Project: Intelligent Document Querying System

1. Project Overview

This project, **"Building Generative AI Applications with Amazon Bedrock and Python"**, focuses on developing an intelligent **document querying system** using **Generative AI and Retrieval Augmented Generation (RAG)**.
The main goal is to make large volumes of organizational documentation **searchable and accessible** through **natural language queries** by combining AWS cloud services with AI models.

Key services used:

- **Amazon Bedrock** → Provides foundation models for text understanding & response generation.
- **Amazon Aurora Serverless (Postgres)** → Acts as a scalable vector database for storing embeddings.
- **Amazon S3** → Stores PDF documents that serve as the knowledge base.
- **Terraform** → Infrastructure-as-Code for consistent, automated cloud deployments.
- **Python scripts** → Handle document ingestion, uploading, and querying.

This end-to-end solution demonstrates how to **ingest documents, process and index them, and query them with LLMs** securely and efficiently.

Base infrastructure creation

Deployment of VPC, Aurora Postgres Serverless, and S3 bucket using Terrafor

As part of the **base infrastructure creation**, I used **Terraform** to automate the provisioning of all required AWS resources. The deployment included:

- **VPC** → A Virtual Private Cloud was created with proper CIDR block allocation to securely host resources.
- **Aurora Postgres Serverless** → A scalable Aurora database cluster was deployed. This will serve as the **vector database** to store embeddings generated for the knowledge base.
- **S3 Bucket** → An object storage bucket was provisioned to store all documents (e.g., PDFs) that form the knowledge corpus for the Bedrock Knowledge Base.

After running:

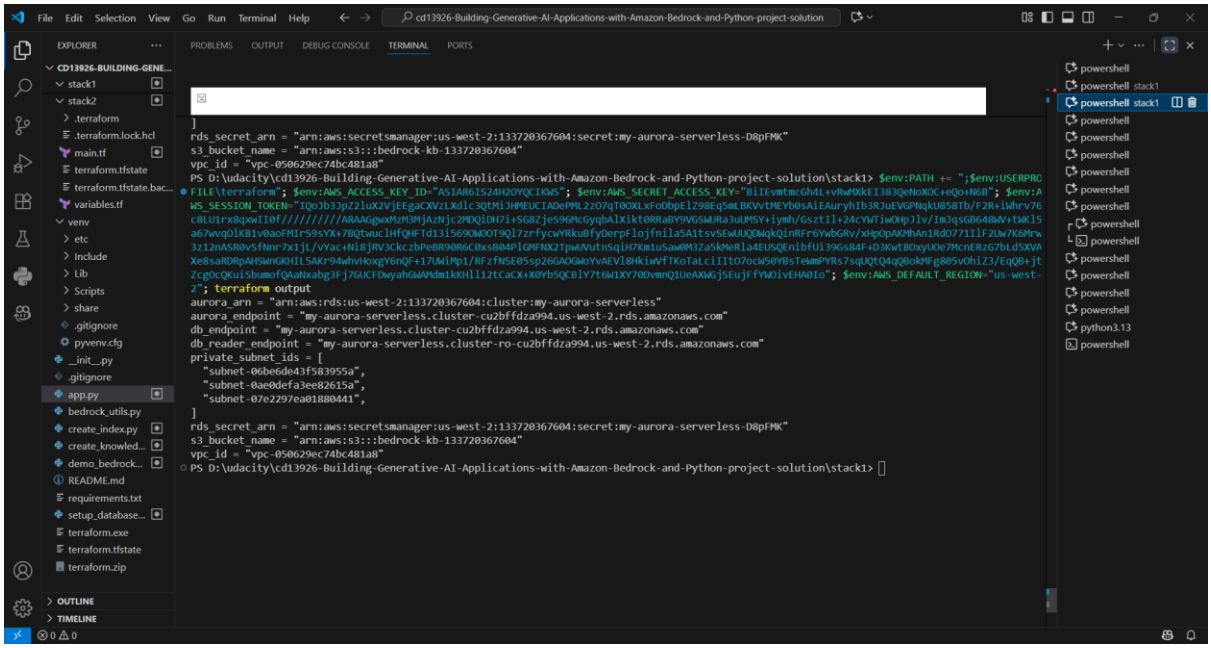
terraform init

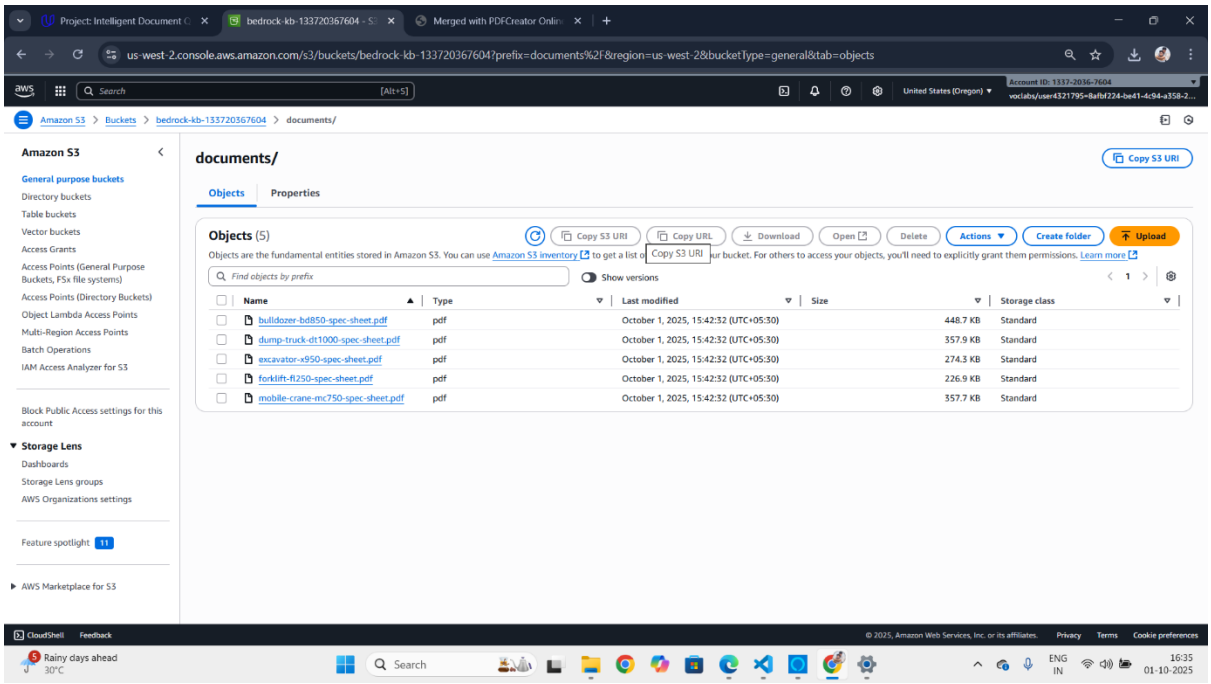
terraform apply

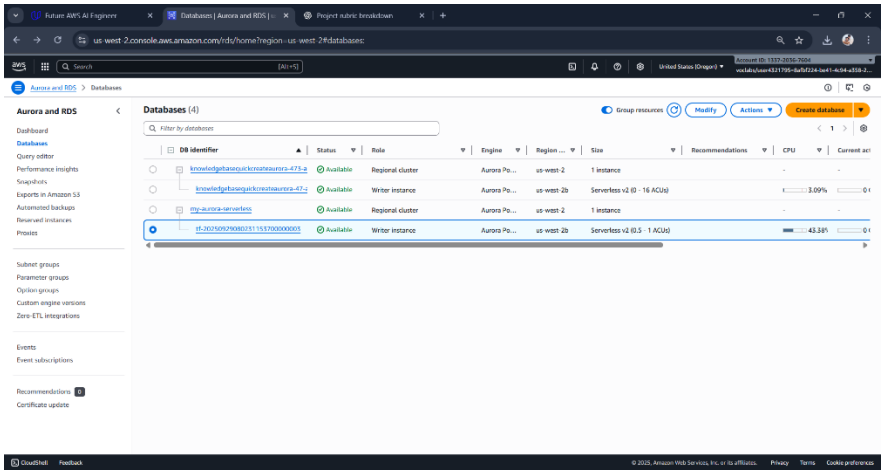
Terraform successfully created these resources, and the **apply output screenshot** confirms resource creation with all identifiers (VPC ID, Aurora endpoint, and S3 bucket name) displayed as outputs.

