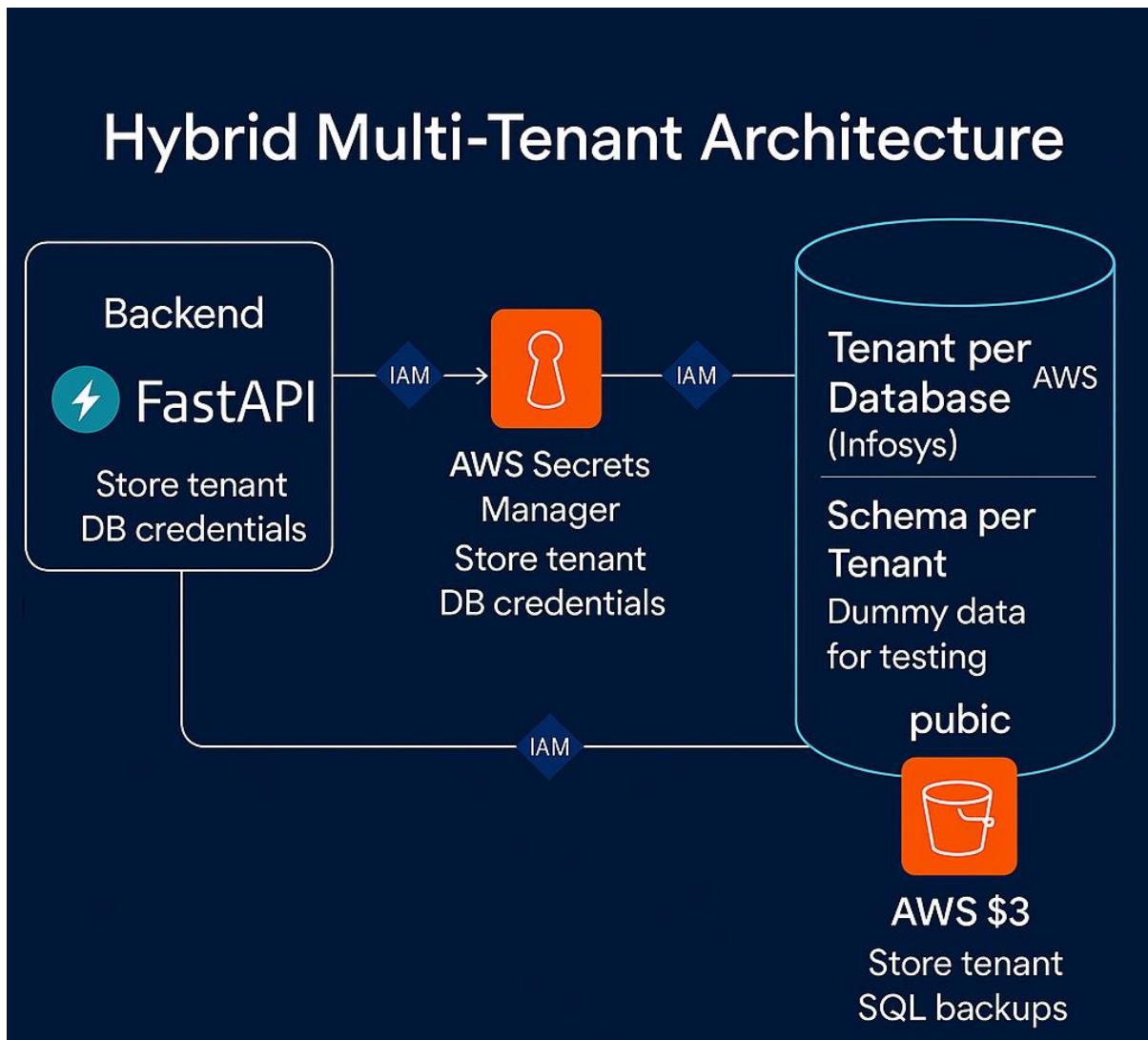


Hybrid Multi-Tenant Architecture (Database-Per-Tenant + Schema per Tenant)



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

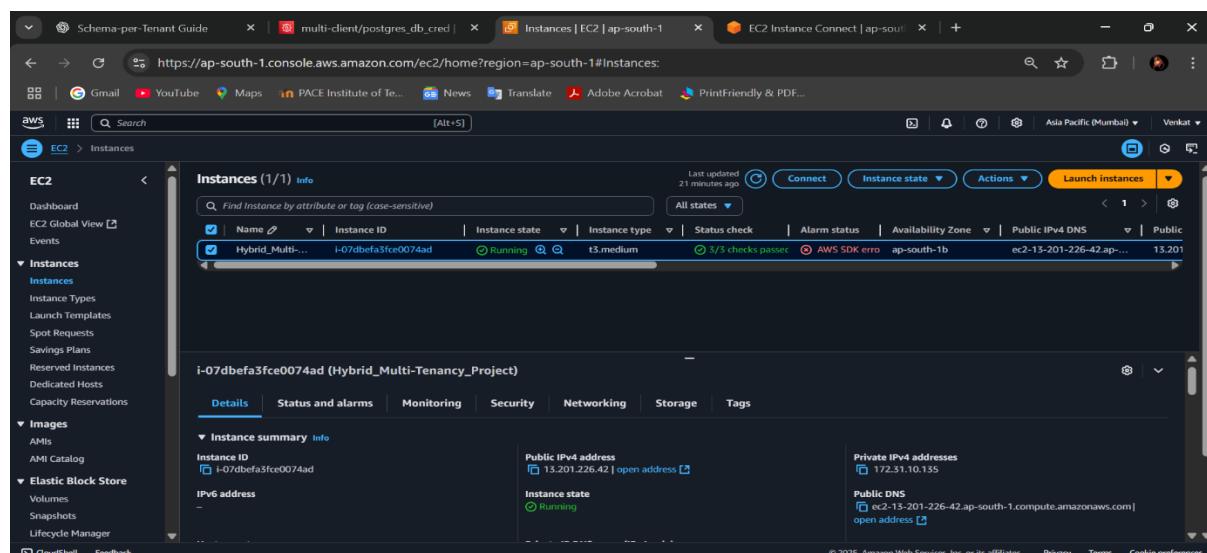
Prerequisites:

- 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)



The screenshot shows the AWS EC2 Security Groups console. On the left, there's a navigation sidebar with options like Dashboard, EC2 Global View, Events, Instances, Images, Elastic Block Store, Network & Security, and CloudShell. The main area displays the details for a security group named 'sg-0ac60fd563cc6de28 - launch-wizard-5'. It shows the security group ID, owner (602351392486), and a description indicating it was created on 2025-07-26T04:03:14.638Z. The VPC ID is vpc-08c0d5e8523902c10. Below this, there are tabs for Inbound rules, Outbound rules, Sharing - new, VPC associations - new, and Tags. The Inbound rules section lists six entries:

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
-	sgr-05d7449f6418f96e	IPv4	HTTP	TCP	80	0.0.0.0/0	-
-	sgr-08fc425af70c50a9	IPv4	PostgreSQL	TCP	5432	0.0.0.0/0	-
-	sgr-066fc0a046bf0f00	IPv4	Custom TCP	TCP	8000	0.0.0.0/0	-
-	sgr-01be9a23e521bbfd	IPv4	SSH	TCP	22	0.0.0.0/0	-
-	sgr-044511666815f323	IPv4	HTTPS	TCP	443	0.0.0.0/0	-
-	sgr-0e79f69fdaca035f4	IPv4	Custom TCP	TCP	9000	0.0.0.0/0	-

→ 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-xxxxxx"  
}
```

The screenshot shows the AWS IAM Users page. The URL in the browser is `https://us-east-1.console.aws.amazon.com/iam/home?region=ap-south-1#users`. The left sidebar shows the IAM navigation menu with 'Users' selected. The main content area displays a table titled 'Users (1/1) Info'. The table has one row for the user 'multi-tenant-db-user'. The columns include 'User name' (multi-tenant-db-user), 'Path' (/), 'Group' (0), 'Last activity' (23 minutes ago), 'MFA' (-), 'Password age' (2 hours), 'Console last sign-in' (-), and 'Access key ID' (Active - AKIAIYYPXQZ...). There are buttons for 'Delete' and 'Create user' at the top right of the table.

The screenshot shows the AWS IAM Roles page. The left sidebar has sections for Identity and Access Management (IAM), Access management, Access reports, and CloudWatch Metrics. The main area displays a table of roles:

Role name	Trusted entities	Last activity
AWSServiceRoleForSupport	AWS Service: support (Service-Linked)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service)	-
multi-tenant-db	AWS Service: ec2	-
multi-tenant-db-role	AWS Service: ec2	22 minutes ago

Below the table, there are sections for 'Access AWS from your non AWS workloads' (using X.509 Standard) and 'Temporary credentials'.

⇒ 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
```

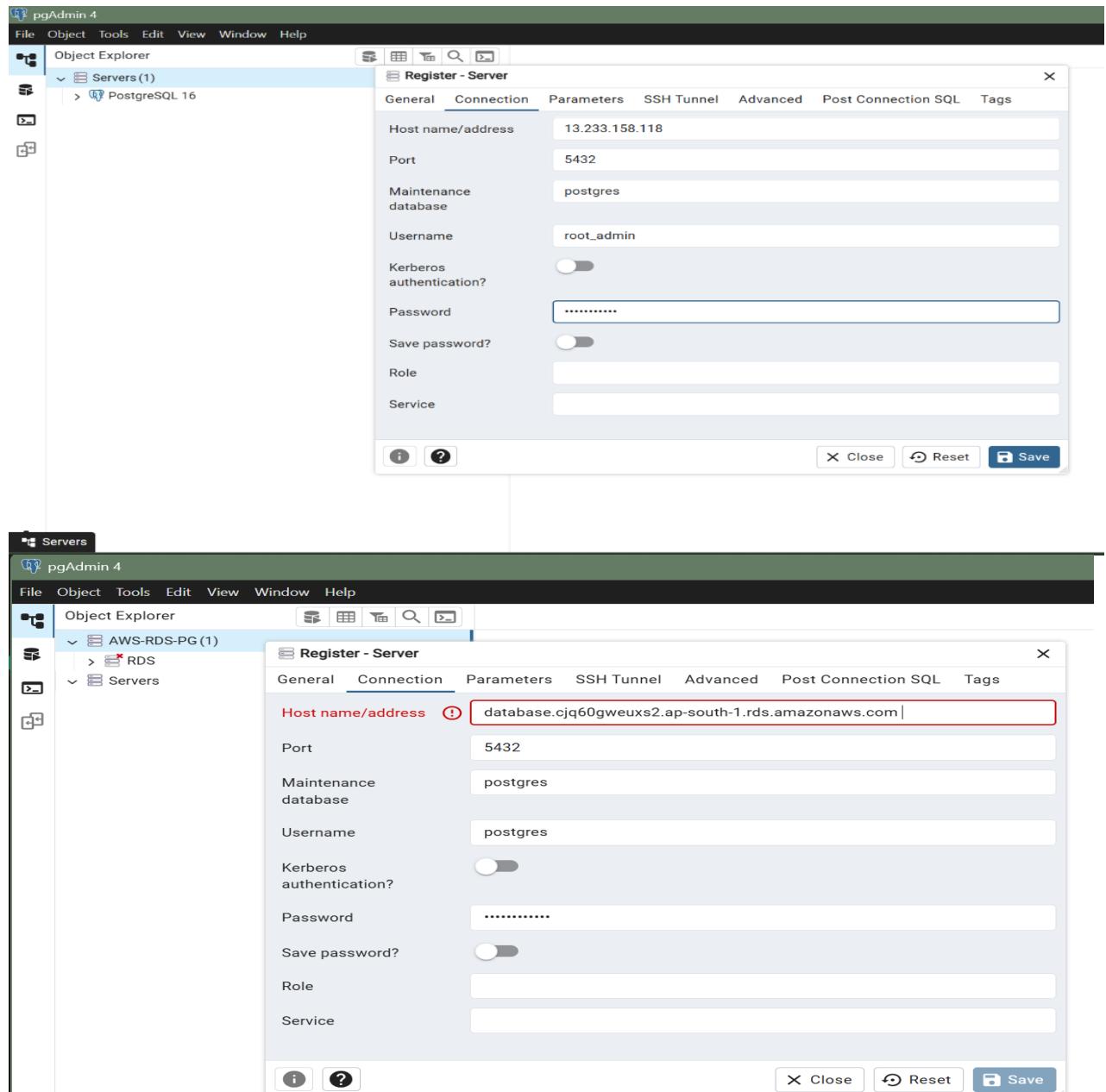
You can replace 0.0.0.0/0 with a specific IP or CIDR for **tighter security** (e.g., 203.0.113.15/32)

