Strecht Simulados · Guia de preparação



Guia de preparação GH-600

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.

6
capítulos por domínio

36
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 GH-600: 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 120 minutos, no formato do exame. Referência de aprovação: 70%.

Domínio (capítulo)	Peso	Neste guia
1. Gerenciar memória, estado e execução	10–15%	17 conceitos · 13 pegadinhas
2. Implementar guardrails e responsabilização	10–15%	18 conceitos · 14 pegadinhas
3. Implementar o uso de ferramentas e a interação com o ambiente	20–25%	18 conceitos · 14 pegadinhas
4. Orquestrar a coordenação multiagente	15–20%	18 conceitos · 14 pegadinhas
5. Preparar a arquitetura de agentes e os processos do SDLC	15–20%	18 conceitos · 14 pegadinhas
6. Realizar avaliação, análise de erros e ajuste	15–20%	17 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 · Gerenciar memória, estado e execução

Um PR de longa duração acumulou dezenas de comentários de discussão já resolvidos, e o agente vem se confundindo com detalhes antigos. O histórico detalhado não é mais necessário. Qual prática do Learn se aplica?

A. Duplicar os comentários em um arquivo de instruções
B. Resumir o histórico em pontos-chave com referências
C. Guardar o histórico na memória de curto prazo do agente
D. Copiar o histórico inteiro para o prompt da sessão

D2 · Gerenciar memória, estado e execução

Uma sessão de agente que implementava um relatório fiscal na Contábil Norte foi interrompida no meio. No dia seguinte, outra sessão deve continuar sem repetir passos nem contradizer decisões. Qual fluxo de retomada o Learn recomenda?

A. Reverter os commits já feitos e reiniciar a implementação para garantir um estado limpo
B. Abrir o PR existente, ler descrição e issue, ver os commits e a aba Checks e seguir dali
C. Colar no chat da nova sessão o histórico completo da conversa anterior e pedir que continue
D. Abrir um novo PR a partir do branch padrão e refazer a tarefa com o mesmo prompt inicial

D3 · Gerenciar memória, estado e execução

Durante uma sessão, um agente recebe do revisor a orientação de renomear uma variável e usa essa orientação nos próximos passos. Quando a sessão termina, essa informação não fica disponível em uma sessão nova. Que tipo de memória descreve esse contexto?

A. Memória de procedimentos
B. Memória de curto prazo
C. Memória de longo prazo
D. Memória externa

D4 · Implementar guardrails e responsabilização

Um agente propõe alterar um arquivo em .github/workflows/ que muda como o deploy é disparado. O diff tem só três linhas. Pela classificação de risco do Learn, qual controle é recomendado?

A. PR com required checks, sem revisão adicional de donos
B. Aprovação de environment com evidência de auditoria
C. Automação completa, pois o diff é pequeno e reversível
D. CODEOWNERS, aprovações explícitas e checks mais rígidos

D5 · Implementar guardrails e responsabilização

Um agente de triagem lê issues novas, resume o problema, sugere rótulos e recomenda quem deve analisar, mas não pode alterar nada no repositório. Qual nível de autonomia está sendo aplicado?

A. Read-only autonomy
B. Execute with guardrails
C. Human-authorized execution
D. Propose-only autonomy

D6 · Implementar guardrails e responsabilização

Mesmo com branch protection, a Seguradora Elmo quer garantir que aprovações de identidades de automação não contem para liberar o merge de PRs de agentes. O que o Learn sugere como defesa em profundidade?

A. Um hook postToolUse que registra em log as aprovações feitas por bots
B. Um banner no README avisando que bots não devem aprovar pull requests
C. Um workflow de guardrail que só aceita aprovações de usuários humanos
D. Um environment de staging com o próprio agente como revisor obrigatório

D7 · Implementar o uso de ferramentas e a interação com o ambiente

Uma engenheira alterou o copilot-setup-steps.yml em um branch de feature para instalar uma nova ferramenta e quer validar a mudança antes do merge. O que acontece com esse arquivo nesse cenário?

A. Fica parado até que um admin aprove a mudança nas configurações
B. Precisa ser disparado à mão pela aba Actions antes do merge
C. O agente já o usa em sessões iniciadas a partir do branch de feature
D. Roda como check no PR, mas o agente só o usa no branch padrão

D8 · Implementar o uso de ferramentas e a interação com o ambiente

Um engenheiro escreveu um GitHub Agentic Workflow em Markdown com frontmatter definindo gatilhos, permissions, tools e safe-outputs. Como esse arquivo chega a ser executado?

A. Ele vira um custom agent em .github/agents na hora do push
B. Ele é lido pelo Copilot Chat toda vez que alguém abre o repositório
C. Ele roda direto no laptop do autor por meio do Copilot CLI
D. Ele é compilado em um lock file, executado pelo GitHub Actions

D9 · Implementar o uso de ferramentas e a interação com o ambiente

Um workflow agendado roda um agente que cria um branch e faz push de commits usando o GITHUB_TOKEN. A equipe esperava que o workflow de CI com gatilho push rodasse em seguida, mas nada aconteceu. Qual explicação e ajuste são corretos?

A. O CI só roda para humanos; mudar o gatilho para pull_request
B. O runner do agente não tinha acesso de rede; liberar o firewall
C. O push foi em branch novo; esperar a próxima execução agendada
D. O GITHUB_TOKEN não dispara novos runs; usar token de GitHub App

D10 · Orquestrar a coordenação multiagente

Dois agentes da Seguradora Pinho abriram PRs diferentes para corrigir o mesmo alerta de segurança, com abordagens incompatíveis. Como o Learn classifica a situação e o que recomenda?

A. Conflito de política; pedir que um agente aprove o PR do outro para desempatar
B. Esforço duplicado com correções incompatíveis; escalar a decisão para um humano
C. Conflito semântico; mergear os dois e corrigir o comportamento em produção
D. Conflito textual; aceitar o PR que terminar os checks primeiro e fechar o outro

D11 · Orquestrar a coordenação multiagente

Um time criou o agente formatador-changelog para ser chamado pelo orquestrador como subagente e não quer que ele apareça no seletor de agentes dos usuários. Qual configuração usar?

A. disable-model-invocation: true
B. user-invocable: false
C. tools: [] no frontmatter
D. target: vscode no perfil

D12 · Orquestrar a coordenação multiagente

Uma startup coordena seus agentes por um script em uma VM própria que decide a ordem de execução e repassa resultados por mensagens, sem registrar nada no GitHub. Qual é o principal problema apontado pelo Learn nesse desenho?

A. Scripts externos executam mais devagar que workflows do Actions
B. A VM impede que os agentes usem modelos de linguagem recentes
C. O GitHub cobra licenças extras de agentes coordenados fora dele
D. Revisores não conseguem entender por que o sistema agiu como agiu

01

CAPÍTULO 1 · 10–15% DA PROVA

Gerenciar memória, estado e execução

Aqui eu preciso saber onde cada informação do trabalho do agente deve morar e como manter o contexto limpo ao longo do tempo: hierarquia de memória (curto prazo, longo prazo, externa), fonte única da verdade, regras de expiração, poda, resumo e reset, retomada de sessão, context drift e continuidade entre ferramentas. Também cai bastante fato de produto: Copilot Memory (28 dias, citações validadas no branch, quem gerencia o quê), arquivos de instruções (AGENTS.md, .instructions.md, prioridade) e retenção de logs e artefatos. Meu erro mais comum é aceitar respostas que guardam decisão em prompt, chat ou cópias, quando o certo é issue, PR e artefatos duráveis do GitHub.

Neste capítulo

17 tópicos-chave · 13 pegadinhas · 6 macetes · 14 gatilhos · 4 questões comentadas

Tópicos-chave

Memória de curto prazo, longo prazo e externa

Curto prazo é o contexto da tarefa atual e some quando a sessão acaba. Longo prazo é conhecimento curado, estável e resumido, reutilizável entre tarefas (decisões-chave, padrões como usar o padrão repository). Externa são artefatos duráveis no GitHub (issues, PRs, documentação, saídas de workflow), a fonte da verdade persistente e revisável por qualquer pessoa.

Memória limitada ao que afeta o resultado

Guarde requisitos e restrições, decisões que afetam a implementação e a abordagem de validação e testes. Saídas de passos intermediários temporários e cópias do mesmo contexto em vários lugares geram confusão e uso de informação desatualizada.

Tabela de fonte da verdade

Requisitos na issue; decisões no PR ou na documentação; regras de validação nas instruções do repositório; resultados nos logs e artefatos de workflow. Processos repetíveis vão para instruções ou skills reutilizáveis. Cada informação tem um único lugar.

Expiração, poda, resumo e reset

Saídas temporárias (logs, artefatos) devem expirar e artefatos desatualizados devem ser podados. Histórico que não precisa mais de detalhe vira pontos-chave com referências. Reset quando requisitos mudam muito, quando se escolhe nova abordagem ou quando premissas deixam de valer: reescreva a descrição do PR ou feche e recrie o PR, documentando a nova direção.

PR como state anchor

O GitHub reúne commits, checks e discussões no PR. A descrição deve ter a tarefa, os critérios de aceite (ou link da issue), o resumo do plano e atualizações quando decisões mudam. Sumarização boa atualiza a descrição com decisões finais, referencia commits-chave e linka workflow runs em vez de copiar logs.

Retomar sem repetir passos

Como o GitHub preserva commits e workflow runs, a nova sessão abre o PR existente, lê descrição e issue, confere os commits e a aba Checks e continua do último ponto. Reverter commits, refazer a tarefa com o mesmo prompt ou colar o chat antigo são distratores.

Context drift

Acontece quando as ações do agente deixam de seguir o objetivo ou decisões anteriores. Sinais: mudanças que não atendem aos critérios, commits que contradizem decisões e checks falhando ou faltando. Correção: reler a issue e o PR, comparar com os critérios, atualizar a descrição se preciso e revalidar com novos runs ou commits.

Contexto conflitante

É a mesma informação em vários lugares com valores diferentes (issue, prompts salvos, comentário antigo). A prevenção é uma fonte da verdade por tipo e atualizar a original, sem criar cópias nem redefinir critérios em vários prompts.

Contexto obsoleto

Aparece quando o PR está atrás da base, os resultados de workflow não refletem o código atual ou os requisitos mudaram. Previna atualizando o branch com a base, revendo commits e runs recentes e atualizando issues e PRs; branch protection pode exigir branch atualizado.

Continuidade entre ferramentas

Do VS Code para o Copilot CLI, ou de uma máquina para outra, o estado viaja por referências duráveis: número do PR, nome do branch, SHA do commit, link do workflow run e URLs de issue e PR. Resumo verbal, cópia do chat ou variáveis de ambiente locais não servem.

Retenção de logs e artefatos

Logs e artefatos de workflow ficam 90 dias por padrão; a retenção é configurável no repositório, na organização ou na enterprise, de 1 a 90 dias em repositórios públicos e até 400 dias em privados. Artefatos podem ser apagados pela aba Actions e artefatos excluídos não podem ser restaurados, então avalie o impacto na trilha de evidências.

Copilot Memory: fatos e preferências

Fatos do repositório trazem citações para o código e, antes de serem usados, as citações são conferidas no branch atual; se não se confirmam, o fato não é usado. Entradas sem uso são apagadas após 28 dias, e o prazo pode reiniciar quando o Copilot valida e usa a entrada. Fatos ficam restritos ao repositório de origem e não viram preferências nem são gravados em arquivos de instruções.

Copilot Memory: quem usa e quem gerencia

É usado pelo cloud agent, code review, CLI e agentic autofix; o code review usa só fatos do repositório, não preferências. Em planos individuais vem habilitado por padrão; em organização e enterprise o admin precisa habilitar, e cada usuário pode optar por não usar. Donos do repositório revisam e excluem fatos; cada usuário vê e exclui as próprias preferências; em Business e Enterprise, admins também podem ver e excluir preferências.

Arquivos de instruções do repositório

.github/copilot-instructions.md vale para o repositório inteiro; arquivos .instructions.md em .github/instructions usam applyTo para caminhos. Quando os dois casam com o arquivo, o Copilot usa os dois. AGENTS.md pode ficar em qualquer pasta e o mais próximo na árvore tem precedência; como alternativa, um único CLAUDE.md ou GEMINI.md na raiz também vale.

Prioridade de custom instructions

Em conflito entre conjuntos de instruções, a ordem é pessoais, depois repositório, depois organização. A organização tem a menor prioridade.

Prompt files e agent skills

Custom instructions ficam sempre ativas. Prompt files em .github/prompts/*.prompt.md são prompts reutilizáveis usados sob demanda no VS Code e no Visual Studio (não no cloud agent). Agent skills ficam em .github/skills/NOME/SKILL.md e são carregadas quando relevantes no cloud agent, CLI, VS Code e outros, podendo trazer scripts e modelos.

Contexto cheio no Copilot CLI

O CLI compacta automaticamente o histórico quando a conversa se aproxima do limite de tokens e oferece /compact para resumir sob demanda sem encerrar a sessão. --continue retoma uma sessão anterior e /agent escolhe um custom agent; nenhum dos dois libera contexto.

Macetes para memorizar

Hierarquia de memória

Curto some, longo resume, externo comprova.

Fonte da verdade

Requisito na Issue, Decisão no PR, Regra nas Instruções, Resultado nos Artefatos: I-P-I-A.

Manutenção de memória

EPRR: Expirar, Podar, Resumir, Resetar.

Copilot Memory

28 dias sem uso, some; citação que não confere no branch, não entra.

Retenção

90 padrão, 400 no privado, e apagado não volta.

Prioridade de instruções

Eu, minha casa, meu bairro: pessoal > repositório > organização.

Tabela de memorização: não confunda

Quando é...	...e quando é
Memória de longo prazo Conhecimento curado e reutilizável entre tarefas, como um padrão de acesso a dados decidido pela equipe.	**Memória externa** Fonte da verdade durável e revisável por todos: issues, PRs, documentação e saídas de workflow.
Regra de reset Requisito ou premissa mudou e o plano antigo deixou de valer.	**Regra de expiração ou resumo** Saídas temporárias antigas (expirar e podar) ou histórico longo que não precisa mais de detalhe (resumir com referências).
Prompt file (.prompt.md) Texto reutilizável disparado sob demanda no VS Code e no Visual Studio.	**Agent skill (SKILL.md)** Procedimento com script e modelo carregado quando relevante no cloud agent, CLI e VS Code.
Contexto conflitante Valores diferentes da mesma informação em vários lugares; uma fonte da verdade e atualizar a original.	**Contexto obsoleto** PR atrás da base ou runs de commits antigos; atualizar o branch e revisar commits e runs recentes.

Fatos do repositório no Copilot Memory Gerenciados pelos donos do repositório, restritos a ele e usados também pelo code review.	**Preferências do usuário** Pertencem ao usuário, que vê e exclui; o code review não as usa.
copilot-instructions.md mais .instructions.md Geral e específico por caminho casando com o mesmo arquivo: os dois valem.	**Vários AGENTS.md** Vale o AGENTS.md mais próximo do arquivo na árvore.

Números que caem

Item	Valor
Expiração de entrada do Copilot Memory sem uso	**28 dias**
Retenção padrão de logs e artefatos de workflow	**90 dias**
Retenção configurável em repositório público	**1 a 90 dias**
Retenção máxima em repositório privado	**Até 400 dias**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Copiar o histórico inteiro ou a decisão para o prompt de cada nova sessão.
Como não cair: Prompt e chat não são duráveis nem compartilhados; registre decisões na issue ou no PR.

⚠ **Alerta de prova:** Aumentar a retenção para nenhum resultado se perder resolve o contexto velho.
Como não cair: Reter mais aumenta o contexto obsoleto; o certo é expirar saídas temporárias e podar artefatos.

⚠ **Alerta de prova:** Artefato excluído volta após o ciclo de 90 dias ou segue disponível pela API.
Como não cair: Artefato excluído não pode ser restaurado; a exclusão é feita na aba Actions por quem tem permissão.

⚠ **Alerta de prova:** Padrão de 400 dias ou 30 dias fixos de retenção.
Como não cair: Padrão de 90 dias; configurável de 1 a 90 em públicos e até 400 em privados, no repositório, na org ou na enterprise.

⚠ **Alerta de prova:** Fato do Copilot Memory é aplicado mesmo se a função citada foi removida no branch.
Como não cair: As citações são validadas no branch atual; sem confirmação o fato não é usado.

⚠ **Alerta de prova:** Fato sem uso fica arquivado esperando um admin aprovar a remoção ou vira instrução do repositório.
Como não cair: Após 28 dias sem uso ele é apagado automaticamente.

⚠ **Alerta de prova:** O AGENTS.md da raiz prevalece por estar no nível mais alto.
Como não cair: A regra é de proximidade: vale o AGENTS.md mais próximo do arquivo editado.

⚠ **Alerta de prova:** O .instructions.md específico substitui o copilot-instructions.md geral.
Como não cair: Quando os dois casam com o caminho, ambos são considerados.

⚠ **Alerta de prova:** Organização acima de repositório na prioridade de instruções.
Como não cair: Pessoais > repositório > organização.

⚠ **Alerta de prova:** CLAUDE.md precisa ser renomeado para copilot-instructions.md ou movido para .github/instructions.
Como não cair: Um CLAUDE.md ou GEMINI.md na raiz já é aceito como instruções de agente.

⚠ **Alerta de prova:** Copilot Memory em org não funciona porque exige GHAS, repositório público ou arquivo memory.md.
Como não cair: Em organização e enterprise o admin precisa habilitar o recurso primeiro.

⚠ **Alerta de prova:** Registrar critérios de aceite no Copilot Memory para substituir a issue e o PR.
Como não cair: Requisitos ficam na issue; o Copilot Memory não substitui a fonte da verdade.

⚠ **Alerta de prova:** Usar --continue ou /agent para liberar janela de contexto no CLI.
Como não cair: Para resumir o histórico atual sem encerrar a sessão, use /compact.

Como cai na prova

📝 **Como cai:** Onde guardar determinada informação (regra de validação, requisito, decisão, resultado, processo repetível).
Como resolver: Aplique a tabela de fonte da verdade e elimine qualquer alternativa com prompt, chat, terminal local ou cópia em vários lugares.

📝 **Como cai:** Classificar o tipo de memória de um cenário.
Como resolver: Pergunte: some com a sessão (curto prazo)? é conhecimento curado reutilizável (longo prazo)? é artefato durável no GitHub que serve de fonte da verdade (externa)?

📝 **Como cai:** Agente desalinhado, contexto velho ou valores divergentes: o que fazer?
Como resolver: Diferencie drift (reler fonte, comparar com critérios, revalidar), obsoleto (atualizar branch com a base) e conflitante (uma fonte, atualizar a original); se o requisito mudou de verdade, é reset documentado no PR.

📝 **Como cai:** Fato de produto sobre Copilot Memory ou arquivos de instruções.
Como resolver: Lembre os números e regras exatos: 28 dias, citação validada no branch, code review só com fatos, admin habilita em org, AGENTS.md mais próximo, ambas as instruções valem, pessoais acima de tudo.

