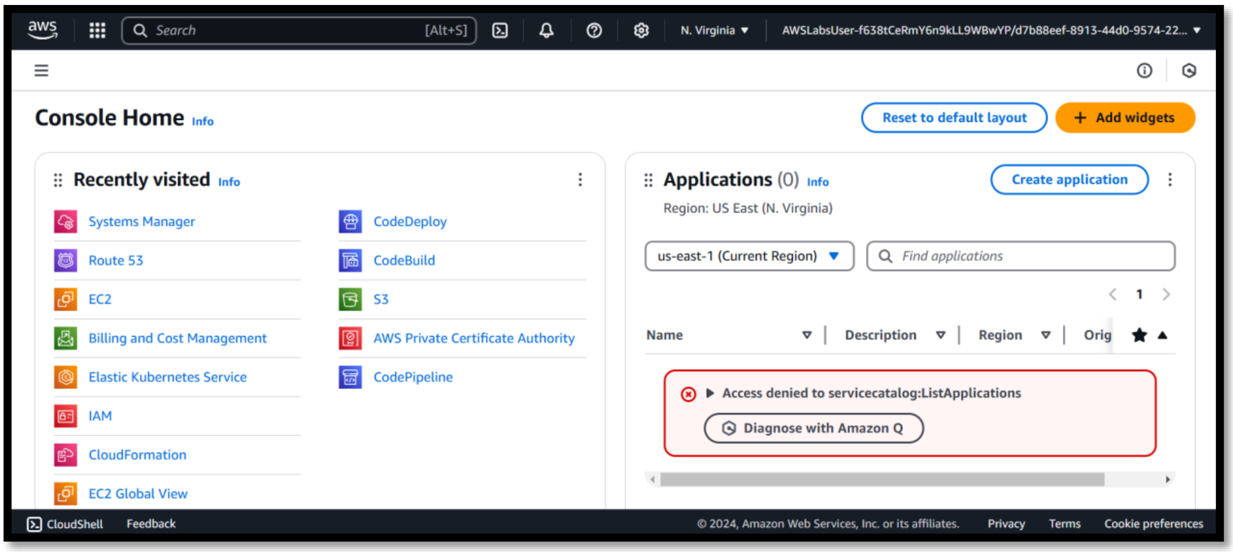
Objective: To use a large language model (LLM) for code generation based on a text prompt.

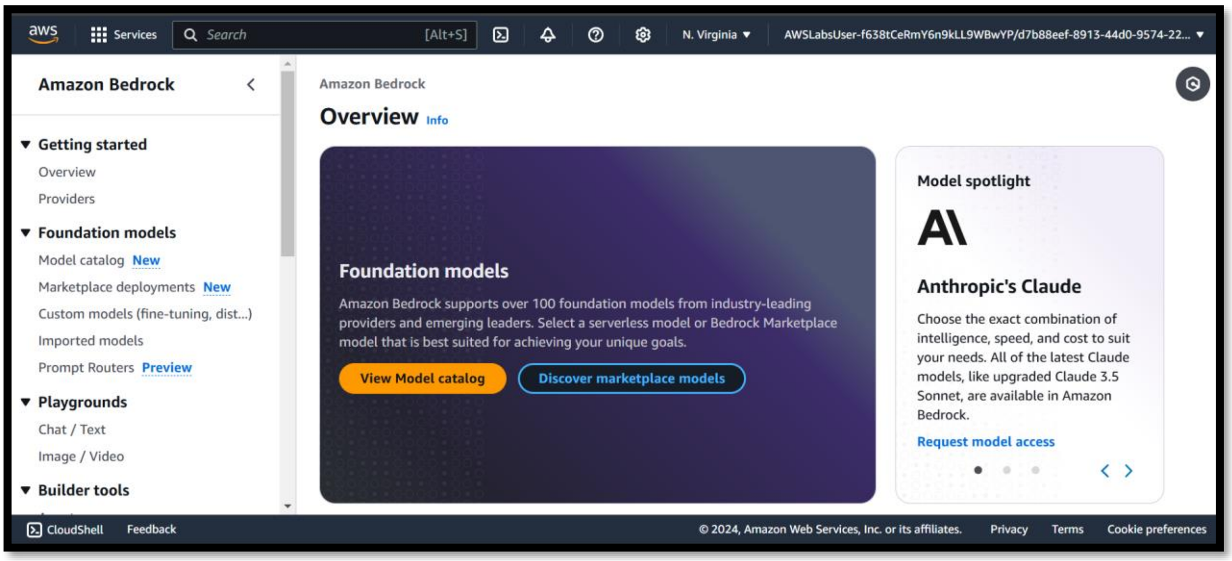
Task 0: Set up the environment

In this task, I registered the base models in the Amazon Bedrock console and launched an Amazon SageMaker Studio application to access my lab resources.



Access denied to servicecatalog:ListApplications

© 2024, Amazon Web Services, Inc. or its affiliates.