Result:

The infrastructure was successfully deployed, verified, and is ready for use in later stages of the project (knowledge base integration and querying).







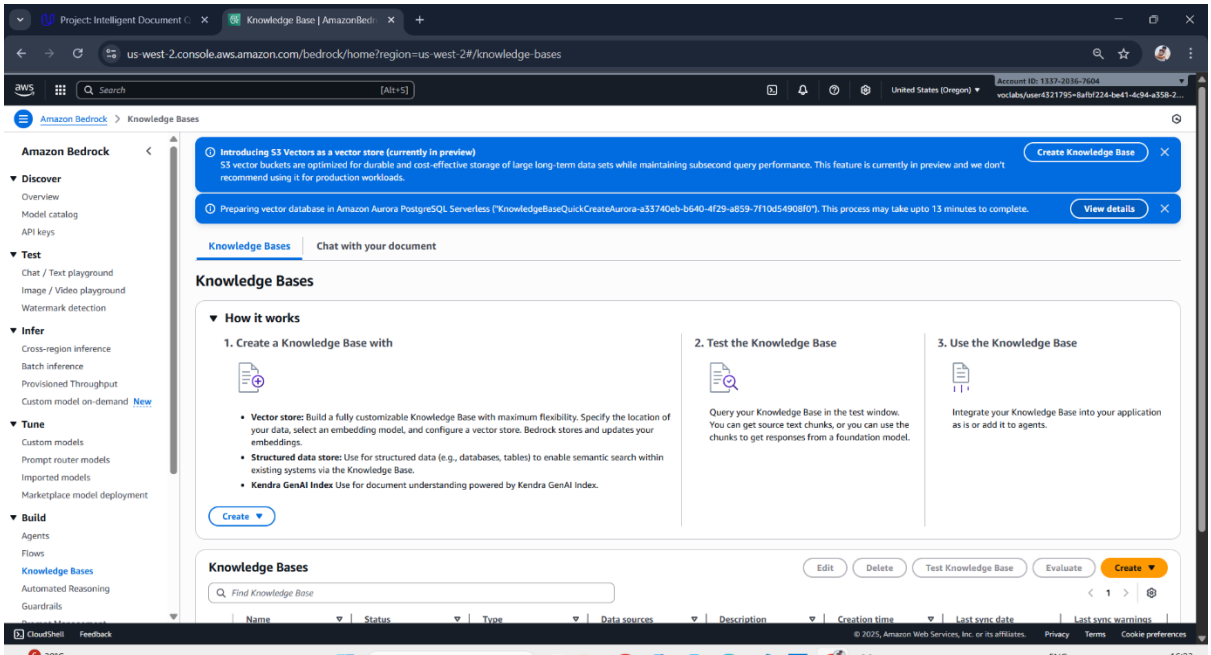

```
PS D:\udacity\cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution\stack2> $env:PATH += ";$env:USERPROFILE\terraform"; terraform output
s3_bucket_name = "bedrock-kb-133720367604"
stack2_status = "Stack 2 completed - Documents uploaded to S3"
uploaded_documents = [
  "documents/bulldozer-bd850-spec-sheet.pdf",
  "documents/dump-truck-dt1000-spec-sheet.pdf",
  "documents/excavator-x950-spec-sheet.pdf",
  "documents/forklift-fl250-spec-sheet.pdf",
  "documents/mobile-crane-mc750-spec-sheet.pdf",
]
PS D:\udacity\cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution\stack2>
```

