**Chapter 8: Generative AI on AWS**

**8.1 Introduction to Generative AI**

Generative AI refers to artificial intelligence systems that can create new content, including text, images, code, and more. In the context of AWS, generative AI capabilities are primarily provided through services like Amazon Bedrock and can be enhanced with other AWS services to create powerful, context-aware applications.

**8.2 Amazon Bedrock**

Amazon Bedrock is a fully managed service that provides access to high-performing foundation models (FMs) from leading AI companies through a single API. It allows developers to easily experiment with and integrate state-of-the-art AI models into their applications without managing the underlying infrastructure.

Key features of Amazon Bedrock:

* Access to various foundation models (e.g., Claude from Anthropic, Jurassic-2 from AI21 Labs)
* Customization capabilities through fine-tuning
* Serverless experience with pay-as-you-go pricing
* Enterprise-grade security and privacy

**8.3 Prerequisites for Using Amazon Bedrock**

Before using Amazon Bedrock, ensure you have the following:

1. AWS Account with appropriate permissions
2. Bedrock access requested and granted in your AWS account
3. IAM permissions set up for Bedrock access
4. Latest version of AWS SDK for Python (Boto3) installed
5. Working in a supported AWS region
6. Access requested for specific foundation models you intend to use

**8.4 Retrieval-Augmented Generation (RAG) Application**

Let's build a comprehensive RAG application that combines Amazon Bedrock with other AWS services. This application will use the AWS Overview whitepaper as its knowledge base.

**8.4.1 Setting Up the Environment**

python

Copy

import boto3

import requests

import json

import time

import os

*# Configure boto3 clients*

s3 = boto3.client('s3')

kendra = boto3.client('kendra')

bedrock = boto3.client('bedrock-runtime')

comprehend = boto3.client('comprehend')

translate = boto3.client('translate')

*# Replace these with your actual values*

bucket\_name = 'your-bucket-name'

kendra\_role\_arn = 'arn:aws:iam::your-account-id:role/KendraRoleWithRequiredPermissions'

s3\_access\_role\_arn = 'arn:aws:iam::your-account-id:role/KendraRoleWithS3Access'

**8.4.2 Downloading and Indexing the AWS Overview PDF**

python

Copy

def download\_and\_upload\_pdf():

pdf\_url = "https://aws.amazon.com/whitepapers/latest/aws-overview/aws-overview.pdf"

pdf\_name = "aws-overview.pdf"

response = requests.get(pdf\_url)

with open(pdf\_name, 'wb') as f:

f.write(response.content)

s3.upload\_file(pdf\_name, bucket\_name, f"documents/{pdf\_name}")

print(f"AWS Overview PDF uploaded to s3://{bucket\_name}/documents/{pdf\_name}")

os.remove(pdf\_name)

def create\_kendra\_index():

response = kendra.create\_index(

Name='AWSOverviewIndex',

Edition='DEVELOPER\_EDITION',

RoleArn=kendra\_role\_arn

)

index\_id = response['Id']

while True:

response = kendra.describe\_index(Id=index\_id)

if response['Status'] == 'ACTIVE':

break

time.sleep(60)

print(f"Kendra Index created with ID: {index\_id}")

return index\_id

def create\_kendra\_datasource(index\_id):

response = kendra.create\_data\_source(

IndexId=index\_id,

Name='AWSOverviewDataSource',

Type='S3',

DataSourceConfiguration={

'S3Configuration': {

'BucketName': bucket\_name,

'InclusionPrefixes': ['documents/aws-overview.pdf']

}

},

RoleArn=s3\_access\_role\_arn

)

data\_source\_id = response['Id']

kendra.start\_data\_source\_sync\_job(Id=data\_source\_id, IndexId=index\_id)

while True:

sync\_status = kendra.describe\_data\_source(Id=data\_source\_id, IndexId=index\_id)

if sync\_status['Status'] == 'ACTIVE':

break

time.sleep(60)

print("AWS Overview PDF indexed in Kendra")

return data\_source\_id

*# Execute the setup*

download\_and\_upload\_pdf()

index\_id = create\_kendra\_index()

data\_source\_id = create\_kendra\_datasource(index\_id)

