

Guia de preparação AZ-401

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

5 capítulos por domínio	**32** questões comentadas	**70%** nota de corte
6 provas completas no curso		

ANTES DE COMEÇAR

Como usar este guia

Este guia foi feito para ser estudado junto com as 6 provas completas do curso AZ-401: 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 100 minutos, no formato do exame. Referência de aprovação: 70%.

Domínio (capítulo)	Peso	Neste guia
1. Implementar uma estratégia de instrumentação	10-15%	17 conceitos · 14 pegadinhas
2. Projetar e implementar pipelines de build e lançamento	35-40%	20 conceitos · 14 pegadinhas
3. Projetar e implementar processos e comunicações	15-20%	18 conceitos · 14 pegadinhas
4. Projetar e implementar um plano de segurança e conformidade	10-15%	18 conceitos · 14 pegadinhas
5. Projetar e implementar uma estratégia de controle do código-fonte	10-15%	18 conceitos · 14 pegadinhas

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

Estude um pouco todo dia, de graça

Instagram @strecht.simulados

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

Seguir no Instagram →

YouTube Strecht Simulados

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

Inscrever-se no canal →

DIAGNÓSTICO

Você já sabe isso?

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

D1 · Implementar uma estratégia de instrumentação

A aplicação de pedidos do Atacarejo Bom Preço tem sete microsserviços instrumentados com Application Insights. Os usuários relatam erros intermitentes e o time quer ver, em uma única visão, os componentes, as chamadas entre eles e onde estão as falhas e a lentidão. Qual recurso usar?

A. Testes de disponibilidade com alertas por região de origem
B. Metrics explorer com a CPU de cada nó do cluster de contêineres
C. Live metrics do Application Insights com filtros por servidor
D. Application map do Application Insights, com os nós e chamadas

D2 · Implementar uma estratégia de instrumentação

O gerente de entrega da Calçados Passo Firme suspeita que o trabalho está se acumulando na coluna de revisão de código do quadro, o que atrasa as entregas. Qual widget do painel mostra, ao longo do tempo, a quantidade de itens em cada coluna e evidencia esse gargalo?

A. Sprint Burndown, com o trabalho restante da sprint atual
B. Velocity, com o esforço planejado e o concluído em cada sprint
C. Cumulative Flow Diagram, com faixas por coluna do quadro
D. Query Tile, com a contagem de uma consulta salva

D3 · Implementar uma estratégia de instrumentação

Um alerta de latência do Azure Monitor disparou para o serviço de rastreamento de entregas da Logística Pronta Entrega. O SRE quer que o Azure Copilot Observability Agent analise as camadas de aplicação, infraestrutura e plataforma e preserve o resultado para a passagem de turno. Qual é o fluxo indicado?

A. Abrir o chat do agente na página de Logs e copiar a conversa para um e-mail
B. Abrir o Log Analytics, escrever uma consulta KQL manual e exportar o resultado para CSV
C. Abrir o alerta e selecionar Investigate, depois salvar a investigação como issue
D. Criar um ticket no ServiceNow e aguardar a correlação automática do agente

D4 · Projetar e implementar pipelines de build e lançamento

O environment de produção da Telefonia Fala Mais tem a verificação Exclusive lock. Várias execuções do pipeline chegam ao stage de produção quase ao mesmo tempo, e a equipe quer que todas sejam implantadas, uma de cada vez, na ordem em que chegaram. O que configurar no stage?

A. condition: always() no stage
B. lockBehavior: runLatest
C. lockBehavior: sequential
D. dependsOn: [] no stage de produção

D5 · Projetar e implementar pipelines de build e lançamento

A Rede Hospitalar Vida Longa usa pools de agentes com conjuntos de dimensionamento de VMs (VMSS) na própria assinatura do Azure e gasta muito tempo mantendo imagens e escala. Ela avalia o Managed DevOps Pools. Qual diferença de arquitetura é correta?

A. No Managed DevOps Pools, as VMs dos agentes ficam em uma assinatura da Microsoft, não na sua
B. O Managed DevOps Pools substitui os agentes por runners do GitHub Actions hospedados no Azure
C. No Managed DevOps Pools, as VMs ficam na sua assinatura, mas com escala manual
D. O Managed DevOps Pools só oferece as imagens padrão dos agentes hospedados, sem personalização

D6 · Projetar e implementar pipelines de build e lançamento

A equipe de segurança da Fábrica de Software Prisma Dev recompilou internamente uma versão corrigida de um pacote open source e a publicou em um feed interno. O feed principal dos projetos tem upstreams para esse feed interno e para o registro público. Como garantir que a versão corrigida seja a usada?

A. Colocar o registro público antes do feed interno, pois públicos vêm primeiro
B. Adicionar os dois feeds no arquivo de configuração de cada projeto
C. Publicar a versão corrigida com número menor que o da versão pública
D. Colocar o feed interno antes do registro público na ordem dos upstreams

D7 · Projetar e implementar processos e comunicações

A Holding Vale Verde quer que o Copilot responda sempre em português do Brasil e siga as diretrizes de segurança da empresa para todos os membros da organização no GitHub. A organização tem Copilot Business. Onde essas instruções valem se forem configuradas como instruções personalizadas da organização?

A. No Copilot Chat, no Copilot code review e no Copilot cloud agent do GitHub.com
B. Nas IDEs suportadas, como VS Code, Visual Studio, JetBrains e Xcode
C. No Copilot CLI, que lê as instruções da organização a partir do servidor
D. Só para quem recebe a licença do Copilot por meio da própria organização no GitHub

D8 · Projetar e implementar processos e comunicações

A equipe de Integrações da Indústria Química Reator registrou no Microsoft Entra um cliente MCP próprio para acessar o servidor MCP remoto do Azure DevOps. O cliente só usa ferramentas de leitura, mas o desenvolvedor concedeu escopos de gerenciamento para não ter problemas. Qual é a orientação da Microsoft?

A. Trocar os escopos por um PAT de administrador no cabeçalho
B. Conceder todos os escopos disponíveis e usar X-MCP-Readonly para impedir qualquer alteração
C. Manter os escopos de gerenciamento, pois habilitar um toolset já limita o que o cliente pode fazer
D. Conceder só escopos de leitura; o escopo nunca ultrapassa as permissões do usuário

D9 · Projetar e implementar processos e comunicações

A analista de qualidade da Gráfica Folha Branca criou casos de teste manuais para validar uma história de usuário e quer garantir rastreabilidade entre a história e os testes que a verificam. Qual tipo de vínculo usar entre a história e os casos de teste?

A. Parent e Child, deixando os testes como filhos da história de usuário
B. Related, o tipo usado para ligar requisitos e testes
C. Predecessor e Successor, com os testes como sucessores da história
D. Tested by, da história de usuário para os casos de teste

D10 · Projetar e implementar um plano de segurança e conformidade

Uma Azure Function da Transportes Expresso Sul precisa gravar em uma conta de armazenamento. O time de segurança proíbe segredos e chaves em configurações do aplicativo e quer que a rotação de credenciais seja responsabilidade do Azure. Qual identidade usar?

A. Identidade gerenciada da Function com papel RBAC no armazenamento
B. Chave de acesso da conta de armazenamento referenciada a partir do Key Vault
C. Entidade de serviço com certificado carregado e renovado pela equipe de operações
D. Entidade de serviço com segredo de cliente guardado nas configurações do app

D11 · Projetar e implementar um plano de segurança e conformidade

A Logística Rio Branco usa OIDC entre GitHub Actions e Azure. Um job de testes de integração, disparado em pull_request e sem environment, precisa acessar um recurso de teste. Qual subject a credencial federada deve usar?

A. repo:rio-branco/api:environment:Production
B. repo:rio-branco/api:ref:refs/heads/main
C. repo:rio-branco/api:pull_request
D. repo:rio-branco/api:job:testes-integracao

D12 · Projetar e implementar um plano de segurança e conformidade

A equipe de nuvem da Operadora Rio Grande quer analisar templates Terraform e Bicep de um repositório do Azure DevOps conectado ao Defender for Cloud, rodando só a varredura de IaC para reduzir o tempo do pipeline. O que usar?

A. O widget Requirements quality configurado para os templates
B. O dependency scanning do GitHub Advanced Security for Azure DevOps
C. A extensão Microsoft Security DevOps com a categoria de IaC
D. O secret scanning com push protection habilitado no repositório

01

CAPÍTULO 1 · 10-15% DA PROVA

Implementar uma estratégia de instrumentação

Aqui eu preciso saber qual painel, gráfico, alerta ou consulta responde a cada pergunta de operação, no Azure DevOps, no GitHub e no Azure Monitor. A prova cobra widgets do Azure DevOps (Cycle Time, CFD, Velocity, Build History, Deployment status), recursos do Application Insights (application map, transação ponta a ponta, live metrics, Standard tests) e KQL básico (summarize, bin, percentile, join). Também cai auditoria (Azure DevOps e GitHub) e os agentes de operação: Azure Copilot Observability Agent e Azure SRE Agent, com papéis e modos de execução. Onde mais erro é escolher a ferramenta que mostra dados parecidos mas não exatamente o pedido. Leio a pergunta procurando o verbo: ver ao vivo, ver topologia, ver uma requisição, alertar por KQL.

Neste capítulo

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

Tópicos-chave

Widgets de fluxo e entrega do Azure DevOps

Cumulative Flow Diagram mostra itens por coluna ao longo do tempo e evidencia gargalos; Velocity mostra trabalho concluído por sprint para planejar capacidade e não deve ser usado como KPI; Lead Time e Cycle Time medem tempos de itens. Build History mostra histograma das execuções com duração e cores; Deployment status combina status por ambiente e taxa de testes; Test Results Trend (Advanced) mostra tendência de testes de vários pipelines; Requirements quality mostra requisitos testados e não testados.

Métricas de disponibilidade

Pela abordagem de monitoramento da Microsoft, os tempos a minimizar são tempo para detectar, mitigar e remediar problemas (TTD, TTM e TTR). Tempos de pipeline e volume de commits não medem recuperação.

Relatórios de pipeline

Pipeline pass rate mostra as tarefas que mais falham com tendência diária; Pipeline duration mostra as tarefas mais demoradas; Test analytics mostra os testes que mais falham com detalhes. GitHub Actions metrics mostram minutos, tempo de fila e falhas e exigem owner ou View organization Actions metrics.

Application Insights: diagnóstico

Application map mostra topologia, chamadas entre componentes e pontos de falha; detalhes de transação de ponta a ponta, a partir da visão Performance, mostram uma requisição por todos os serviços com o tempo de cada dependência; Live metrics acompanha falhas e exceções em tempo real filtrando servidor; a pesquisa de transações localiza telemetria já gravada.

Disponibilidade e anotações

O teste de ping de URL foi descontinuado e será aposentado em 2028; Standard tests validam certificado TLS e aceitam POST e cabeçalhos personalizados. Anotações de implantação são automáticas com a tarefa AzureWebApp quando o app tem APPLICATIONINSIGHTS_CONNECTION_STRING e o Application Insights está na mesma assinatura; a tarefa antiga de anotação deve ser removida.

Instrumentação com OpenTelemetry

Para novos apps, Microsoft OpenTelemetry Distro em .NET, Node.js e Python e Azure Monitor OpenTelemetry Distro em Java. Quem usa o SDK 2.x do .NET deve atualizar para o 3.x ou começar com o Distro.

Métricas no Azure Monitor

No Metrics explorer, splitting por dimensão (instância, código de resposta) mostra qual segmento puxa a métrica. Métricas de host da VM não incluem memória do sistema operacional nem processos; para isso, o monitoramento aprimorado instala o Azure Monitor Agent e coleta métricas do convidado.

Tipos de alerta

Log search alert avalia condição escrita em KQL com junções e filtros; alerta de métrica usa dados pré-calculados; alerta do log de atividades trata de operações de gerenciamento; Service Health avisa sobre problemas da plataforma Azure.

KQL essencial

Série temporal: where timestamp > ago(24h) | summarize count() by bin(timestamp, 5m) | render timechart. p95: summarize percentile(duration, 95) by name. Requisições com exceção: join kind=inner (exceptions) on operation_Id. Agregação com by exige summarize, não project; o simple mode do Log Analytics deixa explorar logs sem KQL.

Auditoria do Azure DevOps

Exige organização ligada ao Microsoft Entra ID, vem desativada por padrão, guarda eventos por 90 dias e não existe no Azure DevOps Server local. Registra mudanças de permissões, exclusões de recursos e políticas de branch; para reter mais e consultar com KQL, use audit stream para Azure Monitor Logs (tabela AzureDevOpsAuditing).

Audit log do GitHub

Streaming para destinos como Azure Event Hubs e Azure Blob Storage, com entrega pelo menos uma vez (pode duplicar) e eventos a partir da ativação. Stream quebrado gera aviso diário por e-mail aos owners e deve ser corrigido em até seis dias. Logs de uso da API do Copilot estão em preview para EMU e residência de dados, com corpo truncado acima de 1 MB.

Auditoria e logs dos agentes do Copilot

O filtro actor:Copilot mostra a atividade agêntica dos últimos 180 dias; agent_session_id liga o evento à sessão, actor_is_agent é sempre true e o campo user indica a pessoa que iniciou. O raciocínio e as ferramentas usadas ficam nos session logs, acessíveis pelo link em cada commit do agente; consultar sessões passadas exige sincronização ativa e política permitida. Logs da extensão no Visual Studio ficam na janela Output, em GitHub Copilot.

GitHub Insights

Pulse resume a atividade recente (PRs, issues, commits); Traffic mostra clones, visitantes dos últimos 14 dias e sites de referência para quem tem push; Contributors mostra commits por pessoa. Nos GitHub Projects, gráficos atuais agrupam e filtram e gráficos históricos (como o Burn up) acompanham mudanças no tempo; ficam visíveis para quem vê o projeto, não contam itens arquivados ou excluídos e o eixo Y pode somar, fazer média, mínimo ou máximo.

Alertas e eventos de pipeline

Para avisar a equipe quando o build quebra, crie assinatura de equipe na categoria Build com o modelo Build fails. Para um sistema externo reagir ao fim de workflows do GitHub Actions, assine o webhook workflow_run com a ação completed.

Azure Copilot Observability Agent

Chat a partir de Logs e deep investigation pelo botão Investigate no alerta funcionam sem provisionar nada; o chat é temporário e o resultado é preservado salvando como issue. O recurso do agente com identidade gerenciada só é exigido para operações autônomas: correlacionar alertas, criar issues e investigá-las, sem alterar o ambiente; para criar issues precisa de Issue Contributor no Azure Monitor workspace. Custom instructions em linguagem natural moldam correlação e criação de issues, mas não mudam deep investigations nem dão permissões.

Azure SRE Agent: criação, papéis e permissões

Criar exige Owner ou Contributor com User Access Administrator. Os papéis de dados são Reader, Standard User, Author e Administrator: SRE Agent Standard User conversa, Author customiza (sozinho não conversa nem conecta repositórios) e só SRE Agent Administrator aprova ações; Owner e Contributor do Azure não substituem esses papéis de dados. Com nível Reader, ações usam o fluxo on-behalf-of autorizado por um Administrator com conta do Microsoft Entra; o nível Privileged dá papéis de contributor por tipo de recurso.

Azure SRE Agent: operação

Run mode Review propõe e espera aprovação; Autonomous age sozinho; o modo é definido por response plan ou tarefa agendada, e a configuração do agente é só o padrão de reserva. Incident response plans roteiam incidentes por filtros (severidade, serviço) para um custom agent e podem ser pausados. Tool access policies: só o escopo global nega, deny global vence allow e ask, allow vence ask e no escopo de thread só há allow; integra Azure Monitor, PagerDuty e ServiceNow para incidentes e GitHub e Azure DevOps para correlacionar deploys.

Macetes para memorizar

Application Insights

Mapa = topologia; ponta a ponta = uma requisição; live = agora; search = passado.

KQL

Filtra, agrupa no balde (bin), desenha (timechart); p95 é percentile; ligar tabelas é join on operation_Id.

Auditoria do Azure DevOps

Entra, desligada, 90 dias; para mais, stream.

Papéis do SRE Agent

Standard User fala, Author ajusta, Administrator aprova.

Políticas de ferramentas do SRE Agent

Só o global diz não; deny > allow > ask.

Observability Agent

Conversa e investiga sem setup; recurso só para autonomia; issue guarda o contexto.

Disponibilidade

Detectar, mitigar, remediar: os três tempos que devem cair.

Tabela de memorização: não confunda

Quando é...	...e quando é
Application map Ver a topologia e onde estão falhas e lentidão entre componentes.	**Detalhes de transação de ponta a ponta** Ver uma requisição específica por todos os serviços com o tempo de cada dependência.