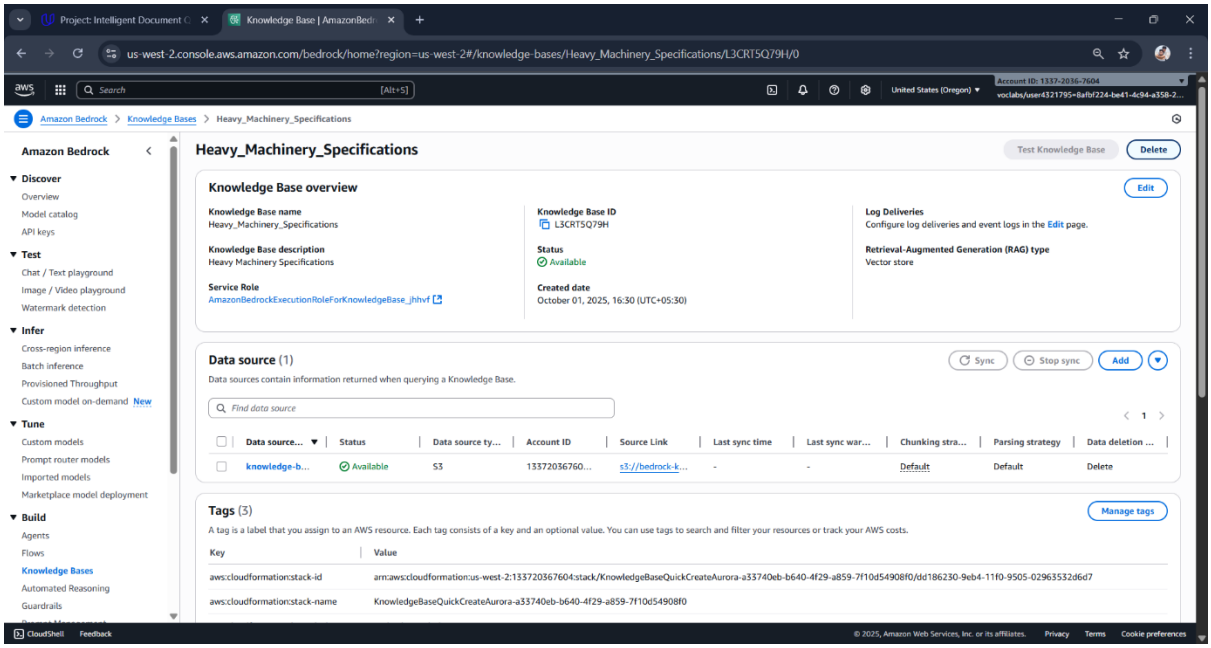
```
\stack1> terraform apply -auto-approve
aurora_arn = "arn:aws:rds:us-west-2:133720367604:cluster:my-aurora-serverless"
aurora_endpoint = "my-aurora-serverless.cluster-cu2bffdza994.us-west-2.rds.amazonaws.com"
db_endpoint = "my-aurora-serverless.cluster-cu2bffdza994.us-west-2.rds.amazonaws.com"
db_reader_endpoint = "my-aurora-serverless.cluster-ro-cu2bffdza994.us-west-2.rds.amazonaws.com"
private_subnet_ids = [
  "subnet-010f91677bcca70d7",
  "subnet-0a1b9b2ee321af52b",
  "subnet-08f8ebdea37df3806",
]
rds_secret_arn = "arn:aws:secretsmanager:us-west-2:133720367604:secret:my-aurora-serverless-3yBTSs"
s3_bucket_name = "arn:aws:s3:::bedrock-kb-133720367604"
vpc_id = "vpc-0d24aad5de0cac8e5"
```