📝 **Como cai:** Retomar ou continuar trabalho em outra sessão ou ferramenta.
Como resolver: A resposta sempre ancora no PR existente e em referências duráveis (PR, branch, SHA, run), nunca em recomeçar ou colar conversa.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
informação que some quando a sessão termina	**Memória de curto prazo**
padrão consolidado para reaproveitar em tarefas futuras	**Memória de longo prazo**
fonte persistente e revisável por qualquer pessoa	**Memória externa em issues, PRs e documentação**
regras de validação diferentes conforme a sessão	**Instruções do repositório**
mesmo passo a passo colado no prompt a cada release	**Instruções ou skills reutilizáveis**
critérios com valores diferentes em issue, prompts e comentários	**Uma fonte da verdade; atualizar a original**
requisito mudou significativamente no meio da tarefa	**Reset: reescrever a descrição ou recriar o PR documentando a nova direção**
sessão interrompida retomada no dia seguinte	**Abrir o PR existente, issue, commits e aba Checks**
PR aberto há semanas e main mudou muito	**Atualizar o branch com a base e revisar commits e runs recentes**
continuar no CLI em outra máquina	**Referências duráveis: PR, branch, SHA, link do run**
monorepo com AGENTS.md na raiz e na pasta do serviço	**Vale o mais próximo**
fato de memória não validado nem usado há 28 dias	**Apagado por expiração**

histórico do CLI ocupando a janela de contexto	**/compact**
manter logs por 180 dias em repositório privado	**Possível: privado vai até 400 dias**

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

- Decisão só no chat ou no prompt se perde: registre na issue ou no PR.
- Cada informação tem um único lugar; não crie cópias.
- Artefato excluído não pode ser restaurado.
- Copilot Memory: 28 dias sem uso, citações conferidas no branch atual, code review só usa fatos.
- AGENTS.md mais próximo vence; instruções gerais e específicas casando valem juntas.

Essenciais para a prova (checklist)

- ☐ Sei classificar um cenário como memória de curto prazo, longo prazo ou externa.
- ☐ Sei a tabela de fonte da verdade (requisitos, decisões, regras de validação, resultados).
- ☐ Sei as regras de expiração, poda, resumo e reset e quando cada uma se aplica.
- ☐ Sei o que a descrição do PR deve ter para funcionar como state anchor.
- ☐ Sei identificar e corrigir context drift, contexto conflitante e contexto obsoleto.
- ☐ Sei compartilhar estado entre ferramentas por referências duráveis.
- ☐ Sei os prazos de retenção de logs e artefatos e que artefato excluído não volta.
- ☐ Sei como o Copilot Memory valida, expira, quem o usa e quem gerencia fatos e preferências.
- ☐ Sei a precedência entre AGENTS.md, copilot-instructions.md, .instructions.md e instruções pessoais e de organização.
- ☐ Sei diferenciar custom instructions, prompt files e agent skills.

Questões de revisão do capítulo

R1.1 · Gerenciar memória, estado e execução

Um fato do Copilot Memory sobre o comando de build de um repositório não foi validado nem usado por sessão alguma nos últimos 28 dias. O que acontece com ele?

A. Ele é promovido para as instruções do repositório
B. Ele vira preferência do último usuário que o usou
C. Ele fica arquivado até um admin do repositório aprovar a remoção
D. Ele é apagado pela regra de expiração por falta de uso

R1.2 · Gerenciar memória, estado e execução

Um repositório já mantém um CLAUDE.md na raiz com convenções usadas por outra ferramenta de IA. O time quer que o Copilot aproveite esse arquivo como instruções de agente sem duplicar o conteúdo. O que a documentação prevê?

A. O Copilot aceita um CLAUDE.md ou GEMINI.md na raiz como alternativa
B. O arquivo precisa ser renomeado para copilot-instructions.md
C. O arquivo só é lido se ficar dentro de .github/instructions
D. O Copilot desconsidera arquivos de outras ferramentas de IA

R1.3 · Gerenciar memória, estado e execução

Uma líder técnica quer saber como o Copilot Memory influencia o Copilot code review em comparação com o cloud agent. Qual afirmação está correta?

A. O code review usa só as preferências pessoais do autor do PR revisado
B. O code review usa fatos de outros repositórios da org
C. O code review usa fatos do repositório, mas não preferências
D. O code review não usa memória, só o cloud agent faz isso

R1.4 · Gerenciar memória, estado e execução

Um agente lê relatórios de teste antigos guardados como artefatos e acaba usando resultados que não refletem mais o código. Que regra de manutenção de memória do Learn previne esse problema?

A. Aumentar a retenção dos artefatos para que nenhum resultado se perca
B. Definir expiração de saídas temporárias e podar artefatos obsoletos
C. Copiar os relatórios antigos para o arquivo de instruções do repositório
D. Mover os relatórios antigos para a memória de curto prazo do agente

Respostas comentadas

R1.1. D) Ele é apagado pela regra de expiração por falta de uso

Por quê: Entradas do Copilot Memory sem uso são apagadas após 28 dias; o prazo pode reiniciar quando o Copilot valida e usa a entrada. É uma regra de expiração que evita memória obsoleta.

✗ A) Incorreta. O Copilot Memory não grava em arquivos de instruções.
✗ B) Incorreta. Fatos do repositório não viram preferências de usuário.
✗ C) Incorreta. A remoção é automática; não depende de aprovação.

> **Dica de prova:** Copilot Memory: 28 dias sem uso, some.

R1.2. A) O Copilot aceita um CLAUDE.md ou GEMINI.md na raiz como alternativa

Por quê: Para instruções de agente, a documentação aceita AGENTS.md em qualquer pasta ou, alternativamente, um único CLAUDE.md ou GEMINI.md na raiz do repositório.

✗ B) Incorreta. Não é preciso renomear o arquivo.
✗ C) Incorreta. Essa pasta é para arquivos .instructions.md.
✗ D) Incorreta. CLAUDE.md e GEMINI.md na raiz são aceitos.

> **Dica de prova:** CLAUDE.md ou GEMINI.md na raiz também valem como instruções de agente.

R1.3. C) O code review usa fatos do repositório, mas não preferências

Por quê: O Copilot Memory é usado pelo cloud agent, code review, CLI e agentic autofix, com conhecimento compartilhado entre eles; o code review usa só fatos do repositório, não preferências de usuário.

✗ A) Incorreta. É o contrário: preferências de usuário ficam de fora no code review.
✗ B) Incorreta. Fatos ficam restritos ao repositório de origem.
✗ D) Incorreta. O code review está entre os recursos que usam o Copilot Memory.

> **Dica de prova:** Code review + Memory: fatos do repositório, sem preferências.

R1.4. B) Definir expiração de saídas temporárias e podar artefatos obsoletos

Por quê: Algumas informações devem expirar, sobretudo saídas temporárias como logs e artefatos; artefatos desatualizados ou desnecessários devem ser removidos para evitar contexto obsoleto.

✗ A) Incorreta. Reter mais tempo aumenta a chance de contexto obsoleto.
✗ C) Incorreta. Instruções guardam regras reutilizáveis, não resultados de execuções passadas.
✗ D) Incorreta. Trazer dados velhos para o contexto atual agrava o problema.

> **Dica de prova:** Artefato velho confunde: expire e pode.

02

CAPÍTULO 2 · 10–15% DA PROVA

Implementar guardrails e responsabilização

Neste domínio eu preciso dimensionar autonomia pelo risco e colocar humanos só onde a decisão importa: os quatro níveis de autonomia, a tabela Low, Medium, High e Critical com o controle de cada um, a governança em camadas (pré-ação, em ação, pós-ação) e os guardrails nativos do cloud agent. A prova traz o cenário de uma ação e pede o nível ou o controle, ou mostra uma política que existe só no papel. Onde mais erro é julgar risco pelo tamanho do diff (alterar .github/workflows/ com três linhas é High) e confundir os controles de High (CODEOWNERS) e Critical (environment). Também caem detalhes como ask virar deny no cloud agent, Require review from Code Owners e Agents secrets.

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

Tópicos-chave

Quatro níveis de autonomia

Read-only autonomy: inspeciona e recomenda sem alterar nada (triagem que sugere rótulos). Propose-only: cria branches e PRs. Execute with guardrails: dispara workflows pré-aprovados (testes, builds, deploy em staging) dentro de limites, sem aprovação a cada vez. Human-authorized execution: ações de alto impacto (deploy de produção, mudança de workflow, acesso a segredos) só depois de aprovação humana explícita.

Classificação de risco e controles

Low (docs, formatação): automação completa depois dos checks. Medium (atualização de dependências, refatoração segura): PR com required checks. High (.github/workflows/, infra/, security/): CODEOWNERS, aprovações explícitas e checks mais rígidos. Critical (deploy e segredos de produção): gate de environment com revisores explícitos e evidência de auditoria.

Fatores de risco

Além da ação, considere onde a mudança é aplicada, quão reversível é e quão rápido afeta usuários. Small diff, big consequence: editar README é baixo risco, mas três linhas em .github/workflows/ afetam o pipeline inteiro. Modelo, tokens, tamanho do diff ou linguagem não definem risco.

Autonomia progressiva

Exemplo do Learn em quatro fases: só abre PRs; mescla PRs de baixo risco após checks; deploy automático em staging; produção apenas com aprovação de environment. Segredos de produção e alteração livre de workflows nunca viram autonomia sem aprovação.

Human-in-the-loop seletivo

Humanos nos pontos de decisão (merge, deploy, acesso a segredo), não em cada passo como leitura, busca, análise, formatação ou checks de rotina. Aprovar tudo gera governance fatigue sem reduzir risco relevante.

Governance fatigue e rubber-stamping

Para reduzir o cansaço sem perder supervisão: exigir evidências (testes, scans, artefatos), rotear revisões por CODEOWNERS, manter PRs pequenos e exigir plano de rollback em mudança de alto impacto. Rubber-stamping é aprovar no automático; mitigação é melhor evidência, CODEOWNERS e PRs menores.

Anti-patterns de governança

Unbounded autonomy (deploy de produção sem aprovação), excess permissions (tokens amplos), missing audit trail, bypass paths (push direto no main, atalho por rótulo urgente, bypass para o bot) e rubber-stamping. Controls in docs only também é anti-pattern: o que a plataforma não impõe deve ser tratado como opcional e será contornado.

Governança em camadas

Restrições pré-ação são permissões; validação em ação são checks; rastreabilidade pós-ação são logs e artefatos. Para agentes locais, bloquear comando destrutivo é regra de deny em hook preToolUse; banner e log não impedem execução.

CODEOWNERS só bloqueia com exigência de revisão

Sem a regra que exige revisão de code owners no ruleset ou na branch protection, o CODEOWNERS é apenas consultivo. Para o cloud agent, o guia de guardrails recomenda proteger .github/agents, .github/hooks, copilot-setup-steps.yml e configurações de MCP com CODEOWNERS e Require review from Code Owners.

Regras do branch protegido e workflows de guardrail

PR obrigatório, required checks, aprovações por risco, revisão de CODEOWNERS, restrição de push direto e bloqueio de force push e exclusão do branch; a aprovação de quem pediu o PR não deve contar como revisão válida. Workflows de guardrail impõem o que a configuração sozinha não expressa, como contar só aprovações de usuários humanos. Renomear ou remover um required check é mudança de alto risco: drift do check vira drift da governança.

Evidência ausente é falha

Se uma mudança não pode ser auditada, não deve ser mergeada, mesmo com checks verdes. Workflows críticos devem gerar resultados de teste, relatórios de scan, plan.json e execution-report. Em vez de narrativa de risco em texto livre, o plano grava risk em plan.json e o workflow roteia o que não é low para um job com environment de aprovação.

Menor privilégio no Actions

permissions: contents: read como padrão no workflow e elevação só no job que escreve; pull-requests: write é capacidade elevada, para criar ou atualizar PRs. Segredos de produção restritos ao environment de produção com revisores obrigatórios; deploys de produção serializados com concurrency group production. No copilot-setup-steps.yml, declare permissões mínimas no job em vez de herdar o padrão de leitura e escrita.

Ferramentas de agentes de terminal

Acesso a ferramentas é a principal fronteira de governança: libere leitura e mantenha escrita e comandos sob controle explícito, proporcional à consequência. Exemplo: copilot -p 'Resuma os últimos 10 commits' --allow-tool 'shell(git)'; com --allow-all-tools o Copilot ganha o mesmo acesso do usuário.

Hooks no cloud agent

No cloud agent, permissionDecision ask em preToolUse é tratado como deny, porque não há usuário para responder. O arquivo de hooks precisa estar no branch padrão e só o campo bash é honrado.

Mitigações nativas do cloud agent

Filtragem de caracteres ocultos contra prompt injection; commits com o desenvolvedor como coautor, assinados pelo Copilot e com link para os session logs; workflows do Actions bloqueados até Approve and run workflows; push restrito a um branch; altera só o repositório onde a tarefa começou; não aprova nem mescla PRs. O admin pode desligar Require approval for workflow runs em Settings > Copilot > Cloud agent, aceitando que código não revisado ganhe acesso de escrita via workflows.

Onde e em que ambiente o cloud agent roda

Na organização, Settings > Copilot > Cloud agent > Repository access aceita All repositories (padrão), Selected repositories ou No repositories; piloto em poucos repositórios é Selected repositories. Para ambiente limpo por sessão, runners hospedados pelo GitHub ou self-hosted efêmeros; self-hosted persistente acumula resíduos e macOS self-hosted não é suportado.

Agents secrets x secrets do Actions

O que o agente precisa vai em Agents secrets and variables (ex.: chave de leitura de API para um MCP server); o que o Copilot não deve acessar fica como secret ou variável do Actions (ex.: chave de deploy de produção). Segredo nunca vai em arquivo de instruções nem em texto puro no YAML.

Content exclusion e tarefas ambíguas

Content exclusion faz o Copilot não usar o conteúdo excluído, mas o agent mode do VS Code e de outros editores não a suporta e ela não vale para symlinks; não serve como guardrail principal de agentes. Tarefa sem critérios de aceite, restrições e não objetivos, arquivos-alvo e expectativa de validação não deve rodar de forma autônoma.

Macetes para memorizar

💡 Níveis de autonomia

Ler, Propor, Executar com trilho, Executar com autorização: LPEE, do mais fraco ao mais forte.

💡 Risco x controle

Low automatiza, Medium pede PR, High chama os donos, Critical tranca no environment.

💡 Fatores de risco

Onde, volta e velocidade: onde muda, se dá para voltar e quão rápido chega ao usuário.

💡 Camadas de governança

Antes permissão, durante check, depois log e artefato.

💡 Cadência de governança

Semana olha falhas e violações; mês olha permissões e segredos; trimestre olha rulesets, CODEOWNERS, revisores e retenção.

💡 Autonomia progressiva

PR, merge pequeno, staging, produção com porteiro.

💡 Hooks no cloud agent

Ninguém para perguntar: ask vira deny.

Tabela de memorização: não confunda

Quando é...	...e quando é
Execute with guardrails Workflows pré-aprovados (testes, builds, staging) disparados sem aprovação a cada vez.	**Human-authorized execution** Ações de alto impacto (produção, workflows, segredos) só após aprovação explícita.
Read-only autonomy Inspeciona e recomenda sem tocar no repositório.	**Propose-only autonomy** Já cria branches e PRs, mas não executa ações de alto impacto.
Controle High CODEOWNERS, aprovações explícitas e checks mais rígidos (workflows, infra/, security/).	**Controle Critical** Gate de environment com revisores explícitos e evidência de auditoria (deploy e segredos de produção).

Controle Low Automação completa depois dos checks (docs, formatação).	**Controle Medium** PR com required checks (dependências, refatoração segura).
Hook preToolUse com deny Bloquear tecnicamente o comando antes da execução.	**Banner, log ou instrução** Apenas orientação ou rastreabilidade; não impede nada.
Agents secrets Valores que o agente precisa usar.	**Secrets do Actions** Valores que o Copilot não pode ver.

Números que caem

Item	Valor
Níveis de autonomia	**4 (read-only, propose-only, execute with guardrails, human-authorized execution)**
Níveis de risco	**4 (Low, Medium, High, Critical)**
Fases do exemplo de autonomia progressiva	**4**
Opções de Repository access do cloud agent na organização	**All repositories (padrão), Selected repositories, No repositories**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** O diff tem só três linhas, então a mudança em .github/workflows/ é de baixo risco.
Como não cair: Workflow, infra/ e security/ são High por padrão; tamanho do diff não define risco.

⚠ **Alerta de prova:** Para acesso a segredos de produção, CODEOWNERS e checks mais rígidos bastam.
Como não cair: Isso é High; Critical exige gate de environment com revisores e trilha de auditoria.

⚠ **Alerta de prova:** Colocar humano em cada passo, inclusive leitura e formatação, aumenta a segurança.
Como não cair: Gera governance fatigue; humanos nos pontos de decisão: merge, deploy e segredo.

⚠ **Alerta de prova:** Agrupar as mudanças da semana em um PR grande para revisar de uma vez.
Como não cair: PRs pequenos e com escopo delimitado reduzem o cansaço.

⚠ **Alerta de prova:** Aprovar com ressalva ou pedir os artefatos depois do merge quando a evidência faltou.
Como não cair: Missing evidence = failure: não mergear.

⚠ **Alerta de prova:** Política escrita no documento interno ou no README basta para o agente.
Como não cair: Controls in docs only: transforme em CODEOWNERS com revisão exigida e ruleset.

⚠ **Alerta de prova:** Um segundo arquivo CODEOWNERS ou um environment fazem o time de /infra/ bloquear o merge.
Como não cair: Falta a regra que exige revisão de code owners no ruleset ou na branch protection.

⚠ **Alerta de prova:** Renomear o job de um required check é cosmético porque o ruleset acompanha o novo nome.
Como não cair: É mudança de alto risco; o check exigido pode deixar de corresponder ao que roda.

⚠ **Alerta de prova:** Hook preToolUse com ask pausa a sessão do cloud agent até alguém aprovar no PR.
Como não cair: Sem usuário, ask vira deny.

⚠ **Alerta de prova:** Serializar deploys de produção com grupo por github.sha, matrix ou espera fixa.
Como não cair: Use um grupo de concorrência fixo chamado production.

⚠ **Alerta de prova:** Contar com content exclusion para proteger pastas sensíveis do agent mode do VS Code.
Como não cair: O agent mode não suporta content exclusion.

⚠ **Alerta de prova:** Trocar o GITHUB_TOKEN por PAT de admin ou com escopo repo completo para resolver permissões.
Como não cair: PAT amplo aumenta o raio de impacto; declare permissões mínimas e eleve só no job que escreve.

⚠ **Alerta de prova:** Colocar as duas chaves em Agents secrets para centralizar.
Como não cair: A chave de deploy de produção fica em secret do Actions, fora do alcance do agente.

⚠ **Alerta de prova:** Limitar o piloto do cloud agent com ruleset, CODEOWNERS ou runs-on no setup steps.
Como não cair: Use Repository access em Selected repositories nas configurações de Copilot da organização.

Como cai na prova

📝 **Como cai:** Uma ação de agente descrita e a pergunta: qual controle ou qual nível de autonomia?
Como resolver: Classifique pelo lugar, reversibilidade e velocidade de impacto (não pelo tamanho do diff) e aplique a tabela: Low automação, Medium PR com checks, High CODEOWNERS e aprovações, Critical environment.

📝 **Como cai:** Um controle que existe mas não funcionou (CODEOWNERS ignorado, política no documento, hook ask).
Como resolver: Procure o que falta para virar enforcement: exigir revisão de code owners, transformar texto em ruleset, lembrar que ask vira deny no cloud agent.

📝 **Como cai:** Pressão para aprovar com evidência faltando ou para reduzir o cansaço dos revisores.
Como resolver: Evidência ausente bloqueia o merge; cansaço se resolve com evidência, CODEOWNERS, PRs pequenos e rollback, nunca com bypass ou merge automático.

📝 **Como cai:** Configuração de guardrail do cloud agent (repositórios, runner, segredos, setup steps, workflows).
Como resolver: Repository access em Selected repositories; GitHub-hosted ou self-hosted efêmero; Agents secrets só para o que o agente precisa; permissões mínimas no job; Approve and run workflows ligado por padrão.

📝 **Como cai:** Qual atividade de governança em qual frequência.
Como resolver: Semanal: runs com falha e violações comuns. Mensal: permissões de workflow e escopos de segredos. Trimestral: rulesets, CODEOWNERS, revisores de environment e retenção de evidências.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
agente lê, resume e sugere, mas não altera nada	**Read-only autonomy**
testes, builds e staging pré-aprovados sem aprovação a cada vez	**Execute with guardrails**
rotação de segredos e mudança de workflow só após aprovação explícita	**Human-authorized execution**
alteração em .github/workflows/, infra/ ou security/	**High: CODEOWNERS, aprovações explícitas, checks mais rígidos**
deploy ou segredos de produção	**Critical: gate de environment com revisores e trilha de auditoria**
aprovadores clicam sem ler o diff	**Rubber-stamping**

regra existe só em documento interno	**Controls in docs only**
aprovações de bots não podem contar	**Workflow de guardrail que só aceita aprovações humanas**
checks verdes, mas artefatos de plano e relatório ausentes	**Missing evidence = failure**
dois deploys de produção rodando ao mesmo tempo	**Concurrency group production**
risco em texto livre e decisões inconsistentes	**risk em plan.json roteando para job com environment**
auditar rulesets, CODEOWNERS, revisores de environment e retenção	**Cadência trimestral**
proteger perfis, hooks e setup steps do cloud agent	**CODEOWNERS + Require review from Code Owners**
ambiente limpo a cada sessão do cloud agent	**GitHub-hosted ou self-hosted efêmero**

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

- Risco se mede por onde, reversibilidade e velocidade de impacto, não pelo tamanho do diff.
- High é CODEOWNERS e aprovações; Critical é environment com revisores e auditoria.
- CODEOWNERS sem exigência de revisão é só consultivo.
- Evidência ausente é falha: não mergeie o que não pode ser auditado.
- No cloud agent, ask em preToolUse vira deny.

Essenciais para a prova (checklist)

- ☐ Sei os quatro níveis de autonomia e reconheço cada um em um cenário.
- ☐ Sei a tabela Low, Medium, High e Critical com o controle de cada nível.
- ☐ Sei os fatores que definem risco e por que workflows são High mesmo com diff pequeno.
- ☐ Sei onde colocar humanos e como reduzir governance fatigue.
- ☐ Sei os anti-patterns de governança, incluindo controls in docs only e rubber-stamping.
- ☐ Sei as camadas pré-ação, em ação e pós-ação e o papel dos hooks preToolUse.
- ☐ Sei configurar CODEOWNERS com exigência de revisão e as regras do branch protegido.
- ☐ Sei aplicar menor privilégio em workflows, environments, segredos e copilot-setup-steps.
- ☐ Sei as mitigações nativas do cloud agent e onde ajustar Repository access e aprovação de workflows.
- ☐ Sei separar Agents secrets de secrets do Actions.

Questões de revisão do capítulo

R2.1 · Implementar guardrails e responsabilização

Os aprovadores de PRs de agentes de um banco passaram a clicar em aprovar só para desbloquear o merge, sem ler o diff. Pela tabela de anti-patterns de governança do Learn, qual é o problema e a mitigação?

A. Bypass paths; permitir push direto no main para evitar a fila de revisão
B. Rubber-stamping; melhorar evidências, usar CODEOWNERS e PRs menores
C. Excess permissions; ampliar o token para os agentes aprovarem sozinhos
D. Unbounded autonomy; liberar o merge automático para os PRs pequenos

R2.2 · Implementar guardrails e responsabilização

Um PR aberto por um agente altera regras de cálculo de juros. Os checks obrigatórios passaram, mas o workflow não publicou os artefatos de plano e de relatório de execução previstos no processo. O revisor está pressionado para aprovar. Qual postura está alinhada ao Learn?

A. Aprovar com ressalva registrada em comentário no PR
B. Aprovar, porque os checks obrigatórios já passaram
C. Tratar a falta de evidência como falha e não mergear
D. Aprovar e pedir que o agente envie os artefatos depois

R2.3 · Implementar guardrails e responsabilização

Um agente precisa acessar segredos de produção para concluir uma migração. Pela tabela de risco do Learn, essa ação é Critical. Qual controle é recomendado?

A. Automação completa com rollback automático em caso de erro
B. PR com required checks e uma revisão de colega do time
C. CODEOWNERS na pasta de infraestrutura e checks mais rígidos
D. Environment com revisores explícitos e trilha de auditoria

R2.4 · Implementar guardrails e responsabilização

A política da Têxtil Aurora diz, em um documento interno, que agentes não podem mexer em .github/workflows/ sem aprovação do time de plataforma, mas nada no GitHub impõe essa regra. Como o Learn classifica essa situação?

A. Controle pré-ação adequado, desde que o documento esteja no README do repositório
B. Boa prática, porque a política escrita basta para orientar o comportamento do agente
C. Controle pós-ação suficiente, pois o audit log registra quem alterou o workflow
D. Anti-pattern controls in docs only: o que a plataforma não impõe será contornado

Respostas comentadas

R2.1. B) Rubber-stamping; melhorar evidências, usar CODEOWNERS e PRs menores

Por quê: A tabela de falhas de governança do Learn lista unbounded autonomy, excess permissions, missing audit trail, bypass paths e rubber-stamping. Neste último, humanos deixam de revisar de fato, e a mitigação é melhor evidência, CODEOWNERS e PRs menores.

✗ A) Incorreta. Bypass paths são caminhos que pulam a política, como push direto no main; criar um deles piora a governança.
✗ C) Incorreta. Excess permissions trata de tokens amplos; deixar o agente aprovar removeria a revisão humana.
✗ D) Incorreta. Unbounded autonomy é deploy de produção sem aprovação; liberar o merge automático agravaria o problema.

> **Dica de prova:** Aprovação no automático: rubber-stamping.

R2.2. C) Tratar a falta de evidência como falha e não mergear

Por quê: O Learn diz que evidência ausente é falha: se uma mudança não pode ser auditada, ela não deve ser mergeada. Workflows críticos devem gerar resultados de teste, relatórios de scan, plan.json e execution-report.

✗ A) Incorreta. Um comentário não substitui os artefatos exigidos e vira rubber-stamping.
✗ B) Incorreta. Checks verdes não substituem a evidência exigida para auditoria.
✗ D) Incorreta. Evidência posterior ao merge não permite auditar a decisão no momento certo.

> **Dica de prova:** Missing evidence = failure.

R2.3. D) Environment com revisores explícitos e trilha de auditoria

Por quê: Tabela do Learn: Low com automação completa; Medium com PR e required checks; High com CODEOWNERS, aprovações explícitas e checks mais fortes; Critical com gate de environment, revisores explícitos e evidência de auditoria.

✗ A) Incorreta. Automação completa é para ações de baixo risco e reversíveis.
✗ B) Incorreta. PR com required checks é o controle de risco médio.
✗ C) Incorreta. CODEOWNERS e checks mais fortes são o controle de risco alto, não crítico.

> **Dica de prova:** Segredo ou deploy de produção: gate de environment.

R2.4. D) Anti-pattern controls in docs only: o que a plataforma não impõe será contornado

Por quê: O Learn afirma que controls in docs only é anti-pattern e que controle não imposto pela plataforma deve ser tratado como opcional. A regra deveria virar CODEOWNERS com revisão exigida e ruleset.

✗ A) Incorreta. Pré-ação é permissão imposta, não texto em README.
✗ B) Incorreta. Política escrita sem enforcement deve ser tratada como opcional.
✗ C) Incorreta. Registro posterior não impede a mudança; é rastreabilidade, não bloqueio.

> **Dica de prova:** Regra só no documento será contornada.

03

CAPÍTULO 3 · 20–25% DA PROVA

Implementar o uso de ferramentas e a interação com o ambiente

É o domínio de maior peso, e eu preciso saber detalhe de configuração: perfis de custom agents, copilot-setup-steps, firewall, servidores MCP no cloud agent e na IDE, registry e allowlist de MCP, Copilot CLI programático e os limites do Copilot cloud agent. As questões são muito factuais: nome exato do job, chave obrigatória, prefixo de segredo, qual opção do CLI, qual política prevalece. Onde mais erro é misturar o que vale para o cloud agent com o que vale para a IDE e o CLI (registry, OAuth, firewall) e esquecer que vários arquivos só valem quando estão no branch padrão. Também caem tratamento de erro, retries, rollback e o comportamento do agente depois que a issue é atribuída.

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

Tópicos-chave

Perfil de custom agent (.agent.md)

Fica em .github/agents/NOME.agent.md e só aparece na lista depois de mesclado no branch padrão. Description é obrigatória e name é opcional; o nome do arquivo (sem .md ou .agent.md) identifica o agente e só aceita ponto, hífen, sublinhado, letras de a a z e números (nada de espaço ou acento). O corpo em Markdown abaixo do frontmatter tem no máximo 30.000 caracteres.

Propriedade tools do custom agent

tools omitido ou '*' habilita o conjunto completo; uma lista habilita só as citadas; tools: [] não habilita ferramenta nenhuma. Aliases como read, search, edit e execute (shell, Bash ou PowerShell) controlam capacidades; ferramentas MCP podem ser listadas uma a uma no formato servidor/ferramenta.

target e precedência de custom agents

target aceita vscode ou github-copilot; sem a propriedade, o agente vale nos dois ambientes. Em conflito de nome no GitHub vale o nível mais baixo: repositório > organização > enterprise. No Copilot CLI, agentes pessoais em ~/.copilot/agents vencem o do repositório quando o ID é igual.

Custom agents de organização e enterprise

De organização: pasta agents do repositório .github ou .github-private, disponíveis para os membros; o .github-private precisa ser privado para não expor os perfis. De enterprise: .github-private de uma organização designada, com rulesets para restringir quem cria ou edita.

copilot-setup-steps.yml

Arquivo em .github/workflows/copilot-setup-steps.yml com um job que precisa se chamar exatamente copilot-setup-steps, e só é usado pelo agente quando está no branch padrão; ao ser alterado num PR, roda como workflow comum e mostra o resultado como check. Chaves personalizáveis do job: steps, permissions, runs-on, services, snapshot e timeout-minutes (máximo 59). Serve para instalar dependências (JDK, uv, pipx), subir serviços (services, como um PostgreSQL) e fazer checkout com lfs: true para Git LFS.

Ambiente e runners do cloud agent

Aceita Ubuntu x64 e Windows 64-bit, sendo Windows só em larger runners ou self-hosted; macOS não é suportado e o firewall integrado não é compatível com Windows. Owners da organização podem definir o runner padrão em Settings > Copilot > Cloud agent e impedir que os repositórios o troquem pelo runs-on do setup steps.

Firewall do cloud agent

Limita a internet dos processos que o agente inicia pela ferramenta Bash, só no ambiente do GitHub Actions; não vale para servidores MCP nem para os setup steps, sempre permite os hosts que o Copilot usa para falar com o GitHub e pode ser contornado por ataques sofisticados. Regras custom aceitam domínio (inclui subdomínios) ou URL (esquema, host e caminho, com descendentes), não IP. A recommended allowlist cobre registries públicos comuns e os hosts de download de navegadores do Playwright; desligar o firewall permite qualquer host e aumenta o risco de exfiltração.

Configuração do firewall na organização

Três ajustes: Enable firewall e Recommended allowlist (Enabled, Disabled ou Let repositories decide) e Allow repository custom rules (Enabled ou Disabled). Para padronizar com allowlist recomendada e sem regras próprias: Enabled, Enabled e custom rules Disabled.

MCP no cloud agent

GitHub MCP server e Playwright MCP server vêm habilitados por padrão. O GitHub MCP padrão usa token só leitura do repositório atual; para ampliar, PAT ou GitHub App, com toolsets escolhidos no header X-MCP-Toolsets. O cloud agent (e o Copilot code review) só usa tools de servidores MCP, não resources nem prompts, não suporta servidores remotos com OAuth e usa as tools configuradas de forma autônoma, sem pedir aprovação.

JSON mcpServers do repositório

type e tools são obrigatórios em toda entrada. type aceita local, stdio, http ou sse; remotos usam url e headers, locais usam command, args e env. Só segredos e variáveis de Agents com nome iniciado por COPILOT_MCP_ ficam disponíveis, referenciados em env ou headers. Para validar, atribua uma issue e expanda Start MCP Servers no session log; se as tools não aparecem por falta de uv ou pipx, instale no copilot-setup-steps.

GitHub MCP server remoto na IDE

É a opção recomendada; com OAuth acessa só os escopos aprovados no login, com PAT valem os escopos do token e as restrições de PAT da organização. Em Copilot Business e Enterprise, a política MCP servers in Copilot precisa estar habilitada na organização.

MCP registry e allowlist

Com Registry only, só servidores do registry rodam, inclusive locais, que precisam constar com ID idêntico ao do instalado. Enterprise prevalece sobre organização e Registry only prevalece sobre Allow all; só em empate de escopo e rigor vale o último enviado. A aplicação por registry vale para IDEs e Copilot CLI, não para o cloud agent. No Azure API Center, informe só a URL base, sem /v0.1/servers.

managed-settings.json

Método mais forte e em GA para restringir servidores MCP, com allowedMcpServers e deniedMcpServers e casamento seguro; um deny de qualquer camada bloqueia, mesmo que o servidor também case com o allow. O registry custom está em preview, casa só por nome ou ID e pode ser contornado editando a configuração.

Copilot CLI programático

Inicie só em diretório confiável (o CLI pede confirmação porque poderá ler, modificar e executar arquivos). No CI: -p com a tarefa, --allow-tool mínimo (ex.: 'shell(git)' ou 'shell(npm:*), write'), --no-ask-user e COPILOT_GITHUB_TOKEN com PAT vindo de secret. --deny-tool vence --allow-tool; --allow-all-tools só em sandbox. --agent NOME escolhe um custom agent; /agent é do modo interativo; --resume e --continue retomam sessão; --share exporta a transcrição.

Como o cloud agent recebe e executa tarefas

Sessões começam por issue atribuída ao Copilot, aba ou painel Agents, Copilot Chat, @copilot em PR, IDEs, GitHub CLI, REST API, GitHub MCP Server e integrações (Jira, Slack, Teams, Azure Boards, Linear). Só quem tem write aciona o agente; ao começar aparecem a reação de olhos e o evento Copilot has started work. Na atribuição ele recebe título, descrição, comentários existentes e instruções adicionais, e dá para escolher base branch e custom agent; comentários posteriores na issue não chegam a ele.

Limites do cloud agent

Um repositório, um branch e um PR por tarefa; sessão de no máximo 59 minutos, sem extensão (nem self-hosted, nem larger runner, nem timeout-minutes). Faz push só no branch copilot/ que cria ou no branch do PR em que foi mencionado; abre PR em rascunho e não pode marcar como pronto, aprovar ou mesclar.

Acompanhar e iterar na sessão

Na página Agents: session log mostra raciocínio e ferramentas; steering redireciona a sessão e é aplicado após a chamada de ferramenta atual; Stop session encerra o run e mantém os commits enviados; sessões são arquivadas, nunca excluídas. Commits do agente trazem link para os session logs. Para iterar no PR: @copilot em comentário ou push no branch do PR; Add to batch delega vários comentários de revisão de uma vez, Fix with Copilot trata um comentário.

Agentic Workflows, contexto de execução e erros

GitHub Agentic Workflows: Markdown com frontmatter (gatilhos, permissions, tools, safe-outputs) compilado em lock file executado pelo Actions; o agente roda com leitura e as escritas passam por safe outputs. Contexto de execução é repositório, branch, workflow e permissões; o escopo de branch vem dele, não do perfil do agente. Tratamento de erro: continue-on-error: false para falhar rápido e retries limitados para falha transitória; rollback é fechar o PR antes do merge e reverter commits depois.

Macetes para memorizar

💡 tools no perfil

Vazio é nada, ausente é tudo: tools: [] fecha, omitir abre.

💡 Chaves do copilot-setup-steps

SPRSST: steps, permissions, runs-on, services, snapshot, timeout-minutes (até 59).

Limites do cloud agent

1-1-1-59: um repositório, um branch, um PR por tarefa, até 59 minutos.

Precedência de custom agents

No GitHub, quem está mais perto vence (repositório > org > enterprise); no CLI, o pessoal vence.

Firewall

Firewall vigia só o Bash do agente: MCP e setup steps passam por fora.

MCP no cloud agent

Só tools, sem OAuth remoto, sem pedir licença: liste pouco.

Copilot CLI no CI

p-a-n-t: -p, --allow-tool mínimo, --no-ask-user, token em secret; deny sempre ganha.

Política de MCP

Enterprise manda, o mais restritivo manda, deny manda.

Tabela de memorização: não confunda

Quando é...	...e quando é
tools omitido ou '*' Libera o conjunto completo de ferramentas (raramente é o que a questão quer).	**tools: []** Agente que só responde pelo próprio prompt, sem ler arquivos, executar comandos nem chamar MCP.
Regra de firewall por domínio Liberar um host inteiro, incluindo subdomínios e qualquer caminho.	**Regra de firewall por URL** Restringir a um caminho específico (esquema, host e caminho, com descendentes), sem subdomínios.
OAuth no GitHub MCP remoto Na IDE, acesso limitado aos escopos aprovados no login (não suportado em servidor remoto do cloud agent).	**PAT no GitHub MCP remoto** Valem os escopos do token e as restrições de PAT da organização.
managed-settings.json Controle mais forte, em GA, com allow e deny e casamento seguro.	**MCP registry com Registry only** Preview, casa só por nome ou ID e pode ser contornado; não vale para o cloud agent.
Add to batch Delegar vários comentários de revisão em uma única rodada.	**Fix with Copilot** Tratar um comentário específico.

Números que caem

Item	Valor
Tempo máximo de uma sessão do cloud agent	**59 minutos, sem extensão**
timeout-minutes no job copilot-setup-steps	**Máximo 59**

Tamanho máximo do prompt (corpo Markdown) de um custom agent	**30.000 caracteres**
Repositórios, branches e PRs por tarefa do cloud agent	**1 repositório, 1 branch por vez, 1 PR por tarefa**
Prefixo de segredos e variáveis de Agents para MCP	**COPILOT_MCP_**
Prefixo dos branches criados pelo cloud agent	**copilot/**
Exemplo de retry limitado do Learn	**Laço de três tentativas**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** O agente não aparece porque falta name no frontmatter ou o hífen no nome do arquivo é inválido.
Como não cair: name é opcional e hífen é permitido; o perfil precisa estar mesclado no branch padrão.

⚠ **Alerta de prova:** O setup steps não funciona porque falta gatilho workflow_dispatch ou porque devia ficar em .github/copilot.
Como não cair: O caminho é .github/workflows/copilot-setup-steps.yml e o job tem de se chamar copilot-setup-steps; workflow_dispatch só serve para testar à mão.

⚠ **Alerta de prova:** Usar env, outputs ou strategy no job copilot-setup-steps.
Como não cair: Só steps, permissions, runs-on, services, snapshot e timeout-minutes são personalizáveis.

⚠ **Alerta de prova:** Estender a sessão com timeout-minutes: 180, larger runner ou self-hosted.
Como não cair: O limite de 59 minutos é fixo; divida o trabalho em tarefas menores, cada uma com seu PR.

⚠ **Alerta de prova:** Confiar no firewall para barrar ferramentas perigosas de um servidor MCP.
Como não cair: O firewall não se aplica a processos de servidores MCP nem aos setup steps; limite a lista tools, de preferência read-only.

⚠ **Alerta de prova:** Comentar o novo requisito na issue depois da atribuição.
Como não cair: O agente não lê comentários novos na issue; peça o ajuste com @copilot no PR (por quem tem write).

⚠ **Alerta de prova:** Registry only da enterprise também limita o MCP do cloud agent.
Como não cair: A aplicação por registry vale para IDEs e CLI; o MCP do cloud agent é configurado por repositório.

⚠ **Alerta de prova:** Org com Allow all vence a enterprise com Registry only por ser mais específica ou mais recente.
Como não cair: Enterprise e o mais restritivo prevalecem; o mais recente só desempata mesmo escopo e rigor.

⚠ **Alerta de prova:** Criar segredo comum do Actions chamado API_KEY e referenciar no headers do MCP.
Como não cair: Só segredos de Agents com prefixo COPILOT_MCP_ ficam disponíveis para a configuração MCP.

⚠ **Alerta de prova:** Trocar --allow-tool amplo por --allow-all-tools ou por instrução no prompt para impedir git push.
Como não cair: Acrescente --deny-tool 'shell(git push)'; deny vence allow e --allow-all-tools é só para sandbox.

⚠ **Alerta de prova:** Autenticar o CLI no Actions com login interativo ou com o GITHUB_TOKEN padrão.
Como não cair: Use COPILOT_GITHUB_TOKEN com PAT guardado em secret.

⚠ **Alerta de prova:** O CI com gatilho push não rodou porque o push foi em branch novo ou por falta de rede.
Como não cair: Push feito com GITHUB_TOKEN não dispara novos runs (salvo dispatch); use token de GitHub App ou PAT e evite loops.

⚠ **Alerta de prova:** Uma issue pedindo três mudanças gera três PRs se você pedir no comentário.
Como não cair: Um PR por tarefa: crie três issues. Para dois repositórios, uma tarefa por repositório.

⚠ **Alerta de prova:** Restringir o branch do agente pelo campo target ou pelo copilot-instructions.md.
Como não cair: Custom agents definem foco, ferramentas e comportamento; o branch é imposto pelo contexto de execução (cloud agent ou workflow).

Como cai na prova

📝 **Como cai:** Algo não funciona no cloud agent (agente sumido, setup ignorado, ferramenta ausente) e a pergunta é a causa.
Como resolver: Verifique na ordem: está no branch padrão? o nome exato está certo (job copilot-setup-steps, arquivo .agent.md)? a dependência está instalada no setup steps? Alternativas sobre aprovação de admin, prefixos de issue ou gatilho manual são distratores.

📝 **Como cai:** Configuração de MCP com campos (type, url, headers, command, tools, segredo).
Como resolver: Lembre que type e tools são obrigatórios, remoto usa url e headers, local usa command, args e env, e segredo vem de Agents com prefixo COPILOT_MCP_. Campos como scopes, endpoint ou token não existem.

📝 **Como cai:** Qual política de MCP vale entre enterprise, organização, registry e managed-settings.
Como resolver: Enterprise acima de organização, Registry only acima de Allow all, deny acima de allow; e lembre que registry não alcança o cloud agent.

📝 **Como cai:** Comando do Copilot CLI para um job de CI com uma restrição.
Como resolver: Escolha a alternativa com -p, --allow-tool estreito e, se houver proibição pontual, --deny-tool; elimine --allow-all-tools fora de sandbox, modo interativo e /agent em CI.

📝 **Como cai:** Como pedir algo ao cloud agent depois de iniciado (novo requisito, vários comentários, ajuste próprio, mudar de rumo).
Como resolver: Novo pedido vai por @copilot no PR; vários comentários com Add to batch; ajuste próprio com push no branch do PR; mudar de rumo em andamento com steering. Editar a issue não adianta.

📝 **Como cai:** Escolher runner e ambiente para o cloud agent (Windows, recursos internos, ambiente limpo).
Como resolver: Windows só em larger ou self-hosted e sem firewall integrado; macOS nunca; recursos internos com runner padrão da org e override bloqueado; ambiente limpo com GitHub-hosted ou self-hosted efêmero.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
agente criado em branch de feature não aparece na lista	**Perfil precisa estar no branch padrão**
Copilot começa sem a ferramenta instalada pelo setup	**Job precisa se chamar copilot-setup-steps**
banco PostgreSQL para os testes do agente	**Chave services no copilot-setup-steps**

arquivos de ponteiro em vez dos binários	**actions/checkout com lfs: true no setup steps**
host bloqueado no curl pelo Bash, mas liberado no setup e no MCP	**Firewall só cobre processos do agente pela ferramenta Bash**
liberar só um caminho de um host interno	**Regra custom por URL**
toolsets actions e code_security no GitHub MCP	**Header X-MCP-Toolsets**
ler issues de outro repositório pelo GitHub MCP do cloud agent	**Token padrão só lê o repositório atual; ampliar com PAT ou GitHub App**
servidor MCP de fornecedor que só aceita OAuth no cloud agent	**Não suportado**
servidor expõe tools, resources e prompts	**Cloud agent usa só as tools**
allowedMcpServers e deniedMcpServers casam ao mesmo tempo	**Deny vence**
colaborador com leitura comenta @copilot e nada acontece	**Só quem tem write aciona o agente**
corrigir o rumo sem parar a sessão	**Steering na página Agents**
relatório diário por agente sem permissão de escrita	**Agentic Workflows com safe outputs**

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

- Perfis de agente, copilot-setup-steps e hooks só valem a partir do branch padrão.
- O cloud agent usa ferramentas MCP sem pedir aprovação: limite a lista tools.
- O firewall cobre só o Bash do agente no Actions; não cobre MCP nem setup steps.
- Depois da atribuição, converse com o Copilot no PR, nunca na issue.
- O cloud agent abre PR em rascunho e não marca como pronto, não aprova e não mescla.

Essenciais para a prova (checklist)

☐ Sei onde ficam e quando aparecem os perfis de custom agent e quais caracteres o nome do arquivo aceita.

☐ Sei o efeito de tools omitido, '*', lista e [] e o papel dos aliases read, search, edit e execute.

☐ Sei as chaves permitidas no job copilot-setup-steps e o nome exato do job.

☐ Sei o que o firewall cobre e não cobre e a diferença entre regra por domínio e por URL.

☐ Sei montar uma entrada mcpServers local e remota e onde guardar o segredo.

☐ Sei as limitações de MCP no cloud agent: só tools, sem OAuth remoto, GitHub MCP só leitura do repositório atual.

- ☐ Sei a precedência entre enterprise, organização, Registry only, Allow all, allow e deny.
- ☐ Sei montar um comando do Copilot CLI para CI com -p, --allow-tool, --deny-tool, --no-ask-user e COPILOT_GITHUB_TOKEN.
- ☐ Sei os limites do cloud agent (59 minutos, um PR por tarefa, branch copilot/) e como iterar num PR dele.
- ☐ Sei por que um push com GITHUB_TOKEN não dispara outro workflow e como resolver.

Questões de revisão do capítulo

R3.1 · Implementar o uso de ferramentas e a interação com o ambiente

Ao iniciar o Copilot CLI em uma pasta recém-clonada de um repositório de terceiros, o desenvolvedor recebe a pergunta se confia naquele diretório. Qual orientação da documentação se aplica?

A. Iniciar o CLI a partir de diretórios em que confia
B. Pular a pergunta, porque ela não afeta permissões
C. Confirmar, porque a pergunta é só informativa
D. Responder que sim e usar --allow-all-tools para agilizar

R3.2 · Implementar o uso de ferramentas e a interação com o ambiente

O servidor MCP interno do Hospital Central expõe tools, resources e prompts. Ao configurá-lo no cloud agent, o que a equipe deve esperar?

A. O agente usa só as tools; resources e prompts do servidor não são suportados
B. O agente usa só os prompts, que substituem as instruções do repositório
C. O agente usa resources e prompts, mas as tools precisam de aprovação manual
D. O agente usa tools, resources e prompts, desde que o tipo seja HTTP

R3.3 · Implementar o uso de ferramentas e a interação com o ambiente

Na Química Fênix, o firewall do cloud agent bloqueia um host externo quando o agente roda curl pelo Bash, mas o mesmo host é acessado sem bloqueio durante o copilot-setup-steps.yml e por um servidor MCP local. Por quê?

A. O servidor MCP e os setup steps rodam com o firewall de outra organização
B. O firewall libera o host após a primeira requisição bloqueada da sessão
C. O firewall só vale para processos que o agente inicia pela ferramenta Bash
D. O host foi incluído na recommended allowlist apenas para os setup steps

R3.4 · Implementar o uso de ferramentas e a interação com o ambiente

Uma equipe quer que o Copilot cloud agent marque seus PRs como prontos para revisão e faça o merge quando os checks passarem, para reduzir trabalho manual. O que o GitHub Docs indica?

A. O agente não marca ready nem aprova ou mescla; isso cabe a humanos
B. Basta conceder ao agente o papel maintain no repositório
C. É preciso habilitar auto-merge no copilot-setup-steps.yml
D. A opção existe para PRs de baixo risco nas configurações do agente

Respostas comentadas

R3.1. A) Iniciar o CLI a partir de diretórios em que confia

Por quê: A documentação orienta iniciar o Copilot CLI só em diretórios confiáveis; ao começar, o CLI pede confirmação de confiança porque poderá ler, modificar e executar arquivos ali.

✗ B) Incorreta. A confiança no diretório afeta o que o CLI pode ler, modificar e executar.
✗ C) Incorreta. A confirmação libera o CLI a trabalhar com os arquivos daquela pasta.
✗ D) Incorreta. Liberar todas as ferramentas numa pasta desconhecida amplia o risco.

> **Dica de prova:** Copilot CLI: só em diretório confiável.

R3.2. A) O agente usa só as tools; resources e prompts do servidor não são suportados

Por quê: A documentação informa que o Copilot cloud agent e o Copilot code review só suportam tools de servidores MCP, e não resources nem prompts.

✗ B) Incorreta. Prompts de MCP não são suportados no cloud agent.
✗ C) Incorreta. É o contrário: só tools são usadas, e de forma autônoma.
✗ D) Incorreta. O tipo do servidor não muda o suporte; resources e prompts não são usados.

> **Dica de prova:** Cloud agent e MCP: só tools.

R3.3. C) O firewall só vale para processos que o agente inicia pela ferramenta Bash

Por quê: Limitações do firewall: vale só para processos iniciados pelo agente pela ferramenta Bash, não para servidores MCP nem para os processos dos setup steps, e só no ambiente do GitHub Actions.

✗ A) Incorreta. Não existe firewall de outra organização nesse fluxo.
✗ B) Incorreta. Não há liberação automática após bloqueio; o firewall registra um aviso.
✗ D) Incorreta. A recommended allowlist não é segmentada por etapa.

> **Dica de prova:** Firewall cobre Bash do agente, não MCP nem setup steps.

R3.4. A) O agente não marca ready nem aprova ou mescla; isso cabe a humanos

Por quê: Mitigação documentada: o cloud agent abre PRs em rascunho e não pode marcá-los como prontos, aprová-los nem mesclá-los; humanos fazem a revisão e o merge.

✗ B) Incorreta. O limite é do produto; mudar papéis não dá ao agente essa capacidade.

✗ C) Incorreta. O copilot-setup-steps prepara o ambiente e não controla merges.

✗ D) Incorreta. Não há configuração que permita ao agente concluir o próprio PR.

> **Dica de prova:** Cloud agent propõe; humano marca ready e mescla.

04

CAPÍTULO 4 · 15–20% DA PROVA

Orquestrar a coordenação multiagente

Aqui eu preciso coordenar vários agentes sem que eles colidam: responsabilidades estreitas por caminho, padrões de orquestração (sequencial, paralela, fan-out/fan-in, coordenação por artefatos), isolamento de branch, workflow, permissões e concorrência, tipos de conflito, limites de escalonamento, modos de falha, observabilidade e ciclo de vida dos agentes. A prova gosta de YAML de Actions (needs, concurrency, workflow_run) e de classificar situações: tipo de conflito ou modo de falha. Onde mais erro é a diferença entre user-invocable e disable-model-invocation, o grupo de concurrency correto e achar que paralelismo é seguro sem isolamento de caminhos. Fronteira central: agentes propõem; humanos e políticas aceitam.

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

Tópicos-chave

Agents propose; humans and policy accept

Agentes abrem PRs e produzem evidência; regras do repositório e revisores decidem o merge. Nenhum agente aprova ou mescla o trabalho de outro, nem o agente de segurança.

Responsabilidades estreitas por caminho

Papel estreito, escopo por caminho e tipo de artefato e sinais de conclusão verificáveis por checks e revisões. Exemplo do Learn: dependências mexe em manifestos e lockfiles; refatoração em src/ sem manifestos, lockfiles ou workflows; segurança valida e reporta com base em checks e scans, sem refatorar salvo tarefa explícita. Agentes generalistas no repositório todo geram colisões, revisões mal roteadas e risco descontrolado.

Artefatos por estágio no sistema multiagente

Planning: seção de plano no PR ou PLAN.md. Implementation: branch, commits e PR. Validation: runs, checks e artefatos. Acceptance: CODEOWNERS, revisões e required checks. Deployment: environments e aprovações.

Orquestração sequencial x paralela

Sequencial quando a saída de um agente precisa de validação antes de seguir (PR, CI, segurança, humano, merge). Paralela só com caminhos isolados; se os agentes podem colidir nos mesmos arquivos, o paralelismo traz instabilidade em vez de vazão, e a saída é sequência ou redefinir escopos.

Gatilhos de orquestração no Actions

schedule para trabalho periódico (relatórios, checagem de dependências); eventos de pull request para validação e iteração; workflow_run com workflows: [nome] e types: [completed] para rodar depois de outro workflow. O filtro workflows usa o nome do workflow, então renomear o anterior quebra a cadeia; completed dispara com sucesso ou falha; workflow_run encadeia no máximo três níveis.

Fan-out/fan-in com needs

Jobs sem dependência rodam em paralelo e o job de consolidação declara needs com a lista (ex.: needs: [spec_analyzer, risk_reviewer]) para começar só quando todos terminam. No exemplo do Learn, concurrency com group multiagent-${{ github.ref }} no plan_merger evita jobs de consolidação sobrepostos no mesmo branch: needs espera, concurrency evita sobreposição.

Coordenação por artefatos

O agente roda com permissões restritas, grava saída estruturada (plano, relatório, proposta em JSON) como artefato e uma etapa controlada aplica as operações permitidas, como criar a issue. Cria fronteira explícita entre raciocínio e escrita e deixa evidência inspecionável, mais confiável que mensagens diretas entre agentes. Orquestração oculta em VM própria impede revisores de entender por que o sistema agiu.

Isolamento de branch, workflow e permissões

Um branch dedicado por agente (ex.: agent/dependency/123, agent/refactor/124) evita commits intercalados e dono ambíguo. Cada agente com workflow ou job dedicado, com gatilhos e permissões próprios, reduz acoplamento. Agente que só atualiza rótulos e descrições de PR usa contents: read e pull-requests: write. Isolamento também dá estabilidade e atribuição: contexto compartilhado deixa falhas difíceis de diagnosticar e gera reexecuções em série.

Concurrency por workflow e branch

Recomendação: concurrency no nível do workflow com group ${{ github.workflow }}-${{ github.ref }} e cancel-in-progress: true; um novo push cancela a execução anterior daquele workflow naquele branch e só a mais nova segue. Usar só ${{ github.ref }} torna a concorrência global ao branch e trava workflows não relacionados; strategy.matrix controla paralelismo dentro de uma execução.

Paralelismo por modo de invocação

Copilot cloud agent e Copilot CLI suportam sessões paralelas em várias tarefas; o modo local é serial, uma tarefa por vez. No cloud agent cada tarefa roda em ambiente efêmero com GitHub Actions, com branch e PR próprios, o que dá isolamento natural.

Quatro tipos de conflito

Textual: mesmas linhas alteradas. Semântico: merge limpo, mas o comportamento combinado quebra. De política: áreas protegidas exigem aprovações diferentes (ex.: PR tocando /src/ e /infra/). Esforço duplicado: dois PRs resolvendo o mesmo problema, como correções incompatíveis para o mesmo alerta.

Arbitragem e merge queue

Sem regras, quem mergeia primeiro vence. Detecte cedo com um passo Validate merge with main (git fetch origin main e git merge --no-commit origin/main) marcado como required check, roteie áreas sensíveis com CODEOWNERS e escale após falhas repetidas. A merge queue testa cada PR sobre o branch alvo atualizado e os PRs já na fila, pegando combinações que quebram.

Limites de escalonamento

Três gatilhos: PR que conflita duas vezes mesmo após rebase, required check que falha duas vezes com a mesma assinatura e dois agentes com correções incompatíveis para o mesmo alerta. O escalonamento traz um relatório curto: o que conflitou ou falhou, o que foi tentado e quais opções existem.

Modos de falha e recuperação

Partial execution: o PR existe, mas falta evidência ou validação. Stalled workflows: checks falhando repetidamente ou aprovação que nunca chega. Conflicting outputs: saídas concorrentes. Flapping: reexecuções sem convergência. Recuperação: retries limitados, escalar quando o mesmo check falha duas vezes, PRs pequenos que facilitam o revert, e escalar ao dono com links e resumo do estado.

Diagnóstico e observabilidade

Diagnostique com timeline do PR, resultados dos required checks, histórico de workflow runs (falhas e cancelamentos) e artefatos. Análise post-hoc: abra o run na aba Actions e baixe da seção Artifacts (ex.: agent-report). Artefato sumido: audit log da org com action:artifact.destroy. Log ao vivo de sessão travada: gh agent-task view --log --follow. Hook preToolUse com 'Command blocked by policy' barra ação destrutiva e aciona revisão humana.

Atribuição e registro de decisões

Com vários agentes, títulos com o agente, rótulos como agent: dependency e seções padronizadas no PR dão atribuição filtrável. O modelo de PR do Learn tem Objective, Plan, Evidence, Decisions and handoffs (Decision, Rationale, Next owner) e Risks and rollback.

Ciclo de vida de agentes

Implantar, monitorar, atualizar e aposentar. Novo agente: dono claro do agente e dos guardrails, runbooks e cadência regular de revisões, com permissões mínimas. Atualizar: mudar o perfil por PR revisado; custom agents são versionados pelo SHA do commit do perfil e as interações dentro de um PR mantêm a mesma versão. Aposentar: remover o perfil do branch padrão por PR, preservando PRs, commits, sessões arquivadas e audit log.

Invocação de agentes e subagentes

user-invocable: false tira o agente do seletor manual mas permite uso como subagente; disable-model-invocation: true impede que ele seja chamado como subagente via orquestração. No Copilot SDK, infer (padrão true) permite seleção automática e infer: false exige pedido explícito; o par recomendado é pesquisador read-only e editor com escrita, com descriptions específicas; trate o evento subagent.failed (há também selected, started, completed e deselected, com toolCallId).

Macetes para memorizar

💡 Fronteira multiagente

Agente propõe, humano e política dispõem.

💡 Tipos de conflito

TSPD: Textual (linhas), Semântico (comportamento), Política (aprovações), Duplicado (mesmo problema).

💡 Modos de falha

Parcial sem evidência, Parado sem aprovação, Conflitante com saídas concorrentes, Flapping sem convergência.

💡 Escalonamento

Duas vezes é o limite: conflitou duas, falhou duas, ou dois agentes brigando pelo mesmo alerta.

💡 Invocação

user-invocable fala com o usuário; disable-model-invocation fala com o modelo e com o orquestrador.

💡 YAML de orquestração

needs espera, concurrency atropela a antiga, workflow_run vem depois de outro workflow.

💡 Paralelismo por modo

Nuvem e CLI em paralelo; local na fila.

Tabela de memorização: não confunda

Quando é...	...e quando é
user-invocable: false Agente chamado só pelo orquestrador como subagente, fora do seletor dos usuários.	**disable-model-invocation: true** Impedir chamada como subagente; se a orquestração parou de chamar um agente que ainda aparece no seletor, a causa é essa.

Orquestração sequencial A saída de um agente precisa ser validada antes do próximo passo.	**Orquestração paralela** Agentes em caminhos isolados, sem tocar os mesmos arquivos.
Conflito semântico O Git não acusa conflito, mas a combinação quebra o comportamento.	**Conflito textual** Os dois PRs alteram as mesmas linhas e o merge acusa.
Conflito de política Um PR toca áreas que exigem aprovações de times diferentes.	**Esforço duplicado** Dois PRs resolvem o mesmo problema; com correções incompatíveis para o mesmo alerta, escale.
Partial execution PR com commits, mas sem check executado nem relatório de validação.	**Stalled workflow** Checks passaram, mas a aprovação não chega, ou checks falham repetidamente.
needs Fazer um job esperar outros terminarem (fan-in).	**concurrency** Evitar execuções sobrepostas no mesmo grupo, cancelando a anterior com cancel-in-progress.
group ${{ github.workflow }}-${{ github.ref }} Isolar por workflow e por branch, mantendo só a execução mais nova.	**group ${{ github.ref }} sozinho** Evitar: deixa a concorrência global ao branch e trava workflows não relacionados.

Números que caem

Item	Valor
Tentativas de rebase com conflito antes de escalar	**2**
Falhas do required check com a mesma assinatura antes de escalar	**2**
Níveis máximos de encadeamento com workflow_run	**3**
Tipos de conflito do Learn	**4 (textual, semântico, política, esforço duplicado)**
Modos comuns de falha	**4 (partial execution, stalled, conflicting outputs, flapping)**

Alertas de prova (pegadinhas)

> ⚠ **Alerta de prova:** Paralelismo é seguro se os agentes usam o mesmo modelo, as mesmas instruções ou mais runners.
> **Como não cair:** O requisito é isolamento de caminhos; sem ele há instabilidade.

⚠ **Alerta de prova:** Colocar os agentes no mesmo branch compartilhado, com prefixo do agente na mensagem de commit.
Como não cair: Um agente, um branch dedicado; branch compartilhado mantém commits intercalados e dono ambíguo.

⚠ **Alerta de prova:** Fazer o job esperar com concurrency ou workflow_run apontando para jobs do mesmo workflow.
Como não cair: Espera entre jobs é needs; workflow_run reage ao término de outro workflow.

⚠ **Alerta de prova:** workflow_run com types: [completed] só dispara se o CI teve sucesso.
Como não cair: Dispara ao fim, com sucesso ou falha; a conclusão pode ser filtrada no job.

⚠ **Alerta de prova:** Cadeia por workflow_run parou porque o GITHUB_TOKEN perdeu permissão após renomear o CI.
Como não cair: O filtro workflows usa o nome do workflow; atualize o nome.

⚠ **Alerta de prova:** Aceitar o PR que terminar os checks primeiro ou dar prioridade a quem abriu primeiro.
Como não cair: Isso é decidir por tempo; use check de validação de merge obrigatório, CODEOWNERS e escalonamento.

⚠ **Alerta de prova:** Aumentar retries até o check estabilizar ou mesclar no último verde.
Como não cair: Flapping pede parar o loop, limitar tentativas e escalar com links e resumo.

⚠ **Alerta de prova:** Aprovação de CODEOWNERS não chega, então reexecutar workflows ou mesclar sem aprovação.
Como não cair: É stalled workflow; escale ao dono com links e resumo.

⚠ **Alerta de prova:** Tentar rebase sem limite ou forçar push no branch do outro agente.
Como não cair: Conflito duas vezes após rebase: escale com relatório do conflito, tentativas e opções.

⚠ **Alerta de prova:** Agente de segurança pode refatorar o que achar inseguro ou aprovar PRs dos outros.
Como não cair: Ele valida e reporta; aceitação é de humanos e políticas.

⚠ **Alerta de prova:** Aposentar agente excluindo sessões e PRs, reescrevendo o histórico ou reaproveitando o nome.
Como não cair: Remova o perfil do branch padrão por PR e preserve a trilha; sessões do cloud agent só podem ser arquivadas.

⚠ **Alerta de prova:** Atualizar o prompt do agente copiando o texto novo nos comentários dos PRs em andamento.
Como não cair: Altere o perfil por PR revisado; a versão por SHA mantém os PRs em andamento estáveis.

⚠ **Alerta de prova:** Usar tools: null ou description genérica para os subagentes do SDK não ficarem sem acesso.
Como não cair: tools: null libera tudo; use pesquisador read-only, editor com escrita e descriptions específicas.

⚠ **Alerta de prova:** gh run view --log ou gh pr checks --watch acompanham a sessão do agente em tempo real.
Como não cair: Para o log da sessão ao vivo, gh agent-task view --log --follow.

Como cai na prova

📝 **Como cai:** Descrição de um incidente entre agentes e a pergunta: que tipo de conflito ou falha é?
Como resolver: Teste as definições: mesmas linhas, comportamento combinado, aprovações diferentes ou mesmo problema; para falhas, PR sem evidência, aprovação que não chega, saídas concorrentes ou reexecução sem convergir.

📝 **Como cai:** Trecho de YAML ou requisito de Actions (esperar jobs, cancelar execução antiga, rodar após outro workflow).
Como resolver: Mapeie: esperar é needs; sobreposição é concurrency com workflow e ref; após outro workflow é workflow_run completed; periódico é schedule.

📝 **Como cai:** Situação travada ou repetida: o que o agente deve fazer agora?
Como resolver: Se bateu um limite (duas falhas, dois conflitos, correções incompatíveis), escale com relatório curto. Elimine retries sem limite, remover check, force push e mesclar sem aprovação.

📝 **Como cai:** Configuração do perfil de agente no ciclo de vida (subagente, atualização, aposentadoria).
Como resolver: Subagente fora do seletor é user-invocable: false; não chamado na orquestração é disable-model-invocation; atualizar é PR revisado com versão por SHA; aposentar é tirar do branch padrão preservando a trilha.

📝 **Como cai:** Como tornar a coordenação observável e auditável.
Como resolver: Escolha artefatos, workflow runs, timeline do PR, rótulos e seções padronizadas; descarte VM externa, mensagens diretas, logs locais do runner e chat.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
merge limpo e sistema quebrado	**Conflito semântico; merge queue**
PR toca áreas com aprovações de times diferentes	**Conflito de política**
dois agentes com correções incompatíveis para o mesmo alerta	**Esforço duplicado; escalar**
conflitos devem falhar cedo como check	**git merge --no-commit do main como required check**
PR existe, sem check nem relatório de validação	**Partial execution**
reexecuções alternando sucesso e falha	**Flapping; limitar e escalar**
rodar validação depois que outro workflow terminar	**workflow_run com types: [completed]**
job de consolidação espera dois analisadores	**needs com a lista de jobs**
manter só a execução mais nova por workflow e branch	**concurrency ${{ github.workflow }}-${{ github.ref }} com cancel-in-progress: true**
agente de análise propõe rótulos sem permissão de escrita	**Coordenação por artefatos com etapa controlada**
agente só como subagente, fora do seletor	**user-invocable: false**
artefato sumiu antes da retenção	**Audit log com action:artifact.destroy**
registrar decisão, motivo e quem assume no handoff	**Seção Decisions and handoffs (Decision, Rationale, Next owner)**
revisores não sabem qual agente fez o PR	**Título com o agente e rótulo agent: nos PRs**

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

- Paralelismo só com caminhos isolados; senão, sequência.
- Um agente, um branch, um workflow ou job com permissões próprias.
- Grupo de concurrency com github.workflow e github.ref; só github.ref trava tudo no branch.
- user-invocable: false tira do seletor; disable-model-invocation: true tira da orquestração.
- Escalar com relatório: o que falhou, o que tentou e quais opções.

Essenciais para a prova (checklist)

☐ Sei definir papéis estreitos por caminho para agentes de dependências, refatoração e segurança.

☐ Sei os artefatos que tornam revisável cada estágio (planning, implementation, validation, acceptance, deployment).

☐ Sei quando usar orquestração sequencial, paralela, fan-out/fan-in e coordenação por artefatos.

☐ Sei configurar workflow_run, needs e concurrency e prever o efeito de cancel-in-progress.

☐ Sei isolar agentes por branch, workflow ou job e permissões mínimas.

☐ Sei os quatro tipos de conflito e as ferramentas de arbitragem (check de merge, CODEOWNERS, merge queue).

☐ Sei os três limites de escalonamento e o conteúdo do relatório.

☐ Sei os quatro modos de falha e as respostas de recuperação.

☐ Sei diagnosticar com timeline, runs, artefatos, audit log e gh agent-task view --log --follow.

☐ Sei adicionar, atualizar e aposentar agentes preservando auditabilidade.

Questões de revisão do capítulo

R4.1 · Orquestrar a coordenação multiagente

Uma líder atribui ao Copilot cloud agent quatro issues independentes do mesmo repositório ao mesmo tempo. Como o GitHub mantém essas execuções paralelas isoladas umas das outras?

A. Cada tarefa grava direto no branch padrão, separada por pastas diferentes
B. Cada tarefa roda em ambiente efêmero próprio, com branch e PR próprios
C. As quatro tarefas compartilham um branch e um PR, com commits intercalados
D. As tarefas entram em fila e rodam uma por vez no mesmo runner do repositório

R4.2 · Orquestrar a coordenação multiagente

O agente de refatoração da Fintech Cobre bateu no limite de escalonamento após o mesmo check falhar duas vezes com a mesma assinatura. O que o relatório de escalonamento deve trazer?

A. Um resumo do custo da sessão e do tempo gasto pelo agente na tarefa
B. Uma nova tentativa automática com outro modelo antes de avisar alguém
C. O que conflitou ou falhou, o que foi tentado e quais opções existem
D. O pedido de remoção do required check para destravar o merge do PR

R4.3 · Orquestrar a coordenação multiagente

O log de uma sessão mostra a mensagem 'Error: Command blocked by policy' com o motivo destructive_operation_detected, seguida de '[agent] Escalating to human review'. O que aconteceu?

A. O firewall do agente recusou o acesso a um domínio externo
B. Um hook postToolUse desfez a ação depois da execução
C. Um hook preToolUse barrou a ação e acionou a revisão humana
D. O required check falhou e reverteu o commit do agente

R4.4 · Orquestrar a coordenação multiagente

Um time usa o Copilot SDK para montar dois subagentes: um que explora o código e outro que aplica as mudanças. Qual desenho segue o padrão recomendado na documentação?

A. Pesquisador com ferramentas de leitura e editor com escrita
B. Editor com leitura e pesquisador com escrita, para equilíbrio
C. Os dois agentes com tools: null, para evitar falta de acesso
D. Um único agente com as duas funções e description genérica

Respostas comentadas

R4.1. B) Cada tarefa roda em ambiente efêmero próprio, com branch e PR próprios
Por quê: O cloud agent trabalha em ambiente efêmero com GitHub Actions, em um branch por vez e com um PR por tarefa. Isso dá isolamento natural às sessões paralelas, que o Learn indica como suportadas no cloud agent e no Copilot CLI.

✗ A) Incorreta. O agente não faz push no branch padrão; trabalha em branch próprio e propõe mudanças por PR.
✗ C) Incorreta. Cada tarefa tem branch próprio; branch compartilhado gera commits intercalados e dono ambíguo.
✗ D) Incorreta. Pela tabela do Learn, o cloud agent suporta sessões paralelas em várias tarefas.

> **Dica de prova:** Várias issues ao mesmo tempo: uma sessão, um branch e um PR por tarefa.

R4.2. C) O que conflitou ou falhou, o que foi tentado e quais opções existem

Por quê: Escalonamento deve vir com um relatório curto do que conflitou ou falhou, do que foi tentado e das opções existentes, para que o humano decida com contexto.

✗ A) Incorreta. Custo e tempo não ajudam o humano a decidir a correção.
✗ B) Incorreta. Escalar é parar a automação, não tentar de novo.
✗ D) Incorreta. Remover o check é abrir brecha na governança.

> **Dica de prova:** Escalar com: o que falhou, o que tentou, quais opções.

R4.3. C) Um hook preToolUse barrou a ação e acionou a revisão humana

Por quê: No Learn, um hook pré-execução (preToolUse) bloqueia ações arriscadas, como apagar infraestrutura, e dispara o handoff para revisão humana.

✗ A) Incorreta. O motivo indica operação destrutiva, não acesso de rede.
✗ B) Incorreta. postToolUse roda depois da ação e serve para auditar, não para bloquear.
✗ D) Incorreta. Checks não revertem commits nem bloqueiam comandos na sessão.

> **Dica de prova:** Blocked by policy antes da execução: preToolUse.

R4.4. A) Pesquisador com ferramentas de leitura e editor com escrita

Por quê: A documentação recomenda o par pesquisador read-only e editor com escrita, com descriptions específicas para o runtime escolher bem e tools restritas por menor privilégio.

✗ B) Incorreta. Os papéis ficariam invertidos em relação às permissões necessárias.
✗ C) Incorreta. tools: null libera tudo e contraria o menor privilégio.
✗ D) Incorreta. Description genérica atrapalha a escolha do runtime, e juntar papéis amplia o acesso.

> **Dica de prova:** SDK: pesquisador só lê, editor escreve.

05

CAPÍTULO 5 · 15–20% DA PROVA

Preparar a arquitetura de agentes e os processos do SDLC

Aqui eu preciso dominar como um agente entra no SDLC sem virar um desenvolvedor solto no repositório: o ciclo plan, act, evaluate, o contrato da tarefa (entradas, saídas e critérios de sucesso), a separação entre planejar e executar e os controles do GitHub que transformam expectativa em garantia. A prova traz cenários curtos de empresa e pede o controle, o anti-pattern ou o item da rubrica certo, quase sempre com distratores que parecem razoáveis mas são só orientação (instrução, aviso, promessa do agente). Onde mais erro é confundir as mitigações dos anti-patterns entre si e trocar controles parecidos: rulesets, required status checks, CODEOWNERS e environments. A regra de ouro do domínio: instrução orienta, allowlist de tools, gates e regras impõem; e quem responde pelo resultado são os humanos que definiram, permitiram e aprovaram.

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

Tópicos-chave

Assistente x agente

O assistente é reativo: sugere e devolve o controle ao usuário a cada passo. O agente mantém um objetivo em várias etapas, usa ferramentas, cria artefatos duráveis (branch, commits, PR) e itera com base em sinais, como um check que falhou. Tamanho do modelo, velocidade ou responder em linguagem natural não definem agente; o teste prático é quem criou o branch e o PR.

Ciclo plan, act, evaluate

É um loop: o agente planeja, age e avalia repetidamente até cumprir os critérios de sucesso ou escalar para um humano. Na fase act ele cria branch, faz commits, abre ou atualiza o PR e responde à revisão, sempre em branch e PR, nunca direto no branch padrão. Na avaliação valem sinais do sistema (workflow runs, status checks de build, teste e lint, feedback de revisão, code scanning, secret scanning, alertas de dependência), nunca a confiança declarada pelo agente.

Contrato da tarefa (inputs, outputs, success criteria)

Entradas: contexto da issue, restrições e limites de escopo. Saídas: plano, PR com changeset limitado e links de evidência. Critérios de sucesso: checks aprovados, sinais de segurança resolvidos, escopo coerente com a intenção e caminho de rollback ou escalonamento. A prova pede para classificar um item em uma dessas três caixas.

Instrução orienta, enforcement impõe

Escrever no prompt ou no arquivo de instruções que o agente não deve editar (mesmo em maiúsculas) é orientação. O que impõe fronteira é allowlist de tools, gates e regras da plataforma. Agente de planejamento deve ficar só com ferramentas de leitura.

Quatro formas de separar planejamento e execução

Agente de planejamento read-only; handoff explícito para um agente de implementação com escrita depois que o plano é aprovado; tool gating no orquestrador (ferramentas só liberadas depois de o plano ser aceito); e fluxos de plan mode. Dar escrita ao próprio agente de planejamento no meio da sessão apaga a fronteira.

Plan-first PR x plano e execução no mesmo PR

Os dois usam os mesmos controles; muda o momento da validação humana. Plan-first (PR só com o plano, aprovado antes do primeiro commit) é para alto risco ou difícil reversão, como autenticação, infraestrutura e produção. Plano e código no mesmo PR serve para baixo risco e fácil reversão, mas traz exposição mais cedo a mudanças desnecessárias e mais esforço de revisão; o merge continua bloqueado pelos checks e revisões.

Plano revisável e template de PR

Um plano fica revisável quando traz escopo (o que muda), critérios de sucesso (como saber que funcionou) e caminho de rollback ou escalonamento; nada de despejar o raciocínio interno do modelo. O template .github/pull_request_template.md do Learn tem Plan (Goal, Scope, Steps, Success criteria, Risks, Rollback/escalation), Evidence (links de workflow runs e resultados de scan) e Review checklist (preenchido na revisão).

Plan gate como required status check

Template com seção de plano é expectativa; para virar garantia do sistema, um workflow de plan gate é marcado como required status check em ruleset ou branch protection. Rótulo depois do merge, aviso no README ou pedido no copilot-instructions.md não bloqueiam nada.

Anti-patterns de agentes e mitigações

Planless execution: PR com diff e sem plano; mitigação é seção de plano no template e revisão pré-merge. Hidden reasoning: só o diff, sem premissas, decisões e contexto; mitigação é exigir plano, linkar workflow runs e registrar decisões no PR. Over-permissioned agent: escrita e segredos amplos (ex.: permissions: write-all); mitigação é GITHUB_TOKEN de menor privilégio, environments com revisores e restringir quem dispara workflows. Blind trust in automation: CI verde basta; mitigação é somar CODEOWNERS, revisões obrigatórias e aprovações por risco.

Contributor model e sua rubrica

O PR do agente é avaliado como qualquer contribuição, pelo padrão do fluxo e não pela autoria. A rubrica tem seis itens: Intent (objetivo claro e plano visível), Scope (arquivos alterados alinhados ao plano), Evidence (checks e logs), Ownership, Policy (rulesets, regras de branch, environments) e Fallback (rollback ou escalonamento claro em mudança de alto risco). Evita dois erros opostos: excessive suspicion e excessive trust.

Responsabilização humana

Agentes executam trabalho, mas humanos continuam responsáveis pelos resultados e pelos controles: quem definiu a tarefa, concedeu permissões, configurou controles e aprovou a mudança. Nem o agente, nem o GitHub (porque checks passaram), nem ninguém em particular; o sistema propõe, humanos decidem. Numa análise de incidente, a pergunta é se o sistema tornou o erro compreensível e evitável (plano, revisores certos, checks condizentes com o risco, trilha).

GitHub como system of record

Repositórios e branches, commits e PRs, issues e discussões (intenção), workflow runs e artefatos (evidência) e histórico de revisão do PR (decisões humanas de aceitar ou rejeitar). CODEOWNERS diz quem revisa, não o que foi decidido.

Tabela de controles do GitHub para agentes

Rulesets ou branch protection: política central de branch (PR obrigatório, checks, bloqueio de force push). Required status checks: exigem evidência de CI antes do merge. CODEOWNERS: roteia revisão por caminho. Environments com revisores: controlam deploys e segredos, pausando o job até a aprovação. Approve and run workflows: evita workflow privilegiado rodando sozinho sobre código não confiável. Templates de PR: padronizam a descrição, mas não bloqueiam.

Controles dependem de configuração

A supervisão funciona, mas o enforcement exige ligar os controles: required checks e rulesets costumam ser tarefa de admin. Eles não vêm ativos por padrão em PRs de agentes nem são exclusivos de um plano.

Responsabilidades do agente por estágio do SDLC

Planning: issues, PR, aba Agents. Implementation: branch, commits, PR. Validation: runs, checks, artefatos. Deployment: geralmente restrito, com environments e aprovações. A maioria dos times começa por implementação e validação, onde PRs e workflows dão pontos de controle naturais. Fronteiras: limitar diretórios por política e ownership, tratar workflows e infraestrutura como risco maior e preferir trabalho via PR mesmo em automação.

Observabilidade mínima e trilha de auditoria

Conjunto mínimo por tarefa: plano estruturado no PR, PR com histórico de commits limitado, links dos workflow runs dos checks obrigatórios, artefatos enviados e resultados de revisão. Trilha mínima: objetivo declarado, plano inspecionável, changeset limitado, evidência automática, julgamento humano e resultado claro (merge, revert ou escalonamento). Evidência boa aponta um workflow run e um commit específicos.

Política de iteração e hooks

Se um check obrigatório falha, o agente pode revisar o branch e reexecutar; se o mesmo check falha duas vezes, ele escala ao humano com o que falhou, o que tentou, a evidência e o próximo passo sugerido. Hooks executam comandos em pontos fixos (pré-ação para bloquear, pós-ação para auditar, erro para escalar), independentes do raciocínio do modelo, e complementam a revisão humana.

Perfil do agente e MCP namespace control

O perfil tem três fronteiras: capability (ferramentas permitidas), visibility (se o agente é selecionável na interface) e delegation (quais subagentes chama e como ocorrem handoffs). Em MCP, namespace control é habilitar um subconjunto estreito em vez de curingas; trocar a lista por curinga é mudança de alto risco, revisada como dependência sensível. Além disso, no cloud agent as ferramentas MCP configuradas são usadas sem pedir aprovação.

Macetes para memorizar

💡 Rubrica do contributor model

I-S-E-O-P-F: Intenção, Escopo, Evidência, Ownership, Política, Fallback. Diff fora do plano é S; sem rollback é F.

💡 Instrução x enforcement

Instrução pede, allowlist proíbe. Se o controle é texto, é distrator.

💡 Plan-first x mesmo PR

Difícil de desfazer, plano primeiro; fácil de desfazer, tudo junto.

💡 Contrato da tarefa

Entra contexto e restrição; sai plano, PR e evidência; passa quando checks, segurança, escopo e rollback batem.

💡 Contextos do Actions

github.* é o evento, vars.* é a configuração central, env.* é o job agora.

💡 Seções do implementation-planner

O-T-I-C-N: Overview, Technical Approach, Implementation Plan, Considerations, Not Included.

💡 Política de iteração

Falhou uma vez, ajuste; falhou duas, chame alguém.

Tabela de memorização: não confunda

Quando é...	...e quando é
Plan-first PR Mudança de alto risco ou difícil de reverter (autenticação, infra/, produção): valida a intenção antes de qualquer código.	**Plano e execução no mesmo PR** Mudança de baixo risco e fácil de reverter (docs, typos) quando velocidade importa; checks e revisões continuam bloqueando o merge.
Excessive suspicion O revisor rejeita o PR porque foi escrito por IA, sem analisar o conteúdo.	**Excessive trust** O revisor aprova sem ler porque veio de automação e seria consistente.
Planless execution O PR chega com diff e sem plano nem justificativa; resolve com seção de plano no template e revisão pré-merge.	**Hidden reasoning** Há diff e checks, mas faltam premissas, pontos de decisão e contexto de execução; resolve exigindo plano, linkando runs e registrando decisões no PR.
Over-permissioned agent Workflow com escrita e segredos amplos; mitigue com GITHUB_TOKEN de menor privilégio, environments com revisores e restrição de quem dispara.	**Blind trust in automation** Regra de que CI verde basta; mitigue somando CODEOWNERS, revisões obrigatórias e aprovações por risco.
Required status check Bloquear o merge até a evidência de CI passar.	**Environment com revisores** Pausar um job específico (deploy, uso de segredo de produção) até alguém aprovar.
Job output lido via needs Passar dado de um job para outro (ex.: resumo do plano do job plan para o job implement).	**Step output** Passar dado entre steps do mesmo job.

vars.* Configuração gerida centralmente e reutilizável por vários workflows (ex.: limite mínimo de cobertura).	**env.* e github.*** env.* configura o job em tempo de execução; github.* traz metadados do evento que disparou o run.

Números que caem

Item	Valor
Falhas do mesmo check obrigatório antes de escalar	**2**
Itens da rubrica do contributor model	**6 (Intent, Scope, Evidence, Ownership, Policy, Fallback)**
Itens da trilha mínima de auditoria	**6 (objetivo, plano, changeset, evidência, julgamento humano, resultado)**
Formas de impor a fronteira entre planejar e executar	**4 (planner read-only, handoff, tool gating, plan mode)**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Repetir a proibição em maiúsculas no prompt ou no arquivo de instruções resolve a fronteira.
Como não cair: Ênfase e instruções continuam sendo orientação; a fronteira vem de allowlist de tools, gates e regras.

⚠ **Alerta de prova:** A descrição do PR diz que o agente está muito confiante, então dá para aprovar.
Como não cair: Confiança do agente não é evidência; avalie por checks, revisões e scans. Alerta aberto ou teste falhando manda voltar ao ciclo ou escalar.

⚠ **Alerta de prova:** Tirar o check da lista de obrigatórios até o agente estabilizar.
Como não cair: Remover check é bypass e causa drift de governança; o mesmo check falhando duas vezes manda escalar.

⚠ **Alerta de prova:** Reexecutar o teste até passar e considerar a tarefa concluída.
Como não cair: Reexecutar sem mudar nada não resolve a causa; o loop termina nos critérios de sucesso ou no humano.

⚠ **Alerta de prova:** Approve and run workflows existe para economizar minutos do Actions.
Como não cair: O propósito é impedir que workflow com acesso privilegiado rode sozinho sobre código não revisado.

⚠ **Alerta de prova:** Como o agente escreveu o código, ele responde pelo incidente; ou o GitHub, porque os checks passaram.
Como não cair: Humanos que definiram a tarefa, concederam permissões, configuraram controles e aprovaram respondem pelo resultado.

⚠ **Alerta de prova:** Salvar o resumo em step output e ler direto no outro job.
Como não cair: Step output vale só dentro do job; entre jobs é job output consumido via needs. Logs e env não são contrato de dados.

⚠ **Alerta de prova:** Omitir tools no frontmatter do agente planejador deixa ele restrito.
Como não cair: Omitir tools libera todas as ferramentas; o implementation-planner do tutorial usa tools read, search e edit (sem execute e sem web).

⚠ **Alerta de prova:** Os bloqueios de merge já vêm ligados para PRs de agentes, ou só existem no Enterprise.
Como não cair: Required checks e rulesets dependem de configuração, geralmente feita por admin; controle desligado não bloqueia nada.

⚠ **Alerta de prova:** Liberar todas as ferramentas MCP com curinga é neutro, porque o agente pede aprovação a cada uso.
Como não cair: No cloud agent, ferramentas MCP configuradas rodam sem aprovação; ampliar com curinga é mudança de alto risco.

⚠ **Alerta de prova:** O agente usa automaticamente o mesmo token que o CI guarda como secret do Actions.
Como não cair: O ambiente do agente tem fronteira de segredos própria e não herda os segredos de CI.

⚠ **Alerta de prova:** CODEOWNERS ou environment resolvem a política de branch para 300 repositórios.
Como não cair: Política central de branch é ruleset ou branch protection; CODEOWNERS roteia revisão por caminho e environment controla deploys e segredos.

⚠ **Alerta de prova:** Deploy é o estágio de menor risco ou pode ser autônomo se os testes passaram.
Como não cair: Deploy é geralmente restrito, com environments e aprovações; comece agentes por implementação e validação.

⚠ **Alerta de prova:** Usar continue-on-error: true no passo que comenta no PR quando o workflow é disparado por workflow_dispatch.
Como não cair: Gating defensivo: if: github.event_name == 'pull_request' no job; mascarar o erro só esconde a falha.

Como cai na prova

📝 **Como cai:** Cenário de empresa com um sintoma no PR do agente e a pergunta: qual anti-pattern e qual mitigação?
Como resolver: Identifique o sintoma (sem plano, sem premissas, permissão ampla, CI verde basta) e depois confira se a mitigação da alternativa é a daquele anti-pattern; os distratores trocam mitigações entre anti-patterns.

📝 **Como cai:** Um controle do GitHub para um objetivo (política central, evidência de CI, revisão por pasta, pausar job, aprovar run).
Como resolver: Mapeie o verbo: centralizar política de branch é ruleset; exigir CI é required status check; rotear revisor por caminho é CODEOWNERS; pausar job com segredo é environment; liberar execução sobre código não confiável é Approve and run workflows.

📝 **Como cai:** Item da rubrica que falha em um PR descrito em detalhes.
Como resolver: Elimine o que o enunciado diz que está atendido (checks passaram é Evidence ok, plano claro é Intent ok) e sobra o item violado.

📝 **Como cai:** Qual o próximo passo depois de falha, alerta aberto ou confiança alta do agente?
Como resolver: Descarte aprovar por confiança, fechar alerta, remover check e reexecutar até passar; a resposta é voltar ao ciclo (revisar plano, ajustar, reavaliar) ou escalar.

📝 **Como cai:** Desenho de workflow: passar dados entre jobs, configuração central, gating de evento, arquivo grande entre jobs.
Como resolver: Job output + needs para dados; vars.* para configuração central; if com github.event_name no job; upload e download de artefato para arquivo sem commitar no repositório.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
agente de planejamento alterou código mesmo com proibição no prompt	**Allowlist de tools só com ferramentas de leitura**
mesmo check falhou duas vezes	**Escalar ao humano com falha, tentativas, evidência e próximo passo**
PR com diff, sem plano nem justificativa	**Planless execution**
só o diff, sem premissas nem pontos de decisão	**Hidden reasoning**
permissions: write-all e acesso amplo a segredos	**Over-permissioned agent**

se o CI passou, pode publicar	**Blind trust in automation; CODEOWNERS, revisões obrigatórias e aprovações por risco**
não diz como desfazer nem a quem escalar	**Fallback da rubrica**
diff mexe em arquivos fora do plano	**Scope da rubrica**
ferramentas só depois de o plano ser aceito, em orquestração	**Tool gating no orquestrador**
Copilot CLI interativo, perguntas de esclarecimento e plano antes de codar	**Plan mode com Shift+Tab**
transformar expectativa de plano em garantia	**Plan gate como required status check em ruleset**
job com segredo de assinatura só após aprovação da segurança	**Environment com revisores obrigatórios**
configuração sensível aplicada do topo do branch	**Implantar por referência explícita: SHA ou tag**
premissas, restrições e riscos no plano do implementation-planner	**Seção Considerations**

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

- Instrução orienta; allowlist de tools, gates e regras impõem.
- Confiança do agente nunca é evidência; sinais do GitHub são.
- O agente não aprova nem mescla o próprio PR: ele propõe e itera; humanos e regras do repositório decidem o merge.
- Humanos que definiram, permitiram e aprovaram respondem pelo resultado.
- Controle desligado não bloqueia nada: required checks e rulesets precisam ser configurados.

Essenciais para a prova (checklist)

- ☐ Sei diferenciar assistente de agente pelo critério de quem cria branch, commits e PR e itera com sinais.
- ☐ Sei classificar um item como entrada, saída ou critério de sucesso do contrato da tarefa.
- ☐ Sei as quatro formas de separar planejamento e execução e quando usar plan-first ou PR único.
- ☐ Sei as seções do template de PR do Learn (Plan, Evidence, Review checklist) e o que vai em cada uma.
- ☐ Sei os quatro anti-patterns de agentes e a mitigação exata de cada um.
- ☐ Sei os seis itens da rubrica do contributor model e os erros excessive suspicion e excessive trust.
- ☐ Sei qual controle do GitHub usar para política de branch, evidência de CI, revisão por caminho, pausa de job e aprovação de run.
- ☐ Sei passar dados entre jobs (job output + needs) e arquivos entre jobs (upload e download de artefato).

- ☐ Sei que o agente não herda segredos do CI e que evidência deve apontar run e commit.
- ☐ Sei as três fronteiras do perfil do agente: capability, visibility e delegation.

Questões de revisão do capítulo

R5.1 · Preparar a arquitetura de agentes e os processos do SDLC

Ao revisar os PRs abertos por um agente de refatoração, a líder técnica da Editora Lumen percebe que o agente começa a alterar código sem nenhuma abordagem inspecionável, e os PRs chegam com o diff, sem plano nem justificativa. Qual anti-pattern é esse e qual mitigação o Learn associa a ele?

A. Planless execution, mitigado com seção de plano no template de PR e revisão pré-merge
B. Over-permissioned agent, mitigado com GITHUB_TOKEN de menor privilégio e environments
C. Blind trust in automation, mitigado com mais testes unitários no workflow de CI do PR
D. Excessive suspicion, mitigado com bloqueio de PRs de agentes até nova auditoria interna

R5.2 · Preparar a arquitetura de agentes e os processos do SDLC

Uma empresa com 300 repositórios quer aplicar a mesma política de branch para PRs de agentes (PR obrigatório, checks e bloqueio de force push) de forma centralizada e consistente. Qual controle da tabela do Learn cumpre esse papel?

A. Environments com revisores
B. Templates de pull request
C. Rulesets ou branch protection
D. CODEOWNERS em cada repositório

R5.3 · Preparar a arquitetura de agentes e os processos do SDLC

Em uma empresa, o Copilot Chat explica por que um teste falhou, enquanto o Copilot cloud agent recebe a issue, cria um branch, abre um PR e ajusta o código quando um check falha. Qual característica torna o segundo um comportamento de agente?

A. Usar um modelo de linguagem maior que o do assistente de chat
B. Responder mais rápido e com menos interação com o usuário
C. Manter o objetivo em várias etapas e iterar com base em sinais
D. Gerar respostas em linguagem natural sobre o código do projeto

R5.4 · Preparar a arquitetura de agentes e os processos do SDLC

Um agente abriu um PR para corrigir uma vulnerabilidade e escreveu na descrição que está muito confiante de que o problema foi resolvido. O alerta de code scanning continua aberto e um teste falhou. Qual é o próximo passo coerente com o ciclo plan, act, evaluate?

A. Aprovar o merge, pois a confiança declarada pelo agente é alta o bastante
B. Fechar o alerta como falso positivo para liberar o pull request
C. Reexecutar o teste até passar e então considerar a tarefa concluída
D. Voltar ao ciclo: revisar o plano, ajustar a mudança, reavaliar ou escalar

Respostas comentadas

R5.1. A) Planless execution, mitigado com seção de plano no template de PR e revisão pré-merge

Por quê: Na tabela do Learn, planless execution aparece como PR com diff e sem plano ou justificativa; a mitigação é exigir seção de plano no template de PR e revisão antes do merge.

✗ B) Incorreta. Over-permissioned trata de acesso amplo demais, não da falta de plano.
✗ C) Incorreta. Blind trust é confiar só no CI; e mais testes não criam um plano inspecionável.
✗ D) Incorreta. Excessive suspicion é um erro de avaliação do contributor model, não esse anti-pattern, e bloquear não resolve a falta de plano.

> **Dica de prova:** Diff sem plano: planless execution, resolvido com template e revisão.

R5.2. C) Rulesets ou branch protection

Por quê: Na tabela do Learn, rulesets e branch protection fornecem política centralizada de branch, evitando merges inseguros e aplicando guardrails de forma consistente.

✗ A) Incorreta. Environments controlam deploys e segredos.
✗ B) Incorreta. Templates padronizam a descrição do PR, mas não bloqueiam nada.
✗ D) Incorreta. CODEOWNERS roteia revisões por caminho; não define a política de branch.

> **Dica de prova:** Política de branch central: rulesets.

R5.3. C) Manter o objetivo em várias etapas e iterar com base em sinais

Por quê: Assistente sugere e devolve o controle ao usuário. Agente mantém um objetivo em várias etapas, usa ferramentas, produz branch, commits e PR e itera com base em sinais.

✗ A) Incorreta. O tamanho do modelo não define se o sistema age como agente.
✗ B) Incorreta. Velocidade de resposta não é o critério; o que importa é agir no fluxo e iterar.
✗ D) Incorreta. Responder em linguagem natural é o que o assistente também faz; não caracteriza agente.

> **Dica de prova:** Agente persegue o objetivo e itera com sinais; assistente sugere.

R5.4. D) Voltar ao ciclo: revisar o plano, ajustar a mudança, reavaliar ou escalar

Por quê: O ciclo plan, act, evaluate é um loop: se checks falham ou o risco continua, o agente revisa o plano, ajusta a ação e reavalia, ou escala para um humano. A avaliação usa sinais do sistema, não a confiança do agente.

✗ A) Incorreta. A avaliação deve se basear em sinais do sistema, não na confiança declarada pelo agente.

✗ B) Incorreta. Descartar o alerta sem análise esconde o sinal que mostra que a tarefa não terminou.

✗ C) Incorreta. Reexecutar sem mudar nada não resolve a causa, e o alerta de segurança continuaria aberto.

> **Dica de prova:** Confiança do agente não é evidência; sinais do GitHub são.

06

CAPÍTULO 6 · 15–20% DA PROVA

Realizar avaliação, análise de erros e ajuste

Neste domínio eu preciso definir o que é sucesso para uma tarefa de agente, gerar sinais objetivos de avaliação, achar a causa-raiz quando algo falha e ajustar o comportamento do jeito certo. A prova cobra muito a classificação de causa-raiz (reasoning error, tool misuse, context issue, problema de ambiente) e a área de ajuste correspondente (prompts e instruções, memória e estado, configuração de ferramentas). Também caem os scans (code scanning, secret scanning, push protection, dependency review), as regras de proteção que transformam avaliação em política e recursos como hooks, excludeAgent e session logs. Meu erro típico é aceitar CI verde como suficiente ou corrigir antes de olhar as evidências.

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

Tópicos-chave

Restrições operacionais do contrato

São o que o agente não pode fazer no caminho: não alterar workflows sem revisão da plataforma, não criar novos segredos, não fazer push direto no main. Fazem parte das entradas; checks obrigatórios são critério de sucesso e nunca podem ser pulados.

Critério alinhado à intenção

CI passou é necessário, mas nem sempre suficiente: se a tarefa é corrigir vulnerabilidade, o critério é o alerta resolvido; se é latência abaixo de uma meta, precisa de um check que meça latência; se é refatoração sem mudar comportamento, os testes de comportamento existentes devem passar sem alteração. Métricas de volume (linhas, arquivos, commits) podem melhorar enquanto o objetivo falha.

Onde escrever critérios e onde ver sinais

Critérios de aceite vão na issue, referenciados na descrição do PR, para agente e revisor usarem a mesma base. Os sinais aparecem nos status checks do PR e na aba Checks; os logs completos ficam na aba Actions. Testes, lint e build viram sinal automático com workflows em .github/workflows/ disparados por pull_request ou push.

Sinais quantitativos x qualitativos

Quantitativos: checks passando, número de alertas de code scanning, contagem de falhas repetidas do mesmo check. Qualitativos: revisão humana, comentários e a rubrica do contributor model; seguir uma rubrica fixa não torna o sinal numérico.

Proteção que impõe a avaliação

Branch protection ou ruleset com status checks obrigatórios selecionados, pelo menos uma revisão aprovando e bloqueio de push direto no branch protegido. Pode-se exigir também que o branch esteja atualizado com a base antes do merge, para que os checks reflitam o código combinado. Scan que roda mas não bloqueia precisa virar required check.

Scans automáticos como sinais

Code scanning (CodeQL e upload de SARIF de terceiros) aponta vulnerabilidades no código; secret scanning detecta segredos e push protection impede o envio de segredos suportados; dependency review mostra dependências adicionadas, removidas ou atualizadas, inclusive indiretas em lockfiles, e a dependency review action falha por padrão ao achar pacote vulnerável. Alerta de segurança e check falhando são bloqueadores, mesmo com testes verdes.

Validação embutida do cloud agent

O Copilot cloud agent valida o próprio código com CodeQL, consulta o GitHub Advisory Database para dependências novas, roda secret scanning e pede segunda opinião ao Copilot code review, sem exigir licença de GHAS, Secret Protection ou Code Security. Essas ferramentas podem ser ligadas ou desligadas individualmente em Validation tools, em Settings > Copilot > Cloud agent.

Copilot code review automático

Donos de repositório e organização podem configurar revisão automática de PRs. Por padrão essas revisões não contam para as aprovações exigidas: são sinal qualitativo adicional ao lado dos checks e da revisão humana.

Categorias de causa-raiz

Reasoning errors: suposições ou decisões incorretas, como interpretar mal requisito, implementar lógica errada ou ignorar critério de aceite explícito. Tool misuse: workflows mal configurados, comandos ou scripts incorretos, falha em disparar ou usar workflows. Context issues: informação ausente, obsoleta ou conflitante (estado de PR desatualizado, decisão que só estava no chat). Problema de ambiente: por exemplo, bloqueio de rede pelo firewall visto no session log.

Evidências para analisar falhas

Logs na aba Actions, mudanças e discussões do PR, histórico de commits do branch e resultados e artefatos de workflow. O primeiro passo é sempre revisar essas evidências. Comparar a intenção (issue e descrição do PR) com o resultado (commits, mudanças, saídas e logs) separa interpretação errada, solução incorreta e falha na execução ou validação.

Loop de feedback

Rodar o agente, observar falhas por logs e artefatos, identificar a causa-raiz, corrigir prompts, memória ou ferramentas e reexecutar o workflow. Corrigir antes de observar, reexecutar até passar ou trocar o modelo junto com outro ajuste atrapalham o diagnóstico. Na prática: abrir o PR, ver mudanças, ir à aba Checks e inspecionar os logs.

Três áreas de ajuste

Prompts e instruções: clarificar critérios de aceite e restrições, registrando regras recorrentes em arquivos de instrução (ex.: .instructions.md com applyTo nos testes). Memória e estado: decisões claras em issue ou PR, remover contexto conflitante, fonte única. Configuração de ferramentas: arquivos em .github/workflows/, gatilhos corretos (push, pull_request), permissões e required checks. Melhorias devem ser rastreáveis em atualizações do PR, citando commits e runs.

Refinar acesso a ferramentas

Ajuste é configuração, não texto: remover execute da lista tools impede shell; restringir tools a read e search e trocar github/* por ferramentas específicas tira capacidades indevidas; tools omitido libera tudo. No Copilot CLI, troque --allow-all-tools por --allow-tool com os comandos necessários, reservando --allow-all-tools para sandbox.

excludeAgent em .instructions.md

Arquivos .instructions.md aceitam applyTo para caminhos e excludeAgent opcional com os valores code-review ou cloud-agent, para que um desses agentes ignore aquelas instruções. Checklist só para revisão: excludeAgent: cloud-agent; padrão só para o cloud agent: excludeAgent: code-review.

Session logs e hooks para análise

Session logs são o trace do cloud agent, com raciocínio e ferramentas, acessíveis pelo PR ou pela aba Agents, e os commits do agente trazem link para eles. Para base de análise sem interferir no agente, hooks postToolUse (ferramenta, argumentos e resultado) e errorOccurred (objeto error, errorContext e indicador recoverable, útil para alertar e escalar).

Ajustes de memória e de MCP

Fato obsoleto no Copilot Memory é removido pelo dono do repositório, revisando e excluindo o fato; desligar o recurso para a empresa é desproporcional. Se Start MCP Servers não lista as tools de um servidor que roda com uv, instale o uv no copilot-setup-steps. Bloqueio de rede a um registry interno se resolve com regra custom do firewall ou pré-instalação no setup steps.

Feedback e desvio de escopo

Para registrar a avaliação de qualidade de um PR ou comentário do Copilot, use thumbs up ou thumbs down com motivo. Se um item listado em Not Included do plano do implementation-planner aparece no diff, é desvio de escopo em relação ao plano aprovado.

Macetes para memorizar

Causa-raiz

Pensou errado, Raciocínio; usou errado, Ferramenta; faltou ou conflitou informação, Contexto; o ambiente barrou, Ambiente.

Área de ajuste

Raciocínio se trata em instruções, contexto em memória e estado, ferramenta em configuração.

Loop de feedback

R-O-C-C-R: Rodar, Observar, Causa, Corrigir, Reexecutar.

Scans

Código é code scanning, segredo é secret scanning, envio de segredo é push protection, pacote é dependency review.

Validação do cloud agent

CASC: CodeQL, Advisory Database, Secret scanning, Code review; sem GHAS.

Critério

CI verde é ingresso, não é prêmio: o prêmio é a intenção da issue.

Tabela de memorização: não confunda

Quando é...	...e quando é
Reasoning error A informação estava disponível (critério escrito na issue) e o agente decidiu ou interpretou errado.	**Context issue** A informação faltava, estava desatualizada ou conflitava entre artefatos (decisão só no chat, valores diferentes em issue e PR).
Tool misuse Script inexistente, parâmetros trocados, workflow no evento errado ou sem permissão.	**Problema de ambiente** O agente tentou o passo certo e foi barrado pelo ambiente, como bloqueio de rede do firewall.
Code scanning Vulnerabilidades no código escrito, por análise estática com CodeQL ou SARIF.	**Dependency review** Dependências novas ou indiretas em lockfiles com vulnerabilidades conhecidas; como required check, bloqueia o merge.

Secret scanning Detectar segredos no repositório.	**Push protection** Impedir que segredos suportados sejam enviados no push; tirar o segredo do commit antes de seguir.
Hook postToolUse Gravar cada uso de ferramenta com argumentos e resultado.	**Hook errorOccurred** Gravar ou alertar erros da sessão com mensagem, contexto e se é recuperável.
excludeAgent: cloud-agent Instruções que devem valer só para o Copilot code review.	**excludeAgent: code-review** Instruções que devem valer só para o cloud agent, sem gerar ruído na revisão.

Números que caem

Item	Valor
Aprovações mínimas na proteção de branch descrita	**Pelo menos 1 revisão aprovando**
Ferramentas da validação embutida do cloud agent	**4 (CodeQL, GitHub Advisory Database, secret scanning, Copilot code review)**
Valores aceitos por excludeAgent	**code-review ou cloud-agent**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Critério de sucesso é só o CI passar.
Como não cair: O critério reflete a intenção: alerta resolvido, meta de latência medida, comportamento preservado.

⚠ **Alerta de prova:** Pedir ao agente que explique no PR que a vulnerabilidade foi tratada ou que ficou mais rápido.
Como não cair: Declaração do agente não é evidência; o critério precisa ser verificável por sinais.

⚠ **Alerta de prova:** Mesclar porque os testes passaram e abrir issue para tratar o dependency review depois.
Como não cair: Falha de segurança é bloqueador antes do merge; corrigir, não remover o check nem reexecutar.

⚠ **Alerta de prova:** Pedir ao agente que leia os alertas ou agendar CodeQL semanal no main para impedir alertas novos.
Como não cair: Marque o check do CodeQL como required check no ruleset.

⚠ **Alerta de prova:** Revisão do Copilot code review substitui a aprovação humana ou o Copilot vira aprovador no CODEOWNERS.
Como não cair: Por padrão ela não conta para aprovações exigidas; é sinal adicional.

⚠ **Alerta de prova:** Validação do cloud agent exige licença de Advanced Security ou só ocorre após o merge.
Como não cair: Roda enquanto o agente trabalha e não exige GHAS, Secret Protection nem Code Security.

⚠ **Alerta de prova:** Desligar a segunda opinião do code review no ruleset ou no copilot-setup-steps.
Como não cair: É em Validation tools, em Settings > Copilot > Cloud agent.

⚠ **Alerta de prova:** Workflow que só roda em push no main é reasoning error ou falha do modelo.
Como não cair: Falha em disparar workflow é tool misuse; ajuste o gatilho para pull_request.

⚠ **Alerta de prova:** Workflow sem permissão e não exigido se resolve reescrevendo o prompt ou aumentando retenção.
Como não cair: Revisar .github/workflows/, gatilhos, permissões e required checks.

⚠ **Alerta de prova:** Trace do cloud agent fica na allowlist do firewall, no CODEOWNERS ou na Copilot Memory.
Como não cair: Fica nos session logs, acessíveis pelo PR ou aba Agents e linkados nas mensagens dos commits.

⚠ **Alerta de prova:** Usar preToolUse negando ferramentas para montar base de análise.
Como não cair: Negar interfere nas decisões do agente; para registrar sem interferir, postToolUse e errorOccurred.

⚠ **Alerta de prova:** Mover checklist de revisão para o AGENTS.md para tirar do cloud agent.
Como não cair: AGENTS.md é lido por agentes; use excludeAgent: cloud-agent no frontmatter do .instructions.md.

⚠ **Alerta de prova:** Listar as tools do MCP no prompt ou trocar o type para http para o agente encontrá-las.
Como não cair: Sem a dependência (uv) instalada no setup steps o servidor não sobe; citar nomes no prompt não resolve.

⚠ **Alerta de prova:** Exigir duas aprovações resolve PRs verdes que quebram o main depois de outro merge.
Como não cair: Exija branch atualizado com a base antes do merge.

Como cai na prova

📝 **Como cai:** Relato de falha do agente seguido de: como classificar a causa-raiz?
Como resolver: Pergunte se a informação estava disponível. Estava e o agente errou: reasoning. Faltava ou conflitava: context. Comando, script ou gatilho errado: tool misuse. Bloqueio externo como rede: ambiente.

📝 **Como cai:** Causa já identificada e a pergunta é qual ajuste fazer.
Como resolver: Case a causa com a área: prompts e instruções, memória e estado ou configuração de ferramentas. Descarte trocar modelo, aumentar retenção ou ampliar permissões.

📝 **Como cai:** PR aceito que não atingiu o objetivo; como reescrever o critério?
Como resolver: Escolha o sinal mensurável ligado à intenção da issue e rejeite declarações do agente, tamanho do diff, mais revisores ou repetir o CI.

📝 **Como cai:** Qual recurso gera ou impõe determinado sinal de segurança?
Como resolver: Identifique o objeto (código, segredo, envio, dependência) e depois se a pergunta pede detectar ou bloquear; bloquear o merge exige required check.

📝 **Como cai:** Ajustar acesso de um custom agent que fez algo indevido.
Como resolver: A resposta é mudar o frontmatter (tools explícito, sem execute, MCP sem curinga); aviso em instruções ou prompt é distrator.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
lógica errada para regra descrita na issue ou critério de aceite ignorado	**Reasoning error**
script inexistente, parâmetros trocados, workflow no evento errado	**Tool misuse**
decisão estava só no chat; issue e PR com valores diferentes	**Context issue; ajuste de memória e estado**
session log com downloads recusados por bloqueio de rede	**Problema de ambiente; regra custom do firewall**

métrica que dá para contar, como checks passando ou número de alertas de code scanning	**Sinal quantitativo**
comentário do revisor ou rubrica do contributor model	**Sinal qualitativo**
push bloqueado por token de API no commit	**Push protection; remover o segredo do commit**
scanner de terceiros que gera SARIF	**Upload do SARIF para o code scanning num workflow do PR**
lockfile com dependência indireta vulnerável e bloqueio de merge	**Dependency review action como required check**
dependência nova adicionada pelo cloud agent	**Consulta ao GitHub Advisory Database**
PR com X vermelho	**Aba Checks e logs do workflow**
aviso no canal de incidentes a cada erro da sessão	**Hook errorOccurred**
custom agent de docs rodando scripts de build	**Remover execute da lista tools**
registrar avaliação de qualidade da saída do Copilot	**Thumbs up ou thumbs down com motivo**

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

- Primeiro as evidências (logs, PR, commits, artefatos), depois a correção.
- CI verde é necessário, mas o critério precisa refletir a intenção da tarefa.
- Sinal de segurança falhando bloqueia o merge, mesmo com testes verdes.
- Revisão automática do Copilot code review não conta como aprovação por padrão.
- Ajustar acesso é mudar configuração (tools, allow-tool), não escrever avisos.

Essenciais para a prova (checklist)

☐ Sei identificar restrições operacionais e escrever critérios alinhados à intenção.

☐ Sei onde ficam critérios (issue referenciada no PR) e onde aparecem os sinais (status checks, aba Checks, aba Actions).

☐ Sei separar sinais quantitativos de qualitativos.

☐ Sei as regras de proteção que impõem a avaliação, incluindo branch atualizado com a base.

☐ Sei o papel de code scanning, SARIF, secret scanning, push protection e dependency review.

☐ Sei o que a validação embutida do cloud agent faz e onde desligar cada ferramenta.

☐ Sei classificar causa-raiz em reasoning error, tool misuse, context issue e ambiente.

☐ Sei as três áreas de ajuste e o loop de feedback.

☐ Sei usar excludeAgent, session logs e os hooks postToolUse e errorOccurred na análise.

- [ ] Sei refinar tools de custom agents e do Copilot CLI.

Questões de revisão do capítulo

R6.1 · Realizar avaliação, análise de erros e ajuste

Na retrospectiva de um incidente, a equipe viu que PRs de agentes estavam sendo mesclados com alertas novos de code scanning, porque o workflow do CodeQL rodava mas não bloqueava nada. Qual ajuste de restrição corrige isso?

A. Pedir ao agente que leia os alertas antes de abrir o PR
B. Agendar o CodeQL para rodar uma vez por semana no main
C. Remover o CodeQL e confiar na revisão do Copilot
D. Marcar o check do CodeQL como required check no ruleset

R6.2 · Realizar avaliação, análise de erros e ajuste

Uma equipe criou um workflow de testes, mas ele só roda quando alguém dispara pela aba Actions, e os PRs dos agentes chegam sem resultado. Qual ajuste faz os testes virarem sinal automático em cada PR?

A. Mover o workflow para a pasta .github/agents do repositório
B. Agendar o workflow com schedule para rodar uma vez por dia
C. Configurar o gatilho on: pull_request com branches: [main]
D. Adicionar workflow_dispatch com entradas para cada agente

R6.3 · Realizar avaliação, análise de erros e ajuste

A equipe de qualidade da Cosméticos Bela montou um guia interno de causas-raiz de falhas de agentes. Qual destes exemplos pertence à categoria reasoning errors?

A. Implementar lógica incorreta para uma regra de negócio descrita na issue
B. Usar um estado desatualizado do pull request ao retomar a tarefa
C. Disparar o workflow de validação no evento errado do repositório
D. Executar um script de migração com parâmetros trocados no CI

R6.4 · Realizar avaliação, análise de erros e ajuste

A issue pedia uma refatoração que não mudasse o comportamento do módulo de faturamento. A equipe avaliava o agente pelo número de arquivos alterados e pela redução de linhas. Qual critério alinha a avaliação à intenção?

A. Testes de comportamento existentes passando sem alteração
B. Redução percentual de linhas acima de uma meta fixa
C. Quantidade de commits feitos pelo agente durante a sessão inteira
D. Tempo total que o agente levou para abrir o PR

Respostas comentadas

R6.1. D) Marcar o check do CodeQL como required check no ruleset

Por quê: Sinais de segurança só funcionam como gate quando são obrigatórios. Ajustar a configuração de ferramentas inclui workflows, eventos corretos, permissões e required checks.

✗ A) Incorreta. Orientação ao agente não impede o merge.
✗ B) Incorreta. Rodar no main depois do merge não impede a entrada do problema.
✗ C) Incorreta. Remover o scanning elimina um sinal objetivo de segurança.

> **Dica de prova:** Scan que não bloqueia: torne-o required check.

R6.2. C) Configurar o gatilho on: pull_request com branches: [main]

Por quê: O Learn recomenda workflows em .github/workflows/ disparados por pull_request ou push, para que testes, lint e build gerem status checks visíveis no PR.

✗ A) Incorreta. Workflows ficam em .github/workflows/; a pasta de agentes guarda perfis de custom agents.
✗ B) Incorreta. Um agendamento diário não avalia cada PR no momento da mudança.
✗ D) Incorreta. workflow_dispatch continua exigindo disparo manual.

> **Dica de prova:** Sinal automático em PR: gatilho pull_request.

R6.3. A) Implementar lógica incorreta para uma regra de negócio descrita na issue

Por quê: Reasoning errors são suposições ou decisões incorretas: interpretar mal requisitos, implementar lógica incorreta, ignorar critérios de aceite.

✗ B) Incorreta. Estado de PR desatualizado é context issue.
✗ C) Incorreta. Evento errado de workflow é tool misuse.
✗ D) Incorreta. Script com parâmetros errados é tool misuse.

> **Dica de prova:** Lógica errada: reasoning error.

R6.4. A) Testes de comportamento existentes passando sem alteração

Por quê: Critérios de avaliação devem refletir a intenção de desenvolvimento. Métricas de volume, como linhas ou arquivos, podem melhorar enquanto o objetivo real falha.

✗ B) Incorreta. Menos linhas não garante comportamento igual.

✗ C) Incorreta. Número de commits não mede preservação de comportamento.

✗ D) Incorreta. Velocidade não mede a intenção da tarefa.

> **Dica de prova:** Avalie a intenção, não o volume de mudança.

DIA DA PROVA

Estratégia de prova

O GH-600 mede se você opera agentes dentro do SDLC usando o GitHub como system of record e control plane, então quase toda questão é um cenário de empresa com um sintoma e pede o controle, a configuração ou a classificação correta. A lógica que mais se repete é instrução orienta, plataforma impõe: alternativas que resolvem com texto no prompt, aviso no README ou confiança do agente costumam estar erradas. A outra metade é factual e exata sobre o Copilot cloud agent, custom agents, MCP, copilot-setup-steps, firewall e Copilot CLI: nomes de chaves, prefixos, limites e precedências. Os distratores trocam controles vizinhos (CODEOWNERS x environment x required check) e mitigações entre anti-patterns. O guia avisa que a maioria das questões cobre recursos em GA, mas recursos em preview de uso comum podem aparecer.

Como responder

✓ A prova é supervisionada, tem 120 minutos e é oferecida só em inglês, com múltipla escolha, múltipla resposta e possíveis componentes interativos; treine os termos em inglês exatamente como aparecem (required status checks, rubber-stamping, flapping, steering).

✓ A aprovação exige 700 de 1000 e o número exato de questões não é divulgado: controle o tempo pelo relógio, não por uma contagem fixa, e não trave em uma questão.

✓ Em múltipla resposta, leia quantas são pedidas (DUAS, TRÊS) e avalie cada alternativa isoladamente como verdadeira ou falsa antes de combinar.

✓ Elimine primeiro alternativas que dependem de orientação (prompt, instrução, banner, documento) ou da confiança do agente quando a pergunta pede bloqueio, garantia ou evidência.

✓ Nas questões de classificação (causa-raiz, conflito, modo de falha, anti-pattern, nível de autonomia), recite a definição de cada categoria e procure o detalhe do enunciado que só uma delas explica.

✓ Distribua a revisão pelo peso: ferramentas e ambiente vale 20 a 25%; arquitetura e SDLC, avaliação e multiagente valem 15 a 20% cada; memória e guardrails, 10 a 15% cada.

✓ Quando duas alternativas parecem corretas, prefira a que usa o mecanismo nativo e mais específico do GitHub (ruleset, CODEOWNERS com revisão exigida, environment, required check) e respeita o menor privilégio.

✓ Marque para revisão as questões de fato de produto em que hesitou e volte a elas no fim, quando outras questões podem ter lembrado o detalhe.

Erros que derrubam candidatos

✗ Julgar o risco de uma mudança pelo tamanho do diff em vez de onde ela é aplicada, reversibilidade e velocidade de impacto.

✗ Aceitar CI verde ou a confiança declarada pelo agente como critério de sucesso.

✗ Misturar o que vale para o cloud agent com o que vale para IDE e CLI (registry de MCP, OAuth, firewall, prompt files).

✗ Esquecer que perfis de agente, copilot-setup-steps e hooks só valem a partir do branch padrão.

✗ Achar que omitir tools restringe o agente, quando libera todas as ferramentas.

✗ Confundir user-invocable: false com disable-model-invocation: true e as mitigações de over-permissioned agent com as de blind trust in automation.

✗ Escolher retries sem limite, remover check ou mesclar sem aprovação em vez de escalar quando um limite é atingido.

Na semana da prova

- ☐ Refaça as provas simuladas cronometrando 120 minutos e revise cada erro pelo motivo da alternativa errada, não só pela resposta certa.
- ☐ Monte uma folha com os números e nomes exatos: 59 minutos, timeout-minutes até 59, 30.000 caracteres, 28 dias, 90 e até 400 dias de retenção, COPILOT_MCP_, copilot/, job copilot-setup-steps.
- ☐ Revise as tabelas que mais caem: controles do GitHub, risco Low a Critical, níveis de autonomia, causas-raiz, tipos de conflito, modos de falha e fonte da verdade.
- ☐ Leia o YAML dos padrões de Actions (needs, concurrency com workflow e ref, workflow_run completed, permissions mínimas) até reconhecer de olho.
- ☐ Como a prova só é oferecida em inglês, verifique no agendamento a possibilidade, prevista no guia oficial, de pedir 30 minutos adicionais quando a prova não está disponível no seu idioma.
- ☐ Na véspera, revise só os pontos de lembrar de cada domínio e durma bem; nada de conteúdo novo.

REVISÃO FINAL

Folha de cola

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

1. Gerenciar memória, estado e execução

‣ informação que some quando a sessão termina → **Memória de curto prazo**

‣ padrão consolidado para reaproveitar em tarefas futuras → **Memória de longo prazo**

‣ fonte persistente e revisável por qualquer pessoa → **Memória externa em issues, PRs e documentação**

‣ regras de validação diferentes conforme a sessão → **Instruções do repositório**

‣ mesmo passo a passo colado no prompt a cada release → **Instruções ou skills reutilizáveis**

⚠ Prompt e chat não são duráveis nem compartilhados; registre decisões na issue ou no PR.

⚠ Reter mais aumenta o contexto obsoleto; o certo é expirar saídas temporárias e podar artefatos.

⚠ Artefato excluído não pode ser restaurado; a exclusão é feita na aba Actions por quem tem permissão.

2. Implementar guardrails e responsabilização

‣ agente lê, resume e sugere, mas não altera nada → **Read-only autonomy**

‣ testes, builds e staging pré-aprovados sem aprovação a cada vez → **Execute with guardrails**

‣ rotação de segredos e mudança de workflow só após aprovação explícita → **Human-authorized execution**

‣ alteração em .github/workflows/, infra/ ou security/ → **High: CODEOWNERS, aprovações explícitas, checks mais rígidos**

‣ deploy ou segredos de produção → **Critical: gate de environment com revisores e trilha de auditoria**

⚠ Workflow, infra/ e security/ são High por padrão; tamanho do diff não define risco.

⚠ Isso é High; Critical exige gate de environment com revisores e trilha de auditoria.

⚠ Gera governance fatigue; humanos nos pontos de decisão: merge, deploy e segredo.

3. Implementar o uso de ferramentas e a interação com o ambiente

‣ agente criado em branch de feature não aparece na lista → **Perfil precisa estar no branch padrão**

‣ Copilot começa sem a ferramenta instalada pelo setup → **Job precisa se chamar copilot-setup-steps**

‣ banco PostgreSQL para os testes do agente → **Chave services no copilot-setup-steps**

‣ arquivos de ponteiro em vez dos binários → **actions/checkout com lfs: true no setup steps**

‣ host bloqueado no curl pelo Bash, mas liberado no setup e no MCP → **Firewall só cobre processos do agente pela ferramenta Bash**

⚠ name é opcional e hífen é permitido; o perfil precisa estar mesclado no branch padrão.

⚠ O caminho é .github/workflows/copilot-setup-steps.yml e o job tem de se chamar copilot-setup-steps; workflow_dispatch só serve para testar à mão.

⚠ Só steps, permissions, runs-on, services, snapshot e timeout-minutes são personalizáveis.

4. Orquestrar a coordenação multiagente

‣ merge limpo e sistema quebrado → **Conflito semântico; merge queue**

‣ PR toca áreas com aprovações de times diferentes → **Conflito de política**

‣ dois agentes com correções incompatíveis para o mesmo alerta → **Esforço duplicado; escalar**

‣ conflitos devem falhar cedo como check → **git merge --no-commit do main como required check**

‣ PR existe, sem check nem relatório de validação → **Partial execution**

⚠ O requisito é isolamento de caminhos; sem ele há instabilidade.

⚠ Um agente, um branch dedicado; branch compartilhado mantém commits intercalados e dono ambíguo.

⚠ Espera entre jobs é needs; workflow_run reage ao término de outro workflow.

5. Preparar a arquitetura de agentes e os processos do SDLC

‣ agente de planejamento alterou código mesmo com proibição no prompt → **Allowlist de tools só com ferramentas de leitura**

‣ mesmo check falhou duas vezes → **Escalar ao humano com falha, tentativas, evidência e próximo passo**

‣ PR com diff, sem plano nem justificativa → **Planless execution**

‣ só o diff, sem premissas nem pontos de decisão → **Hidden reasoning**

‣ permissions: write-all e acesso amplo a segredos → **Over-permissioned agent**

⚠ Ênfase e instruções continuam sendo orientação; a fronteira vem de allowlist de tools, gates e regras.

⚠ Confiança do agente não é evidência; avalie por checks, revisões e scans. Alerta aberto ou teste falhando manda voltar ao ciclo ou escalar.

⚠ Remover check é bypass e causa drift de governança; o mesmo check falhando duas vezes manda escalar.

6. Realizar avaliação, análise de erros e ajuste

‣ lógica errada para regra descrita na issue ou critério de aceite ignorado → **Reasoning error**

‣ script inexistente, parâmetros trocados, workflow no evento errado → **Tool misuse**

‣ decisão estava só no chat; issue e PR com valores diferentes → **Context issue; ajuste de memória e estado**

‣ session log com downloads recusados por bloqueio de rede → **Problema de ambiente; regra custom do firewall**

‣ métrica que dá para contar, como checks passando ou número de alertas de code scanning → **Sinal quantitativo**

⚠ O critério reflete a intenção: alerta resolvido, meta de latência medida, comportamento preservado.

⚠ Declaração do agente não é evidência; o critério precisa ser verificável por sinais.

⚠ Falha de segurança é bloqueador antes do merge; corrigir, não remover o check nem reexecutar.

GABARITO

Respostas do diagnóstico

D1. B) Resumir o histórico em pontos-chave com referências → revise o capítulo 1

Por quê: Manutenção de memória: expirar saídas temporárias, podar artefatos desatualizados e resumir histórico que não precisa mais de detalhe em pontos-chave e referências.

✗ A) Incorreta. Duplicar contexto cria fontes conflitantes e mistura discussão com regras.

✗ C) Incorreta. Memória de curto prazo não persiste e não resolve o excesso de detalhe.

✗ D) Incorreta. Levar tudo para o prompt aumenta o ruído e o risco de usar informação ultrapassada.

> **Dica de prova:** Histórico velho e longo: resuma com referências.

D2. B) Abrir o PR existente, ler descrição e issue, ver os commits e a aba Checks e seguir dali → revise o capítulo 1

Por quê: Como o GitHub preserva commits e workflow runs, o agente retoma a partir do estado armazenado: abre o PR, revisa descrição e issue, confere commits e resultados na aba Checks e continua do último ponto.

✗ A) Incorreta. Descartar o progresso registrado desperdiça trabalho e não alinha a sessão às decisões anteriores.

✗ C) Incorreta. Contexto copiado de chat não é estado durável e pode estar incompleto ou desatualizado.

✗ D) Incorreta. Refazer a tarefa repete passos e pode divergir das decisões já tomadas.

> **Dica de prova:** Retomar é reancorar no PR existente, não recomeçar.

D3. B) Memória de curto prazo → revise o capítulo 1

Por quê: Hierarquia do Learn: curto prazo (contexto da tarefa atual, não persiste), longo prazo (conhecimento curado reutilizável) e externa (artefatos duráveis, fonte da verdade).

✗ A) Incorreta. O Learn organiza a memória em curto prazo, longo prazo e externa; processos repetíveis viram instruções ou skills.

✗ C) Incorreta. Memória de longo prazo guarda conhecimento curado e reutilizável entre tarefas, como decisões e padrões.

✗ D) Incorreta. Memória externa fica em sistemas duráveis, como issues e PRs, e persiste justamente para ser revisada depois.

> **Dica de prova:** Some quando a sessão acaba: memória de curto prazo.

D4. D) CODEOWNERS, aprovações explícitas e checks mais rígidos → revise o capítulo 2

Por quê: O Learn trata .github/workflows/, infra/ e security/ como alto risco por padrão (small diff, big consequence); o controle recomendado é CODEOWNERS, aprovações explícitas e checks mais fortes.

✗ A) Incorreta. PR com required checks é o controle de risco médio, como atualizações de dependência.

✗ B) Incorreta. Gate de environment é o controle de nível Critical, como deploy e segredos de produção.

✗ C) Incorreta. Automação completa é para risco baixo, como docs e formatação; tamanho do diff não define o risco.

> **Dica de prova:** Workflow alterado é High, mesmo com diff pequeno.

D5. A) Read-only autonomy → revise o capítulo 2

Por quê: Os quatro níveis do Learn: read-only, propose-only, execute with guardrails e human-authorized execution. Inspecionar e recomendar sem alterar é read-only.

✗ B) Incorreta. Execute with guardrails envolve rodar workflows pré-aprovados, como testes e builds.
✗ C) Incorreta. Human-authorized execution trata de ações de alto impacto liberadas por aprovação humana.
✗ D) Incorreta. Propose-only já permite criar branches e pull requests, o que o agente de triagem não faz.

> **Dica de prova:** Recomenda sem tocar no repositório: read-only.

D6. C) Um workflow de guardrail que só aceita aprovações de usuários humanos → revise o capítulo 2

Por quê: Workflows de guardrail impõem regras que as configurações sozinhas não expressam, como contar aprovações só de humanos e não de identidades de automação. Continuam valendo branch protection e CODEOWNERS.

✗ A) Incorreta. Registrar depois não bloqueia o merge.
✗ B) Incorreta. Avisos não impedem execução; controle só em documentação é anti-pattern.
✗ D) Incorreta. O agente como revisor anula a aprovação humana.

> **Dica de prova:** Aprovação de bot não conta: workflow de guardrail.

D7. D) Roda como check no PR, mas o agente só o usa no branch padrão → revise o capítulo 3

Por quê: O copilot-setup-steps.yml só é usado pelo agente quando está no branch padrão; ao ser alterado, roda como workflow comum e mostra o resultado no PR para validação.

✗ A) Incorreta. Não existe aprovação de admin nas configurações para esse arquivo.
✗ B) Incorreta. A validação acontece automaticamente no PR quando o arquivo muda.
✗ C) Incorreta. O workflow não é acionado para o agente enquanto não estiver no branch padrão.

> **Dica de prova:** Setup steps: validados no PR, usados a partir do branch padrão.

D8. D) Ele é compilado em um lock file, executado pelo GitHub Actions → revise o capítulo 3

Por quê: O Markdown expressa a intenção e o frontmatter define os limites; o workflow é compilado em um lock file executado pelo GitHub Actions.

✗ A) Incorreta. O workflow não é um perfil de custom agent.
✗ B) Incorreta. Agentic Workflows rodam no Actions, não no Chat.
✗ C) Incorreta. A execução acontece em runners do Actions.

> **Dica de prova:** Agentic Workflow: Markdown compilado em lock file.

D9. D) O GITHUB_TOKEN não dispara novos runs; usar token de GitHub App → revise o capítulo 3

Por quê: A restrição do GITHUB_TOKEN evita execuções recursivas. Para encadear workflows, use token de GitHub App ou PAT e desenhe o fluxo para não criar loops.

✗ A) Incorreta. Não há regra que restrinja o CI a humanos, e trocar o gatilho não ataca a causa.
✗ B) Incorreta. O push foi concluído; o que não ocorre é o disparo de novos workflows.
✗ C) Incorreta. Branch novo também gera evento de push; a causa está no token usado.

> **Dica de prova:** Push com GITHUB_TOKEN não dispara outro workflow (salvo dispatch).

D10. B) Esforço duplicado com correções incompatíveis; escalar a decisão para um humano → revise o capítulo 4

Por quê: O Learn cita esforço duplicado como tipo de conflito e lista dois agentes com correções incompatíveis para o mesmo alerta entre os limites de escalonamento.

✗ A) Incorreta. Agentes não devem aprovar o trabalho uns dos outros para desempatar.
✗ C) Incorreta. Mergear os dois e corrigir depois leva o problema para produção.
✗ D) Incorreta. Decidir por quem termina primeiro é resolver por tempo, não por correção.

> **Dica de prova:** Mesmo alerta, correções incompatíveis: escalar.

D11. B) user-invocable: false → revise o capítulo 4

Por quê: user-invocable: false tira o agente da seleção manual; disable-model-invocation: true faria o oposto do desejado, impedindo o uso como subagente na orquestração.

✗ A) Incorreta. Isso impediria justamente o uso como subagente.
✗ C) Incorreta. Sem ferramentas o agente não faria o trabalho, e continuaria visível.
✗ D) Incorreta. target define o ambiente, não a visibilidade no seletor.

> **Dica de prova:** Só subagente, fora do seletor: user-invocable: false.

D12. D) Revisores não conseguem entender por que o sistema agiu como agiu → revise o capítulo 4

Por quê: A orquestração deve aparecer em eventos do Actions, PRs, runs e artefatos. Coordenação oculta não deixa evidência, torna falhas difíceis de diagnosticar e reduz a confiança.

✗ A) Incorreta. O problema apontado não é desempenho, e sim falta de visibilidade.
✗ B) Incorreta. Onde o script roda não determina o modelo usado pelos agentes.
✗ C) Incorreta. Não há essa cobrança; o risco é de observabilidade e confiança.

> **Dica de prova:** Orquestração invisível gera falha impossível de diagnosticar.

Agora, treine no formato real

Faça as 6 provas completas do curso GH-600: 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 GitHub. As questões são inéditas e não reproduzem a prova real.