Curso Building Generative AI Applications Using Amazon Bedrock''

Explorando o poder da IA Generativa

A inteligência artificial generativa permite diversas aplicações, como:

- **Resumo de texto: condensação automática de conteúdos;**
- **Geração de texto: criação de textos originais com base em instruções;**
- **Sistemas de resposta a perguntas: fornecimento de respostas diretas a partir de dados e contexto.**

Amazon Bedrock Agents é uma solução da AWS que potencializa essas aplicações com recursos avançados. Ele:

- **Compreende solicitações feitas em linguagem natural;**
- **Divide tarefas complexas em etapas automatizadas, como chamadas de API e consultas a bancos de dados;**
- **Mantém o contexto da conversa, garantindo interações coesas e contínuas;**
- **Executa ações automaticamente para atender às demandas do usuário;**
- **Gerencia toda a infraestrutura necessária, como:**
  - **Monitoramento**
  - **Criptografia**
  - **Permissões**
  - **Execução das funções**

Tudo isso sem exigir código personalizado.

Além disso, o sistema integra-se com o Amazon Bedrock Guardrails, que atua como um mecanismo de segurança para:

- **Evitar comportamentos indesejados dos modelos de IA**
- **Filtrar mensagens inapropriadas dos usuários**

Modelos de Fundação (Foundation Models - FMs)

O **Amazon Bedrock** oferece acesso a uma variedade de **Modelos de Fundação (FMs)** desenvolvidos por diferentes empresas líderes em IA. Esses modelos são utilizados em tarefas como geração de texto, tradução, criação de imagens, respostas a perguntas e muito mais.

Modelos disponíveis no Amazon Bedrock

Empresa	Modelo	Descrição
Amazon	Amazon Titan	Modelos versáteis, pré-treinados em grandes volumes

		de dados. Excelentes para aplicações gerais.
AI21 Labs	Jurassic-2, Jamba	LLMs multilíngues para geração de texto em idiomas como português, espanhol, francês, alemão, italiano e holandês.
Anthropic	Claude 3	Modelos potentes com equilíbrio ajustável entre **inteligência**, **velocidade** e **custo**.
Cohere	Command & Embed	• **Command**: Geração de texto para uso empresarial. • **Embed**: Para busca, classificação e agrupamento em 100+ idiomas.
Meta (Facebook)	LLaMA (Chamadas)	Otimizado para **chatbots**, **assistentes virtuais** e **tradução de idiomas**.
Mistral AI	Mistral / Mixtral	Especializados em **texto sintético**, **geração de código**, **RAG** (busca aumentada por recuperação) e agentes autônomos.
Stability AI	Stable Diffusion (SDXL 1.0)	Modelo **texto-para-imagem** usado para gerar imagens, arte, logotipos e designs com alta fidelidade e criatividade.

Parâmetros de Inferência: Ajustes de Geração

Estes parâmetros controlam **como** os modelos geram texto ou imagens. A escolha adequada dos valores impacta diretamente a **criatividade**, **clareza** e **previsibilidade** das respostas.

1. Temperatura (`temperature`)

- Controla a **aleatoriedade** das respostas.
- **Valores**:
  - Mínimo: `0` (mais previsível)
  - Máximo: `1` (mais criativo)
  - Padrão: `0`

2. Top K (`top_k`)

- Limita a escolha às **K palavras mais prováveis**.
- Reduz a chance de respostas incomuns.
- **Quanto menor o valor**, mais previsível é a saída.

3. Top P (`top_p`)

- Corta as palavras menos prováveis com base na **probabilidade cumulativa**.
- **Valores**:
  - Mínimo: `0`
  - Máximo: `1`
  - Padrão: `1`

4. Comprimento da Resposta

- Define o **número de tokens** que a resposta pode ter.
- **Valores**:
  - Mínimo: `0`
  - Máximo: `8.000` (dependente do modelo)
  - Padrão: `512`

5. Penalidade de Comprimento

- Penaliza respostas muito longas, incentivando **respostas mais concisas**.

6. Sequências de Parada

- Sequências específicas de texto que indicam ao modelo **quando parar** de gerar resposta.
- Úteis para **evitar respostas excessivamente longas** ou fora de contexto.

Usando Amazon Bedrock FMs para inferência:

Modelos de Fundação: Amazon Titan (via Amazon Bedrock)

Os modelos **Amazon Titan** são modelos de fundação desenvolvidos pela Amazon e disponíveis na plataforma **Amazon Bedrock**. Atualmente, há quatro principais categorias de modelos Titan:

---

1. Amazon Titan Text

Trata-se de um **modelo generativo de linguagem natural** (LLM) voltado para diversas tarefas de NLP, como:

- Resumo automático;
- Geração e continuação de texto;
- Classificação;
- Respostas a perguntas abertas;
- Extração de informações estruturadas (como JSON, tabelas e CSVs).

Além dos parâmetros comuns como **temperatura**, **Top P** e **comprimento da resposta**, o modelo suporta:

- **Sequências de parada (stopSequences)**: permitem especificar pontos em que o modelo deve encerrar a resposta, com até 20 caracteres por sequência.

Exemplo apresentado: configuração básica de entrada com um prompt, parâmetros de temperatura e comprimento, e uma resposta com número de tokens gerados e motivo do encerramento da geração.

---

2. Amazon Titan Embeddings

Este modelo transforma **palavras e frases em vetores numéricos** (embeddings), permitindo aplicações como:

- Personalização de conteúdo;

- Busca semântica;
- Análise de similaridade textual.

É possível configurar a **dimensão do vetor** (por exemplo, 256, 512 ou 1024) e ativar ou não a **normalização** dos valores.

Exemplo apresentado: criação de um vetor de embeddings a partir de um texto de entrada, com definição de parâmetros como dimensão e normalização. O modelo retorna um vetor com valores numéricos que representam semanticamente o conteúdo.

---

3. Amazon Titan Multimodal Embeddings

Este modelo permite gerar embeddings a partir de **texto, imagem ou uma combinação de ambos**. Ele é ideal para casos como:

- Busca de imagens por descrição textual;
- Comparação de similaridade entre imagens;
- Recuperação de conteúdo multimodal (texto + imagem).

Exemplo apresentado: envio de um texto e uma imagem como entrada, com configuração do comprimento do vetor de saída. O modelo retorna o vetor que representa semanticamente o conteúdo combinado.

---

4. Amazon Titan Image Generator

Este modelo permite a **geração e edição de imagens** a partir de descrições textuais. Ele está disponível em duas versões:

- **Versão 1 (v1)**:
  - Geração de imagens a partir de prompts;
  - Edição de imagens existentes;
  - Uso de máscaras para edição seletiva;
  - Recursos como *inpainting* (preenchimento de áreas ausentes) e *outpainting* (extensão dos limites da imagem).
- **Versão 2 (v2)**:
  - Inclui todas as funcionalidades da versão 1;
  - Adiciona suporte ao uso de **imagens de referência** para orientar a composição da nova imagem;
  - Suporte a **remoção automática de plano de fundo** em imagens com múltiplos objetos.

Exemplo apresentado: envio de prompt textual e imagem (quando aplicável), recebendo como saída uma imagem gerada que respeita a composição e o conteúdo definido pelo usuário.

Modelos AI21 Jurassic-2 (Mid e Ultra)

Disponíveis via **Amazon Bedrock**, os modelos **Jurassic-2**, da AI21 Labs, são voltados para **geração e compreensão de linguagem natural**. Suportam parâmetros clássicos de inferência, como temperatura, Top P e sequências de parada, e oferecem **controles avançados de repetição e diversidade textual**.

Parâmetros exclusivos dos modelos Jurassic-2

- **Comprimento da resposta:**
  - **maxTokens**: Define o número máximo de tokens que o modelo pode gerar.
- **Sequências de parada:**
  - **stopSequences**: Permite configurar até onde o modelo deve continuar a geração.
- **Controle de repetições:**
  - **Penalidade de presença (presencePenalty)**: Reduz a chance de repetir tokens já utilizados, com base apenas na presença.

- **Penalidade de contagem (countPenalty)**: Penaliza tokens com base no número de ocorrências.
- **Penalidade de frequência (frequencyPenalty)**: Penaliza tokens com base na frequência relativa de aparição.

- **Penalização de tokens especiais (opcional):**
  - **Espaços em branco**
  - **Pontuações**
  - **Números**
  - **Palavras irrelevantes (stopwords)**
  - **Emojis** (estes são excluídos da penalização por padrão)

Jurassic-2 Mid

- Modelo de médio porte.
- Ideal para tarefas em que se deseja seguir instruções de forma direta e clara.
- Bom desempenho na geração de texto natural e compreensão de linguagem.
- Recomendado para tarefas como:
  - Resposta a perguntas;
  - Composição de textos;
  - Resumos rápidos;
  - Compreensão geral de linguagem.

Jurassic-2 Ultra

- Modelo de grande porte com maior capacidade de generalização.
- Adequado para tarefas mais complexas, como:
  - Escrita criativa;
  - Geração de conteúdo de marketing;
  - Assistência em tarefas de IA conversacional;
  - Extração de informações específicas;
  - Resumos detalhados.

Jamba-Instruct (AI21)

O **Jamba-Instruct** é um modelo da AI21 com uma **janela de contexto ampliada de 256.000 tokens**, o que o torna ideal para:

- Processamento de documentos extensos;
- Resumos longos;
- Geração de texto em projetos de grande escala;
- Tarefas de QA (question answering) com longos trechos de referência.

Pode ser invocado de duas formas:

- **Via** `invoke_model`: método direto, tradicional em aplicações.
- **Via** `converse`: modo conversacional, ideal para interações dinâmicas com múltiplas mensagens.

Anthropic Claude (Claude 2 e Claude 3)

Os modelos **Claude**, desenvolvidos pela **Anthropic**, são **modelos de linguagem natural generativos** especialmente projetados para:

- Diálogos e conversações sofisticadas;
- Resumos longos e objetivos;
- Respostas a perguntas;
- Automação de fluxos de trabalho;
- Codificação e geração de código;
- Geração de conteúdo criativo.

Esses modelos estão disponíveis na plataforma **Amazon Bedrock** e oferecem duas interfaces principais:

---

1. Claude Text Completions API

Essa API é ideal para **aplicações de turno único** — ou seja, quando se deseja uma resposta direta a um prompt, como:

- Geração de texto criativo (ex: uma postagem de blog);
- Resumo de um trecho textual;
- Traduções e reescritas;
- Classificações e análises de conteúdo.

Parâmetros disponíveis incluem:

- Temperatura;
- Top P;
- Top K;
- Sequências de parada;
- **max_tokens_to_sample** (parâmetro exclusivo Claude): define o número máximo de tokens na resposta.

---

2. Claude Messages API

Essa API é voltada para **aplicações de múltiplos turnos**, como:

- Assistentes virtuais;
- Chatbots personalizados;
- Aplicações de coaching ou tutoria.

A API organiza a conversa como uma lista de mensagens com os papéis: **usuário**, **assistente** e, eventualmente, **sistema**. Cada mensagem contém o conteúdo textual trocado em cada turno.

Exemplo de uso: Uma conversa típica incluiria:

- Uma saudação inicial do usuário.
- Uma resposta cortês do assistente (Claude).
- Uma pergunta subsequente do usuário.
- Uma explicação clara do assistente.

A resposta do modelo inclui:

- Identificador único da conversa;
- Indicação do modelo Claude utilizado (ex: Claude 3 Sonnet);
- A mensagem gerada;
- A razão de parada (ex: fim do turno);
- Uso de tokens de entrada e saída.

---

3. API Converse (opcional)

A **API Converse** funciona como uma camada unificada para todos os modelos compatíveis com o formato de mensagens. Ela aceita os mesmos parâmetros da Claude Messages API e facilita a integração entre modelos diferentes mantendo a mesma estrutura de mensagens.

Conclusão

Os modelos **Claude 2 e 3** são opções robustas e versáteis para diversas aplicações em linguagem natural, tanto para interações pontuais quanto para conversas contínuas. Seu ponto forte está na **clareza, consistência e segurança das respostas**, com foco em **assistência confiável e ética** em ambientes corporativos e criativos.

Stability AI - SDXL (Stable Diffusion XL)

O **SDXL** é um modelo **texto-para-imagem** de última geração, desenvolvido pela Stability AI. Ele permite a **criação de imagens altamente detalhadas e personalizadas a partir de descrições textuais**, com suporte a múltiplas técnicas de manipulação visual.

Funcionalidades Suportadas

1. **Geração de imagem a partir de texto (prompt textual)**
   - Cria uma imagem original com base em uma descrição fornecida pelo usuário.
2. **Imagem para imagem**
   - Gera variações de uma imagem fornecida como entrada, mantendo o estilo ou elementos principais.
3. **Repintura (Inpainting)**
   - Reconstrói ou preenche partes faltantes de uma imagem, útil para restauração ou substituição seletiva de elementos.
4. **Pintura externa (Outpainting)**
   - Expande a imagem original além de seus limites, criando uma continuidade visual coerente com o conteúdo existente.

Principais Parâmetros de Controle

- **Prompt de texto (text_prompts)**:
  Conjunto de instruções textuais com pesos associados para guiar a geração da imagem. Pode-se fornecer múltiplos prompts, cada um com uma **"força" relativa**.
- **Altura e largura (height / width)**:
  Define as dimensões da imagem de saída, em pixels.
- **Força do prompt (cfg_scale)**:
  Controla o quanto a imagem deve se alinhar fielmente ao texto fornecido.
  - Valores mais altos resultam em **maior fidelidade ao prompt**.
  - Valores mais baixos introduzem **mais liberdade criativa/aleatoriedade**.
- **Etapas de geração (steps)**:
  Define **quantas iterações** de amostragem o modelo executa.
  - Mais etapas geralmente resultam em **maior detalhamento e precisão** visual.
- **Semente (seed)**:
  Determina a configuração inicial de ruído aleatório usada no processo.
  - **Definir a mesma semente** com os mesmos parâmetros permite reproduzir imagens semelhantes em execuções diferentes.
- **Estilo (style_preset)**:
  Permite aplicar estilos visuais predefinidos, como pintura a óleo, fotografia realista, desenho a lápis, entre outros.

- **Sampler e orientação de clip (sampler / clip_guidance_preset)**:
  Técnicas específicas de amostragem e orientação que controlam **como a imagem é otimizada** durante a geração.
- **Extras (extras)**:
  Campo adicional para configurações específicas ou experimentais, fornecidas em formato JSON.

Resultado da Inferência

O modelo retorna um objeto contendo:

- **Confirmação de sucesso** da execução;
- **Imagem gerada em formato codificado (base64)**;
- **Semente utilizada**, permitindo reprodução da imagem em execuções futuras.

Conclusão

O **SDXL** é um modelo poderoso para quem deseja criar, modificar ou expandir imagens com base em texto ou imagens de referência. Ele oferece **controle detalhado sobre a fidelidade, estilo, tamanho e consistência visual**, sendo ideal para aplicações criativas, design conceitual, ilustração automatizada e muito mais.

Stability AI - SDXL (Stable Diffusion XL)

O **SDXL** é um modelo **texto-para-imagem** de última geração, desenvolvido pela Stability AI. Ele permite a **criação de imagens altamente detalhadas e personalizadas a partir de descrições textuais**, com suporte a múltiplas técnicas de manipulação visual.

Funcionalidades Suportadas

1. **Geração de imagem a partir de texto (prompt textual)**
   - Cria uma imagem original com base em uma descrição fornecida pelo usuário.
2. **Imagem para imagem**
   - Gera variações de uma imagem fornecida como entrada, mantendo o estilo ou elementos principais.
3. **Repintura (Inpainting)**
   - Reconstrói ou preenche partes faltantes de uma imagem, útil para restauração ou substituição seletiva de elementos.
4. **Pintura externa (Outpainting)**
   - Expande a imagem original além de seus limites, criando uma continuidade visual coerente com o conteúdo existente.

Principais Parâmetros de Controle

- **Prompt de texto (text_prompts)**:
  Conjunto de instruções textuais com pesos associados para guiar a geração da imagem. Pode-se fornecer múltiplos prompts, cada um com uma **"força" relativa**.
- **Altura e largura (height / width)**:
  Define as dimensões da imagem de saída, em pixels.
- **Força do prompt (cfg_scale)**:
  Controla o quanto a imagem deve se alinhar fielmente ao texto fornecido.
  - Valores mais altos resultam em **maior fidelidade ao prompt**.
  - Valores mais baixos introduzem **mais liberdade criativa/aleatoriedade**.

- **Etapas de geração (steps)**:
  Define **quantas iterações** de amostragem o modelo executa.
  - Mais etapas geralmente resultam em **maior detalhamento e precisão** visual.
- **Semente (seed)**:
  Determina a configuração inicial de ruído aleatório usada no processo.
  - **Definir a mesma semente** com os mesmos parâmetros permite reproduzir imagens semelhantes em execuções diferentes.
- **Estilo (style_preset)**:
  Permite aplicar estilos visuais predefinidos, como pintura a óleo, fotografia realista, desenho a lápis, entre outros.
- **Sampler e orientação de clip (sampler / clip_guidance_preset)**:
  Técnicas específicas de amostragem e orientação que controlam **como a imagem é otimizada** durante a geração.
- **Extras (extras)**:
  Campo adicional para configurações específicas ou experimentais, fornecidas em formato JSON.

Resultado da Inferência

O modelo retorna um objeto contendo:

- **Confirmação de sucesso** da execução;
- **Imagem gerada em formato codificado (base64)**;
- **Semente utilizada**, permitindo reprodução da imagem em execuções futuras.

Conclusão

O **SDXL** é um modelo poderoso para quem deseja criar, modificar ou expandir imagens com base em texto ou imagens de referência. Ele oferece **controle detalhado sobre a fidelidade, estilo, tamanho e consistência visual**, sendo ideal para aplicações criativas, design conceitual, ilustração automatizada e muito mais.

Tabela comparativa:

Modelo	Tipo	Funcionalidade Principal	Parâmetros Exclusivos	Casos de Uso
SDXL (Stability AI)	Texto-para-imagem	Geração e manipulação de imagens detalhadas a partir de prompts de texto ou imagens	• Força do prompt (cfg_scale) • Etapas de geração • Semente (seed) • Estilo visual predefinido (style_preset)	• Geração criativa de imagens • Edição de imagens (inpainting, outpainting)
Claude (Anthropic)	Geração de texto e conversação	Geração de conteúdo criativo, assistentes de IA, respostas a perguntas, diálogos sofisticados	• Comprimento máximo (max_tokens_to_sample) • Parâmetros de controle para	• Assistentes virtuais • Respostas a perguntas • Geração de texto criativo

			conversação (role, content)	
Jurassic-2 (AI21 Labs)	Geração de texto	Respostas a perguntas, resumos, geração criativa e tarefas complexas de linguagem natural	• Penalidade de presença (presencePenalty) • Penalidade de contagem (countPenalty) • Penalidade de frequência (frequencyPenalty)	• Resumos • Geração de conteúdo criativo • Compreensão de linguagem natural
Jamba-Instruct (AI21 Labs)	Geração de texto e instrução	Geração de texto com alta capacidade de compreensão e execução de instruções contextuais	• Janela de contexto (256K tokens)	• Assistentes de IA • Execução de instruções complexas e conversação fluida

Destaques de Cada Modelo:

- **SDXL (Stability AI)**: Focado em **criação e manipulação de imagens**, com a capacidade de gerar ou modificar imagens baseadas em descrições textuais detalhadas, oferecendo um alto grau de controle sobre a estética da imagem gerada.
- **Claude (Anthropic)**: Ideal para **assistentes de IA** e **conversações sofisticadas**, com forte capacidade de gerar **conteúdo criativo** e executar tarefas de **respostas a perguntas** de maneira fluida e envolvente.
- **Jurassic-2 (AI21 Labs)**: Um modelo de **geração de texto** altamente flexível e com **controle detalhado de repetição e diversidade de respostas**, ideal para **tarefas complexas de linguagem** e **geração de conteúdo criativo**.
- **Jamba-Instruct (AI21 Labs)**: Excelente para **instruções detalhadas** e **tarefas contextuais complexas**, com grande capacidade de compreender e gerar texto com base em **contexto extenso** (256K tokens).

Set up and configurações relacionadas a APIs:

1. ListFoundationModels

- *Função: Lista todos os modelos de fundação disponíveis no Amazon Bedrock.*
- *Descrição: Esta API retorna informações sobre todos os modelos de fundação que podem ser usados, incluindo metadados e detalhes de cada modelo.*

2. InvokeModel

- *Função: Invoca um modelo de fundação para realizar inferência com base no prompt fornecido.*
- *Descrição: Permite que você execute inferência com um modelo específico. É usado para gerar texto, como respostas a perguntas ou conteúdo criativo. O modelo é configurado com parâmetros como o número máximo de tokens, temperatura e sequências de parada.*

3. InvokeModelWithResponseStream

- *Função: Invoca um modelo de fundação e recebe a resposta como um fluxo de dados (streaming).*

- *Descrição: Similar ao InvokeModel, mas em vez de esperar pela resposta completa, ela é transmitida em blocos de dados enquanto é gerada. Ideal para casos em que você deseja começar a processar a resposta antes que ela esteja completamente gerada.*

4. Conversar

- *Função: Facilita a comunicação contínua com o modelo, simulando uma conversa.*
- *Descrição: Usada para criar interações em um formato de diálogo, onde as mensagens do usuário e do assistente são enviadas de forma sequencial. Isso é ideal para chatbots ou assistentes virtuais que precisam manter o contexto de conversação.*

Parâmetros Comuns:

- *Temperatura: Controla a aleatoriedade da resposta. Valores mais baixos resultam em respostas mais determinísticas.*
- *Top P: Define a probabilidade cumulativa dos tokens selecionados para gerar a resposta. Valores mais altos aumentam a diversidade na saída.*
- *Max Tokens: Limita o número de tokens na resposta gerada.*
- *Stop Sequences: Define sequências de parada, ou seja, quando o modelo deve interromper a geração da resposta.*

Proteja seus aplicativos de IA generativa

- **Criptografia e Controle de Acesso**:
  - **Criptografia**: Seus Modelos de Fundação (FMs) personalizados são criptografados utilizando as chaves do **AWS Key Management Service (KMS)**, garantindo que os dados sejam armazenados de forma segura.
  - **Controle de Acesso**: O **AWS Identity and Access Management (IAM)** permite definir e controlar o acesso aos seus FMs personalizados. Você pode especificar quais usuários e serviços têm permissão para interagir com os modelos, garantindo que o acesso seja concedido apenas a usuários ou serviços autorizados.
- **Apoio à Governança e Auditoria**:
  - O **Amazon Bedrock** oferece ferramentas de **monitoramento e registro** para suportar requisitos de governança e auditoria. Essas ferramentas permitem acompanhar as ações realizadas no sistema, ajudando a manter a conformidade e a garantir que os processos sejam auditáveis.Componentes de aplicativos:

Componentes Típicos de uma Aplicação de IA Generativa

1. Interface FM (Modelo de Fundação)

- **Definição**: No centro de uma aplicação de IA generativa está o Modelo de Fundação (FM), que é treinado em grandes volumes de dados de diversos domínios. Isso permite que o FM seja adaptado a diferentes tarefas, abrangendo áreas como linguagem, áudio e imagens.
- **Capacidade**: Os FMs, em contraste com os modelos tradicionais de machine learning (ML), podem ser aplicados a uma gama ampla de tarefas devido à sua capacidade de generalização.
- **Subconjunto - LLMs**: Modelos de Linguagem Grande (LLMs) são um tipo específico de FM, treinados em grandes corpora de dados textuais e especializados na geração de linguagem natural.

2. Interface e Prompts

- **Acesso ao FM**: Para interagir com um FM, é necessária uma interface, normalmente uma **API**. Esta API pode ser gerenciada na infraestrutura da AWS ou auto-hospedada em um ambiente de machine learning com suporte a computação acelerada.
- **Prompt**: Um **prompt** é uma entrada de texto ou outro tipo de dados fornecido ao FM para gerar uma resposta. Ele é enviado através de chamadas de API, que geram as inferências desejadas.

3. Parâmetros de Inferência

- **Função**: Juntamente com os prompts, os **parâmetros de inferência** são cruciais para guiar a saída do FM. Esses parâmetros influenciam a qualidade e a relevância da resposta gerada, permitindo a customização para casos específicos.
- **Exemplos de Parâmetros**: Parâmetros comuns incluem **Top P**, **Top K** e **temperatura**, que ajudam a controlar a aleatoriedade, diversidade e criatividade da resposta.
- **Tokens**: Os LLMs operam em tokens, que podem ser palavras, letras ou unidades parciais de palavras. A quantidade de tokens influencia o comprimento da resposta e a precisão da geração de texto.

4. Conjuntos de Dados Empresariais

- **Fontes de Dados**: FMs utilizam dados públicos, mas para oferecer soluções personalizadas para empresas, eles precisam acessar **dados corporativos**. Esses dados incluem documentos internos como manuais, relatórios e apresentações.
- **Desafio com Contexto**: Fornecer grandes volumes de dados pode aumentar os custos e a latência, além de causar problemas de **alucinações** no modelo (informações incorretas geradas com confiança). Para evitar isso, a quantidade de contexto fornecido deve ser bem controlada.
- **Formas de Fornecer Contexto**:
  a. **Documento Específico**: Pode-se passar um documento específico como contexto ao modelo.
  b. **Pesquisa nos Dados**: Usando **embeddings de vetor**, é possível pesquisar em grandes volumes de dados corporativos, retornando apenas as partes relevantes como contexto.

5. Incorporações de Vetores (Embeddings)

- **Definição**: **Embeddings** são representações numéricas de dados textuais, imagens ou áudio em um espaço vetorial. Esses vetores ajudam a capturar a semelhança semântica entre diferentes entradas.
- **Modelos de Embedding**: Através de FMs de embeddings, dados como texto e imagens podem ser transformados em vetores que são armazenados em bancos de dados vetoriais para consultas rápidas.
- **Exemplos de Modelos**:
  - **Amazon Titan Embeddings G1**: Modelo de texto que converte texto em embeddings.
  - **Cohere Embed**: Modelo multimodal que converte texto em embeddings em vários idiomas.
  - **Incorporação Multimodal Titan**: Modelo que suporta tanto texto quanto imagens.

6. Bancos de Dados Vetoriais

- **Função**: **Bancos de dados vetoriais** são usados para armazenar vetores de alta dimensão, representando dados como palavras ou entidades. Eles permitem consultas de **similaridade** rápidas em grandes volumes de dados.
- **Algoritmos de Similaridade**: O algoritmo **k-vizinhos mais próximos (k-NN)** ou a **similaridade de cosseno** são comumente usados para comparar vetores e encontrar os mais próximos.
- **Opções de Banco de Dados**:
  - **Amazon OpenSearch Service** (provisionado e sem servidor).
  - **Extensão pgvector** para **PostgreSQL** no **Amazon RDS** e **Amazon Aurora**.
  - **Pinecone** no **AWS Marketplace** e outras opções open-source como **FAISS** e **Chroma**.

7. Dados Empresariais Vetorizados

- **Processo de Vetorização**: Dados corporativos, como documentos ou imagens, são passados para modelos de incorporação, onde são transformados em vetores. Esses vetores são então armazenados em bancos de dados vetoriais para consultas rápidas e eficientes.
- **Redução de Alucinações**: O uso de dados vetorizados como contexto para o FM ajuda a melhorar a precisão das respostas e reduzir o risco de **alucinações** (informações incorretas geradas pelo modelo).

8. Histórico de Prompts de Armazenamento

- **Função**: Um **armazenamento de histórico de prompts** é essencial para manter a continuidade em conversas **multi-turn** (interações com múltiplas trocas). Esse armazenamento permite que a IA acesse o histórico de interações anteriores e forneça respostas mais contextuais e relevantes.
- **Desafio com Janela de Contexto**: Muitos FMs possuem uma **janela de contexto limitada**, o que significa que somente uma quantidade fixa de dados pode ser processada por vez. O armazenamento de histórico ajuda a contornar essa limitação.
- **Propósitos**:
  - **Auditoria e Conformidade**: Permite o monitoramento das interações para garantir que o modelo esteja em conformidade com as políticas e regulamentações.
  - **Depuração**: Auxilia os desenvolvedores a rastrear e corrigir erros em interações anteriores.

