

Project Summary

1. Infrastructure Deployment – Stack 1 (VPC, Aurora Serverless, S3)

I started the project by setting up the basic infrastructure outlined in Stack 1. This included the VPC, subnets, routing settings, a Serverless Aurora PostgreSQL cluster, and the S3 bucket for storing PDF documents for the Bedrock Knowledge Base.

I went into the stack1 directory and initialized Terraform. After checking the variables, which included the AWS region, CIDR ranges and DB settings, I launched the deployment with terraform apply. Once the plan seemed correct, I confirmed it and waited for Terraform to finish the setup.

```
pooja@LAPTOP-EFO33NMK MINGW64 /d/cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution (main)
$ cd stack1
pooja@LAPTOP-EFO33NMK MINGW64 /d/cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution/stack1 (main)
$ terraform init
Initializing the backend...
Initializing modules...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Reusing previous version of hashicorp/random from the dependency lock file
- Using previously-installed hashicorp/aws v6.18.0
- Using previously-installed hashicorp/random v3.7.2

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

```
MINGW64/d/cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution/stack1
module.aurora_serverless.aws_rds_cluster_instance.aurora_instance: Still creating
g... [07m40s elapsed]
module.aurora_serverless.aws_rds_cluster_instance.aurora_instance: Still creating
g... [07m50s elapsed]
module.aurora_serverless.aws_rds_cluster_instance.aurora_instance: Still creating
g... [08m00s elapsed]
module.aurora_serverless.aws_rds_cluster_instance.aurora_instance: Creation complete after 8m1s [id=cf-20251120173448108200000006]

Warning: Deprecated attribute
  on .terraform\modules\vpc\flow-logs.tf line 28, in locals:
  28:   "arn:${data.aws_partition.current[0].partition}:logs:${data.aws_region.current[0].name}:${data.aws_caller_identity.current[0].account_id}:log-group:${log_group.name};""
    The attribute "name" is deprecated. Refer to the provider documentation for
    details.

Apply complete! Resources: 35 added, 0 changed, 0 destroyed.

Outputs:
aurora_arn = "arn:aws:rds:us-east-1:562849332692:cluster:my-aurora-serverless"
aurora_endpoint = "my-aurora-serverless.cluster-cvuiytxommh.us-east-1.rds.amazonaws.com"
db_endpoint = "my-aurora-serverless.cluster-cvuiytxommh.us-east-1.rds.amazonaws.com"
db_reader_endpoint = "my-aurora-serverless.cluster-ro-cvuiytxommh.us-east-1.rds.amazonaws.com"
private_subnet_ids = [
  "subnet-05e1cf95c01cb6dd2",
  "subnet-0021c4978ac1d4523",
  "subnet-05d5f469db60537d5",
]
public_subnet_ids = [
  "subnet-0326ccdfa65272d93",
  "subnet-0ec300ad1f8d26eb",
  "subnet-035346fe0e1285560",
]
rds_secret_arn = "arn:aws:secretsmanager:us-east-1:562849332692:secret:my-aurora-serverless-wdEmCw"
s3_bucket_name = "arn:aws:s3:::bedrock-kb-562849332692"
vpc_id = "vpc-0b48eb1efef8441f0b"

