Amazon Bedrock

Índice

- Índice
  - Description:
  - Course objectives
  - Intended audience
  - Prerequisites
  - Course outline
- Module 1: Introduction to Amazon Bedrock
  - Applications and Use Cases
- Module 2: Foundation Models
  - Inference parameters
    - Randomness and diversity
    - Length
  - Working with Amazon Bedrock FMs
    - Amazon Titan foundation models
  - AI21 Jurassic-2 (Mid and Ultra)
  - Anthropic Claude
    - Using Messages API
    - Using Converse API
  - Stability AI (SDXL)
  - Cohere Command
- Module 3: Application Components
- Module 4: Using LangChain
- Demo 1: Explore Generative AI Use Cases using LangChain and Amazon Bedrock
- Module 5: Using Knowledge Bases
- Demo 2: Build and Evaluate Retrieval Augmented Generation (RAG) Applications Using Amazon Bedrock Knowledge Bases
- Module 6: Using Agents
- Demo 3: Explore Amazon Bedrock Agents integrated with Amazon Bedrock Knowledge Bases and Amazon Bedrock Guardrails

Description:

This course is designed for application developers interested in building generative artificial intelligence (generative AI) applications using either the Amazon Bedrock APIs or AWS-LangChain integration. In this course, you will explore the architecture patterns and implementations to support generative AI use cases such as generating and summarizing text, retrieval augmented generation (RAG), and question answering.

You learn to build RAG application using Amazon Bedrock Knowledge Bases, and AI Assistants that use knowledge bases and user-developed tools to answer questions using Amazon Bedrock Agents. You'll also learn to implement safeguards customized to your application requirements and responsible AI policies using Amazon Bedrock Guardrails.

Course objectives

In this course, you will learn to:

- Identify the components of a generative AI application and the options to customize a foundation model (FM)
- Describe Amazon Bedrock foundation models, inference parameters, and key Amazon Bedrock APIs
- Describe the architecture patterns that can be used to build generative AI applications
- Identify Amazon Web Services (AWS) offerings that help with monitoring, securing, and governing your Amazon Bedrock applications

- Describe LangChain components such as prompt templates, chains, retrievers, and agents
- Apply LangChain components to Amazon Bedrock models to build and test use cases such as text and code generation, summarization, RAG, and question answering
- Use Amazon Bedrock Knowledge Bases to implement RAG applications using best practices
- Use Amazon Bedrock Agents with Amazon Bedrock Knowledge Bases and Amazon Bedrock Guardrails for agent applications

Intended audience

This course is intended for:

- Generative AI application developers

Prerequisites

We recommend that attendees of this course have:

- Intermediate to expert-level proficiency with Python programming language
- *AWS Technical Essentials* (Fundamental)
- *AWS Lambda Foundations* (Fundamental)
- *Amazon Bedrock Getting Started* (Fundamental)
- *Foundations of Prompt Engineering* (Intermediate)

Course outline

Module 1: Introduction to Amazon Bedrock

Building Generative AI Applications on Amazon Bedrock

Applications and Use Cases

The rise of generative artificial intelligence (generative AI) has revolutionized the work of machine learning (ML) engineers, developers, and data scientists. They can use it to work on innovative and impactful projects by automating repetitive implementation tasks.

Data scientists can operate at a higher, more strategic level, designing innovative generative AI solutions to address real business problems that directly impact end users. They can focus on architecting solutions, such as AI assistants, supply chain optimizers, and personalized recommendation systems.

Amazon Bedrock offers several natural language processing (NLP) capabilities that can assist data scientists in their work.