9. Aplicações Web Frontend e Aplicativos Móveis

- **Interação com o Usuário**: Para que os usuários possam interagir com a IA generativa, é necessário um **aplicativo frontend**. Esse aplicativo pode ser uma interface web ou móvel que construa prompts e envie-os para o FM por meio de API.
- **Funções do Frontend**:
  - **Pré-processamento**: O aplicativo pode processar os dados antes de enviá-los ao FM.
  - **Pós-processamento**: Após a resposta do FM, o aplicativo pode formatar ou refinar a saída antes de mostrá-la ao usuário.
  - **Gerenciamento de Falhas**: O frontend também é responsável por lidar com falhas e garantir a melhor experiência para o usuário.

RAG (Recuperação-Complementada com Geração)

1. Definição e Funcionamento

- **Objetivo**: O **RAG** é uma técnica utilizada para construir aplicações de IA generativa que integram fontes de dados empresariais e bancos de dados vetoriais. Seu principal objetivo é superar as limitações de conhecimento dos modelos de linguagem, permitindo que eles acessem informações além dos dados de treinamento originais.
- **Processo**:
  a. **Recuperação de Dados**: O RAG utiliza um módulo recuperador que encontra dados relevantes de fontes externas (como bancos de dados corporativos) com base no prompt do usuário.
  b. **Criação de Prompt Expandido**: As informações recuperadas são combinadas com o prompt original do usuário, criando um **prompt expandido**.
  c. **Geração de Resposta**: Esse prompt expandido é então enviado ao modelo de linguagem, que gera uma resposta incorporando o conhecimento obtido pela recuperação dos dados empresariais.

2. Benefícios do RAG

- **Utilização de Dados Atualizados**: O RAG permite que os modelos de linguagem acessem **informações mais recentes e relevantes**, superando a limitação de depender de dados desatualizados presentes no momento do treinamento do modelo.
- **Aplicabilidade em Ambientes Corporativos**: Essa técnica é especialmente útil em ambientes empresariais onde os dados mudam frequentemente, permitindo que o modelo de linguagem recupere informações atualizadas em vez de se limitar a seu conjunto de dados estático.

3. Limitações do RAG

- **Limitação de Recuperação**: O RAG depende dos dados já presentes nos **bancos de dados vetoriais**. Ou seja, ele só pode recuperar informações que já foram incorporadas ao sistema. Se novos dados empresariais não foram vetorizados ou armazenados, o modelo não poderá acessá-los.
- **Modelo Estático**: O RAG não altera o modelo de linguagem subjacente permanentemente, ou seja, o modelo permanece **estático**. Ele fica mais inteligente apenas temporariamente com base nos dados recuperados, mas não adquire novos conhecimentos de forma contínua ou definitiva.

- **Latência com Janelas de Contexto Grandes**: Ao combinar dados recuperados com o prompt do usuário, o modelo pode ser sobrecarregado com **grandes janelas de contexto**, o que pode causar problemas de **latência**, prejudicando o desempenho.

4. Ajuste Fino do Modelo

- **Necessidade de Ajuste Fino**: Embora o RAG seja útil, ele não resolve todos os problemas, especialmente no que diz respeito à personalização e ao desempenho em tarefas específicas. O ajuste fino do modelo pode ser necessário para melhorar o desempenho em casos mais complexos.
- **Diferenciação entre Ajuste Fino e RAG**:
  - O **ajuste fino** envolve **alterações permanentes** no modelo subjacente, permitindo que o FM aprenda a melhorar em uma tarefa específica, com o auxílio de um conjunto de dados rotulado.
  - O **RAG** oferece uma solução mais temporária, apenas enriquecendo a entrada do modelo com dados recuperados e não alterando o modelo em si.

5. Tipos de Ajuste Fino

Existem duas abordagens principais para o ajuste fino de modelos de linguagem:

1. **Personalização (Aprendizado Baseado em Prompts)**: Refere-se à adaptação do modelo a tarefas específicas, ajustando seu comportamento por meio de novos prompts ou entradas direcionadas.
2. **Pré-Treinamento Contínuo (Adaptação de Domínio)**: Refere-se ao ajuste do modelo com base em novos dados ou em mudanças no domínio, permitindo que o modelo se torne mais adequado a um ambiente ou contexto específico.

Personalização (Aprendizagem Baseada em Prompts)

- **Definição**: O ajuste fino do FM para uma tarefa específica é feito por meio do **aprendizado baseado em prompt**. Neste processo, você direciona o modelo a um conjunto de dados rotulado contendo exemplos que o modelo deve aprender.
- **Funcionamento**:
  - **Exemplos Rotulados**: São formatados como pares de **prompt** e **resposta**, estruturados como instruções.
  - **Modificação dos Pesos**: O ajuste fino altera os **pesos do modelo** para que ele seja mais adequado à tarefa em questão.
  - **Processo Leve**: O ajuste geralmente é **leve**, sendo necessário apenas algumas épocas para ajustar o modelo. Não envolve grandes mudanças no modelo subjacente.
- **Limitação**:
  - O **ajuste fino baseado em prompt** é específico para uma tarefa. Ou seja, ele não pode ser facilmente generalizado para outras tarefas ou contextos fora do seu domínio específico.

Pré-Treinamento Contínuo (Adaptação de Domínio)

- **Definição**: O **ajuste fino de adaptação de domínio** adapta um FM pré-treinado a **várias tarefas específicas** de um domínio usando dados limitados, mas relevantes, para o domínio em questão.
- **Funcionamento**:
  - Você pode ajustar o FM com dados **rotulados ou não rotulados** específicos do domínio. O modelo será ajustado para refletir a **linguagem específica** da sua empresa ou domínio, incluindo termos técnicos.
  - **Adaptabilidade**: O ajuste fino contínuo pode usar **poucos dados** ou aproveitar ao máximo os dados disponíveis para se adaptar melhor ao contexto de um domínio.

Comparação entre RAG e Ajuste Fino

Critério	RAG	Ajuste Fino
Quando Usar	Quando você tem **dados personalizados** e o FM não	Quando os dados são **altamente personalizados ou proprietários** para seu domínio

	tem conhecimento suficiente do domínio.	e o FM tem conhecimento limitado sobre esse domínio. Também pode ser usado para **reduzir a latência**, exigindo menos contexto na consulta.
Esforço para Implementar	Médio	Alto
Custo	Geralmente **baixo a médio**	O custo pode variar de **baixo a alto**, dependendo do **tamanho do modelo**.

Governança e Segurança em Aplicações de IA Generativa

- **Importância**: A **governança** e a **segurança** são fundamentais em cada etapa do desenvolvimento de aplicativos de IA generativa. Elas ajudam a mitigar riscos, garantir supervisão e assegurar a conformidade com os regulamentos.

Práticas Recomendadas:

- **Gerenciamento de Acesso**:
  - **Audite e gerencie o acesso** a todas as partes do aplicativo de IA generativa, incluindo FMs e métodos de API, para controlar quem pode fazer o quê.
- **Monitoramento e Registro**:
  - **Monitore, registre e relate** o acesso ao FM diretamente ou através de abordagens personalizadas, para garantir a **transparência e a responsabilidade**.
- **Conformidade e Explicabilidade**:
  - **Registre solicitações e respostas** do FM, para manter conformidade com os regulamentos e garantir a **explicabilidade** das ações do modelo.
- **Auditoria de Segurança**:
  - **Audite periodicamente** os FMs com dados de teste e simule **ataques de injeção** para garantir que o modelo não tenha consequências não intencionais.
- **Documentação Completa**:
  - **Documente o processo completo** de desenvolvimento e operação do aplicativo, mantendo-a **atualizada** para garantir total transparência e controle sobre todas as facetas da aplicação.

Padrões

Padrão de Geração de Texto ou Código usando Amazon Bedrock

Definição e Casos de Uso

- **Geração de texto** refere-se a qualquer tarefa onde a saída de um modelo é **texto recém-gerado**.
- Pode ser usado para escrever: artigos, blogs, e-mails, livros, poemas etc.
- Também é aplicável à **programação**: geração de código SQL, explicações de código, tradução, correção de bugs e otimizações.

Funcionamento

- O usuário envia um **prompt** para um **Foundation Model (FM)** via o **playground** do Amazon Bedrock ou **API**.
- O FM processa o prompt e retorna a **resposta gerada** com base no conteúdo do prompt.

---

2. Geração de Texto usando LangChain com Amazon Bedrock

Definição

- O **LangChain** é uma biblioteca de código aberto usada para **orquestrar** a interação entre o prompt do usuário e o FM do Amazon Bedrock.
- Serve como **camada intermediária** que organiza chamadas, respostas e fluxo lógico de dados.

Casos de Uso

- Geração de texto personalizada.
- Desenvolvimento de **assistentes de IA conversacionais** mais complexos e dinâmicos.

---

3. Padrão de Resumo de Texto

A abordagem depende do **tamanho do documento** em relação à **janela de contexto** do modelo.

3.1. Documentos Pequenos

- **Quando usar**: O conteúdo do documento **cabe na janela de contexto** do FM.
- **Funcionamento**: O texto é passado diretamente ao FM, que retorna o resumo.

Vantagens:

- Simples e direto.

Limitações:

- Não funciona com textos grandes.

3.2. Documentos Grandes (Map-Reduce + Chunking + Chaining)

Desafios:

- **Limite de tokens** por chamada.
- Riscos de alucinação e consumo excessivo de memória.

Solução (Arquitetura Map-Reduce):

1. **Divisão (Chunking)**: O documento é quebrado em pedaços pequenos (n) com ferramentas como LangChain.
2. **Resumo Parcial**: Cada bloco é resumido individualmente por um FM.
3. **Encadeamento (Chaining)**:
   - O próximo bloco é concatenado ao resumo anterior.
   - O novo conjunto é resumido novamente.
4. **Iteração**: O processo continua até gerar um **resumo final consolidado**.

---

4. Padrão de Resposta a Perguntas (Question Answering)

Objetivo:

- Responder perguntas factuais com **alta precisão**, extraindo informações de fontes estruturadas ou não estruturadas.

4.1. Caso de Uso Base Genérico

Características:

- Baseado apenas no conhecimento do FM.
- Não há personalização ou recuperação de informações externas.

Vantagens:

- Simples de implementar.

Limitações:

- Pode faltar **contexto específico** e precisão, especialmente em áreas como saúde, direito ou finanças.

4.2. Casos de Uso Personalizados e Específicos com RAG

Problema resolvido:

- Limite de tokens/contexto no FM.

Solução (RAG - Retrieval Augmented Generation):

1. **Recuperação**: Identifica e localiza os **blocos de texto relevantes**.
2. **Construção de Contexto**: Os trechos recuperados são **concatenados** e usados como **entrada adicional** ao FM.
3. **Geração da Resposta**: O modelo gera uma resposta mais precisa com base nesse **contexto enriquecido**.

Vantagens:

- Respostas altamente personalizadas e detalhadas.
- Melhora da **confiabilidade** e **precisão** das respostas.

Resumo Visual dos Padrões

Padrão	Quando Usar	Ferramentas Associadas
Geração de Texto ou Código	Criação de textos ou trechos de código	Amazon Bedrock
Geração de Texto com LangChain	Quando se precisa de orquestração ou interação complexa	LangChain + Amazon Bedrock
Resumo de Documentos Pequenos	Quando o texto cabe na janela de contexto do FM	Amazon Bedrock FM direto
Resumo de Documentos Grandes	Para documentos longos e detalhados	LangChain (map-reduce, chaining)
Resposta a Perguntas Genérica	Quando não há necessidade de personalização	Amazon Bedrock FM
Resposta a Perguntas com RAG	Quando o domínio é específico e exige alta precisão	RAG + Amazon Bedrock

Padrão de Assistente de IA com Amazon Bedrock

Definição

Assistentes de IA são interfaces de conversação baseadas em modelos de linguagem que utilizam algoritmos de **Processamento de Linguagem Natural (PNL)** e **Machine Learning (ML)** para compreender e responder às interações dos

usuários. Eles são aplicados em diversas áreas, como:

- Atendimento ao cliente
- Vendas
- Comércio eletrônico

Podem ser acessados por diferentes canais: **sites, redes sociais, aplicativos de mensagens**, entre outros.

Arquitetura Básica de um Assistente de IA com Amazon Bedrock

A arquitetura típica de um assistente de IA neste contexto segue os seguintes passos:

1. **Usuário envia uma consulta** ao assistente de IA.
2. O sistema envia ao modelo do Amazon Bedrock a **consulta atual + histórico de bate-papo** (caso exista).
3. O **modelo gera uma resposta** com base no prompt e no histórico.
4. A resposta é **retornada ao usuário**.
5. A interação é **armazenada no histórico de chat** para manter o contexto ao longo da conversa.

> O diagrama citado mostra um fluxo de conversas entre usuário e assistente, com armazenamento do histórico e resposta gerada pelo FM.

Casos de Uso de Assistentes de IA

1. Assistente de IA Básico

- Modelo de **tiro zero (zero-shot)**.
- Sem histórico ou personalização.
- Executa tarefas simples com base em prompts pontuais.

2. Assistente de IA com Modelo de Prompt

- Inclui um **contexto pré-definido** no prompt enviado ao FM.
- Proporciona respostas mais focadas em um domínio ou tarefa específica.

3. Assistente de IA com Persona

- O FM recebe uma **função específica** (por exemplo, atuar como um coach de carreira).
- Garante que as respostas sigam o estilo, tom e função designados.

4. Assistente de IA com Reconhecimento de Contexto

- Usa **arquivos externos** convertidos em embeddings para manter e utilizar **contexto ao longo do tempo**.
- Ideal para domínios que exigem retenção e referência de informações específicas.

Observação

Outros padrões mais avançados, como o uso de **base de conhecimento** e **agentes de orquestração**, serão explorados nos módulos posteriores.

Usando LangChain

Desafios de Desempenho com LLMs

Modelos de linguagem de grande porte (LLMs) são treinados com grandes volumes de dados e conseguem executar diversas tarefas como:

- Geração e resumo de texto
- Resposta a perguntas
- Análise de sentimentos

No entanto, **enfrentam limitações** quando:

- Precisam lidar com dados fora do domínio conhecido
- Precisam lembrar o contexto de uma conversa anterior
- Realizam tarefas que exigem múltiplas etapas ou raciocínio

Essas limitações podem gerar **alucinações** (respostas incorretas ou inventadas) ou resultados imprecisos. Em muitos casos, **um único prompt não é suficiente** para obter a resposta desejada, sendo necessário **encadear várias chamadas ao modelo** para se chegar a um resultado correto.

LangChain: Simplificando o Desenvolvimento com LLMs

LangChain é uma estrutura (framework) de código aberto criada para **desenvolver aplicativos baseados em LLMs** de forma mais prática e eficiente. Ela oferece **blocos de construção reutilizáveis** para acelerar e simplificar o desenvolvimento, produção e implantação de soluções com LLMs.

Motivos para usar LangChain:

- Os LLMs **não mantêm o estado** entre chamadas: o LangChain ajuda no **gerenciamento de contexto**.
- LLMs podem ser usados para **resolução de problemas complexos** com múltiplas etapas: LangChain fornece ferramentas para **sequenciamento de etapas**.
- Permite **encadear operações e chamadas** de maneira declarativa com LangChain Expression Language (LCEL).

Componentes do LangChain

LangChain está disponível em **Python, TypeScript e JavaScript** e oferece uma série de componentes modulares. Os principais componentes destacados neste módulo são:

1. Modelos

São wrappers em torno de LLMs, usados para enviar prompts e receber respostas.

> Exemplo apresentado: chamada de modelo para gerar uma resposta com base em um prompt.

2. Modelos de Prompt

Permitem estruturar e reutilizar prompts de forma eficiente e organizada.

> Exemplo apresentado: modelo de prompt personalizado com campos dinâmicos.

3. Índices

Conectam os LLMs a fontes externas de dados, como documentos ou bases de conhecimento, permitindo recuperar informações relevantes.

> Exemplo apresentado: indexação de documentos para recuperação com RAG.

4. Memória

Gerencia o **histórico de interações** com o modelo, permitindo manter o contexto de conversas anteriores.

> Exemplo apresentado: armazenamento de mensagens de bate-papo em memória.

5. Cadeias (Chains)

São sequências de passos que um modelo deve seguir. Permitem **encadear prompts, respostas e ações** de forma estruturada.

> Exemplo apresentado: cadeia de operações que envolve recuperação de dados, geração de resposta e resumo.

6. Agentes

Executam **decisões condicionais** e selecionam qual ação tomar com base em ferramentas externas e raciocínio do modelo.

> Exemplo apresentado: agente que escolhe entre diferentes ferramentas para responder a uma pergunta complexa.

Aplicações Práticas dos Componentes

Você pode utilizar os componentes do LangChain para desenvolver:

- Chatbots com **RAG** (Retrieval-Augmented Generation)
- Sistemas de **resumo automático** de textos
- **Geradores de código**
- **Ferramentas de extração de informações**
- Interfaces para **consultas em linguagem natural**
- **Integrações com APIs** externas para compor respostas mais completas

---

Começando com LCEL

A **LangChain Expression Language (LCEL)** é uma linguagem declarativa que permite **definir e conectar componentes** de forma simples e estruturada. Ela é ideal para quem deseja começar rapidamente e evoluir para construções mais sofisticadas.

Integrações com Suporte para AWS no LangChain

Visão Geral

O LangChain oferece suporte a diversas integrações com serviços da AWS, possibilitando a construção de aplicações baseadas em LLMs (Modelos de Linguagem de Grande Porte) de forma flexível e eficiente. Entre as integrações disponíveis estão:

- **Modelos de chat**
- **LLMs**
- **Modelos de incorporação de texto (embeddings)**
- **Carregadores de documentos**
- **Armazenamentos vetoriais (vector stores)**
- **Recuperadores de informações**
- **Ferramentas externas**
- **Memória**
- **Cadeias (chains)**
- **Agentes**
- **Gráficos**

- **Callbacks (retornos de chamada)**

Integração com Amazon Bedrock

O LangChain suporta integração direta com o Amazon Bedrock, permitindo acesso a diversos FMs (Modelos Fundamentais) por meio dos pacotes:

- **langchain.aws**
- **langchain-community**

A integração permite utilizar modelos de diferentes provedores disponíveis no Amazon Bedrock, incluindo:

- **Amazon Titan Text**
- **AI21 Jurassic**
- **Anthropic Claude**
- **Cohere Command e Embed**
- **Meta Llama**
- **Stability AI (Stable Diffusion)**
- **Mistral AI**

Essas integrações oferecem ampla cobertura de modelos para geração de texto, embeddings, imagens e outros casos de uso com IA generativa.

APIs de Integração AWS no LangChain

LLMs (Modelos de Linguagem de Grande Porte)

No LangChain, a classe LLM é uma abstração para interação com modelos de linguagem de diversos provedores, facilitando o uso independente da tecnologia subjacente. Essa classe serve como base para realizar tarefas como geração de texto, completamento e resposta a perguntas.

> **Exemplo apresentado: criação de uma instância do BedrockLLM, utilizando um modelo Titan da Amazon, com parâmetros de inferência como temperatura e número máximo de tokens. A resposta a uma pergunta simples é gerada usando esse modelo.**

Esse padrão de uso é comum para aplicações que requerem entrada textual e resposta automatizada, como assistentes virtuais, sistemas de perguntas e respostas, geração de conteúdo, entre outros.

Modelos Personalizados no Amazon Bedrock

O Amazon Bedrock permite **personalizar Modelos Fundamentais (FMs)** para melhorar o desempenho em casos de uso específicos, por meio de duas abordagens principais:

- **Pré-treinamento contínuo**: treinar o modelo com mais dados após seu pré-treinamento original.
- **Ajuste fino (fine-tuning)**: adaptar o modelo a um domínio ou tarefa específica.

> **Exemplo apresentado:** Criação de uma instância de um modelo personalizado hospedado no Amazon Bedrock, utilizando um ARN (Amazon Resource Name) para identificá-lo, e envio de uma pergunta para receber uma resposta personalizada.

Modelos de Bate-Papo (Chat Models)

Os **modelos de bate-papo** são fundamentais para criar experiências interativas em assistentes virtuais e chatbots. No LangChain, o componente de chat aceita **mensagens** como entrada (com estrutura específica para cada tipo de mensagem, como mensagens humanas, do sistema ou do próprio assistente).

> **Exemplo apresentado:** Criação de uma instância de um modelo da Anthropic (Claude 3), envio de uma mensagem com uma consulta sobre comida indiana, e recebimento de uma resposta detalhada com sugestões de pratos. A resposta também inclui metadados sobre uso de tokens.

Esse padrão é ideal para **aplicações de atendimento ao cliente, assistentes pessoais, sistemas educacionais, entre outros.**

Modelos de Incorporação de Texto (Text Embedding Models)

Os **modelos de incorporação de texto** transformam entradas textuais em **vetores numéricos**, capturando o significado semântico do conteúdo. Essas representações vetoriais (ou *embeddings*) são amplamente usadas em tarefas como:

- Análise de sentimentos
- Classificação de texto
- Sistemas de recomendação
- Recuperação de informações semânticas

Esses vetores podem ser armazenados em **bancos de dados vetoriais**, permitindo **buscas mais precisas e rápidas** com base em similaridade de significado e não apenas por palavras-chave.

Prompts

Um **prompt** é uma instrução textual usada para orientar o comportamento do modelo LLM. Em aplicações mais complexas de IA generativa, os prompts podem incluir:

- Instruções detalhadas
- Contexto da tarefa
- Exemplos
- Variáveis dinâmicas

Esses elementos ajudam a guiar o modelo a produzir respostas mais adequadas às necessidades do usuário.

Modelos de Prompt no LangChain

O LangChain fornece **modelos de prompt predefinidos**, que são estruturas de texto parametrizadas. Eles permitem:

- **Eficiência na engenharia de prompt**: com reutilização e padronização de estruturas de entrada.
- **Flexibilidade**: ao gerar prompts com base em variáveis dinâmicas fornecidas em tempo de execução.

> **Exemplo apresentado:** Um modelo de prompt que aceita variáveis de entrada e gera um prompt formatado dinamicamente com base nessas entradas.

Carregadores de Documentos (Document Loaders)

Ao desenvolver aplicações de IA generativa com a abordagem **RAG (Retrieval-Augmented Generation)**, é essencial que os modelos de linguagem tenham acesso a **dados externos**, que são fornecidos por meio de documentos provenientes de diversas fontes.

O que são os Document Loaders?

No LangChain, os **Document Loaders** são componentes responsáveis por:

- Carregar documentos de **várias fontes externas**
- Preparar os dados para que possam ser **convertidos em embeddings** (vetores semânticos)
- **Indexar e reutilizar** as informações nos fluxos de recuperação de dados para os LLMs

Fontes suportadas

Os Document Loaders podem importar conteúdos de:

- **Bancos de dados**
- **Lojas online**
- **Sistemas locais de arquivos**
- **Serviços em nuvem** como o Amazon S3

Tipos de arquivos comuns suportados

- Arquivos HTML
- Documentos PDF
- Documentos do Word (como .docx)
- Código-fonte
- Outros formatos estruturados e semiestruturados

Exemplo descrito

O exemplo apresentado mostra como carregar um documento armazenado em um bucket do **Amazon S3** utilizando o componente de carregamento específico para esse serviço. Após configurar o loader com o nome do bucket e o caminho do arquivo, o documento é carregado e fica disponível para processamento posterior, como vetorização ou análise semântica.

Recuperador (Retriever)

O **retriever** é um componente essencial do LangChain que busca documentos relevantes com base em uma consulta do usuário. Ele funciona como parte da arquitetura RAG (Retrieval-Augmented Generation), permitindo que os modelos de linguagem operem com base em **dados externos e atualizados**.

Como funciona o Retriever

- Quando o usuário faz uma pergunta, o retriever:
  a. Pesquisa em um índice de documentos previamente carregado.
  b. Retorna os documentos mais relevantes.
  c. Envia esses documentos como **contexto** para o modelo de linguagem gerar uma resposta informada.

Esse processo aumenta a precisão das respostas e reduz o risco de alucinações por parte do LLM.

Integração com Amazon Kendra

Para aplicações na AWS, o LangChain oferece suporte à **integração com o Amazon Kendra**, um serviço de busca empresarial totalmente gerenciado que realiza:

- **Pesquisa semântica inteligente**, usando IA para entender a intenção da consulta.
- **Classificação avançada de documentos e passagens**, com resultados altamente relevantes.
- **Suporte a diversos conectores**, incluindo Amazon S3, SharePoint, Confluence e sites.
- **Compatibilidade com formatos comuns**, como HTML, Word, PowerPoint, PDF, Excel e arquivos de texto simples.

Uso com LangChain

O LangChain permite usar o **AmazonKendraRetriever**, que se conecta a um índice do Kendra para recuperar dados relevantes. Esses dados são então enviados ao LLM, junto com um prompt estruturado, para gerar respostas detalhadas e fundamentadas. O exemplo descrito configura:

- Um modelo de linguagem compatível com o Amazon Bedrock.
- Um retriever apontando para um índice específico no Kendra.
- Um prompt personalizado que define o tom e limita o tamanho da resposta.
- A criação de uma cadeia de recuperação conversacional, que junta todas essas peças para produzir a resposta final.

Lojas de Vetores (Vector Stores)

As lojas de vetores são componentes fundamentais em aplicações de IA generativa que utilizam a abordagem **RAG (Retrieval-Augmented Generation)**. Elas permitem armazenar e consultar **incorporações vetoriais** de dados, como documentos empresariais, para fornecer **contexto relevante aos modelos de linguagem**.

Fluxo da abordagem RAG com Lojas de Vetores

1. **Criação de embeddings**: Dados empresariais (como documentos ou manuais) são transformados em vetores numéricos usando um modelo de incorporação de texto.
2. **Armazenamento vetorial**: Esses vetores são armazenados em um banco de dados vetorial.
3. **Consulta semântica**: Quando o usuário envia uma solicitação, o sistema consulta esse banco vetorial para encontrar os vetores mais semelhantes (documentos relevantes).
4. **Geração de resposta**: Os dados recuperados são enviados como contexto para o LLM, resultando em respostas mais precisas e alinhadas ao domínio do usuário.

Suporte do LangChain a repositórios vetoriais

O LangChain é compatível tanto com **repositórios vetoriais de código aberto** quanto com **soluções específicas de provedores**, como:

- **Amazon OpenSearch Serverless**: Uma solução de busca escalável e sem servidor que pode atuar como banco de vetores.
- **pgvector com Amazon Aurora PostgreSQL-Compatible Edition**: Uma extensão vetorial para bancos relacionais que permite consultas semânticas baseadas em embeddings.

Componentes envolvidos

O LangChain oferece um componente chamado **VectorStore**, que abstrai a interação com o banco de vetores. Esse componente permite:

- Armazenar vetores gerados por modelos de embedding.
- Consultar vetores com base em similaridade semântica.
- Atuar como **retriever**, permitindo que os vetores relevantes sejam usados em cadeias de geração de texto.

Exemplo descrito

O exemplo fornecido na documentação demonstra como:

- Criar embeddings com o serviço da AWS Bedrock.
- Conectar-se a um índice vetorial hospedado no Amazon OpenSearch Serverless.
- Consultar o índice vetorial e recuperar os dados mais relevantes como entrada para um modelo de linguagem.

Memória LangChain

A **memória LangChain** é projetada para armazenar e recuperar elementos de conversas anteriores, o que permite que o modelo mantenha o contexto ao longo de interações subsequentes. Essa memória é útil para construir chatbots e sistemas

conversacionais dinâmicos, já que mantém um histórico que pode ser referenciado para fornecer respostas mais coerentes e contextualmente apropriadas.