pooja@LAPTOP-EFO33NMK MINGW64 /d/cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution/stack1 (main)
```

After the deployment finished, I gathered the outputs, especially the Aurora cluster endpoint. I used this endpoint in the Query Editor while preparing the database.

2. Preparing the Aurora Database (SQL Execution in Query Editor)

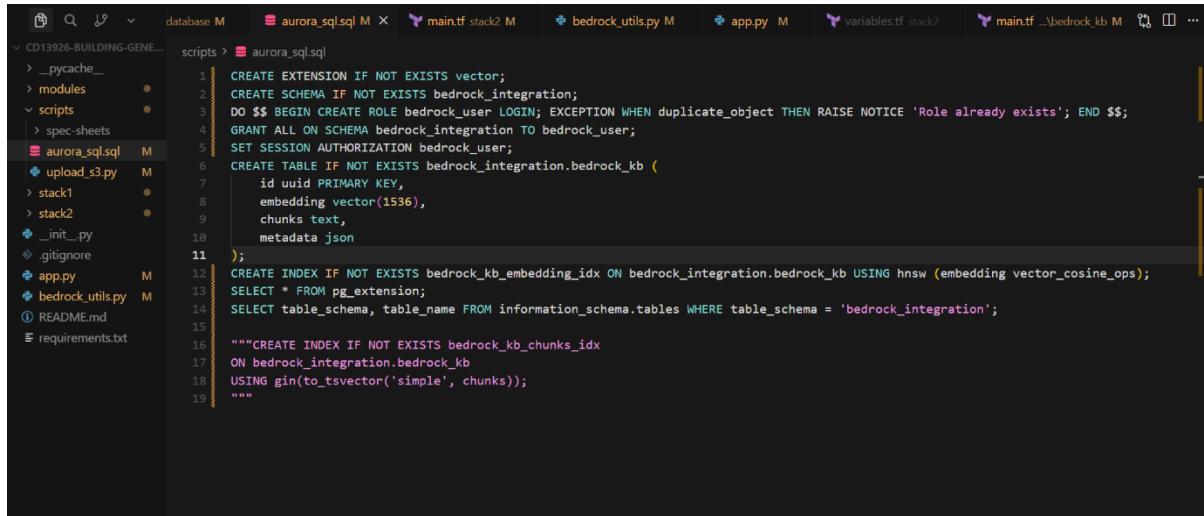
Once Aurora was provisioned, I opened the Query Editor inside the Amazon RDS console and connected using the credentials stored in AWS Secrets Manager.

Inside the Query Editor, I executed the SQL script from scripts/aurora_sql.sql, which created:

- the required bedrock_integration schema
- the vector extension

- user roles
- the bedrock_kb table
- a vector index
- a text search GIN index

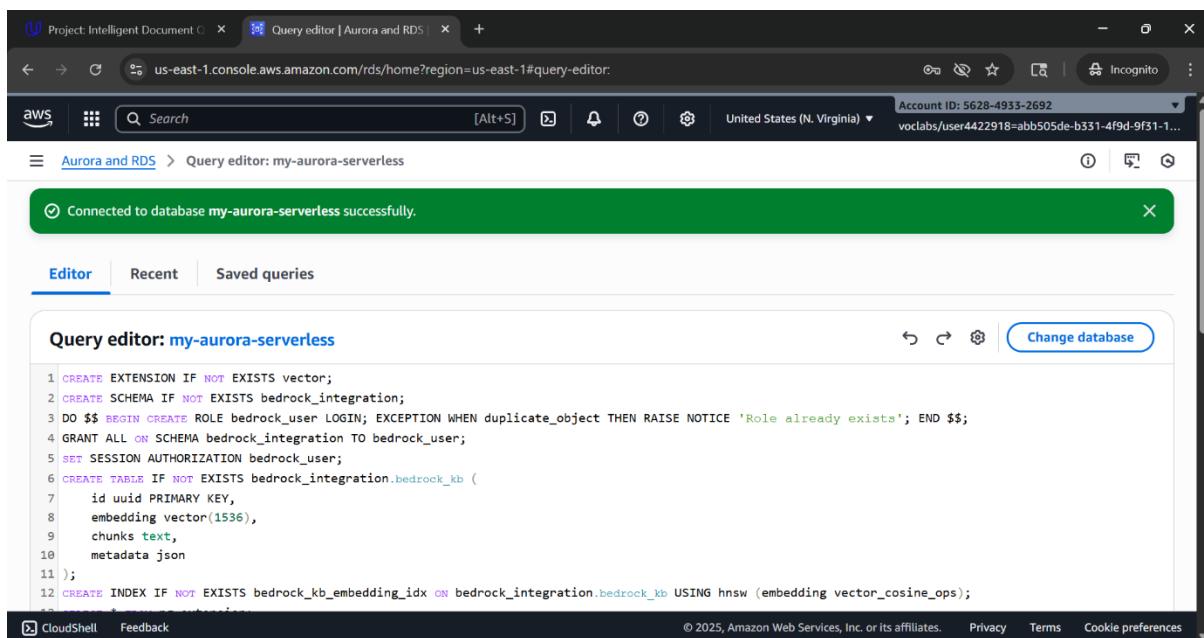
This structure prepares the database to store embeddings, metadata, and chunked content for retrieval.



