

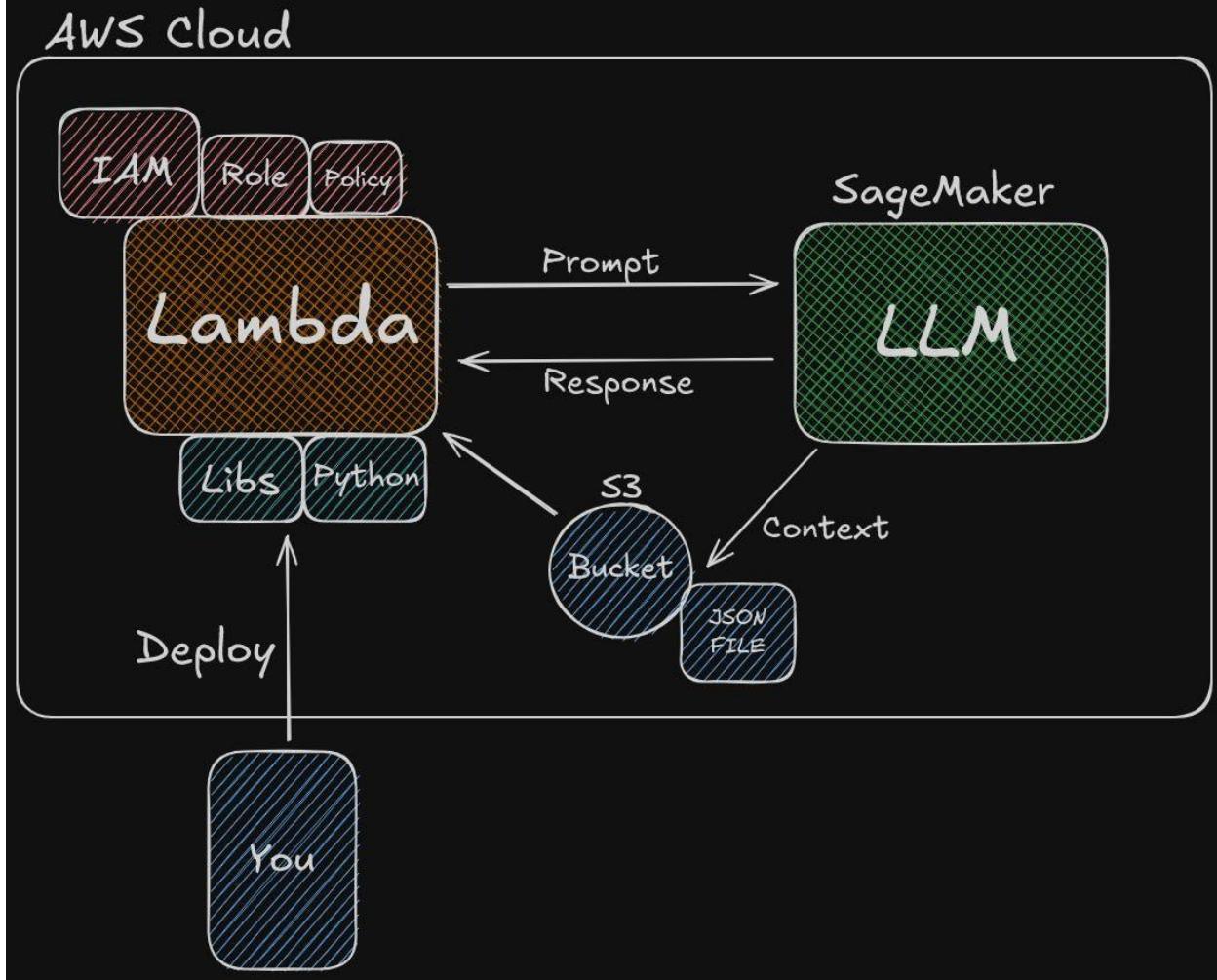
**Project Metadata:**

# AWS AI Basic CV Project

- Technology's this project uses:  
AWS (IAM, LAMBDA SAGEMAKER) + LLM + Python
- Level:  
Easy (ish)
- How to document:  
GitHub + Screen Caps
- Time to complete:  
2 Hours

**Project Plan:**

# Beginner AI Cloud CV Project



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,

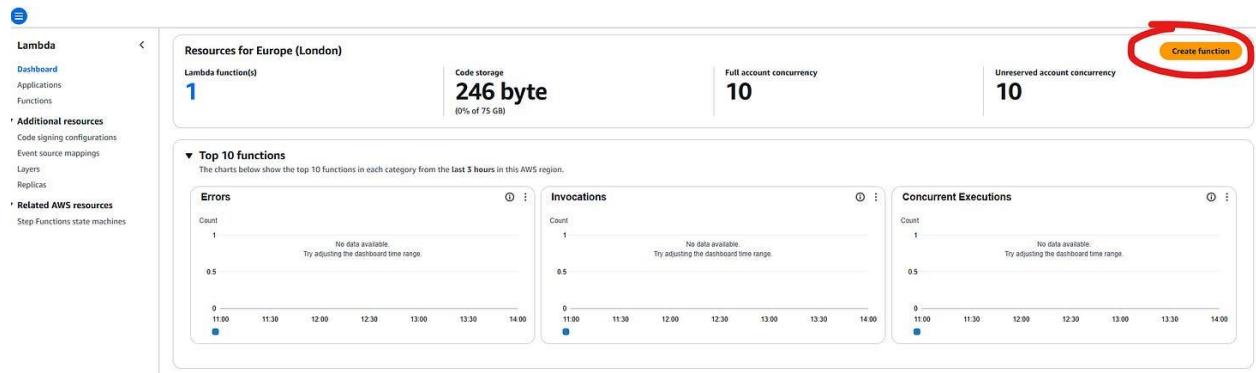
logs, or domain-specific knowledge. This approach is sometimes called **lightweight RAG**, it helps us inject relevant, custom data into the model's prompt without the complexity of a full on RAG (I recommend looking more into this part, but for the project you just need a basic understanding)

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 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='hey cybernotes-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
generated.')