⇒ 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.

	id [PK] integer	name text	job_role text	phone_number text	email text	place text
1	1	Amit Sharma	Software Developer	9876543210	amit.sharma@example.com	Bangalore
2	2	Riya Kapoor	Data Analyst	9871234567	riya.kapoor@example.com	Hyderabad
3	3	Vikram Mehta	DevOps Engineer	9123456789	vikram.mehta@example.com	Pune

```

SELECT * FROM tenant_tcs.candidate

```

	id [PK] integer	name text	job_role text
1	1	Alice Johnson	Data Analyst
2	2	Bob Smith	Software Engineer
3	3	Carol Lee	HR Manager

- ⇒ 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:

Configure secret

Secret name and description

Secret name

/multi-client/postgres_db_cred

Description - optional

Access my Postgres Database

Tags - optional

Resource permissions - optional

Replicate secret - optional

The screenshot shows three sequential steps of the AWS Secrets Manager "Store a new secret" wizard:

Step 1: Choose secret type

Selected: Other type of secret (API key, OAuth token, etc.)

Step 2: Configure secret

Step 3: optional

Step 4: Configure rotation

Configure rotation - optional

Configure automatic rotation (Info) Configure AWS Secrets Manager to rotate this secret automatically.

Automatic rotation (Info)

Rotation schedule (Info)

Schedule expression builder (Time unit: Hours, Value: 24) or schedule expression (Time unit: Days, Value: 40). 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 (Info) Create function

Key/value pairs (Info)

Key/value Plaintext

username	postgres_db	Remove
password	Verkut@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/kms/secretsmanager (selected) Add new key

Next

Step 4: Review

Secret name: multi-client-postgres

Description: My PostgreSQL Secret

Last retrieved (UTC): 30 July 2025

Store a new secret

Secrets

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/GenAllLakes	-	-
multi-client/GenAllLakes.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

→ 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

The screenshot shows two screenshots of the AWS S3 console. The top screenshot displays the 'Buckets' page under the 'General purpose buckets' tab. It shows a single bucket named 'multi-client-postgres-backups' created on July 26, 2025. The bottom screenshot shows the 'Objects' page for the same bucket, listing three folder objects: 'AirIndia/' (Folder), 'GenAllLakes/' (Folder), and 'Infosys-10/' (Folder).

→ Backend Integration (FastAPI)

The screenshot shows a code editor interface with the title "Hybrid_Multi_Tenant_Project". The left sidebar displays a file tree for the project. The "backend" directory contains "routes", "scripts", "venv", and "utils" sub-directories. The "scripts" directory includes "migrate_to_schema.sh", "rollback_schema.sh", and "setup_schema_rls.sql". The "utils" directory contains "db.py", ".env", "env.sh", "main.py", "requirements.txt", "test_s3.py", and "secretsmanager_policy.json". The "sql" directory has "create_schema.sql" and "rls_template.sql". The "secrets" directory contains "secretsmanager_policy.json". The "docs" directory has "architecture_diagram.png". The "README.md" and ".gitignore" files are also listed. The right side of the screen shows the terminal output of running "uvicorn main:app --reload". The terminal shows the application starting up successfully on port 8000.