```

CREATE EXTENSION IF NOT EXISTS vector;
CREATE SCHEMA IF NOT EXISTS bedrock_integration;
DO $$ BEGIN CREATE ROLE bedrock_user LOGIN; EXCEPTION WHEN duplicate_object THEN RAISE NOTICE 'Role already exists'; END $$;
GRANT ALL ON SCHEMA bedrock_integration TO bedrock_user;
SET SESSION AUTHORIZATION bedrock_user;
CREATE TABLE IF NOT EXISTS bedrock_integration.bedrock_kb (
    id uuid PRIMARY KEY,
    embedding vector(1536),
    chunks text,
    metadata json
);
CREATE INDEX IF NOT EXISTS bedrock_kb_embedding_idx ON bedrock_integration.bedrock_kb USING hnsw (embedding vector_cosine_ops);
SELECT * FROM pg_extension;
SELECT table_schema, table_name FROM information_schema.tables WHERE table_schema = 'bedrock_integration';
"""
CREATE INDEX IF NOT EXISTS bedrock_kb_chunks_idx
ON bedrock_integration.bedrock_kb
USING gin(to_tsvector('simple', chunks));
"""

```



Connected to database my-aurora-serverless successfully.

Query editor: my-aurora-serverless

```

1 CREATE EXTENSION IF NOT EXISTS vector;
2 CREATE SCHEMA IF NOT EXISTS bedrock_integration;
3 DO $$ BEGIN CREATE ROLE bedrock_user LOGIN; EXCEPTION WHEN duplicate_object THEN RAISE NOTICE 'Role already exists'; END $$;
4 GRANT ALL ON SCHEMA bedrock_integration TO bedrock_user;
5 SET SESSION AUTHORIZATION bedrock_user;
6 CREATE TABLE IF NOT EXISTS bedrock_integration.bedrock_kb (
7     id uuid PRIMARY KEY,
8     embedding vector(1536),
9     chunks text,
10    metadata json
11 );
12 CREATE INDEX IF NOT EXISTS bedrock_kb_embedding_idx ON bedrock_integration.bedrock_kb USING hnsw (embedding vector_cosine_ops);

```

ID	Start	Statement	Status
1	12:53:40	CREATE EXTENSION IF NOT EXISTS vector	success
2	12:53:40	CREATE SCHEMA IF NOT EXISTS bedrock_integration	success
3	12:53:40	DO \$\$ BEGIN CREATE ROLE bedrock_user LOGIN; EXCEPTION WHEN duplicate_object THEN RAISE NOTICE 'Role already exists'; END \$\$	success
4	12:53:40	GRANT ALL ON SCHEMA bedrock_integration TO bedrock_user	success
5	12:53:40	SET SESSION AUTHORIZATION bedrock_user	success
6	12:53:40	CREATE TABLE IF NOT EXISTS bedrock_integration.bedrock_kb (id uuid PRIMARY KEY, embedding vector(1536), chunks text, metadata json)	success
7	12:53:41	CREATE INDEX IF NOT EXISTS bedrock_kb_embedding_idx ON bedrock_integration.bedrock_kb USING hnsw (embedding vector_cosine_ops)	success
8	12:53:41	SELECT * FROM pg_extension	success
9	12:53:41	SELECT table_schema, table_name FROM information_schema.tables WHERE table_schema = 'bedrock_integration'	success

Rows returned (2)	Export to csv
Search rows	< 1 > ⚙
oid extname extowner extnamespace extrelocatable extversion extconfig extcondition	
14501 plpgsql 10 11 false 1.0 NULL NULL	
16459 vector 10 2200 true 0.8.0 NULL NULL	

This confirmed that the schema, table, indexes, and extensions were all properly created.

3. Infrastructure Deployment – Stack 2 (Bedrock Knowledge Base)

After finishing the database preparation, I went to Stack 2, which manages the Bedrock Knowledge Base and its related resources. I ran Terraform again, updated the variables using the output values from Stack 1, including the Aurora cluster endpoint, S3 bucket, and DB secrets, and then applied the configuration. The apply step created the Bedrock Knowledge Base, set up the data source configuration, and established IAM roles and integration for knowledge retrieval.