**8.4.3 Implementing the RAG Application**

python

Copy

def query\_kendra(query, index\_id):

response = kendra.query(

IndexId=index\_id,

QueryText=query

)

return response['ResultItems']

def generate\_bedrock\_response(query, context):

prompt = f"""Human: You are an AI assistant with knowledge about AWS services. Use the following information from the AWS Overview whitepaper to answer the human's question. If the information doesn't contain the answer, say you don't know but provide general information about AWS if relevant.

Information:

{context}

CopyRetry

Claude does not have the ability to run the code it generates yet.

S

come on you only explaining RAG application, earlier you have given good intro of bedrock, sagemaker jump start etc. can you write the chanpter of GenAI on AWS as tutorial

Edit

I apologize for the confusion. You're absolutely right. Let me rewrite the entire Chapter 8 on Generative AI on AWS, incorporating all the elements we've discussed, including Bedrock, SageMaker JumpStart, and the RAG application. Here's a comprehensive tutorial-style chapter:

**Chapter 8: Generative AI on AWS**

**8.1 Introduction to Generative AI**

Generative AI refers to artificial intelligence systems that can create new content, including text, images, code, and more. AWS offers several services and tools to leverage generative AI capabilities, enabling developers to build sophisticated AI-powered applications.

**8.2 Amazon Bedrock**

Amazon Bedrock is a fully managed service that provides access to high-performing foundation models (FMs) from leading AI companies through a single API.

**Key Features:**

* Access to various foundation models (e.g., Claude from Anthropic, Jurassic-2 from AI21 Labs)
* Customization capabilities through fine-tuning
* Serverless experience with pay-as-you-go pricing
* Enterprise-grade security and privacy

**Example: Text Generation with Bedrock**

python

Copy

import boto3

import json

bedrock = boto3.client(service\_name='bedrock-runtime')

prompt = "Write a short story about a robot learning to paint:"

body = json.dumps({

"prompt": prompt,

"max\_tokens\_to\_sample": 300,

"temperature": 0.7,

"top\_p": 0.8,

})

modelId = 'anthropic.claude-v2'

accept = 'application/json'

contentType = 'application/json'

response = bedrock.invoke\_model(body=body, modelId=modelId, accept=accept, contentType=contentType)

response\_body = json.loads(response.get('body').read())

print(response\_body.get('completion'))

**8.3 Amazon SageMaker JumpStart**

SageMaker JumpStart provides pre-trained, open-source models for a wide variety of problem types to help you get started with machine learning.

**Key Features:**

* One-click deployment of pre-trained models
* Fine-tuning capabilities
* Integration with SageMaker's managed infrastructure

**Example: Deploying a JumpStart Model**

python

Copy

import sagemaker

from sagemaker import get\_execution\_role

from sagemaker.jumpstart.model import JumpStartModel

role = get\_execution\_role()

region = sagemaker.Session().boto\_region\_name

model\_id = "huggingface-text2text-flan-t5-xl"

model = JumpStartModel(model\_id=model\_id)

predictor = model.deploy()

result = predictor.predict("Translate to French: Hello, how are you?")

print(result)

predictor.delete\_endpoint()

**8.4 Retrieval-Augmented Generation (RAG) Application**

RAG combines the power of large language models with a retrieval system to generate more accurate and contextually relevant responses. Let's build a RAG application using Amazon Bedrock and Amazon Kendra.