Final Project Structure (Hybrid_Multi_Tenant_Project/)

bash

Hybrid_Multi_Tenant_Project/

```
├── backend/
│   ├── main.py          # FastAPI app with tenant APIs (create, delete, backup, restore)
│   ├── 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 unicorn 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
⇒ unicorn main:app –reload
⇒ Visit: http://localhost:8000/docs

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

```

Routers or Controllers Created Successfully

The screenshot shows the Multi-Tenant DB Manager Swagger UI interface. At the top, it displays "Multi-Tenant DB Manager - Swagger" and the URL "127.0.0.1:8000/docs". Below the header, there are links for "Import favorites", "PACE Institute of Te...", and "Gmail". The main content area is titled "Multi-Tenant DB Manager 0.1.0 OAS 3.1". It lists several API endpoints under the "default" category:

- GET / Root**
- POST /tenant/create** Create Tenant
- POST /tenant/delete** Delete Tenant
- GET /tenant/list** List Tenants
- POST /tenant/backup** Backup Tenant
- POST /tenant/restore** Restore Tenant

A large blue button labeled "⇒ Create:" is positioned at the bottom left of the main content area.

The screenshot shows the Multi-Tenant DB Manager Swagger UI interface, specifically focusing on the "POST /tenant/create" endpoint under the "default" category. The URL in the browser bar is "127.0.0.1:8000/docs#/default/restore_tenant_tenant_restore_post".

The endpoint details are as follows:

- Method:** POST
- Path:** /tenant/create
- Description:** Create Tenant
- Parameters:** No parameters
- Request body (valid):** application/json
- Request body (Schema):**

```
{  
    "tenant_id": "Aladdin"  
}
```
- Responses:**
 - Curl:**

```
curl -X POST  
http://127.0.0.1:8000/tenant/create  
-H 'Accept: application/json'  
-H 'Content-Type: application/json'  
-d '{  
    "tenant_id": "Aladdin"  
}'
```

Multi-Tenant DB Manager - Swagger

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:

```
{ "message": "Tenant Altridria initialized.", "log": "[2023-08-30 11:19:44] [INFO] [ALTERIDRIA] Tenant initialized successfully.", "migration_log": "[2023-08-30 11:19:44] [INFO] [ALTERIDRIA] Creating schema and user for Altridria tenant.", "details": "[2023-08-30 11:19:44] [INFO] [ALTERIDRIA] Migrating tenant-specific data... [ALTERIDRIA] Migrating tenant schema and user for Altridria tenant.", "version": "1.0.0", "date": "Sun, 30 Aug 2023 11:19:44 GMT", "server": "nginx/1.25.2" }
```

Response headers:

```
Content-Type: application/json; charset=UTF-8
Content-Length: 500
Date: Sun, 30 Aug 2023 11:19:44 GMT
Server: nginx/1.25.2
```

Responses:

Code Description

200 Successful Response

Media type: application/json

Example Value | Schema: "string"

422 Validation Error

Media type: application/json

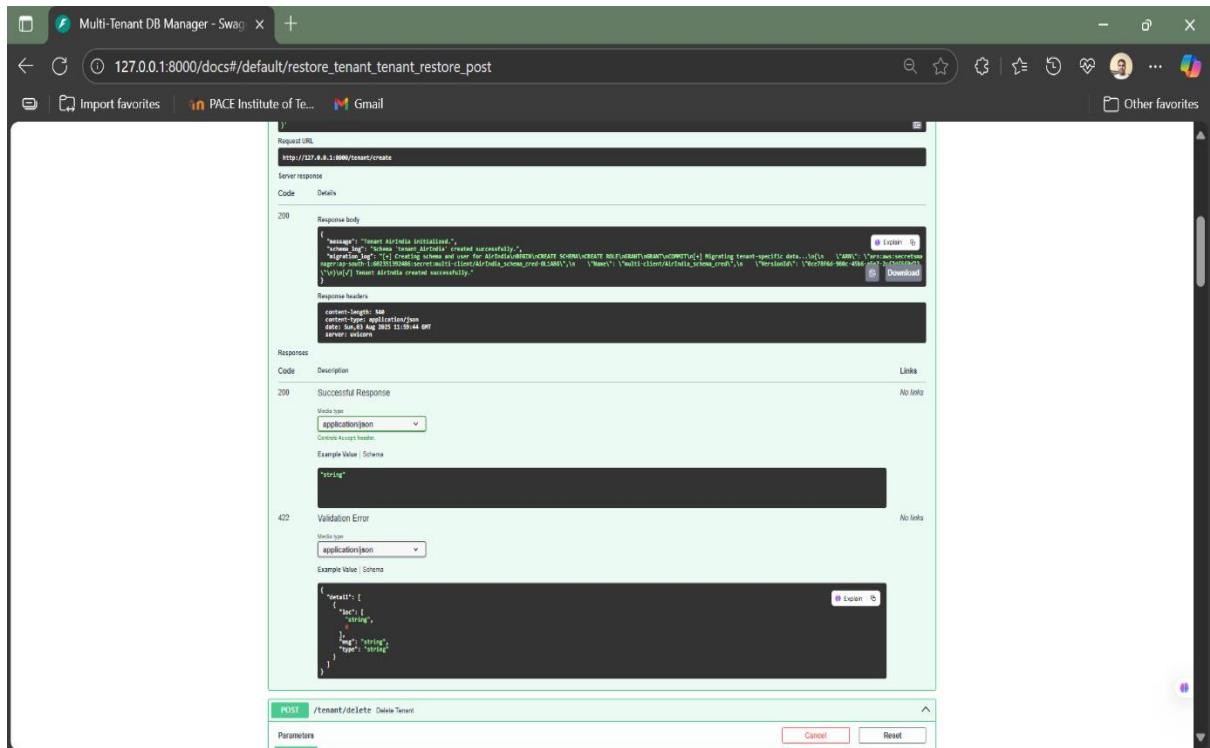
Example Value | Schema:

```
{"details": { "secret": { "type": "string", "log": "string", "type": "string" } } }
```

POST /tenant/delete Delete Tenant

Parameters

Cancel Reset



⇒ Delete:

Multi-Tenant DB Manager - Swagger

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

No parameters

Request body (json)

application/json

Edit Value | Schema:

```
{"tenant_id": "Tenways-1B"}
```

Execute Clear

Responses

Curl:

```
curl -X POST \
  http://127.0.0.1:8000/tenant/delete \
  -H 'Content-Type: application/json' \
  -H 'Accept: application/json' \
  -d '{"tenant_id": "Tenways-1B"}'
```

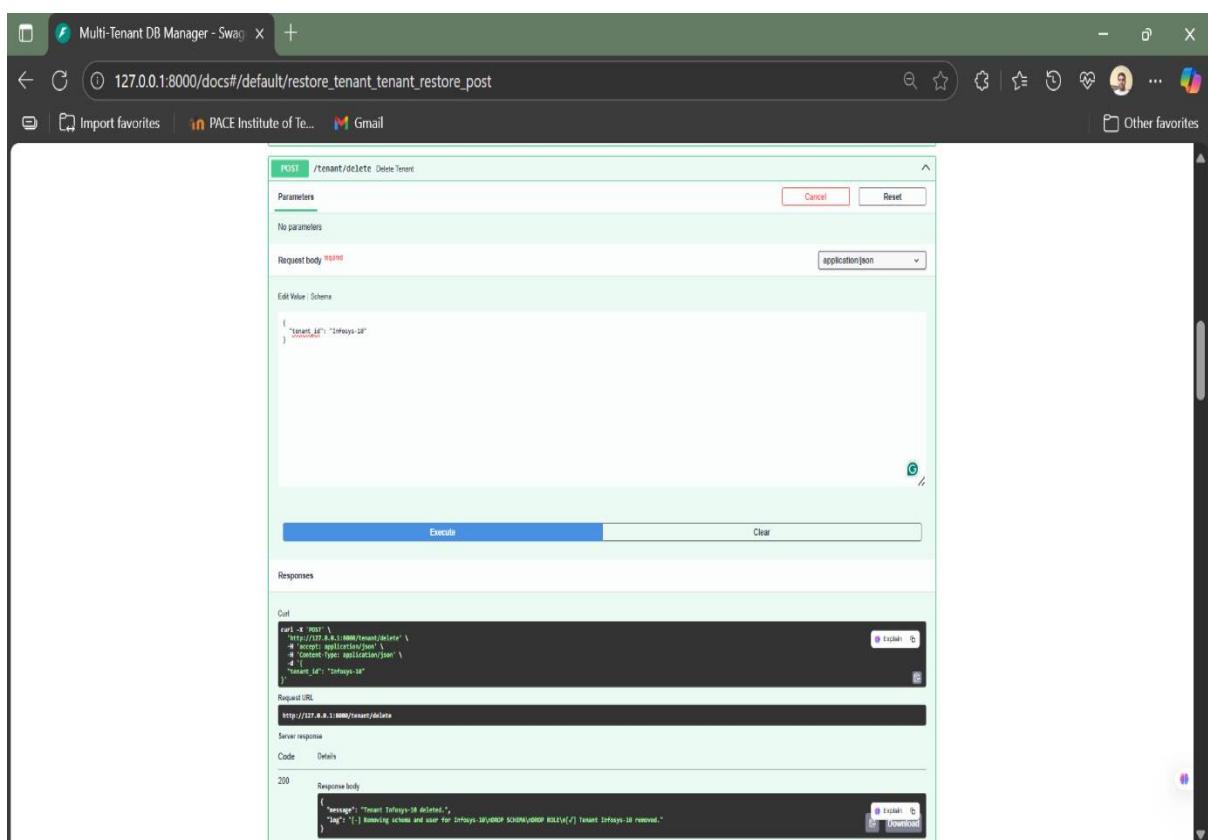
Request URL: http://127.0.0.1:8000/tenant/delete

Server response:

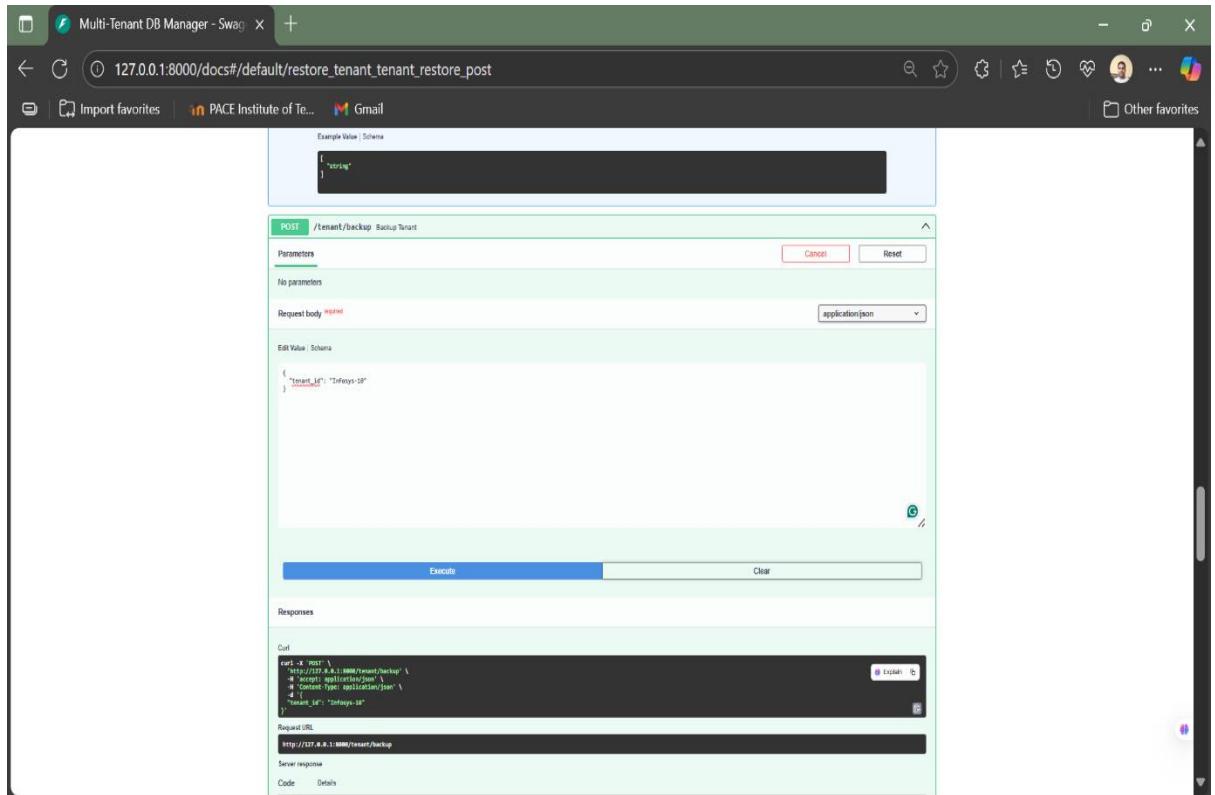
Code: Details

200 Response body:

```
{ "message": "Tenant Tenways-1B deleted.", "log": "[2023-08-30 11:19:44] [INFO] [TENWAYS-1B] Tenant Tenways-1B removed." }
```



⇒ Backup:

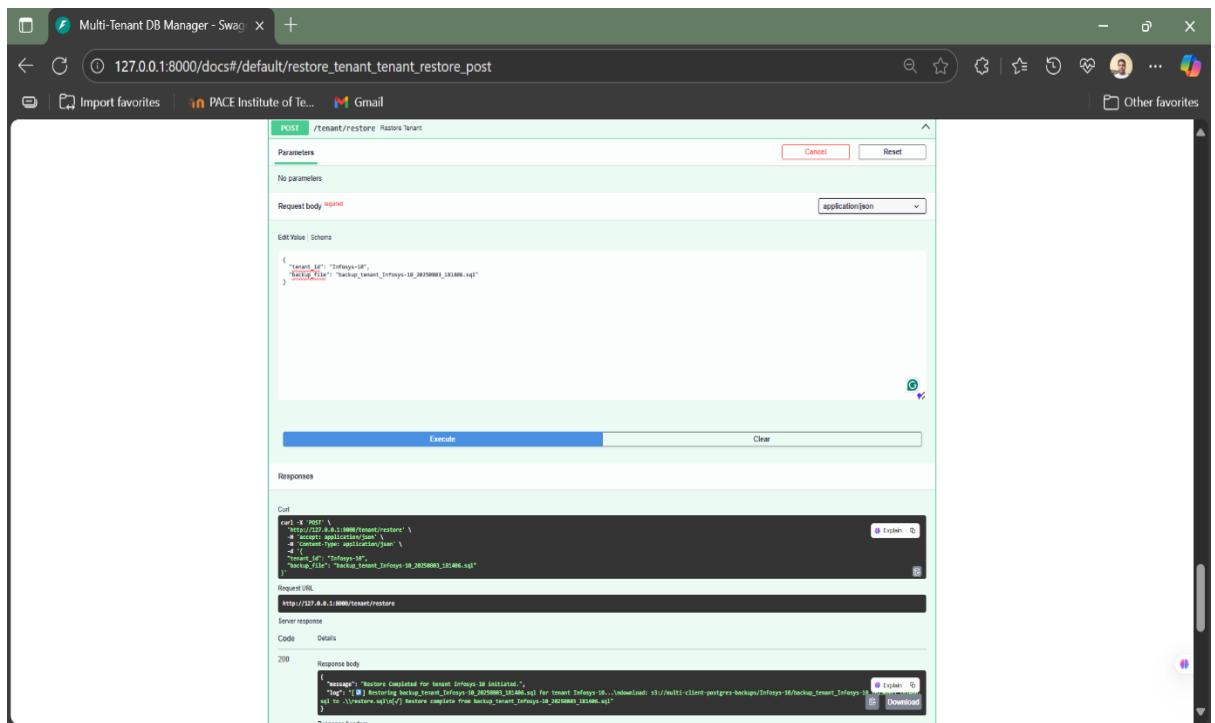


The screenshot shows the Multi-Tenant DB Manager's Swagger UI interface. The URL in the address bar is `127.0.0.1:8000/docs#/default/restore_tenant_tenant_restore_post`. The main panel displays the `POST /tenant/backup` endpoint for "Backup Tenant". The "Request body" section is highlighted in red and contains the following JSON:

```
{"tenant_id": "InfraSys-1B"}
```

The "Content-Type" dropdown is set to `application/json`.