```

MINGW64:/d/cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution/stack2
$ cd ..
MINGW64:/d/cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution/stack1 (main)
$ terraform init
Initializing the backend...
Initializing modules...
- bedrock_kb in ..\modules\bedrock_kb
Initializing provider plugins
- Finding latest version of hashicorp/aws...
  - Found latest version of hashicorp/time...
- Installing hashicorp/aws v6.21.0...
- Installed hashicorp/aws v6.21.0 (signed by HashiCorp)
- Installing hashicorp/time v0.13.1...
- Installed hashicorp/time v0.13.1 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider selections it made above. Include this file in your version control repository so that Terraform can guarantee to make the same selections by default when you run "terraform init" in the future.

Terraform has been successfully initialized!

you may now begin working with Terraform. Try running "terraform plan" to see any changes that are required for your infrastructure. All Terraform commands should now work.

If you ever set or change modules or backend configuration for Terraform, rerun this command to reinitialize your working directory. If you forget, other commands will detect it and remind you to do so if necessary.

MINGW64:/d/cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution/stack2 (main)
$ 

```

```

MINGW64:/d/cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution/stack2
+ rds_configuration {
  + credentials_secret_arn = "arn:aws:secretsmanager:us-east-1:562849332692:secret:my-aurora-serverless-wdmcw"
  + database_name           = "myapp"
  + resource_arn             = "arn:aws:rds:us-east-1:562849332692:cluster:my-aurora-serverless"
  + table_name               = "bedrock_integration.bedrock_kb"

  + field_mapping {
    + metadata_field      = "metadata"
    + primary_key_field   = "id"
    + text_field          = "chunks"
    + vector_field        = "embedding"
  }
}
}

Plan: 2 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ bedrock_knowledge_base_arn = (known after apply)
+ bedrock_knowledge_base_id = (known after apply)

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

module.bedrock_kb.aws_bedrockagent_knowledge_base.main: Creating...
module.bedrock_kb.aws_bedrockagent_knowledge_base.main: Creation complete after
2s [id:SGZDAFDJAW]
module.bedrock_kb.aws_bedrockagent_data_source.s3_bedrock_bucket: Creating...
module.bedrock_kb.aws_bedrockagent_data_source.s3_bedrock_bucket: Creation complete after
1s [id:TABGMUJSU,SGZDAFDJAW]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:
bedrock_knowledge_base_arn = "arn:aws:bedrock:us-east-1:562849332692:knowledge-base/SGZDAFDJAW"
bedrock_knowledge_base_id = "SGZDAFDJAW"

MINGW64:/d/cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution/stack2 (main)
$ 

```

Once deployed, the knowledge base existed but contained no documents yet, that part came next.

4. Uploading PDF Files to S3

I put the machine PDF files in the spec-sheets/ folder. Then, I used the Python script scripts/upload_to_s3.py to upload everything in batches to the S3 bucket created by Stack 1. The script went through the folder and kept the structure. It uploaded all PDFs to the right S3 prefix. After the upload, the Bedrock Knowledge Base data source was able to sync them.

```

bedrock_utils.py M app.py M variables.tf stack2 main.tf ...\\bedrock.kb M variables.tf ...\\bedrock.kb upload_s3.py M ...
scripts > upload_s3.py
    5 def upload_files_to_s3(folder_path, bucket_name, prefix=""):
    6     for root, dirs, files in os.walk(folder_path):
    7         for filename in files:
    8             local_path = os.path.join(root, filename)
    9             relative_path = os.path.relpath(local_path, folder_path)
   10             s3_key = os.path.join(prefix, relative_path).replace("\\", "/")
   11
   12             try:
   13                 # Upload the file to S3
   14                 s3_client.upload_file(local_path, bucket_name, s3_key)
   15                 print(f"Successfully uploaded {relative_path} to {bucket_name}/{s3_key}")
   16             except ClientError as e:
   17                 print(f"Error uploading {relative_path}: {e}")
   18
   19     if __name__ == "__main__":
   20         # Folder path
   21         folder_path = "scripts/spec-sheets"
   22
   23         # S3 bucket name
   24         bucket_name = "bedrock-kb-562849332692" # Replace with your actual bucket name
   25
   26         # S3 prefix (optional)
   27         prefix = "spec-sheets"
   28
   29         upload_files_to_s3(folder_path, bucket_name, prefix)

```