Tipos de Memória no LangChain

1. **ConversationBufferMemory**:
   - O tipo mais comum de memória.
   - Armazena as interações passadas entre o usuário e o modelo, permitindo que o contexto da conversa seja preservado.
   - É útil para simples rastreamento de conversas.
2. **ConversationChain**:
   - Baseado no ConversationBufferMemory, mas projetado para gerenciar as conversas de forma mais estruturada.
   - Facilita o controle e a manipulação das interações dentro de um fluxo conversacional mais complexo.

Esses componentes modulares podem ser encadeados com outros elementos do LangChain, como recuperadores de informações ou modelos, para criar uma experiência conversacional mais envolvente e eficaz.

Componentes de encadeamento

As **cadeias (chains)** no LangChain são compostas por uma sequência de componentes que trabalham em conjunto para processar informações de forma estruturada. Cada componente de uma cadeia pode representar uma chamada a um LLM, uma API, ou até outra cadeia. Essas cadeias permitem criar fluxos de trabalho complexos onde a saída de um componente pode ser utilizada como entrada para o próximo, possibilitando o processamento de dados em múltiplas etapas.

Tipos de Cadeias

1. **Cadeia usando LCEL (LangChain Expression Language)**:
   - Utiliza a linguagem de expressão declarativa do LangChain para encadear componentes e construir fluxos de trabalho complexos.
   - Ideal para criar e controlar cadeias de maneira flexível e eficiente.
2. **Cadeias Legadas**:
   - Construídas por subclasses de uma classe base chamada `Chain` (como o `LLMChain`), essas cadeias são mais tradicionais e prontas para uso.

Uso das Cadeias

- **Processamento de grandes volumes de dados**: Quando os dados excedem o tamanho do contexto permitido pelo LLM, as cadeias podem ser usadas para dividir dados em partes menores, processar cada parte separadamente e combinar os resultados em uma saída única.
- **Fluxos complexos**: As cadeias permitem criar fluxos de trabalho mais avançados, com múltiplas chamadas a LLMs ou outras ferramentas, tornando a solução mais robusta e capaz de lidar com tarefas que envolvem vários estágios de processamento.

Essas cadeias são ferramentas poderosas para organizar e coordenar múltiplos componentes, facilitando a construção de aplicativos mais complexos no LangChain.

Gerenciando recursos externos com agentes LangChain

Os **agentes LangChain** são componentes que permitem aos LLMs interagir com recursos externos, como APIs, mecanismos de busca, ou até mesmo executar código para tarefas que envolvem raciocínio lógico, matemático ou baseado em regras. Embora os LLMs sejam eficientes na geração de texto, os agentes ajudam a superar suas limitações, tornando-os capazes de lidar com questões mais complexas que exigem ações externas.

Funcionamento dos Agentes LangChain

1. **Estrutura de Ação**:

- LangChain fornece ferramentas e kits de ferramentas, compostos por funções específicas que o agente pode chamar para executar ações.
- O agente utiliza a abordagem **ReAct (Raciocínio e Ação)** para selecionar a ferramenta mais adequada com base na entrada do usuário e decidir quais ações tomar em sequência até atingir uma condição de parada.

2. **Tipos de Cadeias e Agentes**:
   - **LLMChain**: Utilizado em aplicativos de Recuperação e Geração (RAG), onde o LLM retorna uma resposta baseada na consulta do usuário e em um contexto fornecido (geralmente, documentos recuperados de um repositório de vetores).
   - **RouterChain**: Um exemplo de cadeia mais complexa, que pode ser usada para selecionar um modelo de prompt adequado com base na entrada do usuário.
   - **Agentes LangChain**: Usam raciocínio lógico para decidir qual ação tomar, executam a ação e observam a saída da ferramenta. As ações podem envolver consultas a mecanismos de busca, cálculos matemáticos, ou interações com APIs externas.

Esses agentes são fundamentais para expandir as capacidades dos LLMs, permitindo-lhes realizar tarefas mais avançadas e interagir de forma mais eficaz com o mundo real.

Tutoriais:

<details>
<summary>Configurar e executar laboratórios de notebook LangChain no Amazon SageMaker</summary>

Configurar e executar laboratórios de notebook LangChain no Amazon SageMaker

Neste laboratório, você percorre os notebooks do Jupyter e faz chamadas de API para modelos de inteligência artificial generativa (IA generativa) hospedados no Amazon Bedrock. Esses laboratórios são baseados nos padrões de arquitetura discutidos nos módulos anteriores. Os notebooks usam o Bedrock para gerar e resumir texto, responder perguntas e construir um chatbot.

Nesta primeira tarefa, você configurará seu ambiente de laboratório para poder concluir esses laboratórios. Isso exigirá que você solicite acesso ao modelo no Amazon Bedrock, bem como configure seu ambiente de laboratório no Amazon SageMaker Studio.

Para começar, você precisará baixar os arquivos do Jupyter notebook que você precisará para concluir os laboratórios. Há 9 arquivos de notebook no total, bem como 1 arquivo .csv e 1 arquivo .txt que serão necessários para alguns dos laboratórios.

Depois de baixar os arquivos para seu computador local, anote o local. Você precisará acessar o conteúdo para executar esses labs.

Para usar o Bedrock, os usuários de conta com as Permissões IAM corretas devem habilitar o acesso aos modelos de fundação Bedrock (FMs) disponíveis. Para selecionar os modelos que você precisa, certifique-se de estar na região us-east-1.

Na parte superior do AWS Management Console, na barra de pesquisa, procure e escolha Amazon Bedrock.

Vá para a página do console do Amazon Bedrock. Selecione Get Started.

Selecione Gerenciar acesso ao modelo.
</details>

Para adicionar ou atualizar seu acesso ao modelo existente, escolha Habilitar modelos específicos.

Na página Editar acesso ao modelo, localize os seguintes modelos e use as caixas de seleção para selecioná-los, caso ainda não tenha acesso a eles:

Titan Embeddings G1 – Texto

Titan Text G1 - Express

Titan Text G1 - Premier

Depois role para baixo.

Nota: Se você notar que o status de acesso já mostra Acesso concedido, nenhuma ação é necessária. Se você não vir o Titan Text G1 – Premier como uma das opções, certifique-se de que você está na região us-east-1.

Em seguida, localize os seguintes modelos e use as caixas de seleção para selecioná-los, caso ainda não tenha acesso a eles:

Soneto Claude 3

Lhama 3 8B Instruir

Depois role para baixo.

Depois de verificar se todos os modelos necessários estão selecionados ou já estão disponíveis, escolha Avançar.

Revise suas seleções, se aplicável, e então escolha Enviar. Ao escolher enviar, você concorda com os preços dos modelos de terceiros e os Contratos de Licença de Usuário Final, bem como com os Termos de Serviço Bedrock.

Revise suas seleções, se aplicável, e escolha Enviar.

Depois que seu acesso for concedido, você poderá usar esses modelos com os cadernos de laboratório.

Nesta etapa, você iniciará um aplicativo Amazon SageMaker Studio para acessar seu ambiente de laboratório.

Na parte superior do AWS Management Console, na barra de pesquisa, procure e escolha Amazon SageMaker.

Para prosseguir, você precisará configurar um domínio. Selecione Configurar para usuário único.

Você verá um banner Preparing SageMaker Domain. Esta é uma configuração única e levará vários minutos.

Quando seu domínio aparecer como Pronto, escolha a aba Perfis de Usuário.

Selecione o botão Iniciar ao lado do perfil de usuário que acabou de ser criado e selecione Estúdio.

Quando estiver no Studio, você poderá ver um pop-up de tour rápido. Selecione Pular tour por enquanto.

Para começar, escolha o ícone JupyterLab.

Selecione o botão Criar espaço JupyterLab.

Insira um nome para seu espaço JupyterLab. Deixe as configurações de Compartilhamento como padrão e escolha Criar Espaço.

Seu laboratório foi criado, escolha Run Space para iniciar o laboratório.

Quando o status mudar para Em execução, escolha o botão Abrir JupyterLab para iniciar o Studio em uma nova aba.

Nesta etapa, você preparará seus recursos de laboratório. Isso inclui carregar arquivos de notebook e executar atualizações no ambiente de laboratório antes de começar suas tarefas.

Assim que o JupyterLabs abrir, você estará trabalhando com a pasta raiz. É aqui que você fará upload de arquivos e criará pastas.

Use o ícone Nova Pasta e crie uma pasta chamada Arquivos. Abra esta pasta.

Use o ícone Upload files para carregar o arquivo do notebook Task0 do seu computador local para o ambiente de laboratório. Retorne para a pasta raiz.

Use o ícone Nova pasta para criar uma nova pasta. Nomeie essa pasta com letras. Abra essa pasta.

Carregue o arquivo 2022-letter.txt do seu computador local para esta pasta.

Navegue de volta para a pasta raiz.

Use o ícone Nova pasta para criar uma nova pasta. Nomeie esta pasta como rag_data. Abra esta pasta.

Carregue o arquivo Amazon_SageMaker_FAQs.csv do seu computador local para esta pasta.

Navegue de volta para a pasta raiz.

Abra a pasta Arquivos e depois abra o arquivo do bloco de notas Task0.

Se solicitado a selecionar uma imagem e kernel para executar este notebook, mantenha as configurações padrão, que devem ser Python 3 (ipykernel). Escolha Select.

Depois que o kernel do notebook iniciar, siga as instruções neste notebook para instalar todos os pacotes necessários para executar os laboratórios de 1 a 6.

Após a conclusão da Tarefa 0, navegue até a barra de menu e selecione Kernel→Reiniciar Kernel.

Selecione Reiniciar quando solicitado.

Seu ambiente está todo configurado e você está pronto para executar os laboratórios.

O passo final é adicionar o restante dos arquivos de tarefa ao ambiente de laboratório. Selecione o ícone Upload.

Selecione os 8 arquivos restantes do caderno de tarefas no seu computador local e escolha Abrir.

Você deve ver todos os arquivos de tarefa enviados para seu ambiente de laboratório. Você está pronto para começar!

Você fez o seguinte com sucesso:

- Solicitou acesso aos modelos do Amazon Bedrock.
- Iniciou e configurou o Amazon SageMaker.
- Arquivos de configuração enviados para os laboratórios.
- Atualizou o kernel e os arquivos no ambiente.

Explore casos de uso de IA generativa com LangChain e Amazon Bedrock

Explore casos de uso de IA generativa com LangChain e Amazon Bedrock

O caso de uso comercial nesta demonstração é a AnyCompany. Eles têm vastos repositórios de dados de log de engenharia e tickets de suporte técnico contendo insights críticos para otimizar suas operações de tecnologia e infraestrutura.

No entanto, os engenheiros lutam para analisar manualmente os dados de texto infinitos para identificar padrões importantes e conhecimento de solução de problemas. A AnyCompany encarregou sua equipe de desenvolvimento de aproveitar o conjunto de IA generativa da Amazon Bedrock e os recursos de agente da LangChain para construir rapidamente soluções de IA corporativa personalizadas.

Neste laboratório, você percorre os notebooks do Jupyter e faz chamadas de API para modelos de inteligência artificial generativa (IA generativa) hospedados no Amazon Bedrock. Esses laboratórios são baseados nos padrões de arquitetura discutidos nos módulos anteriores. Os notebooks usam o Bedrock para gerar e resumir texto, responder perguntas e construir um chatbot.

Nesta tarefa, você executa dois arquivos de notebook: Task1a.ipynb, que invoca um modelo do Amazon Bedrock para geração de texto usando um prompt zero-shot, e Task1b.ipynb, que usa a estrutura LangChain para se comunicar com a API do Amazon Bedrock e cria um modelo de prompt LangChain personalizado para adicionar contexto à solicitação de geração de texto.

Neste notebook, você aprende a usar o Large Language Model (ou LLM) para gerar uma resposta por e-mail para um cliente que forneceu feedback negativo sobre a qualidade do serviço ao cliente que recebeu do engenheiro de suporte. Neste notebook, você gera um e-mail com uma nota de agradecimento com base no e-mail anterior do cliente.

O prompt usado nesta tarefa é chamado de prompt zero-shot. Em um prompt zero-shot, você descreve a tarefa ou a saída desejada para o modelo de linguagem em linguagem simples. O modelo então usa seu conhecimento e capacidades pré-treinados para gerar uma resposta ou concluir a tarefa com base somente no prompt fornecido.

Neste cenário, você é Bob, um gerente de atendimento ao cliente na AnyCompany. Alguns de seus clientes não estão satisfeitos com o atendimento ao cliente e estão fornecendo feedback negativo sobre o serviço fornecido pelos engenheiros de suporte ao cliente. Agora, você gostaria de responder a esses clientes se desculpando pelo serviço ruim e recuperar a confiança. Você precisa da ajuda de um LLM para gerar uma grande quantidade de e-mails para você que sejam amigáveis e personalizados para o sentimento do cliente a partir de correspondências de e-mail anteriores.

Na Tarefa1a.1, você configura seu ambiente.

Execute a primeira célula de código para começar.

Na Tarefa 1a.2, você prepara uma entrada para o serviço Amazon Bedrock para gerar um e-mail.

Execute a primeira célula para criar o prompt.

Execute a segunda célula para definir parâmetros.

Depois de executar a célula, você pode ler os itens a seguir para obter mais informações sobre o modelo Amazon Titan.

Na Tarefa 1a.3, você explora como o modelo gera uma saída com base no prompt criado anteriormente.

Este e-mail é gerado usando o modelo Amazon Titan ao entender a solicitação de entrada e utilizar seu entendimento inerente de diferentes modalidades. A solicitação para a API é síncrona e aguarda que toda a saída seja gerada pelo modelo.

Execute a primeira célula para invocar o modelo.

Execute a segunda célula para criar um rascunho do e-mail.

O e-mail foi fornecido como uma saída para você usar.

O Bedrock também suporta que a saída possa ser transmitida conforme é gerada pelo modelo em forma de blocos. Este e-mail é gerado invocando o modelo com a opção de transmissão. `invoke model with response stream` retorna um `ResponseStream` do qual você pode ler.

Execute esta célula para criar a saída em blocos.

Você vê aqui que a carta foi gerada em pedaços rotulados.

A abordagem de fluxo com resposta ajuda a obter rapidamente a saída do modelo e permite que o serviço a conclua enquanto você lê. Isso auxilia em casos de uso em que você solicita que o modelo gere pedaços maiores de texto. Mais tarde, você pode combinar todos os pedaços gerados para formar a saída completa e usá-la para seu caso de uso.

Execute esta célula para combinar os blocos de saída.

Os pedaços foram combinados e a carta está pronta para uso.

Agora você experimentou usar o boto3 SDK, que fornece exposição básica à API do Amazon Bedrock. Usando essa API, você viu o caso de uso de gerar um e-mail para responder ao feedback negativo de um cliente

Você concluiu este notebook. Feche este arquivo de notebook e continue com a Tarefa 1b.

Neste caderno, você aprenderá como gerar uma resposta de e-mail para um cliente que não ficou satisfeito com a qualidade do serviço de atendimento ao cliente que recebeu do engenheiro de suporte ao cliente. Você fornecerá contexto adicional ao modelo incluindo o conteúdo do e-mail real recebido do cliente insatisfeito.

Você adicionará mais complexidade com a ajuda de PromptTemplates para alavancar a estrutura LangChain para um caso de uso semelhante. PromptTemplates permitem que você crie shells genéricos que podem ser preenchidos com informações posteriormente e obter saídas de modelo com base em diferentes cenários.

Devido ao contexto adicional no prompt, o conteúdo produzido neste notebook é de qualidade e relevância muito melhores do que o conteúdo produzido anteriormente por meio de prompts zero-shot. O prompt usado neste notebook cria um modelo de prompt LangChain personalizado para adicionar contexto à solicitação de geração de texto.

Neste cenário, você ainda é um gerente de atendimento ao cliente na AnyCompany e pode aproveitar o poder dos PromptTemplates da LangChain para criar um shell genérico para gerar respostas de e-mail personalizadas com base no e-mail anterior do cliente. O PromptTemplate incorporará o conteúdo original do e-mail do cliente, permitindo que o LLM entenda o contexto e o sentimento e, então, gere uma resposta relevante e personalizada.

Na Tarefa1b.1, você configura seu ambiente.

Execute a célula de código para criar um cliente de serviço por nome usando a sessão padrão.

Na Tarefa 1b2, você cria uma instância da classe Bedrock a partir de LLMs. Isso espera um `model_ID` que é o Amazon Resource Name (ARN) do modelo disponível no Amazon Bedrock.

Opcionalmente, você pode passar um cliente boto3 criado anteriormente, bem como alguns `model kwargs` que podem conter parâmetros como `temperatura`, `top p`, `contagem máxima de tokens` ou `sequências de parada`.

Execute a célula de código para invocar e configurar o modelo Bedrock LLM.

Na Tarefa 1b.3, você criará um modelo para o prompt que você pode passar diferentes variáveis de entrada em cada execução. Isso é útil quando você tem que gerar conteúdo com diferentes variáveis de entrada que você pode estar buscando de um banco de dados.

Na tarefa anterior, codificamos o prompt. Pode ser o caso de você ter vários clientes enviando feedback negativo semelhante, e agora você quer usar cada um dos e-mails desses clientes e responder a eles com um pedido de desculpas, mas você também quer manter a resposta um pouco personalizada. Na célula a seguir, você explorará como pode criar um `PromptTemplate` para atingir esse padrão.

Execute esta célula de código para criar um modelo de prompt que tenha várias variáveis de entrada.

O código também passará valores para as variáveis de entrada.

Execute esta célula de código para obter o número de tokens.

Depois que o código terminar de executar a célula, você pode observar os resultados. Depois que tudo tiver sido analisado, você sabe que seu prompt tem 156 tokens.

Execute a próxima célula de código para invocar o modelo.

Por fim, execute esta última célula de código para configurar uma cadeia para analisar a saída e criar uma letra.

A carta de desculpas foi criada como resultado para sua análise.

Você aprendeu com sucesso que invocar o LLM sem qualquer contexto pode não produzir os resultados desejados. Ao adicionar contexto e usar ainda mais o modelo de prompt para restringir a saída do LLM, você conseguiu obter com sucesso a saída desejada.

Você concluiu este notebook. Feche este arquivo de notebook e continue com a Tarefa 2.

Nesta tarefa, você executa dois arquivos de notebook: Task2a.ipynb, que resume o texto com arquivos pequenos usando o Titan Text Premier, e Task2b.ipynb, que usa chunking para resumir textos longos com o Amazon Titan.

Neste notebook, você ingere uma pequena sequência de texto diretamente na API do Amazon Bedrock (usando o modelo Titan Text) e a instrui a resumir o texto de entrada. Você pode aplicar essa abordagem para resumir transcrições de chamadas, transcrições de reuniões, livros, artigos, postagens de blog e outros conteúdos relevantes quando o comprimento do texto de entrada estiver dentro dos limites de tamanho de contexto do modelo.

Na Tarefa2a.1, você configura seu ambiente.

Execute a primeira célula de código para começar.

Na Tarefa2a.2, você usa uma curta passagem de texto com menos tokens do que o comprimento máximo suportado pelo modelo de fundação. Como um texto de entrada de amostra para este laboratório, você usa um parágrafo de uma postagem de blog da AWS anunciando o Amazon Bedrock.

O prompt começa com uma instrução `Forneça um resumo do texto a seguir.`. Execute a célula de código.

Na Tarefa 2a.3, você cria um corpo de solicitação com os parâmetros de prompt e inferência acima

Execute a célula de código para criar o corpo da solicitação.

Na Tarefa 2a.4, você envia uma solicitação de API para a Amazon Bedrock especificando os parâmetros de solicitação: `modelID`, `accept` e `contentType`. Seguindo o prompt fornecido, o modelo de base na Amazon Bedrock resume o texto de entrada.

Por padrão, o serviço Amazon Bedrock gera o resumo inteiro para um determinado prompt em uma única saída. Isso pode ser lento se a saída do modelo contiver muitos tokens.

Execute a célula de código para configurar e invocar o modelo.

O resumo da postagem do blog foi enviado para sua análise.

Em seguida, você explora como usar o modelo invoke do Amazon Bedrock com a API de fluxo de resposta para transmitir saídas do modelo para que os usuários possam consumir saídas conforme elas são geradas. Em vez de gerar a saída completa de uma vez, essa API retorna um ResponseStream que envia pedaços menores de saída do modelo conforme eles são produzidos. Você pode exibir essas saídas de streaming em uma visualização contínua e consumível.

Execute esta primeira célula de código para invocar o modelo e transmitir a saída.

Os blocos de saída são transmitidos à medida que são gerados.

Execute a próxima célula de código.

Execute esta célula de código para criar um resumo de saída de streaming.

O resumo é gerado como saída de streaming. Aqui, ele está sendo produzido uma palavra por vez.

O fluxo foi concluído e o resumo está finalizado e pronto para uso.

Agora você experimentou usar o boto3 SDK para acessar a Amazon Bedrock API. Este SDK fornece acesso programático básico aos recursos do Bedrock. Ao aproveitar esta API, você conseguiu implementar dois casos de uso:

Gerando um resumo de texto completo do conteúdo de notícias da AWS de uma só vez,

e 2) Transmitir a saída do resumo em blocos para processamento incremental.

Você concluiu este notebook. Feche este arquivo de notebook e continue com a Tarefa 2b.

Neste caderno, você gerencia os desafios que surgem na sumarização de documentos grandes: o texto de entrada pode exceder o comprimento do contexto do modelo, gerar saídas alucinadas ou acionar erros de falta de memória.

Para atenuar esses problemas, este notebook demonstra uma arquitetura que usa segmentação e encadeamento de prompts com a estrutura LangChain, um kit de ferramentas que permite aplicativos que aproveitam modelos de linguagem.

Você explora uma abordagem que aborda cenários quando os documentos do usuário ultrapassam os limites de token. O chunking divide os documentos em segmentos sob limites de comprimento de contexto antes de alimentá-los sequencialmente aos modelos. Isso encadeia prompts em chunks, mantendo o contexto anterior. Você aplica essa abordagem para resumir transcrições de chamadas, transcrições de reuniões, livros, artigos, postagens de blog e outros conteúdos relevantes.

Na Tarefa2b.1, você configura seu ambiente.

Execute a primeira célula de código para começar.

Na Tarefa 2b.2, você precisa especificar o LLM para a classe LangChain Bedrock e passar argumentos para inferência.

Execute a célula de código para configurar o LangChain com o Boto3.

Na Tarefa 2b.3, você usará o arquivo de texto da carta do CEO da Amazon aos acionistas em 2022 que está no diretório Letters. A célula a seguir carrega o arquivo de texto e conta o número de tokens. Você verá um aviso indicando que o número de tokens no arquivo de texto excede o número máximo de tokens para este modelo.

Execute a célula de código para carregar a letra e obter o número de tokens.

O número total de tokens foi de 6526.

Observe que você pode ignorar com segurança quaisquer avisos e prosseguir para a próxima célula.

Na Tarefa 2b.4, você divide o texto em pedaços menores porque ele é muito longo para caber no prompt. `RecursiveCharacterTextSplitter` em LangChain suporta a divisão de texto longo em pedaços recursivamente até que o tamanho de cada pedaço se torne menor que `chunk size`. Um texto é separado com `separators=["\n\n", "\n"]` em pedaços, o que evita a divisão de cada parágrafo em vários pedaços.

Usando 6.000 caracteres por chunk, você pode obter resumos para cada porção separadamente. O número de tokens, ou pedaços de palavras, em um chunk depende do texto.

Execute a célula de código para começar a dividir o texto.

Execute a segunda célula de código para saber quantos documentos foram criados e quantos blocos o primeiro possui.

Os totais foram gerados para sua revisão. De acordo com a saída, há 10 documentos, e o primeiro tem 439 tokens.

Na Tarefa 2b.5, assumindo que o número de tokens é consistente nos outros documentos, você deve estar pronto para prosseguir. Você pode usar `load summary chain` do LangChain para resumir o texto. `load summary chain` fornece três maneiras de sumarização: `stuff`, `map reduce` e `refine`.

- `stuff`: coloca todos os chunks em um prompt. Assim, isso atingiria o limite máximo de tokens.
- `map reduce`: resume cada pedaço, combina os resumos e resume o resumo combinado. Se o resumo combinado for muito grande, ele geraria um erro.
- `refine`: resume o primeiro chunk, e então resume o segundo chunk com o primeiro resumo. O mesmo processo se repete até que todos os chunks sejam resumidos.

Tanto o map reduce quanto o refine invocam o LLM várias vezes e levam tempo para obter o resumo final. Você pode tentar o map reduce aqui.

Execute a primeira célula de código para usar o map reduce para o resumo.

Execute a segunda célula de código para invocar a cadeia.

Por fim, execute a última célula de código para gerar o resumo final.

O resumo final é criado como saída para sua revisão.

Agora você experimentou usar o encadeamento e o agrupamento de prompts com a estrutura LangChain para resumir documentos grandes e, ao mesmo tempo, atenuar problemas decorrentes de textos de entrada longos.

Você concluiu este notebook. Feche este arquivo de notebook e continue com a Tarefa 3.

Nesta tarefa, você utiliza o modelo Bedrock Titan para fornecer respostas factuais a consultas enviando solicitações incluídas no contexto e recebendo respostas relevantes.

Neste caderno, você aprenderá a usar o modelo Bedrock Titan para fornecer respostas informativas a consultas enviando a solicitação com o contexto relevante completo para o modelo e esperando a resposta de volta, abordando o desafio de fazer com que o modelo retorne respostas factuais para perguntas sem precisar preparar e indexar documentos com antecedência.

Este notebook simula o que **Retrieval-Augmented Generation (ou RAG)** faria, mas não usando RAG de fato. Essa abordagem funciona com documentos curtos ou aplicativos single-ton; pode não ser escalável para responder perguntas de nível empresarial, onde grandes documentos empresariais não podem ser todos encaixados no prompt enviado ao modelo.

Question Answering (ou QA) é uma tarefa importante que envolve extrair respostas para consultas factuais colocadas em linguagem natural. Normalmente, um sistema de QA processa uma consulta em uma base de conhecimento contendo dados estruturados ou não estruturados e gera uma resposta com informações precisas. Garantir alta precisão é essencial para desenvolver um sistema de resposta a perguntas útil, confiável e confiável, especialmente para casos de uso empresarial.

Neste cenário, você tenta modelar uma situação na AnyCompany onde você faz uma pergunta respondendo ao modelo para fornecer informações sobre a troca de pneus para um modelo de veículo específico que eles fabricam. Você primeiro consulta o modelo usando uma abordagem "tiro zero" para ver se ele pode fornecer respostas relevantes com base apenas em seus dados de treinamento.

No entanto, você percebe que o modelo parece estar "alucinando" respostas mais genéricas, como evidenciado quando você tenta um modelo de veículo falso e obtém respostas semelhantes. Isso implica na necessidade de aumentar o treinamento do modelo com os manuais de veículos reais da Example Company para dar detalhes sobre pneus para cada modelo.

Neste laboratório, você simula uma abordagem RAG sem dados externos. Você fornece um trecho detalhado do manual explicando como trocar os pneus no veículo AnyCompany Model Z. Você testa se o modelo agora pode dar uma resposta

personalizada e precisa aproveitando este conteúdo de exemplo em contexto.

Na Tarefa 3.1, você configura seu ambiente.

Execute a primeira célula de código para começar.

Nesta seção, tentamos usar um modelo fornecido pelo serviço Bedrock para responder a perguntas com base no conhecimento adquirido durante a fase de treinamento.

Na Tarefa 3.2, você usa o método invoke_model() do cliente Amazon Bedrock. Os parâmetros obrigatórios necessários para usar esse método são model ID, que representa o ARN do modelo Amazon Bedrock, e body, que é o prompt para sua tarefa.

O prompt do corpo muda dependendo do fornecedor do modelo de base selecionado.

Você tenta usar modelos fornecidos pelo serviço Bedrock para responder perguntas com base no conhecimento adquirido durante a fase de treinamento.

Execute a célula de código para definir os parâmetros do prompt.

Na Tarefa 3.3, você invoca o modelo passando o corpo JSON para gerar a resposta.

Execute a célula de código para invocar o modelo.

Uma lista é gerada como saída para sua revisão.

