

Resumo AWS

Amazon Q Business [🔗](#)

O que é Amazon Q Business? [🔗](#)

💡 Assistente de IA Generativa

- Facilita a busca de informações, geração de conteúdo e automação de fluxos de trabalho usando linguagem natural.

💡 Integração com Ferramentas Corporativas

- Funciona via web e pode ser incorporado a Slack, Microsoft Teams e outros sistemas.

💡 Gestão Simplificada e Sem Infraestrutura

- Serviço totalmente gerenciado, eliminando a necessidade de infraestrutura complexa.
- Configuração intuitiva, sem necessidade de programação.

💡 Conectividade com Fontes de Dados

- Integra-se com Amazon S3, Salesforce, Oracle, entre outros.
- Suporte a dados em nuvem e locais.

💡 Segurança e Controle de Acesso

- Herda permissões dos sistemas empresariais integrados.
- Permite restrições administrativas para bloquear termos ou tópicos específicos.

Como a Amazon Q Business funciona?

✅ Autenticação e Autorização

- Utiliza SAML 2.0 para autenticação com provedores de identidade compatíveis.

✅ Interação via Linguagem Natural

- Usuários enviam perguntas ou comandos em linguagem natural via web ou chat.
- Oferece uma experiência conversacional integrada com IA generativa.

✅ Acesso a Dados Empresariais

- Conecta-se a mais de 40 fontes de dados corporativos por meio de conectores nativos e APIs.
- Respostas personalizadas são geradas com base nos controles de acesso do usuário.

✅ Ações Diretas em Aplicações Empresariais

- Permite criar tickets, casos e incidentes diretamente na interface do Amazon Q Business.
- Integrações com Salesforce, ServiceNow, Jira, Zendesk via plugins nativos.

✅ Integração com Ferramentas de Trabalho

- Pode ser incorporado a Slack e Microsoft Teams usando APIs embutidas.
- Usuários interagem diretamente sem precisar trocar de interface.

📌 Conceitos Básicos [🔗](#)

🔑 IAM Identity Center

- Necessário para criar e usar aplicações no Amazon Q Business.
- Funciona como uma camada adicional ao seu provedor de identidade existente.

- Permite gerenciar usuários e grupos para acesso à aplicação.

RAG (Retrieval Augmented Generation)

- Modelos de IA generativa são treinados offline e podem não incluir dados recentes.
- RAG busca dados externos e os adiciona ao contexto para respostas mais precisas.
- O Amazon Q Business gerencia essa técnica automaticamente, sem necessidade de ajustes manuais.

Controle de Acesso a Dados

- Garante que usuários só acessem conteúdos permitidos com base em suas permissões.
- Suporta autenticação via SAML 2.0 com provedores como Okta, Microsoft Entra ID e Ping Identity.





Integração e Atualização de Dados

- Conectores pré-configurados facilitam a integração com fontes de dados empresariais.
- Suporte a sincronização completa ou incremental de dados.

Plugins

- Integração com Jira, ServiceNow, Salesforce, Zendesk, entre outros.
- Expande funcionalidades sem necessidade de desenvolvimento adicional.

Casos de Uso da Amazon Q Business

-  Acelerar a Criação de Conteúdo
 -  Experiência de Busca Empresarial Otimizada
 -  Geração de Sumários de Documentação
 -  Tomada de Decisões Informadas
-

Criando Buckets no Amazon S3

1 Acessar o Console AWS

- Faça login no AWS Management Console e selecione a Região AWS apropriada.

2 Criar um S3 Bucket

- No campo de busca, digite S3 e selecione a opção correspondente.
- Na página do S3, clique em Create bucket.
- Escolha um nome único e em letras minúsculas.

3 Configurar e Criar o Bucket

- Role até o final da página e mantenha as configurações padrão.
- Clique em Create bucket.

4 Fazer Upload dos Arquivos

- Após a criação, selecione o nome do bucket na lista.
- Escolha Upload ou arraste os arquivos para a página.
- Clique novamente em Upload para confirmar.

Como Criar uma IA no Amazon Q Business

1 Acessar o Console AWS

- ♦ Verifique a Região AWS selecionada.
- ♦ Busque por Amazon Q Business e selecione a opção correspondente.
- ♦ Na página inicial, clique em Get Started.

2 Criar a Aplicação

- ♦ Clique em Create application.
- ♦ Defina um nome para a aplicação e mantenha as configurações padrão.
- ♦ O IAM Identity Center será atribuído automaticamente.
- ♦ Clique em Create.

3 Selecionar um Data Retriever

- ♦ Escolha Use native retriever e clique em Next.

4 Conectar Fontes de Dados

- ♦ Clique no + ao lado de Amazon S3.
- ♦ Dê um nome à fonte de dados e selecione Create a new service role.
- ♦ Escolha o bucket do S3 criado anteriormente.
- ♦ Defina a sincronização como Run on demand e clique em Add data source.

5 Adicionar Usuários e Grupos

- ♦ Clique em Add groups and users.
- ♦ Selecione Assign existing users and groups e clique em Next.
- ♦ Busque o usuário criado (exemplo: demo_user) e clique em Assign.

6 Finalizar a Criação da Aplicação

- ♦ Confirme a atribuição de usuários e clique em Create application.
- ♦ Após a criação, verifique o status de implantação da Web Experience.
- ♦ Clique no nome da aplicação para acessar as configurações.

7 Sincronizar Dados

- ♦ Selecione a fonte de dados e clique em Sync now.
- ♦ O processo pode levar minutos ou horas, dependendo do volume de dados.

8 Configurar Controles de Administração

- ♦ No menu, acesse Admin controls and guardrails.
- ♦ Ative as opções:

✓ Permitir consultas diretas ao LLM

✓ Permitir fallback para o LLM

- ♦ Clique em Save.

9 Personalizar a Web Experience

- ♦ No menu, clique em Customize web experience.
- ♦ Edite o título, subtítulo e mensagem de boas-vindas.
- ♦ Salve as alterações.

10 Acessar a Aplicação

- ♦ Obtenha a Web Experience URL na lista de aplicações.
- ♦ O usuário autenticado poderá interagir com a IA na interface web.

Conversando com o Amazon Q Business

1 Acessar a Web Experience

- ♦ Após a criação da aplicação, utilize a Web Experience URL disponível na lista de aplicações.
- ♦ Abra a URL em um navegador.

2 Autenticação do Usuário

- ♦ Insira o nome de usuário e clique em Next.
- ♦ Digite a senha e clique em Sign in.
- ♦ Insira o código MFA do app autenticador e clique novamente em Sign in.

3 Explorando a Interface

- ◆ Acesse a seção de prompts para iniciar a conversa.
- ◆ A aba Conversations mantém um histórico de conversas.

4 Fazendo Perguntas

- ✓ O Amazon Q Business responde e exibe as fontes de dados usadas.
- ◆ Se a resposta não estiver nos dados carregados, a IA pode recorrer ao LLM.
- ⚠ Respostas dos LLMs podem ser genéricas ou imprecisas.

5 Gerando Conteúdo

- ✓ A IA gera um resumo com referências às fontes.

6 Interagindo com Arquivos

- ◆ Faça upload de arquivos  para conversas específicas.

7 Gerando Resumos com Arquivos

- ✓ A partir de arquivos é possível gerar resumos, basta fazer o upload do arquivo.

Deletando Recursos no Amazon Q Business

1 Removendo a Aplicação

- ◆ Acesse a página Amazon Q Business > Applications.
- ◆ Selecione a aplicação que vai ser excluída.
- ◆ No menu Actions, escolha Delete.
- ◆ Confirme digitando Delete na caixa de texto e clique novamente em Delete.

2 Deletando o Bucket S3

- ◆ Volte para a página inicial do Amazon S3.
- ◆ No menu esquerdo, selecione Buckets.
- ◆ Escolha o bucket criado para a demonstração.

3 Esvaziando o Bucket

- ◆ Antes da exclusão, o bucket precisa estar vazio.
- ◆ Escolha Empty.
- ◆ Na página de confirmação, digite permanently delete e clique em Empty.
- ◆ Após a exclusão dos arquivos, clique em Exit.



4 Excluindo o Bucket

- ◆ Selecione novamente o bucket.
- ◆ Escolha Delete.
- ◆ Digite o nome do bucket para confirmar e clique em Delete bucket.

Amazon Q Developer

Amazon Q Developer: Impulsionando a Produtividade dos Desenvolvedores

O Problema

- ◆ Mais de 70% do tempo dos desenvolvedores é gasto em atividades repetitivas.
- ◆ Essas tarefas incluem:
 - Escrita de código boilerplate (unidade de escrita que pode ser reutilizada continuamente sem alteração).
 - Criação e execução de testes unitários 
 - Tradução de código entre linguagens 

⚡ A Solução

- O Amazon Q Developer acelera o desenvolvimento de novas funcionalidades.
 - A ferramenta:
 - Sugere código automaticamente 📝
 - Executa varreduras de segurança 🔍
 - Reduz custo, tempo e risco no desenvolvimento de aplicações 📊
-

🏗️ Arquitetura [🔗](#)

1 Autenticação e Autorização

- O AWS IAM Identity Center gerencia usuários e permissões.

2 Interação via Chat

- Usuários enviam consultas em linguagem natural ao Amazon Q Developer.
- Respostas são geradas com base:
- Nos dados empresariais disponíveis 📊
- Na documentação da AWS 📖 (ao usar a função "Diagnose with Amazon Q").

