Practical – 1

Accessing AWS Bedrock inside Vscode

Introduction to AWS

Amazon Web Services (AWS) is a cloud computing platform that provides a broad set of services, including storage, computing, machine learning, analytics, security, and more. AWS enables businesses and developers to build, deploy, and scale applications without managing physical infrastructure.

AWS operates on a pay-as-you-go model, meaning users only pay for the resources they consume. Some key benefits of AWS include:

- **Scalability: Easily scale applications up or down based on demand.**
- **Security: AWS provides robust security with compliance and encryption features.**
- **Flexibility: A wide range of services allows businesses to customize their cloud environment.**
- **Global Reach: AWS data centers are distributed worldwide, ensuring high availability and low latency.**

Common AWS Services:

- **Amazon S3: Cloud storage service.**
- **EC2: Virtual servers in the cloud.**
- **AWS Lambda: Serverless computing for running functions without managing infrastructure.**
- **Amazon Bedrock: A service for accessing and integrating AI foundation models.**

What is an IAM User?

AWS Identity and Access Management (IAM) is a security service that helps manage access to AWS resources. Instead of using the root AWS account (which has full permissions), AWS recommends creating IAM users with specific roles and permissions.

Key Features of IAM:

- **Users & Groups: Individuals or teams with assigned permissions.**
- **Roles: Temporary access for AWS services and external users.**

- **Policies: JSON-based documents defining permissions (e.g., allowing access to Amazon Bedrock).**
- **Multi-Factor Authentication (MFA): Extra security by requiring a second authentication factor.**

Steps to Create an IAM User for AWS Bedrock:

1. **Sign in to the AWS Management Console.**
2. **Go to IAM → Click on Users → Add user.**
3. **Set permissions using AWS-managed policies (e.g., AmazonBedrockFullAccess).**
4. **Generate Access Keys (used for programmatic access).**
5. **Store the keys securely and configure them in AWS CLI.**

What is AWS Bedrock?

AWS Bedrock is a fully managed AI service that allows developers to use pre-trained foundation models (FMs) from different AI providers without needing to manage infrastructure.

Key Features of AWS Bedrock:

- **Access to multiple foundation models from Amazon, Meta (Llama), Anthropic (Claude), Cohere, and others.**
- **No infrastructure management, making it easier to integrate AI into applications.**
- **Customizability, allowing fine-tuning of models using proprietary data.**
- **Seamless integration with other AWS services like S3, Lambda, and SageMaker.**

Bedrock is used for applications like chatbots, content generation, text summarization, and AI-powered search.

Models in AWS Bedrock

AWS Bedrock provides access to various foundation models from leading AI companies. Each model is optimized for different tasks.

Popular Models Available in AWS Bedrock:

Model Provider	Model Name	Best Use Case
Amazon	Titan	Search, embeddings, and text generation
Anthropic	Claude	Chatbots, conversational AI
Meta	Llama 3	Natural language processing, reasoning
Cohere	Command R	Retrieval-augmented generation (RAG)

For this implementation, we are using Llama 3, a powerful large language model (LLM) from Meta, which is known for its advanced reasoning and contextual understanding capabilities.

What is Conda?

Conda is an open-source package manager and environment manager that helps developers manage dependencies efficiently. It is commonly used in machine learning (ML) and AI projects.

Why Use Conda?

- **Creates isolated environments to prevent dependency conflicts.**
- **Supports multiple Python versions within different environments.**
- **Works with various package managers, including pip and Conda-specific packages.**

Common Conda Commands:

- **Install Conda (if not installed):**
- **curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh**
- **bash Miniconda3-latest-Linux-x86_64.sh**
- **Create a new environment:**
- **conda create --name aws_bedrock_env python=3.9 -y**
- **Activate the environment:**
- **conda activate aws_bedrock_env**
- **Install packages using Conda:**
- **conda install boto3 awscli -y**

What is requirements.txt?

A requirements.txt file is a text file that lists all dependencies required for a Python project. It helps maintain consistency across environments by allowing developers to install the exact same package versions.

Why Use requirements.txt?