- **Text summarization:** using Amazon Bedrock foundation models (FMs) helps data scientists quickly understand key information in large amounts of text for efficient data exploration and cleaning. Summaries help explain model behaviors, speed up report writing, and improve text data analysis.
- **Text generation:** from language models helps data scientists by augmenting training data, generating code, explaining models, and drafting content. AI assistants act as natural language interfaces to query data and models interactively. This course teaches architectures to generate better and highly relevant summaries. These techniques include Amazon Bedrock, LangChain, and Retrieval Augmented Generation (RAG) with persistent embeddings for contextual awareness and key information retention.
- **Question answering systems:** automate tedious data tasks, like documentation reading. They provide insights by answering analytical questions, generate code snippets, and summarize documents. RAG AI assistants can query knowledge bases interactively and generate contextual answers on demand. This course teaches how to build question answering systems and RAG AI assistants.
- **Agents:** understands natural language user requests, break down complex tasks into API calls and data lookups, maintain conversation context, and take actions to fulfill requests. The service orchestrates prompt engineering with company-specific or domain-specific information and provides natural language responses. Amazon Bedrock Agents handles infrastructure, monitoring, encryption, permissions, and invocation management without custom code. Amazon Bedrock Agents integrates with Amazon Bedrock Guardrails to prevent unwanted behavior from model responses or user messages. This course explains how you can use Amazon

Bedrock Agents to synthesize and manage generative AI workflows. It also explains how to use Amazon Bedrock Agents to accelerate generative AI application development.

Quiz:

1. Amazon Bedrock offers several natural language processing (NLP) capabilities that can assist data scientists in their work. Which of the following NLP applications can be implemented with Amazon Bedrock?
   a. Amazon Bedrock offers several NLP capabilities that can assist data scientists in their work. These capabilities include **text summarization**, **text generation**, and **question answering systems**.
2. Using Amazon Bedrock, data scientists can design innovative generative artificial intelligence (generative AI) solutions to address real business problems that directly impact end users. What are some examples of generative AI solutions?
   a. By using generative AI, data scientists can focus on architecting innovative solutions, such as AI assistants, supply chain optimizers, and personalized recommendation systems. They can focus on architecting solutions such as **AI assistants**, **supply chain optimizers**, and **personalized recommendation systems**.

Module 2: Foundation Models

Objectives	Topics
In this module, you will learn how to do the following: • Identify the foundation models available with Amazon Bedrock. • Describe how to control the inference parameters to tune desired output. • Describe how to use APIs to invoke foundation models or create customization jobs. • Identify monitoring and logging capabilities and tools for governance and audit requirements.	The module is organized into the following topics: • Introduction to Amazon Bedrock Foundation Models • Using Amazon Bedrock FMs for Inference • Amazon Bedrock Methods • Data Protection and Auditability

Amazon Bedrock offers a wide choice of high-performing foundation models (FMs) from leading artificial intelligence (AI) startups and Amazon. Each of these FMs cater to different generative artificial intelligence (generative AI) use cases, such as summarization, language translation, coding, and image generation.

Company	Foundation model	Description
Amazon	**Amazon Titan**	Family of models built by Amazon that are pretrained on large datasets, which makes them powerful, general-purpose models.
AI2I Labs	**Jurassic-2** **Jumba**	Multilingual large language models (LLMs) for text generation in Spanish, French, German, Portuguese, Italian, and Dutch.
Anthropic	**Claude 3**	Claude 3 models offer increasingly powerful performance, allowing users to select the optimal balance of intelligence, speed, and cost for their specific application.
Cohere	**Command and Embed**	Text generation model for business applications and embeddings model for

		search, clustering, or classification in more than 100 languages.
Meta	**Llama**	Llama models for are particularly useful for applications such as chat interfaces, virtual assistants, and language translation.
Mistral AI	**Mistral/Mixtral**	Models for Synthetic Text Generation, Code Generation, RAG, or Agents
Stability AI	**Stable Diffusion**	Text-to-image model for generation of unique, realistic, high-quality images, art, logos, and designs. Now available: Stable Diffusion XL (SDXL) 1.0

Inference parameters

When interacting with an FM, you can configure the inference parameters to customize the FM's response. Generally, you should only adjust one parameter at a time, and the results can vary depending on the FM. The following parameters can be used to modify the output from the LLMs (not all parameters are available with all LLMs).