Live metrics Acompanhar ao vivo, durante uma liberação, filtrando um servidor.	**Pesquisa de transações** Localizar telemetria já gravada.
Log search alert Condição que exige KQL com junções e filtros.	**Alerta de métrica** Valor numérico pré-calculado, como CPU.
Cumulative Flow Diagram Descobrir em que coluna o trabalho está parando.	**Velocity** Planejar capacidade pelo trabalho concluído por sprint, sem usar como KPI.
Build History Histograma das execuções de um pipeline com duração e resultado.	**Deployment status** Status por ambiente e taxa de testes dos builds recentes.
Pulse Resumo da atividade recente de PRs, issues e commits.	**Traffic** Clones, visitantes e sites de referência.
Run mode Review Produção: propor mitigação e esperar aprovação.	**Run mode Autonomous** Ambientes não produtivos em que o agente pode agir sozinho.

Números que caem

Item	Valor
Peso do domínio no exame	**15-20%**
Retenção da auditoria do Azure DevOps	**90 dias**
Prazo para corrigir stream do audit log do GitHub	**até seis dias**
Atividade agêntica no filtro actor:Copilot	**últimos 180 dias**
Corpo nos logs de uso da API do Copilot	**truncado acima de 1 MB**
Visitantes no gráfico Traffic	**últimos 14 dias**
Aposentadoria do teste de ping de URL	**2028**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Usar o log de atividades para ver chamadas da aplicação ou memória da VM.
Como não cair: Log de atividades registra operações de gerenciamento; use transação ponta a ponta ou métricas de convidado com Azure Monitor Agent.

⚠ **Alerta de prova:** Usar Live metrics para ver a topologia entre microsserviços.
Como não cair: Topologia e dependências são do Application map.

⚠ **Alerta de prova:** Manter o ping de URL para validar TLS, POST e cabeçalhos.
Como não cair: Esses recursos estão nos Standard tests; o ping foi descontinuado.

⚠ **Alerta de prova:** Usar top 95 by duration para obter o percentil 95.
Como não cair: top 95 traz as 95 linhas mais lentas; use summarize percentile(duration, 95).

⚠ **Alerta de prova:** Usar project com by ou union para correlacionar requisições e exceções.
Como não cair: Agregação exige summarize; correlação é join on operation_Id.

⚠ **Alerta de prova:** Achar que a auditoria do Azure DevOps vem ligada e guarda sete anos ou que existe no Server local.
Como não cair: Desligada por padrão, 90 dias, exige Entra e não existe no Server.

⚠ **Alerta de prova:** Achar que o streaming do audit log do GitHub entrega exatamente uma vez ou guarda eventos sem limite quando quebra.
Como não cair: Entrega pelo menos uma vez; stream quebrado deve ser corrigido em até seis dias.

⚠ **Alerta de prova:** Procurar o raciocínio do agente no audit log.
Como não cair: Audit log registra eventos; o raciocínio fica nos session logs, linkados no commit.

⚠ **Alerta de prova:** Achar que o Observability Agent exige recurso provisionado para chat e investigação.
Como não cair: Só as operações autônomas exigem o recurso do agente.

⚠ **Alerta de prova:** Achar que o Observability Agent reinicia recursos ou fecha alertas sozinho.
Como não cair: Ele correlaciona, cria issues e investiga; quem muda o ambiente são pessoas.

⚠ **Alerta de prova:** Dar Contributor no Azure para resolver 403 no SRE Agent.
Como não cair: Papéis de dados do SRE Agent: Standard User, Author e Administrator (o único que aprova).

⚠ **Alerta de prova:** Achar que políticas de custom agent ou de thread podem negar ou anular um deny global.
Como não cair: Só o global nega; os demais escopos só ampliam dentro dos limites globais.

⚠ **Alerta de prova:** Usar Velocity como meta de produtividade.
Como não cair: A Microsoft orienta não tratar velocity como KPI.

⚠ **Alerta de prova:** Assinatura pessoal com Build completes para avisar a equipe de falhas.
Como não cair: Assinatura de equipe com Build fails.

Como cai na prova

📝 **Como cai:** Gerente pede um widget ou gráfico que mostre algo específico.
Como resolver: Identifique o objeto (itens, builds, testes, implantações) e a dimensão (tempo, coluna, ambiente); escolha o widget que mostra exatamente essa combinação.

📝 **Como cai:** Problema de produção a diagnosticar no Application Insights ou Azure Monitor.
Como resolver: Defina se a pergunta é topologia, uma requisição, tempo real, segmento da métrica ou dados do convidado e mapeie para Application map, ponta a ponta, Live metrics, splitting ou Azure Monitor Agent.

📝 **Como cai:** Escolher a consulta KQL correta entre quatro parecidas.
Como resolver: Confira a ordem (where antes de summarize), a função (percentile, count), o bin com o intervalo pedido e se a correlação usa join on operation_Id.

📝 **Como cai:** Configurar Observability Agent ou SRE Agent com governança.
Como resolver: Separe o que exige recurso ou permissão (autonomia, criar issues, aprovar ações) do que funciona direto; aplique Review em produção e o papel de dados mínimo.

📝 **Como cai:** Auditoria e retenção de eventos em Azure DevOps ou GitHub.
Como resolver: Lembre retenções e destinos: Azure DevOps 90 dias e stream para Azure Monitor Logs; GitHub streaming para Event Hubs ou Blob, at-least-once, seis dias para corrigir; agentes do Copilot com actor:Copilot por 180 dias.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
onde o trabalho está parando no quadro	**Cumulative Flow Diagram**
histograma de execuções com duração e cores	**Build History**
status por ambiente e taxa de testes	**Deployment status**

tendência de testes de vários pipelines	**Test Results Trend (Advanced)**
uma requisição por todos os serviços com tempos	**Detalhes de transação de ponta a ponta**
validar correção ao vivo filtrando servidor	**Live metrics**
qual instância ou código de resposta puxa o aumento	**Splitting por dimensão no Metrics explorer**
memória e processos da VM	**Monitoramento aprimorado com Azure Monitor Agent**
condição com junções e filtros em KQL	**Log search alert**
auditoria do Azure DevOps por mais de 90 dias com KQL	**Audit stream para Azure Monitor Logs (AzureDevOpsAuditing)**
clones, visitantes e sites de referência	**Traffic em Insights**
por que o agente fez essa mudança	**Session logs, pelo link no commit**
investigar alerta e preservar para a passagem de turno	**Investigate no alerta e salvar como issue**
rotear incidentes por severidade e serviço para custom agent	**Incident response plan do SRE Agent**

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

- Velocity não é KPI.
- Métricas de host da VM não trazem memória do sistema operacional nem processos.
- Chat do Observability Agent é temporário: salve como issue.
- Só SRE Agent Administrator aprova ações.
- Anotação automática de deploy exige connection string e mesma assinatura.

Essenciais para a prova (checklist)

- ☐ Sei qual widget do Azure DevOps responde a cada pergunta de fluxo, build, teste e implantação.
- ☐ Sei usar Application map, transação ponta a ponta, Live metrics e Standard tests.
- ☐ Sei quando usar splitting, métricas de convidado e cada tipo de alerta.
- ☐ Sei escrever KQL com where, summarize, bin, percentile, render e join.
- ☐ Sei as regras da auditoria do Azure DevOps e do streaming do audit log do GitHub.
- ☐ Sei onde encontrar auditoria, session logs e logs do Copilot.
- ☐ Sei usar Pulse, Traffic e os insights dos GitHub Projects.
- ☐ Sei o que o Observability Agent faz com e sem recurso provisionado.
- ☐ Sei configurar papéis, run modes, response plans e tool access policies do SRE Agent.

Questões de revisão do capítulo

R1.1 · Implementar uma estratégia de instrumentação

Um revisor da Fintech Conta Fácil quer entender por que o Copilot cloud agent fez certa mudança em um pull request e quais ferramentas usou para validá-la. Onde encontrar essa informação e como chegar a ela a partir do commit?

A. No audit log da organização, que guarda o texto completo das respostas do agente para cada commit
B. Nos session logs da sessão, que mostram raciocínio e ferramentas usadas; cada commit do agente traz o link para esses logs
C. No histórico de execuções do workflow copilot-setup-steps, que registra as decisões tomadas pelo agente
D. Na aba Insights do repositório, que mostra o raciocínio do agente por arquivo alterado no PR

R1.2 · Implementar uma estratégia de instrumentação

A Plataforma de Ensino Lumina configurou o Azure SRE Agent. Para incidentes de produção, o agente deve propor a mitigação e esperar aprovação antes de reiniciar recursos; para incidentes de homologação, pode agir sozinho. Como configurar?

A. Run mode Review no response plan de produção e Autonomous no response plan de homologação
B. Nível de permissão Reader para produção, já que leitura força aprovação manual
C. Um único run mode no nível do agente, aplicado igualmente a todos os incidentes
D. Papel SRE Agent Standard User para os aprovadores das mitigações de produção

R1.3 · Implementar uma estratégia de instrumentação

A equipe da Startup Agenda Fácil usa GitHub Projects e quer acompanhar, ao longo do tempo, quantos itens estão abertos, concluídos e não planejados, para ver se o trabalho avança. Que tipo de gráfico usar nos insights?

A. Um gráfico atual agrupado por responsável, sem histórico
B. O gráfico Traffic do repositório, com visitantes por dia
C. Um gráfico histórico, como o Burn up padrão
D. O gráfico Contributors, com os commits de cada pessoa

R1.4 · Implementar uma estratégia de instrumentação

Durante a liberação de uma correção na API de agendamentos da Clínica Bem Estar, o SRE quer acompanhar em tempo real se as falhas e as exceções caem logo após a implantação, filtrando um servidor específico. Qual recurso do Application Insights atende?

A. Pesquisa de transações (Search) do Application Insights
B. Live metrics do Application Insights, com filtro por servidor
C. Testes de disponibilidade do Application Insights
D. Application map do Application Insights

Respostas comentadas

R1.1. B) Nos session logs da sessão, que mostram raciocínio e ferramentas usadas; cada commit do agente traz o link para esses logs

Por quê: Sessões do cloud agent podem ser acompanhadas no painel de agentes; os session logs explicam o que o agente fez e cada commit aponta para eles, garantindo rastreabilidade.

✗ A) Incorreta. O audit log registra eventos e ações (com agent_session_id), não o raciocínio completo.
✗ C) Incorreta. O copilot-setup-steps prepara o ambiente antes do agente; não registra as decisões da sessão.
✗ D) Incorreta. Os gráficos de Insights não mostram o raciocínio do agente.

> **Dica de prova:** Por que o agente mudou isso: session logs, acessíveis pelo link no commit.

R1.2. A) Run mode Review no response plan de produção e Autonomous no response plan de homologação

Por quê: No Azure SRE Agent, run modes (Review ou Autonomous) controlam se o agente pede aprovação; a Microsoft recomenda Review para produção e Autonomous para ambientes não produtivos.

✗ B) Incorreta. Permissões controlam o acesso a recursos, não o fluxo de aprovação; sem permissão o agente pede acesso temporário (OBO).
✗ C) Incorreta. A configuração do agente é só o padrão de reserva; o modo é definido por response plan ou tarefa agendada.
✗ D) Incorreta. Só o papel SRE Agent Administrator pode aprovar ações.

> **Dica de prova:** SRE Agent: Review em produção, Autonomous em não produção, configurado por response plan.

R1.3. C) Um gráfico histórico, como o Burn up padrão

Por quê: Insights for Projects oferece gráficos atuais e históricos; os históricos mostram a evolução do estado dos itens.

✗ A) Incorreta. Gráficos atuais mostram a situação presente, não a evolução no tempo.
✗ B) Incorreta. Traffic mostra visitas e clones do repositório.
✗ D) Incorreta. Contributors mostra commits por pessoa.

> **Dica de prova:** Evolução de itens abertos e concluídos no GitHub Projects: gráfico histórico (Burn up).

R1.4. B) Live metrics do Application Insights, com filtro por servidor

Por quê: Live metrics mostra métricas e falhas em tempo real, ideal para validar uma implantação; com OpenTelemetry vem habilitado por padrão em ASP.NET Core, Java, Node.js e Python.

✗ A) Incorreta. A pesquisa localiza itens de telemetria já gravados, sem o acompanhamento ao vivo.

✗ C) Incorreta. Testes de disponibilidade verificam a resposta de endpoints periodicamente.

✗ D) Incorreta. O Application map mostra topologia e saúde dos componentes, não um fluxo em tempo real filtrável por servidor.

> **Dica de prova:** Validar uma correção durante a liberação, ao vivo: Live metrics.

02

CAPÍTULO 2 · 35-40% DA PROVA

Projetar e implementar pipelines de build e lançamento

Este é o maior domínio da prova e eu preciso dominar YAML do Azure Pipelines e do GitHub Actions com precisão de sintaxe: gatilhos, dependsOn e needs, condições, expressões, variáveis de saída, estratégias de deployment job e checks de environment. Também cai muito Azure Artifacts (views, upstream, funções, retenção) e GitHub Packages, agentes e runners (hospedados, auto-hospedados, Managed DevOps Pools, larger runners), estratégias de implantação (slots, canary, rolling, anéis, feature flags) e IaC com Bicep. O tema novo é GitHub Agentic Workflows com gh aw, safe-outputs e engines. Onde mais erro é trocar sintaxe de uma plataforma pela outra e confundir opções com nome parecido, como trigger x pr, always x succeededOrFailed, runLatest x sequential. Leio o enunciado procurando a restrição exata e elimino o que quebra essa restrição.

Neste capítulo

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

Tópicos-chave

Azure Artifacts: views e promoção

Só se publica no feed base, onde o pacote aparece na view @Local; versões validadas são promovidas para views como @Release e os consumidores se conectam a ela. Views comunicam qualidade; SemVer comunica natureza e risco da mudança. Uma versão publicada nunca é sobrescrita nem reutilizada; o pacote excluído vai para a Lixeira por 30 dias.

Upstream sources

O feed salva pacotes de origens públicas, mantendo dependências disponíveis mesmo com o registro público fora do ar; o arquivo de configuração (.npmrc, nuget.config) deve apontar para um único feed. Para pacote open source corrigido internamente, o upstream interno vem antes do público. Para servir de upstream a outros feeds, o feed precisa de visibilidade para membros da organização ou do Entra ID, compartilhado por uma view.

Escopo, funções e limites de feed

Feed com escopo de organização é visível em todos os projetos e não pode virar feed de projeto; projetos públicos estão sendo aposentados. Funções: Feed Reader (lista e baixa), Feed and Upstream Reader (Collaborator, também salva de upstream), Feed Publisher (Contributor, publica, promove, deprecia) e Feed Owner. A identidade Build Service precisa de Feed Publisher para publicar; o nível gratuito é 2 GiB por organização.

Retenção e tipos de pacote

A política de retenção exclui uma versão só quando o máximo de versões é atingido e ela não foi baixada recentemente; versões promovidas a views ficam isentas. As versões excluídas pela política vão para a Lixeira, e a limpeza roda uma vez por dia. Universal Packages, publicados pela Azure CLI, guardam arquivos genéricos grandes; um feed aceita vários tipos (inclusive Python e Cargo). npm: URL do feed no .npmrc do projeto e credenciais no .npmrc do usuário.

GitHub Packages

Oferece Container registry (Docker e OCI, com permissões granulares separadas do repositório) e registros como npm, Maven, Gradle e RubyGems (Maven e Gradle herdam permissões do repositório). O GITHUB_TOKEN publica com packages: write; npm usa setup-node com registry-url e NODE_AUTH_TOKEN. Para baixar pacote de outro repositório, o repositório precisa receber acesso ao pacote.

Versionamento

SemVer: MAJOR para mudança incompatível, MINOR para recurso compatível, PATCH para correção compatível; rótulo de pré-lançamento não substitui o MAJOR. O run number padrão do Azure Pipelines já é $(Date:yyyyMMdd).$(Rev:r), estilo CalVer, e $(Rev:r) reinicia quando o resto do número muda.

Estratégia de testes

L0 e L1 são testes de unidade (L0 rápidos em memória); L2 funcionais com dependências como SQL e sistema de arquivos sem implantação; L3 contra implantação testável; L4 contra produção. No exemplo da Microsoft: até L2 antes do commit, PR reprovado se L3 falhar e implantação bloqueada se L4 falhar. Na detecção do sistema do flaky test management, teste instável é o que falha e passa na nova execução dentro do mesmo pipeline; dá para evitar que testes instáveis derrubem o build mantendo a marcação flaky.

Resultados, cobertura e carga

PublishTestResults@2 só falha o job com failTaskOnFailedTests true; PublishCodeCoverageResults@2 precisa de pathToSources para mostrar o código-fonte. Cobertura como portão de PR: azurepipelines-coverage.yml com target e política de branch, só no Azure Repos. Azure Load Testing: identidade com Load Test Contributor, critérios repetidos na mesma métrica usam o menor limite e auto stop criteria encerram cedo pela taxa de erros.

Agentes do Azure Pipelines

Hospedados pela Microsoft: VM nova a cada job, descartada no fim; no nível gratuito cada job roda até 60 minutos e há 1.800 minutos por mês (pago até 360 por job). Auto-hospedados mantêm caches e software instalado. Managed DevOps Pools: VMs em assinatura da Microsoft, imagens personalizadas, injeção na rede virtual e escala automática; stateless é o padrão (recomendado contra ataques à cadeia de suprimentos) e stateful reaproveita agentes por até sete dias por padrão.

