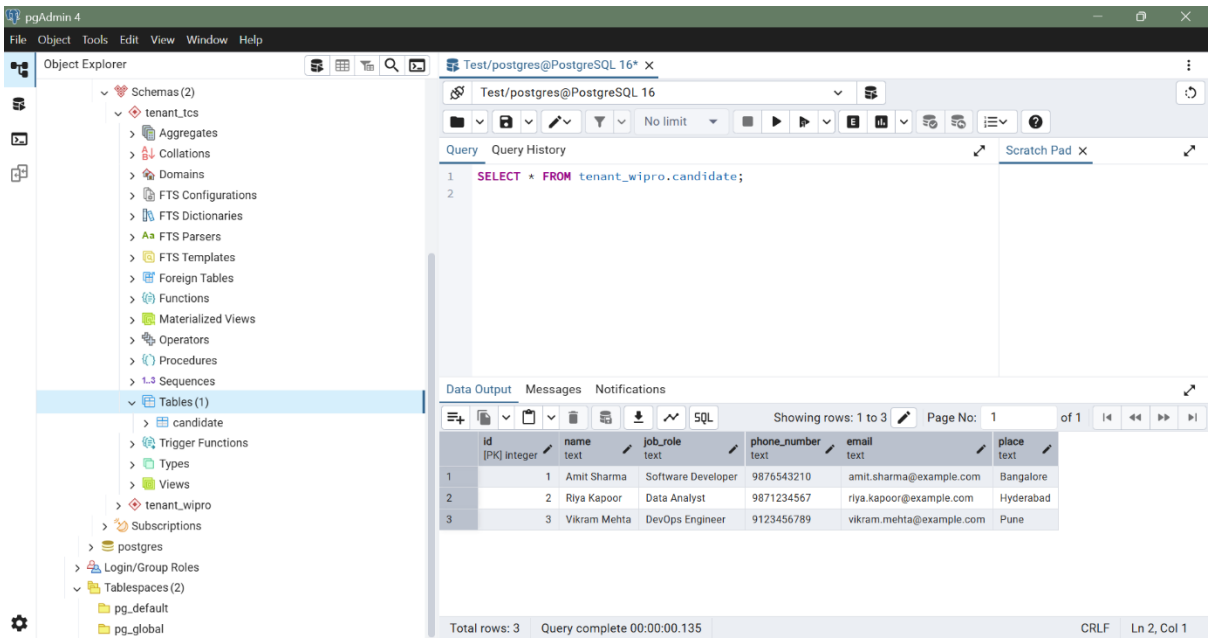
⇨ We can connect through pgadmin4 by AWS RDS EndPoint or PUBLIC_IP

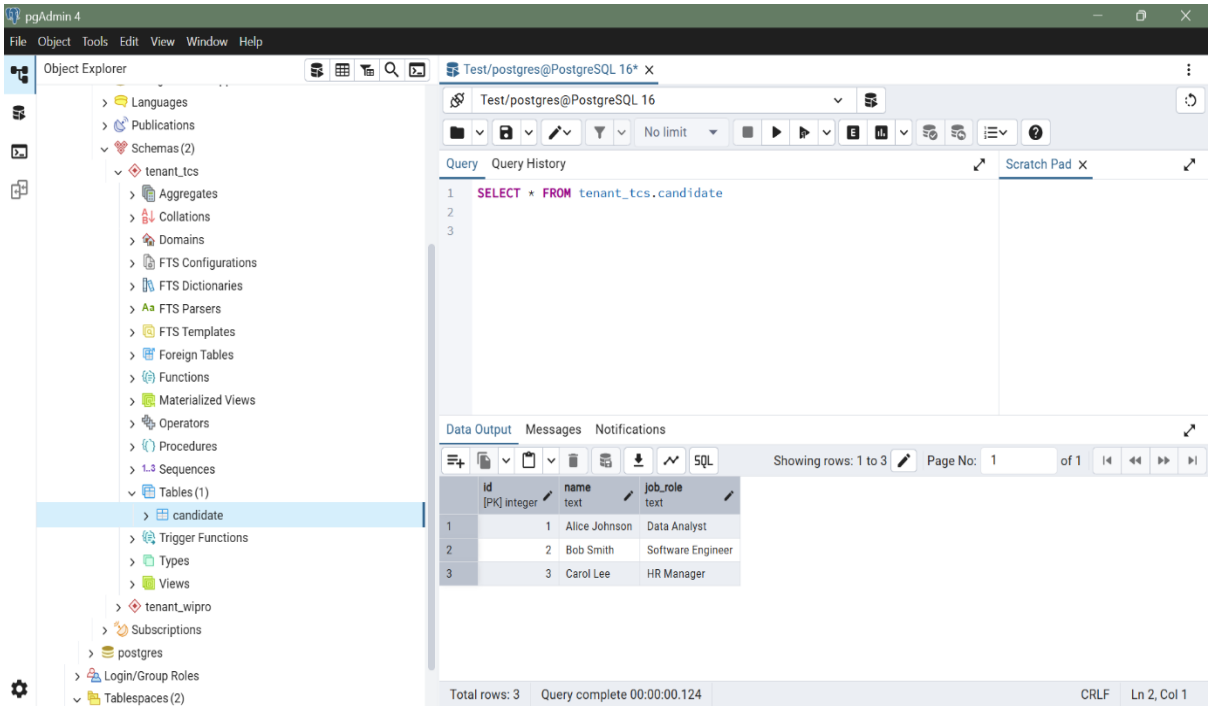






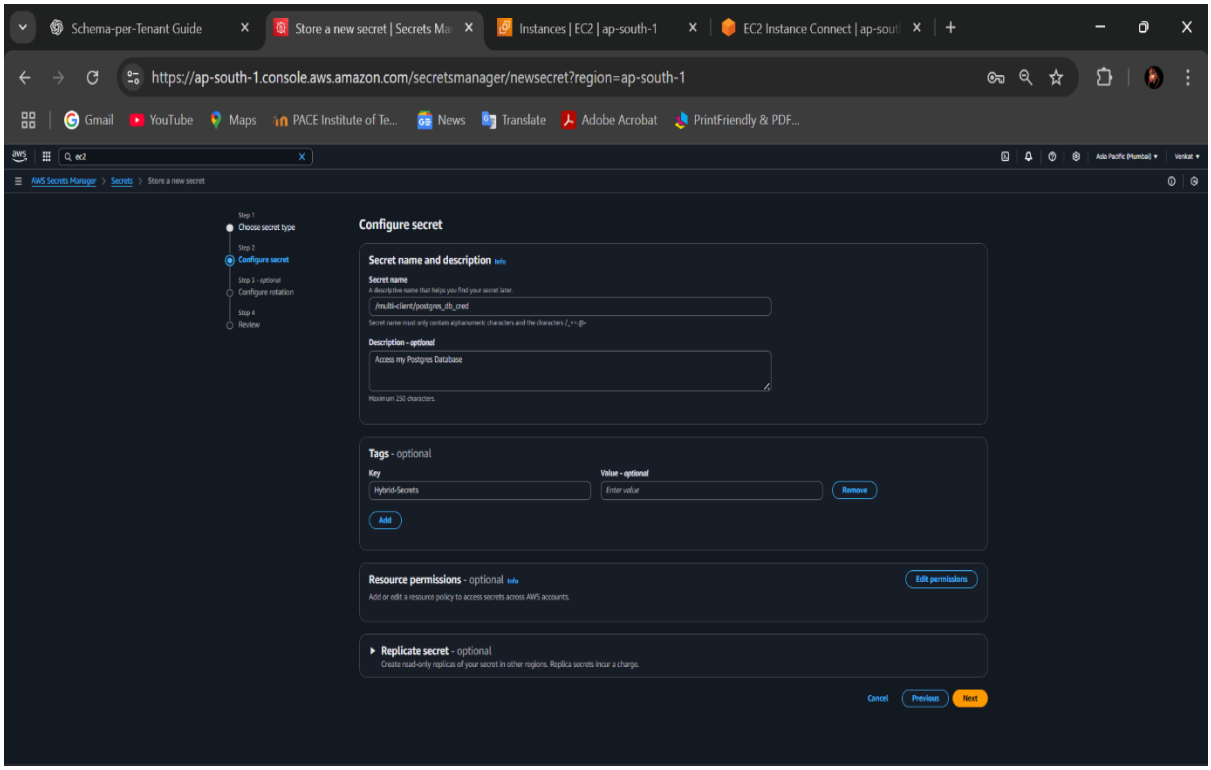
⇨ We installed Single Database in the AWS VM (PostgreSQL) in that created Each Tenant with Database.