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

```
Bucket='heycybernotes-readers', Key='console_pricing.json'
```

b.

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
```

**Upload** Info

Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDKs or Amazon S3 REST API. [Learn more](#)

Drag and drop files and folders you want to upload here, or choose Add files or Add folder.

**Files and folders** (1 total, 96.0 B)

All files and folders in this table will be uploaded.

<input type="checkbox"/>	Name	Folder	Type	Size
<input type="checkbox"/>	console_pricing.json	-	application/json	96.0 B

**Destination** Info

Destination  
[s3://heycybernotes-readers](#)

**Destination details**  
Bucket settings that impact new objects stored in the specified destination.

**Permissions**  
Grant public access and access to other AWS accounts.

**Properties**  
Specify storage class, encryption settings, tags, and more.

[Cancel](#) [Upload](#)

---

### 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.

Amazon SageMaker AI > Domains

**Amazon SageMaker AI**

Getting started

What's new 32

**Applications and IDEs**

- Studio
- Canvas
- RStudio
- Notebooks
- Partner AI Apps NEW

**Admin configurations**

- Domains** (highlighted)
- Role manager
- Images
- Lifecycle configurations

SageMaker AI dashboard

Search

**JumpStart**

- Foundation models
- Computer vision models
- Natural language processing models

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

**Domain details**

Configure and manage the domain.

Domain settings | **User profiles** (highlighted) | Space management | App Configurations | Environment | Resources

**User profiles** Info

A user profile represents a single user within a domain. It is the main way to reference a user for the purposes of sharing, reporting, and other user-oriented features.

Q Search users

Name	Modified on	Created on
No users		

To add a user, choose Add user and enter a user name.

The default name and Execution role are fine to use.

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

A screenshot of the Amazon SageMaker AI console's navigation bar. The bar includes a back arrow, a search icon, and account information. Below the bar, there are several sections:

- Getting started**
- What's new** (32)
- Applications and IDEs**
  - Studio
  - Canvas
  - RStudio
  - Notebooks
  - Partner AI Apps (NEW)
- Admin configurations**
  - Domains
  - Role manager
  - Images
  - Lifecycle configurations
- SageMaker AI dashboard**
- Search**
- JumpStart**
  - Foundation models (highlighted with a red box)
  - Computer vision models
  - Natural language processing models
- Governance**
- HyperPod Clusters**
- Ground Truth**
- Processing**
- Training**
- Inference**
- Augmented AI**
- AWS Marketplace**

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:::<S3BUCKETNAME>*"
            ]
        }
    ]
}
```

#### Review and create Info

Review the permissions, specify details, and tags.

**Policy details**

**Policy name**  
Enter a meaningful name to identify this policy.  
  
Maximum 128 characters. Use alphanumeric and '+', '.', '@', '-' characters.

**Permissions defined in this policy** Info  
Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

Allow (2 of 439 services)			
Service	Access level	Resource	Request condition
S3	Limited: List, Read	Multiple	None
SageMaker	Limited: Read	Multiple	None

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.

The screenshot shows the AWS Lambda 'Test event' configuration page. At the top, a green header bar indicates 'Executing function: succeeded (Logs [2])' and has a 'Details' link. Below this, the main form has the following fields:

- Test event** info: A note to invoke your function without saving an event, configure the JSON event, then choose Test.
- Test event action**: A radio button group where 'Create new event' is selected, and 'Edit saved event' is unselected.
- Event name**: An input field containing 'MyEventName'. A note below says: 'Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.'
- Event sharing settings**: A radio button group where 'Private' is selected, and 'Shareable' is unselected. A note below says: 'This event is only available in the Lambda console and to the event creator. You can configure a total of 10.' and 'Learn more [2]'.
- Template - optional**: A dropdown menu showing 'Hello World' with a 'C' icon for copy.
- Event JSON**: A code editor containing the following JSON:

```
1: {
2:   "user_input": "Tell me what you know about the Gameboy?"
3: }
```

A 'Format JSON' button is to the right, and a 'Copy' icon is at the bottom right of the editor.

At the top right of the main form are three buttons: 'CloudWatch Logs Live Tail', 'Save', and 'Test'.

```
{  
  "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."*

The screenshot shows two identical configurations for a Lambda function's test event. Both configurations include:

- Test event info**: A note to invoke the function without saving an event by configuring a JSON event and choosing Test.
- Test event action**: Set to "Create new event".
- Event name**: "MyEventName".
- Event sharing settings**: Set to "Private".
- Template - optional**: Set to "Hello World".
- Event JSON**: A code editor containing the following JSON:
 

```
1 * {  
2   "user_input": "What is the price of the Playstation 5?"  
3 }
```
- Buttons**: CloudWatch Logs Live Tail, Save, and Test.

```
{
  "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.