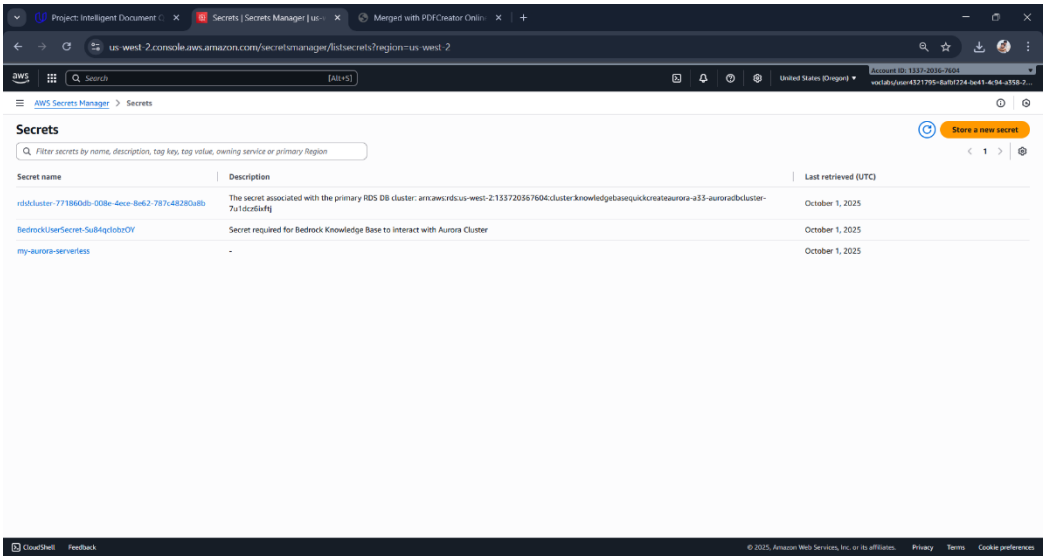
```
PS D:\udacity\cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution\stack2> terraform apply -auto-approve
Outputs:

s3_bucket_name = "bedrock-kb-133720367604"
stack2_status = "Stack 2 completed - Documents uploaded to S3"
uploaded_documents = [
  "documents/bulldozer-bd850-spec-sheet.pdf",
  "documents/dump-truck-dt1000-spec-sheet.pdf",
  "documents/excavator-x950-spec-sheet.pdf",
  "documents/forklift-fl250-spec-sheet.pdf",
  "documents/mobile-crane-mc750-spec-sheet.pdf",
]
```

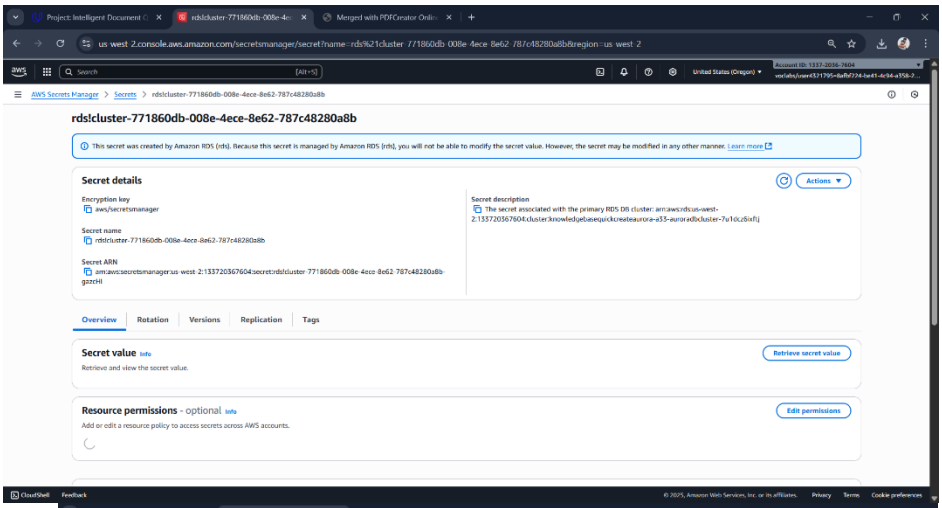
```
PS D:\udacity\cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution\stack2> aws s3 ls s3://bedrock-kb-133720367604/documents/
2025-10-01 15:42:32     459443 bulldozer-bd850-spec-sheet.pdf
2025-10-01 15:42:32     366473 dump-truck-dt1000-spec-sheet.pdf
2025-10-01 15:42:32     280903 excavator-x950-spec-sheet.pdf
2025-10-01 15:42:32     232388 forklift-fl250-spec-sheet.pdf
2025-10-01 15:42:32     366291 mobile-crane-mc750-spec-sheet.pdf
```