Randomness and diversity

Foundation models typically support the following parameters to control randomness and diversity in the response.

- **Temperature:** controls randomness in word choice. Lower values lead to more predictable responses. The following table lists minimum, maximum, and default values for the temperature parameter.

Parameter	JSON Field Format	Minimum	Maximum	Default
Temperature	temperature	0	1	0

- **Top K:** limits word choices to the K most probable options. Lower values reduce unusual responses.
- **Top P:** cuts off low probability word choices based on cumulative probability. It tightens overall response distribution. The following table lists minimum, maximum, and default values for the Top P parameter.

Parameter	JSON Field Format	Minimum	Maximum	Default
Top P	topP	0	1	1

Length

Foundation models typically support the following parameters to control the length of the generated response.

- **Response length:** sets minimum and maximum token counts. It sets a hard limit on response size. The following table lists minimum, maximum, and default values for the response length parameter. Maximum response length is dependent on the specific FM.

Parameter	JSON Field Format	Minimum	Maximum	Default
Response length	maxTokenCount	0	8,000	512

- **Length penalty:** encourages more concise responses by penalizing longer ones. It sets a soft limit on size.

- **Stop sequences:** include specific character combinations that signal the model to stop generating tokens when encountered. It is used for the early termination of responses.

Working with Amazon Bedrock FMs

Some inference parameters are common across most models, such as temperature, Top P, Top K, and response length. You will dive deep into unique model-specific parameters and I/O configuration you can tune to achieve the desired output based on the use case.

Amazon Titan foundation models

Amazon Titan models are Amazon foundation models. Amazon offers the Amazon Titan Text model and the Amazon Titan Embeddings model through Amazon Bedrock.

Amazon Titan models support the following unique inference parameters in addition to temperature, Top P, and response length, which are common parameters across multiple models.

Stop sequences: with stop sequences **(stopSequences)**, you can specify character sequences to indicate where the model should stop. Use the pipe symbol **(|)** to separate different sequences (maximum 20 characters).

Amazon Titan Text: is a generative LLM for tasks such as summarization, text generation, classification, open-ended question answering, and information extraction. The text generation model is trained on many different programming languages and Rich Text Format (RTF), like tables, JSON, comma-separated values (CSV), and others. The following example shows an input configuration used to invoke a response from Amazon Titan Text using Amazon Bedrock. You can pass input configuration parameters along with an input prompt to the model.

- **Input**

```
{
    "inputText": "<prompt>",
    "textGenerationConfig" : {
        "maxTokenCount": 512,
        "stopSequences": [],
        "temperature": 0.1,
        "topP": 0.9
    }
}
```

The following example shows the output from Amazon Titan Text for the input supplied in the previous code block. The model returns the output along with parameters, such as the number of input and output tokens generated, along with the reason the response finished being generated.

- **Output**

```
{
    "inputTextTokenCount": 613,
    "results": [{
        "tokenCount": 219,
        "outputText": "<output>",
"completionReason" : "string"
    }]
}
```

Amazon Titan Text Embeddings: translates text inputs (words and phrases) into numerical representations (embeddings). Applications of this model include personalization and search. Comparing embeddings produces more relevant and contextual responses than word

matching. The following example demonstrates how you can create embeddings vectors from prompts using the Amazon Titan Embeddings model V2.

- **Input**

```
{
    body = json.dumps({"inputText": <prompt>,
"dimensions": <256   512   1024>,
"normalize": True   False,
})
    model_id = 'amazon.titan-embed-text-v2:0'
    accept = 'application/json'
    content_type = 'application/json'

    response = bedrock_runtime.invoke_model(
        body=body,
        modelId=model_id,
        accept=accept,
        contentType=content_type )

    response_body = json.loads(response['body'].read())
    embedding = response_body.get('embedding')
}
```

This will generate an embeddings vector consisting of numbers that look like the following output.