Runners do GitHub Actions

Larger runners hospedados oferecem mais hardware e IP estático; runners hospedados em uma Azure VNet acessam recursos privados sem manter máquinas. Runner groups restringem quais repositórios usam os runners e limitam concorrência. Em repositório público, evite runner auto-hospedado, pois código de forks pode comprometer a máquina e a rede interna; prefira runners efêmeros.

Gatilhos do Azure Pipelines

trigger com branches e paths include dispara CI só para uma pasta (path filter não funciona sem branch correspondente); pr: none desliga validação de PR e trigger: none desliga CI. Cron de schedules é em UTC e always: true roda mesmo sem mudanças; variáveis não podem ser usadas em schedules. resources.pipelines com trigger: true roda quando outro pipeline, até de outro projeto, termina.

Expressões, parâmetros e variáveis

${{ }} é avaliada na compilação; $[ ] em runtime; $( ) é macro que lê variáveis. Parâmetro de runtime tipo string com values restringe a escolha e é resolvido no template para selecionar stages; parâmetros não podem ser opcionais. Para passar valor entre jobs: task.setvariable com isOutput=true, dependsOn no job consumidor e leitura com dependencies; grupo de variáveis novo exige autorizar o pipeline em Pipeline permissions.

Ordem, condições e resiliência de jobs

Fan-out e fan-in com dependsOn (no GitHub Actions é needs; sem needs os jobs rodam em paralelo). succeeded() é o padrão, failed() só após falha, succeededOrFailed() roda após falha mas não após cancelamento, always() roda até com cancelamento. strategy: matrix com maxParallel limita combinações simultâneas; timeoutInMinutes limita o job, cancelTimeoutInMinutes dá tempo à limpeza e retryCountOnTaskFailure repete tarefa com falha transitória.

Reuso: templates, task groups e reusable workflows

Task groups não existem no YAML; o equivalente são templates de steps, jobs, stages e variáveis. extends com a verificação Required template no recurso protegido obriga o uso do template aprovado. Reusable workflows do GitHub aceitam até dez níveis de aninhamento e 50 workflows únicos por arquivo; permissões do GITHUB_TOKEN só se mantêm ou diminuem e segredos não são repassados automaticamente.

Environments e checks

Ordem: checks estáticos (como Branch control), aprovações pré, checks dinâmicos, aprovações pós e Exclusive lock; Branch control falha antes de acionar aprovadores. Query Azure Monitor alerts e Invoke REST API são portões por saúde. Exclusive lock com lockBehavior: sequential enfileira todas as execuções; runLatest (padrão) cancela as anteriores. No GitHub, o environment tem required reviewers, prevent self-review, wait timer e branches selecionadas.

Estratégias de deployment job

runOnce executa os hooks uma vez; rolling com maxParallel atualiza VMs em lotes; canary com increments roda preDeploy uma vez e deploy, routeTraffic e postRouteTraffic por incremento, com monitoramento em postRouteTraffic e rollback em on: failure. Com KubernetesManifest, canary usa percentage e termina com action promote ou reject; o input containers com a tag do build garante a imagem exata.

App Service, feature flags e exposição progressiva

Slot de staging com swap elimina downtime e permite rollback; swap com preview aplica as configurações de produção e pausa para validação; auto swap troca logo após implantar; deployment slot setting prende a configuração ao slot. No App Configuration, targeting filter libera por grupos, usuários e percentuais; time window filter liga por período; variant feature flag faz teste A/B. Implantação por anéis (tiers) amplia o público aos poucos.

IaC e banco de dados

what-if mostra o que o Bicep vai criar, alterar ou excluir sem aplicar. Template specs compartilham templates versionados por RBAC; deployment stacks com deny settings gerenciam recursos como unidade e bloqueiam alterações só no plano de controle; registro privado de módulos Bicep fica no Azure Container Registry; machine configuration do Azure Policy audita e aplica configuração de SO em VMs e servidores Arc. Esquema do Azure SQL por .dacpac usa SqlAzureDacpacDeployment@1.

GitHub Agentic Workflows

Definidos em Markdown e compilados com gh aw compile em .lock.yml, que é o que o Actions executa; os dois arquivos vão para o commit. gh aw init prepara o repositório, gh aw add-wizard escolhe engine, guia segredos e abre PR, gh aw update atualiza da origem preservando mudanças locais, gh aw logs e gh aw audit acompanham consumo e max-ai-credits limita gasto. Escritas passam por safe-outputs com o agente em leitura; engine copilot usa copilot-requests: write, claude exige ANTHROPIC_API_KEY e codex OPENAI_API_KEY; private: true não é importável.

Otimização e retenção

fetchDepth baixo (shallow fetch) acelera checkout; Cache@2 com key contendo o lockfile renova o cache quando dependências mudam. actions/cache remove caches sem acesso há mais de 7 dias e o limite padrão é 10 GB por repositório; concurrency com cancel-in-progress e paths-ignore cortam execuções redundantes, e queue: max (até 100 pendentes) não combina com cancel-in-progress. Artefatos e logs: até 90 dias em públicos e até 400 em privados; retention leases retêm só execuções escolhidas; pipeline artifacts substituem build artifacts.

Macetes para memorizar

Expressões

Chaves ${{ }} = compilação; colchetes $[ ] = runtime; parênteses $( ) = macro.

Ordem dos checks

Estático, aprova, dinâmico, aprova, tranca (Exclusive lock por último).

Hooks do canary

preDeploy uma vez; deploy, routeTraffic e postRouteTraffic por incremento; on: failure reverte.

Níveis de teste

L0/L1 memória, L2 dependências, L3 implantação de teste, L4 produção.

Feeds

Publica no @Local, promove para @Release; número publicado é para sempre.

Plataformas

Azure Pipelines fala dependsOn; GitHub Actions fala needs.

Limites de agentes

Gratuito: 60 minutos por job e 1.800 minutos por mês; pago: 360 minutos por job, sem limite mensal.

Agentic workflows

Edita o .md, compila o .lock.yml, escreve só por safe-outputs.

Tabela de memorização: não confunda

Quando é...	...e quando é
Expressão de compilação ${{ }} Valores conhecidos ao montar o plano, como parâmetros e seleção de stages.	**Expressão de runtime $[ ]** Valores que só existem durante a execução, como resultado de outro job.

succeededOrFailed() Rodar mesmo se algo falhar, mas não quando a execução for cancelada.	**always()** Rodar em qualquer caso, inclusive cancelamento.
lockBehavior: sequential Todas as execuções devem implantar, uma de cada vez, na ordem de chegada.	**lockBehavior: runLatest** Só a mais recente importa; as anteriores são canceladas (padrão).
Agente hospedado pela Microsoft Sem necessidade de rede privada, software licenciado local ou cache persistente.	**Agente auto-hospedado ou Managed DevOps Pools** Software instalado e cache persistente (auto-hospedado) ou rede privada e imagem própria sem cuidar de VMSS (Managed DevOps Pools).
Swap com preview Validar o staging com as configurações de produção antes de concluir a troca.	**Auto swap** Trocar automaticamente logo após cada implantação, sem validação manual.
Template specs Guardar e compartilhar templates ARM versionados com RBAC.	**Deployment stacks** Gerenciar recursos implantados como unidade e impedir alterações com deny settings.
Feature flag com targeting filter Liberar uma funcionalidade para grupos, usuários e percentuais, com desligamento rápido.	**Variant feature flag** Teste A/B com configurações diferentes por variante, alocadas por percentual.

Números que caem

Item	Valor
Peso do domínio no exame	**35-40%**
Armazenamento gratuito do Azure Artifacts	**2 GiB por organização**
Lixeira do Azure Artifacts	**30 dias**
Job hospedado no nível gratuito do Azure Pipelines	**60 minutos por job e 1.800 minutos por mês**
Job hospedado em parallel job pago	**360 minutos**
Reaproveitamento padrão de agente stateful no Managed DevOps Pools	**até sete dias**
actions/cache	**removido após 7 dias sem acesso; 10 GB por repositório por padrão**
Retenção de artefatos e logs do GitHub Actions	**1 a 90 dias em públicos; até 400 dias em privados**
Reusable workflows	**até 10 níveis e 50 workflows únicos por arquivo**
Execuções pendentes com queue: max	**até 100**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Publicar direto na view @Release a partir do pipeline.
Como não cair: Publica-se no feed base (@Local) e depois promove-se para a view.

⚠ **Alerta de prova:** Listar o registro público e o feed juntos no .npmrc.
Como não cair: Várias origens tornam a restauração imprevisível; use um único feed com upstream sources.

⚠ **Alerta de prova:** Excluir o pacote 4.2.0 com defeito e republicar a correção com o mesmo número.
Como não cair: Número publicado não é reutilizado; saia com nova versão.

⚠ **Alerta de prova:** Usar trigger: none para desligar validação de PR.
Como não cair: trigger: none desliga CI de push; validação de PR é pr: none.

⚠ **Alerta de prova:** Agendar cron 0 6 * * * para 6h de Brasília.
Como não cair: O cron é em UTC: 6h em UTC-3 é 0 9 * * *, com always: true para rodar sem mudanças.

⚠ **Alerta de prova:** Usar always() em stage de limpeza que não deve rodar após cancelamento.
Como não cair: always() roda até com cancelamento; use succeededOrFailed().

⚠ **Alerta de prova:** Usar ${{ }} ou parâmetro de template para valor calculado por outro job.
Como não cair: Valor de runtime usa $[ ]; entre jobs, isOutput=true com dependencies e dependsOn.

⚠ **Alerta de prova:** Usar dependsOn no GitHub Actions ou confiar na ordem dos jobs no arquivo.
Como não cair: No Actions é needs; sem needs os jobs rodam em paralelo.

⚠ **Alerta de prova:** Combinar queue: max com cancel-in-progress: true.
Como não cair: A combinação dá erro de validação; queue: max sozinho mantém até 100 pendentes em ordem.

⚠ **Alerta de prova:** Achar que a primeira fase do swap divide tráfego ou copia produção como backup.
Como não cair: Ela aplica as configurações de produção ao staging e pausa para validação.

⚠ **Alerta de prova:** Colocar a connection string no pacote para não trocar no swap.
Como não cair: No pacote ela viaja junto; marque como deployment slot setting.

⚠ **Alerta de prova:** Concluir canary com KubernetesManifest usando scale ou novo deploy com 100%.
Como não cair: O fluxo é deploy com percentage e depois promote ou reject.

⚠ **Alerta de prova:** Ligar Make secrets available to builds of forks para assinatura funcionar em PRs de forks.
Como não cair: Isso expõe segredos a código externo; use GitHub App, segredos bloqueados e comentário de membro antes do build.

⚠ **Alerta de prova:** Fazer commit só do .md do agentic workflow depois de mudar o frontmatter, ou conceder contents: write ao agente.
Como não cair: Rode gh aw compile e faça commit do .md e do .lock.yml; escritas passam por safe-outputs.

Como cai na prova

📝 **Como cai:** Trecho de YAML com uma necessidade (paralelo e depois junta, rodar após falha, passar valor entre jobs).
Como resolver: Identifique a plataforma, depois a palavra-chave exata (dependsOn ou needs, condição, isOutput) e elimine alternativas que quebram o paralelismo ou rodam no momento errado.

📝 **Como cai:** Escolha de agente ou runner com restrições de rede, custo, persistência e manutenção.
Como resolver: Liste as restrições: sem manter máquinas elimina auto-hospedado; rede privada elimina hospedado padrão; cache persistente elimina VM descartável; repositório público elimina auto-hospedado persistente.

📝 **Como cai:** Estratégia de implantação sem downtime, com validação ou exposição gradual.
Como resolver: App Service = slot + swap; VMs = rolling com maxParallel; Kubernetes = canary com increments ou KubernetesManifest promote; público por grupos = feature flag com targeting; descarte parar tudo ou publicar direto em produção.

📝 **Como cai:** Problema de pacote (versão errada, consumidores vendo pré-lançamento, publicação negada).
Como resolver: Verifique views (promover), ordem de upstreams, função da identidade no feed, limite de 2 GiB e se o número da versão já foi usado.

📝 **Como cai:** Agentic workflow com custo, permissão ou mudança que não surtiu efeito.
Como resolver: Mudança no .md exige gh aw compile; escrita vai por safe-outputs; custo com max-ai-credits e gh aw logs/audit; segredo conforme o engine.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
consumidores só veem versões que passaram nos testes	**Promover para a view @Release**
registro público fora do ar e restauração determinística	**Upstream sources com um único feed na configuração**
salvar pacotes do upstream sem publicar	**Feed and Upstream Reader (Collaborator)**
pipeline lê mas não publica no feed	**Feed Publisher (Contributor) para a identidade Build Service**
arquivos genéricos de vários gigabytes versionados	**Universal Packages**
mais hardware e IP estático sem gerenciar máquinas	**Larger runners**
agentes com rede privada e imagem própria sem manter VMSS	**Managed DevOps Pools com injeção de VNet**
atualizar no máximo N VMs por vez	**strategy: rolling com maxParallel**
validar saúde depois de rotear tráfego e reverter se falhar	**postRouteTraffic e on: failure**
prever o que o Bicep vai alterar sem aplicar	**what-if**
obrigar todo pipeline que usa a conexão a estender um template	**Verificação Required template**
reter só execuções de produção por mais tempo	**Retention leases**
funcionários, depois clientes tolerantes, depois todos	**Implantação em anéis (tiers)**
agentic workflow precisa abrir PR sem permissões de escrita	**create-pull-request em safe-outputs**

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

- Cron do Azure Pipelines é em UTC.
- Branch control roda antes das aprovações e falha sem acioná-las.
- Task groups não existem no YAML: use templates.
- Pacote em view é isento da política de retenção.
- Feed com escopo de organização não pode virar feed de projeto.

Essenciais para a prova (checklist)

- ☐ Sei usar views, upstream sources, funções e retenção do Azure Artifacts.

- ☐ Sei publicar e consumir pacotes no GitHub Packages com o GITHUB_TOKEN.
- ☐ Sei aplicar SemVer e o formato de run number com data.
- ☐ Sei os níveis L0 a L4 e como tratar testes instáveis, cobertura e testes de carga.
- ☐ Sei escolher entre agentes hospedados, auto-hospedados, Managed DevOps Pools, larger runners e runner groups.
- ☐ Sei configurar trigger, pr, schedules e resources.pipelines.
- ☐ Sei usar expressões, parâmetros de runtime, variáveis de saída e condições.
- ☐ Sei a ordem dos checks de environment e o efeito de lockBehavior.
- ☐ Sei montar rolling e canary com os hooks certos, swaps de slot e feature flags.
- ☐ Sei usar what-if, template specs, deployment stacks e o fluxo do gh aw.

Questões de revisão do capítulo

R2.1 · Projetar e implementar pipelines de build e lançamento

A Siderúrgica Ferro Novo usa GitHub Enterprise Cloud. Os workflows de build precisam de mais memória e CPU, e o firewall de um parceiro só aceita conexões de endereços IP conhecidos. A empresa não quer administrar máquinas próprias para os runners. Qual opção atende?

A. Runners auto-hospedados em VMs da empresa com IP público fixo
B. Runners padrão hospedados pelo GitHub com cache de dependências
C. Os larger runners hospedados pelo GitHub, com IP estático
D. Actions Runner Controller em um cluster Kubernetes da empresa

R2.2 · Projetar e implementar pipelines de build e lançamento

A equipe da Biblioteca de Componentes Aurora publica um pacote npm com escopo da organização no GitHub Packages sempre que uma release é publicada. O workflow atual falha com erro de autenticação no npm publish, e a segurança proíbe PATs. A equipe revisou o exemplo da documentação e quer saber quais configurações do job são necessárias para publicar com o token do próprio workflow. Qual combinação está correta?

A. packages: write nas permissões, setup-node com registry-url do GitHub Packages e NODE_AUTH_TOKEN com o GITHUB_TOKEN
B. contents: write nas permissões, setup-node sem registry-url e um PAT clássico em NODE_AUTH_TOKEN para o npm
C. packages: read nas permissões, publicação pelo npm login interativo no runner e o GITHUB_TOKEN no package.json
D. id-token: write nas permissões e publicação por OIDC direto no npm, sem setup-node nem NODE_AUTH_TOKEN

R2.3 · Projetar e implementar pipelines de build e lançamento

O feed do Azure Artifacts da Indústria de Autopeças Engrenagem está crescendo rápido, e o dono do feed habilitou retention policies com máximo de 20 versões por pacote e 30 dias para pacotes baixados recentemente. A equipe teme perder versões promovidas para a view @Release e versões ainda usadas. Como a política se comporta?

A. Exclui definitivamente as versões no mesmo instante, sem passar pela Lixeira do feed
B. Exclui as versões mais antigas assim que passam de 20, incluindo as que estão na view @Release
C. Exclui uma versão só quando o máximo é atingido e ela não foi baixada no período; versões em views ficam isentas
D. Exclui todas as versões não baixadas em 30 dias, mesmo que o pacote tenha menos de 20 versões

R2.4 · Projetar e implementar pipelines de build e lançamento

O pipeline multiestágio da Companhia de Logística Rota Norte tem um stage Testes. Depois dele, os stages DeployEUA e DeployBrasil devem rodar em paralelo, e um stage Validacao deve rodar só quando os dois deploys terminarem. Como declarar as dependências?