O modelo fornece uma resposta descrevendo o processo de troca do pneu furado do carro, mas a mesma explicação pode ser válida para qualquer carro. Infelizmente, esta não é a resposta correta para um AnyCompany AC8, que não tem um pneu reserva. Isso ocorre porque o modelo foi treinado em dados contendo instruções sobre troca de pneus em carros.

Outro exemplo desse problema pode ser visto ao tentar fazer a mesma pergunta para uma marca e modelo de carro completamente falsos, digamos, um Amazon Tirana.

Execute a célula de código para alterar o prompt para perguntar especificamente sobre um Amazon Tirana.

Dada a pergunta inicial, o modelo não consegue fornecer uma resposta realista.

Para corrigir esse problema e fazer com que o modelo forneça respostas com base nas instruções específicas válidas para o modelo do seu carro, você pode aumentar o conhecimento do modelo rapidamente, fornecendo uma base de conhecimento adicional como parte do prompt.

Vamos ver como você pode usar isso para melhorar seu aplicativo.

Suponha que o seguinte seja um trecho do manual do AnyCompany AC8. Na realidade, não é o manual real, mas você pode tratá-lo como tal.

Este documento também é convenientemente curto o suficiente para caber inteiramente na janela de contexto do Titan Large.

Execute esta célula de código para criar contexto usando o manual do AC8 que você forneceu.

Agora, execute esta célula de código para passar o trecho inteiro para o modelo junto com a pergunta.

Na Tarefa 3.4, você invoca o modelo via boto3 para gerar a resposta.

Execute a célula de código para gerar a saída do processo de troca de pneu do AC8.

A saída é criada para sua revisão. Você pode observar que a saída é específica para o AnyCompany AC8, e que vem do manual que você forneceu.

Como o modelo demora um pouco para entender o contexto e gerar uma resposta relevante para você, isso pode levar a uma experiência ruim para o usuário, pois ele terá que esperar alguns segundos por uma resposta.

O Bedrock também suporta capacidade de streaming onde o serviço gera saída conforme o modelo gera tokens. Aqui está um exemplo de como você pode implementar isso.

Execute a primeira célula de código.

Execute a segunda célula de código para transmitir a saída.

A resposta fornece instruções resumidas e passo a passo sobre como trocar os pneus.

Agora você aprendeu como aproveitar o processo RAG para gerar uma resposta personalizada e adaptada ao contexto específico e às informações fornecidas.

Você concluiu este notebook. Feche este arquivo de notebook e continue com a Tarefa 4.

Nesta tarefa, você cria uma interface de conversação usando os FMs no Amazon Bedrock e usa llama3-8b-instruct e titan-text-premier como FMs para criar os chatbots.

Interfaces de conversação, como chatbots e assistentes virtuais, podem aprimorar a experiência do usuário para seus clientes. Os chatbots usam processamento de linguagem natural (ou PNL) e algoritmos de aprendizado de máquina para entender e responder às consultas dos usuários. Você pode usar chatbots em uma variedade de aplicativos, como atendimento ao cliente, vendas e comércio eletrônico, para fornecer respostas rápidas e eficientes aos usuários. Os usuários podem acessá-los por meio de vários canais, como sites, plataformas de mídia social e aplicativos de mensagens.

Em interfaces de conversação, como chatbots, lembrar de interações anteriores se torna muito importante, tanto em curto quanto em longo prazo.

O framework LangChain fornece componentes de memória em duas formas. Primeiro, o LangChain fornece utilitários auxiliares para gerenciar e manipular mensagens de bate-papo anteriores. Eles são projetados para serem modulares. Segundo, o LangChain fornece maneiras fáceis de incorporar esses utilitários em cadeias, permitindo que você defina e interaja facilmente com diferentes tipos de abstrações, o que torna chatbots poderosos fáceis de construir.

O primeiro processo na construção de um chatbot sensível ao contexto é gerar embeddings para o contexto. Normalmente, você tem um processo de ingestão que é executado por meio do seu modelo de embedding e gera os embeddings, que serão armazenados em um repositório de vetores. Neste notebook, você usa o modelo Titan Embeddings para isso.

O segundo processo é a orquestração da solicitação do usuário, interação, invocação e retorno dos resultados. Isso envolve orquestrar a solicitação do usuário, interagir com os modelos e componentes necessários para reunir informações, invocar o chatbot para formular uma resposta e, então, retornar a resposta do chatbot de volta ao usuário.

Na Tarefa 4.1, você configura seu ambiente.

Execute a primeira célula de código para começar.

Execute a segunda célula de código para formatar instruções em um prompt de conversação e concluir o processo de configuração.

Na Tarefa 4.2, você habilita o chatbot a carregar contexto de conversação em múltiplas interações com usuários. Ter uma memória de conversação é crucial para que os Chatbots mantenham diálogos significativos e coerentes ao longo do tempo.

Você implementa capacidades de memória conversacional construindo em cima da classe InMemoryChatMessageHistory do LangChain. Este objeto armazena as conversas entre o usuário e o chatbot, e o histórico fica disponível para o agente do chatbot para que ele possa aproveitar o contexto de uma conversa anterior.

Execute a célula de código para iniciar a conversa.

A modelo respondeu com uma mensagem inicial.

Execute a célula de código para pedir dicas ao modelo sobre como começar um novo jardim.

O modelo fornece uma lista de dicas que ajudarão a começar um jardim.

Agora, execute a próxima célula de código para fazer uma pergunta sem mencionar a palavra jardim para ver se o modelo consegue entender a conversa anterior.

A nova saída mostra que o modelo ainda entende que você está falando sobre insetos no contexto de jardinagem.

Execute esta última célula de código para finalizar a conversa.

A saída mostra que a conversa terminou, e o modelo tem algumas palavras de despedida simpáticas para o usuário.

Na Tarefa 4.3, você usa o PromptTemplate padrão que é responsável pela construção dessa entrada. O LangChain fornece várias classes e funções para tornar a construção e o trabalho com prompts fáceis.

Execute a primeira célula de código para começar.

Execute a segunda célula de código para continuar.

Em seguida, execute a última célula de código para iniciar um bate-papo.

A saída é um prompt de bate-papo pronto para interação do usuário.

Perguntar "Qual é seu nome?" fornece algumas informações sobre o modelo. O modelo compartilha alguns antecedentes sobre sua existência.

Execute a célula de código final.

A saída é um log da interação com a interface de bate-papo.

Na Tarefa 4.4, o assistente de Inteligência Artificial (ou IA) desempenha o papel de um coach de carreira. Você pode informar o chatbot sobre sua persona (ou função) usando uma mensagem do sistema. Continue a aproveitar a classe InMemoryChatMessageHistory para manter o contexto de conversação.

Execute a primeira célula para criar a persona.

O coach de carreira em IA fornece uma lista de opções de carreira na área de IA.

Agora, execute a próxima célula de código para fazer uma pergunta que não esteja dentro da especialidade dessa persona. O modelo não deve responder a essa pergunta e deve dar uma razão para isso.

O coach de carreira de IA afirma que não pode dar conselhos sobre como consertar um carro porque não tem experiência suficiente.

Agora, execute esta célula de código final.

A saída é um histórico da interação com essa persona de bate-papo.

Na Tarefa 4.5, você pede ao chatbot para responder perguntas com base no contexto que foi passado a ele. Você pega um arquivo CSV e usa o modelo de embeddings do Titan para criar um vetor que representa esse contexto. Esse vetor é armazenado no Facebook AI Similarity Search (ou FAISS). Quando uma pergunta é feita ao chatbot, você passa esse vetor de volta para o chatbot e ele recupera a resposta usando o vetor.

Embeddings representam palavras, frases ou quaisquer outros itens discretos como vetores em um espaço vetorial contínuo. Isso permite que modelos de aprendizado de máquina realizem operações matemáticas nessas representações e capturem relacionamentos semânticos entre elas. Você usa embeddings para RAG.

Execute esta célula de código para configurar o modelo.

Para usar embeddings para pesquisa, você precisa de um armazenamento que possa executar pesquisas de similaridade de vetores de forma eficiente. Neste notebook, você usa o FAISS, que é um armazenamento na memória. Para armazenar vetores permanentemente, você pode usar Knowledge Bases para Amazon Bedrock, pgVector, Pinecone, Weaviate ou Chroma.

Execute esta célula de código para criar o armazenamento vetorial.

Um resumo dos documentos e do armazenamento de vetores é fornecido como saída.

Você pode usar uma classe Wrapper fornecida pelo LangChain para consultar o armazenamento do banco de dados de vetores e retornar os documentos relevantes.

Execute a célula de código para executar uma cadeia de controle de qualidade com todos os valores padrão.

O teste de baixo código é concluído e os resultados são apresentados como saída.

Para o chatbot, você precisa de gerenciamento de contexto, histórico, armazenamentos de vetores e muitos outros componentes.

Você começa construindo uma cadeia de Geração Aumentada de Recuperação (RAG) que suporte contexto.

Isso usa as funções **create_stuff_documents_chain** e **create_retrieval_chain** .

Execute esta célula de código para começar.

A saída mostra os blocos do documento consultados para chegar à resposta.

Em seguida, execute a célula de código para iniciar um bate-papo.

O chatbot está criado e pronto para uma conversa.

Você utilizou o Titan LLM para criar uma interface de conversação com os seguintes padrões:

- Chatbot (Básico - sem contexto)
- Chatbot usando modelo de prompt (Langchain)
- Chatbot com personas
- Chatbot com contexto

Você concluiu este notebook. Feche este arquivo de notebook e continue com a Tarefa 5.

Nesta tarefa, você usa um LLM para gerar código com base em um prompt de texto.

O prompt usado neste caderno é um prompt de tiro zero, pois não estamos fornecendo nenhum exemplo de texto além do prompt em si.

Para demonstrar a capacidade de geração de código de modelos no Amazon Bedrock, você pega o caso de uso de geração de código. Você explora o uso do cliente Boto3 para se comunicar com a API do Amazon Bedrock e fornece à API uma entrada que consiste em uma tarefa, uma instrução e uma entrada para o modelo sob o capô para gerar uma saída sem fornecer nenhum exemplo adicional. O propósito aqui é demonstrar como os LLMs poderosos entendem facilmente a tarefa em questão e geram saídas atraentes.

Neste cenário, você é um Analista de Dados, na AnyCompany. A empresa quer entender seu desempenho de vendas para diferentes produtos no último ano. Você recebeu um conjunto de dados chamado sales.csv. O conjunto de dados contém as seguintes colunas:

- Formato de data (AAAA-MM-DD)
- ID do produto (identificador exclusivo para cada produto)
- Preço (preço pelo qual cada produto foi vendido)

Neste notebook, você aprende como gerar código para um prompt dado. Você usa o Meta LLama 3 usando a API do Amazon Bedrock com o cliente Boto3.

Na Tarefa 5.1, você configura seu ambiente.

Execute a primeira célula de código para começar.

Na Tarefa 5.2, você prepara uma entrada para o serviço Amazon Bedrock para gerar um programa Python para seu caso de uso.

Execute esta célula de código para criar um arquivo de dados sales.csv de exemplo para este laboratório.

A saída mostra que o arquivo foi criado e podemos vê-lo em nossa lista de arquivos à esquerda.

Na Tarefa 5.3, você analisa as vendas com um programa Python gerado pelo Amazon Bedrock.

Execute a primeira célula de código para definir o modelo de prompt.

Execute a segunda célula de código para criar o prompt e analisar as vendas.

Por fim, execute a terceira célula de código para concluir a análise.

Na Tarefa 5.4, você invoca o modelo para gerar código.

O código foi gerado como saída.

Nesta seção opcional, você pode copiar e colar o código gerado e executá-lo para validação.

O código foi copiado para a célula de teste.

Após executá-lo, a saída mostra o produto da análise. Você pode usar essas informações para avaliar o código gerado.

Agora você experimentou usar o boto3 SDK, que fornece uma exposição vanilla à Amazon Bedrock API. Usando essa API, você gerou um programa Python para analisar e visualizar os dados de vendas fornecidos.

Você concluiu este notebook. Feche este arquivo de notebook e continue com a Tarefa 6.

Nesta tarefa, você aprenderá a usar um agente de planejamento e execução que determina a ordem das ações e as implementa usando as ferramentas disponíveis para os agentes.

Certos aplicativos exigem uma sequência adaptável de chamadas para os modelos de linguagem e vários utilitários para responder à pergunta de um usuário. A interface do Langchain Agent é flexível e pode integrar ferramentas externas com o raciocínio do LLM. Os agentes podem selecionar a ferramenta a ser usada com base na entrada do usuário. Os agentes são capazes de usar várias ferramentas e utilizar a saída de uma ferramenta como entrada para a próxima.

Na Tarefa 6.1, você configura seu ambiente.

Execute a primeira célula de código para começar.

Em seguida, você cria uma instância da classe ChatBedrock do LangChain, que permite interagir com um modelo de IA conversacional hospedado no Amazon Bedrock.

Execute a célula de código para continuar.

Por fim, execute a terceira célula de código para invocar o modelo, perguntando "O que é AWS?" e limitando a resposta a uma única frase.

A saída fornecida mostra informações adicionais, como tokens, ID do modelo e muito mais.

Na Tarefa 6.2, a estrutura ReAct permite que grandes modelos de linguagem interajam com ferramentas externas para obter informações adicionais que resultem em respostas mais precisas e baseadas em fatos.

Grandes modelos de linguagem podem gerar explicações para seu raciocínio e respostas específicas para tarefas de forma alternada.

Produzir explicações de raciocínio permite que os modelos infiram, monitorem e revisem planos de ação, e até mesmo lidem com cenários inesperados. A etapa de ação permite que os modelos interajam e obtenham informações de fontes externas, como bases de conhecimento ou ambientes.

Execute esta célula de código para iniciar a tarefa.

Na próxima célula, você define uma função `get_product_price` que serve como uma ferramenta dentro do framework Langchain e recupera o preço do produto especificado na consulta do arquivo `sales.csv` criado na tarefa anterior. É uma implementação simples para ilustrar como as ferramentas podem ser projetadas para trabalhar com o framework Langchain.

Execute esta célula de código para continuar.

Na próxima célula, você define uma calculadora de função que serve como uma ferramenta dentro da estrutura Langchain. Essa ferramenta permite que um modelo de linguagem execute cálculos matemáticos avaliando uma dada expressão usando a biblioteca numexpr do Python. A ferramenta é projetada para lidar com casos em que a expressão é inválida. Nesse caso, a ferramenta pede ao modelo para repensar sua abordagem para o cálculo.

Execute esta célula para continuar.

Execute a próxima célula para continuar.

Execute esta célula de código para executar funções auxiliares para imprimir a saída de rastreamento em um arquivo.

Na Tarefa 6.3, você criará um gráfico de agente para um sistema de IA conversacional que pode interagir com ferramentas externas. O gráfico de agente é uma máquina de estado que define o fluxo da conversa e a interação com as ferramentas.

Nesta célula de código, você define nós com funções associadas que atualizam o estado com base na entrada. Conecte nós usando arestas, onde o gráfico faz a transição de um nó para o próximo. Incorpore arestas condicionais para rotear o gráfico para diferentes nós com base em condições específicas. Por fim, compile o gráfico do agente para prepará-lo para execução, manipulando transições e atualizações de estado conforme definido.

Execute a célula para continuar.

Em seguida, você visualiza o gráfico compilado. Observe que a transição para fora do nó do agente é condicional, conforme indicado pela linha pontilhada.

Execute a célula de código para continuar.

A saída é a visualização gerada, exatamente como esperado.

Na próxima célula, você executa a função auxiliar para imprimir a saída do gráfico.

Execute a célula de código para continuar.

Em seguida, adicione uma ou mais perguntas que você deseja fazer ao agente sobre preços de produtos do arquivo sales.csv que você criou no notebook anterior.

Execute a célula de código para continuar.

Para entender as etapas envolvidas no raciocínio, habilite o trace. No entanto, mantenha a saída do trace gerenciável **comentando todas as perguntas, exceto uma,** na lista acima. Como alternativa, você pode desabilitar o trace e executar

todas as perguntas.

Execute a célula de código para continuar.

Na etapa final, você invoca o agente com a pergunta da lista acima. A pergunta que o agente responderá é "Quanto custará comprar 3 unidades de P002 e 5 unidades de P003?"

Execute a célula de código para continuar.

O agente começa a executar os cálculos e produz a saída passo a passo.

Os cálculos foram concluídos e o agente tem uma resposta: "O custo total para comprar 3 unidades de P002 e 5 unidades de P003 é de US$ 530."

Você concluiu este notebook. Feche este arquivo de notebook e continue o curso.

Após concluir essas tarefas, você terá feito o seguinte com sucesso:

- Realizada geração de texto.
- Criei resumo de texto.
- Usei o Amazon Bedrock para responder perguntas.
- Criei um chatbot.
- Modelos Amazon Bedrock usados para geração de código.
- Modelos Amazon Bedrock integrados com agentes LangChain.

Obrigado!

Usando Bases de Conhecimento

1. RAG (Recuperação e Geração)

- **Objetivo**: O RAG permite que organizações tenham maior controle sobre as respostas geradas por sistemas de IA. Ele integra o modelo de linguagem (FM) com a capacidade de recuperar informações relevantes de fontes externas confiáveis, aumentando a credibilidade e a precisão das respostas fornecidas.

2. Ingestão de Dados

- **Processo**: O RAG depende da incorporação de dados externos para complementar o treinamento do FM. Esses dados podem vir de APIs, bancos de dados, ou repositórios de documentos e são convertidos em representações numéricas chamadas **embeddings**.
- **Chunking**: Antes da criação de embeddings, os documentos são segmentados em partes menores para facilitar a busca semântica.
- **Modelo de Embeddings**: Um modelo especializado converte o texto segmentado em embeddings. É importante verificar os idiomas suportados por esses modelos.

3. Recuperar Informações Relevantes

- **Busca Semântica**: Após os dados serem ingeridos e armazenados em um repositório de vetores, a consulta do usuário é convertida em uma representação vetorial. Isso permite que o sistema realize uma busca semântica nos dados, retornando as informações mais relevantes.
- **Exemplo de Uso**: Um assistente de IA de recursos humanos pode responder a perguntas como "Quais são meus benefícios de saúde?" ao buscar documentos específicos relacionados aos benefícios do funcionário.

4. Aumentar o Prompt FM

- **Técnicas de Engenharia de Prompt**: A entrada do usuário é aumentada com os dados recuperados, criando um contexto mais completo para o modelo de linguagem. Isso garante que o FM gere respostas precisas e relevantes, baseadas nos dados externos recuperados.

5. Geração

- **Passagem do Prompt Aumentado**: O prompt aumentado é passado para o modelo de fundação (FM) para gerar a resposta final, utilizando a combinação do contexto do usuário e os dados recuperados.

Esses passos formam um fluxo de trabalho que integra recuperação e geração de informações para fornecer respostas mais precisas e contextualmente ricas aos usuários.

Bases de Conhecimento da Amazon Bedrock

- **Recurso Gerenciado**: O Amazon Bedrock Knowledge Bases é uma solução totalmente gerenciada para implementar o fluxo de trabalho RAG (Recuperação e Geração) sem a necessidade de criar integrações personalizadas ou gerenciar fluxos de dados.
- **Integração com Fontes Privadas**: Permite fornecer aos modelos de linguagem (FMs) e agentes informações contextuais provenientes de fontes de dados privadas da empresa, garantindo que as respostas sejam mais relevantes, precisas e personalizadas.
- **Gerenciamento de Contexto de Sessão**: Inclui o gerenciamento integrado do contexto de sessão, o que facilita a criação de aplicativos que suportam conversas multi-turn, ou seja, interações contínuas entre o usuário e a IA.

Esse recurso simplifica a implementação de RAG e permite um gerenciamento eficiente de dados contextuais para melhorar a performance dos modelos de IA.

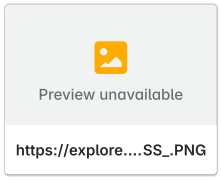
Tutoriais:

Criar uma base de conhecimento

Criar uma base de conhecimento

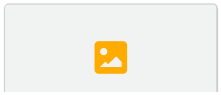
Você pode usar as etapas a seguir para criar uma base de conhecimento usando o Amazon Bedrock.

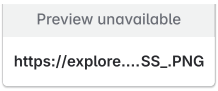
Divisor numerado 1



Amazon Bedrock Crie uma base de conhecimento

Divisor numerado 2





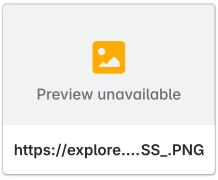
Detalhes e permissões da base de conhecimento do Amazon Bedrock

o nome e a descrição uma função de tempo de execução para

Divisor numerado 3

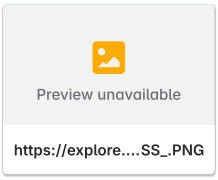
Você pode ingerir conteúdo da web e de repositórios como Amazon Simple Storage Service (Amazon S3), Confluence (pré-visualização), Salesforce (pré-visualização), SharePoint (pré-visualização). Após escolher a fonte de dados, configure os detalhes da fonte de dados (não mostrados).

Depois de apontar para o local dos seus dados proprietários, o Amazon Bedrock Knowledge Bases busca os documentos automaticamente.



Fontes de dados da base de conhecimento

Divisor numerado 4



Escolha do modelo de incorporação do Amazon Bedrock

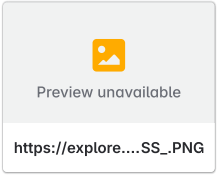
Escolha o modelo de incorporação que você preferir.

Depois que o conteúdo é ingerido, as Bases de Conhecimento dividem o conteúdo em blocos de texto e convertem o texto em incorporações.

Divisor numerado 5

Escolha seu banco de dados de vetores preferido ou deixe que a Amazon Bedrock crie um repositório de vetores **Amazon OpenSearch Serverless** para você. Como alternativa, você pode especificar um repositório de vetores existente em um dos bancos de dados suportados, incluindo OpenSearch Serverless, Pinecone e Redis Cloud, Amazon Aurora e MongoDB.

As Knowledge Bases então armazenam os embeddings no seu banco de dados de vetores. As Knowledge Bases também gerenciam complexidades de fluxo de trabalho, como comparação de conteúdo, tratamento de falhas, controle de throughput, criptografia e muito mais.



Opções de banco de dados de vetores do Amazon Bedrock

Personalização de Bases de Conhecimento para Respostas Precisos

- **Ajuste de Recuperação e Ingestão**: Permite melhorar a precisão da recuperação de dados para casos de uso específicos, com opções avançadas de análise para lidar com dados não estruturados, como PDFs e imagens digitalizadas, e conteúdo

complexo, como tabelas.

- **Chunking Personalizado**: Você pode criar seu próprio código de chunking como uma função Lambda para segmentar dados de maneira personalizada ou utilizar as opções integradas, como chunking de tamanho fixo, sem chunking, chunking hierárquico ou semântico.
- **Reformulação de Consulta**: Estratégias de reformulação de consulta ajudam a melhorar a compreensão do sistema sobre consultas complexas no momento da recuperação de dados.

Recuperação de Dados e Aumento de Prompts

- **API Retrieve**: Utilizada para buscar resultados relevantes de uma consulta em bases de conhecimento.
- **API RetrieveAndGenerate**: Vai além da recuperação, utilizando os resultados obtidos para aumentar o prompt do modelo de linguagem e retornar a resposta.
- **Integração com Amazon Bedrock Agents**: Permite adicionar bases de conhecimento diretamente aos agentes, fornecendo informações contextuais durante as interações.

Atribuição de Fonte

- **Citações para Transparência**: Todas as informações recuperadas de bases de conhecimento vêm com citações para garantir maior transparência e reduzir as alucinações. A opção de visualizar os detalhes da fonte está disponível ao testar a base de conhecimento.

Amazon Bedrock Knowledge Bases: Funcionamento e Considerações

Pré-processamento de Dados

- **Fragmentação de Documentos**: Antes de armazenar os dados em um banco de dados vetorial, é necessário dividir os documentos em pedaços menores e mais gerenciáveis. Esses pedaços são convertidos em embeddings (representações numéricas), que ajudam na recuperação semântica eficiente.
- **Indexação de Embeddings**: Os embeddings gerados são indexados e armazenados em um banco de dados vetorial, mantendo a referência ao documento original para recuperação rápida e precisa.
- **Objetivo**: Facilitar a comparação semântica entre consultas de usuários e textos nos dados indexados, para garantir respostas mais precisas durante a interação com o sistema.

Tempo de Execução

- **Conversão de Consulta em Vetor**: Quando um usuário faz uma consulta, ela é convertida em um vetor utilizando o mesmo modelo de incorporação usado para a base de conhecimento.
- **Recuperação de Blocos Semânticos**: O vetor da consulta é comparado com os vetores armazenados no banco de dados vetorial para recuperar os blocos de texto semanticamente mais relevantes.
- **Aumento do Prompt**: O contexto adicional dos blocos recuperados é adicionado ao prompt do usuário para enriquecer a resposta gerada pelo modelo.

Otimização da Arquitetura RAG

- **Refinamento de Dados e Chunking**: Para uma recuperação mais eficiente, você pode aprimorar o processo de chunking (segmentação) e melhorar a forma como os embeddings e o banco de dados vetorial são usados.
- **Avaliação e Experimentação**: O processo de aprimoramento é iterativo e exige testes rigorosos para verificar a eficácia das configurações de chunking e recuperação.

Considerações para Escolha do Tamanho do Bloco

- **Impacto no Desempenho**: O tamanho do bloco influencia diretamente a eficiência da recuperação e a qualidade do contexto fornecido ao modelo. Blocos menores melhoram a velocidade, mas podem carecer de contexto. Blocos maiores preservam mais contexto, mas podem ser mais caros computacionalmente.
- **Fatores a Considerar**:

- **Natureza do Conteúdo**: Determine o tipo de documento e a forma como ele é melhor fragmentado.
- **Desempenho do Modelo de Incorporação**: Avalie qual o tamanho de bloco ideal para o modelo de incorporação escolhido.
- **Complexidade das Consultas**: A natureza das perguntas dos usuários pode influenciar o tamanho do bloco necessário para respostas precisas.
- **Uso Final dos Resultados**: Considere como os blocos recuperados serão aplicados na prática.

Estratégias de Chunking

- **Fragmentação de Comprimento Fixo**: Divida os textos em blocos de tamanho fixo, o que é simples, mas pode quebrar o contexto e afetar a coerência.
- **Fragmentação Sensível ao Conteúdo**: Inclui abordagens mais avançadas, como fragmentação em nível de frase, chunking recursivo e chunking especializado (como em Markdown ou LaTeX), preservando melhor o contexto semântico.
- **Fragmentação Semântica**: Baseia-se na segmentação do texto por tópicos ou segmentos coerentes, utilizando técnicas de PNL. Embora mais cara, preserva melhor a coerência semântica.
- **Abordagens Híbridas**: Combina diferentes técnicas, como chunking em nível de frase com critérios adicionais como coerência semântica ou comprimento do bloco.

Seleção de Modelos de Incorporação

- **Importância das Incorporações**: Modelos de incorporação transformam texto em representações vetoriais, que são fundamentais para a recuperação semântica eficiente. O modelo escolhido impacta diretamente a qualidade e a velocidade da recuperação.
- **Fatores a Considerar**:
  - **Dimensionalidade**: O tamanho dos vetores de incorporação (ex: 256, 512, 1024 dimensões) afeta o equilíbrio entre precisão e eficiência computacional.
  - **Suporte Multilíngue**: Se a base de conhecimento precisa suportar múltiplos idiomas, é importante escolher um modelo que tenha suporte nativo ou estendido a esses idiomas.
  - **Desempenho**: Avalie a precisão da recuperação e a qualidade da resposta gerada em benchmarks para escolher o modelo mais adequado.
  - **Integração e Compatibilidade**: Certifique-se de que o modelo de incorporação seja compatível com as bibliotecas e frameworks existentes.