3 Principais Funcionalidades

✅ Geração de Código Inline

- Sugestões de código linha por linha ou em blocos completos 🏗️

✅ Customizações

- Recomendações personalizadas com base em bibliotecas internas, APIs e padrões arquiteturais 🎯

✅ Transformação de Código

- Atualização automática de código para versões mais recentes da linguagem ↻

✅ Desenvolvimento de Funcionalidades

- Definição, colaboração e solução de tarefas de engenharia de software 🛠️

✅ Escaneamento de Segurança

- Identificação de vulnerabilidades e sugestões para melhoria do código 🛡️

📊 Manipulação de Dados com ETL [🔗](#)

- O Amazon Q Developer gera scripts ETL para o AWS Glue automaticamente ⚡
 - Suporta diversas fontes de dados, como:
 - Amazon S3 📁
 - Amazon DynamoDB 🔴
 - Amazon RDS 🏛️
 - Amazon Redshift 🚀
-

💻 Integração com IDEs [🔗](#)

- O plugin do Amazon Q Developer para IDEs melhora a experiência de desenvolvimento ao:
 - Conectar-se a ferramentas e serviços essenciais 🔗
 - Automatizar processos, tornando o fluxo de trabalho mais eficiente ✅
-

Plugin para Terminal [🔗](#)

- ♦ O plugin de terminal do Amazon Q Developer transforma o uso do CLI ao:
 - Suportar autocompletar estilo IDE para comandos populares como git, npm, docker e AWS 🚀
 - Interpretar comandos em linguagem natural
 - Gerar códigos Shell prontos para execução ⚡
-

Customização e Repositórios Privados [🔗](#)

- ♦ O Amazon Q Developer pode ser customizado para trabalhar com repositórios privados 🗝️
 - ♦ Isso permite recomendações mais precisas baseadas nos padrões e bibliotecas internas da sua organização 🎯
-

Chat com Amazon Q em Slack e Teams [🔗](#)

- ♦ O Amazon Q pode responder perguntas no Slack ou Microsoft Teams 💡
 - ♦ Você pode perguntar sobre:
 - Recursos da AWS na sua conta ☁️
 - Comparação entre serviços ⚖️
 - Melhores práticas e muito mais 🚀
 - ♦ Ele responderá com guias passo a passo e links diretos para a documentação da AWS 📖
-

Casos de Uso do Amazon Q Developer [🔗](#)

- ♦ O Amazon Q Developer otimiza o desenvolvimento de software em diversas etapas, desde a planeamento até a modernização do código.
-

1. Planejamento (Plan) [🔗](#)

- ♦ A leitura de documentação técnica pode ser demorada e complexa.
 - ♦ O Amazon Q Developer facilita essa etapa com:
 - Guias específicos para o negócio 📊
 - Explicações detalhadas de código 💡
 - Codificação conversacional para ajudar no planejamento
-

2. Criação (Create) [🔗](#)






- ♦ Como assistente de codificação inline, acelera o desenvolvimento e manutenção de software ao:
 - Integrar-se a IDEs e ao AWS CLI ⚡
 - Sugerir códigos e auxiliar na implementação de novas funcionalidades
 - Facilitar iterações rápidas para ajustes e melhorias ↻️
-

3. Testes e Segurança (Test & Secure) [🔗](#)





- ♦ Garantir um código robusto e seguro é essencial! 💪
- ♦ O Amazon Q Developer auxilia ao:
- Gerar testes unitários automaticamente 🖋️
- Identificar vulnerabilidades de segurança 🚨

- Oferecer sugestões para correção com scans de segurança integrados
-

4. Operação (Operate) [🔗](#)

- Para desenvolvedores AWS, o Amazon Q Developer é essencial na solução de problemas e otimização de serviços como:
 - Amazon S3 
 - AWS Lambda 
 - Amazon EC2 
 - Amazon EKS 
 - Integrado diretamente na IDE, terminal ou console, ajuda a manter operações eficientes e resolver problemas rapidamente 
-

5. Manutenção e Modernização (Maintain & Modernize) [🔗](#)

- Com o Amazon Q Code Transformation, a atualização do código fica mais simples! 
 - Ele auxilia na:
 - Modernização de código 
 - Atualização de dependências para versões mais recentes 
 - Minimização de erros e redução da complexidade do processo 
-

Conectando o Amazon Q ao VS Code [🔗](#)

1. Instalação da Extensão Amazon Q [🔗](#)

- 1 Abra o VS Code e clique no ícone de Extensões na barra de atividades.
 - 2 Pesquise por "Amazon Q".
 - 3 Selecione a extensão Amazon Q nos resultados para obter mais informações.
 - 4 Clique em "Instalar".
 - 5 Após a instalação, o ícone do Amazon Q aparecerá na barra de atividades.
-

2. Autenticação no Amazon Q [🔗](#)

- 1 Clique no ícone do Amazon Q no VS Code.
 - 2 No painel Amazon Q: Login, escolha "Use For Free" e clique em "Continue".
 - 3 Um código de confirmação será exibido em um pop-up. Clique em "Proceed To Browser".
 - 4 Na janela que se abrirá, confirme a autorização clicando em "Open".
-

3. Criando um AWS Builder ID [🔗](#)

- 1 No navegador, a tela de Autorização solicitada será exibida. O código do VS Code já estará preenchido.
- 2 Clique em "Confirm and continue".
- 3 Caso já tenha um AWS Builder ID, clique em "Sign In".
- 4 Se precisar criar um, insira seu e-mail e clique em "Next".
- 5 Insira seu nome e clique em "Next".
- 6 Um código de verificação será enviado para seu e-mail. Insira-o e clique em "Verify".
- 7 Crie e confirme uma senha para seu AWS Builder ID e clique em "Create AWS Builder ID".
- 8 Na tela de permissão de acesso, clique em "Allow access".
- 9 Uma mensagem de confirmação será exibida. Você pode fechar o navegador.


✓ 4. Conexão Concluída [↗](#)

Interagindo com o Amazon Q no VS Code [↗](#)

♦ 1. Acessando o Amazon Q Chat [↗](#)

- 1 Verifique se a aba do Amazon Q Chat está aberta no Visual Studio Code.
- 2 Caso não esteja, clique no ícone do Amazon Q na barra de atividades.
- 3 Para aumentar o tamanho do painel do Chat, arraste a linha divisória entre o editor e o Chat.

2. Fazendo Perguntas no Chat [↗](#)

- 1 No campo de entrada do Chat, digite:
 - ♦ *What is the CDK?*
 - 2 Pressione Enter ou clique no ícone do avião de papel para enviar.
 - 3 O Amazon Q responderá com uma explicação e um link para a fonte.
-  *Dica:* Para iniciar uma nova conversa, digite /clear e pressione Enter.

3. Explorando e Analisando Código no VS Code [↗](#)

Enviando Código para Análise [↗](#)

- 1 Selecione o código que deseja enviar para análise.
- 2 Clique com o botão direito e escolha Send to Amazon Q → Send to prompt.
- 3 No Chat, faça perguntas sobre o código.

4. Verificando Vulnerabilidades e Melhorando Segurança [↗](#)

- 1 É possível perguntar ao Amazon Q se há falhas de segurança no código.
- 2 O Amazon Q retornará possíveis problemas e estratégias de mitigação.
- 3 Nesta etapa você pode solicitar uma modificação para adicionar as estratégias sugeridas.
- 4 O Amazon Q responderá com um código atualizado e um resumo das mudanças.

Aplicando as Modificações no Código [↗](#)

- 1 Exclua o trecho do código anterior.
- 2 No Chat, clique em Insert at cursor para colar o código corrigido.
- 3 Corrija a formatação, se necessário.
- 4 Salve o arquivo.

5. Otimizando Código [↗](#)

Refatorando [↗](#)

- 1 Selecione o trecho a ser refatorado com o botão direito e selecione Send to Amazon Q → Optimize.
- 2 O Amazon Q sugerirá otimizações, incluindo exemplos de código.

Aplicando Melhorias [↗](#)

- 1 No Chat, clique em Copy, depois cole o código melhorado.
- 2 Ajuste a importação, se necessário.

3 Salve o arquivo.

Transformando Código de Java 8 para Java 17 com Amazon Q Developer [↗](#)

1. Abra o Projeto [↗](#)

2. Localize o Código Fonte [↗](#)

3. Inicie a Transformação [↗](#)

- 1 Escolha a extensão Amazon Q Developer.
- 2 No **Chat**, digite `/t` e selecione a opção `/transform`.
- 3 Pressione Enter ou clique no ícone para enviar o comando.
- 4 Confirme a transformação

4. Monitore a Transformação [↗](#)

- 1 Se necessário, forneça no Chat o caminho para a instalação do **JDK 8**.
 - 2 A transformação começará com uma análise do projeto e atualização das dependências.
 - 3 O Transformation Hub será aberto automaticamente.
- Caso contrário, acesse pelo ícone de reticências (`...`) no painel do terminal.

5. Baixando e Revisando as Alterações [↗](#)

- 1 No **PROPOSED CHANGES**, veja a lista de arquivos modificados:

A (Added) → Arquivo **adicionado**

M (Modified) → Arquivo **modificado**

2 Compare as versões

- 3 Após revisar, clique em **Accept** para aplicar todas as mudanças.

6. Instalando o Código Atualizado [↗](#)

- 1 No terminal do VS Code, execute:

```
1 mvn clean install
```

- 2 Após alguns segundos, a mensagem **BUILD SUCCESS** confirmará a conclusão da transformação.

7. Verificando e Enviando as Alterações [↗](#)

- 1 Verifique os arquivos modificados para garantir que a migração foi bem-sucedida.
 - 2 Commit as mudanças no repositório de código-fonte.
 - 3 Prossiga com o deployment do código atualizado em Java 17.
-

Implementando uma Nova Feature com Amazon Q Developer [↗](#)

1. Abra o Projeto [↗](#)

2. Acesse a Extensão Amazon Q Developer [↗](#)

- 1 No VS Code, vá até a Activity Bar e escolha a extensão Amazon Q Developer.
- 2 No Chat, digite `/d` e selecione a opção `/dev`.

3. Crie uma nova feature e envie a solicitação [↗](#)

4. Analise o Projeto e Gerando Código [↗](#)

- 1 Amazon Q Developer analisará seu projeto e criará um plano de implementação.
- 2 Após a análise, um plano detalhado e passo a passo será exibido no Chat.
- 3 Revise as etapas e, ao final, escolha Generate code.

5. Revisando e Aplicando as Modificações [↗](#)

- 1 Amazon Q Developer gerará o código e modificará os arquivos necessários.
- 2 A árvore de diretórios será exibida com os arquivos modificados ou adicionados.
- 3 Escolha pom.xml para revisar as mudanças no projeto.

Linhas originais têm um sinal de menos (-).

Linhas modificadas têm um sinal de mais (+).

6. Aplique as Mudanças ao Código [↗](#)

- 1 Role até o final do Chat.
- 2 Escolha Insert code para aplicar as alterações.
- 3 Uma mensagem confirmará que a atualização foi concluída.
- 4 Escolha Close session para finalizar a sessão do Amazon Q Developer.

7. Teste a Nova Feature [↗](#)

- 1 No VS Code, vá até o menu TERMINAL e escolha New Terminal.
- 2 Execute o comando abaixo para garantir que as mudanças não afetaram o build:

```
1 mvn clean install
```

- 3 Se a mensagem BUILD SUCCESS aparecer, o código está pronto para commit e deploy.






Amazon Bedrock [↗](#)

Amazon Bedrock: NLP para Data Scientists [↗](#)

O Amazon Bedrock oferece diversas capacidades de Processamento de Linguagem Natural (NLP) para auxiliar cientistas de dados em suas tarefas.

Amazon Bedrock Agents: Automação Inteligente [↗](#)

Os Amazon Bedrock Agents são capazes de:

-  Entender solicitações em linguagem natural.
-  Dividir tarefas complexas em chamadas de API e consultas a dados.
-  Manter o contexto da conversa para interações naturais.
-  Executar ações automaticamente para atender às solicitações dos usuários.
-  Gerenciar infraestrutura, segurança e permissões sem necessidade de código personalizado.