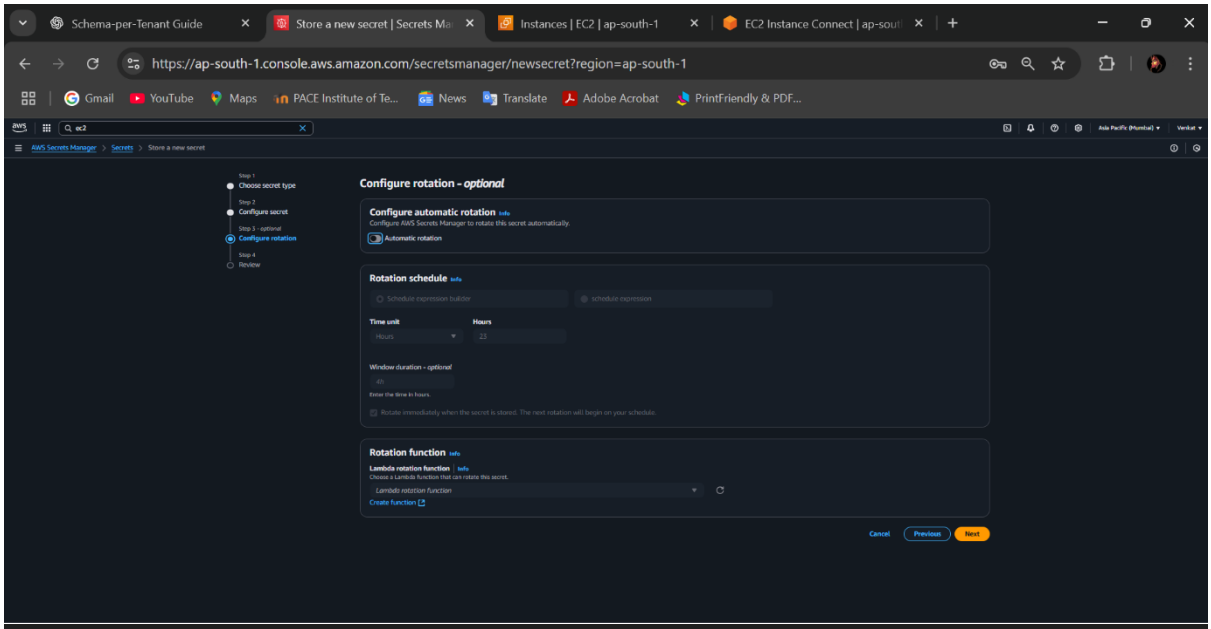


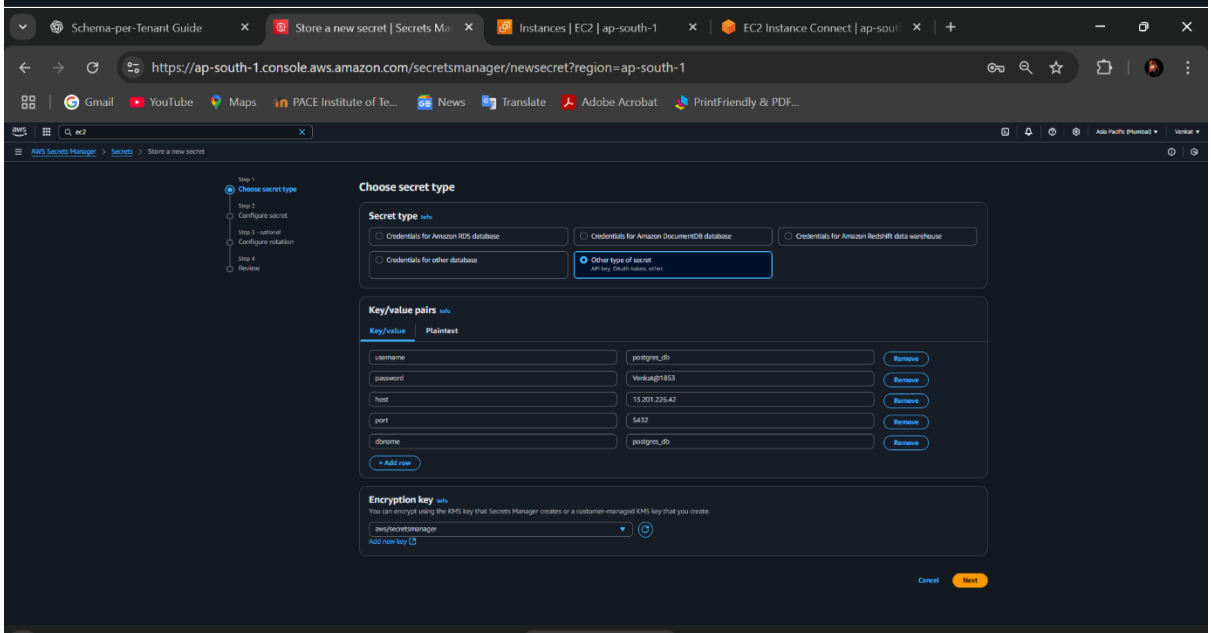


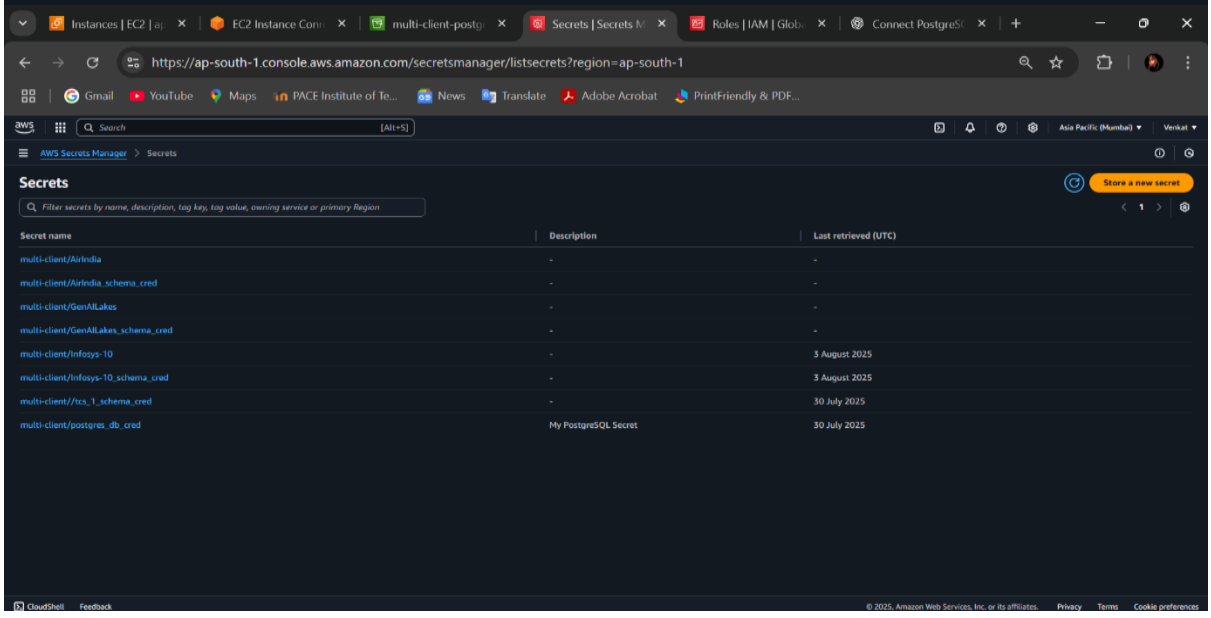
- ⇨ In the Schema -> Tenant -> Tables ->created a tables by using Dummy data Successfully Created.
- ⇨ psql -U postgres -h localhost -d postgres (to check Pgadmin4 is connected or not)

➔Secret Manager:




Schema-per-Tenant Guide
Store a new secret | Secrets Ma
Instances | EC2 | ap-south-1
EC2 Instance Connect | ap-sout
https://ap-south-1.console.aws.amazon.com/secretsmanager/newsecret?region=ap-south-1
Gmail
YouTube
Maps
PACE Institute of Te...
News
Translate
Adobe Acrobat
PrintFriendly & PDF...
AWS Secrets Manager > Secrets > Store a new secret
Step 1 Choose secret type
Step 2 Configure secret
Step 3 - optional Configure rotation
Step 4 Review
Configure rotation - optional
Configure automatic rotation Info
Configure AWS Secrets Manager to rotate this secret automatically.
Automatic rotation
Rotation schedule Info
Schedule expression builder
schedule expression
Time unit
Hours
Hours
23
Window duration - optional
4h
Enter the time in hours.
Rotate immediately when the secret is stored. The next rotation will begin on your schedule.
Rotation function Info
Lambda rotation function Info
Choose a Lambda function that can rotate this secret.
Lambda rotation function
Create function
Cancel
Previous
Next


Schema-per-Tenant Guide
Store a new secret | Secrets Ma
Instances | EC2 | ap-south-1
EC2 Instance Connect | ap-sout
https://ap-south-1.console.aws.amazon.com/secretsmanager/newsecret?region=ap-south-1
Gmail
YouTube
Maps
PACE Institute of Te...
News
Translate
Adobe Acrobat
PrintFriendly & PDF...
AWS Secrets Manager > Secrets > Store a new secret
Step 1 Choose secret type
Step 2 Configure secret
Step 3 - optional Configure rotation
Step 4 Review
Choose secret type
Secret type Info
Credentials for Amazon RDS database
Credentials for Amazon DocumentDB database
Credentials for Amazon Redshift data warehouse
Credentials for other database
Other type of secret
API key, OAuth tokens, other.
Key/value pairs Info
Key/value
Plaintext
username
postgres_db
Remove
password
Venkat@1853
Remove
host
13.201.226.42
Remove
port
5432
Remove
dbname
postgres_db
Remove
+ Add row
Encryption key Info
You can encrypt using the KMS key that Secrets Manager creates or a customer-managed KMS key that you create.
aws/secretsmanager
Add new key
Cancel
Next