⇒ Restore:



The screenshot shows the Multi-Tenant DB Manager's Swagger UI interface. The URL in the address bar is `127.0.0.1:8000/docs#/default/restore_tenant_tenant_restore_post`. The main panel displays the `POST /tenant/restore` endpoint for "Restore Tenant". The "Request body" section is highlighted in red and contains the following JSON:

```
{"tenant_id": "InfraSys-1B", "backup_file": "Backup_Tenant_InfraSys-1B_20230801_181006.sql"}
```

The "Content-Type" dropdown is set to `application/json`.

⇒ List:

The screenshot shows a browser window with the title "Multi-Tenant DB Manager - Swagger". The address bar displays the URL "127.0.0.1:8000/docs#/default/restore_tenant_tenant_restore_post". The main content area is a Swagger UI interface for the "/tenant/list" endpoint. It includes sections for "Parameters" (with "No parameters"), "Responses", and "Call". The "Call" section shows a curl command and a request URL "http://127.0.0.1:8000/tenant/list". The "Server response" section shows a 200 status code with a "Response body" containing a JSON array of tenant objects. One tenant object is highlighted, showing fields like "id", "name", "active", "created_at", and "updated_at". Below the response, there's a "Successful Response" section with a media type of "application/json".

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:

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import List
import subprocess
import os
import psycopg2
from dotenv import load_dotenv
import boto3
import botocore
import json
import re

# Load environment variables from .env
load_dotenv()

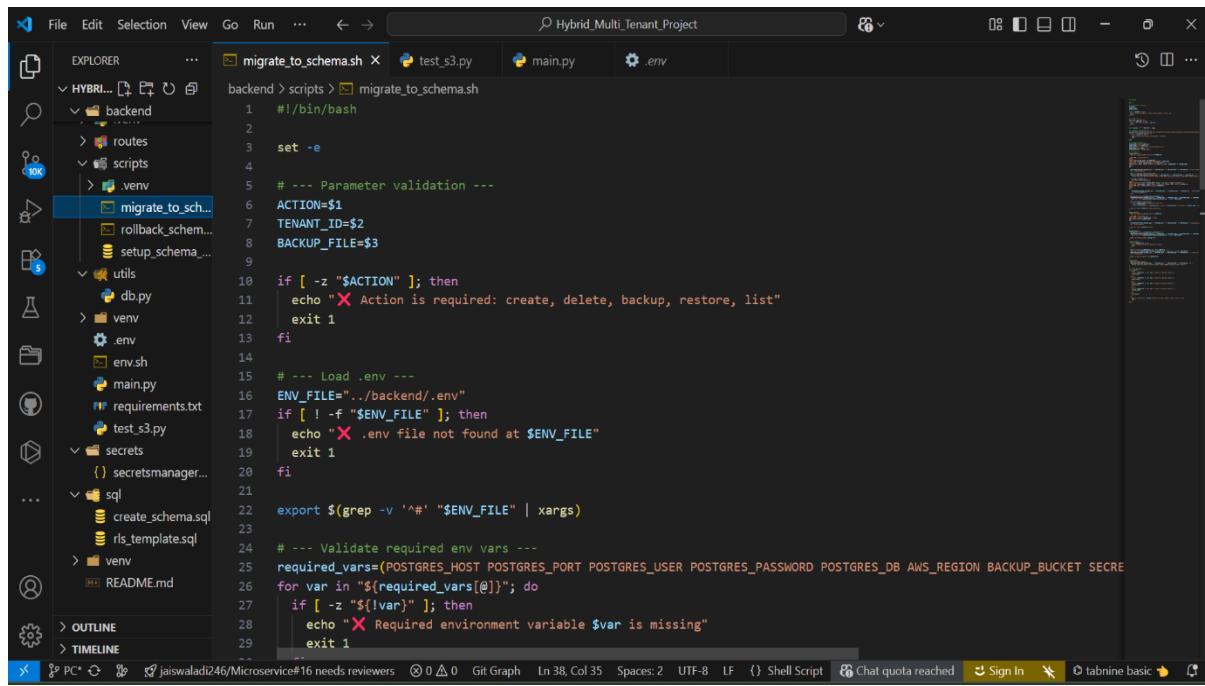
app = FastAPI(title="Multi-Tenant DB Manager")

# Environment variables
DB_HOST = os.getenv("POSTGRES_HOST")
DB_PORT = int(os.getenv("POSTGRES_PORT"))
DB_NAME = os.getenv("POSTGRES_DB")
DB_USER = os.getenv("POSTGRES_USER")
DB_PASS = os.getenv("POSTGRES_PASSWORD")
AWS_REGION = os.getenv("AWS_REGION")
BACKUP_BUCKET = os.getenv("BACKUP_BUCKET")
SECRETS_PREFIX = os.getenv("SECRETS_PREFIX", "multi-client")