Exemplos de Modelos de Incorporação

- **Amazon Titan Text Embeddings**: O modelo Titan Text V2 suporta tamanhos flexíveis de embeddings e oferece alta precisão mesmo com dimensões menores. Ele também suporta mais de cem idiomas, o que o torna ideal para aplicações multilíngues.
- **Cohere Embed**: Suporta embeddings de 1.024 dimensões, com versões em inglês e multilíngue, adequados para diversos casos de uso.

Considerações para Escolha de Banco de Dados Vetorial

- **Função do Banco de Dados Vetorial**: Serve como a espinha dorsal do processo de recuperação, armazenando os embeddings de vetores e permitindo uma recuperação eficiente das informações relevantes durante o RAG.
- **Fatores para Avaliar**:
  - **Escalabilidade**: O banco de dados precisa lidar com grandes volumes de embeddings e suportar consultas rápidas à medida que o conjunto de dados cresce.
  - **Desempenho e Custo**: Avaliar a capacidade de processamento do banco de dados, especialmente em termos de indexação e recuperação eficiente.
  - **Integração com Infraestrutura**: O banco de dados deve ser fácil de integrar com a infraestrutura existente, garantindo uma implementação suave e mínima sobrecarga operacional.

Volume de Dados e Escalabilidade

- O banco de dados de vetores precisa ser capaz de lidar com grandes volumes de embeddings e suportar indexação eficiente conforme o conjunto de dados cresce. Fatores como número de vetores, dimensionalidade e requisitos de throughput devem ser cuidadosamente avaliados.

Esses fatores são essenciais para garantir que o sistema RAG, utilizando o Amazon Bedrock, seja eficiente, escalável e capaz de fornecer respostas precisas e contextualmente relevantes para os usuários.

Desempenho da Consulta e Fatores de Seleção de Banco de Dados Vetorial

Ao escolher um banco de dados vetorial para uso com o Amazon Bedrock Knowledge Bases, diversos fatores devem ser considerados para otimizar o desempenho e a escalabilidade do sistema:

- **Desempenho da Consulta:** O banco de dados de vetores deve oferecer recursos rápidos e eficientes para permitir a recuperação de vetores relevantes com base em consultas de similaridade, especialmente em cenários de baixa latência.
- **Filtragem por Metadados:** A filtragem de metadados pode melhorar a qualidade da pesquisa, pré-filtrando os dados armazenados para resultados mais relevantes.
- **Pesquisa Híbrida:** Utiliza diferentes algoritmos de busca para melhorar a qualidade e relevância dos resultados de pesquisa, combinando as forças de várias abordagens de busca.
- **Integração e Ecossistema:** O banco de dados de vetores deve ser compatível com a infraestrutura existente, incluindo suporte para linguagens de programação, bibliotecas de clientes e integração com outros serviços e ferramentas.
- **Opções de Implantação:** Considerar se o banco de dados de vetores será auto-hospedado, hospedado na nuvem ou gerenciado, conforme as necessidades operacionais e preferências de infraestrutura da organização.
- **Modelo de Custo e Preço:** Avaliar o custo de armazenamento, consulta e taxas adicionais com base nos padrões de uso específicos.
- **Segurança e Conformidade:** O banco de dados deve fornecer recursos adequados de segurança e conformidade, especialmente em casos de dados confidenciais ou regulamentados.
- **Desempenho e Latência:** A latência das consultas e o throughput (taxa de dados processados) são cruciais para garantir a performance desejada.
- **Disponibilidade e Durabilidade dos Dados:** Avaliar as garantias de replicação, tolerância a falhas e recursos de recuperação de desastres para garantir a integridade e a acessibilidade contínua dos dados.

Armazenamento de Vetores Suportado pelo Amazon Bedrock:

Amazon Bedrock Knowledge Bases oferece suporte a diversos bancos de dados vetoriais, incluindo:

- OpenSearch Serverless
- Amazon Aurora (compatível com PostgreSQL)
- Pinecone
- Nuvem Redis
- Atlas MongoDB

Sincronização de Documentos com Amazon Bedrock Knowledge Bases:

Amazon Bedrock facilita a implementação do fluxo de trabalho RAG (Recuperação de Dados e Geração) de forma totalmente gerenciada. Isso inclui adicionar, modificar ou excluir documentos em sua base de conhecimento sem a necessidade de integrações personalizadas.

- **Fontes de Dados Suportadas:** Você pode usar fontes como Amazon S3, Confluence, Salesforce e SharePoint (em pré-visualização).
- **Modelo de Incorporação:** Amazon Titan Embeddings é um exemplo de modelo de incorporação usado para converter dados em vetores.
- **Banco de Dados de Vetores:** OpenSearch Serverless pode ser usado para armazenar vetores.

A atualização dos dados é feita por meio do **AWS Management Console** ou por uma função **AWS Lambda**, que pode ser programada para sincronizar dados incrementalmente. Esse processo pode ser acionado por eventos ou ser realizado de forma

programada. A AWS recomenda o uso de uma fila de eventos para gerenciar múltiplos eventos simultâneos.

Resumindo o Processo:

- O fluxo de dados envolve transformar dados em vetores através de um modelo de incorporação, armazená-los no banco de dados vetorial, e usar essas informações para melhorar a recuperação e a geração de respostas em tempo real.
- A sincronização dos dados pode ser feita de forma incremental, garantindo que os dados atualizados sejam refletidos rapidamente no banco de conhecimento.

Métodos de API no Amazon Bedrock Knowledge Bases para RAG:

1. **Recuperar (Retrieve):**
   - Recupera consultas de uma base de conhecimento para buscar informações relevantes com base na solicitação do usuário.
   - Usado quando se deseja personalizar a parte de geração do RAG.
2. **RetrieveAndGenerate:**
   - Vai além do "Retrieve", utilizando as informações recuperadas para enriquecer o prompt do modelo de linguagem (FM).
   - A resposta gerada cita apenas fontes relevantes da base de conhecimento, com integração de contexto de sessão para suportar conversas multi-turn.

Opções de Personalização do RetrieveAndGenerate:

1. **Prompt Personalizado:**
   - Permite substituir o modelo de prompt padrão com um personalizado, ajustando o tom, formato de saída e comportamento do modelo ao gerar respostas.
2. **Tipo de Pesquisa:**
   - Controla o tipo de pesquisa utilizada para recuperar resultados do repositório de vetores.
   - Pode ser personalizada para usar pesquisa semântica (embeddings de vetores) ou híbrida (embeddings de vetores + texto bruto), dependendo do repositório de vetores escolhido.
3. **Número Máximo de Resultados a Recuperar:**
   - Por padrão, o Amazon Bedrock retorna até cinco resultados, mas você pode configurar para retornar até 100 resultados, conforme necessário.
4. **Metadados e Filtragem:**
   - Permite adicionar metadados aos documentos, que podem ser usados para aplicar filtros durante as consultas.
   - Utiliza o parâmetro de configuração de recuperação (retrievalConfiguration) para otimizar as consultas.
5. **Reformulação de Consulta:**
   - Usando o parâmetro de configuração de orquestração (orchestrationConfiguration), você pode dividir consultas complexas em subconsultas mais simples, melhorando a precisão e relevância dos resultados.
6. **Guarda-corpos (Guardrails):**
   - Permite definir regras de segurança e controle sobre a saída gerada pela base de conhecimento, como bloquear tópicos indesejáveis ou conteúdo prejudicial.
   - Configurações de guardrails personalizam e filtram as respostas para garantir conformidade com as políticas de uso e ética.

Parâmetros de Entrada no RetrieveAndGenerate:

1. **Campos de Entrada Obrigatórios:**
   - **Entrada de consulta:** A consulta do usuário que será processada.
   - **ID da base de conhecimento:** Identificador da base de dados a ser consultada.
   - **FM para geração de resposta:** O modelo de linguagem usado para gerar a resposta a partir das informações recuperadas.
2. **Campos de Entrada Opcionais:**

- **Modelo de prompt:** Permite personalizar o formato e o tom do prompt enviado ao modelo de linguagem.
- **Número total de resultados retornados:** Define quantos resultados a consulta pode retornar, com um limite máximo configurável.
- **Tipo de pesquisa:** Controla o tipo de pesquisa utilizada, como semântica ou híbrida, dependendo das necessidades.
- **ID da sessão com chave de criptografia KMS:** Usado para garantir a segurança e a privacidade da sessão de consulta.

RAG Personalizado Usando Retrieve:

- A chamada de API **Retrieve** permite um controle mais granular sobre o fluxo de trabalho padrão, dando a possibilidade de personalizar como os resultados da pesquisa semântica são processados e integrados. Além disso, pode ser integrada com outras ferramentas de orquestração, como o LangChain.

Campos de Entrada no Retrieve:

1. **Campos de Entrada Obrigatórios:**
   - **Texto de entrada da consulta:** O conteúdo da consulta feita pelo usuário.
2. **Campos de Entrada Opcionais:**
   - **Próximo token:** Usado para obter o próximo conjunto de resultados, se necessário.
   - **Número total de resultados retornados:** Define o limite de resultados que a consulta pode retornar.
   - **Tipo de pesquisa:** Especifica o tipo de pesquisa a ser realizada, como semântica ou híbrida.
   - **Filtros:** Permite aplicar filtros adicionais para refinar os resultados da consulta.
   - **Reclassificação personalizada:** Técnica para refinar e melhorar a qualidade dos resultados retornados, ajustando a relevância dos documentos.

Técnicas de Reclassificação:

1. **Reclassificação Personalizada:**
   - Refina os resultados recuperados, selecionando os documentos mais relevantes com base em uma função personalizada ou outro critério.
2. **Usando FM para Reclassificação:**
   - Em vez de usar uma função personalizada, um modelo de linguagem (FM) pode ser utilizado para classificar os documentos recuperados em ordem de relevância, garantindo que os mais relevantes sejam apresentados ao usuário.

Avaliação de Aplicativos RAG:

A avaliação de aplicativos RAG é essencial para garantir a qualidade e o desempenho de sistemas de processamento de linguagem natural que combinam modelos de linguagem com fontes externas de conhecimento. Uma avaliação bem-feita permite identificar pontos fortes, fracos e potenciais vieses, além de otimizar o desempenho e aumentar a confiabilidade dos aplicativos.

Framework RAGAS:

O **RAGAS** (RAG Assessment Framework) é uma estrutura de código aberto usada para avaliar pipelines RAG. Ele fornece métricas específicas para avaliar cada componente do aplicativo RAG, como as respostas geradas, as recuperações (resultados de pesquisa) e o desempenho geral do sistema.

As principais métricas que o RAGAS utiliza para avaliação são:

1. **Fidelidade:**
   - Mede a consistência factual das respostas geradas em relação ao contexto recuperado.
   - Se as alegações feitas pela resposta podem ser inferidas do contexto dado, a resposta é considerada fiel.
   - A pontuação varia de 0 a 1, sendo que valores mais altos indicam melhor fidelidade.

2. **Relevância da Resposta:**
    - Avalia o quão pertinente a resposta gerada é em relação ao prompt dado.
    - A pontuação é baseada na comparação entre a pergunta original e perguntas artificiais geradas.
    - Respostas incompletas ou redundantes têm pontuações mais baixas, enquanto respostas mais relevantes recebem pontuações mais altas.
3. **Recordação de Contexto (Context Recall):**
    - Mede a extensão em que o contexto recuperado corresponde à resposta correta, considerada a "verdade fundamental".
    - A pontuação varia entre 0 e 1, sendo que valores mais altos indicam uma maior correspondência entre o contexto e a resposta.
4. **Precisão de Contexto (Context Precision):**
    - Avalia se todos os itens relevantes do contexto recuperado são classificados corretamente nas primeiras posições.
    - Pontuações mais altas indicam que os itens relevantes foram bem classificados, enquanto pontuações mais baixas indicam pior desempenho na classificação.

Essas métricas ajudam a garantir que o sistema RAG seja eficiente, preciso e confiável ao integrar dados externos com modelos de linguagem.

Dicas para Construir Aplicações RAG Bem-Sucedidas:

1. **Definindo o Domínio e os Dados Associados:**
    - Defina claramente o domínio da aplicação para determinar as fontes de dados necessárias.
    - Por exemplo, um assistente de IA de suporte ao cliente exigiria manuais de produtos e FAQs, enquanto um sistema médico de perguntas e respostas necessitaria de bancos de dados médicos e artigos de pesquisa.
    - Utilize fontes de dados de alta qualidade e bem curadas para garantir informações confiáveis.
2. **Estratégia de Chunking:**
    - Use o chunking de dados para transformá-los de maneira eficiente, facilitando a recuperação e extração de valor no futuro.
    - A questão importante não é como fragmentar os dados, mas como transformá-los de maneira que favoreça as tarefas do modelo e a recuperação de informações relevantes.
3. **Adicionando Metadados:**
    - Utilize filtros de metadados para recuperar pedaços semanticamente relevantes e bem definidos, restringindo o espaço de pesquisa e melhorando a precisão dos resultados.
4. **Engenharia Rápida:**
    - Personalize a geração em bases de conhecimento para controlar os prompts e padronizar os modelos de prompt.
    - Escolha o modelo de fundação adequado ao seu caso de uso e garanta que os prompts enviados ao modelo sejam eficazes e produzam bons resultados.
5. **Avaliação e Refinamento:**
    - Realize avaliações contínuas do desempenho do seu aplicativo RAG, utilizando métricas e feedback do usuário.
    - O refinamento iterativo envolve ajustar prompts, estratégias de recuperação e melhorar as respostas geradas para aumentar a eficácia do sistema.
    - Utilize frameworks como o RAGAS para realizar avaliações e otimizar seu aplicativo.

Técnicas Avançadas de RAG (Retrieval-Augmented Generation):

1. **Pesquisa Híbrida:**
    - Combina múltiplas abordagens de recuperação de informação, como correspondência por palavras-chave e busca semântica.
    - Aumenta a eficácia na recuperação de dados relevantes, mesmo em consultas complexas ou pouco convencionais.
    - O Amazon Bedrock seleciona automaticamente a melhor técnica com base na consulta e nos dados.

2. **Reformulação ou Decomposição de Consulta:**
    - Divide consultas complexas em subconsultas mais simples e específicas.
    - Cada subconsulta passa por um processo de recuperação separado para obter informações mais precisas.
    - Os resultados são posteriormente agrupados e ordenados antes da geração da resposta pelo modelo.
3. **Cache Semântico:**
    - Armazena pares de consulta/resposta anteriores, incluindo os dados recuperados.
    - Quando uma nova consulta semelhante é feita, o sistema reutiliza as informações do cache.
    - Isso reduz o tempo de processamento e aumenta a eficiência, sendo útil em cenários com perguntas recorrentes.
4. **Protegendo Aplicações RAG:**
    - O Amazon Bedrock Guardrails protege contra a geração de conteúdo inadequado ou tendencioso.
    - Recursos incluem:
        - Detecção de alucinações e bloqueio de tópicos indesejados.
        - Redação de dados sensíveis.
        - Filtragem de conteúdo para evitar discurso de ódio, insultos, conteúdo sexual, violência e má conduta.
        - Uso de filtros personalizados e proteção contra ataques de injeção rápida e jailbreak.
    - Promove o desenvolvimento responsável e seguro de aplicações baseadas em IA.

Tutoriais:

Configurando e executando laboratórios de notebook do SageMaker para bases de conhecimento

Configurando e executando laboratórios de notebook do SageMaker para bases de conhecimento do Amazon Bedrock

Neste laboratório, você cria um aplicativo de perguntas e respostas usando a base de conhecimento da AnyCompany e a API Retrieve e RetrieveAndGenerate da Amazon Bedrock. Você aproveita a base de conhecimento existente, que contém informações abrangentes sobre os produtos, serviços e detalhes corporativos da AnyCompany, como histórico, liderança, desempenho financeiro, esforços de sustentabilidade e muito mais. Você percorre vários notebooks que podem responder efetivamente a perguntas relacionadas aos produtos, serviços e informações corporativas da AnyCompany.

Nesta primeira tarefa, você configurará seu ambiente de laboratório para poder concluir esses laboratórios. Isso exigirá que você solicite acesso ao modelo no Amazon Bedrock, bem como configure seu ambiente de laboratório no Amazon SageMaker Studio.

Para começar, você precisará baixar os arquivos do Jupyter notebook que você precisará para concluir os laboratórios. Há 6 arquivos de notebook no total, bem como 1 arquivo PDF e 6 arquivos de imagem que serão necessários para alguns dos laboratórios.

Depois de baixar os arquivos para seu computador local, anote o local. Você precisará acessar o conteúdo para executar esses laboratórios.

Para usar o Bedrock, os usuários de conta com as Permissões IAM corretas devem habilitar o acesso aos modelos de fundação Bedrock (FMs) disponíveis. Para selecionar os modelos que você precisa, certifique-se de estar na região us-east-1.

Na parte superior do AWS Management Console, na barra de pesquisa, procure e escolha Amazon Bedrock.

Vá para a página do console do Amazon Bedrock. Selecione Get Started.

Se você vir um pop-up de boas-vindas, selecione Gerenciar acesso ao modelo. Caso contrário, use o menu de navegação no lado esquerdo e role para baixo até Acesso ao modelo.

Para adicionar ou atualizar seu acesso ao modelo existente, escolha Habilitar modelos específicos.

Na página Editar acesso ao modelo, localize os seguintes modelos e use as caixas de seleção para selecioná-los, caso ainda não tenha acesso a eles:

- Titan Embeddings G1 – Texto
- Titan Text G1 - Premier

Depois role para baixo.

Nota: Se você notar que o status de acesso já mostra Acesso concedido, nenhuma ação é necessária. Se você não vir o Titan Text G1 – Premier como uma das opções, certifique-se de que você está na região us-east-1.

Por fim, encontre Claude 3 Sonnet. Verifique se ele está selecionado e, se não estiver, use a caixa de seleção para selecioná-lo.

Depois role para baixo.

Depois de verificar se todos os modelos estão selecionados ou já estão disponíveis, escolha Avançar.

Revise suas seleções, se aplicável, e escolha Enviar.

Depois que seu acesso for concedido, você poderá usar esses modelos com os cadernos de laboratório.

Nesta etapa, você configurará e revisará a base de conhecimento que usará com os arquivos do Jupyter Lab Notebook.

Para começar, escolha S3 na barra de favoritos. Se você não tiver uma barra de favoritos, insira-a na barra de pesquisa e escolha-a na lista de serviços.

Selecione criar bucket.

Insira um nome exclusivo para seu bucket RAG e role para baixo.

Deixe todo o resto como padrão e escolha Criar bucket.

O bucket RAG S3 foi criado com sucesso.

Carregue o documento financeiro na pasta configurada para RAG. Selecione Upload.

Para selecionar o documento, escolha Adicionar arquivos.

Escolha o arquivo AnyCompany_financial_10K.pdf nos arquivos de laboratório que você baixou e extraiu anteriormente.

Para finalizar, selecione Upload.

Seu arquivo foi carregado com sucesso.

No campo de pesquisa, digite Amazon Bedrock.

Escolha Amazon Bedrock na lista de serviços.

Selecione Começar.

Selecione Bases de conhecimento no menu de navegação.

Para começar, escolha Criar base de conhecimento.

Em Nome da base de conhecimento, insira e2e-rag-knowledgebase.

Em IAM permissions, certifique-se de que Create and use a new service role esteja selecionado. Mantenha o nome padrão e role para baixo.

Certifique-se de que Amazon S3 esteja selecionado em Escolher fonte de dados.

Deixe todo o resto como padrão e escolha Avançar.

Para o nome da fonte de dados, insira e2e-rag-knowledgebase-ds.

Em seguida, escolha Procurar S3 para selecionar o bucket que você acabou de criar.

Selecione o bucket que você criou no S3 que contém o arquivo para RAG.

Deixe todo o resto como padrão e escolha Avançar.

Em Embeddings model, escolha Titan Embeddings G1 – Text v1.2 e role para baixo.

Deixe todo o resto como padrão e escolha Avançar.

Revise os detalhes da base de conhecimento conforme você rola até o final da página.

Para finalizar, escolha Criar base de conhecimento.

Seu banco de dados de vetores está sendo preparado no Amazon Opensearch Serverless.

A base de conhecimento foi criada com sucesso. O banner fornece um lembrete de que a próxima etapa é sincronizar a fonte de dados. Selecione o botão Ir para fontes de dados no banner.

Com sua fonte de dados selecionada, escolha o botão Sincronizar.

O status mostra que a fonte está sincronizando. Aguarde até que ela apareça como disponível.

A sincronização está completa. Agora é hora de configurar o Amazon SageMaker.

Nesta etapa, você iniciará um aplicativo Amazon SageMaker Studio para acessar seu ambiente de laboratório.

Na parte superior do AWS Management Console, na barra de pesquisa, procure e escolha Amazon SageMaker.

Para prosseguir, você precisará configurar um domínio. Selecione Configurar para usuário único.

Você verá um banner Preparing SageMaker Domain. Esta é uma configuração única e levará vários minutos.

Agora você precisa definir permissões de função. Primeiro, role para baixo até Autenticação e permissões.

Anote a função Execução Padrão, você precisará fazer atualizações nas permissões para essa função.

Agora você precisa definir permissões de função. Use as 3 barras para expandir o menu de navegação.

Selecione Gerente de função no menu de navegação.

Como você está trabalhando com uma função estabelecida, escolha Exibir funções no gerenciador do IAM.

Escolha o link para a função que você acabou de criar.

Para começar, escolha Adicionar permissões e depois Criar política em linha.

Na barra de pesquisa Selecionar um serviço, pressione Enter e escolha Bedrock, depois escolha Avançar.

Expanda a seção List. Escolha List Guardrails e List Knowledge Bases.

Expanda a seção Read. Escolha ApplyGuardrail, InvokeModel e Retrieve.

Expanda a seção Write. Escolha CreateGuardrail e Retrieve and Generate, depois role para baixo.

Expanda a seção Tagging. Escolha TagResource nesta seção.

Por fim, em Recursos, escolha Todos. Depois, escolha Avançar.

Use o campo Nome da política para dar um nome à sua política e selecione Criar política.

Sua política de permissões foi adicionada e está em vigor.

Volte para a aba com Amazon Sagemaker. Escolha Domains no menu de navegação, selecione o domínio atual e, em seguida, escolha a aba User Profiles.

Selecione o botão Iniciar ao lado do perfil de usuário que foi criado e selecione Estúdio.

Quando estiver no Studio, você poderá ver um pop-up de tour. Selecione Skip Tour por enquanto.

Quando estiver no Studio, em Aplicativos, escolha o botão JuypterLab.

Selecione o botão Criar espaço JupyterLab.

Dê um nome ao seu espaço de laboratório e escolha Criar Espaço.

Depois que o Lab Space for criado, escolha o botão Run Space. O Status mudará para Starting.

Quando o status mudar para Em execução, escolha o botão Abrir JupyterLab para iniciar o Studio em uma nova aba.

Nesta etapa, você preparará seus recursos de laboratório. Isso inclui carregar arquivos de notebook e executar atualizações no ambiente de laboratório antes de começar suas tarefas.

Assim que o JupyterLabs abrir, você estará trabalhando com a pasta raiz. É aqui que você fará upload de arquivos e criará pastas.

Use o ícone New Folder e crie uma pasta chamada images. Abra esta pasta.

Selecione o ícone Upload para carregar os arquivos de imagem necessários para este laboratório.

Selecione todos os arquivos de imagem para este laboratório da pasta Images dos arquivos que você extraiu. Depois de selecionar todos eles, escolha Open para carregar os arquivos.

As imagens foram carregadas para a pasta correspondente.

Retorne à pasta raiz.

Selecione o ícone Upload para carregar os arquivos do notebook necessários para este laboratório.

Selecione todos os arquivos de notebook para este laboratório a partir dos arquivos que você extraiu. Depois de selecionar todos eles, escolha Open para carregar os arquivos.

Você deverá ver todos os arquivos de tarefas enviados para seu ambiente de laboratório.

Para finalizar a configuração da Tarefa, você precisará executar a Tarefa-0. Abra o arquivo de notebook da Tarefa-0.

Execute a célula de código usando o botão Run. Isso instalará e atualizará tudo o que você precisa para executar esses laboratórios. Após executar a célula de código, certifique-se de reiniciar o kernel antes de continuar.

Parabéns! Seu ambiente está configurado e você está pronto para continuar.

Você fez o seguinte com sucesso:

- Solicitou acesso aos modelos do Amazon Bedrock.
- Criei e sincronizei sua base de conhecimento.
- Iniciou e configurou o Amazon SageMaker.
- Arquivos de configuração enviados para os laboratórios.
- Atualizou o kernel e os arquivos no ambiente.

Crie e avalie aplicativos RAG usando demonstrações de bases de conhecimento do Amazon Bedrock

Neste laboratório, você cria um aplicativo de perguntas e respostas usando a base de conhecimento da AnyCompany e a API RetrieveAndGenerate da Amazon Bedrock. Você aproveita a base de conhecimento existente, que contém informações abrangentes sobre os produtos, serviços e detalhes corporativos da AnyCompany, como histórico, liderança, desempenho financeiro, esforços de sustentabilidade e muito mais. Você percorre vários notebooks que podem responder efetivamente a perguntas relacionadas aos produtos, serviços e informações corporativas da AnyCompany.

Ao final deste laboratório, você será capaz de fazer o seguinte:

- Aproveite um aplicativo RAG totalmente gerenciado com a API Amazon Bedrock RetrieveAndGenerate.
- Crie um aplicativo de perguntas e respostas usando o Amazon Bedrock Knowledge Bases com a API Retrieve.
- Teste o processo de reformulação de consultas com suporte das bases de conhecimento do Amazon Bedrock.
- Crie e avalie um aplicativo de perguntas e respostas usando as bases de conhecimento do Amazon Bedrock usando a estrutura RAG Assessment (ou RAGAS).
- Teste a funcionalidade do guardrail na Base de Conhecimento do Amazon Bedrock usando a API RetrieveAndGenerate.

Nesta primeira tarefa, você aproveita um aplicativo Retrieval Augmented Generation (ou RAG) totalmente gerenciado com a API RetrieveAndGenerate da Amazon Bedrock. Para esta tarefa, você executa o arquivo de notebook Task-1.ipynb.

Revise as informações iniciais fornecidas para a Tarefa 1, que estão separadas em subtarefas.

Examine a arquitetura final do laboratório.

Na Tarefa 1.1, execute a primeira célula de código para inicializar o cliente boto3 e configurar seu ambiente.

Execute a próxima célula de código para verificar o ID da Base de Conhecimento existente no Amazon Bedrock.

A saída da célula é exibida no notebook. O ID da base de conhecimento é fornecido para verificação.

Na Tarefa 1.2, você testa a Base de Conhecimento usando a função recuperar e gerar.

Você testa inicialmente a base de conhecimento usando a API *RetrieveAndGenerate* . Com essa API, o Amazon Bedrock cuida da recuperação das referências necessárias da base de conhecimento e gera a resposta final usando um modelo de base (ou FM) do Amazon Bedrock.

Execute as duas células de código a seguir para iniciar uma consulta para testar a base de conhecimento. A primeira célula cria a consulta.

A próxima célula usa a consulta para testar a base de conhecimento.

O resultado aqui é um resumo das demonstrações consolidadas de fluxos de caixa da AnyCompany Financial para o ano fiscal encerrado em 31 de dezembro de 2019.

Na Tarefa 1.3, você entenderá a API RetrieveAndGenerate em detalhes.

Execute as duas células de código a seguir para declarar o prompt padrão para a base de conhecimento. A primeira célula declara o prompt padrão da base de conhecimento.

A segunda célula define os parâmetros de recuperação e geração.

Na Tarefa 1.4, você aproveita o recurso de número máximo de resultados. Em alguns casos de uso, as respostas do FM podem não ter contexto suficiente para fornecer respostas relevantes ou depender de que ele não conseguiu encontrar as informações solicitadas. Isso pode ser corrigido modificando o número máximo de resultados recuperados.

Nesta tarefa, você executará a seguinte consulta usando um resultado:

Forneça uma lista de riscos financeiros da AnyCompany em tópicos.

Execute as duas células de código a seguir. A primeira célula define os resultados da geração de impressão.

Em seguida, execute a segunda célula, que contém a consulta e exibirá os resultados.