Instances | EC2 | ap
EC2 Instance Conn
multi-client-postgr
Secrets | Secrets M
Roles | IAM | Glob
Connect PostgreSC
https://ap-south-1.console.aws.amazon.com/secretsmanager/listsecrets?region=ap-south-1
Gmail
YouTube
Maps
PACE Institute of Te...
News
Translate
Adobe Acrobat
PrintFriendly & PDF...
Search
[Alt+S]
Asia Pacific (Mumbai)
Venkat
AWS Secrets Manager > Secrets
Secrets
Store a new secret
Filter secrets by name, description, tag key, tag value, owning service or primary Region
Secret name
Description
Last retrieved (UTC)
multi-client/AirIndia -
multi-client/AirIndia_schema_cred -
multi-client/GenAILakes -
multi-client/GenAILakes_schema_cred -
multi-client/Infosys-10 - 3 August 2025
multi-client/Infosys-10_schema_cred - 3 August 2025
multi-client/tcs_1_schema_cred - 30 July 2025
multi-client/postgres_db_cred My PostgreSQL Secret 30 July 2025
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

➔ Enable S3 Backups

Install AWS CLI and configure IAM role:

sudo apt install awscli -y

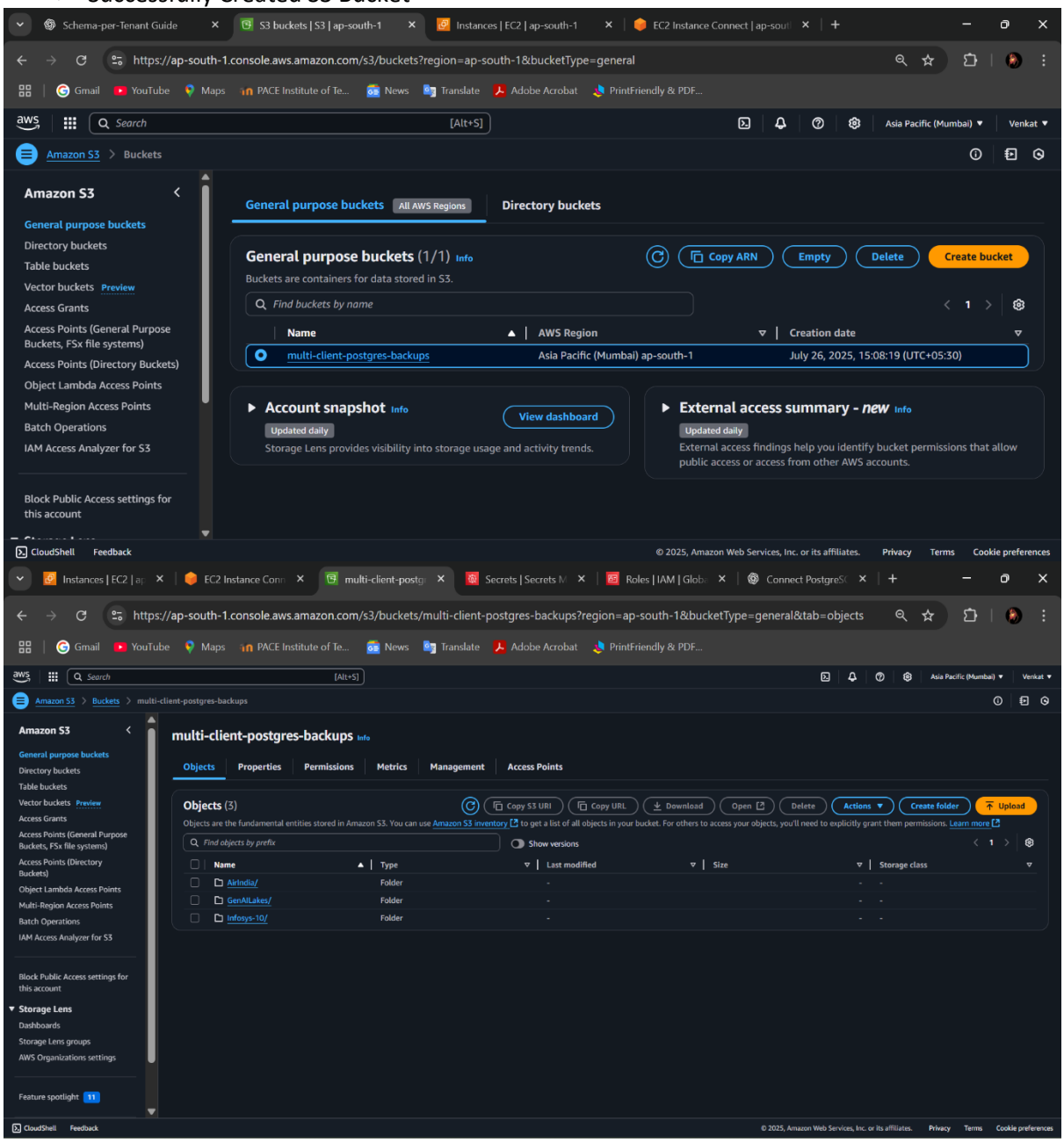
aws configure

Access Key And Secret Access Key

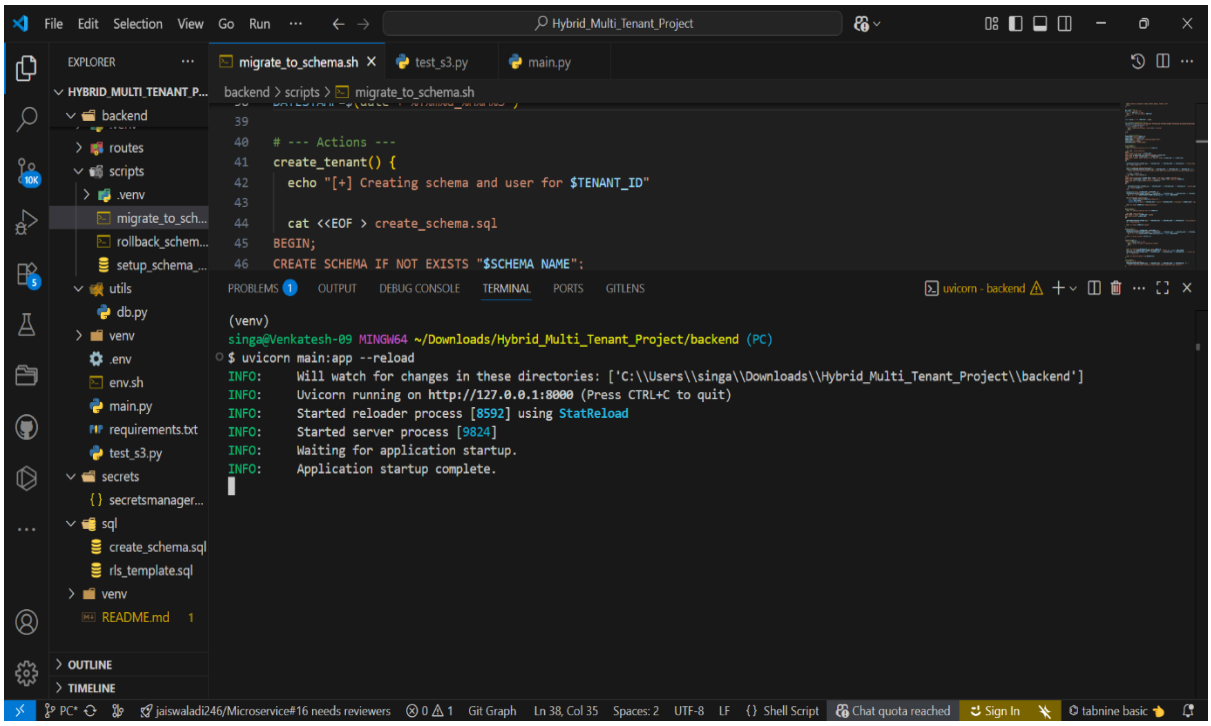
- ⇨ Create a backup script:

```
#!/bin/bash
TIMESTAMP=$(date +"%F")
BACKUP_FILE="/tmp/db_backup_$TIMESTAMP.sql"
pg_dumpall -U postgres > $BACKUP_FILE
aws s3 cp $BACKUP_FILE s3://your-s3-bucket-name/db_backups/
```

- ⇨ Successfully Created S3 Bucket



➔Backend Integration (FastAPI)



✅ Final Project Structure (`Hybrid_Multi_Tenant_Project/`)

bash Copy Edit

```
Hybrid_Multi_Tenant_Project/
├── backend/
│   ├── main.py                 # FastAPI app with tenant APIs (create, delete, backup, restor
│   ├── requirements.txt        # Python dependencies
│   ├── .env                    # Environment variables (DB, AWS, Secrets, etc.)
│   ├── tenants.py              # Tenant management logic (optional)
│   ├── db.py                   # PostgreSQL DB connection logic
│   ├── routes/
│   │   └── backup.py           # Separate router for S3 backup/restore/list
│   └── scripts/
│       ├── migrate_to_schema.sh    # Master script for tenant DB/Schema actions
│       ├── rollbackSchema.sh       # Optional rollback script
│       └── setup_schema_rls.sql    # SQL script to setup RLS policies
├── sql/
│   ├── create_schema.sql       # Base schema creation SQL
│   └── rls_template.sql        # Row-level security policy template
├── secrets/
│   └── secretsmanager_policy.json  # IAM policy for Secrets Manager access
├── docs/
│   └── architecture_diagram.png    # Final system architecture diagram (PNG)
├── README.md                   # GitHub project overview
└── .gitignore                  # Ignore Python envs, IDE, etc.
```

🧭 System Flow Summary

1. **Tenant Onboarding**:
   - FastAPI `/tenant/create` → Shell script creates schema → Migrates data → Dumps backup to S3 → Stores creds in Secrets Manager.
2. **Backup**:
   - `/tenant/backup` → pg_dump runs → file uploaded to S3.
3. **Restore**:
   - `/tenant/restore` → Downloads from S3 → restores to proper schema.
4. **List**:
   - `/tenant/list` → Scans PostgreSQL for all `tenant_%` schemas.
5. **Delete**:
   - `/tenant/delete` → Removes schema, user, optionally Secrets and backups.