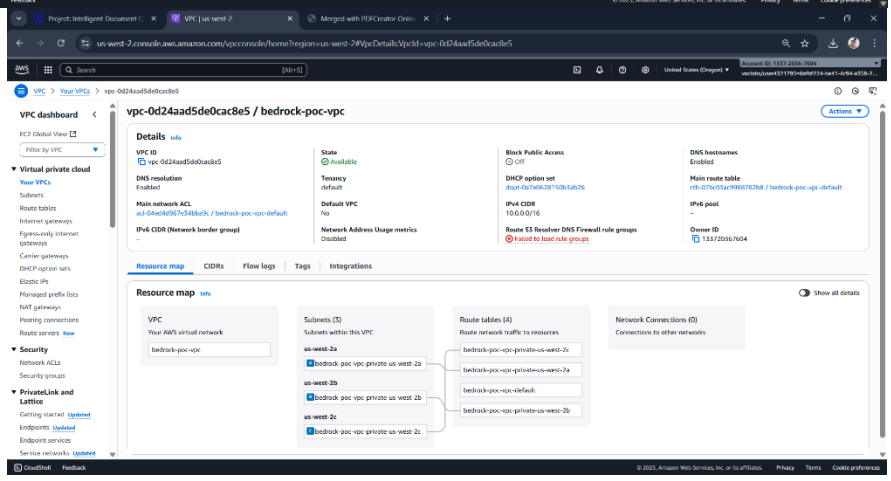
```
🧠 Bedrock Knowledge Base Demo Setup
==================================================
✅ Bedrock Claude 3 model access confirmed!
Response: Heavy machinery generally refers to large, powerful equipment and vehicles used for industrial, cons...
✅ Demo Knowledge Base configuration created:
   KB ID: DEMO-KB-HEAVY-MACHINERY-2025
   Documents: 5 PDFs

🎉 Demo setup complete!
   Focus folder in explorer (ctrl + click)
IERY-2025
PS D:\udacity\cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution> aws sts get
-caller-identity
{
    "UserId": "AROAR6IS24H2B5X6TE3BN:user4321795=8afbf224-be41-4c94-a358-2ae03fba2edf",
    "Account": "133720367604",
    "Arn": "arn:aws:sts::133720367604:assumed-role/voclabs/user4321795=8afbf224-be41-4c94-a358-2ae03fba2edf"
}
```

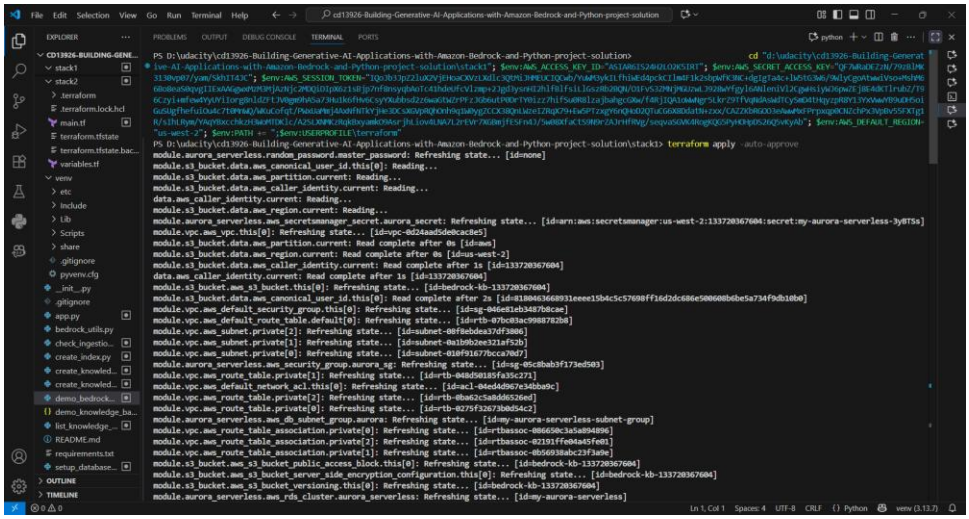





Secrets
Store a new secret
Secret name
Description
Last retrieved (UTC)
rdsl!cluster-771860db-008e-4ece-8e62-787c48280a8b
The secret associated with the primary RDS DB cluster: arn:aws:rds:us-west-2:133720367604:cluster:knowledgebasequickcreateaurora-a33-auroradbcluster-7u1dcz6ixfj
October 1, 2025
BedrockUserSecret-Su84qclobzOY
Secret required for Bedrock Knowledge Base to interact with Aurora Cluster
October 1, 2025
my-aurora-serverless
October 1, 2025

