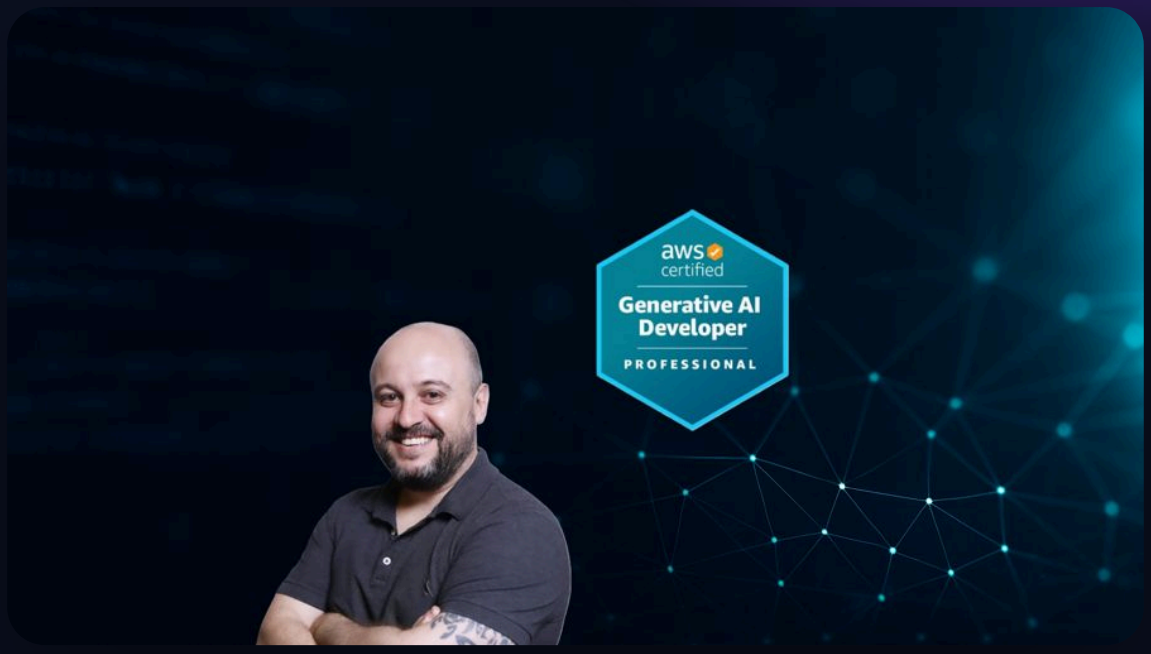
Strecht Simulados · Guia de preparação



Guia de preparação AIP-C01

Tudo o que a prova cobra, organizado por domínio: conceitos-chave, comparações que confundem, pegadinhas reais, gatilhos do enunciado, questões comentadas e um plano para chegar pronto ao exame.

5
capítulos por domínio

32
questões comentadas

75%
nota de corte

6
provas completas no curso

ANTES DE COMEÇAR

Como usar este guia

Este guia foi feito para ser estudado junto com as 6 provas completas do curso AIP-C01: o guia explica o que cai e onde estão as armadilhas; as provas treinam o ritmo e mostram o que você ainda não domina.

Quando	O que fazer
Semana 1	Faça o teste diagnóstico (página seguinte) e leia os capítulos 1 e 2. No fim da semana, faça a Prova 1 do curso no tempo do exame.
Semana 2	Leia os demais capítulos, começando pelos domínios em que você errou mais no diagnóstico. Faça as provas 2 e 3.
Semana 3	Releia os alertas de pegadinha e as tabelas de comparação. Faça as provas 4 e 5 e revise cada erro pela explicação.
Semana 4	Estude a folha de cola, faça a Prova 6 e refaça as provas abaixo de 80%. Agende o exame quando estiver acima de 80% em todas (a nota de corte é 75%).

Mapa do exame

Cada prova do curso tem 75 questões e 180 minutos, no formato do exame. Referência de aprovação: 75%.

Domínio (capítulo)	Peso	Neste guia
1. Domínio 1: Integração de modelos de base, gestão de dados e conformidade	31%	18 conceitos · 14 pegadinhas
2. Domínio 2: Implementação e integração	26%	18 conceitos · 14 pegadinhas
3. Domínio 3: Segurança, proteção e governança de IA	20%	17 conceitos · 14 pegadinhas
4. Domínio 4: Eficiência operacional e otimização de aplicações de IA generativa	12%	16 conceitos · 12 pegadinhas
5. Domínio 5: Testes, validação e solução de problemas	11%	16 conceitos · 12 pegadinhas

Como ler os capítulos: os **tópicos-chave** são o que mais cai; a **tabela de memorização** separa o que a prova costuma confundir; os **alertas de prova** mostram as armadilhas das alternativas erradas; os **gatilhos** ligam palavras do enunciado à resposta; e o **checklist** diz se você está pronto para o domínio.

Estude um pouco todo dia, de graça

Instagram @strecht.simulados

Um post de prova por dia: questão comentada, pegadinhas e o “não confunda” de cada certificação.

Seguir no Instagram →

YouTube Strecht Simulados

Shorts diários com pegadinhas de prova e, aos sábados, um simulado comentado.

Inscrever-se no canal →

DIAGNÓSTICO

Você já sabe isso?

Responda às 12 questões sem consultar nada. As respostas, com a explicação e o capítulo para revisar, estão no fim do guia. Comece o estudo pelos capítulos em que errou.

D1 · Domínio 1: Integração de modelos de base, gestão de dados e conformidade

Uma software house oferece geração de código com o Amazon Bedrock. O front-end envia só um identificador de prompt e alguns parâmetros de negócio; o backend precisa obter o template aprovado, preencher as variáveis, chamar o modelo e registrar, para auditoria, qual versão do prompt gerou cada resposta. Qual implementação atende a esses requisitos com MENOS código próprio de gestão de templates?

A. Deixar o front-end montar o prompt completo a partir de um catálogo de templates e enviar ao backend o texto final junto com o identificador do template para rastreio
B. Guardar os templates em variáveis de ambiente da Lambda, preencher as variáveis no código da função e registrar cada chamada ao modelo no CloudWatch Logs
C. Publicar versões aprovadas dos templates no Prompt Management e, na Lambda, chamar a Converse com o ARN da versão e promptVariables, gravando o histórico no DynamoDB
D. Guardar os templates como configuração no AWS AppConfig, preencher as variáveis no código da Lambda e registrar no histórico só o nome do template usado

D2 · Domínio 1: Integração de modelos de base, gestão de dados e conformidade

Uma seguradora quer usar IA generativa para resumir laudos de sinistro e precisa decidir, em seis semanas, se o projeto vai para produção. A diretoria exige que a decisão se baseie em evidências: qualidade dos resumos comparável à dos analistas, latência aceitável para o fluxo de atendimento e custo por laudo dentro do orçamento. A equipe ainda não escolheu o modelo de base. Qual abordagem atende a esses requisitos com o MENOR esforço operacional?

A. Executar uma PoC no Amazon Bedrock com laudos reais anonimizados, critérios de saída definidos antes (qualidade, latência e custo por laudo) e avaliação dos modelos candidatos
B. Liberar o assistente direto para um grupo piloto de analistas e decidir pela produção com base na quantidade de resumos gerados por dia nas primeiras semanas de uso
C. Comprar Provisioned Throughput para o modelo mais capaz do catálogo já na prova de conceito, para que a latência medida no piloto seja igual à da futura produção, e aprovar se ela ficar estável
D. Fazer fine-tuning de um modelo de base com o histórico de laudos já resumidos e aprovar a produção quando a perda de validação do treinamento parar de cair

D3 · Domínio 1: Integração de modelos de base, gestão de dados e conformidade

Uma operadora logística montou no Amazon Bedrock Flows uma cadeia de dois prompts: o primeiro extrai de um e-mail de cliente o número do pedido e a alteração solicitada, e o segundo redige a resposta. Entre os dois, é preciso consultar o status real do pedido em uma API interna, acessível só pela VPC da empresa, e normalizar a data no padrão interno; depois do segundo prompt, o número do pedido deve sair mascarado. A equipe quer manter tudo no fluxo e reaproveitar o código Python que já usa em outros sistemas. Como o fluxo deve ser montado?

A. Acrescentar ao segundo prompt instruções para consultar o status do pedido, normalizar a data no padrão interno e mascarar o número do pedido antes de devolver o texto ao nó de saída
B. Inserir um nó de base de conhecimento entre os prompts, indexando todo dia os status exportados da API, e aplicar ao segundo prompt um guardrail com filtro de regex para o número do pedido
C. Inserir um nó de função Lambda entre os prompts para consultar a API e normalizar a data, e outro nó Lambda depois do segundo prompt para mascarar o número do pedido antes do nó de saída
D. Inserir nós de código inline entre os prompts e depois do segundo prompt, escrevendo em Python a consulta à API interna, a normalização da data e o mascaramento do número do pedido

D4 · Domínio 2: Implementação e integração

Uma consultoria oferece a analistas uma ferramenta interna em que o backend, em contêineres no Amazon ECS, chama a Converse API do Amazon Bedrock para redigir relatórios de mercado de várias páginas. Os analistas olham uma tela em branco por mais de um minuto, e parte das chamadas falha com timeout de leitura do cliente do SDK. A equipe quer que o texto apareça à medida que é gerado e que as chamadas longas deixem de falhar. Qual mudança atende a esses requisitos?

A. Aumentar o read_timeout do cliente do SDK para alguns minutos, manter a chamada síncrona da Converse e exibir aos analistas um indicador de progresso
B. Usar a StartAsyncInvoke com saída no Amazon S3 e consultar o status da tarefa a cada poucos segundos até o relatório de mercado ficar pronto para o analista
C. Trocar a Converse pela ConverseStream, repassar os eventos contentBlockDelta e conceder à role a permissão bedrock:InvokeModelWithResponseStream
D. Enviar os pedidos como jobs de inferência em lote do Bedrock e avisar o analista por e-mail quando o arquivo de saída do relatório estiver no S3

D5 · Domínio 2: Implementação e integração

Um escritório de advocacia usa um único prompt longo no Amazon Bedrock para, de uma vez, extrair as cláusulas de um contrato, classificar o risco de cada uma e redigir um parecer. As respostas variam muito entre execuções: cláusulas são esquecidas, a classificação mistura critérios e é difícil descobrir em qual etapa o erro aconteceu. A equipe quer mais consistência e poder testar e ajustar cada etapa separadamente, sem trocar de modelo. Qual abordagem atende a esses requisitos?

A. Dividir a tarefa em uma cadeia de prompts em que o primeiro extrai as cláusulas em JSON, o segundo classifica o risco a partir desse JSON e o terceiro redige o parecer, com validação da saída entre as etapas
B. Fazer fine-tuning do modelo com pareceres antigos do escritório, para que ele aprenda a executar extração, classificação e redação em uma única chamada
C. Aumentar o maxTokens e a temperatura do prompt único, para que o modelo tenha espaço e variedade suficientes para cobrir as três tarefas na mesma resposta
D. Executar o mesmo prompt único três vezes em paralelo e escolher por votação a resposta mais frequente, o que reduz a variação percebida pelos advogados sem mudar a estrutura do prompt, o modelo usado nem o fluxo atual de revisão

D6 · Domínio 2: Implementação e integração

Uma corretora de seguros atende pelo Amazon Connect com o Contact Lens habilitado e redação de dados sensíveis ativa. Ao fim de cada contato, o CRM precisa receber um resumo com campos próprios da corretora (motivo, apólice citada, promessa feita ao cliente e próximo passo), gerado por um modelo do Amazon Bedrock a partir da transcrição já mascarada. A equipe não quer manter infraestrutura de transcrição. Qual solução atende a esses requisitos?

A. Gravar o áudio das chamadas no Amazon S3, transcrever cada arquivo com jobs do Amazon Transcribe com redação de PII e chamar o Bedrock em seguida por uma máquina de estados do AWS Step Functions
B. Criar uma notificação de evento do S3 no prefixo Analysis/Voice/Redacted/ do bucket da instância para acionar uma Lambda que lê a transcrição mascarada, chama o Bedrock e grava os campos no CRM
C. Acionar uma função AWS Lambda pelo evento de contato DISCONNECTED no Amazon EventBridge e ler a transcrição do Contact Lens nesse mesmo instante para chamar o Bedrock e gravar o resumo no CRM
D. Transmitir o áudio das chamadas para o Amazon Kinesis Video Streams e consumir o stream com uma aplicação no Amazon ECS que transcreve, resume com o Bedrock e grava no CRM

D7 · Domínio 3: Segurança, proteção e governança de IA

Um varejista expôs um assistente de compras público que usa o Amazon Bedrock Guardrails com filtro de ataques de prompt, tópicos negados e filtro de informações sensíveis. Em um teste de intrusão, a equipe vermelha conseguiu mapear as regras porque a aplicação devolve ao usuário o nome da política que bloqueou cada tentativa. A segurança quer que o usuário receba uma resposta neutra, mas que o time continue investigando qual política foi acionada. Qual solução atende a esses requisitos?

A. Configurar o guardrail para mascarar os prompts bloqueados com o identificador da política, o que esconde o texto do atacante na resposta exibida pela aplicação
B. Retirar o filtro de ataques de prompt e confiar nos tópicos negados, que bloqueiam as tentativas de manipulação sem informar ao usuário qual tópico foi identificado na conversa
C. Trocar a ação dos filtros de bloqueio para detecção, para que o guardrail registre as violações e deixe o modelo responder normalmente ao usuário
D. Definir mensagens genéricas de bloqueio para prompts e respostas no guardrail e devolver só esse texto ao cliente, mantendo a avaliação detalhada do trace nos logs internos

D8 · Domínio 3: Segurança, proteção e governança de IA

Uma seguradora resume laudos de vistoria com um modelo do Amazon Bedrock e passa cada resumo pela ApplyGuardrail com a verificação de grounding contextual, usando o laudo como fonte. O comitê de IA responsável quer quantificar a incerteza dos resumos por tipo de sinistro, ser alertado quando a confiança média cair e enviar à revisão humana os resumos menos confiáveis, sem bloquear a maioria deles. Qual solução atende a esses requisitos?

A. Elevar o limiar de grounding do guardrail para 0,99, para que os resumos incertos sejam bloqueados na hora e reescritos pelos analistas de sinistro da seguradora
B. Usar a métrica InvocationsIntervened do namespace AWS/Bedrock/Guardrails como medida de confiança dos resumos de vistoria e revisar no fim de cada dia os resumos que foram bloqueados pelo guardrail
C. Pedir ao modelo que informe no fim de cada resumo uma nota de 0 a 10 para a própria confiança e mandar à revisão humana os resumos com nota abaixo de 7
D. Publicar como métricas do CloudWatch, com a dimensão tipo de sinistro, as pontuações de grounding e relevância do guardrail, alarmar sobre a média e revisar os resumos abaixo de um limiar

D9 · Domínio 3: Segurança, proteção e governança de IA

Um banco usa um modelo do Amazon Bedrock para recomendar a gerentes o limite de crédito de pequenas empresas, com base em documentos recuperados de uma base de conhecimento. A diretoria de IA responsável quer que o gerente veja por que a recomendação foi feita e quais documentos a sustentam, mas a segurança proíbe expor o prompt de sistema, que contém regras internas da política de risco. A interface precisa exibir esses elementos em um formato fixo. Qual solução atende a esses requisitos?

A. Exigir saída estruturada com JSON Schema no outputConfig.textFormat da Converse, com campos de recomendação, resumo da justificativa e evidências com ID do documento e trecho, mantendo o prompt de sistema no backend
B. Pedir no prompt que o modelo explique a recomendação em texto livre ao final da resposta, sem revelar as regras internas, e exibir esse parágrafo na interface
C. Ativar o raciocínio estendido do modelo e exibir ao gerente o conteúdo de raciocínio completo devolvido pela API, que já traz a justificativa passo a passo da recomendação de crédito com as regras aplicadas em cada etapa da análise
D. Mostrar ao gerente o prompt de sistema e o contexto recuperado ao lado da resposta, já que a transparência sobre as instruções facilita explicar a recomendação

D10 · Domínio 4: Eficiência operacional e otimização de aplicações de IA generativa

Uma empresa está otimizando os custos de uma aplicação de geração de código com um modelo de alta capacidade no Amazon Bedrock. As chamadas usam streaming, mas o time percebeu que muitos tokens gerados depois de certo ponto não são usados: os desenvolvedores interrompem a leitura ou ignoram o restante da resposta. O objetivo é reduzir custo e latência sem degradar a qualidade percebida. Qual abordagem é a MAIS adequada?

A. Ajustar o maxTokens por tipo de requisição com base nas métricas reais de consumo dos usuários e revisar o limite periodicamente
B. Definir stop sequences genéricas, iguais para cada tipo de requisição, para interromper a geração em marcadores comuns de fim de bloco
C. Migrar para um modelo menor e mais barato da mesma família, mantendo os mesmos parâmetros de geração e o mesmo prompt de sistema
D. Ativar o prompt caching no prefixo fixo do system prompt e dos exemplos, reaproveitando os tokens de entrada entre requisições

D11 · Domínio 4: Eficiência operacional e otimização de aplicações de IA generativa

Uma rede de varejo serve um LLM próprio em um endpoint de inferência em tempo real do Amazon SageMaker AI com instâncias de GPU, para um assistente usado pelos vendedores. Em horário comercial há picos intensos de requisições com streaming de tokens; fora dele e nos fins de semana as lojas estão fechadas, não há tráfego, e a equipe aceita alguns minutos de espera na primeira chamada do dia. O objetivo é reduzir o custo de GPU ociosa sem degradar a experiência nos picos. Qual abordagem é a MAIS adequada?

A. Migrar para o SageMaker AI Serverless Inference com provisioned concurrency configurada para o horário de pico
B. Configurar um endpoint multimodelo do SageMaker AI com esse único modelo para aproveitar o carregamento sob demanda
C. Hospedar o modelo como inference component com target tracking em ConcurrentRequestsPerCopy e mínimo de zero cópias fora do expediente
D. Migrar para um endpoint de inferência assíncrona do SageMaker AI, que enfileira as requisições e escala até zero instâncias sem tráfego

D12 · Domínio 4: Eficiência operacional e otimização de aplicações de IA generativa

Uma editora jurídica precisa gerar resumo e palavras-chave para 4 milhões de acórdãos que já estão no Amazon S3, para relançar o acervo em duas semanas. Nenhuma resposta é necessária em tempo real. O mesmo modelo atende, sob demanda e na mesma conta, o chatbot dos assinantes. A equipe quer o menor custo por token e não quer construir uma orquestração de milhões de chamadas síncronas com controle de throttling e retentativas. Qual solução atende a esses requisitos?

A. Enviar as chamadas pela Converse com service_tier flex, a partir de funções Lambda acionadas por uma fila do Amazon SQS, com retentativas e controle de concorrência para não esgotar a cota do modelo
B. Comprar Provisioned Throughput para o modelo durante as duas semanas e processar os acórdãos em paralelo com um estado Map distribuído do AWS Step Functions
C. Gerar arquivos JSONL com os registros no S3 e criar jobs de inferência em lote do Amazon Bedrock com entrada e saída em buckets do S3, acompanhando o estado dos jobs por eventos do EventBridge
D. Usar um perfil de inferência global entre Regiões nas chamadas sob demanda, para ganhar vazão e concluir o processamento do acervo antes do prazo

01

CAPÍTULO 1 · 31% DA PROVA

Domínio 1: Integração de modelos de base, gestão de dados e conformidade

É o maior domínio da prova (31%) e eu preciso dominar o ciclo inteiro de um RAG: preparar e validar os dados, escolher chunking e embeddings, montar o vector store, ajustar a recuperação e escrever e governar os prompts. Também cai muito escolha de modelo e de forma de consumo: Bedrock como padrão, SageMaker AI quando preciso de contêiner próprio, Custom Model Import, Provisioned Throughput ou implantação sob demanda. Os enunciados descrevem um sintoma (trecho certo não aparece, documento revogado na resposta, PDF digitalizado com contexto vazio) e pedem a correção na camada certa. Onde mais erro é atacar a geração (temperatura, janela maior, fine-tuning) quando o defeito está na ingestão ou na recuperação. A regra que mais me salva: conhecimento que muda é RAG, ação ou dado em tempo real é ferramenta, estilo e formato é fine-tuning.

Neste capítulo

18 tópicos-chave · 14 pegadinhas · 7 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Chunking hierárquico

Busca nos chunks filhos (pequenos, precisos) e entrega ao modelo o chunk pai (contexto amplo). Serve para documento longo e estruturado quando quero precisão na busca e contexto na resposta. Os limites de pai e filho são definidos por tamanho em tokens, não pela estrutura de cláusulas ou artigos.

Chunking semântico e sobreposição

O semântico corta onde o assunto muda; a sobreposição repete texto entre trechos vizinhos para não perder a informação cortada na borda. Divisão por estrutura de negócio (cláusula, artigo, item) que as estratégias nativas não respeitam pede 'sem chunking' + Lambda de transformação customizada. Conteúdo já segmentado (um arquivo por item) também vai sem chunking.

A estratégia de chunking não muda depois

O chunking da fonte de dados é fixado na criação; trocar chunking, modelo de embeddings ou metadados exige nova ingestão. Para ajustar a recuperação sem reindexar, mexa no Retrieve: overrideSearchType (busca híbrida) e numberOfResults.

Busca híbrida

Combina busca vetorial com lexical. É a resposta quando código exato, SKU ou termo literal não é encontrado pela busca vetorial. O reranker não resolve isso: ele só reordena o que a busca já trouxe.

Reranker

Reordena os trechos recuperados por relevância semântica. Use quando o trecho certo vem, mas em posição ruim, ou para mandar menos trechos ao modelo (menos ruído, latência e custo). Não ordena por vigência e não traz trecho que a busca nunca encontrou.

Filtros de metadados no Retrieve

Metadado no documento (.metadata.json) + filtro aplicado pelo backend no Retrieve impõem regras obrigatórias: só documentos vigentes, aprovados ou da área do usuário. implicitFilterConfiguration deduz o filtro da pergunta sem código. Metadados e ACLs que mudam muito ficam no DynamoDB pelo ID do documento, com junção na aplicação.

Bedrock Knowledge Bases (Managed)

RAG gerenciado com menor esforço: sincronização incremental (remove excluídos), filtros de metadados e RetrieveAndGenerate. Managed Knowledge Base tem conectores nativos de Confluence e SharePoint; desde 30/09/2026 bases gerenciadas pelo cliente não aceitam novos conectores desses tipos. AgenticRetrieveStream (recuperação agêntica) existe só nas bases totalmente gerenciadas.

Escolha do vector store

OpenSearch Serverless: sem gerenciar cluster e com busca híbrida. S3 Vectors: acervo enorme, poucas consultas, latência de segundos aceitável e só busca semântica, apenas vetores float. pgvector no Aurora: dados já no PostgreSQL e filtro SQL junto da busca vetorial. Neptune Analytics (GraphRAG): perguntas de múltiplos saltos entre entidades.

Índice k-NN criado à mão para a base

O campo knn_vector precisa ter dimension igual à dimensão do modelo de embeddings e engine faiss; nmslib não é suportado pelas bases de conhecimento. Trocar o modelo de embeddings pede índice novo, reindexação em paralelo e troca atômica de alias; não misture vetores de modelos diferentes.

Bedrock Data Automation (BDA)

Extrai conteúdo de documento, imagem, áudio e vídeo com pouco pipeline. Blueprints normalizam campos com confiança por campo sem treinar modelo; um projeto com um blueprint por tipo e divisão de documentos classifica pacotes misturados. PDF com tabelas e gráficos pede parsing com FM ou BDA: chunking não corrige parsing ruim.

Converse API

API unificada: troca de família de modelo mudando só o modelId, inclusive tool use e imagens (blocos image e text na mesma mensagem). Instruções vão no system e os turnos em messages, alternando user e assistant. É sem estado: o histórico precisa ser reenviado pela aplicação e guardado no backend (DynamoDB).

Prompt Management

Template versionado com variáveis (promptVariables), configuração de inferência e versão imutável referenciada pelo ARN. Na Converse com prompt gerenciado, o modelId é o ARN da versão, e inferenceConfig, system e toolConfig vêm do prompt e não podem ir na requisição. A otimização de prompt do Bedrock adapta o prompt ao modelo alvo após uma troca.

Bedrock Flows

Classifica, ramifica por condição e encadeia prompts sem código. Nó Lambda para lógica de negócio ou chamada a sistema interno entre prompts; iterador + coletor para aplicar o mesmo prompt a cada item de uma lista. Produção nunca no TSTALIASID (rascunho): versão imutável + alias, e promover ou reverter é atualizar o alias.

Customização de modelo

Texto sem rótulo = pré-treino continuado; pares prompt/resposta = fine-tuning supervisionado (.jsonl, um registro por linha, formato prompt/completion ou bedrock-conversation-2024). Saída verificável por execução e poucos exemplos = reinforcement fine-tuning com recompensa em Lambda. Modelo menor com a qualidade do grande, sem rótulos = Model Distillation. PEFT/LoRA adapta com pouco custo e base intacta.

Formas de consumo do modelo

Modelo ajustado com tráfego esporádico: implantação sob demanda (ARN da deployment como modelId); carga alta e constante: Provisioned Throughput. Pesos abertos ajustados fora da AWS com arquitetura suportada: Custom Model Import (serverless); arquitetura não suportada ou servidor, biblioteca e decodificação próprios: contêiner em endpoint do SageMaker AI. Modelo só no JumpStart ou no Bedrock Marketplace: implante no endpoint correspondente.

Perfis de inferência entre Regiões

Resolvem ThrottlingException em pico sem compromisso de capacidade e sem código de roteamento. Geográfico mantém na área; global (prefixo global.) vale quando não há restrição de residência. Residência numa Região exata exige invocação in-Region; perfil cross-Region não leva modelo customizado (use model copy).

Qualidade de dados antes do FM

Glue Data Quality com DQDL valida regras conhecidas em pipeline; Data Wrangler explora e limpa de forma visual. Formatos misturados se normalizam em código (Lambda) antes do prompt, não pelo modelo. PDF digitalizado sem texto pede OCR com Textract (TABLES) antes da ingestão.

Seleção de FM e PoC

Filtros eliminatórios primeiro, depois avaliação com dados reais e matriz ponderada de qualidade, latência e custo, comparando no Bedrock Model Evaluation com o próprio conjunto rotulado. Critérios definidos antes da PoC; ir para produção depende de KPI de negócio contra linha de base e custo unitário. Modelo em Legacy: avalie o sucessor no mesmo golden set e migre antes do EOL, não há migração automática.

Macetes para memorizar

💡 RAG x ferramenta x fine-tuning

Muda? RAG. Age ou é agora? Ferramenta. Jeito de falar? Fine-tuning.

💡 Onde corrigir a recuperação

Nem aparece = busca (chunking, embeddings, híbrida). Aparece mal posicionado = reranker. Aparece vazio = parsing/OCR.

💡 toolChoice

auto = modelo escolhe; any = usa alguma; tool = usa aquela.

💡 Tipo de customização pelo dado

Sem rótulo = pré-treino continuado; pares = fine-tuning; nota por execução = reinforcement; professor e aluno = destilação.

💡 Mudanças que exigem reindexar

Chunking, embeddings e metadados reindexam; tipo de busca e numberOfResults não.

💡 Escada de consumo de modelo

Esporádico = sob demanda; constante = provisionado; lote = assíncrono, nunca para resposta em tela.

💡 Converse

System manda, messages conversam, a aplicação lembra.

Tabela de memorização: não confunda

Quando é...	...e quando é
RAG Conhecimento que muda, necessidade de citar a fonte ou dado sensível que precisa poder ser descartado.	**Fine-tuning** Ensinar estilo, formato ou uma tarefa estreita com muitos exemplos rotulados; não dá rastreabilidade nem memoriza fatos por documento de forma confiável.

Busca híbrida O trecho certo nem aparece porque a pergunta tem código, SKU ou termo exato.	**Reranker** O trecho certo aparece, mas em posição ruim, ou quero mandar menos trechos ao modelo.
Chunking hierárquico Documento longo e estruturado: precisão na busca (filho) e contexto amplo na resposta (pai).	**Chunking semântico** Limites de trecho que precisam acompanhar a mudança de assunto.
OpenSearch Serverless Sem gerenciar cluster, com busca híbrida ou vetores binários.	**S3 Vectors** Acervo enorme, poucas consultas, latência de segundos aceitável, só busca semântica e só vetores float.
Provisioned Throughput Carga alta, constante e previsível; compromisso maior (1 ou 6 meses) baixa o preço por hora, e sem compromisso é para testes ou incerteza.	**Implantação sob demanda / sob demanda por token** Tráfego esporádico, intermitente ou incerto; capacidade reservada ficaria ociosa e cobrando por hora.
Custom Model Import Pesos abertos ajustados fora da AWS, arquitetura suportada, serverless e pagamento por uso.	**Endpoint do SageMaker AI com contêiner próprio** Arquitetura não suportada ou exigência de servidor, biblioteca, processador de logits ou decodificação próprios.
Perfil de inferência geográfico/global Throttling em pico e residência por área (geográfico) ou nenhuma restrição (global).	**Invocação in-Region** Processamento exclusivamente em uma Região exata.

Números que caem

Item	Valor
Bases gerenciadas pelo cliente sem novos conectores de Confluence e SharePoint	**desde 30/09/2026 (os existentes continuam)**
Formatos de imagem aceitos na Converse	**png, jpeg, gif, webp**
Formato do dataset de fine-tuning no Bedrock	**.jsonl, um registro por linha (prompt/completion ou bedrock-conversation-2024)**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Reduzir o tamanho dos trechos quando a informação foi cortada entre vizinhos.
Como não cair: Trecho menor aumenta a chance de separar a cláusula da exceção. A correção é sobreposição ou chunking hierárquico.

⚠ **Alerta de prova:** Ligar reranker ou instruir no prompt para ignorar documento revogado.
Como não cair: O reranker ordena por relevância, não por vigência, e o revogado continua no contexto. Filtro de metadados por data/status no Retrieve, ou tirar as versões antigas da fonte e sincronizar.

⚠ **Alerta de prova:** Aumentar a dimensão dos embeddings ou trocar a métrica de distância para corrigir recuperação.
Como não cair: Dimensão não muda o que foi extraído do PDF nem o que entrou no embedding. Produto interno só equivale a cosseno com vetores normalizados; truncamento no limite de tokens do modelo de embeddings explica conteúdo do fim do trecho nunca recuperado.

⚠ **Alerta de prova:** Subir a temperatura para o modelo explorar trechos alternativos ou acertar o formato.
Como não cair: Temperatura afeta a geração, não a recuperação, e aumenta a variação. Formato errado se resolve especificando a saída com exemplo e delimitando os dados.

⚠ **Alerta de prova:** Passar o ID da sessão na Converse achando que o Bedrock guarda o histórico.
Como não cair: A Converse é sem estado: a aplicação reenvia as mensagens a cada chamada e guarda o histórico no backend.

⚠ **Alerta de prova:** Colocar uma URL pré-assinada da foto no texto ou declarar HEIC como jpeg.
Como não cair: O modelo não acessa URLs citadas no prompt e a API não converte imagens. A imagem vai como bloco image na mensagem, em formato suportado (png, jpeg, gif, webp).

⚠ **Alerta de prova:** Usar Intelligent Prompt Routing para comparar provedores, ganhar capacidade de outra Região ou criar ramificações.
Como não cair: Ele escolhe entre modelos da mesma família por qualidade e custo. Comparação é Model Evaluation; capacidade é perfil cross-Region; ramificação é Bedrock Flows.

⚠ **Alerta de prova:** Criar perfil de inferência de aplicação para ganhar cota ou traduzir formato de requisição.
Como não cair: Perfil de aplicação rastreia uso e custo com tags. Não cria tokens por minuto adicionais nem converte o corpo nativo; para trocar de modelo sem reescrever, use a Converse.

⚠ **Alerta de prova:** Invocar o modelo ajustado pelo ID do modelo base.
Como não cair: O ID base invoca o modelo original. O ajustado é chamado pelo ARN da implantação sob demanda ou do Provisioned Throughput.

⚠ **Alerta de prova:** Usar perfil geográfico da UE para cumprir 'processar só em eu-central-1'.
Como não cair: O perfil geográfico pode encaminhar para outras Regiões da área. Residência numa Região exata = invocação in-Region.

⚠ **Alerta de prova:** Trocar o analisador de um campo já mapeado pela API _mapping ou usar _reindex no OpenSearch Serverless.
Como não cair: O analisador não muda em campo existente: índice novo com reindexação. O Serverless não oferece _reindex, e copiar vetores do modelo antigo não serve para o novo modelo.

⚠ **Alerta de prova:** Criar um tópico negado para cada assunto fora do escopo do assistente.
Como não cair: Escopo aberto se resolve com escopo e instrução de recusa no system prompt; tópico negado é para lista fechada de temas proibidos.

⚠ **Alerta de prova:** Aumentar exemplos de uma classe quando o modelo passou a favorecer esse rótulo.
Como não cair: Confira o balanceamento dos exemplos few-shot entre as classes antes de pensar em fine-tuning; mais exemplos da mesma classe aprofundam o viés.

⚠ **Alerta de prova:** Vetorizar linhas de tabela para responder soma, contagem ou filtro.
Como não cair: Busca vetorial devolve trechos parecidos, não agrega. Use text-to-SQL (KB com dados estruturados) ou SQL pré-definido no backend; o banco calcula.

Como cai na prova

📝 **Como cai:** Sintoma de RAG (resposta errada, trecho faltando, conteúdo antigo) com quatro correções em camadas diferentes.
Como resolver: Localize a camada: ingestão (parsing, limpeza, chunking), recuperação (busca, filtro, rerank) ou geração (prompt). Descarte as opções que mexem em temperatura, janela ou fine-tuning quando o defeito é anterior à geração.

📝 **Como cai:** Requisito de 'menor esforço operacional' ou 'sem gerenciar infraestrutura' para RAG ou modelo.
Como resolver: Prefira o gerenciado: Managed Knowledge Base, OpenSearch Serverless, BDA, Custom Model Import. Só escolha vector store próprio ou SageMaker AI se o enunciado exigir controle do índice, runtime ou arquitetura fora do Bedrock.

📝 **Como cai:** Escolher padrão de consumo pelo perfil de tráfego.
Como resolver: Leia o volume e a regularidade: esporádico ou em rajadas leva a sob demanda; constante e alto, Provisioned Throughput; milhões de itens sem pressa, inferência em lote; pico com throttling, perfil cross-Region.

📝 **Como cai:** Governança de prompt: versionar, aprovar e não renderizar no código.
Como resolver: Prompt Management com variáveis e versão pelo ARN; aprovação humana com Step Functions + task token e CreatePromptVersion no final; feedback vira caso no conjunto de avaliação antes da nova versão.

📝 **Como cai:** Escolha de modelo com benchmarks públicos empatados ou PoC.
Como resolver: Filtros eliminatórios, avaliação no seu conjunto rotulado no Bedrock Model Evaluation, matriz ponderada de qualidade, latência e custo, e KPI de negócio contra linha de base.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Código de peça, SKU ou termo exato não encontrado	**Busca híbrida (overrideSearchType no Retrieve, sem reindexar)**
Trecho certo recuperado, mas em posição ruim / contexto grande demais	**Reranker + menos trechos**
PDF digitalizado gera contexto vazio	**OCR com Textract antes da ingestão, ou parsing com FM/BDA**
Pergunta que depende de relações entre entidades de documentos diferentes	**GraphRAG com Neptune Analytics na Knowledge Base**
Segunda busca depende do resultado da primeira	**Recuperação iterativa por agente com a busca como ferramenta; base gerenciada sem código: AgenticRetrieveStream**
Usuários perguntam com palavras diferentes das dos documentos e a consulta não pode mudar	**Enriquecer chunks na ingestão com resumo e perguntas geradas pelo modelo**
Trecho curto não encontrado por perguntas com o nome do produto	**Prefixar o chunk com título e seção antes do embedding**
Filtro restritivo devolvendo menos de k resultados	**Filtragem eficiente dentro da consulta knn, não post_filter**
Índice k-NN gigante e memória cara	**Quantização (byte, FP16, binária/on_disk com rescoring)**
ThrottlingException em pico, sem compromisso de capacidade	**Perfil de inferência entre Regiões**
Trocar de modelo na falha e voltar sozinho	**Circuit breaker (retry e cross-Region não trocam de modelo)**
Modelo aberto que inventa turnos	**Aplicar o chat template do tokenizer**
Gravação estéreo com cada pessoa em um canal	**ChannelIdentification do Transcribe (mono = speaker labels)**
'This model doesn't support tool use'	**Limitação do modelo: trocar por um que suporte ferramentas**

🎯 **Se você lembrar de só uma coisa deste capítulo...**

- Conhecimento que muda é RAG; ação ou dado em tempo real é ferramenta; estilo e formato é fine-tuning.
- Temperatura nunca corrige recuperação nem formato de entrada.
- Chunking, embeddings e metadados exigem nova ingestão; busca híbrida e numberOfResults se ajustam no Retrieve.
- A Converse é sem estado e troca de modelo mudando só o modelId.
- Regra obrigatória sobre documentos se aplica com metadado + filtro no backend, nunca com instrução no prompt.

Essenciais para a prova (checklist)

☐ Sei escolher entre chunking fixo com sobreposição, hierárquico, semântico e sem chunking pelo sintoma descrito.

☐ Sei quando usar busca híbrida, reranker e filtro de metadados, e por que um não substitui o outro.

☐ Sei comparar OpenSearch Serverless, S3 Vectors, pgvector no Aurora e Neptune Analytics.

☐ Sei configurar um índice k-NN para a base (dimension igual ao modelo, engine faiss) e trocar de embeddings com alias.

☐ Sei qual técnica de customização usar para dados sem rótulo, pares, recompensa verificável e destilação.

☐ Sei quando usar sob demanda, implantação sob demanda, Provisioned Throughput, Custom Model Import e SageMaker AI.

☐ Sei estruturar uma chamada Converse (system, messages, image, toolChoice) e usar prompt gerenciado pelo ARN da versão.

☐ Sei usar Bedrock Flows com nó Lambda, iterador/coletor e alias de produção.

☐ Sei escolher entre perfil geográfico, global e invocação in-Region conforme a residência de dados.

☐ Sei validar dados com Glue Data Quality (DQDL) e prepará-los antes do FM com Data Wrangler, Comprehend e Textract.

Questões de revisão do capítulo

R1.1 · Domínio 1: Integração de modelos de base, gestão de dados e conformidade

Uma transportadora gera, com um modelo do Amazon Bedrock, comparativos de frete a partir de registros de três ERPs regionais. Um ERP grava datas como 03/04/2026 e outro como 2026-04-03; os valores chegam como 'R$ 1.234,56', '1234.56 BRL' ou em dólares, e os pesos vêm em quilos ou em libras. O modelo às vezes compara valores em moedas diferentes ou inverte dia e mês, e o mesmo pedido gera conclusões diferentes. A equipe quer respostas consistentes sem trocar de modelo. Qual solução atende a esse requisito?

A. Normalizar os registros em uma função AWS Lambda antes de montar o prompt, convertendo datas para ISO 8601, valores para uma moeda com código explícito e pesos para quilos
B. Extrair as datas, os valores e as quantidades com o reconhecimento de entidades do Amazon Comprehend e anexar ao prompt a lista de entidades encontradas em cada registro
C. Incluir no prompt de sistema uma tabela com os formatos usados por cada ERP e instruir o modelo a converter datas, moedas e unidades antes de fazer a comparação pedida
D. Validar os registros com regras do AWS Glue Data Quality e enviar ao modelo os que passarem, já que o serviço converte formatos de data e de moeda para um padrão comum antes da carga

R1.2 · Domínio 1: Integração de modelos de base, gestão de dados e conformidade

Uma operadora de telecom grava as ligações da central em estéreo, com o atendente em um canal e o cliente no outro, e transcreve os arquivos com o Amazon Transcribe antes de pedir a um modelo do Amazon Bedrock um resumo com o motivo do contato e o que foi prometido. Hoje o prompt recebe a transcrição como texto corrido, cheia de hesitações como ahn e hum, e o modelo atribui ao cliente promessas feitas pelo atendente. A equipe não quer trocar de modelo nem escrever um classificador de falas. Qual solução resolve o problema?

A. Fazer fine-tuning do modelo com transcrições já resumidas pelos supervisores, para que ele aprenda a identificar o atendente e a desconsiderar as hesitações durante a geração do resumo
B. Ativar o particionamento de interlocutores com dois falantes e enviar ao modelo o JSON completo da transcrição, com os itens palavra a palavra e os tempos de cada um, para que ele deduza quem falou cada trecho da ligação
C. Aplicar o filtro de vocabulário com o método mask nas hesitações e aumentar a temperatura do modelo, para que o resumo fique mais fluido e menos preso à literalidade da transcrição
D. Ativar a identificação de canais (ChannelIdentification), aplicar um filtro de vocabulário com o método remove para as hesitações e montar numa Lambda os turnos rotulados como Atendente e Cliente

R1.3 · Domínio 1: Integração de modelos de base, gestão de dados e conformidade

Uma companhia aérea indexa políticas de bagagem e remarcação em uma base de conhecimento do Amazon Bedrock a partir de um bucket do Amazon S3. Cada revisão de política é enviada como um novo arquivo com sufixo de versão, e os arquivos antigos continuam no bucket. Os agentes de atendimento passaram a receber respostas que misturam regras vigentes com regras de versões revogadas. A auditoria quer manter o histórico das revisões. Qual solução resolve o problema de forma duradoura?

A. Habilitar um modelo de reranking na consulta para ordenar os trechos pela data de modificação do arquivo e descartar da resposta final as versões revogadas da política de bagagem
B. Gravar cada revisão na mesma chave com o Versionamento do S3 ativo, excluir os arquivos antigos com sufixo e sincronizar a fonte para a base remover os trechos revogados
C. Adicionar ao prompt de sistema uma instrução para o modelo preferir o trecho com o maior número de versão quando a recuperação trouxer regras conflitantes da mesma política
D. Aumentar o número de resultados recuperados por consulta, para que a versão vigente da política esteja entre os trechos enviados ao modelo junto com as versões antigas

R1.4 · Domínio 1: Integração de modelos de base, gestão de dados e conformidade

Uma seguradora de viagens oferece um assistente que cota seguros com um FM do Amazon Bedrock. Para cotar, são obrigatórios destino, datas de ida e volta e idade de cada viajante, mas muitos clientes escrevem só 'quero um seguro para a Europa', e o modelo inventa as informações que faltam e devolve preços errados. O fluxo já roda no AWS Step Functions, e a empresa quer que o cliente seja perguntado sobre os dados ausentes antes da cotação final. Qual solução atende a esse requisito?

A. Ativar a verificação de grounding contextual do Amazon Bedrock Guardrails para bloquear cotações cujos dados não aparecem na mensagem do cliente
B. Instruir o modelo a assumir valores padrão para os campos ausentes, como sete dias de viagem e um viajante adulto, e informar as premissas usadas no fim da cotação para que o cliente as corrija depois, se quiser
C. Extrair os campos em JSON, usar um estado Choice para detectar os ausentes e pausar com task token enquanto o aplicativo pergunta ao cliente, chamando o modelo da cotação com os dados completos
D. Aumentar o maxTokens e a temperatura da chamada de cotação para que o modelo tenha espaço para considerar mais de uma hipótese de viagem

Respostas comentadas

R1.1. A) Normalizar os registros em uma função AWS Lambda antes de montar o prompt, convertendo datas para ISO 8601, valores para uma moeda com código explícito e pesos para quilos

Por quê: A qualidade e a consistência das respostas de um FM dependem da qualidade da entrada. Dados heterogêneos (datas, moedas, unidades) devem ser normalizados de forma determinística antes de entrar no prompt, por exemplo em uma função Lambda, em vez de pedir ao modelo que interprete cada formato. O guia do exame cita justamente Lambda para normalizar dados, o Bedrock para reformatar texto e o Comprehend para extrair entidades como formas de melhorar a entrada.

✗ B) Incorreta. O Comprehend identifica entidades como datas e quantidades, mas não as normaliza para um padrão nem resolve a ambiguidade entre dia e mês ou a moeda.

✗ C) Incorreta. Delegar a conversão ao modelo mantém a fonte de inconsistência: a interpretação de 03/04 e das moedas continua probabilística e pode variar a cada chamada.

✗ D) Incorreta. O Glue Data Quality avalia regras e aponta registros fora do padrão, mas não converte formatos; descartar registros ainda faria o comparativo perder dados.

> **Dica de prova:** Formatos misturados nos dados: normalize em código (Lambda) antes do prompt; não deixe o modelo converter.

R1.2. D) Ativar a identificação de canais (ChannelIdentification), aplicar um filtro de vocabulário com o método remove para as hesitações e montar numa Lambda os turnos rotulados como Atendente e Cliente

Por quê: Normalizar transcrições antes de resumir melhora a qualidade sem mexer no modelo: rótulos de quem falou (por canal em gravações estéreo, ou por particionamento de interlocutores em mono), remoção de hesitações e formatação em turnos. O Transcribe oferece identificação de canais e filtros de vocabulário com os métodos remove, mask e tag.

✗ A) Incorreta. O problema é de pré-processamento da entrada, não de conhecimento do modelo; o fine-tuning é caro, contraria a restrição de não mexer no modelo e não resolve a falta de rótulos de autoria no texto.

✗ B) Incorreta. O JSON bruto aumenta o ruído e o custo em tokens, e os rótulos genéricos de falante não dizem quem é o atendente; com a gravação já separada por canal, a identificação de canais é o recurso direto.

✗ C) Incorreta. O método mask troca as palavras por asteriscos, que continuam poluindo o texto, e a temperatura maior aumenta a variação da saída sem resolver quem disse o quê.

> **Dica de prova:** Gravação estéreo com um lado em cada canal: ChannelIdentification. Remover palavras: filtro de vocabulário com remove (mask troca por asteriscos).

R1.3. B) Gravar cada revisão na mesma chave com o Versionamento do S3 ativo, excluir os arquivos antigos com sufixo e sincronizar a fonte para a base remover os trechos revogados

Por quê: Versionamento na fonte e higiene do índice: a base de conhecimento indexa os objetos correntes da fonte e, na sincronização, reprocessa os alterados e remove os excluídos. Manter só a versão vigente como objeto corrente (com o histórico no Versionamento do S3) impede que políticas revogadas sejam recuperadas.

✗ A) Incorreta. Rerankers reordenam por relevância semântica em relação à pergunta, não por data do arquivo, e não excluem conteúdo revogado do índice.

✗ C) Incorreta. A instrução não impede que regras revogadas entrem no contexto e depende do modelo interpretar versões corretamente a cada resposta, o que não é duradouro.

✗ D) Incorreta. Trazer mais trechos aumenta a chance de misturar versões e o custo de tokens, sem tirar do índice as regras revogadas.

> **Dica de prova:** RAG devolvendo políticas conflitantes: tire as versões antigas da fonte (histórico no Versionamento do S3) e sincronize. Prompt e reranker não limpam o índice.

R1.4. C) Extrair os campos em JSON, usar um estado Choice para detectar os ausentes e pausar com task token enquanto o aplicativo pergunta ao cliente, chamando o modelo da cotação com os dados completos

Por quê: Clarificação guiada: antes da tarefa final, o fluxo verifica se a solicitação tem os dados obrigatórios e, se não tiver, pergunta ao usuário. No AWS Step Functions, isso se faz com um passo de extração estruturada, um estado Choice que testa os campos e uma tarefa com .waitForTaskToken que pausa a execução até o aplicativo devolver a resposta do usuário com SendTaskSuccess. O FM só recebe a tarefa final com o contexto completo.

✗ A) Incorreta. O grounding bloquearia a cotação ruim, mas não pergunta ao cliente o que falta; o atendimento terminaria sem cotação.

✗ B) Incorreta. Assumir valores padrão continua gerando cotações com dados que o cliente não informou, que é o problema descrito, e empurra a correção para depois.

✗ D) Incorreta. maxTokens e temperatura não trazem as informações ausentes, e uma temperatura maior aumenta a chance de o modelo inventar dados.

> **Dica de prova:** Dados faltando no pedido: pergunte antes de chamar o FM (Choice + task token no Step Functions), em vez de deixar o modelo supor.

02

CAPÍTULO 2 · 26% DA PROVA

Domínio 2: Implementação e integração

Aqui eu preciso saber montar a aplicação de GenAI de ponta a ponta: agentes e ferramentas (Strands, AgentCore Runtime, Gateway, Identity, Memory, Policy, MCP e A2A), padrões assíncronos e de streaming, Step Functions, EventBridge, API Gateway e implantação de modelos em endpoints. Os enunciados quase sempre trazem uma restrição de arquitetura (sem Lambda esperando, sem credencial no cliente, sem acoplamento ao legado, sem deploy para trocar modelo) e a resposta é o serviço gerenciado que cumpre exatamente essa restrição. Onde mais erro é aceitar 'aumentar o timeout' ou 'colocar no prompt' como solução: timeout maior não é arquitetura assíncrona e instrução no prompt não é controle. Também confundo com facilidade Express e Standard no Step Functions e REST API e HTTP API no API Gateway. Lembrar que Bedrock Agents Classic e A2I fecharam a novos clientes: agente novo é Strands + AgentCore e revisão humana é Step Functions com task token.

Neste capítulo
18 tópicos-chave · 14 pegadinhas · 7 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Padrão assíncrono no API Gateway

Geração de minutos não cabe no tempo de integração do API Gateway. Responda na hora com um ID de tarefa e exponha status: POST integrado a StartExecution de uma máquina Standard (devolve o executionArn) e GET ligado a DescribeExecution. StartSyncExecution continua síncrono.

.waitForTaskToken

Padrão de callback de um Standard Workflow: o Step Functions entrega um token a um processo externo e espera, por até um ano, que ele chame SendTaskSuccess ou SendTaskFailure. Elimina Lambda fazendo polling. Também é a base da revisão humana (Choice pela confiança + task token), já que o A2I fechou a novos clientes.

Express x Standard Workflow

Express tem duração máxima de cinco minutos e não suporta .waitForTaskToken. No enriquecimento do EventBridge Pipes (síncrono), Step Functions só pode ser Express. Job longo, callback e histórico por execução pedem Standard.

Streaming de tokens

Bedrock: ConverseStream (permissão bedrock:InvokeModelWithResponseStream). SageMaker AI: InvokeEndpointWithResponseStream (PayloadPart). Para ver o texto na tela: ConverseStream + Lambda response streaming (function URL em RESPONSE_STREAM, não BUFFERED) + REST API em modo STREAM, ou WebSocket.

Tool use na Converse

O modelo decide (toolUse) e o backend executa e devolve toolResult; segredos ficam fora do prompt. Ferramenta que falhou devolve toolResult com status error e causa clara; retry só para erro transitório. Chamada incompleta se valida contra o JSON Schema e o erro volta ao modelo, sem executar com parâmetro vazio.

AgentCore Runtime

Hospeda agente novo com sessões isoladas e tarefas longas sem gerenciar contêineres. Para agir em nome do usuário, valide o JWT do IdP no Runtime e propague a identidade, em vez de usar só a role do agente. Agente novo que decide ferramentas = Strands Agents (@tool) + AgentCore; Agents Classic só se já existir.

AgentCore Gateway

Transforma Lambda e APIs OpenAPI existentes em ferramentas MCP sem escrever servidor (targets). Servidor MCP de terceiro com OAuth entra como target MCP com credencial no AgentCore Identity; legado SOAP entra por uma Lambda adaptadora. Com AgentCore Policy (Cedar) aplica limites rígidos a cada chamada de ferramenta.

AgentCore Identity

Gerencia credenciais de saída e OAuth em nome do usuário (token vault). AgentCore Policy restringe o que o agente pode fazer, mas não obtém nem guarda tokens OAuth. Autorização por usuário em cada chamada de ferramenta = Identity + Gateway com Policy.

AgentCore Memory

Persiste estado entre sessões com estratégias por tipo de informação (preferência, fato, resumo, episódio), que podem ser combinadas; sem estratégia não há memória de longo prazo. Preferências por usuário entre sessões usam namespace no nível do actorId, sem sessionId; namespace no nível da estratégia é compartilhado entre atores.

AgentCore Code Interpreter e Browser

Code Interpreter executa cálculo exato e análise de dados em sandbox; não peça ao modelo para fazer contas no texto nem rode exec de código gerado dentro da VPC. Browser permite ao agente usar um site sem API, com login e formulários.

MCP x A2A

MCP expõe ferramentas e dados a um agente; descoberta dinâmica pelo cliente MCP atende ferramentas que mudam e pertencem a outras equipes. A2A (Agent Card + JSON-RPC) é agente conversando com agente de outra equipe ou framework. MCP simples e sem estado roda em Lambda; complexo ou com estado, em ECS.

Inference components no SageMaker AI

Vários modelos no mesmo endpoint com recursos reservados e escala independente por modelo, inclusive escala a zero (NoCapacityInvocationFailures para voltar). Multi-model endpoint compartilha tudo e carrega sob demanda. Serverless Inference não tem GPU.

Memória de GPU para LLM

Memória = pesos + cache KV (contexto x concorrência) + sobrecarga; escolher só pelos pesos dá OOM sob carga. Escale por métricas de GPU e tokens por segundo, não por CPU. Modelo que não cabe em uma GPU: contêiner LMI com paralelismo de tensor; aceitando pequena perda de qualidade, quantização.

Gateway de GenAI

Lambda + role do IAM + parâmetros externos dão chargeback, nenhuma credencial nos clientes e troca de modelo sem redeploy. Mapa tarefa para modelo fica no AppConfig (extensão da Lambda). API pública: Cognito para autenticar, usage plans + API key por cliente para limitar, Guardrails para prompt injection.

Integração orientada a eventos

Legado publica evento, regra do EventBridge aciona Lambda/Step Functions (com SQS e DLQ antes da Lambda) que chama o Bedrock, sem acoplar o legado ao modelo. Fim de job em lote: evento Batch Inference Job State Change. Saída do BDA: eventBridgeEnabled na invocação + regra por detail-type.

Resiliência e idempotência

Throttling: retry com backoff exponencial e jitter + fallback; ValidationException não se retenta. No Step Functions, Retry só nos erros transitórios e Catch para o resto, nunca States.ALL repetindo erro de validação. Retry após timeout que duplica registro pede consumidor idempotente com chave e gravação condicional.

Ferramentas de desenvolvimento

Amazon Q Developer na IDE gera testes, refatora e explica erro com o stack trace (Fix/Send to prompt). Kiro: specs para requisitos, desenho e tarefas aprovados antes do código; steering é contexto do projeto; hooks são automações. Amplify AI Kit (a.conversation e a.generation) dá chat com histórico e geração estruturada no front-end.

CI/CD de aplicação GenAI

Chamar modelo no CodePipeline: ação de invocação de Lambda com retentativas e parâmetros externos (Parameter Store/AppConfig) e PutJobFailureResult para falha controlada. Aprovação humana: ação de aprovação manual entre testes e deploy. Métrica de avaliação reprovou: rollback automático do estágio (onFailure ROLLBACK).

Macetes para memorizar

💡 Componentes do AgentCore

Runtime hospeda, Gateway conecta, Identity autentica, Memory lembra, Policy proíbe, Browser navega, Code Interpreter calcula.

💡 Express x Standard

Express é expresso: cinco minutos e sem token. Esperou mais, é Standard.

💡 Permissões de streaming

Converse anda com InvokeModel; ConverseStream anda com InvokeModelWithResponseStream. Bloqueou uma, bloqueie a outra.

💡 Erros e retry

Throttling tenta de novo com jitter; Validation corrige antes de tentar.

💡 Timeout não é arquitetura

Se a resposta proposta é aumentar timeout, procure ID de tarefa, fila e status.

💡 Kiro

Spec planeja, steering orienta, hook automatiza.

💡 Memória de GPU

Pesos + KV + folga: quem conta só os pesos estoura no pico.

Tabela de memorização: não confunda

Quando é...	...e quando é
Standard Workflow Jobs longos, callback com .waitForTaskToken, revisão humana que espera dias, histórico por execução.	**Express Workflow** Execuções curtas (até cinco minutos) e enriquecimento síncrono no EventBridge Pipes.
AgentCore Gateway Transformar APIs e Lambdas em ferramentas MCP e aplicar Policy a cada chamada.	**AgentCore Identity** Guardar e obter credenciais de saída e tokens OAuth em nome do usuário.

MCP Expor ferramentas e dados a um agente.	**A2A** Um agente conversar com agente de outra equipe ou framework (Agent Card + JSON-RPC).
Inference components Vários modelos com recursos reservados, escala independente por modelo e escala a zero com GPU.	**Multi-model endpoint** Muitos modelos pouco usados que compartilham a mesma frota e carregam sob demanda.
REST API do API Gateway Validar schema da requisição, transformar o corpo com mapping template VTL, rastreamento ativo do X-Ray, usage plans, API privada.	**HTTP API** Quando nenhuma dessas capacidades é exigida; a HTTP API não valida nem reescreve o body.
Return of control Agente existente com ação sensível que a própria aplicação precisa validar antes de executar.	**AgentCore Policy no Gateway** Limite rígido e determinístico (valor, ferramenta, usuário) em cada chamada de ferramenta de agentes no AgentCore.
Quantização Modelo não cabe numa GPU barata e aceita pequena perda de qualidade.	**Decodificação especulativa** Quer gerar mais rápido sem perder qualidade; não reduz a memória dos pesos.

Números que caem

Item	Valor
Duração máxima de um Express Workflow	**cinco minutos**
Espera máxima de um Standard Workflow com .waitForTaskToken	**até um ano**
Tempo limite de integração do API Gateway (padrão citado)	**29 segundos; aumento possível em APIs regionais, com possível redução da cota de throttling**
Duração máxima de uma função Lambda	**15 minutos**
Novas tentativas na invocação assíncrona da Lambda	**no máximo duas**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Pedir aumento do timeout de integração acima de 29 segundos e manter a Lambda síncrona.
Como não cair: O aumento existe para APIs regionais, mas mantém conexões abertas por minutos e não dá estado nem histórico. Use o padrão assíncrono com ID de tarefa e endpoint de status.

⚠ **Alerta de prova:** Converter para Express Workflow para economizar mantendo a espera longa.
Como não cair: Express dura no máximo cinco minutos e não suporta callback com task token.

⚠ **Alerta de prova:** Usar StartAsyncInvoke para mostrar texto progressivo ao usuário.
Como não cair: A invocação assíncrona grava no S3 e serve para vídeo ou geração de minutos. Texto na tela conforme sai é ConverseStream.

⚠ **Alerta de prova:** Liberar bedrock:InvokeModel só nos ARNs aprovados e deixar o streaming aberto.
Como não cair: Converse = InvokeModel e ConverseStream = InvokeModelWithResponseStream. Restrinja as duas ações pelos ARNs (e na SCP com Deny + NotResource).

⚠ **Alerta de prova:** Descrever um serviço SOAP em OpenAPI e registrar direto no Gateway.
Como não cair: O target OpenAPI faz HTTP com JSON. O XML fica numa Lambda adaptadora que vira o alvo do Gateway.

⚠ **Alerta de prova:** Reduzir a temperatura a zero para o agente parar de chamar a ferramenta errada.
Como não cair: Temperatura zero torna a escolha repetível, não correta. Corrija nomes e descrições das ferramentas e dos parâmetros.

⚠ **Alerta de prova:** Aumentar o timeout das ferramentas para conter agente em loop.
Como não cair: Timeout maior não impede novas chamadas indefinidas. Limite de iterações e timeout na orquestração.

⚠ **Alerta de prova:** Restringir lambda:InvokeFunction no IAM para limitar o valor de um reembolso feito pelo agente.
Como não cair: O IAM não enxerga o valor no payload. Limite rígido em ação do agente é AgentCore Policy no Gateway.

⚠ **Alerta de prova:** Migrar o LLM ajustado para SageMaker Serverless Inference para escalar a zero.
Como não cair: Serverless Inference não oferece GPU. Escala a zero com GPU é inference components; sem gestão de infraestrutura, Custom Model Import.

⚠ **Alerta de prova:** Escalar o endpoint quando o modelo não cabe em uma GPU.
Como não cair: Cada nova instância falha do mesmo jeito. Use contêiner LMI com paralelismo de tensor ou quantização.

⚠ **Alerta de prova:** Distribuir pedidos entre modelos por pesos no AppConfig quando o modelo depende do tipo de pedido.
Como não cair: Pesos não olham o conteúdo. Classificador + estado Choice no Step Functions roteia por tipo com rastreabilidade.

⚠ **Alerta de prova:** Colocar WAF numa REST API regional para tirá-la da internet.
Como não cair: A API regional continua pública. REST API privada + endpoint de interface execute-api + aws:SourceVpce + VPC link.

⚠ **Alerta de prova:** Usar request validator para transformar o corpo de um parceiro.
Como não cair: O validator só aceita ou rejeita. Transformar o body é mapping template VTL da REST API (não proxy).

⚠ **Alerta de prova:** Aumentar retries da invocação assíncrona da Lambda para não perder eventos.
Como não cair: A invocação assíncrona permite no máximo duas novas tentativas. Configure destino on-failure (mais detalhes que a DLQ).

Como cai na prova

📝 **Como cai:** Processo longo (minutos ou horas) com restrição de não manter função ativa esperando.
Como resolver: Identifique a duração: acima de cinco minutos descarta Express; acima de 15 minutos, Lambda; e atrás do API Gateway o tempo limite de integração já pede padrão assíncrono. Escolha Standard com .waitForTaskToken ou StartExecution + DescribeExecution, ou fila com ID de tarefa e notificação.

📝 **Como cai:** Agente precisa acessar sistemas, APIs de terceiros ou dados do usuário com segurança.
Como resolver: Separe as necessidades: hospedar (Runtime), expor ferramenta (Gateway), credencial OAuth (Identity), limite rígido (Policy), estado (Memory). Rejeite respostas que confiam no prompt para controle.

📝 **Como cai:** Integração com legado ou SaaS 'sem acoplamento'.
Como resolver: Procure evento publicado pelo sistema de origem, regra do EventBridge, fila com DLQ e Lambda ou Step Functions chamando o Bedrock. Descarte opções que fazem o legado chamar o modelo direto.

📝 **Como cai:** Implantar LLM em endpoint com custo, memória ou escala como problema.
Como resolver: Calcule memória (pesos + cache KV), escolha inference components para vários modelos e escala a zero, LMI com paralelismo de tensor quando não cabe, e lembre que Serverless não tem GPU.

📝 **Como cai:** Expor GenAI por API com limites por cliente, segurança e streaming.
Como resolver: REST API para validação, VTL, X-Ray e usage plans; Cognito ou IAM para autenticar; streaming com ConverseStream; nenhuma chamada ao Bedrock direto do navegador.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Geração de minutos atrás do API Gateway, cliente aceita consultar depois	**StartExecution (ID de tarefa) + DescribeExecution (status)**
Job externo de horas e Lambda fazendo polling	**.waitForTaskToken + SendTaskSuccess**
Ferramentas que mudam com frequência e pertencem a outras equipes	**Descoberta dinâmica por cliente MCP**
Lambda ou API OpenAPI existente como ferramenta MCP, sem escrever servidor	**Targets no AgentCore Gateway**
OAuth em nome do usuário / token vault	**AgentCore Identity**
Agente esquece ao trocar de sessão	**AgentCore Memory com estratégia e namespace por actorId**
Site sem API, com login e formulários	**AgentCore Browser**
Cálculo exato ou análise de dados pelo agente	**AgentCore Code Interpreter**
Compliance de 'nada persistido após a sessão'	**Estado em memória enviado em messages da Converse**
Bot do Lex respondendo perguntas abertas a partir de base do Bedrock	**AMAZON.QnAIntent, sem Lambda**
Milhares de documentos no S3 com concorrência controlada e falhas toleradas	**Map distribuído do Step Functions**
Dado não sai do prédio, mas a operação é AWS	**Outposts (Wavelength é latência na operadora 5G)**
Pod no EKS chamando serviço da AWS com menor privilégio	**IRSA ou EKS Pod Identity**
Recusar entrada que não cabe no contexto antes da inferência	**Validação no API Gateway + CountTokens (maxTokens é saída)**

🎯 **Se você lembrar de só uma coisa deste capítulo...**

- Agente novo é Strands + AgentCore; Agents Classic e A2I estão fechados a novos clientes.
- Express não tem .waitForTaskToken e dura até cinco minutos.
- Instrução no prompt não é controle: limite rígido vem de Policy, IAM, validação no backend.
- Para streaming, libere e restrinja também InvokeModelWithResponseStream.

Essenciais para a prova (checklist)

☐ Sei montar o padrão assíncrono com StartExecution, DescribeExecution e .waitForTaskToken.

☐ Sei diferenciar Runtime, Gateway, Identity, Memory, Policy, Browser e Code Interpreter do AgentCore.

☐ Sei quando usar MCP, A2A e agents-as-tools no Strands.

☐ Sei tratar toolUse/toolResult, erro de ferramenta e chamada incompleta na Converse.

☐ Sei entregar streaming com ConverseStream, Lambda response streaming e WebSocket.

☐ Sei dimensionar memória de GPU e escolher entre inference components, multi-model endpoint, LMI e quantização.

☐ Sei aplicar Retry/Catch, backoff com jitter e idempotência em integrações com FM.

☐ Sei escolher REST API ou HTTP API pelos recursos exigidos (validação, VTL, X-Ray, API privada).

☐ Sei integrar legado e SaaS por EventBridge, SQS com DLQ e Pipes.

☐ Sei usar Amazon Q Developer, Kiro e Amplify AI Kit no ciclo de desenvolvimento.

Questões de revisão do capítulo

R2.1 · Domínio 2: Implementação e integração

Uma empresa quer gerar código com um modelo do Amazon Bedrock dentro do pipeline de CI/CD que já roda no AWS CodePipeline. Os requisitos são: • o código gerado deve ser versionado no repositório Git que a empresa já usa no AWS CodeCommit; • a invocação deve ser programática, com controle de temperatura e máximo de tokens e logs para auditoria; • o pipeline deve falhar se o modelo não responder em até 10 segundos ou devolver erro; • nenhum servidor adicional deve ser gerenciado. Qual abordagem atende a esses requisitos?

A. Adicionar uma ação do CodePipeline com o provedor Amazon Bedrock, definindo no estágio o modelo, a temperatura e o máximo de tokens, e usar as versões do Prompt Management para guardar o código gerado
B. Adicionar uma ação de invocação do Lambda: a função chama a Converse com timeout de 10 s no SDK, grava o código no CodeCommit com CreateCommit e reporta PutJobFailureResult em erro ou timeout
C. Adicionar uma ação de build do AWS CodeBuild que chama o Amazon Bedrock pela AWS CLI e faz git push para o CodeCommit, mantendo o tempo limite do projeto de build no valor padrão
D. Usar um job Python shell do AWS Glue, acionado por regra do EventBridge na mudança de estágio, que chama o Amazon Bedrock e grava o código no Amazon S3 para sincronizar depois com o CodeCommit

R2.2 · Domínio 2: Implementação e integração

Uma agência de turismo roda um agente feito com Strands Agents no Amazon Bedrock AgentCore Runtime. Quando um cliente volta horas depois para continuar o planejamento de uma viagem, a requisição cai em uma nova sessão de execução e o agente não lembra do que foi combinado. A equipe quer retomar a conversa a partir do identificador do cliente e da sessão, sem criar e manter um banco de dados próprio para o histórico. Qual solução atende a esses requisitos com o MENOR esforço?

A. Aumentar o idleRuntimeSessionTimeout e o maxLifetime do Runtime para manter a microVM da sessão ativa até o cliente voltar, preservando o histórico da conversa na memória do processo do agente entre as visitas
B. Incluir no prompt de sistema um resumo fixo das preferências mais comuns dos clientes, para que o agente pareça lembrar o que foi combinado em cada nova sessão aberta
C. Gravar o histórico de mensagens em um arquivo no sistema de arquivos local da sessão do Runtime e reler esse arquivo quando o mesmo cliente enviar uma nova requisição
D. Criar um recurso do AgentCore Memory e passar ao agente um AgentCoreMemorySessionManager com memory_id, actor_id do cliente e session_id, que grava os eventos e os recarrega na retomada

R2.3 · Domínio 2: Implementação e integração

Uma distribuidora de energia está criando um agente no Amazon Bedrock AgentCore que precisa consultar, em tempo de execução, faturas e protocolos de um sistema legado que expõe só uma API SOAP interna (XML descrito por WSDL). A equipe quer baixo acoplamento entre o agente e o legado, tratamento dos erros da chamada SOAP e observabilidade, com o mínimo de infraestrutura para operar. Qual abordagem atende a esses requisitos?

A. Criar uma função Lambda adaptadora que traduz JSON para SOAP e trata os erros, registrada como alvo no AgentCore Gateway para virar ferramenta MCP
B. Exportar dumps diários do sistema legado para o Amazon S3 e indexá-los em uma base do Amazon Bedrock Knowledge Bases consultada pelo agente
C. Descrever o serviço SOAP em uma especificação OpenAPI e registrá-la no AgentCore Gateway com integração HTTP direta ao endpoint legado
D. Executar no Amazon ECS um servidor MCP que repassa as requisições do agente ao endpoint SOAP e devolve ao modelo as respostas XML brutas, sem tratamento de falhas

R2.4 · Domínio 2: Implementação e integração

Uma administradora de imóveis extrai cláusulas e valores de contratos de locação com um modelo do Amazon Bedrock orquestrado pelo AWS Step Functions. Quando a confiança da extração fica abaixo de um limiar, um analista precisa revisar e corrigir os campos antes da gravação no sistema de gestão, e o fluxo deve esperar essa decisão, que pode levar horas, sem custo de computação ociosa. Qual solução atende a esses requisitos?

A. Publicar os casos de baixa confiança em uma fila do Amazon SQS lida por uma Lambda que reprocessa a extração com o modelo até que a confiança passe do limiar
B. Usar um estado Choice pela confiança e, nos casos baixos, uma tarefa com .waitForTaskToken que notifica o analista e retoma o fluxo quando ele devolve o token
C. Configurar um blueprint do Bedrock Data Automation com limiar de confiança mais alto, descartando os campos abaixo dele antes da gravação no sistema de gestão
D. Enviar os casos de baixa confiança para um fluxo de revisão humana do Amazon Augmented AI (A2I) e gravar no sistema depois que o revisor concluir a tarefa

Respostas comentadas

R2.1. B) Adicionar uma ação de invocação do Lambda: a função chama a Converse com timeout de 10 s no SDK, grava o código no CodeCommit com CreateCommit e reporta PutJobFailureResult em erro ou timeout

Por quê: A ação de invocação do AWS Lambda no CodePipeline entrega um job à função e aguarda que ela chame PutJobSuccessResult ou PutJobFailureResult; é a forma serverless de inserir lógica própria, como uma chamada ao Amazon Bedrock, com regras de falha definidas no código. Timeout configurado no cliente do SDK, parâmetros em inferenceConfig da Converse e logs no CloudWatch cobrem o controle e a auditoria pedidos.

✗ A) Incorreta. O CodePipeline não tem uma ação nativa do Amazon Bedrock, e o Prompt Management versiona prompts, não arquivos de código gerado.

✗ C) Incorreta. O CodeBuild é gerenciado e conseguiria chamar o modelo, mas com o tempo limite padrão do projeto a etapa não falha em 10 segundos quando o modelo demora; a regra de falha pedida não é aplicada.

✗ D) Incorreta. O job do Glue roda fora do fluxo do pipeline: o estágio não espera o resultado nem falha quando a chamada ao modelo dá erro, e a sincronização posterior com o repositório fica sem controle.

> **Dica de prova:** Lógica própria com falha controlada no pipeline: ação de invocação do Lambda + PutJobFailureResult.

R2.2. D) Criar um recurso do AgentCore Memory e passar ao agente um AgentCoreMemorySessionManager com memory_id, actor_id do cliente e session_id, que grava os eventos e os recarrega na retomada

Por quê: Estado de agente que precisa sobreviver entre execuções deve ficar fora do processo. Com Strands Agents, um session manager persiste e restaura a conversa; o AgentCore Memory oferece esse armazenamento gerenciado de curto prazo (e estratégias de longo prazo, se necessário).

✗ A) Incorreta. As sessões têm limite de vida e de ociosidade; manter microVMs paradas por horas gasta recursos e o histórico se perde quando a sessão termina.

✗ B) Incorreta. Um texto fixo não guarda o que cada cliente combinou; o agente continuaria sem o histórico real.

✗ C) Incorreta. O armazenamento local é da sessão de execução e não sobrevive ao fim dela nem é visto por outra sessão.

> **Dica de prova:** Agente esquece ao trocar de sessão: persista o estado com um session manager (AgentCore Memory), não confie na memória da sessão.

R2.3. A) Criar uma função Lambda adaptadora que traduz JSON para SOAP e trata os erros, registrada como alvo no AgentCore Gateway para virar ferramenta MCP

Por quê: O Amazon Bedrock AgentCore Gateway converte funções Lambda, especificações OpenAPI e modelos Smithy em ferramentas MCP que o agente descobre e chama. Para um legado SOAP, uma Lambda adaptadora concentra a tradução entre JSON e XML, a autenticação e o tratamento de erros, e devolve ao agente uma resposta limpa; o Gateway cuida da autenticação de entrada e saída e da observabilidade.

✗ B) Incorreta. Dumps diários não atendem à consulta em tempo de execução e deixam faturas e protocolos desatualizados.

✗ C) Incorreta. Um alvo OpenAPI faz chamadas HTTP com JSON; sem uma camada que monte o envelope SOAP e interprete o XML, a chamada ao legado não funciona e os erros SOAP chegam crus ao agente.

✗ D) Incorreta. Um servidor em ECS exige operar contêineres, e repassar XML bruto sem tratar falhas acopla o modelo ao protocolo legado e piora a confiabilidade.

> **Dica de prova:** Agente consultando legado SOAP: Lambda adaptadora como alvo do AgentCore Gateway. O agente fala MCP e JSON; o XML fica na camada de adaptação.

R2.4. B) Usar um estado Choice pela confiança e, nos casos baixos, uma tarefa com .waitForTaskToken que notifica o analista e retoma o fluxo quando ele devolve o token

Por quê: No padrão de callback do Step Functions (.waitForTaskToken), a tarefa entrega um token, por exemplo em uma mensagem do SQS ou em uma notificação ao analista, e a execução fica pausada sem consumir computação até que SendTaskSuccess ou SendTaskFailure seja chamado com esse token; workflows Standard podem esperar até um ano. Um estado Choice envia para essa etapa só os casos de baixa confiança.

✗ A) Incorreta. Reprocessar com o mesmo modelo não garante confiança maior e elimina a revisão humana exigida, além de gastar inferências repetidas.

✗ C) Incorreta. Um limiar mais alto só descarta mais campos; ninguém revisa nem corrige os dados, e a gravação fica incompleta.

✗ D) Incorreta. O Amazon Augmented AI está fechado a novos clientes desde 30/07/2026, e a revisão humana pode ser orquestrada no próprio Step Functions.

> **Dica de prova:** Revisão humana com espera longa no Step Functions: Choice pela confiança + .waitForTaskToken. A2I está fechado a novos clientes.

03

CAPÍTULO 3 · 20% DA PROVA

Domínio 3: Segurança, proteção e governança de IA

Este domínio pesa 20% e cobra controles reais, não boas intenções: Guardrails (cada tipo de filtro para um problema), controle de acesso a dados antes do contexto, rede privada, criptografia com KMS do cliente, logging de conteúdo, retenção e governança multi-conta. Também cai IA responsável: viés, transparência, explicação sem vazar instruções e revisão humana em decisões de alto impacto. A prova adora colocar 'instruir o modelo no prompt' ou 'confiar no guardrail' como controle de acesso, e isso é sempre errado. Onde mais erro é trocar o filtro do Guardrails (tópico negado x filtro de conteúdo x palavras x informações sensíveis x grounding) e confundir CloudTrail (quem chamou) com Model Invocation Logging (o que foi dito). Lembrar dos serviços fechados a novos clientes: SageMaker Clarify, Model Monitor, A2I e CloudTrail Lake não são resposta para solução nova.

Neste capítulo

17 tópicos-chave · 14 pegadinhas · 6 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Tipos de política do Guardrails

Tema proibido = tópico negado; dano (ódio, violência, insultos) = filtro de conteúdo; termos exatos (marcas, palavrões) = filtro de palavras; PII = informações sensíveis (CPF por regex); resposta fora da fonte = grounding contextual. Imagem nociva = filtros de conteúdo com modalidade IMAGE no mesmo guardrail.

Filtro de ataques de prompt e tags de entrada

Detecta jailbreak e injeção de prompt na entrada. Com InvokeModel e InvokeModelWithResponseStream, o filtro exige que o texto do usuário esteja marcado com as tags de entrada do guardrail; sem tags, ataques não são filtrados. Use tagSuffix aleatório por requisição: sufixo fixo permite fechar a tag e injetar depois dela. Junto dele, contenha o dano com menor privilégio nas ferramentas.

ApplyGuardrail

Aplica as mesmas políticas a modelos fora do Bedrock, a conteúdo de ferramentas e à ingestão (mascarar PII antes do vector store). O guardrail da chamada não olha o toolResult: instrução maliciosa em página ou documento lido pelo agente se verifica com ApplyGuardrail.

Automated Reasoning checks

Provam a resposta contra regras formais com várias condições e explicam a detecção. Tópicos negados bloqueariam o assunto inteiro, inclusive respostas corretas, e não verificam se a conclusão segue a política.

Grounding contextual e modo de detecção

Avalia se a resposta está apoiada na fonte e é relevante para a pergunta; reduz alucinação, não nocividade. Com ação NONE (detecção), a pontuação é lida no trace e a interface avisa sem bloquear. Não valida campos de chamadas de ferramenta: para isso, strict: true no toolSpec.

Streaming com guardrail

Modo síncrono quando nada impróprio pode aparecer ou há mascaramento de PII; assíncrono só quando latência vale mais que esse risco. streamProcessingMode só existe para respostas em streaming.

Model Invocation Logging x CloudTrail

Model Invocation Logging grava prompts e respostas (por Região) no seu destino, com KMS do cliente e política de proteção de dados no grupo de logs para mascarar PII. CloudTrail só audita chamadas de API. InvokeModel e alterações de guardrail são eventos de gerenciamento; Retrieve na base é evento de dados e exige seletor avançado.

Privacidade no Bedrock

Dados de inferência não treinam modelos base nem vão para o provedor do modelo. O histórico só existe se você ativar Model Invocation Logging no seu destino.

Bedrock só pela VPC

Endpoint de interface (PrivateLink) bedrock-runtime + política de endpoint para limitar modelos + condição aws:SourceVpce na IAM. Gateway endpoint só existe para S3 e DynamoDB. Retrieve dando timeout com Converse funcionando = falta o endpoint bedrock-agent-runtime.

Controle de acesso antes do contexto

RAG com dados restritos filtra no backend pela identidade antes de mandar ao modelo; guardrail é camada extra. Agente text-to-SQL com acesso parcial: role de menor privilégio + Lake Formation com LF-Tags; linhas por região e coluna oculta via Athena = data filter do Lake Formation (LF-Tags não filtram linhas).

Descobrir, mascarar e filtrar PII

Macie descobre onde há PII em buckets S3 (não altera nem mascara). Comprehend mascara PII em texto específico (a detecção de PII do Comprehend não aceita português). Guardrails filtra PII na inferência. Logs sem PII mas com o resto do texto: redação por entidade antes de gravar (Firehose + Lambda + Comprehend).

Retenção e imutabilidade

Object Lock em modo de conformidade: ninguém apaga, nem o raiz; governança permite bypass com permissão; legal hold pode ser removido por quem tem s3:PutObjectLegalHold. Lifecycle em bucket versionado precisa de NoncurrentVersionExpiration além de Expiration.

Governança multi-conta

SCP limita modelos e Regiões; política do Amazon Bedrock no AWS Organizations impõe guardrail versionado sem mudar código; Model Invocation Logging guarda o conteúdo. Self-service com padrão obrigatório sem dar IAM às equipes: Service Catalog + launch constraint.

Isolamento multi-tenant

Contrato que exige índice físico e chave próprios por cliente = padrão silo, com base de conhecimento e vector store dedicados. No padrão bridge há índice e base por cliente, mas coleção, bucket e chave KMS são compartilhados.

Linhagem e rastreabilidade

Linhagem automática dos jobs do Glue (OpenLineage) mostra fonte, transformação e índice. Rastrear resposta até a fonte: metadados de origem nas citações + registro das referências, gravando ID e versão da fonte, porque o URI aponta para a versão atual. Explicar por que o agente agiu: traces do AgentCore Observability por sessão.

IA responsável

Viés em respostas geradas: pares contrafactuais + LLM-as-a-judge, por grupo e agendado (Clarify fechado a novos clientes). Confiança do FM sem classificador: várias amostras + votação (self-consistency); confiança autodeclarada é mal calibrada. Decisão de alto impacto: o FM recomenda e a pessoa decide, com pausa por task token e registro de quem aprovou.

Transparência e explicação

Explicar decisão sem vazar instruções: saída estruturada com justificativa resumida e evidências, nunca prompt de sistema ou raciocínio bruto. Rotular resposta de IA é papel da aplicação, não do modelo. Model card do SageMaker AI documenta modelo próprio para regulador; AI Service Card é da AWS.

Macetes para memorizar

Filtros do Guardrails

Assunto = tópico; dano = conteúdo; palavra exata = palavras; dado pessoal = sensíveis; fora da fonte = grounding; regra formal = Automated Reasoning.

Logs

CloudTrail diz quem; Invocation Logging diz o quê.

PII nos serviços

Macie acha, Comprehend mascara texto, Guardrails filtra na conversa.

Endpoint de VPC

Gateway endpoint só para os dois do S: S3 e DynamoDB (o resto é interface).

Object Lock

Conformidade não tem chefe; governança tem quem libere.

Prompt não é cadeado

Se a opção protege dados pedindo ao modelo, ela está errada: controle vem antes do contexto.

Tabela de memorização: não confunda

Quando é...	...e quando é
Tópico negado Assunto proibido em lista fechada (por exemplo, recomendação de ativos).	**Filtro de conteúdo** Dano por categoria (ódio, insultos, violência, sexual), inclusive em imagens com modalidade IMAGE.
Filtro de ataques de prompt Jailbreak e injeção de prompt na entrada.	**Grounding contextual** Resposta que não se apoia na fonte ou não é relevante para a pergunta.
CloudTrail Quem chamou qual API e quando (auditoria de ações).	**Model Invocation Logging** Conteúdo de prompts e respostas, para auditoria e análise com Logs Insights.
Macie Descobrir onde há PII em buckets S3.	**Comprehend / Guardrails** Mascarar PII em texto específico (Comprehend) ou filtrar na inferência (Guardrails).
Object Lock modo de conformidade Exigência de que ninguém, nem o raiz, apague antes do prazo.	**Modo de governança / legal hold** Retenção que usuários com permissão especial ainda podem contornar ou remover.
Padrão silo Contrato exige índice físico e chave KMS exclusivos por cliente.	**Padrão bridge** Índice e base por cliente bastam, com coleção, bucket e chave compartilhados.

Endpoint de interface (PrivateLink) Acessar Bedrock (bedrock-runtime, bedrock-agent-runtime) sem internet.	**Gateway endpoint** Somente S3 e DynamoDB.

Números que caem

Item	Valor
SageMaker Clarify, Model Monitor, Ground Truth, A2I, Bedrock Agents Classic, Kendra e Q Business fechados a novos clientes	**30/07/2026**
CloudTrail Lake fechado a novos clientes	**desde 31/03/2026**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Listar no prompt as tabelas permitidas e pedir ao agente para recusar o resto.
Como não cair: Instrução no prompt não é controle de acesso: injeção ou SQL errado ainda lê as tabelas. Role de menor privilégio + Lake Formation.

⚠ **Alerta de prova:** Usar tópicos negados para detectar imagens nocivas.
Como não cair: Tópicos negados avaliam texto. Imagem nociva é filtro de conteúdo com modalidade IMAGE.

⚠ **Alerta de prova:** Usar o filtro de violência em intensidade alta contra prompt injection.
Como não cair: Injeção escrita em linguagem neutra passa pelo filtro de violência. Use o filtro de ataques de prompt na entrada, com tags.

⚠ **Alerta de prova:** Criar um guardrail por grupo para controlar quais modelos cada perfil pode invocar.
Como não cair: Guardrails filtram conteúdo, não autorizam modelos. Use role do IAM por grupo do user pool e identity pool escolhendo a role pelo token; em várias contas, SCP.

⚠ **Alerta de prova:** Criar gateway endpoint da VPC para o Bedrock.
Como não cair: Gateway endpoint só existe para S3 e DynamoDB. Bedrock usa endpoint de interface.

⚠ **Alerta de prova:** Usar aws:SourceIp com o CIDR privado da VPC na política.
Como não cair: Por endpoint da VPC, aws:SourceIp não traz o IP privado. Use aws:SourceVpce (e lembre que isso bloqueia o console e serviços fora da VPC).

⚠ **Alerta de prova:** Rodar o Macie para mascarar PII antes de indexar.
Como não cair: O Macie só descobre e classifica. Redija antes da ingestão com transformação Lambda + ApplyGuardrail ou Comprehend.

⚠ **Alerta de prova:** Aplicar legal hold para que ninguém apague logs.
Como não cair: Quem tem s3:PutObjectLegalHold remove o hold. Exigência regulatória absoluta é Object Lock em modo de conformidade.

⚠ **Alerta de prova:** Pedir ao modelo uma nota de confiança de 0 a 100 para decidir a revisão humana.
Como não cair: Confiança autodeclarada é mal calibrada, mesmo com chain-of-thought. Use self-consistency ou publique pontuação de confiança como métrica e roteie a faixa baixa para humanos.

⚠ **Alerta de prova:** Filtrar ou esconder dados do cliente para atender ao direito ao esquecimento.
Como não cair: Filtro e guardrail só escondem. Apague na fonte, sincronize a base, limpe o histórico e respeite a retenção dos logs.

⚠ **Alerta de prova:** Exibir ao usuário o raciocínio estendido como explicação da decisão.
Como não cair: O raciocínio bruto pode reproduzir regras do prompt de sistema. Mostre justificativa resumida e evidências em saída estruturada.

⚠ **Alerta de prova:** Contar InvocationsIntervened com alarme para bloquear o usuário que insiste em jailbreak.
Como não cair: A métrica é agregada, sem usuário; o alarme bloquearia todos. Conte as intervenções PROMPT_ATTACK do trace por usuário e bloqueie temporariamente no autorizador.

⚠ **Alerta de prova:** Colocar WAF para barrar toxicidade nas respostas.
Como não cair: O WAF inspeciona requisições HTTP com padrões e reputação; resolve bots e flood (regras gerenciadas + rate-based rule), não conteúdo do modelo.

⚠ **Alerta de prova:** Criptografar com SSE-S3 quando o requisito pede chave editável e revogável.
Como não cair: SSE-S3 usa chaves gerenciadas pelo S3. Política editável e revogável exige KMS gerenciada pelo cliente.

Como cai na prova

📝 **Como cai:** Problema de conteúdo (tema proibido, ódio, PII, jailbreak, alucinação) com várias políticas do Guardrails nas alternativas.
Como resolver: Classifique o problema e escolha a política correspondente; confira se ela precisa atuar na entrada, na saída ou nas duas, e se há imagem (modalidade IMAGE) ou conteúdo de ferramenta (ApplyGuardrail).

📝 **Como cai:** Usuário ou agente não pode acessar certos dados.
Como resolver: Procure controle determinístico antes do contexto: filtro pela identidade no backend, IAM de menor privilégio, Lake Formation, AgentCore Policy. Descarte prompt e guardrail como controle de acesso.

📝 **Como cai:** Auditoria pede evidência de quem fez e do que foi dito.
Como resolver: Combine CloudTrail (ações de API) com Model Invocation Logging (conteúdo), KMS do cliente, retenção com Object Lock e, se houver PII, política de proteção de dados no grupo de logs.

📝 **Como cai:** Rede privada para Bedrock e bases de conhecimento.
Como resolver: Endpoint de interface por API usada (bedrock-runtime, bedrock-agent-runtime), política de endpoint e aws:SourceVpce; vpcConfig no job de customização com endpoint do S3.

📝 **Como cai:** Viés, justiça ou excesso de recusas antes do lançamento.
Como resolver: Avaliação LLM-as-a-judge com métricas integradas por grupo (Stereotyping, Harmfulness, Refusal) em conjunto com casos legítimos e perigosos; guardrail é proteção em produção, não avaliação.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Campo de formulário injetando instruções via InvokeModel	**Tags de entrada do guardrail com tagSuffix aleatório**
Instrução maliciosa dentro de página ou documento lido pelo agente	**ApplyGuardrail sobre o conteúdo da ferramenta + limitar ações**
Modelo inventando campos em chamadas de ferramenta	**strict: true no toolSpec**
Mesmas políticas para modelo fora do Bedrock	**ApplyGuardrail**
Mesmo guardrail em toda a organização sem mudar código	**Política do Amazon Bedrock no AWS Organizations**
Avisar sobre baixa fundamentação sem bloquear	**Grounding com ação NONE e pontuação lida no trace**

Só links de domínios aprovados	**Validação determinística pós-geração (Lambda com allowlist)**
Não revelar ao atacante qual regra bloqueou	**Mensagem genérica ao usuário e trace detalhado só nos logs internos**
Moderar imagens com álcool, tabaco, jogos de azar	**Rekognition DetectModerationLabels com MinConfidence**
Pontuação numérica de toxicidade por categoria sem treinar	**Comprehend DetectToxicContent (texto em inglês)**
Chave de API de provedor externo	**Secrets Manager com rotação e política de recurso restrita à role**
Anonimização consistente e reversível	**Pseudônimo determinístico + mapeamento em cofre criptografado**
Template customizado da base e citações sumiram	**Faltou $output_format_instructions$ no prompt de geração**
Sem trecho relevante, não gerar	**Limiar sobre o score do Retrieve antes da chamada + handoff humano**

🎯 **Se você lembrar de só uma coisa deste capítulo...**

- Prompt e guardrail nunca são controle de acesso a dados.
- CloudTrail não guarda o conteúdo de prompts; Model Invocation Logging guarda.
- Gateway endpoint só existe para S3 e DynamoDB.
- O guardrail da chamada não inspeciona o toolResult; use ApplyGuardrail no conteúdo externo.
- Clarify, Model Monitor, A2I e CloudTrail Lake não entram em solução nova.

Essenciais para a prova (checklist)

☐ Sei escolher a política certa do Guardrails para cada tipo de risco, inclusive imagens.

☐ Sei proteger contra prompt injection direta (tags de entrada) e indireta (ApplyGuardrail no conteúdo da ferramenta).

☐ Sei configurar acesso privado ao Bedrock com endpoint de interface, política de endpoint e aws:SourceVpce.

☐ Sei diferenciar CloudTrail, eventos de dados com seletor avançado e Model Invocation Logging.

☐ Sei onde usar Macie, Comprehend e Guardrails para PII.

☐ Sei aplicar Object Lock, Lifecycle com NoncurrentVersionExpiration e KMS do cliente.

☐ Sei impor governança multi-conta com SCP, política do Bedrock no Organizations e Service Catalog.

☐ Sei medir viés e equidade com LLM-as-a-judge por grupo e pares contrafactuais.

☐ Sei montar revisão humana em decisão de alto impacto com Step Functions e task token.

- [ ] Sei rastrear uma resposta até a fonte com ID e versão do documento.

Questões de revisão do capítulo

R3.1 · Domínio 3: Segurança, proteção e governança de IA

Uma cooperativa de crédito ajustou um modelo de linguagem no Amazon SageMaker AI para analisar pedidos de renegociação. O regulador pede documentação estruturada e versionada do modelo, com uso pretendido, classificação de risco, dados de treinamento, métricas de avaliação e limitações conhecidas, e a possibilidade de exportar o documento para auditoria. Qual recurso atende a esse pedido com o MENOR esforço?

A. Linhagem das tabelas de treinamento registrada no AWS Glue Data Catalog do projeto
B. Metadados de versão e status de aprovação do pacote do modelo no SageMaker Model Registry
C. Model cards do Amazon SageMaker AI, preenchidos e versionados para o modelo ajustado
D. AWS AI Service Cards publicados pela AWS para os serviços de IA usados no projeto

R3.2 · Domínio 3: Segurança, proteção e governança de IA

Um marketplace de artesanato permite que vendedores enviem fotos dos produtos para um modelo multimodal do Amazon Bedrock gerar a descrição do anúncio. Algumas fotos enviadas têm conteúdo sexual ou violento, e a política exige bloquear essas imagens antes que cheguem ao modelo, com os mesmos níveis de filtro já usados para o texto. A equipe quer reaproveitar o guardrail que já protege os textos. Qual configuração atende a esses requisitos?

A. Aplicar a verificação de grounding contextual do guardrail às descrições geradas, bloqueando as respostas que não se apoiem no conteúdo da foto enviada pelo vendedor
B. Adicionar ao guardrail tópicos negados que descrevam conteúdo sexual e violento, que passam a ser avaliados também sobre as imagens enviadas junto com o texto
C. Habilitar a modalidade de imagem nos filtros de conteúdo Sexual e Violence do guardrail, enviar as fotos em PNG ou JPEG e aplicar o guardrail na chamada ao modelo
D. Ativar o filtro de ataques de prompt com modalidade de imagem, que classifica fotos com conteúdo sexual ou violento como tentativa de jailbreak e bloqueia o envio ao modelo

R3.3 · Domínio 3: Segurança, proteção e governança de IA

Um hospital digital tem várias aplicações que chamam modelos do Amazon Bedrock com textos em português. A equipe de segurança quer uma política centralizada que detecte e mascare dados como CPF, número de cartão e dados de contato tanto nos prompts quanto nas respostas, que seja aplicada obrigatoriamente em cada chamada de inferência e que deixe registro das intervenções para auditoria. Qual abordagem atende a esses requisitos?

A. Configurar o Amazon Macie para analisar os prompts e as respostas em trânsito entre as aplicações e o Amazon Bedrock, mascarando os dados pessoais detectados pelo serviço antes da entrega
B. Criar um guardrail com filtro de informações sensíveis em modo de mascaramento, com tipos de PII predefinidos e regex para o CPF, e exigi-lo pela condição bedrock:GuardrailIdentifier
C. Incluir em cada aplicação uma função Lambda que chama o DetectPiiEntities do Amazon Comprehend e remove os dados pessoais do prompt antes de enviá-lo ao modelo de IA
D. Ativar o Amazon GuardDuty na conta para inspecionar as chamadas feitas ao bedrock-runtime e bloquear as requisições que contenham dados pessoais de pacientes

R3.4 · Domínio 3: Segurança, proteção e governança de IA

Uma fintech de crédito extrai dados de propostas com a API Converse e tool use: o toolChoice força a ferramenta registrar_proposta, cujo inputSchema define os campos do contrato e um enum de modalidades. Em produção, parte das chamadas traz campos que não existem no contrato, como observacao_extra, ou uma modalidade fora do enum, e o serviço de destino rejeita o registro. A equipe quer que a própria geração obedeça ao schema da ferramenta, sem escrever lógica de reparo. Qual mudança atende a esse requisito?

A. Reduzir a temperatura a zero e incluir no prompt de sistema exemplos de chamadas corretas da ferramenta, com os campos permitidos e os valores aceitos no enum de modalidades
B. Associar à chamada um guardrail com verificação de grounding contextual, para bloquear respostas que tragam campos não presentes nos documentos da proposta
C. Adicionar strict: true na definição da ferramenta e declarar additionalProperties false no inputSchema, para que o Bedrock restrinja a geração da chamada ao schema
D. Trocar o toolChoice para auto e deixar o modelo decidir quando chamar a ferramenta, devolvendo texto livre quando não tiver certeza da modalidade

Respostas comentadas

R3.1. C) Model cards do Amazon SageMaker AI, preenchidos e versionados para o modelo ajustado

Por quê: Os model cards do Amazon SageMaker AI documentam em um só lugar o uso pretendido, a classificação de risco, os detalhes e métricas de treinamento, os resultados de avaliação e considerações adicionais de um modelo. Edições geram novas versões imutáveis, o card pode ser exportado em PDF para auditores e integra-se ao Model Registry.

✗ A) Incorreta. A linhagem mostra a origem e as transformações dos dados, mas não documenta uso pretendido, métricas de avaliação nem limitações do modelo.

✗ B) Incorreta. O Model Registry controla versões e aprovação do modelo para implantação, mas seus metadados não estruturam uso pretendido, riscos e limitações para um regulador.

✗ D) Incorreta. Os AI Service Cards documentam serviços gerenciados da própria AWS; não descrevem um modelo ajustado pela cooperativa.

> **Dica de prova:** Documentar um modelo próprio para regulador: model card do SageMaker AI. Service Card é da AWS; Model Registry é versão e aprovação.

R3.2. C) Habilitar a modalidade de imagem nos filtros de conteúdo Sexual e Violence do guardrail, enviar as fotos em PNG ou JPEG e aplicar o guardrail na chamada ao modelo

Por quê: O Amazon Bedrock Guardrails estende os filtros de conteúdo a imagens: ao criar ou editar o guardrail, cada categoria pode receber as modalidades TEXT e IMAGE, e a mesma força de filtro vale para as duas. O guardrail avalia imagens de entrada e as geradas pelo modelo, inclusive pelo ApplyGuardrail. O recurso aceita PNG e JPEG, então outros formatos precisam ser convertidos antes.

✗ A) Incorreta. O grounding verifica se a resposta se apoia na fonte; ele não detecta nem bloqueia imagens nocivas na entrada.

✗ B) Incorreta. Tópicos negados avaliam texto; a detecção de imagens nocivas é feita pelos filtros de conteúdo com a modalidade de imagem habilitada.

✗ D) Incorreta. Ataques de prompt tratam tentativas de burlar instruções; conteúdo sexual ou violento é coberto pelas categorias Sexual e Violence.

> **Dica de prova:** Imagem nociva em app multimodal = filtros de conteúdo com modalidade IMAGE no mesmo guardrail; tópicos negados e grounding não analisam a foto.

R3.3. B) Criar um guardrail com filtro de informações sensíveis em modo de mascaramento, com tipos de PII predefinidos e regex para o CPF, e exigi-lo pela condição bedrock:GuardrailIdentifier

Por quê: Os filtros de informações sensíveis do Amazon Bedrock Guardrails detectam tipos de PII predefinidos e padrões definidos por regex, com ação de bloquear ou mascarar em entradas e respostas. Para que a política valha em cada chamada, políticas do IAM com a condição bedrock:GuardrailIdentifier exigem que as inferências informem o guardrail aprovado, e as intervenções aparecem no trace e nas métricas do CloudWatch.

✗ A) Incorreta. O Macie descobre dados sensíveis em objetos armazenados no Amazon S3; não analisa tráfego em trânsito nem mascara prompts e respostas.

✗ C) Incorreta. Uma Lambda por aplicação não é uma política centralizada e trata só o prompt; além disso, a detecção de PII do Comprehend não aceita textos em português.

✗ D) Incorreta. O GuardDuty detecta ameaças e comportamentos suspeitos a partir de logs e eventos; não inspeciona o conteúdo de prompts nem mascara dados pessoais.

> **Dica de prova:** PII em prompts e respostas, inclusive em português: filtro de informações sensíveis do Guardrails; CPF por regex; uso obrigatório via IAM.

R3.4. C) Adicionar strict: true na definição da ferramenta e declarar additionalProperties false no inputSchema, para que o Bedrock restrinja a geração da chamada ao schema

Por quê: Structured outputs no Amazon Bedrock tem dois mecanismos: formato de saída por JSON Schema (outputConfig.textFormat na Converse) e strict tool use (strict: true na definição da ferramenta). No modo estrito, as chamadas de ferramenta seguem o schema de entrada, o que elimina campos inventados e valores fora do enum sem lógica de reparo. O schema precisa usar os recursos suportados do JSON Schema, como additionalProperties false.

✗ A) Incorreta. Temperatura baixa e exemplos melhoram a aderência, mas não garantem conformidade; campos extras e valores fora do enum continuam possíveis.

✗ B) Incorreta. A verificação de grounding avalia se a resposta está apoiada na fonte e é relevante para a pergunta; ela não valida nomes de campos nem valores de enum de uma chamada de ferramenta.

✗ D) Incorreta. Com toolChoice auto o modelo pode nem chamar a ferramenta, e o texto livre piora o problema de contrato com o serviço de destino.

> **Dica de prova:** Modelo inventando campos em chamadas de ferramenta: strict: true no toolSpec. Prompt, temperatura e guardrail reduzem, mas não garantem o schema.

04

CAPÍTULO 4 · 12% DA PROVA

Domínio 4: Eficiência operacional e otimização de aplicações de IA generativa

Com 12% da prova, este domínio cobra três coisas: reduzir custo de tokens, reduzir latência e montar observabilidade útil. Preciso saber escolher entre prompt caching, cache exato, cache semântico e cache de borda, entre sob demanda, Provisioned Throughput, lote, Flex e Priority, e ler as métricas certas (TimeToFirstToken, InvocationThrottles, tokens por minuto no pico). Os enunciados costumam oferecer 'comprar capacidade' como distrator para um problema que é de tokens demais. Onde mais erro é usar CloudTrail ou métricas agregadas do Bedrock para algo que precisa de conteúdo, conversa ou tenant: aí a resposta é requestMetadata + Model Invocation Logging + Logs Insights.

Neste capítulo
16 tópicos-chave · 12 pegadinhas · 6 macetes · 13 gatilhos · 4 questões comentadas

Tópicos-chave

Prompt caching

Reaproveita o processamento de um prefixo longo e repetido com pergunta variável, reduzindo custo e latência da entrada. Cada chamada ainda gera a resposta e cobra os tokens de saída, e os tokens em cache continuam ocupando o contexto. Pergunta repetida pede cache de respostas.

Cache exato, semântico e de borda

Entrada idêntica repetida: cache exato por hash determinístico, compartilhado entre instâncias (ElastiCache). Pergunta parecida com texto variável: cache semântico. Resposta igual para todos e estável: cache de borda no CloudFront.

Controle de tokens

Custo subindo mais que o volume: olhe tokens por requisição de entrada e saída antes de mexer em capacidade. Histórico crescendo: janela dos últimos turnos + resumo dos antigos + CountTokens. Saída desperdiçada: maxTokens calibrado pelo uso real. RAG caro: menos trechos com reranker e prompt de sistema enxuto em cache.

Service tiers: Priority, Flex e Reserved

Mesmo modelo, cargas com urgência diferente, sem reserva: Priority para o crítico e Flex (desconto com tempo de processamento maior) para o tolerante, escolhidos por requisição. Reserved exige reservar capacidade por um período definido.

Inferência em lote do Bedrock

Milhões de itens, sem pressa e menor custo, sem orquestrar chamadas: lote com entrada e saída no S3. Pode levar horas e exige número mínimo de registros, então não serve para pedido individual ou interativo. Para um formato só entre vários modelos: modelInvocationType Converse.

Provisioned Throughput e carga mista

Compensa com uso alto e constante; dimensionado para o pico fica ocioso. Carga base previsível + picos: Provisioned na base e sob demanda no excedente. Capacidade provisionada não reduz tokens nem latência de geração.

performanceConfig latency optimized

Reduz latência no mesmo modelo, sem reservar capacidade, nos modelos compatíveis. Provisioned Throughput garante vazão, não a versão otimizada para latência.

Dimensionamento do Bedrock

Use tokens de entrada e de saída por minuto no pico; cotas e model units são por minuto. Média diária e contagem de requisições enganam, e a projeção de custo é tokens médios por tipo de interação x volume x preço por token.

Métricas AWS/Bedrock

Tokens, latência e throttling por ModelId sem código. Throttling é InvocationThrottles, não InvocationClientErrors. Latência de streaming se compara por TimeToFirstToken e tokens de saída por segundo, não pela média de InvocationLatency.

Custo por conversa, funcionalidade ou tenant

requestMetadata + tokens dos registros de invocação + Logs Insights; Cost Explorer e métricas agregadas não enxergam conversa. Custo por aplicação no Cost Explorer: perfil de inferência de aplicação com tags. Rateio sem mudar código, uma role por equipe: tags no principal do IAM ativadas para alocação de custos.

Detecção de anomalia

Métrica de tokens com padrão diário ou semanal: alarme com detecção de anomalia do CloudWatch, não limite fixo. Gasto anormal considerando sazonalidade: AWS Cost Anomaly Detection, que usa faturamento e chega com atraso (não avisa em minutos).

Tracing ponta a ponta

Código novo: ADOT/OpenTelemetry enviando ao X-Ray, um span por etapa. Agente: traces OpenTelemetry (ADOT SDK) + CloudWatch Transaction Search + painel GenAI Observability. Bedrock Flows: enableTrace mostra entrada, saída e condições de cada nó.

Monitoramento de qualidade em produção

Infra verde e resposta errada: meça alucinação e drift contra golden set reexecutado, com métricas de faithfulness/correctness no CloudWatch e alarme. Detectar falha sem tráfego: canário do CloudWatch Synthetics com uma conversa de teste. Indicador de negócio: métrica personalizada (EMF) com dimensões no painel.

Vector store eficiente

HNSW: m e ef_construction exigem reindexar; ef_search ajusta recall x latência na consulta. OpenSearch Serverless: teto de custo é a capacidade máxima de OCUs (indexação e busca); mínimo em zero só corta ociosidade; monitore latência de busca e SearchOCU frente ao máximo.

Endpoints e escala

LLM com picos e ociosidade: inference components + target tracking por concorrência + escala a zero. Escolher instância por custo e latência com tráfego próprio: Inference Recommender, job Advanced. EKS com vários modelos: um deployment por modelo, HPA por métrica de inferência.

Otimização de chamadas

Cliente do SDK fora do handler (Provisioned Concurrency não resolve se o cliente é criado dentro). Subtarefas independentes: Parallel ou Map no Step Functions. ThrottlingException com fila: limite a concorrência máxima do event source mapping do SQS. Modelo pequeno primeiro e escalonamento para o grande quando a verificação falha.

Macetes para memorizar

💡 Escolha de cache

Prefixo igual = prompt caching; pergunta igual = hash; pergunta parecida = semântico; resposta de todos = CloudFront.

💡 Tiers

Priority corre, Flex espera com desconto, lote dorme no S3.

💡 Custo de LLM

Antes de comprar capacidade, conte tokens.

💡 Métrica de throttling

Estrangulou? InvocationThrottles. ClientErrors é outra coisa.

💡 HNSW

m e ef_construction nascem com o índice; ef_search muda a cada consulta.

💡 Streaming

Usuário sente o primeiro token: meça TimeToFirstToken.

Tabela de memorização: não confunda

Quando é...	...e quando é
Prompt caching Prefixo longo e repetido com pergunta variável.	**Cache semântico** Muitas perguntas reformuladas com a mesma resposta; com perguntas sempre diferentes ele raramente acerta.
Flex Chamadas síncronas tolerantes a atraso, com desconto.	**Priority** Latência crítica sem reservar capacidade.
Inferência em lote Milhões de itens sem pressa e sem orquestrar chamadas.	**Flex** Chamadas síncronas individuais que toleram latência.
Alarme de detecção de anomalia do CloudWatch Métrica operacional (tokens, invocações) com padrão por hora e dia, alerta em minutos.	**AWS Cost Anomaly Detection** Gasto anormal com sazonalidade, sobre faturamento (com atraso).
Métricas AWS/Bedrock Tokens, latência e throttling agregados por ModelId, sem código.	**Model Invocation Logging + Logs Insights** Conteúdo e tokens por requisição, conversa, tenant ou funcionalidade (com requestMetadata).
Perfil de inferência de aplicação com tags Custo por aplicação no Cost Explorer, aceitando trocar o modelId para o ARN do perfil.	**Tags no principal do IAM** Rateio por equipe sem mudar código, com uma role por equipe.

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Comprar Provisioned Throughput para reduzir custo de prompts enormes.
Como não cair: Vazão dedicada evita throttling, mas cada chamada continua processando os mesmos tokens. Reduza tokens: RAG com trechos relevantes, resumo, cache.

⚠ **Alerta de prova:** Ativar prompt caching para resolver estouro de contexto do histórico.
Como não cair: Tokens em cache continuam no contexto. Janela de turnos recentes + resumo + CountTokens.

⚠ **Alerta de prova:** Comparar latência de streaming pelo p90 de InvocationLatency.
Como não cair: InvocationLatency mede até o último token e mistura início com tamanho. Compare TimeToFirstToken e tokens de saída por segundo.

⚠ **Alerta de prova:** Dimensionar pela média diária de tokens dividida por 1.440 minutos.
Como não cair: Cotas e vazão são por minuto; a média esconde o pico. Use tokens por minuto no pico.

⚠ **Alerta de prova:** Usar Cost Anomaly Detection como alarme principal de pico de tokens.
Como não cair: Faturamento chega com atraso. Para minutos, alarme de detecção de anomalia do CloudWatch na métrica de tokens.

⚠ **Alerta de prova:** Contar chamadas no CloudTrail para custo por tenant ou alucinação.
Como não cair: O CloudTrail registra API e a role (a mesma para todos os tenants), não tokens nem qualidade. requestMetadata + Invocation Logging; qualidade com golden set.

⚠ **Alerta de prova:** Aumentar memória e timeout da Lambda para acabar com ThrottlingException.
Como não cair: Isso não muda a concorrência contra a cota. Limite a concorrência máxima do event source mapping.

⚠ **Alerta de prova:** Colocar mínimo de OCUs em zero para conter o estouro do orçamento no OpenSearch Serverless.
Como não cair: O mínimo só corta ociosidade. Teto é a capacidade máxima de OCUs de indexação e busca.

⚠ **Alerta de prova:** Aumentar o m do HNSW em produção para melhorar recall.
Como não cair: m é definido na criação e exige reindexar. Em tempo de consulta, ajuste ef_search.

⚠ **Alerta de prova:** Ativar Provisioned Concurrency para eliminar a latência antes da chamada ao modelo.
Como não cair: Se o cliente do SDK é criado dentro do handler, ele é recriado em toda invocação. Mova o cliente para fora do handler.

⚠ **Alerta de prova:** Criar perfil de inferência de aplicação para converter JSONL entre modelos no lote.
Como não cair: Perfil rastreia uso e custo. Formato único no lote é modelInvocationType Converse.

⚠ **Alerta de prova:** Usar inferência em lote para acelerar poucas chamadas independentes de um pedido interativo.
Como não cair: Lote leva horas e exige número mínimo de registros. Subtarefas independentes vão em Parallel ou Map no Step Functions.

Como cai na prova

📝 **Como cai:** Custo de LLM subiu; alternativas misturam capacidade, troca de modelo e redução de tokens.
Como resolver: Verifique tokens por requisição de entrada e saída; ataque o que cresce (histórico, contexto de RAG, saída). Capacidade só entra se o problema for throttling em carga constante.

📝 **Como cai:** Cargas com urgências diferentes no mesmo modelo.
Como resolver: Crítico em Priority, tolerante em Flex, massivo sem pressa em lote, base previsível em Provisioned com excedente sob demanda.

📝 **Como cai:** Observabilidade: o que medir e com qual serviço.
Como resolver: Agregado por modelo: AWS/Bedrock. Conteúdo e dimensão de negócio: Invocation Logging com requestMetadata e Logs Insights. Etapas: ADOT/X-Ray. Qualidade: golden set + métricas. Painel único: Managed Grafana.

📝 **Como cai:** Reduzir custo sem perder qualidade.
Como resolver: Experimento controlado: avalie candidatos no mesmo dataset, escolha o mais barato acima do limiar, compare saídas caso a caso e use modelo pequeno com escalonamento para o grande.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Prefixo longo repetido + pergunta variável	**Prompt caching**
Muitas perguntas reformuladas	**Cache semântico**
Resposta igual para todos e estável	**Cache de borda no CloudFront**
Latência subindo a cada turno	**Prompt crescendo: janela recente + resumo da sessão**
Custo por conversa, funcionalidade ou tenant	**requestMetadata + Model Invocation Logging + Logs Insights**
Um painel com métricas, logs e custo de fontes diferentes, sem operar servidor	**Amazon Managed Grafana**
Detectar falha sem tráfego de usuários e validar a resposta	**Canário do CloudWatch Synthetics**
Mesmo modelo, menos latência, sem reservar	**performanceConfig latency optimized**
Onde o agente perde tempo	**ADOT + Transaction Search + GenAI Observability**
Em qual passo do fluxo está o erro	**enableTrace do Bedrock Flows**
Escolher instância de endpoint com tráfego próprio	**Inference Recommender, job Advanced**

Extração, classificação, JSON	**Temperatura baixa (stop sequences e maxTokens controlam tamanho)**
Notas médias iguais entre duas versões	**Comparar as saídas caso a caso (diff) antes da troca**

🎯 **Se você lembrar de só uma coisa deste capítulo...**

- Capacidade provisionada não reduz tokens.
- Prompt caching barateia a entrada; a saída continua sendo cobrada.
- Dimensione por tokens por minuto no pico.
- CloudTrail não serve para custo por tenant nem para qualidade.

Essenciais para a prova (checklist)

- ☐ Sei escolher entre prompt caching, cache exato, cache semântico e cache de borda.
- ☐ Sei quando usar sob demanda, Provisioned Throughput, lote, Flex, Priority e latency optimized.
- ☐ Sei dimensionar o Bedrock por tokens por minuto no pico e projetar custo por token.
- ☐ Sei ler TimeToFirstToken, InvocationThrottles e métricas por ModelId.
- ☐ Sei medir custo por conversa ou tenant com requestMetadata e Logs Insights.
- ☐ Sei alocar custo por aplicação ou equipe com perfil de inferência de aplicação ou tags no principal do IAM.
- ☐ Sei montar tracing com ADOT, X-Ray, Transaction Search e enableTrace no Flows.
- ☐ Sei ajustar HNSW e limitar OCUs no OpenSearch Serverless.

Questões de revisão do capítulo

R4.1 · Domínio 4: Eficiência operacional e otimização de aplicações de IA generativa

Uma plataforma de educação vai lançar um tutor por voz que lê a resposta em voz alta enquanto ela é gerada, e a experiência só funciona se a fala começar rápido e o texto chegar mais depressa do que a leitura. Três modelos do Amazon Bedrock passaram nos testes de qualidade. Um teste anterior comparou a média de InvocationLatency de chamadas Converse e escolheu o modelo que gerava respostas mais curtas, que depois se mostrou lento para começar a falar. Qual abordagem de benchmark atende a esse objetivo?

A. Chamar os três modelos pela ConverseStream com prompts reais do tutor e comparar TimeToFirstToken e os tokens de saída por segundo, calculados com OutputTokenCount e a latência
B. Comparar a métrica EstimatedTPMQuotaUsage dos três modelos durante um teste de carga e escolher o que consumir menos cota por minuto, que tende a responder mais depressa
C. Repetir o teste com a Converse e comparar o p90 de InvocationLatency, normalizando o resultado pelo número médio de tokens de entrada de cada modelo para deixar a comparação justa
D. Escolher o modelo com o menor número de parâmetros entre os três, porque o tamanho do modelo determina a latência do primeiro token independentemente do tamanho dos prompts

R4.2 · Domínio 4: Eficiência operacional e otimização de aplicações de IA generativa

Uma empresa de streaming opera uma aplicação de GenAI em que o Amazon Bedrock publica métricas e registros de invocação no CloudWatch, os microsserviços no Amazon EKS enviam métricas ao Amazon Managed Service for Prometheus e o custo diário por funcionalidade fica em tabelas consultáveis pelo Amazon Athena. Engenharia, finanças e produto querem um único painel com métricas, logs e custo lado a lado, acesso pelo login corporativo do IAM Identity Center e nenhum servidor de visualização para manter. Qual solução atende a esses requisitos?

A. Criar um workspace do Amazon Managed Grafana autenticado pelo IAM Identity Center, com fontes de dados CloudWatch, Amazon Managed Service for Prometheus e Athena no mesmo painel
B. Exportar métricas, logs e custos para o Amazon S3 a cada hora e montar relatórios estáticos no Amazon Quick Sight, compartilhados por e-mail com as três áreas
C. Instalar o Grafana de código aberto em um cluster do Amazon EKS com volumes persistentes, plugins das três fontes de dados e integração SAML com o provedor de identidade corporativo
D. Criar um painel do CloudWatch com widgets de métricas e de Logs Insights, copiando para ele as métricas do Prometheus e os custos do Athena por funções Lambda agendadas

R4.3 · Domínio 4: Eficiência operacional e otimização de aplicações de IA generativa

Uma software house oferece um assistente de geração de código baseado no Amazon Bedrock que recebe milhares de requisições por minuto. Cada requisição envia ao modelo todo o contexto: histórico, descrição completa do repositório e uma longa lista de exemplos, o que gera prompts enormes e alto consumo de tokens. Muitas perguntas são reformulações umas das outras, e os usuários reclamam de respostas diferentes para pedidos equivalentes. Qual abordagem oferece a MELHOR combinação de custo, latência e consistência, mantendo a qualidade?

A. Comprar Provisioned Throughput para o modelo atual e manter o contexto completo em cada requisição, garantindo vazão constante e preservando toda a informação do repositório
B. Usar cache semântico com limiar de similaridade e mover a documentação e os exemplos para uma base vetorial, injetando por RAG só os trechos relevantes
C. Habilitar logs detalhados de prompts no CloudWatch Logs e trocar para um modelo com janela de contexto maior, evitando truncamento do histórico
D. Reduzir o limite de tokens de saída e manter no histórico só a última interação, preservando o restante do contexto enviado em cada chamada

R4.4 · Domínio 4: Eficiência operacional e otimização de aplicações de IA generativa

Uma rede de farmácias vai lançar no aplicativo um assistente de bulas com um modelo do Amazon Bedrock e precisa decidir quantas model units de Provisioned Throughput comprar e se as cotas sob demanda seriam suficientes. O piloto mostrou cerca de 1.800 tokens de entrada e 350 de saída por pergunta, e a previsão de pico é de 400 perguntas por minuto nas noites de campanha, contra 60 por minuto na média do dia. Como a equipe deve dimensionar a capacidade?

A. Pelos tokens por minuto de entrada e de saída no pico (perguntas de pico vezes os tokens médios de cada tipo), comparados com a vazão por model unit ou com a cota de TPM
B. Pelos tokens de entrada no pico, já que os tokens de saída não contam para a cota de tokens por minuto nem para a vazão contratada da model unit
C. Pelo total diário de tokens dividido por 1.440 minutos, porque as cotas e as model units são calculadas sobre o consumo médio de um dia inteiro
D. Pela média diária de requisições por minuto, já que a vazão do Provisioned Throughput é medida em requisições e os picos das noites de campanha se compensam ao longo do dia

Respostas comentadas

R4.1. A) Chamar os três modelos pela ConverseStream com prompts reais do tutor e comparar TimeToFirstToken e os tokens de saída por segundo, calculados com OutputTokenCount e a latência

Por quê: Para interfaces em streaming, a latência percebida tem duas partes: o tempo até o primeiro token (prefill, ligado ao tamanho da entrada) e a velocidade de decodificação, em tokens de saída por segundo. O Amazon Bedrock publica TimeToFirstToken para ConverseStream e InvokeModelWithResponseStream, e a documentação mostra que OTPS = OutputTokenCount / (InvocationLatency − TimeToFirstToken). O benchmark deve usar prompts reais e a mesma API de streaming que irá para produção.

✗ B) Incorreta. EstimatedTPMQuotaUsage é uma estimativa de consumo de cota, não de latência; um modelo pode consumir pouca cota e ainda demorar para emitir o primeiro token.

✗ C) Incorreta. InvocationLatency mede até o último token e mistura o tempo de início com o tamanho da resposta; normalizar por tokens de entrada não separa as duas componentes que importam para a voz.

✗ D) Incorreta. O número de parâmetros é só um indicativo: a latência real depende da infraestrutura, do tamanho do prompt e da otimização de cada modelo, e só é conhecida com medição.

> **Dica de prova:** Latência de streaming: compare TimeToFirstToken e tokens de saída por segundo, não a média de InvocationLatency.

R4.2. A) Criar um workspace do Amazon Managed Grafana autenticado pelo IAM Identity Center, com fontes de dados CloudWatch, Amazon Managed Service for Prometheus e Athena no mesmo painel

Por quê: Observabilidade unificada de GenAI: o Amazon Managed Grafana conecta várias fontes (CloudWatch, Prometheus gerenciado, Athena, X-Ray, OpenSearch) e mostra métricas técnicas, logs e custo no mesmo painel, com login corporativo e sem servidores para operar.

✗ B) Incorreta. Relatórios estáticos por e-mail não oferecem painel interativo com métricas e logs atuais lado a lado, e a exportação horária é mais uma rotina para operar.

✗ C) Incorreta. Atende funcionalmente, mas exige operar cluster, armazenamento, atualizações e alta disponibilidade do Grafana, contrariando a restrição de nada para manter.

✗ D) Incorreta. Copiar métricas e custos por funções próprias cria um pipeline para manter, com atraso e duplicação de dados, só para caber em um painel do CloudWatch.

> **Dica de prova:** Um painel com métricas, logs e custo de fontes diferentes, sem operar servidor: Amazon Managed Grafana com várias fontes de dados.

R4.3. B) Usar cache semântico com limiar de similaridade e mover a documentação e os exemplos para uma base vetorial, injetando por RAG só os trechos relevantes

Por quê: O cache semântico guarda pares de pergunta e resposta com o embedding da pergunta; quando um novo pedido fica acima de um limiar de similaridade, a resposta salva é devolvida sem chamar o modelo, o que reduz custo e latência e dá a mesma resposta a pedidos equivalentes. Mover contexto estático extenso para uma base vetorial e recuperar só os trechos relevantes por RAG diminui os tokens de entrada de cada chamada restante.

✗ A) Incorreta. Provisioned Throughput garante capacidade, mas não reduz os tokens processados nem as chamadas redundantes, e não trata a inconsistência entre pedidos equivalentes.

✗ C) Incorreta. Logs ajudam a observar, mas um contexto maior processa ainda mais tokens por chamada, elevando custo e latência sem resolver a inconsistência.

✗ D) Incorreta. Cortar saída e histórico reduz um pouco o consumo, mas perde contexto útil e mantém o principal desperdício, que é reenviar o contexto estático extenso; a inconsistência continua.

> **Dica de prova:** Muitas perguntas reformuladas: cache semântico. Contexto estático enorme em cada prompt: RAG com só os trechos relevantes. Capacidade provisionada não reduz tokens.

R4.4. A) Pelos tokens por minuto de entrada e de saída no pico (perguntas de pico vezes os tokens médios de cada tipo), comparados com a vazão por model unit ou com a cota de TPM

Por quê: Capacity planning de GenAI é feito em tokens por minuto, não em requisições. Uma model unit de Provisioned Throughput entrega uma vazão definida de tokens de entrada e de saída por minuto, e as cotas sob demanda do Bedrock também são em tokens e requisições por minuto (o max_tokens é reservado no início da requisição). Estime o pico multiplicando as requisições por minuto pelos tokens médios de entrada e de saída, com margem.

✗ B) Incorreta. Tokens de saída contam: a model unit especifica vazão de saída, e na cota sob demanda a saída é descontada, inclusive com a reserva do max_tokens no início da requisição.

✗ C) Incorreta. Cotas e vazão são por minuto; a média diária esconde o pico de 400 perguntas por minuto, que é justamente quando a capacidade falta.

✗ D) Incorreta. A vazão de uma model unit é definida em tokens de entrada e de saída por minuto, não em requisições, e dimensionar pela média deixa o pico de campanha sem capacidade, com throttling.

> **Dica de prova:** Dimensionar o Bedrock: tokens de entrada e de saída por minuto no pico. Média diária e contagem de requisições enganam.

05

CAPÍTULO 5 · 11% DA PROVA

Domínio 5: Testes, validação e solução de problemas

É o menor domínio (11%), mas as questões são diagnósticas: eu recebo um sintoma (resposta cortada, ValidationException, busca irrelevante, documento novo invisível) e preciso achar a causa exata antes de escolher a correção. A outra metade é avaliação: como comparar modelos e prompts com o mesmo dataset e juiz, quando usar avaliação humana, retrieve-only ou retrieve-and-generate, e como bloquear regressão no pipeline. Onde mais erro é responder com tentativa e erro (mudar prompt, temperatura, mais trechos) em vez de isolar a etapa com a avaliação certa. Também confundo métricas: correspondência exata, ROUGE e BLEU subestimam respostas corretas com outra redação.

Neste capítulo
16 tópicos-chave · 12 pegadinhas · 5 macetes · 12 gatilhos · 4 questões comentadas

Tópicos-chave

stopReason e resposta cortada

stopReason max_tokens indica que a resposta atingiu o limite de saída: aumente o maxTokens ou peça concisão; stop_sequence é outra causa. Resposta cortada sem exceção: olhe stopReason, tokens e o timeout de integração do API Gateway, com traces ponta a ponta.

ReadTimeoutError

Após cerca de 60 s sem throttling é o timeout de leitura do cliente: aumente o read_timeout do SDK e controle os retries. Falta de memória daria outro erro, não timeout do cliente HTTP.

ValidationException

Requisição inválida: repetir o mesmo payload falha de novo. Em conversa longa, entrada grande pede resumir ou podar o histórico (maxTokens é saída; cache e streaming não aumentam a janela). Com anexo, confira as regras de Message: documento exige bloco text e anexos só no papel user.

Avaliação retrieve-only

Mede só a recuperação (Retrieve + perguntas rotuladas com fontes esperadas, precision/recall@k e latência). Queda aqui aponta chunking e embeddings; geração, temperatura e guardrail não entram nessa métrica.

Avaliação retrieve-and-generate

Mede a resposta com o contexto (faithfulness, correctness). Recuperação boa e resposta ruim: ajuste o prompt de geração. Isolar as etapas: retrieve-only primeiro, depois geração com contexto fixo.

LLM-as-a-judge no Bedrock Model Evaluation

Avaliação semântica automática, usada como gate no pipeline. Compare modelos com mesmo dataset, mesmo juiz e mesmas métricas, uma avaliação por tarefa, sem misturar tarefas numa média. Conjunto próprio (prompt + referenceResponse) quando é preciso dado do domínio.

Avaliação humana do Bedrock

Para critério subjetivo que só especialistas da casa julgam: equipe própria e métricas customizadas. Notas em respostas já existentes sem reprocessar: bring your own inference responses. Métrica computada e repetível não pede humano.

Validador determinístico

Erro com critério exato (rota, parâmetro, schema) se testa contra o contrato, não com juiz LLM. JSON fora do formato sem saída estruturada: validação por JSON Schema e retry de correção com o erro, com limite e fila de revisão.

Métricas de similaridade

Correspondência exata, ROUGE e BLEU penalizam paráfrases corretas. Regressão em que só a redação muda: similaridade de embeddings contra a referência; consistência entre paráfrases: similaridade semântica; relevância: LLM-as-a-judge.

Lost in the middle

Modelo acerta quando o trecho está no começo ou no fim e erra no meio: reduza e reordene os trechos (reranking), não aumente o contexto. Context relevance baixa com a resposta escondida no trecho = chunk grande demais; reduza o chunk, não aumente k.

Diagnóstico de RAG

Documento novo invisível: confira primeiro se houve StartIngestionJob depois do upload. Busca sem erro mas irrelevante após mudança: consulta com modelo de embeddings diferente da indexação. Recuperação piorou só para conteúdo novo: suspeite do modelo de embeddings e prove com retrieve-only.

Testes de regressão e implantação segura

Gate de métricas no pipeline bloqueia regressão antes do usuário. Problema que só aparece com dados reais: shadow testing + dataset de regressão. Versão nova com pequena parcela e rollback por alarmes: blue/green CANARY com AutoRollbackConfiguration (ALL_AT_ONCE manda tudo de uma vez).

A/B de prompts

Muitas amostras por caso, tudo registrado e decisão com intervalo de confiança; temperatura 0 não elimina a variação em todos os modelos. Regressão após mudança de prompt: compare versões do Prompt Management no mesmo golden set, um fator por vez.

Observabilidade para diagnóstico

Agente inconsistente entre sessões: trace do agente passo a passo; logs de invocação não mostram a orquestração. No X-Ray, anotação é filtrável (versão do prompt, tenant) e metadado só guarda detalhe. Padrões em prompts e respostas: Logs Insights sobre o Model Invocation Logging.

AgentCore Evaluations

Avaliação online de agente em produção (objetivo atingido, ferramenta certa) com avaliadores integrados. O Model Evaluation julga pares de prompt e resposta em lote, não a trajetória do agente.

AccessDenied em perfil cross-Region

AccessDenied citando foundation-model de outra Região: libere o modelo em todas as Regiões do perfil e revise a SCP. É erro de autorização: backoff e retry não resolvem.

Macetes para memorizar

Isolar o RAG

Primeiro busca, depois fala: retrieve-only antes de retrieve-and-generate.

Quem julga

Exato = código; semântico = juiz LLM; gosto da casa = humano.

maxTokens

maxTokens é saída: não ajuda entrada grande.

Comparação justa

Mesmo dataset, mesmo juiz, mesma métrica, um fator por vez.

Contexto

Meio do contexto é ponto cego: menos e melhor ordenado.

Tabela de memorização: não confunda

Quando é...	...e quando é
Retrieve-only Suspeita na recuperação (chunking, embeddings, filtros); mede precision/recall@k.	**Retrieve-and-generate** Recuperação boa e resposta ruim ou alucinação; mede faithfulness e correctness.
LLM-as-a-judge Qualidade semântica automática, repetível e como gate no pipeline.	**Avaliação humana com equipe própria** Critério subjetivo que só especialistas da casa sabem julgar.
Validador determinístico Critério exato: rota, parâmetro, schema, código que precisa rodar em sandbox com testes.	**Juiz LLM** Relevância, utilidade e outros critérios semânticos.
Blue/green CANARY Expor só uma pequena parcela primeiro, com rollback automático por alarmes.	**Blue/green ALL_AT_ONCE** Mandar todo o tráfego à versão nova de uma vez.
Anotação no X-Ray Precisa filtrar traces por um valor (versão do prompt, tenant).	**Metadado no X-Ray** Só guardar detalhe no trace, sem filtrar.
Shadow testing Validar com dados reais sem afetar usuários.	**A/B em produção** Comparar variantes com amostras grandes e intervalo de confiança, aceitando expor usuários.

Números que caem

Item	Valor
ReadTimeoutError típico do SDK sem throttling	**cerca de 60 s**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Reduzir a pergunta do usuário quando stopReason é max_tokens.
Como não cair: max_tokens é o limite de saída configurado, não falta de espaço na janela. Aumente o maxTokens ou peça concisão.

⚠ **Alerta de prova:** Trocar Converse por ConverseStream para caber uma entrada grande.
Como não cair: Streaming muda só a entrega da resposta; a entrada segue o mesmo limite de contexto. Resuma ou pode o histórico.

⚠ **Alerta de prova:** Mandar o documento em additionalModelRequestFields.
Como não cair: Esse campo é para parâmetros de inferência específicos do modelo. Documento vai em Message, com bloco text, no papel user.

⚠ **Alerta de prova:** Culpar o modelo de geração por queda em avaliação retrieve-only.
Como não cair: O gerador não participa dessa métrica. Olhe chunking e embeddings.

⚠ **Alerta de prova:** Aumentar numberOfResults ou ligar reranker para PDF digitalizado com chunks vazios.
Como não cair: Não há texto a recuperar. É parsing: BDA ou FM como parser e nova sincronização.

⚠ **Alerta de prova:** Suspeitar de capacidade de OCUs quando o documento novo não aparece.
Como não cair: Falta de capacidade apareceria como lentidão ou falha na ingestão. Primeiro confira se houve StartIngestionJob.

⚠ **Alerta de prova:** Fixar seed e temperatura 0 quando o mesmo modelo se comporta diferente em dois ambientes.
Como não cair: Se os parâmetros já são iguais, a diferença está no payload. Compare o payload completo (exemplos e contexto).

⚠ **Alerta de prova:** Usar ROUGE, BLEU ou igualdade exata como gate de qualidade.
Como não cair: Penalizam respostas corretas com outra redação. Use similaridade semântica e LLM-as-a-judge.

⚠ **Alerta de prova:** Fazer fine-tuning do gerador para corrigir bloqueio indevido do classificador de segurança.
Como não cair: O classificador é componente separado. Calibre-o com dados rotulados por humanos e métricas.

⚠ **Alerta de prova:** Repetir com backoff um AccessDeniedException intermitente.
Como não cair: Falta de capacidade gera ThrottlingException; AccessDenied é autorização. Libere o modelo nas Regiões do perfil e revise a SCP.

⚠ **Alerta de prova:** Usar SageMaker Model Monitor para comparar duas versões num conjunto fixo.
Como não cair: Ele observa produção depois do deploy e está fechado a novos clientes. Use gate de avaliação no pipeline com o mesmo golden set.

⚠ **Alerta de prova:** Trocar o chunking para semântico quando há alucinação só em produção com throttling na busca.
Como não cair: Em staging a recuperação funciona; em produção a busca falha. Retry com backoff na recuperação e fallback para contexto vazio.

Como cai na prova

📝 **Como cai:** Mensagem de erro ou sintoma técnico (stopReason, ReadTimeoutError, ValidationException, AccessDenied).
Como resolver: Leia o erro literalmente: limite de saída, timeout do cliente, payload inválido ou autorização. Elimine alternativas de retry para erros não transitórios e de mais capacidade para erros de configuração.

📝 **Como cai:** Qualidade de RAG caiu e não se sabe a etapa.
Como resolver: Rode retrieve-only; se a recuperação está boa, avalie geração com contexto fixo (faithfulness). Verifique ingestão (StartIngestionJob, parsing) e coerência do modelo de embeddings.

📝 **Como cai:** Escolher método de avaliação para um critério.
Como resolver: Critério exato: validador determinístico. Critério semântico automatizável: LLM-as-a-judge. Critério subjetivo de especialista: avaliação humana. Agente em produção: AgentCore Evaluations.

📝 **Como cai:** Evitar que a regressão chegue ao usuário.
Como resolver: Gate de métricas no pipeline com golden set, shadow testing com dados reais, implantação CANARY com rollback por alarmes.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
stopReason max_tokens	**Aumentar maxTokens ou pedir concisão**
ReadTimeoutError após cerca de 60 s	**read_timeout do SDK**
Documento novo no S3 e invisível na resposta	**StartIngestionJob depois do upload**
Busca sem erro mas irrelevante após mudança	**Mesmo modelo de embeddings na consulta e na indexação**
Acerta no começo e no fim, erra no meio	**Menos trechos e reranking**
Modelo ignora o contexto do RAG	**Grounding contextual do Guardrails com limiares**
Busca vazia gera alucinação	**Corrigir o que zerou a recuperação + instruir a dizer que não sabe**
Notas de especialistas em respostas já dadas	**Avaliação humana com bring your own inference responses**
Código gerado precisa ser avaliado	**Executar testes, lint e segurança em sandbox com prompts fixos**

Painel de negócio sobre resultados de avaliação no S3	**Glue Data Catalog + Athena + Amazon Quick Sight**
Feedback útil para curadoria	**ID por resposta em requestMetadata gravado junto com o voto**
Deriva em conversas longas	**Regressão com diálogos completos + histórico com resumos**

🎯 Se você lembrar de só uma coisa deste capítulo...

- max_tokens no stopReason = limite de saída; maxTokens não aumenta a janela de entrada.
- Retrieve-only isola a recuperação; retrieve-and-generate avalia a resposta.
- Documento novo invisível: confira o StartIngestionJob antes de qualquer outra coisa.
- Igualdade exata, ROUGE e BLEU subestimam GenAI.

Essenciais para a prova (checklist)

- ☐ Sei diagnosticar resposta cortada pelo stopReason e pelo timeout do API Gateway.
- ☐ Sei tratar ReadTimeoutError, ValidationException e AccessDenied sem retry inútil.
- ☐ Sei separar avaliação retrieve-only e retrieve-and-generate.
- ☐ Sei escolher entre LLM-as-a-judge, avaliação humana e validador determinístico.
- ☐ Sei comparar modelos e prompts de forma justa (mesmo dataset, juiz e métricas, um fator por vez).
- ☐ Sei montar gate de regressão, shadow testing e implantação CANARY com rollback.
- ☐ Sei investigar agentes com traces e usar anotações do X-Ray.
- ☐ Sei por que métricas lexicais falham em respostas geradas.

Questões de revisão do capítulo

R5.1 · Domínio 5: Testes, validação e solução de problemas

Uma consultoria gera relatórios de auditoria com a Converse API do Amazon Bedrock. Vários relatórios terminam no meio de uma frase, e o campo stopReason da resposta traz max_tokens. O prompt e a pergunta estão bem abaixo do limite de contexto do modelo. Qual ajuste resolve o problema?

A. Aumentar o maxTokens em inferenceConfig ou pedir ao modelo um relatório mais conciso, dentro do limite de saída
B. Remover as stop sequences configuradas na chamada, que estão interrompendo a geração antes do fim do relatório
C. Reduzir o tamanho da pergunta enviada pelo usuário, liberando espaço para a resposta dentro da janela de contexto
D. Trocar a Converse pela ConverseStream, que entrega a resposta em partes e não tem o mesmo limite de saída

R5.2 · Domínio 5: Testes, validação e solução de problemas

Um jornal resume notas de agências de notícias com um modelo do Amazon Bedrock. O template junta, num único bloco de texto corrido, as instruções de estilo, uma nota de exemplo com o resumo esperado e a nota do dia. Em parte das respostas, o resumo mistura fatos da nota de exemplo com os da nota real ou repete frases das instruções. Os resumos não estão sendo cortados, e a equipe não pode trocar de modelo. Qual ajuste ataca a causa?

A. Indexar as notas de agência numa base de conhecimento e recuperar a nota do dia com RetrieveAndGenerate, em vez de enviá-la junto do exemplo e das instruções no mesmo bloco de texto do prompt enviado ao modelo
B. Remover a nota de exemplo e aumentar a temperatura, para que o modelo produza resumos mais variados e menos presos ao texto do template usado pela redação
C. Separar instruções, exemplo e nota do dia em seções nomeadas e delimitadas, levar as instruções ao prompt de sistema, apontar a seção a resumir e proibir que o exemplo apareça na resposta
D. Aumentar o maxTokens da resposta, porque o resumo está sendo truncado e o modelo completa o texto com trechos do exemplo e das instruções de estilo

R5.3 · Domínio 5: Testes, validação e solução de problemas

Uma marca de cosméticos usa um FM do Amazon Bedrock para escrever descrições de produtos e quer escolher entre o modelo atual e um candidato. O critério decisivo é a aderência ao tom de voz da marca, algo que só os redatores sêniores sabem julgar e que não existe como métrica automática confiável. O comitê exige que esses redatores deem notas de 1 a 5 por critério às respostas dos dois modelos para as mesmas 300 solicitações, com os resultados consolidados em um relatório. Qual solução atende a esses requisitos com o MENOR esforço?

A. Publicar as respostas dos dois modelos para uma força de trabalho pública do Amazon SageMaker Ground Truth e pedir notas de 1 a 5 sobre o tom de voz da marca
B. Criar uma avaliação automática do Amazon Bedrock com a tarefa de geração de texto geral e as métricas de robustez e toxicidade, calculadas sobre as respostas às 300 solicitações
C. Criar uma avaliação do Amazon Bedrock com avaliadores humanos da própria equipe, métricas customizadas de tom de voz em escala Likert e os dois modelos como fontes
D. Criar uma avaliação com LLM-as-a-judge e as métricas internas de correção e completude, usando o modelo atual como avaliador das respostas do candidato e das suas próprias

R5.4 · Domínio 5: Testes, validação e solução de problemas

Uma prefeitura indexou em uma base de conhecimento do Amazon Bedrock um acervo de PDFs de leis municipais, parte deles digitalizada a partir de papel. O job de ingestão termina com sucesso, mas perguntas sobre as leis digitalizadas não recuperam trechos delas, e uma inspeção dos chunks mostra que esses arquivos geraram texto vazio. A fonte de dados usa o parser padrão e a estratégia de chunking padrão. Qual é a correção?

A. Criar uma fonte de dados no mesmo bucket com o parser do Amazon Bedrock Data Automation, que extrai o texto das páginas digitalizadas, sincronizá-la e remover a fonte com o parser padrão
B. Trocar a estratégia de chunking para semântica com um buffer maior, para que o modelo de embeddings encontre os limites das frases nas páginas digitalizadas do acervo
C. Aumentar o numberOfResults nas consultas e adicionar um modelo de reranking, para que os trechos das leis digitalizadas apareçam entre os resultados devolvidos à aplicação
D. Aumentar a dimensão dos vetores do Titan Text Embeddings V2 para 1.024 e sincronizar de novo a fonte de dados atual, para que as páginas digitalizadas ganhem uma representação vetorial mais rica no índice

Respostas comentadas

R5.1. A) Aumentar o maxTokens em inferenceConfig ou pedir ao modelo um relatório mais conciso, dentro do limite de saída

Por quê: O stopReason da Converse indica por que o modelo parou: end_turn (terminou), max_tokens (atingiu o limite de saída configurado), stop_sequence (encontrou uma sequência de parada), guardrail_intervened ou content_filtered, entre outros. Com max_tokens, a solução é aumentar o maxTokens em inferenceConfig, até o máximo do modelo, ou reduzir o tamanho esperado da resposta.

✗ B) Incorreta. Se uma stop sequence tivesse interrompido a geração, o stopReason seria stop_sequence, não max_tokens.

✗ C) Incorreta. A entrada já está bem abaixo do limite; max_tokens indica que a resposta atingiu o limite de saída configurado, não falta de espaço na janela.

✗ D) Incorreta. A ConverseStream respeita o mesmo maxTokens; a resposta seria cortada no mesmo ponto, só que entregue em partes.

> **Dica de prova:** Resposta cortada com stopReason max_tokens: aumente o maxTokens (ou peça concisão). stop_sequence seria outra causa.

R5.2. C) Separar instruções, exemplo e nota do dia em seções nomeadas e delimitadas, levar as instruções ao prompt de sistema, apontar a seção a resumir e proibir que o exemplo apareça na resposta

Por quê: Modelos prestam mais atenção às partes certas do prompt quando ele tem seções claramente delimitadas, com Markdown, XML ou outra estrutura, e quando as instruções se referem a essas seções pelo nome. Também é possível instruir o modelo a não reproduzir o conteúdo das seções de instruções e de exemplos na resposta.

✗ A) Incorreta. Recuperar a nota não muda a estrutura do prompt final, que continuaria misturando instruções, exemplo e conteúdo sem separação.

✗ B) Incorreta. Tirar o exemplo perde a padronização do resumo, e temperatura alta aumenta a chance de o modelo inventar ou misturar conteúdo.

✗ D) Incorreta. O enunciado diz que os resumos não estão sendo cortados; o problema é o modelo não distinguir as partes do prompt.

> **Dica de prova:** Resposta mistura exemplo, instruções e conteúdo: delimite e nomeie as seções do prompt. Não é caso de temperatura, maxTokens nem RAG.

R5.3. C) Criar uma avaliação do Amazon Bedrock com avaliadores humanos da própria equipe, métricas customizadas de tom de voz em escala Likert e os dois modelos como fontes

Por quê: Avaliação com humanos no Amazon Bedrock (bring your own work team): os avaliadores da empresa recebem as respostas de até duas fontes de inferência para o mesmo conjunto de prompts e dão notas em métricas customizadas, com métodos como Likert individual ou comparativo, botões de escolha, ranking ordinal ou polegar para cima e para baixo. É a opção para critérios subjetivos que exigem especialistas, como tom de voz, estilo e adequação à marca.

✗ A) Incorreta. Uma força de trabalho pública não conhece o tom de voz da marca, o que viola a exigência de que os redatores sêniores avaliem; além disso, o Ground Truth está fechado a novos clientes.

✗ B) Incorreta. Robustez e toxicidade são métricas computadas que não medem aderência ao tom de voz, e o comitê exige o julgamento dos redatores.

✗ D) Incorreta. LLM-as-a-judge automatiza o julgamento, mas o comitê exige notas dos redatores, e usar o modelo atual como juiz das próprias respostas introduz viés na comparação.

> **Dica de prova:** Critério subjetivo que só especialistas da casa sabem julgar: avaliação humana do Bedrock com equipe própria e métricas customizadas.

R5.4. A) Criar uma fonte de dados no mesmo bucket com o parser do Amazon Bedrock Data Automation, que extrai o texto das páginas digitalizadas, sincronizá-la e remover a fonte com o parser padrão

Por quê: Na ingestão de uma base de conhecimento, o parser vem antes do chunking e dos embeddings. O parser padrão do Bedrock só extrai texto de arquivos textuais; para PDFs digitalizados, imagens, tabelas e gráficos, use o Bedrock Data Automation ou um modelo de base como parser. Texto vazio nos chunks é sintoma de falha de pré-processamento, não de busca.

✗ B) Incorreta. O chunking atua sobre o texto já extraído; como o parser não extraiu texto das páginas digitalizadas, não há conteúdo para dividir, qualquer que seja a estratégia.

✗ C) Incorreta. Recuperar mais resultados ou reordená-los não cria chunks que não existem; os documentos digitalizados continuam sem texto no índice.

✗ D) Incorreta. A dimensão do embedding afeta a precisão da representação do texto, mas um texto vazio continua vazio; o problema está na etapa de parsing, antes da vetorização.

> **Dica de prova:** Chunks vazios em PDF digitalizado: problema de parsing. Use BDA ou FM como parser e sincronize de novo.

DIA DA PROVA

Estratégia de prova

A AIP-C01 é de nível profissional: cada enunciado descreve uma empresa, um sintoma ou requisito e uma ou duas restrições (menor esforço operacional, sem código, sem reservar capacidade, sem copiar dados, residência numa Região). Quase todas as alternativas 'funcionariam', mas só uma cumpre todas as restrições com o serviço gerenciado certo. A prova pune soluções que dependem do comportamento do modelo (instrução no prompt, temperatura, confiança autodeclarada) quando o requisito pede garantia, e pune atacar a camada errada (geração quando o problema é recuperação). Ela usa os serviços vigentes: AgentCore, Knowledge Bases e Bedrock Evaluations no lugar de Agents Classic, Kendra, Clarify, Model Monitor e A2I.

Como responder

✓ São 75 questões (65 pontuadas e 10 não pontuadas, sem identificação) em 180 minutos: controle o tempo para chegar ao fim e responder todas.

✓ O resultado é de 100 a 1000 com mínimo de 750 e o modelo é compensatório: foque nos Domínios 1 (31%) e 2 (26%), que somam mais da metade da prova.

✓ Leia primeiro a pergunta final e sublinhe as restrições (menor esforço, sem código, sem compromisso, sem copiar dados); elimine toda alternativa que viola uma delas.

✓ Em múltipla resposta, conte quantas o enunciado pede e avalie cada alternativa isoladamente contra o cenário.

✓ Identifique a camada do problema (ingestão, recuperação, geração, orquestração, segurança) antes de olhar as alternativas.

✓ Desconfie de alternativas com 'aumentar timeout', 'aumentar temperatura', 'mais trechos', 'instruir no prompt' ou 'comprar Provisioned Throughput': quase sempre são distratores.

✓ Entre duas corretas, prefira a nativa e gerenciada que atende sem código extra, a menos que o enunciado exija controle do runtime ou do índice.

✓ Não deixe nenhuma em branco: questão sem resposta conta como errada e não há penalidade por chute.

Erros que derrubam candidatos

✗ Escolher serviço fechado a novos clientes (Agents Classic, Kendra, Q Business, Clarify, Model Monitor, A2I, CloudTrail Lake) para solução nova.

✗ Tratar instrução no prompt ou guardrail como controle de acesso a dados.

✗ Confundir CloudTrail (quem chamou) com Model Invocation Logging (o que foi dito).

✗ Corrigir problema de recuperação com temperatura, janela maior ou fine-tuning.

✗ Usar Express Workflow para espera longa ou callback com task token.

✗ Comprar capacidade quando o custo vem de tokens demais por requisição.

✗ Liberar só InvokeModel e esquecer InvokeModelWithResponseStream nas políticas.

Na semana da prova

- ☐ Rever os cenários em que errou, entendendo por que cada alternativa errada falha, com prioridade nos Domínios 1 e 2.
- ☐ Revisar as tabelas de decisão: tipos de chunking, vector stores, formas de consumo do modelo, políticas do Guardrails, caches e tiers.
- ☐ Revisar os componentes do AgentCore (Runtime, Gateway, Identity, Memory, Policy, Browser, Code Interpreter) e quando usar MCP ou A2A.
- ☐ Revisar a lista de serviços fechados a novos clientes e seus substitutos.
- ☐ Fazer um simulado completo cronometrado em 180 minutos para treinar o ritmo.
- ☐ Na véspera, revisar só os blocos 'lembrar' de cada capítulo e descansar.

REVISÃO FINAL

Folha de cola

Leia na véspera e na manhã da prova: os gatilhos e as pegadinhas mais importantes de cada domínio.

1. Domínio 1: Integração de modelos de base, gestão de dados e conformidade

‣ Código de peça, SKU ou termo exato não encontrado → **Busca híbrida (overrideSearchType no Retrieve, sem reindexar)**

‣ Trecho certo recuperado, mas em posição ruim / contexto grande demais → **Reranker + menos trechos**

‣ PDF digitalizado gera contexto vazio → **OCR com Textract antes da ingestão, ou parsing com FM/BDA**

‣ Pergunta que depende de relações entre entidades de documentos diferentes → **GraphRAG com Neptune Analytics na Knowledge Base**

‣ Segunda busca depende do resultado da primeira → **Recuperação iterativa por agente com a busca como ferramenta; base gerenciada sem código: AgenticRetrieveStream**

⚠ Trecho menor aumenta a chance de separar a cláusula da exceção. A correção é sobreposição ou chunking hierárquico.

⚠ O reranker ordena por relevância, não por vigência, e o revogado continua no contexto. Filtro de metadados por data/status no Retrieve, ou tirar as versões antigas da fonte e sincronizar.

⚠ Dimensão não muda o que foi extraído do PDF nem o que entrou no embedding. Produto interno só equivale a cosseno com vetores normalizados; truncamento no limite de tokens do modelo de embeddings explica conteúdo do fim do trecho nunca recuperado.

2. Domínio 2: Implementação e integração

‣ Geração de minutos atrás do API Gateway, cliente aceita consultar depois → **StartExecution (ID de tarefa) + DescribeExecution (status)**

‣ Job externo de horas e Lambda fazendo polling → **.waitForTaskToken + SendTaskSuccess**

‣ Ferramentas que mudam com frequência e pertencem a outras equipes → **Descoberta dinâmica por cliente MCP**

‣ Lambda ou API OpenAPI existente como ferramenta MCP, sem escrever servidor → **Targets no AgentCore Gateway**

‣ OAuth em nome do usuário / token vault → **AgentCore Identity**

⚠ O aumento existe para APIs regionais, mas mantém conexões abertas por minutos e não dá estado nem histórico. Use o padrão assíncrono com ID de tarefa e endpoint de status.

⚠ Express dura no máximo cinco minutos e não suporta callback com task token.

⚠ A invocação assíncrona grava no S3 e serve para vídeo ou geração de minutos. Texto na tela conforme sai é ConverseStream.

3. Domínio 3: Segurança, proteção e governança de IA

‣ Campo de formulário injetando instruções via InvokeModel → **Tags de entrada do guardrail com tagSuffix aleatório**

‣ Instrução maliciosa dentro de página ou documento lido pelo agente → **ApplyGuardrail sobre o conteúdo da ferramenta + limitar ações**

‣ Modelo inventando campos em chamadas de ferramenta → **strict: true no toolSpec**

‣ Mesmas políticas para modelo fora do Bedrock → **ApplyGuardrail**

‣ Mesmo guardrail em toda a organização sem mudar código → **Política do Amazon Bedrock no AWS Organizations**

⚠ Instrução no prompt não é controle de acesso: injeção ou SQL errado ainda lê as tabelas. Role de menor privilégio + Lake Formation.

⚠ Tópicos negados avaliam texto. Imagem nociva é filtro de conteúdo com modalidade IMAGE.

⚠ Injeção escrita em linguagem neutra passa pelo filtro de violência. Use o filtro de ataques de prompt na entrada, com tags.

4. Domínio 4: Eficiência operacional e otimização de aplicações de IA generativa

‣ Prefixo longo repetido + pergunta variável → **Prompt caching**

‣ Muitas perguntas reformuladas → **Cache semântico**

‣ Resposta igual para todos e estável → **Cache de borda no CloudFront**

‣ Latência subindo a cada turno → **Prompt crescendo: janela recente + resumo da sessão**

‣ Custo por conversa, funcionalidade ou tenant → **requestMetadata + Model Invocation Logging + Logs Insights**

⚠ Vazão dedicada evita throttling, mas cada chamada continua processando os mesmos tokens. Reduza tokens: RAG com trechos relevantes, resumo, cache.

⚠ Tokens em cache continuam no contexto. Janela de turnos recentes + resumo + CountTokens.

⚠ InvocationLatency mede até o último token e mistura início com tamanho. Compare TimeToFirstToken e tokens de saída por segundo.

5. Domínio 5: Testes, validação e solução de problemas

‣ stopReason max_tokens → **Aumentar maxTokens ou pedir concisão**

‣ ReadTimeoutError após cerca de 60 s → **read_timeout do SDK**

‣ Documento novo no S3 e invisível na resposta → **StartIngestionJob depois do upload**

‣ Busca sem erro mas irrelevante após mudança → **Mesmo modelo de embeddings na consulta e na indexação**

‣ Acerta no começo e no fim, erra no meio → **Menos trechos e reranking**

⚠ max_tokens é o limite de saída configurado, não falta de espaço na janela. Aumente o maxTokens ou peça concisão.

⚠ Streaming muda só a entrega da resposta; a entrada segue o mesmo limite de contexto. Resuma ou pode o histórico.

⚠ Esse campo é para parâmetros de inferência específicos do modelo. Documento vai em Message, com bloco text, no papel user.

GABARITO

Respostas do diagnóstico

D1. C) Publicar versões aprovadas dos templates no Prompt Management e, na Lambda, chamar a Converse com o ARN da versão e promptVariables, gravando o histórico no DynamoDB → revise o capítulo 1

Por quê: O Prompt Management do Amazon Bedrock guarda templates com variáveis, configuração de inferência e versões imutáveis. A aplicação executa uma versão diretamente pela Converse, informando o ARN da versão no modelId e os valores em promptVariables; system prompt, toolConfig e inferenceConfig ficam definidos no próprio prompt. Gravar o ARN da versão junto com a resposta dá a rastreabilidade exigida pela auditoria.

✗ A) Incorreta. Com o prompt montado no cliente, o backend perde o controle do template aprovado, e qualquer usuário pode alterar o texto enviado ao modelo.

✗ B) Incorreta. Variáveis de ambiente exigem reconfigurar a função a cada mudança de template, não têm versionamento próprio de prompts e obrigam a escrever a renderização no código.

✗ D) Incorreta. O AppConfig versiona configuração, mas a renderização das variáveis fica no código, e registrar só o nome do template não prova qual versão gerou a resposta.

> **Dica de prova:** Prompt versionado e rastreável sem renderizar no código: Prompt Management + Converse com o ARN da versão.

D2. A) Executar uma PoC no Amazon Bedrock com laudos reais anonimizados, critérios de saída definidos antes (qualidade, latência e custo por laudo) e avaliação dos modelos candidatos → revise o capítulo 1

Por quê: Prova de conceito com critérios de saída mensuráveis: antes de aprovar a produção, define-se um conjunto de avaliação representativo e metas objetivas (qualidade, latência, custo por transação). O Amazon Bedrock permite testar vários modelos sob demanda e avaliá-los com Model Evaluation, medindo tokens e latência em cada chamada, o que gera a evidência pedida com pouca operação.

✗ B) Incorreta. Volume de uso não mede qualidade, latência nem custo por transação, e expor um modelo não avaliado a usuários antes dos critérios de saída inverte a ordem da validação.

✗ C) Incorreta. Provisioned Throughput cobra por hora e por unidade de modelo, e comprá-lo antes de escolher o modelo encarece a PoC; latência estável sozinha não prova qualidade nem custo por laudo.

✗ D) Incorreta. Fine-tuning antes de saber se um modelo de base já atende é caro e lento, e a perda de validação mede ajuste ao treino, não qualidade percebida, latência nem custo por laudo.

> **Dica de prova:** PoC de GenAI: dados reais (anonimizados), critérios definidos ANTES e comparação de modelos no Bedrock. Fine-tuning e capacidade provisionada vêm depois, se a evidência pedir.

D3. C) Inserir um nó de função Lambda entre os prompts para consultar a API e normalizar a data, e outro nó Lambda depois do segundo prompt para mascarar o número do pedido antes do nó de saída → revise o capítulo 1

Por quê: No Amazon Bedrock Flows, o nó de função Lambda envia ao código um evento com as entradas resolvidas pelas expressões e devolve a resposta da função como functionResponse para o próximo nó. Colocado entre prompts encadeados, ele faz enriquecimento e normalização; colocado depois, faz pós-processamento como mascaramento. O nó de código inline é útil para transformações simples, mas roda isolado e sem acesso à rede.

✗ A) Incorreta. O modelo não executa chamadas de API por conta própria; ele inventaria o status, e o mascaramento ficaria dependente de uma instrução probabilística.

✗ B) Incorreta. Um índice atualizado uma vez por dia devolve status desatualizado, e a busca semântica não é a forma confiável de obter o registro exato de um pedido.

✗ D) Incorreta. O nó de código inline roda em um ambiente isolado gerenciado pela AWS, sem acesso à internet nem à VPC da empresa, então não alcança a API interna; além disso, está em versão de pré-visualização.

> **Dica de prova:** Lógica de negócio ou chamada a sistema interno entre prompts encadeados: nó Lambda no Bedrock Flows.

D4. C) Trocar a Converse pela ConverseStream, repassar os eventos contentBlockDelta e conceder à role a permissão bedrock:InvokeModelWithResponseStream → revise o capítulo 2

Por quê: A ConverseStream retorna a resposta como um fluxo de eventos (messageStart, contentBlockDelta com o texto incremental, messageStop e metadata), o que reduz o tempo até o primeiro token percebido pelo usuário e evita conexões ociosas esperando gerações longas. Para chamá-la, a identidade precisa da ação bedrock:InvokeModelWithResponseStream (a Converse usa bedrock:InvokeModel). Aumentar o timeout do cliente resolve as falhas, mas não mostra o texto progressivamente.

✗ A) Incorreta. Evita o timeout, mas o analista continua esperando o relatório inteiro; não atende ao requisito de ver o texto à medida que é gerado.

✗ B) Incorreta. A invocação assíncrona do Bedrock atende a modelos e casos específicos com saída no S3 e não entrega texto progressivo; o analista seguiria esperando.

✗ D) Incorreta. A inferência em lote é para grandes volumes sem urgência, com conclusão em horas; o oposto de uma ferramenta interativa.

> **Dica de prova:** Texto longo em ferramenta interativa: ConverseStream (e permissão InvokeModelWithResponseStream); aumentar timeout só evita a falha.

D5. A) Dividir a tarefa em uma cadeia de prompts em que o primeiro extrai as cláusulas em JSON, o segundo classifica o risco a partir desse JSON e o terceiro redige o parecer, com validação da saída entre as etapas → revise o capítulo 2

Por quê: Prompt chaining quebra uma tarefa complexa em subtarefas, cada uma com instruções curtas e saída estruturada que vira entrada da seguinte. Isso reduz a variação, permite validar os resultados intermediários e localizar a etapa que falhou, sem trocar o modelo.

✗ B) Incorreta. Fine-tuning cria um modelo customizado, tem custo e prazo altos e continua sem permitir testar cada etapa de forma isolada.

✗ C) Incorreta. Mais tokens não resolve a sobrecarga de instruções, e temperatura maior aumenta a variação entre execuções, o oposto do pedido.

✗ D) Incorreta. Triplica o custo por contrato e não ajuda a descobrir em qual etapa o erro ocorre, já que o prompt continua monolítico.

> **Dica de prova:** Prompt gigante com várias tarefas e resultados inconsistentes: encadeie prompts com saída estruturada entre as etapas.

D6. B) Criar uma notificação de evento do S3 no prefixo Analysis/Voice/Redacted/ do bucket da instância para acionar uma Lambda que lê a transcrição mascarada, chama o Bedrock e grava os campos no CRM → revise o capítulo 2

Por quê: Integrar GenAI a um contact center reaproveita o que o serviço já produz: o Contact Lens grava no bucket da instância a transcrição analisada, inclusive a versão mascarada em Analysis/Voice/Redacted/. Um evento do S3 nesse prefixo aciona a função que chama o Bedrock com o template de campos da corretora e grava o resultado no CRM. O gatilho no arquivo evita a corrida entre o fim da chamada e o fim da análise.

✗ A) Incorreta. Duplica a transcrição e a redação que o Contact Lens já entrega, com pipeline e custo adicionais que a equipe quer evitar.

✗ C) Incorreta. A análise pós-contato é produzida depois do encerramento da chamada; no instante do DISCONNECTED o arquivo de transcrição pode ainda não existir.

✗ D) Incorreta. Exige operar a transcrição por conta própria em contêineres, justamente a infraestrutura que a equipe não quer manter.

> **Dica de prova:** Pós-chamada no Amazon Connect: dispare o resumo quando o arquivo do Contact Lens chegar ao S3, não no evento de desconexão.

D7. D) Definir mensagens genéricas de bloqueio para prompts e respostas no guardrail e devolver só esse texto ao cliente, mantendo a avaliação detalhada do trace nos logs internos → revise o capítulo 3

Por quê: Mensagens de bloqueio customizadas: todo guardrail define o texto devolvido quando bloqueia um prompt (blockedInputMessaging) ou uma resposta (blockedOutputsMessaging). Usar uma mensagem genérica evita dar ao atacante pistas sobre as regras, e o trace do guardrail, guardado só nos logs, preserva o detalhe para a equipe.

✗ A) Incorreta. Mascaramento existe no filtro de informações sensíveis para substituir PII por identificadores; ele não esconde o motivo do bloqueio e expor o identificador da política mantém o vazamento.

✗ B) Incorreta. Retirar o filtro de ataques de prompt enfraquece a defesa contra injection e jailbreak, e o vazamento continuaria se a aplicação seguisse expondo o nome da política.

✗ C) Incorreta. No modo de detecção o guardrail não bloqueia, então o conteúdo malicioso chega ao modelo e a resposta é entregue; a segurança quer bloquear com mensagem neutra.

> **Dica de prova:** Não revelar ao atacante qual regra bloqueou: mensagem genérica de bloqueio no guardrail ao usuário e trace detalhado só nos logs internos.

D8. D) Publicar como métricas do CloudWatch, com a dimensão tipo de sinistro, as pontuações de grounding e relevância do guardrail, alarmar sobre a média e revisar os resumos abaixo de um limiar → revise o capítulo 3

Por quê: Pontuações de confiança (como as de grounding e relevância da verificação de grounding contextual do Guardrails) podem virar métricas do CloudWatch com dimensões de negócio. Isso quantifica a incerteza, permite alarmes sobre tendências e define faixas: bloquear o que é claramente sem apoio e mandar à revisão humana a faixa intermediária.

✗ A) Incorreta. Com o limiar perto do máximo, quase todos os resumos seriam bloqueados, contrariando o requisito, e não haveria métrica de tendência por tipo de sinistro.

✗ B) Incorreta. Essa métrica conta intervenções, não mostra a distribuição das pontuações nem a separação por tipo de sinistro, e só cobre o que já foi bloqueado.

✗ C) Incorreta. A autoavaliação do modelo costuma ser mal calibrada em relação aos acertos e não usa a fonte como referência; é uma medida fraca de incerteza.

> **Dica de prova:** Quantificar incerteza e acionar revisão: publique a pontuação de confiança como métrica com dimensões e roteie a faixa baixa para humanos.

D9. A) Exigir saída estruturada com JSON Schema no outputConfig.textFormat da Converse, com campos de recomendação, resumo da justificativa e evidências com ID do documento e trecho, mantendo o prompt de sistema no backend → revise o capítulo 3

Por quê: Transparência em IA responsável não exige revelar o prompt: um resumo do raciocínio e as evidências usadas, em formato estruturado e validado por esquema, explicam a decisão sem expor instruções internas.

✗ B) Incorreta. Texto livre não garante formato nem que as regras internas fiquem de fora; a interface teria de interpretar parágrafos variáveis.

✗ C) Incorreta. O raciocínio bruto é longo, sem formato fixo e pode reproduzir as regras internas do prompt de sistema, que a segurança proíbe expor.

✗ D) Incorreta. Viola diretamente a proibição de expor as regras internas do prompt de sistema.

> **Dica de prova:** Explicar a decisão sem vazar instruções: saída estruturada com justificativa resumida e evidências; nunca exiba prompt de sistema ou raciocínio bruto.

D10. A) Ajustar o maxTokens por tipo de requisição com base nas métricas reais de consumo dos usuários e revisar o limite periodicamente → revise o capítulo 4

Por quê: No Amazon Bedrock, custo e latência crescem com os tokens de entrada e de saída, e a geração da saída é a parte mais lenta. O parâmetro maxTokens limita quantos tokens o modelo pode gerar; calibrá-lo por tipo de requisição, com base no ponto em que os usuários param de ler, elimina saída desperdiçada sem mexer na qualidade do trecho que é usado.

✗ B) Incorreta. Stop sequences genéricas cortam respostas em pontos que variam por tipo de pedido; podem truncar código útil ou não disparar, sem relação com o volume que o usuário consome.

✗ C) Incorreta. Um modelo menor reduz o preço por token, mas continua gerando os tokens descartados e exige revalidar a qualidade do código gerado.

✗ D) Incorreta. O prompt caching reduz custo e latência dos tokens de entrada repetidos; o desperdício descrito está nos tokens de saída que ninguém lê.

> **Dica de prova:** Tokens de saída desperdiçados: maxTokens calibrado pelo uso real; prompt caching só barateia a entrada.

D11. C) Hospedar o modelo como inference component com target tracking em ConcurrentRequestsPerCopy e mínimo de zero cópias fora do expediente → revise o capítulo 4

Por quê: No SageMaker AI, o auto scaling de endpoints em tempo real usa o Application Auto Scaling. Para LLMs, as métricas de alta resolução ConcurrentRequestsPerModel e ConcurrentRequestsPerCopy, emitidas a cada 10 segundos, refletem a carga real melhor que a CPU. Com inference components, a escala pode chegar a zero instâncias sem tráfego, e a primeira requisição dispara o retorno da capacidade em alguns minutos.

✗ A) Incorreta. O Serverless Inference aceita provisioned concurrency, mas não oferece GPUs, o que inviabiliza servir o LLM com a latência exigida nos picos.

✗ B) Incorreta. Endpoints multimodelo servem muitos modelos pouco usados que compartilham a mesma frota; com um único modelo não há o que compartilhar, e o recurso não elimina a capacidade ociosa fora do expediente.

✗ D) Incorreta. A inferência assíncrona escala até zero, mas grava o resultado no Amazon S3 depois do processamento; não atende a um assistente interativo com streaming de tokens.

> **Dica de prova:** LLM em endpoint com picos e ociosidade: inference components + target tracking por concorrência + escala a zero.

D12. C) Gerar arquivos JSONL com os registros no S3 e criar jobs de inferência em lote do Amazon Bedrock com entrada e saída em buckets do S3, acompanhando o estado dos jobs por eventos do EventBridge → revise o capítulo 4

Por quê: Para grandes volumes sem exigência de tempo real, a inferência em lote do Amazon Bedrock é a opção de menor custo e menor operação: os prompts vão em arquivos JSONL no S3, o serviço processa de forma assíncrona e grava a saída no S3. O nível Flex também é mais barato, mas é síncrono e compartilha a cota sob demanda do modelo.

✗ A) Incorreta. O nível Flex também tem desconto, mas continua sendo inferência síncrona: a equipe teria de orquestrar milhões de chamadas e tratar throttling, e a cota sob demanda é compartilhada entre os níveis, disputando capacidade com o...

✗ B) Incorreta. Provisioned Throughput cobra por capacidade reservada e não é o caminho de menor custo para um processamento pontual sem urgência; além disso, a orquestração das chamadas continuaria por conta da equipe.

✗ D) Incorreta. O perfil global aumenta a capacidade disponível, mas mantém o preço sob demanda e toda a orquestração de chamadas síncronas.

> **Dica de prova:** Milhões de itens, sem pressa, menor custo e sem orquestrar chamadas: inferência em lote com S3. Flex é para chamadas síncronas tolerantes a latência.

Agora, treine no formato real

Faça as 6 provas completas do curso AIP-C01: 450 questões inéditas, tempo do exame, explicação de cada alternativa e link da documentação para revisar.

Ver as 6 provas completas e o cupom do mês →

Até o dia da prova, uma revisão por dia no seu feed

Instagram @strecht.simulados

Um post de prova por dia: questão comentada, pegadinhas e o “não confunda” de cada certificação.

Seguir no Instagram →

YouTube Strecht Simulados

Shorts diários com pegadinhas de prova e, aos sábados, um simulado comentado.

Inscrever-se no canal →

Material independente de preparação, sem afiliação ou endosso de Amazon Web Services (AWS). As questões são inéditas e não reproduzem a prova real.