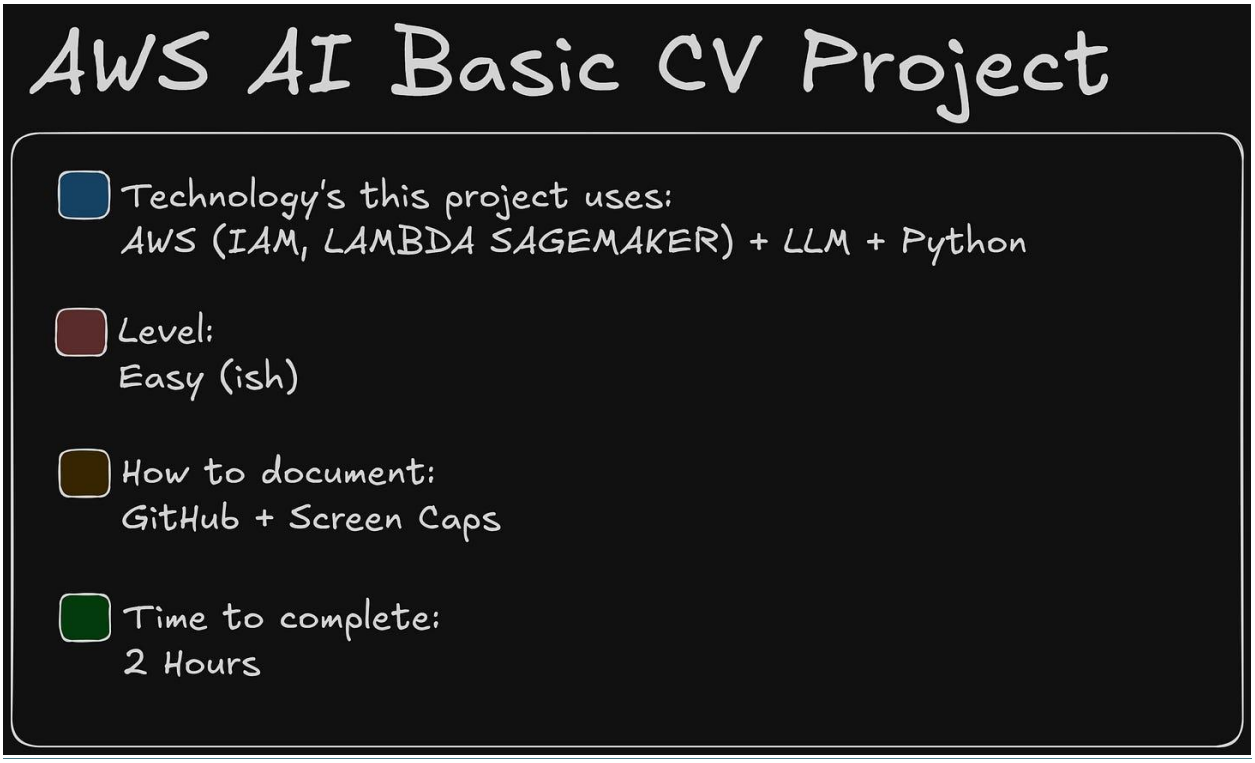
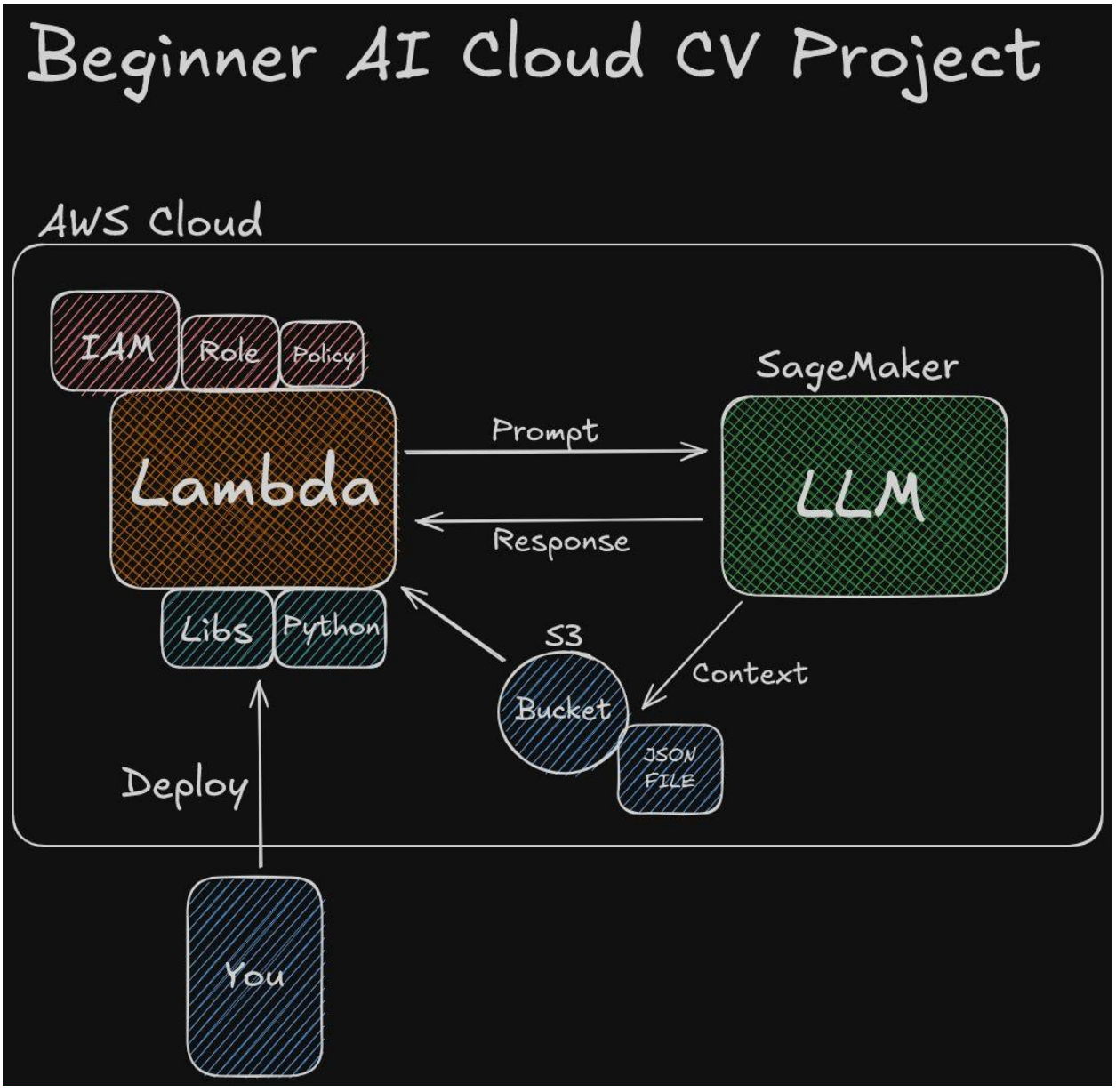
Project Metadata:



Project Plan:



This week, we're going to deploy an AWS Lambda function containing a Python script with the appropriate permissions to query a large language model that we're hosting in SageMaker. We'll also add contextualised data using Amazon S3, which the model can reference.

Why?

LLMs are powerful, but when deployed in real world scenarios, we often want them to understand and respond using our own relevant data. Maybe that's internal documents,

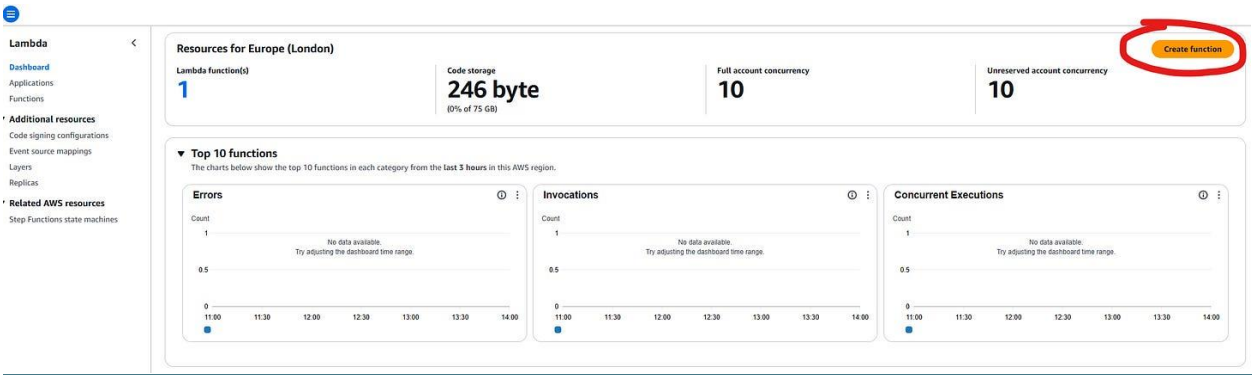
logs, or domain-specific knowledge. This approach is sometimes called **lightweight RAG**, it helps us inject relevant, custom data into the model's prompt without the complexity of a full on RAG (I recommend looking more into this part, but for the project you just need a basic understanding)

Let's get to it...

Step One:

First, we need to deploy our Lambda function. It is recommended to do this via Terraform or the AWS CDK, but if you are more comfortable with the console, feel free to use that for now. Since most of my readers are just starting out, we will use the console for this guide.

Navigate to the Lambda console and select **Create function**.



Configure the Lambda function to use Python (choose the latest supported version, such as Python 3.12), and set the timeout to 1 minute.

This next part is really simple...

Attach the AWSLambdaBasicExecutionRole to your function. This role gives it basic permissions to write logs to CloudWatch. We'll create a more tightly scoped role with the correct permissions for accessing SageMaker and S3 later in the tutorial.

Now that we've created our Lambda function, let's deploy some Python code to it.

I recommend using Infrastructure as Code (like Terraform) to make this repeatable, but if you're new, that might feel like too much right now.

So for now, you can copy and paste this script directly into the Lambda code editor. The script has comments so you can understand each block, if there’s something you don’t understand, spend the the time here to ensure you do.

```
import json
import boto3

# Init the SageMaker and S3 clients
sagemaker_client = boto3.client('sagemaker-runtime')
s3_client = boto3.client('s3')

def lambda_handler(event, context):
    # Get user input from the incoming Lambda event
    user_input = event.get('user_input', '')

    try:
        # Fetch data from S3 (optional, adjust key if you're using this for
game data)
        s3_response = s3_client.get_object(
            Bucket='heycybernotes-readers',
            Key='console_pricing.json'
        )
        pricing_data = json.loads(s3_response['Body'].read().decode('utf-8'))

        # Check if the user is asking about console/game pricing
        if any(keyword in user_input.lower() for keyword in ['price', 'cost',
'console', 'game']):
            prompt = f"""
            User: {user_input}

            You are a video game expert with up-to-date knowledge of console
and game pricing. Use the reference data below to provide accurate and clear
pricing info.

            Reference data:
            {pricing_data}

            Please provide relevant pricing details and short recommendations
if applicable.
            """
        else:
            prompt = f"""
            User: {user_input}

            You are a professional video game expert. Answer the user's
question clearly and helpfully, using examples from gaming history, game
mechanics, or recent industry trends where helpful.

            Keep it beginner-friendly and enthusiastic.
            """

        # Call the SageMaker endpoint to invoke the LLM with the prompt
        print("Prompt: {}".format(prompt))
        response = sagemaker_client.invoke_endpoint(
            EndpointName='meta-textgeneration-llama-2-7b',
            ContentType='application/json',
```

```
        # Call the SageMaker endpoint to invoke the LLM with the prompt
        print("Prompt: {}".format(prompt))
        response = sagemaker_client.invoke_endpoint(
            EndpointName='meta-textgeneration-llama-2-7b',
            ContentType='application/json',
            Body=json.dumps({
                "inputs": prompt,
                "parameters": {
                    "temperature": 0.2
                }
            }),
            InferenceComponentName='meta-textgeneration-llama-2-7b'
        )

        # Decode model response
        raw_response = response['Body'].read().decode('utf-8')
        print("Raw SageMaker response:", raw_response)
        response_body = json.loads(raw_response)

        if isinstance(response_body, list) and len(response_body) > 0:
            response_body = response_body[0]

        generated_text = response_body.get('generated_text', 'No response
enerated.')

        return {
            "statusCode": 200,
            "body": json.dumps({"response": generated_text})
        }

    except Exception as e:
        print("Lambda error:", str(e))
        return {
            "statusCode": 500,
            "body": json.dumps({"error": str(e)})
        }
```

This Python script runs in your Lambda function and sends your input to your LLM hosted on SageMaker. The model acts like a video game expert, answering your questions or giving pricing info.

The script checks your input for keywords like **"price"** or **"cost" "**. If it finds them, it assumes you're asking about the price of something, like a game console and it pulls in pricing data from a file stored in Amazon S3 or **ELSE** it goes ahead with prompt number two.

Key parts to change:

1. Key parts to change:
    a. **S3 bucket and key**
    b.
```
Bucket='heycybernotes-readers', Key='console_pricing.json'
```
    Change this to your own S3 bucket and file if you're storing game or console data.
    c. **Prompt content**
    Inside the two prompt = f"""..."""" sections - these control how the model behaves. You can edit these to change the tone or focus (retro games, indie titles or even something completely different).
    d. EndpointName **and** InferenceComponentName - For now just leave these blank. We will have them once we've deployed our model.

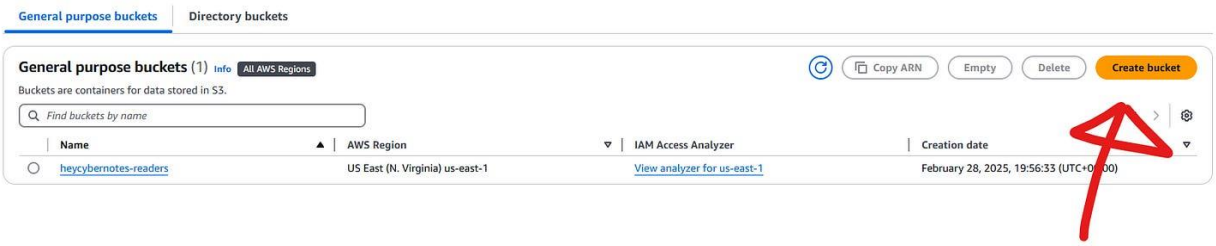
```
EndpointName='meta-textgeneration-llama-2-7b' - This is mine, yours might
differ  //

InferenceComponentName='meta-textgeneration-llama-2-7b'
```

Step Two:

Next, let's deploy an S3 bucket with some JSON data inside....

To do this, head over to the S3 console and select **Create bucket**.



Walk through the setup wizard, ensuring that Block Public Access is enabled and other default settings are left as they are.

Back on your desktop, we need to create some data that our large language model will reference before it’s response

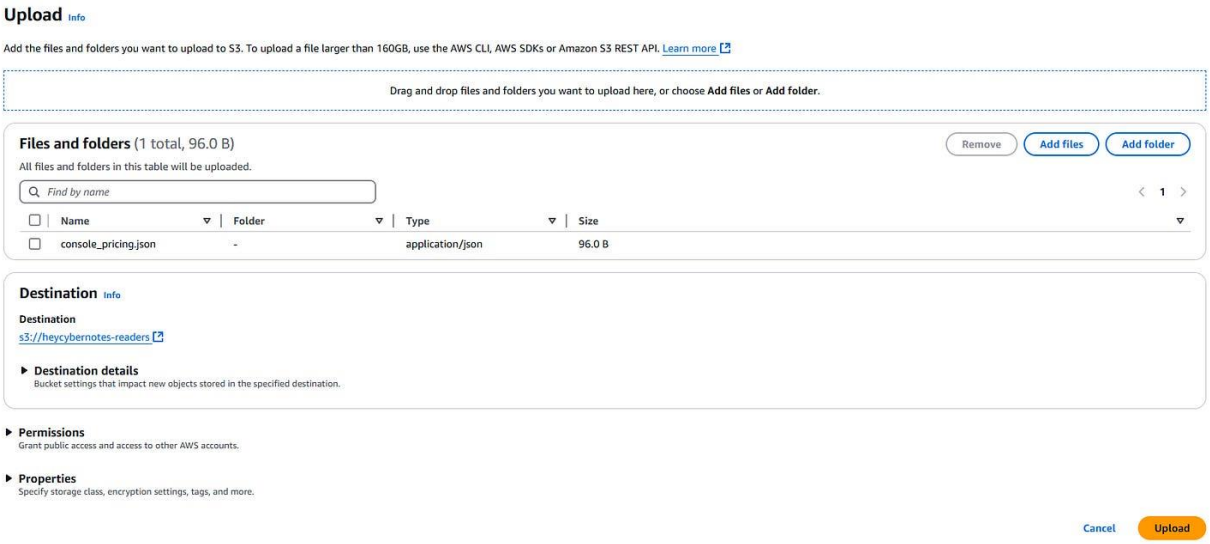
Create a new file and place this inside:

```
[
  "Nintendo Switch - £249.99",
  "Xbox Series X £449.99",
  "PlayStation 5 £449.99"
]
```

You can create what data you like here, so long as it’s JSON array of strings - Each item in the array is just a string containing both a product name and a price, but there's no structured key value format. You can experiment with different formats after you’ve completed this.

Upload this data to your new S3 bucket with this command OR by dragging and dropping:

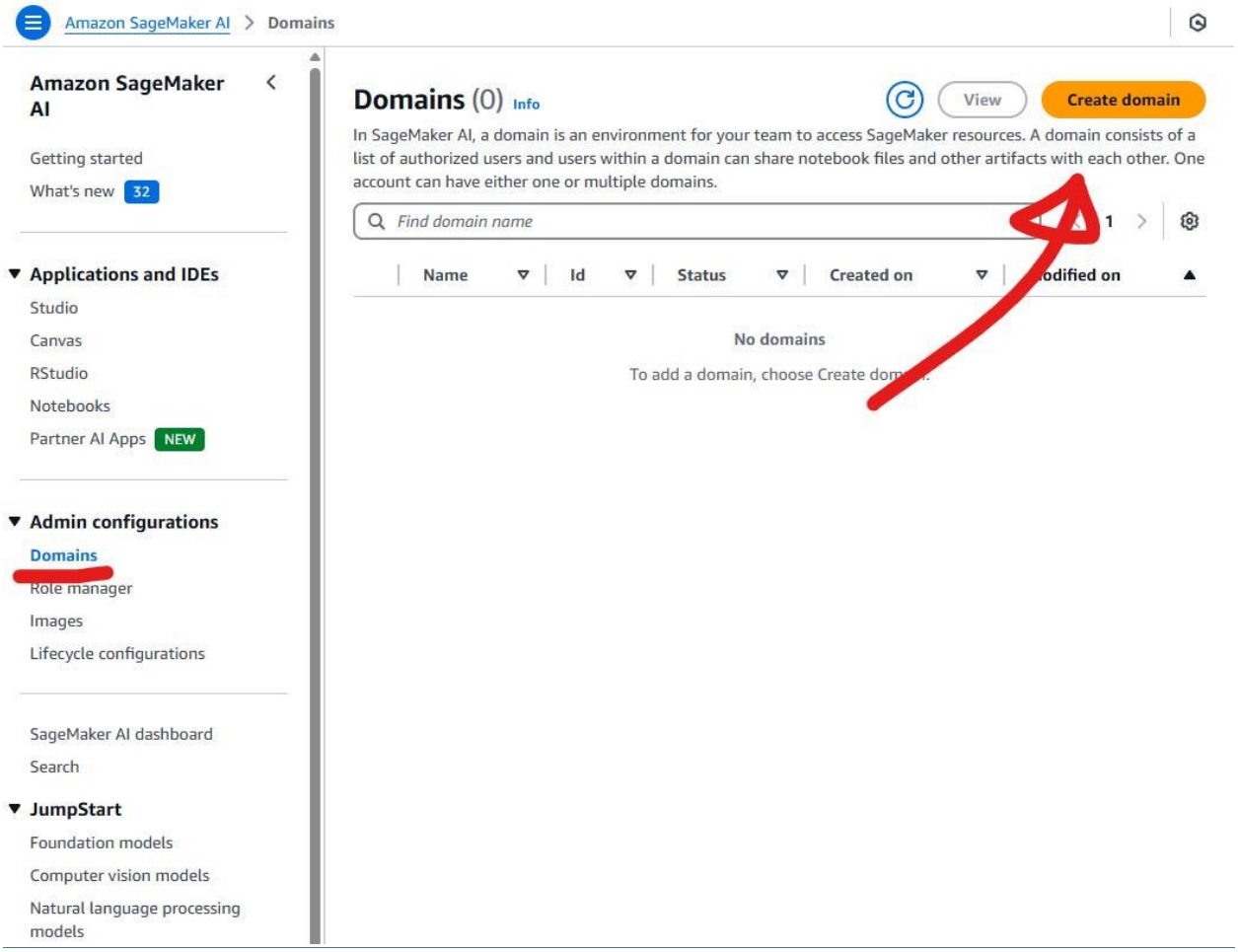
```
aws s3 cp console_pricing.json s3://<yourbucket/console_pricing.json
```



Step Three:

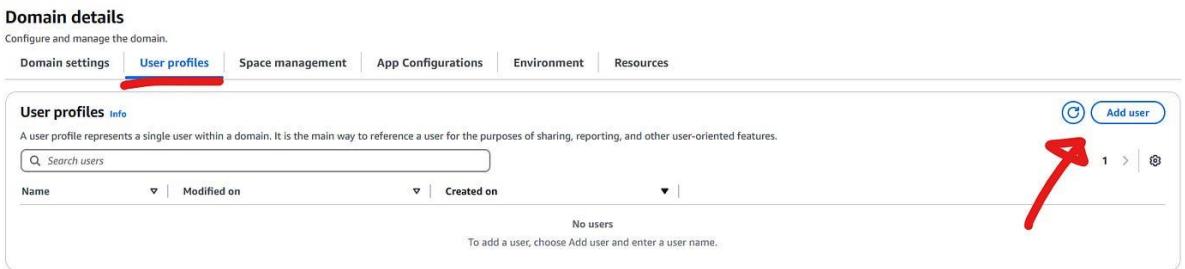
First, we need to create a SageMaker Domain, which is required for launching and managing SageMaker Studio apps, including the ones used to host LLMs.

1. In the AWS Console, navigate to SageMaker.
2. On the left-hand menu, click Domains.
3. Click “Create domain”, then choose Quick setup.
4. Follow the prompts to finish the setup.



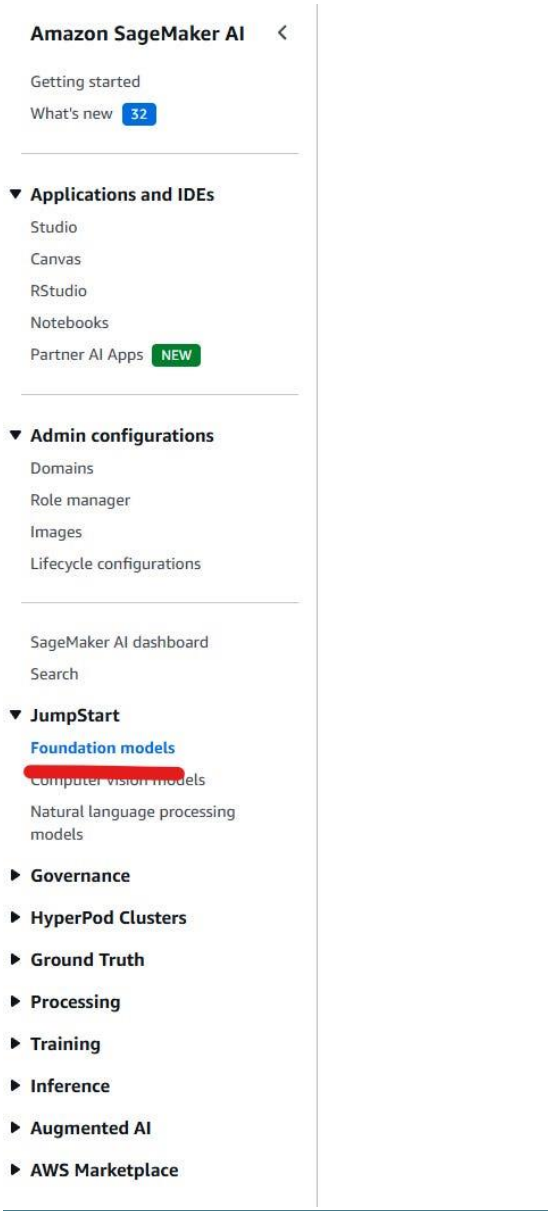
Now, you'll need to wait around 10 minutes for the domain status to change from "Pending" to "Ready".

When ready, click into it and navigate to the User Profiles tab then create a new domain user



The default name and Execution role are fine to use.

With your domain and user ready we can now go to Foundation Models.



From here we can see all the available models.
You can use what you like here but if you want to follow along exactly I recommend using

the “**Meta Llama 2 7B**” it’s a smallish model that’s perfect for text to text generation. if you can’t see it, search for ***“Meta Llama 2 7B Chat”***

⚠️ **Cost Warning:**
This next step will incur charges on your AWS account, as we need to launch a **large instance** to run the LLM.

To keep costs down, I recommend **deleting the instance immediately after documenting your project**. If you're quick, the total cost should stay under **£2–£5**, but this is only an estimate and may vary depending on your region and usage.

📜 **Important:**

- **I’m not responsible for any charges you may incur.**
- You may need to request a service increase to launch the instance.
- I **highly recommend** setting up **AWS billing alerts** to monitor your spending.

Step Four:

Let’s configure the Lambda role properly now. It’s best to create a new role for this function.