- **Output**

```
[0.82421875, -0.6953125, -0.115722656, 0.87890625, 0.05883789, -0.020385742,
0.32421875, -0.00078201294, -0.40234375, 0.44140625, ...]
```

Amazon Titan Multimodal Embeddings: is used for use cases like searching images by text, by image for similarity, or by a combination of text and image. It translates the input image or text into an embedding that contain the semantic meaning of both the image and text in the same semantic space. The following example demonstrates how you can create embeddings vectors from a prompt and image using the Amazon Titan Multimodal Embeddings G1.

- **Input**

```
{
   body = json.dumps({"inputText": <prompt>,
      "inputImage": <image>,
      "embeddingConfig":  { "outputEmbeddingLength":  <output_embedding_length> }
})
   model_id = 'amazon.titan-embed-image-v1'
   accept = 'application/json'
   content_type = 'application/json'

   response = bedrock_runtime.invoke_model(
       body=body,
       modelId=model_id,
       accept=accept,
       contentType=content_type )

   response_body = json.loads(response.get('body').read())
   embedding = response_body.get('message')
```

```
}
```

Amazon Titan Image Generator: is an image generation model. It comes in two versions v1 and v2.

- **With Amazon Titan Image Generator v1**, users can create images that match their text-based descriptions by inputting natural language prompts. They can upload and edit existing images, apply text-based prompts, or edit specific parts of an image using an image mask. The model also supports **outpainting**, which extends the boundaries of an image. **Inpainting** is used to fill in missing image areas.
- **Amazon Titan Image Generator v2** supports all the existing features of Titan Image Generator v1 with additional capabilities. It allows users to leverage reference images to guide image generation. The output image will then align with the layout and composition of the reference image while still following the textual prompt. It also includes an automatic background removal feature to remove backgrounds from images containing multiple objects.

AI21 Jurassic-2 (Mid and Ultra)

Common parameters for Jurassic-2 models include temperature, Top P, and stop sequences. Jurassic-2 models support the following unique parameters to control randomness, diversity, length, or repetition in the response:

Length

- **Max completion length (maxTokens):** specify the maximum number of tokens to use in the generated response.
- **Stop sequences (stopSequences):** configure stop sequences that the model recognizes and after which it stops generating further tokens.

Repetitions

- **Presence penalty (presencePenalty):** use a higher value to lower the probability of generating new tokens that already appear at least once in the prompt or in the completion.
- **Count penalty (countPenalty):** use a higher value to lower the probability of generating new tokens that already appear at least once in the prompt or in the completion. The value is proportional to the number of token appearances.
- **Frequency penalty (frequencyPenalty):** use a higher value to lower the probability of generating new tokens that already appear at least once in the prompt or in the completion. The value is proportional to the frequency of the token appearances (normalized to text length).
- **Penalize special tokens:** reduce the probability of repetition of special characters. The default values are true as follows:
  - **Whitespaces (applyToWhitespaces):** a **true** value applies the penalty to white spaces and new lines.
  - **Punctuations (applyToPunctuation):** a **true** value applies the penalty to punctuation.
  - **Numbers (applyToNumbers):** a **true** value applies the penalty to numbers.
  - **Stop words (applyToStopwords):** a **true** value applies the penalty to stop words.
  - **Emojis (applyToEmojis)**: a **true** value excludes emojis from the penalty.

Jurassic-2 Mid: this is a mid-sized model that is optimized to follow natural language instructions and context, so there is no need to provide it with any examples. It is ideal for composing human-like text and solving complex language tasks, such as question answering, and summarization.

Jurassic-2 Ultra: is a large-sized model that you can apply to language comprehension or generation tasks. Use cases include generating marketing copy, powering AI assistantsassisting with creative writing, performing summarization, and extracting information.