```

MINGW64:/d/cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution
$ python scripts/upload_s3.py
Requirement already satisfied: s3transfer<1.5 in c:\users\pooja\appdata\roaming\python\python311\site-packages (from boto3>=1.17.0>=2.1>botocore<1.42.0,>=1.41.0->boto3> requirements.txt (line 1)) (1.17.0)
Requirement already satisfied: s3transfer<1.4.0 in c:\users\pooja\appdata\roaming\python\python311\site-packages (from botocore<1.42.0,>=1.41.0->boto3> requirements.txt (line 1)) (1.17.0)
Requirement already satisfied: botocore<1.42.0,>=1.41.0 in c:\users\pooja\appdata\roaming\python\python311\site-packages (from boto3>=1.17.0>=2.1>botocore<1.42.0,>=1.41.0->boto3> requirements.txt (line 1)) (1.42.0)
Requirement already satisfied: jmespath<1.0.1,>=0.26.0 in c:\users\pooja\appdata\roaming\python\python311\site-packages (from botocore<1.42.0,>=1.41.0->boto3> requirements.txt (line 1)) (0.26.0)
Requirement already satisfied: s3transfer<1.4.0 in c:\users\pooja\appdata\roaming\python\python311\site-packages (from botocore<1.42.0,>=1.41.0->boto3> requirements.txt (line 1)) (1.17.0)
Requirement already satisfied: botocore<1.42.0,>=1.41.0 in c:\users\pooja\appdata\roaming\python\python311\site-packages (from boto3>=1.17.0>=2.1>botocore<1.42.0,>=1.41.0->boto3> requirements.txt (line 1)) (1.42.0)
Using cached jmespath-1.0.1-py3-none-any.whl (20 kB)
Requirement already satisfied: s3transfer-0.14.0-py3-none-any.whl (85 kB)
Requirement already satisfied: botocore-1.41.0-py3-none-any.whl (14.2 MB)
Successfully installed boto3-1.41.0 botocore-1.41.0 jmespath-1.0.1 s3transfer-0.14.0
[notice] A new release of pip is available: 25.0.1 => 25.1.0
[notice] To update, run: python.exe -m pip install --upgrade pip
pooja@LAPTOP-EFO33NMK: MINGW64 /d/cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution (main)
$ python scripts/upload_s3.py
Error: The folder 'spec-sheets' does not exist.

pooja@LAPTOP-EFO33NMK: MINGW64 /d/cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution (main)
$ python scripts/upload_s3.py
Error: The folder 'spec-sheets' does not exist.