To Setup the Environment (venv):

```
#!/bin/bash

ACTION=$1

if [ -z "$ACTION" ]; then
  echo "❌ Usage: $0 {activate|deactivate}"
  exit 1
fi

if [ "$ACTION" == "activate" ]; then
  echo "🔍 Checking for virtual environment..."

  # Create if missing
  if [ ! -d ".venv" ]; then
    echo "📦 Creating .venv virtual environment..."
    python3 -m venv .venv || py -3 -m venv .venv
  fi

  # Activate for Git Bash (Windows)
  if [ -f ".venv/Scripts/activate" ]; then
    echo "✅ Activating virtual environment (Windows)"
    source .venv/Scripts/activate
  # Activate for Linux/macOS
  elif [ -f ".venv/bin/activate" ]; then
    echo "✅ Activating virtual environment (Unix)"
    source .venv/bin/activate
  else
    echo "❌ Activation script not found (.venv/Scripts/activate or .venv/bin/activate)"
    exit 1
  fi
```

```
  # Install dependencies
  if [ -f "requirements.txt" ]; then
    echo "📦 Installing requirements..."
    pip install -r requirements.txt
  fi

  echo "🚀 Environment is ready."

elif [ "$ACTION" == "deactivate" ]; then
  if type deactivate &>/dev/null; then
    deactivate
    echo "🛑 Virtual environment deactivated."
  else
    echo "⚠️ No virtual environment is currently active."
  fi

else
  echo "❌ Invalid option: $ACTION"
  echo "Usage: $0 {activate|deactivate}"
  exit 1
fi
```

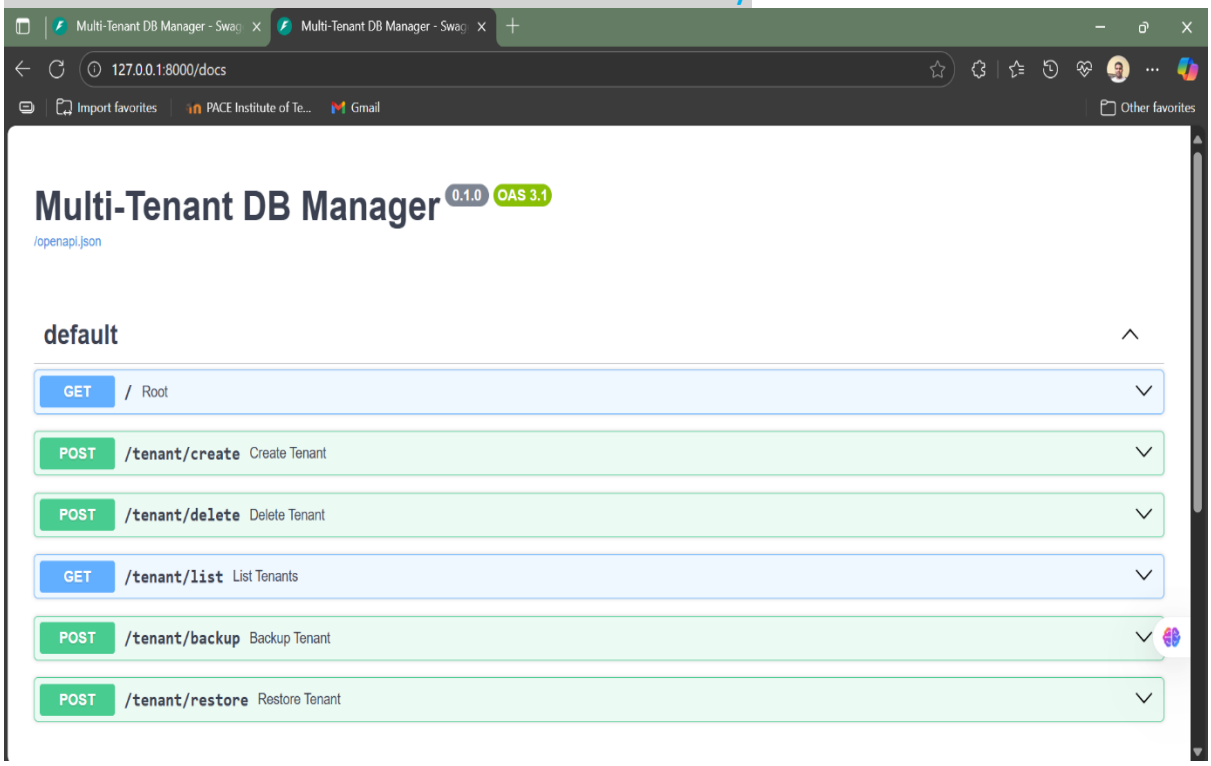
- ⇨ ./env.sh activate
  # work...
- ⇨ ./env.sh deactivate

Download Packages:

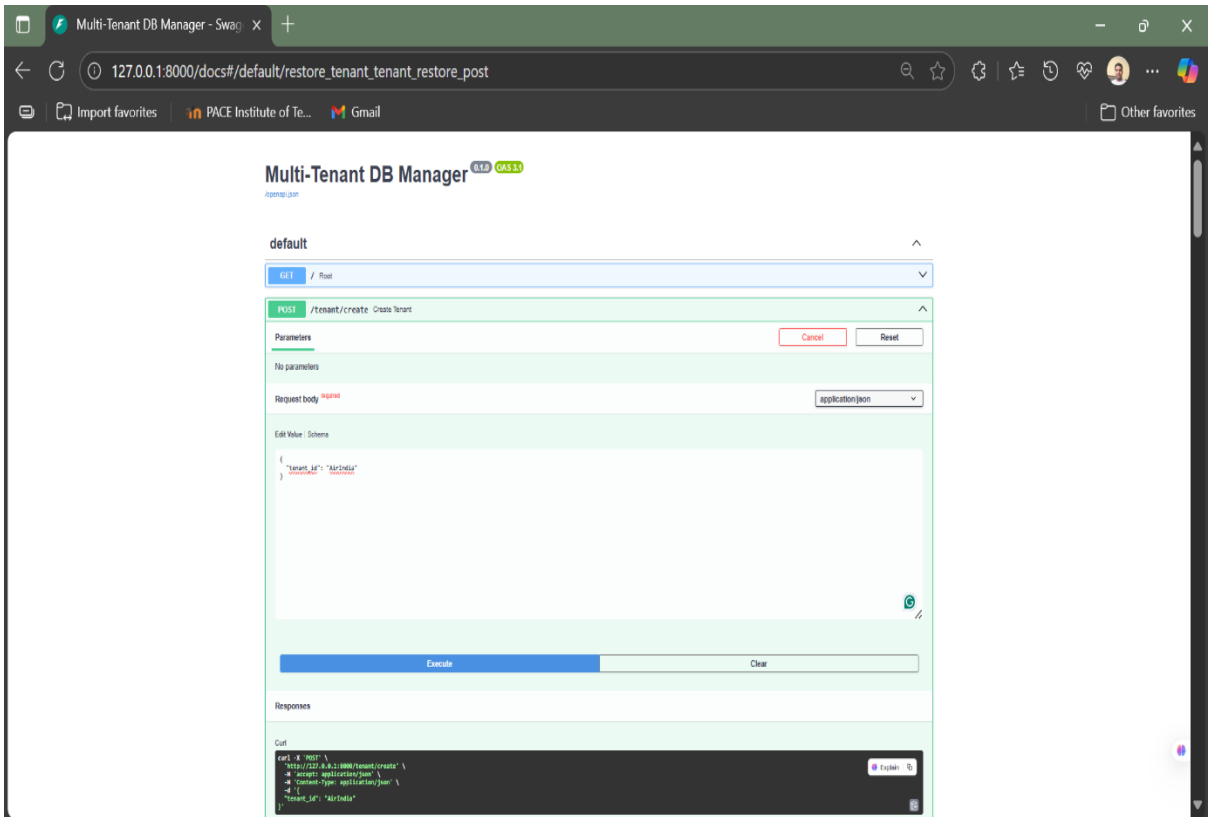
- ⇨ pip install fastapi uvicorn python-dotenv boto3 psycopg2 (/Hybrid_Multi-Tenant/backend)
- ⇨ python -m venv venv (Creates Environment)
- ⇨ source venv/Scripts/activate (bash)
- ⇨ .\venv\Scripts\Activate.ps1 (Powershell)
- ⇨ pip install -r requirements.txt
- ⇨ uvicorn main:app –reload
- ⇨ Visit: http://localhost:8000/docs