- **Ensures dependency consistency in different environments.**
- **Makes it easier to share projects with others.**
- **Allows quick reinstallation of dependencies when switching environments.**

Creating and Using requirements.txt

1. **Generate a requirements.txt file from an existing environment:**
2. **pip freeze > requirements.txt**
3. **Install dependencies from requirements.txt:**
4. **pip install -r requirements.txt**
5. **Example requirements.txt for AWS Bedrock:**
6. **boto3**
7. **awscli**
8. **langchain**
9. **llama-index**

Step 1: Create a Conda Environment

Command:

conda create -p myenv python=3.10 -y

What This Command Does:

- **conda create**: This command is used to create a new Conda environment. Conda is a package manager and environment manager that helps in creating isolated environments for different projects.
- **-p myenv**: The -p option specifies the **path** where the environment will be created. In this case, the environment will be created inside the myenv folder on your Desktop (e.g., C:\Users\YourUser\Desktop\amazonAWS\myenv). This option allows you to specify a custom directory instead of using the default Conda environments location.

- **python=3.10**: This specifies that **Python version 3.10** should be installed in this environment. You can choose a different version if needed, but Python 3.10 is a stable version that is compatible with most packages.
- **-y**: The -y flag automatically accepts all the prompts that Conda would normally ask you during the environment creation process. Without this flag, you would need to confirm the installation manually.

Why You Need This Step:

This step sets up an isolated environment with Python 3.10, ensuring that dependencies and packages installed in this environment do not interfere with other projects or system-wide installations. It is a best practice to use isolated environments to avoid conflicts between dependencies.

Step 2: Activate the Conda Environment

Command:

conda activate C:\Users\YourUser\Desktop\amazonAWS\myenv

What This Command Does:

- **conda activate**: This command is used to **activate** a Conda environment, making it the active environment in which all subsequent Python commands and package installations will occur.
- **C:\Users\YourUser\Desktop\amazonAWS\myenv**: This is the full **path** to the environment you just created. When you activate the environment, Conda changes the context to this environment, meaning Python and any packages you install or run will be isolated to this specific environment.

Why You Need This Step:

By activating the environment, you switch to using the specific environment you created, ensuring that any Python code or libraries you use will rely on the Python version and packages installed inside myenv. Without activating the environment, the system-wide Python installation would be used instead.

Step 3: Install Dependencies from requirements.txt

Command:

pip install -r requirements.txt

What This Command Does:

- **pip install**: This is the Python package manager used to install libraries and dependencies.
- **-r**: The -r option allows you to install all the dependencies listed in a requirements file (in this case, requirements.txt).
- **requirements.txt**: This file contains a list of all the Python libraries and their specific versions needed for your project. For instance, it may include boto3, awscli, and other libraries required to interact with AWS and perform specific tasks.

Why You Need This Step:

This step ensures that all the necessary libraries and dependencies required for your project are installed. By using requirements.txt, you ensure consistency across different environments or machines. The requirements.txt file typically contains essential libraries for AWS interactions (boto3, awscli, etc.), along with any other libraries your project may depend on.

Step 4: Configure AWS CLI

Command:

aws configure

What This Command Does:

- **aws configure**: This command is used to configure the AWS Command Line Interface (CLI) with your AWS credentials and region. The AWS CLI allows you to interact with AWS services from the terminal or command prompt.
- When you run this command, it will prompt you to enter the following information:
    1. **AWS Access Key ID**: This is a unique identifier for your AWS account's access. You can find it in your AWS Console under **IAM > Users > [Your User] > Security Credentials**.
    2. **AWS Secret Access Key**: This is the secret key associated with your Access Key ID. It is also available in the AWS Console and is used to authenticate your requests.
    3. **Default Region Name**: The AWS region where your resources (such as EC2, S3, Bedrock) are located. Example regions include us-east-1 (N. Virginia), us-west-2 (Oregon), etc.
    4. **Default Output Format**: The format in which AWS CLI should return results, typically json (you can also choose text or table).

Why You Need This Step:

The aws configure command sets up your AWS credentials and region locally on your machine. This ensures that your applications (like Python scripts using **boto3**) can authenticate with AWS services and interact with the resources you create in the AWS Console. Without this configuration, your AWS SDK (like boto3) wouldn't be able to access your AWS account to perform tasks like invoking models in **Bedrock**.