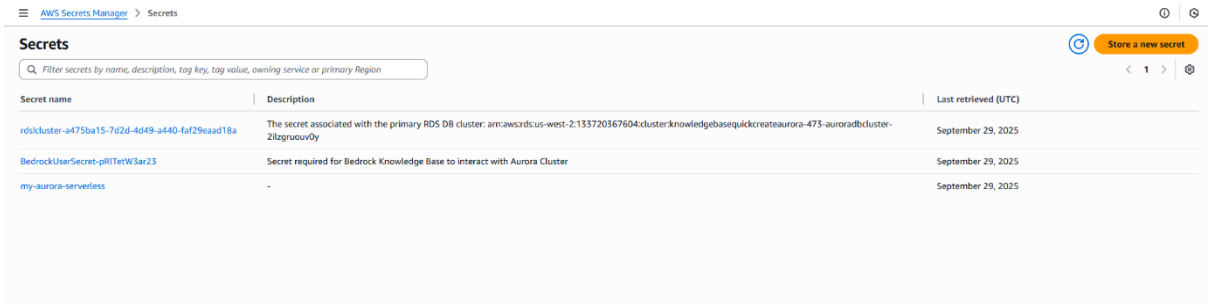

rdsl!cluster-771860db-008e-4ece-8e62-787c48280a8b
Secret details
Actions
Overview
Rotation
Versions
Replication
Tags
Secret value
Retrieve secret value
Resource permissions - optional
Edit permissions


vpc-0d24aad5de0cac8e5 / bedrock-poc-vpc
Actions
Details
Resource map
CIDRs
Flow logs
Tags
Integrations
VPC dashboard
bedrock-poc-vpc
bedrock-poc-vpc-private-us-west-2a
bedrock-poc-vpc-private-us-west-2b
bedrock-poc-vpc-private-us-west-2c
bedrock-poc-vpc-private-us-west-2c
bedrock-poc-vpc-private-us-west-2a
bedrock-poc-vpc-default
bedrock-poc-vpc-private-us-west-2b



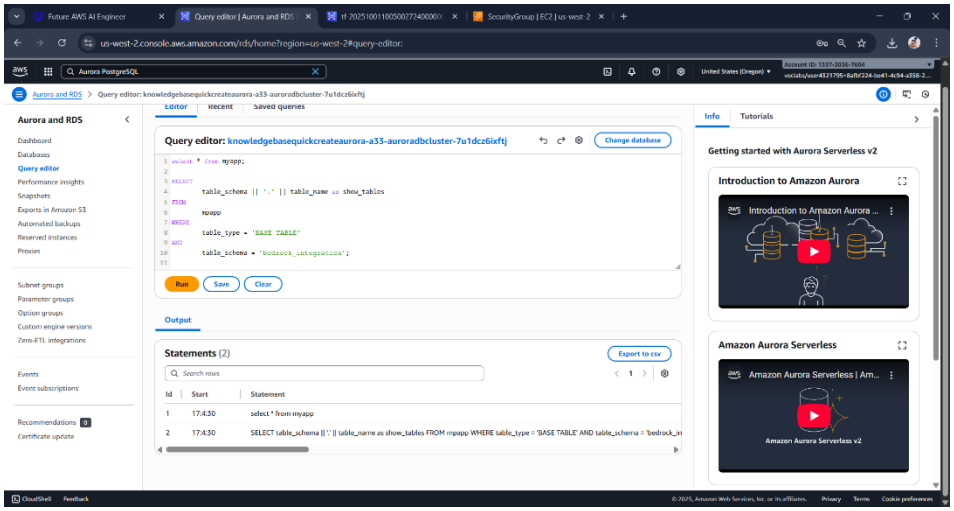
Proper configuration and security settings

Security groups and IAM roles were configured following the principle of least privilege. The Aurora database only allows inbound traffic on port 5432. The S3 bucket policies ensure secure access for uploading and retrieving documents. Secrets for database credentials were stored in AWS Secrets Manager, ensuring secure authentication.



Database properly configured for vector storage.