A. DeployBrasil com dependsOn: DeployEUA e Validacao com dependsOn: DeployBrasil
B. Os dois deploys com dependsOn: Testes e a Validacao com dependsOn listando ambos
C. Todos os stages com dependsOn: [] para rodarem juntos e Validacao com condition: always()
D. Validacao com dependsOn: Testes, e os deploys sem dependsOn para começarem primeiro

Respostas comentadas

R2.1. C) Os larger runners hospedados pelo GitHub, com IP estático

Por quê: Larger runners combinam infraestrutura gerenciada pelo GitHub com recursos extras: hardware maior, IP estático, rede privada do Azure, GPU e grupos de runners.

✗ A) Incorreta. Runners auto-hospedados exigem que a empresa administre as máquinas.
✗ B) Incorreta. Runners padrão não oferecem IP estático nem mais recursos de hardware.
✗ D) Incorreta. O ARC roda runners auto-hospedados no cluster da empresa, que também precisa ser administrado.

> **Dica de prova:** Mais hardware e IP estático sem gerenciar máquinas: larger runners.

R2.2. A) packages: write nas permissões, setup-node com registry-url do GitHub Packages e NODE_AUTH_TOKEN com o GITHUB_TOKEN

Por quê: Para publicar no GitHub Packages pelo Actions: permissão packages: write, setup-node configurando o registro e o GITHUB_TOKEN como credencial.

✗ B) Incorreta. Publicar exige packages: write, e PATs são proibidos e desnecessários.
✗ C) Incorreta. packages: read não publica, e não se coloca token no package.json.
✗ D) Incorreta. O exemplo usa o GITHUB_TOKEN com NODE_AUTH_TOKEN, não OIDC direto no registro.

> **Dica de prova:** npm no GitHub Packages: packages: write + registry-url + NODE_AUTH_TOKEN = GITHUB_TOKEN.

R2.3. C) Exclui uma versão só quando o máximo é atingido e ela não foi baixada no período; versões em views ficam isentas

Por quê: Retention policies do Azure Artifacts combinam máximo de versões e dias desde o último download; views são isentas e o job roda diariamente.

✗ A) Incorreta. As versões são excluídas de forma reversível e movidas para a Lixeira; a limpeza roda uma vez por dia.

✗ B) Incorreta. Pacotes em views são isentos, e o critério de download recente também precisa ser atendido.

✗ D) Incorreta. O limite de versões também precisa ser atingido.

> **Dica de prova:** Retenção no Azure Artifacts: limite de versões E sem download recente; views isentas.

R2.4. B) Os dois deploys com dependsOn: Testes e a Validacao com dependsOn listando ambos

Por quê: Com dependsOn é possível montar fan-out (vários stages após um) e fan-in (um stage após vários); dependsOn: [] remove dependências.

✗ A) Incorreta. Encadear os deploys elimina o paralelismo pedido.

✗ C) Incorreta. dependsOn: [] faria os stages começarem sem esperar os testes, e always() rodaria mesmo com falhas.

✗ D) Incorreta. Validacao rodaria logo após os testes, sem esperar os deploys.

> **Dica de prova:** Paralelo depois junta: fan-out e fan-in com dependsOn.

03

CAPÍTULO 3 · 15-20% DA PROVA

Projetar e implementar processos e comunicações

Aqui eu preciso dominar como o trabalho flui do Azure Boards até o código no GitHub e como os agentes do Copilot entram nesse fluxo. A prova cobra muito a integração Azure Boards x GitHub (AB#, transição de estado, menções com !, links Integrated in build) e o servidor MCP do Azure DevOps (remoto x local, cabeçalhos X-MCP-*). Também caem personalizações do Copilot: instruções, skills, custom agents, hooks e copilot-setup-steps.yml, sempre com o detalhe de local do arquivo e nome exato. Onde mais erro é confundir mecanismos parecidos: service hook x notificação, skill x custom agent x instrução, lead time x cycle time. A regra é ler o cenário procurando a palavra que define o recurso: calendário de várias equipes, critério de saída por coluna, evento para serviço externo.

Neste capítulo

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

Tópicos-chave

Servidor MCP remoto do Azure DevOps

Hospedado pela Microsoft, usa streamable HTTP e autenticação Microsoft Entra (OAuth); não aceita PAT e exige organização ligada a um locatário do Entra (organização só com contas MSA não autentica). É o recomendado para VS Code e Visual Studio. Clientes MCP próprios devem receber escopos OAuth mínimos, e o escopo nunca ultrapassa a permissão do usuário no Azure DevOps.

Servidor MCP local do Azure DevOps

Roda com Node.js 20+, transporte stdio e autenticação por PAT ou Entra. É a saída para clientes que não suportam o fluxo Entra do remoto (como Claude Desktop e Codex) ou para organizações sem Entra. O servidor MCP do Azure DevOps é gratuito; valem só os custos normais do Azure DevOps e da plataforma de IA.

Cabeçalhos X-MCP-* e ferramentas de work item

X-MCP-Toolsets e X-MCP-Tools escolhem quais ferramentas ficam disponíveis; X-MCP-Readonly true bloqueia operações de escrita; X-MCP-Insiders dá acesso antecipado a ferramentas novas, como wit_query_by_wiql. Consultas salvas do Boards são executadas com wit_query na ação get_results; search_workitem é só busca textual. Habilitar toolset não concede nem limita permissões OAuth.

MCP do Azure DevOps no assistente

O assistente precisa estar em agent mode para acessar os dados; no modo Ask responde de forma genérica. O usuário pergunta em linguagem natural e o servidor traduz para consultas como WIQL. Se o assistente repetir dados antigos na mesma conversa, acrescente ao prompt Do not use previously fetched data.

Copilot iniciado pelo Azure Boards

Exige repositório no GitHub (Azure Repos não é suportado), conexão do Boards com o GitHub por GitHub App (não PAT) e assinatura ativa do Copilot. Vão para o Copilot título, campos de texto grandes, comentários e link do item (não anexos nem histórico de revisões); são criados branch e PR em rascunho vinculados. O item mostra status In Progress, Ready for Review ou Error; a operação leva de 5 a 15 minutos, não pode ser cancelada e pode ser refeita com Rerun Copilot.

AB# e transição de estado

AB#ID cria vínculo quando está na mensagem do commit ou na descrição do PR ou da issue; título do PR, comentários, nome da branch e tags não vinculam. Palavras como fix, fixes e fixed antes de AB#ID levam o item ao primeiro estado da categoria Resolved (ou Completed), mas só quando o PR é mesclado na branch padrão.

Recursos da integração Azure Boards x GitHub

Menção com ! cita um PR do GitHub dentro do item; a seção Development mostra os insights do PR (rascunho, revisão e checks); pipelines YAML do Azure Pipelines que usam repositório GitHub criam links Integrated in build automaticamente. O Azure Boards app autentica como app e suporta até 2.000 repositórios por conexão; cada repositório deve ficar ligado a uma única organização do Azure DevOps para evitar vínculos AB# inesperados.

Processos do Azure Boards

Basic usa Epic, Issue e Task; Agile usa User Story, Bug, Feature e Epic; Scrum usa Product Backlog Item e Impediment; CMMI inclui Requirement, Change Request, Risk e Review. Mudança formal, risco, revisão e trilha auditável apontam para CMMI.

Tipos de vínculo

Tested by (da história para o caso de teste) expressa verificação e alimenta widgets como Requirements quality. Parent/Child organiza hierarquia; Predecessor/Successor representa dependência de agenda e é o que aparece no Delivery Plans; Related é genérico.

Delivery Plans

Reúne backlogs de várias equipes, até de projetos diferentes da organização, em uma linha do tempo por iteração, sem mudar o modo como cada equipe planeja. Mostra progresso de Features, dependências entre itens ligados por Predecessor/Successor e permite ajustar datas arrastando os itens.

Configuração do quadro

Limite WIP é restrição flexível: deixa exceder e só destaca a contagem da coluna. Colunas divididas (Doing/Done) mostram o que terminou uma etapa e espera a próxima, apoiando o mecanismo de puxar. Definition of Done por coluna deixa visíveis os critérios de saída; swimlanes e estilos de cartão só organizam ou destacam.

Bugs no backlog

Cada equipe escolhe tratar bugs como requisitos (backlog com priorização), como tarefas (Taskboard, para quem registra horas) ou fora dos backlogs. Bugs como tarefas fazem a equipe perder Planning, velocity, forecast, rollup e Delivery Plans, mas continuam em consultas e podem ser vinculados a commits e PRs.

Service hooks e notificações

Service hooks publicam eventos do Azure DevOps em serviços externos como Microsoft Teams, Slack, Jenkins ou Web Hooks (JSON para endpoint HTTPS) e exigem Project Administrator ou Project Collection Administrator. Assinaturas de notificação por e-mail podem ser de equipe, projeto ou organização, entregando para Members of Team, por função ou endereço personalizado.

Webhooks do GitHub com segurança

Configure um segredo no webhook; o GitHub assina cada entrega com HMAC SHA-256 no cabeçalho X-Hub-Signature-256. O receptor recalcula a assinatura e compara com função de tempo constante; X-GitHub-Delivery e User-Agent não provam origem.

Documentação como código

O wiki do Azure DevOps aceita diagramas Mermaid, sumário, seções recolhíveis e consultas incorporadas. Publish code as wiki publica uma pasta do repositório, revisada por PR e versionada por branch. Licença fica em LICENSE na raiz; notas de versão automáticas são organizadas por rótulo com .github/release.yml (categories e exclude.authors).

Instruções personalizadas do Copilot

Repositório: geral em .github/copilot-instructions.md (o cloud agent pode gerá-lo pela aba de agentes) e por caminho em .github/instructions/*.instructions.md com applyTo em glob; AGENTS.md é outro tipo voltado a agentes. Instruções de organização exigem Business ou Enterprise, valem para todos os membros e se aplicam ao Copilot Chat, code review e cloud agent no GitHub.com, não às IDEs.

Skills, custom agents e hooks

Skill: pasta em .github/skills (ou .claude/skills, .agents/skills) com SKILL.md; name (minúsculas e hífens) e description obrigatórios, license e allowed-tools opcionais; carregada sob demanda; skills pessoais em ~/.copilot/skills não chegam ao cloud agent. Custom agent: arquivo .agent.md em .github/agents (ou no repositório .github/.github-private da organização), com tools, mcp-servers, model (IDEs) e target; prompt de até 30.000 caracteres. Hooks: JSON em .github/hooks, na branch padrão para valer no cloud agent, com version 1, chaves bash e powershell e timeoutSec padrão de 30 segundos.

Ambiente e MCP do Copilot cloud agent

copilot-setup-steps.yml na branch padrão com um único job copilot-setup-steps preinstala ferramentas; só steps, permissions, runs-on, services, snapshot e timeout-minutes (até 59) são respeitados. O MCP do GitHub e o do Playwright vêm habilitados; o do GitHub usa token só de leitura do repositório atual (PAT ou GitHub App para ir além), segredos precisam do prefixo COPILOT_MCP_ e as ferramentas são usadas sem pedir aprovação. O firewall vem ligado, mas não cobre servidores MCP nem setup steps.

Macetes para memorizar

💡 Processos do Boards

Basic = Issue, Agile = User Story, Scrum = PBI, CMMI = Change Request + Risk + Review.

💡 Onde AB# funciona

CDD: Commit, Descrição do PR, Descrição da issue. Título e comentário não contam; estado só muda na branch padrão.

💡 Símbolos de menção

AB# no GitHub aponta para o Boards; ! no Boards aponta para o PR; # no Boards é item; @ é pessoa.

💡 MCP do Azure DevOps

Remoto = HTTP + Entra, sem PAT. Local = Node 20 + stdio + PAT ou Entra.

💡 Arquivos do Copilot

instructions orientam sempre, skills carregam sob demanda, agents definem papel, hooks disparam em eventos, setup-steps prepara a máquina.

💡 copilot-setup-steps

Branch padrão, job único com o mesmo nome, timeout até 59.

Lead x Cycle

Lead começa quando o cliente pede (criação); Cycle começa quando a equipe pega (In Progress).

Tabela de memorização: não confunda

Quando é...	...e quando é
MCP remoto do Azure DevOps Cliente suporta o fluxo Entra (VS Code, Visual Studio), organização ligada ao Entra e política que proíbe instalar componentes; streamable HTTP, sem PAT.	**MCP local do Azure DevOps** Cliente sem suporte ao fluxo Entra do remoto (Claude Desktop, Codex) ou organização só com MSA; Node.js 20+, stdio, PAT ou Entra.
Lead time Medir do registro do pedido (criação do item) até a entrega: visão do cliente.	**Cycle time** Medir do primeiro In Progress até a conclusão: eficiência da equipe, ignora a espera antes do início.
Service hook Enviar o evento para outro sistema (Teams, Slack, endpoint HTTPS com JSON).	**Assinatura de notificação** Avisar pessoas por e-mail, por exemplo todos os membros da equipe com Members of Team.
Agent skill (SKILL.md) Procedimento reutilizável com instruções e scripts carregado só quando a tarefa é daquele tipo.	**Custom agent (.agent.md)** Papel especializado no seletor de agentes, com tools restritas (por exemplo só leitura e busca) e mcp-servers próprios.
.github/copilot-instructions.md Regras gerais que valem para todo o repositório.	**.github/instructions/*.instructions.md com applyTo** Regras só para uma parte do repositório, como arquivos Terraform da pasta infra ou testes em tests/.
Delivery Plans Calendário interativo de várias equipes com dependências e ajuste de datas arrastando.	**Sprint Burndown ou gráfico de consulta** Trabalho restante de uma sprint de uma equipe ou contagens e tendências, sem dependências entre equipes.
Colunas divididas (Doing/Done) Mostrar o que terminou a etapa e aguarda ser puxado pela próxima.	**Limite WIP** Sinalizar excesso de itens em andamento numa coluna, sem bloquear o movimento.

Números que caem

Item	Valor
Peso do domínio no exame	**15-20%**
Node.js mínimo para o MCP local do Azure DevOps	**20+**
Prompt máximo de um custom agent	**30.000 caracteres**

timeoutSec padrão de hooks do Copilot	**30 segundos**
timeout-minutes máximo no copilot-setup-steps	**59**
Repositórios por conexão do Azure Boards app	**até 2.000**
Duração típica do Copilot iniciado pelo Boards	**5 a 15 minutos**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Mandar um PAT no cabeçalho do servidor MCP remoto do Azure DevOps para clientes sem Entra.
Como não cair: O remoto não aceita PAT; PAT é opção do servidor local.

⚠ **Alerta de prova:** Achar que escrever Fixes AB#123 em PR mesclado em release/x muda o estado do item.
Como não cair: O vínculo aparece, mas a transição de estado só ocorre no merge para a branch padrão.

⚠ **Alerta de prova:** Colocar AB# no título do PR, em comentários, no nome da branch ou em tag.
Como não cair: Só mensagem de commit e descrição do PR ou da issue criam o vínculo.

⚠ **Alerta de prova:** Usar # para citar um PR do GitHub dentro do item do Boards.
Como não cair: No Boards, # referencia itens de trabalho; PR do GitHub é citado com !. AB# é usado no lado do GitHub.

⚠ **Alerta de prova:** Supor que o limite WIP bloqueia o cartão ou exige aprovação.
Como não cair: WIP no Azure Boards é soft constraint: só destaca a contagem.

⚠ **Alerta de prova:** Criar o copilot-setup-steps.yml numa branch de feature com job chamado setup.
Como não cair: Precisa estar na branch padrão e o job deve se chamar exatamente copilot-setup-steps.

⚠ **Alerta de prova:** Usar hook sessionEnd para preparar o ambiente ou para rodar lint após cada ferramenta.
Como não cair: sessionEnd roda no fim da sessão; preparação é copilot-setup-steps e lint após ferramenta é postToolUse.

⚠ **Alerta de prova:** Esperar que a skill pessoal em ~/.copilot/skills seja usada pelo cloud agent.
Como não cair: Skills pessoais ficam na máquina; para o cloud agent, a skill deve estar no repositório.

⚠ **Alerta de prova:** Achar que instruções de organização valem nas IDEs ou no Copilot CLI, ou só para quem recebe licença pela organização.
Como não cair: Valem no Copilot Chat, code review e cloud agent do GitHub.com, para todos os membros.

⚠ **Alerta de prova:** Criar um arquivo MCP separado para o Copilot code review ou esperar que ele use resources e prompts.
Como não cair: A configuração MCP do repositório é compartilhada com o cloud agent; o code review só usa ferramentas com annotations.readOnlyHint true.

⚠ **Alerta de prova:** Configurar o MCP remoto do Azure DevOps com OAuth para o cloud agent ou guardar PAT Full access no .vscode/mcp.json.
Como não cair: O cloud agent não suporta MCP remoto com OAuth; use azure/login com OIDC no copilot-setup-steps, segredos de Agents e o servidor ado.