É gerada uma lista dos principais riscos para a AnyCompany Financial mencionados nos resultados da pesquisa.

Na próxima caixa de código, o número máximo de resultados foi alterado de um para três.

Execute a célula de código com o valor modificado para o número de resultados recuperados.

A resposta gerada é uma lista com o dobro do tamanho da anterior, com mais riscos identificados.

Como você pode ver na saída, ao modificar o número de resultados recuperados para três, você obtém mais resultados, o que leva a uma resposta muito mais abrangente.

Na Tarefa 1.5, você personaliza o prompt padrão com seu próprio prompt com base no caso de uso. Esse recurso ajuda a adicionar mais contexto ao FM, o que requer formato de saída específico, idiomas e outros contextos.

Você usa a classe *PromptTemplate* do Langchain para parametrizar o template *de prompt personalizado* . Você usa a variável *output format* para modificar a saída em tempo de execução.

Execute esta célula de código para criar uma função auxiliar que suporte a personalização do modelo.

Para a Tarefa 1.5.1, use o mesmo exemplo de consulta e defina o FM como padrão para gerar a saída em um idioma diferente.

Execute as três células de código a seguir para testar um exemplo que fornece uma saída em francês. A primeira célula fornece o prompt.

A segunda célula solicita que a resposta seja fornecida em francês.

Aqui você pode ver a saída combinada das duas primeiras células.

A terceira célula fornecerá os resultados da pesquisa no idioma especificado, que é o francês.

Uma lista dos principais riscos para a AnyCompany Financial é gerada, em francês, conforme solicitado.

Por fim, na Tarefa 1.5.2, você usa o mesmo exemplo de consulta e define o FM para gerar a saída no formato JSON.

Execute a célula de código para testar um exemplo que fornece uma saída no formato JSON.

A resposta é uma lista gerada no formato JSON.

Você concluiu este notebook. Feche este arquivo de notebook e continue com a Tarefa 2.

Na Tarefa 2, você cria um aplicativo de perguntas e respostas usando Amazon Bedrock Knowledge Bases com Retrieve API. Aqui, você consulta a base de conhecimento para obter o número desejado de pedaços de documentos com base na pesquisa de similaridade. Em seguida, você aumenta o prompt com documentos relevantes e executa uma consulta que atua como entrada para Anthropic Claude para gerar resposta.

Revise as informações iniciais fornecidas para a tarefa, incluindo o cenário.

Revise as informações iniciais fornecidas para a tarefa. Na Tarefa 2.1, você inicia o cliente Amazon Bedrock e executa as seguintes operações:

- Verifique o ID da Base de Conhecimento.
- Importe as bibliotecas necessárias e configure os clientes necessários

Na Tarefa 2.1.1, você verifica o ID da Base de Conhecimento.

Para executar este notebook, você precisa verificar e atribuir o ID da Base de Conhecimento à variável *kb ID* e instalar os pacotes necessários.

Execute a célula de código para verificar o ID da Base de Conhecimento existente no Amazon Bedrock.

O ID da base de conhecimento é fornecido e pode ser verificado.

Na Tarefa 2.1.2, você inicia o cliente Amazon Bedrock.

Execute a célula de código para importar as bibliotecas necessárias para configurar seu ambiente.

A região é fornecida para verificação para mostrar que o ambiente está configurado corretamente.

Na Tarefa 2.2, você define uma função de recuperação que chama a API *de recuperação* fornecida pela Base de conhecimento para Amazon Bedrock, que converte consultas de usuários em incorporações, pesquisa na base de conhecimento e retorna os resultados relevantes, dando a você mais controle para criar fluxos de trabalho personalizados com base nos resultados da pesquisa semântica.

Revise as informações iniciais fornecidas para a tarefa.

Execute a célula de código para definir uma função *de recuperação* que chama a API *Retrieve* .

Na Tarefa 2.2.1, você inicializa seu Knowledge Base ID antes de consultar respostas do LLM inicializado. Você chama a API *Retrieve* e passa o *Knowledge Base ID* , *o número de resultados* e *a consulta* como parâmetros.

Execute a célula de código a seguir para chamar a API *Retrieve* passando o *ID da Base de Conhecimento* , *o número de resultados* e *a consulta* como parâmetros.

Você pode visualizar a pontuação associada de cada pedaço de texto retornado, o que descreve sua correlação com a consulta em termos de quão próximo ela corresponde a ela.

Na Tarefa 2.2.2, você extrai os blocos de texto da resposta da API *Retrieve* .

Execute as duas células de código a seguir para buscar o contexto dos resultados da recuperação e imprimi-los.

A primeira caixa de código busca o contexto da resposta.

Esta segunda célula imprime os resultados do contexto.

Aqui estão os resultados.

Todos os pedaços foram extraídos do texto.

Na Tarefa 2.2.3, você usa um prompt específico para o modelo atuar como um sistema de IA de consultor financeiro que fornece respostas a perguntas usando informações estatísticas e baseadas em fatos quando possível. Você fornece as respostas da API *Retrieve* da tarefa anterior como parte dos *{contextos}* no prompt para o modelo consultar, junto com a *consulta* do usuário .

Execute a célula de código para usar um prompt específico para o modelo atuar como um sistema de IA de consultor financeiro.

Na Tarefa 2.2.4, você invoca um modelo de base do Amazon Bedrock.

Execute as duas células de código a seguir para invocar o modelo de base anthropic.claude-3-sonnet-20240229-v1:0 do Amazon Bedrock.

Você está passando tanto o contexto quanto a consulta para o modelo. Execute a segunda célula.

O texto de resposta é fornecido como saída.

A tarefa 2.3 envolve a integração do LangChain.

Nesta tarefa, você cria um aplicativo de perguntas e respostas usando a classe *AmazonKnowledgeBasesRetriever* do LangChain. Você consulta a base de conhecimento para obter o número desejado de pedaços de documentos com base na pesquisa de similaridade. Depois disso, você a integra com a cadeia do LangChain para passar os pedaços de documentos e a consulta para o LLM (que é o Anthropic Claude 3 Sonnet) para responder às perguntas.

Na Tarefa 2.3.1, você configura seu ambiente. Execute a célula de código para importar os pacotes necessários para configurar seu ambiente.

Na Tarefa 2.3.2, você cria um objeto *AmazonKnowledgeBasesRetriever* do LangChain que chama a API *Retrieve* fornecida pelo Amazon Bedrock Knowledge Bases. Isso converte consultas de usuários em embeddings, pesquisa a base de conhecimento e retorna os resultados relevantes, dando a você mais controle para criar fluxos de trabalho personalizados sobre os resultados da pesquisa semântica.

Execute a célula de código para criar o objeto *AmazonKnowledgeBasesRetriever .*

Os resultados são apresentados para você.

Na Tarefa 2.3.3, você usa um prompt específico para o modelo atuar como um sistema de IA de consultor financeiro que fornece respostas a perguntas usando informações estatísticas e baseadas em fatos quando possível. Você fornece as respostas da API *Retrieve* acima como parte do *{context}* no prompt para o modelo consultar, junto com a *consulta* do usuário .

Execute a célula de código para usar um prompt específico para o modelo atuar como um sistema de IA de consultor financeiro.

Por fim, na Tarefa 2.3.4, você integra o recuperador e o LLM usando a Linguagem de Expressão Langchain (ou LCEL) para criar o aplicativo de perguntas e respostas.

A resposta é fornecida como uma saída.

Você concluiu este notebook. Feche este arquivo de notebook e continue com a Tarefa 3.

Na Tarefa 3, você entende e testa o processo de Reformulação de Consulta suportado pelas Bases de Conhecimento Amazon Bedrock. Com a reformulação de consulta, você pega um prompt de entrada complexo e o divide em várias subconsultas. Essas subconsultas passam separadamente por suas próprias etapas de recuperação para blocos relevantes. Os blocos resultantes são então agrupados e classificados juntos antes de passá-los para o Modelo Fundamental para gerar uma resposta. A reformulação de consulta é outra ferramenta que ajuda a aumentar a precisão para consultas complexas que seu aplicativo pode enfrentar na produção.

Nesta primeira tarefa, Tarefa 3.1, você configura o ambiente do notebook importando os pacotes necessários.

Execute as duas células de código a seguir para importar os pacotes necessários e configurar seu ambiente.

Após executar a primeira célula, a versão boto3 é fornecida como saída.

Execute a segunda célula.

A região e o ID da conta são fornecidos como saída.

Execute a célula de código para verificar o ID da Base de Conhecimento existente no Amazon Bedrock.

O ID da base de conhecimento foi fornecido para verificação.

Execute a célula de código para definir o FM a ser usado neste notebook.

Na Tarefa 3.2, você demonstra a reformulação de consulta. Você investiga uma consulta simples e uma mais complexa que poderiam se beneficiar da reformulação de consulta e vê como isso afeta as respostas geradas.

Na Tarefa 3.2.1, você vê como o resultado gerado se parece para a seguinte consulta sem usar reformulação de consulta. Você usa a seguinte consulta: *Onde fica o edifício da orla da empresa AnyCompany e como o escândalo do denunciante prejudica a empresa e sua imagem?*

Execute as três células de código a seguir para gerar resultados sem reformulação de consulta. A primeira cria a consulta.

A segunda caixa de código imprimirá a resposta quando executada.

A saída é fornecida para você.

Execute a terceira célula para examinar os blocos que geram a resposta.

O número de citações ou blocos usados para gerar a resposta é fornecido, juntamente com o bloco, a localização e os metadados.

Como visto nas citações acima, sua recuperação com a consulta complexa não retornou nenhum bloco relevante para o edifício, concentrando-se em vez disso nas incorporações mais semelhantes ao incidente do denunciante.

Isso pode indicar que a incorporação da consulta resultou em alguma diluição da semântica daquela parte da consulta.

Na Tarefa 3.2.2, você vê como a reformulação da consulta pode beneficiar a recuperação de contexto mais alinhada, o que, por sua vez, aumentará a precisão da geração de respostas.

Execute as duas células de código para gerar os resultados com reformulação da consulta.

A primeira célula fornece a resposta como uma saída de texto gerada.

Execute a próxima célula para dar uma olhada nos blocos recuperados com reformulação da consulta.

Como você pode ver, com a reformulação da consulta ativada, os blocos recuperados agora fornecem contexto para o escândalo do denunciante e a localização dos componentes da propriedade à beira-mar.

Você concluiu este notebook. Feche este arquivo de notebook e continue com a Tarefa 4.

Nesta próxima tarefa, você cria e avalia um aplicativo de perguntas e respostas usando a API Retrieve fornecida pela Amazon Bedrock Knowledge Bases, juntamente com LangChain e RAGAS para avaliar as respostas. Aqui, você consulta a base de conhecimento para obter o número desejado de pedaços de documentos com base na pesquisa de similaridade, solicita a consulta usando Anthropic Claude e, em seguida, avalia as respostas efetivamente usando métricas de avaliação, como fidelidade, relevância da resposta, recall de contexto, precisão do contexto, recall de entidade de contexto, similaridade da resposta, correção da resposta, nocividade, malícia, coerência, correção e concisão.

Na Tarefa 4, você usa os relatórios financeiros 10k da AnyCompany (um conjunto de dados gerado sinteticamente) como um corpus de texto para executar perguntas e respostas. Esses dados já estão ingeridos na Base de Conhecimento no Amazon Bedrock.

Revise as informações iniciais fornecidas para a tarefa.

Na Tarefa 4.1, você configura o ambiente

Para executar este notebook, você precisa instalar as dependências LangChain e RAGAS e os pacotes boto3 e botocore atualizados.

Execute a primeira célula de código para verificar o ID da Base de Conhecimento existente no Amazon Bedrock.

O ID da base de conhecimento é fornecido como uma saída para verificação.

Execute a próxima célula de código para instalar dependências.

Na Tarefa 4.2, você cria um objeto *AmazonKnowledgeBasesRetriever* do LangChain para pesquisar na base de conhecimento e retornar os resultados relevantes, dando a você mais controle para criar fluxos de trabalho personalizados sobre os resultados da pesquisa semântica.

Execute a célula de código para criar um objeto *AmazonKnowledgeBasesRetriever .*

Na Tarefa 4.3, você invoca o modelo e visualiza a resposta usando as seguintes informações:

Pergunta: Forneça uma lista de alguns riscos financeiros para a AnyCompany em uma lista numerada e sem descrição."

Resposta da verdade fundamental

1. Preços das commodities
2. Taxas de câmbio estrangeiras
3. Preços de ações
4. Risco de crédito
5. Risco de liquidez

Execute a célula de código para criar um prompt com contexto e pergunta como variáveis.

Em seguida, execute a seguinte célula de código para invocar o modelo usando uma consulta predefinida e imprimir o resultado.

Os resultados da consulta são fornecidos como uma saída.

Na Tarefa 4.4, você prepara os dados de avaliação.

Como o RAGAS pretende ser uma estrutura de avaliação sem referência, as preparações necessárias do conjunto de dados de avaliação são mínimas. Nesta tarefa, você prepara os pares *de perguntas* e *verdades básicas* a partir dos quais você pode preparar as informações restantes por meio de inferência, conforme mostrado abaixo.

Execute a seguinte célula de código para preparar os pares *de perguntas* e *verdades básicas* para avaliação.

Execute a próxima célula de código para ver as respostas do LLM e as verdades básicas para o conjunto de perguntas de avaliação.

As respostas são fornecidas como saída.

A verdade básica também é fornecida após cada resposta.

Na Tarefa 4.5, você avalia o aplicativo RAG.

Nesta tarefa, você importa todas as métricas que deseja usar de *ragas.metrics* . Então, você usa a função *assess()* e simplesmente passa as métricas relevantes e o conjunto de dados preparado.

Execute a célula de código para importar todas as métricas de *ragas.metrics* e use a função *assess()* .

O processo de avaliação começa. A barra de status ilustra o processo.

Você pode ignorar com segurança quaisquer avisos na saída acima. Certifique-se de que a avaliação esteja 100 por cento concluída antes de prosseguir.

Execute a célula de código para ver as pontuações RAGAS resultantes.

As pontuações RAGAS são apresentadas em formato de tabela. Para melhor legibilidade, é possível exportar a tabela para uma planilha.

Execute a célula de código para exportar as pontuações RAGAS resultantes como um arquivo do Microsoft Excel.

Após executar com sucesso a célula de código acima, um arquivo chamado styled.xlsx deverá estar disponível para revisão no painel de navegação esquerdo.

Observe que as pontuações acima dão uma ideia relativa sobre o desempenho do seu aplicativo RAG e devem ser usadas com cautela e não como pontuações independentes. Observe também que você usou apenas 5 pares de perguntas/respostas para avaliação. Como prática recomendada, você deve usar dados suficientes para cobrir diferentes aspectos do seu documento para avaliar o modelo. Com base nas pontuações, você pode revisar outros componentes do seu fluxo de trabalho RAG para otimizar ainda mais as pontuações.

Você concluiu este notebook. Feche este arquivo de notebook e continue com a Tarefa 5.

Nesta tarefa final, você testa a funcionalidade do guardrail na Amazon Bedrock Knowledge Base usando a API RetrieveAndGenerate. Você utiliza uma base de conhecimento existente usada com os notebooks anteriores, que foi baseada

no documento AnyCompany's Financial 10K.

Na primeira tarefa, Tarefa 5.1, você inicializa o cliente boto3 para configurar o ambiente para este notebook. Em todo o notebook, você utiliza a API RetrieveAndGenerate para testar os recursos da base de conhecimento.

Execute a seguinte célula de código para inicializar o cliente boto3 para configurar seu ambiente.

Para executar este notebook, você precisa verificar e atribuir o ID da Base de Conhecimento à variável *KB_ID* .

Execute a seguinte célula de código para verificar o ID da Base de Conhecimento existente no Amazon Bedrock.

O ID da base de conhecimento é fornecido para verificação.

Na Tarefa 5.2, você testa se um guardrail com guardrailName já existe. Se não existir, você cria um guardrail com esse nome. Caso contrário, você captura o ID do guardrail e a versão do guardrail para um guardrail existente.

Execute as duas células de código a seguir para testar um guardrail existente e criar um novo se não houver um. A primeira célula testa um guardrail existente.

A segunda célula cria uma proteção que impede o modelo de fornecer aconselhamento fiduciário.

A saída mostra que o guardrail foi criado.

Na Tarefa 5.3, você testa a base de conhecimento usando a API RetrieveAndGenerate.

Com esta API, a Bedrock cuida de recuperar as referências necessárias da base de conhecimento e gerar a resposta final usando um modelo de fundação da Bedrock. Sem o guardrail, o modelo de fundação fornece uma resposta. Quando você inclui o guardrail, o modelo se abstém de fornecer consultoria financeira.

Execute a célula de código para iniciar uma consulta.

Execute a próxima célula de código para criar duas configurações separadas: uma sem o guardrail e outra com o guardrail.

Execute esta célula de código para definir a função de obtenção, recuperação e geração de resposta.

Execute a célula de código a seguir e peça ao modelo para responder sem o guardrail e você poderá ver se o modelo fornece consultoria de investimento.

O resultado é uma resposta que fornece consultoria de investimento.

Execute a seguinte célula de código e peça ao modelo para responder com o guardrail.

Após invocar o guardrail, o modelo se recusa a fornecer qualquer aconselhamento financeiro.

Execute as duas células de código a seguir que permitem que o trace veja o guardrail em ação para uma consulta que pergunta: *Devo investir na AnyCompany Financial?*

A primeira célula importa o cliente em tempo de execução.

A segunda célula envia a solicitação para Bedrock usando o guardrail existente.

A saída do trace mostra como o modelo respondeu, mas essa resposta não chegou até o usuário. O Guardrail interveio e enviou a mensagem de saída enlatada. Você também pode ver a política que foi violada.

Você concluiu este notebook e este laboratório. Feche este arquivo do notebook e continue.

Após concluir essas tarefas, você terá feito o seguinte com sucesso:

- Aproveitou um aplicativo RAG totalmente gerenciado com a API Amazon Bedrock RetrieveAndGenerate.
- Criei um aplicativo de perguntas e respostas usando o Amazon Bedrock Knowledge Bases com a API Retrieve.
- Testei o processo de reformulação de consultas com suporte das bases de conhecimento do Amazon Bedrock.
- Criei e avaliei um aplicativo de perguntas e respostas usando as bases de conhecimento do Amazon Bedrock e a estrutura RAG Assessment (ou RAGAS).
- Testou a funcionalidade de guardrail na Base de Conhecimento Amazon Bedrock usando a API RetrieveAndGenerate.

Módulo 6: Usando agentes com LLMs no Amazon Bedrock

Agentes são componentes fundamentais para potencializar o uso de **grandes modelos de linguagem (LLMs)**, permitindo que eles executem tarefas complexas de forma autônoma e segura. Abaixo estão os principais conceitos abordados:

O que são agentes

- **Agentes** são aplicativos que utilizam LLMs para resolver tarefas, dividindo problemas em etapas lógicas com técnicas como *Chain of Thought (CoT)* e *ReAct* (Raciocínio + Ação).
- Permitem executar ações como chamadas de API, consultas a bases de conhecimento (via Amazon Bedrock Knowledge Bases) e execução de código.
- Os agentes seguem instruções definidas (o que devem/não devem fazer) e são protegidos por **guardrails**, que refletem as políticas de IA responsável da empresa.

Relação entre LLMs e agentes

- LLMs entendem linguagem natural e geram texto com qualidade humana, mas não executam tarefas ou controlam sistemas externos por conta própria.
- Agentes preenchem essa lacuna ao orquestrar ações com base em saídas intermediárias do LLM, como:
  - Executar código
  - Fazer chamadas a APIs

    - Recuperar dados de bases de conhecimento
- Essa abordagem permite construir **fluxos de trabalho complexos, adaptáveis e eficientes**, com menos codificação manual.

Funcionamento de um agente no Amazon Bedrock

- **Entrada**: O agente recebe uma instrução em linguagem natural.
- **Interpretação**: Ele entende a intenção, lida com ambiguidade e contexto.
- **Execução**: Divide a tarefa em etapas e utiliza recursos como LLMs, chamadas de função ou integrações com sistemas externos.
- **Resposta**: Retorna uma solução ou ação coerente ao usuário.

Recursos avançados dos agentes

- **Decomposição e delegação**: Capacidade de dividir tarefas complexas em subtarefas e delegar para funções ou sistemas adequados.
- **Interação com sistemas externos**: Agentes podem realizar ações como:
    - Atualizar bancos de dados
    - Controlar dispositivos
    - Iniciar processos
- **Integração com bases de conhecimento**: Acesso a dados atualizados e específicos do domínio, aumentando a relevância e a confiança da resposta.
- **Autocorreção e verificação de erros**: O agente usa o raciocínio do LLM para revisar e melhorar respostas automaticamente.
- **Transparência e interpretabilidade**: Explica decisões tomadas e etapas executadas, essencial em setores como saúde, finanças e direito.

Técnicas utilizadas

- **CoT (Chain of Thought)**: Técnica que orienta o LLM a resolver problemas complexos por meio de raciocínio passo a passo, facilitando a tomada de decisão e a interpretação lógica.
- **Guardrails no Amazon Bedrock**: Conjunto de proteções que evita respostas inadequadas, filtra conteúdos sensíveis e assegura conformidade com diretrizes da empresa.

Estrutura ReAct e Funcionamento dos Agentes no Amazon Bedrock

ReAct (Raciocínio e Ação)

- ReAct é uma estrutura de raciocínio que permite aos agentes intercalarem **pensamento lógico** com **execução de ações específicas**.
- Isso possibilita que os agentes:
    - Interajam com LLMs (modelos de linguagem);
    - Forneçam **contexto adicional** ou **feedback** ao modelo;
    - Atualizem as respostas com base em novas informações ou saídas de etapas anteriores.
- Essa abordagem **iterativa** é especialmente útil em cenários **ambíguos ou complexos**.

Componentes Essenciais de um Agente

- **Instruções do agente**: Definem o que o agente pode e não pode fazer.

- **Modelo de base (FM)**: Responsável por interpretar entradas e gerar respostas.
- **Memória do agente**: Permite persistência da conversa e uso de contextos anteriores.
- **Grupo de ações**: Define o que o agente pode executar, podendo incluir:
  - Funções customizadas;
  - APIs definidas via JSON ou OpenAPI;
  - Funções AWS Lambda;
  - Execução de código analítico gerado por LLMs.

Integrações Possíveis

- **Base de conhecimento**: Permite que o agente acesse informações confiáveis e atualizadas.
- **Guardrails**: Aplicam políticas de uso responsável da IA com base em diretrizes da empresa.
- **Serviços AWS**: Os agentes podem ser implantados e gerenciados usando ferramentas como AWS CDK, CloudFormation, ou Terraform.
- **Idiomas compatíveis**: C++, Go, Java, JavaScript, Kotlin, .NET, PHP, Python e Ruby.

Monitoramento e Segurança

- Agentes podem ser monitorados via **Amazon CloudWatch Logs**.
- Suporte a sessões com **atributos personalizados**, **controle de retorno**, e **parâmetros dinâmicos**.
- Suporte ao encerramento de sessões e uso de identificadores de memória.

Funcionamento do Agente: Dois Momentos-Chave

1. Tempo de Construção

- Fase de **definição e configuração do agente**:
  - Criação de instruções, grupos de ação, bases de conhecimento e guardrails.
  - Configuração do prompt base usado no tempo de execução.
  - Esses elementos definem como o agente interpreta entradas e executa tarefas.

> *Exemplo visual omitido: Ilustração dos componentes envolvidos na configuração do agente.*

---

2. Tempo de Execução

- Fase onde o agente interage com o usuário e executa as tarefas.
- Principais etapas:
  - **Pré-processamento**: Valida e interpreta a entrada do usuário.
  - **Orquestração**: Executa ações ou consulta bases de conhecimento.
  - **Pós-processamento**: Formata a resposta final.
- **Fluxo iterativo**:
  - O agente pode retornar controle ao desenvolvedor;
  - O desenvolvedor executa ações externas e envia resultados de volta ao agente;
  - O processo continua até que o agente gere a resposta final ou solicite mais dados do usuário.

> *Exemplo visual omitido: Demonstra o ciclo de orquestração de tarefas pelo agente.*

Prompts Avançados no Amazon Bedrock

- Permitem **personalização profunda**:
  - Habilitar/desabilitar etapas (pré/pós-processamento, orquestração, etc.);
  - Substituir prompts padrão;
  - Ajustar parâmetros de inferência;
  - Inserir exemplos personalizados (poucos disparos) para refinar o comportamento do agente.
- Os **prompts orientam** como o agente:
  - Interpreta entradas;
  - Coordena ações e consultas;
  - Gera saídas coerentes e contextuais.

> *Exemplo visual omitido: Demonstra o uso de placeholders e eventos do Lambda nos prompts.*

Personalização de Funções Lambda

- Funções Lambda podem ser substituídas por implementações próprias.
- Importante: As respostas dessas funções devem seguir o **formato esperado** pelo agente para garantir a continuidade da orquestração.

Configuração de Prompts e Criação de Agentes no Amazon Bedrock

1. Modelos de Prompt Avançado

No Amazon Bedrock, os agentes são configurados com quatro tipos principais de prompts, cada um com uma função específica no fluxo de execução:

1.1 Prompt de Pré-processamento

- **Objetivo**: Modificar e filtrar a entrada do usuário antes de seguir para a orquestração.
- **Casos de uso**:
  - Filtrar entradas maliciosas.
  - Detectar múltiplas solicitações numa mesma mensagem.
  - Identificar entradas que o agente não pode lidar.
- **Personalização**: Permite definir regras de categorização e validação específicas.

1.2 Prompt de Orquestração

- **Objetivo**: Controlar como o agente decide entre usar o modelo de linguagem, bases de conhecimento, grupos de ação ou interpretação de código.
- **Casos de uso**:
  - Ajustar lógica de orquestração.
  - Incluir exemplos para lidar com situações específicas.
  - Substituir diretrizes de comportamento do agente.

1.3 Prompt de Geração de Resposta da Base de Conhecimento

- **Objetivo**: Alterar como o agente consulta e usa informações das bases de conhecimento.
- **Casos de uso**:
  - Ajustar o método de recuperação de informação.
  - Forçar citações diretas de fontes confiáveis.

1.4 Prompt de Pós-processamento

- **Objetivo**: Modificar a resposta final antes de ser apresentada ao usuário.
- **Casos de uso**:
  - Aplicar formatações específicas.
  - Adicionar contexto final.
  - Realizar ajustes finos na linguagem da resposta.

---

2. Criação e Implantação de Agentes

2.1 Formas de Criação

- **Ferramentas**: AWS Console, Amazon Bedrock API, AWS CDK, CLI, CloudFormation, Terraform.
- **Elementos obrigatórios**:
  - Função IAM com permissões.
  - Seleção de Foundation Model (FM).
  - Instruções em linguagem natural para definir comportamento do agente.

2.2 Instruções do Agente

- **Inserção**: No prompt de orquestração.
- **Importância**:
  - Definem o nome, modo e escopo do agente.
  - Descrevem como ele deve interagir com o usuário e validar dados antes de agir.

---

3. Configurações Adicionais do Agente

- **Code Interpreter**: Permite que o agente execute código.
- **User Input**: Permite que o agente peça informações adicionais ao usuário quando necessário.
- **Chaves KMS**: Permite criptografia com chaves personalizadas.
- **Tempo limite da sessão**: Por padrão, 10 minutos de inatividade encerram a sessão do usuário.

---

4. Criação de Agente com API do Bedrock

- **SDKs disponíveis**: Suporte para várias linguagens como Python, JavaScript, Go, Java, Swift, .NET, PHP, entre outras.
- **Requisito prévio**: Criar política e função do IAM com permissões mínimas para invocar o modelo.

---

5. Grupos de Ação do Agente

5.1 Definição

- Componentes que contêm ações específicas que o agente pode executar.
- Tipos:
  - **Função Lambda**
  - **Retorno de controle ao aplicativo chamador**

5.2 Métodos de Criação

- **Com Esquema OpenAPI**:
  - Define as APIs disponíveis.