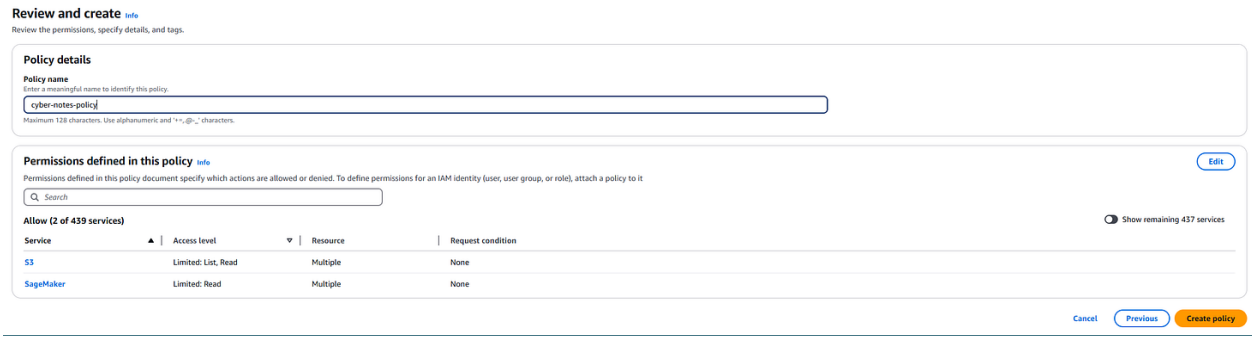
1. Go to the **IAM** section in the AWS Console.
2. Find the current Lambda function and make a note of any IAM policies it already has attached, like those for logging to CloudWatch.
3. Now, create a new IAM role with a clear, meaningful name, such as lambda-video-game-assistant-role.
4. In the new role, add a new inline policy with the JSON I’ll provide below.

This approach keeps things clean and lets you tightly control permissions for this specific project.

If this part feels confusing or overwhelming, that’s completely normal. I’d recommend bookmarking this project and coming back to it after spending some time learning how IAM works; especially roles, permissions, and policies. Note the two resources you need to change here.

{

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowInvokeEndpoint",
            "Effect": "Allow",
            "Action": "sagemaker:InvokeEndpoint",
            "Resource": [
                "arn:aws:sagemaker:eu-west-2:<YourAccountID>:endpoint/meta-
textgeneration-llama-2-7b",
                "arn:aws:sagemaker:eu-west-2:<YourAccountID:inference-
component/meta-textgeneration-llama-2-7b"
            ]
        },
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::<S3BUCKETNAME>",
                "arn:aws:s3:::<S3BCUKETNAME>*"
            ]
        }
    ]
}
```

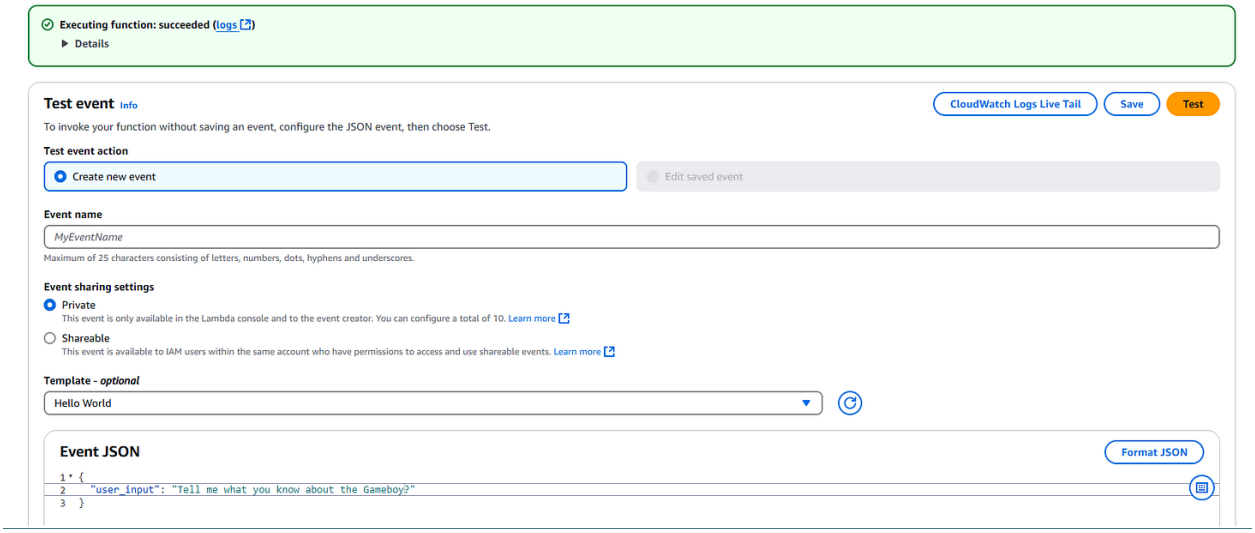


Click save and navigate back to your lambda.

Step Five:

Let’s test this!

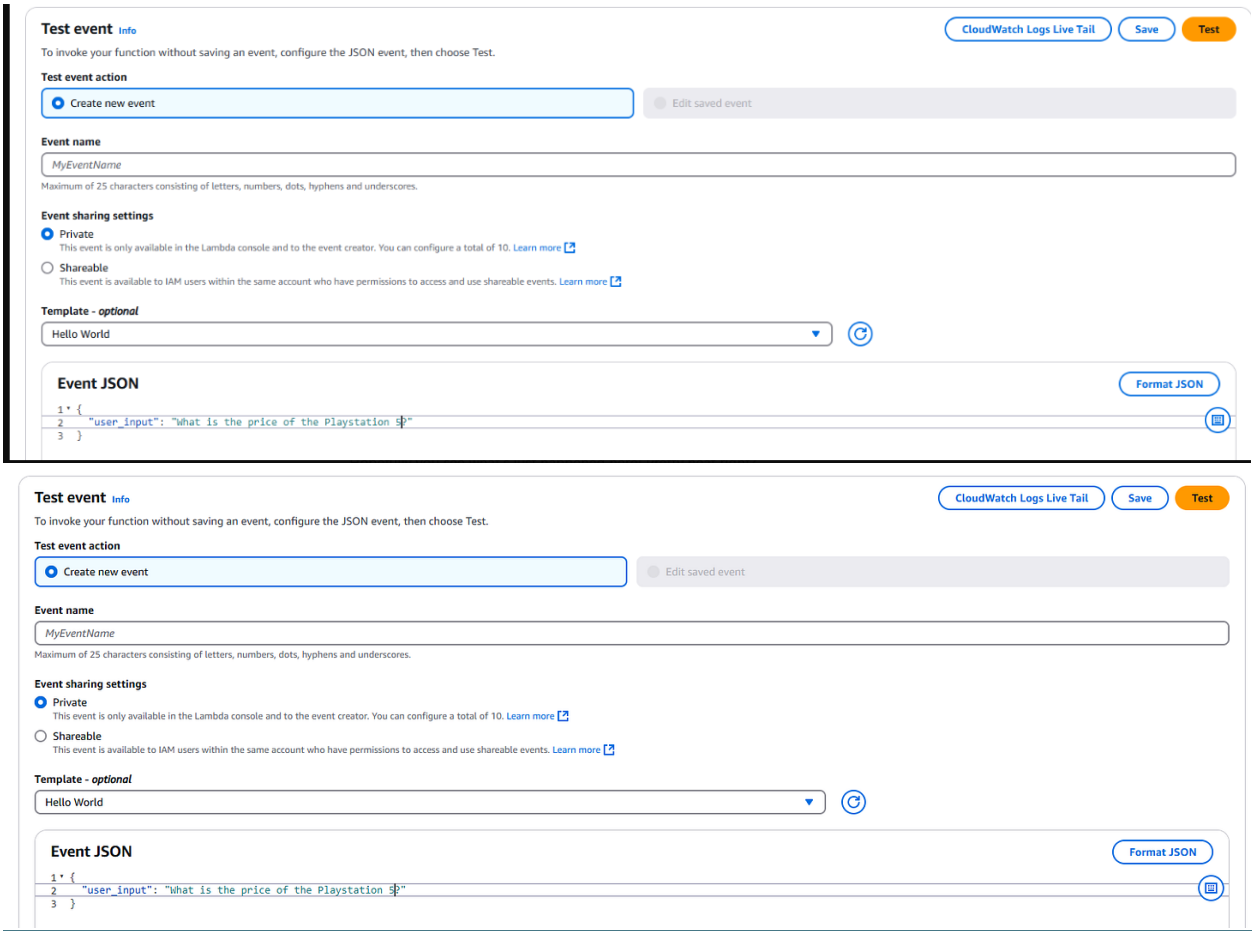
Create a new test event for your lambda and pass it the user_input event with a prompt.




```
{
  "user_input": "Tell me what you know about the Gameboy?"
}
```

As you can see it responds with some basic information about what it knows here.

“The Game Boy is a small handheld gaming device made by Nintendo. It first came out in Japan on April 21, 1989, and then later in North America and other places. It was different from earlier Nintendo handhelds because it could play different games using cartridges.”




```
{
  "user_input": "What is the price of the Playstation 5?"
}
```

Hopefully you see what's just happened here! Pretty neat right?

I'm curious to see how many of you complete these projects. Run your model and comment below with what it responds - I'll send a free copy of TechOneTwenty! to the first 3.

🎉🎉**Congratulations**🎉🎉What you have deployed here is awesome!

Step Six:

Document... Create a GitHub repository and include everything you've built, your Python scripts, any Infrastructure as Code files like Terraform, and a well written README that explains what the project does, how to deploy it, and any costs or limitations. This not only

helps you track your progress, but also builds your public portfolio for future employers or collaborators to see.