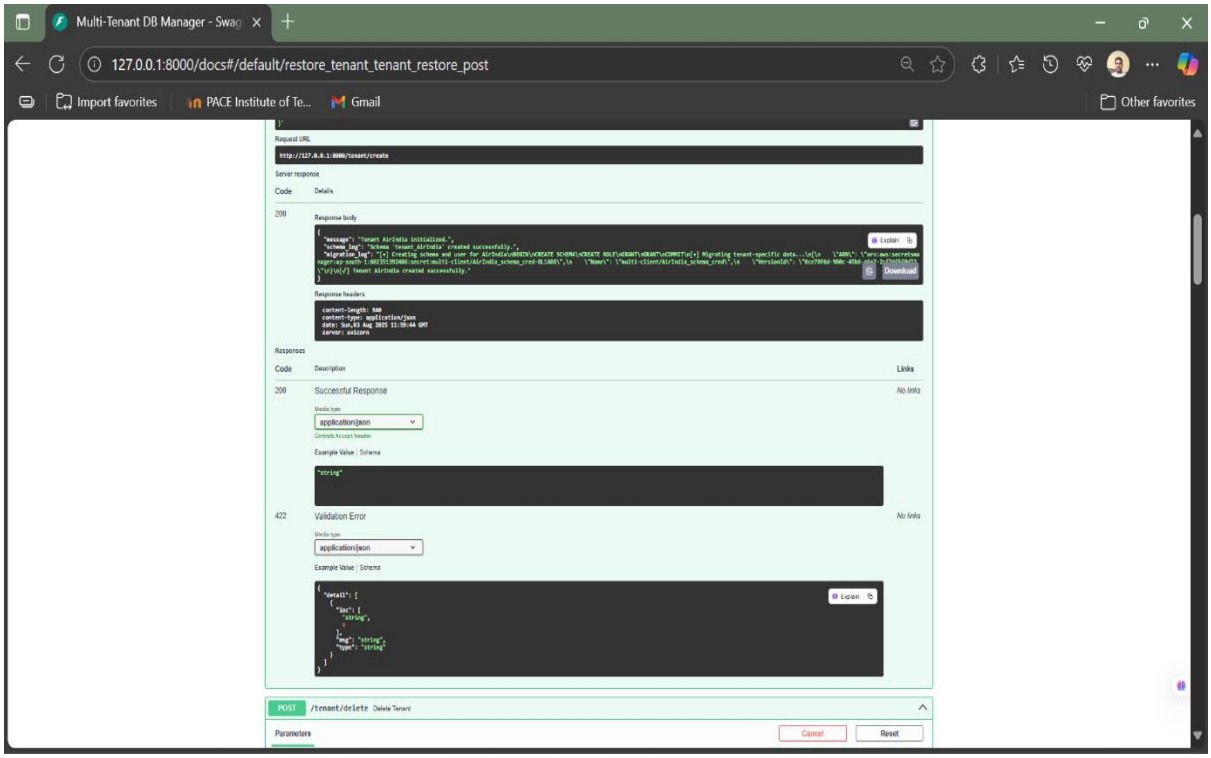
- ⇨ deactivate
- ⇨ rm -rf venv
- ⇨ python -m venv venv (Creates Environment)

Routers or Controllers Created Successfully

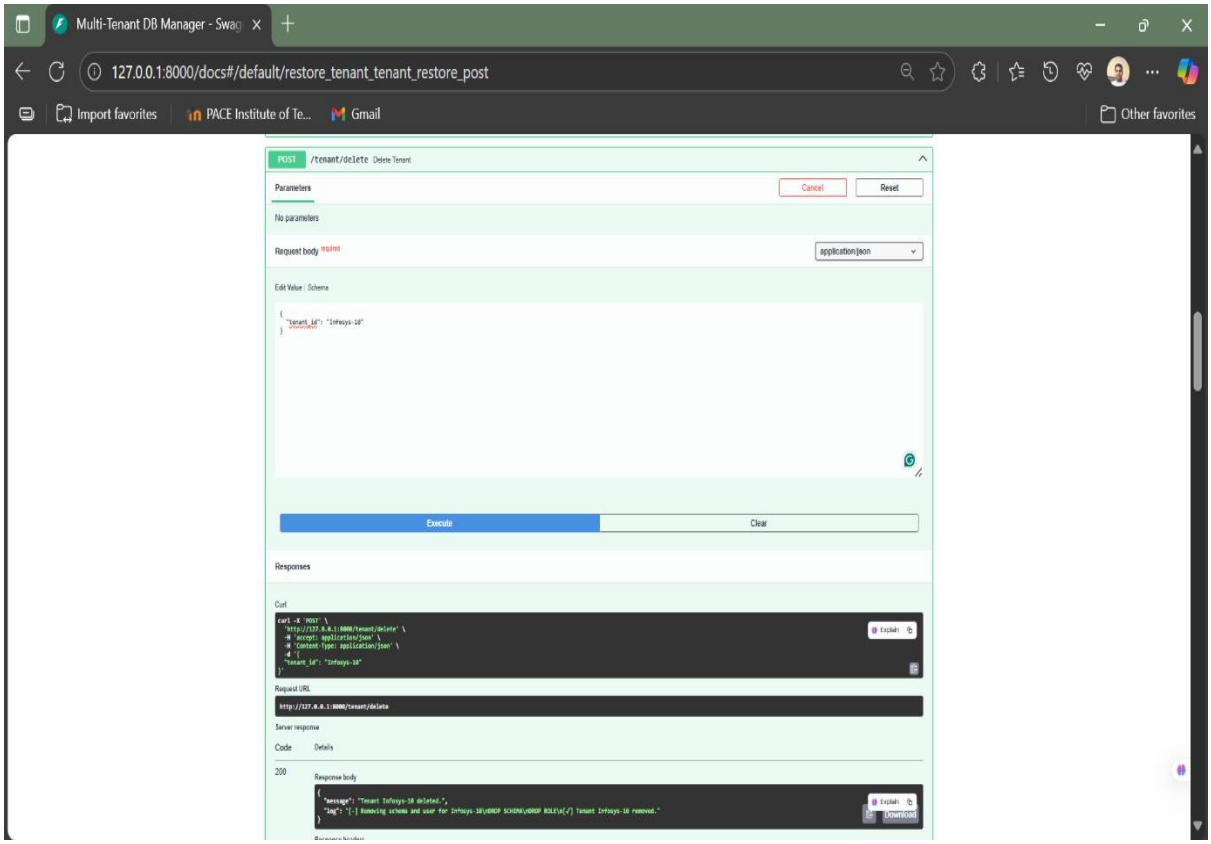


⇨ Create:

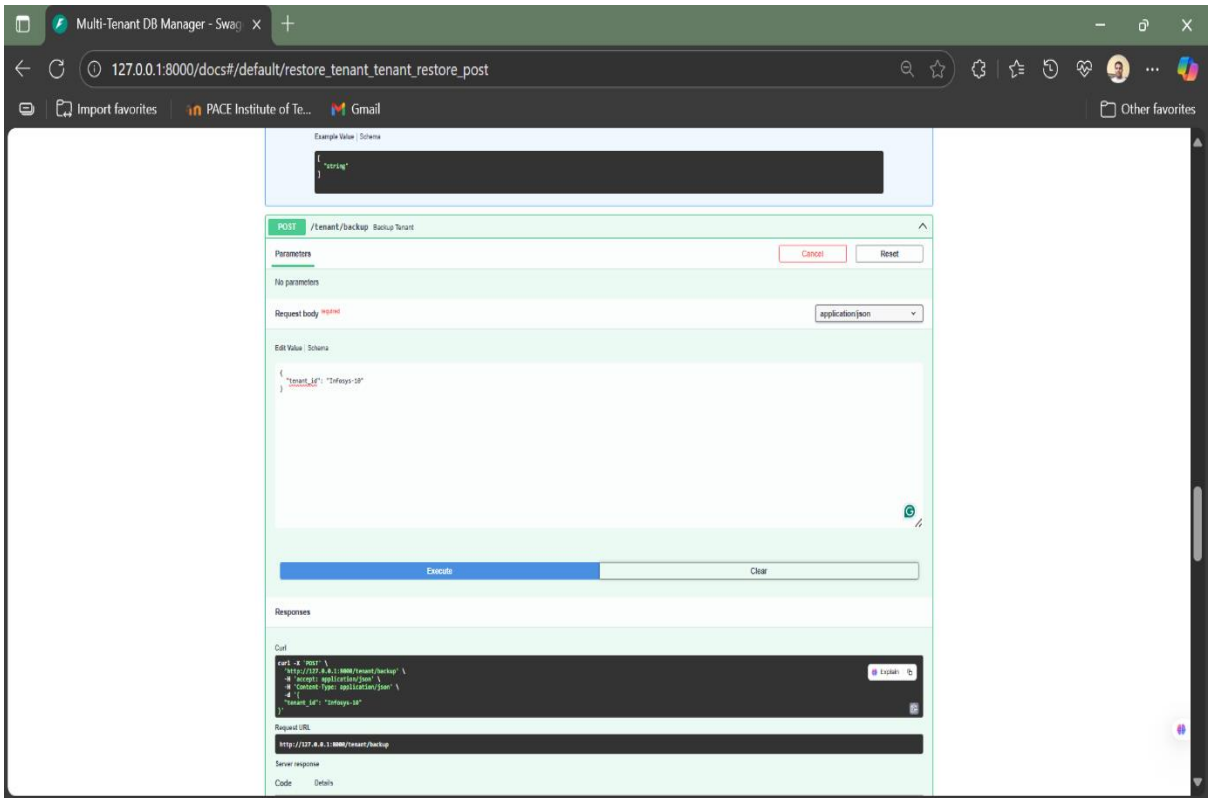



Multi-Tenant DB Manager - Swag
127.0.0.1:8000/docs#/default/restore_tenant_tenant_restore_post
Import favorites
PACE Institute of Te...
Gmail
Other favorites
Request URL
http://127.0.0.1:8000/tenant/create
Server response
Code
Details
200
Response body
Download
Response headers
content-length: 588
content-type: application/json
date: Sun,03 Aug 2025 11:39:44 GMT
server: uvicorn
Responses
Code
Description
Links
200
Successful Response
No links
application/json
Example Value | Schema
"string"
422
Validation Error
No links
application/json
Example Value | Schema
"detail": [
"loc": [
"string",
0
],
"msg": "string",
"type": "string"
]
POST
/tenant/delete
Delete Tenant
Parameters
Cancel
Reset

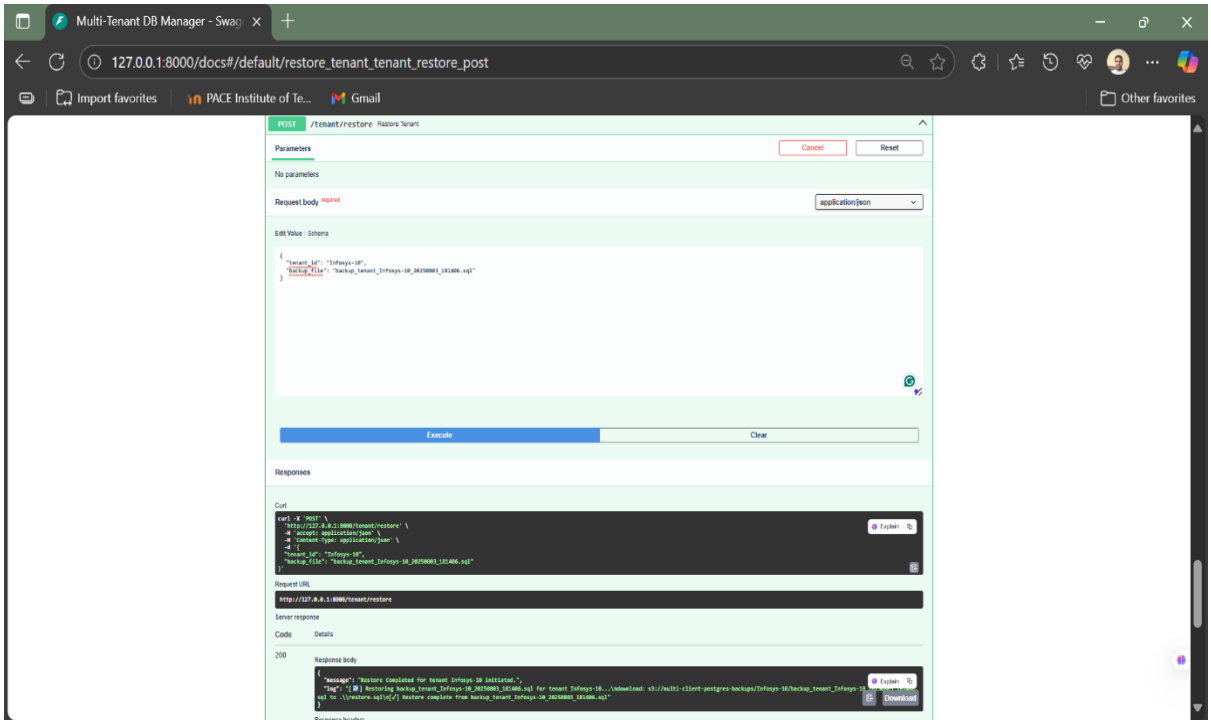
⇨ Delete:


Multi-Tenant DB Manager - Swag
127.0.0.1:8000/docs#/default/restore_tenant_tenant_restore_post
Import favorites
PACE Institute of Te...
Gmail
Other favorites
POST
/tenant/delete
Delete Tenant
Parameters
Cancel
Reset
No parameters
Request body
required
application/json
Edit Value | Schema
"tenant_id": "Infosys-10"
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://127.0.0.1:8000/tenant/delete' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
"tenant_id": "Infosys-10"
}'
Request URL
http://127.0.0.1:8000/tenant/delete
Server response
Code
Details
200
Response body
"message": "Tenant Infosys-10 deleted.",
"log": "[-] Removing schema and user for Infosys-10\nDROP SCHEMA\nDROP ROLE\n[✓] Tenant Infosys-10 removed."
Download

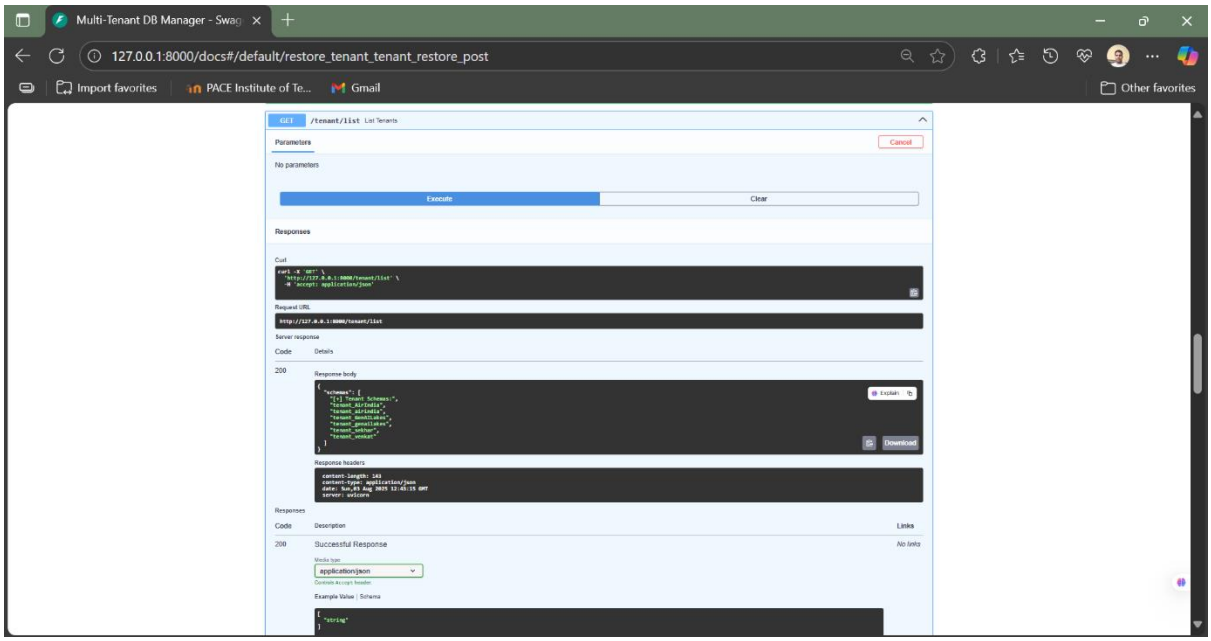
⇨ Backup:



⇨ Restore:



⇨ List:

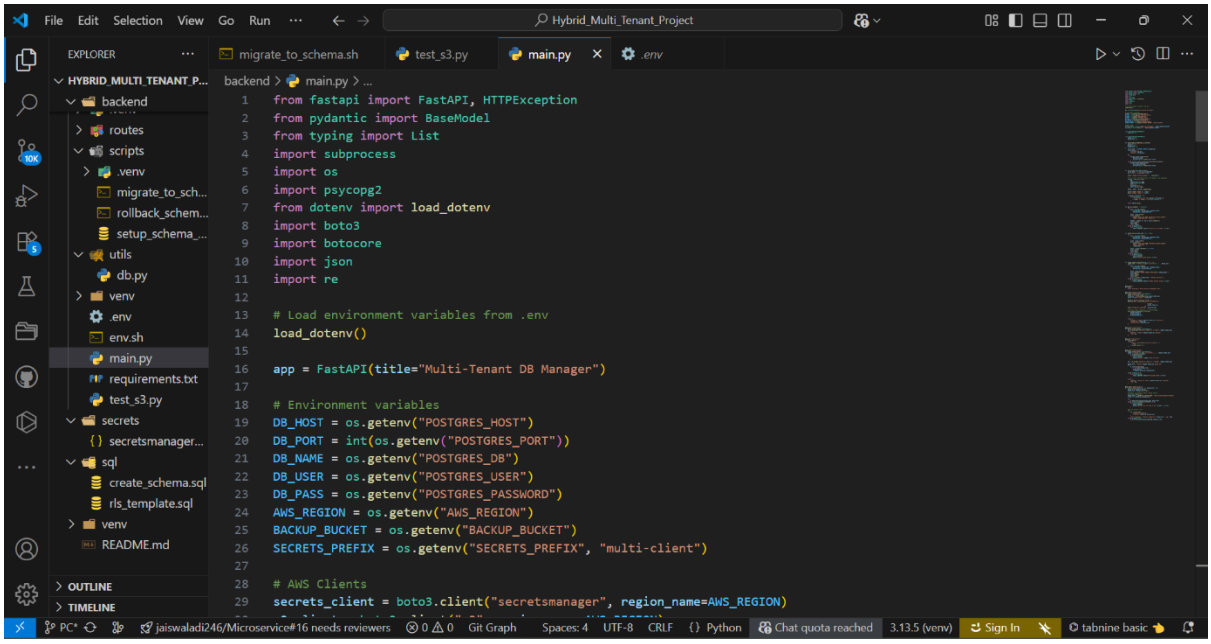


FastAPI Backend Code Structure

Routes:

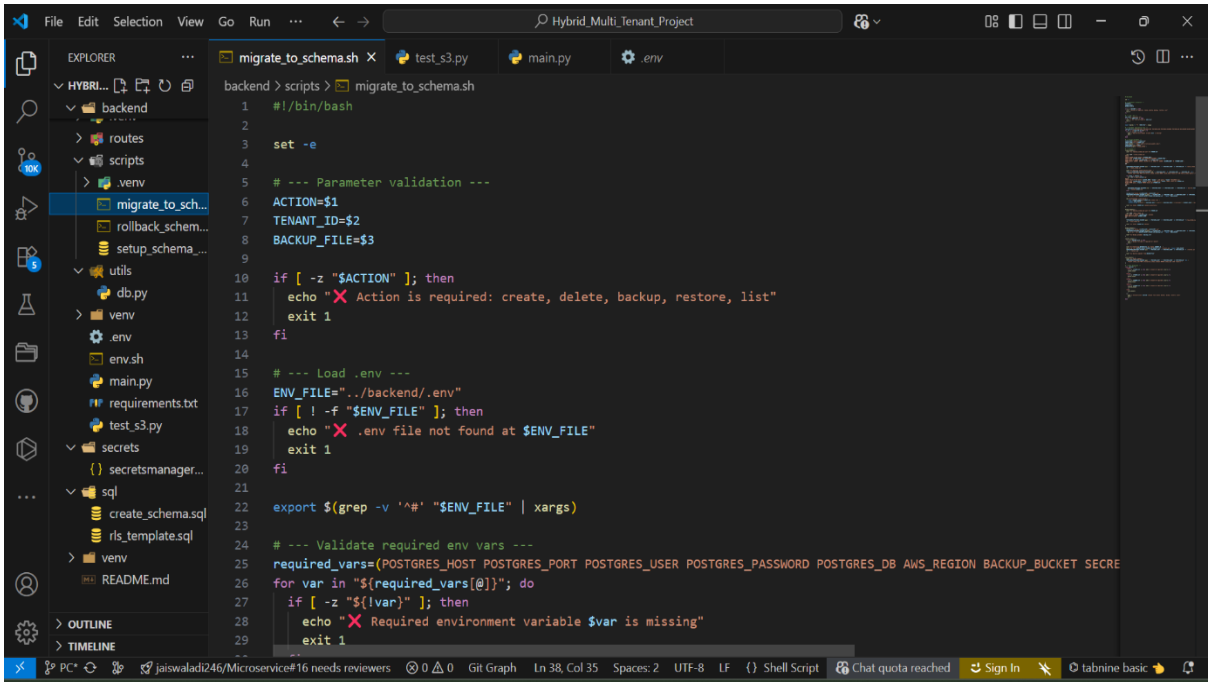
- POST /tenant/create - Creates new tenant schema or DB, Stores Secret in Secret Manager.
- POST /tenant/delete - Deletes tenant after backing up
- GET /tenant/list - Lists current tenants (schemas/DBs)
- POST /tenant/backup - Backup to S3
- POST /tenant/restore - Restore tenant from S3 backup

⇨ main.py file:

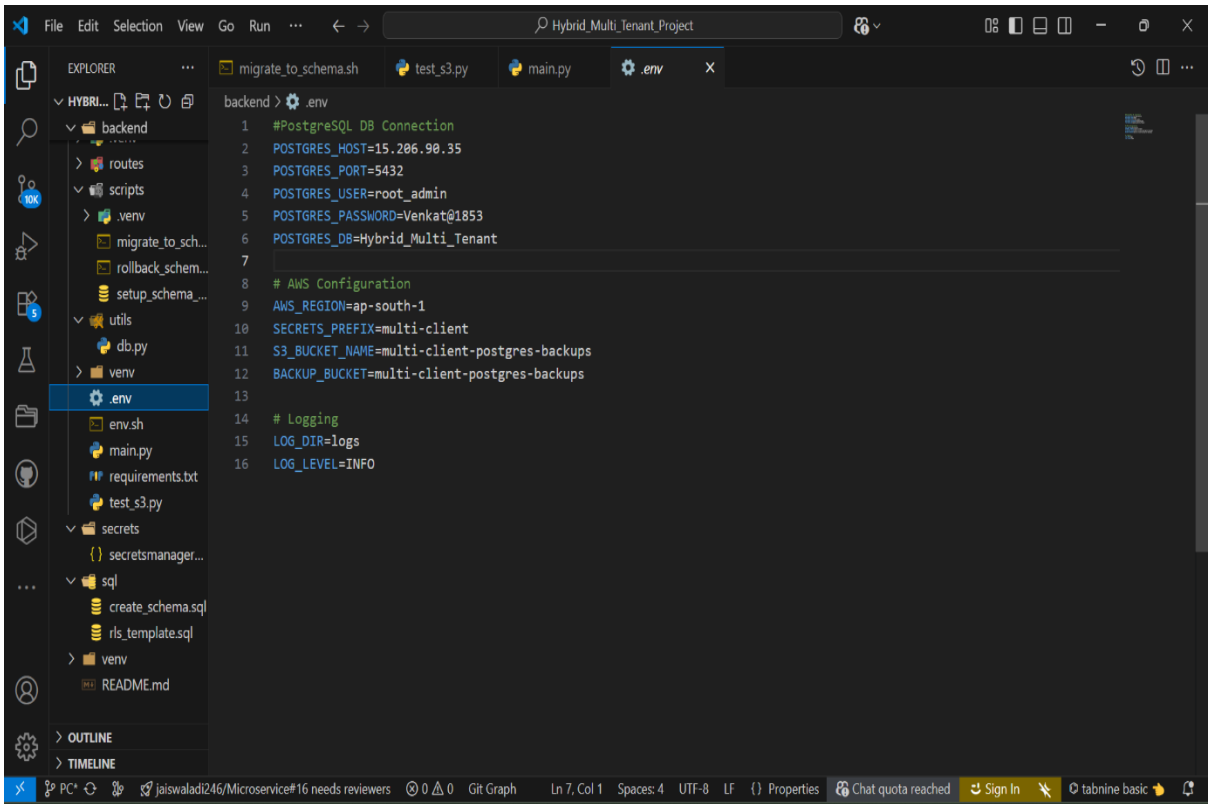


Bash Script Integration:

- ./migrate_to_schema.sh/create_tenant.sh
- ./ migrate_to_schema.sh/backup_tenant.sh
- ./ migrate_to_schema.sh/restore_tenant.sh
- ./ migrate_to_schema.sh/delete_tenant.sh
- ./ migrate_to_schema.sh/list_tenant.sh



.env file:



S3 Backup Flow:

Step-by-Step Guide

Create an S3 Bucket

aws s3 mb s3://multi-client-postgres-backups --region ap-south-1

Create a Backup Script

Create File /opt/pg_s3_backup.sh

sudo vi /opt/pg_s3_backup.sh

Paste the following:

```
#!/bin/bash

DATE=$(date +%F-%H-%M)
DB_USER="postgres"
DB_HOST="localhost"
S3_BUCKET="multi-client-postgres-backups"
BACKUP_DIR="/tmp/db_backups"
mkdir -p $BACKUP_DIR

# Get non-template databases
databases=$(psql -U $DB_USER -h $DB_HOST -t -c "SELECT datname FROM pg_database WHERE datistemplate = false;")

for db in $databases; do
  CLEAN_DB=$(echo $db | xargs)
  FILENAME="${BACKUP_DIR}/${CLEAN_DB}_${DATE}.sql.gz"
  echo "📦 Backing up database: $CLEAN_DB"
  pg_dump -U $DB_USER -h $DB_HOST $CLEAN_DB | gzip > $FILENAME

  # Upload to S3
  aws s3 cp $FILENAME s3://$S3_BUCKET/$CLEAN_DB/$DATE.sql.gz

  # Clean up
  rm -f $FILENAME
done
```