⚠ **Alerta de prova:** Achar que a operação do Copilot iniciada pelo item pode ser cancelada ou que mudar o estado do item a reinicia.
Como não cair: Não há cancelamento; trate o PR gerado e use Rerun Copilot.

⚠ **Alerta de prova:** Validar a origem do webhook do GitHub pelo User-Agent GitHub-Hookshot ou pelo X-GitHub-Delivery.
Como não cair: Só a assinatura HMAC SHA-256 em X-Hub-Signature-256, comparada em tempo constante, prova origem e integridade.

⚠ **Alerta de prova:** Usar Web Hooks com URL HTTP para evitar erro de certificado.
Como não cair: HTTP expõe dados e cabeçalhos de autenticação; a recomendação é HTTPS.

Como cai na prova

📝 **Como cai:** Cenário de integração Boards x GitHub em que o vínculo não aparece ou o estado não muda.
Como resolver: Verifique onde o AB# foi escrito (commit, descrição de PR ou issue) e em qual branch o PR foi mesclado (só a padrão transiciona estado); descarte respostas sobre iteração ou tipo de autenticação.

📝 **Como cai:** Escolher entre MCP remoto e local para grupos com clientes diferentes.
Como resolver: Separe clientes que suportam o fluxo Entra (remoto) dos que não suportam (local); elimine qualquer alternativa com PAT no remoto ou remoto com stdio.

📝 **Como cai:** Onde colocar uma personalização do Copilot (regra, procedimento, papel, automação de evento).
Como resolver: Identifique se é regra sempre válida (instructions), procedimento sob demanda (skill), papel com ferramentas próprias (custom agent) ou ação em evento (hook); confira local e nome exatos do arquivo e se precisa estar na branch padrão.

📝 **Como cai:** Pedido de visão gerencial (calendário, KPI, Power BI, gargalo).
Como resolver: Mapeie a palavra-chave: várias equipes e dependências = Delivery Plans; do pedido à entrega = Lead Time; Power BI = Analytics views; critério de saída = Definition of Done.

📝 **Como cai:** Integração por evento com sistema externo ou com a equipe.
Como resolver: Sistema externo recebe evento = service hook (HTTPS); pessoas recebem e-mail = assinatura de notificação de equipe; webhook do GitHub = validar X-Hub-Signature-256.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
visão de calendário de várias equipes com dependências	**Delivery Plans com links Predecessor/Successor**
mudança formal, risco, revisão, trilha auditável ao criar projeto	**Processo CMMI**
requisito verificado por caso de teste	**Vínculo Tested by**
critérios de saída visíveis em cada coluna do quadro	**Definition of Done da coluna**
ver o que terminou a etapa e aguarda a próxima	**Colunas divididas Doing/Done**
do registro do pedido até a entrega	**Widget Lead Time**
Power BI sem escrever OData à mão	**Analytics views**
novo item ou build vira mensagem no Teams ou JSON para serviço externo	**Service hooks (Teams ou Web Hooks com HTTPS)**
Claude Desktop, Codex ou organização com contas MSA usando MCP do Azure DevOps	**MCP local (Node.js 20+, stdio, PAT ou Entra) ou ligar a organização ao Entra**
assistente responde genérico no modo Ask	**MCP exige agent mode**
diagrama versionado como texto no wiki	**Mermaid**
documentação na pasta docs revisada por PR e versionada por branch	**Publish code as wiki**

agente que lê e pesquisa, mas nunca edita	**Custom agent .agent.md com tools restritas**
do Application Insights para o backlog com telemetria	**Work item integration com modelos de workbook (permissão de escrita em workbooks)**

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

- MCP remoto do Azure DevOps não aceita PAT e exige organização ligada ao Entra.
- AB# transiciona estado só com merge na branch padrão.
- Integração Copilot pelo Boards não funciona com Azure Repos e exige conexão por GitHub App.
- O cloud agent usa ferramentas MCP sem pedir aprovação: libere só ferramentas de leitura.
- Hooks do Copilot: JSON em .github/hooks na branch padrão, com bash e powershell; timeoutSec padrão de 30 segundos.

Essenciais para a prova (checklist)

- ☐ Sei escolher entre MCP remoto e local do Azure DevOps e o que cada cabeçalho X-MCP-* faz.
- ☐ Sei onde AB# cria vínculo e quando o estado do item muda.
- ☐ Sei os requisitos e o comportamento do Copilot iniciado pelo Azure Boards.
- ☐ Sei associar cada processo (Basic, Agile, Scrum, CMMI) aos tipos de item.
- ☐ Sei diferenciar Delivery Plans, WIP limit, colunas divididas e Definition of Done.
- ☐ Sei a diferença entre lead time e cycle time.
- ☐ Sei quando usar service hook, assinatura de notificação e webhook validado por HMAC.
- ☐ Sei onde ficam instructions, skills, custom agents, hooks e copilot-setup-steps.yml e suas regras.
- ☐ Sei como o MCP funciona no cloud agent e no code review (padrões, segredos COPILOT_MCP_, readOnlyHint).

Questões de revisão do capítulo

R3.1 · Projetar e implementar processos e comunicações

A Seguros Horizonte vai criar um projeto no Azure DevOps para um sistema regulado. A área de conformidade exige registrar solicitações formais de mudança, riscos e revisões, com histórico auditável das decisões. Qual processo deve ser escolhido ao criar o projeto?

A. O processo CMMI, que inclui Requirement, Change Request, Risk e Review
B. O processo Scrum, que inclui Product Backlog Item e Impediment
C. O processo Basic, com os tipos Epic, Issue e Task
D. O processo Agile, que inclui User Story, Bug, Feature e Epic

R3.2 · Projetar e implementar processos e comunicações

A Fundação Saber Livre vai abrir no GitHub o código de uma biblioteca e quer que a licença MIT seja exibida corretamente na página do repositório. Onde colocar o texto da licença?

A. Em um arquivo LICENSE (ou LICENSE.md) na raiz do repositório
B. Em um comentário no topo de cada arquivo de código
C. Na descrição curta do repositório, em Settings
D. Em uma página do wiki chamada Licença do projeto

R3.3 · Projetar e implementar processos e comunicações

Na Corretora de Seguros Prumo, uma desenvolvedora iniciou o GitHub Copilot a partir de um item de trabalho do Azure Boards e, dez minutos depois, percebeu que as instruções estavam incompletas. Ela quer interromper a operação e rodar de novo com instruções melhores. O que é correto sobre esse fluxo?

A. A operação não pode ser cancelada; ela trata o PR gerado no GitHub e usa Rerun Copilot com novas instruções
B. A operação pode ser cancelada pelo botão Stop do item, que apaga a branch e o PR criados automaticamente no GitHub
C. A operação só termina depois de 24 horas, quando o status passa sozinho para Error e pode ser refeita
D. A operação é cancelada quando o item volta para New, e o Copilot recomeça na próxima mudança de estado

R3.4 · Projetar e implementar processos e comunicações

Um desenvolvedor da Startup Rota Verde criou uma skill pessoal em ~/.copilot/skills para gerar notas de versão e espera que o Copilot cloud agent a use quando trabalhar nos PRs do repositório no GitHub. O agente não usa a skill. Qual é a explicação mais provável?

A. Skills precisam ser compiladas com gh aw compile antes de o agente usá-las
B. Skills exigem o campo license no frontmatter para serem carregadas pelo agente na nuvem
C. Skills só funcionam no Copilot code review e nunca são lidas pelo cloud agent
D. Skills pessoais ficam na máquina; para o cloud agent, a skill deve estar no repositório

Respostas comentadas

R3.1. A) O processo CMMI, que inclui Requirement, Change Request, Risk e Review

Por quê: Na escolha do processo do Azure Boards, o CMMI é o indicado para gestão formal de mudanças e trilha auditável, porque inclui os tipos Requirement, Change Request, Risk e Review.

✗ B) Incorreta. O Scrum atende equipes que seguem Scrum com Product Backlog Items e Bugs; não oferece os tipos de gestão formal de mudança pedidos.

✗ C) Incorreta. O Basic é o processo mais simples (Epic, Issue e Task) e não oferece os tipos de controle formal exigidos pela conformidade.

✗ D) Incorreta. O Agile atende equipes que usam métodos ágeis e acompanham User Stories, mas não traz os tipos Change Request, Risk e Review.

> **Dica de prova:** Mudança formal, risco, revisão e auditoria apontam para CMMI; User Story para Agile; PBI para Scrum; Issue para Basic.

R3.2. A) Em um arquivo LICENSE (ou LICENSE.md) na raiz do repositório

Por quê: Sem licença, valem as leis de direitos autorais padrão; o GitHub recomenda incluir o arquivo de licença na raiz do repositório.

✗ B) Incorreta. Cabeçalhos em cada arquivo não substituem o arquivo de licença na raiz.

✗ C) Incorreta. A descrição do repositório não é o local do texto da licença.

✗ D) Incorreta. Uma página de wiki não é o arquivo de licença do repositório.

> **Dica de prova:** Licença do repositório: arquivo LICENSE na raiz.

R3.3. A) A operação não pode ser cancelada; ela trata o PR gerado no GitHub e usa Rerun Copilot com novas instruções

Por quê: O item mostra o status do Copilot (In Progress, Ready for Review, Error); a operação não é cancelável e pode ser refeita com Rerun Copilot.

✗ B) Incorreta. Não há cancelamento da operação pelo item de trabalho.

✗ C) Incorreta. A operação costuma levar de 5 a 15 minutos; não há espera de 24 horas.

✗ D) Incorreta. Mudar o estado do item não cancela nem reinicia a operação.

> **Dica de prova:** Copilot iniciado pelo Azure Boards: não cancela; trate o PR e use Rerun Copilot.

R3.4. D) Skills pessoais ficam na máquina; para o cloud agent, a skill deve estar no repositório

Por quê: Skills de projeto vivem no repositório; skills pessoais vivem no diretório do usuário e não estão disponíveis para agentes na nuvem.

✗ A) Incorreta. gh aw compile é para GitHub Agentic Workflows, não para skills.

✗ B) Incorreta. O campo license é opcional; name e description são os obrigatórios.

✗ C) Incorreta. Skills funcionam no cloud agent, no code review, no CLI, no Copilot app e no modo agente das IDEs.

> **Dica de prova:** Skill para o cloud agent: coloque no repositório, não no ~/.copilot.

04

CAPÍTULO 4 · 10-15% DA PROVA

Projetar e implementar um plano de segurança e conformidade

Neste domínio eu preciso eliminar segredos de longa duração: identidade gerenciada, workload identity federation e OIDC são quase sempre a resposta quando o cenário pede autenticação sem segredo. A prova cobra o subject exato da credencial federada, o que id-token: write realmente faz, como funcionam segredos no GitHub Actions e no Azure Pipelines e a integração com Key Vault. Também caem GitHub Advanced Security, GitHub Advanced Security for Azure DevOps, Dependabot e Defender for Cloud com suas permissões. Onde mais erro é confundir o que cada varredura cobre (código, dependências, segredos) e qual produto inclui o quê. Sempre escolho o menor privilégio que resolve.

Neste capítulo

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

Tópicos-chave

Identidade gerenciada

Código em recurso do Azure acessando outro serviço do Azure usa identidade gerenciada com papel RBAC, e a rotação de credenciais fica com o Azure. A atribuída pelo sistema nasce e morre com o recurso; a atribuída pelo usuário tem ciclo de vida independente e pode ser compartilhada por várias VMs recriadas com frequência.

Objeto de aplicativo x entidade de serviço

Um app multi-locatário tem um objeto de aplicativo só no locatário de origem (modelo global) e uma entidade de serviço em cada locatário que o usa (instância local). São objetos diferentes e não dependem de identidade gerenciada.

Workload identity federation no Azure Pipelines

Service connection do Azure Resource Manager com WIF elimina segredos que expiram. Sem permissão para criar entidades de serviço, use identidade gerenciada atribuída pelo usuário com credencial federada; conexões antigas devem ser convertidas para o emissor do Microsoft Entra antes de 1º de julho de 2027.

OIDC no GitHub Actions

Credencial federada no app do Entra ou na identidade gerenciada, azure/login por OIDC e client ID, tenant ID e subscription ID no repositório. id-token: write só permite solicitar o token OIDC, sem dar escrita em recursos, e é obrigatório para o login. O subject é repo:ORG/REPO:environment:NOME quando o job usa environment, repo:ORG/REPO:pull_request para pull_request e repo:ORG/REPO:ref:refs/heads/BRANCH nos demais casos.

GitHub Apps

Recomendadas para integrações duradouras no lugar de PAT de pessoa, com permissões granulares. Para agir por conta própria nos repositórios, o app usa installation access token; o JWT serve para gerar tokens de instalação e gerenciar o app; user access token atribui a atividade a uma pessoa.

GITHUB_TOKEN

Defina o padrão restrito (leitura) na organização e libere escritas pontuais com permissions no workflow. Vale só para o job e o repositório do workflow; em runner auto-hospedado não passa de 24 horas, e runners hospedados têm job de no máximo 6 horas.

PATs

No GitHub, fine-grained PAT limita repositórios e permissões e permite que owners exijam aprovação; o clássico usa escopos amplos. No Azure DevOps, PATs globais foram aposentados: migre para tokens do Entra ou PATs de organização com escopos mínimos e use políticas de locatário para restringir PATs globais, de escopo total e a validade máxima.

Segredos no GitHub Actions

Segredo de environment prevalece sobre os de repositório e organização com o mesmo nome; limite de 48 KB; forks (exceto GITHUB_TOKEN) e eventos do Dependabot não recebem segredos; reusable workflows só recebem se forem repassados. Não use JSON ou YAML como segredo, registre valores gerados com ::add-mask:: e, se algo vazar no log, exclua o log e rotacione.

Variáveis secretas no Azure Pipelines

Ficam criptografadas e não são exibidas depois de definidas; não viram variáveis de ambiente sozinhas e precisam ser mapeadas; builds de PRs de forks do GitHub não as recebem por padrão. Dados estruturados não são mascarados campo a campo.

Key Vault nos pipelines

Grupo de variáveis ligado ao Key Vault busca os valores a cada execução, mas segredos novos precisam ser adicionados ao grupo; só segredos são mapeados e cofres RBAC funcionam com endpoint público. AzureKeyVault@2 com RunAsPreJob true disponibiliza os segredos para todas as tarefas do job; a identidade da service connection precisa de Key Vault Secrets User. No GitHub Actions: OIDC, az keyvault secret show e add-mask no valor lido.

Proteção e rotação no Key Vault

Purge protection, que exige soft delete, impede eliminar definitivamente antes do fim da retenção. Rotação automática: o evento near expiry, 30 dias antes do vencimento, via Event Grid aciona uma Function que gera e aplica a nova senha.

Secure files e recursos protegidos

Keystore, certificados e chaves SSH ficam em Secure files, com permissões e checks, e são baixados com DownloadSecureFile@1. Recursos protegidos não devem ter Open access: autorize só pipelines confiáveis. Limit job authorization scope to current project for non-release pipelines restringe o token do job ao próprio projeto.

Níveis de acesso e grupos do Azure DevOps

Stakeholder acompanha itens e painéis e pode ver e aprovar releases, mas não acessa código; Basic acessa repositórios; Basic + Test Plans cria planos de teste manuais. Grupos padrão por menor privilégio: Readers, Contributors, Build Administrators e Project Administrators. Criar projeto exige Project Collection Administrators ou Create new projects; nova equipe com backlog próprio recebe caminho de área e iterações.

Permissões e gatilhos arriscados no GitHub

Base permission Read dá leitura a membros sem atingir colaboradores externos. pull_request_target roda o workflow da branch padrão do repositório base com segredos e token de escrita; o risco é fazer checkout e executar o código do fork. Para ações de terceiros, a recomendação mais forte é lockfile de dependências de workflow e, sem ele, fixar pelo SHA completo.

GitHub Advanced Security no GitHub

GitHub Secret Protection inclui secret scanning e push protection; GitHub Code Security inclui code scanning e dependency review. Bypass de push protection aceita o push e gera alerta para revisão. Default setup do CodeQL roda em pushes e PRs para a branch padrão ou protegida (exceto PRs de forks) e semanalmente; outras branches, linguagens ou build próprio pedem configuração avançada; Require code scanning results no ruleset bloqueia merge por severidade.

Dependabot, autofix e dependency review

Alertas avisam sobre vulnerabilidades sem bloquear PRs, security updates abrem PRs para corrigir dependências com alerta e version updates (dependabot.yml) abrem PRs para manter dependências atualizadas mesmo sem vulnerabilidade. A dependency review action em workflow de PR falha quando o PR adiciona dependência vulnerável. Copilot Autofix dá sugestão pontual; agentic autofix investiga, corrige, valida e abre PR em rascunho.

GitHub Advanced Security for Azure DevOps

Habilitado como GitHub Code Security for Azure DevOps (code scanning com CodeQL e dependency scanning, que exige a tarefa AdvancedSecurity-Dependency-Scanning@1 ou a configuração padrão) e GitHub Secret Protection for Azure DevOps (secret scanning e push protection). A varredura inicial de segredos cobre todo o histórico; credenciais em par só geram alerta com as duas partes no arquivo. Contributor no repositório vê o resumo de alertas e dispensar alertas exige Project Administrator; cobrança por committer ativo em 90 dias, contado uma vez por assinatura. Na detecção de segredos, substituiu o CredScan do MSDO; a configuração padrão do code scanning pode usar pool auto-hospedado e exige tarefas do Marketplace liberadas.

Defender for Cloud DevOps security

Conector Azure DevOps: conta com Project Collection Administrator e acesso Basic (Stakeholder não serve), OAuth de aplicativos de terceiros ligado e Contributor na assinatura. Conector GitHub: Organization Owner para autorizar e Contributor para criar; planos Free, Pro, Team e Enterprise Cloud; Security Reader basta para ler a postura. Microsoft Security DevOps (MSDO) roda só a categoria IaC quando pedido; anotações em PR exigem MSDO e, no GitHub, GHAS.

Macetes para memorizar

Sem segredos

Azure para Azure = identidade gerenciada; Pipelines para Azure = WIF; Actions para Azure = OIDC.

Subject OIDC

Tem environment? environment ganha. Não tem e é PR? pull_request. Senão, ref da branch.

Varreduras

CodeQL = código; dependency = bibliotecas; secret = credenciais; push protection = segredo barrado no push.

Produtos GHAS

Secret Protection guarda segredos; Code Security cuida de código e dependências.

Dependabot

Alerta avisa, security update corrige, version update atualiza.

Key Vault

Soft delete recupera; purge protection tranca.

Segredos no Actions

Environment vence; fork e Dependabot não recebem; 48 KB; nada de JSON.

Tabela de memorização: não confunda

Quando é...	...e quando é

Identidade gerenciada atribuída pelo sistema Um único recurso com ciclo de vida próprio.	**Identidade gerenciada atribuída pelo usuário** Várias VMs recriadas com frequência com o mesmo acesso, sem refazer RBAC.
Fine-grained PAT Acesso pessoal restrito a repositórios e permissões específicas.	**GitHub App** Integração duradoura da organização, independente de pessoas.
Copilot Autofix Sugestão pontual de correção para o alerta.	**Agentic autofix** Investigar, gerar correção, validar e abrir PR em rascunho.
GitHub Secret Protection Secret scanning e push protection.	**GitHub Code Security** Code scanning e dependency review.
Dependabot security updates Abrir PRs que corrigem dependências já com alerta.	**Dependency review action** Fazer o check do PR falhar quando ele adiciona dependência vulnerável.
Soft delete Permitir recuperar objeto excluído durante a retenção.	**Purge protection** Impedir que alguém elimine definitivamente antes do fim da retenção.
Stakeholder Acompanhar itens e painéis e aprovar releases sem licença paga.	**Basic** Quem precisa acessar código e PRs.

Números que caem

Item	Valor
Peso do domínio no exame	**15-20%**
Tamanho máximo de segredo no GitHub Actions	**48 KB**
Validade máxima do GITHUB_TOKEN em runner auto-hospedado	**24 horas**
Tempo máximo de job em runner hospedado pelo GitHub	**6 horas**
Evento near expiry do Key Vault	**30 dias antes do vencimento**
Janela de committers ativos no GitHub Advanced Security for Azure DevOps	**90 dias**
Prazo para converter service connections WIF para o emissor do Entra	**antes de 1º de julho de 2027**

Alertas de prova (pegadinhas)

> ⚠ **Alerta de prova:** Achar que id-token: write dá escrita no Azure ou no repositório.
> **Como não cair:** Só permite pedir o token OIDC; o acesso vem dos papéis da identidade federada.

⚠ **Alerta de prova:** Usar subject de branch ou environment para job de pull_request sem environment.
Como não cair: Nesse caso o subject é repo:ORG/REPO:pull_request; com environment no job, vira repo:ORG/REPO:environment:NOME.

⚠ **Alerta de prova:** Guardar a chave da conta de armazenamento no Key Vault para cumprir a regra de sem segredos.
Como não cair: A chave ainda é segredo compartilhado; use identidade gerenciada com RBAC.

⚠ **Alerta de prova:** Usar identidade gerenciada do agente hospedado pela Microsoft para o pipeline.
Como não cair: Agentes hospedados não têm identidade da sua assinatura; use service connection com WIF.

⚠ **Alerta de prova:** Guardar várias credenciais num único segredo em JSON.
Como não cair: Segredo estruturado reduz a chance de redação correta; crie um segredo por valor.

⚠ **Alerta de prova:** Esperar que um segredo novo no cofre apareça sozinho no grupo de variáveis.
Como não cair: Valores atualizam sozinhos, mas segredos novos precisam ser adicionados ao grupo.

⚠ **Alerta de prova:** Confiar em soft delete para impedir purge.
Como não cair: Só purge protection impede eliminação definitiva antes da retenção.

⚠ **Alerta de prova:** Fixar ações de terceiros pela tag de versão maior achando que é imutável.
Como não cair: Tags podem ser movidas; use lockfile ou SHA completo.

⚠ **Alerta de prova:** Achar que no pull_request_target o checkout traz o código do fork por padrão e o token é só leitura.
Como não cair: Sem ref, o checkout usa a branch padrão do base e o token é de leitura e escrita; o perigo é executar o código do fork.

⚠ **Alerta de prova:** Ligar push protection para achar SQL injection ou dependências vulneráveis.
Como não cair: Push protection só procura segredos; código é CodeQL e dependências é dependency scanning.

⚠ **Alerta de prova:** Habilitar o CredScan do MSDO ou adicionar Trivy para detectar segredos.
Como não cair: CredScan foi descontinuado; segredos no Azure Repos são do GitHub Advanced Security for Azure DevOps; Trivy é para contêiner e IaC.

⚠ **Alerta de prova:** Criar dependabot.yml no Azure Repos ou esperar varredura semanal automática de dependências.
Como não cair: Falta a tarefa AdvancedSecurity-Dependency-Scanning@1 ou a configuração padrão.

⚠ **Alerta de prova:** Dar Owner na assinatura ou Key Vault Administrator para ler segredos.
Como não cair: Key Vault Secrets User para a identidade da service connection.

⚠ **Alerta de prova:** Achar que Stakeholder lê código ou não pode aprovar releases.
Como não cair: Stakeholder não acessa repositórios, mas pode ver e aprovar pipelines de release.

Como cai na prova

📝 **Como cai:** Escolher a forma de autenticar uma automação no Azure ou no GitHub com requisitos de segurança.
Como resolver: Elimine primeiro tudo que guarda segredo de longa duração (PAT, client secret, chave); depois escolha entre identidade gerenciada, WIF, OIDC ou GitHub App conforme a origem e o destino.

📝 **Como cai:** Login OIDC falha depois de mudar o workflow (environment, PR, branch).
Como resolver: Monte o subject a partir do contexto do job (environment, pull_request ou ref) e recrie a credencial federada com ele; confirme id-token: write.

📝 **Como cai:** Cenário de varredura de segurança pedindo o recurso certo.
Como resolver: Classifique o alvo (código, dependências, segredos, IaC) e a plataforma (GitHub, Azure Repos, Defender) e escolha a ferramenta; descarte as que procuram outro tipo de problema.

📝 **Como cai:** Mapeamento de permissões e níveis de acesso por perfil.
Como resolver: Para cada perfil, escolha o menor grupo ou nível que cumpre a tarefa: leitura = Readers, código = Contributors com Basic, build = Build Administrators, testes manuais = Basic + Test Plans.

📝 **Como cai:** Segredo do Key Vault que não aparece ou não chega a tempo no pipeline.
Como resolver: Verifique se o segredo foi adicionado ao grupo, se a identidade tem Key Vault Secrets User e se a tarefa precisa de RunAsPreJob.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
sem segredos e rotação responsabilidade do Azure	**Identidade gerenciada**

service connection com segredo que expira	**Azure Resource Manager com workload identity federation**
implantar no Azure pelo GitHub Actions sem segredos	**OIDC com azure/login e credencial federada**
integração com o GitHub usa PAT de pessoa que vai sair	**GitHub App instalada na organização**
keystore ou certificado fora do repositório	**Secure files + DownloadSecureFile@1**
segredo do Key Vault necessário antes da tarefa no mesmo job	**AzureKeyVault@2 com RunAsPreJob true**
pipeline com token que acessa todos os projetos	**Limit job authorization scope to current project for non-release pipelines**
service connection com Open access	**Remover Open access e autorizar pipelines específicos**
bloquear merge por alertas de alta severidade do CodeQL	**Ruleset Require code scanning results**
barrar PR que adiciona dependência vulnerável	**Dependency review action**
CodeQL em Go, branches de release ou build próprio	**Code scanning em configuração avançada**
custo do GitHub Advanced Security for Azure DevOps	**Committers ativos em 90 dias, deduplicados por assinatura**
senha vence e deve ser trocada sem scripts manuais	**Near expiry do Key Vault + Event Grid + Function**
gestores só acompanham e analistas criam planos de teste	**Stakeholder e Basic + Test Plans**

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

- id-token: write só permite pedir o token OIDC.
- Forks e eventos do Dependabot não recebem segredos no GitHub Actions.
- Grupo ligado ao Key Vault atualiza valores sozinho, mas não inclui segredos novos.
- Push protection procura segredos; não analisa código nem dependências.
- Stakeholder não acessa código.

Essenciais para a prova (checklist)

- ☐ Sei quando usar identidade gerenciada do sistema ou do usuário.
- ☐ Sei configurar WIF no Azure Pipelines e OIDC no GitHub Actions, incluindo o subject.

- ☐ Sei quando usar GitHub App, fine-grained PAT e GITHUB_TOKEN.
- ☐ Sei as regras de segredos do GitHub Actions e de variáveis secretas do Azure Pipelines.
- ☐ Sei integrar Key Vault a pipelines e proteger o cofre com purge protection.
- ☐ Sei proteger recursos com Secure files, remoção de Open access e escopo do token do job.
- ☐ Sei atribuir níveis de acesso e grupos de segurança por menor privilégio.
- ☐ Sei o que cobrem code scanning, dependency scanning, secret scanning e push protection.
- ☐ Sei os requisitos dos conectores do Defender for Cloud para Azure DevOps e GitHub.

Questões de revisão do capítulo

R4.1 · Projetar e implementar um plano de segurança e conformidade

Um desenvolvedor da Startup Bike Já tentou enviar um commit com um token de API para um repositório com push protection habilitada e o push foi bloqueado. Ele afirma que o token é de teste e escolhe ignorar o bloqueio informando o motivo. O que acontece?

A. A push protection fica desligada no repositório por 24 horas
B. O commit é aceito sem registro, pois o motivo foi informado
C. O token é revogado na hora pelo GitHub junto ao provedor dele
D. O push é aceito e um alerta de bypass é gerado para revisão da equipe

R4.2 · Projetar e implementar um plano de segurança e conformidade

A equipe de AppSec da Varejista Lar & Cia tem dezenas de alertas de code scanning no GitHub. Para alguns, quer apenas uma sugestão de correção que o desenvolvedor revise e aplique; para outros, quer que o Copilot investigue, gere a correção, valide e abra um pull request em rascunho. Quais recursos correspondem a esses usos?

A. Agentic autofix para a sugestão única; Copilot Autofix para a correção completa com PR em rascunho no repositório
B. Dependabot security updates para os dois casos, já que eles abrem PRs para qualquer alerta
C. Copilot Autofix para a sugestão única; agentic autofix para a correção completa com PR em rascunho
D. Secret scanning com push protection para os dois casos, que bloqueia o código vulnerável no push

R4.3 · Projetar e implementar um plano de segurança e conformidade

A Seguradora Pacto Firme usa code scanning no GitHub e quer impedir o merge de PRs quando a ferramenta exigida encontrar alertas de severidade alta ou crítica, ou quando a análise ainda estiver em andamento. Como configurar?

A. Um alerta do Dependabot com severidade alta, que bloqueia merges automaticamente
B. Uma regra Require signed commits no ruleset, que valida a segurança do código enviado ao repositório
C. Uma proteção de tag no ruleset, que impede releases com alertas de code scanning
D. Uma regra Require code scanning results no ruleset, com a ferramenta e as severidades

R4.4 · Projetar e implementar um plano de segurança e conformidade

Na organização do Azure DevOps da Distribuidora Água Pura, o code scanning do GitHub Advanced Security for Azure DevOps está em configuração padrão. Os repositórios precisam acessar uma rede privada durante a varredura e há pipelines bloqueados por falta de tarefas. O que ajustar?

A. Adicionar as tarefas do CodeQL em cada pipeline, pois a configuração padrão não permite escolher agentes
B. Selecionar um pool auto-hospedado nas opções da configuração padrão e deixar Disable Marketplace tasks desligado
C. Trocar para a configuração padrão do GitHub, que roda as varreduras do Azure Repos nos runners do GitHub
D. Ligar Disable Marketplace tasks na organização para que as tarefas internas do CodeQL sejam usadas

Respostas comentadas

R4.1. D) O push é aceito e um alerta de bypass é gerado para revisão da equipe

Por quê: Push protection para repositórios exige GitHub Secret Protection, vem desabilitada por padrão e gera alertas quando há bypass.

✗ A) Incorreta. O bypass vale para aquele push; a proteção continua ativa no repositório.
✗ B) Incorreta. O bypass é registrado como alerta justamente para revisão posterior.
✗ C) Incorreta. O bypass não revoga o token; a revogação depende do dono do segredo.

> **Dica de prova:** Bypass de push protection: o push passa, mas fica um alerta para revisão.

R4.2. C) Copilot Autofix para a sugestão única; agentic autofix para a correção completa com PR em rascunho

Por quê: Há duas experiências de autofix para code scanning: Copilot Autofix (sugestão por alerta) e agentic autofix (o Copilot corrige de ponta a ponta e abre PR).

✗ A) Incorreta. Os papéis estão invertidos.
✗ B) Incorreta. Dependabot security updates corrigem dependências vulneráveis, não alertas de code scanning.
✗ D) Incorreta. Secret scanning trata de segredos, não de vulnerabilidades no código.

> **Dica de prova:** Sugestão pontual: Copilot Autofix. Correção completa com PR: agentic autofix.

R4.3. D) Uma regra Require code scanning results no ruleset, com a ferramenta e as severidades

Por quê: A merge protection do code scanning é configurada em rulesets e bloqueia PRs por severidade de alertas, análise pendente ou ferramenta ausente.

✗ A) Incorreta. Alertas do Dependabot não bloqueiam merges por si só.
✗ B) Incorreta. Commits assinados tratam da autoria, não de vulnerabilidades.
✗ C) Incorreta. Proteção de tag não avalia resultados de code scanning em PRs.

> **Dica de prova:** Bloquear merge por alertas do CodeQL: ruleset Require code scanning results.

R4.4. B) Selecionar um pool auto-hospedado nas opções da configuração padrão e deixar Disable Marketplace tasks desligado

Por quê: Code scanning em configuração padrão no Azure DevOps usa um pool (Azure Pipelines ou auto-hospedado) e agendamento definidos na organização; depende de tarefas do Marketplace.

✗ A) Incorreta. A configuração padrão permite escolher o pool de agentes e o agendamento.
✗ C) Incorreta. O GitHub Advanced Security for Azure DevOps roda nos agentes do Azure Pipelines.
✗ D) Incorreta. Com tarefas do Marketplace desabilitadas, a configuração padrão não consegue rodar as varreduras.

> **Dica de prova:** Default setup do CodeQL com rede privada: pool auto-hospedado e tarefas do Marketplace liberadas.

05

CAPÍTULO 5 · 10-15% DA PROVA

Projetar e implementar uma estratégia de controle do código-fonte

Neste domínio eu preciso escolher a estratégia de branching certa e saber exatamente qual política, ruleset ou permissão impõe cada exigência, no Azure Repos e no GitHub. A prova adora nomes exatos de opções: Reset all approval votes, Require linear history, Block force pushes, Limit merge types, Check for linked work items. Também cai recuperação e remoção de dados (revert, branch excluída, segredo e arquivo grande no histórico) e a migração com o GitHub Enterprise Importer. Onde mais erro é confundir opções vizinhas que soam iguais e esquecer que rulesets se agregam pela regra mais restritiva. Antes de responder, pergunto: isso é Azure Repos ou GitHub?

Neste capítulo
18 tópicos-chave · 14 pegadinhas · 7 macetes · 13 gatilhos · 4 questões comentadas

Tópicos-chave

Desenvolvimento baseado em trunk (modelo da Microsoft)

Pequenas mudanças integradas com frequência na main por PR, branches de tópico curtas, feature flags para esconder trabalho inacabado e branches de release cortadas da main a cada sprint. Branches de release não são mescladas de volta na main.

Hotfix no release flow

A correção entra primeiro na main por PR e depois é levada à branch de release com cherry-pick, que pode ser feito pela página do PR abrindo um novo PR para a release. Push direto, merge da main inteira ou esperar a próxima sprint estão errados.

GitHub flow x Git flow

GitHub flow: criar branch, fazer commits, abrir PR, revisar, mesclar e excluir a branch; mudanças só chegam à padrão pelo merge do PR. Git flow mantém branch develop e branches de feature e release de vida longa, o que gera conflitos.

Git worktree

Permite várias árvores de trabalho do mesmo repositório, cada uma com sua branch, para tarefas ou agentes em paralelo sem clonar de novo. O GitHub Copilot app usa uma worktree dedicada por sessão; stash, submódulos e forks não resolvem isso.

Rulesets do GitHub

Rulesets e proteções de branch funcionam juntos: todas as regras aplicáveis são impostas e vale a mais restritiva, sem prioridade entre organização e repositório e sem soma. Exceções controladas são feitas pela lista de bypass com equipes ou GitHub Apps específicas.

Regras de ruleset mais cobradas

Require signed commits (autoria verificada), Block force pushes (histórico intocável), Require linear history (exige permitir squash ou rebase merge), Restrict file size com exceções, Require code scanning results, Automatically request Copilot code review com Review new pushes e Require branches to be up to date before merging.

Merge queue

Para branch muito movimentada, valida combinações de PRs contra a versão mais recente antes de entrarem, sem obrigar autores a atualizar a branch e esperar checks de novo. Auto-merge, squash ou linear history não fazem essa validação.

Proteção de branch clássica

Por padrão bloqueia push forçado e exclusão, mas administradores contornam as regras; para incluí-los é preciso estender a regra a eles. Vale para qualquer push, não só PRs de forks.

Políticas de branch do Azure Repos

Mínimo de revisores com Reset all approval votes (zera só aprovações) ou Reset all code reviewer votes (zera tudo); Check for linked work items e Check for comment resolution; Optional só avisa, Required bloqueia. Validação externa usa PR status pela API com política de status check.

Build validation

Path filter evita builds em PRs que mudam só documentação. A expiração controla revalidação: Immediately when main is updated (branches com poucas mudanças), After n hours if main has been updated (main movimentada) ou Never (risco de validar contra main antiga).

Limit merge types e squash

Histórico linear no Azure Repos: permita Squash merge e Rebase and fast-forward. Basic merge (no fast-forward) e Rebase with merge commit criam commit de merge. Squash merge transforma cada PR em um único commit na main.

Permissões do Azure Repos

Force push (rewrite history, delete branches and tags) e Bypass policies when completing pull requests não são concedidas a nenhum grupo por padrão. Create branch pode ser configurada por pasta (negar na raiz, permitir em feature/ e users/). Lock congela uma branch sem impedir clonar e buscar.

Funções de repositório do GitHub

Read, Triage, Write, Maintain e Admin. Triage rotula, fecha e reabre issues e PRs e solicita revisores sem enviar código; Write é para quem envia código; Maintain gerencia sem ações sensíveis; Admin tem acesso total.

CODEOWNERS

Fica em .github, na raiz ou em docs (buscado nessa ordem); a última linha que casa com o arquivo define os donos. Com Require review from Code Owners basta a aprovação de um dos donos listados; para proteger o próprio CODEOWNERS ele precisa ter dono definido.

PRs do Copilot cloud agent

O agente envia commits só para a própria branch, respeita as proteções e não aprova, não marca como pronto nem mescla PRs. Quem pediu o PR não pode aprová-lo, e sem autoria humana há uma aprovação extra; workflows só rodam após alguém com escrita clicar em Approve and run workflows; comentários de quem não tem escrita não chegam ao agente.

Git hooks locais

commit-msg lê a mensagem e pode barrar o commit saindo com código diferente de zero (por exemplo, exigir AB#); pre-commit roda antes da mensagem existir e checa conteúdo (lint, testes, segredos); post-commit roda depois e serve para notificação e métricas.

Recuperação e remoção de dados

Commit compartilhado é desfeito com git revert (novo commit), nunca reset com push forçado ou amend. Branch excluída no Azure Repos é restaurada pela busca do nome exato em Branches, sem prazo de retenção. Segredo vazado: revogar ou rotacionar primeiro; arquivo grande: congelar o repositório, reescrever o histórico e push forçado com a permissão Force push.

GitHub Enterprise Importer (gh ado2gh)

Migra do Azure DevOps Cloud (não do Server) código com histórico, PRs com histórico e anexos, vínculos de itens nos PRs e políticas de branch do repositório. Não migra Boards, Pipelines, wikis, feeds, service connections, objetos LFS nem políticas por usuário; generate-script com --all também reconfigura pipelines, cria equipes e integra o Azure Boards.

Macetes para memorizar

Hotfix

Main primeiro, cherry-pick depois: a release nunca volta para a main.

Rulesets

Camadas não somam: vence a mais chata.

CODEOWNERS

Último que casa leva; um dono aprovando basta.

Hooks Git

pre = conteúdo, msg = mensagem, post = aviso.

Votos no Azure Repos

approval votes zera só o sim; code reviewer votes zera tudo.

Histórico linear

Squash ou Rebase and fast-forward; qualquer opção com merge commit no nome está fora.

Segredo vazado

Revoga, depois reescreve.

Tabela de memorização: não confunda

Quando é...	...e quando é
Reset all approval votes Novos commits devem remover só aprovações, mantendo votos de rejeitar e aguardar.	**Reset all code reviewer votes** Novos commits devem remover todos os votos.
git revert Desfazer commit já compartilhado sem reescrever o histórico.	**git reset --hard com push forçado** Nunca em branch compartilhada: reescreve o histórico e prejudica quem já baixou.
Require branches to be up to date Garantir que o PR foi testado contra a main atual, aceitando mais builds e retrabalho.	**Merge queue** Branch muito movimentada: mesma garantia sem fazer autores atualizar a branch e esperar checks de novo.
Squash merge Cada PR vira um commit, histórico linear.	**Basic merge (no fast-forward)** Mantém os commits da branch e cria commit de merge.
commit-msg Validar o texto da mensagem antes de criar o commit.	**pre-commit** Validar o conteúdo alterado (lint, testes, segredos).
Tag anotada Registrar autor, data e mensagem; pode ser criada no portal web e no Visual Studio.	**Tag leve** Só um ponteiro para um commit.
Trunk-based com feature flags Integrar cedo em branches curtas, escondendo funcionalidade inacabada.	**Git flow** Modelo com develop e branches de feature longas, que gera conflitos dolorosos no final.

Números que caem

Item	Valor

Peso do domínio no exame	**10-15%**
Tamanho máximo de arquivo durante a migração com GEI	**400 MiB**
Tamanho máximo de repositório na migração com GEI	**40 GiB**
Tamanho máximo de arquivo no GitHub após a migração	**100 MiB**

Alertas de prova (pegadinhas)

⚠ **Alerta de prova:** Somar aprovações de rulesets ou dar prioridade à ruleset da organização.
Como não cair: Não se soma nem há prioridade: vale a exigência mais restritiva.

⚠ **Alerta de prova:** Criar um segundo ruleset vazio para liberar a equipe de plantão.
Como não cair: Rulesets se somam; exceção é feita pela lista de bypass.

⚠ **Alerta de prova:** Usar Require linear history para impedir reescrita de histórico.
Como não cair: Linear history só impede commits de merge; quem impede reescrita é Block force pushes.

⚠ **Alerta de prova:** Achar que a primeira linha do CODEOWNERS prevalece ou que todos os padrões se combinam.
Como não cair: A última linha correspondente vence; padrões não se somam.

⚠ **Alerta de prova:** Achar que a proteção de branch clássica vale para administradores por padrão.
Como não cair: Administradores contornam por padrão; inclua-os explicitamente.

⚠ **Alerta de prova:** Marcar build validation como Optional ou trocar para Manual para economizar builds em PRs de documentação.
Como não cair: Use path filter; Optional enfraquece a validação e Manual depende de lembrar.

⚠ **Alerta de prova:** Escolher Rebase with merge commit para histórico linear.
Como não cair: Ele reaplica os commits, mas também cria commit de merge.

⚠ **Alerta de prova:** Usar pre-commit para validar a mensagem ou post-commit para bloquear.
Como não cair: Mensagem é commit-msg; post-commit roda depois que o commit já existe.

⚠ **Alerta de prova:** Executar git-filter-repo como primeiro passo após vazar um token.
Como não cair: Primeiro revogue ou rotacione o token; reescrever o histórico não invalida o segredo exposto.

⚠ **Alerta de prova:** Apagar o arquivo grande em um novo commit ou restaurar a branch excluída para limpar o histórico.
Como não cair: O arquivo continua no histórico; é preciso reescrevê-lo e fazer push forçado com Force push.

⚠ **Alerta de prova:** Supor que branches excluídas no Azure Repos ficam retidas só 30 dias ou que o portal roda git reflog.
Como não cair: Não há prazo de retenção; restaure pela busca de branches excluídas. Reflog é local do clone.

⚠ **Alerta de prova:** Corrigir na release e mesclar a branch de release de volta na main.
Como não cair: Corrija na main e leve com cherry-pick para a release.

⚠ **Alerta de prova:** Achar que o GEI migra do Azure DevOps Server, objetos LFS ou que rulesets do destino são ignorados.
Como não cair: Só do Azure DevOps Cloud; objetos LFS são enviados depois da migração; rulesets da organização podem fazer a migração falhar.

⚠ **Alerta de prova:** Achar que o Copilot cloud agent pode aprovar ou mesclar o próprio PR, ou que workflows rodam sozinhos nele.
Como não cair: Uma pessoa revisa e mescla; workflows dependem de Approve and run workflows.

Como cai na prova

📝 **Como cai:** Requisito de proteção descrito em linguagem de negócio (ninguém reescreve, todo commit assinado, só alguns contornam).
Como resolver: Traduza cada exigência em uma regra: reescrita = Block force pushes, autoria = Require signed commits, exceção = bypass list; descarte regras que tratam de outra coisa (linear history, file size).

📝 **Como cai:** Política do Azure Repos com opções de nome parecido.
Como resolver: Leia o detalhe do requisito (quais votos resetar, avisar ou bloquear, quais PRs disparam build) e escolha a opção exata: Reset all approval votes, Optional, path filter, After n hours.

📝 **Como cai:** Erro no histórico (commit errado, segredo, arquivo grande, branch excluída).
Como resolver: Pergunte se o histórico já foi compartilhado e se o dado é sensível: compartilhado = revert; segredo = revogar primeiro; arquivo grande = congelar, reescrever e Force push; branch excluída no Azure Repos = restaurar pelo portal.

📝 **Como cai:** Migração do Azure DevOps para o GitHub.
Como resolver: Liste o que o GEI leva (código, PRs, políticas de branch) e o que fica (Boards, Pipelines, wikis, feeds, LFS); confira a origem (só Cloud), limites de tamanho e a opção --all para rewire.

📝 **Como cai:** Agente do Copilot criando PRs em branch protegida.
Como resolver: Lembre que proteções continuam valendo, o agente não aprova nem mescla, quem pediu não aprova e workflows precisam de aprovação para rodar.

Gatilhos do enunciado

Se o enunciado trouxer...	Pense em
integrar pequeno e frequente na main, esconder trabalho inacabado	**Trunk-based com feature flags**
várias tarefas ou agentes em paralelo no mesmo repositório sem clonar de novo	**git worktree**
PRs testados contra main antiga quebraram a main	**Require branches to be up to date before merging (ou merge queue se o retrabalho incomoda)**
commit verificado e ninguém reescreve o histórico	**Require signed commits + Block force pushes**
bloquear arquivos de mídia grandes com exceção	**Restrict file size com exceções**
Copilot revisa todo PR e cada novo push	**Ruleset com Automatically request Copilot code review e Review new pushes**
avisar sem bloquear na fase de adoção	**Política Optional**
serviço externo aprova o PR no Azure Repos	**PR status pela API + política de status check**
branches só em feature/ e users/, release/ só para administradores	**Permissão Create branch por pasta**
branch não recebe commits durante auditoria, mas pode ser clonada	**Lock da branch**
gerenciar issues e PRs sem enviar código	**Função Triage**
migração com rewire de pipelines, equipes e Boards	**gh ado2gh generate-script --all**

histórico linear exigido no GitHub dá aviso de pré-requisito	**Permitir squash merge ou rebase merge no repositório**

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

- Rulesets e branch protection se agregam e vale a regra mais restritiva.
- Force push e Bypass policies não são dadas a nenhum grupo por padrão no Azure Repos.
- Require linear history no GitHub exige permitir squash ou rebase merge.
- Branch de release nunca é mesclada de volta na main; a correção vai por cherry-pick.
- Segredo exposto: revogar antes de reescrever o histórico.

Essenciais para a prova (checklist)

- ☐ Sei descrever trunk-based, GitHub flow, Git flow e o fluxo de hotfix da Microsoft.
- ☐ Sei quando usar git worktree.
- ☐ Sei como rulesets e proteções se combinam e como dar bypass.
- ☐ Sei as regras de ruleset para assinatura, force push, linear history, file size e code review do Copilot.
- ☐ Sei configurar votos, itens vinculados, build validation e status check no Azure Repos.
- ☐ Sei quais tipos de merge mantêm histórico linear no Azure Repos.
- ☐ Sei as permissões especiais do Azure Repos e as funções de repositório do GitHub.
- ☐ Sei como CODEOWNERS escolhe donos e quantas aprovações exige.
- ☐ Sei recuperar commit, branch excluída, segredo e arquivo grande no histórico.
- ☐ Sei o que o GitHub Enterprise Importer migra e suas limitações.

Questões de revisão do capítulo

R5.1 · Projetar e implementar uma estratégia de controle do código-fonte

Um desenvolvedor da Construtora Solo Firme fez push de um token de API em um commit da main no GitHub há duas semanas. O líder quer reescrever o histórico com git-filter-repo imediatamente. Qual deve ser o primeiro passo segundo o GitHub?

A. Tornar o repositório privado durante a limpeza
B. Excluir o repositório e recriá-lo com outro nome
C. Revogar ou rotacionar o token exposto
D. Executar git-filter-repo e forçar o push na main

R5.2 · Projetar e implementar uma estratégia de controle do código-fonte

Na Produtora Audiovisual Claquete, alguém fez commit e push de um vídeo de 2 GB no repositório Git do Azure Repos, e todos os clones ficaram enormes. Ninguém mais enviou alterações depois disso. Qual é o procedimento indicado para remover o arquivo do histórico da branch?

A. Excluir a branch e restaurá-la logo em seguida pela lista de branches excluídas do repositório
B. Apagar o vídeo em um novo commit, o que também o remove do histórico baixado pelos novos clones
C. Criar uma política de branch que bloqueie arquivos grandes, o que limpa automaticamente os commits antigos do repositório
D. Congelar o repositório, refazer o histórico local sem o vídeo e fazer push forçado com a permissão Force push

R5.3 · Projetar e implementar uma estratégia de controle do código-fonte

Na Rede de Postos Rota 66, a main do GitHub exige o status check de testes. Depois que dois PRs foram mesclados em sequência, a main quebrou, embora cada PR tivesse passado nos testes contra uma versão antiga da main. A equipe aceita fazer mais builds para evitar isso. Qual opção da regra habilitar?

A. Require branches to be up to date before merging
B. Require signed commits para todos os colaboradores
C. Restrict deletions na regra que protege a main
D. Require linear history na branch main do repositório

R5.4 · Projetar e implementar uma estratégia de controle do código-fonte

A Plataforma de Pagamentos Troco Certo criou um ruleset no GitHub exigindo aprovações e checks na main. Em incidentes graves, só a equipe de plantão e uma GitHub App de automação de releases podem contornar essas regras. Como configurar isso?

A. Criar um segundo ruleset sem regras para a equipe de plantão e outro para a GitHub App
B. Desativar o ruleset durante os incidentes
C. Dar a função Admin do repositório a toda a equipe de plantão e à GitHub App de automação
D. Adicionar a equipe de plantão e a GitHub App à lista de bypass do ruleset

Respostas comentadas

R5.1. C) Revogar ou rotacionar o token exposto

Por quê: Segredo vazado: primeiro revogue ou rotacione; reescrever histórico com git-filter-repo exige coordenação e tem efeitos colaterais.

✗ A) Incorreta. Mudar a visibilidade não revoga o token, que pode já ter sido copiado.

✗ B) Incorreta. Recriar o repositório não invalida o segredo que já foi exposto.

✗ D) Incorreta. Reescrever o histórico tem efeitos colaterais (hashes alterados, risco de recontaminação, proteções de branch) e não invalida o token já exposto.

> **Dica de prova:** Segredo no histórico: revogue antes de pensar em reescrever.

R5.2. D) Congelar o repositório, refazer o histórico local sem o vídeo e fazer push forçado com a permissão Force push

Por quê: Para tirar um arquivo grande do histórico de uma branch é preciso reescrevê-lo e fazer push forçado; isso exige congelar o repositório e a permissão Force push.

✗ A) Incorreta. Restaurar a branch excluída traz de volta o mesmo histórico com o vídeo.

✗ B) Incorreta. Um novo commit apagando o arquivo não o tira do histórico; ele continua sendo baixado nos clones.

✗ C) Incorreta. Uma política impede novos arquivos, mas não reescreve commits já enviados.

> **Dica de prova:** Arquivo grande no histórico: congelar, reescrever e push forçado (Force push).