- **Input**

```
{
   "prompt": "<prompt>",
   "maxTokens": 200,
   "temperature": 0.5,
   "topP": 0.5,
```

```
    "stopSequences": [],
    "countPenalty": {"scale": 0.0},
    "presencePenalty": {"scale": 0.0},
    "frequencyPenalty": {"scale": 0.0}
}
```

- **Output**

```
{
    "id": 1234,
    "prompt": {
        "text": "<prompt>",
        "tokens": [
            {
                "generatedToken": {
                    "token": "\u2581who\u2581is",
                    "logprob": -12.980147361755371,
                    "raw_logprob": -12.980147361755371
                },
                "topTokens": null,
                "textRange": {"start": 0, "end": 6}
            },
            //...
        ]
    },
    "completions": [
        {
            "data": {
                "text": "<output>",
                "tokens": [
                    {
                        "generatedToken": {
                            "token": "<|newline|>",
                            "logprob": 0.0,
                            "raw_logprob": -0.01293118204921484
                        },
                        "topTokens": null,
                        "textRange": {"start": 0, "end": 1}
                    },
                    //...
                ]
            },
            "finishReason": {"reason": "endoftext"}
        }
    ]
}
```

Jamba-Instruct: offers a 256K context window for text generation, summarization, and question answering tasks for the enterprise. To call the AI21 Labs Jamba-Instruct model, you can use either ***invoke_model*** or ***converse*** API.

- **Input with *invoke_model***

```
response = bedrock.invoke_model(
modelId='ai21.jamba-instruct-v1:0',
body=json.dumps({
'messages': [
{ 'role': 'user',
'content': <prompt>
}
],
})
)
```

- **Input with *converse***

```
response = bedrock.converse(
modelId='ai21.jamba-instruct-v1:0',
'messages': [
{ 'role': 'user',
'content': [{'text':<prompt>}]
}
],
)
```

Anthropic Claude

Anthropic Claude 2 and Claude 3 are additional models available for text generation on Amazon Bedrock. Claude is a generative AI model by Anthropic. It is purpose built for conversations, summarization, question answering, workflow automation, coding, and more. It supports everything from sophisticated dialogue and creative content generation to detailed instruction following.

You can use Amazon Bedrock to send Anthropic Claude Text Completions API or Anthropic Claude Messages API inference requests. You use the messages API to create conversational applications, such as a virtual assistant or a coaching application. Use the text completion API for single-turn text generation applications. For example, generating text for a blog post or summarizing text that a user supplies.

Claude uses common parameters, such as temperature, Top P, Top K, and stop sequences. In addition, Claude models use the following unique parameter to further tune the response output.

- **Maximum length (max_tokens_to_sample)**: specify the maximum number of tokens to use in the generated response.
-

Using Messages API

You can use the **Messages API** to create AI assistant applications. The API manages the conversational exchanges between a user and an Anthropic Claude model (assistant). In Anthropic Claude Messages API, each input message must be an object with a role and content.

Here is an example with a single user message:

```
[{"role": "user", "content": "Hello, Claude"}]
```

The following example shows an input configuration used to invoke a response from Anthropic Claude 3 using Amazon Bedrock with multiple conversational turns:

- **Input**

```
[  {"role": "user", "content": "Hello there."},
  {"role": "assistant", "content": "Hi, I'm Claude. How can I help you?"},
  {"role": "user", "content": "Can you explain LLMs in plain English?"},
]
```

- **Output**

```
{
  "id": "<identifier>",
  "type": "message",
  "role": "assistant",
  "model": "claude-3-sonnet-20240229",
  "content": [
     {
        "type": "text",
        "text": " Sure, I'll try to explain …"
     }
  ],
  "stop_reason": "end_turn",
  "stop_sequence": null,
  "usage": {
     "input_tokens": <token count>,
     "output_tokens": <token count>
  }
}
```

Using Converse API

The Converse API provides a unified set of parameters that work across all models that support messages.