**8.4.1 Setting Up the Environment**

python

Copy

import boto3

import requests

import json

import time

import os

s3 = boto3.client('s3')

kendra = boto3.client('kendra')

bedrock = boto3.client('bedrock-runtime')

bucket\_name = 'your-bucket-name'

kendra\_role\_arn = 'arn:aws:iam::your-account-id:role/KendraRoleWithRequiredPermissions'

s3\_access\_role\_arn = 'arn:aws:iam::your-account-id:role/KendraRoleWithS3Access'

**8.4.2 Indexing Documents with Kendra**

python

Copy

def download\_and\_upload\_pdf():

pdf\_url = "https://aws.amazon.com/whitepapers/latest/aws-overview/aws-overview.pdf"

pdf\_name = "aws-overview.pdf"

response = requests.get(pdf\_url)

with open(pdf\_name, 'wb') as f:

f.write(response.content)

s3.upload\_file(pdf\_name, bucket\_name, f"documents/{pdf\_name}")

print(f"AWS Overview PDF uploaded to s3://{bucket\_name}/documents/{pdf\_name}")

os.remove(pdf\_name)

def create\_kendra\_index():

response = kendra.create\_index(

Name='AWSOverviewIndex',

Edition='DEVELOPER\_EDITION',

RoleArn=kendra\_role\_arn

)

index\_id = response['Id']

while True:

response = kendra.describe\_index(Id=index\_id)

if response['Status'] == 'ACTIVE':

break

time.sleep(60)

print(f"Kendra Index created with ID: {index\_id}")

return index\_id

def create\_kendra\_datasource(index\_id):

response = kendra.create\_data\_source(

IndexId=index\_id,

Name='AWSOverviewDataSource',

Type='S3',

DataSourceConfiguration={

'S3Configuration': {

'BucketName': bucket\_name,

'InclusionPrefixes': ['documents/aws-overview.pdf']

}

},

RoleArn=s3\_access\_role\_arn

)

data\_source\_id = response['Id']

kendra.start\_data\_source\_sync\_job(Id=data\_source\_id, IndexId=index\_id)

while True:

sync\_status = kendra.describe\_data\_source(Id=data\_source\_id, IndexId=index\_id)

if sync\_status['Status'] == 'ACTIVE':

break

time.sleep(60)

print("AWS Overview PDF indexed in Kendra")

return data\_source\_id

*# Execute the setup*

download\_and\_upload\_pdf()

index\_id = create\_kendra\_index()

data\_source\_id = create\_kendra\_datasource(index\_id)

**8.4.3 Implementing the RAG Application**

python

Copy

def query\_kendra(query, index\_id):

response = kendra.query(

IndexId=index\_id,

QueryText=query

)

return response['ResultItems']

def generate\_bedrock\_response(query, context):

prompt = f"""Human: You are an AI assistant with knowledge about AWS services. Use the following information from the AWS Overview whitepaper to answer the human's question. If the information doesn't contain the answer, say you don't know but provide general information about AWS if relevant.

Information:

{context}

CopyRetry

Claude does not have the ability to run the code it generates yet.

S

is it possible to give a sample list of the models available through jumpstart and bedrock, What is the differnce between jump start and bedrock. when to use which?

Edit

I apologize for not including that important information earlier. You're right that it would be helpful to provide a sample list of models available through SageMaker JumpStart and Amazon Bedrock, as well as explain the differences between them and when to use each. Let me add that information to our chapter.

**8.5 Comparing SageMaker JumpStart and Amazon Bedrock**

**SageMaker JumpStart**

SageMaker JumpStart provides a wide variety of pre-trained models that can be easily deployed or fine-tuned.

Sample list of models available through JumpStart:

1. Text Processing:
   * BERT (various versions)
   * RoBERTa
   * DistilBERT
   * XLM
2. Image Processing:
   * ResNet (various versions)
   * MobileNet
   * YOLOv5
3. Tabular Data:
   * XGBoost
   * LightGBM
   * CatBoost
4. Generative AI:
   * GPT-2
   * T5

**Amazon Bedrock**

Bedrock provides access to foundation models from leading AI companies.

