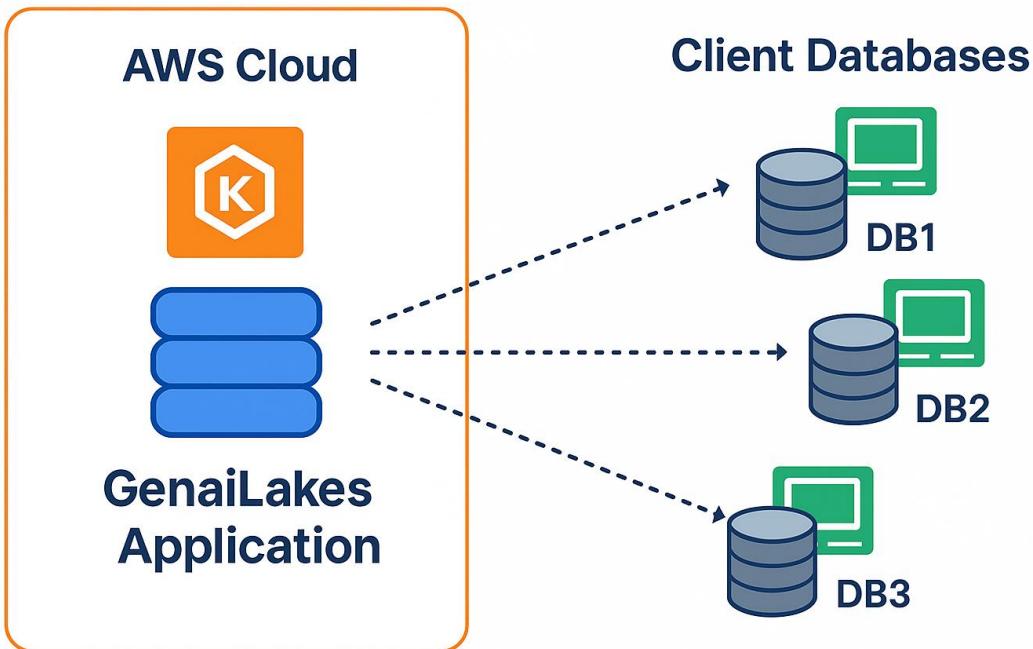


# Multiple Tenant Databases



## Project Overview

- GenAI Lakes is a **secure, scalable, and cost-effective** application designed to manage **multiple tenant databases** across different AWS environments. It enables each client to have **their own isolated PostgreSQL database** (hosted on AWS VMs) while providing **centralized application control, secure credential storage, and automated backup/restore** processes.
- With this architecture, businesses can ensure **data security, tenant isolation, and high availability**, while still benefiting from a **single, unified application interface**.

## Architecture Overview

### Components Used:

- **AWS EC2 (2 Virtual Machines)** – Each hosting PostgreSQL databases.
- **AWS S3 Bucket** – Stores automated and on-demand backups for quick recovery.
- **AWS Secrets Manager** – Securely stores and retrieves tenant database credentials.
- **AWS IAM** – Manages user roles, permissions, and security policies.
- **pgAdmin4 (Local)** – Centralized database administration from a local machine.

## Flow:

1. **GenAI Lakes Application** connects to the appropriate tenant database (DB1, DB2, ...) based on the client.
2. **Tenant credentials** are securely fetched from AWS Secrets Manager.
3. **Automated backups** are stored in AWS S3, with on-demand restore capability.
4. **pgAdmin4** allows centralized monitoring and administration.

## Main Aim of the Project:

To provide a single application that can securely manage multiple client databases across AWS environments with:

- **Tenant isolation** for security and compliance.
- **Automated & on-demand backups** for disaster recovery.
- **Centralized control** for easier management and scalability.

This architecture is ideal for **multi-client SaaS products, data platforms, and enterprise services** that require **separate databases per customer**.

## Key Features:

- **Multi-Tenant Isolation** – Each client has their own **PostgreSQL** database to ensure no data leakage.
- **Secure Credential Storage** – Uses **AWS Secrets Manager** to protect database credentials.
- **Automated & On-Demand Backups** – AWS S3 stores versioned backups with restore capability.
- **High Availability** – Spread across multiple AWS VMs for redundancy.
- **Centralized Administration** – Manage all tenant databases via **pgAdmin4** from a single interface.
- **Scalable Architecture** – Easily add new tenants without affecting existing ones.

## Benefits (Pros)

- **Security** – Credentials and data are stored securely using AWS Cloud.
- **Scalability** – Add or remove tenants quickly without downtime.
- **Disaster Recovery** – S3 backup and restore ensures business continuity.
- **Performance Isolation** – One tenant's workload does not impact another's.

## Why Tenant Databases Are Important

Tenant databases are **the backbone** of secure, multi-tenant applications. They:

- Ensure **data isolation**, preventing one client from accessing another's data.
- Allow **customized configurations** for each client (indexes, extensions, optimizations).

- Improve **performance predictability**, as queries are not competing across tenants.
- Simplify **compliance audits** by maintaining separate database instances.

## Example Use Cases

- **SaaS Platforms** – One app, multiple clients, each with its own database.
- **HR & Payroll Systems** – Each company's employee data is isolated.
- **Healthcare Data Management** – Separate patient data per hospital or clinic.
- **Educational Portals** – Different schools/universities have separate datasets.

## AWS EC2 Virtual Machines:

The image contains two screenshots of the AWS EC2 Instances page, demonstrating the creation of separate database instances in different regions.

**Screenshot 1 (Top): Asia Pacific (Mumbai)**

- Region:** ap-south-1
- Instance Details:**
  - Name:** Database1
  - Instance ID:** i-0c5771c924871c465
  - State:** Running
  - Type:** t2.micro
  - Public DNS:** 15.206.92.94
  - Private IP:** 172.31.37.1

**Screenshot 2 (Bottom): United States (N. Virginia)**

- Region:** us-east-1
- Instance Details:**
  - Name:** Database2
  - Instance ID:** i-0fa84f171305b629f
  - State:** Running
  - Type:** t2.micro
  - Public DNS:** 54.197.2.231
  - Private IP:** 172.31.21.9

## Security Groups:

- Security Groups Created for testing purpose (Allowed All Traffic) in the Production we can allow Specific required Ports.

The screenshot shows the AWS EC2 Security Groups page for a specific security group (sg-0588835fb36ddb485). The page displays the owner (602351392486), inbound rules count (4 Permission entries), and outbound rules count (1 Permission entry). The 'Inbound rules' tab is selected, showing four rules:

Name	Security group rule ID	IP version	Type	Protocol
-	sgr-03ef1320f143f4894	IPv4	HTTPS	TCP
-	sgr-0be6109dde6ba20b8	IPv4	All traffic	All
-	sgr-049477a14031b57d0	IPv4	HTTP	TCP
-	sgr-00b3ffee2d0f0a6e9	IPv4	SSH	TCP

## Identity and Access Management (IAM):

- Created IAM Role for Allowed the permissions for S3 Bucket and Secret Manager.
- Attach the IAM Roles to your EC2 Virtual machines.
- Created User for AWS CLI Access.

The screenshot shows the AWS IAM Roles page. It lists three roles: AWSServiceRoleForSupport, AWSServiceRoleForTrustedAdvisor, and Multiple-Tenant-DB. The Multiple-Tenant-DB role is selected. Below the table, there are sections for 'Access AWS from your non AWS workloads', 'X.509 Standard', and 'Temporary credentials'.

Role name	Trusted entities	Last activity
AWSServiceRoleForSupport	AWS Service: support (Service-Linker)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service)	-
Multiple-Tenant-DB	AWS Service: ec2	15 minutes ago

The screenshot shows the AWS Identity and Access Management (IAM) service interface. On the left, a sidebar menu includes 'Dashboard', 'Access management' (with 'Users' selected), 'Roles', 'Policies', 'Identity providers', and 'Account settings'. Under 'Access reports', there is an 'Access Analyzer' option. At the bottom of the sidebar are links for 'CloudShell' and 'Feedback'. The main content area is titled 'Users (1/1)' and contains a table with one row. The table columns are 'User name', 'Path', 'Groups', 'Last activity', 'MFA', 'Password age', and 'Console last sign'. The user 'Multiple-Tenant-DB' is listed with a checkmark next to it. The 'Last activity' column shows '3 hours ago'. The 'Password age' column shows 'Yesterday'. The 'Delete' button is visible at the top right of the table. The top navigation bar shows tabs for 'Instances | EC2 | ap-south-1', 'EC2 | us-east-1', 'Multi/Tenant/db2 | Secret...', 'Users | IAM | Global', and 'S3 buckets | S3 | ap-south-1'. The browser address bar shows the URL: <https://us-east-1.console.aws.amazon.com/iam/home?region=ap-south-1#/users>. The top right corner displays the account ID: 6023-5139-2486 and the user name Venkat.

## Secret Manager:

➤ Created Secret Manager to store the Tenants Credentials

The screenshot shows the AWS Secrets Manager service interface. On the left, a sidebar menu includes 'AWS Secrets Manager' (selected) and 'Secrets'. The main content area is titled 'Secrets' and contains a table with ten rows. The table columns are 'Secret name', 'Description', 'Last retrieved (UTC)', and 'Deleted on (UTC)'. The secrets listed are: Multi/RDS/db3, Multi/Tenant/db2, Multi/tenant/db1, Tech\_credentials, TechMahindra\_credentials, Genailakes\_credentials, 5647-TCS\_credentials, JIO\_credentials, and two unnamed secrets represented by '-' in the Description column. The 'Store a new secret' button is located at the top right of the table. The top navigation bar shows tabs for 'Instances | EC2 | ap-south-1', 'EC2 | us-east-1', 'Secrets | Secrets Manager' (selected), 'Dashboard | IAM | Global', and 'S3 buckets | S3 | ap-south-1'. The browser address bar shows the URL: <https://ap-south-1.console.aws.amazon.com/secretsmanager/listsecrets?region=ap-south-1>. The top right corner displays the account ID: 6023-5139-2486 and the region: Asia Pacific (Mumbai). The user name Venkat is also shown. The bottom of the screen includes standard AWS footer links for 'CloudShell' and 'Feedback'.

The screenshot displays two separate AWS Secrets Manager secret details pages, one for 'Multi/tenant/db1' and one for 'Multi/Tenant/db2'. Both pages have a similar layout with sections for 'Secret details', 'Secret value', and 'Resource permissions'.

**Secret details:**

- Encryption key:** aws/secretsmanager
- Secret name:** Multi/tenant/db1 (Multi/Tenant/db2)
- Secret ARN:** arn:aws:secretsmanager:ap-south-1:162351392486:secret:Multi/tenant/db1-7CgAg5 (arn:aws:secretsmanager:ap-south-1:162351392486:secret:Multi/Tenant/db2-4Xc2S)

**Secret value:**

**Key/value** (selected) **Plaintext**

Secret key	Secret value
host	15.206.97.94
port	5432
database	mycompany
user	admin
password	admin@123

**Resource permissions - optional** info

Add or edit a resource policy to access secrets across AWS accounts.

**Actions:** Copy Edit

**Overview** **Rotation** **Versions** **Replication** **Tags**

**Secret details:**

- Encryption key:** aws/secretsmanager
- Secret name:** Multi/Tenant/db2
- Secret ARN:** arn:aws:secretsmanager:ap-south-1:162351392486:secret:Multi/Tenant/db2-4Xc2S

**Secret value:**

**Key/value** (selected) **Plaintext**

Secret key	Secret value
host	54.197.2.23
port	5432
database	mycompany1
user	Verklat
password	Verklat@123

**Resource permissions - optional** info

Add or edit a resource policy to access secrets across AWS accounts.

**Actions:** Copy Edit

**Overview** **Rotation** **Versions** **Replication** **Tags**

## S3 Bucket:

- Created S3 Bucket for to store the Tenants data for Backups & Restore if we required.

The screenshot shows the AWS S3 console interface. On the left, there's a sidebar with various options like General purpose buckets, Directory buckets, Table buckets, etc. The main area is titled 'General purpose buckets (1/1)' and shows a table with one row. The row details a bucket named 'multi-client-postgres-backups' located in the 'Asia Pacific (Mumbai) ap-south-1' region, created on July 26, 2025, at 15:08:19 (UTC+05:30). There are buttons for 'Copy ARN', 'Empty', 'Delete', and 'Create bucket'. To the right, there are sections for 'Account snapshot' and 'External access summary - new'.

## Backend:

- python -m venv venv
- source venv/Scripts/activate
- pip install -r requirements.txt
- uvicorn main:app –reload
- Select the URL <http://127.0.0.1:8000>
- You can connect successfully

The screenshot shows a browser window with the URL '127.0.0.1:8000'. The page content is a JSON response: {"service": "multi-tenant-backend", "status": "ok"}. This indicates that the application is running successfully.

The screenshot shows the VS Code interface with the project structure on the left and the code editor on the right. The code editor displays `main.py` with the following content:

```

4  from fastapi.middleware.cors import CORSMiddleware
5
6  app = FastAPI()
7
8  app.add_middleware(
9      CORSMiddleware,
10     allow_origins=["*"], # Adjust this to your needs
11     allow_credentials=True,
12     allow_methods=["*"],
13     allow_headers=["*"],
14 )
15
16 app.include_router(auth.router)
17 app.include_router(tenant.router)

```

The terminal tab shows the following command history:

```

singh@Venkatesh-09 MINGW64 ~/Downloads/Multiple_DB (PC)
$ cd backend/
$ source venv/Scripts/activate
(venv)
singh@Venkatesh-09 MINGW64 ~/Downloads/Multiple_DB/backend (PC)
$ unicorn main:app --reload
INFO: Will watch for changes in these directories: ['C:\\Users\\singh\\Downloads\\Multiple_DB\\backend']
INFO: Unicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [15180] using Watchfiles
INFO: Started server process [10228]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:62250 - "GET / HTTP/1.1" 200 OK

```

## .env

- To store each Tenants Credentials in .env file in the backend so it can fetch the credentials and it will connect/Routes to Correct Tenants Databases.

The screenshot shows the VS Code interface with the project structure on the left and the code editor on the right. The code editor displays the `.env` file with the following content:

```

#tenant_1_db_url=postgres://ec2_user:password123@localhost:5432/ec2_tenant_db
#tenant_rds_db_url=postgres://rds_user:password456@rds-endpoint:5432/rds_tenant_db

# App
FRONTEND_ORIGIN=http://localhost:3000
TMP_DIR=/tmp

# AWS
AWS_REGION=ap-south-1
AWS_ACCESS_KEY_ID=AKIAIYPPXQ23TIYUWGDHU
AWS_SECRET_ACCESS_KEY=yTgWID0P0CpZ159FXMRSASYR75Jtg9POZuzlWA
S3_BUCKET_NAME=multi-client-postgres-backups
S3_REGION=ap-south-1

TENANT_SECRETS = (
    Tenant_1=Multi/Tenant/db1,
    Tenant_2=Multi/Tenant/db2,
)

#Testing environment variables
[{"host": "15.206.92.94", "port": 5432, "database": "mycompany", "user": "admin", "password": "admin@123"}, {"host": "18.215.147.33", "port": 5432, "database": "mycompany1", "user": "Venkat", "password": "Venkat@123"}]

# PG DB-postgres
# PG_HOST=localhost
# PG_PORT=5432
# PG_USER=postgres
# PG_PASSWORD=160524