Stability AI (SDXL)

This is a text-to-image model used to generate detailed images. SDXL includes support for the following types of image creation:

- **Image-to-image prompting:** this involves inputting one image to get variations of that image.
- **Inpainting:** this involves reconstructing the missing parts of an image.
- **Outpainting:** this involves constructing a seamless extension of an existing image.

Stability AI Diffusion models support the following controls:

- **Prompt strength (cfg_scale):** this control determines how much the final image portrays the prompt. Use a lower number to increase randomness in the generation.
- **Generation step (steps):** this control determines how many times the image is sampled. More steps can result in a more accurate result.
- **Seed (seed):** this control determines the initial noise setting. Use the same seed and the same settings as a previous run so inference can create a similar image. If you don't set this value, it is set as a random number.

The following example shows an input configuration used to invoke a response from SDXL using Amazon Bedrock.

- **Input**

```
{       "text_prompts": [
         {
```

```
            "text": string,
            "weight": float
        }
    ],
    "height": int,
    "width": int,
    "cfg_scale": float,
    "clip_guidance_preset": string,
    "sampler": string,
    "samples",
    "seed": int,
    "steps": int,
    "style_preset": string,
    "extras" :JSON object

}
```

- **Output**

```
{
    "result": "success",
    "artifacts": [
        {
            "seed": 123,
            "base64": "<image in base64>",
            "finishReason": "SUCCESS"
        },
        //...
    ]
}
```

Cohere Command

Command is the flagship text generation model by Cohere. It is trained to follow user commands and be useful instantly in practical business applications, such as summarization, copywriting, dialogue, extraction, and question answering. Optimized for business priorities, Cohere is System and Organizations Control (SOC) 2 compliant and emphasizes security, privacy, and responsible AI.

In addition to temperature, Top P, Top K, maximum length, and stop sequences, the Cohere Command model supports the following unique controls:

- **Return likelihoods (return_likelihoods):** specify how and if the token likelihoods are returned with the response. You can specify the following options:
  a. **GENERATION:** this option only returns likelihoods for generated tokens.
  b. **ALL:** this option returns likelihoods for all tokens.
  c. **NONE:** this option doesn't return any likelihoods. This is the default option.
- **Stream (stream):** specify *true* to return the response piece by piece in real time and *false* to return the complete response after the process finishes.

The following example shows an input configuration used to invoke a response from Cohere Command using Amazon Bedrock.

- **Input**

```
{
"prompt": "string",
"temperature": float,
"p": float,
"k": float,
"max_tokens": int,
"stop_sequences": ["string"],
"return_likelihoods": "GENERATION|ALL|NONE",
"stream": boolean,
"num_generations": int
}
```

Module 3: Application Components

Objectives	Topics
In this module, you will learn how to do the following: • Describe the components of a generative AI application. • Work with embeddings and vector databases. • Customize a foundation model using RAG and model fine-tuning.	The module is organized into the following topics: • Overview of Generative AI Application Components • Foundation Models and the FM Interface • Working with Datasets and Embeddings • Additional Application Components • RAG

• Describe the text generation and text summarization architecture patterns. • Explain how to use the question answering pattern for generative AI applications. • Describe how the AI assistant pattern is used to enhance the user experience.	• Model Fine-Tuning • Securing Generative AI Applications • Generative AI Application Architecture

Module 4: Using LangChain

Objectives	Topics
In this module, you will learn how to do the following: • Identify common challenges practitioners face when developing LLMs. • Describe the benefits of using LangChain for training LLMs. • Integrate LangChain with LLMs, prompt templates, chains, chat models, text embeddings models, document loaders, retrievers, and agents in Amazon Bedrock. • Use LangChain agents to manage external resources.	The module is organized into the following topics: • Optimizing LLM Performance • Integrating AWS and LangChain • Using Models with LangChain • Constructing Prompts • Structuring Documents with Indexes • Storing and Retrieving Data with Memory • Using Chains to Sequence Components • Managing External Resources with LangChain Agents