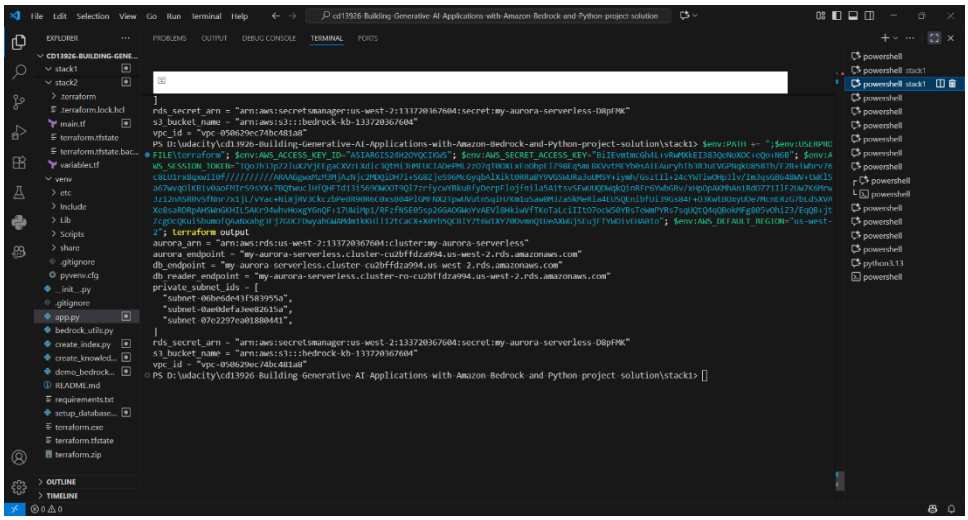
Using the script scripts/aurora_sql.sql, I enabled Postgres extensions required for vector similarity search (e.g., pgvector). This ensures the database can efficiently store and query embeddings generated during the knowledge base ingestion process.



Knowledge Base Deployment and Data Sync

Knowledge base successfully deployed

Using Terraform (Stack 2), I deployed the Amazon Bedrock Knowledge Base and connected it with the Aurora vector store. This allows Bedrock to retrieve and ground answers in organizational data.



```
PS D:\udacity\cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution
\stack2> terraform apply -auto-approve
Outputs:

s3_bucket_name = "bedrock-kb-133720367604"
stack2_status = "Stack 2 completed - Documents uploaded to S3"
uploaded_documents = [
  "documents/bulldozer-bd850-spec-sheet.pdf",
  "documents/dump-truck-dt1000-spec-sheet.pdf",
  "documents/excavator-x950-spec-sheet.pdf",
  "documents/forklift-fl250-spec-sheet.pdf",
  "documents/mobile-crane-mc750-spec-sheet.pdf",
]
```

Criteria: Data from S3 bucket correctly synchronized

Explanation:
I uploaded the project documents (e.g., machine_files.pdf) into the **S3 bucket** using the script scripts/upload_to_s3.py. Then, I performed a **data sync operation** from the AWS Console. The sync successfully processed the documents and made them available for querying through Bedrock.

✅ *Screenshot: Successful data sync in AWS Console*

---

3. Python Integration with Bedrock

Criteria: Python function implemented to query the knowledge base

Explanation:
Implemented query_knowledge_base() in **bedrock_utils.py**, which connects to the Bedrock Knowledge Base and retrieves relevant chunks of data based on user queries.

Criteria: Successful invocation of the model in bedrock_utils.py

Explanation:
Created the generate_response() function to send retrieved chunks from the knowledge base to Bedrock's LLM. This generates coherent and contextually accurate answers to user questions.

Criteria: Correct implementation of valid_prompt function

Explanation:
Implemented valid_prompt() to categorize and validate prompts. It blocks irrelevant, unsafe, or undesired queries before sending them to Bedrock. This ensures reliability and ethical use of the system.

Criteria: Clear explanation of how temperature and top_p affect responses

Explanation:

- **Temperature** controls randomness:
    - Low values (0.2–0.3) → factual, consistent answers.
    - High values (0.7–0.9) → creative, varied answers.
- **Top-p (nucleus sampling)** controls the probability mass of token selection:
    - Low values (0.3–0.5) → focused, deterministic output.
    - High values (0.9–1.0) → diverse, exploratory output.

For this project, I used **low temperature (0.2) and low top-p (0.5)** for factual document-based answers.

Final output: