Strecht Simulados · Guia de preparação



Guia de preparação Google Professional Agentic Architect

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

70%
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 Google Professional Agentic Architect: 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 70%).

Mapa do exame

Cada prova do curso tem 50 questões e 115 minutos, no formato do exame. Referência de aprovação: 70%.

Domínio (capítulo)	Peso	Neste guia
1. Avaliar e Implantar Fluxos de Trabalho Agênticos	22%	18 conceitos · 14 pegadinhas
2. Criar Agentes com Ferramentas Low-Code	13%	18 conceitos · 14 pegadinhas
3. Desenvolver Agentes Personalizados	33%	18 conceitos · 14 pegadinhas
4. Proteger e Governar Fluxos de Trabalho Agênticos	15%	18 conceitos · 14 pegadinhas
5. Usar Agentes de Programação no Desenvolvimento de Aplicações	17%	18 conceitos · 14 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 · Avaliar e Implantar Fluxos de Trabalho Agênticos

A área de segurança do Hospital Santa Luz exige VPC Service Controls e CMEK em todos os serviços de agente. A arquiteta revisa a tabela de recursos de segurança da Agent Platform. Qual serviço NÃO atende a essa exigência atualmente?

A. Agent Platform Sessions
B. Agent Runtime
C. Memory Bank
D. Example Store

D2 · Avaliar e Implantar Fluxos de Trabalho Agênticos

Um agente ADK da Editora Mirante implantado no Agent Runtime recebe picos de 300 requisições simultâneas às segundas-feiras de manhã, e as primeiras respostas demoram vários segundos a mais que as seguintes. A implantação usa os valores padrão dos parâmetros. Qual ajuste ataca diretamente esse sintoma?

A. Habilitar context caching para guardar as respostas das segundas-feiras
B. Aumentar min_instances para cobrir a carga de base e evitar cold starts
C. Trocar o modelo por uma versão maior para acelerar o raciocínio do agente
D. Reduzir container_concurrency para 1 e forçar mais escalonamento horizontal

D3 · Avaliar e Implantar Fluxos de Trabalho Agênticos

Um projeto criado com agents-cli create -d gke na Logtech Rota Certa vai ser implantado pela primeira vez. O que o comando agents-cli deploy faz para esse alvo de implantação?

A. Envia o código ao Agent Runtime, que constrói o container pelo Dockerfile
B. Usa Terraform, build de imagem Docker e kubectl apply no cluster GKE
C. Executa gcloud beta run deploy e publica um serviço no Cloud Run
D. Publica o agente direto no Gemini Enterprise sem infraestrutura própria

D4 · Criar Agentes com Ferramentas Low-Code

A Prefeitura de Campo Novo vai ingerir no Agent Search milhares de PDFs digitalizados de processos antigos, alguns com mais de 800 páginas, e outros que misturam texto pesquisável com páginas escaneadas. Qual afirmação sobre o parser OCR para PDFs está correta?

A. Detecta tabelas, listas e títulos melhor que o layout parser em todo arquivo
B. Processa PDFs de qualquer tamanho, mas descarta as páginas com texto nativo
C. Substitui o parser digital para DOCX, PPTX e XLSX enviados no mesmo lote
D. Processa só as primeiras 500 páginas, e useNativeText mescla o texto nativo

D5 · Criar Agentes com Ferramentas Low-Code

No flow de segunda via de boleto da Companhia de Saneamento Rio Limpo, o agente precisa reagir quando o cliente fica em silêncio no canal de voz e quando o webhook de consulta de débitos retorna erro. Qual recurso do Dialogflow CX trata esses dois casos?

A. Condition routes que verificam a cada turno se o cliente falou algo
B. Form parameters obrigatórios que repetem a pergunta até haver resposta
C. Intent routes com intents de treinamento para silêncio e para erro
D. Event handlers para eventos internos, como no-input e erro de webhook

D6 · Criar Agentes com Ferramentas Low-Code

A Distribuidora de Bebidas Gelo Fino criou no Workflow Builder um workflow que deve rodar quando chega um e-mail novo de pedido no Gmail. Nos testes, o workflow às vezes começa vários minutos depois do e-mail e, num lote, dois e-mails foram processados fora da ordem de chegada. A equipe acha que é defeito. Qual explicação está de acordo com a documentação?

A. O conector do Gmail só suporta gatilhos agendados uma vez por hora
B. Gatilhos fazem polling: há atraso e a ordem não é garantida
C. Os e-mails só disparam o workflow depois de uma aprovação humana
D. O workflow precisa de um passo Loop para ordenar os eventos recebidos

D7 · Desenvolver Agentes Personalizados

Um agente ADK da Holding Atlântica tem acesso a centenas de ferramentas de servidores MCP e de APIs internas, e os custos de tokens e a latência dispararam, com piora no raciocínio do modelo. Qual abordagem a documentação de governança de skills recomenda?

A. Dividir o agente em vários LoopAgents com as mesmas ferramentas cada um
B. Aumentar a janela de contexto para caber todas as descrições de ferramentas
C. Colocar o catálogo completo de ferramentas no prompt de sistema do agente
D. Usar Agent Skills com divulgação progressiva de ferramentas

D8 · Desenvolver Agentes Personalizados

Um agente da Rede Postos Caminho precisa de duas integrações: ler os tickets do Jira de cada gerente de posto em nome dele e enviar dados de vendas a uma API de parceiro que aceita OAuth de máquina para máquina. Quais tipos de auth provider configurar no auth manager?

A. OAuth 3-legged para o Jira e 2-legged para o parceiro
B. API key para o Jira e HTTP basic para a API do parceiro
C. OIDC ID token para o Jira e API key para a API do parceiro
D. OAuth 2-legged para o Jira e OAuth 3-legged para a API do parceiro

D9 · Desenvolver Agentes Personalizados

No ADK, um engenheiro da Logística Cerrado colocou três subagentes dentro de um ParallelAgent para consultar, ao mesmo tempo, frete, estoque e previsão do tempo. Nos testes, o subagente de frete não enxerga o que o subagente de estoque acabou de descobrir na mesma execução. Qual explicação está correta?

A. Cada ramo roda de forma independente, sem troca automática de histórico ou estado
B. O ParallelAgent precisa de um LoopAgent externo para propagar o estado aos ramos
C. O ParallelAgent consulta o modelo para decidir a ordem e só repassa o estado no fim
D. Os subagentes só compartilham estado quando usam o mesmo modelo de linguagem

D10 · Proteger e Governar Fluxos de Trabalho Agênticos

Um agente ADK da Rede Supermercados Bom Preço consulta o BigQuery por meio de uma ferramenta de SQL. A equipe quer garantir, de forma determinística, que o agente execute só SELECT e só nas tabelas de vendas, independentemente do que o modelo decidir passar como argumento. Qual abordagem a documentação de segurança do ADK descreve?

A. Guardrail na ferramenta com política fixada no Tool Context
B. Temperatura zero no modelo para tornar as consultas determinísticas
C. Context caching com a lista de tabelas permitidas no início do prompt
D. Instrução de sistema pedindo ao modelo que nunca gere comandos de escrita

D11 · Proteger e Governar Fluxos de Trabalho Agênticos

A Agência de Comunicação Voz Ativa automatizou no Workflow Builder a produção de comunicados à imprensa: um passo Gemini agent redige o texto a partir de um briefing recebido por e-mail, outro passo busca dados do cliente no Drive, e o último publica o comunicado num portal externo por um conector. A diretoria exige que nenhum comunicado seja publicado sem a revisão e a assinatura de um editor humano, de forma determinística, sem depender de o modelo lembrar de pedir aprovação. Os editores devem ver o rascunho gerado antes de liberar a publicação no portal do cliente. Qual configuração atende?

A. Escrever na instrução do Gemini agent que peça aprovação antes
B. Adicionar um passo Filter que remove rascunhos não revisados
C. Inserir um passo Approval antes do passo que publica no portal
D. Agendar o workflow para rodar só no horário comercial do editor

D12 · Proteger e Governar Fluxos de Trabalho Agênticos

Num aplicativo de saúde da Clínica Bem Estar, todos os pacientes conversam com o mesmo agente, que usa o Memory Bank. O encarregado de proteção de dados quer garantia de que a consolidação e a recuperação de memórias de um paciente não misturem fatos de outro paciente. Qual característica do Memory Bank atende a essa exigência?

A. Isolamento por escopo: consolidação e recuperação ficam restritas à identidade
B. Busca por similaridade global, que compara memórias de todos os usuários
C. Memory revisions, que bloqueiam a leitura de versões antigas da memória
D. TTL curto configurado para apagar as memórias ao fim de cada dia

01

CAPÍTULO 1 · 22% DA PROVA

Avaliar e Implantar Fluxos de Trabalho Agênticos

Aqui eu preciso saber avaliar agentes antes e depois do deploy (ADK eval, Agents CLI eval, Agent Platform Evals, simulação e Online Monitors), escolher o runtime certo (Agent Runtime, Cloud Run ou GKE) e diagnosticar problemas com Cloud Trace, Cloud Monitoring, Cloud Logging e ajustes como container_concurrency e max_llm_calls. A prova cobra nomes exatos de critérios e métricas, quais exigem referência e quais não, e as flags do agents-cli deploy. O que mais derruba é confundir critério com referência (response_match_score, final_response_match_v2) com critério sem referência (rubric_based, hallucinations_v1) e misturar avaliação offline com monitoramento de produção. Também é fácil errar o recurso do Cloud Run (service, instance, worker pool, job) quando o enunciado descreve o comportamento do agente.

Neste capítulo

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

Tópicos-chave

O que se avalia num agente

Como modelos são probabilísticos, testes de passa ou falha não bastam: avaliam-se a trajetória e o uso de ferramentas, além da resposta final, com métricas baseadas em referência ou em rubricas. Cada inferência de um eval case gera um trace, o registro imutável de entradas, respostas e chamadas de ferramentas sobre o qual as métricas são calculadas.

Eval case

Especifica a tarefa do agente e pode conter um ou mais passos de conversa, o contexto (estado do agente no início do caso) e uma especificação para simular as respostas do usuário durante a inferência.

Avaliação no ADK

Arquivos .test.json têm uma sessão simples cada, rápidos como testes unitários; evalsets têm várias sessões potencialmente longas, para integração. Caminhos: adk web (interativo), pytest (pipeline) e adk eval (terminal). Padrão sem configuração: tool_trajectory_avg_score 1.0 e response_match_score 0.8; test_config.json define outros critérios.

Critérios do ADK com e sem referência

Com referência: tool_trajectory_avg_score, response_match_score (ROUGE-1), response_evaluation_score e final_response_match_v2 (equivalência semântica). Sem referência: rubric_based (como rubric_based_final_response_quality_v1), hallucinations_v1, safety_v1 e os multi-turn (multi_turn_task_success_v1, multi_turn_trajectory_quality_v1, multi_turn_tool_use_quality_v1), que suportam simulação de usuário.

Trajetória e critérios de eficiência

tool_trajectory_avg_score compara as ferramentas chamadas com EXACT (sem extras), IN_ORDER (mesma ordem, extras permitidas) ou ANY_ORDER (qualquer ordem, extras permitidas). O ADK reporta sempre quatro critérios de eficiência (contagem de ferramentas e de chamadas ao modelo, tokens e duração, como tool_call_count_v1 e token_usage_v1): são informativos, não reprovam caso e não aceitam limiar.

Avaliação no Agents CLI

Usa o GenAI Eval SDK. eval run = eval generate + eval grade, com dataset em tests/eval/datasets/basic-dataset.json e métricas em tests/eval/eval_config.yaml; --url avalia um agente implantado e eval grade reavalia traces existentes sem reexecutar. O fluxo é iterativo: comece com 1 ou 2 casos centrais e amplie. Métricas como hallucination (afirmações atômicas contra o contexto das ferramentas) e final_response_match (contra referência).

Agent Platform Evals e Metric Registry

Avaliação gerenciada (antes Gen AI evaluation service). O Metric Registry guarda métricas predefinidas (Final Response Quality com rubrica adaptativa, Hallucination com rubrica estática por afirmação, Tool Use Quality, multi-turn), métricas LLM customizadas (rubrica em linguagem natural com juiz) e métricas de código customizado (Python). No SDK, as predefinidas são acessadas por types.RubricMetric.

Simulação antes da produção

Gera cenários (eval cases com prompt inicial e conversation plan oculto) a partir das instruções e ferramentas e usa um User Simulator para conversas de várias rodadas com limite de turnos, produzindo traces. Environment simulation intercepta chamadas de ferramenta com respostas simuladas. Exige ao menos uma versão do agente no Agent Registry.

Online Monitors e quality alerts

Pontuam de forma assíncrona os traces ao vivo para detectar quality drift. Exigem agente implantado, Cloud Trace habilitado e sinais OpenTelemetry (gen_ai.agent.name, gen_ai.conversation.id, mensagens e definições de ferramentas); mídias grandes vão para o Cloud Storage. Rodam com a P4SA, então restrinja quem cria OnlineEvaluator. As notas viram métricas no Cloud Monitoring, e uma política de alerta sem canal ainda gera incidente, sem notificação.

Otimização e exemplos

adk optimize aplica o algoritmo GEPA contra a suíte de testes no ciclo de qualidade. O Example Store guarda exemplos few-shot e recupera os mais relevantes a cada requisição. O Feedback service (pré-GA) coleta reações ligadas à sessão e ao evento, sem banco próprio.

Implantação no Agent Runtime

Runtime gerenciado e opinativo. Métodos: Developer Connect (repositório Git e CI/CD), arquivos-fonte, Dockerfile, imagem de container no Artifact Registry e objeto do Agent Platform SDK (notebook); Dockerfile e imagem exigem o runtime contract. Fixe versões no requirements. ADK tem integração completa (CLI adk para Python, adkgo para Go); LangChain, LangGraph, AG2 e LlamaIndex usam templates do SDK.

Escala e recursos do Agent Runtime

min_instances cobre a carga de base e evita cold start. Cada instância tem 9 processos de agente; para agentes assíncronos aumente container_concurrency para um múltiplo de 9, como 36. Revisões são snapshots imutáveis com divisão de tráfego (pré-GA, v1beta1), e revisão arquivada não volta. Bidirectional streaming (pré-GA) atende áudio e vídeo em tempo real; a interface Private Service Connect dá saída privada à VPC e on-premises.

agents-cli deploy

Lê o deployment_target do agents-cli-manifest.yaml: agent_runtime (build sempre pelo Dockerfile, sem --image), cloud_run (gcloud beta run deploy, aceita imagem pronta, limites e --dry-run) ou gke (Terraform, build Docker e kubectl apply). --no-wait com --status acompanha implantação assíncrona e --list lista. O pipeline tem CI no PR, CD de staging no merge com build, deploy e teste de carga, e produção após aprovação manual com a mesma imagem.

Agentes no Cloud Run

Services atendem agentes sem estado guiados por requisição (escalam a zero); instances, agentes singleton com estado sempre ligados; worker pools, frotas que consomem filas sem HTTP público; jobs, fluxos que rodam até concluir. Memória: Memorystore for Redis (curto prazo), Firestore (longo prazo) e Memory Bank; busca vetorial com pgvector no Cloud SQL ou AlloyDB; interação por streaming HTTP e WebSockets.

GKE e Code Execution

GKE dá mais controle da implantação e suporte a modelos abertos. O Code Execution da Agent Platform cria sandboxes em menos de um segundo, aceita arquivos de até 100 MB por requisição, mantém estado com TTL e funciona com qualquer framework e modelo, sem exigir implantar na Agent Platform.

Observabilidade do Agent Runtime

A aba Observability tem visões Overview, Evaluation, Models e Tools. Métricas internas do ReasoningEngine (requisições, latência, CPU, memória) aparecem no Cloud Monitoring sem configuração; GOOGLE_CLOUD_AGENT_ENGINE_ENABLE_TELEMETRY acrescenta métricas GenAI no ADK. Cloud Trace mostra spans de cada operação. Logs: stdout e stderr como texto (reasoning_engine_stdout e reasoning_engine_stderr), logging do Python estruturado ou cliente do Cloud Logging; recursos filhos (Sessions, Memory Bank, Code Execution, Example Store) não têm Cloud Logging. BigQuery Agent Analytics é opcional, via --bq-analytics.

Diagnóstico de falhas e loops

Falhas intermitentes: retry_options com HttpRetryOptions na definição do modelo. Loop de raciocínio: RunConfig com max_llm_calls, e traces e avaliações para corrigir a causa. Histórico grande demais: filtros num_recent_events ou after_timestamp no carregamento da sessão. Topologia sem o agente novo: aguarde a sincronização e repita.

Controles de segurança por recurso

Agent Runtime, avaliação de agentes, Sessions, Memory Bank e Code Execution suportam VPC-SC, CMEK e residência de dados em repouso; o Example Store não.

Macetes para memorizar

💡 Padrão do ADK eval

Trajetória perfeita (1.0), resposta quase igual (0.8).

💡 Trajetória

EXACT = fila indiana sem penetras; IN_ORDER = fila com penetras; ANY_ORDER = bagunça permitida.

Agents CLI eval

run = generate + grade; grade sozinho reaproveita traces.

Cloud Run para agentes

Service atende, Instance mora, Worker pool trabalha na fila, Job termina e vai embora.

container_concurrency

Nove processos por instância: pense em múltiplos de 9.

Antes e depois da produção

Antes: simula (usuário e ambiente). Depois: monitora (Online Monitors) e alerta.

Contra loops e falhas

Loop = max_llm_calls; falha intermitente = retry; histórico gordo = num_recent_events.

Tabela de memorização: não confunda

Quando é...	...e quando é
Arquivo .test.json Sessão simples, rápida, como teste unitário frequente.	**Evalset** Várias sessões longas, como teste de integração menos frequente.
final_response_match_v2 ou response_match_score Há resposta de referência aprovada (golden dataset).	**rubric_based ou hallucinations_v1** Não há referência; avalia por rubrica ou pela fundamentação no contexto.
Simulação de usuário e de ambiente Antes da produção, sem tráfego real, testando resiliência.	**Online Monitors** Produção, pontuando traces ao vivo para detectar quality drift.
Agent Runtime Time quer runtime gerenciado, sessões gerenciadas e foco na lógica.	**GKE** Mais controle da implantação e modelos abertos, operando o cluster.
Cloud Run service Chatbot sem estado, guiado por requisição, escalando a zero.	**Cloud Run instance** Assistente singleton com estado, sempre ligado.
Cloud Run worker pool Frota que consome fila (Pub/Sub) sem HTTP público.	**Cloud Run job** Processamento em lote que roda até concluir.
Cloud Trace Achar onde o tempo é gasto dentro de cada consulta (spans).	**Cloud Monitoring** Contagem, latência, CPU e memória agregadas, alertas e PromQL.

Números que caem

Item	Valor
Critérios padrão do ADK eval	**tool_trajectory_avg_score 1.0 e response_match_score 0.8**
Processos de agente por instância do Agent Runtime	**9 (concorrência por processo = container_concurrency / 9)**
Exemplo de container_concurrency para agente assíncrono	**36**
Critérios de eficiência sempre reportados pelo ADK	**4**
Arquivos no Code Execution	**até 100 MB por requisição; sandbox em menos de um segundo**
Casos para começar a avaliação no Agents CLI	**1 ou 2 casos centrais**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Avaliar contra golden dataset com text_quality ou tool_use_quality.
Como não cair: Golden dataset pede métrica com referência: final_response_match (Agents CLI) ou response_match_score e final_response_match_v2 (ADK).

⚠ **Alerta de prova:** Usar final_response_reference_free quando há referência.
Como não cair: Essa métrica dispensa referência e exige rubric_groups no caso.

⚠ **Alerta de prova:** Achar que EXACT aceita chamadas extras ou que ANY_ORDER exige ordem.
Como não cair: EXACT é idêntico; IN_ORDER mantém a ordem com extras; ANY_ORDER aceita qualquer ordem com extras.

⚠ **Alerta de prova:** Inverter o padrão do ADK para trajetória 0.8 e resposta 1.0.
Como não cair: Padrão: tool_trajectory_avg_score 1.0 e response_match_score 0.8.

⚠ **Alerta de prova:** Configurar limiar nos critérios de eficiência para reprovar casos caros.
Como não cair: São informativos, sempre ativos e um limiar configurado neles é rejeitado.

⚠ **Alerta de prova:** Usar Online Monitors ou divisão de tráfego para testar falhas antes do lançamento.
Como não cair: Sem produção, use environment simulation; monitores e tráfego real expõem clientes.

⚠ **Alerta de prova:** Achar que a simulação exige golden dataset ou agente publicado no Gemini Enterprise.
Como não cair: Ela gera os casos e exige só uma versão do agente no Agent Registry.

⚠ **Alerta de prova:** Reduzir container_concurrency para 1 contra filas em agentes assíncronos.
Como não cair: Para ADK assíncrono, aumente para múltiplo de 9 (ex.: 36).

⚠ **Alerta de prova:** Usar context caching ou modelo maior para resolver cold start.
Como não cair: Cold start se resolve com min_instances cobrindo a carga de base.

⚠ **Alerta de prova:** Passar --image no deploy para Agent Runtime.
Como não cair: No Agent Runtime o build é sempre pelo Dockerfile; imagem pronta só no Cloud Run.

⚠ **Alerta de prova:** Procurar logs de Sessions e Memory Bank no Cloud Logging.
Como não cair: Recursos filhos do Agent Runtime não têm suporte a Cloud Logging.

⚠ **Alerta de prova:** Acreditar que a política de alerta não é criada sem canal de notificação.
Como não cair: Ela é criada, gera verificação e incidente, só não notifica.

⚠ **Alerta de prova:** Usar max_llm_calls para reduzir o histórico carregado ou para tratar falhas transitórias.
Como não cair: Histórico: num_recent_events ou after_timestamp. Falhas: HttpRetryOptions. max_llm_calls contém loops.

⚠ **Alerta de prova:** Reimplantar ou criar o nó à mão quando o agente novo não aparece na topologia.
Como não cair: Agentes do Agent Runtime são registrados sozinhos; aguarde a sincronização.

Como cai na prova

📝 **Como cai:** Time tem ou não tem resposta de referência e precisa escolher critérios de avaliação.
Como resolver: Separe os casos: com referência, critérios match; sem referência, rubric_based, hallucinations_v1 ou multi-turn. Lembre que critérios de eficiência só informam.

📝 **Como cai:** Escolha do runtime pelo perfil do time e do agente.
Como resolver: Sem capacidade de operar infraestrutura: Agent Runtime. Controle e modelos abertos: GKE. Containers com recursos específicos (service, instance, worker pool, job): Cloud Run.

📝 **Como cai:** Sintoma em produção: lentidão, loop, falha intermitente, cold start ou fila.
Como resolver: Lentidão dentro da consulta: Cloud Trace. Loop: max_llm_calls. Intermitente: HttpRetryOptions. Cold start: min_instances. Fila em agente assíncrono: container_concurrency maior.

📝 **Como cai:** Comando ou flag do Agents CLI para implantar, acompanhar ou avaliar.
Como resolver: deploy --no-wait e --status para assíncrono, --list para listar, --dry-run só no Cloud Run, --image nunca no Agent Runtime; eval run --url para implantado; eval grade para traces existentes.

📝 **Como cai:** Onde encontrar uma métrica, log ou visão de desempenho.
Como resolver: Cota e uso por modelo: visão Models da aba Observability. Latência de ferramentas: visão Tools. Requisições, CPU e memória: Cloud Monitoring (ReasoningEngine). Spans: Cloud Trace.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Avaliar contra respostas de referência aprovadas	**final_response_match_v2 ou response_match_score**
Cada afirmação conferida com o contexto das ferramentas	**hallucination ou hallucinations_v1**
Objetivo da conversa inteira cumprido	**multi_turn_task_success_v1**
Mesma ordem de ferramentas, aceitando chamadas extras	**IN_ORDER no tool_trajectory_avg_score**
Reavaliar traces com nova métrica sem reexecutar o agente	**agents-cli eval grade**
Avaliar o serviço já implantado	**agents-cli eval run com --url**
Qualidade caindo em produção ao longo do tempo	**Online Monitors e quality alerts (quality drift)**
Testar a API de terceiros falhando antes do lançamento	**Environment simulation**
Melhorar instruções automaticamente contra a suíte	**adk optimize com GEPA**
Picos de segunda-feira com cold start	**min_instances para a carga de base**
Agente ADK assíncrono com filas e instâncias ociosas	**container_concurrency múltiplo de 9**
Rollout gradual de nova versão do agente	**Revisões com divisão de tráfego**
Agente no Agent Runtime acessando API interna on-premises	**Interface Private Service Connect com network attachment**
Agente em loop gastando chamadas ao modelo	**RunConfig com max_llm_calls**

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

- Avalie trajetória e resposta final, não só a saída.
- Com referência: response_match_score e final_response_match_v2. Sem referência: rubric_based, hallucinations_v1, safety_v1 e multi-turn.
- Simulação antes da produção; Online Monitors depois, com Cloud Trace e OpenTelemetry.
- Agent Runtime não aceita --image no agents-cli deploy.
- Sessions, Memory Bank, Code Execution e Example Store não têm Cloud Logging.

Essenciais para a prova (checklist)

- ☐ Sei a diferença entre .test.json e evalset e os três caminhos de avaliação do ADK.
- ☐ Sei quais critérios do ADK exigem referência e quais não.
- ☐ Sei o que EXACT, IN_ORDER e ANY_ORDER aceitam.
- ☐ Sei como eval run, eval generate e eval grade se relacionam no Agents CLI.
- ☐ Sei os requisitos dos Online Monitors e como viram alertas.
- ☐ Sei os cinco métodos de implantação do Agent Runtime e quando usar cada um.
- ☐ Sei ajustar min_instances e container_concurrency.
- ☐ Sei escolher entre service, instance, worker pool e job no Cloud Run.
- ☐ Sei onde olhar métricas, traces e logs de um agente no Agent Runtime.
- ☐ Sei o fluxo do pipeline de CI/CD gerado pelo Agents CLI.

Questões de revisão do capítulo

R1.1 · Avaliar e Implantar Fluxos de Trabalho Agênticos

Na Fábrica Móveis Carvalho, uma implantação do agente no Agent Runtime que funcionava ontem falhou hoje sem mudança no código, porque uma dependência ganhou uma versão nova. Qual boa prática a documentação recomenda para builds reprodutíveis?

A. Deixar o Agent Runtime escolher as versões compatíveis
B. Fixar as versões dos pacotes na lista de requirements
C. Usar sempre a versão mais recente de cada pacote
D. Remover o requirements.txt e instalar pacotes em runtime

R1.2 · Avaliar e Implantar Fluxos de Trabalho Agênticos

O Banco Comunitário Raízes tem um conjunto de perguntas com respostas-padrão aprovadas pelo jurídico (golden answers) e quer medir se o agente responde de forma equivalente a elas. Qual métrica do Agents CLI compara com essa referência?

A. text_quality, que mede fluência, coerência e gramática
B. final_response_match, comparando com a resposta de referência
C. tool_use_quality, que avalia seleção e parâmetros das ferramentas
D. final_response_reference_free, que dispensa qualquer referência

R1.3 · Avaliar e Implantar Fluxos de Trabalho Agênticos

O time de qualidade da Seguradora Prudência usa as métricas predefinidas da avaliação de agentes da Agent Platform. Qual métrica verifica se cada afirmação da resposta está fundamentada no uso de ferramentas registrado nos eventos intermediários?

A. Exact Match, que compara com uma resposta de referência
B. Agent Final Response Quality, com uma rubrica adaptativa gerada automaticamente
C. Agent Hallucination, que checa afirmações atômicas contra as ferramentas
D. Agent Tool Use Quality, que avalia a escolha das ferramentas

R1.4 · Avaliar e Implantar Fluxos de Trabalho Agênticos

Na simulação de agentes da Agent Platform, o time da Imobiliária Casa Nova Sul gera eval cases para um agente de locação. Além do prompt inicial, o que cada caso traz para orientar o usuário simulado?

A. Um script fixo com todas as respostas que o agente deve dar
B. Uma resposta de referência que o agente deve copiar palavra por palavra
C. Uma lista de ferramentas proibidas para o usuário simulado
D. Um conversation plan oculto com objetivos e reações do usuário

Respostas comentadas

R1.1. B) Fixar as versões dos pacotes na lista de requirements

Por quê: Ao implantar no Agent Runtime, os pacotes são informados por lista ou arquivo de requirements (ou requirements.txt nos arquivos-fonte). Fixar versões evita que mudanças em dependências quebrem a implantação.

✗ A) Incorreta. O Agent Runtime instala o que é declarado; a escolha das versões é responsabilidade do time.
✗ C) Incorreta. Usar a versão mais recente é o que causou a quebra.
✗ D) Incorreta. Instalar em runtime aumenta a imprevisibilidade e o tempo de inicialização.

> **Dica de prova:** Builds reprodutíveis do agente: versões de pacotes fixadas.

R1.2. B) final_response_match, comparando com a resposta de referência

Por quê: Para avaliar um sistema agêntico contra um golden dataset, use métricas baseadas em referência, como final_response_match no Agents CLI ou response_match_score e final_response_match_v2 no ADK.

✗ A) Incorreta. text_quality avalia aspectos linguísticos, não fidelidade à referência.

✗ C) Incorreta. tool_use_quality avalia uso de ferramentas, não equivalência com a resposta aprovada.

✗ D) Incorreta. Essa métrica avalia a resposta sem referência e exige rubric_groups no caso.

> **Dica de prova:** Comparar com respostas aprovadas (golden): final_response_match.

R1.3. C) Agent Hallucination, que checa afirmações atômicas contra as ferramentas

Por quê: O Metric Registry traz métricas predefinidas de agente, como Final Response Quality (rubrica adaptativa), Hallucination (rubrica estática por afirmação) e Tool Use Quality, além de métricas multi-turn.

✗ A) Incorreta. Exact Match exige resposta de referência e não verifica fundamentação.

✗ B) Incorreta. Final Response Quality gera critérios a partir das instruções, ferramentas e do prompt para avaliar a resposta como um todo.

✗ D) Incorreta. Tool Use Quality avalia o uso das ferramentas, não a fundamentação das afirmações.

> **Dica de prova:** Afirmações sem base nas ferramentas: métrica Agent Hallucination.

R1.4. D) Um conversation plan oculto com objetivos e reações do usuário

Por quê: A simulação gera cenários a partir das instruções e ferramentas do agente e usa um User Simulator para conduzir conversas de várias rodadas, com limite de turnos, produzindo traces para avaliação.

✗ A) Incorreta. O agente responde livremente; o plano orienta o usuário simulado, não o agente.

✗ B) Incorreta. A simulação não exige resposta de referência para o agente copiar.

✗ C) Incorreta. O usuário simulado não recebe lista de ferramentas proibidas.

> **Dica de prova:** Usuário simulado é guiado por um conversation plan oculto.

02

CAPÍTULO 2 · 13% DA PROVA

Criar Agentes com Ferramentas Low-Code

Aqui eu preciso dominar três frentes: o Workflow Builder do Gemini Enterprise (chamado de Agent Designer no guia do exame), o CX Agent Studio com a herança do Dialogflow CX e a conexão de dados corporativos ao Gemini Enterprise e ao Agent Search. A prova traz cenários de negócio e pede o passo, o tipo de agente, o parser ou o tipo de sincronização certo. O ponto que mais derruba é confundir o que é determinístico (passos, callbacks, handoff rules, flows) com o que depende do modelo (instruções, exemplos, ferramenta que o agente decide chamar). Também erro quando troco ingestão por federação, full sync por incremental sync e layout parser por OCR. Quando o enunciado fala em garantia ou em sempre, a resposta quase nunca é escrever melhor a instrução.

Neste capítulo

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

Tópicos-chave

Workflow Builder (Agent Designer no guia)

Ferramenta no-code e low-code do Gemini Enterprise para criar chat agents e workflows. Criar por prompt é o caminho mais rápido e serve de ponto de partida; o flow builder dá controle granular para agentes de várias etapas com subagentes, e os dois métodos podem ser combinados.

Chat agent x workflow

Chat agents atendem interações conversacionais em que a entrada do usuário guia o fluxo. Workflows executam passos a partir de um gatilho, com lógica determinística ou não e aprovações humanas, como num processo de admissão.

Passos de controle e human-in-the-loop

Condition ramifica, Filter filtra listas e Loop itera sobre itens. Os passos human-in-the-loop são Request info, que pausa para coletar informação do usuário, e Approval, que exige aprovação humana antes dos passos seguintes.

Event triggers

Disparam workflows quando dados mudam em conectores (novos e-mails, eventos de calendário, arquivos). O conector precisa estar conectado como data store, e a detecção é por polling: há atraso e a ordem dos eventos não é garantida.

Modo designer x modo chat

O modo chat ajuda a criar e ajustar agentes por linguagem natural, mas data stores de fontes da categoria Cloud (BigQuery, Cloud Storage, Cloud SQL) só são habilitados pelo modo designer.

Passo MCP x ferramenta MCP no Gemini agent

O passo MCP chama ferramentas, lista recursos e consulta servidores em toda execução, de forma garantida. Num passo Gemini agent, ferramentas MCP autorizadas em Tools & Integrations são chamadas quando o modelo decide.

Governança do Gemini Enterprise

Usuários não administradores podem criar agentes e workflows, mas só usam conectores, ferramentas e permissões provisionados pelo administrador. A política da organização discoveryengine.managed.allowedDataSources (valores como google_drive, google_mail, sharepoint) controla quais fontes podem ser conectadas.

CX Agent Studio

Construtor de agentes conversacionais com pouco código, baseado no ADK e evolução do Dialogflow CX. Tem recursos de IA como restructure instructions (converte instruções para XML), refine instructions (clareza) e test case hill climbing (rigor e estabilidade dos casos de teste).

Variáveis e referências nas instruções do CX Agent Studio

{{variavel}} é estática: compilada no prompt antes da chamada, e atualizá-la invalida o prompt caching. {variavel} é dinâmica: atualizada por ferramentas, callbacks ou API e entra no histórico como state update. {@AGENT: nome} e {@TOOL: nome} referenciam agentes e ferramentas, e modular components aceitam condições como _session.modality == 'VOICE'.

Guardrails do CX Agent Studio

Incluem prompt guard (ataques do tipo ignore suas instruções), blocklists (palavra inteira, qualquer menção ou regex), filtros de segurança Relaxed, Balanced ou Strict, regras em linguagem natural ou código (after_model_callback) e supervisor agents, cada um com um resultado como Say exactly, handoff ou Generate a response.

Determinismo no CX Agent Studio

Handoff rules forçam ou bloqueiam transferências por condição, sem código. Callbacks rodam fora do controle do agente e são determinísticos. Uma tool tem interior determinístico, mas a decisão de chamá-la é do agente; instruções e exemplos continuam dependendo do modelo.

Flows e flow-based agents

Flows servem para lógica determinística que não cabe em callbacks, como coleta sequencial e autenticação; não servem para resposta fixa ou classificação de intenção. O flow-based agent conduz a conversa até END_SESSION e devolve o controle ao agente, com variáveis mapeadas; é chamado por DetectIntent, o que exige o papel Dialogflow API Client no Customer Engagement Suite Service Agent.

Dialogflow CX como máquina de estados

Cada page é um estado e só uma page fica ativa por vez. State handlers dividem-se em routes (intent e/ou condição; só condição é condition route) e event handlers (eventos internos como no-input, no-match e erro de webhook). Fora da start page, routes do flow só ficam em escopo se tiverem intent.

Exemplos de playbook

Funcionam como prompts few-shot, incluindo ações do agente. A estratégia de seleção define se o exemplo entra no prompt: Always select, Dynamically select (por relevância) ou Never select.

Data stores e sincronização

Ingestão copia e indexa os dados no Gemini Enterprise (melhor busca, mais armazenamento e tempo); federação consulta a origem sem cópia. Full sync captura o estado inteiro, inclusive exclusões, e pode ser agendada como entity sync (dados) e identity sync (usuários dos grupos de ACL); incremental sync traz adições e atualizações, sem identidade nem exclusões.

Conectores de terceiros e rede

Ao conectar uma fonte como o Jira, o Gemini Enterprise cria um data store e data stores de entidades (issues, comentários). Static IP egress fixa os IPs de saída para liberar no firewall da origem. Conectar vários data stores a um app é blended search.

Parsers do Agent Search

Parser digital é o padrão e detecta blocos de texto. OCR é para PDFs digitalizados, mais barato que o layout parser, processa só as primeiras 500 páginas e mescla texto nativo com useNativeText. Layout parser detecta títulos, tabelas, listas e imagens, alimenta o chunking para RAG e oferece image annotation e table annotation.

Dados multimodais e contexto

Modelos Gemini aceitam imagem, vídeo, áudio e documentos direto na requisição. Arquivos da seção Knowledge ou anexados no chat vão para a janela de contexto a cada turno, até 800.000 tokens combinados; bibliotecas grandes devem ir para um data store conectado. O gemini-embedding-2 mapeia texto, imagem, vídeo e PDF num só espaço; o LLM parser do RAG Engine interpreta gráficos com custo de tokens.

Macetes para memorizar

💡 Human-in-the-loop no Workflow Builder

Request info PERGUNTA, Approval ASSINA.

💡 Controle de fluxo

Condition escolhe o caminho, Filter peneira, Loop repete por item.

💡 Variáveis do CX Agent Studio

Duas chaves {{ }} = dado parado; uma chave { } = dado que se mexe.

💡 Determinismo

Instrução pede, exemplo sugere, tool o agente escolhe; callback, handoff rule e passo GARANTEM.

💡 Sincronização

Full = foto inteira (inclui o que sumiu); incremental = só o que é novo; identity = quem pode ver.

💡 Parsers do Agent Search

Digital lê, OCR enxerga papel escaneado (até 500 páginas), layout entende a estrutura.

💡 Ingestão x federação

Ingestão copia e busca melhor; federação pergunta na origem sem copiar.

Tabela de memorização: não confunda

Quando é...	...e quando é
Chat agent Dúvidas abertas em que a entrada do usuário guia a conversa.	**Workflow** Sequência definida disparada por gatilho, com aprovações humanas e passos garantidos.

Request info Preciso coletar uma informação nova do usuário, como o centro de custo.	**Approval** Preciso de assinatura ou liberação humana antes de uma ação crítica, como publicar.
Ingestão Quero melhor qualidade de busca e aceito copiar e indexar os dados.	**Federação** Quero consultar a origem em tempo real, sem cópia, aceitando busca possivelmente pior.
Full sync Preciso capturar exclusões e mudanças de identidade e ACL (identity sync).	**Incremental sync** Basta trazer entidades adicionadas ou atualizadas.
Variável estática {{nome}} Conteúdo fixo como catálogo ou regras de negócio, compilado no prompt.	**Variável dinâmica {nome}** Valor que muda na conversa, como número da reserva, atualizado por ferramenta ou callback.
Layout parser Documentos com tabelas, listas, títulos e imagens, e chunking estruturado para RAG.	**OCR parser** PDFs escaneados, com custo menor, até as primeiras 500 páginas.
Condition route Transição pela condição da sessão, como formulário completo levando à confirmação.	**Event handler** Reação a eventos internos, como no-input, no-match ou erro de webhook.

Números que caem

Item	Valor
Limite do OCR parser do Agent Search	**primeiras 500 páginas do PDF**
Arquivos da seção Knowledge e anexos no contexto do Gemini Enterprise	**até 800.000 tokens combinados, carregados a cada turno**
Níveis dos filtros de segurança do CX Agent Studio	**Relaxed, Balanced ou Strict**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Escrever na instrução do agente que peça aprovação antes de publicar.
Como não cair: Instrução não é determinística. Revisão humana garantida é passo Approval antes do passo que age.

⚠ **Alerta de prova:** Usar Filter para remover rascunhos não revisados.
Como não cair: Filter só filtra listas de dados; não cria etapa humana.

⚠ **Alerta de prova:** Achar que o webhook muda a página por conta própria.
Como não cair: O webhook gera respostas dinâmicas; quem muda de página é o transition target do handler.

⚠ **Alerta de prova:** Tratar silêncio do cliente ou falha de webhook com intent routes ou condition routes.
Como não cair: Silêncio e erro de webhook são eventos: use event handlers (no-input, erro de webhook).

⚠ **Alerta de prova:** Usar flow para resposta fixa de boas-vindas ou para classificar intenção.
Como não cair: Resposta fixa vai num callback; classificação de intenção é tarefa do LLM. Flow é para coleta sequencial e autenticação.

⚠ **Alerta de prova:** Colocar o número da reserva numa variável estática para melhorar a adesão.
Como não cair: Estáticas não são atualizadas no meio da sessão e, se mudarem, invalidam o prompt caching. Valor mutável é dinâmico.

⚠ **Alerta de prova:** Callback em Python para controlar transferência entre agentes quando o time não quer código.
Como não cair: Handoff rules dão controle determinístico sem código.

⚠ **Alerta de prova:** Confiar numa tool Python para garantir um aviso em toda resposta.
Como não cair: A decisão de chamar a tool é do agente. Garantia exige callback.

⚠ **Alerta de prova:** Achar que a incremental sync traz exclusões e mudanças de permissão.
Como não cair: Exclusões e ACL vêm da full sync, com identity sync para usuários dos grupos.

⚠ **Alerta de prova:** Usar floor settings do Model Armor ou identity sync para limitar quais conectores podem ser usados.
Como não cair: Quem limita fontes é a política discoveryengine.managed.allowedDataSources.

⚠ **Alerta de prova:** Dividir PDFs grandes e anexar todos de novo na seção Knowledge.
Como não cair: Tudo vai para o contexto a cada turno (até 800.000 tokens). Biblioteca grande vai num data store conectado.

⚠ **Alerta de prova:** Achar que o OCR parser processa PDFs de qualquer tamanho ou entende tabelas melhor que o layout parser.
Como não cair: OCR para nas primeiras 500 páginas; estrutura complexa e imagens pedem layout parser.

⚠ **Alerta de prova:** Adicionar um segundo data store de outra região na mesma data store tool.
Como não cair: Cada ferramenta fica presa à região do primeiro data store; crie outra data store tool.

⚠ **Alerta de prova:** Chamar de hybrid search a busca em vários data stores ligados a um app.
Como não cair: Vários data stores num app é blended search; hybrid search é semântica mais tokens no Vector Search.

Como cai na prova

📝 **Como cai:** Cenário de negócio com exigência de garantia (nunca, sempre, obrigatório) e alternativas que mexem na instrução do modelo.
Como resolver: Descarte tudo que depende do modelo (instrução, exemplo, agendamento). Procure o mecanismo determinístico: passo Approval, callback, handoff rule, passo MCP ou flow.

📝 **Como cai:** Lista de passos do Workflow Builder e um processo descrito em etapas.
Como resolver: Mapeie cada verbo: perguntar ao usuário é Request info, aprovar é Approval, ramificar é Condition, filtrar é Filter, repetir por item é Loop.

📝 **Como cai:** Problema de dados no Gemini Enterprise: permissão desatualizada, exclusão não refletida, firewall bloqueando.
Como resolver: ACL e usuários: identity sync. Exclusões: full sync. IPs fixos: static IP egress. Restringir fontes: allowedDataSources.

📝 **Como cai:** Dialogflow CX com página, route e evento; pergunta sobre escopo ou transição.
Como resolver: Lembre que só uma page fica ativa, que eventos tratam silêncio e erro, e que fora da start page routes do flow precisam de intent para estar em escopo.

📝 **Como cai:** Documentos com tabelas, imagens ou escaneados e qualidade ruim no RAG.
Como resolver: Escaneado e barato: OCR (até 500 páginas). Estrutura e imagens: layout parser com anotação. Gráficos no RAG Engine: LLM parser, com custo de tokens.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Nenhuma publicação sem revisão e assinatura de um editor humano	**Passo Approval antes do passo que publica**
Coletar um dado do usuário no meio do workflow e depois aprovar	**Request info e, em seguida, Approval**
Enviar um e-mail para cada item de uma lista filtrada	**Filter para filtrar e Loop para iterar**
Workflow disparado por e-mail novo com atraso e fora de ordem	**Event triggers fazem polling: atraso e ordem não garantida**
BigQuery, Cloud Storage ou Cloud SQL não aparecem ao criar pelo chat	**Habilitar as fontes Cloud no modo designer**
Registro que precisa ocorrer em toda execução versus uso opcional pelo agente	**Passo MCP para o garantido; ferramenta MCP no Gemini agent para o opcional**
Roteamento por autenticação sem escrever código	**Handoff rules com condição na variável**
Respostas diferentes para voz e texto sem duplicar agentes	**Modular components com _session.modality**
Exemplo que deve sempre entrar no prompt do playbook	**Always select nesse exemplo e Dynamically select nos demais**
Firewall da origem só libera IPs conhecidos	**Static IP egress**
Diagramas e figuras dos PDFs não são entendidos pelo RAG	**Layout parser com image annotation**
Busca com texto encontrando imagens, vídeos e PDFs	**gemini-embedding-2 multimodal no Agent Retrieval**
Formulário completo deve levar à página de confirmação	**Condition route com transition target**
Flow legado chamado pelo agente falha por permissão	**Dialogflow API Client no Customer Engagement Suite Service Agent**

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

- Garantia nunca vem da instrução do modelo: use passo, callback, handoff rule ou flow.
- Request info coleta dado; Approval exige aprovação antes de agir.
- Incremental sync não traz exclusões nem identidade; isso é full sync (entity e identity sync).
- Fontes Cloud no Workflow Builder só pelo modo designer.
- Silêncio e erro de webhook são eventos, não routes.

Essenciais para a prova (checklist)

- ☐ Sei escolher entre chat agent e workflow pelo tipo de interação e pela necessidade de gatilho e aprovação.
- ☐ Sei diferenciar Condition, Filter, Loop, Request info e Approval.
- ☐ Sei por que event triggers têm atraso e não garantem ordem.
- ☐ Sei usar {{estática}}, {dinâmica}, {@TOOL: nome} e {@AGENT: nome} nas instruções do CX Agent Studio.
- ☐ Sei quando usar handoff rules, callbacks, tools e flows no CX Agent Studio.
- ☐ Sei explicar pages, routes, event handlers e escopo de handlers no Dialogflow CX.
- ☐ Sei distinguir ingestão de federação e full sync de incremental sync, entity sync e identity sync.
- ☐ Sei qual parser do Agent Search usar para PDF escaneado, tabelas e imagens.
- ☐ Sei por que bibliotecas grandes vão para data store e não para a seção Knowledge.
- ☐ Sei qual política da organização restringe os conectores do Gemini Enterprise.

Questões de revisão do capítulo

R2.1 · Criar Agentes com Ferramentas Low-Code

A Fundação Previdência Futuro usa o Gemini Enterprise e quer impedir, em toda a organização, que administradores de apps conectem fontes de terceiros não homologadas, permitindo somente Google Drive, Gmail e SharePoint. Qual controle atende a isso?

A. Model Armor com floor settings aplicadas aos conectores
B. Static IP egress configurado em cada data store
C. Identity sync limitada aos grupos dos administradores
D. Política da organização discoveryengine.managed.allowedDataSources

R2.2 · Criar Agentes com Ferramentas Low-Code

No Dialogflow CX, a equipe da Rede de Cafeterias Grão Nobre tem dezenas de exemplos num playbook e percebe que alguns quase nunca são relevantes, enquanto um exemplo de fluxo de reembolso deve orientar o modelo em toda conversa. Como configurar a estratégia de seleção desses exemplos?

A. Never select no exemplo de reembolso e Always select nos demais
B. Dynamically select em todos, já que só o primeiro é enviado
C. Apagar os exemplos, porque playbooks não usam few-shot
D. Always select no exemplo de reembolso e Dynamically select nos demais

R2.3 · Criar Agentes com Ferramentas Low-Code

A equipe de qualidade da Central de Relacionamento Ouvir Mais acha que os casos de teste de avaliação do agente no CX Agent Studio são frágeis e dão resultados instáveis entre execuções. Qual recurso de IA do CX Agent Studio foi criado para melhorar o rigor e a estabilidade desses casos de teste?

A. Prompt guard
B. Handoff rules
C. Test case hill climbing
D. Restructure instructions

R2.4 · Criar Agentes com Ferramentas Low-Code

A Cooperativa Vale do Café está migrando o atendimento para o CX Agent Studio e ainda mantém alguns flows do Dialogflow CX. Segundo as boas práticas para flow-based agents, qual tarefa continua sendo um bom uso de um flow?

A. Gerar respostas criativas sobre produtos com base no catálogo
B. Responder com uma mensagem fixa de boas-vindas ao abrir a conversa
C. Coletar e validar dados em sequência, como CPF e número do contrato
D. Classificar a intenção do cliente a partir de mensagens livres

Respostas comentadas

R2.1. D) Política da organização discoveryengine.managed.allowedDataSources

Por quê: Cada conector do Gemini Enterprise tem um valor correspondente (por exemplo google_drive, google_mail, sharepoint) na política da organização discoveryengine.managed.allowedDataSources, usada para controlar quais fontes podem ser conectadas.

✗ A) Incorreta. Floor settings definem limiares mínimos de templates do Model Armor, não quais conectores podem ser usados.
✗ B) Incorreta. Static IP egress fixa os IPs de saída para a origem liberar o acesso; não limita conectores.
✗ C) Incorreta. Identity sync atualiza usuários e grupos de ACL; não restringe tipos de fonte.

> **Dica de prova:** Limitar conectores permitidos na organização: allowedDataSources.

R2.2. D) Always select no exemplo de reembolso e Dynamically select nos demais

Por quê: Cada playbook deve ter exemplos de conversas entre usuário e playbook, incluindo ações do agente; eles funcionam como prompts few-shot. A estratégia de seleção (Dynamically, Always ou Never select) controla se o exemplo entra no prompt.

✗ A) Incorreta. Never select tiraria justamente o exemplo que deve sempre aparecer.
✗ B) Incorreta. A seleção dinâmica não envia só o primeiro exemplo; ela escolhe por relevância.
✗ C) Incorreta. Exemplos de playbook são, na prática, exemplos few-shot para o LLM.

> **Dica de prova:** Exemplo que deve estar sempre no prompt: Always select.

R2.3. C) Test case hill climbing

Por quê: Entre os recursos de IA do CX Agent Studio estão criar agente com Gemini, restructure instructions, refine instructions e test case hill climbing, voltados a acelerar o desenvolvimento e melhorar a qualidade.

✗ A) Incorreta. Prompt guard protege contra ataques de prompt.

✗ B) Incorreta. Handoff rules controlam transferências entre agentes.

✗ D) Incorreta. Restructure instructions reformata instruções em XML.

> **Dica de prova:** Casos de teste instáveis no CX Agent Studio: test case hill climbing.

R2.4. C) Coletar e validar dados em sequência, como CPF e número do contrato

Por quê: As boas práticas do CX Agent Studio recomendam flows para lógica determinística que não cabe em callbacks, como coleta sequencial e autenticação, e desaconselham flows para respostas fixas ou classificação de intenção.

✗ A) Incorreta. Geração criativa a partir de catálogo é tarefa do modelo, não de um flow determinístico.

✗ B) Incorreta. Uma resposta fixa é melhor atendida por um callback do agente do CX Agent Studio, segundo as boas práticas.

✗ D) Incorreta. A documentação aponta classificação e detecção de intenção como mau uso de flows; para isso devem ser usados LLMs.

> **Dica de prova:** Flow: coleta e validação determinística. Intenção: LLM. Resposta fixa: callback.

03

CAPÍTULO 3 · 33% DA PROVA

Desenvolver Agentes Personalizados

É o maior domínio, com cerca de um terço da prova, e eu preciso dominar o ADK por dentro (agentes, workflows, grafos, state, callbacks, ferramentas, A2A e MCP), Sessions e Memory Bank, RAG Engine, Vector Search e Agent Retrieval, escolha de modelo e serving, Agent Identity, Agent Registry e os servidores MCP do Google Cloud. As questões costumam pedir a API, o prefixo, o modo ou o serviço exato para um requisito técnico. Onde mais erro é confundir operações do Memory Bank (GenerateMemories, CreateMemory, IngestEvents, RetrieveMemories), inverter consumir e expor A2A (RemoteA2aAgent x to_a2a) e trocar MaaS por self-deploy. Vale ler o requisito com lupa: determinístico, independente, entre sessões, sem provisionar e tráfego previsível são palavras que decidem a resposta.

Neste capítulo

18 tópicos-chave · 14 pegadinhas · 8 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

LlmAgent e workflow agents

LlmAgent é a unidade com modelo, instruções e ferramentas opcionais; com mais de um agente ou nó vira workflow. Workflow agents (Sequential, Parallel, Loop) orquestram de forma determinística, sem consultar o modelo. ParallelAgent roda ramos independentes sem troca automática de estado; LoopAgent precisa de max_iterations ou escalonamento de um subagente para terminar.

Grafos no ADK 2.0

Workflow recebe edges com a sequência de nós (agentes, funções, ferramentas ou outros workflows) e a saída de cada nó vira entrada do próximo; schemas podem definir o contrato. Permite ramificações e funções sem chamar modelo. Nós que emitem RequestInput pausam para entrada humana de forma previsível.

Equipes colaborativas e modos de subagente

O coordenador recebe automaticamente uma ferramenta de delegação com o nome de cada subagente. Modos: chat (padrão, interação completa, retorno manual), task (perguntas de esclarecimento, retorno automático, com input_schema e output_schema) e single_turn (sem interação, retorno automático, pode rodar em paralelo).

AgentTool, sub_agents e A2A

AgentTool envolve um agente local como ferramenta; sub_agents delega com transferência de controle. RemoteA2aAgent consome um agente remoto pelo Agent Card; to_a2a(root_agent) expõe um agente local gerando o Agent Card automaticamente. A2A é indicado para serviço separado, outra equipe, linguagem ou framework, ou contrato formal; subagente local roda no mesmo processo, sem rede.

MCP x A2A e McpToolset

MCP liga o agente a ferramentas, recursos e prompts; A2A liga agentes entre si. O McpToolset descobre ferramentas com list_tools e executa call_tool, por stdio ou HTTP; em produção deve ser definido de forma síncrona no agent.py. Servidor MCP local usa stdio na máquina; remoto roda na infraestrutura do serviço com endpoint HTTP.

Prefixos de state

Sem prefixo: escopo da sessão. user: vale para o usuário em todas as sessões. app: vale para toda a aplicação. temp: existe só na invocação atual, não é persistido e é compartilhado com subagentes chamados na mesma invocação.

Sessions, state, eventos e memória

Sessão é a sequência cronológica de eventos; evento guarda conteúdo e ações; state guarda dados temporários da conversa; memória é informação do usuário entre sessões. Com ADK implantado, as Agent Platform Sessions são gerenciadas automaticamente. No Agents CLI, --session-type aceita in_memory (padrão do protótipo), cloud_sql e agent_platform_sessions.

Memory Bank: geração

Só persiste o que casa com os memory topics (gerenciados como USER_PERSONAL_INFO ou customizados) e consolida com as memórias do escopo. GenerateMemories extrai fatos do histórico na hora; CreateMemory deixa o agente escrever memórias diretamente (memory-as-a-tool); IngestEvents acumula eventos em streams com gatilhos, deduplicação e geração assíncrona; RetrieveMemories só recupera.

Memory Bank: escopo, TTL, revisões, perfis e acesso

O escopo é definido na criação, é imutável e a recuperação exige correspondência exata. TTL expira memórias antigas; memory revisions mostram como cada memória mudou; memory profiles têm esquema fixo e um único perfil por esquema e escopo; IAM Conditions limitam quais principais leem ou gravam cada escopo. Mesmo instruído, pode guardar dado sensível: use SDP e Model Armor.

MemoryService do ADK e Memory Bank x RAG

InMemoryMemoryService não persiste (protótipo); VertexAiMemoryBankService traz memórias extraídas e evolutivas (produção); VertexAiRagMemoryService faz busca vetorial sobre o corpus de conversas. O RAG usa base externa estática; o Memory Bank evolui com o que o agente aprende de cada usuário.

Callbacks do ADK

before_model_callback inspeciona o prompt e, se devolver um LlmResponse, pula a chamada ao modelo. before_tool_callback (tool, args, tool_context) valida ou bloqueia argumentos; after_tool_callback (com tool_response) pode gravar no state. on_tool_error_callback só roda quando a ferramenta falha.

Ferramentas, skills e contexto no ADK

Em function tools, docstring e type hints viram descrição e esquema da ferramenta. Skills (experimentais) seguem a especificação Agent Skills, carregadas pelo SkillToolset com list_skills e load_skill para divulgação progressiva. Artifacts são binários nomeados e versionados por escopo; context caching fica no objeto App com ContextCacheConfig (min_tokens, ttl_seconds); a Context Compaction tem estratégia por tokens, prioritária, e janela deslizante por turnos.

Escolha de modelo e serving

MaaS oferece modelos abertos como APIs serverless com cobrança por uso: ideal para protótipo e tráfego variável. Self-deploy no Model Garden atende pesos customizados, tráfego alto e previsível (menor TCO) e controle de hardware; há containers pré-criados (vLLM, Hex-LLM, TGI) e vLLM customizado. O ADK aceita Gemini, Claude e endpoints por string, wrappers ApigeeLlm e LiteLlm no campo model e hospedagem local com Ollama, vLLM e LiteRT-LM.

RAG Engine

Fluxo: ingestão, transformação (chunks), embedding, indexação (corpus), recuperação e geração. chunk_size padrão de 1.024 tokens e chunk_overlap de 256; chunk menor gera embedding mais preciso. O modelo de embedding (text-embedding-005 padrão, text-multilingual-embedding-002, e5 e multilingual-e5) fica fixo no corpus. RagManagedDb é o padrão: sem provisionamento, KNN padrão e ANN (rebuild após grandes mudanças), cosseno e CMEK; Vector Search, Pinecone e Weaviate ficam sob gestão da empresa.

Reranking e controles do RAG Engine

Ranking API: muito rápida (abaixo de 100 ms), cobrança por requisição, exige a Discovery Engine API. LLM reranker: chamada extra ao Gemini, 1 a 2 s, cobrança por token. O RAG Engine suporta VPC Service Controls e CMEK, mas não residência de dados nem Access Transparency (AXT).

Vector Search e Agent Retrieval

Vector Search 1.0 é índice ANN como serviço com ScaNN; custo de VMs dos índices implantados e de build e atualização. Hybrid search une busca semântica e por tokens para códigos e termos fora do domínio; token restricts e numeric restricts filtram na busca. O Agent Retrieval (antes Vector Search 2.0) traz Collections de Data Objects, schema, embeddings automáticos (como gemini-embedding-001) ou próprios, e o VertexRanker em combine.ranker reordena no servidor os candidatos do RRF.

Agent Identity

Identidade SPIFFE por agente, com X.509 de 24 horas renovado automaticamente; não é compartilhada, não pode ser personificada nem gerar chaves longas. Requer identity_type AGENT_IDENTITY na criação (SDK, --agent-identity ou config do adk deploy); sem isso, segue a conta de serviço. Reimplantar como novo recurso gera novo principal. Recomendados: roles/aiplatform.expressUser, roles/serviceusage.serviceUsageConsumer e roles/browser; papéis Legacy Bucket não são aceitos. Suportada no Agent Runtime, no Gemini Enterprise e no Cloud Run.

Agent Registry e servidores MCP do Google Cloud

O registro cataloga agentes, servidores MCP com ferramentas, endpoints e skills; bindings ligam agentes a destinos. Agentes A2A do Agent Runtime entram sozinhos com skills do Agent Card; componentes de outros projetos ou nuvens entram manualmente como Service resources; standalone skills têm revisões imutáveis. Nos MCP do Google Cloud, o IAM exige mcp.tools.call (roles/mcp.toolUser) mais a permissão do recurso; toolsets reduzem ferramentas e a descoberta é por tools/list e resources/list. Servidor próprio se publica com Apigee ou Cloud Run.

Macetes para memorizar

Prefixos de state

Nada = sessão, user = pessoa, app = todo mundo, temp = agora e nunca mais.

Modos de colaboração

chat conversa e espera, task pergunta e volta sozinho, single_turn não fala com ninguém.

A2A no ADK

to_a2a = TO fora (exponho); Remote = busco o REMOTO (consumo).

Operações do Memory Bank

Generate extrai, Create escreve, Ingest acumula, Retrieve só lê.

Serving de modelos abertos

MaaS para rajada e protótipo; self-deploy para peso próprio, volume previsível e GPU escolhida.

Chunk do RAG Engine

1.024 de chunk, 256 de overlap; menor é mais preciso, maior é mais genérico.

Rerankers

Ranking API é relâmpago (menos de 100 ms); LLM reranker pensa 1 a 2 s e cobra token.

Agent Identity

SPIFFE, 24 horas, sem chave, sem crachá emprestado.

Tabela de memorização: não confunda

Quando é...	...e quando é
RemoteA2aAgent Consumir um agente remoto lendo o Agent Card dele.	**to_a2a()** Expor o meu agente local via A2A, gerando o Agent Card.
Subagente local Módulo interno de baixa latência, mesmo processo, acesso direto ao estado.	**Agente remoto via A2A** Serviço separado de outra equipe, linguagem ou framework, com contrato formal.
GenerateMemories Extrair e consolidar fatos automaticamente do histórico.	**CreateMemory** O agente decide exatamente o que gravar (memory-as-a-tool).
MaaS Protótipo, tráfego variável ou em rajadas, sem operar infraestrutura.	**Self-deploy no Model Garden** Pesos customizados, tráfego alto e previsível com menor TCO, controle de hardware.
Ranking API Latência muito baixa (abaixo de 100 ms) e cobrança por requisição.	**LLM reranker** Reordenação com o Gemini aceitando 1 a 2 s e custo por token.
Modo task Subagente pode fazer perguntas de esclarecimento e volta sozinho com schemas.	**Modo single_turn** Subagente sem interação com o usuário, retorno automático, pode rodar em paralelo.
OAuth 3-legged Agir em nome do usuário, com consentimento dele (ex.: Jira de cada gerente).	**OAuth 2-legged ou OIDC ID token** Agir com a autoridade própria do agente em APIs de parceiros ou backends externos.

Números que caem

Item	Valor
chunk_size padrão do RAG Engine	**1.024 tokens**
chunk_overlap padrão do RAG Engine	**256 tokens**
Latência da ranking API	**abaixo de 100 ms**
Latência do LLM reranker	**1 a 2 segundos**
Context caching do ADK	**modelos como Gemini 2.0 ou superior**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Achar que o ParallelAgent repassa estado entre ramos ou consulta o modelo para decidir a ordem.
Como não cair: Ramos são independentes e a orquestração é determinística; troca de dados deve ser explícita.

⚠ **Alerta de prova:** Confiar que o LoopAgent percebe sozinho quando parar.
Como não cair: Defina max_iterations ou escalonamento a partir de um subagente.

⚠ **Alerta de prova:** Usar RemoteA2aAgent para publicar o próprio agente ou McpToolset para consumir A2A.
Como não cair: RemoteA2aAgent consome, to_a2a expõe; McpToolset é só para servidores MCP.

⚠ **Alerta de prova:** Usar app: para dados da sessão ou temp: para preferências entre sessões.
Como não cair: app: é global da aplicação; temp: some ao fim da invocação; preferências entre sessões usam user:.

⚠ **Alerta de prova:** Achar que o Memory Bank descarta todo dado pessoal ou que TTL curto impede a gravação.
Como não cair: Ele é instruído a evitar, mas pode falhar; TTL só apaga depois. Use SDP e Model Armor.

⚠ **Alerta de prova:** Usar GenerateMemories síncrono ao fim de cada mensagem para evitar processamento excessivo.
Como não cair: IngestEvents acumula eventos com gatilhos e deduplicação e gera de forma assíncrona.

⚠ **Alerta de prova:** Trocar o modelo de embedding de um corpus existente do RAG Engine.
Como não cair: O vínculo é fixo pela vida do corpus; crie outro corpus.

⚠ **Alerta de prova:** Aumentar o chunk_size ou zerar o overlap para melhorar a precisão.
Como não cair: Chunk maior gera embedding mais genérico; o overlap padrão de 256 melhora a recuperação. Para precisão, reduza o chunk.

⚠ **Alerta de prova:** Resolver busca por códigos de produto com chunks maiores, mais dimensões ou reranking.
Como não cair: O modelo de embedding não conhece os códigos; use hybrid search.

⚠ **Alerta de prova:** Achar que o VertexRanker substitui o RRF.
Como não cair: O ranker primário (só RRF) é obrigatório; o VertexRanker reordena depois, no servidor.

⚠ **Alerta de prova:** Esperar que instâncias antigas passem a usar Agent Identity sozinhas no próximo deploy.
Como não cair: Sem identity_type AGENT_IDENTITY na criação, a instância continua com conta de serviço.

⚠ **Alerta de prova:** Reaproveitar o principal de um agente excluído com o mesmo nome de exibição.
Como não cair: Novo recurso gera novo principal: conceda os papéis a ele e limpe os vínculos antigos.

⚠ **Alerta de prova:** Usar tool.isReadOnly ou registro no Agent Registry para o IAM controlar um servidor MCP de terceiros.
Como não cair: O IAM do MCP vale só para servidores do Google Cloud; terceiros se governam com Agent Gateway e Model Armor.

⚠ **Alerta de prova:** Passar o wrapper LiteLlm ou ApigeeLlm no campo tools.
Como não cair: O wrapper vai no parâmetro model do agente.

Como cai na prova

📝 **Como cai:** Arquitetura multiagente descrita em texto, pedindo o padrão de orquestração.
Como resolver: Etapas dependentes e fixas: sequencial. Independentes ao mesmo tempo: paralelo. Refinar até condição: loop com término. Ramificações e pausas humanas: grafo. Decisão flexível: coordenador LLM com subagentes.

📝 **Como cai:** Requisito de memória ou estado com escopo e persistência específicos.
Como resolver: Defina quem vê e por quanto tempo: invocação (temp:), sessão (sem prefixo), usuário entre sessões (user: ou Memory Bank), aplicação (app:). Para o Memory Bank, escolha a operação pela forma de gravação.

📝 **Como cai:** Problema de qualidade na recuperação (RAG ou busca vetorial).
Como resolver: Precisão baixa: chunk menor. Códigos e termos fora do domínio: hybrid search. Filtros de atributo: restricts. Ordem ruim: reranker. Outro idioma: novo corpus com modelo multilíngue.

📝 **Como cai:** Escolha entre MaaS, self-deploy, containers e Gemini gerenciado.
Como resolver: Pesos customizados ou hardware escolhido levam a self-deploy; equipe sem operação e tráfego variável levam a MaaS; tráfego alto e previsível sugere migrar para self-deploy pelo TCO.

📝 **Como cai:** Erro de permissão com Agent Identity, Agent Registry ou MCP do Google Cloud.
Como resolver: Confira se a instância tem identity_type AGENT_IDENTITY, se o principal mudou após reimplantar, se o papel é Legacy Bucket e se há mcp.tools.call mais a permissão do recurso.

📝 **Como cai:** Integração de autenticação para ferramentas externas.
Como resolver: Em nome do usuário: OAuth 3-legged. Autoridade do agente: identidade no Google Cloud para serviços do Google, OIDC ID token para backends, OAuth 2-legged ou API key via auth manager.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Etapas fixas em que cada uma depende da anterior	**Workflow sequencial**
Ramificações por tipo, funções sem modelo e pausa para entrada humana	**Grafo do ADK 2.0 com nó RequestInput**
Subagente que consulta sistemas sem conversar com o usuário	**Modo single_turn**
Especialista usado como ferramenta pelo agente principal	**AgentTool**
Preferência que vale em todas as sessões do usuário	**Prefixo user: no state**
Resposta bruta de API que não deve ser persistida	**Prefixo temp:**
Perfil consistente do cliente lido rápido a cada sessão	**Memory profiles**
Como a memória mudou ao longo do tempo	**Memory revisions**
Arquivo binário versionado pelo nome, por sessão ou usuário	**Artifacts do ADK**
Bloquear o pedido antes de chamar o modelo e responder direto	**before_model_callback devolvendo LlmResponse**
Muitas ferramentas inchando o contexto	**Skills com divulgação progressiva ou toolsets no MCP do Google Cloud**
Filtrar por cidade e faixa de preço na busca vetorial	**Token restricts e numeric restricts**
Agente pai não consegue descobrir nem invocar subagente remoto registrado	**roles/agentregistry.viewer e roles/aiplatform.user na identidade do pai**
Chamada a ferramenta MCP do BigQuery negada	**mcp.tools.call (roles/mcp.toolUser) mais a permissão do dataset**

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

- Workflow agents (Sequential, Parallel, Loop) são determinísticos; LoopAgent precisa de condição de término.
- RemoteA2aAgent consome; to_a2a expõe; AgentTool usa agente como ferramenta.
- Escopo do Memory Bank é imutável e a recuperação exige escopo exato.
- Modelo de embedding do corpus do RAG Engine não muda: crie outro corpus.
- MCP do Google Cloud: mcp.tools.call mais permissão do recurso; IAM não governa MCP de terceiros.

Essenciais para a prova (checklist)

- ☐ Sei escolher entre sequencial, paralelo, loop, grafo e coordenador LLM.
- ☐ Sei os modos chat, task e single_turn e quem volta ao pai automaticamente.
- ☐ Sei quando usar AgentTool, sub_agents, RemoteA2aAgent e to_a2a.
- ☐ Sei os quatro prefixos de state e seus escopos.
- ☐ Sei diferenciar GenerateMemories, CreateMemory, IngestEvents e RetrieveMemories.
- ☐ Sei para que servem TTL, memory revisions, memory profiles e IAM Conditions no Memory Bank.
- ☐ Sei os padrões de chunk do RAG Engine, os rerankers e os controles suportados.
- ☐ Sei quando usar hybrid search, restricts e Agent Retrieval.
- ☐ Sei escolher entre MaaS, self-deploy e containers de serving.
- ☐ Sei provisionar Agent Identity e quais papéis recomendar ou evitar.

Questões de revisão do capítulo

R3.1 · Desenvolver Agentes Personalizados

Ao publicar agentes com o Agents CLI, o time da Rede de Supermercados Bom Dia tem agentes ADK e agentes servidos via A2A. Qual skill do Agents CLI cobre os modos de registro ADK e A2A e a gestão da frota no Agent Registry?

A. google-agents-cli-eval
B. google-agents-cli-observability
C. google-agents-cli-scaffold
D. google-agents-cli-publish

R3.2 · Desenvolver Agentes Personalizados

Na arquitetura da Plataforma Imobiliária Lar Digital, o arquiteto precisa explicar ao time a diferença de papel entre MCP e A2A. Qual descrição está correta?

A. MCP liga agentes entre si; A2A liga o agente a bancos de dados
B. MCP liga o agente a ferramentas e dados; A2A liga agentes entre si
C. A2A substitui o MCP em agentes que rodam no Agent Runtime
D. MCP e A2A são nomes diferentes para o mesmo protocolo de ferramentas

R3.3 · Desenvolver Agentes Personalizados

Ao criar um projeto com agents-cli create, o arquiteto da Seguradora Escudo quer que as sessões fiquem no serviço gerenciado da Agent Platform, e não em memória local nem num banco que ele precise provisionar. Qual valor de --session-type usar?

A. in_memory, que guarda as sessões no processo local
B. agent_platform_sessions, o serviço gerenciado de sessões
C. cloud_sql, que guarda as sessões num banco que você provisiona
D. memory_bank, que transforma cada sessão em memória

R3.4 · Desenvolver Agentes Personalizados

Num workflow em grafo do ADK 2.0 da Rede de Pizzarias Forno Quente, um nó gera o nome de uma cidade, uma função consulta o horário local e outro agente monta a frase final. O desenvolvedor pergunta se precisa gravar cada resultado no state para o próximo nó usar. Qual resposta está correta?

A. Não; o grafo envia todo o histórico da sessão ao próximo nó
B. Sim; nós de grafo só leem dados gravados com o prefixo temp:
C. Sim; é obrigatório usar output_key em todos os nós do grafo
D. Não; a saída de cada nó vira entrada do próximo

Respostas comentadas

R3.1. D) google-agents-cli-publish

Por quê: As skills do Agents CLI são instaladas nos agentes de programação: workflow (sempre ativa), adk-code, scaffold, eval, deploy, publish e observability, cada uma com conhecimento de uma fase do ciclo de vida.

✗ A) Incorreta. A skill de eval trata de métricas, datasets e avaliação.
✗ B) Incorreta. A skill de observability trata de Cloud Trace, logs e integrações de terceiros.
✗ C) Incorreta. A skill de scaffold trata de criar, melhorar e atualizar projetos.

> **Dica de prova:** Registro ADK ou A2A e frota no Agent Registry: skill publish.

R3.2. B) MCP liga o agente a ferramentas e dados; A2A liga agentes entre si

Por quê: Num sistema agêntico, o MCP expõe ferramentas, recursos e prompts aos agentes, enquanto o A2A permite que agentes de serviços, equipes ou linguagens diferentes colaborem por um contrato formal.

✗ A) Incorreta. Inverte os papéis dos protocolos.

✗ C) Incorreta. Os dois coexistem, e o Agent Runtime suporta agentes A2A e ferramentas MCP.

✗ D) Incorreta. São protocolos distintos, com objetivos diferentes.

> **Dica de prova:** MCP: agente com ferramentas. A2A: agente com agente.

R3.3. B) agent_platform_sessions, o serviço gerenciado de sessões

Por quê: No Agents CLI, o tipo de armazenamento de sessão é escolhido na criação do projeto. O modo de protótipo com --yes usa o padrão in_memory; para produção gerenciada, há agent_platform_sessions.

✗ A) Incorreta. in_memory perde as sessões quando o processo reinicia.

✗ C) Incorreta. cloud_sql guarda sessões num banco Cloud SQL, que não é o serviço gerenciado de sessões.

✗ D) Incorreta. memory_bank não é um valor de --session-type.

> **Dica de prova:** Sessões gerenciadas no Agents CLI: --session-type agent_platform_sessions.

R3.4. D) Não; a saída de cada nó vira entrada do próximo

Por quê: No ADK 2.0, Workflow recebe edges com a sequência de nós (agentes, funções, ferramentas ou outros workflows), e schemas de entrada e saída podem definir o contrato entre os nós.

✗ A) Incorreta. O que passa adiante é a saída do nó, não o histórico inteiro.

✗ B) Incorreta. Não há essa exigência de prefixo.

✗ C) Incorreta. output_key não é obrigatório para passar dados entre nós.

> **Dica de prova:** Grafo do ADK: a saída de um nó é a entrada do próximo.

04

CAPÍTULO 4 · 15% DA PROVA

Proteger e Governar Fluxos de Trabalho Agênticos

Neste domínio eu preciso saber como Agent Identity, auth manager, Agent Registry, Agent Gateway, políticas IAM, Principal Access Boundary, Model Armor e Sensitive Data Protection se encaixam numa defesa em camadas. A prova traz um risco concreto (token vazando, agente acessando outra organização, prompt injection, loop gastando tokens, ação fora da intenção do usuário) e pede o controle exato. Onde mais erro é usar Model Armor para algo que é de autorização, ou usar IAM para algo que é de intenção (semantic governance). Também preciso decorar atributos, papéis e modos: tool.isReadOnly, request.auth.oauth.client_id, roles/iap.egressor, principal://, DRY_RUN e ENFORCE, REQUEST_AUTHZ e CONTENT_AUTHZ.

Neste capítulo

18 tópicos-chave · 14 pegadinhas · 8 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Credenciais da Agent Identity

Certificados X.509 provisionados e renovados automaticamente, com validade de 24 horas. Uma política de Context-Aware Access gerenciada pelo Google, ativa por padrão, exige mTLS nas APIs do Google e DPoP além do gateway, vinculando os tokens ao runtime confiável; sair dela não é recomendado. Não há chaves de longa duração nem senhas estáticas.

Auth manager do Agent Identity

Cofre e broker de credenciais de saída. Auth providers definem o tipo de autenticação por destino (API key, OAuth 2-legged ou 3-legged); com 3-legged, gerencia consentimento, visibilidade e revogação, atribuídos ao SPIFFE ID do agente. HTTP basic não é recomendado; se inevitável, guarde como API key no auth manager.

Credenciais de usuário nunca chegam ao agente

Com Agent Identity, Agent Registry e Agent Gateway, as credenciais de usuário dos conectores passam criptografadas pelo auth manager e só são decifradas no gateway. O agente nunca vê o token bruto, e o gateway termina o mTLS e cuida de OAuth 2.0 e Workload Identity Federation.

Agent auth x user auth no ADK

Agent auth usa a identidade própria do agente e serve quando todos compartilham o mesmo acesso. User auth usa o token OAuth do usuário e limita o agente ao que o usuário poderia fazer, sem reimplementar permissões no prompt. Em fluxos delegados, os logs de auditoria mostram a identidade do usuário e a do agente juntas.

Allow, deny e Principal Access Boundary

Allow e deny policies são anexadas a recursos; deny nega um recurso específico apesar de papel amplo. PAB é anexada a conjuntos de principais e limita o que eles podem acessar, mesmo com papéis concedidos fora (por exemplo, recursos de outra organização). PAB só bloqueia permissões presentes na enforcement version escolhida.

IAM para servidores MCP do Google Cloud

Os atributos valem para a permissão mcp.tools.call. Deny policies aceitam tool.isReadOnly, request.auth.oauth.client_id, resource.service e tool.name; allow policies aceitam só resource.service e tool.name. Não controlam servidores MCP de terceiros.

Agent Gateway: caminhos governados

Client-to-Agent (ingress) autentica clientes e inspeciona prompts; Agent-to-Anywhere (egress) avalia IAM, PAB, Model Armor e semantic governance nas chamadas de saída, usando a Agent Identity do agente. No Agent Runtime há os dois; no Gemini Enterprise só egress. Para esses dois, o Google gerencia o gateway; Envoy próprio é o modelo para outras nuvens.

Agent Gateway: recursos para segurança e desenvolvimento

Governança centralizada entre runtimes, guardrails com Model Armor, perímetro VPC-SC por anexos de rede VPC privados e observabilidade com traces e logs. Faz mediação de protocolos (MCP, A2A, REST, gRPC) e dispensa gerenciar roteamento de VPC, Envoy e rotação de certificados mTLS.

Políticas IAM do Agent Gateway

Membro é o principal SPIFFE (principal://), não o urn:agent do registro. Recursos são serviços do Agent Registry; destino não registrado gera NOT_FOUND na criação. O papel é roles/iap.egressor, e o IAP verifica iap.resources.egressViaIAP. DRY_RUN registra violações nos Cloud Audit Logs; ENFORCE bloqueia; teste em DRY_RUN no staging antes.

Extensões de autorização do Agent Gateway

Chamada gRPC em tempo real a um serviço de autorização, que pode inspecionar, modificar ou bloquear o tráfego. REQUEST_AUTHZ envia cabeçalhos (ex.: IAP para identidade); CONTENT_AUTHZ envia o payload completo (ex.: Model Armor para o prompt).

Semantic governance e token management

Semantic governance policies (pré-GA) avaliam em tempo de execução, contra o prompt e o histórico, regras em linguagem natural antes de cada chamada de ferramenta, cobrindo ações que o IAM permite mas fogem da intenção do usuário. Token management policies (acesso antecipado) controlam orçamento de tokens de entrada, saída e cache por agente e departamento e devolvem HTTP 429 ao esgotar; rate limit por requisição não basta.

Model Armor: filtros e limiares

Filtros de IA responsável: discurso de ódio, assédio, sexualmente explícito, perigoso (limiares configuráveis) e CSAM (sempre aplicado, não desliga). Limiares: High (poucos falsos positivos), Medium and above (equilíbrio) e Low and above (máxima detecção, recomendado para prompt injection, não para categorias gerais). Use templates separados para entrada e saída. Também inspeciona texto de PDFs e imagens.

Model Armor: floor settings e integrações

Floor settings na organização e em pastas garantem que templates sejam pelo menos tão restritivos quanto o padrão; no projeto, habilitam aplicação inline em modelos Gemini e servidores MCP do Google Cloud suportados, sem mudar código (MCP de terceiros fica fora). No Agent Gateway, BLOCK no ingresso devolve erro e o prompt nem chega ao agente. Endpoints regionais acessados de dentro da VPC exigem endpoint Private Service Connect.

Model Armor x Sensitive Data Protection

No Gemini Enterprise, as content policies do SDP protegem dados de origem (conectores, apps, notebooks e uploads) e impedem a indexação de dados sensíveis; o Model Armor protege prompts e respostas. Para CPF e outros dados específicos, use um template avançado do SDP com os infoTypes certos.

Agentes ADK registrados no Gemini Enterprise

Processam as consultas no Agent Runtime, então as configurações do Model Armor no console do Gemini Enterprise não os protegem: chame o Model Armor pela API REST no código do agente. O agente recebe o e-mail do usuário e a conexão é compatível com VPC-SC.

Segurança em camadas no ADK

Identidade e autorização, guardrails de entrada e saída, guardrail na ferramenta com política fixada no Tool Context (determinístico, como só SELECT em tabelas permitidas), recursos de segurança do Gemini, callbacks e plugins, Gemini Flash-Lite como guardrail barato, código gerado executado em sandbox, avaliação e tracing e controles de rede com VPC-SC.

Human-in-the-loop

No ADK, Tool Confirmation (experimental) pausa a ferramenta: Boolean Confirmation (sim ou não, require_confirmation=True) ou Advanced Confirmation (texto explicativo e dados estruturados), com diálogo na interface web e resposta remota pela API REST. No Workflow Builder, passo Approval; no Antigravity SDK, ask_user em run_command.

Outros controles de perímetro e visibilidade

APIs agentidentity.googleapis.com e agentidentitycredentials.googleapis.com num perímetro VPC-SC usam o Restricted VIP (restricted.googleapis.com). Decisões do Agent Gateway ficam nos logs gateway_requests (recurso networkservices.googleapis.com/Gateway). A aba Security da Agent Platform exige Security Command Center Premium ou Enterprise. O Memory Bank isola consolidação e recuperação por escopo.

Macetes para memorizar

💡 Quem faz o quê na governança

Identity diz QUEM, Registry diz PARA ONDE, Policies dizem O QUE PODE, Gateway APLICA.

💡 Deny x PAB

Deny tranca a porta do recurso; PAB põe coleira na identidade.

💡 Atributos do MCP

client_id só no deny; allow só conhece serviço e nome da ferramenta.

💡 Credencial da Agent Identity

X.509 vive 24 horas, mTLS e DPoP prendem o token ao runtime.

💡 Limiares do Model Armor

Muito falso positivo: sobe para High. Injection crítico: desce para Low and above.

💡 Caminhos do gateway

Agent Runtime entra e sai; Gemini Enterprise só sai.

💡 Perfis de extensão

REQUEST olha o envelope (cabeçalho); CONTENT abre a carta (payload).

💡 SDP x Model Armor

SDP cuida da despensa (dados de origem); Model Armor cuida da conversa (prompts e respostas).

Tabela de memorização: não confunda

Quando é...	...e quando é
Deny policy Negar um recurso específico que eu controlo, apesar de papel amplo.	**Principal Access Boundary** Limitar pelo lado da identidade o que o agente pode alcançar, inclusive recursos de terceiros que eu não controlo.
IAM (allow, deny, PAB) Controle estático de quem pode acessar o quê.	**Semantic governance** Ação permitida pelo IAM, mas fora da intenção do usuário ou das regras de negócio.

Model Armor Filtrar prompts e respostas contra prompt injection, conteúdo nocivo e vazamento.	**Sensitive Data Protection** Proteger dados de origem e uploads, impedindo indexação de dados sensíveis.
REQUEST_AUTHZ Verificar identidade pelos cabeçalhos (IAP).	**CONTENT_AUTHZ** Inspecionar o payload completo (Model Armor).
DRY_RUN Validar políticas registrando violações sem bloquear.	**ENFORCE** Bloquear de fato depois de validado.
Agent auth Todos os usuários compartilham o mesmo acesso.	**User auth** Cada usuário tem permissões diferentes e o agente deve respeitá-las.
Limiar High Categoria com muitos falsos positivos, como discurso de ódio bloqueando perguntas legítimas.	**Limiar Low and above** Máxima detecção onde o risco é crítico, como prompt injection.

Números que caem

Item	Valor
Validade do certificado X.509 da Agent Identity	**24 horas, renovado automaticamente**
Resposta das token management policies ao esgotar o orçamento	**HTTP 429**
Tokens que uma única requisição pode consumir	**de dezenas a mais de 100.000**
Limiares do Model Armor	**High, Medium and above, Low and above**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Usar deny policy com tool.isReadOnly para evitar que o agente venda quando o cliente só consultou.
Como não cair: IAM é estático e bloquearia ordens legítimas; ação fora da intenção é semantic governance.

⚠ **Alerta de prova:** Usar allow policy com request.auth.oauth.client_id para bloquear um cliente OAuth.
Como não cair: Esse atributo só existe em deny policies.

⚠ **Alerta de prova:** Garantir restrição de SQL com temperatura zero, context caching ou instrução de sistema.
Como não cair: Determinismo vem do guardrail na ferramenta com política no Tool Context.

⚠ **Alerta de prova:** Usar Model Armor para controlar acesso a recursos, gerenciar OAuth ou guardar tokens.
Como não cair: Model Armor filtra conteúdo; autorização é IAM e PAB, credenciais são do auth manager.

⚠ **Alerta de prova:** Guardar tokens ou senhas no Memory Bank ou em variáveis de ambiente.
Como não cair: Memory Bank guarda fatos de conversa; credenciais vão em auth providers do auth manager.

⚠ **Alerta de prova:** Colocar deny policy nos buckets do parceiro para impedir acesso a outra organização.
Como não cair: A empresa não controla os recursos do parceiro; use PAB nas identidades dos agentes.

⚠ **Alerta de prova:** Concluir que a PAB não vale para agentes quando uma permissão passou.
Como não cair: A PAB só bloqueia permissões da enforcement version escolhida; atualize a versão.

⚠ **Alerta de prova:** Usar o urn:agent do Agent Registry como membro da política IAM.
Como não cair: O membro é o principal SPIFFE no formato principal://.

⚠ **Alerta de prova:** Liberar a saída do gateway com regra de firewall na porta 443.
Como não cair: Falta política IAM com roles/iap.egressor no destino registrado.

⚠ **Alerta de prova:** Achar que o Gemini Enterprise tem Client-to-Agent no Agent Gateway.
Como não cair: No Gemini Enterprise o gateway suporta só Agent-to-Anywhere.

⚠ **Alerta de prova:** Colocar Low and above em todos os filtros para máxima proteção.
Como não cair: Em categorias gerais isso bloqueia conteúdo inofensivo; ajuste por categoria.

⚠ **Alerta de prova:** Esperar que o Model Armor do console do Gemini Enterprise proteja o agente ADK registrado.
Como não cair: Para agentes ADK no Agent Runtime, configure o Model Armor no código do agente.

⚠ **Alerta de prova:** Achar que floor settings inline cobrem qualquer servidor MCP.
Como não cair: Cobrem modelos Gemini e servidores MCP do Google suportados; MCP de terceiros precisa de outra proteção.

⚠ **Alerta de prova:** Usar rate limiting por requisições por segundo contra loop que esgota a cota.
Como não cair: Uma requisição pode consumir de dezenas a mais de 100.000 tokens; use token management policies.

Como cai na prova

📝 **Como cai:** Risco descrito em linguagem de negócio e quatro controles de segurança diferentes.
Como resolver: Classifique o risco: quem acessa (IAM, deny, PAB), para onde vai (Registry e gateway), o que é dito (Model Armor, SDP), se a ação faz sentido (semantic governance) ou quanto gasta (token management).

📝 **Como cai:** Política do Agent Gateway que não funciona ou falha na criação.
Como resolver: Confira o membro (principal://), o destino no Agent Registry (NOT_FOUND), o papel roles/iap.egressor e o modo (DRY_RUN só registra).

📝 **Como cai:** Credenciais para ferramentas externas e auditoria de quem agiu.
Como resolver: Use auth providers no auth manager; para acesso delegado, OAuth 3-legged; lembre que os logs trazem usuário e agente e que o token bruto não chega ao agente.

📝 **Como cai:** Ajuste fino do Model Armor ou proteção de dados pessoais.
Como resolver: Templates separados para entrada e saída, limiar por categoria, CSAM sempre ligado, SDP avançado com infoTypes para dados específicos e floor settings para piso e proteção inline.

📝 **Como cai:** Exigência de aprovação humana antes de ação crítica.
Como resolver: Mecanismo determinístico do produto: Approval no Workflow Builder, Tool Confirmation no ADK (avançada se precisar de dados estruturados), ask_user no Antigravity SDK.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Ação permitida pelo IAM, mas fora da intenção do usuário	**Semantic governance policies**
Bloquear um único cliente OAuth nos servidores MCP	**Deny policy com request.auth.oauth.client_id**
Só ferramentas MCP de leitura em toda a organização	**Deny policy com tool.isReadOnly no nível da organização**
Consentimento, visibilidade e revogação de OAuth delegado	**Auth manager com OAuth 3-legged**
Token do usuário não pode chegar ao agente	**Auth manager criptografa, Agent Gateway decifra**

Agente não pode acessar recursos de outras organizações	**Principal Access Boundary**
Saída para servidor MCP bloqueada após ENFORCE	**roles/iap.egressor no destino registrado**
NOT_FOUND ao criar política do gateway	**Destino não registrado no Agent Registry**
Loop consumindo cota compartilhada de tokens	**Token management policies com HTTP 429**
Templates de equipes menos restritivos que o padrão e proteção sem mudar código	**Floor settings do Model Armor**
Erro de certificado ao chamar Model Armor de dentro da VPC	**Endpoint Private Service Connect**
Dado sensível não pode nem ser indexado	**Content policies do Sensitive Data Protection nos conectores**
Usuários com permissões diferentes no mesmo agente	**User auth com o token OAuth do usuário**
Por que o gateway negou a chamada	**Logs gateway_requests no Cloud Logging**

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

- Model Armor filtra conteúdo; nunca concede ou nega acesso a recursos.
- PAB limita a identidade; allow e deny ficam nos recursos.
- request.auth.oauth.client_id só em deny; IAM do MCP só vale para servidores do Google Cloud.
- Política do gateway: principal://, destino no Agent Registry, roles/iap.egressor, DRY_RUN antes de ENFORCE.
- Agente ADK registrado no Gemini Enterprise precisa do Model Armor no próprio código.

Essenciais para a prova (checklist)

- ☐ Sei como mTLS, DPoP e Context-Aware Access protegem as credenciais da Agent Identity.
- ☐ Sei configurar auth providers no auth manager e quando usar OAuth 3-legged.
- ☐ Sei diferenciar allow, deny e Principal Access Boundary e o papel da enforcement version.
- ☐ Sei os atributos de IAM aceitos em deny e em allow para servidores MCP do Google Cloud.
- ☐ Sei os caminhos Client-to-Agent e Agent-to-Anywhere e onde cada um é suportado.
- ☐ Sei montar uma política IAM do Agent Gateway e testar em DRY_RUN.
- ☐ Sei quando usar semantic governance e token management policies.
- ☐ Sei configurar filtros, limiares, templates e floor settings do Model Armor.
- ☐ Sei dividir o trabalho entre Sensitive Data Protection e Model Armor no Gemini Enterprise.
- ☐ Sei as camadas de segurança do ADK, incluindo Tool Context, Flash-Lite e Tool Confirmation.

Questões de revisão do capítulo

R4.1 · Proteger e Governar Fluxos de Trabalho Agênticos

A equipe da Plataforma Pet Amigo quer uma camada extra de segurança no agente ADK que avalie cada entrada do usuário e cada resposta antes de seguir, sem encarecer muito a solução. O que a documentação de segurança do ADK sugere?

A. Copiar cada conversa para revisão humana antes de responder
B. Usar o maior modelo disponível como juiz em cada etapa da conversa
C. Desligar os filtros de conteúdo do Gemini para reduzir a latência
D. Usar o Gemini Flash-Lite como guardrail via callbacks

R4.2 · Proteger e Governar Fluxos de Trabalho Agênticos

Um agente da Imobiliária Lar Seguro acessa a agenda de corretores no Google Calendar por OAuth delegado. O gestor quer que o próprio sistema abra a tela de login e consentimento quando necessário e que seja possível ver e revogar os acessos concedidos por cada corretor. Qual componente oferece isso?

A. Agent Registry, com um binding que guarda a senha de cada corretor
B. Model Armor, com template de dados sensíveis para tokens OAuth
C. Auth manager do Agent Identity, com OAuth 3-legged gerenciado
D. Memory Bank, com memórias que armazenam os tokens de cada usuário

R4.3 · Proteger e Governar Fluxos de Trabalho Agênticos

Para que o Agent Gateway libere as chamadas de um agente da Montadora Ônibus Estrada a um endpoint REST de um parceiro, o que precisa existir na governança da Agent Platform?

A. Uma chave de API do parceiro gravada no código do agente e um firewall liberado
B. Uma memória no Memory Bank com a URL do parceiro e o modo dry-run ativo
C. O endpoint cadastrado no Model Garden e um template do Model Armor criado para esse parceiro
D. O endpoint registrado no Agent Registry e uma política IAM ligando o principal do agente a ele

R4.4 · Proteger e Governar Fluxos de Trabalho Agênticos

A segurança da Distribuidora de Energia Linha Viva exige que o tráfego governado pelo Agent Gateway respeite as regras do perímetro do VPC Service Controls contra exfiltração de dados. Como a documentação descreve essa proteção?

A. Usar o modo dry-run permanente para que o perímetro não bloqueie nada
B. Direcionar o tráfego por anexos de rede VPC privados para aplicar o perímetro
C. Liberar o gateway na internet pública com IPs fixos permitidos no perímetro
D. Desligar o VPC Service Controls nos projetos que usam o Agent Gateway

Respostas comentadas

R4.1. D) Usar o Gemini Flash-Lite como guardrail via callbacks

Por quê: A segurança de agentes no ADK é em camadas: identidade e autorização, guardrails nas ferramentas, recursos de segurança do Gemini, callbacks e plugins, Gemini como guardrail, execução de código em sandbox, avaliação e tracing, e controles de rede.

✗ A) Incorreta. Revisão humana de toda conversa não escala e não é a camada automática sugerida.
✗ B) Incorreta. O maior modelo encarece e aumenta a latência, o oposto do requisito.
✗ C) Incorreta. Desligar filtros reduz a segurança.

> **Dica de prova:** Guardrail barato e rápido: Gemini Flash-Lite em callbacks.

R4.2. C) Auth manager do Agent Identity, com OAuth 3-legged gerenciado

Por quê: O Agent Identity auth manager é cofre e broker de credenciais: com um auth provider OAuth 3-legged, gerencia consentimento e tokens dos usuários, oferece visibilidade e revogação, e os acessos ficam atribuíveis ao SPIFFE ID do agente.

✗ A) Incorreta. Bindings ligam agentes a destinos e auth providers; senhas de usuários não ficam no registro.
✗ B) Incorreta. Model Armor filtra conteúdo; não gerencia consentimento OAuth.
✗ D) Incorreta. Memory Bank guarda fatos de conversas, não credenciais.

> **Dica de prova:** Consentimento, visibilidade e revogação de OAuth delegado: auth manager.

R4.3. D) O endpoint registrado no Agent Registry e uma política IAM ligando o principal do agente a ele

Por quê: O Agent Registry mantém o registro autoritativo de agentes, servidores MCP, ferramentas, skills e endpoints externos e gera os identificadores usados por clientes de descoberta e por políticas de segurança aplicadas pelo Agent Gateway.

✗ A) Incorreta. Chave no código contraria as boas práticas e não cria a autorização no gateway.
✗ B) Incorreta. Memórias não autorizam tráfego, e dry-run não bloqueia nem libera.
✗ C) Incorreta. Model Garden é catálogo de modelos, e Model Armor filtra conteúdo.

> **Dica de prova:** Gateway libera o destino: registrado no Agent Registry + política IAM.

R4.4. B) Direcionar o tráfego por anexos de rede VPC privados para aplicar o perímetro

Por quê: Para equipes de segurança, o Agent Gateway oferece governança centralizada entre runtimes, guardrails de IA com Model Armor, proteção de perímetro com anexos de rede VPC privados e observabilidade com traces e logs.

✗ A) Incorreta. Dry-run registra violações sem bloquear e não substitui o perímetro.
✗ C) Incorreta. Expor o gateway na internet pública contraria a proteção do perímetro.
✗ D) Incorreta. Desligar o VPC-SC remove a proteção contra exfiltração.

> **Dica de prova:** Agent Gateway e VPC-SC: tráfego por anexos de rede VPC privados.

05

CAPÍTULO 5 · 17% DA PROVA

Usar Agentes de Programação no Desenvolvimento de Aplicações

Neste domínio eu preciso conhecer a fundo a customização do Antigravity (rules, skills, hooks, plugins, subagentes, permissões, presets e sandbox), o Antigravity CLI e SDK, o Agents CLI, o Claude Code na Agent Platform, os sandboxes seguros no GKE e no Cloud Workstations e o CodeMender. A prova cobra o nome exato do arquivo, da pasta, do evento de hook ou do comando, então detalhe decorado vale ponto. O erro mais comum é trocar rule por skill, PreToolUse por PostToolUse e configuração global por configuração de workspace. Outra armadilha frequente é achar que rule bloqueia algo: rule orienta, hook e permissão bloqueiam.

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

Tópicos-chave

Rules do Antigravity

Instruções persistentes. AGENTS.md e GEMINI.md não têm frontmatter e ficam sempre ativos (always_on) no escopo da pasta. Arquivos em .agents/rules/ exigem frontmatter YAML com trigger válido (always_on, model_decision, glob ou manual, em snake_case); sem isso são descartados em silêncio, e subpastas exigem registro em .agents/rules.json.

Combinação de rules

Ao ler ou editar um arquivo, o Antigravity sobe da pasta do arquivo até a raiz do workspace e carrega AGENTS.md, GEMINI.md e .agents/rules/ em cada nível, somando às rules globais. Em conflito, vale a do diretório mais específico.

Skills

Pasta com SKILL.md cujo frontmatter tem description (obrigatório) e name (opcional, assume o nome da pasta). Divulgação progressiva: o agente vê nome e descrição e só lê o SKILL.md quando a skill é relevante. No App ficam em .agents/skills/ (workspace, versionado) ou ~/.gemini/config/skills/ (global da máquina).

Hooks

Scripts em pontos do loop: SessionStart, PreToolUse, PostToolUse, PreInvocation, PostInvocation e Stop. Em PreToolUse e PostToolUse o matcher (regex) escolhe as ferramentas; a saída de PreToolUse pode ter decision allow, ask ou deny. Ficam em .agents/hooks.json, ~/.gemini/config/hooks.json ou em plugins, com enabled true por padrão.

Plugins

Pasta com plugin.json (manifesto) e subpastas opcionais skills/, agents/ e rules/, além de mcp_config.json e hooks.json. Instalação pelo Marketplace, no workspace (.agents/plugins/) ou global (~/.gemini/config/plugins/); no CLI, /plugin abre o gerenciador.

Subagentes customizados

Arquivos .md com frontmatter: name e description obrigatórios, tools (lista explícita), mainAgent, subagent, model (inherit, flash ou pro), commandExecutionPolicy (padrão sandbox), mcpServers e skills; o corpo vira o prompt de sistema. Via invoke_subagent, o subagente começa sem o histórico do pai e usa workspace inherit, branch (worktree Git isolada) ou share.

Permissões action(target)

Operações sensíveis são avaliadas em três listas: Deny (bloqueia), Ask (pede aprovação) e Allow (libera), com precedência Deny > Ask > Allow e acima do preset. command(prefixo) casa com comandos que começam pelo prefixo.

Presets do Antigravity

Do mais restrito ao mais permissivo: Request Review (sem sandbox, pede aprovação para tudo), Default (comandos no terminal sandbox; sair dele pede aprovação, salvo regra Allow), Auto (Preview; avalia sozinho comandos fora do sandbox; indisponível para contas business e enterprise) e Turbo (sem sandbox, sem prompt, acesso a todo o sistema de arquivos).

Terminal sandbox

Usa primitivas do sistema operacional (namespaces no Linux, sandbox-exec no macOS), sem VMs nem Docker. Grava na pasta do projeto e em temporários, bloqueia arquivos sensíveis como ~/.ssh e .env e roda sem rede por padrão. Caminhos em read_file entram só leitura, em write_file leitura e escrita, e domínios em read_url entram na allowlist.

Modos do Antigravity CLI

default (pausa para revisar cada edição e pergunta antes de comandos), accept-edits (aprova edições e pergunta antes de comandos), auto (Preview) e plan, alternados com Shift+Tab. /plan gera o artefato Implementation Plan, que pode ser comentado antes do Proceed. O modo headless agy -p roda um prompt e encerra, com resposta em stdout, diagnósticos em stderr e saídas text, json ou stream-json.

MCP no Antigravity CLI

Servidores ficam em ~/.gemini/config/mcp_config.json (global) ou .agents/mcp_config.json (workspace). Remotos exigem Streamable HTTP com o endpoint em serverUrl; HTTP+SSE legado e httpUrl não são aceitos. timeoutSeconds pode ser aumentado, por exemplo para 600.

Antigravity SDK

Motor declarativo de políticas com regras deny, allow e ask_user por ferramenta. Funções Python customizadas e leitura (view_file) são permitidas por padrão; run_command e outras operações de alto risco exigem permissão. policy.safe_defaults pede aprovação em escrita com o handler interativo.

Agents CLI

CLI com sete skills (workflow, adk-code, scaffold, eval, deploy, publish, observability) instaladas por uvx google-agents-cli setup em agentes como Antigravity, Claude Code e Codex. Agent Mode é o consumo por agentes de programação; Human Mode é o uso direto no terminal. Fluxo prototype-first: create --prototype e depois scaffold enhance.

Comandos e extensões do Agents CLI

publish gemini registra no Gemini Enterprise; infra cicd cria o pipeline com --staging-project e --prod-project; infra single-project provisiona com Terraform; scaffold enhance -d troca o deployment_target do agents-cli-manifest.yaml. Extensões em agents-cli-extension.yaml sobrescrevem ou adicionam comandos (run sem shell, AGENTS_CLI_DISABLE_OVERRIDES=1 evita recursão) e distribuem skills para .agents/skills; o Claude Code lê .claude/skills.

Claude Code no Google Cloud

Requer faturamento, API aiplatform.googleapis.com ativa, acesso aos modelos Claude no Model Garden, gcloud e cota. O usuário precisa de roles/aiplatform.user (ou aiplatform.endpoints.predict). Em CI usa CLAUDE_CODE_USE_VERTEX=1, CLOUD_ML_REGION (global ou us) e ANTHROPIC_VERTEX_PROJECT_ID; fixar versões controla migração, pois sem pin o primário padrão pode ser um Opus, mais caro.

Agent Sandbox no GKE

Add-on gerenciado com gVisor e rede Default Deny. CRDs: Sandbox, SandboxTemplate, SandboxClaim e SandboxWarmPool, mais o Sandbox Router. O SandboxClaim referencia um SandboxTemplate; warm pools entregam ambientes em menos de um segundo. No Standard: node pool --sandbox=type=gvisor e cluster com --enable-agent-sandbox; no Autopilot basta a flag na criação.

Cloud Workstations

Ambientes de desenvolvimento gerenciados: clusters definem região e VPC, configurações funcionam como templates (máquina, disco, imagem, IDE) e cada desenvolvedor cria a sua workstation, acessada por navegador, IDE local ou SSH. Não é sandbox para código de usuários nem runtime de produção.

CodeMender

Agente de segurança que encontra, verifica e corrige vulnerabilidades. cm find tem deep scan (padrão, auditoria completa), diff scan (--diff, mudanças em CI/CD, falha com achados CRITICAL ou HIGH) e quick scan (--quick). cm fix gera o patch, aplica numa cópia no sandbox, compila, testa e reexecuta a PoC antes de aplicar ao workspace; --unrestricted desliga as proteções do sandbox.

Macetes para memorizar

💡 Rule x skill x hook

Rule é lei sempre presente, skill é receita sob demanda, hook é segurança armada no evento.

💡 Precedência de permissões

D-A-A: Deny ganha de Ask, que ganha de Allow.

💡 Presets do mais seguro ao mais livre

Request Review, Default, Auto, Turbo: RDAT, do freio de mão ao pé no fundo.

💡 Eventos de hook

Sessão começa (SessionStart), antes e depois da ferramenta (Pre/PostToolUse), antes e depois do modelo (Pre/PostInvocation), fim do loop (Stop).

💡 Onde fica cada coisa

Ponto-agents (.agents/) é do projeto; til-gemini (~/.gemini/config/) é da máquina inteira.

💡 Workspace de subagente

inherit herda, branch isola numa worktree, share divide a pasta.

💡 CRDs do Agent Sandbox

Template é a receita, Claim é o pedido, WarmPool é a fila de pratos já prontos.

Tabela de memorização: não confunda

Quando é...	...e quando é
Rule Restrição ou padrão que deve valer sempre, como usar zod, injetado no contexto.	**Skill** Procedimento de várias etapas, como migrações, carregado só quando relevante.
Hook PreToolUse Barrar ou aprovar antes de a ferramenta rodar (decision deny).	**Hook PostToolUse** Agir depois que a ferramenta rodou, como rodar o Prettier após gravar arquivos.

AGENTS.md na pasta Regra sempre ativa no escopo de uma pasta, sem configurar gatilho.	**.agents/rules/ com frontmatter** Ativação controlada por trigger: glob, model_decision, manual ou always_on.
Configuração de workspace (.agents/...) Vale só para aquele projeto e pode ser versionada com o time.	**Configuração global (~/.gemini/config/...)** Vale para todos os projetos da máquina do usuário.
Preset Default Código não confiável: comandos no sandbox e aprovação para sair dele.	**Preset Turbo** Executar sem sandbox e sem prompt, com acesso a todo o sistema de arquivos.
Agent Sandbox no GKE Executar código não confiável de usuários ou agentes com isolamento de kernel.	**Cloud Workstations** Dar a cada desenvolvedor um ambiente de desenvolvimento padronizado.
cm find --diff Verificar só as mudanças de um PR no CI/CD.	**cm find deep scan** Auditoria completa e prontidão de release.

Números que caem

Item	Valor
Skills instaladas pelo Agents CLI	**7 (workflow, adk-code, scaffold, eval, deploy, publish, observability)**
Exemplo de timeoutSeconds para ferramenta MCP demorada	**600**
Tempo de entrega de ambientes com SandboxWarmPool	**menos de um segundo**
Severidades que reprovam o diff scan do CodeMender no CI	**CRITICAL ou HIGH**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Criar .agents/rules/arquivo.md sem frontmatter achando que fica always_on.
Como não cair: Sem frontmatter com trigger válido a rule é descartada em silêncio; always_on sem frontmatter só vale para AGENTS.md e GEMINI.md.

⚠ **Alerta de prova:** Usar valores camelCase como alwaysOn ou modelDecision no trigger.
Como não cair: São inválidos; o correto é always_on, model_decision, glob ou manual.

⚠ **Alerta de prova:** Proibir comando perigoso com rule always_on.
Como não cair: Rule orienta o modelo; bloqueio determinístico é hook PreToolUse com decision deny ou regra Deny.

⚠ **Alerta de prova:** Usar PostToolUse com deny para impedir uma exclusão.
Como não cair: PostToolUse roda depois que a ferramenta já executou; é tarde demais.

⚠ **Alerta de prova:** Usar o hook Stop para aprovar cada comando.
Como não cair: Stop roda quando o loop termina; decision continue serve para fazer o agente voltar ao loop.

⚠ **Alerta de prova:** Achar que command(*) em Allow vence um Ask, ou que Ask é avaliado por último.
Como não cair: Precedência é Deny > Ask > Allow.

⚠ **Alerta de prova:** Achar que o subagente perdeu o histórico do pai por falha.
Como não cair: Começar sem o contexto do pai é o comportamento esperado; para isolar arquivos use workspace branch.

⚠ **Alerta de prova:** Definir servidor MCP remoto com httpUrl ou HTTP+SSE no Antigravity CLI.
Como não cair: Remoto exige Streamable HTTP com serverUrl.

⚠ **Alerta de prova:** Colocar servidores MCP, hooks ou plugins em GEMINI.md ou em settings.json.
Como não cair: GEMINI.md só contém rules; MCP vai em mcp_config.json, hooks em hooks.json e plugins em .agents/plugins/.

⚠ **Alerta de prova:** Escolher Turbo para acelerar o trabalho com código de terceiros.
Como não cair: Turbo desliga o sandbox e libera tudo sem prompt; o risco é justamente esse.

⚠ **Alerta de prova:** Esperar o preset Auto em conta business ou enterprise.
Como não cair: Auto (Preview) não está disponível para contas business e enterprise.

⚠ **Alerta de prova:** Configurar o Claude Code na Agent Platform com GEMINI_API_KEY, ANTHROPIC_API_KEY ou GOOGLE_GENAI_USE_ENTERPRISE.
Como não cair: O correto é CLAUDE_CODE_USE_VERTEX=1 com CLOUD_ML_REGION e ANTHROPIC_VERTEX_PROJECT_ID.

⚠ **Alerta de prova:** Adicionar capabilities como NET_ADMIN, privileged ou hostPort no Pod do Agent Sandbox.
Como não cair: A política exige runtimeClassName gvisor, automountServiceAccountToken false, runAsNonRoot true e capabilities.drop ALL; capabilities.add e hostPort são rejeitados.

⚠ **Alerta de prova:** Achar que Kata Containers ou Autopilot são obrigatórios para o Agent Sandbox.
Como não cair: O requisito é gVisor; Autopilot é recomendado, mas Standard também funciona.

Como cai na prova

📝 **Como cai:** Cenário pede um comportamento do Antigravity e as alternativas misturam rule, skill, hook e subagente.
Como resolver: Pergunte: precisa bloquear? hook ou Deny. Precisa valer sempre? rule. É procedimento ocasional? skill. É delegar trabalho? subagente.

📝 **Como cai:** Pergunta sobre onde gravar um arquivo de configuração para valer só no projeto ou para todos.
Como resolver: Projeto: pasta .agents/ na raiz do workspace. Máquina: ~/.gemini/config/. Rules em pasta: AGENTS.md ou GEMINI.md.

📝 **Como cai:** Comando bloqueado ou com prompt inesperado no Antigravity.
Como resolver: Verifique o preset (sandbox sem rede no Default), a precedência Deny > Ask > Allow e se o caminho ou domínio está em write_file ou read_url.

📝 **Como cai:** Sequência de comandos do Agents CLI para levar um protótipo a produção.
Como resolver: create --prototype, depois scaffold enhance (alvo e CI/CD), infra cicd com projetos de staging e prod, deploy e por fim publish gemini.

📝 **Como cai:** Requisitos de Pod ou de habilitação do Agent Sandbox no GKE.
Como resolver: Procure gVisor, automountServiceAccountToken false, drop ALL e flags --enable-agent-sandbox e --sandbox=type=gvisor; descarte privileged, capabilities.add e hostPort.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
Regra que vale sempre numa pasta, sem configurar gatilho	**AGENTS.md dentro da pasta**

Barrar um comando de shell antes de executar	**Hook PreToolUse com matcher run_command e decision deny**
Formatar arquivos depois que o agente grava	**Hook PostToolUse**
Distribuir skills, rules, MCP, hooks e subagentes num pacote só	**Plugin com plugin.json**
Muitas skills instaladas sem inchar o prompt	**Divulgação progressiva: nome e description primeiro**
Comando fora do sandbox liberado sem prompt	**Regra command(prefixo) em Allow**
Build precisa baixar pacotes e gravar cache dentro do sandbox	**Domínio em read_url e caminho em write_file**
Aprovar edições automaticamente, mas perguntar antes de comandos	**Modo accept-edits do Antigravity CLI**
Antigravity CLI em pipeline de CI com saída estruturada	**agy -p com --output-format json**
Executar código não confiável com isolamento de kernel e baixa latência	**Agent Sandbox no GKE com gVisor e SandboxWarmPool**
Varredura de vulnerabilidades só nas mudanças do PR	**cm find PATH --diff**
Acrescentar um passo próprio ao deploy do Agents CLI sem fork	**Extensão em agents-cli-extension.yaml sobrescrevendo o deploy**
Registrar o agente implantado para pessoas e agentes o encontrarem	**agents-cli publish gemini**
Ambientes de desenvolvimento padronizados por template	**Cloud Workstations com configurações de workstation**

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

- Arquivo em .agents/rules/ sem frontmatter com trigger válido é descartado em silêncio.
- Para bloquear antes de executar: PreToolUse com decision deny.
- Precedência de permissões: Deny > Ask > Allow, acima do preset.
- Servidor MCP remoto no Antigravity CLI: Streamable HTTP em serverUrl.
- Subagente começa sem o histórico do pai.

Essenciais para a prova (checklist)

- ☐ Sei onde ficam rules, skills, hooks, plugins e mcp_config.json no workspace e no global.
- ☐ Sei os quatro triggers de rules e por que camelCase invalida a rule.
- ☐ Sei os seis eventos de hook e o que decision allow, ask, deny e continue fazem.

- ☐ Sei os quatro presets e o que o terminal sandbox libera e bloqueia.
- ☐ Sei os modos default, accept-edits, auto e plan do Antigravity CLI e o modo headless.
- ☐ Sei montar o frontmatter de um subagente customizado restrito (description, tools explícitas, mainAgent false e model).
- ☐ Sei a sequência create, scaffold enhance, infra cicd, deploy e publish do Agents CLI.
- ☐ Sei configurar o Claude Code com a Agent Platform (API, Model Garden, papel e variáveis).
- ☐ Sei os CRDs e requisitos de Pod do Agent Sandbox no GKE.
- ☐ Sei os três modos do cm find e como o cm fix valida o patch.

Questões de revisão do capítulo

R5.1 · Usar Agentes de Programação no Desenvolvimento de Aplicações

No preset Default do Antigravity, um agente da Fintech Troco Certo precisa rodar npm publish, que exige acesso à rede, fora do terminal sandbox. O que acontece por padrão e como evitar o prompt para esse comando específico?

A. Pede aprovação; só um hook PostToolUse consegue liberar o comando
B. Pede aprovação; uma regra command(npm publish) em Allow libera sem prompt
C. É bloqueado; só o preset Turbo permite comandos fora do sandbox
D. Roda direto, porque o preset Default libera a rede para todos os comandos npm

R5.2 · Usar Agentes de Programação no Desenvolvimento de Aplicações

Um estagiário da Agência Digital Onda Azul mudou o preset de permissões do Antigravity para Turbo num repositório que recebe código de terceiros ainda não revisado. O líder técnico pede para reverter. Qual risco do Turbo justifica o pedido?

A. Desliga o sandbox e libera comandos sem prompt e acesso a todo o sistema de arquivos
B. Exige aprovação para cada edição, o que atrasa o trabalho com terceiros
C. Envia o código do repositório para treinar modelos públicos do Google
D. Bloqueia a leitura de arquivos fora do workspace e quebra o build e os testes de todo o projeto

R5.3 · Usar Agentes de Programação no Desenvolvimento de Aplicações

Uma desenvolvedora da Fábrica de Jogos Pixel Raro usa o Antigravity CLI e quer que as edições de arquivos no workspace sejam aprovadas automaticamente, mas continuar sendo consultada antes de qualquer comando de terminal. Qual modo de execução escolher?

A. auto, que também libera comandos seguros fora do sandbox
B. accept-edits: aprova edições e pergunta antes de comandos
C. default, que mostra o diff e pede aprovação para cada gravação
D. plan, que só analisa e descreve os passos antes de codar

R5.4 · Usar Agentes de Programação no Desenvolvimento de Aplicações

A equipe de plataforma do Grupo Educacional Trilhas quer distribuir, num único pacote para todos os times, as skills internas, um subagente revisor, as rules de estilo, dois servidores MCP e os hooks de segurança do Antigravity. Qual estrutura usar?

A. Um arquivo AGENTS.md global com o conteúdo de cada componente colado
B. Um plugin com plugin.json, skills, agents, rules, MCP e hooks
C. Uma skill única cujo SKILL.md descreve todos os outros componentes
D. Uma extensão do Agents CLI publicada no Agent Registry como endpoint

Respostas comentadas

R5.1. B) Pede aprovação; uma regra command(npm publish) em Allow libera sem prompt
Por quê: No Antigravity, o preset Default executa comandos no terminal sandbox (workspace e temporários, sem rede). Quando um comando precisa de rede ou recursos do host, o agente pede para rodar fora do sandbox, com aprovação, salvo regras command em Allow.

✗ A) Incorreta. PostToolUse roda depois da ferramenta; a liberação é feita por regras de permissão.
✗ C) Incorreta. O Default permite comandos fora do sandbox mediante aprovação.
✗ D) Incorreta. No Default, os comandos no sandbox não têm rede, e sair dele pede aprovação.

> **Dica de prova:** Comando fora do sandbox sem prompt: regra command(...) em Allow.

R5.2. A) Desliga o sandbox e libera comandos sem prompt e acesso a todo o sistema de arquivos
Por quê: Os presets do Antigravity vão do mais restrito ao mais permissivo: Request Review, Default, Auto (Preview) e Turbo. Com código não confiável, presets com sandbox e aprovação reduzem o risco de comandos destrutivos.

✗ B) Incorreta. Pedir aprovação para tudo é o comportamento do Request Review.
✗ C) Incorreta. Essa não é uma característica do preset.
✗ D) Incorreta. O Turbo amplia o acesso; não bloqueia leitura.

> **Dica de prova:** Turbo: sem sandbox e sem prompts; evite com código não confiável.

R5.3. B) accept-edits: aprova edições e pergunta antes de comandos
Por quê: Os modos do Antigravity CLI são default, accept-edits, auto (Preview) e plan, alternados com Shift+Tab. Regras explícitas de allow, ask e deny continuam valendo em todos os modos.

✗ A) Incorreta. auto também aprova automaticamente comandos seguros, o que ela não quer.
✗ C) Incorreta. default pausa para revisar cada edição e pergunta antes de comandos.
✗ D) Incorreta. plan prefixa a instrução /plan para planejar antes de escrever código.

> **Dica de prova:** Editar sem perguntar, mas perguntar antes de comandos: accept-edits.

R5.4. B) Um plugin com plugin.json, skills, agents, rules, MCP e hooks

Por quê: Um plugin do Antigravity é uma pasta com plugin.json (manifesto) e subpastas opcionais skills/, agents/ e rules/, além de mcp_config.json e hooks.json. Pode ser instalado por workspace (.agents/plugins/), globalmente ou pelo Marketplace.

✗ A) Incorreta. AGENTS.md contém rules; não configura MCP, hooks ou subagentes.

✗ C) Incorreta. Uma skill é um pacote de instruções para uma tarefa, não um contêiner de MCP, hooks e subagentes.

✗ D) Incorreta. Extensões do Agents CLI sobrescrevem ou adicionam comandos da CLI; não são registradas como endpoints.

Dica de prova: Distribuir skills, subagentes, rules, MCP e hooks juntos: plugin.

DIA DA PROVA

Estratégia de prova

A prova é de arquiteto: quase toda questão é um cenário de empresa com um requisito decisivo escondido no meio do texto, como determinístico, sem código, sem provisionar, entre sessões, em nome do usuário ou sem mudar o código. As alternativas erradas costumam ser produtos reais do Google Cloud usados fora do seu papel (Model Armor para autorização, Memory Bank para credenciais, Example Store para logs) ou o par certo com papéis invertidos. O guia usa os nomes novos da Gemini Enterprise Agent Platform (Agent Runtime, Agent Search, Agent Retrieval, Agent Platform Evals), e a prova é em inglês, então reconhecer os termos originais é obrigatório. Desenvolver Agentes Personalizados pesa cerca de 33% e Avaliar e Implantar cerca de 22%, somando mais da metade da nota.

Como responder

✓ Gerencie o tempo: são cerca de 80 questões de múltipla escolha em 3 horas, pouco mais de 2 minutos por questão; marque as longas e volte depois.

✓ Antes de ler as alternativas, sublinhe mentalmente o requisito decisivo (garantia, custo, latência, sem código, menor privilégio) e o que já está pronto no cenário.

✓ Elimine primeiro as alternativas que usam um serviço fora do papel dele, como Model Armor para acesso, Memory Bank para tokens ou Example Store para métricas.

✓ Quando houver duas alternativas com os mesmos itens em ordem trocada, decida pelo papel de cada item: a pegadinha de inversão é muito frequente.

✓ Se o enunciado pedir garantia ou determinismo, descarte respostas que dependem de instrução, exemplo ou decisão do modelo.

✓ Em questões de escolha DUAS, confira se cada alternativa marcada é verdadeira sozinha; não escolha duas que se contradizem.

✓ Distribua revisão pelo peso: Desenvolver Agentes Personalizados (~33%) e Avaliar e Implantar (~22%) merecem mais horas que Low-Code (~13%).

✓ Pense nos nomes em inglês como aparecem na documentação: Agent Runtime (antes Agent Engine), Agent Search (antes Vertex AI Search), Agent Retrieval (antes Vector Search 2.0) e Workflow Builder (citado no guia como Agent Designer).

Erros que derrubam candidatos

✗ Confundir autorização (IAM, deny, PAB, roles/iap.egressor) com filtragem de conteúdo (Model Armor) e com intenção (semantic governance).

✗ Escolher a alternativa que reforça a instrução do modelo quando o cenário exige comportamento determinístico.

✗ Inverter pares clássicos: RemoteA2aAgent x to_a2a, PreToolUse x PostToolUse, Request info x Approval, estática x dinâmica, MaaS x self-deploy.

✗ Trocar critérios de avaliação com referência por critérios sem referência e esquecer que os critérios de eficiência são só informativos.

✗ Errar o local de configuração no Antigravity (workspace .agents/ x global ~/.gemini/config/) e o arquivo certo (mcp_config.json, hooks.json, AGENTS.md).

✗ Achar que recursos novos de uma identidade herdam sozinhos papéis antigos: reimplantar como novo recurso gera novo principal.

✗ Estranhar os nomes novos dos produtos e descartar a alternativa certa por não reconhecer Agent Runtime ou Agent Retrieval.

Na semana da prova

- ☐ Refaça as provas simuladas cronometrando o ritmo de pouco mais de 2 minutos por questão e revise cada erro pelo motivo da alternativa errada.
- ☐ Releia as seções de pegadinhas e gatilhos dos cinco domínios, começando por Desenvolver Agentes Personalizados e Avaliar e Implantar.
- ☐ Monte uma folha com os pares que você inverte (A2A, hooks, Memory Bank, critérios do ADK, Cloud Run) e revise todo dia.
- ☐ Revise os nomes em inglês e as equivalências de nomenclatura da Agent Platform para não travar na leitura.

☐ Na véspera, revise só os números e os itens para lembrar de cada capítulo e confirme se a prova será online ou presencial.

REVISÃO FINAL

Folha de cola

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

1. Avaliar e Implantar Fluxos de Trabalho Agênticos

‣ Avaliar contra respostas de referência aprovadas → **final_response_match_v2 ou response_match_score**

‣ Cada afirmação conferida com o contexto das ferramentas → **hallucination ou hallucinations_v1**

‣ Objetivo da conversa inteira cumprido → **multi_turn_task_success_v1**

‣ Mesma ordem de ferramentas, aceitando chamadas extras → **IN_ORDER no tool_trajectory_avg_score**

‣ Reavaliar traces com nova métrica sem reexecutar o agente → **agents-cli eval grade**

⚠ Golden dataset pede métrica com referência: final_response_match (Agents CLI) ou response_match_score e final_response_match_v2 (ADK).

⚠ Essa métrica dispensa referência e exige rubric_groups no caso.

⚠ EXACT é idêntico; IN_ORDER mantém a ordem com extras; ANY_ORDER aceita qualquer ordem com extras.

2. Criar Agentes com Ferramentas Low-Code

‣ Nenhuma publicação sem revisão e assinatura de um editor humano → **Passo Approval antes do passo que publica**

‣ Coletar um dado do usuário no meio do workflow e depois aprovar → **Request info e, em seguida, Approval**

‣ Enviar um e-mail para cada item de uma lista filtrada → **Filter para filtrar e Loop para iterar**

‣ Workflow disparado por e-mail novo com atraso e fora de ordem → **Event triggers fazem polling: atraso e ordem não garantida**

‣ BigQuery, Cloud Storage ou Cloud SQL não aparecem ao criar pelo chat → **Habilitar as fontes Cloud no modo designer**

⚠ Instrução não é determinística. Revisão humana garantida é passo Approval antes do passo que age.

⚠ Filter só filtra listas de dados; não cria etapa humana.

⚠ O webhook gera respostas dinâmicas; quem muda de página é o transition target do handler.

3. Desenvolver Agentes Personalizados

‣ Etapas fixas em que cada uma depende da anterior → **Workflow sequencial**

‣ Ramificações por tipo, funções sem modelo e pausa para entrada humana → **Grafo do ADK 2.0 com nó RequestInput**

‣ Subagente que consulta sistemas sem conversar com o usuário → **Modo single_turn**

‣ Especialista usado como ferramenta pelo agente principal → **AgentTool**

‣ Preferência que vale em todas as sessões do usuário → **Prefixo user: no state**

⚠ Ramos são independentes e a orquestração é determinística; troca de dados deve ser explícita.

⚠ Defina max_iterations ou escalonamento a partir de um subagente.

⚠ RemoteA2aAgent consome, to_a2a expõe; McpToolset é só para servidores MCP.

4. Proteger e Governar Fluxos de Trabalho Agênticos

‣ Ação permitida pelo IAM, mas fora da intenção do usuário → **Semantic governance policies**

‣ Bloquear um único cliente OAuth nos servidores MCP → **Deny policy com request.auth.oauth.client_id**

‣ Só ferramentas MCP de leitura em toda a organização → **Deny policy com tool.isReadOnly no nível da organização**

‣ Consentimento, visibilidade e revogação de OAuth delegado → **Auth manager com OAuth 3-legged**

‣ Token do usuário não pode chegar ao agente → **Auth manager criptografa, Agent Gateway decifra**

⚠ IAM é estático e bloquearia ordens legítimas; ação fora da intenção é semantic governance.

⚠ Esse atributo só existe em deny policies.

⚠ Determinismo vem do guardrail na ferramenta com política no Tool Context.

5. Usar Agentes de Programação no Desenvolvimento de Aplicações

‣ Regra que vale sempre numa pasta, sem configurar gatilho → **AGENTS.md dentro da pasta**

‣ Barrar um comando de shell antes de executar → **Hook PreToolUse com matcher run_command e decision deny**

‣ Formatar arquivos depois que o agente grava → **Hook PostToolUse**

‣ Distribuir skills, rules, MCP, hooks e subagentes num pacote só → **Plugin com plugin.json**

‣ Muitas skills instaladas sem inchar o prompt → **Divulgação progressiva: nome e description primeiro**

⚠ Sem frontmatter com trigger válido a rule é descartada em silêncio; always_on sem frontmatter só vale para AGENTS.md e GEMINI.md.

⚠ São inválidos; o correto é always_on, model_decision, glob ou manual.

⚠ Rule orienta o modelo; bloqueio determinístico é hook PreToolUse com decision deny ou regra Deny.

GABARITO

Respostas do diagnóstico

D1. D) Example Store → revise o capítulo 1

Por quê: A página Scale da Agent Platform traz uma tabela de recursos de segurança: Agent Runtime, avaliação de agentes, Sessions, Memory Bank e Code Execution suportam VPC-SC, CMEK e residência de dados em repouso; o Example Store não.

✗ A) Incorreta. As Sessions suportam VPC Service Controls e CMEK.

✗ B) Incorreta. O Agent Runtime suporta VPC Service Controls e CMEK.

✗ C) Incorreta. O Memory Bank suporta VPC Service Controls e CMEK.

> **Dica de prova:** Exigência de VPC-SC e CMEK: atenção ao Example Store.

D2. B) Aumentar min_instances para cobrir a carga de base e evitar cold starts → revise o capítulo 1

Por quê: Cold start ocorre quando chega uma requisição e não há instância ociosa. A documentação do Agent Runtime recomenda min_instances suficiente para a carga de base e, se possível, carga contínua e previsível, por exemplo com uma fila.

✗ A) Incorreta. Context caching reaproveita contexto de entrada enviado ao modelo; não guarda respostas nem elimina cold start.

✗ C) Incorreta. Um modelo maior tende a aumentar a latência e não resolve o tempo de inicialização das instâncias.

✗ D) Incorreta. Para agentes assíncronos como os do ADK, a recomendação é aumentar container_concurrency, não reduzir; reduzir agrava as filas.

> **Dica de prova:** Primeiras respostas lentas em pico: cold start; ajuste min_instances.

D3. B) Usa Terraform, build de imagem Docker e kubectl apply no cluster GKE → revise o capítulo 1

Por quê: O agents-cli deploy lê o deployment_target do agents-cli-manifest.yaml: agent_runtime (gerenciado, build pelo Dockerfile), cloud_run (gcloud beta run deploy) ou gke (Terraform, build Docker e kubectl apply).

✗ A) Incorreta. Esse é o fluxo do alvo agent_runtime.

✗ C) Incorreta. Esse é o fluxo do alvo cloud_run.

✗ D) Incorreta. Registrar no Gemini Enterprise é tarefa do comando publish, depois da implantação.

> **Dica de prova:** Alvo gke no Agents CLI: Terraform + Docker + kubectl.

D4. D) Processa só as primeiras 500 páginas, e useNativeText mescla o texto nativo → revise o capítulo 2

Por quê: O Agent Search (antes Vertex AI Search) oferece parser digital (padrão), OCR para PDFs digitalizados e layout parser. O OCR é alternativa de menor custo ao layout parser para PDFs escaneados, limitado às primeiras 500 páginas.

✗ A) Incorreta. Para hierarquia complexa, tabelas e elementos visuais, o layout parser tende a dar melhores resultados.

✗ B) Incorreta. Páginas além da 500ª não são processadas, e o texto nativo pode ser aproveitado com useNativeText.

✗ C) Incorreta. O OCR só se aplica a PDFs; os demais tipos de arquivo usam o parser digital.

> **Dica de prova:** OCR para PDFs: até 500 páginas; useNativeText aproveita o texto nativo.

D5. D) Event handlers para eventos internos, como no-input e erro de webhook → revise o capítulo 2

Por quê: No Dialogflow CX, state handlers dividem-se em routes (intent e/ou condição) e event handlers (eventos internos ou customizados). Todo flow tem event handlers para no-match e no-input, e é possível tratar erros de webhook por página.

✗ A) Incorreta. Condition routes avaliam o status da sessão, mas o caso típico de silêncio e erro de webhook é evento.

✗ B) Incorreta. O reprompt de formulário não trata erro de webhook.

✗ C) Incorreta. Intents casam frases do usuário; silêncio e falha de webhook são tratados como eventos.

> **Dica de prova:** Silêncio do usuário ou falha de webhook: event handlers.

D6. B) Gatilhos fazem polling: há atraso e a ordem não é garantida → revise o capítulo 2

Por quê: No Workflow Builder, event triggers executam workflows quando dados mudam em conectores, como novos e-mails, eventos de calendário ou arquivos. O conector precisa estar conectado como data store, e a detecção é por polling.

✗ A) Incorreta. O Gmail está entre os conectores com event triggers, sem limite de uma vez por hora.

✗ C) Incorreta. Aprovação humana é um passo opcional, não requisito para o gatilho.

✗ D) Incorreta. Loop itera sobre listas; não garante a ordem de chegada dos eventos.

> **Dica de prova:** Gatilho atrasado ou fora de ordem no Workflow Builder: polling, por design.

D7. D) Usar Agent Skills com divulgação progressiva de ferramentas → revise o capítulo 3

Por quê: Expor muitas descrições de ferramentas de uma vez gera inchaço de contexto, com mais custo, mais latência e raciocínio pior. Skills implementam divulgação progressiva; no ADK, ferramentas como list_skills e load_skill gerenciam esse carregamento.

✗ A) Incorreta. Repetir as ferramentas em vários agentes não reduz o contexto de cada um.

✗ B) Incorreta. Mais contexto aumenta custo e latência e não resolve a piora de raciocínio.

✗ C) Incorreta. Expor o catálogo inteiro causa justamente o inchaço de contexto.

> **Dica de prova:** Ferramentas demais no contexto: skills com divulgação progressiva.

D8. A) OAuth 3-legged para o Jira e 2-legged para o parceiro → revise o capítulo 3

Por quê: O Agent Identity suporta autoridade delegada do usuário (OAuth 3-legged) e autoridade própria do agente (identidade no Google Cloud, OIDC ID tokens, OAuth 2-legged, API key). Os auth providers do auth manager guardam e obtêm essas credenciais.

✗ B) Incorreta. API key não representa o consentimento de cada gerente, e HTTP basic não é recomendado.

✗ C) Incorreta. OIDC ID token é para o agente se autenticar com a própria identidade, não para agir em nome do usuário no Jira.

✗ D) Incorreta. Inverte os fluxos: o acesso delegado do usuário é 3-legged.

> **Dica de prova:** Em nome do usuário: OAuth 3-legged. Máquina a máquina: OAuth 2-legged.

D9. A) Cada ramo roda de forma independente, sem troca automática de histórico ou estado → revise o capítulo 3

Por quê: O ParallelAgent inicia os subagentes de forma concorrente, e cada um opera em um ramo independente. Por isso, os subagentes devem conseguir trabalhar sem depender uns dos outros; se precisarem trocar dados, isso deve ser feito explicitamente.

✗ B) Incorreta. Não existe essa dependência de um LoopAgent; o LoopAgent serve para repetição até uma condição de término.

✗ C) Incorreta. Workflow agents como o ParallelAgent são determinísticos e não consultam um modelo para orquestrar a execução.

✗ D) Incorreta. O modelo usado por cada subagente não muda o isolamento dos ramos paralelos.

> **Dica de prova:** ParallelAgent: ramos independentes, sem estado compartilhado automático.

D10. A) Guardrail na ferramenta com política fixada no Tool Context → revise o capítulo 4

Por quê: Ferramentas recebem argumentos definidos pelo modelo e Tool Context definido pelo desenvolvedor. Ao ler a política do Tool Context, a ferramenta elimina deterministicamente classes de ações indesejadas, como escrita ou acesso a tabelas não autorizadas.

✗ B) Incorreta. Temperatura baixa reduz variação, mas não garante restrição de comandos ou tabelas.

✗ C) Incorreta. Context caching reduz custo de contexto repetido; não aplica política.

✗ D) Incorreta. Instruções orientam o modelo, mas não impedem de forma determinística um comando proibido.

> **Dica de prova:** Restrição determinística dentro da ferramenta: política no Tool Context.

D11. C) Inserir um passo Approval antes do passo que publica no portal → revise o capítulo 4

Por quê: Human-in-the-loop é parte dos frameworks de segurança de agentes. No Workflow Builder, Request info pede informações e Approval exige aprovação antes de ações críticas, como publicar conteúdo.

✗ A) Incorreta. Depender da instrução do modelo não é determinístico.

✗ B) Incorreta. Filter filtra listas de dados; não cria uma etapa de assinatura humana.

✗ D) Incorreta. Agendar não garante revisão humana.

> **Dica de prova:** Assinatura humana obrigatória antes de agir: passo Approval.

D12. A) Isolamento por escopo: consolidação e recuperação ficam restritas à identidade → revise o capítulo 4

Por quê: Cada memória pertence a um escopo (por exemplo, agente e usuário). RetrieveMemories só devolve memórias com o mesmo escopo, e a consolidação considera memórias daquele escopo, garantindo isolamento entre identidades.

✗ B) Incorreta. A busca por similaridade do Memory Bank é limitada ao escopo informado, não global.

✗ C) Incorreta. Revisões permitem inspecionar como uma memória mudou; não isolam usuários.

✗ D) Incorreta. O TTL expira memórias antigas, mas não impede que dados de pacientes diferentes se misturem.

> **Dica de prova:** Memórias de um usuário não vazam para outro: isolamento por escopo.

Agora, treine no formato real

Faça as 6 provas completas do curso Google Professional Agentic Architect: 300 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 Google Cloud. As questões são inéditas e não reproduzem a prova real.