Demo 1: Explore Generative AI Use Cases using LangChain and Amazon Bedrock

Objectives	Topics
In this lab/demo, you will explore several generative AI use cases using Amazon Bedrock API and AWS-LangChain integration. These examples can be modified to build practical applications. You will employ the architecture patterns from Module 3 to build these applications.	The module is organized into the following topics: • Introduction to Labs • Task 1: Performing Text Generation • Task 2: Creating Text Summarization • Task 3: Using Amazon Bedrock for Question Answering • Task 4: Building a Chatbot • Task 5: Using Amazon Bedrock Models for Code Generation • Task 6: Integrating Amazon Bedrock Models with LangChain Agents

Module 5: Using Knowledge Bases

Objectives	Topics
In this module, you will learn how to do the following: • Identify Retrieval-Augmented Generation (RAG) applications and use cases. • Explore RAG Architecture.	The module is organized into the following topics: • Retrieval-Augmented Generation (RAG) overview, use cases, architecture, and challenges while building RAG applications. • Amazon Bedrock Knowledge Bases

• Understand the challenges that come with building RAG applications. • Explore Amazon Bedrock Knowledge Bases. • Recognize best practices and advanced techniques for RAG.	• Data Ingestion into Knowledge Bases • Fully managed and customized RAG using Amazon Bedrock Knowledge Bases, • Evaluating RAG applications. • Best practices and advanced techniques for RAG.

Demo 2: Build and Evaluate Retrieval Augmented Generation (RAG) Applications Using Amazon Bedrock Knowledge Bases

Objectives	Topics
In this lab/demo, you will build a question-answering application using the AnyCompany knowledge base and Amazon Bedrock's RetrieveAndGenerate API. You leverage the existing knowledge base, which contains comprehensive information about AnyCompany's products, services, corporate details like history, leadership, financial performance, sustainability efforts, and more. You run through various notebooks that can effectively answer questions related to AnyCompany's products, services, and corporate information.	The module is organized into the following topics: • Set up the environment • Task 1: Leverage a fully-managed RAG application with Amazon Bedrock's RetrieveAndGenerate API method. • Task 2: Build a Q&A application using Amazon Bedrock Knowledge Bases with Retrieve API method. • Task 3: Test the Query Reformulation process supported by Amazon Bedrock Knowledge Bases. • Task 4: Build and evaluate Q&A Application using Amazon Bedrock Knowledge Bases using RAG Assessment (RAGAS) framework. • Task 5: Test the guardrail functionality on Amazon Bedrock Knowledge Base using RetrieveAndGenerate API method.

Module 6: Using Agents

Objectives	Topics
In this module, you will learn how to do the following: • Understand the concept of Amazon Bedrock Agents. • Describe different use cases for agents. • Describe how Amazon Bedrock Knowledge Bases and Amazon Bedrock Guardrails work in conjunction with agents. • Understand how to create and deploy agents using different methods. • Understand how to create and invoke action groups.	The module is organized into the following topics: • Introduction to agents • Use cases for agents • Overview of Amazon Bedrock Agents • Creating and deploying agents • Agent action groups • Deploying and invoking agents

Demo 3: Explore Amazon Bedrock Agents integrated with Amazon Bedrock Knowledge Bases and Amazon Bedrock Guardrails

Objectives	Topics
In this lab/demo, you use a shopping assistant agent to answer questions about lawn maintenance products from two companies.	The module is organized into the following topics: • Task 1: Configure the Agent's Short-Term Memory

You will use the console to study and test the prebuilt knowledge base, guardrails, and the agent application.	• Task 2: Setup the SageMaker Studio Environment • Task 3: Test and Trace the Agent Steps With and Without Amazon Guardrails