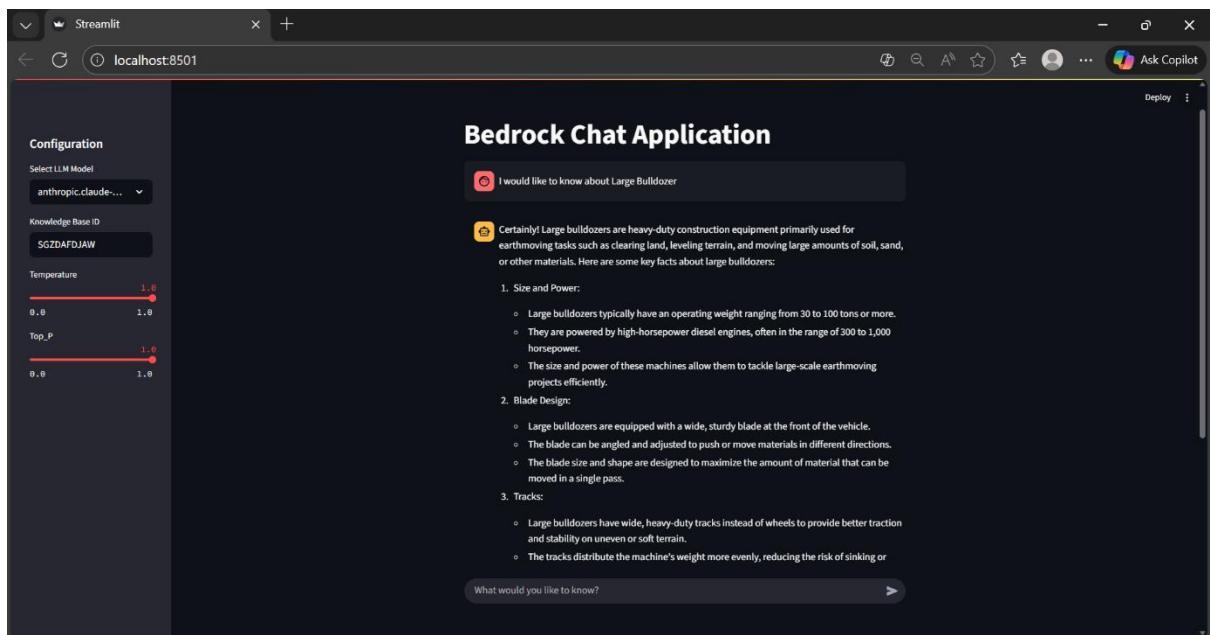
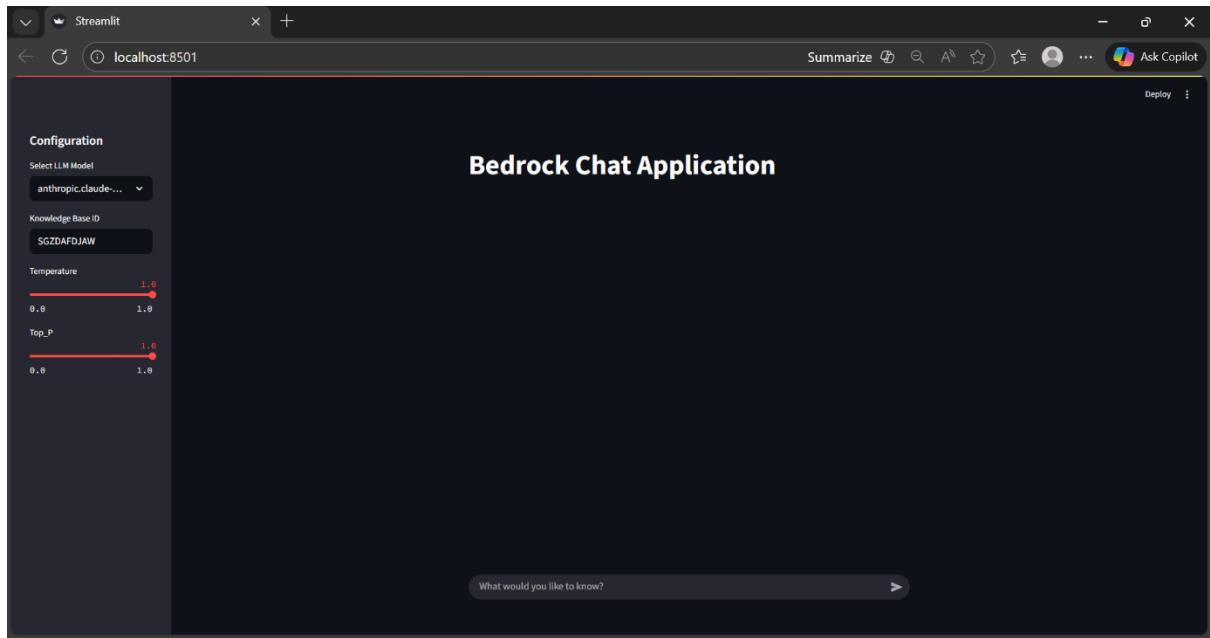
pooja@LAPTOP-EFO33NMK: MINGW64 /d/cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution (main)
$ python scripts/upload_s3.py
Error: The folder 'spec-sheets' does not exist.