- Pode ser escrito em JSON ou YAML.
- Inserido diretamente no console ou carregado via Amazon S3.

- **Com Detalhes da Função**:
  - Descreve diretamente os parâmetros e lógica da função a ser executada.

---

6. Componentes do Esquema OpenAPI

Cada esquema OpenAPI deve seguir os seguintes elementos para funcionar com grupos de ação:

- **Versão da especificação**: Deve ser 3.0.0 ou superior.
- **Paths (Caminhos)**: Representam rotas de API (ex: `/sendEmail`).
- **Métodos HTTP**: Como POST, GET, PUT etc., indicam o tipo de operação.
- **Descrição**: Explica o propósito da rota e como usá-la.
- **Parâmetros**: Definem as entradas aceitas pela rota, incluindo se são obrigatórios.
- **Request Body**: Estrutura e formato do corpo da solicitação.
- **Responses**: Códigos de resposta esperados, descrições e formatos de retorno.

---

7. Invocação de Grupos de Ação

7.1 Tipos de Invocação

- **Função Lambda**: Executa a lógica de negócios e retorna o resultado ao agente.
- **Return Control**: O aplicativo cliente gerencia a execução e resposta da ação.

7.2 Implementação da Função Lambda

- Recebe um evento JSON com detalhes da solicitação.
- Deve usar lógica condicional para determinar qual ação executar.
- A resposta deve ser estruturada como um objeto JSON com chave `TEXT`.

7.3 Criação Rápida (Quick Create)

- Gera automaticamente um esqueleto de função Lambda com permissões e estrutura de resposta básica.
- **Observação**: O desenvolvedor ainda precisa implementar a lógica de negócio.

Considerações Finais

- Os **prompts avançados** permitem ajustes finos em todas as fases do processamento do agente.
- É possível controlar completamente como o agente processa, decide e responde às entradas.
- A configuração de **grupos de ação e funções Lambda** é essencial para tarefas dinâmicas e interativas.
- O uso do **OpenAPI** como descrição das ferramentas de ação permite um alto grau de integração e automação.

Fluxo de ação de grupo com função Lambda

1. Definição da função Lambda no grupo de ações

- O agente é criado com um **grupo de ação que utiliza uma função Lambda** como executor.
- Essa configuração é feita através do parâmetro `actionGroupExecutor`, que associa o grupo à função Lambda responsável por executar a lógica de negócios.

2. Invocação do agente pelo aplicativo

- O **aplicativo cliente** envia uma solicitação para o agente com um texto de entrada fornecido pelo usuário.

- Esse texto contém os **dados necessários para realizar a ação**, como identificadores e datas no caso de uma solicitação de férias, por exemplo.

3. Pré-processamento da entrada

- O agente pode **pré-processar o texto de entrada** para classificá-lo, validar informações ou identificar múltiplas intenções.
- Esse passo é opcional e pode ser ativado para **adicionar regras de controle e segurança**, embora possa aumentar a latência da resposta.
- Pode envolver uma chamada a um **modelo de fundação (FM)** para apoio na análise.

4. Geração de justificativa

- Durante a fase de orquestração, o agente interpreta a entrada com um FM e gera uma **justificativa lógica**.
- Essa justificativa define o **próximo passo na tomada de decisão**, com base no conteúdo e na intenção detectada.

5. Escolha da ação

- O agente decide que deve **executar uma ação específica** por meio da função Lambda.
- Ele identifica o **nome da função Lambda** apropriada e determina os **parâmetros necessários** com base na entrada do usuário.

6. Invocação da função Lambda

- O agente invoca a função Lambda com os parâmetros previamente definidos.
- A função Lambda **executa a lógica de negócios**, como verificar disponibilidade e registrar uma ação.
- Retorna ao agente uma **resposta com sucesso ou erro**, conforme o resultado do processamento.

7. Finalização e resposta ao usuário

- O agente **recebe a resposta da Lambda**, finaliza o processo e entrega o resultado final ao aplicativo cliente.
- O **aplicativo então exibe essa resposta ao usuário**, encerrando o fluxo.

Implementando o Controle de Retorno (Return Control)

O que é o Controle de Retorno

- Um tipo de invocação para grupos de ação que permite **executar ações de forma assíncrona**.
- Útil tanto para testes rápidos no console quanto em **cargas de produção** quando **não há necessidade de uma função Lambda**.
- O agente **retorna o controle para o aplicativo**, que decide como processar e responder ao usuário.

Etapas do Fluxo com Controle de Retorno

1. Definição com `RETURN_CONTROL`

- O agente é criado com um grupo de ação cujo executor é configurado com o tipo `RETURN_CONTROL`.
- Isso define que o controle será retornado ao aplicativo em vez de invocar uma função diretamente.

2. Invocação do agente

- O aplicativo envia ao agente uma solicitação com o **texto de entrada do usuário**.
- A entrada inclui os **dados relevantes para a ação solicitada**, como identificadores e datas.

3. Pré-processamento opcional

- O agente pode validar e categorizar a entrada antes de prosseguir.

- Pode envolver chamadas a modelos de fundação (FM), embora **isso aumente a latência**.

4. Geração de justificativa

- O agente utiliza um FM para interpretar a solicitação e **produz uma justificativa lógica** para decidir o próximo passo.
- Essa lógica orienta qual ação precisa ser executada.

5. Escolha da ação com retorno de controle

- O agente decide que a ação será tratada via controle de retorno.
- Ele responde ao aplicativo com uma estrutura que inclui a **ação a ser tomada e seus parâmetros**.
- Essa estrutura indica, por exemplo, qual função local o aplicativo deve executar.

6. Invocação da função pelo aplicativo

- O aplicativo executa a **função local apropriada** com os parâmetros informados.
- A função realiza a **lógica de negócios**, como verificar disponibilidade, criar registros ou validar regras.
- Retorna um resultado com sucesso ou erro.

7. Retorno dos resultados ao agente

- O aplicativo envia de volta ao agente os **resultados da função local**, utilizando um campo apropriado no estado de sessão.
- O agente então conclui o processamento com base na resposta recebida.

8. Resposta ao usuário

- O agente finaliza a ação e o aplicativo responde ao usuário com a **mensagem final resultante**.

Resumo Geral dos Grupos de Ação no Bedrock Agents

- Grupos de ação definem o **que um agente pode fazer** para auxiliar os usuários.
- São construídos por meio de:
  - **Definições de função** (objetos JSON que definem funções e parâmetros).
  - **Esquemas OpenAPI** (definições agnósticas de linguagem para APIs/funções).

Dois métodos de invocação disponíveis:

1. **Funções Lambda**
   - Executa ações definidas com lógica de negócios diretamente via Lambda.
   - O agente cuida de toda a orquestração, incluindo chamadas e tratamento de erros.
2. **Controle de Retorno (Return Control)**
   - O agente **devolve o controle ao aplicativo**, que toma a decisão e executa as ações.
   - Ideal para lógica executada localmente ou quando se quer maior controle do fluxo.

Integrações de Agentes do Amazon Bedrock

Visão Geral

- O Amazon Bedrock permite integrar dois tipos principais de recursos aos agentes:
  - **Bases de conhecimento (Knowledge Bases)**: Para permitir buscas contextuais e semânticas em dados privados.
  - **Guardrails (Salvaguardas)**: Para impor políticas de uso responsável e proteger contra respostas inadequadas.

---

1. Integração com Bases de Conhecimento

Objetivo

- Conectar os agentes a fontes de dados privadas e confiáveis.
- Permitir que o agente recupere **automaticamente** informações relevantes da base de conhecimento e as utilize para formular respostas mais precisas.

Funcionalidade

- Ao associar uma base de conhecimento ao agente:
  - Ele pode consultar o conteúdo quando necessário, sem intervenção externa.
  - Decide **quando e como usar as informações** disponíveis na base.
- É possível **adicionar uma base existente** no console por meio do Agent Builder, ou programaticamente via APIs.

Personalização da resposta

- Durante a execução, você pode configurar:
  - **Filtros de metadados**.
  - **Tipo de pesquisa** (semântica ou híbrida).
  - **Quantidade de documentos recuperados**.
- Essa personalização é feita utilizando configurações específicas no estado de sessão (`sessionState`) ao invocar o agente.

Benefícios principais

- **Maior precisão nas respostas**: As informações vêm de fontes confiáveis e atualizadas.
- **Melhor entendimento contextual**: Ajuda o agente a compreender o domínio, produtos e serviços envolvidos nas perguntas.
- **Consistência e escalabilidade**: As bases de conhecimento podem ser atualizadas e versionadas, mantendo a uniformidade nas respostas.
- **Facilidade de desenvolvimento**: Atua como fonte única de verdade, simplificando manutenção e integração.

Formas de conexão

- A base de conhecimento pode ser integrada:
  - Via **console (Agent Builder)**.
  - **Programaticamente**, por meio das APIs da AWS.
  - Utilizando ferramentas de infraestrutura como código:
    - AWS CloudFormation
    - AWS CDK
    - Terraform

---

2. Integração com Guardrails (Salvaguardas)

Objetivo

- Aplicar políticas de uso responsável da IA.
- Controlar e filtrar o comportamento do agente com base em diretrizes de segurança e conformidade específicas.

Integração com Guardrails no Amazon Bedrock

Objetivo

- Implementar salvaguardas e políticas de **IA responsável** em aplicações de IA conversacional.
- Proporcionar uma experiência de usuário **segura, consistente e conforme** com padrões organizacionais.

Como funciona a integração

- **Criação e conexão do guardrail**:
  - O guardrail é criado previamente e então conectado a um agente.
  - Essa integração pode ser feita via console, APIs, SDKs, ou ferramentas como CloudFormation, AWS CDK e Terraform.
- **Dupla verificação por consulta**:
  - O guardrail é ativado **duas vezes** por interação do usuário:
    i. **Verificação da entrada**: Avalia se a pergunta do usuário é segura. Se não for, o agente não responde.
    ii. **Verificação da saída**: Avalia se a resposta gerada é apropriada. Se for inadequada, a resposta é substituída por uma mensagem predefinida.

Benefícios da integração com Guardrails

- **Implementação responsável de IA**
  Garante alinhamento com diretrizes e princípios internos de IA ética e responsável.
- **Moderação de conteúdo**
  Permite configurar tópicos proibidos e filtros para bloquear conteúdo impróprio ou sensível, tanto na entrada quanto na resposta.
- **Experiência de usuário consistente**
  Mantém um padrão uniforme de segurança e conduta entre diferentes agentes e modelos de base.
- **Políticas personalizadas**
  Adapta os guardrails para diferentes casos de uso, aplicando regras específicas por contexto.
- **Controle sobre a aplicação**
  Fornece um mecanismo confiável para garantir que as interações respeitem os limites definidos.
- **Alinhamento ético**
  Promove transparência e confiança, reforçando o compromisso com práticas de IA ética.
- **Proteção da marca**
  Evita respostas que possam comprometer a reputação, melhorando a segurança da comunicação com os usuários.
- **Conformidade regulatória**
  Contribui para o atendimento de normas legais relacionadas à moderação de conteúdo, segurança e privacidade de dados.

Monitoramento e rastreamento

- Após a integração, o uso e o comportamento do guardrail podem ser acompanhados:
  - Pela interface do console.
  - Por meio das respostas retornadas nas chamadas de API.

Versões e Aliases do Agente

- **Versões do agente:**
  - São snapshots imutáveis do agente no momento da criação.
  - Preservam links para recursos como funções Lambda, bases de conhecimento e guardrails.
  - Para atualizar os recursos vinculados a um agente, é necessário criar uma nova versão.
  - Se um recurso vinculado for atualizado, a versão do agente continuará apontando para esse recurso atualizado.

- **Aliases do agente:**
  - Permitem alternar facilmente entre versões diferentes sem alterar o código do aplicativo.
  - Exemplo de uso: aliases para **DESENVOLVIMENTO**, **TESTE** e **PRODUÇÃO**.
  - Podem ser criados via Console AWS, SDKs, CloudFormation, AWS CDK ou Terraform.

Invocando um Agente

- **Métodos de execução:**
  - Agentes podem ser invocados via Console AWS ou por meio do SDK da API `bedrock-agent-runtime` (por exemplo, usando o boto3 em Python).
  - Métodos de tempo de execução não são compatíveis com infraestrutura como código (CloudFormation, CDK, Terraform).
- **Etapas de invocação:**
  a. **Atualizar SDKs** para garantir acesso aos recursos mais recentes.
  b. **Criar cliente do agente** com o serviço `bedrock-agent-runtime`.
  c. **Invocar o agente** usando a API `InvokeAgent`, fornecendo:
    - `agentId`: identificador exclusivo do agente.
    - `agentAliasId`: identificador do alias que será usado.
    - `sessionId`: identificador único da sessão de conversa.
    - `inputText`: prompt do usuário (opcional em alguns contextos).

Funcionalidades Adicionais do Agente

1. Rastreabilidade (Tracing)

- Permite visualizar as etapas internas da lógica do agente durante uma invocação.
- O rastreamento está disponível no console e via API, e é ativado com o parâmetro `enableTrace`.
- **Tipos de rastreamento disponíveis:**
  - **Pré-processamento:** valida a entrada do usuário.
  - **Orquestração:** lida com a lógica do agente, consultas, geração de código ou consulta a bases de conhecimento.
  - **Pós-processamento:** determina a resposta final enviada ao usuário.
  - **Guardrail:** verifica a conformidade da entrada e saída com as regras de segurança.
  - **Falha:** identifica erros e motivos de falhas.
- Cada rastreamento (exceto de falha) inclui o prompt usado na etapa, por meio do objeto `ModelInvocationInput`.

---

2. Sessões

- Um agente mantém o contexto da conversa durante uma sessão identificada por um `sessionId`.
- A sessão tem um tempo de vida limitado, configurável (máximo de uma hora).
- Após a expiração, os dados da conversa anterior são descartados.
- Sessões expiradas podem ser reutilizadas sem contexto anterior.
- No console, a conversa persiste até que a sessão seja encerrada manualmente ou expire.

---

3. Memória (em visualização pública)

- Permite que o agente retenha informações entre sessões diferentes com o mesmo usuário.

- Habilitada ao passar um `memoryId` na chamada `InvokeAgent`.
- Após cada sessão, é criado um **resumo da interação** do usuário.
- Esse resumo é utilizado para enriquecer interações futuras com o mesmo `memoryId`.
- É possível:
  - Visualizar a memória com `GetAgentMemory`.
  - Excluir memórias com `DeleteAgentMemory`.
- **Boas práticas:** usar um `memoryId` único por usuário para manter históricos individualizados.

---

4. Interpretação de Código (em visualização pública)

- Permite que o agente:
  - Gere código.
  - Execute o código em ambiente seguro.
  - Solucione problemas de forma automatizada.
- Casos de uso incluem: visualização de dados, operações matemáticas, transformações de dados.
- O recurso é habilitado durante a criação do agente.
- O agente pode decidir automaticamente quando usar esse recurso no tempo de execução.
- Também é possível forçar o uso ao:
  - Enviar arquivos usando o parâmetro `files` na chamada `InvokeAgent`.
  - Indicar o tipo de uso com o parâmetro `useCase` (valores possíveis: `CHAT` ou `CODE_INTERPRETER`).
- Os arquivos podem ser enviados como:
  - URL do Amazon S3.
  - Conteúdo de bytes diretamente.

Uso de sessionAttributes e promptSessionAttributes

Amazon Bedrock Agents não têm acesso automático a informações de contexto, como a data atual ou dados do usuário. Para fornecer esse tipo de informação ao agente sem exibir isso ao usuário, são utilizados os atributos `sessionAttributes` e `promptSessionAttributes`.

1. sessionAttributes

- **Persistência**: Os `sessionAttributes` permanecem disponíveis durante toda a sessão, desde que não expire (tempo de inatividade controlado pelo parâmetro `idleSessionTTLInSeconds`).
- **Não visível no prompt**: Essas informações **não são adicionadas ao prompt** enviado ao modelo de linguagem.
- **Aplicações típicas**:
  - Armazenar tokens de autenticação.
  - Controlar permissões de usuários.
  - Manter o estado de uma aplicação ao longo de múltiplas interações.
- **Acesso**: Disponível na função Lambda por meio do objeto de evento que o agente envia ao invocá-la.

2. promptSessionAttributes

- **Volatilidade**: Persistem **apenas durante um único turno de conversa**.
- **Visível no prompt**: São incorporadas ao prompt e compartilhadas com o modelo subjacente.
- **Aplicações típicas**:
  - Fornecer dados temporários ao modelo, como:
    - Data e hora atuais.
    - Informações contextuais para respostas relativas (ex: “ontem”, “amanhã”).

- **Edição do prompt**: É possível referenciar os atributos no prompt usando o marcador especial `$prompt_session_attributes$` no modelo de prompt de orquestração.

3. Integração com Funções Lambda

- `sessionAttributes` são acessados pela função Lambda diretamente do objeto de evento.
- `promptSessionAttributes` não exigem código extra para serem processados, pois já estão disponíveis no prompt usado pelo agente.

Considerações Técnicas

- Ambos os atributos só estão disponíveis via **API** (não disponíveis diretamente no console até o momento).
- Devem ser usados como boas práticas para fornecer **contexto contextual e personalizado** sem interferir na experiência do usuário.

Conclusão - Invocando Agentes no Amazon Bedrock

Nesta seção, você aprendeu como interagir com os agentes do Amazon Bedrock de duas maneiras principais:

1. **Interface de Usuário de Teste (Console AWS Management)**: Utilizando a interface gráfica para invocar e testar os agentes diretamente no console.
2. **Chamada de API usando SDKs**: Utilizando o SDK Amazon Bedrock para Python (boto3) para invocar os agentes programaticamente.

Você também aprendeu a aplicar funcionalidades avançadas, incluindo:

- **Rastreamento**: Habilitar o rastreamento para inspecionar e depurar a cadeia de execução do agente.
- **Sessões**: Manter uma conversa contínua com o agente ao longo de múltiplas interações usando sessões de conversa e IDs de sessão.
- **Retenção de Memória**: Usar a funcionalidade de memória para permitir que o agente lembre de interações passadas e melhore a experiência do usuário ao longo do tempo.
- **Interpretação de Código**: Habilitar a execução e a solução de problemas de código diretamente no ambiente do agente.
- **Atributos de Sessão**: Passar informações contextuais adicionais ao agente usando `sessionAttributes` e `promptSessionAttributes`, sem interferir na experiência do usuário.

Essas técnicas ajudam a criar interações mais dinâmicas e inteligentes, proporcionando uma melhor experiência ao usuário enquanto mantêm o controle sobre o fluxo da conversa e o contexto.

Tutorial:

Configuração e execução da demonstração do Amazon Bedrock Agents Integrated Lab

Configuração e execução da demonstração do Amazon Bedrock Agents Integrated Lab

Neste laboratório, você usa um agente assistente de compras para responder a perguntas sobre produtos de manutenção de gramados de duas empresas: AnyCompany Outdoor Power Equipment e AnyCompany LawnCare Solutions. Você usará o console para estudar e testar a base de conhecimento pré-construída, os guardrails e o aplicativo do agente. A base de conhecimento inclui detalhes do produto, como fabricante, descrição e classificação. O agente pode acessar funções que suportam cálculos simples e pesquisa de preço. Você usará um notebook Jupyter para praticar o uso de um agente em um aplicativo. Você lista os agentes e invoca o agente assistente de compras com parâmetros de sessão e memória para entender o impacto da memória de curto e longo prazo no comportamento do agente. Você também capturará e estudará o rastreamento para entender o funcionamento do agente, incluindo o uso de um assistente de compras com Amazon Bedrock Guardrails.

Nesta tarefa, você verificará se suas permissões do IAM permitem acesso aos recursos necessários para conduzir a demonstração.

Certifique-se de ter permissões para criar todos os ativos. Verifique o usuário ou a função sob a qual você está executando. Para facilitar a criação de todos os ativos, você pode querer executar com uma política que tenha todas as permissões, como neste exemplo.

No entanto, se você estiver executando em uma conta corporativa, isso pode não ser possível. Consulte seu administrador de sistema corporativo da AWS.

Verifique se você tem permissões para executar o laboratório. Se você configurar todas as permissões na etapa 1, isso não será necessário, mas em uma conta corporativa, isso pode ser necessário. Você pode usar a política "ReadOnlyAccess" gerenciada pela AWS para isso. Consulte seu administrador de sistema corporativo da AWS

Nesta tarefa, você usará o Amazon Bedrock para solicitar acesso aos modelos necessários para esta demonstração.

Navegue até o serviço Amazon Bedrock. Selecione Request model access.

Selecione Habilitar modelos específicos.

Certifique-se de que os seguintes serviços estejam selecionados:

- Soneto Claude 3
- Incorporações de texto Titan V2

Em seguida, role até a parte inferior da tela e escolha Avançar.

Aceite os termos de serviço escolhendo Enviar.

Role para baixo e certifique-se de que o acesso aos modelos foi concedido.

Você concluiu esta tarefa.

Nesta tarefa, você usará o Amazon SageMaker AI para criar seu domínio e perfil de usuário para acessar o SageMaker Studio. Você também criará seu ambiente JupyterLab.

Navegue até o serviço SageMaker AI.

Selecione Configuração para usuário único.

Selecione Setup for single user (Configuração para usuário único). Observe que isso leva aproximadamente 10 minutos.

Isso também cria um usuário padrão. Navegue até User profiles.

Selecione o botão Iniciar e escolha Estúdio.

Isso abrirá o SageMaker Studio em uma nova aba. Escolha Skip Tour for now se solicitado.

Escolha JupyterLab.

Escolha Criar espaço JupyterLab.

Insira MySpace no campo Nome e certifique-se de que Private esteja selecionado. Escolha Create space.

Após a criação do espaço, clique em Executar Espaço

Depois que o status do espaço for definido como em execução, escolha Abrir JupyterLab.

Isso abre o JupyterLab em uma nova aba.

Você concluiu esta tarefa.

Nesta tarefa, você usa o IAM para fornecer permissões adicionais para a função de execução do SageMaker AI.

Selecione a aba Domínio no seu navegador para retornar ao Amazon SageMaker AI.

Selecione a aba Configurações de domínio.

Role para baixo até Autenticação e permissões. Anote a função de execução Padrão.

Navegue até o IAM.

Selecione Funções no menu de navegação.

Financie a função SageMaker Execution que você acabou de anotar. Selecione-a.

Em Políticas de permissões, escolha Adicionar permissões e, em seguida, escolha Criar política em linha.

No editor de políticas, escolha JSON e adicione o código mostrado aqui. Você adicionará os seguintes itens em ação:

- base:Lista
- base:InvokeAgent
- base:Agente de atualização
- base:PrepareAgent
- base:ObterMemória do Agente
- Use um curinga depois de list e depois de Resource para permitir tudo.

Quando tiver feito isso, role para baixo e escolha Avançar.

Insira um nome no campo PolicyName e escolha Criar política.

Sua política foi criada.

Esta tarefa foi concluída.

Nesta tarefa, você cria os buckets do Amazon Simple Storage Solution (S3) necessários para hospedar seus documentos da base de conhecimento.

Navegue até S3.

Selecione Criar bucket no painel principal.

Na página Criar bucket, em Nome, insira knowledgebasebucket e adicione a região e o nome da sua conta para tornar o nome globalmente exclusivo.

Deixe todo o resto como padrão e role para baixo. Escolha Create bucket.

Escolha o link para o bucket que você acabou de criar.

Selecione Carregar link para carregar os arquivos da base de conhecimento.

Selecione Adicionar arquivos.

Selecione e carregue os seguintes arquivos do grupo de arquivos que você baixou anteriormente:

- Integração Agents_KB.docx
- Integração Agents_KB.docx.metadata.json
- Agentes_KB integração_2.docx
- Agentes_KB integração_2.docx.metadata.json

Selecione Abrir para selecionar os arquivos.

Os arquivos selecionados estão prontos para upload.

Role para baixo e escolha o botão Upload.

Os arquivos da base de conhecimento foram carregados com sucesso.

Você concluiu esta tarefa.

Em seguida, você precisará criar um balde para armazenar os logs do Bedrock.

Selecione Buckets no link embutido.

Selecione Criar bucket.

No campo Nome do bucket, comece inserindo o seguinte: bedrock-logging-

Em seguida, insira sua região AWS. Isso pode ser determinado usando o menu suspenso de regiões.

Por fim, você precisará do seu Account ID. Para obtê-lo, escolha a seta ao lado do AWS User/Role no canto superior direito para expandir o painel de informações.

Use o botão Copiar em ID da conta para copiar o número de ID.

O nome do seu bucket agora está completo.

Role para baixo até o final e selecione Criar bucket.

O banner mostra que você criou o bucket com sucesso.

Esta tarefa agora está concluída.

Nesta tarefa, você criará uma função Lambda que servirá para executar os agentes que você criará em uma tarefa posterior.

Navegue até o serviço Lambda.

Para criar uma função lambda do assistente de compras necessária para este laboratório, escolha Criar uma função.

Para esta função, deixe Autor do zero selecionado.

No nome da função, insira shopping-assistant-lambda.

Para o tempo de execução, escolha Python 3.13 ou a versão mais recente do Python no tempo de execução.

Deixe a arquitetura definida como x86_64.

Expanda Alterar função de execução padrão. Certifique-se de que Criar uma nova função com permissões básicas do Lambda esteja selecionado.

Deixe todo o resto como padrão e escolha Criar função.

O banner mostra que a função Lambda foi criada com sucesso

Role para baixo até o painel de código-fonte. Escolha a guia Configuration.

Em Configuração geral, escolha Editar.

Você precisará atualizar algumas das configurações básicas.

Para descrição, insira Lambda usado para execução do agente

Para Memória, digite 512.

Para Tempo limite, insira 12 minutos e 0 segundos.

Role para baixo e deixe todo o resto como padrão. Escolha Salvar.

O banner mostra que você atualizou a função com sucesso.

Role para baixo e escolha Permissões no menu de navegação.

Role para baixo até Declarações de política baseadas em recursos e escolha Adicionar permissões.

Role para baixo até Declarações de política baseadas em recursos e escolha Adicionar permissões.

Em Editar declaração de política, escolha Serviço AWS.

Em Serviço, use o menu suspenso, role até o final e escolha Outro.

Em StatementID, insira CustomPolicy ou algo semelhante.

Em Principal, insira o seguinte conforme mostrado:

bedrock.amazonaws.com

Para o ARN de origem, comece inserindo o seguinte:

arn:aws:base rochosa:

Insira a região em que você está trabalhando no momento. Neste caso, é us-east-1, mas você pode usar o menu suspenso de regiões para verificar.

Em seguida, você precisará do seu ID de conta. Para obtê-lo, escolha a seta ao lado do AWS User/Role no canto superior direito para expandir o painel de informações.

Use o botão Copiar para copiar o ID da conta.

Cole o ID da conta no campo ARN de origem após o texto digitado, sem espaços.

Por fim, adicione o seguinte conforme mostrado na tela para completar o Source ARN::agent/*

A mensagem ARN inválido deve desaparecer.

Para a última etapa, no campo Action, use o menu suspenso e role para baixo até lambda:InvokeFunction. Escolha esse item.

Role para baixo e escolha Salvar para concluir esta etapa.

O banner mostra que suas permissões foram atualizadas e salvas.

Role para baixo e selecione a aba Código.

Selecione o código na janela de código e exclua-o.

Abra o arquivo lambda_function.py do grupo de arquivos que você baixou anteriormente. Selecione todo o código no arquivo.

Cole o código copiado na janela Lambda Code source. Após colar o código, escolha Deploy.

O banner mostra que seu código foi atualizado.

Você concluiu esta tarefa.

Nesta tarefa, você criará uma base de conhecimento para o laboratório usando os documentos enviados ao S3.

Navegue até o serviço Amazon Bedrock.

Selecione Bases de conhecimento no menu de navegação.

No painel Como funciona, selecione Criar e escolha Base de conhecimento com armazenamento de vetores.

Você precisará inserir algumas informações nos detalhes da Base de conhecimento.