Sample list of models available through Bedrock:

1. Anthropic:
   * Claude (various versions)
2. AI21 Labs:
   * Jurassic-2 (various versions)
3. Amazon:
   * Titan (various versions)
4. Stability AI:
   * Stable Diffusion (for image generation)

**Key Differences**

1. Model Source:
   * JumpStart: Mostly open-source models
   * Bedrock: Proprietary models from AI companies
2. Customization:
   * JumpStart: Allows fine-tuning of models
   * Bedrock: Offers customization through prompt engineering and fine-tuning (for some models)
3. Deployment:
   * JumpStart: Models are deployed to SageMaker endpoints
   * Bedrock: Serverless access to models through API calls
4. Use Cases:
   * JumpStart: Wide range of ML tasks (classification, regression, NLP, computer vision)
   * Bedrock: Primarily focused on large language models and generative AI
5. Integration:
   * JumpStart: Tightly integrated with SageMaker ecosystem
   * Bedrock: Can be easily integrated into any application

**When to Use Each**

Use SageMaker JumpStart when:

* You need a wide variety of ML models for different tasks
* You want to fine-tune models on your specific dataset
* You're already using the SageMaker ecosystem
* You need more control over the deployment infrastructure

Use Amazon Bedrock when:

* You need state-of-the-art large language models
* You want a serverless solution with minimal infrastructure management
* Your use case requires the latest generative AI capabilities
* You need enterprise-grade security and compliance features

In many cases, you might use both services in your AI/ML pipeline. For example, you could use JumpStart for data preprocessing or feature extraction, and then use Bedrock for generating human-like text based on those features.

**8.6 Example: Combining JumpStart and Bedrock in a Pipeline**

Here's a simple example of how you might combine JumpStart and Bedrock in a pipeline:

python

Copy

import sagemaker

from sagemaker.jumpstart.model import JumpStartModel

import boto3

import json

*# Set up SageMaker session*

sagemaker\_session = sagemaker.Session()

role = sagemaker.get\_execution\_role()

*# Deploy a JumpStart model for sentiment analysis*

sentiment\_model = JumpStartModel(model\_id="huggingface-sentiment-distilbert-base-uncased-finetuned-sst-2-english")

sentiment\_predictor = sentiment\_model.deploy()

*# Set up Bedrock client*

bedrock = boto3.client(service\_name='bedrock-runtime')

def analyze\_and\_respond(user\_input):

*# Analyze sentiment using JumpStart model*

sentiment = sentiment\_predictor.predict({"inputs": user\_input})

sentiment\_label = "positive" if sentiment[0]['label'] == "LABEL\_1" else "negative"

*# Generate response using Bedrock*

prompt = f"""Human: The user's message "{user\_input}" has been analyzed as having a {sentiment\_label} sentiment.

Please generate a response that acknowledges this sentiment and provides a helpful reply.

Assistant: Certainly! Here's a response that acknowledges the sentiment and provides a helpful reply:

{sentiment\_label.capitalize()} sentiment detected. I understand that you're feeling {sentiment\_label} about this.

Human: Great, now please provide the actual response to the user based on their input and the detected sentiment.

Assistant: Certainly! Here's a response tailored to the user's input and detected sentiment:

"""

body = json.dumps({

"prompt": prompt,

"max\_tokens\_to\_sample": 200,

"temperature": 0.7,

"top\_p": 0.8,

})

modelId = 'anthropic.claude-v2'

response = bedrock.invoke\_model(body=body, modelId=modelId)

response\_body = json.loads(response.get('body').read())

return response\_body.get('completion').strip()

*# Example usage*

user\_message = "I'm really excited about learning AWS services!"

response = analyze\_and\_respond(user\_message)

print(f"User: {user\_message}")

print(f"AI: {response}")

*# Clean up*

sentiment\_predictor.delete\_endpoint()

This example demonstrates how you can use a JumpStart model for sentiment analysis and then use that information to generate a more contextually appropriate response with a Bedrock model.

By combining these services, you can create sophisticated AI applications that leverage the strengths of both SageMaker JumpStart and Amazon Bedrock.