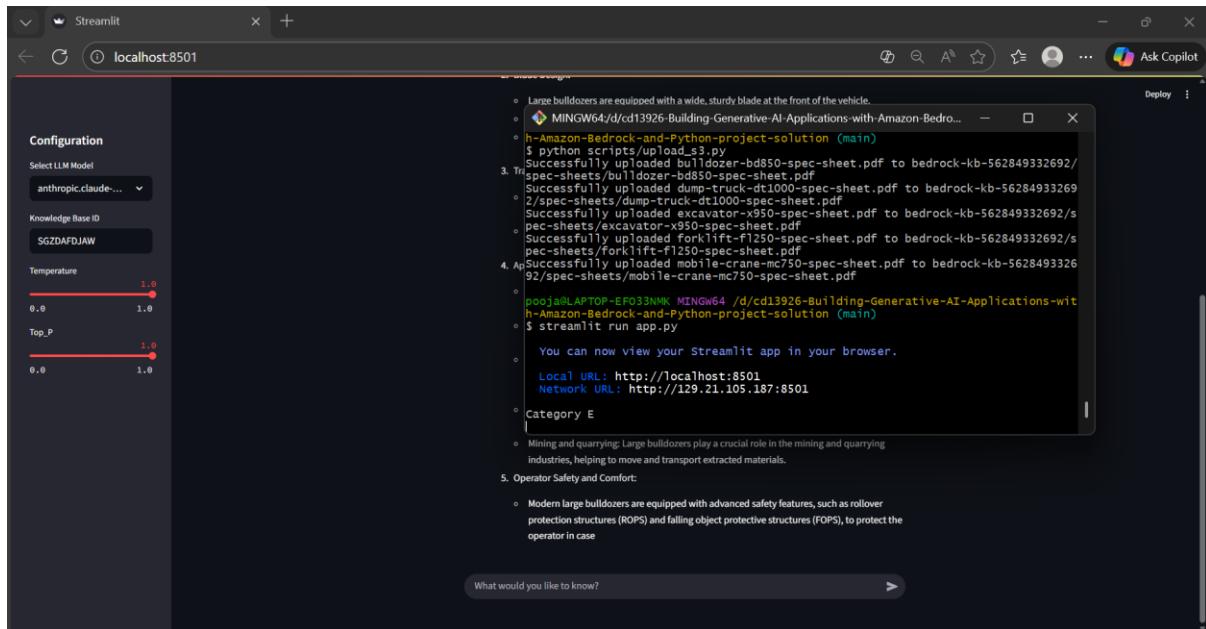
pooja@LAPTOP-EFO33NMK: MINGW64 /d/cd13926-Building-Generative-AI-Applications-with-Amazon-Bedrock-and-Python-project-solution (main)
$ python scripts/upload_s3.py
Successfully uploaded bulldozer-bd50-spec-sheet.pdf to bedrock-kb-562849332692/spec-sheets/bulldozer-bd50-spec-sheet.pdf
Successfully uploaded dump-truck-dt1000-spec-sheet.pdf to bedrock-kb-562849332692/2/spec-sheets/dump-truck-dt1000-spec-sheet.pdf
Successfully uploaded excavator-ex50-spec-sheet.pdf to bedrock-kb-562849332692/spec-sheets/excavator-ex50-spec-sheet.pdf
Successfully uploaded forklift-f1250-spec-sheet.pdf to bedrock-kb-562849332692/spec-sheets/forklift-f1250-spec-sheet.pdf
Successfully uploaded mobile-crane-mc750-spec-sheet.pdf to bedrock-kb-562849332692/spec-sheets/mobile-crane-mc750-spec-sheet.pdf

```

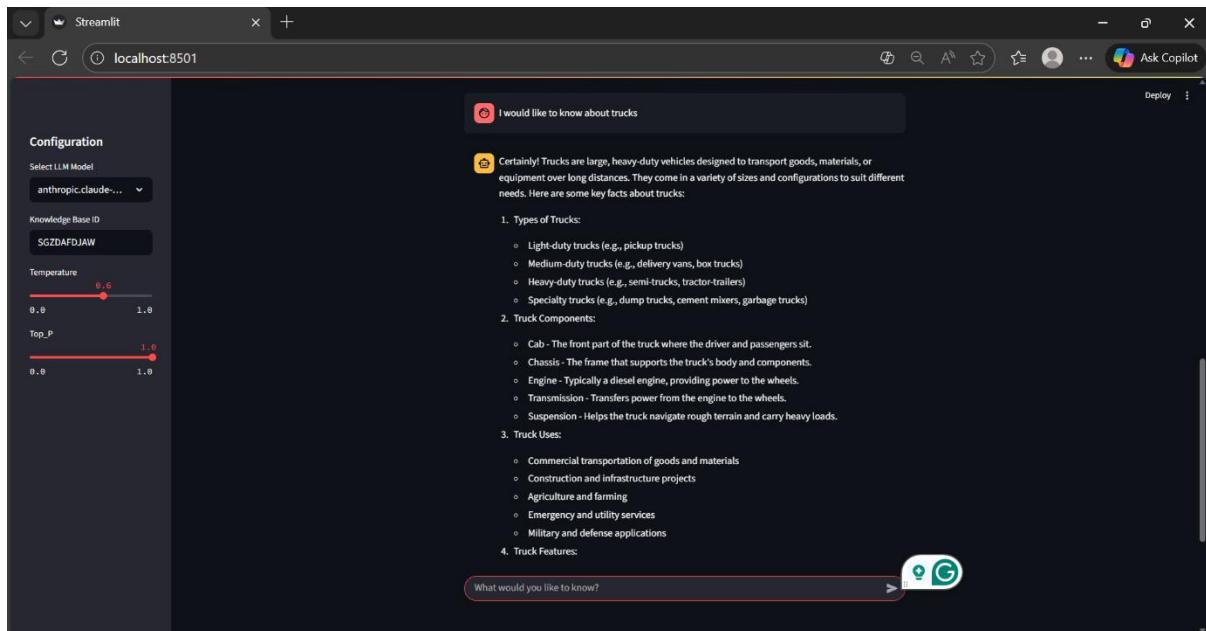
6. Completing the Application Logic

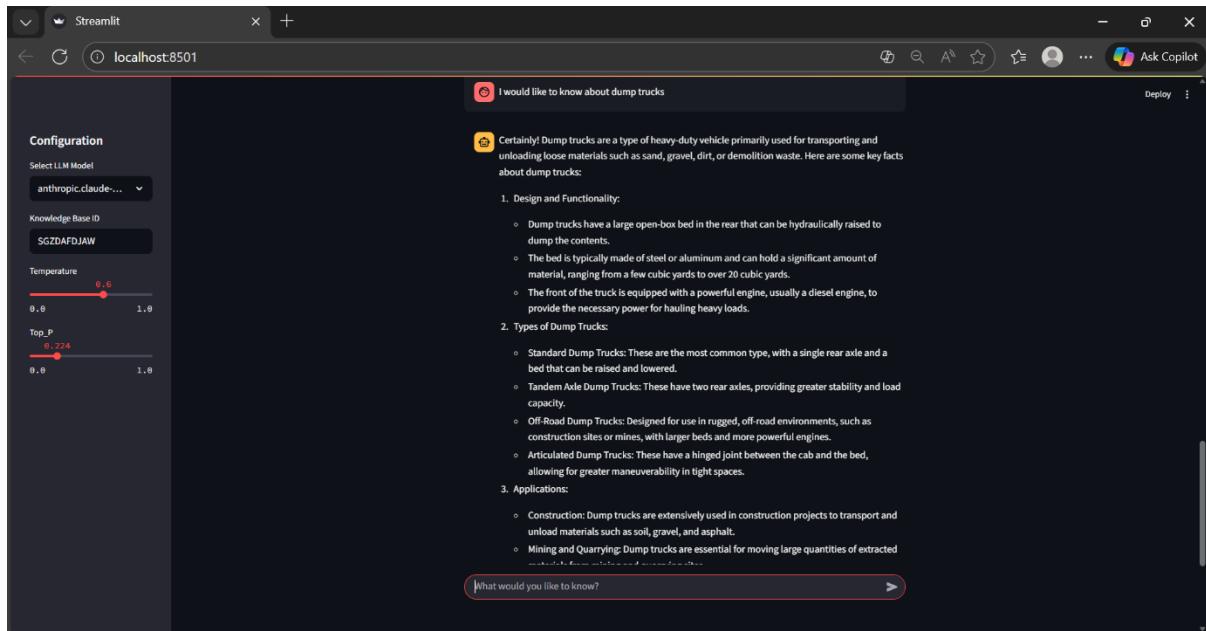
Since the project included a working sample application, I simply ran the provided app.py file without needing to implement or modify any of the core logic. The application was already configured to call the Knowledge Base, retrieve document chunks from Aurora, and generate grounded model responses using Bedrock. Running the app verified the full retrieval-augmented workflow end-to-end.





To complete the project requirements, I also generated different responses using various temperature and top_p settings inside the application. These settings helped me observe how the model output changes under different sampling conditions. Lower temperature and top_p settings produced highly consistent, factual responses, while higher values generated more expressive and varied outputs. I captured screenshots of these tests as part of the required submission.





After completing all steps, I collected the required screenshots, organized them into the Screenshots directory, verified my Terraform stacks, SQL setup, ingestion job, and application run were all correct, and packaged the entire project for submission exactly as required in the rubric.