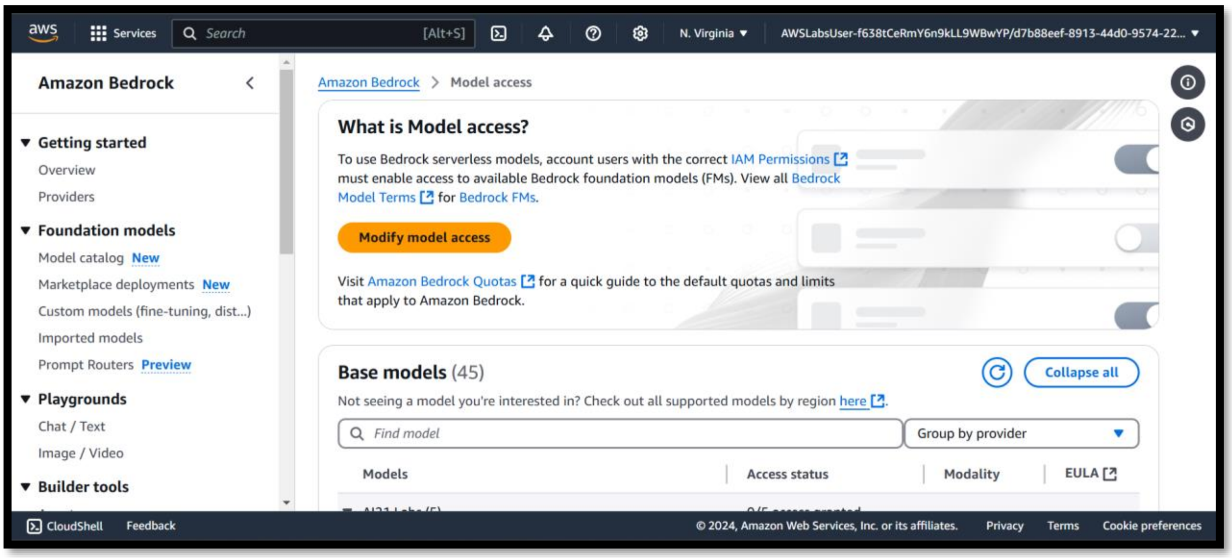


Amazon Bedrock supports over 100 foundation models from industry-leading providers and emerging leaders. Select a serverless model or Bedrock Marketplace model that is best suited for achieving your unique goals.

Choose the exact combination of intelligence, speed, and cost to suit your needs. All of the latest Claude models, like upgraded Claude 3.5 Sonnet, are available in Amazon Bedrock.

© 2024, Amazon Web Services, Inc. or its affiliates.

I reviewed the Access Status for each of the models. If the Access Status for one or more of the models was set to Available to request, I expanded this menu and followed the steps to enable access for them.



What is Model access?

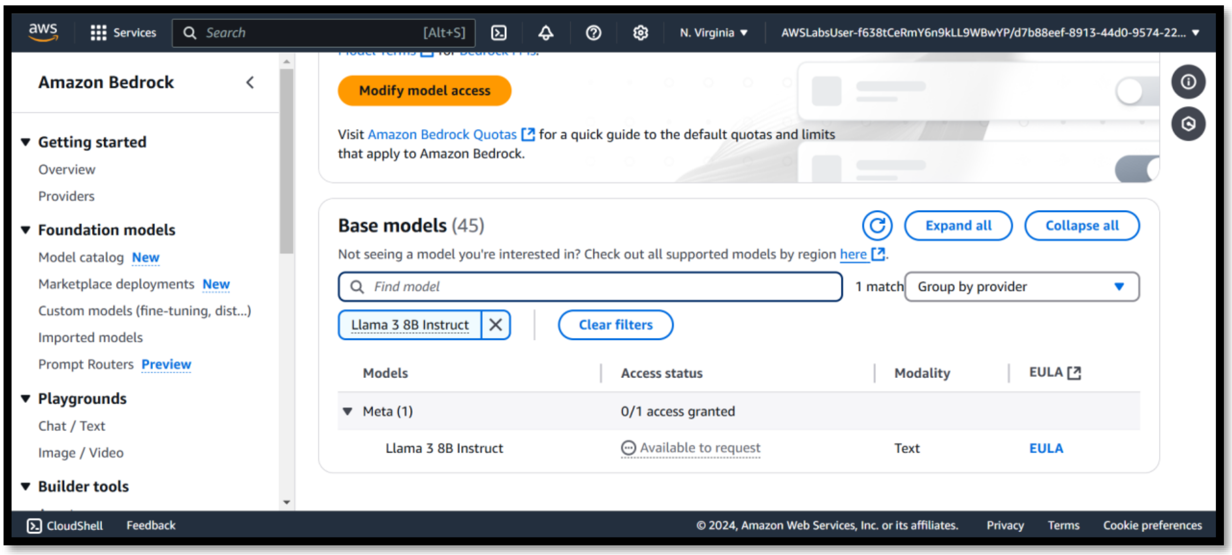
To use Bedrock serverless models, account users with the correct IAM Permissions must enable access to available Bedrock foundation models (FMs). View all Bedrock Model Terms for Bedrock FMs.

Visit Amazon Bedrock Quotas for a quick guide to the default quotas and limits that apply to Amazon Bedrock.

Base models (45)

Not seeing a model you're interested in? Check out all supported models by region here.

Models	Access status	Modality	EULA



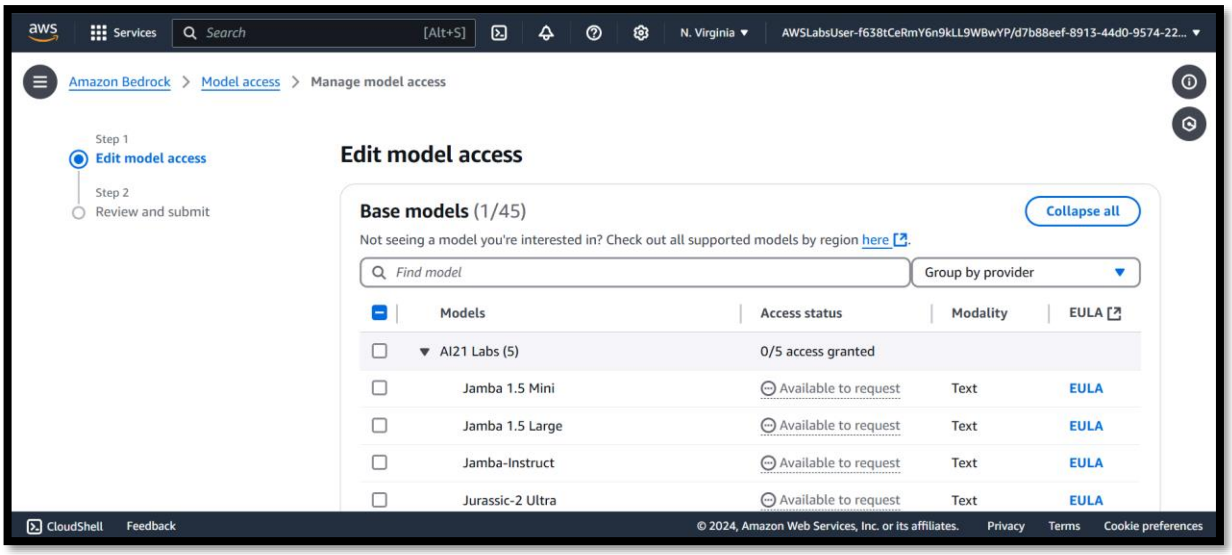
Visit Amazon Bedrock Quotas for a quick guide to the default quotas and limits that apply to Amazon Bedrock.

Base models (45)

Not seeing a model you're interested in? Check out all supported models by region here.

Models	Access status	Modality	EULA
Meta (1)	0/1 access granted		
Llama 3 8B Instruct	Available to request	Text	EULA

I chose Modify model access at the top of the screen.



aws Services Search [Alt+S] N. Virginia AWSLabsUser-f638tCeRmY6n9kLL9WBwYP/d7b88eef-8913-44d0-9574-22...

Amazon Bedrock > Model access > Manage model access

Step 1
Edit model access

Step 2
Review and submit

Edit model access

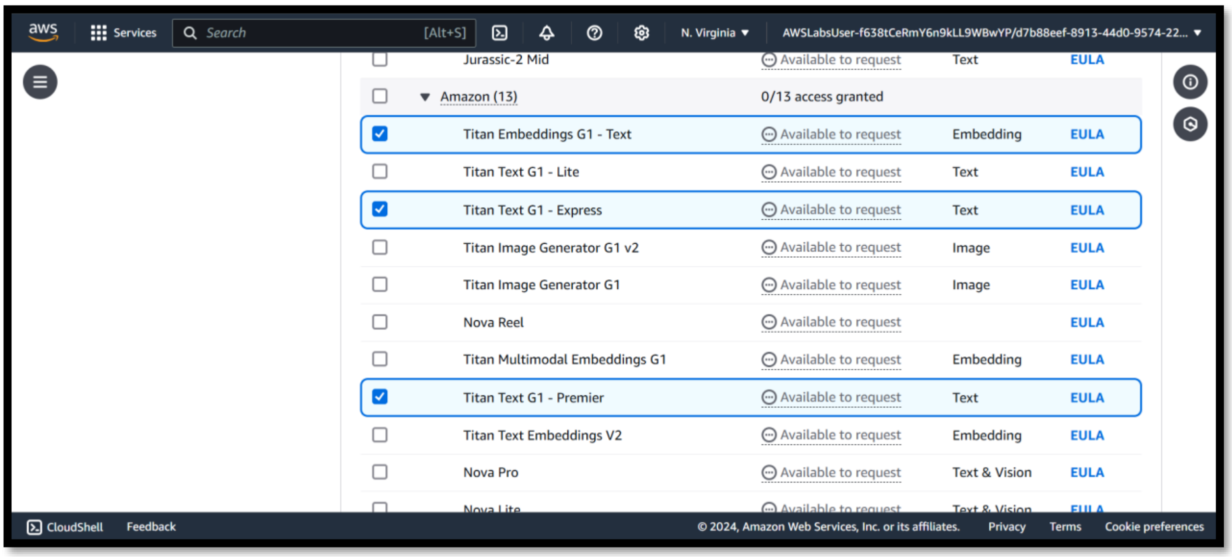
Base models (1/45) Collapse all

Not seeing a model you're interested in? Check out all supported models by region here.

Find model | Group by provider

	Models	Access status	Modality	EULA
	▼ AI21 Labs (5)	0/5 access granted		
	Jamba 1.5 Mini	Available to request	Text	EULA
	Jamba 1.5 Large	Available to request	Text	EULA
	Jamba-Instruct	Available to request	Text	EULA
	Jurassic-2 Ultra	Available to request	Text	EULA

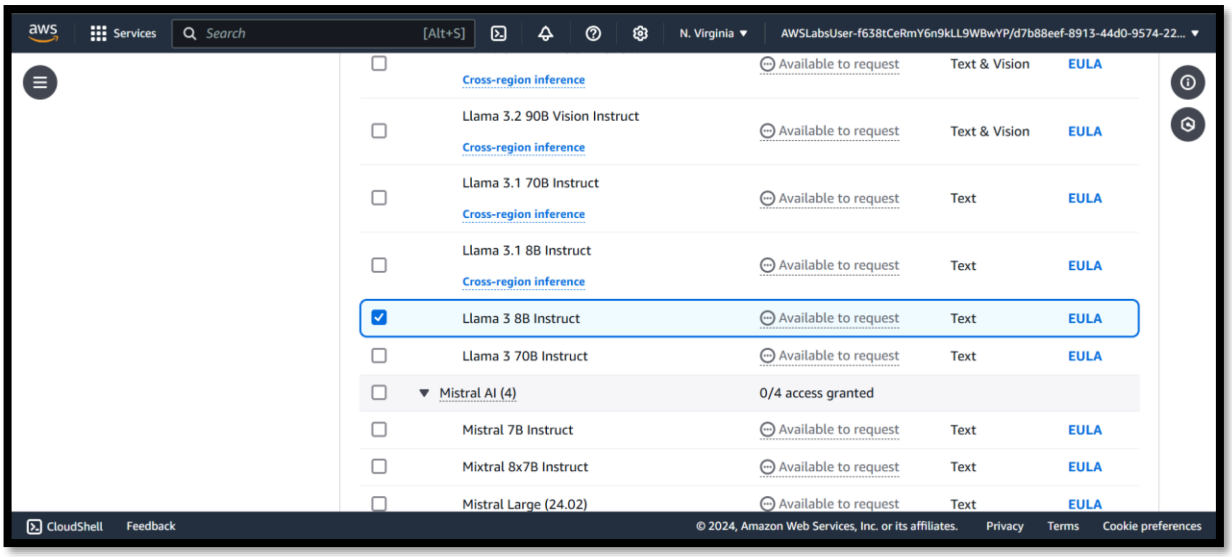
CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences



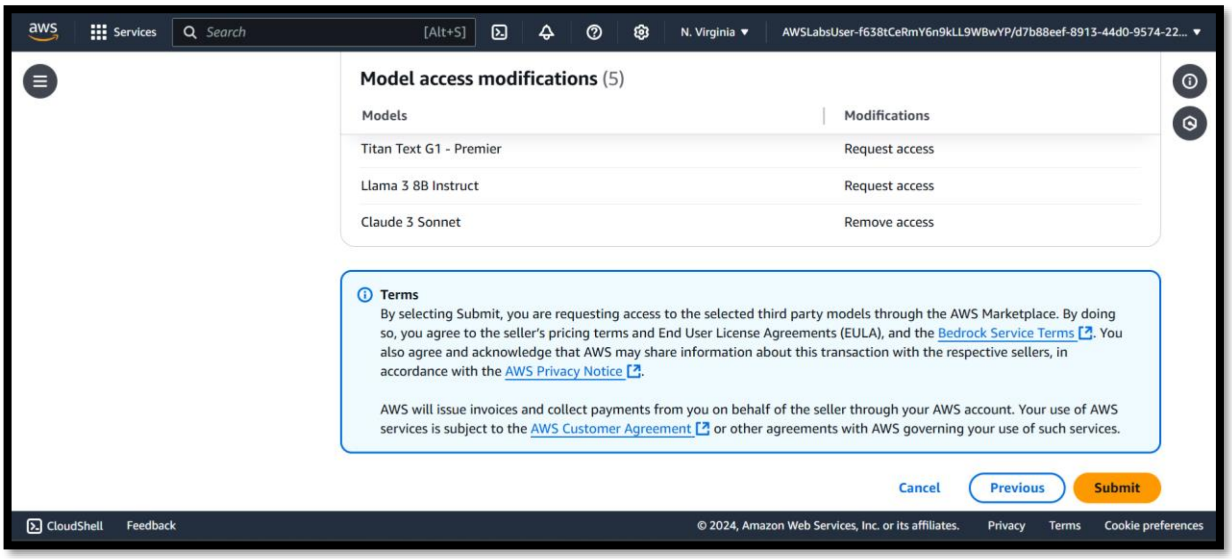
aws Services Search [Alt+S] N. Virginia AWSLabsUser-f638tCeRmY6n9kLL9WBwYP/d7b88eef-8913-44d0-9574-22...

	Models	Access status	Modality	EULA
	Jurassic-2 Mid	Available to request	Text	EULA
	▼ Amazon (13)	0/13 access granted		
✓	Titan Embeddings G1 - Text	Available to request	Embedding	EULA
	Titan Text G1 - Lite	Available to request	Text	EULA
✓	Titan Text G1 - Express	Available to request	Text	EULA
	Titan Image Generator G1 v2	Available to request	Image	EULA
	Titan Image Generator G1	Available to request	Image	EULA
	Nova Reel	Available to request		EULA
	Titan Multimodal Embeddings G1	Available to request	Embedding	EULA
✓	Titan Text G1 - Premier	Available to request	Text	EULA
	Titan Text Embeddings V2	Available to request	Embedding	EULA
	Nova Pro	Available to request	Text & Vision	EULA
	Nova Lite	Available to request	Text & Vision	EULA

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences



	Model	Access status	Modality	EULA
☐	Cross-region inference	Available to request	Text & Vision	EULA
☐	Llama 3.2 90B Vision Instruct Cross-region inference	Available to request	Text & Vision	EULA
☐	Llama 3.1 70B Instruct Cross-region inference	Available to request	Text	EULA
☐	Llama 3.1 8B Instruct Cross-region inference	Available to request	Text	EULA
☑	Llama 3 8B Instruct	Available to request	Text	EULA
☐	Llama 3 70B Instruct	Available to request	Text	EULA
☐	▼ Mistral AI (4)	0/4 access granted		
☐	Mistral 7B Instruct	Available to request	Text	EULA
☐	Mixtral 8x7B Instruct	Available to request	Text	EULA
☐	Mistral Large (24.02)	Available to request	Text	EULA



Model access modifications (5)

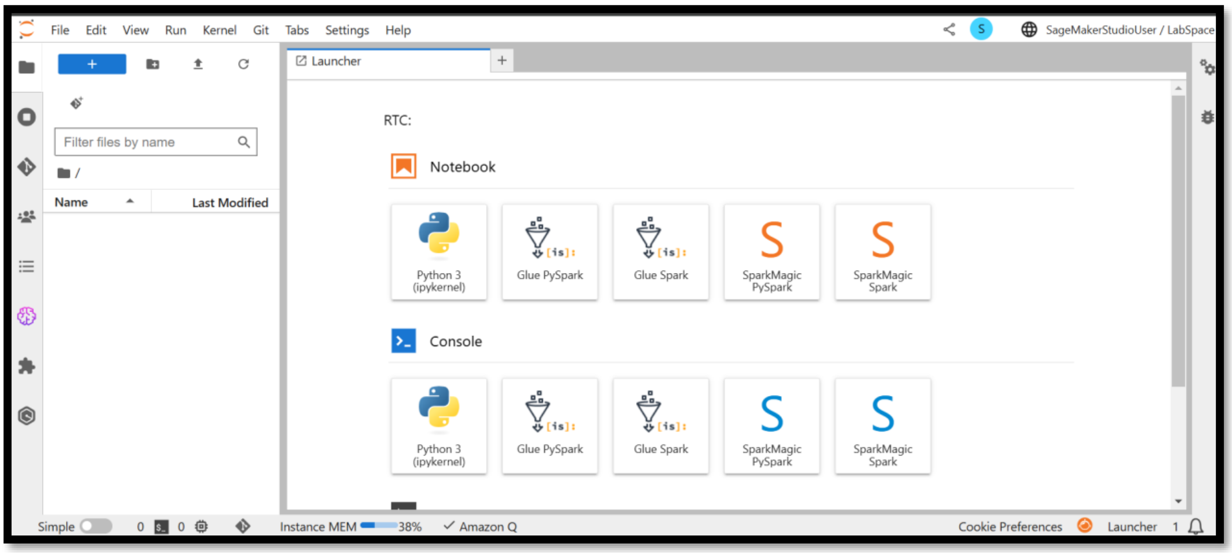
Models	Modifications
Titan Text G1 - Premier	Request access
Llama 3 8B Instruct	Request access
Claude 3 Sonnet	Remove access

Terms

By selecting Submit, you are requesting access to the selected third party models through the AWS Marketplace. By doing so, you agree to the seller's pricing terms and End User License Agreements (EULA), and the Bedrock Service Terms. You also agree and acknowledge that AWS may share information about this transaction with the respective sellers, in accordance with the AWS Privacy Notice.

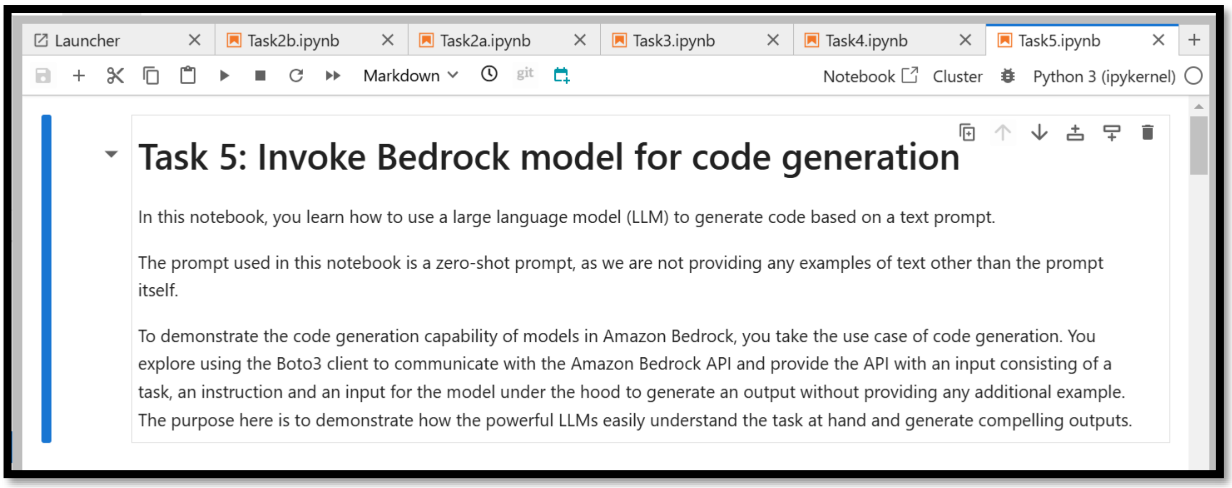
AWS will issue invoices and collect payments from you on behalf of the seller through your AWS account. Your use of AWS services is subject to the AWS Customer Agreement or other agreements with AWS governing your use of such services.

Launch an Amazon SageMaker Studio application



Task 5: Use Amazon Bedrock Models for Code Generation

I used a large language model (LLM) to generate code based on a text prompt.

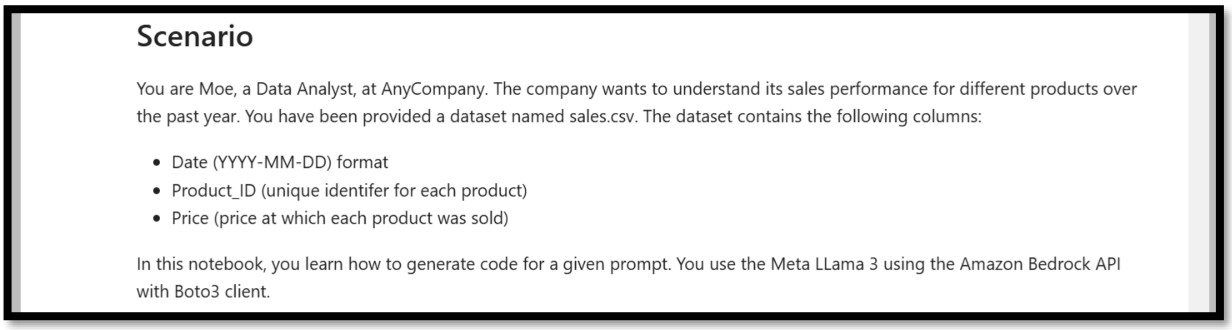


Task 5: Invoke Bedrock model for code generation

In this notebook, you learn how to use a large language model (LLM) to generate code based on a text prompt.

The prompt used in this notebook is a zero-shot prompt, as we are not providing any examples of text other than the prompt itself.

To demonstrate the code generation capability of models in Amazon Bedrock, you take the use case of code generation. You explore using the Boto3 client to communicate with the Amazon Bedrock API and provide the API with an input consisting of a task, an instruction and an input for the model under the hood to generate an output without providing any additional example. The purpose here is to demonstrate how the powerful LLMs easily understand the task at hand and generate compelling outputs.

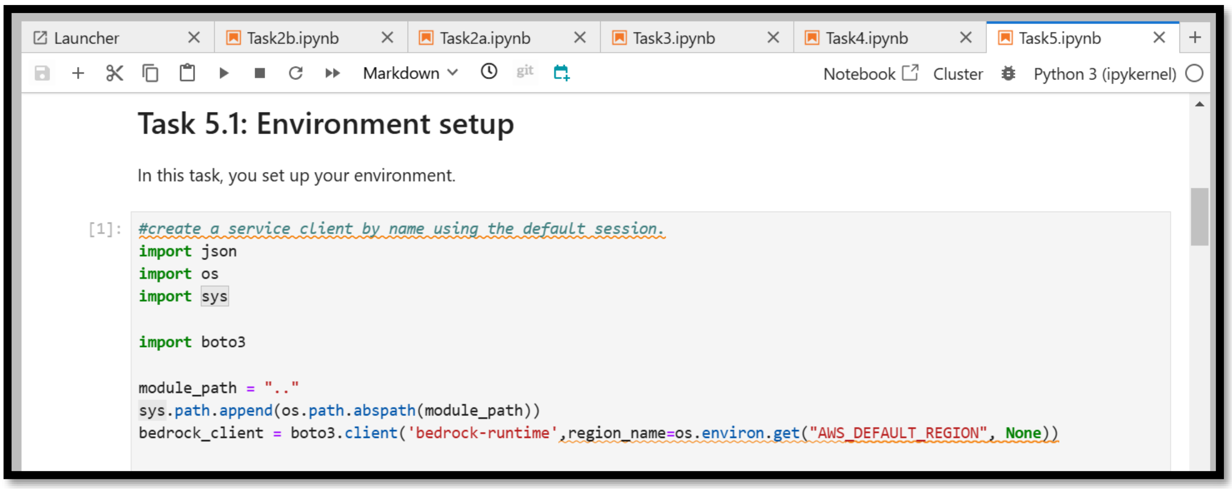


Scenario

You are Moe, a Data Analyst, at AnyCompany. The company wants to understand its sales performance for different products over the past year. You have been provided a dataset named sales.csv. The dataset contains the following columns:

- Date (YYYY-MM-DD) format
- Product_ID (unique identifer for each product)
- Price (price at which each product was sold)

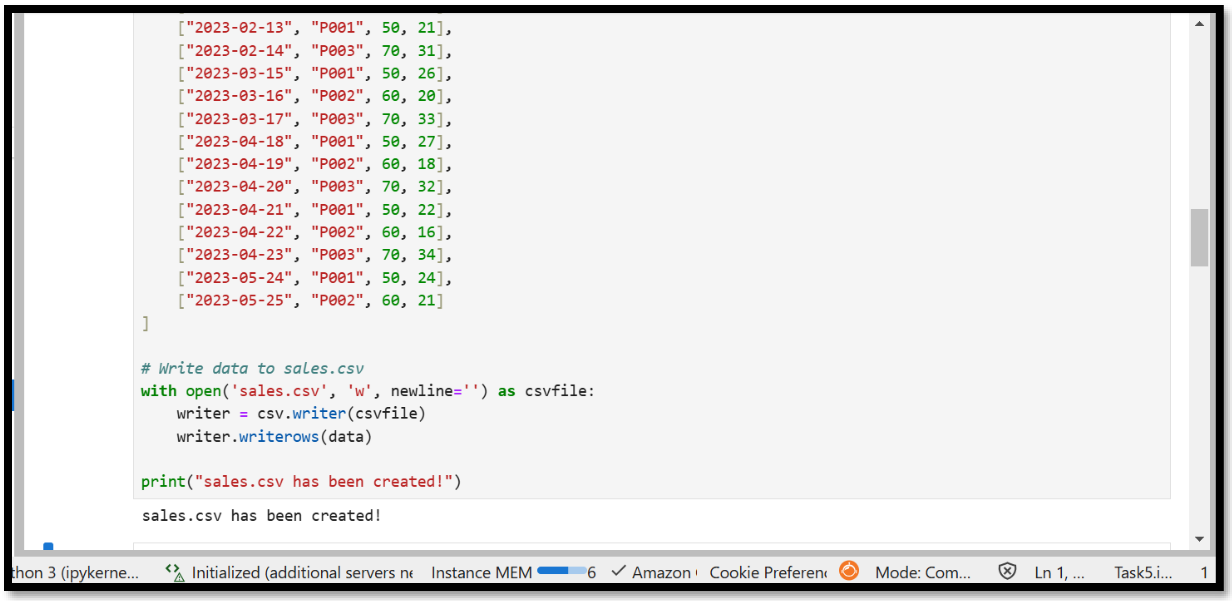
In this notebook, you learn how to generate code for a given prompt. You use the Meta LLama 3 using the Amazon Bedrock API with Boto3 client.



Task 5.1: Environment setup

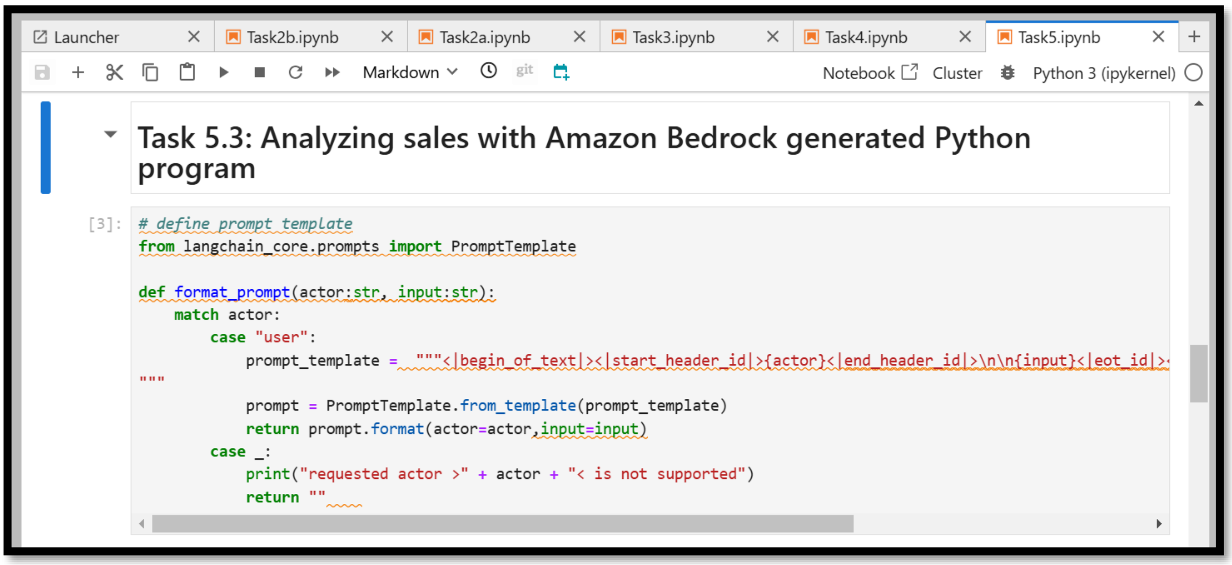
In this task, you set up your environment.

[1]:
```
#create a service client by name using the default session.
import json
import os
import sys

import boto3

module_path = ".."
sys.path.append(os.path.abspath(module_path))
bedrock_client = boto3.client('bedrock-runtime',region_name=os.environ.get("AWS_DEFAULT_REGION", None))
```



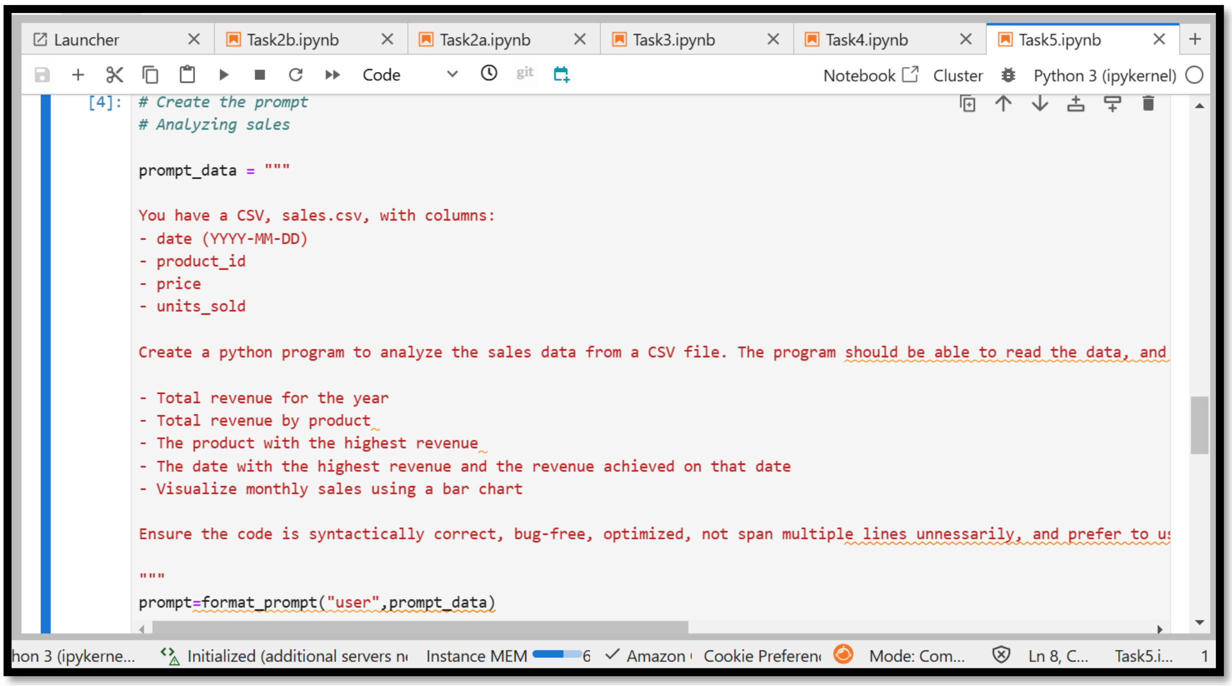
```
    ["2023-02-13", "P001", 50, 21],
    ["2023-02-14", "P003", 70, 31],
    ["2023-03-15", "P001", 50, 26],
    ["2023-03-16", "P002", 60, 20],
    ["2023-03-17", "P003", 70, 33],
    ["2023-04-18", "P001", 50, 27],
    ["2023-04-19", "P002", 60, 18],
    ["2023-04-20", "P003", 70, 32],
    ["2023-04-21", "P001", 50, 22],
    ["2023-04-22", "P002", 60, 16],
    ["2023-04-23", "P003", 70, 34],
    ["2023-05-24", "P001", 50, 24],
    ["2023-05-25", "P002", 60, 21]
]

# Write data to sales.csv
with open('sales.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerows(data)

print("sales.csv has been created!")
```

```
sales.csv has been created!
```

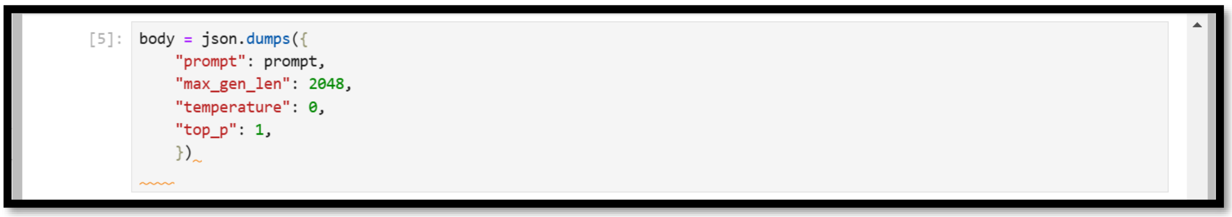


Task 5.3: Analyzing sales with Amazon Bedrock generated Python program

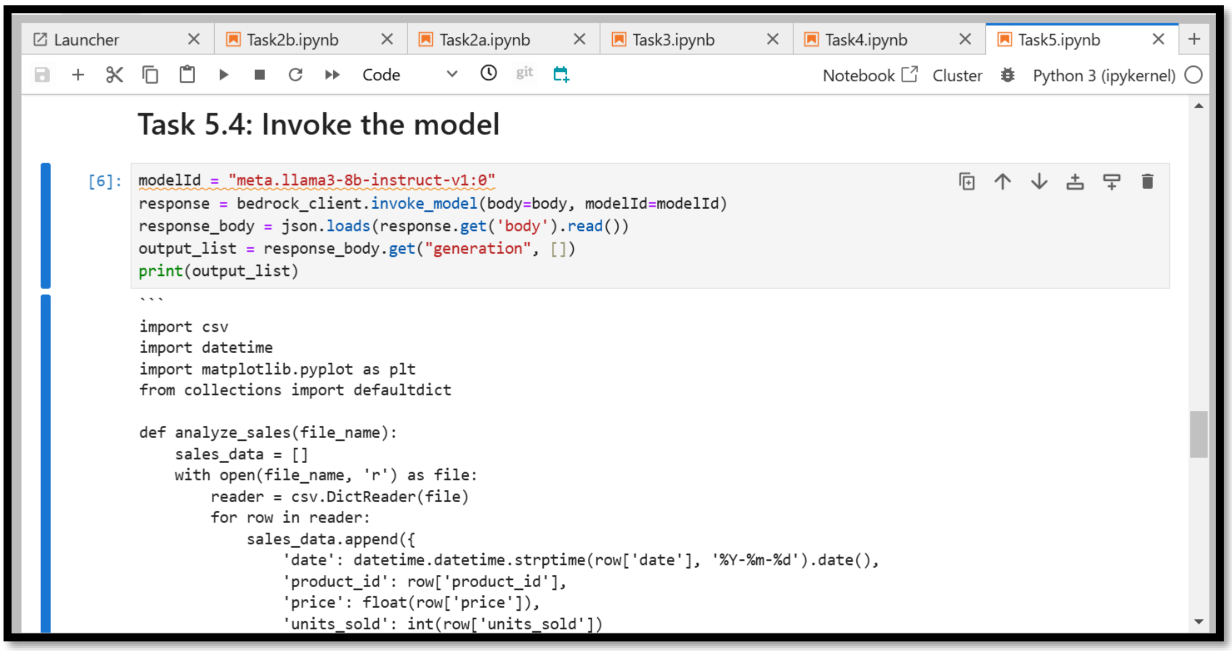
[3]:
```
# define prompt template
from langchain_core.prompts import PromptTemplate

def format_prompt(actor:str, input:str):
    match actor:
        case "user":
            prompt_template = """<|begin_of_text|><|start_header_id|>{actor}<|end_header_id|>\n\n{input}<|eot_id|><
"""
            prompt = PromptTemplate.from_template(prompt_template)
            return prompt.format(actor=actor,input=input)
        case _:
            print("requested actor >" + actor + "< is not supported")
            return ""
```



```
[4]: # Create the prompt
     # Analyzing sales

     prompt_data = """

     You have a CSV, sales.csv, with columns:
     - date (YYYY-MM-DD)
     - product_id
     - price
     - units_sold

     Create a python program to analyze the sales data from a CSV file. The program should be able to read the data, and

     - Total revenue for the year
     - Total revenue by product
     - The product with the highest revenue
     - The date with the highest revenue and the revenue achieved on that date
     - Visualize monthly sales using a bar chart

     Ensure the code is syntactically correct, bug-free, optimized, not span multiple lines unnessarily, and prefer to us

     """
     prompt=format_prompt("user",prompt_data)
```



```
[5]: body = json.dumps({
         "prompt": prompt,
         "max_gen_len": 2048,
         "temperature": 0,
         "top_p": 1,
         })
```



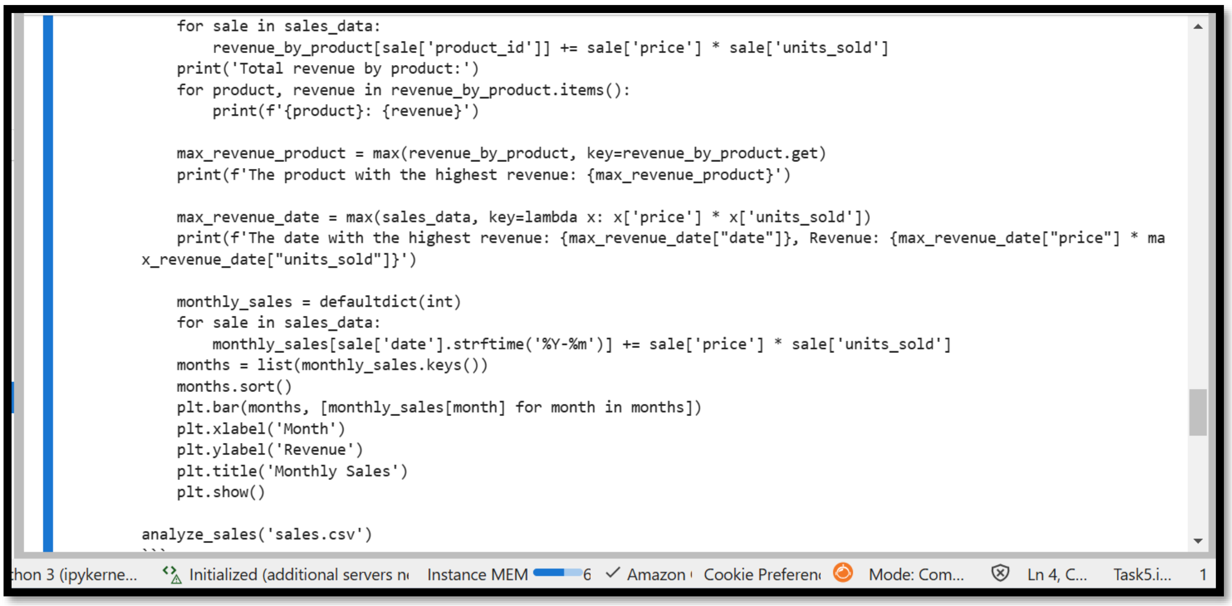
Task 5.4: Invoke the model

```
[6]: modelId = "meta.llama3-8b-instruct-v1:0"
     response = bedrock_client.invoke_model(body=body, modelId=modelId)
     response_body = json.loads(response.get('body').read())
     output_list = response_body.get("generation", [])
     print(output_list)
```

```
```

import csv
import datetime
import matplotlib.pyplot as plt
from collections import defaultdict

def analyze_sales(file_name):
    sales_data = []
    with open(file_name, 'r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            sales_data.append({
                'date': datetime.datetime.strptime(row['date'], '%Y-%m-%d').date(),
                'product_id': row['product_id'],
                'price': float(row['price']),
                'units_sold': int(row['units_sold'])
```
```



```
    for sale in sales_data:
        revenue_by_product[sale['product_id']] += sale['price'] * sale['units_sold']
    print('Total revenue by product:')
    for product, revenue in revenue_by_product.items():
        print(f'{product}: {revenue}')

    max_revenue_product = max(revenue_by_product, key=revenue_by_product.get)
    print(f'The product with the highest revenue: {max_revenue_product}')

    max_revenue_date = max(sales_data, key=lambda x: x['price'] * x['units_sold'])
    print(f'The date with the highest revenue: {max_revenue_date["date"]}, Revenue: {max_revenue_date["price"] * ma
x_revenue_date["units_sold"]}')

    monthly_sales = defaultdict(int)
    for sale in sales_data:
        monthly_sales[sale['date'].strftime('%Y-%m')] += sale['price'] * sale['units_sold']
    months = list(monthly_sales.keys())
    months.sort()
    plt.bar(months, [monthly_sales[month] for month in months])
    plt.xlabel('Month')
    plt.ylabel('Revenue')
    plt.title('Monthly Sales')
    plt.show()

analyze_sales('sales.csv')
```