Amazon Bedrock: Modelos Fundamentais (FMs) de IA [↗](#)

O Amazon Bedrock oferece uma ampla seleção de modelos fundamentais (FMs) de alto desempenho, desenvolvidos pela Amazon e startups líderes de IA. Esses modelos são voltados para diversas aplicações de IA generativa, como:

- ✓ Geração de texto
- ✓ Tradução de idiomas
- ✓ Criação de código
- ✓ Geração de imagens

◆ Modelos Disponíveis no Amazon Bedrock [🔗](#)

💡 Exemplos de Modelos:

- 🔥 Amazon Titan
- 🔥 Jurassic-2 (Jumbo, Mid e Ultra)
- 🔥 Claude 3
- 🔥 Command e Embed
- 🔥 Llama
- 🔥 Mistral
- 🔥 Stable Diffusion

🚀 Amazon Titan [🔗](#)

- 1 Amazon Titan Foundation Models → Modelos de base para IA generativa.
- 2 Amazon Titan Text → Especializado em geração de texto.
- 3 Amazon Titan Text Embeddings → Criação de embeddings para NLP avançado.
- 4 Amazon Titan Multimodal Embeddings → Integração de texto e imagens.
- 5 Amazon Titan Image Generator → Geração de imagens com IA.

🤖 AI21 Jurassic-2 (Mid e Ultra) [🔗](#)

Os modelos Jurassic-2 oferecem personalização avançada para geração de texto e compreensão de linguagem natural.

◆ Principais Configurações: [🔗](#)

- ◆ Tamanho do Texto → Definir o número máximo de tokens gerados.
- ◆ Stop Sequences → Parar a geração de texto em palavras-chave.

◆ Controle de Repetições: [🔗](#)

- 🔴 Presence Penalty → Evita repetição de palavras do prompt.
- 🔴 Count Penalty → Penaliza palavras já mencionadas.
- 🔴 Frequency Penalty → Penaliza palavras mais frequentes no texto.
- 🔴 Penalização de Tokens Especiais → Reduz a repetição de espaços, pontuações, números, stopwords e emojis.
- ◆ 🌐 Jurassic-2 Mid → Modelo médio, ideal para resumos e respostas a perguntas sem necessidade de exemplos.
- ◆ 💡 Jurassic-2 Ultra → Modelo grande, otimizado para copywriting, assistentes de IA e extração de informações.

📖 Jamba-Instruct [🔗](#)

O Jamba-Instruct é um modelo avançado que oferece:

- ✓ Janela de contexto de 256K tokens.
 - ✓ Suporte para geração de texto, resumo e respostas a perguntas.
 - ✓ Foco em aplicações empresariais de IA.
-

Anthropic Claude (Claude 2 e Claude 3) [↗](#)

O Claude é um modelo de IA generativa desenvolvido pela Anthropic, ideal para:

- ✓ Conversações naturais
- ✓ Geração de conteúdo criativo
- ✓ Resumos automáticos
- ✓ Resolução de perguntas
- ✓ Automação de fluxos de trabalho
- ✓ Geração de código

APIs disponíveis no Amazon Bedrock:

- ◆ Claude Text Completions API → Para geração de texto única (exemplo: criação de artigos ou resumos).
- ◆ Claude Messages API → Para aplicativos conversacionais (exemplo: assistentes virtuais).

Parâmetros Exclusivos do Claude [↗](#)

- 📌 max_tokens_to_sample → Define o número máximo de tokens gerados.
- 📌 Temperature, Top P, Top K, Stop Sequences → Parâmetros comuns para ajuste fino das respostas.

Claude Messages API - Assistentes de IA [↗](#)

A Messages API gerencia interações entre o usuário e o modelo Claude. Cada mensagem precisa ter um "role" (usuário ou assistente) e um conteúdo.

- 📌 Exemplo de uso: Criar assistentes virtuais e aplicativos de coaching.

Claude Converse API [↗](#)

A Converse API fornece uma interface unificada para modelos baseados em mensagens.

Stability AI - Geração de Imagens (SDXL) [↗](#)

O Stability AI SDXL é um modelo de text-to-image, usado para criar imagens detalhadas a partir de descrições textuais.

Modos de Criação de Imagens [↗](#)

- ◆ Image-to-image prompting → Gera variações a partir de uma imagem existente.
- ◆ Inpainting → Reconstrói partes ausentes de uma imagem.
- ◆ Outpainting → Expande uma imagem além dos seus limites originais.

Controles do Stability AI Diffusion [↗](#)

- 📌 Prompt strength (cfg_scale) → Ajusta a fidelidade da imagem ao prompt.
- 📌 Generation step (steps) → Define a quantidade de amostragens para refinar a imagem.
- 📌 Seed (seed) → Permite reproduzir imagens idênticas com a mesma configuração.

Cohere Command no Amazon Bedrock [↗](#)

O Cohere Command é um modelo de geração de texto otimizado para atender comandos do usuário e oferecer suporte imediato a aplicações empresariais, incluindo:

- ✓ Resumos automáticos
- ✓ Criação de conteúdo (copywriting)
- ✓ Diálogos e interações naturais

- ✓ Extração de informações
- ✓ Resolução de perguntas (Q&A)

🔒 **Segurança:** O Cohere é SOC 2 compliant (é um relatório de auditoria que atesta a eficácia dos controles de uma organização de serviços relacionados à segurança, disponibilidade, integridade de processamento, confidencialidade e privacidade).

⚙️ Parâmetros Exclusivos do Cohere Command [↗](#)

📌 Return likelihoods (return_likelihoods) → Define como os tokens de probabilidade são retornados:

- 1 GENERATION → Retorna apenas os tokens gerados.
- 2 ALL → Retorna tokens gerados + tokens de entrada.
- 3 NONE → Não retorna probabilidade dos tokens (padrão).

📌 Stream (stream) → Define se a resposta será retornada em tempo real (true) ou após a conclusão (false).

📌 Parâmetros padrão → Temperature, Top P, Top K, Max Length e Stop Sequences.

🔒 Segurança de Dados no Amazon Bedrock [↗](#)

O Amazon Bedrock garante proteção total dos dados com os seguintes recursos:

- ◆ Privacidade → Seus dados não são usados para melhorar o serviço ou compartilhados com terceiros.
- ◆ AWS PrivateLink → Conexão privada entre os modelos e sua VPC (Virtual Private Cloud), sem expor tráfego à internet.
- ◆ Criptografia → Dados protegidos em trânsito e em repouso.
- ◆ Treinamento privado → Personalização de modelos sem compartilhar dados sensíveis.
- ◆ Monitoramento e auditoria → Amazon CloudWatch: Métricas personalizadas para auditoria.
- 🔍 AWS CloudTrail → Rastreia atividades da API para integração segura de sistemas.

📌 Enterprise Datasets no Amazon Bedrock [↗](#)

Os **modelos fundacionais (FMs)** são treinados com dados públicos, mas, para aplicações empresariais personalizadas, é necessário fornecer **dados internos** (documentos, manuais, relatórios, etc.) como **contexto** no prompt.

⚠️ Cuidado!

- ◆ **Muito contexto** → Pode gerar **alucinações** no modelo, aumentar latência e custo.
 - ◆ **Pouco contexto** → Pode resultar em respostas genéricas ou imprecisas.
-

📊 Como fornecer contexto de forma eficiente? [↗](#)

- 1 **Contexto direto** → Inserir um documento específico no prompt.
- 2 **Busca inteligente** → Utilizar **vetores de embeddings** para encontrar os dados mais relevantes com base na consulta.

📌 **Embeddings** são representações numéricas que permitem buscas mais **precisas e eficientes** em grandes bases de dados empresariais.

◆ Benefícios

- ✓ Respostas mais precisas
- ✓ Menor latência e custo
- ✓ Redução de alucinações

O uso estratégico de **enterprise datasets** no Amazon Bedrock melhora a personalização e eficiência da IA para aplicações empresariais. 🚀

📌 O que são Vector Embeddings? [↗](#)

Os **vector embeddings** transformam textos, imagens e áudios em **representações numéricas** dentro de um espaço vetorial. Isso permite que informações **semelhantes semanticamente** fiquem próximas, facilitando buscas eficientes em grandes volumes de dados.

♦ Como funciona?

- 1 Dados empresariais (textos, imagens, áudios) são convertidos em **tokens** e vetorizados.
- 2 Esses vetores, com metadados, são armazenados em um **banco de dados vetorial** para rápida recuperação.
- 3 **Consulta eficiente:** O modelo compara a similaridade dos vetores com o prompt do usuário.

♦ Modelos de Embeddings no Amazon Bedrock

- ✓ **Amazon Titan Embeddings G1** → Texto para embeddings.
- ✓ **Cohere Embed** → Suporte a múltiplas línguas.
- ✓ **Titan Multi-modal Embedding** → Suporte a texto e imagens.

✚ Benefícios:

- ✓ Melhora a recuperação de informações.
- ✓ Reduz latência e custo em buscas complexas.
- ✓ Garante respostas mais relevantes para aplicações empresariais.

✚ O que são Vector Databases? [↗](#)

Os **vector databases** armazenam bilhões de vetores de alta dimensão e permitem **buscas eficientes por similaridade**, usando algoritmos como **k-NN** e **cosseno de similaridade**.

♦ Opções da AWS para Vector Databases:

- ✓ **Amazon OpenSearch Service** (provisionado)
- ✓ **Amazon OpenSearch Serverless**
- ✓ **pgvector no Amazon RDS para PostgreSQL**
- ✓ **pgvector no Amazon Aurora PostgreSQL**

♦ Como funciona?

- 1 **Vectorização** → Dados empresariais são convertidos em vetores.
- 2 **Busca vetorial** → O banco de dados encontra os vetores mais semelhantes ao prompt do usuário.
- 3 **Contexto para IA** → Os resultados são enviados para o modelo de IA, melhorando a qualidade da resposta.

♦ Vantagens:

- ✓ Melhora a precisão das respostas.
- ✓ Reduz **alucinações** em LLMs.
- ✓ Suporte ao método **RAG (Retrieval-Augmented Generation)**.

✚ O que é RAG (Retrieval-Augmented Generation)? [↗](#)

RAG é uma técnica que **combina modelos generativos de IA com bancos de dados vetoriais** para aprimorar respostas com **dados atualizados** e relevantes.

♦ Como funciona?

- 1 O **usuário envia um prompt** para o sistema.
- 2 Um **módulo de recuperação (retriever)** busca informações em bancos de dados empresariais/vetoriais.
- 3 O sistema **expande o prompt** com os dados recuperados.
- 4 O **modelo de linguagem gera uma resposta** enriquecida com o conhecimento corporativo.