Em Nome da Base de Conhecimento, insira shopping_assistant_kb

Na descrição da Base de conhecimento, insira Base de conhecimento para o agente do Assistente de compras.

Para permissões do IAM, certifique-se de que Criar e usar uma nova função de serviço esteja selecionado.

Role para baixo. Em Choose data source, certifique-se de que Amazon S3 esteja selecionado.

Role até o final. Selecione Avançar.

No campo Nome da fonte de dados, insira shopping_assistant_kb-ds

Para Local da fonte de dados, certifique-se de que Esta conta da AWS esteja selecionada.

Para o URI S3, escolha o botão Navegar S3

Selecione o botão de opção knowledgebasebucket e clique em Escolher

Selecione o botão de opção para o knowledgebasebucket que você criou na Tarefa 6 e selecione Escolher.

Para a estratégia de análise, certifique-se de que o analisador padrão do Amazon Bedrock esteja selecionado.

Role para baixo. Em Chunking strategy, escolha o menu suspenso e selecione Fixed-size chunking.

Certifique-se de que o número máximo de tokens esteja definido como 300 e a porcentagem de sobreposição entre blocos esteja definida como 20.

Deixe todo o resto como padrão e escolha Avançar.

Em Modelo de incorporação, certifique-se de que Titan Text Embeddings v2 esteja selecionado.

Role para baixo.

Em Banco de dados de vetores, certifique-se de que Criar rapidamente um novo armazenamento de vetores – Recomendado tenha sido selecionado.

Em Loja de vetores, certifique-se de que Amazon OpenSearch Serverless tenha sido selecionado.

Selecione Avançar.

Revise suas configurações nesta página. Em seguida, role para baixo e escolha Create Knowledge Base.

Observe que a criação do banco de dados de vetores do OpenSearch leva aproximadamente 5 minutos.

O banner mostra que a Knowledge Base foi criada. Selecione Ir para fontes de dados.

Selecione o link para a fonte de dados que foi criada.

Role para baixo até Documentos e escolha Adicionar documentos.

Em URI do S3, escolha Procurar S3.

Selecione o link para o bucket da Base de Conhecimento que você criou anteriormente.

Selecione o Agents_KB integration.docx que você carregou anteriormente para este bucket. Selecione o botão Choose.

O primeiro documento foi adicionado. Para adicionar o segundo, escolha Add document.

No segundo painel Adicionar documento, em URI do S3, escolha Procurar no S3.

Selecione o link para o bucket da Base de Conhecimento.

Selecione o Agents_KB integration_2.docx que você carregou anteriormente para este bucket. Selecione o botão Choose.

Ambos os documentos foram selecionados. Escolha Adicionar.

Observe que adicionar os documentos leva aproximadamente 1 minuto.

Os banners mostram que ambos os itens foram adicionados com sucesso à Base de Conhecimento.

Selecione o link shopping_assistant_kb para a Base de Conhecimento.

Role para baixo até o painel Data Source e selecione a caixa de seleção para a fonte de dados. Após selecioná-la, escolha Sync.

O processo de sincronização começa, mas não deve demorar muito, aproximadamente 15 segundos.

O banner mostra que o processo de sincronização foi concluído com sucesso.

Por fim, você precisa adicionar o bucket de registro s3 à base de conhecimento.

Role até o topo da página e escolha Editar.

Role um pouco para baixo para ver todo o painel Log deliveries. Escolha o menu suspenso Add.

Usando o menu suspenso, escolha Para Amazon S3. Em seguida, selecione Na conta atual.

Use o botão Procurar para escolher um bucket S3 de destino.

Selecione o bucket que você criou que começa com bedrock-logging. Então escolha Select.

Selecione o bucket que você criou que começa com bedrock-logging. Então escolha Select.

Selecione Salvar alterações para finalizar a configuração das entregas de log.

O banner mostra que suas atualizações foram bem-sucedidas.

Esta tarefa está concluída.

Nesta tarefa, você criará proteções para um dos seus agentes.

No painel de navegação, escolha Guardrails.

Selecione o botão Criar guardrail para começar.

Insira algumas informações para começar a trabalhar no guardrail.

No campo Nome, digite shopping-assistant-guardrail

No campo Descrição, insira Guardrail para assistente de compras on-line para ajudar os usuários com dúvidas sobre compras

No campo Mensagens para prompts bloqueados, insira Desculpe, sua consulta viola nossas políticas de uso. Não fornecemos consultoria sobre manutenção de gramados. Para discutir as melhores opções para manter seu gramado, entre em contato com um profissional de gramados.

Role para baixo e selecione Avançar.

Para a Etapa 2, deixe todas as configurações como padrão e escolha Avançar.

Na Etapa 3, escolha Adicionar tópico negado.

No campo Nome, insira Conselho de Manutenção de Gramado

No campo Definição, insira Conselhos sobre manutenção de gramado.

Expanda a seção Adicionar frases de exemplo.

Adicione as seguintes frases, uma de cada vez:

- Com que frequência devo cortar a grama?
- Que tipo de fertilizante devo usar?
- Como se livrar de ervas daninhas?

Quando terminar, selecione Confirmar.

O guardrail foi configurado. Selecione Pular para revisar e criar.

Revise as informações inseridas rolando a página para baixo. Selecione Create guardrail para finalizar.

O guarda-corpo foi instalado e o status mostra que ele está pronto.

Você concluiu esta tarefa.

Nesta tarefa, você criará o primeiro agente para usar com o assistente de compras. Este agente não usará guardrails.

Selecione Agentes no painel de navegação.

Selecione Criar agente no painel Agentes.

No campo Nome, digite shopping-assistant

No campo Descrição, insira Agente de demonstração que implementa um assistente de compras.

Deixe todo o resto como padrão e escolha Criar.

Isso leva você para a tela Agent Builder. O banner mostra que o agente foi criado com sucesso.

Verifique a função de recurso do agente para certificar-se de que Criar e usar uma nova função de serviço foi criada.

Role para baixo até Selecionar modelo. Escolha o botão Selecionar modelo.

No pop-up Selecionar modelo, escolha Antrópico em Categorias.

Na coluna Modelos, escolha Claude 3 Sonnet.

Para Inferência, Sob demanda deve ser selecionado por padrão.

Quando terminar, escolha Aplicar.

O modelo Claude 3 Sonnet foi selecionado.

No campo Instruções para o agente, insira o seguinte:

Você é um assistente de compras de IA e responde a perguntas relacionadas a produtos. Não faça suposições ou invente uma resposta. Você não é bom em matemática. Então, use a ferramenta MultiFunctionCalculatorTool fornecida para responder a perguntas de matemática. Você SEMPRE responde educadamente e concisamente, usando SOMENTE a BASE DE CONHECIMENTO e as funções PriceLookup e MultiFunctionCalculatorTool disponíveis no ACTION_GROUP.

Expanda a seção Configurações adicionais.

No campo Tempo limite da sessão ociosa, altere o tempo para 3600 segundos.

Deixe todo o resto como padrão e role para cima até o topo.

Selecione o botão Salvar para salvar suas alterações e permanecer no Construtor de agentes.

Role para baixo até Action groups. Selecione Add.

No painel de detalhes do grupo Ação, adicione o seguinte:

No campo Nome, digite shopping-assistant

No campo Descrição, insira o grupo de ação que implementa a funcionalidade básica de consulta do produto

Para o tipo de grupo de ação, certifique-se de que Definir com detalhes da função esteja selecionado.

Role para baixo até Invocação do grupo de ação.

Para selecionar como definir a função Lambda, selecione o botão de opção para Selecionar uma função Lambda existente

Selecione a lista suspensa Selecionar função Lambda e selecione shopping-assistant-lambda

Deixe tudo como padrão e role para baixo até a função 1 do grupo de ação.

No campo Nome, digite PriceLookup

No campo Descrição, insira Útil quando precisar consultar o preço do produto. Forneça o ID do produto ou o nome do produto

Depois de inseri-los, escolha Adicionar parâmetro.

Selecione o espaço vazio na coluna Name e insira productid no campo disponível. Selecione a marca de seleção para confirmar sua entrada.

Selecione o espaço vazio na coluna Descrição e insira o identificador do produto no campo disponível. Selecione a marca de seleção para confirmar sua entrada.

Selecione a entrada False na coluna Required e use o menu suspenso para selecionar True. Selecione a marca de seleção para confirmar sua entrada.

Selecione o botão Adicionar função de grupo de ação para concluir a configuração desta função de grupo de ação.

Role para baixo até a função do grupo de ação 2.

No campo Nome, digite MultiFunctionCalculatorTool

No campo Descrição, insira Útil quando você precisa adicionar, subtrair, multiplicar ou dividir dois números. Para usar a ferramenta, você deve fornecer os dois números e o operador.

Depois de inseri-los, escolha Adicionar parâmetro.

Selecione o espaço vazio na coluna Name e insira oper1 no campo disponível. Selecione a marca de seleção para confirmar sua entrada.

Selecione o espaço vazio na coluna Description e insira o primeiro operando no campo disponível. Selecione a marca de seleção para confirmar sua entrada.

Selecione a entrada False na coluna Required e use o menu suspenso para selecionar True. Selecione a marca de seleção para confirmar sua entrada.

Selecione Adicionar parâmetro para adicionar um segundo parâmetro.

Selecione o espaço vazio na coluna Name e insira oper2 no campo disponível. Selecione a marca de seleção para confirmar sua entrada.

Selecione o espaço vazio na coluna Description e insira o segundo operando no campo disponível. Selecione a marca de seleção para confirmar sua entrada.

Selecione a entrada False na coluna Required e use o menu suspenso para selecionar True. Selecione a marca de seleção para confirmar sua entrada.

Selecione Adicionar parâmetro para adicionar um terceiro parâmetro.

Selecione o espaço vazio na coluna Nome e insira operador no campo disponível. Selecione a marca de seleção para confirmar sua entrada.

Selecione o espaço vazio na coluna Descrição e insira operador no campo disponível.

Selecione a marca de seleção para confirmar sua entrada.

Selecione a entrada False na coluna Required e use o menu suspenso para selecionar True. Selecione

a marca de seleção para confirmar sua entrada.

Role para baixo até o final. Selecione Criar.

Após a conclusão, você será levado de volta ao Agent Builder.

Selecione o botão Salvar para salvar suas alterações e permanecer no Construtor de agentes.

No Agent builder, role para baixo até Knowledge Bases. Selecione Add.

Em Bases de conhecimento, use o menu suspenso para selecionar shopping_assistant_kb.

No campo Instruções, insira Acessar a base de conhecimento quando os clientes fizerem perguntas sobre a descrição do produto, a classificação do produto ou os produtos disponíveis.

Quando terminar, escolha Adicionar.

De volta ao Agent Builder, clique no botão Salvar e Sair

Usando o painel Teste, clique no botão Preparar.

O banner e o status mostram que o agente foi preparado.

Você concluiu esta tarefa.

Nesta tarefa, você criará o primeiro agente para usar com o assistente de compras. Este agente estará sujeito a guardrails.

No console Bedrock, no painel esquerdo, selecione Agentes

Selecione Criar agente.

No campo Nome, digite shopping-assistant-with-guardrails.

No campo Descrição, insira Assistente de compras com guarda-corpos.

Quando terminar, escolha Criar.

O agente foi criado.

Para este agente, para a função de recurso Agente, selecione o botão de opção Usar uma função de serviço existente.

Como você já criou uma função de agente na tarefa anterior, você pode reutilizá-la para este agente.

Role para baixo até a seção Selecionar modelo e escolha o botão Selecionar modelo.

Faça as mesmas seleções da tarefa anterior. Neste caso, é Anthropic em Categories

e Claude 3 Sonnet em Models. Todo o resto é padrão.

Quando terminar, escolha Aplicar.

O modelo Claude 3 Sonnet foi selecionado.

No campo Instruções para o agente, insira o seguinte:

Você é um assistente de compras de IA e responde a perguntas relacionadas a produtos. Não faça suposições ou invente uma resposta. Você não é bom em matemática. Então, use a ferramenta MultiFunctionCalculatorTool fornecida para responder a perguntas de matemática. Você SEMPRE responde educadamente e concisamente, usando SOMENTE a BASE DE CONHECIMENTO e as funções PriceLookup e MultiFunctionCalculatorTool disponíveis no ACTION_GROUP.

Expanda a seção Configurações adicionais.

No campo Tempo limite da sessão ociosa, altere o tempo para 3600 segundos.

Deixe todo o resto como padrão e role para cima até o topo.

Selecione o botão Salvar para salvar suas alterações e permanecer no Construtor de agentes.

Role para baixo até Action groups. Selecione Add.

No painel de detalhes do grupo Ação, adicione o seguinte:

No campo Nome, digite shopping-assistant

No campo Descrição, insira o grupo de ação que implementa a funcionalidade básica de consulta do produto

Para o tipo de grupo de ação, certifique-se de que Definir com detalhes da função esteja selecionado.

Role para baixo até Invocação do grupo de ação.

Para selecionar como definir a função Lambda, selecione o botão de opção para Selecionar uma função Lambda existente

Selecione a lista suspensa Selecionar função Lambda e selecione shopping-assistant-lambda

Deixe tudo como padrão e role para baixo até a função 1 do grupo de ação.

No campo Nome, digite PriceLookup

No campo Descrição, insira Útil quando precisar consultar o preço do produto. Forneça o ID do produto ou o nome do produto

Depois de inseri-los, escolha Adicionar parâmetro.

Selecione o espaço vazio na coluna Name e insira productid no campo disponível. Selecione a marca de seleção para confirmar sua entrada.

Selecione o espaço vazio na coluna Descrição e insira o identificador do produto no campo disponível. Selecione a marca de seleção para confirmar sua entrada.

Selecione a entrada False na coluna Required e use o menu suspenso para selecionar True. Selecione a marca de seleção para confirmar sua entrada.

Selecione o botão Adicionar função de grupo de ação para concluir a configuração desta função de grupo de ação.

Role para baixo até a função do grupo de ação 2.

No campo Nome, digite MultiFunctionCalculatorTool

No campo Descrição, insira Útil quando você precisa adicionar, subtrair, multiplicar ou dividir dois números. Para usar a ferramenta, você deve fornecer os dois números e o operador.

Depois de inseri-los, escolha Adicionar parâmetro.

Selecione o espaço vazio na coluna Name e insira oper1 no campo disponível. Selecione a marca de seleção para confirmar sua entrada.

Selecione o espaço vazio na coluna Description e insira o primeiro operando no campo disponível. Selecione a marca de seleção para confirmar sua entrada.

Selecione a entrada False na coluna Required e use o menu suspenso para selecionar True. Selecione a marca de seleção para confirmar sua entrada.

Selecione Adicionar parâmetro para adicionar um segundo parâmetro.

Selecione o espaço vazio na coluna Name e insira oper2 no campo disponível. Selecione a marca de seleção para confirmar sua entrada.

Selecione o espaço vazio na coluna Description e insira o segundo operando no campo disponível. Selecione a marca de seleção para confirmar sua entrada.

Selecione a entrada False na coluna Required e use o menu suspenso para selecionar True. Selecione a marca de seleção para confirmar sua entrada.

Selecione Adicionar parâmetro para adicionar um terceiro parâmetro.

Selecione o espaço vazio na coluna Nome e insira operador no campo disponível. Selecione a marca de seleção para confirmar sua entrada.

Selecione o espaço vazio na coluna Descrição e insira operador no campo disponível. Selecione a marca de seleção para confirmar sua entrada.

Selecione a entrada False na coluna Required e use o menu suspenso para selecionar True. Selecione a marca de seleção para confirmar sua entrada.

Role para baixo até o final. Selecione Criar.

Após a conclusão, você será levado de volta ao Agent Builder.

Selecione o botão Salvar para salvar suas alterações e permanecer no Construtor de agentes.

No Agent builder, role para baixo até Knowledge Bases. Selecione Add.

Em Bases de conhecimento, use o menu suspenso para selecionar shopping_assistant_kb.

No campo Instruções, insira Acessar a base de conhecimento quando os clientes fizerem perguntas sobre a descrição do produto, a classificação do produto ou os produtos disponíveis.

Quando terminar, escolha Adicionar.

Selecione o botão Salvar para salvar suas alterações e permanecer no Construtor de agentes.

Role para baixo até Guardrail details. Selecione Edit.

Usando o menu suspenso Selecionar guardrail, selecione o shopping-assistant-guardrail que já foi criado.

Deixe todo o resto como padrão e escolha Salvar e Sair.

Seu agente com guardrails foi criado.

Na parte superior do construtor do agente, clique no botão Salvar e sair

Usando o painel Teste, clique no botão Preparar.

O banner e o status mostram que o agente foi preparado.

Você concluiu esta tarefa.

Após concluir essas tarefas, você concluiu com sucesso o seguinte:

- Verificou suas permissões
- Acesso de modelo habilitado no Bedrock
- Crie seu ambiente de demonstração usando o SageMaker AI
- Crie seus buckets S3
- Criou uma função lambda
- Criei uma base de conhecimento e guardrails
- Agentes criados

Explore os agentes do Amazon Bedrock integrados com as bases de conhecimento do Amazon Bedrock e os Amazon Bedrock Guardrails

Explore os agentes do Amazon Bedrock integrados com as bases de conhecimento do Amazon Bedrock e os Amazon Bedrock Guardrails

Neste laboratório, você usa um agente assistente de compras para responder a perguntas sobre produtos de manutenção de gramados de duas empresas: AnyCompany Outdoor Power Equipment e AnyCompany LawnCare Solutions. Você usará o console para estudar e testar a base de conhecimento pré-construída, os guardrails e o aplicativo do agente. A base de conhecimento inclui detalhes do produto, como fabricante, descrição e classificação. O agente pode acessar funções que suportam cálculos simples e pesquisa de preço. Você usará um notebook Jupyter para praticar o uso de um agente em um aplicativo. Você lista os agentes e invoca o agente assistente de compras com parâmetros de sessão e memória para entender o impacto da memória de curto e longo prazo no comportamento do agente. Você também capturará e estudará o rastreamento para entender o funcionamento do agente, incluindo o uso de um assistente de compras com Amazon Bedrock Guardrails.

Nesta tarefa, você acessa o ambiente JupyterLab e carrega o arquivo de notebook Agents_Lab.ipynb.

Se você acabou de concluir a configuração, você ainda deve ter o ambiente do laboratório Jupyter configurado e aberto. Use a aba do seu navegador para alternar para o JupyterLab.

Quando estiver no JupyterLab, escolha o ícone de upload.

Navegue até o arquivo de notebook Agents_Lab no seu computador local e escolha Abrir.

O arquivo de notebook Agents_Lab agora está disponível para você usar.

Nesta tarefa, você habilitará a memória do Agente no console Bedrock.

Use as abas do seu navegador para retornar à aba Amazon Bedrock.

Agora você editará a Configuração para o Agente do Assistente de Compras. No painel de navegação esquerdo, em Ferramentas do Builder, escolha Agentes.

Você tem os dois agentes que criou e configurou. Shopping-assistant será usado para testar as respostas sem Guardrails, e Shopping-assistant-with-guardrails será usado para testar respostas com Guardrails habilitados.

Selecione o agente assistente de compras escolhendo o link.

Observação: na seção Visão geral do agente, observe que a Memória está desabilitada.

Selecione Editar no Agent Builder.

Role a página para baixo até a seção Memória.

Na seção Memória, em Habilitar memória, escolha Habilitar.

Role de volta para o topo e escolha Salvar.

No painel direito, escolha Preparar.

Aguarde alguns segundos para que o Agente seja preparado. Quando o banner mostrar que o agente foi atualizado com sucesso, escolha Salvar e sair.

Observe que o Status está definido como Preparado e a Memória está Ativada.

Tarefa concluída! Você configurou e habilitou com sucesso a Short-Term Memory para o Bedrock Agent.

Nesta tarefa, você executa o arquivo de notebook Agents_Lab para praticar o uso de um Bedrock Agent em um aplicativo. Por meio do Agent, você acessa a Knowledge Base, que inclui detalhes do produto, como fabricante, descrição e classificação. Além disso, o agente pode acessar funções que oferecem suporte a cálculos simples e pesquisa de preço.

Use as abas do navegador para retornar ao JupyterLab.

Para executar este laboratório, no painel de navegação esquerdo, abra o notebook Agents_Lab.ipynb para sua tarefa.

Neste caderno, você usa um agente de assistente de compras para responder perguntas sobre produtos de manutenção de gramados de duas empresas: AnyCompany Outdoor Power Equipment e AnyCompany LawnCare Solutions.

Execute as células neste notebook para simular uma conversa de bate-papo com o agente, para testar a base de conhecimento pré-criada, os guardrails e o aplicativo do agente.

A base de conhecimento inclui detalhes do produto, como fabricante, descrição e classificação. O agente pode acessar funções que suportam cálculos simples e pesquisa de preço.

Nas etapas a seguir, você lista os agentes e invoca o agente do assistente de compras com parâmetros de sessão e memória para entender o impacto da memória de curto prazo e de longo prazo no comportamento do agente. Você também captura e estuda o rastro para entender o funcionamento do agente, incluindo o uso de um assistente de compras com guardrails.

Nesta primeira tarefa, Tarefa 3.1.1, você configura o ambiente do notebook importando os pacotes necessários.

Execute as duas células de código a seguir para importar os pacotes necessários e configurar seu ambiente.

Na próxima tarefa, Tarefa 3.1.2, você configura objetos e funções auxiliares do Boto3.

Execute a primeira célula de código para configurar os clientes Session, AWS Region e Bedrock.

Execute a próxima célula de código para configurar uma função auxiliar para encontrar o ID do agente pelo nome do agente.

Execute a terceira célula de código para invocar a função auxiliar find_agent_id_by_agent_name para obter o ID do agente associado ao agente do assistente de compras.

Na célula, você verá que um **alias_id** está definido. Esse **alias_id** será usado em etapas posteriores quando o Resumo da Sessão for solicitado.

Por fim, execute a quarta célula de código para criar uma função auxiliar para invocar o Bedrock Agent e capturar memória.

Na próxima tarefa, Tarefa 3.1.3, você interagirá com o agente Bedrock sem guardrails.

Ao interagir e conversar com agentes da Bedrock, há dois componentes que ajudam a manter o estado:

- session_id representa uma conversa com um agente em várias perguntas. Para continuar a mesma conversa com um agente, use o mesmo valor session_id na solicitação.
- memory_id é onde o histórico de conversas e o contexto de cada usuário são armazenados com segurança, garantindo a separação completa entre os usuários.

Execute a primeira célula de código para configurar o session_id e o memory_id.

O session_id e o memory_id foram gerados e listados aqui no notebook.

Em seguida, você simulará um diálogo com o agente. Faça uma pergunta inicial ao agente para iniciar a conversa. Algumas das perguntas ao agente serão respondidas a partir de informações armazenadas na Base de Conhecimento. Como o registro está ativo, você notará latência. Observe que o agente **shopping-assistant** está sendo invocado. Este agente não tem nenhum Guardrails associado.

Observe que este laboratório usa uma conta AWS de não produção com um limite de taxa de invocação de modelo. Se qualquer uma das células subsequentes que postarem perguntas para o agente retornar um **Throttling Error** , aguarde até 60 segundos e execute novamente essa célula.

Execute a próxima célula de código para simular um diálogo com o agente.

O diálogo com o agente é publicado no arquivo do notebook.

A resposta mostra que os produtos são fabricados por duas empresas, o que era esperado.

Continue fazendo perguntas sobre os produtos.

Execute a próxima célula para fazer uma pergunta sobre os comentários dos clientes sobre os produtos.

A resposta contém informações como classificações de clientes e número de avaliações para cada produto.

Execute a próxima célula para fazer uma pergunta sobre o custo de cada produto.

A resposta inclui o preço individual de cada produto.

Execute a próxima célula para fazer uma pergunta que levará o agente a usar um cálculo matemático. Neste caso, ele está calculando o custo de dois aparadores de cordas.

A resposta mostra que o agente calculou o custo de dois aparadores de cordas e divulgou o total.

Faça uma pergunta final nesta sessão. Como você está ligando para o agente **de assistência de compras** sem Guardrails, faça uma pergunta que não será bloqueada.

Execute esta célula para fazer uma pergunta não relacionada a produtos ou estoque.

O agente responde que não tem nenhuma informação relacionada à pergunta que foi feita. Como não havia guardrails, essa pergunta não foi bloqueada.

O último bloco de código para esta tarefa fecha a sessão definindo **end_session=True** para que você possa acessar o resumo da sessão mais tarde no armazenamento de memória.

Execute o bloco de código.

A resposta confirma que a última ação foi bem-sucedida.

Na Tarefa 3.1.4, você interage com os logs de invocação do agente Bedrock.

Aqui, você define um conjunto de funções auxiliares para localizar, baixar e salvar o log de invocação recente do Bedrock e imprimir o conteúdo json no arquivo gzipado baixado. Essa abordagem reduz a quantidade de saída na tela.

A outra opção para obter o log de invocação é habilitar o rastreamento enquanto faz uma pergunta ao agente.

Você habilitará o rastreamento ao usar o agente com guardrail.

Execute a primeira célula para encontrar o registro recente do Bedrock.

Execute a próxima célula para baixar o log Bedrock recente. Aguarde aproximadamente um minuto para que o log Bedrock apareça.

Execute a próxima célula para baixar e imprimir um log json.

Por fim, você imprimirá o trace para a invocação recente do agente. Observe que o código dorme por um minuto para garantir que os logs mais recentes estejam sendo buscados.

Execute a última célula para prosseguir.

A saída do log mostra o raciocínio usado pelo agente para responder às perguntas do usuário. O log foi baixado e descompactado e está disponível no seu diretório de laboratório.

Abra o arquivo de log.

Reserve alguns minutos para estudar o registro.

Feche o arquivo de log quando terminar de revisá-lo.

Na tarefa 3.1.5, você vai interagir com o agente Bedrock com guardrails. Você vai fazer a última pergunta buscando conselhos sobre ervas daninhas para o agente **shopping-assistant-with-guardrails** .

Execute a primeira célula para fazer a pergunta.

Como enable_trace está definido como true, você pode ver a saída do trace na tela. A saída do trace também está disponível no arquivo de log.

Observe que o agente se recusa a fornecer conselhos sobre manutenção do gramado para a última pergunta da lista acima.

Execute a próxima célula para imprimir o log para a invocação recente do agente. O código dorme por um minuto para garantir que os logs mais recentes sejam buscados.

O log foi gerado e está pronto para download.

Por fim, na Tarefa 3.1.5, você interagirá com a memória de sessão do agente.

Mais de dois minutos se passaram desde o fechamento da primeira sessão. O armazenamento de memória com um resumo da primeira sessão deve estar pronto agora. Isso pode ser útil para auditoria, depuração ou compreensão do raciocínio do agente em várias conversas.

A função abaixo verifica se o Resumo da Sessão foi finalizado, imprimindo o Resumo da Sessão.

Se o Resumo ainda não estiver disponível, o código abaixo aguardará 30 segundos e tentará novamente até 5 vezes.

Execute esta célula. Após executar esta célula, se o Resumo ainda não estiver disponível, você pode esperar mais um minuto e executar esta célula novamente.

O Resumo está disponível e impresso para revisão.

Execute a última célula para finalizar esta conversa com o agente, pedindo que ele se lembre do que foi discutido.

O agente fornece um resumo da sua atividade de perguntas recapitulando o que você pediu e compartilhando como ele calculou o custo total com base na sua solicitação.

Conclua as tarefas de limpeza para finalizar este laboratório.

Tarefa concluída: Você executou com sucesso todas as células no Agents_Lab.ipynb, permitindo que o

Agente para coletar informações da Base de Conhecimento, como pesquisas de preços, além de realizar cálculos matemáticos básicos.

Após concluir essas tarefas, você terá feito o seguinte com sucesso:

- Analisou a interação de uma base de conhecimento e guardrails com um Bedrock Agent
- Analisou os componentes de um agente, incluindo grupos de ação, memória e guardrails
- Criei um aplicativo básico que usa um agente que tem acesso a funções, bases de conhecimento e guardrails.
- Capturou e analisou um rastro para entender o comportamento do agente, incluindo o uso da memória de curto e longo prazo.