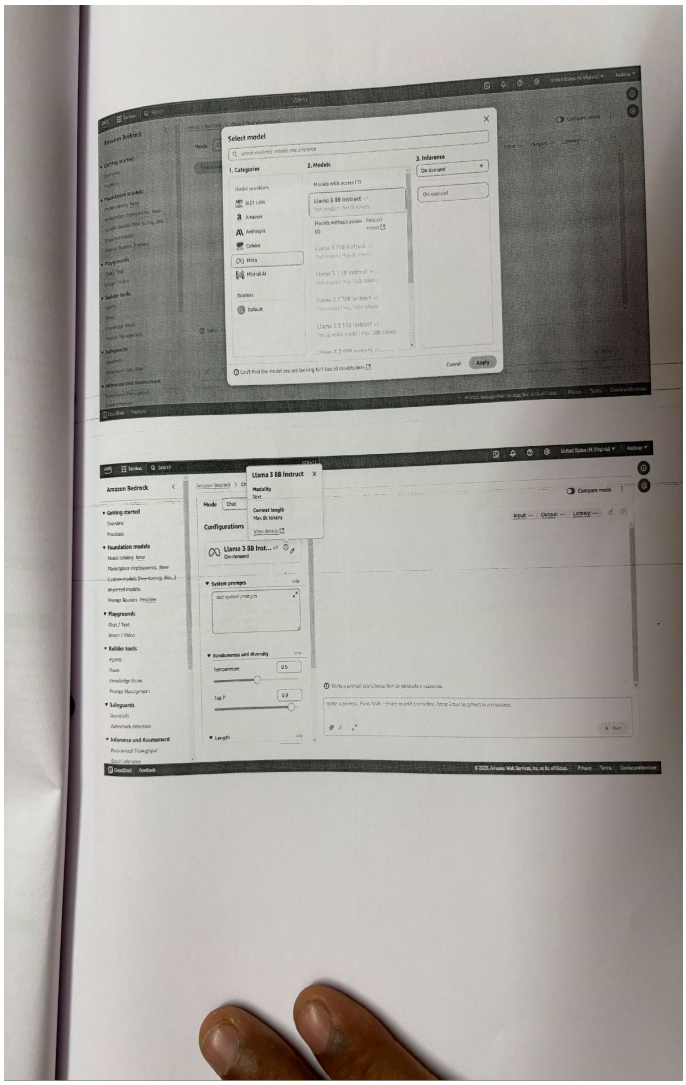
Summary:

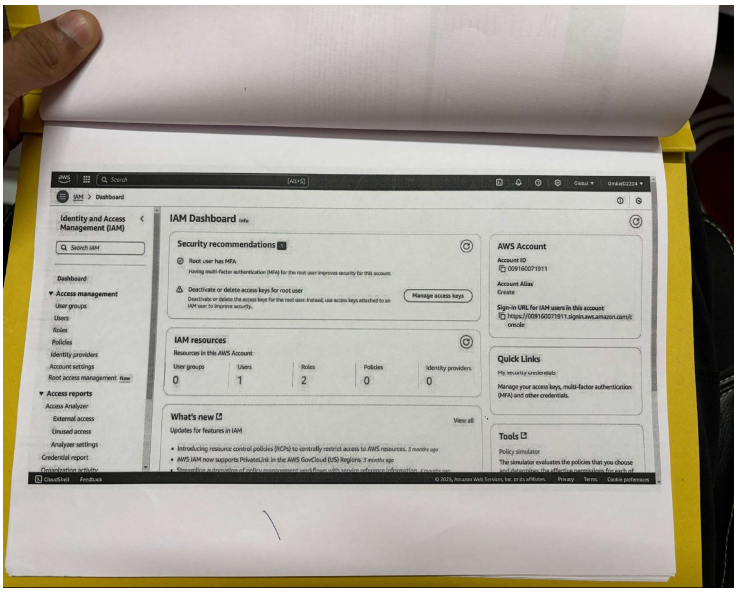
- **Step 1**: **Create a Conda environment** (conda create -p myenv python=3.10 -y): This creates an isolated environment with a specific Python version (3.10), ensuring no conflicts with other projects.
- **Step 2**: **Activate the Conda environment** (conda activate): This sets the newly created environment as the active one for your project.
- **Step 3**: **Install dependencies** (pip install -r requirements.txt): Installs all necessary libraries listed in the requirements.txt file, such as boto3 and awscli, to interact with AWS.
- **Step 4**: **Configure AWS CLI** (aws configure): Configures the AWS CLI with your access credentials and region, enabling authentication and communication with AWS services.

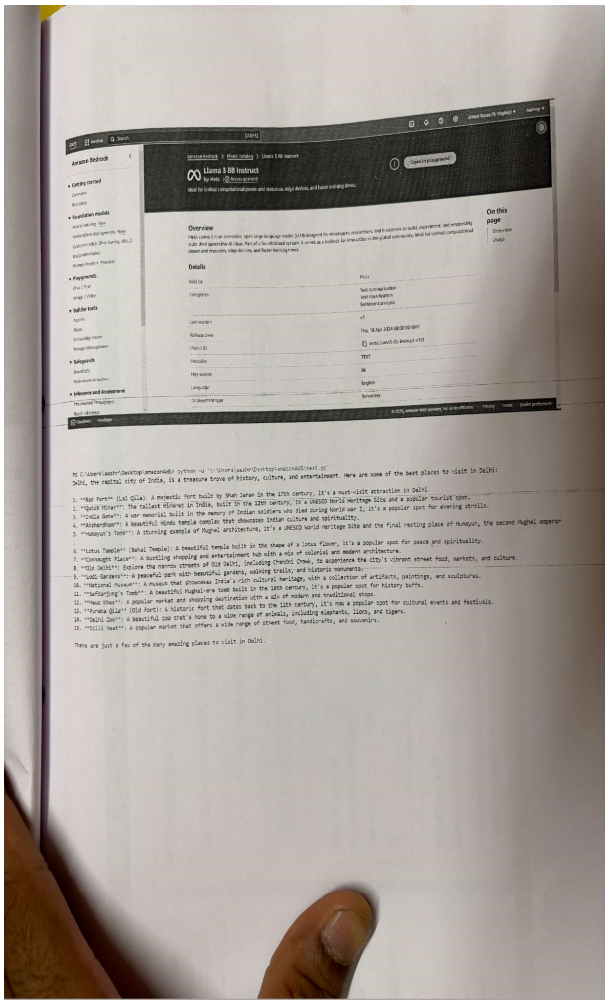
Python Code :-

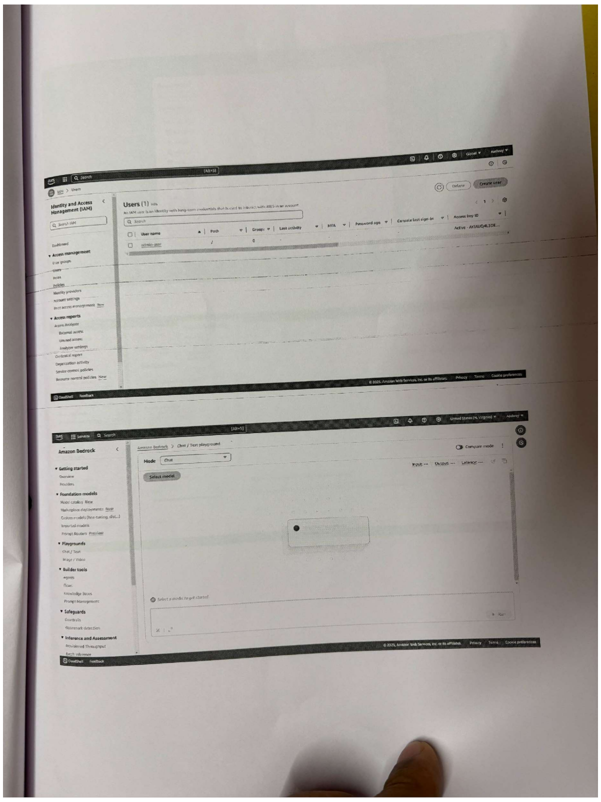
```
import boto3
import json


from botocore.exceptions import ClientError


client = boto3.client("bedrock-runtime", region_name="us-east-1")


model_id = "meta.llama3-8b-instruct-v1:0"


prompt = "Best places to visit in delhi"
```

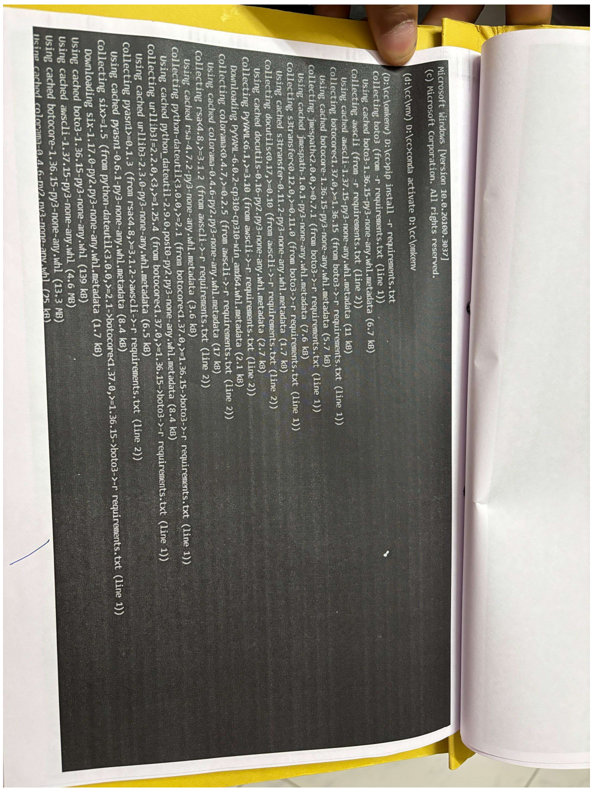
```
formatted_prompt = f"""
<|begin_f_text|><|start_header_id|>user<|end_header_id|>
{prompt}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
"""

native_request = {
   "prompt": formatted_prompt,
   "max_gen_len": 512,
   "temperature": 0.5,
   }
request = json.dumps(native_request)

try:
   response = client.invoke_model(modelId=model_id, body=request)

except(ClientError, Exception) as e:
   print(f"ERROR: Can't invoke '{model_id}'. Reason: {e}")
   exit(1)

model_response = json.loads(response["body"].read())

response_text = model_response["generation"]
print(response_text)
```

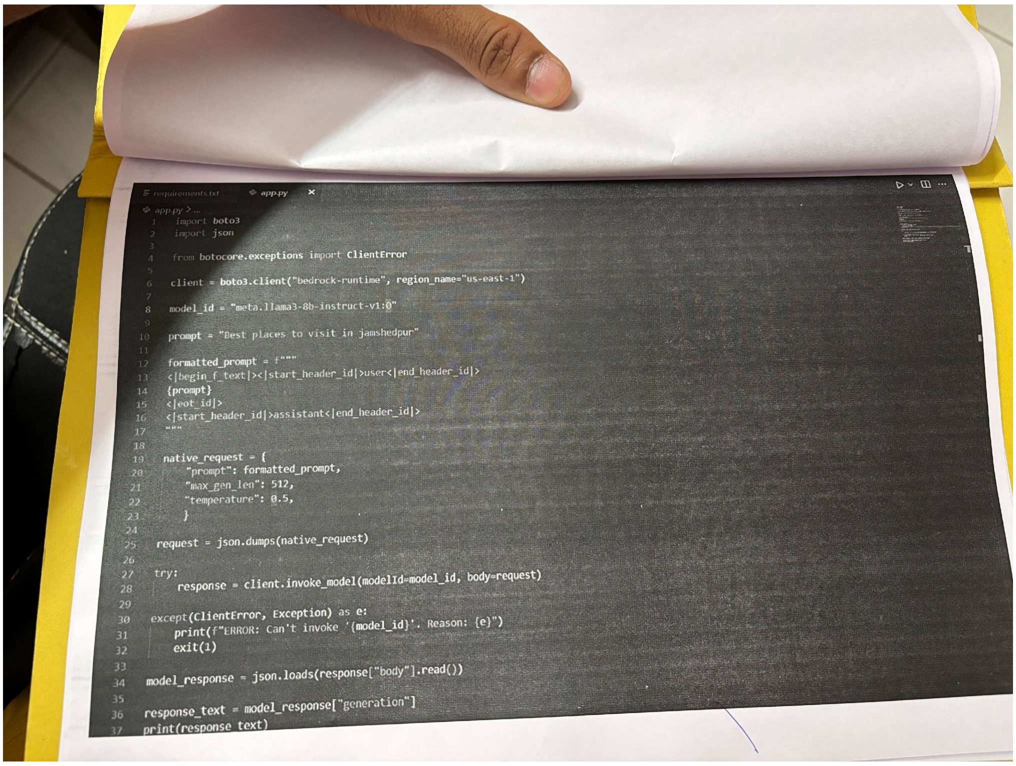

Select model
Amazon Bedrock
Llama 3 8B Instruct
Configurations


IAM Dashboard
Identity and Access Management (IAM)
Security recommendations
AWS Account
IAM resources
Quick Links
What's new
Tools


Amazon Bedrock
Llama 3 8B Instruct
Overview
Details
On this page
Delhi, the capital city of India, is a treasure trove of history, culture, and entertainment. Here are some of the best places to visit in Delhi:
1. **Red Fort** (Lal Qila): A majestic fort built by Shah Jahan in the 17th century, it's a must-visit attraction in Delhi.
2. **Qutub Minar**: The tallest minaret in India, built in the 12th century, it's a UNESCO World Heritage Site and a popular tourist spot.
3. **India Gate**: A war memorial built in the memory of Indian soldiers who died during World War I, it's a popular spot for evening strolls.
4. **Akshardham**: A beautiful Hindu temple complex that showcases Indian culture and spirituality.
5. **Humayun's Tomb**: A stunning example of Mughal architecture, it's a UNESCO World Heritage Site and the final resting place of Humayun, the second Mughal emperor
6. **Lotus Temple** (Bahai Temple): A beautiful temple built in the shape of a lotus flower, it's a popular spot for peace and spirituality.
7. **Connaught Place**: A bustling shopping and entertainment hub with a mix of colonial and modern architecture.
8. **Old Delhi**: Explore the narrow streets of Old Delhi, including Chandni Chowk, to experience the city's vibrant street food, markets, and culture.
9. **Lodi Gardens**: A peaceful park with beautiful gardens, walking trails, and historic monuments.
10. **National Museum**: A museum that showcases India's rich cultural heritage, with a collection of artifacts, paintings, and sculptures.
11. **Safdarjung's Tomb**: A beautiful Mughal-era tomb built in the 18th century, it's a popular spot for history buffs.
12. **Hauz Khas**: A popular market and shopping destination with a mix of modern and traditional shops.
13. **Purana Qila** (Old Fort): A historic fort that dates back to the 11th century, it's now a popular spot for cultural events and festivals.
14. **Delhi Zoo**: A beautiful zoo that's home to a wide range of animals, including elephants, lions, and tigers.
15. **Dilli Haat**: A popular market that offers a wide range of street food, handicrafts, and souvenirs.
There are just a few of the many amazing places to visit in Delhi.




Microsoft Windows [Version 10.0.26100.3037]
(c) Microsoft Corporation. All rights reserved.


requirements.txt
app.py
import boto3
import json
from botocore.exceptions import ClientError
client = boto3.client("bedrock-runtime", region_name="us-east-1")
model_id = "meta.llama3-8b-instruct-v1:0"
prompt = "Best places to visit in jamshedpur"
formatted_prompt = f"""
<|begin_f_text|><|start_header_id|>user<|end_header_id|>
{prompt}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
"""
native_request = {
"prompt": formatted_prompt,
"max_gen_len": 512,
"temperature": 0.5,
}
request = json.dumps(native_request)
try:
response = client.invoke_model(modelId=model_id, body=request)
except(ClientError, Exception) as e:
print(f"ERROR: Can't invoke '{model_id}'. Reason: {e}")
exit(1)
model_response = json.loads(response["body"].read())
response_text = model_response["generation"]
print(response_text)