Make it executable:

sudo chmod +x /opt/pg_s3_backup.sh

Schedule via cron (Daily at 1AM)

sudo crontab -e

Add this line:
0 1 * * * /opt/pg_s3_backup.sh >> /var/log/pg_backup.log 2>&1

Result

Every night at **1 AM**, all PostgreSQL databases will be:

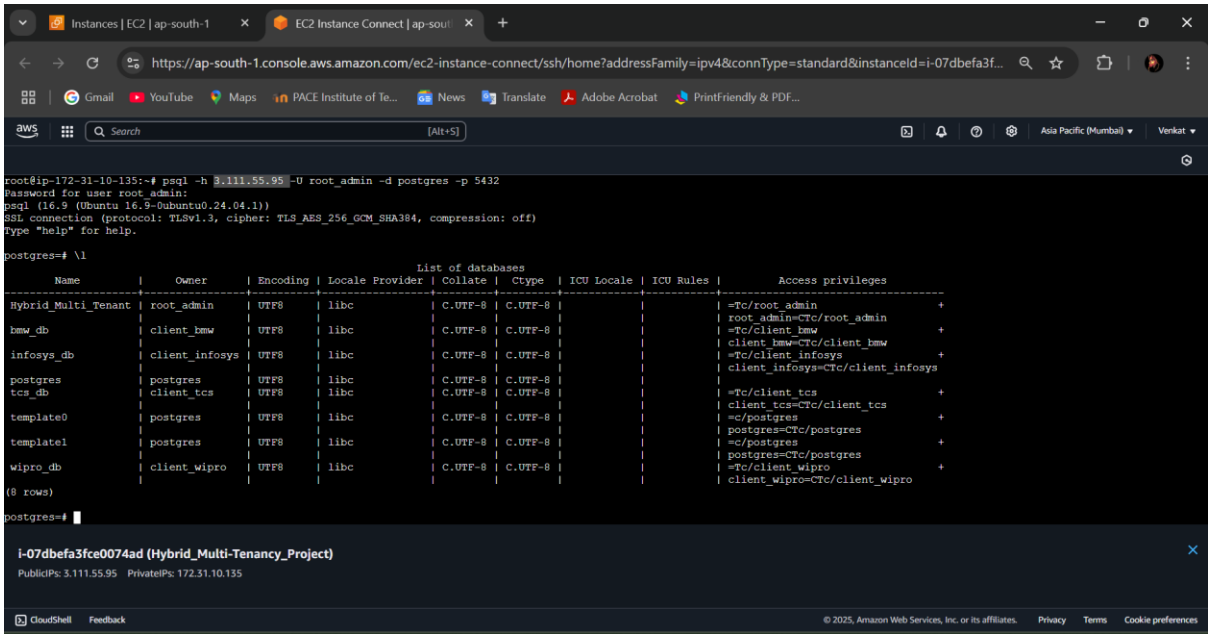
- Backed up as compressed .sql.gz files
- Uploaded to s3://multi-client-postgres-backups/<db_name>/<timestamp>.sql.gz

---

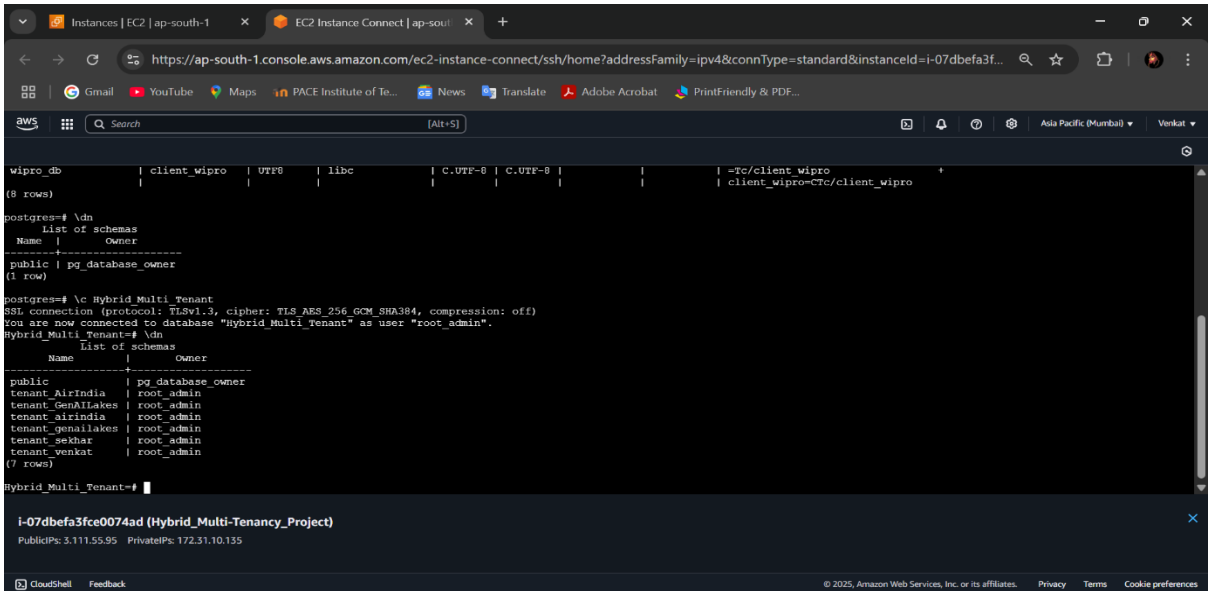
1. Use pg_dump to dump schema/db
2. Upload to s3://tenant-db-backups/{tenant_id}/backup_YYYYMMDD.sql
3. On restore, download and psql -f

Secret Manager JSON Format:

```
{
 "host": "<db-host>",
 "port": 5432,
 "username": "user_tenant1",
 "password": "<secure-password>",
 "dbname": "maindb" or "tenant1db"
}
```



⇨ Created Database Per Tenant In Single VM (PostgreSQL) Database



⇨ In the Tenant Schema per tenant created sucessfully

➔step by step to Test:

1. **Check schemas and databases**
2. **Insert dummy data (per tenant) into tenant-specific schema**

Step 1: Check Databases and Schemas in PostgreSQL

⇨ **Go to Connect EC2 Virtual Machine**

⇨ **Login to PostgreSQL**

```
psql -h <your-vm-ip> -U root_admin -d postgres -p 5432
```

⇨ Ex: psql -h 15.206.90.35 -U root_admin -d postgres -p 5432

⇨ Password for user postgres:

⇨ **If You Need to Create Client DBs:**

Here's a sample for **multi-tenant setup**:

⇨ sudo -u postgres psql

⇨ -- Create user and DB

```
CREATE USER client_tcs WITH PASSWORD 'Tcs@123';
CREATE DATABASE tcs_db OWNER client_tcs;
GRANT ALL PRIVILEGES ON DATABASE tcs_db TO client_tcs;

-- Create another
CREATE USER client_wipro WITH PASSWORD 'Wipro@123';
CREATE DATABASE wipro_db OWNER client_wipro;
GRANT ALL PRIVILEGES ON DATABASE wipro_db TO client_wipro;
```

⇨ **List all databases**

\l

Output: You'll see: infosys_db, tcs_db, wipro_db, etc.

⇨ **Connect to a tenant database**
For example:
\c infosys_db

⇨ **List all schemas in this database**
\dn
You should see something like:
public
tenant_Infosys_10
...

Summary

Task	**Command/Action**
List databases	\l
Connect to a DB	\c <db_name>
List schemas	\dn
Use a schema	SET search_path TO tenant_<name>
Create & insert	SQL CREATE TABLE, INSERT INTO
Verify inserted data	SELECT * FROM <table>

Insert Dummy Data into a Tenant's Schema

⇨ **1. Connect to the database:**
\c infosys_db

⇨ **2. Switch to tenant schema**
Set the schema path:
SET search_path TO tenant_Infosys_10;

⇨ **3. Create a dummy table (if not yet created)**

```
CREATE TABLE candidates (
    id SERIAL PRIMARY KEY,
    candidate_name VARCHAR(100),
    job_role VARCHAR(100),
    phone_number VARCHAR(15),
     email VARCHAR(100),
    place VARCHAR(100)
    );
```

⇨ **4. Insert dummy records**

```
INSERT INTO candidates (candidate_name, job_role, phone_number, email, place) VALUES
 ('Rahul Sharma', 'Software Engineer', '9876543210', 'rahul@example.com', 'Mumbai'),
('Priya Verma', 'Data Analyst', '9123456780', 'priya@example.com', 'Bangalore');
```

⇨ **5. Check inserted data**
SELECT * FROM candidates;
You should see:

	id	candidate_name	job_role	phone_number	email	place
1		Rahul Sharma	Software Engineer	9876543210	rahul@example.com	Mumbai
2		Priya Verma	Data Analyst	9123456780	priya@example.com	Bangalore

---

⇨ **Check Remaining (for multiple tenants)**

Repeat the above:

- Connect to other DBs (\c wipro_db, \c tcs_db)
- Set schema path to tenant_Wipro_xx or tenant_TCS_xx
- Run same create/insert queries