```

## Frontend:

```

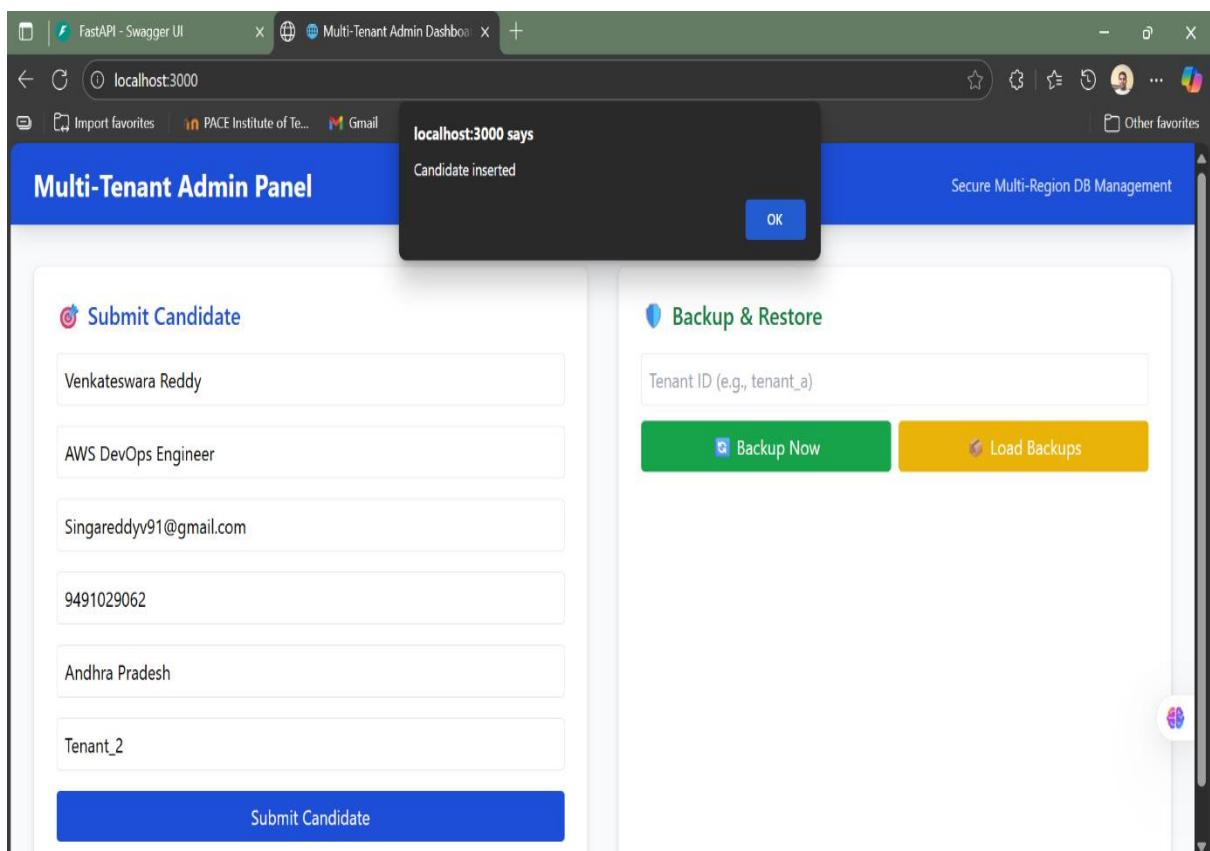
singh@Venkatesh-09 MINGW64 ~/Downloads/Multiple_DB (PC)
$ cd frontend
singh@Venkatesh-09 MINGW64 ~/Downloads/Multiple_DB/frontend (PC)
$ python -m http.server 3000
Serving HTTP on :: port 3000 (http://[::]:3000) ...
::1 - - [12/Aug/2025 16:17:50] "GET / HTTP/1.1" 304 -
::1 - - [12/Aug/2025 16:17:54] "code 404, message File not found"
::1 - - [12/Aug/2025 16:17:54] "GET /favicon.ico HTTP/1.1" 404 -

```

## Successfully Connected to the Correct Tenant Databases like (DB1 & DB2)

The screenshot displays two windows side-by-side. The left window is the "Multi-Tenant Admin Panel" with a blue header. It shows a "Submit Candidate" form with fields for name, job role, email, phone, location, and tenant ID. A success message "Candidate inserted" is displayed in a modal. The right window is "pgAdmin 4" showing the "public.candidates" table from the "DB\_1" database. The table contains 10 rows of candidate data.

	id	name	job_role	email	phone	location
1	1	Venkatesh	AWS DevOps Engineer	Venka123@gmail.com	9491029062	Hyderabad
2	2	string	string	string	string	string
3	3	Eswar	DATA SCIENCE	eswar123@gmail.com	7897866687	Chennai
4	4	Venkatesh	data	data123@gmail.com	939459485	Delhi
5	5	Ramesh	engineer	ramesh123@gmail.com	394398889	Lucknow
6	6	Rajesh	Data Engineer	Rajesh456@gmail.com	3883437998	Lucknow
7	7	Kumari	manager	kumari1456@gmail.com	893887490	Chennai
8	8	Kumari	manager	kumari1456@gmail.com	893887490	Chennai
9	9	venkatesh	AWS DevOps	singareddyv91@gmail.com	9491029062	Mumbai
10	10	Rajasekhar Reddy	Azure DevOps Engineer	Sekhar1532@gmail.com	6301999310	Hyderabad



The screenshot shows the pgAdmin 4 interface. The left sidebar is the Object Explorer, displaying the database structure. The main area shows a query editor with the following SQL code:

```
1 SELECT * FROM public.candidates
2 ORDER BY id ASC
```

The results of the query are displayed in a table:

	<b>id</b> [PK] integer	<b>name</b> text	<b>job_role</b> text	<b>email</b> text	<b>phone</b> text	<b>location</b> text
1	1	Sekhar	Azure	Sekhar123@gmail.com	3842483489	AP
2	2	Narayana	Data Analytic	narayana198@gmail.com	439043549	Pune
3	3	Eswar	AWS DevOps	eswar123@gmail.com	3487347888	Pune
4	4	Pawan	Software	pawan12345@gmail.com	93948938	string
5	5	Sampath	Engineer	Sampath2678@gmail.com	6839958843	Jammu
6	6	Srinivasulu	DevOps	Srinivasulu123@gmail.com	9364899979	Hyderabad
7	7	Venkateswara Reddy	AWS DevOps Engineer	Singareddy91@gmail.com	9491029062	Andhra Pradesh

Total rows: 7 Query complete 00:00:02.556 CRLF Ln 1, Col 1

## Pgadmin4 Centralized Database Connections:

The screenshot shows the pgAdmin4 interface with a focus on a centralized database connection configuration. The left sidebar displays servers and databases. A context menu is open over the 'DB\_1' database entry, showing options like 'General', 'Connection', 'Parameters', 'SSH Tunnel', 'Advanced', 'Post Connection SQL', and 'Tags'. The 'Connection' tab is selected, showing fields for 'Host name/address' (15.206.92.94), 'Port' (5432), 'Maintenance database' (mycompany), 'Username' (admin), and 'Kerberos authentication?' (disabled). To the right, a table viewer displays a list of records with columns 'phone' and 'location'. The table data is as follows:

phone	location
3842483489	AP
439043549	Pune
3487347888	Pune
93948938	string
6839958843	Jammu
9364899979	Hyderabad
9491029062	Andhra Pradesh

This screenshot is identical to the one above, showing the pgAdmin4 interface with a focus on a centralized database connection configuration. The left sidebar displays servers and databases. A context menu is open over the 'DB\_1' database entry, showing options like 'General', 'Connection', 'Parameters', 'SSH Tunnel', 'Advanced', 'Post Connection SQL', and 'Tags'. The 'Connection' tab is selected, showing fields for 'Host name/address' (15.206.92.94), 'Port' (5432), 'Maintenance database' (mycompany), 'Username' (admin), and 'Kerberos authentication?' (disabled). To the right, a table viewer displays a list of records with columns 'phone' and 'location'. The table data is as follows:

phone	location
3842483489	AP
439043549	Pune
3487347888	Pune
93948938	string
6839958843	Jammu
9364899979	Hyderabad
9491029062	Andhra Pradesh

- IF you try to Connect to the Another Database, it will Failures due to the Credentials are not included in the (.env file & Secret Manager).

The screenshot shows a code editor interface with multiple tabs open. The active tab is `main.py`, which contains Python code. A terminal window is open below the code editor, displaying an error message:

```

return await get_async_backend().run_sync_in_worker_thread(
    func, args, abandon_on_cancel=abandon_on_cancel, limiter=limiter
)
File "C:\Users\singa\Downloads\Multiple_DB\backend\venv\lib\site-packages\anyio\_backends\asyncio.py", line 2476, in run_sync_in_worker_thread
    return await future
File "C:\Users\singa\Downloads\Multiple_DB\backend\venv\lib\site-packages\anyio\_backends\asyncio.py", line 967, in run
    res = Open file in editor (ctrl + click) ("ps")
File "C:\Users\singa\AppData\Local\Programs\Python\Python313\lib\contextlib.py", line 141, in __enter__
    return next(self.gen)
File "C:\Users\singa\Downloads\Multiple_DB\backend\utils\db_router.py", line 55, in get_db_connection_for_tenant
    cfg = get_tenant_db_config(tenant_name)
File "C:\Users\singa\Downloads\Multiple_DB\backend\utils\db_router.py", line 49, in get_tenant_db_config
    raise Exception(f"Tenant credentials not found in Secrets Manager"
                  f"for env for '{tenant_name}'")
Exception: Tenant credentials not found in Secrets Manager for env for 'Tenant_3'

```

The terminal also shows other tabs like `frontend` and `backend`.

## Conclusion

The GenAI Lakes Application enables secure, scalable, and client-specific data management with full AWS integration. It ensures tenant data safety, operational efficiency, and disaster recovery, making it a reliable choice for multi-client businesses.