# AWS Clients
secrets_client = boto3.client("secretsmanager", region_name=AWS_REGION)
```

Bash Script Integration:

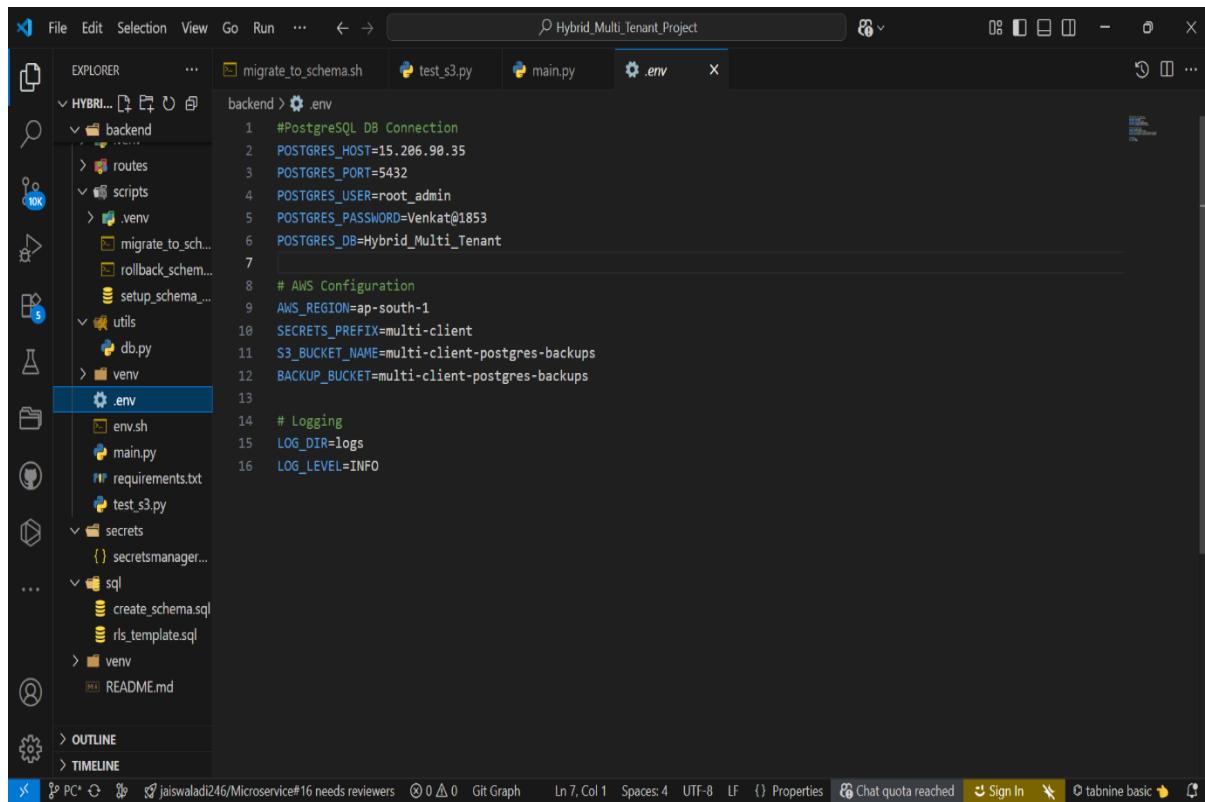
- ./migrate_to_schema.sh/create_tenant.sh
- ./migrate_to_schema.sh/backup_tenant.sh
- ./migrate_to_schema.sh/restore_tenant.sh
- ./migrate_to_schema.sh/delete_tenant.sh
- ./migrate_to_schema.sh/list_tenant.sh



The screenshot shows the Visual Studio Code interface with the file 'migrate_to_schema.sh' selected in the Explorer panel. The code editor displays the following bash script:

```
#!/bin/bash
set -e
# --- Parameter validation ---
ACTION=$1
TENANT_ID=$2
BACKUP_FILE=$3
if [ -z "$ACTION" ]; then
    echo "X Action is required: create, delete, backup, restore, list"
    exit 1
fi
# --- Load .env ---
ENV_FILE="../backend/.env"
if [ ! -f "$ENV_FILE" ]; then
    echo "X .env file not found at $ENV_FILE"
    exit 1
fi
export $(grep -v '^#' "$ENV_FILE" | xargs)
# --- Validate required env vars ---
required_vars=(POSTGRES_HOST POSTGRES_PORT POSTGRES_USER POSTGRES_PASSWORD POSTGRES_DB AWS_REGION BACKUP_BUCKET SECRETS_PREFIX)
for var in "${required_vars[@]}"; do
    if [ -z "${!var}" ]; then
        echo "X Required environment variable $var is missing"
        exit 1
    fi
done
```

.env file:



The screenshot shows the Visual Studio Code interface with the file '.env' selected in the Explorer panel. The code editor displays the following environment variables:

```
POSTGRES_HOST=127.0.0.1
POSTGRES_PORT=5432
POSTGRES_USER=root
POSTGRES_PASSWORD=Venkat@1853
POSTGRES_DB=Hybrid_Multi_Tenant
AWS_REGION=ap-south-1
SECRETS_PREFIX=multi-client
S3_BUCKET_NAME=multi-client-postgres-backups
BACKUP_BUCKET=multi-client-postgres-backups
LOG_DIR=logs
LOG_LEVEL=INFO
```

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

```
wipro_db      | client_wipro | UTF8   | libc          | C.UTF-8 | C.UTF-8 |           | -Tc/client_wipro
(8 rows)
postgres=# \dn
          List of schemas
 Name    | Owner
 public  | pg_database_owner
(1 row)

postgres=# \c Hybrid_Multi_Tenant
SSL connection (protocol: TLSv1.3, cipher: AES_256_GCM_SHA384, compression: off)
You are now connected to database "Hybrid_Multi_Tenant" as user "root_admin".
Hybrid_Multi_Tenant=# \dn
          List of schemas
 Name    | Owner
 public  | pg_database_owner
tenant_AirIndia | root_admin
tenant_GenLakes | root_admin
tenant_airIndia | root_admin
tenant_genLakes | root_admin
tenant_sekhar   | root_admin
tenant_venkat   | root_admin
(7 rows)

Hybrid_Multi_Tenant# \q
i-07dbefa3fce0074ad (Hybrid_Multi-Tenancy_Project)
PublicIPs: 3.111.55.95 PrivateIPs: 172.31.10.135
```

⇒ 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
-