- ✓ **Vantagens:** Permite que modelos usem informações **atuais e reais**.

- ✗ **Limitação:** Depende dos dados armazenados no momento da consulta.

✚ Model Fine-Tuning: Melhorando Modelos [↗](#)

O **fine-tuning** ajusta o modelo base para melhorar o desempenho em tarefas específicas. Diferente do RAG, o fine-tuning **muda o modelo permanentemente**.

- ♦ **Tipos de Fine-Tuning:**
- ♦ **1. Customização (Prompt-Based Learning):** Ajuste leve, sem modificar pesos do modelo.
- ♦ **2. Pré-treinamento Continuado (Domain Adaptation):** Re-treino do modelo em um novo domínio de conhecimento.

Qual usar?

- **RAG** = Para respostas temporariamente enriquecidas com dados externos.
- **Fine-Tuning** = Para aprimorar permanentemente o desempenho do modelo em tarefas específicas.

Personalização de Modelos (Fine-Tuning) [↗](#)

Existem **dois métodos principais** para ajustar modelos de IA para aplicações específicas:

1 Customização (Prompt-Based Learning) [↗](#)

- ✓ Técnica **mais leve** e rápida.
- ✓ **Treinamento em poucas épocas** (iterations).
- ✓ Usa **pares de prompt e resposta** como exemplos rotulados.
- ✓ Focado em **tarefas específicas, não se generaliza bem**.

Exemplo:

- Treinar um modelo para responder **perguntas jurídicas** usando um conjunto de perguntas e respostas formatadas corretamente.

2 Pré-Treinamento Continuado (Domain Adaptation) [↗](#)

- ✓ Ajusta o modelo com **dados do domínio específico**.
- ✓ Permite **uso em múltiplas tarefas** dentro do domínio.
- ✓ O modelo aprende **vocabulário técnico e jargões**.
- ✓ Requer **infraestrutura de ML e capacidade computacional** para treinar.

Exemplo:

- Ajustar um modelo para entender **termos médicos** usando grandes volumes de registros médicos.

Quando usar?

- ✓ **Prompt-Based Learning** → Quando precisa de ajustes rápidos e específicos.
- ✓ **Domain Adaptation** → Quando precisa de um modelo robusto para **diferentes tarefas dentro de um domínio especializado**.

Boas Práticas para Segurança em Aplicações de IA Generativa [↗](#)

- 1** Controle de Acesso
- 2** Monitoramento e Logs
- 3** Conformidade e Transparência
- 4** Testes de Segurança
- 5** Documentação Atualizada

Padrão de Arquitetura para Geração de Texto e Código com Amazon Bedrock [↗](#)

Casos de Uso

- ✓ Escrita de artigos, blogs, emails, livros 

- ✓ Geração de código (SQL, Python, Java) 🖥️
- ✓ Explicação, otimização e correção de código 🔧

📌 Fluxo da Arquitetura

- 1 O usuário envia um **prompt** via **Amazon Bedrock Playground** ou **API**.
- 2 O modelo de IA processa o input e **gera um texto ou código** como resposta.
- 3 O resultado é retornado ao usuário.

📌 Benefícios

- 🚀 **Agilidade** na criação de conteúdo e prototipação de ideias.
- 🔍 **Auxílio para desenvolvedores** na geração e otimização de código.

🖥️ Padrões de Arquitetura para Geração e Resumo de Texto com Amazon Bedrock 🔗

📌 Geração de Texto com LangChain 🔗

- ✓ LangChain atua como uma **camada de orquestração** entre o usuário e os modelos da Amazon Bedrock.
- ✓ Suporta **text generation** e **AI Assistants**.
- ✓ Fluxo:
 - 1 O usuário envia um **prompt** para o LangChain.
 - 2 LangChain gerencia a requisição para o modelo da Amazon Bedrock.
 - 3 O modelo gera o **output**, retornado ao usuário.

-
- ♦ **Para documentos pequenos:** Enviar diretamente para o modelo da Amazon Bedrock.
 - ♦ **Para documentos grandes:** Aplicar **Map-Reduce** e técnicas de **chunking** e **chaining prompts**.
-

Fluxo para documentos grandes:

- 1 **Dividir** o documento em múltiplos **chunks** menores.
- 2 Enviar cada **chunk** para o modelo e gerar um **resumo parcial**.
- 3 Concatenar o próximo chunk ao resumo gerado e **resumir novamente**.
- 4 **Iterar** até obter um resumo final.

Padrão de Arquitetura para Assistente de IA 🔗

Os **assistentes de IA** utilizam **NLP** e **ML** para processar consultas e fornecer respostas eficientes em diversas aplicações, como **atendimento ao cliente, vendas e e-commerce**.

♦ Fluxo de Funcionamento 🔗

- 1 O usuário faz uma consulta ao assistente.
- 2 O histórico do chat (se houver) e a nova consulta são enviados para o **modelo FM da Amazon Bedrock**.
- 3 O modelo **gera uma resposta** considerando o contexto.
- 4 A resposta é retornada ao usuário.

📌 Tipos de Assistentes de IA 🔗

- ♦ **Assistente Básico:** Usa um **modelo FM padrão**, sem informações adicionais (**zero-shot**).
- ♦ **Assistente com Template de Prompt:** Inclui um **contexto pré-definido** para refinar respostas.
- ♦ **Assistente com Persona:** Assume um **papel específico** (ex: consultor de investimentos, professor virtual).
- ♦ **Assistente com Consciência Contextual:** Usa **embeddings** para buscar informações externas relevantes e gerar respostas mais assertivas.

- 🚀 **Benefícios:** Melhor personalização, maior precisão e experiência aprimorada para o usuário.

♦ Desafios de Desempenho dos LLMs [↗](#)

Os LLMs são treinados em grandes volumes de dados e podem executar tarefas como **geração de texto, sumarização, perguntas e respostas**. No entanto, apresentam desafios como:

- ✓ **Dados fora do domínio treinado** → Podem gerar respostas imprecisas (**alucinações**).
- ✓ **Falta de memória contextual** → Cada chamada ao modelo é independente, dificultando conversas coerentes.
- ✓ **Execução em múltiplas etapas** → Algumas respostas exigem **encadeamento de prompts** para melhorar a precisão.

🚀 Simplificando o Desenvolvimento com LangChain [↗](#)

LangChain é um **framework para aplicações LLM** que reduz a complexidade do desenvolvimento ao fornecer **blocos de construção** para lidar com desafios comuns.

- ♦ **Gerenciamento de contexto** → Permite reter e utilizar o histórico da conversa.
- ♦ **Encadeamento de raciocínio** → Executa **tarefas multietapas** de forma estruturada.
- ♦ **Facilidade de produção e escalabilidade** → Otimiza a integração de **LLMs** em aplicações empresariais.

💡 Com LangChain, é possível estruturar fluxos eficientes e melhorar a precisão das respostas, reduzindo falhas e custos operacionais.

🔧 Componentes do LangChain [↗](#)

O LangChain está disponível em **Python, TypeScript e JavaScript** e oferece componentes essenciais para aplicações com LLMs:

- ♦ **Modelos** → Integração com diferentes FMs.
- ♦ **Templates de Prompt** → Estruturam prompts eficazes.
- ♦ **Índices** → Facilitam a busca de informações.
- ♦ **Memória** → Mantém o contexto da conversa.
- ♦ **Encadeamentos (Chains)** → Automação de fluxos complexos.
- ♦ **Agentes** → Permitem decisões baseadas em múltiplas interações.

💡 Esses componentes são fundamentais para criar chatbots RAG, sumarização de textos, geração de código, análise de consultas e mais.

🔗 Integração com AWS e Amazon Bedrock [↗](#)

O LangChain oferece integração com **Amazon Bedrock** via **langchain.aws** e **langchain-community**, permitindo acesso a diversos modelos:

- ✓ **Amazon Titan (Texto)**
- ✓ **AI21 Jurassic**
- ✓ **Anthropic Claude**
- ✓ **Cohere Command & Embed**
- ✓ **Meta Llama**
- ✓ **Stable Diffusion (Imagens)**
- ✓ **Mistral AI**

💡 Isso possibilita o uso de modelos avançados para diversas aplicações, combinando o poder do LangChain com a infraestrutura escalável da AWS.

📌 Trabalhando com Modelos no LangChain [↗](#)

O LangChain facilita a interação com **LLMs** e oferece suporte para modelos customizados, chatbots e embeddings.

🚀 Usando LLMs no LangChain [↗](#)

O LangChain fornece uma **classe LLM** para interagir com modelos de linguagem de diversos provedores.

Exemplo: Chamando o modelo **Amazon Titan** via **Amazon Bedrock**.

```
1 import boto3
2 from langchain_aws import BedrockLLM
3
4 bedrock_client = boto3.client('bedrock-runtime', region_name="us-east-1")
5 inference_modifiers = {"temperature": 0.3, "maxTokenCount": 512}
6
7 llm = BedrockLLM(
8     client=bedrock_client,
9     model_id="amazon.titan-tg1-large",
10    model_kwargs=inference_modifiers,
11    streaming=True,
12 )
13
14 response = llm.invoke("What is the largest city in Vermont?")
15 print(response)
16
```

Modelos Customizados [🔗](#)

O **Amazon Bedrock** permite **pré-treinamento contínuo** ou **fine-tuning** de modelos para casos específicos.

Exemplo: Chamando um **modelo customizado**.

```
1 import boto3
2 from langchain_aws import BedrockLLM
3
4 custom_llm = BedrockLLM(
5     credentials_profile_name="bedrock-admin",
6     provider="cohere",
7     model_id="<Custom model ARN>",
8     model_kwargs={"temperature": 1},
9     streaming=True,
10 )
11
12 response = custom_llm.invoke("What is the recipe for mayonnaise?")
13 print(response)
14
```

Modelos de Chat [🔗](#)

Chatbots otimizam o suporte ao cliente e melhoram a experiência do usuário.

Exemplo: Criando um **Chatbot com Anthropic Claude**.

```
1 from langchain_aws import ChatBedrock as Bedrock
2 from langchain.schema import HumanMessage
3
4 chat = Bedrock(model_id="anthropic.claude-3-sonnet-20240229-v1:0", model_kwargs={"temperature":0.1})
5
6 messages = [HumanMessage(content="I would like to try Indian food, what do you suggest?")]
7 chat.invoke(messages)
8
```

Modelos de Embeddings [↗](#)

Os embeddings transformam texto em **vetores numéricos**, úteis para **busca semântica, análise de sentimentos e classificação de texto**.