R5.3. A) Require branches to be up to date before merging

Por quê: Status checks obrigatórios podem ser estritos (branch atualizada antes do merge) ou flexíveis; o estrito gera mais builds e evita quebras por combinação de PRs.

✗ B) Incorreta. Commits assinados tratam da verificação de autoria.

✗ C) Incorreta. Restringir exclusões impede apagar a branch, sem relação com a validação dos PRs.

✗ D) Incorreta. Histórico linear impede commits de merge, mas não revalida o PR contra a main atual.

> **Dica de prova:** PRs testados contra main antiga quebrando a main: exigir branch atualizada antes do merge.

R5.4. D) Adicionar a equipe de plantão e a GitHub App à lista de bypass do ruleset

Por quê: A lista de bypass de um ruleset define funções, equipes ou GitHub Apps que podem contornar as regras, sem desativar a proteção para os demais.

✗ A) Incorreta. Rulesets se somam e vale a regra mais restritiva; um ruleset vazio não anula o outro.

✗ B) Incorreta. Desativar o ruleset deixa a branch desprotegida para todos durante o período.

✗ C) Incorreta. Dar Admin concede acesso amplo e destrutivo além do necessário.

> **Dica de prova:** Exceção controlada a um ruleset: lista de bypass com equipes ou apps específicas.

DIA DA PROVA

Estratégia de prova

O AZ-401 é um exame de cenários de DevOps que mistura Azure DevOps e GitHub, com o GitHub Copilot e agentes de IA espalhados por todos os domínios. Quase toda alternativa errada é um recurso real que resolve outro problema, então a prova testa se eu sei o detalhe que separa recursos vizinhos: nome exato da opção, local do arquivo, sintaxe da plataforma certa. O critério de desempate é sempre segurança e menor privilégio: sem segredos de longa duração, sem Owner, sem PAT de pessoa. Há múltipla escolha e múltipla resposta (escolha DUAS ou TRÊS), e o domínio de pipelines pesa mais de um terço da nota.

Como responder

✓ Planeje o tempo para cerca de 50 itens em 100 minutos, uns 2 minutos por item, e marque para revisão o que travar. A Microsoft não divulga a quantidade de questões, e a página do exame não informa a duração; 100 minutos é a duração oficial dos exames role-based sem laboratórios (com laboratórios são 120).

✓ Antes de ler as alternativas, identifique a plataforma (Azure DevOps ou GitHub) e a restrição principal do cenário: sem segredos, sem manter máquinas, sem downtime, só leitura.

✓ Elimine alternativas com sintaxe da plataforma errada (dependsOn no GitHub Actions, needs no Azure Pipelines, ruleset no Azure Repos, webhook do GitHub no Azure Repos).

✓ Entre duas alternativas plausíveis, prefira a que usa identidade gerenciada, WIF, OIDC ou GitHub App e o papel mínimo.

✓ Nas de múltipla resposta, avalie cada alternativa isoladamente como verdadeira ou falsa; não escolha uma só porque a outra parece certa.

✓ Leia o verbo e o momento pedido: ao vivo, antes do commit, depois do merge, após cancelamento; muitas pegadinhas estão no momento em que o recurso atua.

✓ Distribua o estudo pelo peso: pipelines 35-40%, processos 15-20%, segurança 15-20%, instrumentação 15-20% e controle do código-fonte 10-15%.

✓ Treine com as 6 provas do curso e, a cada erro, anote o par de recursos que você confundiu.

Erros que derrubam candidatos

✗ Escolher o recurso que parece fazer a mesma coisa sem conferir o detalhe exigido (Reset all approval votes x Reset all code reviewer votes, always x succeededOrFailed).

✗ Errar o local ou o nome de arquivos do Copilot (copilot-setup-steps.yml fora da branch padrão, skill fora do repositório, job com outro nome).

✗ Aceitar PAT, client secret ou Owner como solução quando o cenário pede segurança.

✗ Esquecer que o cron do Azure Pipelines é em UTC e que AB# só muda estado na branch padrão.

✗ Confundir varreduras: push protection não acha vulnerabilidade de código e dependency scanning não acha segredo.

✗ Somar regras de rulesets em vez de aplicar a mais restritiva.

✗ Usar velocity como KPI ou métricas de host para memória da VM.

Na semana da prova

- ☐ Revise os pares de comparação e as pegadinhas de cada capítulo, começando por pipelines, que tem o maior peso.
- ☐ Refaça as provas do curso com tempo cronometrado e revise cada alternativa errada, não só a certa.
- ☐ Lembre que a prova é aplicada somente em inglês: revise os nomes das opções e recursos em inglês como aparecem nas telas (Bypass policies, Require linear history, Run mode Review).
- ☐ Confirme o agendamento: o exame está em beta, o resultado do beta não sai na hora (sai cerca de 10 dias depois que o exame entra em disponibilidade geral, prevista para dezembro de 2026) e a aprovação no beta conta para a certificação.
- ☐ Lembre que a certificação Agentic DevOps Engineer Expert exige o AZ-401 e mais o GH-300 ou o AZ-104.
- ☐ Na véspera, revise só a seção lembrar e os números de cada capítulo.

REVISÃO FINAL

Folha de cola

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

1. Implementar uma estratégia de instrumentação

‣ onde o trabalho está parando no quadro → **Cumulative Flow Diagram**

‣ histograma de execuções com duração e cores → **Build History**

‣ status por ambiente e taxa de testes → **Deployment status**

‣ tendência de testes de vários pipelines → **Test Results Trend (Advanced)**

‣ uma requisição por todos os serviços com tempos → **Detalhes de transação de ponta a ponta**

⚠ Log de atividades registra operações de gerenciamento; use transação ponta a ponta ou métricas de convidado com Azure Monitor Agent.

⚠ Topologia e dependências são do Application map.

⚠ Esses recursos estão nos Standard tests; o ping foi descontinuado.

2. Projetar e implementar pipelines de build e lançamento

‣ consumidores só veem versões que passaram nos testes → **Promover para a view @Release**

‣ registro público fora do ar e restauração determinística → **Upstream sources com um único feed na configuração**

‣ salvar pacotes do upstream sem publicar → **Feed and Upstream Reader (Collaborator)**

‣ pipeline lê mas não publica no feed → **Feed Publisher (Contributor) para a identidade Build Service**

‣ arquivos genéricos de vários gigabytes versionados → **Universal Packages**

⚠ Publica-se no feed base (@Local) e depois promove-se para a view.

⚠ Várias origens tornam a restauração imprevisível; use um único feed com upstream sources.

⚠ Número publicado não é reutilizado; saia com nova versão.

3. Projetar e implementar processos e comunicações

‣ visão de calendário de várias equipes com dependências → **Delivery Plans com links Predecessor/Successor**

‣ mudança formal, risco, revisão, trilha auditável ao criar projeto → **Processo CMMI**

‣ requisito verificado por caso de teste → **Vínculo Tested by**

‣ critérios de saída visíveis em cada coluna do quadro → **Definition of Done da coluna**

‣ ver o que terminou a etapa e aguarda a próxima → **Colunas divididas Doing/Done**

⚠ O remoto não aceita PAT; PAT é opção do servidor local.

⚠ O vínculo aparece, mas a transição de estado só ocorre no merge para a branch padrão.

⚠ Só mensagem de commit e descrição do PR ou da issue criam o vínculo.

4. Projetar e implementar um plano de segurança e conformidade

‣ sem segredos e rotação responsabilidade do Azure → **Identidade gerenciada**

‣ service connection com segredo que expira → **Azure Resource Manager com workload identity federation**

‣ implantar no Azure pelo GitHub Actions sem segredos → **OIDC com azure/login e credencial federada**

‣ integração com o GitHub usa PAT de pessoa que vai sair → **GitHub App instalada na organização**

‣ keystore ou certificado fora do repositório → **Secure files + DownloadSecureFile@1**

⚠ Só permite pedir o token OIDC; o acesso vem dos papéis da identidade federada.

⚠ Nesse caso o subject é repo:ORG/REPO:pull_request; com environment no job, vira repo:ORG/REPO:environment:NOME.

⚠ A chave ainda é segredo compartilhado; use identidade gerenciada com RBAC.

5. Projetar e implementar uma estratégia de controle do código-fonte

‣ integrar pequeno e frequente na main, esconder trabalho inacabado → **Trunk-based com feature flags**

‣ várias tarefas ou agentes em paralelo no mesmo repositório sem clonar de novo → **git worktree**

‣ PRs testados contra main antiga quebraram a main → **Require branches to be up to date before merging (ou merge queue se o retrabalho incomoda)**

‣ commit verificado e ninguém reescreve o histórico → **Require signed commits + Block force pushes**

‣ bloquear arquivos de mídia grandes com exceção → **Restrict file size com exceções**

⚠ Não se soma nem há prioridade: vale a exigência mais restritiva.

⚠ Rulesets se somam; exceção é feita pela lista de bypass.

⚠ Linear history só impede commits de merge; quem impede reescrita é Block force pushes.

GABARITO

Respostas do diagnóstico

D1. D) Application map do Application Insights, com os nós e chamadas → revise o capítulo 1

Por quê: O Application map identifica componentes pelo roleName na telemetria e desenha as chamadas HTTP entre eles, com saúde e falhas por nó.

✗ A) Incorreta. Testes de disponibilidade verificam se um endpoint responde, sem mostrar as chamadas internas.

✗ B) Incorreta. CPU dos nós não revela qual chamada entre serviços está falhando.

✗ C) Incorreta. Live metrics mostra telemetria em tempo real, mas não a topologia das dependências entre serviços.

> **Dica de prova:** Ver topologia, dependências e pontos de falha entre microsserviços: Application map.

D2. C) Cumulative Flow Diagram, com faixas por coluna do quadro → revise o capítulo 1

Por quê: O Cumulative Flow Diagram acompanha o fluxo de itens pelos estados/colunas; faixas que crescem revelam filas e gargalos.

✗ A) Incorreta. O burndown mostra o trabalho restante da sprint, não a distribuição por coluna.

✗ B) Incorreta. Velocity mostra o volume entregue por sprint, sem detalhar onde o trabalho para.

✗ D) Incorreta. O Query Tile mostra um número atual, sem a evolução por coluna.

> **Dica de prova:** Onde o trabalho está parando no quadro: Cumulative Flow Diagram.

D3. C) Abrir o alerta e selecionar Investigate, depois salvar a investigação como issue → revise o capítulo 1

Por quê: A deep investigation coleta e correlaciona sinais de aplicação, infraestrutura e plataforma e gera um relatório; issues preservam o contexto para continuar e compartilhar.

✗ A) Incorreta. O chat é temporário; a preservação e o compartilhamento são feitos por issues.

✗ B) Incorreta. Consultas manuais não fazem a correlação entre camadas que a deep investigation faz.

✗ D) Incorreta. O Observability Agent é iniciado pelo Azure Monitor, não por tickets externos.

> **Dica de prova:** Investigar um alerta com o Observability Agent: Investigate no alerta e salvar como issue.

D4. C) lockBehavior: sequential → revise o capítulo 2

Por quê: A verificação Exclusive lock permite uma execução por vez no recurso; lockBehavior define se só a mais recente segue (runLatest) ou todas em fila (sequential).

✗ A) Incorreta. Uma condição não controla a fila de execuções no recurso.

✗ B) Incorreta. runLatest, o padrão, faz só a execução mais recente adquirir o lock, e as anteriores são canceladas.

✗ D) Incorreta. Remover dependências não serializa as execuções.

> **Dica de prova:** Exclusive lock com todas as execuções em fila: lockBehavior sequential.

D5. A) No Managed DevOps Pools, as VMs dos agentes ficam em uma assinatura da Microsoft, não na sua → revise o capítulo 2

Por quê: Managed DevOps Pools simplifica pools personalizados (imagens próprias, escala, rede corporativa) e roda a infraestrutura em uma assinatura gerenciada pela Microsoft.

✗ B) Incorreta. O serviço fornece agentes do Azure Pipelines, não runners do GitHub Actions.

✗ C) Incorreta. As VMs não ficam na assinatura do cliente, e a escala é configurável e automática.

✗ D) Incorreta. Ele aceita imagens personalizadas e também imagens com o mesmo software dos agentes hospedados.

> **Dica de prova:** Managed DevOps Pools: evolução do VMSS, infraestrutura em assinatura da Microsoft.

D6. D) Colocar o feed interno antes do registro público na ordem dos upstreams → revise o capítulo 2

Por quê: A ordem dos upstreams define de onde o pacote vem; quem republica pacotes open source personalizados deve listar essa origem antes dos registros públicos.

✗ A) Incorreta. Colocar públicos primeiro é a recomendação geral, mas, para pacotes personalizados, a origem interna deve vir antes.

✗ B) Incorreta. Vários feeds no arquivo de configuração tornam a restauração imprevisível; use um único feed com upstreams.

✗ C) Incorreta. Mudar o número da versão não controla de qual origem o feed resolve o pacote.

> **Dica de prova:** Pacote open source corrigido internamente: upstream interno antes do público.

D7. A) No Copilot Chat, no Copilot code review e no Copilot cloud agent do GitHub.com → revise o capítulo 3

Por quê: Tipos de instruções: pessoais, de repositório (geral, por caminho e de agente) e de organização; as de organização exigem Business ou Enterprise e valem em superfícies do GitHub.com.

✗ B) Incorreta. Instruções de organização não têm suporte geral nas IDEs; lá valem instruções de repositório e pessoais.

✗ C) Incorreta. O Copilot CLI usa arquivos locais para instruções pessoais; as da organização não são o mecanismo dele.

✗ D) Incorreta. Elas valem para todos os membros da organização, independentemente de onde vem a licença do Copilot.

> **Dica de prova:** Instruções da organização: Copilot Chat, code review e cloud agent no GitHub.com.

D8. D) Conceder só escopos de leitura; o escopo nunca ultrapassa as permissões do usuário → revise o capítulo 3

Por quê: Clientes MCP personalizados devem receber escopos OAuth mínimos; a operação exige tanto o escopo do aplicativo quanto a permissão do usuário no Azure DevOps.

✗ A) Incorreta. O servidor remoto não aceita PAT.

✗ B) Incorreta. O cabeçalho restringe ferramentas, mas conceder escopos excessivos contraria o mínimo privilégio.

✗ C) Incorreta. Habilitar uma ferramenta ou toolset não concede nem limita permissões OAuth; o mínimo privilégio é feito nos escopos.

> **Dica de prova:** Cliente MCP próprio: escopos OAuth mínimos; escopo não supera a permissão do usuário.

D9. D) Tested by, da história de usuário para os casos de teste → revise o capítulo 3

Por quê: Vínculos de teste (Tests/Tested by) ligam casos de teste a requisitos e bugs, base para a rastreabilidade de qualidade e para widgets como Requirements quality.

✗ A) Incorreta. Parent e Child organizam a hierarquia de trabalho, não a relação de verificação por testes.

✗ B) Incorreta. Related é genérico e não expressa que o caso de teste verifica o requisito.

✗ C) Incorreta. Predecessor e Successor representam dependências de agenda.

> **Dica de prova:** Requisito verificado por caso de teste: vínculo Tested by.

D10. A) Identidade gerenciada da Function com papel RBAC no armazenamento → revise o capítulo 4

Por quê: Para código que roda em recursos do Azure, identidades gerenciadas eliminam a gestão de credenciais; entidades de serviço fazem sentido quando não há suporte a identidade gerenciada.

✗ B) Incorreta. A chave da conta ainda é um segredo compartilhado que precisa ser rotacionado, mesmo guardado no Key Vault.

✗ C) Incorreta. O certificado elimina o segredo em texto, mas a equipe continua responsável por emitir e renovar a credencial.

✗ D) Incorreta. Um segredo de cliente nas configurações viola a regra e exige rotação manual.

> **Dica de prova:** Código em recurso do Azure acessando outro serviço do Azure: identidade gerenciada.

D11. C) repo:rio-branco/api:pull_request → revise o capítulo 4

Por quê: O subject do token OIDC muda conforme o contexto: environment (se houver), pull_request, branch ou tag; a credencial federada precisa corresponder.

✗ A) Incorreta. O filtro de environment só vale quando o job referencia o environment.

✗ B) Incorreta. O filtro de branch só vale para eventos que não são de pull request e sem environment.

✗ D) Incorreta. O subject padrão não usa o nome do job.

> **Dica de prova:** OIDC para job de pull_request sem environment: subject repo:ORG/REPO:pull_request.

D12. C) A extensão Microsoft Security DevOps com a categoria de IaC → revise o capítulo 4

Por quê: O MSDO executa analisadores como Checkov, Terrascan, Template Analyzer e Trivy; pode ser limitado à categoria de IaC no pipeline.

✗ A) Incorreta. O widget acompanha requisitos e testes.

✗ B) Incorreta. Dependency scanning analisa dependências de código aberto, não templates de IaC.

✗ D) Incorreta. Secret scanning procura segredos, não configurações inseguras de infraestrutura.

> **Dica de prova:** Varredura só de IaC para o Defender for Cloud: MSDO com a categoria IaC.

Agora, treine no formato real

Faça as 6 provas completas do curso AZ-401: 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 Microsoft. As questões são inéditas e não reproduzem a prova real.