- ♦ Os vetores podem ser armazenados em **bancos de dados vetoriais** para consultas mais rápidas e precisas.

💡 O LangChain fornece suporte nativo para embeddings com Amazon Bedrock e outras plataformas! 🚀

Trabalhando com Embeddings e Templates no LangChain [↗](#)

Criando Embeddings no LangChain [↗](#)

Os embeddings transformam texto em **vetores numéricos**, permitindo busca semântica e comparação de significados entre frases e idiomas.

Exemplo: Criando um **embedding** com **Amazon Titan** via **Amazon Bedrock**.

```
1 from langchain_community.embeddings import BedrockEmbeddings
2
3 embeddings = BedrockEmbeddings(
4     region_name="us-east-1",
5     model_id="amazon.titan-embed-text-v1"
6 )
7
8 vector = embeddings.embed_query("Cooper is a puppy that likes to eat beef")
9 print(vector) # Retorna um vetor numérico representando o texto
10
```

Comparação de Similaridade [↗](#)

💡 O poder dos embeddings:

- **Troca de palavras similares:** “dog” → “puppy” mantém alta similaridade.
- **Mudança na estrutura da frase:** Exemplo, “Beef is what a dog named Cooper likes” ainda mantém significado.
- **Tradução entre idiomas:** Frases em diferentes idiomas com mesmo significado são mais próximas.

Construção de Prompts com LangChain [↗](#)

O LangChain oferece **templates de prompts** reutilizáveis, otimizando a **engenharia de prompts**.

- ♦ **Vantagens dos Prompt Templates:**

- ✓ Reduz a necessidade de escrever prompts do zero.
- ✓ Facilita a **personalização dinâmica** dos prompts.
- ✓ Melhora a **consistência** na interação com LLMs.

Estruturando Documentos com LangChain e AWS [↗](#)

Carregamento de Documentos (Document Loaders) [↗](#)

Ao construir aplicações **RAG (Retrieval-Augmented Generation)**, é essencial carregar documentos de **diferentes fontes**.

💡 **Componente:** O LangChain oferece **document loaders** para carregar documentos de:

- ✓ **Bancos de dados**

✅ Armazenamento local (arquivos HTML, PDF, etc.)

✅ Fontes online (APIs, S3, SharePoint, etc.)

🔍 Recuperação de Documentos (Retriever) [↗](#)

O **retriever** busca documentos relevantes no índice para combinar com **LLMs**.

♦ Como funciona?

- 1 O usuário envia uma **consulta**.
- 2 O **retriever** pesquisa no índice.
- 3 Retorna **documentos mais relevantes**.
- 4 O LLM processa e gera a resposta.

📌💡 AWS Amazon Kendra:

- Serviço gerenciado para **busca semântica**.
- Conecta-se a **Amazon S3, SharePoint, Confluence** etc.
- Suporta **HTML, PDF, Word, Excel, PureText**.

🔧 Exemplo prático com Amazon Kendra [↗](#)

```
1 from langchain.retrievers import AmazonKendraRetriever
2
3 retriever = AmazonKendraRetriever(index_id="meu-kendra-index-id")
4
5 query = "Quais são os benefícios da energia solar?"
6 documents = retriever.get_relevant_documents(query)
7
8 for doc in documents:
9     print(doc.page_content) # Exibe o conteúdo do documento relevante
10
```

💡 Resumo:

- ✅ Carregue documentos de múltiplas fontes 📄
- ✅ Recupere informação relevante dinamicamente 🔍
- ✅ Use Amazon Kendra para buscas semânticas poderosas ⚡

📌 Armazenamento e Recuperação com LangChain e AWS [↗](#)

🧠 Vector Stores: Guardando e Buscando Informações [↗](#)

Ao criar aplicações **RAG (Retrieval-Augmented Generation)**, é essencial armazenar dados para serem usados como **contexto** nos LLMs.

💡 Passos para um RAG eficiente:

- 1 Converter documentos em **embeddings** (representação numérica do texto).
- 2 Armazenar os embeddings em um **banco vetorial (vector store)**.
- 3 Consultar o vector store para recuperar **documentos relevantes**.
- 4 Enviar os documentos ao **LLM** para obter respostas mais precisas.

📌💡 AWS Suporte:

- **Amazon OpenSearch Serverless (vector engine)**
- **pgvector com Amazon Aurora PostgreSQL**
- Outras opções: Pinecone, Weaviate, FAISS, Chroma

🔍 Exemplo de uso de Vector Store com LangChain [↗](#)

```
1 from langchain.vectorstores import FAISS
2 from langchain_community.embeddings import BedrockEmbeddings
3
4 # Criar embeddings com Amazon Bedrock
5 embeddings = BedrockEmbeddings(model_id="amazon.titan-embed-text-v1")
6
7 # Carregar documentos para o vector store FAISS
8 vectorstore = FAISS.from_texts(["O plástico é reciclável", "Energia solar é renovável"], embeddings)
9
10 # Consultar documentos relevantes
11 query = "O que pode ser reciclado?"
12 docs = vectorstore.similarity_search(query)
13
14 # Exibir os resultados mais relevantes
15 for doc in docs:
16     print(doc.page_content)
17
```

📁 Memória em LangChain [↗](#)

LLMs **não armazenam contexto** entre interações. Para um chatbot, é fundamental **manter o histórico da conversa**.

💡 Tipos de Memória em LangChain:

- ♦ **ConversationBufferMemory** – Mantém o histórico completo do chat.
- ♦ **ConversationChain** – Gerencia interações passadas e contexto.

📌 Exemplo: Memória no Chatbot

```
1 from langchain.memory import ConversationBufferMemory
2 from langchain.chains import ConversationChain
3 from langchain_aws import ChatBedrock
4
5 memory = ConversationBufferMemory()
6 chatbot = ConversationChain(
7     llm=ChatBedrock(model_id="anthropic.claude-3-sonnet"),
8     memory=memory
9 )
10
11 # Simular uma conversa
12 print(chatbot.run("Olá, qual o seu nome?"))
13 print(chatbot.run("Qual era minha pergunta anterior?"))
14
```

💡 Resumo:

- ✅ **Armazene informações relevantes em vector stores** 📁
- ✅ **Use embeddings para buscas semânticas** 🔍
- ✅ **Mantenha o histórico do chatbot** para conversas naturais 🧠

🔗 Criando Cadeias (Chains) no LangChain [↗](#)

💡 O que são Chains?

Uma **Chain** é um conjunto de componentes que trabalham juntos para processar informações. Esses componentes podem ser:

- ♦ Chamadas para um LLM
- ♦ Requisições a APIs
- ♦ Sequências de outras chains

✅ Vantagens das Chains:

- 🔗 Processamento de **grandes volumes de dados** (ex.: sumarização de documentos longos).
- 🔗 **Encadeamento estruturado** de chamadas para melhorar a precisão.
- 🔗 **Automatização** de fluxos complexos de NLP.

🔗 Exemplo: Criando uma Chain com LangChain [🔗](#)

🚀 **LLMChain:** Usa um LLM + um Prompt Template para processar a entrada do usuário.

```
1 from langchain_aws import BedrockLLM
2 from langchain.chains import LLMChain
3 from langchain.prompts import PromptTemplate
4
5 # Criar o modelo LLM
6 llm = BedrockLLM(model_id="amazon.titan-tg1-large")
7
8 # Criar um Prompt Template
9 template = PromptTemplate(
10     input_variables=["cidade"],
11     template="Qual é a população de {cidade}?"
12 )
13
14 # Criar uma LLMChain
15 chain = LLMChain(llm=llm, prompt=template)
16
17 # Executar a chain
18 resposta = chain.run("São Paulo")
19 print(resposta)
20
```

🔗 Processando Grandes Volumes de Dados com Chains [🔗](#)

💡 Para processar **documentos extensos**, podemos dividir o texto em **chunks** e gerar resumos parciais.

🔗 Exemplo: Resumo de Texto Longo

```
1 from langchain.chains.summarize import load_summarize_chain
2 from langchain_aws import BedrockLLM
3
4 # Criar o modelo LLM
5 llm = BedrockLLM(model_id="amazon.titan-tg1-large")
6
7 # Criar uma chain de sumarização
8 summarize_chain = load_summarize_chain(llm, chain_type="map_reduce")
9
10 # Texto longo
11 documento = ["0 plástico é um dos materiais mais utilizados..."] * 10 # Simulando um texto grande
12
13 # Gerar resumo
14 resumo = summarize_chain.run(documento)
15 print(resumo)
16
```

🔗 Tipos de Chains no LangChain [↗](#)

- 1 **LCEL (LangChain Expression Language):** Forma recomendada para construir chains.
- 2 **Chains Legadas:** Usam a classe `Chain`, como `LLMChain`.

🚀 **Próximo passo:** Criar **workflows completos** para aplicações baseadas em LLMs! 🔥

🔍 Visão Geral do RAG (Retrieval-Augmented Generation) [↗](#)

💡 O que é RAG?

RAG (**Retrieval-Augmented Generation**) é uma técnica que melhora o desempenho dos **Modelos Fundamentais (FM)** ao integrar conhecimento externo.

🔴 Problemas dos FMs sem RAG:

- ❌ Podem apresentar **informações falsas ou desatualizadas**.
- ❌ Respostas **genéricas** quando os usuários precisam de algo **específico**.
- ❌ Baseiam-se em **dados não confiáveis**.
- ❌ Confusão de **terminologia**, gerando respostas imprecisas.

✅ Como o RAG resolve isso?

RAG **combina um FM com um repositório externo de conhecimento**. O modelo busca informações **autoridade e atualizadas**, melhorando a precisão das respostas.

🚀 Fluxo de Funcionamento do RAG [↗](#)

- 1 **O usuário faz uma pergunta**
- 2 **O sistema busca informações relevantes** em fontes autorizadas (banco de dados, documentos, APIs, etc.).
- 3 **O modelo LLM recebe os dados encontrados** e os usa como contexto para gerar uma resposta precisa.
- 4 **A resposta final é gerada e enviada ao usuário.**

🔧 Exemplo de Implementação de RAG com LangChain + Amazon Kendra [↗](#)

```
1 from langchain_aws import BedrockLLM
2 from langchain.retrievers import AmazonKendraRetriever
3 from langchain.chains import RetrievalQA
4
5 # Criar o modelo LLM
6 llm = BedrockLLM(model_id="amazon.titan-tg1-large")
7
8 # Criar o Retriever para buscar informações no Amazon Kendra
9 retriever = AmazonKendraRetriever(
10     index_id="meu-kendra-index",
11     region_name="us-east-1"
12 )
13
14 # Criar a RAG Chain para responder perguntas com busca em fontes externas
15 qa_chain = RetrievalQA(llm=llm, retriever=retriever)
16
17 # Fazer uma pergunta ao modelo
18 resposta = qa_chain.run("Qual é a política de devolução da minha empresa?")
19 print(resposta)
20
```

🔥 Benefícios do RAG [🔗](#)

- ✓ Respostas mais confiáveis 📖
- ✓ Informações sempre atualizadas 🌐
- ✓ Maior controle sobre a fonte dos dados 🔍

🚀 **Próximo passo:** Implementar **armazenamento vetorial** para otimizar a recuperação de informações! 💡

🔍 Casos de Uso do RAG [🔗](#)

🚀 1. Sistemas de Perguntas e Respostas [🔗](#)

- 🔗 **Aplicações:** Atendimento ao cliente, suporte técnico, assistentes de pesquisa.
- 🔗 **Benefício:** Respostas precisas e específicas, baseadas em um banco de conhecimento atualizado.

📝 2. Geração de Conteúdo Aprimorada [🔗](#)

- 🔗 **Aplicações:** Redação de artigos, relatórios, histórias criativas.
- 🔗 **Benefício:** Produção de conteúdos mais ricos, com dados verificáveis e bem embasados.

🤖 3. IA Conversacional Inteligente [🔗](#)

- 🔗 **Aplicações:** Chatbots, assistentes virtuais, interfaces de conversação.
- 🔗 **Benefício:** Respostas mais naturais, contexto aprimorado e maior assertividade.

📊 4. Análise de Dados e Relatórios [🔗](#)

- 🔗 **Aplicações:** Geração de relatórios detalhados, apresentações e insights baseados em dados.
- 🔗 **Benefício:** Integração de estatísticas relevantes para análises mais completas e embasadas.

✓ 5. Verificação de Fatos e Fact-Checking [🔗](#)

- 🔗 **Aplicações:** Jornalismo, compliance, auditoria.
- 🔗 **Benefício:** Respostas confiáveis baseadas em fontes verificadas e confiáveis.

🎯 6. Personalização de Conteúdo [🔗](#)

- 🔗 **Aplicações:** Descrições de produtos, marketing, recomendações personalizadas.
- 🔗 **Benefício:** Geração de conteúdos adaptados aos interesses e preferências dos usuários.

🔥 **Conclusão:** O RAG amplia a capacidade dos modelos de IA, garantindo informações mais precisas, atualizadas e relevantes para cada contexto! 🚀

🔧 Arquitetura RAG: Como Funciona? [🔗](#)

O **Retrieval-Augmented Generation (RAG)** aprimora a capacidade dos Modelos Fundamentais (**FMs**) ao integrar um mecanismo de recuperação de informações.

🔗 🔍 **Diferença Chave:** Em vez de gerar respostas apenas com base no conhecimento aprendido durante o treinamento, o **RAG busca informações externas**, combinando-as com a consulta do usuário antes de produzir uma resposta.

📁 Processo de Ingestão de Dados [🔗](#)

🔗 **Objetivo:** Incorporar dados externos (APIs, bancos de dados, repositórios de documentos) para expandir o conhecimento da IA.

🔗 **Passos do Processo:**

1 Coleta de Dados Externos:

- Fontes: Arquivos, registros de bancos de dados, textos extensos.

2 Segmentação de Dados (Chunking):

- Divide grandes documentos em partes menores para facilitar o processamento.

3 Conversão em Embeddings:


- Transforma o texto em representações numéricas (vetores) com um modelo de embeddings.

4 Armazenamento em Banco de Vetores:

- Permite a recuperação eficiente de informações relevantes com base no significado semântico.

Fluxo Completo do RAG [↗](#)


1 Recuperação de Informações (Retrieve) [↗](#)

 **Objetivo:** Encontrar os dados mais relevantes para a consulta do usuário.

 **Como Funciona?**

- A consulta do usuário é convertida em um **vetor numérico**.
- Uma **busca semântica** compara esse vetor com os armazenados no banco de vetores.
- O sistema recupera os documentos mais relevantes com base na similaridade.

2 Aprimoramento do Prompt (Augment) [↗](#)

 **Objetivo:** Enriquecer a entrada do modelo de IA com os dados recuperados.

 **Como?**

- Técnicas de **engenharia de prompts** são usadas para estruturar a consulta.
- O modelo recebe tanto a pergunta original quanto os documentos relevantes.

3 Geração da Resposta (Generate) [↗](#)

 **Objetivo:** Criar uma resposta precisa e informativa.

 **Fluxo Final**

 **Prompt Aprimorado** → **Modelo de IA** →  **Resposta Baseada em Dados Atualizados**

Desafios na Construção de Aplicações RAG [↗](#)

1 Atualização dos Dados Externos [↗](#)

 **Problema:** Os dados externos podem ficar desatualizados, reduzindo a precisão das respostas.

 **Solução:**

- Implementar **atualizações assíncronas** das fontes externas e embeddings.
 - Usar **atualizações em tempo real** para dados dinâmicos.
 - Adotar **processamento em lote periódico** (diário, semanal, mensal) para bases grandes.
-

2 Escalabilidade [🔗](#)

✓ **Problema:** Quanto maior a base de conhecimento, mais lento e custoso se torna o processo de recuperação.

🔧 **Solução:**

- Criar **índices eficientes** e aplicar **otimização de consultas**.
 - Usar **técnicas de busca semântica** e algoritmos de recuperação rápidos.
 - Implementar **cache** para reutilizar consultas frequentes.
-

3 Relevância e Precisão [🔗](#)

✓ **Problema:** Nem sempre os documentos recuperados são os mais relevantes.

🔧 **Solução:**

- Melhorar **métricas de similaridade** para recuperação mais precisa.
 - Implementar **ajuste fino** no modelo de embeddings.
 - Usar **feedback do usuário** para refinar os resultados ao longo do tempo.
-

4 Viés e Justiça [🔗](#)

✓ **Problema:** A base de conhecimento pode conter vieses, levando a respostas tendenciosas.

🔧 **Solução:**

- Auditar os dados de treinamento e recuperação.
 - Usar **técnicas de de-biasing** e curadoria de fontes confiáveis.
 - Implementar verificações de **justiça e imparcialidade** nos outputs.
-

5 Avaliação e Métricas [🔗](#)

✓ **Problema:** Métricas tradicionais nem sempre capturam a qualidade real das respostas.

🔧 **Solução:**

- Criar **métricas específicas para RAG**, combinando avaliação de recuperação e geração.
 - Incluir **avaliação manual e automática** para medir a precisão das respostas.
 - Utilizar **benchmarks específicos** para monitorar desempenho e melhorias.
-

Amazon Bedrock Knowledge Bases: Simplificando RAG na AWS [🔗](#)

Amazon Bedrock Knowledge Bases oferece uma solução **totalmente gerenciada** para implementar RAG sem precisar desenvolver integrações personalizadas e gerenciar fluxos de dados.

✓ **Principais Benefícios:**

- Integração automática com fontes de dados privadas.
 - Melhoria na precisão e relevância das respostas geradas.
 - Suporte nativo para **conversas multi-turno**.
-

Como Criar uma Knowledge Base no Amazon Bedrock [🔗](#)

1 Criar uma Nova Knowledge Base [🔗](#)

- ♦ Acesse o **AWS Management Console** e selecione **"Create knowledge base"**.

2 Definir Nome e Permissões [↗](#)

- ♦ Insira um **nome** e uma **descrição**.
- ♦ Configure um **runtime role** para conceder permissões ao Amazon Bedrock.

3 Ingerir Conteúdo [↗](#)

- ♦ Escolha a fonte de dados (exemplo: **Amazon S3, Confluence, Salesforce, SharePoint**).
- ♦ Configure os detalhes e **Bedrock buscará os documentos automaticamente**.

4 Escolher o Modelo de Embeddings [↗](#)

- ♦ O Bedrock Knowledge Bases **segmenta o texto e gera embeddings** automaticamente.

5 Armazenar os Embeddings em um Banco Vetorial [↗](#)

- ♦ Escolha entre:
 - ✓ **Amazon OpenSearch Serverless** (gerenciado pelo Bedrock).
 - ✓ **Bancos suportados:** Pinecone, Redis Cloud, Amazon Aurora, MongoDB.

[🔍](#) Personalizando Knowledge Bases para Respostas Mais Precisas [↗](#)

Amazon Bedrock Knowledge Bases permite **ajustar a recuperação e a ingestão** de dados para melhorar a precisão das respostas em diferentes cenários.

⚡ Como Personalizar a Knowledge Base? [↗](#)

1 Ajuste a Ingestão de Dados [↗](#)

- ♦ Use **opções avançadas de parsing** para interpretar **PDFs, imagens escaneadas** e conteúdos complexos (como tabelas).
- ♦ Utilize **opções de chunking** para segmentação eficiente dos documentos:
 - ✓ **Chunking customizado** → Defina sua própria lógica usando **AWS Lambda**.
 - ✓ **Chunking embutido** → Escolha entre **tamanho fixo, sem chunking, chunking hierárquico ou chunking semântico**.
- ♦ **Frameworks compatíveis:** LangChain e LlamaIndex.

2 Aprimore a Recuperação de Dados [↗](#)

- ♦ **Query Reformulation** → Reformula consultas complexas para melhorar a recuperação de informações relevantes.
- ♦ Use a **Retrieve API** para buscar documentos mais relevantes de sua Knowledge Base.
- ♦ Use a **RetrieveAndGenerate API** para recuperar informações e **gerar respostas otimizadas automaticamente**.

3 Integre Knowledge Bases com Bedrock Agents [↗](#)

- ♦ Forneça **contexto adicional** aos agentes para melhorar a precisão das respostas e gerar interações mais inteligentes.

[🚀](#) Com esses ajustes, você pode garantir que seu sistema RAG entregue respostas mais precisas e contextualizadas!

[🔍](#) Amazon Bedrock Knowledge Bases: Como Funciona? [↗](#)

Amazon Bedrock Knowledge Bases fornece **citações e atribuições de fontes** para melhorar a transparência e minimizar alucinações. Você pode visualizar os detalhes da fonte ao testar sua Knowledge Base.

⚡ Como Amazon Bedrock Knowledge Bases Funciona? [↗](#)

1 Pré-processamento de Dados [↗](#)

- ♦ **Divisão de documentos** → Os documentos são segmentados em **chunks** para otimizar a recuperação.
- ♦ **Geração de embeddings** → Cada chunk é convertido em um vetor e armazenado em um **índice vetorial**.
- ♦ **Mapeamento de fontes** → O índice vetorial mantém uma relação com o documento original para rastreamento da fonte.

2 Processamento em Tempo de Execução (Runtime) [↗](#)

- ♦ A **consulta do usuário** é convertida em um **vetor** pelo mesmo modelo de embeddings usado na Knowledge Base.
- ♦ O **índice vetorial** compara o vetor da consulta com os vetores armazenados para encontrar os trechos mais relevantes.
- ♦ A **consulta do usuário é enriquecida** com o contexto dos trechos recuperados.
- ♦ O **modelo de linguagem** gera uma resposta utilizando a consulta aprimorada.

3 Otimização do RAG [↗](#)

- ♦ **Expandir e refinar seus dados** para uma melhor recuperação de informações.
- ♦ **Implemente estratégias avançadas de chunking** para melhor segmentação de textos.
- ♦ **Otimizar embeddings e VectorDB** para maior precisão na busca de dados.
- ♦ **Explore técnicas avançadas de recuperação** para maximizar o desempenho do sistema.

 Com essas estratégias, o RAG se torna mais eficaz, garantindo respostas mais precisas e contextuais!

Ingestão de Dados em Knowledge Bases [↗](#)

Para que o RAG recupere informações de maneira eficaz, os dados precisam ser:

- 1 **Convertidos para texto**
- 2 **Divididos em partes gerenciáveis (chunks)**
- 3 **Transformados em vetores** (embedding vectors)
- 4 **Armazenados em um banco vetorial** para busca semântica

- ♦ **Fatores essenciais na recuperação eficiente de dados RAG:** [↗](#)





- ✓ **Tamanho do chunk**
- ✓ **Modelo de embeddings**
- ✓ **Tipo de banco vetorial**

Como Escolher o Tamanho Ideal dos Chunks? [↗](#)

O tamanho do chunk afeta diretamente a **eficiência da busca e a preservação do contexto**.

- ♦ **Chunks pequenos** → Processamento mais rápido, mas podem perder contexto
- ♦ **Chunks grandes** → Maior contexto, mas podem ser custosos e conter informações irrelevantes

? Perguntas para definir o chunk ideal: [↗](#)

-  Qual é a natureza do conteúdo?
-  Qual modelo de embeddings será usado? Ele tem um tamanho ideal de chunk?
-  Qual a complexidade das consultas esperadas?
-  Como os resultados recuperados serão usados na aplicação?

Estratégias de Chunking [↗](#)

1 Chunking de Tamanho Fixo [↗](#)

📌 Divide o texto em partes fixas (palavras ou caracteres)

- ✓ Simples e rápido
- ✗ Pode cortar frases ou comprometer o contexto

2 Chunking Baseado no Conteúdo [🔗](#)

📌 Adapta-se ao tipo de conteúdo para preservar o contexto

- **Sentence-level** → Quebra no final das frases
 - **Recursive chunking** → Divide o texto hierarquicamente
 - **Semantic chunking** → Identifica limites semânticos e divide por tópicos
- ✓ Mantém coerência
 - ✗ Pode exigir mais poder computacional

3 Estratégia Híbrida [🔗](#)

📌 Combina diferentes abordagens

- ✓ Exemplo: Usar **sentence-level chunking** e depois agrupar ou dividir conforme o contexto
- ♦ A escolha ideal depende da aplicação e do comportamento esperado do RAG!

📌 Como Escolher um Modelo de Embeddings? [🔗](#)

Os **modelos de embeddings** convertem textos em vetores numéricos, permitindo buscas semânticas eficientes. A escolha do modelo influencia diretamente o desempenho da aplicação RAG.

♦ Principais Fatores na Escolha do Modelo [🔗](#)

1 Dimensionalidade dos Embeddings [🔗](#)

📌 Quanto maior a dimensão, maior o detalhamento semântico, mas maior o custo computacional.

- ✓ **Embeddings grandes (1024 dimensões)** → Mais contexto, mas exigem mais processamento.
- ✓ **Embeddings pequenos (100 dimensões)** → Mais eficientes, mas podem perder detalhes semânticos.

2 Suporte Multilíngue [🔗](#)

- 📌 Se o knowledge base contém múltiplos idiomas, o modelo deve:
- ✓ Ter suporte nativo a esses idiomas.
- ✓ Ser adaptável para estender o suporte.

3 Métricas de Desempenho [🔗](#)

- 📌 Escolha o modelo considerando:
- ✓ **Precisão na recuperação** de informações.
- ✓ **Pontuações de similaridade semântica.**
- ✓ **Qualidade das respostas geradas.**
- ✓ Teste diferentes modelos em benchmarks antes de decidir.

4 Compatibilidade e Facilidade de Integração [🔗](#)

- 📌 O modelo deve:
- ✓ Ser compatível com a infraestrutura existente.
- ✓ Integrar-se facilmente com bibliotecas e frameworks do projeto.

- ♦ Escolher o modelo certo de embeddings é essencial para garantir buscas eficientes e respostas precisas! 🚀

📌 Como Escolher um Banco de Dados Vetorial para RAG? [🔗](#)

A escolha de um **banco de dados vetorial** é crucial para garantir um sistema RAG eficiente. Diferentes bancos possuem características específicas, e a decisão deve considerar desempenho, escalabilidade e custo.

♦ Principais Fatores para Escolha [🔗](#)

1 Volume de Dados e Escalabilidade [🔗](#)

- ✓ Suporta grandes volumes de embeddings sem perda de eficiência?
 - ✓ Pode escalar horizontalmente conforme a necessidade?
-

2 Desempenho das Consultas [🔗](#)

- ✓ O tempo de resposta para buscas é adequado?
 - ✓ O índice vetorial otimiza consultas semânticas?
-

3 Filtros em Metadados [🔗](#)

- ✓ Permite filtrar buscas por atributos como data, categoria ou usuário?
 - ✓ Suporta filtragem eficiente para conjuntos grandes de documentos?
-

4 Busca Híbrida (Texto + Vetor) [🔗](#)

- ✓ Combina busca semântica (similaridade vetorial) e busca por palavras-chave?
 - ✓ Integra NLP para melhores resultados?
-

5 Integração e Ecossistema [🔗](#)

- ✓ Compatível com ferramentas como LangChain, LlamaIndex e frameworks de IA?
 - ✓ Suporte nativo para AWS, Azure ou Google Cloud?
-

6 Opções de Implantação [🔗](#)

- ✓ Disponível como SaaS (nuvem), open-source ou on-premise?
 - ✓ Fácil de gerenciar e escalar conforme a necessidade?
-

7 Custo e Modelo de Preços [🔗](#)

- ✓ Tem um custo previsível conforme a demanda?
 - ✓ Oferece um bom custo-benefício para consultas frequentes?
-

8 Segurança e Conformidade [🔗](#)

- ✓ Suporta criptografia e controle de acesso granular?
 - ✓ Está em conformidade com GDPR, LGPD e outras normas?
-

9 Performance e Latência [🔗](#)

- ✓ Consegue processar consultas em tempo real?
- ✓ Oferece baixa latência mesmo em consultas complexas?

♦ Sincronização no Amazon Bedrock Knowledge Bases [🔗](#)

📌 O Amazon Bedrock automatiza a criação e atualização dos embeddings no banco vetorial.

- ✓ Atualizações podem ser feitas via **AWS Console** ou **AWS Lambda**.
- ✓ O **Lambda** pode rodar sob demanda ou em cronogramas pré-definidos.
- ✓ O Amazon Bedrock gerencia a ingestão de dados e está pronto para chamadas **Retrieve** e **RetrieveAndGenerate API**.

♦ Escolher o banco certo impacta diretamente na eficiência do seu RAG! 🚀

♦ RAG Gerenciado com Amazon Bedrock Knowledge Bases [🔗](#)

O Amazon Bedrock Knowledge Bases oferece duas formas principais de consulta para aplicações **RAG**:

📌 Métodos Disponíveis [🔗](#)

1 Retrieve

- ✓ Retorna **documentos relevantes** do Knowledge Base.
- ✓ Ideal para quem deseja personalizar a geração da resposta.

2 RetrieveAndGenerate

- ✓ Recupera os documentos e **gera uma resposta** usando o FM.
- ✓ **Gerencia contexto de sessão** para conversas multi-turno.
- ✓ Retorna **somente fontes citadas**, reduzindo alucinações.

♦ Fluxo de Trabalho (Workflow) [🔗](#)

- 1 Usuário faz uma consulta (query).
- 2 Geração do embedding da consulta.
- 3 Recuperação de documentos semelhantes do Knowledge Base.
- 4 Os documentos são adicionados ao prompt.
- 5 O FM gera uma resposta baseada nos documentos.
- 6 Resposta final é entregue ao usuário.

♦ Campos Obrigatórios [🔗](#)

📌 Para RetrieveAndGenerate:

- ✓ **Query Input** (consulta do usuário).
- ✓ **Knowledge Base ID**.
- ✓ **Foundation Model (FM)** usado para geração.
- ✓ **Template de Prompt** para estruturar a resposta.
- ✓ **Número máximo de resultados retornados**.
- ✓ **Tipo de Busca** (exemplo: similaridade semântica).
- ✓ **Session ID** com **chave de criptografia KMS**

📊 Avaliação de Aplicações RAG com RAGAS [🔗](#)

O **RAG Assessment (RAGAS)** é um framework open-source para avaliar **pipelines RAG**. Ele fornece métricas específicas para avaliar separadamente:

- ✓ **Respostas geradas** (FM Output).
- ✓ **Recuperação de contexto** (search results).
- ✓ **Desempenho global** da aplicação RAG.

✚ Principais Métricas [↗](#)

- ♦ **Context Precision** → Mede a relevância do contexto recuperado.
- ♦ **Context Recall** → Mede a completude do contexto recuperado.
- ♦ **Resposta gerada** → Mede a precisão das respostas com base no contexto.

🚀 Melhores Práticas para RAG [↗](#)

- ♦ **1 Definir Domínio e Fontes de Dados**
 - ✓ Escolha dados confiáveis e relevantes para seu caso de uso (exemplo: manuais de produtos para suporte ao cliente).
- ♦ **2 Estratégia de Chunking**
 - ✓ Use chunking eficaz (fixo, semântico, hierárquico) para otimizar a recuperação e a performance.
- ♦ **3 Uso de Metadados**
 - ✓ Filtre documentos usando metadados para melhorar a precisão na busca.
- ♦ **4 Engenharia de Prompts**
 - ✓ Defina prompts otimizados e padronizados para diferentes cenários.
- ♦ **5 Avaliação Contínua e Refinamento**
 - ✓ Integre métricas RAGAS para iterar melhorias.
 - ✓ Ajuste prompts, refine estratégias de recuperação e colete feedback.

🚀 Técnicas Avançadas para RAG [↗](#)

🔍 Hybrid Search [↗](#)

- ✓ Combina **busca semântica** e **busca por palavras-chave** para melhorar a recuperação de informações.
- ✓ Útil para consultas complexas ou mal formuladas.
- ✓ O **Amazon Bedrock** escolhe automaticamente o melhor método de busca.

✚ Query Reformulation / Decomposition [↗](#)

- ✓ Divide consultas **complexas** em **subconsultas mais simples**.
- ✓ Recupera fragmentos mais precisos antes de gerar a resposta.
- ✓ Exemplo:

✓ **Consulta original:**

"Onde fica o prédio da AnyCompany na orla e como o escândalo do denunciante afetou sua imagem?"

✓ **Consulta reformulada:**

- 1 "Onde fica o prédio da AnyCompany na orla?"
- 2 "Como o escândalo do denunciante afetou a imagem da empresa?"

- ✓ As respostas das subconsultas são combinadas antes de gerar a resposta final.

📁 Semantic Cache [↗](#)

- ✓ **Armazena consultas repetidas** para evitar reprocessamento desnecessário.
- ✓ **Melhora a eficiência e reduz custos** em aplicações com padrões recorrentes de consulta.

🛡️ Protegendo Aplicações RAG com Amazon Bedrock Guardrails [↗](#)

- ✓ **Filtragem de conteúdo indesejado** (violência, discurso de ódio, etc.).
- ✓ **Bloqueio de temas sensíveis e redação de informações confidenciais**.

- ✓ **Proteção contra ataques de prompt injection e jailbreak.**
- ✓ **Redução de alucinações em até 75%.**
- 💡 **Resultado:** Aplicações RAG **mais seguras, confiáveis e eficientes!** 🚀

🧠 Introdução aos Agentes de IA [↗](#)

Os **Agentes** são aplicações que utilizam **LLMs (Large Language Models)** para realizar tarefas complexas, dividindo problemas em etapas menores e raciocinando sobre cada uma delas.

🔍 Como Funcionam os Agentes? [↗](#)

- ✓ **Orquestração de tarefas** → Os agentes estruturam e executam os passos necessários para completar uma solicitação.
- ✓ **Integração com Knowledge Bases** → Permite acesso a informações atualizadas.
- ✓ **Execução de Código** → Os agentes podem processar código para realizar cálculos ou automações.
- ✓ **Aplicação de Guardrails** → Protegem a aplicação contra respostas inadequadas e seguem políticas de IA responsável.

💡 Diferença Entre LLMs e Agentes [↗](#)

Característica	LLMs	Agentes
Geram texto?	✓ Sim	✓ Sim
Fazem raciocínio passo a passo?	⚠ Limitado	✓ Sim, com CoT e ReAct
Executam código e chamadas de API?	✗ Não	✓ Sim
Mantêm contexto da conversa?	⚠ Limitado	✓ Sim

🔧 O Poder dos Agentes [↗](#)

- ✓ **Planejamento dinâmico** → Reagem ao resultado de cada etapa antes de seguir para a próxima.
 - ✓ **Autonomia** → Não precisam de lógica fixa para cada cenário.
 - ✓ **Eficiência** → Automatizam processos complexos sem necessidade de programação manual extensiva.
- 🚀 **Resumo:** Agentes são interfaces inteligentes que permitem aos usuários interagir naturalmente com sistemas de IA para resolver problemas complexos de forma dinâmica e eficiente.

🔧 Como os Agentes Funcionam? [↗](#)

Os agentes processam **entrada em linguagem natural**, interpretam intenções, formulam prompts apropriados e executam tarefas para o usuário.

- ✓ **Interpretam consultas complexas** → Identificam contexto, ambiguidade e linguagem figurativa.
 - ✓ **Decompõem tarefas** → Dividem grandes problemas em subtarefas para maior eficiência.
 - ✓ **Integram sistemas externos** → Interagem com bancos de dados, APIs e knowledge bases.
 - ✓ **Corrigem erros** → Verificam respostas e se autocorrigem quando necessário.
 - ✓ **Tornam as decisões mais transparentes** → Explicam cada passo do raciocínio.
-

Casos de Uso de Agentes com Amazon Bedrock [↗](#)

- ♦ **RH:** Consulta dias de férias e faz reservas de folga.
- ♦ **Seguros:** Gerencia sinistros e solicita documentação pendente.
- ♦ **Suporte técnico:** Recupera documentos de bases de conhecimento.
- ♦ **Restaurantes:** Gerencia cardápios e reservas de mesas.
- ♦ **E-commerce:** Ajuda clientes a encontrar produtos.
- ♦ **CRM:** Facilita interações com dados de clientes.
- ♦ **Automação de seguros:** Acelera processamento de sinistros.
- ♦ **Text-to-SQL:** Converte linguagem natural em consultas SQL.

Visão Geral dos Amazon Bedrock Agents [↗](#)

Os **Amazon Bedrock Agents** são sistemas inteligentes que usam **modelos fundacionais (FMs)** para interpretar comandos, executar ações e manter a memória de interações passadas.

Componentes Principais [↗](#)

- ✓ **Instruções** → Definem o que o agente pode e não pode fazer.
 - ✓ **Foundation Model (FM)** → Processa entradas e gera respostas.
 - ✓ **Memória** → Retém contexto da sessão para respostas mais coerentes.
 - ✓ **Grupos de Ação** → Permitem que o agente execute funções via **AWS Lambda** ou código personalizado.
 - ✓ **Base de Conhecimento (Knowledge Base)** → O agente consulta informações externas relevantes.
 - ✓ **Guardrails (Regras de Segurança)** → Evitam respostas inadequadas e garantem compliance.
-

Benefícios dos Bedrock Agents [↗](#)

- ♦ **Execução Autônoma** → Agentes podem tomar decisões e agir automaticamente.
- ♦ **Personalização e Segurança** → Guardrails garantem conformidade com regras empresariais.
- ♦ **Escalabilidade Empresarial** → Suporte para **CloudFormation, CDK, Terraform**.
- ♦ **Monitoramento e Logging** → Integração com **Amazon CloudWatch** para análise e auditoria.
- ♦ **Integração Flexível** → Disponível em **C++, Go, Java, Python, .NET, PHP**, entre outros.

Configuração e Execução dos Amazon Bedrock Agents [↗](#)

Configuração em Tempo de Construção (Build-time) [↗](#)

Na fase de **construção**, você define os componentes principais do agente, como:

- ✓ **Instruções** → Direcionam o comportamento do agente.
- ✓ **Grupos de Ação (Action Groups)** → Permitem executar funções/APIs externas (opcional).
- ✓ **Base de Conhecimento (Knowledge Base)** → Fornece dados específicos ao agente (opcional).
- ✓ **Regras de Segurança (Guardrails)** → Protegem contra respostas inadequadas (opcional).

Esses elementos são usados para criar **prompts base**, garantindo que o agente compreenda o contexto e execute as tarefas corretamente.

Processo de Execução em Tempo de Execução (Run-time) [↗](#)

- 1 **Pré-processamento** → O agente valida e interpreta a entrada do usuário.
- 2 **Orquestração** →
 - ♦ Invoca **Grupos de Ação** para executar funções.
 - ♦ Consulta a **Base de Conhecimento** para obter dados relevantes.

- ♦ Gera a melhor resposta com base no **modelo fundacional (FM)**.

3 Retorno de Controle (Opcional) → O agente pode retornar parâmetros à aplicação, que decide como executar ações externas.

4 Pós-processamento (Opcional) → Formata a resposta final antes de retornar ao usuário.

Se a **memória** estiver ativada, o agente pode armazenar um **resumo da conversa** para uso futuro.

Criando e Implantando um Agente no Amazon Bedrock [↗](#)

Criando um Agente na AWS Console [↗](#)

1 Acesse a área de agentes: Vá até *Amazon Bedrock* → *Builder tools* → *Agents*.

2 Crie um novo agente: Clique em *Create agent* e forneça um nome e uma descrição (opcional).

3 Configure o Agente: No *Agent Builder*, defina:

✓ **IAM Role** para permissões.

✓ **Modelo Fundacional (FM)** que o agente usará.

✓ **Instruções** para direcionar o comportamento do agente.

4 Configurações adicionais:

- ♦ **Interpretador de código** → Permite ao agente executar código.
- ♦ **Entrada do usuário** → Define se o agente pode solicitar mais informações.
- ♦ **Chave KMS** → Configuração de criptografia de recursos do agente.
- ♦ **Tempo limite de sessão** → Define quando a conversa expira (padrão: 10 minutos).

5 Configurações avançadas:

- ♦ *Grupos de Ação*, *Bases de Conhecimento* e *Guardrails* podem ser configurados aqui.
- ♦ *Advanced Prompts* permitem editar modelos de prompt padrão.

6 Salvar e preparar: Clique em *Save* e *Prepare* para ativar o agente.

Implantação e Testes [↗](#)

Após preparar o agente, use o **Test Agent Chat** na console da AWS para validar seu funcionamento antes da implantação final.

Criando e Implantando Agentes Programaticamente no Amazon Bedrock [↗](#)

Criando um Agente com a API do Amazon Bedrock [↗](#)

O Amazon Bedrock SDK permite que desenvolvedores criem e integrem agentes diretamente em seus aplicativos usando diversas linguagens, como C++, Go, Java, JavaScript, Kotlin, .NET, PHP, Python, Ruby, Rust, SAP ABAP e Swift.

Abaixo, um exemplo utilizando o SDK *boto3* em Python.

Criando uma Política IAM para o Agente [↗](#)

Antes de criar o agente, é necessário definir uma **IAM Role** com permissões mínimas para invocar o modelo fundacional (FM).

O seguinte exemplo cria uma **IAM Policy** chamada `hr-agent-bedrock-policy`, permitindo a chamada do método `bedrock:InvokeModel` no modelo **Anthropic Claude 3 Sonnet FM**:

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": "bedrock:InvokeModel",
7       "Resource": "arn:aws:bedrock:us-east-1::foundation-model/anthropic.claude-3-sonnet"
8     }
9   ]
10 }
```

```
9     ]
10 }
11
```

Criando um Agente com o SDK boto3 (Python) [↗](#)

Após configurar a **IAM Role**, podemos criar um agente usando *boto3*:

```
1 import boto3
2
3 # Criando o cliente do Bedrock
4 client = boto3.client('bedrock')
5
6 # Criando o agente
7 response = client.create_agent(
8     agentName='HR-Agent',
9     description='Agente de RH para consultar dias de férias disponíveis',
10    roleArn='arn:aws:iam::123456789012:role/hr-agent-role',
11    foundationModelArn='arn:aws:bedrock:us-east-1::foundation-model/anthropic.claude-3-sonnet',
12    instructions='Responda perguntas sobre saldo de férias dos funcionários e auxilie no agendamento.',
13    idleSessionTTLInSeconds=600
14 )
15
16 print("Agent criado com sucesso:", response)
17
```